N° d'ordre : 2632

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Guillaume LATU**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Algorithmique parallèle et calcul haute performance dédiés à la simulation d'un système hôte-macroparasite

---

Soutenue le : 17 décembre 2002

Après avis des rapporteurs :

Frédéric Desprez	Directeur de recherche INRIA
Ralf Gruber	Adjoint scientifique à l'EPFL
Alain Pavé	Professeur

Devant la commission d'examen composée de :

Michel Langlais	Professeur	Président
Patrick Silan ...	Chargé de Recherche CNRS .	Rapporteur
Jean Roman	Professeur	Directeur de thèse
Frédéric Desprez	Directeur de recherche INRIA	
Ralf Gruber	Adjoint scientifique à l'EPFL	
Alain Pavé	Professeur	

– 2002 –

Algorithmique parallèle et calcul haute performance dédiés à la simulation d'un système hôte-macroparasite

Résumé : Ce travail contribue à un modèle déterministe discret d'un système hôte-macroparasite et propose un modèle stochastique équivalent. Le phénomène d'agrégation des macroparasites est classiquement pris en compte par une distribution des parasites sur les hôtes de type loi binomiale négative. Les hétérogénéités de la population parasitaire y sont reproduites plus finement, ce qui constitue une approche originale. Une application du modèle est donnée pour un macroparasite constituant un élément pathogène en aquaculture : le système Bar-*Diplectanum aequans*. Ce système est étudié quantitativement à l'aide de deux simulateurs numériques parallèles.

Les temps d'exécution d'un simulateur séquentiel déterministe étant rédhibitoires, une mise en œuvre parallèle est incontournable. Nous présentons deux distributions des données et des calculs, ainsi qu'une étude algorithmique détaillée des solutions parallèles. Des résultats expérimentaux sont donnés sur IBM SP2, SP3, Regatta et SGI Origin 3800. Après optimisation, une simulation engendre des coûts de calcul allant jusqu'à 105 TFLOP, voire 1.45 PFLOP. L'extensibilité de très bonne qualité est évaluée théoriquement et testée. Nous avons pu constater plus de 77 % d'efficacité sur 448 processeurs au niveau de simulations entières, le noyau de calcul parallèle utilisé atteignant lui 92 %. Une étude sur l'utilisation des mémoires caches permet d'atteindre 60 % de la puissance crête au cœur des calculs. La précision des calculs a été améliorée, on reproduit ainsi des dynamiques observées sur le terrain qui n'étaient pas accessibles avec le simulateur précédent.

Les techniques algorithmiques mises en œuvre pour un deuxième simulateur basé sur une méthode de type Monte-Carlo sont présentées. Ce simulateur individu-centré utilise 3 niveaux de parallélisme différents. On donne les performances associées à une programmation hybride sur une grappe de nœuds SMP, particulièrement bien adaptée à ce type d'application. On compare les simulateurs stochastiques et déterministes en termes de résultats qualitatifs. L'étude quantitative effectuée donne un éclairage nouveau sur l'interaction des mécanismes des systèmes hôte-macroparasite régulés principalement par la mort des hôtes sur-infestés. L'analyse des simulations conduit à améliorer le modèle ; elle permet de mieux comprendre et clarifier les mécanismes du système biologique réel.

Mots clés : Algorithmique parallèle, simulation numérique, modèle discret, modèle individu-centré, système hôte-parasite, dynamique de populations, *Monogenea*.

Discipline: Informatique

LaBRI,
351, cours de la libération,
33405 Talence Cedex (FRANCE).

LABRI, UMR 5800, Thème ALienor,
Université Bordeaux 1 et ENSEIRB,
Projet ScAlApplix, INRIA UR Futurs.

Parallel algorithmic and high performance computing dedicated to a simulation of a host-macroparasite system

Abstract: We are interested in a host-macroparasite system. Ecological and epidemiologic interests are motivating the study of their population dynamics. The model we have contributed to is both deterministic and stochastic, and includes non-uniform dynamics within the cohort of parasites. Several biological processes are combined in order to reach a good level of realism, such as the parasite aggregation. Therefore, the parasite distribution on the hosts is not predetermined but explicitly computed.

For this system, a previous deterministic discrete simulations brought about too large computations, and some simulations had very long execution times. We implemented a high-performance simulator working on parallel machines (IBM SP2, IBM SP3, SGI Origin 3800, IBM Regatta) which provides more accurate computations. The algorithmic study and the performance analysis establish the efficiency and scalability (checked up to 448 processors) of the parallel algorithm. The parallel efficiency reaches 77% on 448 processors for a complete simulation, and the computation kernel corresponds to an efficiency of 92%. A study of memory accesses and cache utilization lead to an implementation reaching 60 % of peak performance in the computation kernel on the IBM SP3. We give qualitative results for the Sea bass-*Diplectanum aequans* system (corresponding to a pathological problem in fish farming). A full analysis of numerical simulations has allowed us to tune the model to get a realistic qualitative behavior. Results of numerical simulations show the effect of overdispersion, parasite and host mortality on the parasite distribution, and host regulation (occurring through cycles).

We developed another individual-based model. We describes the Monte Carlo algorithm of the stochastic simulator and its three levels of parallelism. Analysis and performance, up to 256 processors, of a hybrid MPI/OpenMP code are then presented for a cluster of SMP nodes. The qualitative results of both parallel simulators are finally compared. The detailed quantitative study gives some insights to the interaction of mechanisms taking place in a host-parasite system with parasite-induced host mortality. Improving the model leads to a deeper understanding of the processes occurring in the real biological system.

Keywords : parallel algorithmics, numerical simulation, discrete model, individual-based model, host-parasite system, population dynamics, *Monogenea*.

Discipline: Computer-Science

LaBRI,
351, cours de la libération,
33405 Talence Cedex (FRANCE).

LaBRI, UMR 5800, ALienor Team,
Bordeaux 1 University and ENSEIRB,
ScAlApplix Project, INRIA UR Futurs.

Remerciements

Je tiens vivement à remercier ici :

Frédéric Desprez, directeur de recherche à l'INRIA, pour l'honneur qu'il m'a fait en acceptant de rapporter sur ce travail. Je suis particulièrement sensible au fait qu'il ait pris de son temps pour cette lourde tâche ;

Ralf Gruber, adjoint scientifique à l'EPFL, qui a bien voulu rapporter cette thèse. Ses remarques m'ont permis de compléter utilement ce document et son aide a été précieuse pour y jeter un regard neuf ;

Michel Langlais, professeur à Bordeaux 2, pour avoir accepté de présider le jury de ma thèse. Je voudrais lui exprimer toute ma gratitude pour le suivi et l'intérêt qu'il a manifesté tout au long de cette thèse. Sa disponibilité et ses conseils m'ont guidé dans la complexité inhérente à la modélisation ;

Alain Pavé, professeur des Universités et directeur de recherche du CNRS, pour l'immense honneur qu'il m'a fait en acceptant d'être rapporteur de ce travail ;

Jean Roman, professeur à l'ENSEIRB, pour avoir contribué à l'aboutissement de cette thèse, pour son dynamisme, pour sa présence de tous les instants. Ses compétences et sa rigueur dans le travail, m'ont permis de mener à bien les travaux que nous avons entrepris. Enfin, merci pour m'avoir fait découvrir le parallélisme.

Un grand merci à Patrick Silan, chargé de recherches au CNRS, pour sa confiance, et son accompagnement au long de ces quatres années. J'ai découvert grâce à lui la communauté de l'Ecologie, des problématiques scientifiques différentes de celles qui nous intéressent en informatique et la manière de les aborder.

J'ai vécu et partagé l'enthousiasme et une certaine passion à enseigner à l'IUT informatique. Un grand merci à tous ces membres et à la chaleureuse pause café qui m'a permis d'y apprécier la vie d'une communauté d'enseignants. Merci aussi aux joueurs de ping-pong de l'IUT pour la détente et le grand plaisir que nous avons pris à faire ensemble du sport (Oui Nico, c'en est).

Je remercie aussi mes précieux relecteurs qui ont permis de peaufiner ce document : Jean, Michel et Patrick bien sûr, mais aussi Akka, François, Olivier, Pierre-André, Pierre, Raymond, Robert, Valère. Dans l'urgence pour certain, ils ont su consacrer de leur temps pour détecter les coquilles, les bourdes, et les phrases à rallonge.

Un grand merci aux très nombreux co-bureau avec qui nous avons partagé beaucoup d'esclaffades, et autre rigolades. Merci pour l'odyssée de la première année aux galériens de la première heure de la spacieuse salle bateau : Arnault, Davy, Pascal (le gourou), Valère, et Olivier ... Les discussions à rallonge dans lesquelles nous n'hésitions pas à sombrer agrémentaient alors nos journées. Et puis, merci aux habitants de la salle A de ces dernières années, Irek, Nicolas et les nouveaux arrivés. La bonne humeur quotidienne (tout à fait guillaume) a permis d'oublier la diminution progressive des locaux.

Merci aux nombreux membres du LaBRI avec qui j'ai passé d'excellents moments. Leur ouverture d'esprit alliée à une propension à faire la fête de temps en temps m'ont accompagné au long de ces dernières années. J'y ai beaucoup apprécié la disponibilité de chacun, et la joie de partager notamment ses sujets de recherche. Merci à Bernard, sans lequel ni moto ni vélo ne saurait fonctionner. Je pense aussi aux inestimables compagnons de ce voyage, Akka, Pierre R., David, P.A., Gwenola, Olivier, Pierre C., Jaco, Laurent et bien d'autres que je ne devrais pas oublier, et pourtant ...

Je remercie l'équipe ALiENor du LaBRI parmi laquelle j'ai passé ses dernières années et qui m'a toujours soutenue. Je pense ici aussi à tous les membres de l'<avant projet–presque projet–projet accepté–nouveau projet> ScAlApplix auquel je participe.

J'ai une pensée aussi pour mes parents Danielle et Jean ainsi qu'à mes sœurs Frédérique et Marylise. Ils ont accepté mes déplacements professionnels, et la rédaction de thèse, qui à maintes reprises nous ont privés de la joie de nous réunir. Je n'oublierai pas les Sartrouvillois de la première heure Diegman, Samuel et Thomas, malheureusement loin, bien loin de Bordeaux.

Le dernier merci et non des moindres revient à ma fiancée, Isabelle, qui m'a toujours soutenu. Elle m'a donné la force nécessaire pour naviguer entre, mais aussi exercer, les metiers d'enseignant et de chercheur. Sans son inestimable amour, je n'aurais jamais écrit les 231 pages qui composent cette thèse.

Table des matières

Partie I

Introduction et cadre de l'étude

Chapitre 1

Modélisation en dynamique de populations

1.1 Outils de modélisation

La dynamique de populations fait partie intégrante de l'écologie. Elle a notamment un but socio-économique : elle s'intéresse par exemple à la lutte contre les ravageurs, à la diffusion d'épidémies, au suivi des espèces menacées de disparition, et enfin à l'élevage, à la gestion et l'exploitation de ressources vivantes. Cet objectif va de pair avec des questions scientifiques importantes : quels sont les mécanismes principaux intervenant dans la dynamique de populations ? Peut-on identifier des structures au sein des populations ? Une stabilité, un équilibre, des cycles peuvent-ils être constatés ? Ces problèmes sont naturellement abordés par des observations directes de populations. Ensuite, la conception de modèles (statistique, conceptuel ou mécaniste) constitue un outil pour comprendre les dynamiques.

Un modèle *statistique* va par exemple étudier les corrélations entre une densité de population et des facteurs qui dépendent du temps ou de la position spatiale, sans nécessairement formuler explicitement les raisons qui ont conduit à la situation constatée. Il s'appuie sur une analyse approfondie des données de terrain ; il peut éventuellement être utilisé pour prédire l'évolution temporelle des densités de population. Qualifiés parfois de "boîte noire", les processus en œuvre dans le système ne sont pas nécessairement identifiés. Au contraire, un modèle *conceptuel* va répertorier les variables théoriques pertinentes qui sont susceptibles d'influer sur l'évolution des populations et avance une relation de cause à effet entre les variables en question. Il conviendra ensuite de vérifier la justesse des hypothèses en comparant les résultats du modèle aux mesures expérimentales. Un modèle *mécaniste* s'attache à décrire l'interaction de phénomènes bien identifiés. Il permet d'appréhender des comportements dits émergents qui sont issus de la combinaison dynamique des éléments du modèle. Les travaux présentés dans les prochains chapitres relèvent d'un modèle mécaniste.

1.1.1 Systèmes et modèles

Dans ce chapitre sont présentés les différents types de mécanismes utilisés pour modéliser des dynamiques de populations. On se focalisera sur les éléments utiles pour une étude quantitative.

Un *système* en dynamique de populations est circonscrit par un ensemble d'entités en interaction. Ces dernières possèdent un état qui peut être modifié par un certain nombre de processus et correspondent à une échelle d'observation fixée. Dans un premier temps, il conviendra donc de distinguer ce qui est inclus dans le système de ce qui en est exclus. Un modèle mécaniste procède nécessairement à une simplification du système afin d'en comprendre certains aspects, ou de répondre à certaines questions. En fonction des objectifs visés, on procède à des hypothèses de modélisation qui permettent de reproduire au mieux le phénomène que l'on souhaite observer [16]. Le modèle peut nous permettre d'accéder à des phénomènes macroscopiques insoupçonnables de prime abord à la vue des mécanismes et entités qui le constituent. D'autre part, des mesures quantitatives permettent à l'expérimentateur de mieux appréhender la dynamique issue de la combinaison de plusieurs processus. En général, le modèle n'est pas destiné à appréhender tous les aspects du système, voire à le reproduire entièrement avec tous ses détails. Il permet entre autre chose, de formaliser un ensemble de connaissances, de tester des hypothèses, d'explorer des interactions entre mécanismes.

Le modélisateur définit le degré de finesse de son modèle. En intégrant beaucoup de détails, il doit disposer de nombreuses mesures expérimentales et la simulation informatique correspondante devient complexe et coûteuse en temps d'exécution. De plus, l'interprétation des résultats sera délicate (nombreux jeux de paramètres), et les incertitudes de mesure sur les entrées peuvent éventuellement augmenter l'incertitude sur les sorties. D'un autre côté, un niveau d'abstraction plus haut, qui vise un nombre restreint de paramètres et peu de variables d'état, peut négliger des mécanismes importants. On s'expose alors à travailler sur un modèle qui n'est pas adapté vis-à-vis des questions qui lui sont adressées.

1.1.2 Mécanismes à modéliser

La dynamique de populations et l'épidémiologie en particulier envisagent des systèmes intégrant différents types de mécanismes. Les principaux vont être présentés dans cette Section. Les systèmes comportent une ou plusieurs espèces ; les interactions se font entre individus d'une certaine espèce (*intra-spécifique*) ou entre individus n'appartenant pas à la même espèce (*inter-spécifique*). Le système peut par exemple représenter :

- une épidémie dont l'origine est un virus ou des parasites (inter-spécifique),
- la croissance et la structure d'une forêt constituée d'arbres d'une seule espèce (intra-spécifique),
- un système comportant des prédateurs, des proies et la ressource alimentaire des proies (inter-spécifique).

Les entités du système sont affectées par les ressources vivantes présentes (*facteur biotique*, *c.a.d.* qui a pour origine un être vivant) et l'espace où elles évoluent. D'autre part, la température (*facteur abiotique*, c'est-à-dire non lié à l'action d'êtres vivants) ou d'autres facteurs spécifiques influent sur le système. Le modélisateur choisit les éléments les plus pertinents à retenir dans le modèle. Nous avons brièvement énuméré qui sont les acteurs du système et leur environnement. Attardons-nous maintenant sur les phénomènes que l'on veut modéliser au sein du système.

1.1.2.1 Densité-dépendance et intensité-dépendance

Dans certains cas, les populations voient leur mortalité s'élever, ou la taille des individus se réduire, ou encore la fécondité diminuer lorsque le nombre d'individus devient trop important. On dit alors que certains phénomènes dépendent de la densité des populations (*densité-dépendance*). Notamment, cette notion s'illustre lorsqu'il y a lutte pour le territoire, compétition pour la nourriture ou pour d'autres ressources. Trois types de densité-dépendance sont fréquemment rencontrées dans la littérature [2] :

- le type *contest* correspond à un partage inéquitable d'une ressource entre individus. Lorsque le nombre d'individus augmente, il y en a toujours certains qui ont assez de ressources pour survivre et se reproduire. D'autres en seront privés et pourront éventuellement ne pas survivre ;
- le type *scramble* signifie qu'il y a un partage équitable d'une ressource. Lorsqu'il y a trop d'individus, et que la ressource vient à manquer, tous en subissent les conséquences ;
- le type *Allee effect* exprime le fait qu'au dessous d'un seuil de densité de population, le taux de survie diminue. On peut prendre le cas d'une population d'ours : dans des conditions critiques, une décroissance de la densité conduit à une probabilité de rencontre moindre, et finalement à un faible taux de reproduction.

L'*intensité-dépendance* est un autre mécanisme proche conceptuellement de la densité dépendance. L'*intensité* est définie comme le nombre d'individus d'une espèce parasite présente sur un individu hôte. L'abondance est le nombre moyen de parasites fixés par hôte. L'intensité moyenne correspond au nombre moyen de parasites fixés uniquement parmi les hôtes infestés [15]. Lorsqu'un mécanisme est fonction de l'intensité parasitaire sur les hôtes et non pas de leur seule présence, on dit qu'il est intensité-dépendant. Par exemple, l'hôte pourra développer une réponse immunitaire graduelle, ou adopter un comportement particulier en fonction de l'intensité parasitaire.

1.1.2.2 Structure en âge ou variable structurante

L'âge détermine une ou plusieurs caractéristiques des individus dans les populations que l'on considère. Si cela est prépondérant sur la variation inter-individuelle (*i.e.* la variation de cette caractéristique entre plusieurs individus du même âge), le modélisateur peut en tenir compte. Dans d'autres cas, ce ne sera pas l'âge, mais le sexe ou la taille des individus qui conditionnera par exemple un comportement. Il est alors possible de définir des compartiments à l'intérieur d'une population pour prendre en compte les caractéristiques propres à certains compartiments. Introduire une variable structurante affine le modèle et enrichit la dynamique de populations pour la rendre plus réaliste.

1.1.2.3 Métapopulation

Les populations étudiées ne sont pas toujours réparties de manière homogène dans l'espace, ce qui peut infléchir la dynamique. Il peut notamment exister des ensembles d'individus isolés avec peu d'échanges entre les groupes. Cette collection de groupes en interaction est appelée métapopulation. On représente ces ensembles comme des compartiments entre lesquels circulent des flux d'individus ou d'autres flux.

1.1.2.4 Modèle structuré en espace

La représentation spatiale des individus sur une ou plusieurs dimensions peut permettre de rendre compte de certains phénomènes liés par exemple aux déplacements des individus, à leur territoire. La dispersion de graines, la migration d'individus, la diffusion d'une épidémie, la lutte pour le territoire sont exprimables dans un modèle spatialisé. D'autre part, l'espace considéré est classiquement entre une et trois dimensions (mais pas obligatoirement). Par exemple, on peut modéliser le flux de planctons à différentes profondeurs z (une seule dimension) ; par analogie, on peut regarder le nombre d'hôtes ayant l parasites et étudier une distribution dans l'espace des hôtes (une dimension). Ce qui distingue les modèles structurés en espace et les métapopulations réside dans la continuité de l'espace considéré et dans le fait d'utiliser les coordonnées dans cet espace.

1.1.2.5 Effet Retard

Certains événements n'ont pas d'effets immédiats mais agissent après une latence. Par exemple, la notion de délai est intégré à certains modèles pour séparer l'acte reproducteur de la naissance d'un nouvel individu, ou bien pour différencier les différents stades d'une maladie (période d'incubation, période infectieuse et non-infectieuse). Le fait d'ajouter des délais dans le modèle peut radicalement changer son évolution dans le temps.

1.1.3 Conclusion

On a présenté dans cette partie quelques mécanismes et structures employés pour aborder un système où interagissent des populations. Sans être exhaustif, cela donne un aperçu des outils élémentaires qui serviront ensuite à construire un modèle mécaniste.

1.2 Modèles mathématique et/ou informatique

1.2.1 Objectifs

La complexité de certains systèmes naturels (biologiques, écologiques et sociaux) représente un défi à l'entendement [54]. De nombreuses informations doivent être intégrées pour obtenir une représentation maîtrisable de multiples mécanismes interagissant à différentes échelles. En construisant une représentation des phénomènes observés, et en tenant compte des données recueillies sur le terrain, on peut approfondir nos connaissances d'un système. De tels travaux permettent de réaliser la synthèse d'événements interactifs pour expliquer des comportements macroscopiques. Là où l'expérimentation réelle n'est pas possible, la modélisation constitue un moyen pour valider les mécanismes que l'on a "plaqués" sur le système. Ainsi, des hypothèses formulées *a priori* peuvent être infirmées, et il est quelquefois possible de prédire l'évolution du système biologique. Il existe trois grandes classes de modèles que l'on précisera dans le paragraphe suivant : *déterministe*, *stochastique*, ou *hybride*. On distingue ensuite ceux qui sont *analytiques*, de ceux qui emploient la *simulation informatique*.

1.2.2 Types et exemples de modèle

Les modèles déterministes décrivent l'évolution moyenne temporelle de certains *compartiments* définis dans une ou plusieurs populations. Chaque compartiment (considéré homogène), induit par une structure en âge, spatiale ou autre, contient des variables d'état qui le caractérisent. Des équations explicitent les lois qui relient ces différentes variables, les paramètres du modèle et le temps. Les modèles déterministes sont particulièrement bien adaptés lorsque les compartiments correspondent à de larges ensembles d'individus [54]. Des phénomènes aléatoires minimes du système réel ne doivent pas influer notablement sur la dynamique du système, car cela invaliderait l'emploi du modèle déterministe.

Les modèles stochastiques sont utilisés lorsque l'on veut savoir quelle est la dynamique des lois de distribution de certaines variables d'état, ou lorsque certains événements aléatoires joue un rôle prépondérant dans la dynamique. Ils donnent les probabilités que le système et les variables qui le décrivent se trouvent dans tel ou tel état. Les mathématiques mises en œuvre dans ce type de modèle sont peu maniables, et elles font largement appel aux statistiques.

Outre ces deux grands types qui peuvent s'hybrider dans certains cas, on qualifie les modèles selon d'autres caractéristiques. Un modèle peut tout d'abord être continu ou discret en *temps*. D'un autre côté, il peut être non spatialisé, continu, ou bien discret en *espace*. Dans le cadre des modèles analytiques, les mathématiques servent à extraire des propriétés générales ou propres à la dynamique du système. Elles améliorent ainsi directement la compréhension des phénomènes. La simulation permet, elle, d'intégrer plus de paramètres, d'être éventuellement de plus bas niveau d'abstraction, de gérer des hétérogénéités multiples. Elle a moins de contraintes structurelles que les modèles analytiques. La prédiction des sorties du système est accessible ; par contre, la simulation n'apporte pas immédiatement une meilleure compréhension du système.

La simulation d'un modèle d'un système réel ne fournit pas de solution générale mais l'évolution séquentielle des variables d'état de ce système connaissant les paramètres d'entrée. On peut faire l'analogie entre une simulation effectuée à partir du modèle du système réel, et une véritable expérience qui porte sur le système réel lui-même. C'est à travers de multiples simulations, des analyses précises des sorties en fonction des entrées, et en isolant certains mécanismes, que des caractéristiques du système peuvent être dégagées. En cela, la simulation constitue un *outil d'investigation*. Dans les Sections suivantes, des exemples de modèles sont présentés.

1.2.2.1 Modèle déterministe discret en temps

La discrétisation du temps permet d'introduire des non-linéarités temporelles (naissance d'un individu, mort, modification physiologique). Un autre atout est celui de pouvoir définir des classes d'âge sur la base d'un pas de temps, pour incorporer un taux de fécondité ou de mort différent selon la classe d'âge. Prenons par exemple un modèle déterministe discret en temps, non spatialisé, qui considère l'évolution d'une seule population et regardons quel type de développement mathématique peut être fait [30]. Trois classes d'âge sont considérées, et seuls les individus des deux dernières classes peuvent se reproduire (taux de fécondité $f > 0$). Les individus des deux premières classes d'âge passent dans la classe d'âge suivante avec des taux de survie s_0 et s_1 (réels positifs), tandis que ceux qui sont dans la dernière classe meurent dès le pas de temps suivant. Si $n(t) = (n_0(t), n_1(t), n_2(t))$ représente les effectifs des trois classes d'âge par ordre croissant, on a le système :

$$\begin{cases} n_0(t+\Delta t) = f \times (n_2(t) + n_1(t)) \\ n_1(t+\Delta t) = s_0\, n_0(t) \\ n_2(t+\Delta t) = s_1\, n_1(t) \end{cases} \quad \text{ou} \quad n(t+\Delta t) = \underbrace{\begin{bmatrix} 0 & f & f \\ s_0 & 0 & 0 \\ 0 & s_1 & 0 \end{bmatrix}}_{P} n(t) \ .$$

La matrice de passage P permet de calculer $n(t+\Delta t)$ en fonction de $n(t)$; elle est appelée matrice de Leslie [14]. Avec une telle écriture, il est possible d'analyser sur le long terme la structure de la population. Regardons s'il existe une répartition en âge stable lorsque t est grand. On cherche donc à savoir s'il existe λ tel que $n(t+\Delta t) = \lambda\, n(t)$. Les calculs donnent par exemple (on prend $f = 1, s_0 = \frac{1}{2}, s_1 = 1$) :

$$n(t+\Delta t) = P\, n(t) = \lambda\, n(t) \ \Rightarrow \ det(P - \lambda\, I) = 0 \ \Rightarrow$$

$$\begin{vmatrix} -\lambda & 1 & 1 \\ \frac{1}{2} & -\lambda & 0 \\ 0 & 1 & -\lambda \end{vmatrix} = 0 \ \Rightarrow \ \frac{1}{2} + \frac{1}{2}\lambda - \lambda^3 = 0 \Leftrightarrow \ (\lambda - 1)\,(\lambda + \frac{1}{2} + \frac{1}{2}\mathrm{i})\,(\lambda + \frac{1}{2} - \frac{1}{2}\,\mathrm{i}) = 0 \ .$$

Lorsque la matrice P admet une valeur propre réelle simple λ, dominante et strictement positive, on interprète λ comme le taux de croissance asymptotique de la population. Le vecteur propre associé à λ donne la structure de la distribution en âge lorsque t est grand. Ici, la valeur propre simple dominante est $\lambda = 1$. Pour cette valeur de λ, on cherche le vecteur propre n^*, telle que la somme des composantes de n^* soit égale à 1. Cela implique les équations suivantes :

$$\begin{cases} n_0^* = n_2^* + n_1^* \\ n_1^* = \frac{1}{2}\, n_0^* \\ n_2^* = 1\, n_1^* \\ \text{et} \quad n_0^* + n_1^* + n_2^* = 1 \text{ (normalisation).} \end{cases} \qquad \Rightarrow \quad n^* = \begin{pmatrix} 0,5 \\ 0,25 \\ 0,25 \end{pmatrix}$$

Asymptotiquement, le vecteur $n(t)$ est un multiple de n^*. Les deux dernières classes auront donc un nombre identique d'individus, et la première classe d'âge contiendra autant d'individus que les deux dernières réunies.

1.2.2.2 Modèle stochastique continu en temps

Modèle déterministe analytique Nous allons considérer un modèle déterministe continu en temps et étudier dans les paragraphes qui suivent des équivalents stochastiques [47]. Soit un système comportant une population initiale de $n(0) = 0$ individu. On modélise un phénomène d'immigration ou de recrutement dont le flux moyen est $g > 0$ et une mortalité de taux $m > 0$. L'équation déterministe de ce modèle d'immigration et de mort (*Linear Immigration Death model* dans la litérature) s'écrit immédiatement :

$$\frac{d\,n(t)}{dt} = g - m\,n(t) \ , \quad \text{qui a pour solution} \;\; n(t) = (1 - e^{-m\,t})\,\frac{g}{m} \ .$$

Le nombre d'individus tend vers $\frac{g}{m}$ pour $t \to \infty$. Dans la modélisation déterministe, on suppose de manière implicite que la mortalité suit dans le temps une loi de distribution exponentielle[1] (s'il n'y avait pas d'immigration, on aurait $n(t) = n(0)\, e^{-m\,t}$). Si la loi de distribution temporelle observée n'est pas exponentielle, on envisagera par exemple des simulations de type *Discret-Event Systems* [23].

Modèle stochastique analytique Pour l'exemple considéré, on suppose maintenant que $N(t)$ est la variable aléatoire désignant le nombre d'individus au temps t. On définit les probabilités $(p_i(t) = Prob[N(t) = i])_{i \in [0,+\infty[}$ que le système ait i individus au temps t. Le vecteur des probabilités $p_i(t)$ est noté $p(t)$. Durant un pas de temps Δt très petit, $N(t)$ a une probabilité infime de gagner ou de perdre strictement plus de 1 individu (les variations de deux ou plus individus sont classiquement négligées). Les probabilités de transition instantanée peuvent donc être énoncées comme suit :

$$\begin{cases} \text{Prob}\{N(t) \text{ augmente de 1 pour une immigration}\} = g\,\Delta t + o(\Delta t) \\ \text{Prob}\{N(t) \text{ diminue de 1 car un individu est mort}\} = N(t)\, m\,\Delta t + o(\Delta t) \\ \text{Prob}\{N(t) \text{ ne change pas de valeur}\} = 1 - g\,\Delta t - N(t)\, m\,\Delta t + o(\Delta t) \ . \end{cases}$$

1. L'espérance du temps de vie est donné par $\frac{1}{m}$.

On en déduit que $p_i(t+\Delta t)$ (pour $i > 0$) s'établit en fonction de $p_{i-1}(t), p_i(t), p_{i+1}(t)$ selon qu'il y ait un mort, aucun changement ou une immigration. L'équation de $p_i(t+\Delta t)$ (pour $i > 0$) est donc :

$$p_i(t + \Delta t) = g\,\Delta t\, p_{i-1}(t) + (1 - g\,\Delta t - i\,m\,\Delta t)\, p_i(t) + (i+1)\, m\,\Delta t\, p_{i+1}(t) \ , \tag{1}$$

$$\text{et on en déduit} \quad \frac{d\,p_i(t)}{dt} = g\,p_{i-1}(t) - (g + i\,m)\,p_i(t) + (i+1)\,m\,p_{i+1}(t) \ , \tag{2}$$

$$\text{avec} \quad \frac{d\,p_0(t)}{dt} = -g\,p_0(t) + m\,p_1(t) \ . \tag{3}$$

Supposons qu'il existe une distribution limite pour la variable $N(t)$ lorsque t tend vers ∞. Si l'on note p^* la distribution limite, la situation à l'équilibre impose $(\frac{d\,p_i^*}{dt} = 0)_{i \in [0,+\infty[}$. Calculons p^* ; de l'équation (3) il vient :

$$p_1^* = \frac{g}{m} p_0^* \ .$$

De l'équation (2) on déduit :

$$p_2^* = \frac{g^2}{2\,m^2} p_0^* \quad \text{puis par récurrence} \quad p_i^* = \frac{g^i}{i!\,m^i} p_0^* \ .$$

En sommant la distribution de probabilité p^*, on obtient nécessairement la valeur 1, donc :

$$1 = \sum_{i \in [0,+\infty[} p_i^* = p_0^* \sum_{i \in [0,+\infty[} \frac{g^i}{i!\,m^i} = p_0^*\, e^{\frac{g}{m}} \quad \text{soit} \quad p_0^* = e^{\frac{-g}{m}} \ .$$

On en tire $p_i^* = \frac{g^i}{i!\,m^i} e^{\frac{-g}{m}}$, et donc p^* est distribuée selon une loi de Poisson de paramètre g/m. On retrouve alors le résultat déterministe lorsque t est grand puisque l'espérance de cette loi est $N^* = g/m$. En reprenant l'équation (1), on a une expression de l'espérance $N^\diamond(t)$ qui est :

$$\begin{aligned}
N^\diamond(t + \Delta t) &= \sum_{i=0}^{\infty} i\, p_i(t+\Delta t) = g\Delta t \sum_{i=1}^{\infty} p_{i-1}(t) + \sum_{i=0}^{\infty} i\, p_i(t) - m\Delta t \sum_{i=0}^{\infty} i\, p_i(t) \ , \\
N^\diamond(t + \Delta t) &= g\Delta t + N^\diamond(t) - m\Delta t\, N^\diamond(t) \ , \\
N^\diamond(t) &= (1 - e^{-m\,t})\, \frac{g}{m} \ .
\end{aligned}$$

La moyenne stochastique équivaut ici au résultat déterministe. Ce n'est pas toujours le cas lorsque l'on adjoint des termes non-linéaires à l'équation différentielle [47] (voir aussi p. 153). La connaissance de la loi de $N(t)$ permet de répondre à certaines questions. Ici par exemple, le calcul à l'équilibre donne ainsi la variance de N^*, la probabilité d'avoir 0 individu dans le système à l'équilibre ou d'en avoir plus de x. Le modèle analytique donne donc des informations qui qualifient la dynamique.

Modèle par chaîne de Markov L'exemple que nous avons abordé aurait pu l'être à l'aide des chaînes de Markov. On considère alors un modèle stochastique discrétisé en temps, pour lequel une variable d'état au pas de temps t dépend linéairement des variables d'état au temps $t-\Delta t$ et de certaines probabilités de passage. Le pas de temps Δt doit être très proche de 0 afin qu'il soit quasi-certain qu'il y aura en Δt au plus une immigration et/ou une mort. Dans ce cadre et pour notre exemple on aurait :

$$p(t+\Delta t) = \underbrace{\begin{bmatrix} (1-g\,\Delta t) & m\,\Delta t & & 0 & \\ g\,\Delta t & (1-(g+m)\,\Delta t) & \ddots & & \\ & g\,\Delta t & \ddots & i\,m\,\Delta t & \\ & & \ddots & (1-(g+i\,m)\,\Delta t) & \\ & 0 & & g\,\Delta t & \end{bmatrix}}_{T} p(t)\ .$$

La matrice de transition T d'un pas de temps à l'autre est ici tridiagonale et infinie. On peut néanmoins travailler avec celle-ci en la tronquant au delà du nombre limite d'individus de la population si cette limite existe. La distribution au temps t se calcule à partir de la distribution initiale $p(0)$ et de la matrice T élevée à une certaine puissance (si $t=x\,\Delta t$) : $p(t) = T^x\,p(0)$. Les techniques spécifiques des chaînes de Markov s'appliquent alors et permettent d'approfondir la compréhension du modèle stochastique.

Modèle par simulation individu-centrée Pour clore cette Section, on peut revoir notre exemple sous l'angle d'un modèle individu-centré (modèle stochastique, discret en temps, simulé informatiquement). Le système comprend des individus que l'on gère indépendamment. Des tirages aléatoires sont utilisés pour faire évoluer chaque individu. On produit R réalisations possibles du système que l'on appelle réplications (*replicates* ou *simulation runs* en anglais). Une seule réplication ne caractérise pas le système car elle est dirigée par le hasard. Par contre, la moyenne pour chaque sortie bornée v_i du système sur toutes les réplications se stabilise lorsque R augmente (par application du théorème de la limite centrale : pour i fixé, les sorties $[(v_i)_r]_{r\in[1,R]}$ sont les échantillons d'une variable aléatoire qui suit une certaine loi). Ce résultat constitue une synthèse de l'évolution temporelle probable du système. Revenons à l'exemple et considérons que l'on ait $N_r(t)$ individus à l'instant t dans la réplication numéro r. Pour déduire l'état du modèle au pas de temps suivant $t+\Delta t$, on procède comme suit :

- pour chaque individu parmi les $N_r(t)$, on effectue un tirage uniforme sur $[0,1]$. Si le tirage est inférieur à $m\,\Delta t$, l'individu meurt et il ne sera plus présent dans le système en $t+\Delta t$;
- un tirage uniforme sur $[0,1]$ est effectué. Si le nombre généré est inférieur à $g\,\Delta t$, un nouvel individu immigrant est incorporé au système.

En fonction des flux entrants et sortants du système, on déduit ainsi le nombre d'individus en $t+\Delta t$ qui sera $N_r(t+\Delta t)$. Si l'on dispose de R réplications numérotées $r\in[1,R]$, on peut déduire la probabilité $p'_i(t)$ d'avoir i individus au temps t dans le système. On a :

$$p'_i(t) = \frac{1}{R} \sum_{r \in [1,R] \ / \ N_r(t)=i} 1 \ .$$

On peut les comparer aux $p_i(t)$ des modèles stochastiques précédents. Lorsque le nombre de réplications augmente, les courbes issues de la simulations tendent bien vers les courbes analytiques. Les probabilités $p'_i(t)$ tendent comme précédemment vers une loi de Poisson quand R est grand (voir Figure 1 à droite). Quoique l'on retrouve les mêmes résultats numériquement, la simulation ne fournit pas directement l'information : " cette distribution suit une loi de Poisson " ; c'est l'utilisateur qui fait l'observation que les distributions convergent vers cette loi lorsque R est grand.

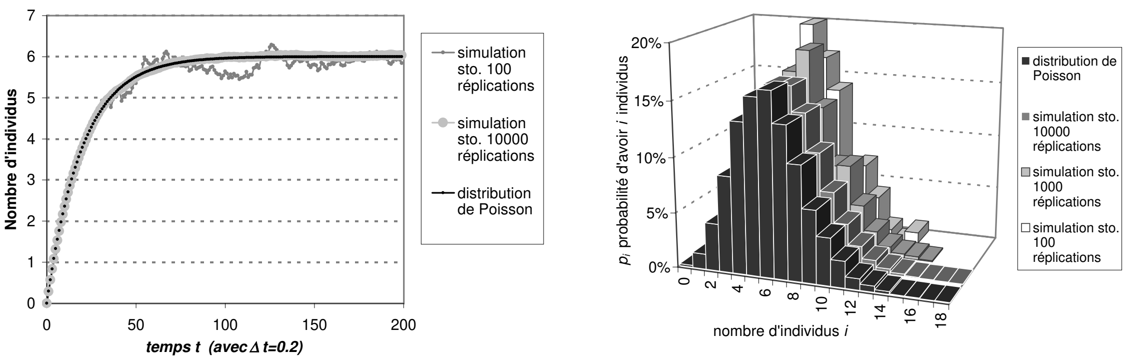


Figure 1. Comparaison de simulations stochastiques (avec 100, 1000 ou 10000 réplications) avec le modèle stochastique analytique. Les paramètres sont $g = 0.3$; $m = 0.05$; $\Delta t = 0.2$; $N(0) = 0$. A gauche, l'évolution du nombre d'individus fonction du pas de temps. A droite, la probabilité d'avoir i individus dans le système pour t grand ($t = 200$).

La méthode est donc moins efficace du point de vue de la facilité d'interprétation. Mis à part cet inconvénient, on peut facilement ajouter au niveau des individus des taux de mortalité différents en fonction du temps, de l'espace et de l'environnement, dans le cas de la simulation individu-centrée. De cette manière on peut facilement incorporer des hétérogénéités, ce qui les distinguent des précédents modèles analytiques.

Conclusion Les modèles analytiques et ceux issus d'une approche de type chaîne de Markov ne suffisent pas pour décrire et expliquer la complexité de toutes les dynamiques de populations. La simulation informatique se révèle donc être un outil de compréhension équivalent à la modélisation théorique et expérimentale. Les principaux éléments pour réaliser une simulation consistent à développer un modèle théorique, à produire des résultats en utilisant un algorithme adéquat, enfin à analyser les sorties.

Plusieurs modélisations mécanistes sont à notre disposition pour aborder un système biologique [16]. Premièrement, les modèles classiques à base d'équations différentielles schématisent des flux entre compartiments correspondant à des sous-populations homogènes du système. Deuxièmement, des modèles en temps discret expriment des probabilités de transitions entre différents états à chaque pas de temps (approches de type chaîne de Markov et matrices de Leslie). Finalement, les modèles individu-centrés représentent l'interaction des individus entre eux et avec l'environnement. Dans ce dernier cas, on ne spécifie pas des relations entre variables d'état macroscopiques parce que la dynamique émerge d'elle-même au cours de la simulation. Les modèles individu-centrés sont bien adaptés pour aborder des systèmes complexes car ils permettent facilement de tenir compte de l'hétérogénéité des caractères individuels et de l'environnement.

1.3 Calcul haute-performance et dynamique de populations

1.3.1 Langages et environnements pour la simulation

Une grande puissance de calcul est souvent nécessaire pour aborder les modèles de systèmes biologiques complexes [38, 39]. Certaines classes de modèles comme les approches de modélisation spatialisée, ou ceux ayant un niveau de détail fin, conduisent à une consommation mémoire et des coûts de calculs importants. Dans le domaine de la simulation par événement discret [23, 55] (en anglais *discrete event simulations*), le parallélisme est fréquemment utilisé pour réduire le temps des simulations. Des environnements généralistes existent pour construire de telles simulations basées sur le concept d'événements discrets, et pour générer une application parallèle. Durant la dernière décennie, il a été proposé dans la littérature plusieurs langages de description de modèles et des environnements intégrés. Ceux-ci permettent de construire et exécuter différents types de simulation en écologie. Certains outils visent les modèles spatialement explicites [11, 18, 42], d'autres les modèles individu-centrés [53], voir les automates cellulaires [44]. Les outils cités utilisent des algorithmes distribués et fonctionnent typiquement sur des machines parallèles ou sur des réseaux de stations. Ils fournissent une interface utilisateur, un outil de contrôle et de gestion pour l'exécution des simulations, et souvent des outils d'analyse des sorties. Néanmoins des modèles spécifiques ne s'intègrent pas obligatoirement dans les cadres proposés par ces outils. Des simulations qui prennent en compte plusieurs types de mécanisme (au sens où nous l'avons défini Section 1.1.2) ou ont besoin de différentes techniques de calcul (stochastique, déterministe) ne peuvent pas être intégrées dans de tels environnements. Dans des cas semblables, on veut une implémentation efficace pour disposer d'un simulateur utilisable, et être en mesure de valider le modèle. La simulation parallèle a prouvé ses qualités indéniables dans le cadre de la modélisation de dynamique de populations à travers les environnements de simulations et les applications développés spécifiquement (voir par exemple [3, 41]). Il reste cependant à développer d'autres techniques pour élargir le spectre des simulations qui tirent profit du parallélisme.

1.3.2 Importance du niveau de détail

La modélisation déterministe est très utilisée en bio-mathématique. Ce type de modélisation est habituellement basé, soit sur un système d'équations différentielles non linéaires, soit sur une discrétisation d'un tel système [54]. Les problèmes traités dans ce cadre reposent généralement sur un nombre restreint de paramètres ; ils ont pour objectif principal de reproduire des dynamiques de populations "théoriques" en exploitant peu de données quantitatives sur les populations. Ces modèles font abstraction de certains processus se déroulant à l'échelle des individus, et ils cherchent à saisir les phénomènes macroscopiques au niveau des populations. Une approche radicalement différente est basée sur les modèles individu-centrés aussi appelés multi-agents [7, 28]. Ces modèles se proposent de reproduire fidèlement des processus élémentaires naturels, mais de ce fait conduisent parfois à des coûts de calcul élevés [21]. Le choix de l'échelle d'étude, par exemple au niveau de l'individu ou au niveau d'un compartiment d'une population structurée, ou encore au niveau d'une population entière, dépend des mécanismes à intégrer, des questions posées au modèle, et d'autres contraintes gérées par le modélisateur. La qualité de réalisme des simulations diffère selon le type d'approche envisagée [8]. Les modèles présentés dans la littérature s'échelonnent sur l'axe du nombre de mécanismes pris en compte, et sur l'axe du nombre de paramètres intégrés. Le coût de simulation dépend fortement de la position du modèle par rapport à ces deux axes.

Des modèles individu-centrés ont été implémentés sur machine parallèle [21], car certains systèmes complexes nécessitent de grandes puissances de calcul disponibles uniquement sur ces machines [41]. On trouve dans la littérature peu de références à des simulateurs réalistes qui utilisent des modèles déterministes en temps discret [40], et encore moins qui sont implantés sur machine parallèle [3].

Chapitre 2

Modélisation de systèmes hôte-macroparasite

Les systèmes hôte-macroparasite comportent une grande diversité de mécanismes. Les modèles déterministes généraux se révèlent insuffisants pour décrire simultanément l'ensemble des systèmes existants. Dans les Sections qui suivent, nous donnons des raisons de la complexité de ces systèmes et des éléments concernant leur modélisation.

Le modèle que nous proposons s'appuie sur les travaux initiaux de Michel Langlais et Patrick Silan [13, 67], qui furent poursuivis par Catherine Bouloux [13, 63]. Le travail de thèse de Catherine Bouloux [12] combine deux approches : tout d'abord un modèle discret en temps est proposée et conduit à un simulateur déterministe, ensuite des versions du modèle continues en temps donnent lieu à une étude mathématique. Notre travail reprend et approfondit le modèle discret pour plus de réalisme, puis l'étend en proposant une simulation individu-centrée.

2.1 Introduction aux systèmes hôte-parasite

On distingue deux grandes classes de parasites fondées en partie sur leurs tailles et sur leurs comportements [1]. En général, les microparasites sont très petits et se reproduisent dans l'hôte sans avoir à en sortir. Des bactéries, virus et de nombreux protozoaires font partie de cette catégorie.

Les microparasites se reproduisent à l'intérieur de leur hôte relativement rapidement. On classe les hôtes en plusieurs catégories : les Susceptibles à l'infection, les hôtes Infectés, les Résistants à l'infection. La sévérité de l'infection n'est pas nécessairement prise en compte car le parasite prolifère dès qu'il a colonisé un hôte susceptible. Les hôtes qui sortent d'une infection peuvent acquérir une immunité totale ou partielle contre une nouvelle infection. La *prévalence* désigne le pourcentage de la population hôte qui est infectée, soit le ratio : I (nombre d'infectés) divisé par $S + I + R$ le nombre d'hôtes Infectés ou Susceptibles ou Résistants. Les microparasites ont été étudiés sous l'angle de la dynamique de populations et de l'épidémiologie (modèles SIR et SEIR par exemple), mais nous ne développerons pas plus avant ce domaine de recherche.

En général, les macroparasites sont plus gros et colonisent un nouvel hôte à chaque nouvelle génération. C'est le cas pour de nombreux helminthes, comme le ténia (cestode), l'ascaris (nématodes), la douve du foie (trématode). Ils sont à l'origine d'infections persistantes, avec des hôtes qui sont continuellement réinfectés. Dans ce cadre, il est essentiel de tenir compte du nombre d'individus présents sur l'hôte [2]. En effet, l'intensité parasitaire peut influer sur la dynamique lorsque certains hôtes sont plus infestés que d'autres. Par exemple, la mortalité ou la fertilité d'un hôte et/ou de ses parasites dépendent éventuellement de l'intensité parasitaire individuelle de l'hôte. Prendre en compte la distribution des parasites sur les hôtes est alors nécessaire dans un modèle du système. En effet, des structures y apparaissent et elles peuvent influer sur la dynamique. Cette distribution est souvent interprétée comme une loi de distribution d'une variable aléatoire, ce qui introduit une composante stochastique dans le modèle.

Nous allons développer quelques aspects des systèmes comprenant des macroparasites. Parmi les macroparasites, on distingue ceux qui ont un cycle indirect (*hétéroxène*) de ceux qui ont un cycle direct (*holoxène*). Durant leur vie, les parasites à cycle indirect colonisent des *hôtes intermédiaires* pour passer d'un stade à l'autre de leur évolution. Leur *cycle* de vie, c'est-à-dire l'ensemble des stades qu'ils traversent, comprend ainsi plusieurs hôtes. Lorsque l'hôte définitif est colonisé, les parasites ont une reproduction sexuée ; leur progéniture doit alors recommencer le cycle entier. Pour les parasites à cycle direct, il y a un hôte final et pas d'hôte intermédiaire. La modélisation des phases où le parasite n'est pas encore en mesure de se reproduire (pas encore fixé sur l'hôte) en est simplifiée.

Pour de nombreux systèmes hôte-parasite, il est capital de comprendre le procédé que les parasites utilisent pour s'introduire chez l'hôte. En effet, cela induit une certaine répartition des intensités parasitaires chez les hôtes. En fonction de cette intensité, les hôtes et les parasites peuvent réagir de façon très différente. L'hôte peut par exemple développer une défense spécifique contre les parasites à partir d'un certain seuil. A partir d'un autre seuil, l'hôte peut aussi s'affaiblir, mourir, ou bien devenir un vecteur très contagieux du parasitisme. D'autre part, les parasites auront plus l'occasion de se reproduire avec des intensités fortes, ou bien n'auront pas la possibilité d'être plus d'un certain nombre par hôte faute de place ou de nourriture. Selon les systèmes hôte-parasite, de nombreux phénomènes intensité-dépendant s'expriment donc ; en conséquence, leur impact sur la dynamique de populations doit être pris en compte. Un ensemble de processus intensité-dépendant conduit fréquemment à l'hétérogénéité des intensités parasitaires de la population hôte [69]. Pour de nombreux macroparasites, l'agrégation contribue à structurer la dynamique de l'infection ; de ce fait, la distribution des parasites sur les hôtes est une information importante dans un modèle du système. De nombreuses études théoriques [29, 49, 50] ont montré que l'agrégation parasitaire conduit à des distributions qui peuvent stabiliser le système hôte-parasite en milieu naturel. Cette thèse contribue

2. cela est aussi réalisable pour les microparasites.

à améliorer la compréhension des mécanismes actifs de régulation dans de nombreux systèmes hôte-parasite.

La régulation d'une population parasitaire consiste en une série de rétroaction positive ou négative qui maintient l'effectif de cette population dans un certain intervalle. On distingue 4 modes principaux de régulation du parasitisme [15]. Premièrement, les parasites colonisent des hôtes sans jamais atteindre un seuil dangereux pour chacun de ces hôtes. Dans ce cas, les flux entrants et sortant des parasites dans l'espace des hôtes sont équilibrés. Deuxièmement, lorsque les stades infestants arrivent sur un hôte, la présence de parasites en trop grand nombre peut limiter le développement des nouveaux arrivés par manque de ressources. Cette régulation correspond à une densité dépendance de type *scramble* (*cf* p. 5). Troisièmement, les parasites déjà présents sur l'hôte peuvent diffuser des substances qui empêchent le développement normal des larves qui viennent d'arriver (régulation de type *contest*). Quatrièmement, le parasitisme peut provoquer au dessus d'un certain niveau la mort de l'hôte et des parasites qu'il porte. Dans cette thèse, nous nous intéresserons plus particulièrement à ce dernier type de régulation par la mort des hôtes surinfestés.

2.2 Distribution des macroparasites sur les hôtes

2.2.1 L'agrégation

2.2.1.1 Loi des puissances

L'agrégation fait référence à une distribution des parasites sur les hôtes qui n'est pas uniforme. Elle est caractérisée par le ratio $\sigma^2/\overline{p}$: variance du nombre de parasites par hôte divisée par la moyenne du nombre de parasites par hôte. Lorsque la variance équivaut à la moyenne, on suppose que la distribution est aléatoire et suit une loi de Poisson. Si la variance est plus grande que la moyenne, la distribution est dite *agrégée* ou *sur-dispersée*, *c.à.d.* que les parasites ont tendance à être placés en paquets sur quelques hôtes. Enfin pour une variance plus faible que la moyenne, la distribution est *sous-dispersée*, et tous les hôtes ont à peu près le même nombre de parasites.

La loi des puissances de Taylor [68] correspond à une relation observée dans ce système entre la moyenne et la variance du type : $log(\sigma^2) \approx log(a) + b\,log(\overline{p})$ (ou bien $\sigma^2 \approx a\,\overline{p}^b$). Ces formules se vérifient pour différents systèmes hôte-macroparasite. Cette "loi" se retrouve dans de nombreux exemples de populations naturelles [60] et les valeurs de b sont généralement comprises entre 1 et 2. Lorsque b vaut 1 la distribution des parasites est du type aléatoire, lorsque b est strictement supérieur à 1, la configuration est dite agrégée.

2.2.1.2 Loi binomiale négative

La distribution des macroparasites sur les hôtes est en général agrégée [19, 20, 60]. Dans la littérature, ceci est souvent représenté par une distribution du nombre de parasites sur les hôtes qui suit une loi binomiale négative de paramètre k. Cette loi de distribution a servi à analyser et à répertorier de nombreux systèmes hôte-parasite en milieu naturel [60]. Pour un nombre moyen de parasites par hôte $\overline{p}$, elle définie la proportion d'hôtes $Prob(i|k,\overline{p})$ portant i parasites par les relations :

$$Prob(i|k,\overline{p}) = \frac{(k+i-1)!}{i!(k-1)!}\,(\frac{\overline{p}}{k})^i\,(1+\frac{\overline{p}}{k})^{-k-i} \tag{4}$$

$$\text{cela sécrit aussi} \quad Prob(0|k,\overline{p}) = (1+\frac{\overline{p}}{k})^{-k}$$

$$\text{et} \quad Prob(i{+}1|k,\overline{p}) = Prob(i|k,\overline{p})\,\frac{\overline{p}\,(k{+}i)}{k\,(i+1)(1+\frac{\overline{p}}{k})}\ . \tag{5}$$

La variance d'une telle loi est $\sigma^2 = \overline{p} + \frac{\overline{p}^2}{k}$. Sur la Figure 2 sont présentées quelques distributions qui suivent la loi binomiale négative.

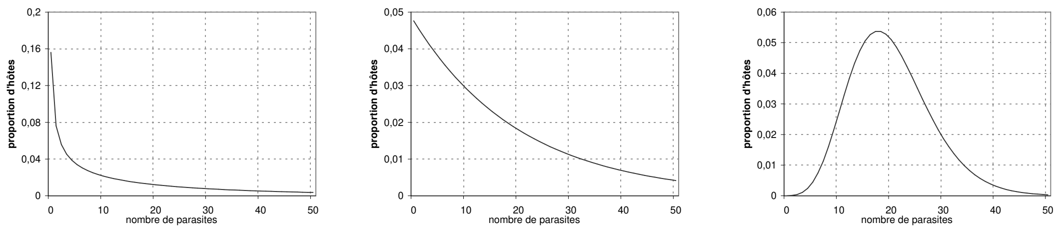


Figure 2. Exemples de distributions binomiales négatives en fonction de k ; on a $\overline{p} = 20$ et de la gauche vers la droite $k = 0.5$; $k = 1$; $k = 5$.

Il est possible de construire un modèle simple conduisant à cette distribution de macroparasites parmi les hôtes. Prenons un taux d'infection aléatoire g, une mortalité des parasites de m, et enfin un taux r de reproduction des parasites au sein de l'hôte (on considère le cas $m > r$). On a le même type de modèle stochastique qu'en Section I-1.2.2.2, p. 10. De manière similaire, on aboutit aux deux équations suivantes

$$\frac{d\,p_i(t)}{dt} = [g + r\,(i{-}1)]\,p_{i-1}(t) - (g{+}i\,m{+}i\,r)\,p_i(t) + (i{+}1)\,m\,p_{i+1}(t) \tag{6}$$

$$\text{avec} \quad \frac{d\,p_0(t)}{dt} = -g\,p_0(t) + m\,p_1(t)\ . \tag{7}$$

Cherchons une distribution limite lorsque t tend vers $+\infty$. Soit p^* la distribution limite, à l'équilibre on a $(\frac{d\,p_i^*}{dt} = 0)_{i\in[0,+\infty[}$. L'équation (7) implique $p_1^* = \frac{g}{m}p_0^*$. Supposons l'hypothèse de récurrence $HR(i)$: $p_i^* = p_{i-1}^*\,\frac{g+r\,(i-1)}{i\,m}$. L'équation (7) donne :

$$0 = [g + r\,(i{-}1)]\,\frac{i\,m}{g{+}r\,(i{-}1)}\,p_i^* - (g{+}i\,m{+}i\,r)\,p_i^* + (i{+}1)\,m\,p_{i+1}^*$$

$$\text{soit} \quad 0 = -(g{+}i\,r)\,p_i^* + (i{+}1)\,m\,p_{i+1}^* \qquad \text{d'où} \qquad p_{i+1}^* = p_i^*\,\frac{g{+}r\,i}{(i{+}1)\,m}\ .$$

L'hypothèse est vérifiée en $i+1$, et on en déduit qu'elle est vraie pour tout i. Prenons maintenant la loi binomiale négative de paramètres $\overline{p} = g/(m-r)$ et $k = g/r$; on a d'après (5)

$$Prob(i+1|g,m,r) = Prob(i|\frac{g}{r},\frac{g}{m-r})\,\frac{\frac{g}{m-r}\,(\frac{g}{r}+i)}{\frac{g}{r}\,(i+1)(1+\frac{g/(m-r)}{g/r})}$$

$$\text{soit}\quad Prob(i+1|g,m,r) = Prob(i|\frac{g}{r},\frac{g}{m-r})\,\frac{(g+r\,i)}{(i+1)\,m}\ .$$

On peut donc identifier p_i^* à cette loi de probabilité binomiale négative. Un modèle élémentaire suffit donc pour reproduire une loi binomiale négative. Dans ce cadre, il est même possible de calculer la prévalence ($p_{rev}^* = 1 - p_0^*$), soit $p_{rev}^* = 1 - (1 + \frac{r}{m-r})^{-g/r}$.

2.2.2 Comment gérer la distribution ?

Une des questions posée aux modélisateurs est de savoir comment représenter la distribution des parasites sur les hôtes. Parmi les stratégies employées pour simplifier l'analyse des modèles hôte-macroparasite, on a vu se développer des modèles hybrides [56] utilisant des techniques déterministes et stochastiques. Une approche productive a consisté à prendre un modèle déterministe pour la dynamique des hôtes tout en considérant au niveau des parasites une loi de distribution des intensités parasitaires parmi les hôtes [26]. Des modèles déterministes tenant compte de cette distribution considèrent la donnée $N(l,t)$ qui donne le nombre d'hôte ayant l parasites au temps t. La population des hôtes se trouve ainsi spatialisée selon la variable uni-dimensionnelle l. On se trouve donc à gérer une infinité de variables d'état $N(l,t)$ ($l \in [0,+\infty]$), chacune régie par une équation différentielle [19, 20, 31]. Ces modèles se sont révélés insuffisants pour une étude analytique [48] si l'on n'intègre pas plus explicitement la forme de la distribution en fréquence, ce qui induit des hypothèses supplémentaires. De nombreux modèles ont été formulés, intégrant ou non des distributions stochastiques. En appliquant ces modèles à des systèmes hôte-parasite particuliers, les modélisateurs ont souvent considéré que la forme de la distribution reste figée au cours du temps. Typiquement la forme de la distribution est choisie de type loi de Poisson, loi binomiale négative, ou loi binomiale positive. Il est alors impossible de laisser le système évoluer indépendamment par le jeu du recrutement des parasites sur les hôtes [6, 25], puis de constater après coup la forme de la distribution des parasites sur les hôtes.

Un modèle déterministe *discret* utilisant la simulation a l'avantage de pouvoir reproduire les mécanismes du système sans préjuger de la forme de la distribution [63]. Ici, on se contente de simuler par pas de temps successifs jusqu'au temps t, et de constater les valeurs des variables d'état. Dans ce cadre, des distributions de fréquence sont issues de mécanismes de recrutement et de mortalité qui peuvent être déterministes et/ou stochastiques. Le modèle peut intégrer un recrutement sur les hôtes intensité-dépendant. Cet élément est essentiel car on peut ainsi traduire les contraintes liées au recrutement différentiel des parasites et à l'impact que ce recrutement aura sur la fécondité et la mortalité de l'hôte et du parasite. L'un des résultats de cette thèse est que la forme de la distribution, qui évolue au cours du temps, joue un rôle prépondérant dans la dynamique. Par contre pour ce type de modèle discret, un problème de mise en œuvre informatique se pose car la quantité de calcul pour la simulation peut être importante [32, 36].

Une autre approche, via une simulation de type Monte-Carlo, constitue une alternative aux modèles basés sur une prise en compte de la distribution. Grace à une simulation individu-centrée, on obtient en effet la distribution sans la gérer explicitement [37]. Les techniques utilisées et les perspectives nouvelles offertes par la modélisation déterministe discrète et la simulation individu-centrée sont présentées dans cette thèse.

2.3 Deux modèles pour simuler un système hôte-macroparasite

Au cours de cette Section, nous développons les calculs nécessaires pour gérer la distribution de parasites sur les hôtes dans deux cadres : premièrement pour le modèle déterministe discret qui sera décrit dans la prochaine Section, et deuxièmement pour le modèle individu-centré approfondi dans la Section 2.3.2. On décrira l'interaction directe entre les parasites fixés et l'hôte final sans détailler les phases intermédiaires du développement du parasite (stade larvaire, hôte intermédiaire ...) qui sont spécifiques au système considéré. Plusieurs hypothèses fondamentales sont faites dans nos modèles :

– les parasites fixés sur les hôtes sont structurés en K classes d'âge numérotées de 1 à K. La classe d'âge K représente la classe des adultes capables de se reproduire. A chaque pas de temps, les parasites vont d'une classe d'âge à la suivante avec un certain taux de survie, sauf pour la classe des adultes où les parasites restent dans la même classe en subissant un taux de mortalité. La première classe d'âge est réalimentée ou non à chaque pas de temps par le recrutement de larves de parasites par les hôtes. La durée de résidence d'un parasite dans une classe d'âge suit une loi exponentielle ;
– les hôtes ne se reproduisent pas et la seule cause de mortalité possible ici est le parasitisme. Le taux de survie d'un hôte en t et durant un pas de temps est $p(l)$, l étant l'intensité parasitaire ; ce taux peut donc être intensité-dépendant.

La deuxième hypothèse est restrictive et relève d'un choix de modélisation [26] ; cela permet de se focaliser sur des dynamiques plus simples. La démographie des hôtes pourra être prise en compte lorsque les modèles seront bien compris.

On pose les notations suivantes :

- Δt : le pas de temps de discrétisation ;
- $\varphi(j, l, t)$: la probabilité que j larves soient recrutées par un hôte ayant l parasites au temps t. Cette probabilité tient compte du mécanisme par lequel les larves colonisent l'hôte (intensité-dépendance éventuelle), et du nombre de larves présentes dans le milieu ;

- $\delta_k(u,l)$: probabilité que u parasites d'âge k meurent parmi l parasites durant le pas de temps Δt.

Ces éléments peuvent servir pour de nombreux systèmes hôte-macroparasite. Le détail des mécanismes actifs qui mènent à l'agrégation est peu souvent bien connu, surtout au niveau quantitatif. Par contre, on est en mesure d'identifier s'il y a une hétérogénéité du recrutement ou non. On pourra alors estimer une fonction $\varphi(j,l,t)$ pour en tenir compte.

2.3.1 Modèle déterministe discret

Pour le modèle déterministe discret, on considère la distribution des parasites sur les hôtes. Pour cela, on définit :

- $N(l,t)$: nombre d'hôtes ayant l parasites au temps t. La variable $N(.,t)$ correspond à la distribution des parasites sur les hôtes ;
- $N_k(i,l,t)$: nombre d'hôtes ayant l parasites, i étant plus vieux que $k\,\Delta t$. Il s'agit là d'une distribution sur les hôtes qui tient compte de la structure en âge des parasites.
- $\forall k \in [1,K]$ on a $\delta_k(u,l) = \delta(u,l)$, la mortalité des parasites ne dépend pas de l'âge $k\,\Delta t$.

Le modèle est déterministe avec des composantes stochastiques puisqu'il utilise des lois de probabilités pour les fonctions φ et δ. En temps discret, la question qui se pose maintenant est de savoir comment calculer l'état du modèle à l'instant $t+\Delta t$ en fonction de son état au temps t. Les formules qui suivent permettent d'effectuer cette opération. Ce modèle est issu des articles de Bouloux, Langlais et Silan [12, 13, 34, 63].

2.3.1.1 Mise à jour de $N(l,t)$

Considérons un hôte avec l parasites au temps $t+\Delta t$. Durant l'intervalle de temps de t à $t+\Delta t$, il peut avoir recruté j larves alors que u parasites mourraient de mort naturelle (voir Figure 3). Donc au pas de temps précédant t, cet hôte avait $l+u-j$ parasites avec $0 \leq j \leq l$ parce qu'aucune des j larves n'est morte durant la phase de recrutement ; de plus, l'hôte a survécu à une intensité de $l+u-j$ parasites. Finalement, on a $\forall l \in [0,+\infty[$:

$$N(l,t+\Delta t)=\sum_{j=0}^{l}\sum_{u=0}^{+\infty} N(l+u-j,t)\,\varphi(j,l+u-j,t)\,\delta(u,l+u-j)\,p(l+u-j) \ . \tag{8}$$

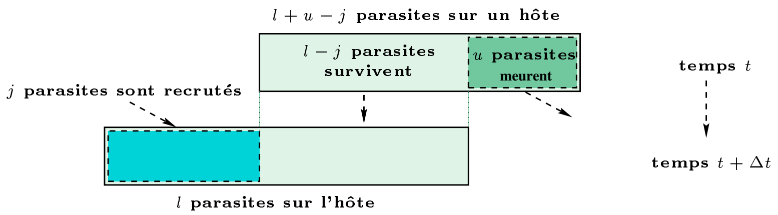


Figure 3. Recrutement et mortalité des parasites sur un hôte (mise à jour de N)

2.3.1.2 Mise à jour de $N_1(i,l,t)$

Considérons un hôte ayant l parasites parmi lesquels i sont plus vieux que Δt. Au pas de temps t cet hôte avait au moins i parasites à cause du processus de vieillissement, c'est à dire $i+u$ parasites ; entre t et $t+\Delta t$, u parasites sont décédés de mort naturelle alors que $l-i$ étaient recrutés. De plus, l'hôte doit survivre à une charge de $i+u$ parasites (voir Figure 4). On en déduit que $\forall l \in [0,+\infty[, \forall i \in [0,l]$:

$$N_1(i,l,t+\Delta t) \;=\; \sum_{u=0}^{+\infty} N(i+u,t)\,\varphi(l-i,i+u,t)\,\delta(u,i+u)\,p(i+u)\;. \tag{9}$$

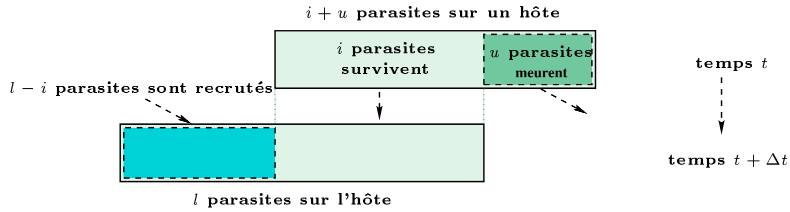


Figure 4. Recrutement et mortalité des parasites sur un hôte (mise à jour de N_1)

2.3.1.3 Mise à jour de $N_k(i,l,t)$

Considérons maintenant, en $t+\Delta t$ un hôte ayant l parasites parmi lesquels i sont plus vieux que $k\Delta t$, $2 \le k \le K$ (voir Figure 5). Au temps t cet hôte avait au moins i parasites plus vieux que $(k-1)\,\Delta t$, c'est à dire $i+\nu$ parmi lesquels ν sont morts de mort naturelle entre t et $t+\Delta t$; la charge totale de parasites au temps t était $m+u$ parmi lesquels u meurent, $0 \le \nu \le u$, ce qui conduit à un recrutement de $l-m$ larves. Enfin l'hôte survit à une charge de $m+u$ parasites (illustré dans la Figure 5). On a donc (pour $2 \le k \le K,\;\; 0 \le l \le +\infty,\;\; 0 \le i \le l$)

$$\begin{aligned} N_k(i,l,t+\Delta t) = \sum_{m=i}^{l}\sum_{u=0}^{+\infty}\sum_{v=0}^{u} \; & N_{k-1}(i+v,m+u,t)\,\varphi(l-m,m+u,t) \\ & \delta(v,i+v)\,\delta(u-v,m+u-i-v)\,p(m+u)\quad . \end{aligned} \tag{10}$$

Les trois formules de mise à jour qui précèdent constituent les lois d'évolution du modèle dans le temps lors d'une simulation. *Elles se substituent au système infini d'équations aux dérivées partielles d'un modèle déterministe en temps continu [12].* La réalisation d'un simulateur de ce modèle pour un système hôte-parasite donné nécessite d'implanter des algorithmes qui calculent les formules (8,9,10) de mise à jour énoncées. Cette thèse donne des techniques parallèles et performantes pour mener à bien cet objectif.

2.3.1.4 Evolution du modèle précédent, contribution

Le modèle déterministe présenté a fait l'objet d'une première étude dans [12]. Le simulateur séquentiel utilisé alors avait recours à des méthodes de réduction des calculs consistant à omettre la réalisation de certaines sommes dont le résultat était jugé

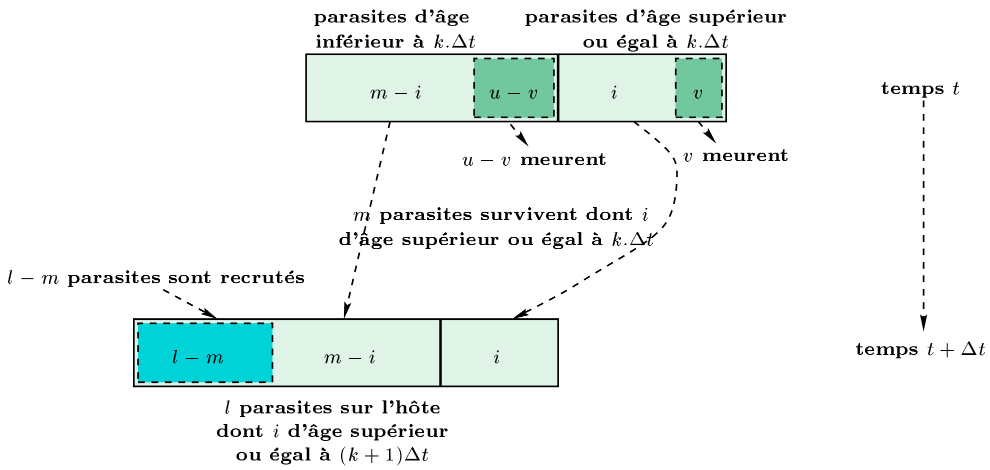


Figure 5. Recrutement et mortalité des parasites structurés en âge sur un hôte (mise à jour de N_k)

négligeable. Ceci introduisait des biais au niveau des sorties. Le modèle se révèle en effet sensible aux perturbations numériques (voir Section III-6.2.7 p. 81). D'autre part, la simulation séquentielle demandait des temps de calculs importants. Les temps d'exécution pouvaient atteindre plusieurs semaines à un mois pour une seule simulation sur une station de travail. Dans ces conditions, l'outil ne pouvait pas servir à explorer complètement le modèle, ni à le valider. Les techniques du parallélisme vont être utilisées pour obtenir un outil efficace d'étude et d'exploration. Les gains en performance du simulateur donneront l'opportunité de comparer les sorties pour une gamme étendue de paramètres en entrée.

2.3.2 Modèle basé sur une simulation de Monte-Carlo

Le modèle basé sur une simulation de Monte-Carlo prend en compte chaque hôte individuellement et les parasites présents sur celui-ci. Nous l'avons présenté initialement dans les articles [33, 37]. Pour cela, on définit les notations suivantes :

- $\mathbb{H}_i$: désigne l'hôte numéro i ;
- $P_i(k,t)$: nombre de parasites d'âge $k\,\Delta t$ sur l'hôte $\mathbb{H}_i$ entre t et $t{+}\Delta t$ ($\forall k \in [0, K{-}1]$) ;
- $P_i(K,t)$: nombre de parasites en âge de procréer sur l'hôte $\mathbb{H}_i$ entre t et $t{+}\Delta t$, *i.e.* d'âge supérieur ou égal à $K\,\Delta t$;
- $P_i(t)$: nombre total de parasites sur l'hôte $\mathbb{H}_i$ entre t et $t{+}\Delta t$;
- $VH_i(t)$: est vrai si l'hôte $\mathbb{H}_i$ est vivant entre t et $t{+}\Delta t$, faux sinon ;
- $MP_i(k,t)$: nombre de parasites qui meurent parmi ceux de la classe d'âge k sur l'hôte $\mathbb{H}_i$ entre t et $t{+}\Delta t$ ($\forall k \in [0, K]$) ;
- $R_i(t)$: nombre de larves recrutées sur l'hôte $\mathbb{H}_i$ entre t et $t{+}\Delta t$.

A chaque pas de temps t, on procède à des tirages aléatoires selon des lois énoncées ci-après pour déterminer les valeurs des variables $VH_i(t)$, $MP_i(k,t)$, $R_i(t)$. Ensuite, les autres variables sont mises à jour pour le pas de temps t en fonction des valeurs trouvées.

La probabilité de survie d'un hôte entre t et $t+\Delta t$ est donnée par $p(P_i(t-\Delta t))$. Un tirage aléatoire permet donc pour chaque hôte vivant de fixer la valeur de $VH_i(t)$. Le nombre u de parasites morts, issus d'une certaine classe d'âge $k \in [2, K]$, sur l'hôte i au pas de temps t suit la loi $\delta_k(u, P_i(k, t-\Delta t))$. Pour un couple (k, i) donné, un tirage aléatoire selon cette loi fournit le nombre de parasites morts $MP_i(k, t)$ dans la classe d'âge k pour l'hôte i. En ce qui concerne le recrutement, il s'agit de répartir $L(t)$ larves sur les hôtes.

$$L(t) = \sum_{j \in [0,+\infty[} \sum_{l \in [0,+\infty[} j\, \varphi(j, l, t)$$

Pour chaque larve qui se fixe en t, tout hôte vivant $\mathbb{H}_i$ a pour espérance $e_i(t)$ de la recruter.

$$e_i(t) = \sum_{j \in [0,+\infty[} j\, \varphi(j, P_i(t), t)/L(t)$$

Les indices $i_1, i_2.., i_{H(t)}$ désignent les hôtes vivants au temps t. La loi de probabilité la plus simple s'adaptant à ce problème consiste à dire que les recrutements sur les hôtes vivants suivent la loi multinomiale $\mathcal{B}(L(t); e_{i_1}(t), e_{i_1}(t), .., e_{i_{H(t)}}(t))$. Les valeurs de $R_i(t)$ sont obtenues en utilisant une séquence de tirages aléatoires. On a finalement les formules suivantes ($\forall i$ tel que VH_i) :

$$\begin{cases} P_i(0,t) = R_i(t) \\ \forall k \in [2, K-1], P_i(k,t) = P_i(k-1, t-\Delta t) - MP_i(k-1,t) \\ P_i(K,t) = P_i(K-1, t-\Delta t) + P_i(K, t-\Delta t) - MP_i(K-1,t) - MP_i(K,t) \\ P_i(t) = \sum_{k \in [0,K]} P_i(k,t) \end{cases}$$

La simulation de Monte-Carlo procède à plusieurs réplications dont on fait la moyenne ; comme on l'a déjà vu celle-ci admet une limite. Les techniques mises en œuvre pour calculer la simulation de Monte-Carlo sont présentées dans le chapitre III-7.

2.4 Enjeux et contribution

Ce travail met en œuvre des compétences scientifiques complémentaires pour une recherche interdisciplinaire dans les domaines de l'informatique hautes performances et de la modélisation biomathématique. Un des objectifs visés est d'analyser et de résoudre efficacement des problèmes de calcul scientifique provenant d'applications complexes qui nécessitent des puissances de calcul téraflopiques, voire au delà du petaflop. La résolution effective de cette gamme de problèmes est un véritable challenge qui nécessite une synergie concernant : en mathématiques appliquées le domaine de la modélisation, et en informatique les domaines de l'algorithmique parallèle et du savoir-faire de la mise en œuvre de codes hautes performances sur diverses plates-formes de calcul. Il s'agit dans ce cadre de contribuer à toutes les étapes de la chaîne qui va de la conception du modèle à la mise en œuvre optimisée et efficace de codes de simulation pour des applications de très grande taille. Cette chaîne contient aussi la partie interprétation des résultats passant par une analyse du rôle des paramètres dans le modèle hôte-macroparasite.

La complexité de certains systèmes naturels (biologiques, écologiques) représente un défi à l'entendement. De nombreuses informations doivent être intégrées pour obtenir une représentation maîtrisable de multiples mécanismes interagissant à différentes échelles. Cette modélisation multi-échelle donne lieu à des couplages fins entre les mécanismes qui sont impliqués. Les modèles analytiques, qui considèrent un nombre restreint de processus pour modéliser de tels systèmes, ne peuvent généralement pas appréhender la richesse des dynamiques potentielles. Ceci rend la simulation incontournable. L'objectif que l'on vise est d'effectuer la synthèse d'événements interactifs pour expliquer des comportements macroscopiques émergents. Les simulations informatiques rendues possibles par les hautes performances des ordinateurs et les outils mathématiques, permettent l'exploration de modèles de tels systèmes. Ensuite, là ou l'expérimentation réelle n'est pas possible, ils constituent un moyen pour valider de tels modèles.

La grande variabilité et la richesse des systèmes hôte-macroparasite fait l'objet de nombreuses études pratiques et théoriques. Des tests d'hypothèses sont réalisées concernant les hôtes, l'environnement, les parasites et leurs stratégies de rencontre avec les hôtes. Les modèles déterministes généraux se révèlent insuffisants pour décrire l'ensemble des cas de figures existants. Des travaux dans ce domaine sont d'autant plus nécessaires que des problèmes de pathologie parasitaire et d'épidémiologie sont liés aux différentes formes d'élevages en environnement marin. De plus, ils contribuent à améliorer la gestion et le contrôle des ressources vivantes. La deuxième partie de la thèse décrit un modèle hôte-macroparasite discrétisé en temps qui se décline en deux variantes, l'une déterministe, l'autre individu-centrée. Le modèle individu-centré (ou stochastique) apporte des informations non disponibles dans le cas déterministe : le spectre des variables de sortie, de nouvelles sorties peu accessibles auparavant, des phénomènes particuliers liés à l'introduction de la stochasticité. En outre, les simulateurs peuvent être comparés afin de détecter les différences issues des modèles (et non du système réel) ou d'éprouver la robustesse des logiciels. Les deux modèles sont nécessaires, les écarts entre les deux simulateurs nous donnent de informations précieuses sur les phénomènes qui relèvent exclusivement des modèles et non du système réel.

Des environnements et langages de simulation généralistes intègrent le parallélisme et permettent au modélisateur d'obtenir des résultats pour des modèles simples ou classiques. Néanmoins le développement de techniques parallèles sont nécessaires pour certains modèles originaux. Dans cette thèse, des mises en œuvre parallèles de deux simulateurs déterministe, et individu-centré sont présentés. On abordera des problèmes de très grande taille requérant des techniques de calcul haute-performance. Nous chercherons une solution algorithmique dédiée pour une très grande plate-forme (plusieurs centaines de processeurs). Dans la troisième partie de ce travail, les algorithmes de deux simulateurs déterministe et individu-centré sont exposés, ainsi que leur adéquation avec les architectures de grappes de nœuds SMP. Nous avons eu l'opportunité d'évaluer la qualité du simulateur sur des configurations ayant jusqu'à 448 processeurs (IBM SP3 NH2), 169 processeurs (IBM SP2) et sur les machines SGI Origin 3800 et IBM Regatta. Une analyse en profondeur du code, ainsi qu'une modélisation mathématique de la répartition des charges ont conduit à des performances de qualité, par exemple une efficacité relative de 77 % pour le simulateur déterministe sur 448 processeurs (SP3). Les calculs sont beaucoup plus précis que dans le simulateur séquentiel précédent et des approximations ont été supprimées. Des temps d'exécution courts permettent désormais d'effectuer de nombreuses simulations. La robustesse du programme a également été éprouvée.

La quatrième partie met en évidence les dynamiques obtenues à l'aide des simulateurs. Des comportements dynamiques nouveaux, correspondant à des cas observés *in situ*, sont mis en évidence. A titre d'exemple, une régulation provisoire de l'épizootie[3] par la mortalité des hôtes est observée dans certaines simulations. On voit aussi que l'allure de la distribution des parasites sur les hôtes n'est pas figée, et qu'elle est couplée à la dynamique du système. Du point de vue de l'écologie, un objectif de ce travail est de découvrir l'importance relative de chacun des mécanismes biologiques qui interviennent dans les systèmes hôte-macroparasite via la simulation. En faisant varier les entrées, nous évaluons donc quels sont les rôles des paramètres de la simulation. D'autre part, les résultats de simulation mettent en évidence certaines limitations du modèle. L'étude approfondie des sorties numériques des simulateurs conduit à modifier le modèle pour améliorer la qualité de réalisme. Notamment, des ajustements successifs suppriment des artéfacts numériques. Nous regardons enfin la sensibilité du modèle considéré, ce qui nécessite un grand nombre de simulations.

Le lecteur intéressé par les aspects algorithmiques ou ceux touchant au parallélisme pourra sans attendre passer à la troisième partie. Les éléments fondamentaux des modèles ont en effet déjà été présenté précédemment. De la même manière, la partie numéro 3 pourra tout à fait être évitée par le modélisateur ou l'écologue sans que cela nuise à sa compréhension.

Ce travail est issu d'une collaboration interdisciplinaire: la dynamique de populations avec Patrick Silan (UPR CNRS de Sète), les mathématiques appliquées avec Michel Langlais (MAB de l'Université Bordeaux 2), et enfin Jean Roman pour l'informatique (LaBRI de l'Université Bordeaux 1). Ces recherches ont été supportées par le programme "Environnement, Vie et Sociétés", le GDR "Architecture, Réseaux et système, et Parallélisme" (thème iHPerf) du CNRS, et puis l'action bioinformatique inter-EPST. Finalement, cette thèse s'incrit dans le projet ScAlApplix de l'unité de recherche INRIA Futurs.

3. processus infectieux qui se développe dans une population animale.

Partie II

Modèle d'un système hôte-macroparasite

Chapitre 3

Cadre biologique

Nous nous intéressons dans cette partie à la parasitose de téléostéens par des ecto-parasites [4]. Nous allons voir premièrement quel est le contexte biologique de ce système hôte-macroparasite [12, 34, 35, 61], puis nous décrirons le modèle biomathématique utilisé.

3.1 Hôte en situation d'élevage

Nous allons étudié le modèle hôte-macgroparasite à travers un cas particulier. L'hôte que l'on considère est le bar, *Dicentrarchus labrax* (ou loup de mer) et le parasite est *Diplectanum aequans* (groupe des *Monogènes*). Essentiellement, on s'intéresse au début de vie de l'hôte dans une situation d'élevage. Les œufs de bars sont générallement recueillis en janvier et février. Les alevins se développent ensuite dans une écloserie, de mars à juin (voir Figure 6), sans être parasités. Dans certains élevages, les jeunes poissons sont placés en juin dans des bassins de pré-grossissement (*race-ways* en anglais) près d'un étang ou en bord de mer. Le modèle décrit le développement d'un macroparasite durant l'année que passent les jeunes bars en bassin de prégrossissement. La mortalité naturelle des hôtes est négligée [5], et l'on considère ici que la mortalité est uniquement due à une charge parasitaire importante. On ne tient pas compte de la reproduction des jeunes hôtes à cause de la situation en élevage qui sert de motivation à l'étude.

De l'eau est prélevée dans l'étang pour renouveler celle des bassins. Or, depuis la fin du printemps jusqu'à l'automne, des bars sauvages sont présents à proximité des lieux de l'élevage. On peut avancer une raison pour expliquer cela : l'environnement y est plus favorable, la nourriture est disponible en plus grande quantité (d'autant plus que celle dispensée aux poissons des bassins se répand autour de celui-ci). Les poissons sauvages sont parasités et sont à l'origine de la présence de larves nageantes de parasite dans le milieu. Ces très petites larves sont introduites via le pompage de l'eau dans les *race-ways*. Elles y contaminent les poissons. Les bars d'élevage sont soustraits des bassins au

4. se dit d'un parasite externe (puce, punaise des lits).
5. bien que d'autres sources de mortalité existent néanmoins.

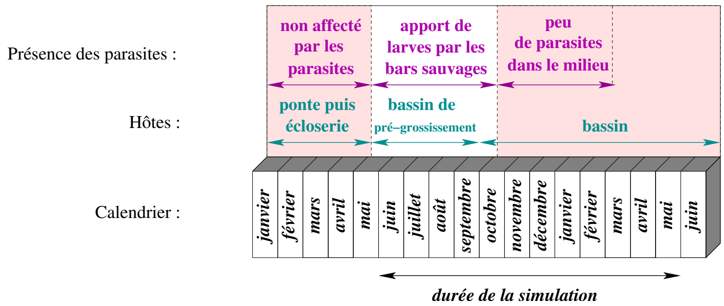


Figure 6. Elevage du bar

printemps de l'année suivante. Sur une première période, le système étudié comporte en entrée un apport en larves permanent. Dans une seconde période, l'épizootie se développe sans apport extérieur de larves (voir Figure 6). Lorsqu'elles ont lieu, les morts massives d'hôtes dues à l'épizootie surviennent habituellement durant les mois de janvier, février, mars. Des traitements chimiques existent mais souvent ils nuisent à la santé des poissons, ou bien ils affectent directement la qualité commerciale du produit final.

3.2 Le parasite

Le parasite étudié est *Diplectanum aequans*, ectoparasite des branchies du bar. Ce macroparasite appartient à l'embranchement des Helminthes (vers), au sous-embranchement des Plathelminthes (vers plats) et à la classe des Monogènes[6][12]. Il est ovipare et ses œufs une fois pondus se dispersent dans le milieu aquatique. Leur temps d'incubation dépend de la température de l'eau. Les larves nageantes *Oncomiracidium*, issues des œufs, cherchent à rejoindre un poisson qui leur servira d'hôte. Si cet événement arrive, la larve se déplace sur le poisson jusqu'aux branchies. Elle perd alors la capacité de nage et se fixe sur un arc branchial (elle garde néanmoins la possibilité de se déplacer sur cet arc).

Ce parasite est hermaphrodite protandre (dans un premier temps, lorsqu'ils sont jeunes, ils sont mâles, une fois adultes ils sont à la fois mâles et femelles). Un accouplement croisé est néanmoins nécessaire pour assurer une descendance. La vie de ces parasites se décompose en plusieurs phases qui constituent un cycle direct[7] (Figure 7). Sur cette figure, trois entités interagissent entre elles : l'ensemble des œufs, les larves nageantes, l'ensemble des parasites. A l'étape numérotée 2, des œufs n'éclosent pas et disparaissent du système. De même, aux étapes 5 et 7 des larves et des parasites meurent. Les étapes 6 et 7 sont les plus complexes et le modèle les décrit finement, comme nous le verrons.

6. Plathelminthes qui sont des parasites de vertébrés aquatiques.
7. *cf* p. 16.

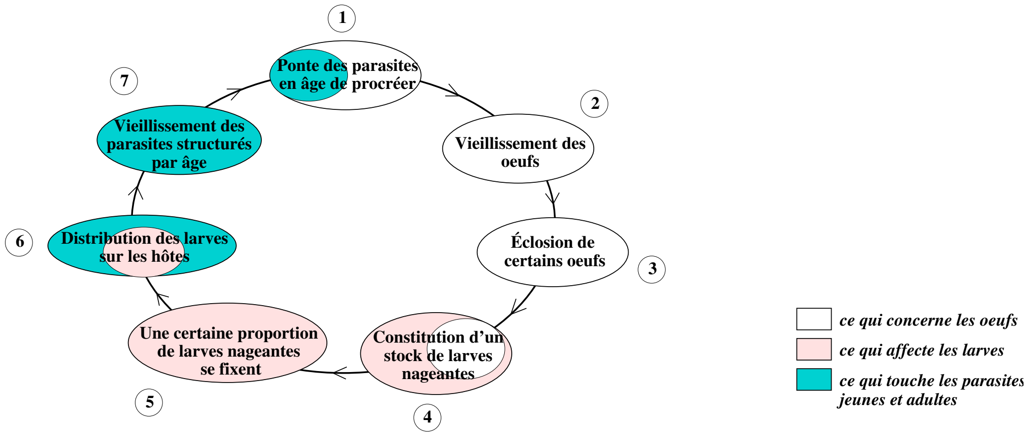


Figure 7. Cycle du développement parasitaire.

3.3 Distribution des larves sur les hôtes

Les larves nageantes ne se fixent pas de manière homogène dans la population d'hôtes. Certains hôtes (ceux qui sont déjà très atteints par le parasitisme) recrutent beaucoup plus de larves que les autres, on parle de phénomène de surdispersion ou agrégatif. En effet, les poissons les plus parasités vont être les cibles privilégiées des larves pour plusieurs raisons. Premièrement, les hôtes fortement touchés par le parasitisme sont affaiblis et se déplacent moins rapidement. Les larves cherchant activement un hôte, colonisent avec plus de facilité ce type d'hôte. Deuxièmement, les hôtes affaiblis restent la plupart du temps au fond des bassins où les larves sont en concentration plus élevée. Ce facteur augmente encore la probabilité de rencontre entre les larves et ces hôtes. Mais dans ce système hôte-macroparasite comme pour d'autres, il y a de nombreuses causes qui peuvent favoriser l'hétérogénéité des hôtes face au parasitisme. La propagation de l'épizootie dépend entre autre de la manière de modéliser ce mécanisme. Ceci est d'autant plus important que l'agrégation est un phénomène courant dans les systèmes hôte-macroparasite ; il résulte fréquemment de processus intensité-dépendant.

Chapitre 4

Modèle déterministe discret d'un système Bar-Monogène

4.1 Introduction

4.1.1 Fonctionnement global du modèle

Après avoir pris connaissance du contexte, intéressons-nous au fonctionnement global du modèle. Nous étudierons ce dernier à travers la simulation de la dynamique d'un système hôte-macroparasite dans un bassin de prégrossissement sur une année entière (à partir de juin, voir Figure 6). Les informations produites par la simulation du modèle sont par exemple le nombre d'hôtes en vie, le nombre total de parasites sur les poissons, le nombre de larves nageantes, et surtout la répartition des parasites (par classe d'âge) sur les poissons. Pour réaliser la simulation, on avance par pas de temps successifs sur une année, en remettant à jour les données à chaque fois. Le pas de cette boucle dans le programme est $\Delta t = 2$ jours, durée qui correspond à l'événement le plus court du modèle, à savoir l'espérance de vie d'une larve nageante (*Oncomiracidium*). Pour effectuer la mise à jour des données, on parcourt les sept étapes de la Figure 7. Ces étapes tiennent compte :

- des facteurs abiotiques (température de l'eau $\theta(t)$, arrivée saisonnière de larves),
- de la population des hôtes (à la mort d'un hôte, les parasites disparaissent avec lui ce qui modifie les données qui concernent les parasites ; d'autre part un excès de parasites sur un hôte peut le faire mourir).

On parcourt 183 pas de temps pour une observation sur 366 jours. Ce que l'on obtient d'une simulation, ce sont les valeurs des différentes variables internes à chaque pas de temps. Le simulateur dispose de très nombreux paramètres (les données biologiques concernant les parasites sont par exemple gérées). Certains d'entre eux ont des valeurs qui constituent des hypothèses de travail. Les résultats numériques permettent, en les comparant aux données relevées sur le terrain, de confirmer ou d'infirmer en partie ces hypothèses.

4.1.2 Interactions entre hôte et parasites

Les parasites adultes fixés sur les hôtes pondent des œufs à chaque pas de temps. Des classes d'âge sont définies pour ces œufs ; d'un pas de temps au suivant, ils passent d'une classe d'âge à l'autre (processus de vieillissement). Lors du passage d'une classe à l'autre une certaine proportion d'œufs meurt. Le taux de mortalité $\mu_e(\theta(t))$ appliqué alors dépend de la température $\theta(t)$ de l'eau. Selon cette température, les œufs ont un certain temps d'incubation $I_p(\theta(t))$ avant qu'ils n'éclosent. Un contingent de larves nageantes est alors présent dans le bassin durant une période de deux jours à l'issue de laquelle les larves meurent si elles n'ont pas trouvé d'hôte. Il est augmenté d'un ensemble de larves nageantes extérieures contenues dans l'eau alimentant le bassin. Le mois de l'année détermine le nombre moyen de larves apportées dans le milieu. En fait, cet apport extérieur de larves nageantes constitue le *facteur déclenchant* du système hôte-parasite.

Une proportion de larves nageantes se fixe sur les hôtes. Celle-ci est déterminée [59] par un facteur de transmission noté $T(H(t))$ qui dépend du nombre d'hôtes $H(t)$ disponibles. Le nombre de larves fixées étant ensuite connu, elles sont réparties sur les différents hôtes du bassin. Le modèle fait l'hypothèse qu'à partir d'un certain seuil, plus l'hôte a de parasites plus sa probabilité d'en recruter s'accroît. Ceci est modélisé dans la fonction $\varphi(j, l, t)$ qui donne la probabilité qu'un hôte ayant l parasites recrute j larves à l'instant t.

Les cohortes de parasites vieillissent sur les hôtes. Les cohortes de parasites sont d'abord jeunes, puis adultes 18 jours après le recrutement. Le taux de mortalité des parasites quel que soit leur âge est donné par $\mu(\theta(t))$ qui dépend de la température $\theta(t)$. La mortalité différenciée en âge n'est pas incluse actuellement dans le modèle [64]. Les hôtes ayant un nombre l de parasites ont un taux de survie qui décroît lorsque l est grand et s'annule pour l supérieur au *seuil létal* fixé à 700 ou 800 parasites. La fonction $p(l)$ donne la probabilité qu'un hôte porteur de l parasites meurt. Les différents types de mécanismes intervenant dans le modèle sont maintenant décrits. Les paragraphes suivants les détaillent un à un.

4.2 Les œufs de parasites

4.2.1 Mortalité des œufs

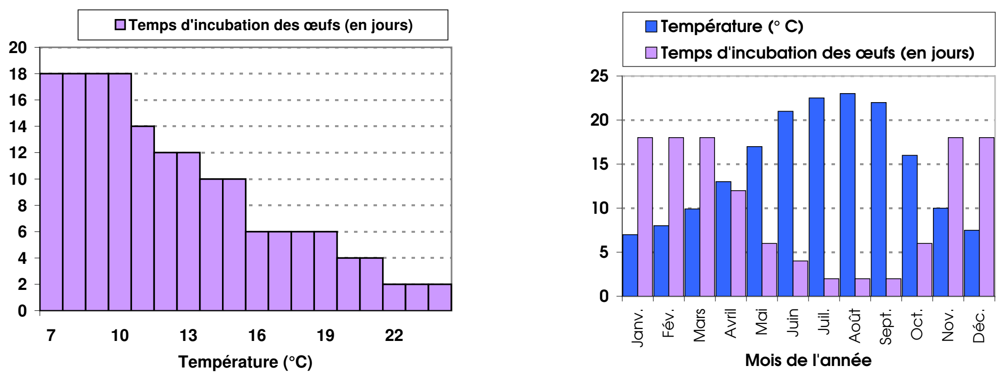
A chaque pas de temps, les œufs pondus viennent former la cohorte d'age 0 ; posons $E_0(t)$ le nombre d'œufs dans cette cohorte. Par extension, la cohorte d'âge $\tau\Delta t$ comporte $E_\tau(t)$ œufs. A la fin de chaque pas de temps, la cohorte d'œufs d'âge $\tau\Delta t$ devient la cohorte d'âge $(\tau{+}1)\Delta t$. Un taux de mortalité $\mu_e(\theta(t))$ sur un intervalle de temps Δt dépendant de la température est alors appliqué à chaque cohorte. Les œufs dont l'âge devient supérieur ou égal à la durée d'incubation $I_p(\theta(t))$ éclosent. Les larves nageantes issues de ces œufs viennent s'ajouter au stock de larves nageantes présentes dans le milieu ; ces nouvelles larves sont dénombrées par $L_B(t)$. Le nombre de larves présentes dans le bassin au pas de temps t est égal à $L(t)$. Ces larves proviennent des éclosions dans le bassin $L_B(t)$ et des larves rentrant dans le système via l'alimentation en eau $L_{ext}(t)$. On en déduit donc l'égalité : $L(t) = L_B(t) + L_{ext}(t)$. Les équations reprenant les différents points énoncés ci-dessus sont les suivantes :

$$\begin{cases} \text{pour} & 0 < \tau < I_p(\theta(t)) & E_\tau(t+\Delta t) = E_{\tau-1}(t)\,(1-\mu_e(\theta(t))) \\ \text{pour} & \tau \geq I_p(\theta(t) & E_\tau(t+\Delta t) = 0 \\ \text{et} & L_B(t+\Delta t) = \sum_{\tau=I_p(\theta(t))}^{I_p(\theta(t+\Delta t))} & E_{\tau-1}(t)\,(1-\mu_e(\theta(t)))\ . \end{cases}$$

Le cadre d'étude choisi pour l'application numérique du modèle correspond à un bassin de prégrossissement situé sur l'étang de Thau. Les paramètres numériques sont issus de [12, 13, 62, 65–67]. Pour les simulations, l'apport de larves nageantes extérieures au bassin $L_{ext}(t)$ sera de 40 larves par jour pendant 6 mois à partir du 1er juin et nul durant le reste de l'année.

4.2.2 Non-linéarité dans l'éclosion des œufs

L'un des paramètres abiotiques du système est la température de l'eau. Elle détermine le temps d'incubation des œufs en jours via une fonction définie par palier. La Figure 8(a) présente cette fonction, et l'on constate que pour les basses températures le temps d'incubation est de 18 jours pour descendre à 2 jours à 24°C [13]. Au terme du temps d'incubation, des larves nageantes sortent des œufs.



(a) (b)

Figure 8. Courbes présentant le temps d'incubation des œufs fonction du mois ou de la température

La courbe de température moyenne au cours de l'année est donnée sur la Figure 8(b). Sur le même schéma, le temps d'incubation des œufs en fonction du mois se déduit directement de la courbe des températures. Lorsque le temps d'incubation s'allonge en début de mois, cela se traduit par le fait qu'aucun œuf n'éclot. Cette absence d'éclosion introduit une non-linéarité dans le système qui peut avoir des conséquences sur la dynamique[8]. D'un autre côté, lorsque le temps d'incubation diminue avec des températures

8. il s'agit là d'un artefact que l'on ne supprime pas car ce type de variabilité ponctuelle peut se

croissantes, plusieurs cohortes d'œufs donnent des larves nageantes en même temps. Cela engendre une arrivée massive de jeunes parasites sur les poissons qui peut altérer la dynamique qui prévalait auparavant. Par exemple d'avril à mai (Figure 9, 334$^{\text{ème}}$ jour) , le temps d'incubation baisse de 6 jours soit $3\Delta t$. L'augmentation correspond à 3 cohortes supplémentaires qui viennent s'adjoindre à l'unique cohorte qui donnent des larves nageantes habituellement. Le début de la simulation se situe au premier juin. Les pics des jours 304 et 334 des courbes descriptives des nombres de larves correspondent aux changements de mois mars/avril et avril/mai. La courbe du nombre d'hôtes chute lors de ces changement de mois, conséquence directe de ces arrivées massives de larves nageantes. La courbe de mortalité des œufs en fonction de la température est présentée dans la Figure 10, avec celle de la mortalité des œufs fonction du mois qui en découle.

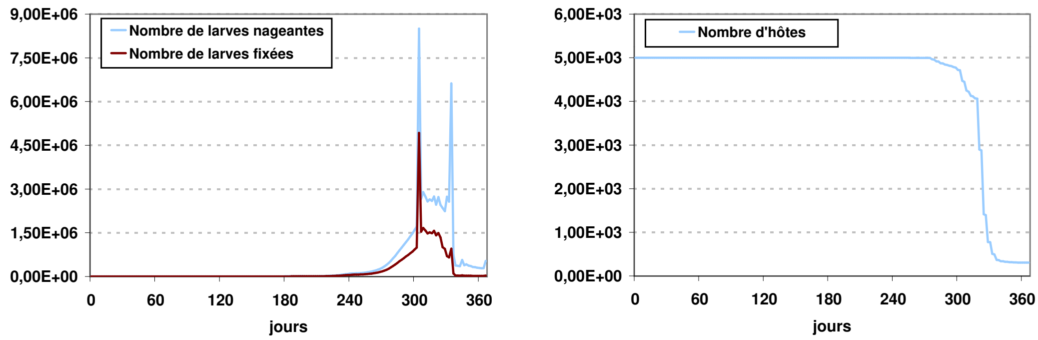


Figure 9. Nombre de larves et d'hôtes $\rho = 0,6$

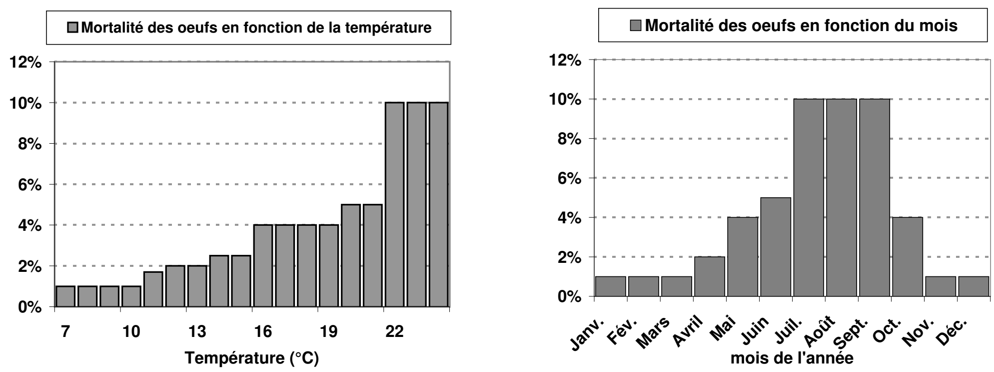
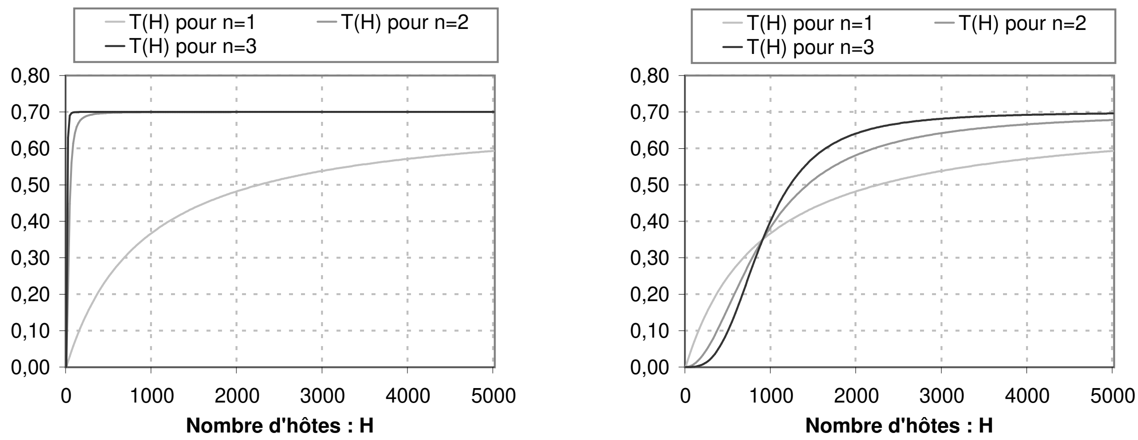


Figure 10. Courbes donnant la mortalité des œufs fonction du mois ou de la température

produire, et il est interessant d'en constater les effets sur la dynamique.



(a) Première version de T (b) Seconde version de T

Figure 11. Le facteur de transmission $T(H(t))$ pour différents n et $C_0 = 900$

4.3 Recrutement des larves

La modélisation du recrutement des larves nageantes sur les hôtes a un rôle essentiel dans le phénomène d'agrégation de parasites. La solution originale proposée a pour objectif de reproduire des processus rarement détaillés et peu validés.

4.3.1 Facteur de transmission

Le nombre de larves fixées $L_r(t)$ est égal au nombre de larves nageantes que multiplie le facteur de transmission :

$$L_r(t) = T(H(t))\, L(t) \ .$$

Reprenant [19, 49, 50], le facteur de transmission des larves nageantes (stade libre) vers le stade fixé peut être choisi de la forme suivante (comme dans [13]) :

$$T(H(t)) = \rho \frac{H(t)^n}{C_0 + H(t)^n} \ .$$

Ce facteur intègre trois paramètres : ρ, C_0 et n. On nomme ρ le facteur de saturation de la fonction T, et on a $0 < \rho < 1$. Il est introduit pour tenir compte du fait qu'il n'est pas réaliste de supposer que toutes les larves nageantes se fixent. Il permet de poser une limite supérieure au facteur de transmission puisque $0 \leq T(H(t)) \leq \rho$. Les courbes correspondant à n variable pour $C_0 = 900$ et $\rho = 0.7$ fixés sont données dans la Figure 11(a). On remarque que les courbes pour des n différents sont vraiment très dissemblables. Il est assez difficile, dans ces conditions, de faire varier facilement le paramètre n. Il paraît en fait plus cohérent que les paramètres C_0 et n soient liés pour permettre plus de continuité lorsqu'un seul paramètre varie. Nous avons proposé une légère modification de cette fonction :

$$T(H(t)) = \rho \frac{H(t)^n}{C_0^n + H(t)^n} \ .$$

En reprenant les paramètres des courbes précédentes, le changement opéré conduit à la courbe de la Figure 11(b). On remarque que les courbes se croisent en $(C_0; \rho/2) = (900; 0.35)$, *i.e.* à 50 % de saturation. Considérons le scénario suivant : un certain nombre d'hôtes meurent à cause du parasitisme, $H(t)$ décroît. Si $n = 1$, le facteur de transmission baisse alors petit à petit jusqu'à $H(t) = C_0$. Si l'on prend $n > 1$, on constate que le facteur de transmission baisse peu au début, mais se met à chuter ensuite brusquement en approchant de C_0. Un effet retard apparaît donc avec un paramètre n élevé. Cet effet lié à la densité d'hôtes s'accorde avec les données recueillies sur le terrain. Actuellement nous nous servons pour les simulations du facteur de transmission modifié avec pour paramètres $C_0 = 900,\ n = 3$ et ρ variable.

4.3.2 Définition de $pmax$ et $psup$

La densité d'hôtes ayant un nombre très important de parasites est limitée par le fait que l'on considère qu'un hôte ayant plus de ($lethal = 700$ ou 800) parasites, meurt dans les deux jours. La distribution des parasites sur les hôtes $N(l,t)$ tend vers 0 lorsque l tend vers l'infini. Pour limiter le coût des calculs et la place mémoire occupée par les structures de données, il est intéressant d'introduire $pmax(t)$ qui sera le seuil après lequel la distribution $N(l,t)$ n'est plus calculée. Il faut tenir compte du fait que les parasites adultes, fixés sur un hôte mourant, pondent des œufs qui restent dans le système au pas de temps $t+\Delta t$. Le seuil $pmax(t)$ est choisi tel que pour tout l supérieur à celui-ci, $N(l,t) < 10^{-8}$:

$$\forall t,\ \exists\ pmax(t) \text{ tel que } N(pmax(t),t) \geq 10^{-8} \text{ et } \ \forall l > pmax(t),\ N(l,t) < 10^{-8}$$

A partir de $pmax(t)$, on peut aussi définir un autre seuil $psup(t)$ qui intervient dans certains calculs. Il est défini par $psup(t) = min(lethal, pmax(t))$. On se servira par la suite de la propriété suivante : tous les hôtes ayant moins de $psup(t)$ parasites ont une probabilité non nulle de survivre au pas de temps $t+\Delta t$ ($psup(t)$ est renommé $S(t)$ ou S dans certaines parties de ce document). Lorsque l'on met à jour les distributions N des parasites sur les hôtes, on ne tient compte que des hôtes ayant moins de $psup(t)$ parasites, ce qui a pour conséquence de réduire les calculs.

4.3.3 Fonction F, objectifs

Des observations expérimentales ont permis d'établir qu'à partir d'un certain seuil de parasitisme les hôtes changent de comportement. Le nombre de parasites correspondant à ce seuil sera noté *palier* dans la suite de ce document. Une des hypothèses faite dans le modèle est que les hôtes ayant plus de *palier* parasites vont recruter prioritairement les larves présentes dans le milieu. Ce constat est générateur d'agrégation, phénomène courant chez les macroparasites. On considère deux groupes distincts d'hôtes. Le groupe $\dot{\mathrm{P}}$ est constitué des hôtes qui ont moins de *palier* parasites et sont peu parasités. Le groupe $\dot{\mathrm{I}}$ englobe les hôtes très infestés ayant plus de *palier* parasites. La fonction F donne la proportion de larves qui va être recrutée par les hôtes du groupe $\dot{\mathrm{I}}$. On pose que cette fonction F dépend uniquement du pourcentage $x(t)$ d'hôtes appartenant à i. Précisons que la densité d'hôtes est $H(t) = \sum_{l \in [0,pmax(t)]} N(l,t)$. On a la relation :

$$x(t) = \frac{\sum_{l=palier+1}^{pmax(t)} N(l,t)}{H(t)} .$$

On en déduit que le nombre de larves qui seront recrutées par les poissons ayant plus de *palier* parasites est $L_r(t)\, F(x(t))$. Le nombre de larves qui se fixeront sur les autres poissons est le complémentaire $L_r(t)\,(1 - F(x(t))$. Pour définir la fonction F, on se donne uniquement son point d'inflexion (x_i, a_0) choisi convenablement. La fonction F choisie possède la forme paramétrique suivante :

$$\begin{cases} \text{si } 0 \leq x \leq x_i \ : \ F(x) = e^{\alpha x} - 1, \\ \text{si } x_i < x \ : \ F(x) = a_0 + \gamma\, log(\beta + \delta\, x). \end{cases}$$

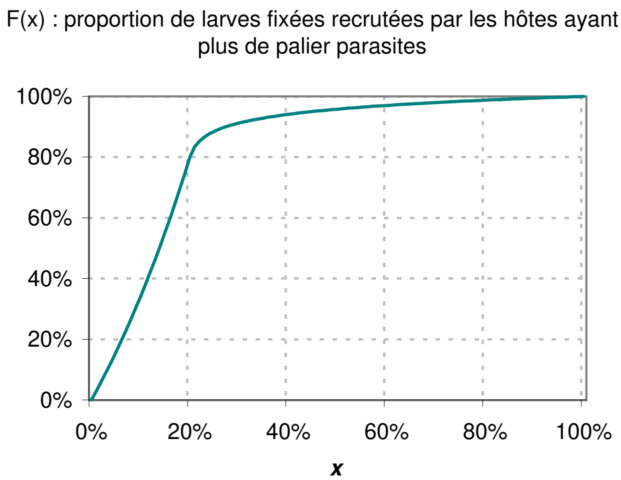


Figure 12. Graphe de la fonction F, avec point d'inflexion à $(x_i = 20\%, a_0 = 80\%)$

Nous avons déterminé par le calcul (section 4.3.4 suivante) l'espace admissible pour ce point d'inflexion. On en déduit ensuite les valeurs de α, β, δ et γ à l'aide des trois conditions naturelles suivantes : le point $(1,1)$ appartient au graphe de F, le point (x_i, a_0) appartient au graphe de F, en ce dernier point les dérivées à droite et à gauche sont égales. Un exemple de courbe F pour un point (x_i, a_0) fixé à (20 %,80 %) est donné sur la Figure 12. L'interprétation que l'on peut en donner est la suivante : si 20 % des hôtes ont plus de *palier* parasites alors ces mêmes hôtes recrutent 80 % des larves qui se fixent. Cette représentation est issue de la règle des 20:80 introduite par Pareto en 1906. Elle fut ensuite reprise dans de nombreuses disciplines, par exemple en épidémiologie : souvent 80 % ou plus des macroparasites est détenu par 20 % ou moins des hôtes [6].

4.3.4 Fonction F, calcul des paramètres

Il s'agit maintenant de calculer la valeur des inconnues α, β, δ et γ. Les contraintes énoncées précédemment s'écrivent de la manière suivante (on suppose $\gamma \neq 0$) :

$$\begin{cases} a_0 + \gamma\, log(\beta + \delta) = 1 \\ a_0 + \gamma\, log(\beta + \delta\, x_i) = a_0 \\ e^{\alpha\, x_i} - 1 = a_0 \\ \gamma\, \delta\, /\, (\beta + \delta\, x_i) = \alpha\, e^{\alpha\, x_i} \end{cases} \quad soit \quad \begin{cases} \gamma\, log(\beta + \delta) = 1 - a_0 \\ \beta + \delta\, x_i = 1 \\ \alpha = log(1 + a_0)/x_i \\ \gamma\, \delta = \alpha\, (1 + a_0)\ . \end{cases}$$

On peut déduire directement $\alpha = log(1 + a_0)/x_i$. Par contre, pour trouver δ, on doit résoudre l'équation suivante :

$$\gamma\, log(1 - \delta\, x_i + \delta) = 1 - a_0,\ (\text{avec}\gamma\, \delta = \alpha\, (1 + a_0)),$$
$$\text{soit}\ \ log(1 + \delta(1 - x_i))/(\delta(1 - x_i)) = (1 - a_0)/(\alpha(1 + a_0)(1 - x_i)).$$

Posons $z = \delta(1 - x_i)$ et $y = x_i\, (1 - a_0)/((1 + a_0)(1 - x_i) log(1 + a_0))$, on a alors

$$log(z + 1)/z = y \quad . \tag{11}$$

Comme $log(z + 1)/z$ est compris entre 0 et 1 pour z strictement positif, on en déduit l'inégalité $y \leq 1$. Ceci impose *a posteriori* une contrainte sur les données a_0 et x_i sous la forme suivante (domaine non grisé sur la Figure 7) :

$$y = \frac{x_i\, (1 - a_0)}{log(1 + a_0)\, (1 + a_0)(1 - x_i)} \leq 1 \quad .$$

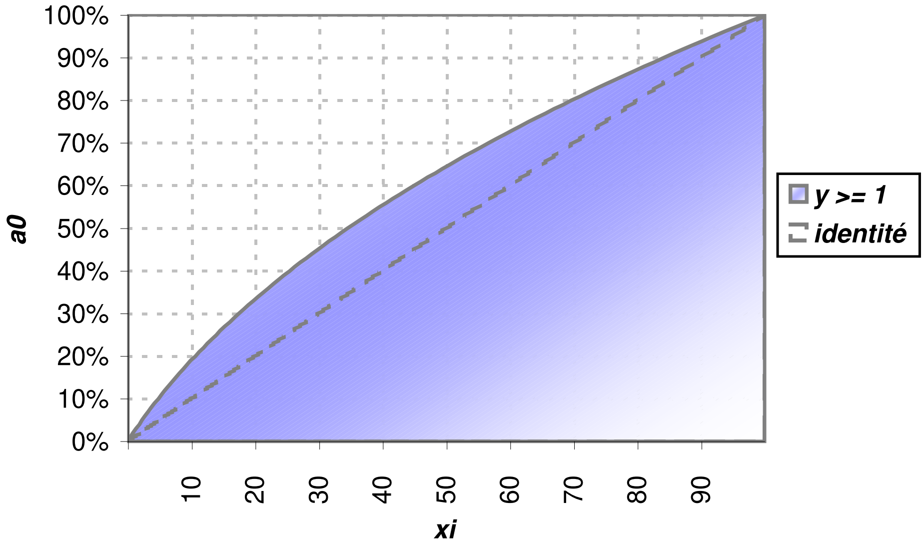


Figure 13. Contrainte sur les paramètres a_0 et x_i

On connaît x_i, a_0 donc y et l'on recherche z. On a les inégalités et $0 < x_i < 1$ et $0 < a_0 < 1$; le système d'équations posé permet de déduire que α, δ puis γ, z, y sont des réels strictement positifs. D'autre part, l'équation $log(z+1) = z\,y$ a deux solutions pour z, dont l'une d'elle est $z_0 = 0$ et l'autre $z_1 > 0$. Nous présentons une résolution de cette équation en annexe D. On utilise la fonction W de Lambert, définie par $W_c(x)\,e^{W_c(x)} = x$. On a alors

$$\delta = z/(1-x_i) = \frac{1}{y} ln(-\frac{1}{y} W_{-1}(-y\,e^{-y}))/(1-x_i),$$
puis $\gamma = \alpha(1+a_0)/\delta$ et $\beta = 1 - \delta\,x_i$.

4.3.5 Fonction f

La fonction F a permis de distinguer deux groupes d'hôtes différents, ceux ayant plus de *palier* parasites et les autres. Une deuxième fonction intervient dans le recrutement, $f(l,t)$ qui fixe le taux de recrutement moyen d'un hôte ayant l parasites. Soit $C(l,t)$ la classe des hôtes ayant un même nombre de parasites l. La fonction f va indiquer que chaque classe d'hôtes ayant moins de *palier* parasites recrute un nombre identique de larves ; f est donc constante. Elle permet aussi d'exprimer qu'au-delà de *palier* parasites les classes d'hôtes recrutent d'autant plus de parasites que le nombre de parasites l sur les hôtes est grand ($f(l,t)$ est alors une fonction croissante pour l supérieur à *palier*). Il s'agit là encore d'un phénomène d'agrégation sur les hôtes déjà parasités. La quantité $f(l,t)\,N(l,t)$ représente la proportion du nombre de larves recrutées par les hôtes ayant l parasites. On a donc la relation suivante (la somme des probabilités valant 1) :

$$\sum_{l \in [0, pmax(t)]} f(l,t) N(l,t) = 1 \ . \tag{12}$$

La classe d'hôte ayant l parasites recrute en tout $f(l,t)N(l,t)L_r(t)$ parasites. Comptons maintenant le nombre de larves recrutées par les hôtes les moins parasités ($\dot{\text{P}}$) :

$$L_r(t)(1 - F(x(t))) = \sum_{l=0}^{palier} f(l,t)N(l,t)L_r(t), \text{ ce qui implique}$$
$$(1 - F(x(t))) = \sum_{l=0}^{palier} f(l,t)N(l,t)$$

et pour les hôtes les plus parasités ($\dot{\text{I}}$), on a

$$L_r(t)F(x(t)) = \sum_{l=palier+1}^{pmax(t)} f(l,t)N(l,t)L_r(t), \text{ donc}$$
$$F(x(t)) = \sum_{l=palier+1}^{pmax(t)} f(l,t)N(l,t) \ .$$

Ces relations imposent des contraintes sur la fonction $f(l,t)$ car la valeur $F(x(t))$ est connue ; elles seront explicitées après la définition de f ci-après. Le recrutement moyen d'un hôte ayant l parasites $f(l,t)$ est constant pour $0 \leq l \leq palier$, puis croissant pour

$l > palier$. On introduit un paramètre nf qui fixe le degré du monôme déterminant la croissance de la fonction f. La proposition de [34] est la suivante :

$$\begin{cases} \text{si } 0 \leq l \leq palier \ : \ f(l,t) = f_0(t), \\ \text{si } palier < l \leq pmax(t) \ : f(l,t) = f_0(t) + \lambda(t) \ \ f_1(l - palier) \text{ avec } f_1(l) = l^{nf}. \end{cases}$$

Dans ces conditions on obtient la fonction f présentée Figure 14 lorsque $nf = 2$.

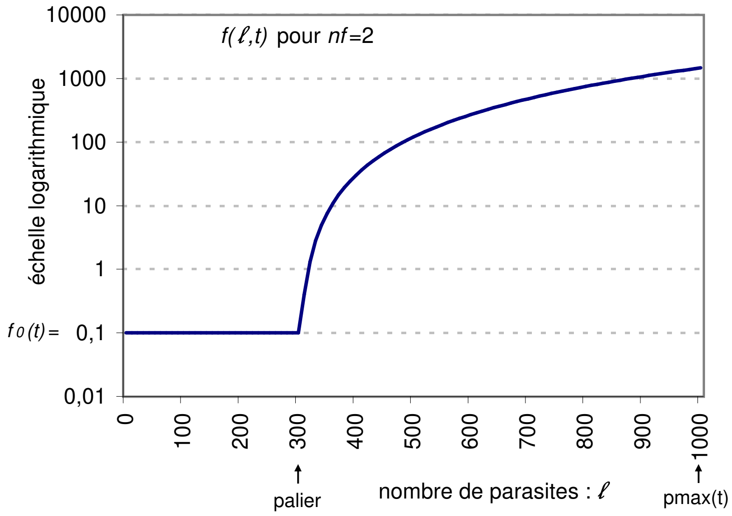


Figure 14. Fonction $f(l,t)$: espérance de larves recrutées à l'instant t par un hôte ayant l parasites avec $f_0(t) = 0.1$ et $\lambda(t) = 0.003$

Les relations précédentes permettent alors de déterminer $f_0(t)$:

$$(1 - F(x(t))) = \sum_{l=0}^{palier} f_0(l,t)\, N(l,t), \quad \text{donc}$$

$$f_0(t) = (1 - F(x(t))) / \sum_{l=0}^{palier} N(l,t) \ .$$

D'autre part on cherche la valeur de $\lambda(t)$:

$$F(x(t)) = \sum_{l=palier+1}^{pmax(t)} f(l,t) N(l,t)$$

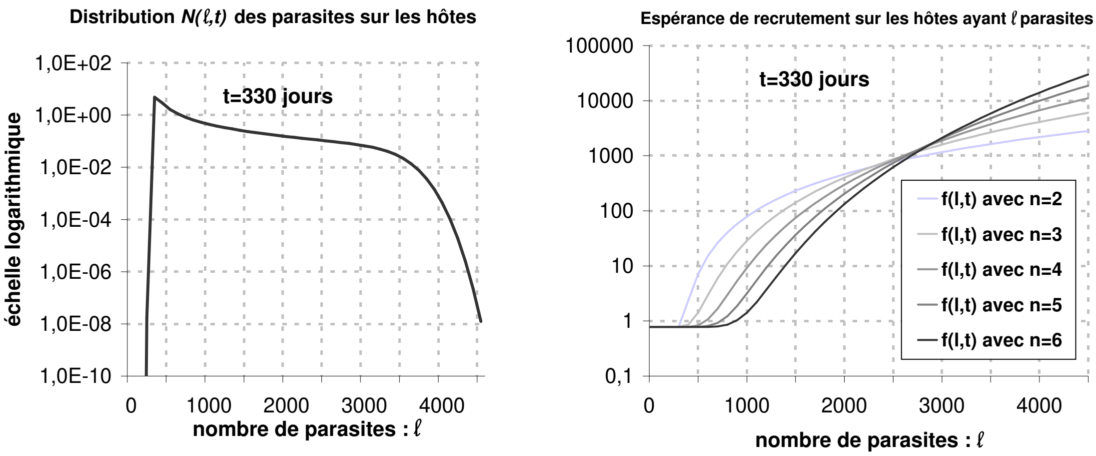
$$F(x(t)) = f_0(t) \sum_{l=palier+1}^{pmax(t)} N(l,t) + \lambda(t) \sum_{l=palier+1}^{pmax(t)} f_1(l)\, N(l,t)$$

$$1 - f_0(t) \sum_{l=0}^{palier} N(l,t) = f_0(t) \sum_{l=palier+1}^{pmax(t)} N(l,t) + \lambda(t) \sum_{l=palier+1}^{pmax(t)} f_1(l)\, N(l,t) \ .$$

Or on a $\sum_{l \in [0, pmax(t)]} N(l,t) = H(t)$, donc on peut déterminer $\lambda(t)$:

$$1 - f_0(t)\, H(t) = \lambda(t) \sum_{l=palier+1}^{pmax(t)} f_1(l)\, N(l,t) \quad \text{et} \quad \lambda(t) = \frac{1 - f_0(t)\, H(t)}{\sum_{l=palier+1}^{pmax(t)} f_1(l)\, N(l,t)} \ .$$

Pour illustrer le type de graphes de f que l'on obtient durant une simulation, un exemple suit Figure 15. La distribution des parasites sur les hôtes est présentée dans la Figure (a) au pas $t = 330$ jours d'une simulation ($\rho = 0,65$). Les graphes f possibles selon le paramètre nf sont présentés sur la Figure (b).



(a) (b)

Figure 15. Distribution des parasites sur les hôtes au temps 330 jours en (a), espérance de recrutement en larve sur les hôtes ayant l parasites sur (b)

On constate que l'espérance de recrutement sur un hôte ayant plus de 3000 parasites est supérieure à 1000 larves quelle que soit la valeur de nf. L'espérance de recrutement sur un hôte ayant 800 parasites est comprise entre 0.77 et 40.6 larves selon nf. Or les parasites recrutés par les hôtes qui vont mourir à cause d'une charge parasitaire importante, n'iront pas infester les hôtes peu parasités. Le choix de nf influe donc considérablement sur la distribution future des parasites sur les hôtes. Une surdispersion importante favorisera dans certaines conditions la survie des hôtes peu parasités, comme on le verra cela peut contribuer à une régulation de la population parasitaire. L'agrégation générée dans le modèle se contrôle donc notamment avec cette valeur du paramètre nf et le choix du point d'inflexion de F.

4.3.6 Loi de Poisson ou loi binomiale pour la fonction φ

Le nombre moyen de larves qui se fixent sur un hôte ayant l parasites est maintenant déterminé, il est égal à $L_r(t)\, f(l,t)$. Etant donnée cette moyenne, la distribution explicite de ce nombre de larves sur les hôtes reste à fixer (quelle loi de probabilité utiliser ?). Parmi les hôtes ayant l parasites (classe d'hôtes $C(l,t)$), on a noté $\varphi(j,l,t)$ la probabilité de recruter j larves à l'instant t. Les relations suivantes caractérisent cette loi de probabilité :

$$\sum_{l=0}^{\infty} \varphi(j,l,t) = 1 \ , \quad \sum_{l=0}^{\infty} j\, \varphi(j,l,t) = L_r(t)\, f(l,t) \ .$$

Dans une première approche [34], la loi $\varphi(j,l,t)$ correspondait à une loi de Poisson de paramètre j et de moyenne $L_r(t)\, f(l,t)$. Néanmoins, ce choix occasionne une incohérence sémantique lorsque j devient supérieur au nombre de larves disponibles $L_r(t)$. En effet, que cela signifie-t-il de recruter plus de larves en moyenne que le nombre de larves disponibles à un instant t ? Nous avons donc proposé l'utilisation de la loi binomiale $\mathcal{B}(L_r(t), f(l,t))$. On a dans ce second cas :

$$\sum_{j=0}^{L_r(t)} \varphi(j,l,t) = 1 \ , \quad \sum_{j=0}^{L_r(t)} j\, \varphi(j,l,t) = L_r(t)\, f(l,t) \ .$$

Cela signifie que la probabilité pour une larve de se fixer sur un hôte ayant l parasites est égale à $f(l,t)$. Mathématiquement, lorsque $L_r(t)$ tend vers $+\infty$, la loi binomiale $\mathcal{B}(L_r(t), f(l,t))$ tend vers la loi de Poisson de la première approche. Néanmoins, lorsqu'il y a peu de larves dans l'environnement et que $L_r(t)$ est faible, la loi binomiale est la plus adaptée. Quelle que soit la loi choisie, on a la propriété :

$$\sum_{l=0}^{\infty} \sum_{j=0}^{\infty} j\, \varphi(j,l,t)\, N(l,t) = L_r(t) \ .$$

Cette égalité signifie que toutes les larves recrutées par toutes les classes d'hôtes ayant l parasites est $L_r(t)$ qui est le nombre de larves à recruter.

4.4 Mortalité des parasites

Dans cette partie on introduit le problème de la structuration en âge des parasites et de la difficulté de prise en compte d'une *mortalité différenciée selon l'âge.*

4.4.1 Modélisation de la structure en âge

Dans une première version, le modèle décrivait de manière détaillée la densité d'hôtes possédant une certaine répartition de parasites classés par âge [13]. On pouvait alors considérer isolément les parasites d'âges différents et avoir une mortalité différenciée dans chaque classe d'âge. On appelait $N(b_0, b_1 \ldots, b_K, t)$ le nombre d'hôtes ayant b_0 parasites d'âge 0 et b_i parasites d'âge $i\Delta t$ (avec K=10). On calculait $N(b_0, b_1 \ldots, b_K, t + \Delta t)$ au temps $t + \Delta t$ en fonction de $N(b_0, b_1 \ldots, b_K, t)$. Cette mise à jour de la structure de données N possédant 11 dimensions en plus du temps t induisait en pratique des coûts de calcul considérables. Pour étayer cette affirmation, supposons que pour chacune des dimensions de N, les indices b_i soient dans l'intervalle [0,700] ; l'accès en écriture à tous les éléments de la structure, indispensable pour la mise à jour de cette dernière, demanderait $701^{11} = 2.0\,10^{31}$ opérations d'écriture [9]. Ceci rend inaccessible toute validation par

9. A la vitesse d'accès mémoire de 2 Go/s, ce qui représente actuellement une bonne performance, l'accès en écriture à l'ensemble de la structure nécessiterait $2.0\,10^{31}/(2\,10^9) = 1.0\,10^{22}$ secondes soit plus de 300 milliards de siècles.

la simulation numérique en un temps raisonnable, même dans les années à venir et en tenant compte des évolutions du matériel informatique en terme de performance. Il était donc indispensable de trouver un modèle mathématique moins fin, engendrant moins de calculs.

Langlais et Silan [63] ont décrit un modèle simplifié de ce type. Soit $N_k(l,i,t)$ la densité d'hôtes ayant l parasites dont i d'âge supérieur à $k\,\Delta t$, la mise à jour de cette structure à chaque pas de temps est alors donnée par l'équation (13).

$$N_k(i,l,t+\Delta t) = \sum_{m=i}^{l} \sum_{u=0}^{psup(t)-m} \sum_{v=0}^{u} N_{k-1}(i+v, m+u, t)\, \varphi(l-m, m+u, t) \\ \phi(u, m+u)\, \psi(v, i+v, u, m+u)\, p(m+u) \quad . \tag{13}$$

La répartition des parasites sur les hôtes n'est cependant plus détaillée ; la mortalité différenciée en fonction de l'âge des parasites n'est donc plus aussi simple. Notons que dans cette formule, la fonction $\phi(u,x)$ est la probabilité que u parasites meurent sachant qu'il y en avait x à l'instant t. La quantité $\psi(v,y,u,x)$ est la probabilité que v parasites d'âge supérieur à $q\Delta t$ meurent durant Δt (q étant inconnue), sachant qu'à l'instant t, y parasites d'âge supérieur à $q\Delta t$ sont présents, et que u parasites meurent parmi x. Ce modèle utilise donc la notion de mortalité moyennée des parasites avec la fonction ϕ. Cette mortalité doit être cohérente avec la mortalité différenciée en fonction des classes d'âge et introduite par la fonction ψ. Il se pose donc un problème de cohérence directement lié à la modélisation mathématique choisie.

4.4.2 Mortalité différenciée selon l'âge

Considérons un hôte à un instant t ayant y parasites d'âge supérieur à $k\Delta t$ parmi un total de x parasites ; et supposons que l'on connaisse le taux de mortalité μ_k des parasites d'âge supérieur à $k\Delta t$; l'objectif est de calculer la contribution de cet hôte à la densité $N_{k+1}(i,l,t)$. En utilisant une loi de Poisson de paramètre μ_k, on peut simuler la mort des y parasites d'âge supérieur à $k\Delta t$; ce n'est pas si simple pour les $x-y$ parasites d'âge strictement inférieur à $k\Delta t$. En effet, la répartition en âge de ces derniers est inconnue et leur mortalité ne peut pas être déduite. Dans le cas général, moyenne et variance de la loi de mortalité des $x-y$ parasites ne peuvent pas être connues. Pour illustrer ce propos, considérons $N_1(20,100,t)$ le nombre d'hôtes ayant 100 parasites dont 20 d'âge supérieur strictement à Δt ; à un moment t_0 de la simulation, supposons qu'il n'y ait pas de parasites d'âge 0 ; les hôtes de $N_1(20,100,t_0)$ ont alors $x-y=80$ parasites d'âge inférieur ou égal à Δt, et les 80 parasites d'âge Δt meurent avec une loi de moyenne μ_1. A l'instant t_0, les parasites d'âge 0 étant absents, il n'y a logiquement pas de parasites d'âge Δt au temps $t_0+\Delta t$. Les hôtes de $N_1(20,100,t_0+\Delta t)$ ont donc chacun 80 parasites d'âge 0 qui meurent avec une loi de moyenne μ_0. Pour ces deux pas de temps, les 80 parasites d'âge inférieur ou égal à Δt ont donc des taux de mortalité différents μ_0 et μ_1 (les lois de mortalité peuvent aussi avoir des variances différentes). Ceci est incohérent avec le modèle

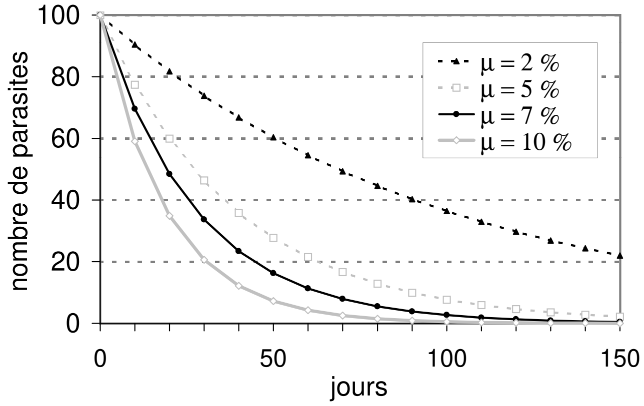


Figure 16. Courbes de survie des parasites (100 à $t = 0$) pour différentes valeurs du taux de mortalité μ

qui suppose un taux de mortalité constant qui dépend juste de t, $i = 20$ et $l = 100$ et rien d'autre. Finalement, ne connaissant pas la répartition en âge complète des parasites sur les hôtes, nous concluons que les lois de ϕ et ψ ne peuvent pas être déterminées dans le cadre proposé [64].

4.4.3 Loi de mortalité unique

Il existe néanmoins un cas particulier où il est possible de déterminer ces lois ; c'est le cas où la loi de mortalité est la même pour tous les parasites, quelle que soit leur classe d'âge. Le taux de mortalité commun est appelé μ. Cette loi peut alors être illustrée en représentant l'effectif d'une population de parasites en fonction du temps. La Figure 16 traduit, pour des taux de mortalité différents, la survie des parasites à partir d'une population initiale de 100 individus.

La solution adoptée dans [13] visait à faire dépendre la mortalité unique de la moyenne d'âge des parasites. Dans ce cas, le taux de mortalité est d'autant plus faible que la population est jeune ; inversement, une population plus âgée sera concernée par une mortalité plus forte. Illustrons par un exemple la méthode de pondération de la mortalité en fonction de l'âge moyen de la population. Soit p_a le nombre de parasites adultes et p_j le nombre de parasites jeunes ; deux taux de mortalité sont fixés, l'un pour les adultes $\mu_a(\theta_e)$ et l'autre pour les jeunes $\mu_j(\theta_e)$; ces deux taux dépendent dans le modèle de la température du milieu. Le taux de mortalité moyen est calculé à l'aide de la formule $\mu(\theta_e) = (p_j\mu_j(\theta_e) + p_a\mu_a(\theta_e))/(p_a + p_j)$. L'avantage d'une telle méthode est de pouvoir moduler la mortalité en fonction de la structure en âge de la population parasitaire présente. L'inconvénient majeur réside dans le fait que cette mortalité moyenne s'applique aussi bien aux parasites jeunes qu'adultes. Cela n'est formellement pas satisfaisant, les jeunes peuvent mourir aussi vite que les adultes si, par exemple, il y a beaucoup de parasites âgés. Ces fluctuations peuvent être évitées en fixant une mortalité moyenne indépendante de la structure en âge de la population parasitaire. Les courbes de la Figure 16 s'appliquent alors seulement à la population parasitaire prise dans son ensemble.

4.4.4 Limitations et solutions

Par ailleurs, la formulation mathématique relative à la modélisation de ψ avec une loi de probabilité simple pose problème. Dans le modèle initial, des approximations utilisant une loi de Poisson tronquée puis normalisée ont été réalisées. L'espérance mathématique n'était pas connue exactement, à cause de la normalisation. La relation indispensable pour une loi de probabilité était en revanche respectée : la somme, pour toutes les valeurs possibles de v des quantités $\psi(v, y, u, x)$, y, u, x étant fixés, était égale à 1. La mortalité était également liée au produit $\phi(u, x)\, \psi(v, y, u, x)$; cela introduisait des corrélations entre variables v, y, u et x qui n'était pas énoncées. Nous proposons une autre formulation des calculs qui utilise des variables indépendantes.

Considérons un hôte entre l'instant t et $t + \Delta t$; l'événement E est défini par : v parasites meurent sur y d'âge supérieur à $q\Delta t$ et au total u parasites meurent parmi x ; on a les relations $v \leq u$ et $y \leq x$. Une autre manière de décrire l'événement E est de dire que v parasites sur y d'âge supérieur à $q\Delta t$ meurent et $u - v$ parasites sur $x - y$ d'âge strictement inférieur à $q\Delta t$ meurent. Ainsi, le nombre de morts d'âge supérieur à $q\Delta t$ n'est plus corrélé au nombre de morts d'âge strictement inférieur à $q\Delta t$. La loi de mortalité des parasites est décrite par une loi binomiale δ de paramètre μ. L'événement " v parasites sur y d'âge supérieur à $q\Delta t$ meurent " a pour probabilité $\delta(v, y)$ et suit la loi $\mathcal{B}(v, y, \mu)$. Dans ce cadre, l'événement E a pour probabilité : $\delta(v, y)\delta(u-v, x-y)$. Une formule de mise à jour équivalente à la relation (10) peut alors être écrite à l'aide de δ :

$$N_k(i, l, t + \Delta t) = \sum_{m=i}^{l} \sum_{u=0}^{psup(t)-m} \sum_{v=0}^{u} N_{k-1}(i+v, m+u, t)\, \varphi(l-m, m+u, t) \qquad (14)$$
$$\delta(v, i+v)\, \delta(u-v, m+u-(i+v))\, p(m+u) \quad .$$

Avec cette nouvelle formulation que nous donnons dans [64], il n'y a plus d'ambiguïté sur le sens de la fonction de mortalité des parasites.

4.4.5 Cohérence des calculs, propriétés mathématiques

Cette dernière formule (14) peut être elle-même réécrite à l'aide de variables qui simplifieront la lecture par la suite. Le symbole x est défini comme le nombre de parasites sur un hôte à l'instant t ; dans le contexte de la formule précédente on a $x = m + u$. Le symbole j constitue le nombre de parasites recrutés sur un certain hôte en $t + \Delta t$; il est égal à $u - x + l$. L'emploi de ces variables (permutations de sommes avec changements de variables) conduit à la formule :

$$N_k(i, l, t+\Delta t) = \sum_{x=i}^{psup(t)+l-i} \sum_{j=max(l-x,0)}^{min(l-i, psup(t)-i+l-x)} \qquad (15)$$
$$\sum_{v=0}^{min(j+x-l, x-i)} N_{k-1}(i+v, x, t)\, \varphi(j, x, t)\, \delta(v, i+v)\, \delta(u-v, x-(i+v))\, p(x) \quad .$$

Par définition, $U_k(t)$ est le nombre de parasites dont l'âge est supérieur à $k\Delta t$. Cette quantité se calcule avec la formule suivante :

$$U_k(t) = \sum_{l=0}^{psup(t)} \sum_{i=0}^{l} i\, N_k(i,l,t) \ . \tag{16}$$

Cette formule peut être interprétée de la manière suivante : la quantité $i\, N_k(i,l,t)$ est le nombre de parasites d'âge supérieur à $k\Delta t$ fixés sur l'ensemble des hôtes ayant l parasites dont i d'âge supérieur à $k\Delta t$. En sommant ce nombre de parasites pour tous les hôtes ayant l parasites (l variant entre 0 et $psup(t)$), dont i parasites sont d'âge supérieur à $k\Delta t$ (i variant entre 0 et l), on obtient tous les parasites d'âge supérieur à $k\Delta t$. A partir de (16), nous allons prouver mathématiquement que le nombre total de parasites sur les hôtes au cours d'un pas de temps diminue logiquement avec une mortalité moyenne de μ. Pour se placer dans le cas général et surtout rendre les calculs plus lisibles, on considère que $psup(t) = \infty$. Supposons que lors d'un pas de temps, il n'y ait pas de mortalité due au parasitisme parmi les hôtes ; cela signifie mathématiquement que le terme $p(x)$ vaut 1 lors de l'évaluation de la formule (15). La formule de mise à jour de N_k devient ans ces conditions :

$$N_k(i,l,t+\Delta t) = \sum_{x=i}^{\infty} \sum_{j=max(l-x,0)}^{l-i} \sum_{y=i}^{j+x-l+i} N_{k-1}(y,x,t)\, \varphi(j,x,t) \tag{17}$$
$$\delta(y-i,y)\, \delta(x-y-l+j, x-y) \ .$$

Cette expression est utilisée pour exprimer le nombre de parasites d'âge supérieur à $(k+1)\Delta t$ en $t+\Delta t$ de la manière suivante (en combinant (16) et (17)) :

$$U_k(t+\Delta t) = \sum_{l=0}^{\infty}\sum_{i=0}^{l}\sum_{x=i}^{\infty} \sum_{j=max(l-x,0)}^{l-i} \sum_{y=i}^{j+x-l+i} i\, N_{k-1}(y,x,t)\varphi(j,x,t)\delta(y-i,y)\delta(x-y-l+j,x-y) \ .$$

Des permutations de sommes et des changements de variables conduisent à

$$U_k(t+\Delta t) = \sum_{x=0}^{\infty}\sum_{y=0}^{x} N_{k-1}(y,x,t) \sum_{v=0}^{y} (y-v)\delta(v,y) \sum_{z=0}^{x-y} \delta(z,x-y) \sum_{j=0}^{\infty} \varphi(j,x,t) \ . \tag{18}$$

Or la fonction δ suit une loi binomiale et φ une loi de Poisson ; les propriétés intrinsèques de ces lois permettent d'écrire les relations suivantes :

$$\sum_{j=0}^{\infty} \varphi(j,x,t) = 1 \quad , \qquad \sum_{z=0}^{x-y} \delta(z,x-y) = 1 \quad ,$$
$$\sum_{v=0}^{y} \delta(v,y) = 1 \quad , \qquad \sum_{v=0}^{y} v\, \delta(v,y) = \mu(\theta(t))\, y \ .$$

En utilisant ces propriétés, la dernière expression de U_k se réécrit de la manière suivante

$$U_k(t+\Delta t) = \sum_{x=0}^{\infty}\sum_{y=0}^{x} N_{k-1}(y,x,t)\, y\,(1-\mu(\theta(t)))\ , \tag{19}$$

$$\text{soit}\quad U_k(t+\Delta t) = (1-\mu(\theta(t)))\, U_{k-1}(t)\ . \tag{20}$$

Finalement, le nombre de parasites au cours d'un pas de temps diminue bien avec une mortalité moyenne $\mu(\theta(t))$. Cette preuve garantit une cohérence numérique qui était tout simplement inaccessible auparavant avec les fonctions ϕ et ψ. On constatait d'ailleurs dans les résultats de certaines simulations que la mortalité observée, $\mu_{obs}(t) = 1 - U_k(t+\Delta t)/U_{k-1}(t)$, était différente de celle spécifiée par le paramètre d'entrée $\mu(\theta(t))$. C'est d'ailleurs en faisant cette observation au cours des simulations que le problème a été détecté puis résolu. Ce résultat souligne bien l'intérêt de l'interaction entre modélisation bio-mathématique et simulation informatique. L'utilisation de la fonction δ plutôt que les fonctions ψ, ϕ signifie que l'espérance de la mortalité des parasites peut désormais être contrôlée durant la simulation. Ceci permet, de manière incidente, d'améliorer la robustesse du code. Cet exemple de la mortalité des parasites sur les hôtes illustre bien quelques unes des contraintes qu'un modèle, que l'on souhaite de plus en plus réaliste, pose à ses développeurs.

4.5 Survie de l'hôte liée à l'intensité

Le taux de survie des hôtes $p(l)$ dépend du nombre de parasites présents sur l'hôte. Cette fonction est composée de deux parties distinctes. Lorsque le nombre de parasites l est relativement faible tous les hôtes survivent, le taux de survie est égal à 1. Au-delà d'une certaine limite (modulable à l'aide d'un paramètre nommé *critique*), le taux de survie de l'hôte décroît rapidement. La fonction utilisée pour modéliser ceci est basée sur une fraction rationnelle qui est définie par :

$$\begin{cases} A = (lethal^3 + (lethal - critique)^3)/lethal^3, & \\ \text{si } 0 \le l \le lethal, & p(l) = A\,(lethal - l)^3/((lethal - l)^3 + (lethal - critique)^3)) \\ \text{si } lethal < l, & p(l) = 0\ . \end{cases}$$

La Figure 17 reproduit la courbe de la fonction $p(l)$ sur l'intervalle $400 < l < 810$ pour $lethal = 800$. Les courbes représentent la probabilité de survie d'un hôte fonction de l'intensité individuelle pour différents intervalles de temps.

D'après les expériences de terrain réalisées sur les jeunes bars élevés en bassin, les hôtes de moins d'un an peuvent supporter une charge allant jusqu'à 800 parasites [13]. Au-delà de ce seuil, la probabilité de survie est très faible. Avec le jeu de paramètres $lethal = 800$, $critique = 770$, on observe qu'un hôte à une probabilité de survie à 50 % avec 770 parasites sur un pas de temps. On observe de même qu'un hôte à une probabilité de survie à 50 % avec 717 parasites pendant un mois, ou de 50 % avec 695 parasites durant deux mois.

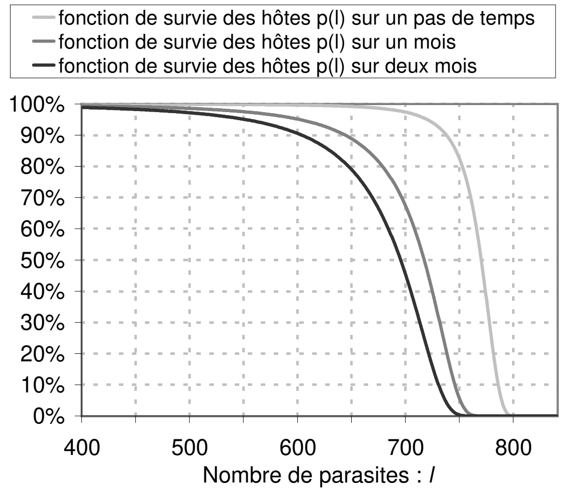


Figure 17. Fonctions de survie des hôtes $p(l)$, pour $lethal = 800,\ critique = 770$

4.6 Mise à jour des distributions de parasites

Configuration initiale Au temps $t = 0$, on suppose que les 5000 hôtes considérés n'ont pas de parasite :

$$N(0,0) = H(0) = 5000,\ \forall l > 0\ \ N(l,0) = 0\ .$$

Dans les paragraphes suivants, on reprend les équations (8) (9) et (10) que l'on adapte au calcul numérique en supprimant les bornes infinies.

Mise à jour de la distribution non différenciée en âge $N(l,t)$ Considérons un hôte avec l parasites au temps $t+\Delta t$. Durant l'intervalle de temps de t à $t+\Delta t$, il peut avoir recruté j larves alors que u parasites mourraient de mort naturelle. Donc au pas de temps précédant t, cet hôte avait $l+u-j$ parasites avec $0 \leq j \leq l$ parce qu'aucune des j larves n'est morte durant la phase de recrutement ; de plus, il a survécu $l+u-j$ parasites et l'on a $0 \leq l \leq l+u-j \leq psup(t)$. Finalement :

$$N(l,t+\Delta t) = \sum_{j=0}^{l} \sum_{u=0}^{psup(t)-l+j} N(l+u-j,t)\,\varphi(j,l+u-j,t)\,\delta(u,l+u-j)\,p(l+u-j)\ . \qquad (21)$$

Mise à jour de la distribution différenciée en âge $N_k(l,i,t)$ Considérons un hôte ayant l parasites parmi lesquels i sont plus vieux que Δt. Au pas de temps t cet hôte avait au moins i parasites à cause du processus de vieillissement, c'est à dire $i+u$ parasites ; entre t et $t+\Delta t$, u parasites sont morts d'une mort naturelle alors que $l-i$ étaient recrutés. De plus, l'hôte doit survivre à une charge de $i+u$ parasites. On en déduit que :

$$N_1(i,l,t+\Delta t) \;=\; \sum_{u=0}^{psup(t)-i} N(i+u,t)\,\varphi(l-i,i+u,t)\,\delta(u,u+i)\,p(i+u)\;.$$

Considérons maintenant, en $t+\Delta t$ un hôte ayant l parasites parmi lesquels i sont plus vieux que $k\Delta t$, $2 \leq k \leq K$. Au temps t cet hôte avait au moins i parasites plus vieux que $k\Delta t$, c'est à dire $i+v$ parmi lesquels v sont morts de mort naturelle entre t et $t+\Delta t$; la charge totale de parasites au temps t était $m+u$ parmi lesquels u meurent, $0 \leq v \leq u$, ce qui conduit à un recrutement de $l-m$ larves. Enfin l'hôte survit à une charge de $m+u$ parasites. On a donc, pour $2 \leq k \leq K$, $0 \leq i \leq l$, $0 \leq l \leq psup(t)$ et si $\psi(i,i{+}v,m,m{+}u) = \delta(v,i{+}v)\,\delta(u{-}v,m{+}u-i{-}v)$:

$$N_k(i,l,t+\Delta t) = \sum_{m=i}^{l} \sum_{u=0}^{psup(t)-m} \sum_{v=0}^{u} N_{k-1}(i{+}v,m{+}u,t)\,\varphi(l{-}m,m{+}u,t) \qquad (22)$$
$$\psi(i,i{+}v,m,m{+}u)\,p(m{+}u)\quad .$$

4.7 La ponte

Le taux de fécondité $\lambda_a(\theta(t))$ des parasites adultes est de 4 œufs par pas de temps jusqu'à 15°C inclus, puis 6 œufs à partir de 16°C. L'ensemble des parasites adultes qui peuvent pondre est de taille $\sum_{l\in[0,pmax(t)]}\sum_{i\in[0,l]} i\,N_K(l,i,t)$. Avant d'établir une fonction de ponte, il nous faut considérer certains éléments biologiques. Un hôte possède deux séries d'arcs branchiaux distincts. Un parasite une fois fixé ne peut se déplacer vers l'autre série d'arcs branchiaux. Il se peut donc que deux parasites adultes sur un même hôte n'aient pas la possibilité de s'inter-féconder [61] et donc de pondre [10]. D'autre part, un parasite qu'il soit adulte ou sub-adulte a la possibilité féconder un adulte. Lorsque l'abondance est faible, une distribution aléatoire des parasites sur les hôtes conduit éventuellement à une croissance lente de la population parasitaire (voir p. 153) due à une probabilité de fécondation petite. Les hypothèse prises dans le modèle [12] reproduisent partiellement le fonctionnement de la reproduction parasitaire, phénomène complexe et variable :

- si un hôte n'a qu'un parasite, ce parasite n'est pas fécondé et ne pond pas ;
- s'il y a deux parasites sur un hôte, la probabilité qu'ils soient sur une même série d'arcs branchiaux (et donc de pondre pour un adulte) est de 0.5 ;
- dans les autres cas, on considère que tous les parasites adultes de l'hôte pondent.

Le nombre d'œufs pondus E_0 en $t{+}\Delta t$ est donc :

$$E_0(t{+}\Delta t) = \lambda_a(\theta(t))\,(\frac{1}{2}\sum_{i\in[1,2]} i\,N_K(2,i,t) + \sum_{l\in[3,pmax(t)]}\sum_{i\in[1,l]} i\,N_K(l,i,t))\;.$$

10. ce fonctionnement est proche du concept de "mating function" étudié par May [6].

Chapitre 5

Modèle stochastique

Dans les deux paragraphes qui suivent, les différentes étapes du cycle de vie du parasite *Diplectanum aequans* et les interactions avec les hôtes qui interviennent dans les simulations stochastiques sont explicitées. Toutes les données gérées sont mises à jour tous les $\Delta t = 2$ jours comme pour la simulation déterministe. Il est admis [54] que lorsque les populations considérées dans un système ont des effectifs faibles, les modélisations déterministe et stochastique ont potentiellement des dynamiques différentes. Dans le modèle hôte-parasite considéré, les effectifs d'hôtes peuvent devenir très faibles en situation d'épizootie. D'autre part, le nombre de parasites par hôte est limité dans certains cas. Le recours à une modélisation individu-centrée se justifie donc afin de quantifier l'effet de la stochasticité démographique sur le modèle. On veut en effet savoir si des variations aléatoires concernant le recrutement ou la mortalité des parasites a un impact sur la dynamique.

D'autre part, pour des modèles non-linéaires simples de systèmes biologiques [47], des approches stochastiques et déterministes ne conduisent pas nécessairement au même résultat. Puisque notre modèle déterministe présente des non-linéarités notamment lié à l'agrégation, il est intéressant de savoir si un modèle stochastique a le même comportement. De plus, la simulation stochastique nous donne accès à la variance des sorties. Elle nous renseignera sur la sensibilité du modèle à la fluctuations des effectifs.

5.1 Partie commune aux deux modèles déterministe et stochastique

Des classes d'âge sont définies pour les œufs de parasites présents dans le bassin ; d'un pas de temps au suivant, les œufs passent d'une classe d'âge à une autre avec un certain taux de survie afin de modéliser leur vieillissement. Le taux appliqué dépend de la température $\theta(t)$. Selon la température, les œufs ont un certain temps d'incubation avant d'éclore. Après l'éclosion, un contingent de larves nageantes est présent dans le bassin durant une période de deux jours. Ce contingent de larves se voit renforcé d'un plus ou moins grand nombre de larves nageantes extérieures contenues dans l'eau alimentant le bassin. En fait, cet apport extérieur constitue le facteur déclenchant du système hôte-parasite. Une certaine proportion des larves nageantes se fixent sur les poissons. Le nombre de larves fixées $L(t)$ étant connu, elles sont réparties sur les différents poissons du bassin. Le simulateur stochastique reprend du simulateur déterministe la partie de l'application qui gère l'environnement, les œufs et les larves. Les éléments du chapitre précédent qui concernent cette partie se retrouvent à l'identique dans le modèle stochastique. Ce choix permet de comparer les deux simulateurs seulement au niveau des interactions directes entre hôtes et parasites fixés. On se focalise ainsi sur les processus de recrutement des larves, de mortalité des parasites, et de mortalité des hôtes. Dans la Figure 18, les différents processus intégrés au modèle sont résumés.

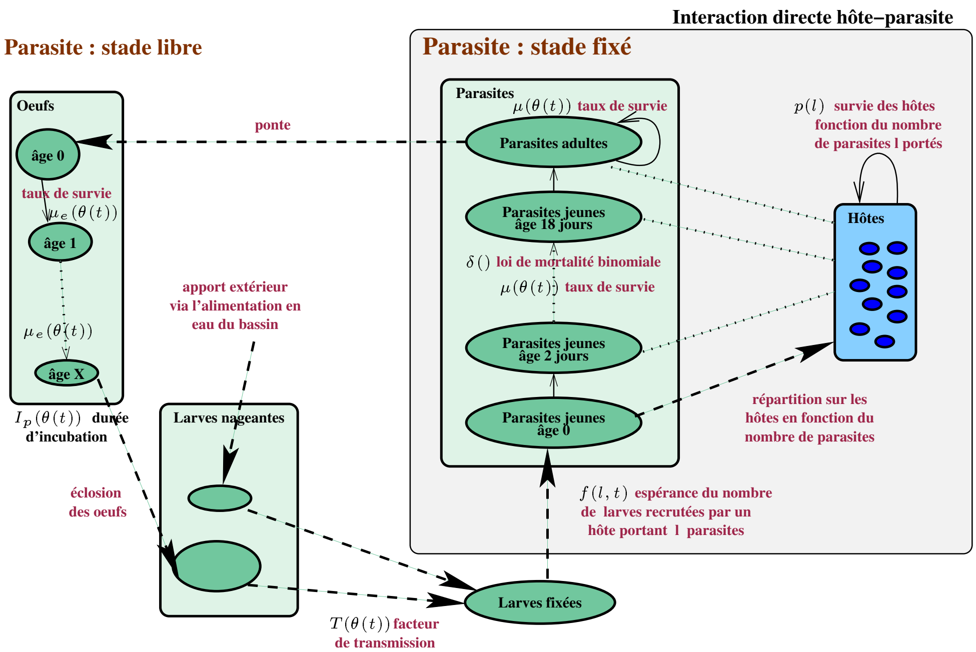


Figure 18. Schéma de l'interaction entre le cycle de vie du parasite et l'hôte

5.2 Partie différente selon le modèle

Les modèles font l'hypothèse forte qu'à partir d'un certain seuil, un processus est générateur d'agrégation sur les hôtes les plus parasités. Ceci est modélisé notamment par la fonction $f(l,t)$ qui donne l'espérance du pourcentage de larves fixées qui seront recrutées par un hôte ayant l parasites à l'instant t. Les parasites sont structurés en âge : 9 classes d'âge de jeunes parasites, 1 classe d'adultes. Les cohortes de parasites sont d'abord jeunes (2 jours dans chaque classe d'âge), puis adultes 18 jours après le recrutement (ils restent ensuite dans cette dernière classe d'âge). Les cohortes de parasites vieillissent sur les hôtes, leur probabilité de mort est donnée par $\mu(\theta(t))$. On considère que la mortalité des hôtes ne dépend que de leur charge parasitaire. Les hôtes ayant un nombre x de parasites ont un taux de survie $p(x)$ qui décroît lorsque x est grand et s'annule pour x supérieur au seuil létal de parasites. Les parasites adultes fixés sur des poissons pondent des œufs à chaque pas de temps. La modélisation stochastique de cette partie a été décrite dans la Section I.2.3.2. Elle utilise les mêmes fonctions que celles décrites dans le modèle déterministe, à savoir : f, F, p, δ. La différence réside uniquement dans les structures de données utilisées et la méthode de calcul de la mise à jour pour les hôtes et les parasites fixés. Dans le modèle déterministe, les structures de données $N(l,t)$, $N_k(l,i,t)$ sont actualisées d'un pas de temps à l'autre grâce à plusieurs formules mathématiques. D'un autre côté, la simulation stochastique fait appel à des tirages aléatoires selon certaines lois afin de faire évoluer les hôtes et les parasites gérés individuellement. On ajoute ainsi de la stochasticité démographique ; par contre, des aléas issus de l'environnement (stochasticité environnementale), issus d'une catastrophe ou de la stochasticité génétique ne sont pas considérés pour l'instant dans le modèle.

Les interactions hôte-parasite sont donc évaluées par deux approches distinctes. Leur coût en calcul dépend de paramètres différents comme on le verra dans de la Partie suivante. Les différences entre modélisation stochastique et déterministe n'est pas évidente à comprendre en profondeur sauf peut être pour des systèmes simples [47] (processus de Yule-Furry, modèles de type *Linear Immigration Death*, *NonLinear Birth Immigration Death* ...). Dans la partie IV, nous envisagerons quelques aspects qui caractérisent et différencient les deux approches dans nos simulations. Cependant, nous n'approfondirons pas cette problématique complexe qui touche de nombreuses disciplines telles que l'économie, l'automatique, l'écologie, et bien d'autres encore.

Partie III

Algorithmique et performance

Chapitre 6

Simulation parallèle d'un modèle déterministe pour un système hôte-macroparasite

Notre démarche a consisté à mettre au point un simulateur parallèle robuste et efficace pour valider le modèle déterministe discret. Afin de réaliser cet objectif, nous réduisons dans un premier temps les coûts en calcul de la simulation, et ceci sans modifier le modèle. Il serait envisageable de simplifier le modèle ou d'aborder le système avec un modèle différent, mais il n'est pas question de cela dans cette thèse. Nous nous sommes focalisé sur la parallélisation de la simulation déterministe dans le chapitre 1. Dans le second chapitre, on décrit la parallélisation d'une simulation stochastique.

6.1 Analyse séquentielle

Le temps cumulé de l'ensemble des mises à jour de $N_*(*,*,t)$ et $N(*,t)$ (voir Section II-4.6 p. 51) représente plus de 99 % du temps total de simulation lorsque l'épizootie se développe et que les hôtes commence à mourir. Le temps de calcul de ces étapes dépend notoirement, comme nous le verrons, du nombre léthal de parasites pour un hôte. Le travail visant à raccourcir les temps d'exécution de l'application a surtout porté sur cette partie de la simulation. Dans les prochaines sections, nous allons préciser comment effectuer la mise à jour de $N_*(*,*,t)$ en $N_*(*,*,t+\Delta t)$.

6.1.1 Evaluation de la complexité

Etudions la complexité algorithmique de la mise à jour de $N_k(i,l,t{+}\Delta t),\ \ 2 \leq k \leq K$. On prendra comme hypothèse de départ que les fonctions $\varphi,\ p,\ \psi$ utilisées dans la formule ne demandent aucun calcul (on révisera cette hypothèse par la suite p. 61). Précisons que pour une somme de n termes, on comptera pour simplifier n additions et non pas $n-1$.

De plus, supposons pour un instant t fixé que l'on pose $S = p_{sup}(t)$ pour simplifier les notations. Deux changements de variables sur l'équation (22) p. 51 conduisent à :

$$N_k(i,l,t+\Delta t) \;=\; \sum_{m=i}^{l}\sum_{c=m}^{S}\underbrace{\varphi(l-m,c)\,p(c)}_{1}\;\sum_{d=i}^{c-m+i}\underbrace{N_{k-1}(d,c,t)\,\psi(d-i,d,c-m,c)}_{2}\;.$$
$$0 \le i \le l,\quad 0 \le l \le S,\; 2 \le k \le K\;.$$

Comptons le nombre d'opérations effectuées :

- si l'on comptabilise le nombre de multiplications effectuées dans la partie 2 de l'expression, on en trouve une seule. Lors de l'évaluation de l'expression entière, le nombre de multiplications sera :

$$F_1 = \sum_{k=2}^{K}\sum_{l=0}^{S}\sum_{i=0}^{l}\sum_{m=i}^{l}\sum_{c=m}^{S}\sum_{d=i}^{c-m+i} 1\;;$$

- il y a deux multiplications dans la partie 1 de l'expression. Lors de l'évaluation de l'expression entière, on en aura :

$$2\,F_2 = 2\sum_{k=2}^{K}\sum_{l=0}^{S}\sum_{i=0}^{l}\sum_{m=i}^{l}\sum_{c=m}^{S} 1\;;$$

- quant au nombre d'additions, il y en a au total (si l'on pose $F_3 = \sum_{k=2}^{K}\sum_{l=0}^{S}\sum_{i=0}^{l}\sum_{m=i}^{l} 1$) :

$$F_3 + F_2 + F_1 \quad .$$

La complexité $C_o(S,K)$ de la mise à jour de la donnée $N_k(i,l,t+\Delta t)$ pour $0 \le i \le l$, $0 \le l \le S$, $2 \le k \le K$ est donc :

$$C_o(S,K) = 1\,F_1 + 2\,F_2 + 1\,[F_1 + F_2 + F_3] = 2\,F_1 + 3\,F_2 + F_3\;.$$

Les calculs concernant la simplification des expressions de F_1, F_2, F_3 se trouvent en annexe B.1.2 (p. 185), et on utilise ces polynômes pour évaluer C_o :

$$\begin{aligned} C_o(S,K) = \quad & 2\,(K-1)\,[\frac{1}{40}\,S^5 + \frac{7}{24}\,S^4 + \frac{31}{24}\,S^3 + \frac{65}{24}\,S^2 + \frac{161}{60}\,S + 1] \\ & + 3\,(K-1)\,[\frac{1}{12}\,S^4 + \frac{2}{3}\,S^3 + \frac{23}{12}\,S^2 + \frac{7}{3}\,S + 1] \\ & + 1\,(K-1)\,[\frac{1}{6}\,S^3 + S^2 + \frac{11}{6}\,S + 1]\;, \end{aligned}$$

$$C_o(S,K) = (K-1)\,[\frac{1}{20}\,S^5 + \frac{5}{6}\,S^4 + \frac{19}{4}\,S^3 + \frac{73}{6}\,S^2 + \frac{71}{5}\,S + 6]\;.$$

Bien que polynômiale, cette complexité en $O(K\,S^5)$ est tout de même très coûteuse en termes de nombre d'opérations à effectuer. De plus, on a supposé pour l'instant que les évaluations des fonctions ϕ, p, ψ, φ ne demandent aucune opération.

6.1.2 Précalculs et Factorisations

Il nous faut tout d'abord revenir sur l'hypothèse qui considère que l'évaluation des fonctions qui apparaissent dans la formule de mise à jour ne demande aucun calcul (par exemple lorsque l'on retrouve les valeurs dans un tableau). Considérons la fonction $\psi(v,i,u,l)$ qui se calcule de la manière suivante (*cf* p. 47) :

$$\psi(v,y,u,x) = \delta(v,y)\,\delta(u-v,x-y) = \mathcal{B}(v,y,\mu(\theta(t)))\mathcal{B}(u-v,x-y,\mu(\theta(t))) \ .$$

Au cœur de la formule de mise à jour, ce calcul peut nécessiter beaucoup d'opérations élémentaires s'il n'est pas optimisé. La complexité C_o sera alors nécessairement plus élevée. Notre première tâche est d'organiser les calculs afin d'éviter des calculs coûteux dans la boucle la plus interne de l'algorithme. Notamment, en effectuant une reformulation mathématique de la fonction ψ, on peut réduire son coût moyen de calcul à une seule multiplication. Pour cela, on précalcule un tableau ψ_1 qui a deux dimensions, et cela à chaque pas de temps t où la mortalité $\mu(\theta(t))$ change. Ce tableau ψ_1 contient simplement la loi binomiale $\mathcal{B}(v,y,\mu(\theta(t)))$, et il est indexé sur les indices v et y . On en déduit alors $\psi(v,y,u,x) = \psi_1(v,y)\,\psi_1(u-v,x-y)$; le coût de calcul consiste en une seule multiplication lors de l'évaluation de la formule de mise à jour. L'utilisation de ce tableau précalculé constitue un compromis entre calcul et place mémoire occupée. En effet, on aurait pu précalculer la fonction ψ fonction de 4 paramètres avec un coût en mémoire prohibitif pour un coût total en calcul moindre. La quantité S admet pour borne supérieure le paramètre $lethal = 800$ (voir p. 38). Le choix de stocker ψ_1 implique d'avoir en mémoire un tableau de taille $(lethal+1)\,(lethal+2)/2$ en réel double précision. Un réel prenant 8 octets, la place occupée en mémoire est donc $801 * 802/2 * 8 = 2.45$ MO. Nous sommes prêt à investir 2.45 Mo de mémoire pour cette optimisation qui permet une très grosse économie en calcul. D'autres précalculs sont effectués pour les différentes fonctions qui interviennent dans la formule de mise à jour. Dans la suite de ce document, on considère qu'au niveau de la mise à jour de $N_*(*,*,t)$ et $N(*,t)$, les fonctions φ, p nécessitent un accès mémoire, tandis que ψ demande une multiplication et deux accès mémoire.

Pour diminuer le nombre total d'opérations effectuées au cours de l'algorithme, il est intéressant de faire apparaître un maximum de facteurs multiplicatifs. Dans la formule de mise à jour, le nombre de multiplications dépend de l'ordre dans lequel sont faites les sommes. Par exemple, prenons deux formules équivalentes :

$$N_k(i,l,t+\Delta t) \ = \ \sum_{m=i}^{l}\sum_{c=m}^{S}\varphi(l-m,c)\,p(c)\sum_{d=i}^{c-m+i}N_{k-1}(d,c,t)\,\psi(d-i,d,c-m,c) \ ,$$

$$N_k(i,l,t+\Delta t) \ = \ \sum_{c=i}^{S}p(c)\sum_{m=i}^{min(c,l)}\varphi(l-m,c)\sum_{d=i}^{c-m+i}N_{k-1}(d,c,t)\,\psi(d-i,d,c-m,c) \ .$$

On remarque que dans la deuxième formule la permutation des deux premières sommes a permis de mettre en facteur $p(c)$ diminuant ainsi le nombre de multiplications effectuées au total. La formule de mise à jour comporte trois sommes, et on a donc $3! = 6$ combinaisons possibles pour l'ordre des sommes. La combinaison qui nécessite le moins de multiplications est cette deuxième écriture que l'on adopte donc pour la suite de l'étude.

6.1.3 Algorithme élémentaire

On déduit directement de la formule de mise à jour l'algorithme suivant :

```
Procédure MISE_A_JOUR(S) {
 Pour k:=2 à K faire {
 .  Pour l:=0 à S faire {
 .  .  Pour i:=0 à l faire {
 .  .  .  somme1:=0;
 .  .  .  Pour c:=i à S faire {
 .  .  .  .  somme2:=0;
 .  .  .  .  Pour m:=i à min(c,l) faire {
 .  .  .  .  .  somme3:=0;
 .  .  .  .  .  Pour d:=i à c-m+i faire {
 .  .  .  .  .  .  somme3:=somme3+N_{k-1}(d,c,t) (ψ1.ψ1)(d-i,d,c-m,c);
 .  .  .  .  .  }
 .  .  .  .  .  somme2:=somme2+φ(l-m,c) somme3;
 .  .  .  .  }
 .  .  .  .  somme1:=somme1+p(c) somme2;
 .  .  .  }
 .  .  .  N_k(i,l,t+Δt):=somme1;
 .  .  }
 .  }
 }
}
```

Evaluons la complexité de l'algorithme :

- au niveau de la boucle la plus interne en d, on a 2 multiplications et une addition, au total $3\,F_1$ opérations ;
- pour la boucle en m, on a une multiplication et une somme, soit $2\,F_2$ opérations ;
- enfin, pour la boucle en c, on a une addition et une multiplication :

$$\text{soit} \quad 2\,F_3' \quad \text{si} \quad F_3' = (\sum_{k=2}^{K} \sum_{l=0}^{S} \sum_{i=0}^{l} \sum_{c=i}^{S} 1) \quad \text{opérations.}$$

Au total, la complexité $C_1(S,K)$ de l'algorithme est donc[11] : $C_1(S,K) = 3\,F_1 + 2\,F_2 + 2\,F_3'$. En remplaçant F_1 , F_2 , F_3' par leurs valeurs, on trouve

$$C_1(S,K) = (K-1)\,[\frac{3}{40}\,S^5 + \frac{25}{24}\,S^4 + \frac{47}{8}\,S^3 + \frac{359}{24}\,S^2 + \frac{341}{20}\,S + 7]\ .$$

11. voir les calculs de F_1, F_2 et de F_3' en annexe B.1.2.

C_1 tend asymptotiquement vers $\frac{3}{40}\,K\,S^5$. Pour se rendre compte de la masse de calcul que cela peut représenter, prenons le cas limite suivant (atteint régulièrement lors des simulations). Nous voudrions une modélisation avec $K = 10$ et S valant 800.

$$C_1(S = 800, K = 10) = 225\ 10^{12} \quad .$$

Plusieurs centaines de TFLOP à réaliser semble déraisonnable pour les machines actuelles. Rappelons que cet algorithme de mise à jour est appelé une fois à chaque itération de la boucle sur le temps du simulateur. Il est donc possible d'avoir à faire plusieurs fois des calculs très coûteux comme celui-ci durant l'ensemble de la simulation.

6.1.4 Factorisation en k et l

On peut aller plus avant dans la factorisation en évitant des redondances de calcul.

$$\forall l \in [0, S(t)], \forall i \in [0, l], \forall k \in [2, K],$$

$$N_k(l,i,t+\Delta t) = \sum_{c=i}^{S(t)} p(c).\sum_{m=i}^{min(c,l)} \varphi(l-m,c,t) \underbrace{\sum_{d=i}^{c-m+i} N_{k-1}(c,d,t).\underbrace{(\psi_1.\psi_1)(d-i,d,c-m,c)}_{\mathbf{1}}}_{\mathbf{2}} \quad . \qquad (23)$$

Des parties de l'équation (23) sont en fait réévaluées plusieurs fois pour certaines combinaisons des variables l, i, k. Dans l'équation (23), on voit par exemple que le terme **1** ne dépend pas de la variable k, ce qui implique que pour différents k il va être calculé plusieurs fois. A l'aide d'un remaniement de l'algorithme, on élimine ce recalcul. Mais combien gagne-t-on ainsi ? Comme la multiplication $(\psi_1.\psi_1)(d-i,d,c-m,c)$ est faite une fois seulement pour l'ensemble des $k \in [2, K]$, il reste uniquement dans la boucle la plus interne de l'algorithme deux opérations en moyenne (une multiplication plus une addition), à la place de trois opérations en moyenne (deux multiplications plus une addition). Ainsi, la nouvelle complexité de l'algorithme représente approximativement $\frac{2}{3}$ de la complexité précédente. De manière similaire, il est intéressant de noter que le terme **2** de (23) est réévalué pour l variant sur son intervalle de définition. En stockant ce terme et en le réutilisant pour tous les l, on réduit notoirement le nombre des calculs. L'algorithme remanié est présenté sur la Figure 19.

Le nombre d'opérations réalisées est : $c{-}m{+}1$ multiplications (ligne 8), $(K{-}1)(c{-}m{+}1)$ multiplications et additions (ligne 10), $(K{-}1)$ multiplications (ligne 13), puis $(K{-}1)(S{-}m{+}1)$ additions et multiplications (ligne 15), $(K-1)(S-i+1)$ additions (ligne 20).

Soient D_1, D_2, D_3, D_4 les expressions suivantes (évaluées sous forme de polynômes en S et K dans l'annexe B.1.3 p. 186) :

$$D_1 = (\sum_{i=0}^{S}\sum_{c=i}^{S}\sum_{m=i}^{c}\sum_{d=i}^{c-m+i} 1) \qquad D_2 = (\sum_{i=0}^{S}\sum_{c=i}^{S}\sum_{m=i}^{c}\sum_{l=m}^{S} 1)$$

$$D_3 = \sum_{i=0}^{S}\sum_{c=i}^{S}\sum_{m=i}^{c} 1 \qquad D_4 = (\sum_{i=0}^{S}\sum_{c=i}^{S}\sum_{l=i}^{S} 1) \ .$$

```
Procédure MISE_A_JOUR(S) {
    Pour i:=0 à S faire {
    .    somme1(2:K,0:S):=0;
    .    Pour c:=i à S faire {
    .    .    somme2(2:K,0:S):=0;
    .    .    Pour m:=i à c faire {
    .    .    .    somme3(2:K):=0;
    .    .    .    Pour d:=i à c-m+i faire {
    .    .    .    .    tmp3:=(ψ1.ψ1)(d − i, d, c − m, c);
    .    .    .    .    Pour k:=2 à K faire
    .    .    .    .    somme3(k):=somme3(k)+N_{k−1}(d, c, t) tmp3;
    .    .    .    }
    .    .    .    Pour k:=2 à K faire {
    .    .    .    .    tmp3(k):=p(c) tmp3(k)
    .    .    .    .    Pour l:=m à S faire
    .    .    .    .    somme2(k,l):=somme2(k,l)+φ(l − m, c) somme3(k);
    .    .    .    }
    .    .    }
    .    .    Pour k:=2 à K faire {
    .    .    .    Pour l:=i à S faire
    .    .    .        somme1(k,l):=somme1(k,l)+somme2(k,l);
    .    .    }
    .    }
    .    Pour k:=2 à K faire {
    .        Pour l:=i à S faire N_k(i, l, t + Δt):=somme1(k,l);
    .    }
    }
}
```

Figure 19. Algorithme évitant les calculs redondants

Au total, la complexité C_2 est donc :

$$C_2(S,K) = (2\,K-1)\,D_1 + (2\,K-2)\,D_2 + (K-1)\,D_3 + (K-1)\,D_4 \ .$$

En remplaçant D_1, D_2, D_3, D_4 par leurs valeurs, il vient

$$C_2(S,K) = (\frac{1}{4}\,K-\frac{5}{24})\,S^4+(\frac{8}{3}\,K-\frac{9}{4})\,S^3+(\frac{37}{4}\,K-\frac{187}{24})\,S^2+(\frac{77}{6}\,K-\frac{43}{4})\,S+6\,K-5 \ .$$

Par rapport à l'algorithme élémentaire de complexité $C_1(S,K)$, pour $K = 10$ et S grand, le nombre d'opérations est donc divisé par :

$$\frac{C_1(S,K)}{C_2(S,K)} \approx \mathbf{0,29\,S + 0,95 + O(\frac{1}{S})} \ .$$

On a alors

$$C_2(S=800, K=10) = 951\ 10^9 \ .$$

Lorsque K=10, S=800, on est ramené à 0,95 TFLOP ; on fait de réelles économies par rapport aux solutions précédentes.

6.1.5 Vectorisation et utilisation des BLAS

6.1.5.1 Intérêt des BLAS

Les Basic Linear Algebra Subroutines (BLAS) constituent une bibliothèque de fonctions d'Algèbre linéaire. Les BLAS [4] conduisent à une réduction des temps d'exécution pour les calculs vectoriels. Elles visent deux objectifs :

- fournir des routines ayant de très bonnes performances sur différentes architectures,
- permettre une portabilité des programmes qui les utilisent (en changeant de machine, une version des BLAS garantira des optimisations adaptées à la nouvelle architecture).

A la création des BLAS, il a été décidé que les constructeurs proposeraient eux-mêmes les bibliothèques correspondant à leur matériel. La majorité des calculateurs et super-calculateurs en sont actuellement équipés. Elles permettent d'ailleurs très souvent d'approcher les performances de crête de ces machines. Les types acceptés par les BLAS sont les réels ou les complexes et cela en simple ou double précision.

On distingue trois niveaux dans la bibliothèque BLAS : le niveau 1 concerne les opérations vecteur-vecteur (ce sont elles qui ont été historiquement créées en premier), le niveau 2 les opérations matrice-vecteur, le niveau 3 les opérations matrice-matrice.

BLAS 1 Si X et Y sont des vecteurs de taille n, les opérations disponibles sont par exemple :

$$\begin{array}{lcl} \texttt{axpy} & : & Y \leftarrow \alpha * X + Y \\ \texttt{dot} & : & Y \leftarrow X^T + Y \\ \texttt{nmr2} & : & Y \leftarrow ||X||_2 \end{array}$$

Dans tous les cas, il y a $O(n)$ éléments à traiter et $O(n)$ opérations à réaliser. En jouant sur les tailles des vecteurs on peut gagner une accélération comprise entre 2 et 3.

BLAS 2 Soit X,Y des vecteurs et A une matrice carrée dont le côté est de taille n, les routines auxquelles on a accès sont par exemple :

$$\begin{array}{lcl} \texttt{gemv} & : & Y \leftarrow \alpha * A * X + \beta * Y \\ \texttt{ger} & : & A \leftarrow \alpha * X * Y^T + A \end{array}$$

Le nombre d'opérations ainsi que d'éléments à prendre en compte est $O(n^2)$. Les performances des BLAS 2 sont meilleures que celles des BLAS 1.

BLAS 3 Si A, B et C sont des matrices carrées (côté de taille n), on peut effectuer les opérations suivantes :

$$\begin{array}{lcl} \texttt{gemm} & : & C \leftarrow \alpha * A * B + \beta * C \\ \texttt{syrk} & : & C \leftarrow \alpha * A * A^T + \beta * C \end{array}$$

Les routines sont à ce niveau beaucoup plus efficaces que pour les BLAS 1 et 2. En effet, on doit réaliser $O(n^3)$ opérations pour seulement $O(n^2)$ références mémoire. Cela veut dire que l'on peut utiliser efficacement le cache du processeur et faire beaucoup de calculs sur des groupes de données présents dans le cache, ce qui augmente notablement les performances. Des détails sont donnés dans l'annexe C pour les architectures utilisées dans ce travail.

Pour profiter des gains en temps de calcul accessibles via les BLAS, il faut modifier les calculs et faire apparaître un maximum d'opérations vectorielles et matricielles.

6.1.5.2 Algorithme utilisant des opérations matricielles

Pour construire l'algorithme basé sur des opérations matricielles, on reprend l'algorithme avec les factorisations en k et l que l'on adapte. Parmi les formulations vectorielles de l'algorithme, on prend celle qui permet de ne faire que des opérations matricielles (donc uniquement du BLAS 3).
On introduit la notation `*` utilisée pour désigner dans un objet à plusieurs dimensions (par exemple une matrice), l'ensemble des composantes selon une certaine dimension de cet objet. On pourra éventuellement spécifier le nom de la dimension dont on prend toutes les composantes juste après le symbole `*`. Par exemple si $(s(m,k))_{m \in DM, k \in DK}$ est une matrice, $\mathtt{s}(*\mathtt{m}, \mathtt{k}_0)$ désigne le vecteur $s(m,k_0)_{m \in DM}$. On va calculer le nombre d'opérations effectuées au cours de l'algorithme en tenant compte des creux éventuels ($c.a.d$ les termes nuls) des matrices. Dans cet algorithme (Figure 20), on a deux boucles imbriquées (en i et en c), au cœur desquelles :

- `phase 1`, des matrices temporaires sont générées ;
- `phase 2`, des multiplications et une addition de matrices sont effectuées.

Evaluons les complexités de ces deux phases pour un i et un c donnés. Nous en déduirons la complexité globale de l'algorithme en sommant sur ces deux variables. Cet algorithme correspond à l'algorithme précédent mais en version matricielle ; nous devrions donc retrouver à la fin la même complexité opératoire.

6.1.5.3 Génération des matrices intermédiaires (`phase 1`)

Pour constituer les matrices intermédiaires, pour un couple (i,c) donné, on effectue $B_{mat\_int}(i,c)$ opérations. Au total, pour tous les i et c, le nombre de multiplications et d'additions que l'on aura effectuées tout au long de l'algorithme est noté C_{mat_int}.

```
Procédure MISE_A_JOUR(S) {

    (* initialisation de N_*(*,*,t + Δt) *)
    N_*(*,*,t + Δt):=0;
    Pour i:=0 à S faire {
    .   Pour c:=i à S faire {
    .   .   (* préparation des matrices intermédiaires : phase 1 *)
    .   .   Pour m:=i à c faire {
    .   .   .       Pour d:=i à c faire
    .   .   .           si (d < c-m+i) alors tmp_1(m,d):=(ψ_1.ψ_1)(d − i, d, c − m, c);
    .   .   .                          sinon tmp_1(m,d):=0;
    .   .   }
    .   .   Pour m:=i à c faire  {
    .   .   .       Pour l:=0 à S faire
    .   .   .           si (m < l) alors tmp_3(l,m):=φ(l − m, c);
    .   .   .                      sinon tmp_3(l,m):=0;
    .   .   }
    .   .   (* réalisation des calculs matriciels : phase 2 *)
    .   .   s(*m,*k):=multiplie_matrices(selon d,tmp_1(*m,*d).N_*k(*d, c, t));
    .   .   s(*l,*k):=scalaire_multiplie_matrice(p(c).s(*l,*k));
    .   .   s(*l,*k):=multiplie_matrices(selon m,tmp_3(*l,*m).s(*m,*k));
    .   .
    .   .   (* répercussion du calcul sur N_k(i, l, t + Δt) *)
    .   .   N_*k(i, *l, t + Δt):= addition_matrices(N_*k(i, *l, t + Δt)+s(*l,*k));
    .   }
    }
}
```

Figure 20. Algorithme utilisant des opérations matricielles

On remarque que seule la matrice tmp_1 nécessite des multiplications pour être formée :

$$B_{mat\_int}(i,c) = \sum_{m=i}^{c} \sum_{d=i}^{c-m+i} 1 = \frac{(c-i+1)*(c-i+2)}{2}$$

$$C_{mat\_int} = \sum_{i=0}^{S} \sum_{c=i}^{S} \sum_{m=i}^{c} \sum_{d=i}^{c-m+i} 1 = D_1 \ .$$

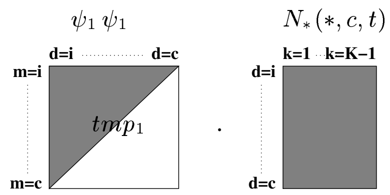
6.1.5.4 Complexité des calculs (phase 2)

Comptabilisons le nombre d'opérations réalisées au cours des produits matriciels. On ne compte pas les multiplications et additions par zéro dues aux creux dans les structures des matrices, et on peut éviter de les faire en utilisant les routines BLAS 3 adaptées aux formes des matrices considérées[12]. Pour chaque $(i,c) \in [0..S] \times [i..S]$, on a $B_{calcul}(i,c)$

12. En pratique on utilise des combinaisons des routines DGEMM et DTRMM.

opérations à effectuer. A la fin de l'algorithme, on aura effectuer C_{calcul}. Enumérons tous les calculs intervenant dans $B_{calcul}(i,c)$:

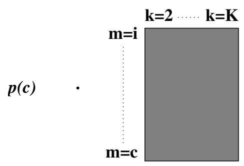
– La contribution de la ligne 18 de l'algorithme est :



$$(K-1) * \sum_{m=i}^{c} \sum_{d=i}^{c-m+i} 1$$

multiplications et additions ;

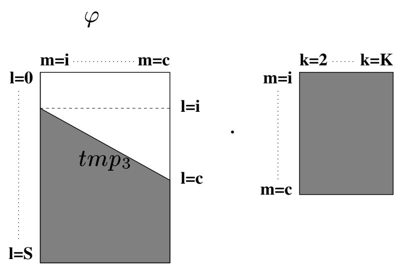
– ligne 19 :



$$(K-1) * \sum_{m=i}^{c} 1$$

multiplications ;

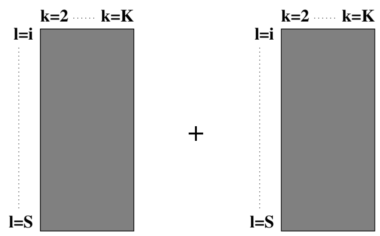
– ligne 20 :



$$(K-1) * \sum_{m=i}^{c} \sum_{l=m}^{S} 1$$

multiplications et additions ;

– ligne 23 :



$$(K-1) * \sum_{l=i}^{S} 1$$

additions.

On a pour résumer :

$$B_{calcul}(i,c) \;=\; (2K-2) \sum_{m=i}^{c} \sum_{d=i}^{c-m+i} 1 + (2K-2) \sum_{m=i}^{c} \sum_{l=m}^{S} 1 + (K-1) \sum_{l=i}^{S} 1 + (K-1) \sum_{m=i}^{c} 1.$$

Calculons le nombre total d'opérations engendrées par les calculs matriciels :

$$\begin{aligned}
C_{calcul} &= \sum_{i=0}^{S}\sum_{c=i}^{S} B_{calcul}(i,c) \\
C_{calcul} &= (2\,K-2)\sum_{i=0}^{S}\sum_{c=i}^{S}\sum_{m=i}^{c}\sum_{d=i}^{c-m+i} 1 + (2\,K-2)\sum_{i=0}^{S}\sum_{c=i}^{S}\sum_{m=i}^{c}\sum_{l=m}^{S} 1 \\
&\quad +(K-1)\sum_{i=0}^{S}\sum_{c=i}^{S}\sum_{m=i}^{c} 1 + (K-1)\sum_{i=0}^{S}\sum_{c=i}^{S}\sum_{l=i}^{S} 1 \\
C_{calcul} &= (2\,K-2)\,D_1 + (2\,K-2)\,D_2 + (K-1)\,D_3 + (K-1)\,D_4
\end{aligned}$$

6.1.5.5 Complexité totale

En ajoutant cette complexité C_{calcul} à celle trouvée pour constituer les matrices intermédiaires, on trouve le nombre total de multiplications de l'algorithme (W) :

$$W(S,K) = C_{mat\_int} + C_{calcul} = (2\,K-1)\,D_1 + (2\,K-2)\,D_2 + (K-1)\,D_3 + (K-1)\,D_4 \ .$$

$$W(S,K) = (\frac{1}{4}\,K-\frac{5}{24})\,S^4+(\frac{8}{3}\,K-\frac{9}{4})\,S^3+(\frac{37}{4}\,K-\frac{187}{24})\,S^2+(\frac{77}{6}\,K-\frac{43}{4})\,S+6\,K-5 \ .$$

En comparant à C_2 (qui correspond à la version non matricielle de l'algorithme), on a bien : $W = C_2$. On aura besoin par la suite de la complexité du noyau de calcul, que l'on nomme $CM(c,i)$, et qui vaut $B_{mat\_int}(i,c)+B_{calcul}(i,c)$:

$$CM(c,i) = (c-i+1)\,(2\,(K-1)\,(S-i+1)+\frac{c}{2}-\frac{i}{2}-2+3\,K)+(K-1)\,(S+1-i) \ .$$

Nous avons décomposé l'algorithme factorisé en k et l en un enchaînement d'opérations matricielles. Maintenant que nous possédons ceci, nous pouvons essayer de réaliser les produits matriciels de manière à minimiser encore le nombre de calculs effectués. En effet, l'ordre dans lequel on multiplie les matrices influe sur le nombre d'opérations effectuées au total.

6.1.5.6 Réordonnancement optimal des produits matriciels

A l'étape $(i,c) \in [0..S] \times [i..S]$ de l'algorithme, le calcul à réaliser est :

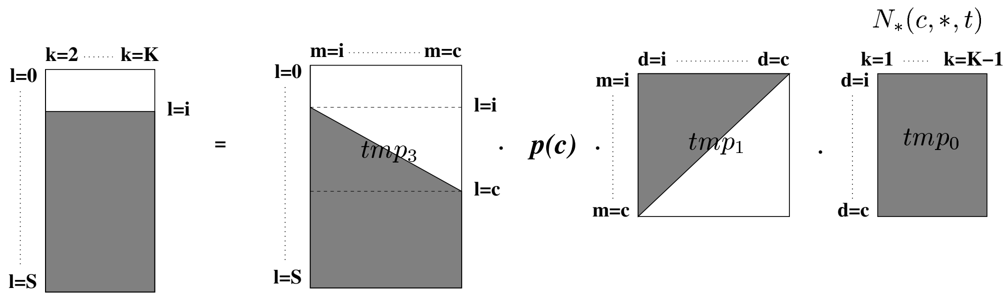


Figure 21. Noyau de calcul sous forme matricielle

Posons $M = c-i+1$. Les cinq enchaînements possibles pour les trois multiplications à effectuer sont présentés ci-après. Le facteur multiplicatif $p(c)$ peut être déplacé dans la formule sans changer le résultat (commutativité), mais la complexité se ramène alors à l'une des suivantes :

- **solution 1** : $(tmp_3\, p(c))\,(tmp_1\, tmp_0)$
  Nombre d'opérations :
  - $aux1 \leftarrow (tmp_3\, p(c))$: $\frac{M\,(M+1)}{2} + (S-c)\,M$ multiplications ;
  - $aux2 \leftarrow (tmp_1\, tmp_0)$: $\frac{(K-1)\,M\,(M+1)}{2}$ multiplications et additions ;
  - $aux1\, aux2$: $(K-1)\,(\frac{M\,(M+1)}{2} + (S-c)\,M)$ multiplications et additions.
- **solution 2** : $((tmp_3\, p(c))\, tmp_1)\, tmp_0)$
  Nombre d'opérations :
  - $aux1 \leftarrow (tmp_3\, p(c))$: $\frac{M\,(M+1)}{2} + (S-c)\,M$ multiplications ;
  - $aux2 \leftarrow (aux1\, tmp_1)$: $\frac{M\,(M+1)\,(2M+1)}{6} + (S-c)\,\frac{M\,(M+1)}{2}$ multiplications et additions ;
  - $aux2\, tmp_0$: $(K-1)\,M\,(S-i+1)$ multiplications et additions.
- **solution 3** : $((tmp_3\,(p(c)\, tmp_1))\, tmp_0)$
  Nombre d'opérations :
  - $aux1 \leftarrow (p(c)\, tmp_1)$: $\frac{M\,(M+1)}{2}$ multiplications ;
  - $aux2 \leftarrow (tmp_3\, aux1)$: $\frac{M\,(M+1)\,(2M+1)}{6} + (S-c)\,\frac{M\,(M+1)}{2}$ multiplications et additions ;
  - $aux2\, tmp_0$: $(K-1)\,M\,(S-i+1)$ multiplications et additions.
- **solution 4** : $(tmp_3\,((p(c)\, tmp_1)\, tmp_0))$
  Nombre d'opérations :
  - $aux1 \leftarrow (p(c)\, tmp_1)$: $\frac{M\,(M+1)}{2}$ multiplications ;
  - $aux2 \leftarrow (aux1\, tmp_0)$: $(K-1)\,\frac{M\,(M+1)}{2}$ multiplications et additions ;
  - $tmp_3\, aux2$: $(K-1)\,\frac{M\,(M+1)}{2} + (K-1)\,M\,(S-c)$ multiplications et additions.
- **solution 5** : $(tmp_3\,(p(c)\,(tmp_1\, tmp_0)))$
  Nombre d'opérations :
  - $aux1 \leftarrow (tmp_1\, tmp_0)$: $\frac{(K-1)\,M\,(M+1)}{2}$ multiplications et additions ;
  - $aux2 \leftarrow (p(c)\, aux1)$: $M\,(K-1)$ multiplications ;
  - $tmp_3\, aux2$: $(K-1)\,\frac{M\,(M+1)}{2} + (K-1)\,M\,(S-c)$ multiplications et additions.

Les solutions **1** et **2** peuvent être exclues dès maintenant de notre étude. En effet, elles ont un coût strictement supérieur aux solutions **4** et **3** respectivement. Effectuons donc un bilan des coûts sur les solutions **3**, **4** et **5** ; on ne compte que les opérations dues aux multiplications entre les termes tmp_0, tmp_1, $p(c)$, tmp_3, et non pas le coût des opérations nécessaires pour former et remplir les matrices qui est en fait identique pour chaque solution. Soient s_3, s_4, s_5 les nombres d'opérations réalisées pour les solutions **3**, **4** et **5** pour (i, M) fixé.

- **solution 3**: $((tmp_3.(p(c).tmp_1)).tmp_0)$

$$s_3(i,M) = -\frac{1}{3}\,M^3 + (\frac{3}{2} + S - i)\,M^2 + (\frac{11}{6} + S - i + 2\,(K-1)\,(S - i + 1))\,M\ .$$

- **solution 4**: $(tmp_3.((p(c).tmp_1).tmp_0))$

$$s_4(i,M) = \frac{1}{2}\,M^2 + (2\,(K-1)\,S + (4 - 2\,i)\,(K-1) + \frac{1}{2})\,M\ .$$

- **solution 5**: $(tmp_3.(p(c).(tmp_1.tmp_0)))$

$$s_5(i,M) = (2\,S + 5 - 2\,i)\,(K-1)\,M\ .$$

Appelons H_3, H_4, H_5 les complexités algorithmiques totales des solutions **3**, **4** et **5** ($H_j = \sum_{i=0}^{S}\sum_{M=1}^{S-i+1} s_j(i,M)$). En sommant les expressions s_3, s_4, s_5, on déduit[13] :

$$H_3 = \frac{S^5}{20} + O(S^4)\ ;\ H_4 = \frac{K\,S^4}{4} - \frac{5\,S^4}{24} + O(K\,S^3)\ ;\ H_5 = \frac{K\,S^4}{4} - \frac{S^4}{4} + O(K\,S^3).$$

Il apparaît donc que la solution **5** est la plus économe pour les grandes valeurs de S au niveau des multiplications du noyau de calcul. On garde finalement cette solution **5** qui correspond au dernier algorithme présenté à la sous-Section précédente (p. 66). L'algorithme matriciel est donc de complexité $W(S,K)$.

6.1.6 Conclusion

Au cours de ce chapitre, nous avons fait apparaître des précalculs qui réduisent le nombre d'opérations à effectuer lors de la mise à jour de $N_*(*,*,t+\Delta t)$. Nous nous sommes ainsi ramenés à un calcul pour lequel aucune fonction n'est appelée, et qui utilise uniquement des valeurs stockées en mémoire. Ensuite, nous avons factorisé les calculs selon les deux variables k et l, ce qui nous a permis de passer d'une complexité en $O(K\,S^5)$ à $O(K\,S^4)$ en divisant asymptotiquement et pour K=10, la complexité par un facteur $0.29\,S$. Puis, l'algorithme de mise à jour a été transformé en un algorithme matriciel. Le calcul ainsi transformé permet de réaliser le même nombre d'opérations qu'auparavant en moins de temps grâce à la bibliothèque BLAS qui optimise ce type de calcul. Cette analyse séquentielle est préliminaire aux solutions parallèles qui sont envisagées dans la suite de ce chapitre.

13. Ces résultats de calculs symboliques peuvent être obtenus avec un logiciel tel que *Maple* par exemple.

6.2 Solution parallèle : distribution 1D

6.2.1 Introduction

$$N_*(*, i, t+\Delta t) = \sum_{c=i}^{S(t)}$$

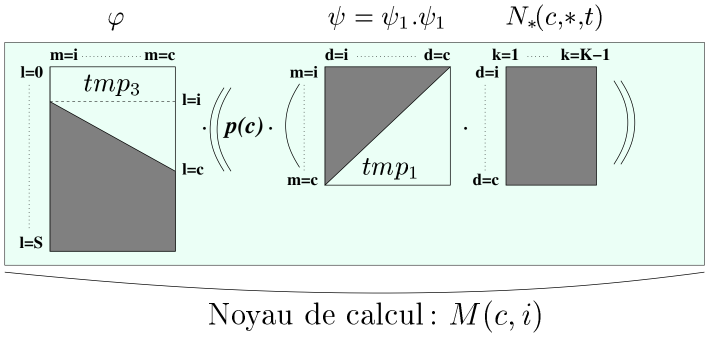


Figure 22. Mise à jour de $N_*(*,*,t)$ utilisant une expression matricielle

La mise à jour de $N_*(*,*,t)$ est réécrite dans la Figure 22. Les matrices intermédiaires tmp_1, tmp_3 contiennent par construction des termes nuls qui sont représentés par la couleur gris clair dans la Figure 22. Les dépendances entre les différents indices de boucles impliquent que certains espaces ne sont pas couverts par les indices, ce qui introduit des termes nuls de construction dans les matrices intermédiaires. En reprenant par exemple l'équation (23), on note que l'indice d varie entre i et $c-m+i$; ceci explique par exemple que pour $d > c - m + i$ dans la matrice $tmp1$ les coefficients sont à zero. Par définition, on pose $M(c, i)$ le noyau de calcul contenant les multiplications de matrices se situant à l'intérieur de la somme. L'algorithme auquel on est conduit est celui de la Figure 23 (rappelons qu'à t donné, on a un $S = p_{sup}(t)$). Pour conserver la performance des calculs BLAS que l'on a fait apparaître, on parallélise en préservant le noyau de calcul $M(c, i)$. Si l'on considère l'ensemble des tâches $M(c, i)$ qui sont à réaliser, l'algorithme est à grain moyen avec $(S+1)(S+2)/2$ noyaux à distribuer sur les P processeurs. Par contre, si l'on considère la parallélisation de la boucle en i de l'algorithme, on se ramène à $S+1$ tâches à répartir, d'où un parallélisme à plus gros grain. Ces différentes possibilités vont être envisagées tour à tour. Nous présentons d'abord la parallélisation de la boucle en i.

```
Procédure Mise_à_jour_de_N(S) {
   Pour i:=0 à S faire {
   .   Pour c:=i à S faire {
   .   .   Initialisation des matrices tmp1, tmp3 ;
   .   .   Aux ← tmp3.(p(c).(tmp1.N*(c,*,t))) ;   /* tâche M(c,i) */
   .   .   N*(*,i,t+Δt) ← N*(*,i,t+Δt) + Aux ;
   .   }
   }
}
```

Figure 23. Algorithme de mise à jour de $N_*(*,*,t)$

6.2.2 Distribution des calculs et des données

Par la suite, la donnée $N_*(c,*,t)$ est appelée «*plaque ligne* (c,t)», et $N_*(*,i,t)$ est appelée «*plaque colonne* (i,t)». A l'itération $(i=i_0, c=c_0)$ de l'algorithme (Figure 23), la plaque ligne (c_0,t) est utilisée pour calculer une contribution pour la plaque colonne $(i_0, t+\Delta t)$. La plaque colonne $(i_0, t+\Delta t)$ dépend de l'ensemble des plaques lignes (c,t) pour lesquelles $S \geq c \geq i_0$ (voir Figure 24). Les données sont donc les plaques lignes et les plaques colonnes constituent les résultats. Ceci implique que l'on ne peut pas écraser les données du temps t par celles du temps $t+\Delta t$; on doit disposer de deux structures de données pour $N_*(*,*,t)$ lors de sa mise à jour (une pour le temps t, et l'autre pour le temps $t+\Delta t$). Ceci ne représente globalement (en réel double-précision) que $8\,K\,S\,(S+1)$ octets, soit 49 MO pour $S=800$, à distribuer sur la machine parallèle (le nombre d'éléments dans N étant de $K\,S\,(S+1)/2$).

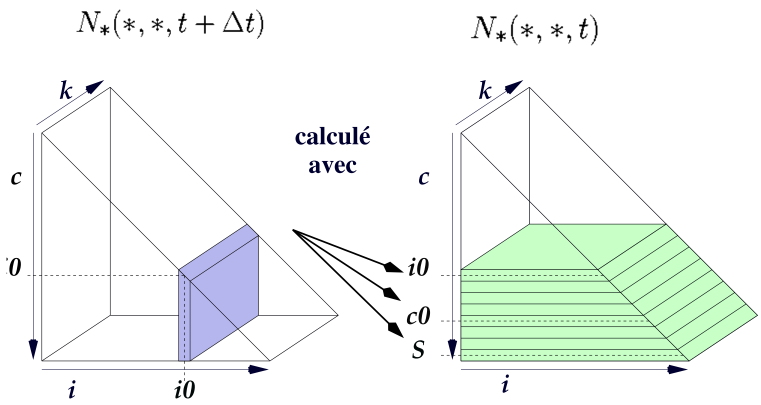


Figure 24. Dépendances des données

En ce qui concerne le flot de calcul, les tâches individuelles M sont indépendantes entre elles. Les tâches $M(*,i_0)$ servent au calcul des contributions de la plaque colonne $N_*(*,i_0,t+\Delta t)$. Il est donc intéressant de placer sur le même processeur les tâches de calcul $M(*,i_0)$ et la sous-structure à laquelle elles contribuent $N_*(*,i_0,t+\Delta t)$. Il est possible ainsi de réaliser les additions des contributions `Aux` à une plaque colonne $N_*(*,i_0,t+\Delta t)$ de l'algorithme localement. Les indices i_0 sont distribués sur les processeurs avec une distribution *en serpentin* (*snake* en anglais). Ainsi on obtient un équilibrage de la charge de bonne qualité. Une distribution cyclique peut aussi être envisagée, mais elle donne en de moins bons résultats dans notre cadre[14]. On place les calculs les plus coûteux en premier

14. Soit $2\,P$ taches indicées de 0 à $2\,P-1$ dont la fonction de coût en calcul $\Upsilon(.)$ est strictement décroissante. Lorsque l'on distribue cycliquement les tâches sur P processeurs, le processeur 0 a le plus lourd calcul a effectué de coût $c_{cycl//} = \Upsilon(0) + \Upsilon(P)$. D'un autre côté, on remarque que le coût en calcul d'un processeur $z \in [0..P-1]$ avec la distribution en serpentin est inférieur à $c_{cycl//}$ à cause de la décroissance de Υ : $c_{cycl//} > \Upsilon(z) + \Upsilon(2P-1-z)$. Avec la distribution en serpentin, le coût en calcul sur le processeur ayant le plus gros calcul est $c_{serp//}$; on a $c_{cycl//} > c_{serp//}$. Le résultat ce généralise lorsque le nombre de tâches est divisible par $2\,P$. L'équilibrage issu de la distribution cyclique est donc nécessairement de moins bonne qualité que celle en serpentin.

Numéros des processeurs	**0**	**1**	**2**	**3**
Numéros des plaques colonnes i_0 locales	0	1	2	3
	4	5	6	7
	8	9	10	11
	12	13	14	15
	16	17	18	19
	20	21		

(a) cyclique

Numéros des processeurs	**0**	**1**	**2**	**3**
Numéros des plaques colonnes i_0 locales	0	1	2	3
	7	6	5	4
	8	9	10	11
	15	14	13	12
	16	17	18	19
			21	20

(b) serpentin

Figure 25. Distributions possibles des tâches $M(*, i_0)$ et des plaques lignes $(i_0, t{+}\Delta t)$ sur les processeurs

sur les processeurs ce qui correspond aux tâches $M(*, i_0)$ avec des indices i_0 faibles (un exemple où $S = 21$ est présenté Figure 25).

Il a été décidé de stocker dans la structure $N_*(*, *, t)$ au temps t des plaques lignes, tandis qu'au temps $t + \Delta t$ la structure $N_*(*, *, t + \Delta t)$ calculée contient des plaques colonnes. L'accès aux données lors des calculs est ainsi facilité. Finalement des plaques lignes appartenant à la structure $N_*(*, *, t)$ et des plaques colonnes de la structure $N_*(*, *, t{+}\Delta t)$ doivent être réparties conjointement sur les processeurs. Tout comme pour les plaques colonnes, on utilise une distribution en serpentin pour les plaques lignes. L'algorithme a été reformulé pour tenir compte explicitement de ces deux distributions (voir Figure 26). Dans celui-ci, la fonction `mapc(c)` renvoie le numéro du processeur à qui appartient la plaque ligne (c, t) . De manière similaire `mapi(i)` désigne le numéro du processeur propriétaire de la plaque colonne $(i, t{+}\Delta t)$.

Comme l'algorithme utilise des plaques lignes et génère des plaques colonnes, une étape de transposition est nécessaire pour réaliser une série de mises à jour (à la fin de l'algorithme Figure 26). Considérons le vecteur $N_*(c_0, i_0, t + \Delta t)$ qui a été placé sur le processeur `mapi(i`$_0$`)` ; il doit simplement être transmis au processeur `mapc(c`$_0$`)`. La transposition consiste donc à déplacer tous les vecteurs $N_*(c_0, i_0, t{+}\Delta t)$ vers leur nouvelle place ; ainsi le coût en communication est le même que l'espace mémoire occupé par $N_*(*, *, t)$, c'est-à-dire $\frac{KS^2}{2}$ (soit 24 MO au maximum pour $S <= 800$ en double précision). Cette transposition globale est effectuée à l'aide de la routine `MPI_AllToAll` de la bibliothèque MPI [52].

Comme on peut le constater dans la boucle en c de l'algorithme, deux étapes peuvent être distinguées : une de communication, l'autre de calcul. La diffusion totale (*broadcast*) utilisée dans l'étape de communication est en pratique anticipée et se déroule en communication non bloquante. Ceci permet de tirer profit d'un recouvrement calcul-communication. Un anneau logique est défini parmi l'ensemble des processeurs. Sur cet anneau, un processeur a un voisin en amont et un autre en aval. Les plaques lignes sont reçues depuis l'amont, stockées pour un calcul futur, puis transmises en aval. Entre les plages de temps CPU consacrées aux noyaux de calculs, on prend connaissance des plaques lignes reçues et on les réémet de manière asynchrone et non bloquante. D'autre part, on peut remarquer

```
Procédure Mise_à_jour_1D_de_N(S) {
 Pour c:=0 à S faire {
 .  /* étape de communication */
 . Si (mapc(c)=my_id) {
 .  Diffusion totale de N_*(c,*,t)
 . } Sinon {
 .  Reçoit N_*(c,*,t)
 . }
 .  /* étape de calcul */
 . Pour i:=0 à c faire {
 . . Si (mapi(i)=my_id) {
 . .  /* tâche M(c,i) */
 . ..Initialisation des matrices tmp_1,tmp_3;
 . . Aux ← tmp_3.(p(c).(tmp_1.N_*(c,*,t)));
 . . N_*(*,i,t+Δt) ← N_*(*,i,t+Δt) + Aux;
 . .}
 . }
 }
 Transposition de N_*(*,*,t) des plaques
 colonnes vers les plaques lignes;
}
```

Figure 26. Mise à jour de $N_*(*,*,t)$ avec la distribution 1D

qu'en faisant tous les calculs concernant la plaque colonne c en une fois (la boucle en i est effectuée avant de passer à $c+1$), on augmente la localité temporelle de la donnée $N_*(c,*,t)$, et on optimise ainsi les effets de cache.

Les diffusions asynchrones anticipées des plaques lignes sur les P processeurs impliquent un coût de communication de $\frac{KPS^2}{2}$. D'une part cette quantité est très inférieure à la complexité $W(S,K)$, et d'autre part un recouvrement calcul-communication total est donc effectivement réalisable. Le volume de données communiquées pour une transposition de $N_*(*,*,t)$ est de $\frac{KS^2}{2}$; bien que non recouverte, on peut négliger le temps pris par cette étape lorsque le nombre de processeurs impliqués reste peu important[15]. Lorsque P augmente et que le volume de données est constant, on verra que ce coût n'est plus minime. L'implémentation confirme ces deux affirmations sur nos machines de validation de type IBM SP2 et SP3 (description des machines en section 6.2.5). La communication n'altère donc pas les performances du programme parallèle ; nous allons maintenant étudier l'équilibrage des charges et la scalabilité du programme.

15. La primitive MPI_AllToAll utilisée classiquement se décompose en P-1 étapes. A chaque étape e les processeurs i et $(i-e \bmod P-1)$ échangent un message de taille $\lceil V/P \rceil$. Si α est la latence et β la bande passante, le coût en temps est $(P-1)\,(\alpha+\beta\,\lceil \frac{V}{P} \rceil)$. Pour P grand, cet algorithme aura donc un coût en temps en $\Theta(P)$.

6.2.3 Coûts en calcul sur chaque processeur

On a déterminé la complexité $CM(c,i)$ des noyaux de calculs $M(c,i)$ en comptant le nombre de multiplications et d'additions qui y sont réalisées. Le noyau de calcul utilisant les routines GEMM et TRMM des BLAS 3, on suppose effectivement qu'il existe une relation linéaire entre le temps de calcul de $M(c,i)$ et le nombre d'opérations effectuées[16] :

$$CM(c,i) = (c-i+1)\left[(2K-\frac{3}{2})(c-i+2)+K-1+(2K-2)(S-c)\right]+(K-1)(S-i+1).$$

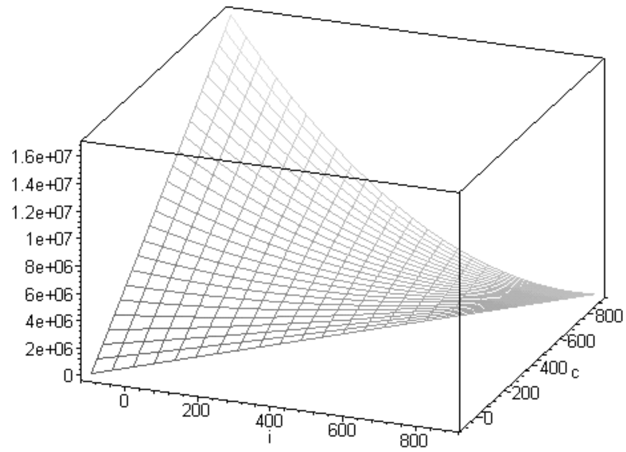


Figure 27. Courbe $CM(i,c)$ indiquant les coûts en calcul des noyaux $M(i,c)$ pour $S = 800$, $K = 10$

Comme l'illustre la Figure 27, la fonction CM est positive, décroissante en i et croissante en c. D'autre part elle est convexe en i et c. Connaissant le nombre d'opérations à réaliser pour chaque tâche M, et la répartition de celles-ci sur les processeurs, nous pouvons calculer combien d'opérations chaque processeur doit réaliser.

Comme le placement des calculs sur les processeurs a été réalisé statiquement (il est décidé une fois pour toute dès le début des calculs de mise à jour de $N_*(*,*,t)$), une étude théorique portant sur l'efficacité est possible. Pour calculer cette efficacité, on détermine quel processeur a le plus d'opérations à effectuer et le nombre d'opérations à effectuer sur celui-ci. Par la suite, nous utilisons la Proposition 1 suivante (elle découle de la Proposition 3 en annexe p. 175) :

Proposition 1. *Soit P le nombre de processeurs ; un nombre fini $(s+1)$ de tâches est distribué sur ces processeurs avec une distribution en serpentin (on suppose que $s+1=2Pq$ avec $q \in \mathbb{N}^*$). Soit $g(i)$ la fonction donnant le nombre d'opérations à effectuer pour une tâche d'indice i. On suppose $g(i)$ convexe décroissante sur $[0,s]$. Soit* charge(p_x) *la fonction*

16. On néglige dans un premier temps les effets non-linéaires liés à la réutilisation des caches.

qui donne le coût en nombre d'opérations sur chaque processeur $p_x \in [0, P-1]$ *; il vient*

$$\mathsf{charge}(p_x) = \sum_{j=1}^{\left(\frac{s+1}{2P}-1\right)} g(2\,(j-1)\,P + p_x) + g(2\,j\,P - 1 - p_x)\ .$$

La fonction $\mathsf{charge}(p_x)$ *est alors positive décroissante sur* $p_x \in [0, P-1]$.

En pratique, on ne dispose pas toujours de l'hypothèse $S+1=2\,P\,q$. On peut alors appliquer la Proposition 1 pour les $s'+1 = \lfloor (S+1)/(2P) \rfloor\, 2\,P$ premières tâches. On considère alors que la distribution des $S-s'$ tâches restantes qui ont les coûts les plus faibles (g est décroissante) ne vient pas infirmer la propriété comme quoi, de fait, la fonction $\mathsf{charge}(p_x)$ est positive décroissante sur $[0, P-1]$.

On a distribué sur les processeurs les groupes de tâches $M(*, i)$. Appelons $g'(i)$ la complexité du groupement de tâches : $g'(i) = \sum_{c=i}^{S} CM(c, i)$. La fonction $g'(i)$ est une somme de fonctions décroissantes convexes en i, elle est donc elle-même une fonction décroissante convexe. Comme on distribue les indices i sur les processeurs selon la méthode du serpentin, on se trouve dans les conditions d'application de la Proposition 1 avec $s = S$. On en déduit donc directement que la charge des processeurs $[0, P–1]$ est une suite positive décroissante. Le processeur 0 est donc celui qui a le plus de calculs à réaliser.

Si les temps d'initialisation des matrices intermédiaires, les effets de caches, l'accélération des calculs due au BLAS ne biaisent pas ce résultat, les temps d'exécution des processeurs doivent être de la même manière une fonction décroissante des indices des processeurs. Notre implémentation a cette propriété comme l'illustre la Figure 29 (p. 79).

Enfin, on aurait pu envisager une distribution des tâches de calculs $M(*, i)$ à l'aide d'un algorithme de placement du type *bin packing* [27], ce qui conduirait à une distribution encore meilleure que celle de type serpentin. Cependant cette méthode se prête moins facilement à une étude théorique et l'efficacité parallèle est moins directement accessible. D'autre part, les indices i vont être placés de manière plus irrégulière, ce qui impliquera une répartition des données et des schémas de communication plus inégaux, et donc un algorithme plus difficile à gérer en pratique.

6.2.4 Efficacité, scalabilité

On va maintenant déduire les conditions pour lesquelles on a une exécution efficace du programme parallèle, en étudiant l'efficacité et la scalabilité théoriques de l'algorithme.

Soit $T_{in}(S, K, c, i)$ la complexité en temps d'une itération de la boucle en c la plus interne de l'algorithme de la Figure 23 (p. 72), et soit $T_{out}(S, K, i)$ la complexité de la boucle en i externe. La complexité totale en temps de la mise à jour de $N_*(*, *, t)$ est notée $T_{seq}(S, K)$. Comme évoqué précédemment, $T_{in}(S, K, c, i)$ ne correspond pas exactement à la complexité en nombre d'opérations pour trois raisons : l'initialisation des matrices intermédiaires n'est pas négligeable, les effets de caches dûs aux réutilisations successives de certaines matrices dans le noyau de calcul ont un impact sur les temps d'exécution, et enfin l'utilisation des BLAS modifie le coût en temps. On en déduit que la complexité en temps $T_{seq}(S,K)$ peut être différente du nombre d'opérations de la mise à jour $W(S,K)$, bien que ces deux quantités soient du même ordre asymptotique ; on pose $T_{seq}(S,K) = \Theta(S^4)$. La constante K est fixée pour une simulation donnée. On observe que le nombre d'opérations $CM(c,i)$ est un polynôme homogène du second degré en $i,\ c,$ et S. Il en est de même pour la complexité en temps, bien que les coefficients du polynôme puissent être différents et fonction de K. On a aussi les équations suivantes :

$$T_{out}(S,K,i) = \sum_{c=i}^{S} T_{in}(S,K,c,i) \quad \text{et} \quad T_{seq}(S,K) = \sum_{i=0}^{S} T_{out}(S,K,i) \quad .$$

Soit $T_{//}(S,K,P)$ le temps d'exécution parallèle sur P processeurs. Par commodité, on suppose que $S+1$ est un multiple de $2P$. Le temps d'exécution parallèle correspond au temps d'exécution du processeur 0 comme il a été dit précédemment. La distribution en serpentin conduit à l'expression suivante :

$$T_{//}(S,K,P) = \sum_{j=0}^{\frac{S+1}{2P}-1} T_{out}(S,K,2Pj) + T_{out}(S,K,2Pj+2P-1) \quad .$$

$T_0(S,K,P) = P.T_{//}(S,K,P) - T_{seq}(S,K)$ constitue dans cette étude (la communication n'engendre pas de surcoût significatif) le surcoût induit par le déséquilibre des charges. En utilisant la forme polynômiale de $T_{in}(S,K,c,i)$, quels que soient les coefficients de ce polynôme, un calcul formel donne[17]

$$T_0(S,K,P) = \Theta(P^2\,S^2) \quad .$$

Ceci montre que le surcoût pour chaque mise à jour augmente de manière quadratique en fonction du nombre de processeurs P et de la taille du problème S. Soit $\gamma(K)$ un facteur que l'on souhaite déterminer (on rappelle que $T_{seq}(S,K) = \Theta(S^4)$) tel que

$$\frac{T_0(S,K,P)}{T_{seq}(S,K)} \approx \gamma(K)\frac{P^2}{S^2} \quad . \tag{24}$$

L'efficacité (définie dans [24]) peut s'exprimer par

$$E(S,K,P) = \frac{1}{1+\frac{T_0(S,K,P)}{T_{seq}(S,K)}} \quad \text{donc } E(S,K,P) \approx \frac{1}{1+\gamma(K)\frac{P^2}{S^2}} \quad . \tag{25}$$

Cette relation lie directement efficacité, nombre de processeurs, et valeur de S. Pour $K=10$, des séries d'exécutions tests sur IBM SP2 faisant varier S ont permis de déterminer avec une regression linéaire que $\gamma(10) \approx 14$. On dispose ainsi d'une fonction qui approxime

17. Le résultat peut être obtenu en utilisant *Maple*, par exemple. L'ensemble des calculs symboliques réalisés dans cette Partie sont disponibles sur le Web à l'adresse suivante `http://dept-info.labri.u-bordeaux.fr/~latu/these/etude.mws`.

correctement l'efficacité observée lors des simulations (voir Figure 28). Si l'on se fixe une efficacité minimale de 80%, on peut écrire les relations :

$$E(S, 10, P) > 0.80 \ ; \qquad 1 > 56\frac{P^2}{S^2} \ ; \qquad \frac{S}{P} > \sqrt{56} \approx 7.5 \ .$$

On obtient ainsi une relation entre S et P qui donne les conditions pour une exécution efficace. Dans le cas de 32 processeurs, la mise à jour parallèle de $N_*(*, *, t)$ s'effectue avec une efficacité supérieure à 80 % pour $S \in [240, 800]$ (*c.à.d* pour des problèmes dont le coût est compris entre 8 GFLOP et 951 GFLOP) ; pour 64 processeurs, S doit être dans l'intervalle $[479, 800]$ (problèmes de 123 à 951 GFLOP). Jusqu'à 64 processeurs, on observe donc que les mises à jour réellement coûteuses sont réalisées avec une efficacité parallèle de bonne qualité.

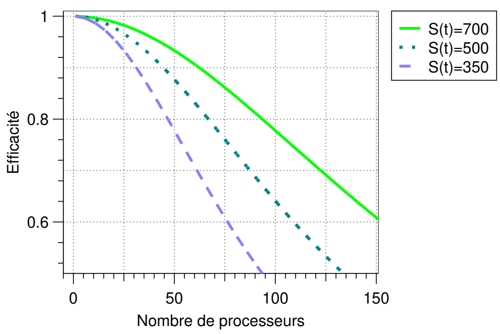


Figure 28. Efficacité approchée avec $\gamma(K=10) = 14$

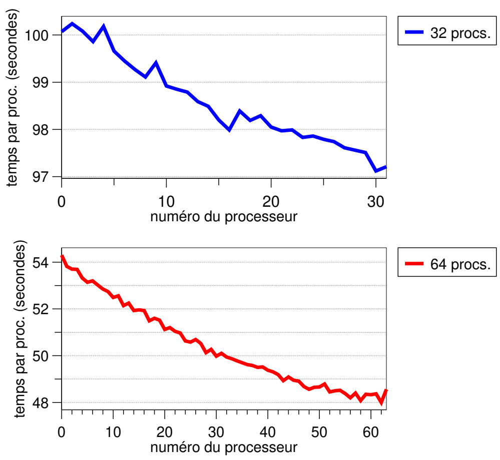


Figure 29. Temps de calcul pour chaque processeur de la mise à jour de $N_*(*, *, t)$ $(S=700)$ pour les simulations du Tableau 6.1 (*cf.* p. 81)

Comme $T_{seq}(S, K) = S^4$, et $T_0(S, K, P) = \Theta(P^2 S^2)$, alors $T_0(S, K, P) \ll T_{seq}(S, K)$ lorsque P est petit devant S. Sous cette hypothèse on en déduit que l'algorithme est à coût optimal :

$$P.T_{//}(S, K, P) = \Theta(T_{seq}(S, K)) \ .$$

Exprimons l'iso-efficacité en réécrivant l'équation (25), on a

$$1 + \gamma(K)\frac{P^2}{S^2} = \frac{1}{E(S, K, P)} \ , \tag{26}$$

$$\text{d'où} \quad S = \sqrt{\gamma(K)\frac{E(S, K, P)}{(1 - E(S, K, P))}} \quad P \ . \tag{27}$$

A l'aide de cette formule d'iso-efficacité pour laquelle on fixe $E(K,S,P)$ et K, on a plus simplement $S = \Theta(P)$. Lorsque le nombre de processeurs est doublé, S doit de la même manière être doublé pour conserver la même efficacité, ce qui est illustré dans la Figure 28 (pour $S = 350$ et $S = 700$).

6.2.5 Machines utilisées

Dans un premier temps, de nombreuses simulations ont été réalisées sur deux IBM SP2. Au LaBRI[18], la machine possédait 16 Power 2 nœuds fins (66 MHz), avec 64 ou 128 Mo de mémoire. Au CINES[19], il y avait 207 Power 2 Super Chip nœuds fins (120 MHz), avec 256 Mo de mémoire ; un switch TB3 assurait l'interconnexion des nœuds. Les résultats présentés ici ont été obtenus sur la machine du CINES. Le code a été développé en FORTRAN 90 avec le compilateur XL Fortran et utilise la bibliothèque MPI (version propriétaire de IBM).

Ensuite, la machine cible est devenu l'IBM SP3. Le SP3 du CINES est composé de 28 nœuds SMP (Symmetric Multi Processing) de calcul interconnectés par un switch Colony. Chaque nœud IBM NH2 dispose de 16 processeurs qui accèdent à une mémoire partagée de 16 GO et utilise un unique système d'exploitation. Les 448 processeurs[20] sont cadencés à 375 Mhz. Au LABRI, une plus petite configuration de 7 nœuds WH2 (375Mhz) a été employée. La différence essentielle entre ce IBM SP3 et le précédent consiste en des tailles de cache de premier et second niveau et de mémoire plus faibles, et le fait qu'un nœud ne possède que 4 processeurs[21]. D'autre part, le réseau reliant ces nœuds est un GigaEthernet.

Enfin, certains tests ont été réalisés sur la SGI ORIGIN 3800 du CINES et les 8 nœuds à base de Power 4 de l'IDRIS. La machine SGI a 512 processeurs 500 Mhz avec une mémoire partagée de 512 MO. La machine de l'IDRIS est constituée de nœuds SMP de 32 processeurs chacun se partageant plus de 64 GO. Les processeurs ont une fréquence horloge de 1.5 Ghz.

6.2.6 Expérimentation numérique sur IBM SP2

Exceptée la mise à jour de $N_*(*,*,t)$, de nombreuses autres parties du code ont été parallélisées, et il ne reste que quelques opérations séquentielles. De 32 à 64 processeurs, un très fort pourcentage du temps total reste consacré au calcul de la mise à jour de $N_*(*,*,t)$ (entre 80 % et 90 %). Pour des simulation réalistes, la valeur de S est 700 durant une grande partie du temps d'exécution de la simulation, et il est donc important d'avoir de bonnes performances dans cette configuration (voir Tableau 6.1).

18. Laboratoire Bordelais de Recherche en Informatique.
19. Centre Informatique National de l'Enseignement Supérieur (Montpellier).
20. Des détails de l'architecture des processeurs est donné en annexe p. 189.
21. Au total la machine compte donc 28 processeurs.

	Nombre de processeurs: P				
	16	32	64	112	192
Temps (sec.)	201,0	102,8	55,4	37,6	28,5
Efficacité relative	100,0 %	97,8 %	90,8 %	76,4 %	58,7 %

Tableau 1. Temps d'exécution, efficacité relative par rapport à 16 processeurs et nombre moyen d'opérations par seconde et par processeur pour une mise à jour de $N_*(*,*,t)$ ($S=700$, $K=10$ *i.e.* 560 GFLOP)

Une mise à jour de $N_*(*,*,t)$ avec $S=700$ atteint par exemple une efficacité relative de 91 % sur 64 processeurs et 59 % sur 192 processeurs. Pour une simulation complète, on remarque que le temps d'exécution est à peu près divisé par deux lorsque le nombre de processeurs est doublé. Cependant, la performance des processeurs n'atteint que le tiers des performances de crête. Il y a deux raisons à cela : premièrement le remplissage des matrices intermédiaires prend du temps, et deuxièmement les multiplications BLAS se font avec une des dimensions égale à $K{-}1{=}9$ (Figure 22), ce qui n'autorise pas les plus grandes vitesses de calcul. Le temps d'exécution pour une simulation complète et coûteuse[22] (qui comprend 88 pas de temps avec $S=700$) est de 38600 secondes sur 8 processeurs et 5500 secondes sur 64 processeurs.

6.2.7 Précision des calculs

Lorsqu'on change le nombre de processeurs, les calculs parallèles sont ordonnés différemment sur les processeurs, ce qui peut causer des différences minimes dans les résultats. De plus, le modèle hôte-parasite est non-linéaire et les résultats de la simulation sont sensibles aux paramètres d'entrée[23]. En utilisant la double précision, toutes les variables que l'on observe durant la simulation demeurent identiques sur 9 chiffres significatifs lorsque le nombre de processeurs change. Pour la simple précision, si le nombre de processeurs est modifié, les résultats ne différent pas uniquement sur les 2 premiers chiffres significatifs. La double précision améliore donc notablement la précision des résultats mais le coût en temps est de 60 % de plus sur le SP2 et 40 % sur le SP3 (la bande passante vers la mémoire comptabilisée en nombre de flottants par seconde est en effet divisée par deux). La stabilité numérique est importante du point de vue des biologistes et des mathématiciens : les phénomènes observés dans les simulations numériques ne doivent pas provenir d'artéfacts de calcul et le modèle doit être fidèlement reproduit. La double précision (réels codés sur 64 bits) a donc été conservée.

22. Les simulations coûteuses sont celles pour lequel S est égal à *letal* sur de nombreux pas de temps. Elles correspondent à des épizooties qui s'inscrivent dans la durée.
23. Voir aussi p. 155.

6.2.8 Conclusion

Après une analyse des calculs effectués par un simulateur numérique concernant un système hôte-parasite, nous avons pu élaborer une solution parallèle. Une simulation complète et coûteuse dure moins de 11 heures sur 8 processeurs et 1 heure 30 sur 64 processeurs. L'analyse de performance a permis de montrer l'efficacité et l'extensibilité de l'algorithme parallèle ; l'efficacité relative atteint 76,4 % avec 112 processeurs.

Les simulations réalisées avec ce simulateur ont permis d'observer certaines dynamiques, en accord avec les observations de terrain, qui n'étaient pas accessibles avec le simulateur séquentiel (voir Section IV-8.1.7 p. 131).

6.3 Solution parallèle : distribution 2D

Considérons maintenant la parallélisation des deux boucles de l'algorithme de la Figure 23 à la fois en i et en c. On pose toujours $S = p_{sup}(t)$ pour l'étude algorithmique de la mise à jour de N au temps t. Pour cet algorithme à grain moyen, on a $(S+1)(S+2)/2$ tâches à répartir sur P processeurs. Ce schéma est nommé 2D car comme nous allons le voir, les couples d'indices (c, i) correspondant au calcul $M(c, i)$ sont distribués sur une grille de processeurs. Le grain de ce schéma étant plus fin que celui précédemment décrit, on compte améliorer la scalabilité par rapport au schéma 1D en exploitant plus de parallélisme. Ainsi, l'apport de cette deuxième distribution consiste en un meilleur équilibrage des charges. Contrairement à ce que l'on pourrait supposer, on verra qu'elle réduit aussi le volume de communications à effectuer et offre la possibilité d'exploiter des communications rapides localement (architecture de type SMP par exemple).

6.3.1 Distribution des données et des calculs

6.3.1.1 Grille de processeurs

Deux boucles parallèles doivent être réalisées sur un ensemble de P processeurs. Pour cela, on travaille sur une grille de processeurs comprenant $|G|$ lignes dans la première dimension, et $|H|$ colonnes dans la seconde ($|H|\,|G| = P$). La Figure 30 illustre la répartition des processeurs $p_{e,u}$.

	h_0	h_1	h_2	h_3	h_4	h_5
g_0	$p_{0,0}$	$p_{0,1}$	$p_{0,2}$	$p_{0,3}$	$p_{0,4}$	$p_{0,5}$
g_1	$p_{1,0}$	$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	$p_{1,4}$	$p_{1,5}$
g_2	$p_{2,0}$	$p_{2,1}$	$p_{2,2}$	$p_{2,3}$	$p_{2,4}$	$p_{2,5}$

Figure 30. Grille des processeurs : $p_{.,.}$ désigne chaque processeur, $g_.$ et $h_.$ correspondent aux groupes de processeurs

Les indices de la boucle en i de l'algorithme sont répartis sur $|G|$ ensembles de processeurs. L'algorithme exact de placement des indices sur les deux ensembles de processeurs sera discuté plus tard. Soit e un numéro de colonne de la grille de processeurs ($e \in [0, |H| - 1]$), le groupe de processeurs h_e est responsable des indices c d'un ensemble DH_e. De même, si u est le numéro d'une ligne de la grille de processeurs, le groupe g_u est responsable des indices i de l'ensemble DG_u. On remarque que le processeur appartenant à la fois à g_u et h_e est nommé $p_{e,u}$. Ainsi ce processeur $p_{e,u}$ doit effectuer les tâches $M(c, i)$ avec $i \in DG_u$ et $c \in DH_e$. La fonction de répartition qui va donner u et e en fonction de i et c doit tenir compte notamment de l'équilibrage des charges et des accès aux données $N_*(c, *, t)$ et $N_*(*, i, t + \Delta t)$ lors de la mise à jour de N. On observe que le schéma 1D correspond au cas où $|H| = 1$ et $|G| = P$.

6.3.1.2 Distribution des données

On a vu (algorithme de la Figure 23) que les tâches $M(*, i_0)$ calculent des contributions à la plaque colonne $(i_0, t+\Delta t)$. Comme dans le cas 1D, le groupe de processeurs g_u étant responsable des indices $i_0 \in DG_u$, il est intéressant d'affecter la plaque colonne $(i_0, t+\Delta t)$ et les tâches $M(*, i_0)$ au même groupe de processeurs afin de réaliser l'addition de la sous-structure $N_*(*, i_0, t + \Delta t)$ localement et de réduire en conséquence les coûts de communications. Dans chaque ensemble g_u, tous les processeurs calculent localement une somme partielle de la plaque colonne $(i_0, t+\Delta t)$ pour chaque $i_0 \in DG_u$. Dans l'étape finale de la mise à jour de N, ces sommes partielles sont sommées à l'intérieur du groupe g_u puis stockées. Pour chaque i_0 de DG_u, un seul processeur de g_u est choisi pour enregistrer la plaque colonne $(i_0, t+\Delta t)$. Ce processeur est désigné par la fonction `mapi(`i_0`)` qui sera explicitée plus tard. De manière similaire, un processeur de chaque groupe h_e est désigné comme propriétaire de la plaque ligne (c_0, t) avec $c_0 \in DH_e$; l'indice de ce processeur est connu grâce à une fonction `mapc(`c_0`)`. L'algorithme de mise à jour de N est réécrit dans la Figure 31 en utilisant les fonctions `mapc` et `mapi`. Dans cet algorithme, des plaques lignes $N_*(c, *, t)$ sont utilisées au niveau du noyau de calcul, et des plaques colonnes $N_*(*, i_0, t+\Delta t)$ sont générées. Une étape de transposition est donc réalisée entre deux mises à jour successives. Il s'agit là de la tâche de communication $comm_3$ qui apparaît dans l'algorithme. Le coût de communication de cette réorganisation des données est égal à la taille de la donnée $N_*(*, *, t + \Delta t)$ qui est $\frac{KS^2}{2}$. La technique adoptée pour réaliser cette tâche a consisté à faire appel à la routine `MPI_AllToAll` [51, 52] à l'intérieur du groupe g_u. On choisira `mapi` et `mapc` de façon à réduire les coûts de communication et à équilibrer la charge.

6.3.1.3 Coûts des communications

Les coûts de communication sont évalués en considérant une à une les trois tâches de communications de l'algorithme parallèle de la Figure 31.

En ce qui concerne la tâche de communication $comm_1$, dans chaque groupe de processeurs h_e, une diffusion au sein du groupe est réalisée à chaque itération $c \in DH_e$. Le processeur d'indice `mapc(c)` envoie sa plaque ligne (c, t) comptant $K(c+1)$ réels à tous les processeurs

```
Procédure Mise_à_jour_2D_de_N(S) { (* sur le processeur p_{u,e} *)
    Pour tous les c ∈ DH_e faire{
    .   (* tâche de communication comm_1 *)
    .   Si mapc(c) = my_id alors
    .       broadcast dans mon groupe h_e de
    .       la plaque ligne N_*(c,*,t);
    .   Sinon
    .       reçoit N_*(c,*,t);
    .   Pour tous les i ∈ DG_u faire{
    .   .   Si (i ≤ c) alors {
    .   .   .   (* tâche M(i,c) *)
    .   .   .   initialisation des matrices A_1,A_2,A_3;
    .   .   .   Aux:=noyau_de_calcul(i,c,N_*(c,*,t));
    .   .   .   (* sommation partielle *)
    .   .   .   N_*(*,i,t+Δt)=
    .   .   .       addition_de_matrices(N_*(*,i,t+Δt)+Aux);
    .   .   }
    .   }
    }

    (* tâche de communication comm_2 *)
    Pour tous les i ∈ DG_u faire{
    .   Addition des sommes partielles N_*(*,i,t+Δt) de
    .   tous les processeurs de g_u sur le processeur mapi(i);
    }

    (* tâche de communication comm_3 *)
    Transposition des plaques colonnes (i,t+Δt) placées
    sur les processeurs mapi(i), vers des plaques lignes (c,t+Δt)
    placées sur les processeurs mapc(c).
}
```

Figure 31. Algorithme parallèle 2D de mise à jour de N

du groupe h_e. Le nombre de réels dans la donnée N est de l'ordre de $K\,S^2/2$. Dans chaque ensemble de processeurs h_e, une fraction $1/|H|$ de cette quantité mémoire est transmise à chaque processeur durant une mise à jour de N. A la fin d'une mise à jour complète, chaque processeur dispose donc approximativement de $K\,S^2/(2\,|H|)$ réels parmi les plaques lignes reçues. Comme le processeur est responsable en moyenne d'une fraction $1/|G|$ des plaques lignes au sein de son propre groupe h_e, il diffuse à tous les processeurs du groupe une fraction $1/|G|$ de $K\,S^2/(2\,|H|)$, et reçoit de ceux-ci $(1-1/|G|)$ de $K\,S^2/(2\,|H|)$. En multipliant le coût des communications reçues par un processeur, par le nombre de processeurs $P = |H|\,|G|$, on déduit le coût de communication global sur la machine parallèle qui est de l'ordre de $(|G| - 1)\,K\,S^2/2$. Le coût en calcul de l'algorithme parallèle est lui égal à la complexité séquentielle $55\,S^4/24$ plus le surcoût lié au parallélisme. Quel que soit ce surcoût, les communications de la tâche $comm_1$ sont en $5\,(|G|-1)\,S^2$ et asymptotiquement moindres que la complexité séquentielle pour $G \ll S$. En pratique, la tâche de communication $comm_1$ est pipelinée et met en œuvre des communications non-bloquantes. Cela signifie que pendant que les calculs sont effectués, des plaques lignes sont envoyées par avance aux autres processeurs. De cette façon, on a la possibilité de recouvrir les communications par les calculs. De plus, comme les coûts de communications sont faibles devant ceux des calculs ($P \ll S$), on observe durant les mesures de performances un recouvrement quasi-total des communications.

Considérons maintenant la tâche de communication $comm_2$. Dans chaque ensemble de processeurs g_u, des additions de sommes partielles sont réalisées pour obtenir $N_*(*, i_0, t)$ avec $i_0 \in DG_u$. Pour une itération i_0 de la tâche de communication (et pour une plaque colonne (i_0, t)), tous les processeurs de l'ensemble, sauf le propriétaire de cette plaque colonne `mapi(`i_0`)`, envoient leurs contributions à `mapi(`i_0`)`. La taille des données $N_*(*, *, t + \Delta t)$ est $K\,S^2/2$; dans chaque groupe g_u une fraction de $1/|G|$ de cette quantité est manipulée durant la tâche de communication $comm_2$, soit $K\,S^2/(2\,|G|)$. Il y a $|H|$ processeurs dans l'ensemble g_u, donc un processeur est réellement responsable de $K\,S^2/(2\,|G|)$ flottants divisé par H en moyenne. Durant la réduction somme, un processeur reçoit de tous les processeurs de g_u les contributions pour les plaques colonnes dont il est responsable, c'est-à-dire $|H|-1$ processeurs. Le coût des communications reçues par processeur est donc $(|H|-1)\,K\,S^2/(2\,|G|\,|H|)$. On en déduit que le coût global sur la machine parallèle pour la tâche de communication $comm_2$ est $(|H|-1)\,K\,S^2/2$ en multipliant par le nombre de processeurs. On note que cette valeur est symétrique par rapport au coût de communication pour la tâche $comm_1$, en substituant $|G|$ par $|H|$. Donc lorsque $|G|$ augmente, le coût de la tâche de communication $comm_1$ croît et celui de la tâche de communication $comm_2$ diminue (et vice-versa pour H). Là encore, le coût de communication est asymptotiquement négligeable devant la complexité séquentielle. Cette tâche de communication n'est pas recouverte par le calcul comme la précédente. Néanmoins l'utilisation de la routine `MPI_REDUCE` au sein d'un de chaque groupe g_u peut permettre de réaliser des communications relativement rapides selon les caractéristiques du réseau.

La tâche de communication $comm_3$ correspond à la transposition de la donnée $N_*(*, *, t + \Delta t)$. Là encore, le vecteur de K éléments $N_*(c_0, i_0, t + \Delta t)$ qui appartient au processeur `mapi(i`$_0$`)` doit simplement être transféré au processeur `mapc(c`$_0$`)`. Le volume des communications est égal au nombre de flottants dans la donnée $N_*(*, *, t)$ *i.e.* $\frac{K\,S^2}{2}$. L'implémentation utilise la routine MPI [51] `MPI_AllToAll`.

L'organisation des communications qui ont été décrites et l'utilisation de routines MPI adaptées conduisent à une implémentation comportant un faible surcoût lié à la communication. On le constatera effectivement lorsque les performances seront présentées en p. 95. On peut noter que pour le schéma 1D ($|H| = 1, \text{et} |G| = P$), il n'y a pas de coût de communication pour la tâche $comm_2$, mais le coût de la tâche $comm_1$ est plus important : $(|G| - 1)\,K\,S^2/2$. Les tâches de communication $comm_1$ et $comm_2$ dans le schéma 1D coûtent $(P-1)\,K\,S^2/2$ alors que dans le schéma 2D on les évalue à $(|G|+|H|-2)\,K\,S^2/2$. Ce schéma 2D réduit donc le coût des communications et favorise les communications locales à l'intérieur d'un groupe de processeurs. On verra que l'on peut tirer parti de cette propriété en allouant les groupes de processeurs dans un nœud à mémoire partagée sur une architecture constituée d'un réseau de nœuds SMP. Ainsi on pourra réaliser des communications intra-nœud très rapides.

Après cette étude sur les communications, la distribution des calculs va être évaluée. Les schémas de communications que nous avons choisis laisse la possibilité de choisir le placement des indices c et i dans DH_e et DG_u, à la seule condition que les données soient réparties équitablement entre les processeurs. Il reste à déterminer ce placement en minimisant le seul surcoût parallèle potentiel restant, celui issu du déséquilibre des charges.

6.3.2 Coûts, efficacité, scalabilité

La complexité en nombre d'additions et de multiplications du noyau de calcul $M(c,i)$ est donné pour un pas de temps t fixé (avec $p_{sup}(t) = S$ et $K = 10$) par :

$$CM(c,i) = \frac{1}{2}(c-i+1)\,[\,c-37\,i+36\,S+92\,] + 9\,(S+1-i). \tag{28}$$

Dans le domaine $i \in [0,S], c \in [i,S]$, la fonction CM est positive, décroissante par rapport à la variable i et croissante en c. De plus, cette fonction est convexe en i pour une valeur de c fixée, et convexe en c lorsque i est fixée :

$$\frac{\partial^2 CM}{\partial^2 c}(c,i_0) = 1 > 0$$

$$\frac{\partial^2 CM}{\partial^2 i}(c_0,i) = 37 > 0 \ .$$

On a donc ce que l'on appellera par la suite la Propriété 1 :

Propriété 1 *Pour $0 \leq i \leq c \leq S$, la fonction $CM(S-c,i)$ est décroissante convexe pour les variables i et c.*

6.3.2.1 Propriétés des distributions

De façon analogue à la solution 1D, on réalise une distribution en serpentin pour les indices i et c. On considère le placement des indices c sur les groupes de processeurs h_e, et celui des indices i sur les groupes g_u. Une distribution optimale des tâches $M(*,*)$ peut être recherchée pour minimiser le déséquilibre de charges. Néanmoins, afin d'étudier analytiquement certaines propriétés de l'algorithme parallèle, une distribution régulière a été choisie. Une analyse théorique des qualités de la distribution en serpentin va être présentée ci-après.

indices e du groupe h_e	**0**	**1**	**2**
indices locaux	17	16	15
	12	13	14
	11	10	9
	6	7	8
	5	4	3
	0	1	2

Figure 32. Distribution en serpentin des indices c pour $|H| = 3$, $S = 17$

indices u du groupe g_u	**0**	**1**	**2**
indices locaux	0	1	2
	5	4	3
	6	7	8
	11	10	9
	12	13	14
	17	16	15

Figure 33. Distribution en serpentin des indices i pour $|G| = 3$, $S = 17$

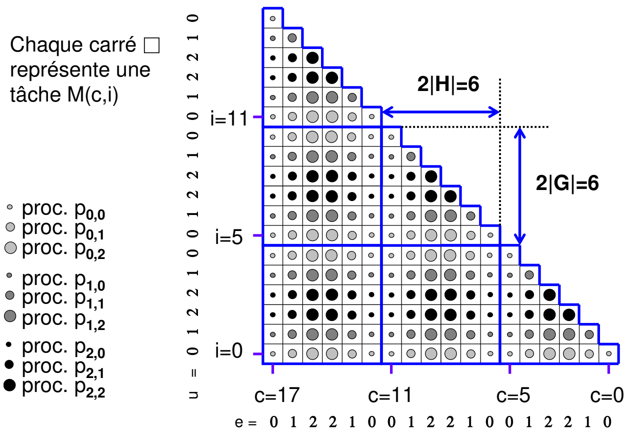


Figure 34. Les tâches $M(S-c, i)$ affectées aux $p_{e,u}$ pour $|H| = |G| = 3$ et $S = 17$

La complexité d'une tâche $M(c, i)$ est une fonction décroissante de la variable c. On commence la distribution des indices c en commençant par le plus grand c, soit $c = S$. On affecte aux processeurs la plus coûteuse des tâches en premier puis les tâches moins coûteuses. De cette manière, l'affectation des dernières tâches de calcul peu coûteuses ne risque pas de changer notoirement l'équilibrage des charges. Comme la complexité d'une tâche $M(c, i)$ est une fonction croissante de la variable i, on prend $i = 0$ pour initier la distribution des indices i. Supposons que $|G| = 3$ et $|H| = 3$ pour $S = 21$, on obtient les distributions des figures 32 et 33. Les fonctions `mapi` et `mapc` sont directement déduites à partir des distributions décrites.

On voit sur la Figure 34 quelle est l'allure d'une telle distribution en serpentin 2D au niveau de l'ensemble des tâches $M(c, i)$ (une organisation très régulière est notée lorsque $S+1$ divise $2\,|H| = 2\,|G|$, comme dans cet exemple). On remarque immédiatement que le schéma comporte des blocs similaires de côté $2|H| = 6$ et $2|G| = 6$. On observe que l'affectation des processeurs sur les blocs diagonaux est toujours identique, ainsi que celle des blocs extra-diagonaux (correspondant à $2\,|G| \times 2\,|H| = 4\,P$ tâches). Dans les études suivantes, on utilisera la structure répétitive des blocs extra-diagonaux contenant

$4\,P$ tâches $M(c,i)$ qui reste valable même sans l'hypothèse $S+1$ divise $2\,|H|=2\,|G|$.

Les performances de ce placement des calculs vont maintenant être évaluées.

6.3.2.2 Coût des calculs par processeur

La distribution des tâches $M(c,i)$ étant connue ainsi que leur complexité, on peut évaluer le coût des calculs pour chaque processeur. La distribution est statique donc une étude sur l'efficacité fonction des variables S, $|H|$ et $|G|$ est possible. Au préalable, il est nécessaire d'identifier le processeur qui a le plus d'opérations à réaliser de manière à déduire le coût parallèle. On suppose pour l'instant que la grille de processeurs est un carré de côté $Y=|H|=|G|$ (nous reviendrons sur cette hypothèse plus tard).

Proposition 2. *Supposons que $2\,Y$ divise $S+1$. On distribue les tâches avec une distribution en serpentin bidimensionnelle sur la grille de processeurs carrée de $P=Y^2$ processeurs. On utilise les règles suivantes pour déterminer le processeur $p_{e,u}$ qui calculera la tâche $M(S-c,i)$ (voir Figure 35):*

- *soit* $r_i = i$ *modulo* $2Y$ *:*
  si $0 \leq r_i \leq Y-1$ *alors* $e = r_i$,
  si $Y \leq r_i \leq 2Y-1$ *alors* $e = 2Y-1-r_i$;
- *soit* $r_c = (S-c)$ *modulo* $2Y$ *:*
  si $0 \leq r_c \leq Y-1$ *alors* $u = r_c$,
  si $Y \leq r_c \leq 2Y-1$ *alors* $u = 2Y-1-r_c$.

Soit $charge(e,u)$ la fonction donnant la complexité sur le processeur $p_{e,u}$. On a la relation

$$\forall (e,u) \in [0,Y-1]^2 \qquad charge(e,u) \leq charge(0,0) \quad . \tag{29}$$

Le processeur $p_{0,0}$ réalise donc plus de calculs que les autres.

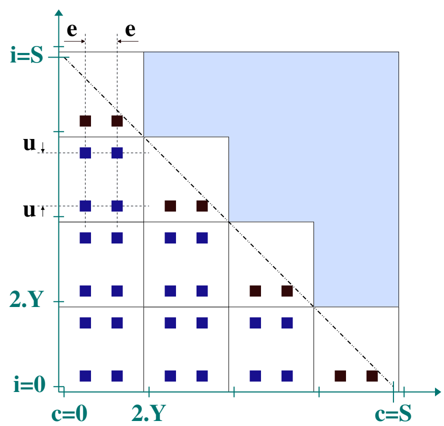


Figure 35. Les tâches $M(S-c,i)$ distribuées sur le processeur $p_{e,u}$ apparaissent comme des carrés noirs

Cette Proposition est prouvée en annexe A p. 169. Le processeur qui a la charge la plus importante est donc $p_{0,0}$ (si la grille de processeur est carrée de côté Y, et si $2Y$ divise $S+1$). Un exemple des charges supportées par les processeurs durant la mise à jour de N est donné sur la figure 36 dans la Section suivante.

6.3.2.3 Surcoût du schéma 2D

Le coût parallèle de l'algorithme de mise à jour de N se déduit directement de la charge de $p_{0,0}$ et il vaut $P\,charge(0,0)$. On peut déterminer une formule pour le surcoût théorique dû au déséquilibre des charges. On doit néanmoins considérer que la complexité en nombre d'opérations est différente de celle en temps. En effet, les temps d'initialisation des matrices intermédiaires ne sont pas négligeables, et des améliorations des performances dues à la réutilisation des caches doivent être considérées. L'utilisation des BLAS introduit cependant une relation non triviale entre les dimensions des matrices et les coûts en temps. La complexité en temps diffère donc du nombre d'opérations $W(S,K)$ (*cf* p. 69), bien qu'asymptotiquement ces deux grandeurs soient identiques $T_{seq}(S) = \Theta(S^4)$.

Dans cette partie, on évalue le surcoût théorique toujours avec $2Y$ qui divise $S+1$. On pose $Z = S+1$, le coût de la tâche $M(c,i)$ en fonction de Z devient :

$$CM(c,i) = \frac{1}{2}(c-i+1)\,[\,c-37\,i+36\,Z+56\,] + 9\,(Z-i)\ .$$

On définit $W_{//}(Z-1,Y)$ le coût parallèle sur $Y^2 = P$ processeurs :

$$W_{//}(Z-1,Y) = Y^2\,charge(0,0)\ . \tag{30}$$

L'expression de $charge(0,0)$ dépend de $CM(c,i)$ et de la distribution en serpentin :

$$\begin{aligned}
charge(0,0) = & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (31)\\
& \sum_{j=1}^{q}\sum_{g=1}^{q-j+1}[CM(Z-1-(2Y(g-1)),2Y(j-1)) + CM(Z-1-(2Yg-1),2Y(j-1))+\\
& \qquad CM(Z-1-(2Y(g-1)),2Yj-1) + CM(Z-1-(2Yg-1),2Yj-1)]\\
& +\sum_{j=1}^{q+1}[CM(Z-1-(2Y(q-j+1)),2Y(j-1)) + CM(Z-1-(2Y(q-j+2)-1),2Y(j-1))+\\
& \qquad CM(Z-1-(2Y(q-j+1)),2Yj-1)]\ .
\end{aligned}$$

Le surcoût de l'algorithme $W_{ov}(Z-1,Y)$ est donné par la formule classique :

$$\begin{aligned}
W_{ov}(Z-1,Y) &= W_{//}(Z-1,Y) - W(Z-1)\\
&= Y^2\,charge(0,0) - W(Z-1)\ .
\end{aligned}$$

On définit la variable $X = Y - 1$ pour simplifier les expressions qui suivent. Des calculs formels[24] conduisent au résultat suivant :

$$W_{ov}(Z-1,X+1) = \left[\frac{28\,X^2}{3}+\frac{137\,X}{12}\right] Z^2 + \left[17\,X^2+\frac{117\,X}{14}\right] Z \ .$$

Pour effectuer des simulations coûteuses, le simulateur parallèle est employé avec des variables X et Z dans les domaines suivants : $Z \in [240, 2000]$, et $Y \in [4, 20]$. Pour ce type d'utilisation, le surcoût est approximativement (avec une erreur de $\pm 1\,\%$) : $W_{ov}(Z-1,X+1) \approx (\frac{28}{3}\,X^2 + \frac{137}{12}\,X)\,Z^2$, ce qui conduit à la relation suivante (asymptotiquement en S) :

$$W_{ov}(S,\sqrt{P}) \underset{S\to\infty}{=} \frac{1}{12}\,(112\,P - 87\,\sqrt{P} - 25)\,S^2 \ . \tag{32}$$

Pour le schéma 1D, le surcoût était à égal $\Theta(P^2\,S^2)$; le bénéfice du schéma 2D est donc flagrant car le surcoût n'est plus qu'en $\Theta(P\,S^2)$. Lorsque l'on supprime l'hypothèse $2Y$ divise $S+1$, les résultats restent valables[25] lorsque S est grand devant Y.

6.3.2.4 Efficacité et extensibilité

Si $E(S,P)$ désigne l'efficacité théorique du schéma 2D, on peut l'écrire comme une fonction du surcoût de la manière suivante [24] :

$$E(S,P) = \frac{1}{1+\frac{W_{ov}(S,\sqrt{P})}{W_{seq}(S)}} \ .$$

On obtient le résultat suivant asymptotiquement en S :

$$E(S,P) = \frac{1}{1+\frac{224\,P-174\sqrt{P}-50}{55\,S^2}+o(\frac{1}{S^3})} \ . \tag{33}$$

Si l'on fixe l'efficacité, on peut déduire une formule d'iso-efficacité théorique :

$$224\,P - 174\sqrt{P} - 50 = 55\,S^2 + o(\frac{1}{S}) \ , \tag{34}$$

$$S \underset{P\to\infty}{=} \Theta(\sqrt{112\,P - 87\sqrt{P}}) \ . \tag{35}$$

L'extensibilité de l'algorithme est donc de très bonne qualité.

24. Ces résultats de calculs symboliques peuvent être obtenus avec un logiciel tel que *Maple* par exemple.
25. Voir annexe p. 179.

6.3.2.5 Expérimentation numérique

Un exemple de temps constatés en pratique pour une mise à jour de $N_*(*,*,t)$ est présenté sur la Figure 36 sur un IBM SP2 (nœuds fins 120 Mhz *cf* p. 80). Comme l'illustre cette Figure, on constate bien que le processeur 0 est celui qui a le plus long temps de calcul.

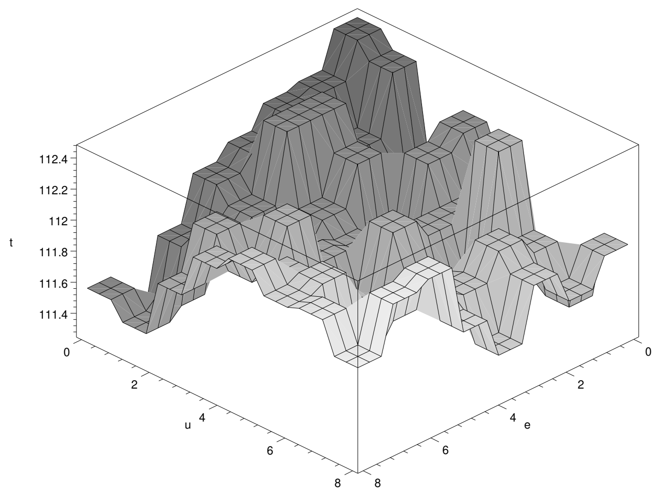


Figure 36. Distribution effective des temps de calcul sur les processeurs pour une mise à jour de $N_*(*,*,t)$ (y compris les communications) pour $P = Y^2 = 64$ processeurs

6.3.2.6 Grille rectangulaire

Supposons maintenant que la grille de processeurs ne soit plus carrée mais rectangulaire. Les résultats théoriques s'appuyant sur la Proposition 2 ne sont alors plus valables. Il s'agit de réaliser une étude simplifiée sous cette nouvelle hypothèse. On distribue les indices c sur les $|G|$ groupes de processeurs et les indices i sur les $|H|$ groupes de processeurs avec la méthode du serpentin. On forme les blocs extra-diagonaux de $4\,P$ tâches à réaliser $M(c,i)$ que l'on a introduit précédemment p. 87. Dans l'étude de cette section, nous ne tiendront pas compte des blocs diagonaux qui contiennent moins de $4\,P$ tâches (lorsque $c \leq i$ pour certain des couples (c,i) du bloc considéré). Sur la Figure 37, cela correspond aux blocs qui ne sont pas grisés, *i.e.* ceux qui sont au-dessous et qui ne croisent pas la diagonale $c = i$. Sur l'ensemble des $4\,P$ noyaux de calculs d'un bloc extra-diagonal chaque processeur de la grille de processeurs doit réaliser exactement 4 tâches. Chacune des tâches du processeur $p_{e,u}$ est symbolisée par un petit disque sur la Figure 37.

On considère des blocs indicés par (g,j) contenant les tâches $(c,i) \in [S - 2\,G\,g + 1, S - 2\,G\,(g-1)] \times [2\,H\,(j-1), 2\,H\,j-1]$. La contrainte $c \geq i$ implique que le minimum des indices c du bloc soit supérieur ou égal au maximum des indices i dans le bloc :

$$S - 2\,G\,g+1 \geq 2\,H\,j-1 \qquad \text{donc} \qquad \frac{S+2-2\,H\,j}{2\,G} \geq g \quad .$$

D'autre part, les indices i sont majorés par S, d'où :

$$2\,H\,j-1 \leq S \qquad \text{donc} \qquad \frac{S+1}{2\,H} \geq j \ .$$

On en déduit que les blocs extra-diagonaux considérés $ED(g,j)$ (blocs non grisés dans la Figure 37) sont tels que pour les variables j et g, on ait $1 \leq j \leq \lfloor \frac{S+1}{2\,H} \rfloor$, $1 \leq g \leq \lfloor \frac{S+2-2\,H\,j}{2\,G} \rfloor$. Pour un bloc extra-diagonal indicé par (g,j), le coût en calcul du processeur $p_{e,u}$ est :

$$\begin{aligned} ED_{e,u}(g,j) = \quad & CM(S-(2\,G\,(g-1)+e), 2\,H\,(j-1)+u) \\ & + CM(S-(2\,G\,g-e-1), 2\,H\,(j-1)+u) \\ & + CM(S-(2\,G\,(g-1)+e), 2\,H\,j-u-1) \\ & + CM(S-(2\,G\,g-e-1), 2\,H\,j-u-1) \ . \end{aligned}$$

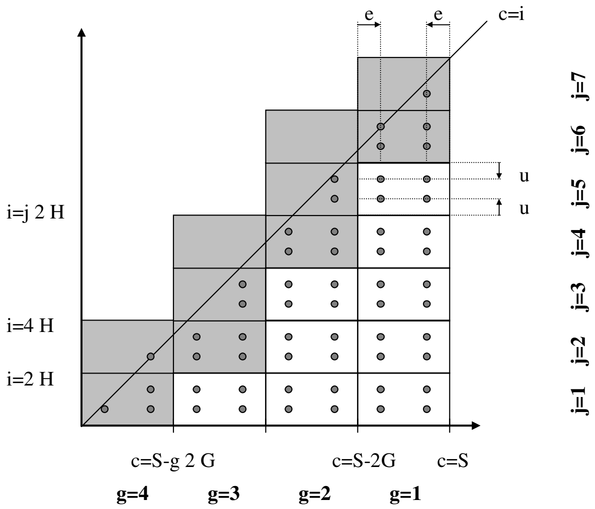


Figure 37. Schéma de l'ensemble des blocs contenant $4\,P$ tâches $M(c,i)$ (blocs blancs) ; les blocs grisés correspondent aux blocs qui coupe la droite $c = i$ que l'on nomme blocs diagonaux ; les petits disques correspondent à l'ensemble des tâches $M(c,i)$ affectées par exemple au processeur $p_{e,u}$

Le coût parallèle en calcul pour ce bloc de tâches est $c_{//}(g,j) = P\,ED_{0,0}(g,j)$, car $p_{0,0}$ est le processeur le plus chargé (même preuve que pour un bloc carré). D'un autre côté, le coût séquentiel de ce bloc est la somme du coût de chaque tâche *i.e.* $c_{seq}(g,j) = \sum_{e=0}^{G-1} \sum_{u=0}^{H-1} ED_{e,u}(g,j)$. Par un calcul direct nous allons maintenant évaluer le surcoût parallèle généré par un seul bloc extra-diagonal. On peut exprimer $ED_{e,u}(g,j)$ en fonction de la valeur du polynôme CM au point (c_0, i_0) placé au centre du bloc (g,j) (en posant $c_0 = S - (2\,G\,(g-1) + G - \frac{1}{2})$, $i_0 = 2\,H\,(j-1) + H - \frac{1}{2})$. Pour cela, on va réaliser un développement limité. Comme la fonction de coût CM est un polynôme homogène du second degré le développement à l'ordre 2 est exact. On a donc :

$$\begin{aligned} CM(c_0+\epsilon_c, i_0+\epsilon_i) = \; & CM(c_0,i_0) + \epsilon_c \frac{\partial CM}{\partial c}(c_0,i_0) + \epsilon_i \frac{\partial CM}{\partial i}(c_0,i_0) + \\ & \frac{1}{2}\epsilon_c^2 \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0) + \frac{1}{2}\epsilon_i^2 \frac{\partial^2 CM}{\partial^2 i}(c_0,i_0) + \epsilon_i\,\epsilon_c \frac{\partial^2 CM}{\partial c\,\partial i}(c_0,i_0) \end{aligned} \tag{36}$$

Pour u, e, G, H donnés, on pose $\epsilon_c = G-e-\frac{1}{2}$ et $\epsilon_i = H-u-\frac{1}{2}$. On a alors le coût en calcul sur le processeur $p_{e,u}$ qui vaut :

$$\begin{aligned} ED_{e,u}(g,j) = \; & CM(c_0-\epsilon_c, i_0-\epsilon_i) + \; CM(c_0+\epsilon_c, i_0-\epsilon_i) \\ & + CM(c_0-\epsilon_c, i_0+\epsilon_i) + \; CM(c_0+\epsilon_c, i_0+\epsilon_i)\ . \end{aligned}$$

En utilisant l'équation 36, on peut réécrire la formule précédente. On a alors

$$ED_{e,u}(g,j) = \; 4\,CM(c_0,i_0) + 2\,(G-e-\frac{1}{2})^2 \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0) + 2\,(H-u-\frac{1}{2})^2 \frac{\partial^2 CM}{\partial^2 i}(c_0,i_0)\ .$$

Evaluons maintenant le surcoût parallèle en calcul $c_{over}(g,j)$ sur ce bloc de tâches :

$$\begin{aligned} c_{over}(g,j) = \; & c_{//}(g,j) - c_{seq}(g,j) = P\,ED_{0,0}(g,j) - \sum_{e=0}^{G-1}\sum_{u=0}^{H-1} ED_{e,u}(g,j)\ , \\ c_{over}(g,j) = \; & P\,(4\,CM(c_0,i_0) + \; 2\,(G-\frac{1}{2})^2 \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0) + 2\,(H-\frac{1}{2})^2 \frac{\partial^2 CM}{\partial^2 i}(c_0,i_0)) - \\ & \sum_{e=0}^{G-1}\sum_{u=0}^{H-1} (4\,CM(c_0,i_0) + \; 2\,(G-e-\frac{1}{2})^2 \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0) + 2\,(H-u-\frac{1}{2})^2 \frac{\partial^2 CM}{\partial^2 i}(c_0,i_0))\ . \end{aligned}$$

Or comme on a $P = H\,G$, avec H et G les dimensions de la grille de processeurs, on en déduit donc :

$$\begin{aligned} c_{over}(g,j) = \; & \sum_{e=0}^{G-1}\sum_{u=0}^{H-1} \Big[2\,(G-\frac{1}{2})^2 \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0) - 2\,(G-e-\frac{1}{2})^2 \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0) \\ & \quad + 2\,(H-\frac{1}{2})^2 \frac{\partial^2 CM}{\partial^2 i_0}) - 2\,(H-u-\frac{1}{2})^2 \frac{\partial^2 CM}{\partial^2 i}(c_0,i_0) \Big]\ , \\ = \; & 2 \sum_{e=0}^{G-1}\sum_{u=0}^{H-1} \Big[e\,(2\,G-e-1) \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0) + u\,(2\,H-u-1) \frac{\partial^2 CM}{\partial^2 i}(c_0,i_0) \Big]\ , \\ = \; & 2 \sum_{e=0}^{G-1}\sum_{u=0}^{H-1} \Big[((2\,G-1)\,e-e^2) \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0) + ((2\,H-1)\,u-u^2) \frac{\partial^2 CM}{\partial^2 i}(c_0,i_0) \Big]\ , \\ = \; & 2 \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0) \Big[(2\,G-1)\,H\,\frac{G\,(G-1)}{2} - H\,\frac{G\,(2G-1)\,(G-1)}{6} \Big] \\ & \quad + 2 \frac{\partial^2 CM}{\partial^2 i}(c_0,i_0) \Big[(2\,H-1)\,G\,\frac{H\,(H-1)}{2} - G\,\frac{H\,(2H-1)\,(H-1)}{6} \Big]\ , \\ = \; & \frac{2}{3}\,P \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0)\,(2\,G-1)\,(G-1) + \frac{2}{3}\,P \frac{\partial^2 CM}{\partial^2 i}(c_0,i_0)\,(2\,H-1)\,(H-1)\ , \\ = \; & \frac{4}{3}\,P \Big[(G-\frac{1}{2})\,(G-1) \frac{\partial^2 CM}{\partial^2 c}(c_0,i_0) + (H-\frac{1}{2})\,(H-1) \frac{\partial^2 CM}{\partial^2 i}(c_0,i_0) \Big]\ . \end{aligned}$$

En explicitant (28), on a

$$CM(c,i) = \frac{1}{2}\,c^2 + \frac{37}{2}\,i^2 - 19\,c\,i + 18\,c\,S - 18\,i\,S + \frac{93}{2}\,c - \frac{147}{2}\,i + 27\,S + 55\ , \tag{37}$$

et on obtient finalement

$$c_{over}(g,j) = \frac{2}{3}\,P\left[(G-\frac{1}{2})\,(G-1) + 37\,(H-\frac{1}{2})\,(H-1)\right]\ .$$

On remarque que le surcoût $c_{over}(g,j)$ est identique quelles que soient les coordonnées du bloc (g,j) ! Il va donc être possible trouver une taille (H,G) pour la grille de processeurs qui optimise les surcoûts de tous les blocs extra-diagonaux en même temps.

6.3.3 Minimisation du surcoût

La modélisation du surcoût engendré par les blocs diagonaux est plus difficile, et l'on trouve des contres-exemples pour lesquels le coût du processeur $p_{0,0}$ ne dépasse pas le coût des autres processeurs sur les blocs diagonaux. Par contre le coût d'un bloc diagonal est généralement faible par rapport au coût d'un bloc extra-diagonal (la fonction de coût CM décroît en i et en $S-c$ et admet son minimum au niveau de la diagonale, *cf* p. 179). D'autre part, le nombre de bloc diagonaux est asymptotiquement plus faible que le nombre de blocs extra-diagonaux. On considère de ce fait l'hypothèse suivante : le surcoût moyen d'un bloc quelconque est approximée par le surcoût d'un bloc extra-diagonal. Cherchons donc à minimiser le surcoût approximé $c_{approx}(H)$ (pour un nombre de processeurs P fixé) que l'on pose égal à c_{over} :

$$c_{approx}(H) = \frac{2}{3}\,P\left[37\,H^2 - \frac{111\,H}{2} + 19 + \frac{P^2}{H^2} - \frac{3\,P}{2\,H}\right]$$
$$c'_{approx}(H) = \frac{2}{3}\,P\left[\frac{-2\,P^2}{H^3} + \frac{3\,P}{2\,H^2} + 74\,H - \frac{111}{2}\right]$$

La fonction $c'_{approx}(H)$ s'annule au minimum de la fonction $c_{approx}(H)$. En cherchant la valeur de H_{min} qui annule la dérivée, une résolution numérique donne pour $P = 120$ la valeur $H_{min} = 4.61$. La courbe théorique du surcoût pour $P = 120$ est présentée à la Figure 38. La courbe expérimentale des temps de calculs d'une mise à jour ($S = 800$) pour différents H sur l'IBM SP2 du CINES se trouve en Figure 39. On constate que ces courbes ont une allure similaire, justifiant ainsi les calculs théoriques. D'autre part, le calcul de H_{min} est conforté par l'expérimentation qui donne $H_{min} = 5$.

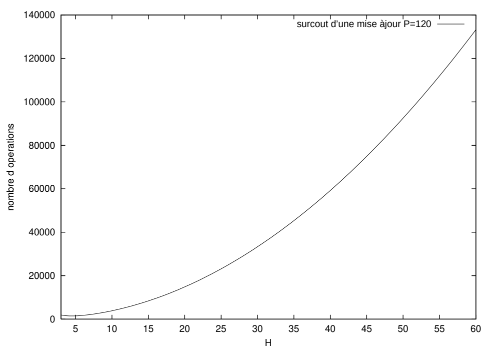


Figure 38. Courbe théorique du nombre d'opérations d'une mise à jour de N fonction de H pour $P = 120$

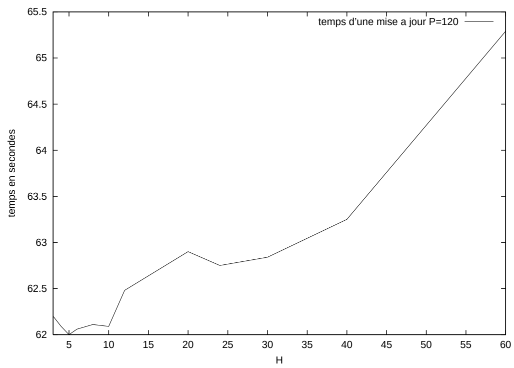


Figure 39. Temps observé d'une mise à jour de N fonction de H sur IBM SP2 pour 120 processeurs

Sur une architecture de machine telle que l'IBM SP3, une donnée supplémentaire doit être prise en compte. En effet, lors d'une communication entre deux processeurs de la machine, le fait d'être sur le même nœud SMP ou non change les coûts en communication. Avec l'IBM SP2 les coûts de communication point à point sont globalement similaires quel que soit le couple de processeurs considéré. Lorsque les groupes de processeurs DG_u et DH_e sont choisis sur un IBM SP3, cela introduit un coût en communication non uniforme entre les processeurs. On en déduit qu'en choisissant des groupes DG_u et DH_e adéquats, on peut diminuer les coûts de communications en favorisant les communications intra-nœuds. En effet, les communications collectives concernant plusieurs nœuds simultanément peuvent infirmer l'hypothèse initiale concernant des coûts en communication négligeable. Le surcoût dû au parallélisme est donc modifié et pour l'IBM SP3 le placement et la taille de la grille de processeurs doivent tenir compte de la topologie de la machine en termes de communications intra et inter-nœuds. La modélisation du surcoût que l'on a développé doit donc être adapté sur l'IBM SP3 et favoriser les communications intra-nœuds.

6.3.4 Performances sur IBM SP2 et SP3

Les machines employées pour ces prises de performances sont les IBM SP2 et IBM SP3 du CINES (*cf* p. 80). Sur l'IBM SP2 et pour $S = 800$, le temps passé pour la réduction (étape de communication) prend moins de 1 % du temps de mise à jour de N (voir Figure 40).

Sur l'IBM SP3, ce pourcentage monte à 6 % pour 169 processeurs. Cette hausse s'explique parce qu'on a pris des groupes DG_u contenant des processeurs à cheval sur plusieurs nœuds SMP pour 100 et 169 processeurs[26], alors que pour les autres nombres de processeurs chaque groupe DG_u est contenu entièrement dans un nœud SMP (qui contient 16 processeurs). Lorsque la tâche de communication 2 s'effectue au sein d'un nœud, la

26. 10 groupes de 10 processeurs pour $P = 100$, et 13 groupes de 13 processeurs pour $P = 169$.

réduction somme dans DG_u se fait de manière plus performante en mémoire partagée. Ce fait explique la hausse constatée pour 100 et 169 processeurs au niveau de cette réduction somme. L'autre tâche de communication MPI_AllToAll prend moins de 5 % du temps de la mise à jour de N sur les IBM SP2 et SP3. La communication représente donc un faible surcoût pour l'application.

La Figure 41 donne l'efficacité relative pour les deux machines. Sur l'IBM SP2, l'efficacité ne baisse quasiment pas jusque 169 processeurs si l'on tient compte de la variation de performances entre processeurs qui est non négligeable. Pour l'IBM SP3, l'analyse est différente car l'efficacité relative baisse entre 4 et 16 processeurs, puis entre 16 et 64 processeurs. Premièrement le fait d'employer 4 puis 16 processeurs au sein d'un nœud implique que chaque processeur voit sa bande passante mémoire diminuée ; comme la mémoire est partagée, les calculs saturant la bande passante sont ralentis et les communications utilisant la mémoire partagée le sont aussi. Deuxièmement, entre 16 et 64 processeurs on passe d'une situation pour laquelle toutes les communications se font en mémoire partagée (à l'intérieur du nœud) à des communications inter-nœuds, ce qui explique la baisse constatée. Entre 64 et 448 processeurs l'efficacité relative est stable et supérieure à 92 %. Dans la Figure 42, on constate que les calculs autres que la mise à jour de N représentent un faible pourcentage du temps d'exécution. Ceci est dû au fait que ces parties ont été parallélisées pour la plupart. Dans plusieurs routines, les calculs sont distribués au sein des groupes de processeurs DG_u, puis une étape de communication collective au sein du groupe permet aux processeurs de récupérer l'ensemble des résultats. Comme chaque groupe est habituellement affecté à un nœud SMP, les communications collectives intra-nœud ont un faible coût. La loi d'Amdhal limite peu les performances, la partie strictement séquentielle de l'application a été réduite au minimum. On constate ainsi sur la Figure 42 que les calculs annexes se limitent à 15 % du temps avec 448 processeurs. La Figure 43 montre que l'efficacité relative pour une simulation entière et coûteuse de plus de 105 TFLOP décroît peu sur IBM SP2 avec près de 90 % sur 169 processeurs. Sur IBM SP3, l'efficacité relative baisse davantage à deux reprises : entre 16 et 64 processeurs, cela est lié à l'utilisation de communications inter-nœuds comme on l'a déjà dit, entre 100 et 169 processeurs le temps pris par les calculs annexes (Figure 42) commençant à prendre plus d'importance.

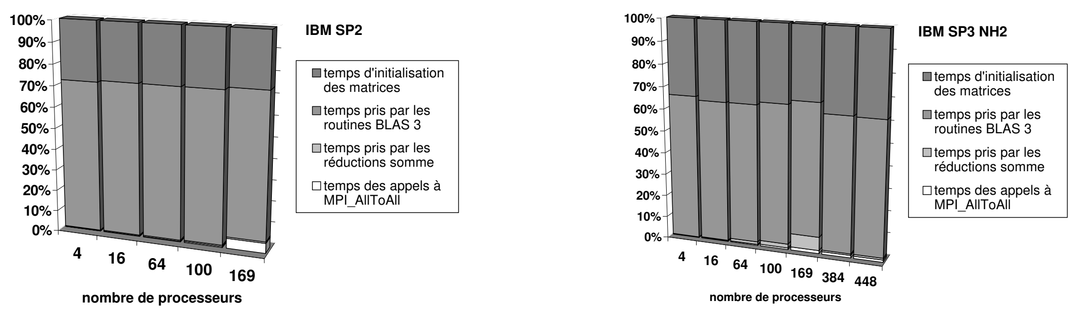


Figure 40. Répartition des temps pour les différentes tâches au sein d'une mise à jour de N ($S = 800$)

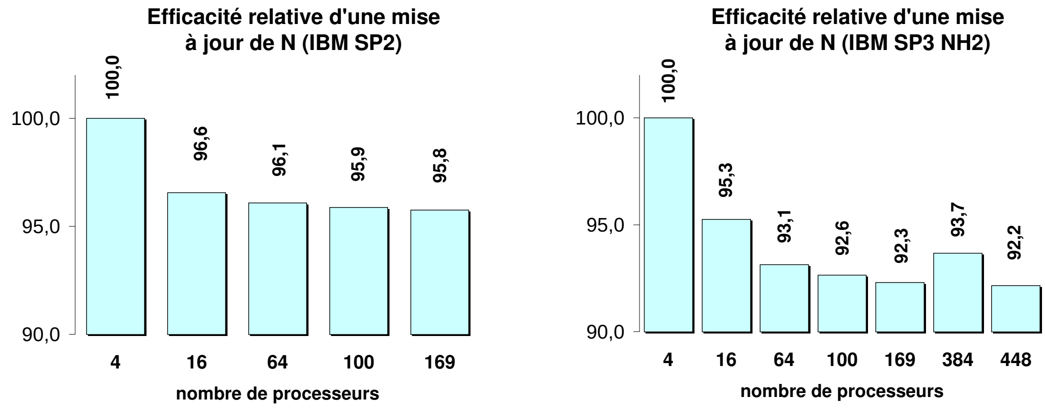


Figure 41. Efficacité relative d'une mise à jour de N ($S = 800$)

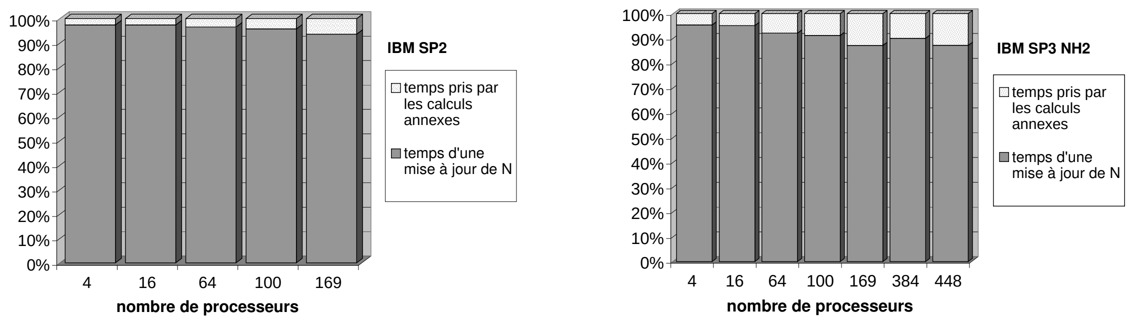


Figure 42. Répartition des temps pour les différents tâches réalisées par le simulateur ($S = 800$) au cours d'un pas de calcul

L'efficacité relative est supérieure à 77 % avec 448 processeurs, montrant ainsi la bonne extensibilité de l'algorithme (l'accélération est de 346). Dans le Tableau 6.2, on donne un aperçu des temps de simulation pour une simulation coûteuse. Les résultats de simulation sont disponibles après 50 minutes sur 64 processeurs et après moins de 9 minutes sur 448 processeurs (tous les nœuds de calcul NH2 du CINES).

6.4 Etude fine de la localité des données

6.4.1 Problèmes de grande taille, baisse des performances

La complexité maximale de la mise à jour de N dépend directement du nombre de parasites léthal pour un hôte. En effet, on a la relation $S \leq lethal$. Dans la plupart des simulations, on a employé les valeurs $lethal = 800$ ou 700, qui correspondent à celles utilisées précédemment dans [12, 34, 63]. Néanmoins, pour des hôtes résistants au parasitisme, on souhaiterait envisager des cas avec un paramètre $lethal$ plus grand. En l'augmentant

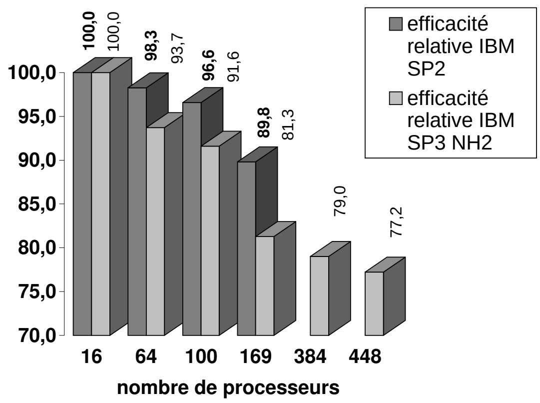


Figure 43. Efficacité relative pour une simulation complète ($>$ 105 TFLOP, $S = 800$)

	Nombre de processeurs : P						
	4	16	64	100	169	384	448
Temps (sec.) sur l'IBM SP2	-	55 781	14 192	9 242	5 879	-	-
Temps (sec.) sur l'IBM SP3	42 996	11 320	3 020	1 977	1 319	597	523

Tableau 2. Temps pour une simulation complète ($>$ 105 TFLOP pour $S = 800$) avec des calculs en double précision

de 800 à 1600, le coût d'une mise à jour de N pour laquelle S vaut *lethal* est multiplié par 16 (une mise à jour correspond alors à 15.1 TFLOP).

Lors de prises de performances, on a constaté que la puissance par processeur pour une mise à jour de N avec $S = 1600$ était réduite de plus de 20 % par rapport au cas où $S = 800$. L'outil HPM[27] a été utilisé pour compter le nombre moyen de références mémoire sans défauts de cache L2 et le nombre de MFLOPS par processeur. On observe alors que le nombre moyen de références sans défaut de cache est plus faible lorsque $S = 1600$, par rapport au cas $S = 800$. La localité mémoire est donc dégradée pour un problème de grande taille. En fait, les références mémoires effectuées au niveau du noyau de calcul et des initialisations de matrices intermédiaires nécessitent proportionnellement plus d'accès mémoire et moins d'accès aux caches lorsque $S = 1600$. On va donc modifier les noyaux de calculs pour améliorer, si possible, la localité temporelle et spatiale des données.

6.4.2 Réduction à un algorithme représentatif du problème

Le noyau de calcul est constitué des étapes suivantes (voir Figure 21 p. 69) :

1. initialisation de tmp_1 et multiplication de 2 matrices : $aux_1 = tmp_1\ N_*(c, *, t)$;
2. multiplication scalaire-matrice : $aux_2 = p(c)\, aux_1$;
3. initialisation de tmp_3 et multiplication de matrices : $aux_3 = tmp_3\ aux_2$;

27. Hardware Performance Monitor Toolkit d'IBM.

4. addition de 2 matrices : $N_*(*, i, t + \Delta t) = N_*(*, i, t + \Delta t) + aux_3$;

Les phases 1 et 3 nécessitent proportionnellement plus de calcul que les phases 2 et 4. Leur rôle se réduit au problème plus général suivant (avec g la fonction d'initialisation de tmp_1 ou tmp_3) :

$$\underline{\text{Problème } \Omega} \ : \ \forall m \in [1, dm], \forall n \in [1, dn] \quad c_{m,n} = \sum_{k=1}^{dk} g(k, n)\, a_{m,k} \ .$$

Dans ce problème, on pose dm nombre de lignes de la matrice A, dk nombre de colonnes de la matrice A, dn nombre de colonnes de la matrice C. Nous allons maintenant rechercher un algorithme efficace permettant de calculer le problème Ω qui se trouve à l'origine des mauvaises performances. Une solution immédiate consiste à écrire simplement 3 boucles imbriquées en m, n, k et une affectation au cœur des boucles. L'algorithme simple `algo_A` est donc :

Procédure `algo_A`(dm, dn, dk, A, C, g) {
  Pour $n := 1$ **à** dn **faire** {
  . **Pour** $k := 1$ **à** dk **faire** {
  . . $b := g(k,n)$;
  . . **Pour** $m := 1$ **à** dm **faire** {
  . . . $c_{m,n} := c_{m,n} + a_{m,k}\, b$;
  . . }
  . }
  }
}

Figure 44. Algorithme `algo_A` basique

Pour cet algorithme `algo_A`, on peut donner la responsabilité au compilateur d'optimiser les calculs. En conséquence le calcul peut ne pas être performant.

On peut faire apparaître des multiplications matrice-matrice dans le problème Ω. Ces multiplications peuvent alors être effectuées en faisant appel à la routine BLAS 3 GEMM (produit de deux matrices pleines[28]). Cela correspond à la solution employée précédemment dans le simulateur au niveau du noyau de calcul. C'est donc avec l'algorithme `algo_B` de la Figure 45 que l'on va détecter des problèmes des performances qui dépendent du paramètre *lethal*.

28. En pratique, l'étape 1 correspond à un appel à la primitive TRMM et l'étape 3 à deux appels successifs aux primitives TRMM et GEMM. Par soucis de lisibilité, on ne donne ici que le cas d'une multiplication de type GEMM, des détails sont disponibles en annexe pour TRMM.

```
Procédure algo_B(dm, dn, A, C, g) {
    B : matrice temporaire;
    Pour k := 1 à dk faire {
    .    Pour n := 1 à dn faire {
    .    .    b_{k,n} := g(k,n);
    .    }
    }
    C := A B;    /* utilisation de GEMM */
}
```

Figure 45. Algorithme `algo_B` utilisant le produit matriciel. Il constitue la brique de base du noyau de calcul utilisé dans les chapitres précédents.

Regardons les performances de ces deux algorithmes pour une matrice A quelconque et pour une fonction g dépendant[29] des indices k et n. On peut analyser l'influence des dimensions des matrices : dm, dn, dk. Dans les tests qui suivent sur IBM SP3 WH2, on prend $dn = dk$ (la matrice temporaire B est donc carrée).

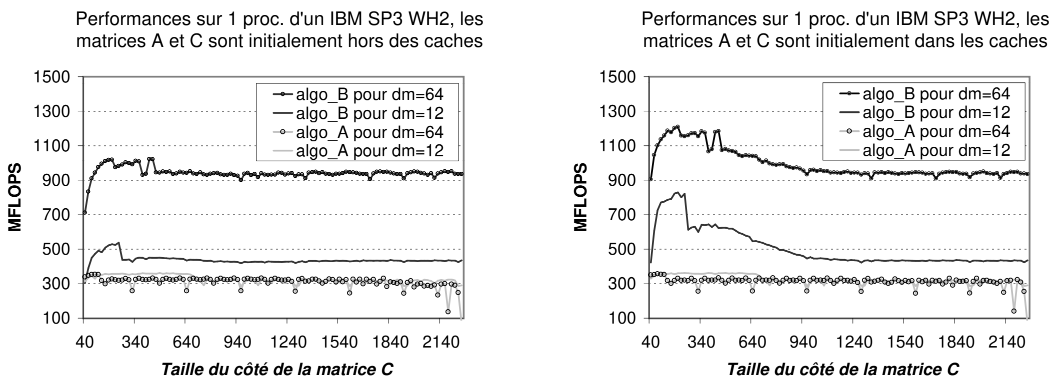


Figure 46. Comparaison des performances des 2 algorithmes sur un seul processeur d'un IBM WH2 pour $dm = 12$ ou $dm = 64$

Sur la Figure 46, les performances des algorithmes sont présentés pour des matrices A et C présentes ou non dans les caches au début des calculs. On remarque que l'algorithme `algo_A` présente une mauvaise performance de 300 MFLOPS que les matrices soient ou non dans le cache. L'algorithme `algo_B` réalise une bonne performance surtout lorsque dm est élevé. En effet, le nombre d'accès mémoire pour la multiplication de matrices est en $O(dn\,(dn + 2\,dm))$ alors que le nombre de calcul est en $O(dm\,dn^2)$; lorsque l'on augmente dm, on fait croître le ratio masse de calcul par rapport au nombre d'accès mémoire, et on réutilise donc plus efficacement les données du cache ce qui augmente les performances. A gauche, on constate que `algo_B` ($dm = 64$) atteint 900 MFLOPS en moyenne lorsque `algo_B` ($dm = 12$) correspond à moins de 500 MFLOPS. Autre remarque, la présence des matrices A et C dans les caches, Figure de droite, favorise les

29. Pour les tests nous avons pris $g(k,n) = 1 + (\frac{n}{10000})^{k-1}$, mais on peut prendre une autre fonction.

performances de `algo_B` lorsque dn est inférieur à 1000, par rapport au cas où elles ne sont pas dans le cache, Figure de gauche. Ceci est lié au fait que `algo_B` initialise B avant de réaliser la multiplication de matrices. Lorsque dn est très grand, les matrices A, B, et C peuvent sortir du cache entièrement ou partiellement pendant l'initialisation de B. En conséquence, les différentes matrices peuvent ne plus être dans le cache lorsque l'on réalise l'opération matricielle GEMM, ce qui dégrade les performances. On a constaté pour d'autres prises de performances sur les architectures IBM SP3 NH2 et Origin 3800 que l'ensemble des remarques précédentes étaient aussi pertinentes.

Dans le cadre du simulateur, il est important d'avoir de bonnes performances pour $dm = 9$ et $dn \in [1, 1600]$. D'autre part, d'un appel à l'autre le noyau de calcul du simulateur réutilise fréquemment la donnée $N_*(*, *, t)$, ce qui équivaut dans `algo_B` au cas où les matrices A et C sont dans le cache. On développe dans les prochains paragraphes une solution efficace qui améliore `algo_B` et évite les pertes de performances lorsque dn augmente. Pour cela, on va premièrement récapituler les techniques utilisées pour implémenter l'algorithme du GEMM et on donne une implémentation pour le SP3 WH2 en annexe. Deuxièmement, on verra que l'algorithme doit être adapté lorsque $dm = 9$ pour que les matrices A et B ne soient chargées dans le cache qu'une seule et unique fois, ceci afin d'économiser de la bande passante vers la mémoire. Dans un troisième temps, on couplera l'initialisation de la matrice B avec la multiplication de matrices afin d'améliorer l'exploitation de la localité en mémoire cache.

6.4.3 Algorithme DGEMM dédié

Une collection de techniques logicielles sont utilisées pour réaliser le produit de matrices de manière performante sur les machines superscalaires. Dans le cas des routines BLAS 3, elles dépendent fortement de paramètres architecturaux. La taille des blocs de données utilisées dans les calculs dépend notamment des caractéristiques des caches et des registres de la machine cible. Ce constat a conduit à l'élaboration de plusieurs types de bibliothèques de calcul utilisant les techniques d'optimisation de code standard :

- l'implémentation de référence des BLAS qui est portable mais n'est généralement pas la plus performante (disponible via `www.netlib.org`) ;
- les BLAS propriétaires développés par les constructeurs pour leurs machines (par exemple ESSL chez IBM, SCSL pour SGI) ;
- les BLAS paramétrables, qui peuvent s'ajuster en fonction d'un jeu de tests sur une machine (bibliothèques ATLAS [30], PHiPac [31]).

Même si les techniques pour approcher le puissance de crête avec les BLAS 3 sont intégrées par les bibliothèques, elles sont donc adaptées et paramétrées pour chaque machine. Nous avons développé des routines GEMM et TRMM optimisées pour les SP3 NH2 et WH2

30. Voir `http://math-atlas.sourceforge.net/`.
31. Voir `http://www.icsi.berkeley.edu/~bilmes/phipac/`.

qui tiennent compte des spécificités de l'architecture du processeur Power 3. Ce travail a ensuite été adapté pour d'autres machines de type superscalaire : SGI Origin 3800 et Power 4. Ensuite, le code de ces routines a été transformé afin de s'intégrer complètement au sein du simulateur.

En annexe C, on présente une implémentation d'une routine GEMM. Elle utilise les techniques de blocage de boucle et de blocage de registre qui optimisent la réutilisation des registres et des caches de données. Le stockage et l'accès aux données des matrices sont structurées pour utiliser efficacement le mécanisme de *prefetch* matériel, pour recouvrir les latences lors de références mémoire, et pour bénéficier de l'entrelacement mémoire.

6.4.4 Adaptation à une matrice avec une petite dimension

La routine GEMV des BLAS 2 consiste en une multiplication matrice-vecteur. La complexité des calculs (matrice carré de côté n) est $O(n^2)$ pour des données de taille aussi en $O(n^2)$. Ceci implique que le transfert de données vers le processeur constitue le goulot d'étranglement au niveau des performances. Pour chaque flottant chargé de la matrice à multiplier, seulement une multiplication et une addition vont être réalisées. La performance en MFLOPS est alors intrinsèquement liée à la bande passante mémoire. Ceci est d'autant plus limitant sur des machines à mémoire partagée lorsque la bande passante mémoire peut être divisée par le nombre de processeurs qui accèdent à la mémoire.

Pour la routine GEMM des BLAS 3 de multiplication de matrice-matrice, la situation est différente. Elle se distingue par une complexité en calcul (matrices carrés de côté n) en $O(n^3)$ pour des données de taille $O(n^2)$. Comme il y a beaucoup de calcul par rapport aux données, il est possible de réutiliser les données et de tirer partie de la mémoire cache. Afin d'améliorer l'accès aux données, on est amené à recopier initialement des données afin de diminuer les latences d'accès à la mémoire au cours des calculs (utilisation de l'entrelacement mémoire et du prefetch matériel). L'objectif visé est de réaliser le maximum des calculs au niveau du processeur (parallélisme d'instruction) en préparant toujours les données à l'avance pour éviter qu'il soit bloqué en attente de données.

Il existe une situation intermédiaire entre le GEMV et le GEMM, celle où l'on fait le produit de deux matrices $A.B$ et où la hauteur de A est petite. Dans ce cas, on adapte les techniques du GEMM : 1) en évitant les recopies mémoire comme pour le GEMV ; 2) en réutilisant intensivement les références chargées dans les caches pour ne pas avoir à y accéder par la suite (amélioration des localités spatiales et temporelles).

La Figure 47 présente une routine optimisée[32] `pmulti_6` comparées à l'emploi du GEMM constructeur (IBM ESSL) pour $dm = 9$. A droite, les 16 processeurs du nœuds IBM NH2 calculent simultanément, tandis qu'à gauche un processeur seulement calcule parmi tous ceux du nœud SMP. La bande passante mémoire est le facteur limitant des performances ; lorsque les processeurs d'un même nœud accèdent en même temps à la mémoire partagée, cela provoque une forte baisse des performances.

---

32. Le détail des techniques utilisées sont décrites en annexe C.3 p. 209. Le numéro qui termine le nom de la routine `pmulti_6` correspond à un niveau d'optimisation poussé.

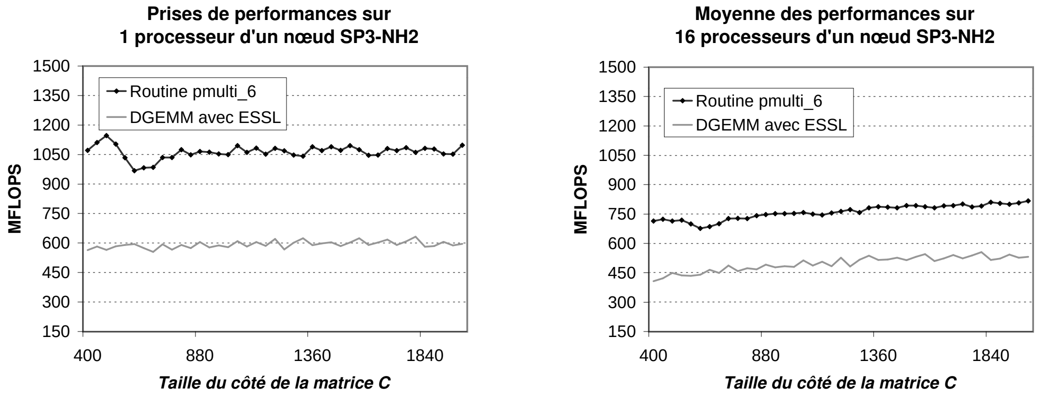


Figure 47. Comparaison des performances des routines `pmulti_6` et DGEMM constructeur sur un seul processeur IBM Power 3 d'un nœud (à gauche) ; comparaison des moyennes des performances lorsque tous les processeurs du nœud NH2 font le calcul simultanément (à droite) ; les Power 3 cadencés à 375Mhz ont chacun une puissance de crête de 1.5 GFLOP

On dispose maintenant d'un algorithme de multiplication performant même si l'une des dimensions de la matrice A est petite. Il s'agit maintenant de coupler cet algorithme avec l'initialisation de la matrice B par la fonction g.

6.4.5 Couplage de l'initialisation et de la multiplication

Reprenons l'algorithme `algo_B`. Si B est une matrice de grande taille, il est possible que l'on doive charger en mémoire B pour l'initialisation et une seconde fois pour la multiplication des matrices. Lorsque $dm = 9$ sur l'IBM SP3, le temps de chargement de la matrice B depuis la mémoire représente une fraction de temps non négligeable par rapport au temps de calcul du produit de matrices.

(∗ On suppose pour simplifier que $mod(dn, pas_n) = 0$, $mod(dk, pas_k) = 0$ ∗)
Procédure `algo_C(`$dm, dk, dn, A, C, \Phi, pas_k, pas_n$`)` {
 β `: matrice intermédiaire`
 Pour $kk := 1$ **à** dk **par pas de** pas_k **faire** {
 . **Pour** $nn := 1$ **à** dn **par pas de** pas_n **faire** {
 . . `call` $\Phi(\beta, kk, kk + pas_k - 1, nn3, nn3 + pas_n - 1)$
 . . $C(1..dm, nn..nn{+}pas_n{-}1) := C(1..dm, nn..nn{+}pas_n{-}1) +$
 . . $\qquad A(1..dm, kk..kk{+}pas_k{-}1)\,\beta$;
 . . }
 . }
 }
}

Figure 48. Algorithme par bloc `algo_C` utilisant une fonction d'initialisation par bloc Φ

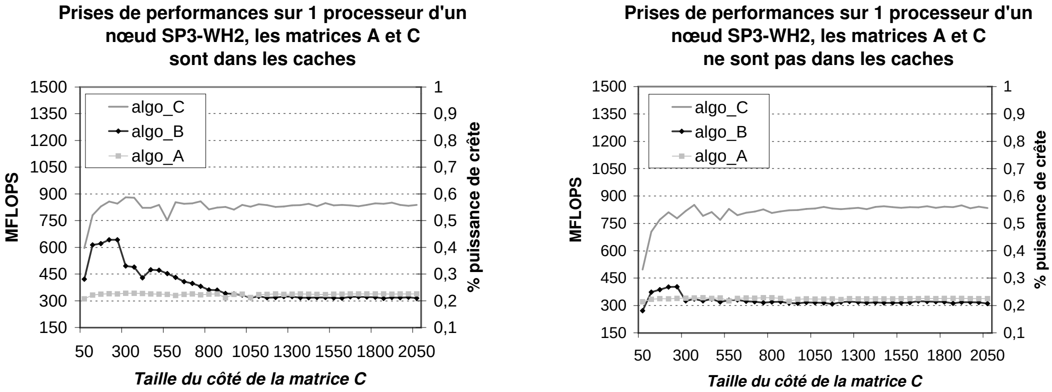


Figure 49. Comparaison des performances des 3 algorithmes sur un seul processeur IBM Power 3 d'un nœud WH2

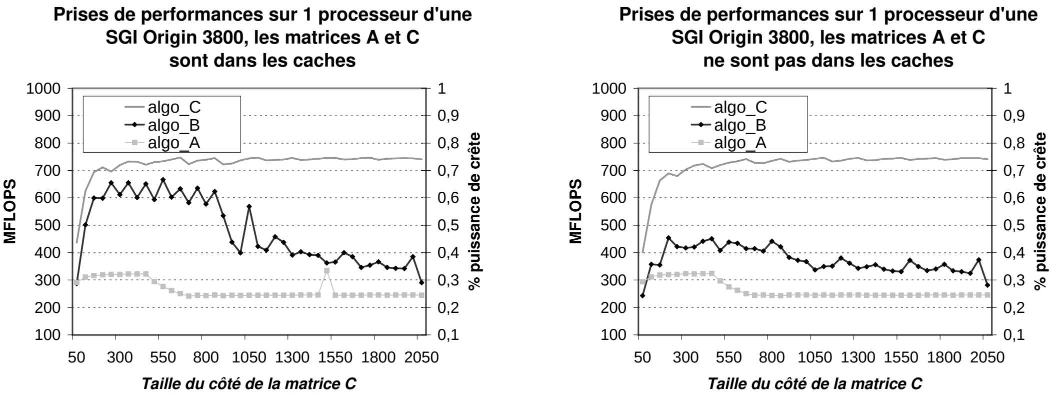


Figure 50. Comparaison des performances des 3 algorithmes sur un seul processeur R14000 de l'Origin 3800

Or, il est possible de coupler les étapes pour éviter tout rechargement. On prend, dans la procédure de la Figure 48, le principe d'une routine de multiplication avec blocage des boucles. Au niveau des deux boucles les plus externes, on ajoute l'initialisation d'une sous-matrice β de B en utilisant une fonction que l'on appelle Φ. L'appel à cette fonction $\Phi(\beta, k_a, k_b, n_a, n_b)$ affecte les éléments dans la matrice β de la manière suivante : $\forall k \in [k_a, k_b]$, $\forall n \in [n_a, n_b]$, $\beta_{k-k_a, n-n_a} = g(k, n)$. De cette manière, on va reconstituer B par morceaux, et utiliser les morceaux au fur et à mesure de l'exécution. On peut ainsi éviter de parcourir B deux fois grâce à cette réutilisation immédiate de sous-matrices de B. Les variables pas_k et pas_n constituent les dimensions du bloc pour l'algorithme `algo_C` de la Figure 48.

On consomme moins de mémoire en utilisant uniquement des sous-matrices β de taille $pask_k \times pas_n$ initialisées sur demande. Le seul bémol à cette méthode provient du passage d'une fonction Φ en paramètre. En effet, cela induit des changements de contexte

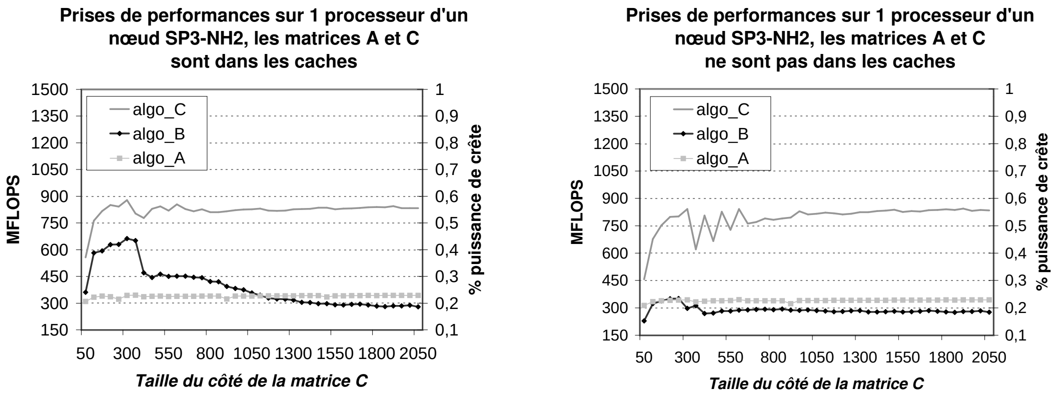


Figure 51. Comparaison des performances des 3 algorithmes sur un seul processeur IBM Power 3 d'un nœud NH2

à l'appel de la fonction dans les boucles, et abaisse ainsi un peu les performances. Cet inconvénient peut être minimisé en initialisant des sous-matrices β de plus grande taille ! Il y a ici un juste équilibre à trouver car si la sous-matrice β est trop grande, il se pose à nouveau le problème de rester dans le cache. Sur les figures 49, 50 et 51, on note le très bon comportement de l'`algo_C` par rapport aux autres algorithmes. Même pour de très petites matrices ($dm = 9$ et $dn = dk = 50$), l'`algo_C` est meilleur que l'`algo_B` qui utilise les BLAS 3 et ceci quelle que soit la machine : IBM SP3 ou SGI Origin 3800. Grâce au blocage des boucles, on conserve une bonne performance même lorsque la taille des matrices augmentent. La matrice β reste en effet dans les caches, on économise ainsi la bande passante mémoire. Entre les courbes de droite et celles de gauche, on constate que `algo_C` n'est pas sensible à la présence des matrices A et C dans le cache. La technique est de plus portable puisqu'elle se fonde sur une meilleure exploitation de la localité temporelle des données dans les caches.

6.4.6 Performance dans le simulateur

6.4.6.1 Haute performance de la solution intégrée

On mesure maintenant la performance des tâches $M(c, i)$ qui intègrent `algo_B` ou `algo_C` lors des simulations (voir les figures 52 et 53). Chaque tâche $M(c, i)$ s'exécute sur un unique processeur ; il est donc possible de calculer la puissance réalisée par celle-ci. Lors d'une mise à jour N telle que $S = 800$ (respectivement $S = 1600$), on a échantillonné les tâches $M(c, i)$ pour lesquelles c et i divisent 100. Les résultats pour `algo_B` sont sur la Figure 52 ; la courbe plafonne à 770 MFLOPS. Lorsque c est grand et i petit, le coût en calcul est maximal pour $M(c, i)$, mais malheureusement la performance est faible à moins de 400 MFLOPS.

La version couplée de `algo_C` est présentée Figure 53. On remarque que le nombre de MFLOPS est uniformément plus élevé qu'auparavant. Les performances sont généralement au dessus de 900 MFLOPS (60 % de la puissance crête). On distingue néanmoins une perte

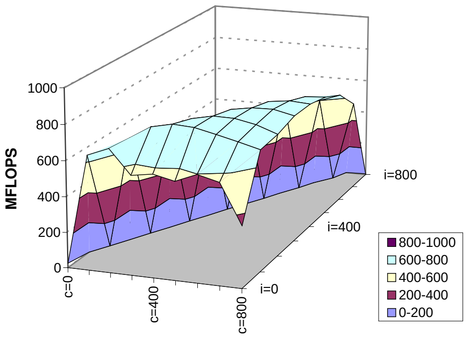


Figure 52. Performances des tâches $M(c,i)$ lorsque `algo_B` est utilisé sur l'IBM SP3 NH2 (simulation avec $S = 800$ à gauche et $S = 1600$ à droite)

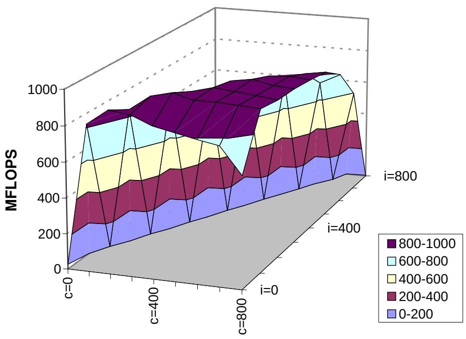


Figure 53. Performances des tâches $M(c,i)$ lorsque `algo_C` est utilisé sur l'IBM SP3 NH2 (simulation avec $S = 800$ à gauche et $S = 1600$ à droite)

de puissance lorsque c est grand et i proche de zéro. Premièrement, c'est une conséquence directe du rapport entre la taille des calculs qui utilisent la routine GEMM ou la routine TRMM au niveau du noyau de calcul. En effet, la routine TRMM est en général moins performante que la routine GEMM. Or ce rapport dépend uniquement des valeurs de i et c (voir Figure 21 p. 69). Deuxièmement dans ce cas de figure, le programme manipule les matrices les plus grosses qu'il ait à considérer au niveau de l'ensemble de l'algorithme.

L'algorithme couplé que nous avons développé permet d'avoir de bonnes performances lorsque $S = 1600$ et de plus améliore les résultats du simulateur pour $S = 800$. Le Tableau 6.3 qui suit l'illustre : pour $S = 800$, `algo_C` permet de réduire les temps d'exécution de 22 % par rapport à `algo_B`, et pour $S = 1600$ de 43 %. Sur l'IBM SP3 NH2, la mise à jour lorsque $S = 1600$ s'effectue à 58 % de la puissance de crête sur 128 processeurs (soit 112 GFLOPS).

Une simulation complète ($S = 800$) de 105 TFLOP sur 128 processeurs prend 19 minutes sur 128 processeurs (`algo_C`) soit 48 % de la puissance crête, contre 24 minutes pour `algo_B`. Une simulation de 1.15 PFLOP ($S = 1600$) a été calculée en 6h30 sur 32 processeurs de l'IBM Regatta (à base de Power 4), soit 30 % de la puissance de crête soutenue. Une simulation de 1.45 PFLOP ($S = 1600$) effectuée sur 128 processeurs demande

	$S = 800$			$S = 1600$		
	Nb. proc. : P			Nb. proc. : P		
	16	128	256	16	128	256
Temps (sec.) `algo_B`	85,7	12,0	6,1	1855	235,0	125,5
Temps (sec.) `algo_C`	66,5	9,4	4,8	1060	134,9	70,0

Tableau 3. Temps pour une mise à jour de N : 951 GFLOP pour $S = 800$ et 15.1 TFLOP pour $S = 1600$

4h13 de calcul au CINES sur IBM SP3 NH2. L'application soutient alors 50 % de la puissance crête des 8 nœuds.

6.4.6.2 Améliorations portables

La version optimisée du simulateur hôte-parasite dépend de 8 paramètres de blocage de boucles qui ajustent la version optimisée `algo_C` ; ils sont déterminés pour chaque architecture cible[33]. Afin de déterminer les valeurs de ces paramètres, on a mesuré les temps d'exécution de la mise à jour de N pour un échantillon de 1440 combinaisons de paramètres. Cette méthode est jugée préférable à l'utilisation d'une formule qui donnerait ces paramètres à partir de la description de la machine ; il y a en effet de nombreuses données liées à l'architecture à intégrer, ce qui conduirait à une combinatoire lourde. Il est plus simple, et pas moins efficace, de recourir à une exploration systématique des paramètres, à condition de les chercher dans un espace de valeurs qui considère un minimum d'hypothèses sur l'architecture considérée.

processeur	R14000, 500 Mhz		Power 3 WH2, 375Mhz		Power 3 NH2, 375Mhz		Power 4, 1.3 Ghz	
mesure	temps	MFLOPS % crête	temps	MFLOPS % crête	temps	MFLOPS % crête	temps	MFLOPS % crête
mise à jour de N `algo_C`	97,1s	612 61 %	69,1s	860 57 %	66,4s	895 60 %	36,8s	1544 30 %
simulation 92 TFLOP `algo_C`	9828s	581 58 %	7371s	774 52 %	6959s	820 55 %	4226s	1350 26 %
mise à jour de N `algo_B`	111,8s	532 53 %	112,5s	528 35 %	86,8s	685 46 %	71,0s	837 16 %
simulation 92 TFLOP `algo_B`	11270s	506 51 %	11485s	497 33 %	8942s	638 43 %	7119s	802 15 %

Tableau 4. Performances des algorithmes `algo_C` et `algo_B` pour 16 processeurs sur différentes architectures superscalaires

Dans le Tableau 6.4, les performances de l'algorithme couplé `algo_C` avec les paramètres trouvés sont comparées à celle de l'algorithme `algo_B` utilisant les BLAS constructeurs sur différentes architectures superscalaires (passage sur 16 processeurs et pour

33. ceci dépend de la taille des caches L1 et L2.

$S = 800$). Sur les architectures qui ont servi pour les tests, on constate que quelle que soit la taille des matrices du noyau de calcul, on gagne toujours à utiliser l'algorithme couplé optimisé plutôt que celui avec les BLAS constructeurs ($S = 800$). Si l'on compare la performance en nombre d'opérations réalisées par seconde et par processeur entre les deux versions, on constate que la version optimisée augmente la performance de 15 % sur ORIGIN 3800, 31 % sur IBM NH2, 63 % sur IBM WH2 et 93 % sur IBM Regatta (sur cette dernière machine les performances exploitables sont limitées par la bande passante mémoire). On conclut que l'utilisation des BLAS 3 comme solution pour atteindre de bonnes performances n'est pas forcément idéale même si les calculs semblent y être adaptés (`algo_B`). Les techniques d'optimisation qui sont à l'origine d'une utilisation efficace du processeur par la bibliothèque BLAS 3 sont par contre adaptables à un problème de calcul flottant régulier spécifique (`algo_C`). Les techniques présentées en annexe C et dans cette section visent notamment à minimiser l'utilisation de la bande passante et augmenter l'utilisation des caches. Ces méthodes sont essentiellement réutilisables lorsque le code effectue des accès redondants à la mémoire.

6.5 Conclusion

La complexité des calculs $C(S, K)$ d'une mise à jour est liée à deux variables du problème : S qui est bornée par le nombre maximum de parasites au-delà duquel le poisson meurt, et K qui est le nombre de classes d'âge de parasites utilisé (K=10). Le coût d'une mise à jour pour un pas de temps t a été ramené à $C(S, K) = O(K\,S^4)$ [32, 36]grâce à des précalculs, des factorisations, et des réordonnancement pour des multiplications de matrices. Ce coût est majoré par $C(800, 10) = 950$ GFLOP lorsque $S = 800$. Certaines parties de l'application, autres que la mise à jour de N, ont aussi été parallélisées afin de réduire au maximum la partie intrinsèquement séquentielle du simulateur.

L'algorithme de la mise à jour de N s'exprime sous forme vectorielle, ce qui permet de réduire les temps d'exécution en utilisant la bibliothèque de calcul BLAS 3. Deux distributions des calculs et des données ont été formulées, elles se basent sur le paradigme du parallélisme de données. L'efficacité, la scalabilité de ces distributions ont été évaluées théoriquement et en pratique. L'équilibrage des charges est bon dans le schéma 1D, mais encore meilleur avec le schéma 2D. Pour ce deuxième schéma, on peut estimer la taille de la grille de processeurs qui conduit aux meilleures performances. Une simulation complète nécessite 105 TFLOP lorsque $S = 800$ (un peu plus de 100 pas de temps à 950 GFLOP), ce qui prend 9 minutes pour 448 processeurs sur un IBM SP3 NH2. L'efficacité relative est bonne et reste au dessus de 77 % avec 448 processeurs pour ce type de simulation.

Nous effectuons aussi une étude fine de la localité des données pour améliorer les performances du noyau de calcul sur architecture superscalaire. Des simulations de grandes tailles (plus de 1 PFLOP) requièrent ces optimisations pour conserver des performances comparables aux simulations ordinaires. Pour cela, nous avons développé un algorithme spécifique, mais portable, dérivé de celui de la multiplication de matrices. Le couplage de l'initialisation et de la multiplication par blocs conduisent à de très bons résultats. On obtient un noyau de calcul qui atteint ainsi 60 % de la puissance crête sur 16 processeurs de l'IBM NH2 ou de l'Origin 3800, aussi bien lorsque $S = 800$ que lorsque $S = 1600$.

Chapitre 7

Simulation parallèle d'un modèle stochastique pour un système hôte-macroparasite

<< The tragedy is that too few researchers realize that both deterministic and stochastic models have important roles to play in the analysis of any particular system >> E. Renshaw, 1993

7.1 Présentation de la simulation stochastique

Les techniques algorithmiques mises en œuvre pour une simulation stochastique basée sur des méthodes de type Monte-Carlo sont présentées. Ce chapitre fait suite aux parties I-2.3.2 et II-5. Les performances associées à une programmation hybride sur une grappe de nœuds SMP sont exposées. Une comparaison des deux simulateurs du système hôte-macroparasite est donnée en termes de performances.

7.1.1 Description du contexte

Le modèle biomathématique utilisé ici est issu d'une modélisation déterministe d'un problème de dynamique de population. Ce type de modèle fait généralement abstraction de certains processus se déroulant à l'échelle des individus et privilégie les phénomènes macroscopiques au niveau des populations. D'un autre côté, les modèles individu-centrés se proposent de reproduire fidèlement des processus élémentaires naturels, mais de ce fait conduisent parfois à des coûts de calcul élevés. Des modèles individu-centrés ont été implémentés sur machine parallèle, car certains systèmes complexes nécessitent donc de grandes puissances de calcul. Un nombre important de *réplications* (simulations élémentaires) doivent être réalisées. Aucune réplication prise individuellement n'est représentative, car chacune d'elles est dirigée par des *événements aléatoires*. Finalement, on prendra donc la *moyenne* de R réplications distinctes.

La comparaison de ces modèles est pertinente tant mathématiquement que biologiquement. Le modèle individu-centré devrait apporter des informations non disponibles dans le cas déterministe, notamment le spectre des variables de sortie. La version stochastique permettra aussi de savoir si certaines modifications de bas niveau peuvent influer sur la dynamique globale, car ceci n'est pas aisé à évaluer sur le modèle déterministe plus rigide à cause des fondements mathématiques sous-jacents. Il sera par exemple possible d'avoir une mortalité différenciée en âge, ce qui n'était pas le cas pour le modèle déterministe (*cf* p. 44). La présente étude se propose d'examiner une simulation stochastique pour le système *Bar–Diplectanum aequans* dans un contexte d'élevage. Après une présentation du contexte de la simulation stochastique parallèle, l'algorithme utilisé dans l'application est donné. Les performances d'une programmation hybride MPI/OpenMP sont présentées jusqu'à 256 processeurs pour un IBM SP3 NH2. Les courbes produites par le simulateur stochastique parallèle sont comparées à celle du simulateur déterministe dans la partie suivante.

7.1.2 État de l'art de la simulation parallèle stochastique

Plusieurs stratégies de parallélisation dans le cadre de simulations de type stochastique sont présentes dans la littérature. La première consiste à utiliser l'ensemble des processeurs disponibles pour effectuer une réplication. Les réplications sont alors effectuées les unes après les autres. Le problème est généralement décomposé spatialement. S'il s'agit d'agents, l'espace où ils peuvent évoluer est typiquement réparti entre les processeurs [41, 43]. Pour un algorithme reposant sur un automate cellulaire, des sous-ensembles de la grille sont distribués [9]. Néanmoins, ceci est possible essentiellement lorsque la taille des calculs est suffisante sur chaque processeur pour que le grain de calcul soit adapté à la machine cible.

Une autre stratégie plus générale dans le cadre des simulations de type Monte-Carlo consiste à distribuer les réplications ou simulations élémentaires sur les processeurs. Dans cette optique, les réplications sont exécutées séparément sur les processeurs sans interactions entre elles. A la fin des calculs, il y a une réduction des diverses sorties réparties sur les processeurs de manière à générer une synthèse des sorties (historiques de certaines réplications, moyenne des variables d'état, variance et autres statistiques). Il y a certaines limitations dans cette manière de procéder. D'abord, si les réplications ne prennent pas un temps équivalent sur chaque processeur un surcoût est généré. Un équilibrage dynamique de charge doit alors être envisagé. Ensuite, le nombre R de réplications nécessaires limite le nombre de processeurs P que l'on peut employer, car $P \leq R$. Enfin, le surcoût dû à la fusion qui synthétise le résultat final ne doit pas être trop important relativement aux coûts des réplications. Cette deuxième approche est aussi envisagée dans la littérature [10] car elle se prête à une parallélisation massive. Les problèmes à résoudre sont : la génération aléatoire de séries de nombres disjointes et non corrélées sur les processeurs, et l'équilibrage des charges. Nous reviendrons sur ces points ultérieurement.

Enfin, la validation et l'évaluation de modèles dans le cadre de la simulation peut comporter une analyse de sensibilité. En effet, cette exploration du comportement du modèle en fonction des paramètres d'entrée améliore la compréhension du modèle. Il est intéressant de savoir quels sont les paramètres qui influencent le plus les sorties. La navigation dans l'espace des entrées du modèle peut prendre un temps important et peut être difficile à mettre en œuvre par l'utilisateur. L'agrégation des résultats et leur structuration dans ce cadre consomment de l'espace disque et du temps. Or la réalisation d'une série de simulations peut être effectuée en parallèle et de manière automatique. La synthèse des résultats nécessite au final la coopération de tous les processeurs. Ce troisième niveau de parallélisme est peu documenté dans la littérature [22], mais remarquons qu'il reste hors de portée pour les simulations lourdes.

7.1.3 Modélisation stochastique des interactions hôte-parasite

Dans une modélisation individu-centrée, on considère classiquement les interactions élémentaires entre les éléments du système. On représente d'une part les hôtes présents dans le bassin, d'autre part les parasites fixés. Soit $\mathbb{H}_i$ l'objet hôte numéro i. Par rapport au modèle déterministe donnant un résultat avec une simulation, la synthèse de plusieurs réplications est nécessaire dans le cadre stochastique pour générer un résultat représentatif. Le nombre R de réplications sera fonction de la précision voulue pour les sorties du simulateur. La gestion de ces interactions hôte-parasite est décrite ci-après (voir Figure 54) :

- La probabilité de survie d'un hôte ayant l parasites est $p(l)$. On procède à un tirage aléatoire pour déterminer si $\mathbb{H}_i$ vivant à l'instant t meurt d'ici au temps $t+\Delta t$. On tire aléatoirement et uniformément un réel x sur l'intervalle $[0, 1]$. Si x est supérieur à $p(l)$, l'hôte $\mathbb{H}_i$ meurt.
- On considère $P_i(q, t)$ la quantité de parasites d'âge q fixé sur l'hôte $\mathbb{H}_i$ au pas de temps situé entre t et $t+\Delta t$ et $P_i(t)$ le nombre total de parasites sur ce même hôte. Si la température de l'eau est $\theta(t)$, le taux de mortalité des parasites est $\mu(\theta(t))$. Une loi binomiale $\mathcal{B}(P_i(q, t); \mu(\theta(t)))$ est utilisée pour calculer combien de parasites meurent parmi $P_i(q, t)$ durant le pas de temps t. Tous les parasites survivants sont transférés dans la classe d'âge suivante $P_i(q+1, t)$ (pour $0 < q < K$), voir Figure 54. En bref, pour chaque hôte et chaque classe d'âge de parasites, un tirage aléatoire selon une loi binomiale fixe le nombre de parasites qui survivent au pas de temps t. Pour la dernière classe d'âge K, les parasites survivants restent dans cette même classe d'âge.

- Le recrutement de $L_r(t)$ larves sur $H(t)$ hôtes doit être réalisé. L'espérance de recrutement en larves fixées d'un hôte ayant l parasites est $f(l,t)\,L_r(t)$ (comme dans le modèle déterministe). Soit $P_i(t)$ le nombre de parasites sur l'hôte $\mathbb{H}_i$ et $R_i(t)$ la variable aléatoire désignant le nombre de larves que $\mathbb{H}_i$ recrute au pas de temps t. La fonction f est telle que :

$$\sum_{i/avec\ \mathbb{H}_i\ vivant\ au\ temps\ t} f(R_i(t),t)\,L_r(t) = L_r(t)\,. \qquad (38)$$

Pour chaque larve qui doit se fixer, tout hôte vivant $\mathbb{H}_i$ a la probabilité $f(P_i(t),t)$ de la recruter. Soit $i_1, i_2.., i_{H(t)}$ désignant les indices des hôtes vivants au temps t. La loi de probabilité la plus simple correspondant à ce problème consiste à dire que $(R_{i_1}(t), R_{i_2}(t), ..R_{i_{H(t)}}(t))$ suit la loi multinomiale $\mathcal{B}(L_r(t);\ f(P_{i_1}(t),t),\ f(P_{i_2}(t),t)..,\ f(P_{i_{H(t)}}(t),t))$. On note que dans ce cadre, chaque $R_j(t)$ suit la loi binomiale $\mathcal{B}(L_r(t); f(P_j(t),t))$, et que l'on a la propriété $R_{i_1}(t)+R_{i_2}(t)+\,..\,+R_{i_{H(t)}}(t)=L_r(t)$. Les variables aléatoires $R_j(t)$ ne sont pas indépendantes.

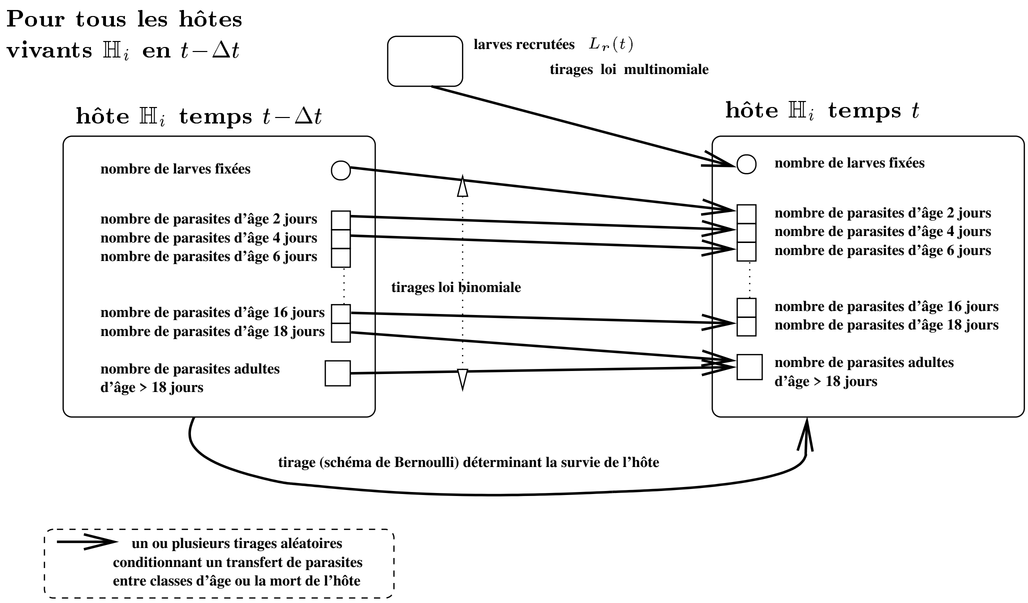


Figure 54. Schéma de la mise à jour des hôtes et des parasites au pas de temps t

Pour pouvoir comparer les deux modèles stochastique et déterministe, il est important que les différentes composantes du modèle soient représentables de manière similaire dans les deux cas. Par exemple le modèle stochastique n'envisage que des nombres entiers d'individus, alors que le déterministe non. Le modèle déterministe a été modifié pour en tenir compte. D'autre part, les lois de distributions utilisées dans le modèle déterministe sont choisies telles que la simulation stochastique soit possible ; par exemple, le recrutement est fait avec une loi binomiale et non pas avec une loi de Poisson dans le modèle déterministe (*cf* p. 43).

7.2 Algorithme stochastique

7.2.1 Complexité de l'algorithme

L'algorithme utilisé pour le modèle stochastique est présenté dans la Figure 55.

1. lecture des paramètres d'entrée ;
2. Pour r := 1 au nombre de réplications voulues R ;
2.1 calcul des valeurs initiales des variables ;
2.2 Pour t := 0 à 366 avec un pas de temps de 2 jours
2.2.1 mise à jour des données environnementales ;
2.2.2 ponte des œufs par les parasites adultes ;
2.2.3 mise à jour de la population d'œufs (vieillissement) ;
2.2.4 éclosion des œufs (donnant des larves nageantes) ;
2.2.5 mise à jour de la population larvaire (vieillissement) ;
2.2.6 recrutement de larves par les hôtes (mise à jour avec tirages aléatoires) ;
2.2.7 mort des hôtes trop parasités (mise à jour avec tirages aléatoires) ;
2.2.8 vieillissement, mortalité des parasites (mise à jour avec tirages aléatoires) ;
Fin Pour
2.3 enregistrement des sorties de la réplication « r » ;
Fin Pour
3. fusion des résultats des réplications et affichage des données significatives (moyenne, variance des variables d'état).

Figure 55. Algorithme stochastique global

Les étapes concernant les interactions directes hôte-parasite (*c.a.d.* 2.2.6, 2.2.7 et 2.2.8) constituent la partie la plus coûteuse de l'algorithme (comme dans le modèle déterministe *cf* p. 59). Un relevé des temps d'exécution des différentes étapes a été produit sur un ensemble de jeux de paramètres représentatifs. Celui-ci révèle que pour les cas examinés, les étapes 2.2.6, 2.2.7 et 2.2.8 prennent toujours au moins 89 % du temps d'exécution total pour l'ensemble des réplications. Examinons la répartition des coûts pour des simulations qui gèrent de nombreux hôtes et parasites. En effet, c'est dans ces cas qu'il y a le plus de calculs à réaliser. Dans ces simulations, l'épizootie s'étale sur au moins la moitié de l'année. Lors de l'épizootie, plus de $1.2\,10^6$ parasites et plus de 4000 hôtes peuvent être gérés par pas de temps. Nous verrons que la complexité des calculs est directement fonction du nombre de larves recrutées et du nombre d'hôtes.

L'étape 2.2.6 est une étape relativement coûteuse. Dans cette étape, il faut en effet répartir tous les parasites qui se fixent sur les hôtes présents dans le milieu à l'aide d'une loi multinomiale. Avec la méthode élémentaire pour simuler une loi multinomiale, cela nécessite autant de tirages que de parasites qui se fixent. Avec la notation introduite plus haut, cela signifie que la complexité de l'étape est en $\Theta(L_r(t))$.

L'étape 2.2.7 est une étape moins coûteuse. Pour chaque hôte, il faut déterminer s'il survit à sa charge parasitaire ou non. Il suffit pour cela d'un seul tirage aléatoire par hôte, d'où une complexité $\Theta(H(t))$.

L'étape 2.2.8 consiste à déterminer pour chaque parasite fixé sur un hôte s'il survit ou non au pas de temps suivant. On peut effectuer un tirage aléatoire par parasite, et à partir d'une valeur seuil, savoir si le parasite meurt ou non. En fait, on peut réduire substantiellement le nombre de tirages à effectuer. En effet, si l'on considère les parasites fixés sur un hôte et dans une certaine classe d'âge, ils ont tous le même taux de survie. On peut donc simuler la mort de ces parasites par une loi binomiale $\mathcal{B}(\mu(\theta(t)), u)$, avec u le nombre de parasites dans une classe d'âge d'un certain hôte. Finalement, on remplace une suite d'expériences de Bernoulli par une variable aléatoire qui suit la loi binomiale. Si l'on résume maintenant ce qu'il reste à simuler, il y a $H(t)$ hôtes, et sur chacun d'entre eux K classes d'âge. Les parasites vieillissent en passant d'une classe d'âge à l'autre ; ces classes constituent des compartiments. En tirant $K\,H(t)$ nombres aléatoirement avec une loi binomiale, on réalise le vieillissement de tous les classes d'âge présentes sur les hôtes (complexité de $\Theta(K\,H(t))$).

La complexité d'une mise à jour pour les étapes 2.2.6, 2.2.7 et 2.2.8 s'élève donc à $\Theta(L_r(t) + H(t))$. Pour des simulations relativement coûteuses en calcul, on constate que l'étape 2.2.6 représente plus de 90 % du temps total d'exécution, la relation $H(t) \ll L_r(t)$ se vérifie alors. Pour ces simulations coûteuses, on en déduit que la complexité totale de toutes les mises à jour pour une réplication est approximativement $\Theta(\sum_{t\in[0,366]} L_r(t))$. Or la somme des larves recrutées sur une année peut atteindre $1.8\,10^8$ pour certaines simulations. Avec R réplications, la complexité atteint donc $\Theta(R \sum_{t\in[0,366]} L_r(t))$. On remarque que celle-ci est complètement différente de la complexité du simulateur déterministe, et qu'*a priori* et selon les situations, l'un ou l'autre des simulateurs sera donc plus économique en temps de calcul.

Les données principales manipulées dans l'algorithme sont explicitées dans la figure 54. Une quantité $H(t)$ d'objets « hôte » sont gérés. Chaque objet dispose de $K+1 = 11$ compteurs pour comptabiliser les parasites dans les différentes classes d'âge. La complexité en mémoire est donc $\Theta(11\,H(t))$. L'étude porte sur des bassins de 5000 poissons, ce qui induit donc un faible coût mémoire. Néanmoins, si on souhaite conserver J variables d'état à chaque pas de temps pour les statistiques réalisées à la fin des R réplications, la complexité de l'historique est $366\,J\,R$.

7.2.2 Algorithme parallèle

Trois niveaux de parallélisme sont évoqués dans la littérature concernant les simulations stochastiques. L'architecture cible choisie est ici une grappe de multiprocesseurs à mémoire partagée. Elle comporte deux niveaux de communication : inter-nœuds et intra-nœuds plus rapide. Pour le premier niveau de parallélisme, le domaine que l'on peut distribuer est constitué par l'ensemble des hôtes et parasites vivants à l'instant t. Notamment, les calculs de l'étape 2.2.6 décrite précédemment peuvent être distribués. Dans cette étape, une fois les nombres aléatoires générés, il faut déterminer sur quel hôte se fixe une larve donnée. On parallélise donc en découpant la boucle sur les larves. Pour ce parallélisme à grain fin, la mémoire partagée est ici un atout pour réaliser de bonnes performances en réduisant les coûts de communication. Dans ce contexte, il est possible d'employer OpenMP [57], dont la mise en œuvre ne se heurte à aucune difficulté. Le nombre de processeurs utilisés devra être adapté au faible coût de calcul concerné par ce premier niveau de parallélisme.

Le second niveau de parallélisme consiste à distribuer les différentes réplications sur les processeurs. Il s'agit là d'un parallélisme à plus gros grain. Si l'on utilise uniquement ce niveau de parallélisme, le nombre de processeurs doit être inférieur ou égal au nombre de réplications, $P \leq R$. Il se pose le problème de l'équilibrage de charge lorsque plusieurs réplications doivent être réalisées sur chaque processeur. Or des séries d'expérimentations ont permis de constater qu'il n'y a généralement pas de variation très importante des temps pour les réplications (pour un jeu de paramètres donné). Toutes les réplications étant effectuées, une phase de réduction synthétise le résultat de sortie. Le second niveau de parallélisme est basé sur un ensemble de processus MPI. En effet, on ne souhaite pas utiliser ici OpenMP et s'interdire de réaliser une exécution sur des nœuds différents. La réduction est effectuée à l'aide des routines de communications globales MPI et fait intervenir tous les processeurs. L'algorithme parallèle est présenté dans la Figure 56.

Lorsque une analyse de sensibilité est effectuée (troisième niveau de parallélisme), la boucle externe (variable a) est distribuée sur sp ensembles de processeurs. Les indices $a \in [1, A]$ sont assignés aux sp groupes cycliquement afin d'équilibrer les charges. Pour l'analyse de sensibilité, les simulations utilisent des paramètres d'entrée qui explorent le voisinage d'une simulation de référence. Les simulations se ressemblent donc du point de vue du comportement qualitatif de la dynamique. En première approximation, on suppose que le coût des simulations est quasi-constant pour une analyse de sensibilité ; les charges seront donc bien réparties si A divise sp.

```
Pour les simul. de l'analyse de sensibilité a ∈ [1,A] faire en parallèle {
.   Lecture des paramètres d'entrée ;
.   Pour toutes les réplications r ∈ [1,R] faire en parallèle {
.   .  calcul des valeurs initiales des variables ;
.   .  Pour t:=0 à 366 par pas de 2 faire
.   .  .  mises à jour des étapes 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5 ;
.   .  .  mise à jour parallèle de l'étape 2.2.6 ; (threads OpenMP)
.   .  .  mises à jour des étapes 2.2.7, 2.2.8 ;
.   .  }
.   }
.   réduction, synthèse des sorties (communication collective MPI) ;
}
```

Figure 56. Algorithme stochastique parallèle

La bibliothèque PRNGlib [46] a été utilisée pour la génération aléatoire parallèle. Plusieurs bibliothèques sont disponibles sur le web ; elles permettent la génération indépendante de vecteurs de nombres aléatoires [45] et disposent de qualités reconnues. Pour être utilisable dans une application de type Monte-Carlo, un générateur aléatoire doit premièrement passer avec succès des tests statistiques théoriques et empiriques. Il doit avoir une longue période et se calculer rapidement en parallèle. Le générateur parallèle doit prendre en charge plusieurs types de générateurs élémentaires pour comparer leur adéquation avec le simulateur. En effet, certains générateurs peuvent introduire des biais statistiques dans les applications. On doit pouvoir aussi générer la même série de nombres quel que soit le nombre de processeurs, car cela conduit à la reproductibilité des simulations. Le générateur intégré au simulateur est initialisé à partir d'une graine (*seed*). Il donne R séries de nombres que l'on peut répartir sur les processeurs, selon convenance. Sur 1 ou 128 processeurs, il est ainsi possible de reproduire la même simulation et des résultats finaux identiques. Ceci permet de déboguer plus facilement en levant l'indéterminisme. Cela confère à l'application de la robustesse et améliore la portabilité. Bien que le simulateur soit parallèle, les événements générés dépendent uniquement de la graine initiale. Bien sûr, si la graine change, les résultats diffèrent quelque peu, mais d'autant moins que le nombre de réplications est important (*cf* p. 121).

7.2.3 Programmation hybride sur IBM SP3 NH2

Pour les prises de performances, un jeu de paramètres représentatif des simulations coûteuses en calcul a été choisi. Pour obtenir une synthèse et une moyenne acceptable, on estime que $R=32$ à 512 réplications sont nécessaires. Dans un premier temps, on va étudier les deux premiers niveaux de parallélisme en excluant celui concernant l'analyse de sensibilité (donc $sp = 1$). Les performances en termes de temps et d'efficacité relative sont données dans la figure 57 ($R=32$). Dans ces tests, le nombre de processeurs est égal à $m \times nt$, avec m le nombre de processus MPI et nt le nombre de processus légers dans chaque processus MPI. La machine utilisée pour les tests est le IBM SP3 du CINES à base de nœud NH2 à 16 processeurs (375 Mhz). Une simulation comportant 32 réplications permet d'avoir le comportement global du système pour le jeu de paramètres utilisé. La distribution autour de la moyenne peut alors être analysée, ce qui constitue un résultat de la dynamique du système. Dans le cas où $R=32$, l'utilisation du parallélisme à grain fin donne la possibilité d'exploiter plus de processeurs que le nombre de réplications. Néanmoins on constate que l'efficacité relative est plus faible lorsque l'on utilise de 2 à 4 threads (Figure 57). Par exemple pour 32 processeurs, on regarde la séquence soulignée dans le Tableau avec $m \times nt = 32 \times 1$, 16×2, 8×4. Ces résultats montrent que le code exclusivement MPI est plus performant que le code hybride. Par contre, la programmation hybride permet d'utiliser 128 processeurs lorsque $R=32$ et permet donc d'avoir plus de processeurs que de réplications. La programmation hybride permet de paralléliser finement l'application de telle façon qu'une simulation s'exécute en 81s sur 64 processeurs et 59,7s sur 128 processeurs.

Les directives OpenMP ajoute donc un parallélisme à grain fin au simulateur. Ce premier niveau de parallélisme correspond à la parallélisation de la boucle sur les larves comprenant typiquement de nombreuses itérations (par exemple 2×10^8 durant l'épizootie). Les structures de données utilisées dans cette boucle peuvent être partagées

	Nombre de processus MPI (m)				
	1	4	8	16	32
$nt=1$ thread	3669,9s	935,3 s	471,3s	238,8s	123,8s
$nt=2$ threads	2385,5s	609,1s	307,5s	155,9s	81,0s
$nt=3$ threads	1963,5s	500,3s	252,4s	127,8s	67,3s
$nt=4$ threads	1745,9s	469,4s	228,1s	119,1s	59,7s

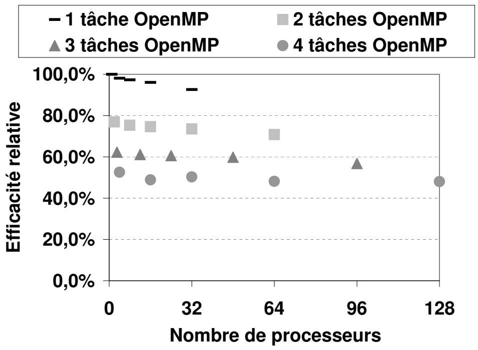


Figure 57. Mesures de temps et efficacité relative sur IBM SP3 NH2 ($R = 32$) ; on a m processus MPI, nt processus légers dans chaque processus MPI, soit $P=m\times nt$ processeurs

grâce à la programmation en mémoire partagée. Si nous considérions une version MPI équivalente, cela engendrerait un surcoût de communication pour transmettre ces données entre les processeurs.

De plus, le temps de calcul passé dans cette boucle représente en moyenne 81 % du temps de calcul en séquentiel t_{seq}. La loi d'Amdahl pose une limite à l'accroissement de vitesse due à la parallélisation d'un programme qui comporte une partie strictement séquentielle. Soient p_{seq} la fraction du temps d'exécution correspondant à la partie séquentielle, P le nombre de processeurs utilisés, E l'efficacité et S_{peedup} le speedup, on a :

$$S_{peedup} \le \frac{1}{p_{seq} + (1-p_{seq})/P} \le \frac{1}{p_{seq}} \qquad E \le \frac{1}{P\,p_{seq} + (1-p_{seq})} \le \frac{1}{P\,p_{seq}}\ .$$

La partie séquentielle représente 19 % du temps d'une réplication dans le cas présent, la boucle est découpée en nt blocs. Par application de la loi d'Amdahl, on a $E \le \frac{1}{0{,}19\,nt+0{,}81}$; pour $nt = 2$, on a $E \le 84$ % et pour $nt = 4$, on a $E \le 64$ %. Précisons que cette limitation n'implique que le parallélisme de premier niveau et non pas celui de second niveau. Ces efficacités sont des limites supérieures, et les efficacités observées correspondantes sont sur la Figure 57. On constate que l'efficacité décroît très vite lorsque le nombre de threads OpenMP augmente (c'est-à-dire nt). On a vérifié d'autre part que les performances d'OpenMP sont exactement similaires à une implémentation avec des threads POSIX pour cette application. La loi d'Amdahl explique complètement la décroissance de l'efficacité fonction de nt.

Le temps minimal d'une simulation coûteuse sur l'IBM SP3 tombe de 28 minutes sur 128 processeurs pour la simulation déterministe à 1 minute pour la simulation stochastique (avec $m=32,\ nt=4$). La méthode offre donc un avantage important du point de vue des temps de calcul. Dans cette application, le parallélisme de deuxième niveau offre de meilleures performances que celui de premier niveau. D'autre part, on remarque que le placement d'un nombre identique de réplications sur chaque processus MPI suffit en général à équilibrer les charges. Précisons que les communications MPI sont utilisées essentiellement à deux reprises dans le simulateur : premièrement pour enclencher l'initialisation des générateurs aléatoires sur les processeurs (quasi-instantané), au final pour réaliser la synthèse des résultats. Les communications se réduisent donc à la réduction sur un processeur de l'historique de toutes les réplications de taille 366 J R flottants. Par exemple, lorsque l'on se place dans le cas $nt = 1$ avec une simulation coûteuse et R processeurs, les temps de communications représentent au plus 1.5 % du temps d'exécution total. Les communications ne représentent clairement pas un goulot d'étranglement.

Lorsque l'on utilise le simulateur avec $R = 512$, on obtient le spectre des variables de sortie. Le coût en temps est naturellement plus important, mais cela permet d'avoir des informations plus précises sur la forme de la loi de distribution des différentes variables de sorties.

7.2.4 Parallélisme de troisième niveau

La combinaison des deux premiers niveaux de parallélisme a été décrite. On va maintenant insister sur l'utilisation simultanée des deuxième et troisième niveaux, et on exclut donc le parallélisme à grain fin ($nt = 1$). Le nombre de processeurs est donné par $P = sp \times m$. On distribue cycliquement les simulations sur les ensembles de processeurs. Chaque groupe de processeurs n'est pas forcément responsable du même nombre de simulations. La Figure 58 illustre les performances pour deux analyses de sensibilité pour lesquelles $A = 15$ ou $A = 41$.

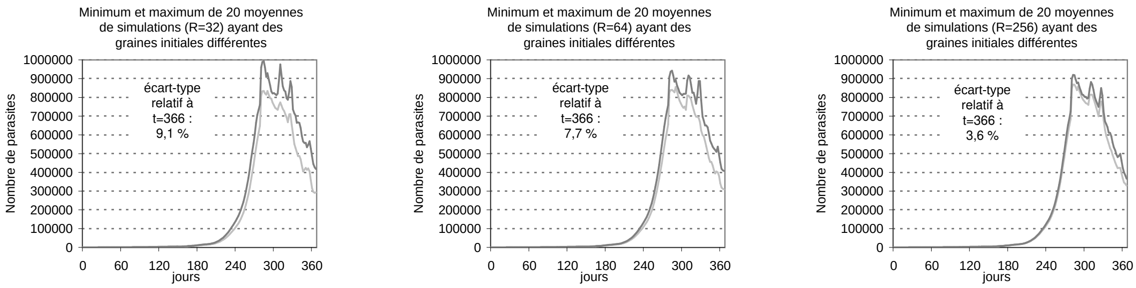
A=15	Nombre d'ensemble de processeurs (sp)			
	1	2	4	16
$P=32$	1677s 100,0%	1725s 97,2%	1754s 95,6%	1697s 98,8%
$P=64$	–	897s 93,5%	861s 97,4%	861s 97,4%
$P=128$	–	–	445s 94,2%	438s 95,7%
$P=256$	–	–	–	223s 94,0%

A=41	Nombre d'ensemble de processeurs (sp)			
	1	2	4	16
$P=32$	4444s 100,0%	4607s 96,5%	4531s 98,1%	5438s 81,7%
$P=64$	–	2319s 95,8%	2298s 96,7%	2566s 86,6%
$P=128$	–	–	1197 92,8%	1344s 82,6%
$P=256$	–	–	–	682s 81,4%

Figure 58. Temps d'exécution et efficacité relative de deux analyses de sensibilité avec $A=15$ et $A=41$; on a $P=sp\times m$ processeurs avec $R=32$

Le nombre de processeurs dans un groupe est au plus 32 car $R = 32$ et $P \leq R$. Ainsi le nombre de processeurs utilisables est au total $sp\times R$ (les configurations impossibles sont notées par un signe moins dans les Tableaux). Lors d'une série de A simulations, on note que les temps d'exécution sont approximativement divisés par deux lorsque le nombre de processeurs double. Jusque 256 processeurs, les simulations réellement coûteuses ont une très bonne efficacité parallèle ; on conclut que l'implémentation est extensible. Cependant on peut se demander pourquoi l'efficacité semble moindre lorsque $A = 41$ et $sp = 16$. Supposons que le **t**emps d'**e**xécution d'une **r**éplication varie peu autour de la valeur ter. La complexité séquentielle est alors $A \times R \times ter$. La distribution cyclique du troisième niveau donne un coût parallèle de $P\times\lceil A/sp\rceil\times(R\times ter/m)$. On déduit alors que l'efficacité théorique est bornée par $A/(sp\times\lceil A/sp\rceil)$. Lorsque $A=41$, $sp=16$, l'efficacité parallèle doit ainsi être inférieure à 85 % parce que les groupes de processeurs prennent en charge un nombre de simulations différent. L'hypothèse que les temps d'exécutions des simulations sont identiques est approximative mais explique ainsi la baisse des performances lorsque $A=41$, $sp=16$. En général cependant, le comportement des deuxième et troisième niveaux de parallélisme est très satisfaisant. Une analyse de sensibilité coûteuse ($A = 41$) ne demande que 12 minutes sur 256 processeurs. Lorsque les temps d'exécution des réplications et celles des simulations varient peu, les parallélismes de deuxième et troisième niveaux ont la même qualité et procurent une bonne scalabilité.

7.2.5 Indéterminisme lié à la génération aléatoire



Selon la graine que l'on choisit pour initialiser le générateur aléatoire du simulateur stochastique, la moyenne et la variance finales changent. Mais elles varient d'autant moins que le nombre R de réplications augmente. Le théorème de la limite centrale appuie cette remarque : la loi de la moyenne M de R variables indépendantes V_r de même loi (de moyenne $\overline{m}$ et de variance σ^2) converge en loi vers la loi Normale $\mathcal{N}(\overline{m}, \frac{\sigma}{\sqrt{R}})$. On identifie V_r à l'une des sorties du simulateur au temps $t = 366$ jours pour la réplication r, par exemple le nombre de parasites. D'après le théorème, l'écart-type de la loi de la moyenne diminue en $R^{-\frac{1}{2}}$. On constate cette relation en pratique sur les figures précédentes : lorsque R est multiplié par 4 ($R = 64 \rightarrow R = 256$), l'écart-type de la moyenne estimée est approximativement divisé par deux. La simulation dont sont issues les courbes ci-dessus correspond à un cas pour lequel la variance des résultats finaux est élevée. On déduit que dans le cas général, on aura intérêt à fixer R à plus de 256 si l'on souhaite une précision correcte sur les sorties.

7.2.6 Conclusion

Les performances en temps du simulateur stochastique devancent celles du simulateur déterministe dans les situations qui motivent cette étude, en utilisant une autre méthode de calcul. Du point de vue qualitatif, la méthode stochastique fournit en supplément des spectres des sorties du simulateur. La programmation hybride permet de paralléliser finement l'application stochastique de telle façon qu'une simulation s'exécute en moins d'une minute sur 128 processeurs d'un IBM SP3 NH2. Les trois niveaux de parallélisme permettent d'employer potentiellement un très grand nombre de processeurs. Ces résultats probants permettent d'utiliser ces simulations dont le coût en temps est faible pour réaliser une analyse de sensibilité du modèle. Notamment, une analyse de sensibilité de 41 simulations demande moins de 12 minutes sur 256 processeurs ($R = 32$ réplications).

Partie IV

Etude qualitative du modèle bar-Diplectanum aequans

Chapitre 8

Interprétation des résultats des simulations déterministes

Les principaux comportements qualitatifs des dynamiques du modèle déterministe sont exposés dans ce chapitre. On verra aussi quels rôles jouent les paramètres de mortalité des parasites μ (voir Section II-4.4.2 p. 45) et le seuil du recrutement sur-dispersé *palier* (voir Section II-4.3.3 p. 38). Des études concernant l'influence des autres paramètres d'entrée seront présentées dans les Chapitres suivants.

8.1 Spectre des dynamiques

Les cinq types de simulations décrites dans cette Section caractérisent les sorties du simulateur. Nous définissons cinq catégories de dynamiques des populations sur des critères qui sont explicités ci-après. Lorsque l'on analysera les sorties de nombreuses simulations, cela nous permettra de classer et comparer les simulations entre elles. Tous ces types correspondent à des dynamiques qui peuvent être observées sur le terrain.

Pour obtenir les cinq types de simulations, on a pris un unique jeu de paramètres d'entrée présenté ci-après pour lequel on a juste modifié le paramètre ρ. Selon la valeur de ce taux de saturation ρ du facteur de transmission (*cf* p. 37), les larves de parasites présentes dans le milieu vont se fixer plus ou moins efficacement sur les hôtes. Ce paramètre est une inconnue majeure qui est lié à toutes les interactions entre les larves nageantes et les hôtes, et qui dépend de facteurs abiotiques (configuration spatiale, qualité du milieu, *etc*). Il est donc légitime de faire varier ρ sur un intervalle assez large (ici de 0.3 à 0.6). Ces valeurs correspondent à des situations plus ou moins favorables à la colonisation des hôtes par les larves.

8.1.1 Paramètres d'entrée

Les paramètres du modèle utilisés dans les simulations sont maintenant donnés. Ils reprennent les observations de terrain [62, 65–67], puis les données utilisées dans la partie

II et les articles [12, 13]. Ils constituent le jeu de paramètres de référence qui sera utilisé dans l'ensemble de la partie IV (sauf mention contraire).

Paramètres scalaires

- Effectif initial d'hôtes (correspond à $N(0,0)$) : 5000,
- *critiquecritique*) (intervient dans la survie de l'hôte $p(.)$, voir p. 49) : 770,
- *lethal* (nombre de parasites létal pour l'hôte, voir p. 49) : 800,
- C_0 (paramètre de la fonction de transmission T, voir p. 37) : 900,
- n (paramètre de la fonction de transmission T, voir p. 37) : 2,
- ρ (taux de saturation de la fonction de transmission T, voir p. 37) : $\rho \in [0, 1]$,
- *palier* (seuil du recrutement agrégé, intervient dans $f(.,t)$, voir p. 41) : 300,
- *nf* (degré du polynôme de $f(.,t)$, voir p. 41) : 3,
- $(x_i; a_0)$ (point d'inflexion de la fonction F, voir p. 41) :=(0.05 ; 0.95),
- Date de début de la simulation : 1$^{\text{er}}$ juin,
- Durée en jours : 366, $\Delta t = 2$ jours.

Paramètres dépendant du mois

mois	ja.	fé.	ma.	av.	ma.	ju.	ju.	ao.	se.	oc.	no.	dé.
température	7.0	8.0	9.9	13.0	17.0	21.0	22.5	23.0	22.0	16.0	10.0	7.5
apport extérieur de larves par Δt	0	0	0	0	0	80	80	80	80	80	80	0

Paramètres dépendant de la température

Temp. °C	7	8	9	10	11	12	13	14	15
Mortalité $\mu(\theta)$ des parasites en % sur Δt	5	5	5	5	5	5	5	5	5
Temps d'incubation en Δt	9	9	9	9	7	6	6	5	5
Fécondité en Δt	4	4	4	4	4	4	4	4	4
Mortalité des œufs en % sur Δt	1	1	1	1	1.7	2	2	2.5	2.5

Temp. °C	16	17	18	19	20	21	22	23	24
Mortalité $\mu(\theta)$ des parasites en % sur Δt	5	5	5	5	5	5	5	5	5
Temps d'incubation en Δt	3	3	3	3	2	2	1	1	1
Fécondité en Δt	6	6	6	6	6	6	6	6	6
Mortalité des œufs en % sur Δt	4	4	4	4	5	5	10	10	10

Les paramètres étant les plus difficiles à estimer sont les suivants :

1. le taux de mortalité $\mu(\theta)$ des parasites,
2. l'apport extérieur de larves,
3. le taux de saturation ρ de T,
4. le *palier* de recrutement,
5. le degré nf du polynôme de f,
6. le point d'inflexion de F : $(x_i; a_0)$.

Notons que les deux premiers paramètres ne sont pas mesurables *in situ* avec les moyens actuels. Les quatres derniers paramètres servent à modéliser le recrutement des larves par les hôtes et sont inconnus. On analyse dans cette Partie quel est l'impact de la variation de ces paramètres sur les sorties de la simulation. Nous avons identifié 5 types de simulation qui caractérisent les dynamiques possibles du système.

8.1.2 Simulation de type I

La Figure 59 illustre une extinction progressive de la population parasitaire pour un effectif d'hôtes constant égal à 5000 individus. Le flux entrant de larves dans le système via l'apport extérieur ne suffit pas à contrebalancer la mortalité naturelle des parasites fixés. Dans ces conditions la population parasitaire ne subsiste pas. Cette situation prévaut quand moins de 30 % des larves disponibles se fixent sur les hôtes (ρ=0.30).

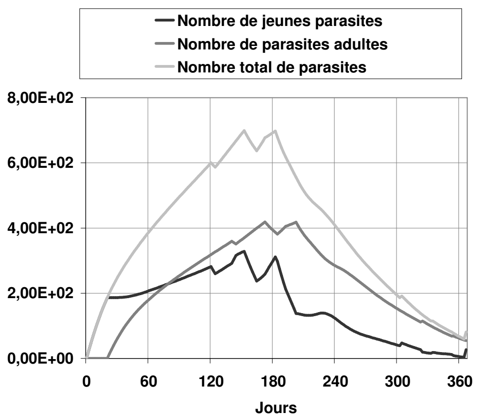


Figure 59. Evolution du nombre de parasites pour $\rho = 0.30$

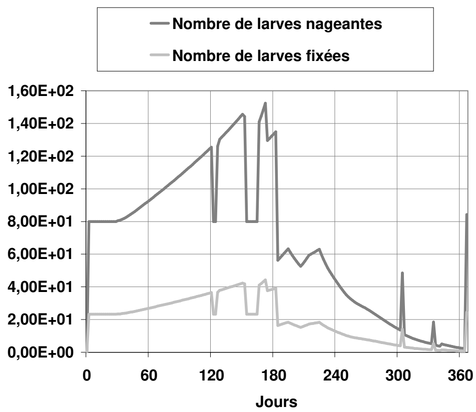


Figure 60. Evolution du nombre de larves pour $\rho = 0.30$

Cela se traduit par une différence d'effectifs entre larves nageantes et larves fixées (voir la Figure 60). On notera les décalages temporels entre les cohortes de jeunes non reproducteurs et celles des adultes. Ces effets retard sont notamment visibles lorsque l'effectif des jeunes parasites présente un maximum local ($t = 180$ par exemple). Le type I est défini comme une simulation pour laquelle l'effectif d'hôtes est stable et la population parasitaire tend à s'éteindre à la fin de l'année.

8.1.3 Simulation de type II

Un accroissement du facteur de transmission de 5 % suffit à éviter l'extinction (Figure 61), créant une situation de faible endémie[34].

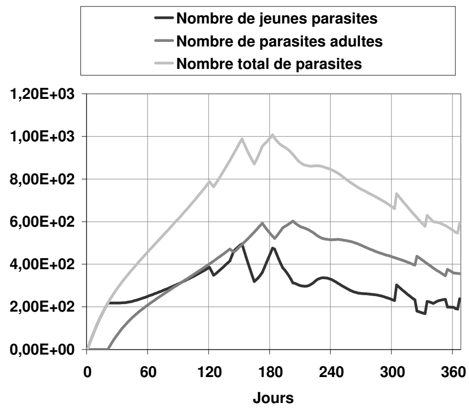


Figure 61. Evolution du nombre de parasites pour $\rho = 0.35$

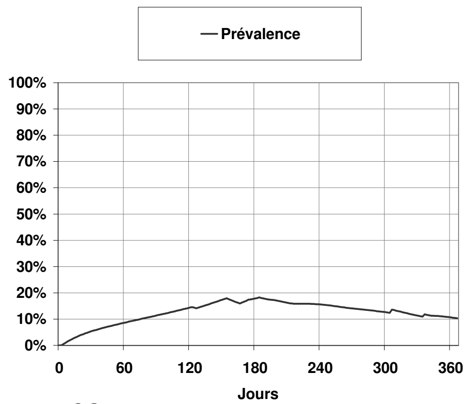


Figure 62. Evolution de la prévalence pour $\rho = 0.35$

Les adultes reproducteurs parviennent à entretenir la population, sans accroissement ou diminution spectaculaire. Certains changements mineurs de trajectoires sont dus à l'influence de facteurs environnementaux (notamment la variation de la durée d'incubation liée à la température). On suit sur la Figure 62 la prévalence[35] ; elle ne dépasse pas les 20 %, la population hôte restant stable à 5000 individus. Par définition, le type II caractérise une simulation dont le nombre d'hôtes est égal à 5000 toute l'année et dont le nombre de parasites se stabilise au bout des 366 jours de simulation.

8.1.4 Simulation de type III

A partir de certains seuils, une population de macroparasites passe obligatoirement dans une phase d'accroissement qui est naturellement fonction des différentes variables biologiques et celles liées au milieu. Les parasites géniteurs sont alors assez nombreux et les autres freins démographiques ne sont pas suffisants pour stabiliser la population parasitaire (Figure 63).

Il n'y a pas de régulation intensité-dépendante des hôtes ; cette situation est souvent transitoire et locale dans les populations naturelles. Le nombre de parasites jeunes est alors plus important que le nombre d'adultes. En effet, la population parasitaire est dans une phase de croissance, et il existe un délai de maturation de jeune en adulte. La prévalence ne cesse de croître durant l'année, pour rejoindre finalement un taux d'hôtes infestés égal à 100 % (Figure 64).

34. Se dit d'un phénomène de santé présent en permanence, ou épisodiquement dans l'année.
35. La prévalence désigne le pourcentage d'hôtes portant au moins 1 parasite.

Une simulation de type III se caractérise par un nombre d'hôtes constant sur les 366 jours et une population parasitaire qui globalement s'accroît durant toute la simulation.

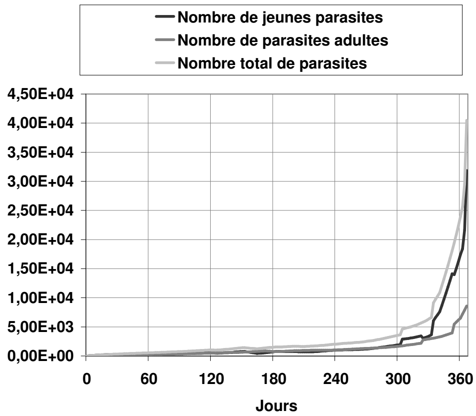


Figure 63. Evolution du nombre de parasites pour $\rho = 0.40$

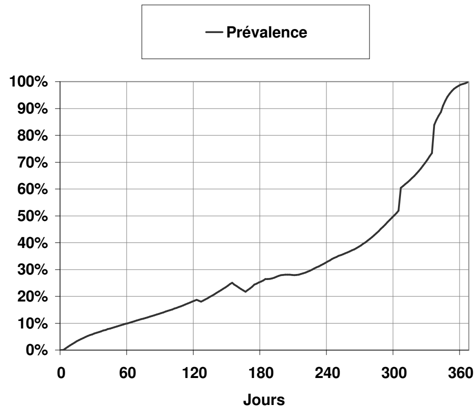


Figure 64. Evolution de la prévalence pour $\rho = 0.40$

8.1.5 Simulation de type IV

Dans un premier temps, et tant qu'il n'y pas d'hôtes ayant plus de *palier* parasites, les larves se fixent sur les hôtes par un processus aléatoire (Figure 65, ρ=0.55). Après une certaine période, tous les hôtes sont infestés (la prévalence atteint 100 %). Les hôtes ayant plus de *palier* parasites recrutent d'autant plus qu'il ont déjà un nombre élevé de parasites. Les jeunes parasites colonisent alors prioritairement les hôtes très parasités ce qui amorce le phénomène de surdispersion. Lorsque la population parasitaire devient vraiment importante, le seuil létal de parasites est atteint par certains hôtes (Figure 66).

La mort des hôtes et des parasites qu'ils portent occasionne une régulation du parasitisme. La population parasitaire globale tend à se stabiliser au commencement de l'épizootie (entre $t = 280$ et $t = 330$), ce qui n'est pas le cas des cohortes. Le nombre de jeunes diminue alors que le nombre d'adultes continue d'augmenter. La diminution des premiers s'explique : 1) par l'arrivée de grandes quantités de larves infestantes accompagnée d'une surdispersion importante dans le recrutement, puis 2) par la disparition des jeunes parasites entraînée par la mort des hôtes les plus parasités. L'accroissement du nombre d'adultes tient au vieillissement naturel des larves déjà fixées, jusqu'à ce que la mortalité des adultes ne soit plus compensée par l'arrivée de nouveaux jeunes. Au moment où ils se sont fixés, les parasites adultes ont été recrutés par un processus Poissonnien. Par contre les jeunes qui se sont fixés récemment sont agrégés. On explique ainsi pourquoi l'effectif des jeunes décroît fortement aux premières morts d'hôtes, contrairement à celui des parasites adultes. La distinction que réalise le modèle entre parasites jeunes et adultes prend donc ici tout son sens, car les dynamiques propres des cohortes induit une évolution macroscopique particulière. Un peu moins de 50 % des hôtes ont disparu à termes (Figure

66). Le type IV correspond à des simulations pour lesquelles le nombre d'hôtes diminue mais reste au-dessus d'un certain seuil. Arbitrairement, on fixe ce dernier à 2000.

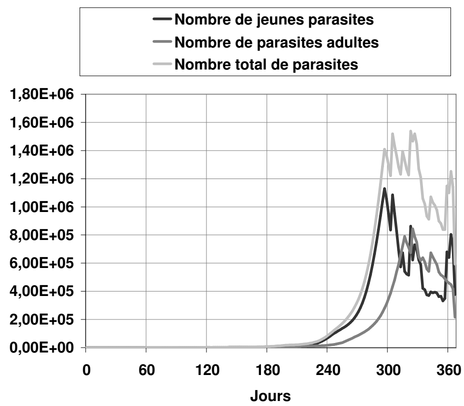


Figure 65. Evolution du nombre de parasites pour $\rho = 0.55$

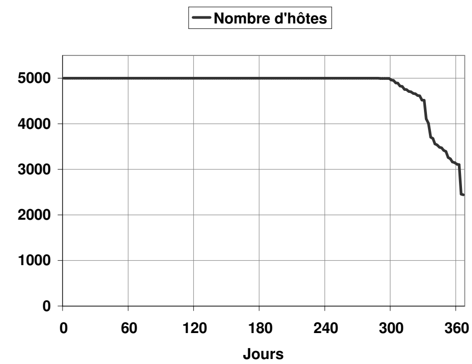


Figure 66. Evolution du nombre d'hôtes pour $\rho = 0.55$

8.1.6 Simulation de type V

Considérons maintenant un facteur de transmission encore plus élévé $\rho = 60$ % (Figures 67 et 68). Les premières étapes sont similaires, mais les adultes présents produisent tellement de stades infestants que la population parasitaire s'accroît à plusieurs reprises, conduisant à une mortalité des hôtes désormais massive.

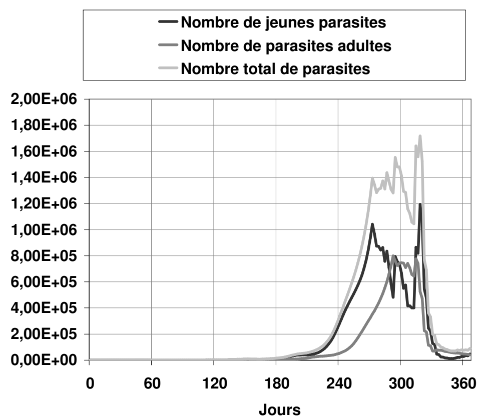


Figure 67. Evolution du nombre de parasites pour $\rho = 0.60$

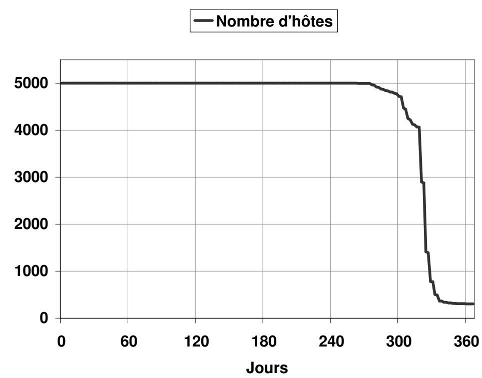


Figure 68. Evolution du nombre d'hôtes pour $\rho = 0.60$

On assiste alors à un effondrement des deux populations couplées, puis à une nouvelle stabilisation inévitable. En effet, le facteur de transmission étant fonction de la densité des hôtes (*cf.* p. 37), il baisse notablement avec des mortalités importantes d'hôtes ; il freine de ce fait le recrutement et conduit à un état d'endémie provisoire. La situation est alors nouvelle : le nombre de larves est encore élevé, mais peu trouvent un hôte. De telles vagues épizootiques sont observées en situation aquicole, quand le confinement des hôtes est trop élevé. Le parasitisme a pour effet de décaler l'équilibre du système vers une densité moindre d'hôtes. On pose que le type V correspond aux simulations dont le nombre final d'hôte est inférieur à 2000, pour lesquelles la population hôte est finalement décimée.

8.1.7 Conclusion

Avec la version parallèle du simulateur déterministe, les calculs sont précis, et des approximations ont été enlevées par rapport au simulateur séquentiel. Des temps d'exécution courts permettent désormais d'effectuer de nombreuses simulations. Avec le simulateur séquentiel, seulement deux comportements dynamiques étaient observés : premièrement, l'extinction des hôtes après une épidémie ; deuxièmement la disparition des parasites sans aucune mort d'hôte. Des comportements dynamiques nouveaux, correspondant à des cas observés *in situ*, ont été mis en évidence avec le simulateur parallèle. A titre d'exemple, une régulation provisoire de l'épizootie par la mortalité des hôtes a été observée dans certaines simulations. Sur de nouveaux jeux de paramètres, il a été possible d'obtenir au total 5 types de dynamique distincts qui correspondent à des situations réelles.

8.2 Etude sur μ, le taux de mortalité des parasites

Nous allons analyser des résultats de simulations pour lesquels les paramètres d'entrée μ et ρ varient. Ceci constitue une étude de la réponse du système hôte-macroparasite aux variations possibles des paramètres d'entrée. Sur certaines des Figures qui suivent, les simulations qui correspondent au jeu de paramètres de référence sont désignées par une flèche. Le paramètre ρ n'étant pas fixé, la flèche pourra pointer sur une série de simulations pour laquelle ρ varie.

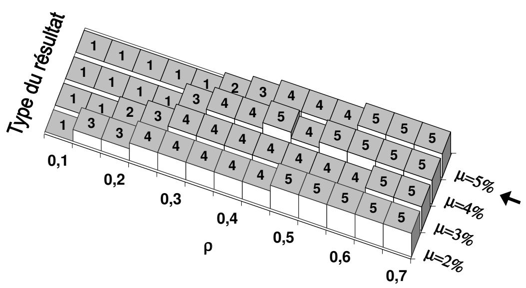


Figure 69. Type de la simulation en fonction de ρ et μ

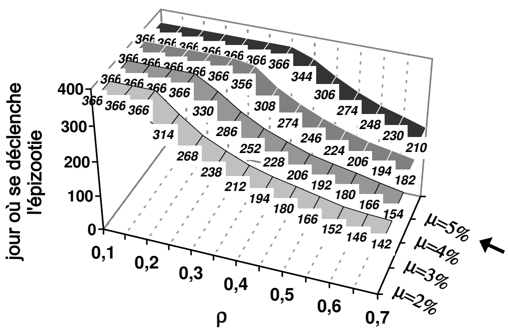


Figure 70. Jour où commence l'épizootie en fonction de ρ et μ ; la valeur $t = 366$ jours apparaît si l'épizootie ne se déclare pas

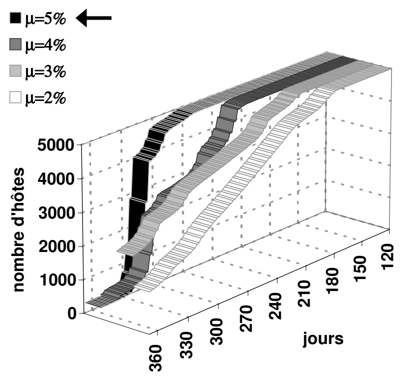


Figure 71. Evolution du nombre d'hôtes pour $\rho = 0.60$ et μ variable

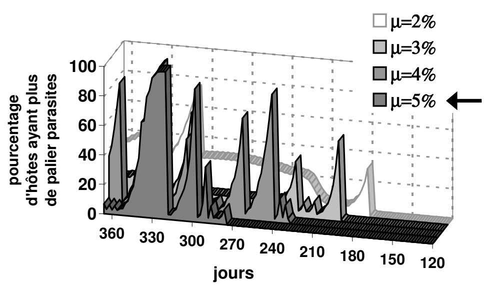


Figure 72. Evolution du pourcentage d'hôtes ayant plus de *palier* parasites pour $\rho = 0.60$ et μ variable

Sur la Figure 69, on a un aperçu de tous les types de résultats obtenus par simulation pour $\rho \in [0.1..0.7]$ et $\mu \in [0.1..0.7]$. Lorsque μ augmente[36], on constate que les simulations de type I ont lieu jusqu'à une valeur de ρ de plus en plus grande. L'interprétation est la suivante : un taux de transmission ρ élevé ou une mortalité faible des parasites favorise de deux manières différentes le développement de la population parasitaire. La Figure 70

36. On peut voir p. 46 Figure 16 à quoi correspondent de tels taux de mortalité μ.

indique le jour (compté à partir du premier juin) auquel le nombre d'hôtes commence à baisser. On note deux choses : plus la transmission des larves est favorisée, plus l'épizootie commence tôt. D'autre part un taux de mortalité élevé retarde le début de l'épizootie.

Puis, on peut s'interroger sur les différences qu'il peut y avoir entre des épizooties associées à des taux de mortalité distincts. Pour cela, on regarde spécifiquement le cas où $\rho = 0.60$ qui présente une décroissance du nombre d'hôtes relativement précoce dans l'année. La Figure 71 montre les courbes du nombre d'hôtes. On constate que pour $\mu =$ 2 % ou 3 % la décroissance est continue, alors pour $\mu =$ 4 % ou 5 % elle présente des non-linéarités de plus en plus importantes. Ce comportement non-linéaire se retrouve au niveau du pourcentage d'hôtes ayant plus de *palier* parasites (variable $x(t)$ pour la fonction $F(x(t))$, *cf* p. 38) sur la Figure 72. Voici une interprétation : lorsque le taux de mortalité est faible, les hôtes du groupe[37] $\dot{\mathrm{I}}$ vont en moyenne perdre peu de parasites pendant un pas de temps et rester dans la classe $\dot{\mathrm{I}}$. Par contre si le taux de mortalité est plus grand, certains hôtes sortent de $\dot{\mathrm{I}}$ et rejoignent $\dot{\mathrm{P}}$. En conséquence, lorsque μ est élevé, il arrive plus fréquemment qu'il y ait peu d'hôtes dans $\dot{\mathrm{I}}$. Dans ce cas les parasites sont peu surdispersés, les larves nageantes ont tendance à se fixer à la fois sur les quelques hôtes très parasités mais aussi sur ceux qui le sont moins. La parasitisme affecte donc l'ensemble des hôtes et non spécifiquement ceux qui portent beaucoup de parasites. Le modèle provoque dans ce cas une alternance entre les situations où les parasites sont agrégés sur les hôtes et des périodes où ils le sont moins. C'est ce phénomène qui provoque les morts massives d'hôtes sur la Figure 71 et les pics sur la Figure 72.

8.3 Etude sur *palier*, seuil de la surdispersion

Le seuil *palier* permet d'indiquer à partir de quelle charge parasitaire les hôtes commencent à devenir les cibles de l'agrégation parasitaire ; il permet de distinguer les hôtes faisant partie des groupes $\dot{\mathrm{P}}$ et $\dot{\mathrm{I}}$.

Tout d'abord, ce paramètre n'intervient que lorsque certains hôtes ont plus de *palier* parasites. Il est donc normal sur la Figure 73 de ne remarquer aucune différence à ρ constant pour différentes valeurs de *palier* ; en fait les résultats sont identiques pendant le début des simulations tant qu'il n'y pas d'hôtes dans le groupe $\dot{\mathrm{I}}$. Lorsque l'épizootie se déclare (à partir de $\rho = 0.45$), les plus grandes valeurs de *palier* (300 et 400) conduisent à des morts d'hôtes plus importantes correspondant à des simulations de type 5.

Les deux graphiques 75 et 76 présentent le cas où $\rho = 0.6$ et *palier* prend plusieurs valeurs. En analysant les sorties de ces simulations, on remarque que l'abondance[38] oscille autour du *palier* dès les premières morts d'hôtes. De façon étonnante, le modèle est très contraint par ce paramètre *palier*. On note au niveau du nombre d'hôtes que l'épizootie

37. Rappelons que $\dot{\mathrm{I}}$ est le groupe des hôtes ayant plus de *palier* parasites, et que $\dot{\mathrm{P}}$ est le groupe des hôtes ayant moins de *palier* parasites.

38. L'abondance est le nombre moyen de parasites fixés par hôte.

est d'autant plus sévère que *palier* est grand.

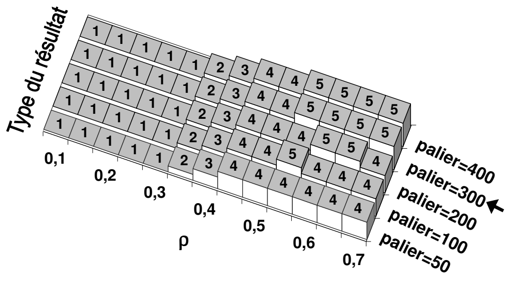


Figure 73. Type de la simulation en fonction de ρ et *palier*

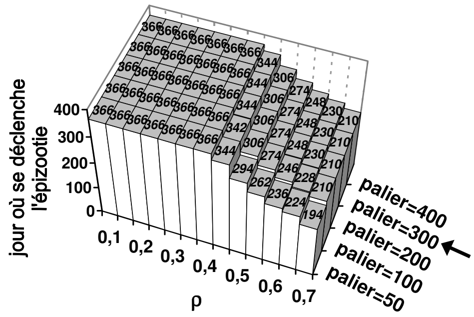


Figure 74. Jour où commence l'épizootie en fonction de ρ et *palier* ; la valeur $t = 366$ jours apparaît si l'épizootie ne se déclare pas

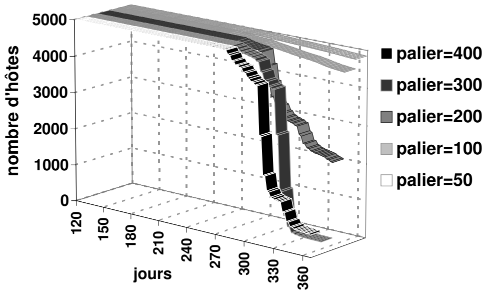


Figure 75. Evolution du nombre d'hôtes pour $\rho = 0.60$ et *palier* variable

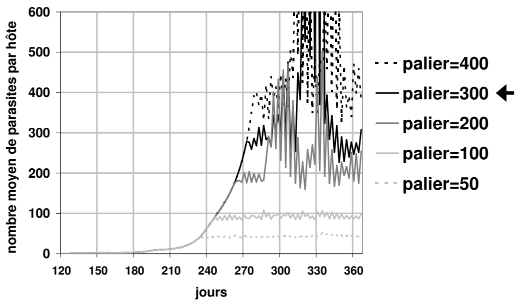


Figure 76. Evolution de l'abondance pour $\rho = 0.60$ et *palier* variable

Il est très surprenant que l'abondance oscille autant sur la Figure 76. De plus, la période des oscillations est de deux pas de temps. Nous allons voir dans le prochain Chapitre quelle est l'origine de ce phénomène, puis le supprimer.

Chapitre 9

Evolution du modèle déterministe

Certains phénomènes au niveau des sorties sont singuliers et demandent à être étudiés en détail. En considérant des simulations particulières, nous allons mettre en évidence l'origine des situations artificielles. La détection d'artéfacts numériques et de dynamiques non conformes à l'esprit du modèle conduisent à remanier les fonctions f et F. Par l'analyse des résultats de la simulation, il est ainsi possible de revenir sur le modèle pour remanier des composants qui se révèlent pas assez finement modélisés *a posteriori*.

9.1 Transformation de la fonction f

9.1.1 Description préliminaire

Avant de regarder certaines simulations précisément et de proposer des modifications concernant le modèle, nous effectuons quelques rappels sur la fonction f. La fonction f désigne l'espérance de larves recrutées par un hôte ayant l parasites ; elle est définie par :

$$f(l,t) = \begin{cases} f_0(t) & \text{si } 0 \leq l \leq palier \\ f_0(t) + \lambda(t) f_1(l - palier) & \text{si } palier < l \leq p_{max}(t) \ . \end{cases} \tag{39}$$

La fonction f_1 utilisée est de type $f_1(l) = l^{nf}$, avec nf un des paramètres de la simulation. On obtient dans ces conditions la courbe de la Figure 77. Les valeurs de $f_0(t)$ et $\lambda(t)$ sont calculées à partir des formules (*cf* p. 42)

$$f_0(t) = (1 - F(R_p(t)))/\sum_{l=0}^{palier} N(l,t) \quad , \tag{40}$$

$$\lambda(t) = (1 - f_0(t)\, H(t))/\sum_{l=palier+1}^{pmax(t)} f_1(l - palier)\, N(l,t) \quad . \tag{41}$$

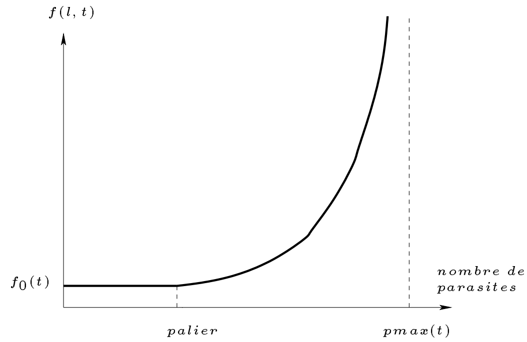


Figure 77. $f(l,t)$: espérance de larves recrutées à l'instant t par un hôte ayant l parasites pour $f_1(l) = l^2$.

La variable $p_{max}(t)$ représente la charge parasitaire maximale au delà de laquelle la probabilité qu'un hôte porte autant de parasites est quasi nulle. La queue de distribution des parasites sur les hôtes $N(l,t)$ (lorsque l est grand) joue un rôle important dans l'évaluation des paramètres f_0 et λ. Regardons le dénominateur de l'équation (41), pour l assez grand : $N(l,t)$ décroît alors que $f_1(l - palier)$ croît de manière très significative. De ce fait, la queue de distribution de $N(l,t)$ (lorsque l est grand) a parfois une importance significative dans le calcul de $\lambda(t)$ à cause de la multiplication avec des quantités importantes. Or on peut se demander si cela correspond à un phénomène biologique. En effet $f(l,t)$ ne vise pas cela, elle a pour objectif d'encourager le recrutement sur les hôtes disposant de nombreux parasites. En pratique, la quantité $\lambda(t)$ dépend fortement de la borne $p_{max}(t)$ dans certaines simulations. Lorsque $p_{max}(t)$ est grand, $\lambda(t)$ est abaissé et vice-versa. Ceci entraîne des artéfacts de calcul conduisant à un phénomène de "pompage" d'un pas de temps à l'autre. Ce mécanisme est illustré dans la Figure 78 issues d'une simulation ($t \in [228, 260]$).

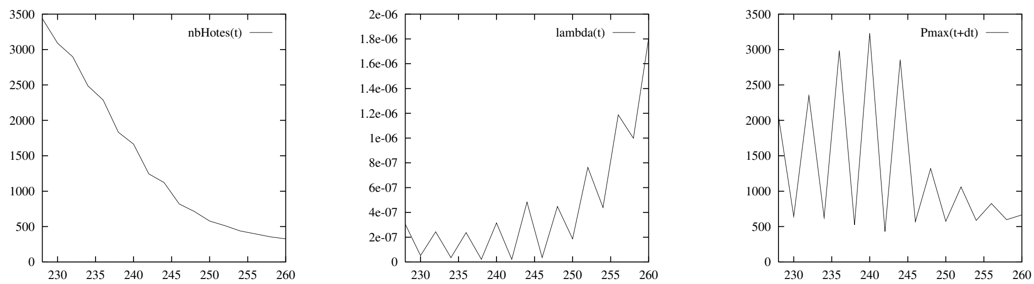


Figure 78. Oscillations dues à un artéfact numérique et constatées au niveau du nombre d'hôtes $H(t)$, de $\lambda(t)$ et de $p_{max}(t)$ pour $t \in [228, 260]$

Schématiquement, au pas de temps $t - \Delta t$ la queue de distribution contient peu d'hôtes et $p_{max}(t - \Delta t)$ est faible. Au pas de temps suivant t, $\lambda(t)$ est grand entre autre parce que son dénominateur est lié à $p_{max}(t - \Delta t)$ petit. Ainsi, comme $\lambda(t)$ est élevé, des hôtes du groupe $\dot{\text{I}}$ passe dans le groupe $\dot{\text{P}}$ et la queue de distribution n'est plus vide, et donc $p_{max}(t)$ est grand. Au pas de temps $t+\Delta t$, le dénominateur de $\lambda(t+\Delta t)$ est lié à $p_{max}(t)$ donc $\lambda(t+\Delta t)$ est petit. Il s'en suit un recrutement modeste sur les hôtes très parasités et donc $p_{max}(t+\Delta t)$ sera faible.

Pour éliminer cet artéfact, on change la fonction $f(l,t)$ sur la dernière partie lorsque l est grand. En effet, il n'y a aucune raison qui justifie que f doive tendre vers l'infini pour l grand et provoquer ainsi cette dépendance entre $\lambda(t)$ et $p_{max}(t)$.

9.1.2 Solution au phénomène de "pompage"

On construit une fonction $f_1(l)$ qui reste constante pour l supérieur à *lethal* ; on créé un seuil maximal de recrutement. Ainsi, on signifie qu'au-delà du nombre de parasites létal pour l'hôte, le recrutement n'est plus intensité-dépendant.

$$f_1(l) = \begin{cases} l^{nf}\left(\left(1+nf^{-1}\right)(lethal - palier) - l\right) & \text{si } 0 \leq l \leq lethal - palier \\ (lethal - palier)^{nf+1}/nf & \text{si } lethal - palier \ < l \ . \end{cases} \tag{42}$$

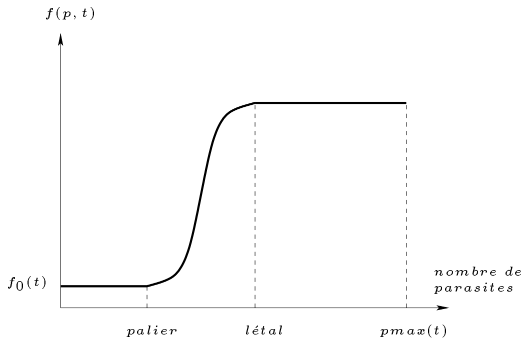


Figure 79. $f(l,t)$: espérance de larves recrutées à l'instant t par un hôte ayant l parasites, $f_1(l) = (lethal - palier)\,\frac{3\,l^2}{2} - l^3$.

Les calculs de $f_0(t)$ et $\lambda(t)$ restent inchangées. Sur la Figure 79, on présente la courbe $f(l,t)$ pour $nf = 2$, soit $f_1(l) = (lethal - palier)\,\frac{3\,l^2}{2} - l^3$. Avec cette nouvelle fonction f_1, le dénominateur de $\lambda(t)$ est en général moins lié à $p_{max}(t)$. Cela permet de stopper les oscillations artificielles comme on le voit sur les Figures 80 et 81. On constate un lissage des courbes lorsque l'on échange l'ancienne fonction f_1 contre la nouvelle. Sur la courbe de la densité d'hôtes (à gauche), on remarque que le crénelage de la courbe disparaît avec la nouvelle fonction f_1. Au niveau de l'abondance (courbe de droite), il est très net que le pompage cesse. Par contre, les allures des courbes restent semblables et c'est aussi le cas sur l'ensemble des simulations que nous avons pu tester. Ainsi, on peut raisonnablement penser que ce changement de fonction n'altère pas le type des dynamiques et supprime uniquement l'artéfact numérique.

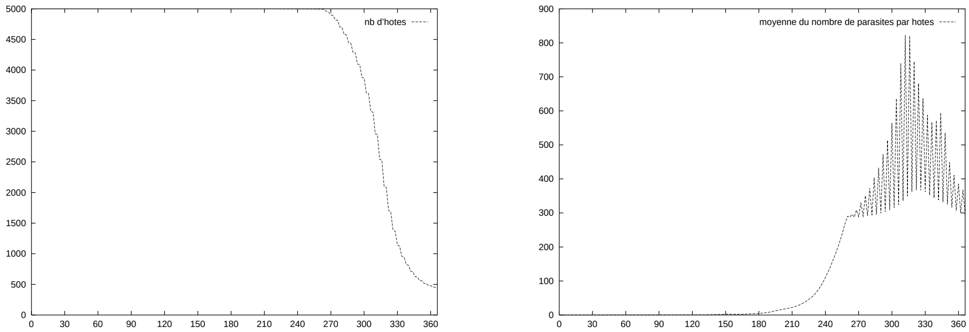


Figure 80. Nombre d'hôtes fonction du jour et abondance (avec l'ancienne fonction $f_1(l) = l^2$)

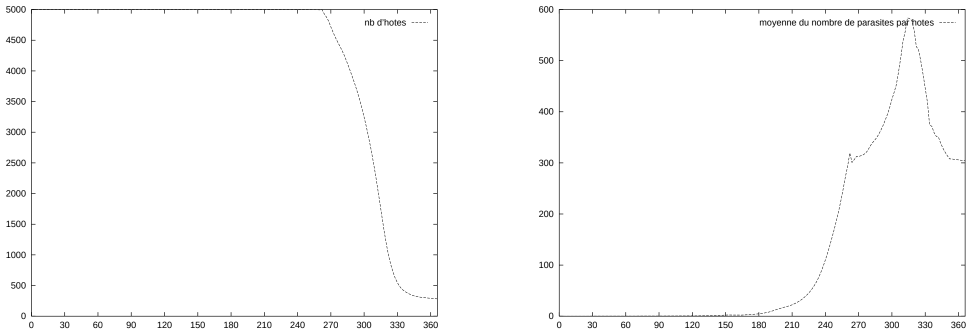


Figure 81. Nombre d'hôtes fonction du jour et abondance (avec la nouvelle fonction $f_1(l) = l^2\,(\frac{3}{2}(lethal-palier) - l))$

9.2 Modification de la fonction F

Dans notre modèle, on considère deux groupes distincts d'hôtes $\dot{\mathrm{I}}$ et $\dot{\mathrm{P}}$. Les hôtes de $\dot{\mathrm{I}}$ recrutent des larves selon une loi de Poisson qui ne dépend pas de l'intensité parasitaire[39]. Les larves se fixent sur les hôtes du groupe $\dot{\mathrm{P}}$ de façon intensité-dépendante. La valeur $x(t)$ représente la proportion d'hôtes qui se trouvent dans le groupe $\dot{\mathrm{I}}$. La fonction $F(x(t))$ détermine la proportion de larves qui se fixent sur les hôtes du groupe $\dot{\mathrm{I}}$. Cette fonction définie p. 38 possède un unique paramètre : son point d'inflexion (x_i, a_0).

La fonction F possède une dérivée seconde positive sur $[0, x_i]$, elle est donc convexe sur cet intervalle (voir F_{old} sur la Figure 82). Par contre sur $[x_i, 1]$, F a une dérivée seconde négative, elle y est donc concave. D'autre part, la courbe possède une dérivée à gauche très importante au point (x_i, a_0), et la tangente à la courbe a une pente très supérieure à 1. Par contre à droite de x_i, sur $[xi, 1]$, la pente baisse subitement et la dérivée première est rapidement au-dessous de 1.

La valeur de $x(t)$ se met souvent à osciller autour de x_i. On peut commenter les courbes de la Figure 83 issues d'une simulation en essayant de décrire le mécanisme à

39. L'*intensité* est définie comme le nombre d'individus d'une espèce parasite présente sur un hôte.

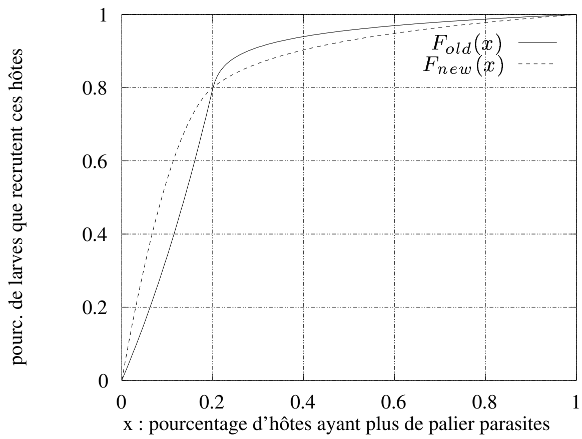


Figure 82. Deux versions de la fonction donnant le pourcentage de larves que recrutent les hôtes ayant plus de *palier* parasites : F_{old} et F_{new}

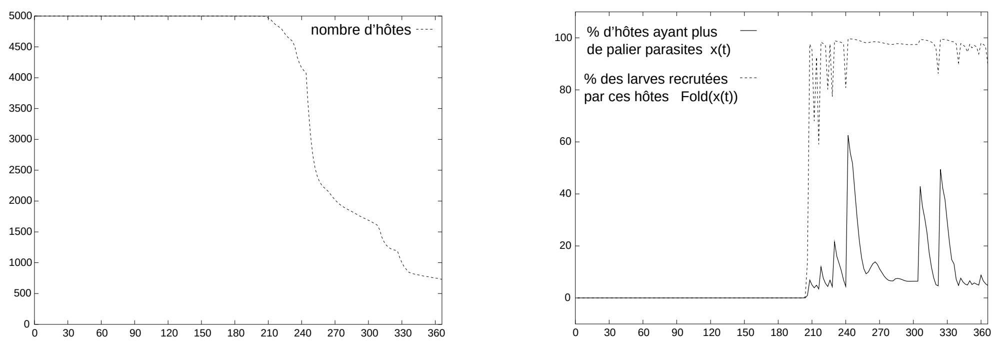


Figure 83. Nombre d'hôtes (à gauche), puis $x(t)$ et $F_{old}(x(t))$ présentées à droite (rappelons que le paramétrage de $F_{old}(x)$ est $x_i = 5$ %, $a_0 = 95$ %)

l'origine de l'oscillation. Au temps $t - \Delta t$, supposons que la variable $x(t-\Delta t)$ ait décru jusqu'à une valeur très proche de x_i, mais telle que $x(t-\Delta t) > x_i$. Au temps t, supposons que x décroisse encore : $x(t) - x(t-\Delta t) = -h < 0$, $x(t) > x_i$. Enfin en $t + \Delta t$, prenons $x(t+\Delta t) < x_i$ et supposons que $x(t+\Delta t) - x(t) = -h < 0$. Les valeurs de la dérivée première de F impliquent qu'entre $t - \Delta t$ et t, la valeur de $F(x)$ décroît faiblement et $F(x(t)) - F(x(t-\Delta t)) = -h\,a(t)$ avec $a(t)$ ayant une valeur peu élevée (par exemple proche de 1). Au pas de temps suivant, $x(t)$ et $x(t+\Delta t)$ encadrent x_i, on a $F(x(t+\Delta t)) - F(x(t)) = -h\,a(t+\Delta t)$ avec $a(t+\Delta t)$ maintenant très supérieure à 1 à cause de la dérivée à gauche de F en x_i. Ce décrochement et cette décroissance très rapide de la valeur de $F(x(t+\Delta t))$ provoque une agrégation moindre et un recrutement plus important sur les hôtes de $\dot{\mathrm{P}}$. Ceci implique que certains hôtes de $\dot{\mathrm{P}}$ passent dans le groupe d'hôtes $\dot{\mathrm{I}}$. Le nombre d'hôtes ayant plus de *palier* parasites augmente soudainement et $x(t+\Delta t)$ prend une valeur élevée. Il y a donc un phénomène de répulsion ; lorsque $x(t)$ diminue et qu'il descend au-dessous de x_i, ce pourcentage remonte brusquement. On observe ainsi sur le graphique que $x(t)$

effectue des remontées subites juste après avoir atteint x_i.

On doit se demander si ce phénomène reproduit un phénomène biologique ou bien s'il s'agit d'un artéfact. Comme le changement brutal de la dérivée de F n'a pas été nettement spécifié dans le modèle et qu'il constitue l'origine des oscillations, on conjecture la présence d'un artéfact numérique.

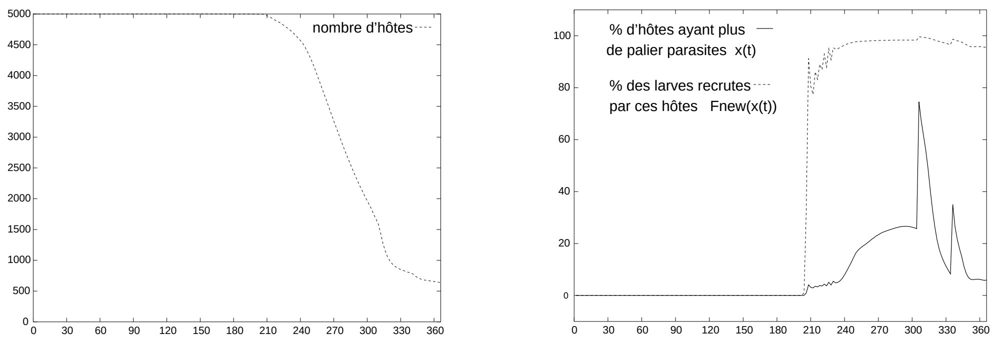


Figure 84. Nombre d'hôtes (à gauche), puis $x(t)$ et $F_{new}(x(t))$ présentées à droite (rappelons que le paramétrage de $F_{new}(x)$ est $x_i = 5$ %, $a_0 = 95$ %)

On construit une nouvelle fonction F_{new} qui ne présente pas de rupture importante de sa dérivée première. On impose une courbe concave à la fois sur $[0, x_i]$ et $[x_i, 1]$. Comme précédemment on souhaite pouvoir paramétrer un point $(x_i, F(x_i))$ de la courbe. Pour éviter les ennuis que l'on a rencontrés avec la fonction précédente, on fixe la dérivée à droite et à gauche en $x = x_i$ égale à 1. On pose $F(x)$ valant $GF_1(x)$ sur $[0, x_i]$ et $GF_2(x)$ sur $[xi, 1]$, avec :

$$GF_1(x) = Oa\,x^2 + Ob\,x$$
$$GF_2(x) = F(x_i) + (1/Om) * log(1 + Om * (x - x_i))$$

Les coefficients de ces deux fonctions sont calculés ainsi[40] $(F(x_i) > x_i)$:

$$Oa = -(F(x_i) - x_i)/{x_i}^2 \ ; \qquad Ob = 1 - 2\,Oa\,x_i \ ;$$
Om est solution de l'équation $F(x_i) + (1/Om)\,log(1 + Om\,(1 - x_i)) = 1$.

On a les propriétés suivantes :

$$GF_1(0) = 0, \quad GF_1(x_i) = F(x_i),$$
$$GF_1'(0) = 1 + 2\,(F(x_i) - x_i)/x_i, \quad GF_1'(x_i) = 1,$$
$$\forall x \in [0, x_i] \ : \ GF_1''(x) = -2\,(F(x_i) - x_i)/x_i^2 < 0,$$
(on a toujours $F(x_i) > x_i$ dans notre modèle.)

40. En pratique on utilise une dichotomie pour trouver Om.

$$GF_2(x_i) = F(x_i), \quad GF_2(1) = 1,$$
$$GF_2'(x_i) = 1, \quad GF_2'(1) = 1/(1 + Om\,(1 - x_i)),$$
$$\text{pour } x \in [x_i, 1] \ : GF_2''(x) = -Om\,x/(1 + Om\,(x - x_i))^2,$$
$$\text{comme } Om > 0 \text{ on a } GF_2''(x) < 0.$$

La Figure 84 présente les résultats que l'on obtient lorsque l'on reprend la simulation de la Figure 83 en modifiant la fonction F. Cette modification élimine un phénomène qui semble artificiel et que le modélisateur n'a pas voulu explicitement.

9.3 Compatibilité déterministe/stochastique

Pour comparer les simulateurs déterministe et stochastique, nous avons dû les modifier l'un et l'autre. Les modifications présentées dans cette partie ont permis d'harmoniser leurs caractéristiques afin que leur comparaison ait un sens.

La simulation stochastique considère des nombre entiers de larves nageantes, alors que la simulation déterministe utilise des flottants. La solution choisie est d'utiliser le type entier dans la simulation déterministe.

Le recrutement des larves sur les hôtes dans la simulation stochastique suit une loi multinomiale. On considère approximativement que les hôtes ayant q parasites recrutent selon une loi binomiale de paramètre $\mathcal{B}(L_r(t), f(q,t))$. Dans le modèle déterministe, la fonction $\varphi(j,q,t)$ donne la probabilité de recruter j larves pour un hôte ayant q parasites, elle a pour moyenne $f(q,t)\,L_r(t)$. On souhaite donc adopter une loi binomiale $\mathcal{B}(L_r(t), f(q,t))$ pour la fonction φ au lieu de la loi de Poisson (voir Section II-4.3.6 p. 43). Or la loi binomiale $\mathcal{B}(L_r(t), f(q,t))$ n'est pas définie lorsque $f(q,t) > 1$. On va voir que la fonction $f(l,t)$ peut être supérieure à 1 sur la fin de la queue de distribution N des parasites sur les hôtes ($l \in [0, p_{max}(t)]$). Supposons qu'il existe un entier z tel que $\forall l > z,\ f(l,t) > 1$ (rappelons que f et f_1 sont croissantes). On va considérer l'hypothèse suivante : $\sum_{l=z..\infty} N(l,t) \geq 1$, puis l'infirmer. La démonstration est rapide ; on sait que

$$\sum_{l=0}^{\infty} f(l,t)\,N(l,t) = 1$$

Or d'après l'hypothèse, on a

$$1 \leq \sum_{l=z}^{\infty} N(l,t) < \sum_{l=z}^{\infty} f(l,t)\,N(l,t)$$

Il y a donc contradiction. L'hypothèse de départ est donc fausse, d'où

$$\sum_{l=z..\infty} N(l,t) < 1 \qquad \square$$

Le problème d'une espérance incohérente $f(l,t) > 1$ ne se produit que sur la toute fin de la queue de distribution de N. Dans le modèle stochastique, il n'y pas de phénomène semblable pour une raison simple ; soit d le nombre d'hôtes qui ont le maximum de parasites parmi tous les hôtes au temps t (soit q ce nombre de parasites), on a $\sum_{l=q..\infty} N(l,t) = d$; or d est un nombre entier d'hôtes et il est supérieur à 1, il est donc impossible de trouver z tel que $\sum_{l=z..\infty} N(l,t) < 1$. La solution partielle nous permettant de régler ce problème dans le simulateur déterministe consiste à déterminer z à chaque pas de temps et considérer que $\forall l \geq z, \quad f(l,t) = f(z,t)$.

9.4 Interprétation et études de cas

Les modifications des fonctions effectuées sont intéressantes car sans invalider les idées qui ont prévalues dans la construction du modèle, elles suppriment des effets non désirables. Les adaptations nécessaires pour harmoniser les deux simulateurs déterministe et stochastique permettent dorénavant leur comparaison.

9.4.1 Etude sur μ, le taux de mortalité des parasites

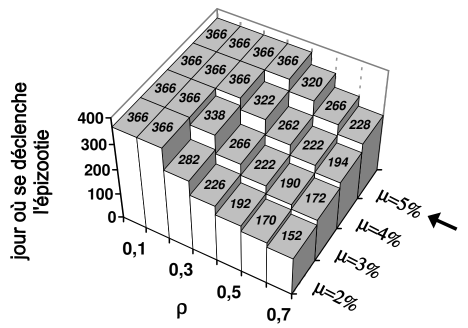


Figure 85. Jour où commence l'épizootie fonction de ρ et μ ; la valeur $t = 366$ jours apparaît si l'épizootie ne se déclare pas

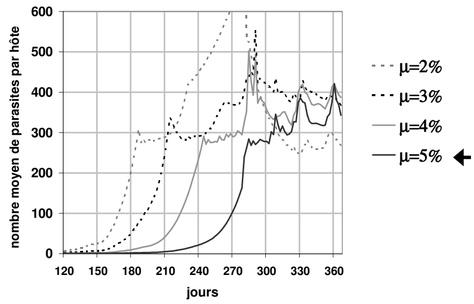


Figure 86. Abondance pour $\rho = 0.60$ et μ variable

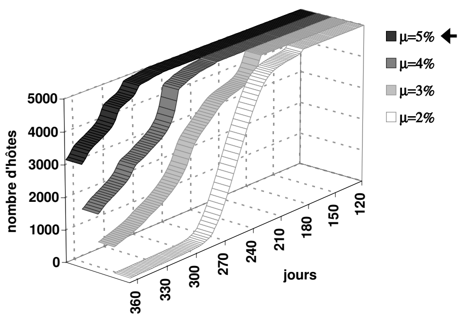


Figure 87. Evolution du nombre d'hôtes pour $\rho = 0.60$ et μ variable

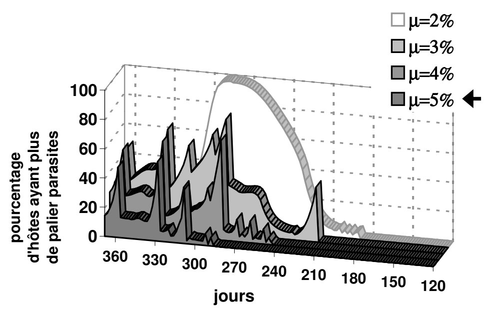


Figure 88. Evolution du pourcentage d'hôtes ayant plus de *palier* parasites pour $\rho = 0.60$ et μ variable

Nous allons analyser des résultats de simulations pour lesquels les paramètres d'entrée μ et ρ varient. Les Figures 85 à 88 illustrent l'influence des modifications apportées au modèle sur les simulations. Le jour où débute l'épizootie est tout à fait comparable à la Figure 70, même si elle est un peu retardée.

En effet, les changements apportés concernent le processus de recrutement surdispersif qui agit une fois que l'épizootie a commencé. Les types de simulation fonction de ρ et μ ne sont pas présentés, car ce graphique est quasi-identique à celui de la Figure 69. Les courbes du nombre d'hôtes (Figure 87) ont changé significativement, la réponse du système en fonction de μ est maintenant plus linéaire. Précédemment (Figure 72), la quantité $x(t)$ prenaient de très grandes valeurs et présentaient peu de sections continues ; ce n'est plus le cas Figure 88.
Lorsqu'il y a un pic de parasitisme ($x(t)$ augmente subitement), un taux de mortalité élevé combiné à l'agrégation permet de limiter les morts d'hôtes. En voici l'explication : 1) un groupe d'hôtes recrute la majorité des nombreuses larves disponibles en t, 2) des hôtes meurent dans ce groupe, d'autres restent très parasités et meurent durant les pas de temps qui suivent, 3) une mortalité élevée des parasites permet à certains hôtes du groupe de redescendre à une intensité parasitaire non léthale. Ceci entraîne un ralentissement des morts d'hôtes. On remarque d'ailleurs sur la Figure 88 que plus la mortalité μ est importante, plus $x(t)$ redescend rapidement après un pic.

9.4.2 Etude sur *palier*

Le début de l'épizootie est retardée lorsque l'on compare la Figure 74 correspondant aux anciennes fonctions f et F et la Figure 89 dans la nouvelle version du modèle. D'autre part, l'abondance (Figure 90) n'oscille plus en permanence avec les nouvelles fonctions f et F.

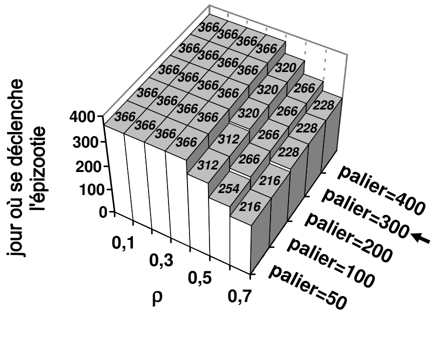


Figure 89. Jour où commence l'épizootie fonction de ρ et *palier* ; la valeur t = 366 jours apparaît si l'épizootie ne se déclare pas

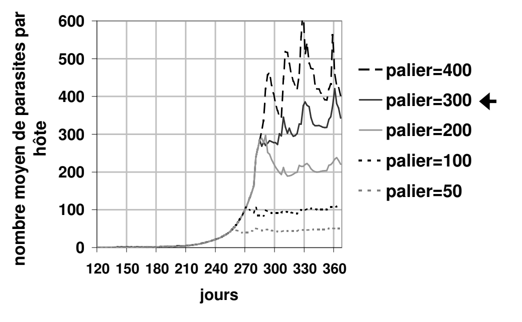


Figure 90. Abondance pour $\rho = 0.60$ et *palier* variable

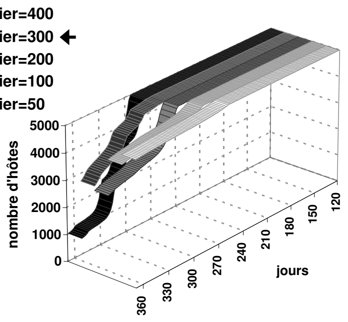


Figure 91. Evolution du nombre d'hôtes pour $\rho = 0.60$ et *palier* variable

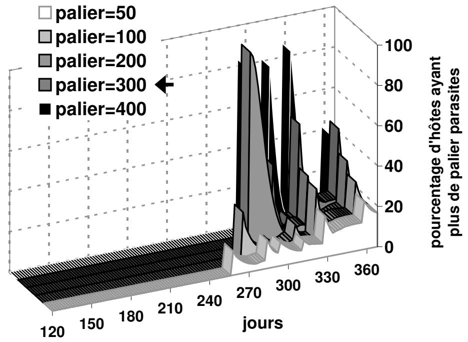


Figure 92. Evolution du pourcentage d'hôtes ayant plus de *palier* parasites pour $\rho = 0.60$ et *palier* variable

On note que l'abondance est de nouveau très proche du paramètre *palier*, et le modèle semble atteindre une stabilité lorsque l'abondance rejoint *palier* : 1) au dessous, la population parasitaire est suffisamment bien implantée pour croître, 2) au dessus de *palier*, les hôtes ont une mortalité élevée liée au parasitisme et en mourant, ils font disparaître des parasites dont la population totale diminue.
La Figure 92 montre que l'amplitude de $x(t)$ dépend de la valeur de *palier*. En fait, lorsque *palier* augmente le nombre de parasites jeunes et adultes fixés sur les hôtes s'accroît. Le nombre de larves nageantes est de fait plus élevé avec un *palier* haut, et les hôtes sont finalement d'autant plus colonisés par ces larves (Figure 91). Un *palier* faible favorise la survie des hôtes.

9.4.3 Etude sur *nf*

Le type de simulation et le jour où meurent les premiers hôtes en fonction de ρ et *nf* sont identiques à ceux de la précédente étude sur *palier* (lorsque *palier* = 300).

Le paramètre *nf* correspond au degré du polynôme utilisé pour $f_1(l)$. Plus ce paramètre est grand, plus le recrutement des larves est favorisé sur les hôtes déjà très parasités (*cf* p. 43). Le cas extrême lorsque *nf* est élevé est de considérer que les hôtes qui sont colonisés par un nombre proche de *letal* parasites recrutent pratiquement toutes les larves disponibles et puis meurent. Par exemple pour $nf = 8$, et après une décroissance forte du nombre d'hôtes, la population d'hôtes se stabilise (figure 93). La mortalité des hôtes devient alors faible. Dans ce cas, la population des parasites est régulée par la combinaison de l'agrégation avec la mortalité des hôtes fortement parasités. Dans un autre cas où la surdispersion est peu favorisée ($nf = 2$), beaucoup d'hôtes sont touchés par le parasitisme et la mortalité dans la population hôte est élevée. Le nombre de parasites total (Figure 94) dépend du paramètre *nf*, mais l'abondance varie très peu pour $nf = 3, 5$ ou 8. Le processus agrégatif bénéficie au parasitisme, il lui permet de renouveler la population parasitaire sans risquer l'extinction des hôtes.

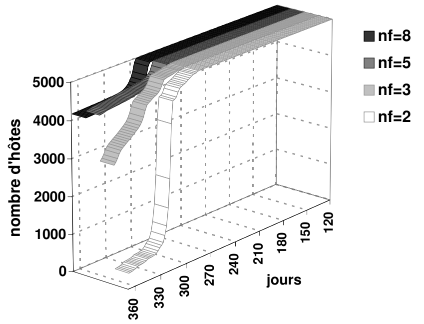


Figure 93. Evolution du nombre d'hôtes pour $\rho = 0.60$ et nf variable

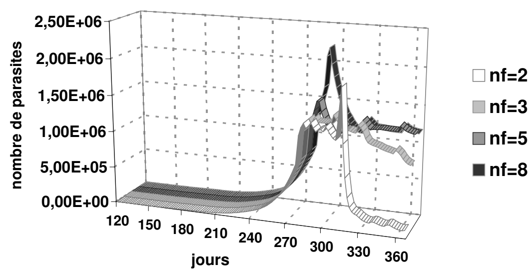


Figure 94. Evolution du nombre de parasites ayant pour $\rho = 0.60$ et nf variable

9.5 Suppression de la fonction F

Afin de travailler sur un modèle encore plus cohérent, on revient encore sur la fonction F afin de supprimer un comportement peu réaliste du modèle.

9.5.1 Distribution des parasites et agrégation

Analysons la relation qui existe entre le logarithme de l'abondance (notée $\overline{p}$) et le logarithme de la variance (notée σ^2) du nombre de parasites par hôte. Pour cette étude, on prend la simulation de référence[41] et $\rho = 0.6$. Sur la Figure 95, la taille du point est proportionnel au temps écoulé depuis $t = 0$. On a tracé la courbe identité qui correspond à une distribution aléatoire des parasites sur les hôtes pour laquelle l'abondance est égale à la variance. Au début de la simulation présentée, on remarque que la courbe suit celle de l'identité. Cela correspond au recrutement selon une loi de Poisson des parasites par les hôtes, tant qu'ils sont faiblement parasités. Ensuite, lorsque les hôtes commencent à être colonisés par plus de *palier* parasites, la surdispersion s'enclenche. Les points de la courbe sont alors alignés sur une droite pratiquement verticale. Cela correspond à une loi des puissances $log(\sigma^2) \approx log(a) + b\,log(\overline{p})$ avec un paramètre b très élevé. On en déduit que les paramètres d'entrée induisent une agrégation très forte sur les hôtes puisque la variance oscille tandis que la moyenne varie peu.
Sur le graphique de droite (Figure 95), on a formé 12 classes d'hôtes selon l'intensité parasitaire sur les hôtes et affiché la fraction d'hôtes dans chaque classe. L'échelle logarithmique permet d'observer simultanément les classes contenant beaucoup de parasites (ici lorsque

41. *cf* p. 126

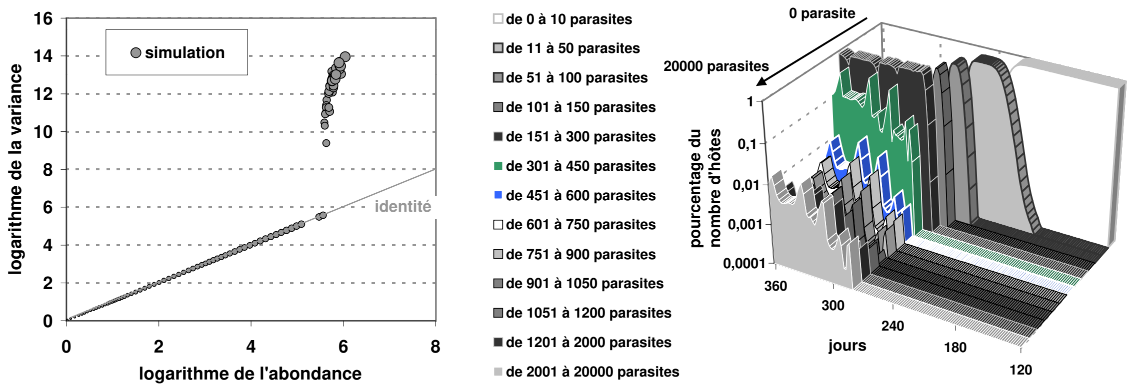


Figure 95. Variance fonction de l'abondance pour la simulation de référence avec $\rho = 0.6$, puis distribution (en proportion) des hôtes fonction du nombre de parasites portés (c'est-à-dire $N(l,t)/H(t)$)

les hôtes ont moins de 450 parasites) et les classes en ayant peu. Jusqu'à 280 jours, la distribution est centrée sur un nombre toujours croissant de parasites par hôte. Après cette date, tous les hôtes ont plus de 150 parasites et la majorité d'entre eux ont moins de 450 parasites. D'autre part, une minorité d'hôtes appartenant à la queue de distribution ont entre 450 et 20000 parasites. En regardant aussi les Figures 86 et 87 ($\mu = 5$ %), on note que les pics dans la queue de distribution provoquent la mort des d'hôtes très infestés et la décroissance de l'abondance. Chaque pic de la queue de distribution correspond à une nombre important de larves qui se fixent au temps t. Les larves colonisent majoritairement les hôtes très parasités ce qui provoque une hausse importante de la variance du nombre de parasites par hôte, et dans une moindre mesure de l'abondance. Une particularité de ce modèle est de tenir explicitement compte de l'évolution de la queue de distribution qui, on le voit ici, joue un rôle central dans la dynamique.

La loi binomiale négative abondamment utilisée pour modéliser l'agrégation de macroparasite n'est pas adéquate pour décrire des cas observés et certaines sorties de nos simulations. En effet, on constate deux choses simultanément : 1) il n'y a plus d'hôtes indemnes, 2) il y a beaucoup d'agrégation donc la variance est très élevée par rapport à la moyenne. En utilisant l'expression de $Prob(0|k,\overline{p})$ de l'équation (5) p. 18, le premier point implique que k doit être très grand pour que cette probabilité soit proche de zéro ($k \gg 1$). D'un autre côté, le fait d'avoir une variance très supérieure à la moyenne implique une petite valeur de k ($k = \overline{p}^2/(\sigma^2 - \overline{p})$, il est possible d'avoir $k < 1$). Cette contradiction implique que le choix de la loi binomiale négative n'est en aucun cas adapté à ce type de situation.

9.5.2 Dysfonctionnement des fonctions f et F

Envisageons les deux cas A et B (voir Figure 96) afin de comprendre l'interaction des fonctions F et f. Le cas A correspond par exemple à une situation pour laquelle les hôtes les plus parasités ont une intensité qui est proche mais au-dessus de *palier*. Le cas B

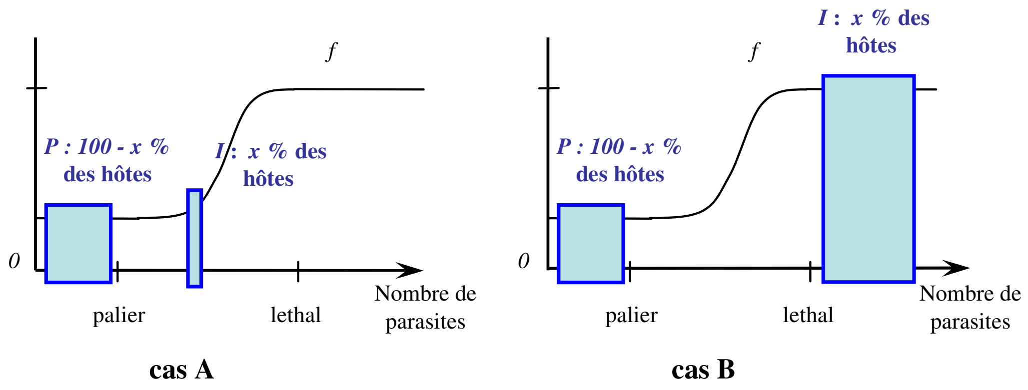


Figure 96. Deux cas de figure schématiques se présentant lors des simulations

représente une configuration qui peut se présenter lors d'une épizootie, avec certains hôtes ayant une intensité parasitaire proche de *lethal*.

On suppose pour les deux cas considérés que certaines variables sont égales : x la proportion d'hôtes dans i, L_r le nombre de larves à recruter, H le nombre d'hôtes. Il vient que le nombre de parasites qui se fixent sur les hôtes de i est $L_r\,F(x)$ dans les deux situations. On déduit que chaque hôte de i va recruter $\frac{L_r\,F(x)}{x\,H}$. La fonction f ne tient alors plus son rôle, car le nombre moyen de parasites recrutés est identique dans les deux situations. Il est anormal que les hôtes de i ayant tout juste plus de *palier* parasites ou bien beaucoup plus de *lethal* parasites recrutent autant de larves.

L'allure de la courbe f a été choisie pour que les hôtes ayant un peu plus de *palier* parasites recrutent légèrement plus que les hôtes ayant moins de *palier* parasites. Or la fonction F vient altérer ceci, dès que certains hôtes ont un peu plus de *palier* parasites, ils recrutent énormément. En conséquence, la variance s'élève subitement sur la Figure 95 lorsque certains hôtes acquièrent plus de *palier* parasites en t= 280 jours. Dernier point, l'abondance reste ensuite relativement figée, très contrainte par le système. On peut s'interroger sur le sens de ce phénomène qui peut être qualifié de non-linéaire. La fonction f devait introduire un recrutement agrégatif progressif en fonction du nombre de parasites portés. Nous allons donc remanier le modèle afin de réduire cet effet.

Enfin, il serait souhaitable de diminuer le nombre de paramètres de modélisation qui concernent le recrutement ; en effet, les 5 paramètres disponibles actuellement sont *palier*, *lethal*, nf, x_i, a_0.

9.5.3 Suppression de la fonction F

Supposons que l'on soit dans la situation suivante : le groupe i est constitué d'hôtes ayant plus de *lethal* parasites et représente x% des hôtes (cas B Figure 96). Nous allons supprimer F en raisonnant à partir de ce cas particulier tout en gardant un modèle très semblable.

On pose que parmi les hôtes de i, la valeur de $f(l,t)$ est a fois plus élevée que pour les hôtes de $\dot{\mathrm{P}}$. Notre nouvelle hypothèse est simplement que les hôtes ayant plus de *lethal*

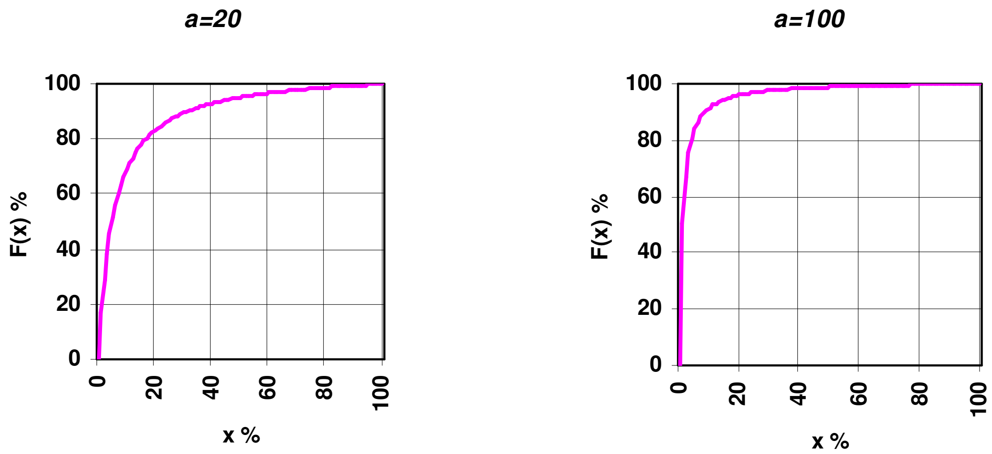


Figure 97. Fonction F implicite pour $a = 20$ et $a = 100$

parasites ont une espérance de recrutement a fois plus élevée que les hôtes ayant moins de *palier* parasites.

Pour les hôtes du groupe $\dot{\text{I}}$, la fonction f vaut $f_0(t)$ par convention ; pour les hôtes de $\dot{\text{P}}$, la fonction f est égale à $a\, f_0(t)$. Le groupe $\dot{\text{I}}$ va recruter la proportion suivante des larves disponibles $L_r(t)$:

$$\frac{x(t)\, a\, f_0(t)\, L_r(t)}{(100 - x(t))\, f_0(t)\, L_r(t) + x(t)\, a\, f_0(t)\, L_r(t)} = \frac{x(t)\, a}{100 + x(t)\,(a-1)} \ .$$

On peut tracer la courbe de la proportion de parasites recrutés dans $\dot{\text{I}}$ en fonction de $x(t)$ et de a. On obtient une allure de la fonction F implicite (voir Figure 97). Cette courbe est implicite parce que la fonction F n'est pas définie *a priori* mais elle se déduit dans le cas B pour une valeur de a donnée. Dans ce cadre, on note que la fonction F implicite est de la même forme que la fonction F précédente. La fonction f conserve sa définition antérieure (*cf* p. 135) ; les valeurs de $f_0(t)$ et $\lambda(t)$ sont maintenant données par les formules :

$$f_0(t) = \left(\sum_{l=0}^{pmax(t)} N(l,t) + (a-1) \sum_{l=palier+1}^{pmax(t)} N(l,t) \frac{f_1(l-palier)}{f_1(lethal-palier)} \right)^{-1}$$

$$\lambda(t) = \frac{(a-1)\, f_0(t)}{f_1(lethal-palier)}$$

La relation de consistance de l'équation (12) p. 41 est vérifiée pour ces définitions. Dans la suite de ce document, on substitue ces nouvelles relations donnant $f_0(t)$ et $\lambda(t)$ aux précédentes. Les deux paramètres x_i et a_0 sont maintenant remplacés par le seul paramètre a.

9.5.4 Utilisation du modèle sans fonction F

Cette modification du modèle permet de résoudre le problème issu de l'interdépendance des fonctions f et F. Dans ce paragraphe, nous regardons l'effet de la modification apportée sur la distribution des parasites sur les hôtes. Nous remplaçons le couple ($x_i = 5$ %, $a_0 = 95$ %) par le paramètre $a = 3333$ dans la simulation de référence (*cf* p. 125). Ce paramétrage conduit à des sorties qui sont quantitativement proches de celles issues du simulateur qui utilisait la fonction F auparavant. On observe sur la Figure 98 que la distribution des parasites sur les hôtes évolue de manière plus continue que sur la Figure 95. Le nombre d'hôtes qui ont beaucoup de parasites et se trouvent dans la queue de distribution oscille. Dans cette nouvelle configuration, la distribution des parasites sur les hôtes est moins figée et semble jouer un rôle dans la dynamique. Nous détaillerons cette simulation en tenant compte d'autres sorties dans le prochain chapitre.

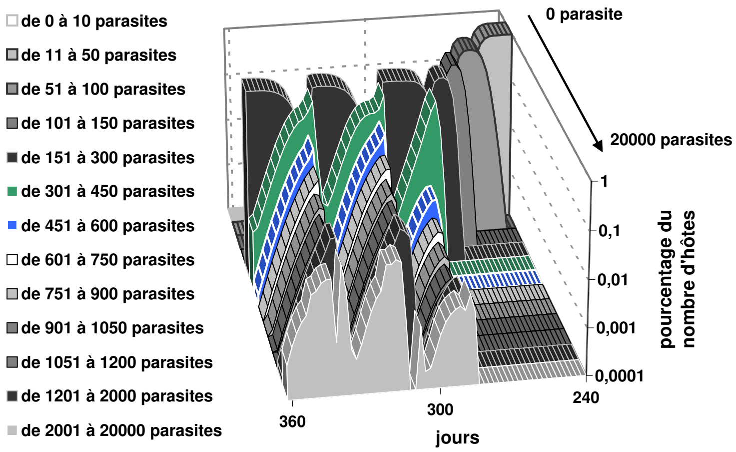


Figure 98. Distribution (en proportion) des hôtes fonction du nombre de parasites portés, c'est-à-dire $N(l,t)/H(t)$

9.6 Conclusion

La simulation a permis d'exhiber cinq types de dynamique possibles pour un système Bar-*Diplectanum aequans* dans le contexte d'un élevage en aquaculture. Ces types correspondent aux observations effectuées sur le terrain par le biologiste.

Néanmoins, certaines propriétés du modèle mathématique et des simulations n'étaient pas compatibles avec ce que l'on attendait. Nous sommes donc revenus sur le modèle et nous avons remis en cause certaines hypothèses implicites. L'outil de simulation permet cette rétroaction qui est porteuse d'information [58]. L'amélioration du modèle en découle et consiste à intégrer de nouvelles connaissances sur le système en précisant les mécanismes en cause. En ajustant le modèle, des artéfacts numériques ont été supprimés améliorant ainsi le réalisme.

On observe quantitativement une régulation par la mort des hôtes. Elle est liée au pourcentage d'hôtes ayant plus de *palier* parasites qui présente des oscillations. La surdispersion réalise des liens entre les variables du système et agit fortement au cœur de la dynamique de populations. Il est bien connu que l'agrégation a un rôle primordial dans la dynamique de populations observée [15]. Cependant, la simulation apporte des informations qualitatives et quantitatives supplémentaires pour comprendre finement les mécanismes impliqués.

Chapitre 10

Comparaison qualitative des modèles déterministe et basé sur une méthode de Monte-Carlo

Dans ce chapitre, nous allons comparer les résultats des simulateurs déterministe et stochastique et donner une brève analyse de sensibilité. On verra que le comportement des deux modèles sont qualitativement proches, bien que l'on puisse isoler des cas pour lesquels ils se distinguent.

10.1 Etude sur μ

Sauf indication contraire, toutes les simulations stochastiques présentées dans ce chapitre ont été réalisées à l'aide de $R = 512$ réplications (*cf* p. 121). Regardons les simulations pour lesquelles les paramètres d'entrée μ et ρ varient. Sur les Figures 99 et 100, le comportement qualitatif de la courbe déterministe (gris foncé plein) et de la courbe moyenne stochastique (pointillé noir) sont proches. Le *premier quartile* est la valeur pour laquelle 25 % de l'échantillon est en-dessous, et 75 % de l'échantillon au-dessus. Le *deuxième quartile* correspond à la médiane. Le *troisième quartile* est la valeur pour laquelle 75 % de l'échantillon est en-dessous, et 25 % de l'échantillon au-dessus. L'*étendue interquartile* est définie comme la différence entre le troisième et le premier quartile. Cette étendue (le fuseau blanc entouré de deux fuseaux gris sur les Figures qui suivent) peut être utilisée pour caractériser l'étalement d'une distribution. A un certain temps t, un échantillon (une variable observée pour une réplication donnée) appartient à l'étendue interquartile avec une probabilité de $\frac{1}{2}$.

Les Figures 101 et 102 présentent le cas où la mortalité des parasites est abaissée à un taux de 2 % par pas de temps. L'effet est significatif sur la population hôte, la décroissance de l'effectif commence dès $t = 180$ et ralentit progressivement. On constate qu'il y a peu de ralentissements cycliques de la population parasitaire par les morts d'hôtes comme sur la dynamique précédente. Ces cycles sont visibles à la fois sur une réplication stochastique et sur la courbe déterministe ; par contre la moyenne stochastique efface ce type de comportement qui n'est pas synchrone pour toutes les réplications. On remarque aussi que l'étendue interquartile est étroite lorsque le taux de survie des parasites augmente (μ décroît). Les paragraphes qui suivent détaillent les raisons du phénomène cyclique et la dépendance entre μ et la variance des sorties.

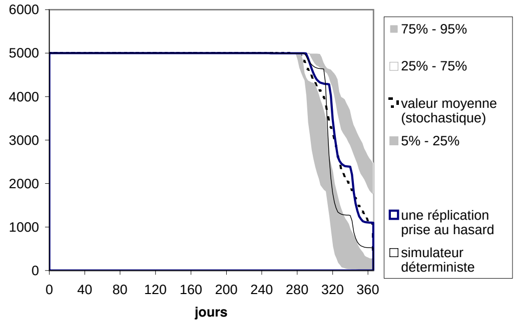


Figure 99. Spectre du nombre d'hôtes pour $\rho = 0.6$ et $\mu = 5\%$ (simulation de référence)

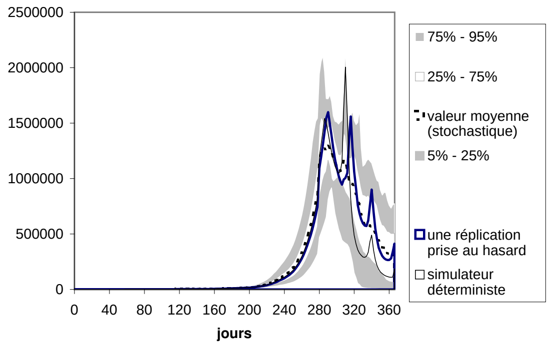


Figure 100. Spectre du nombre de parasites pour $\rho = 0.6$ et $\mu = 5\%$ (simulation de référence)

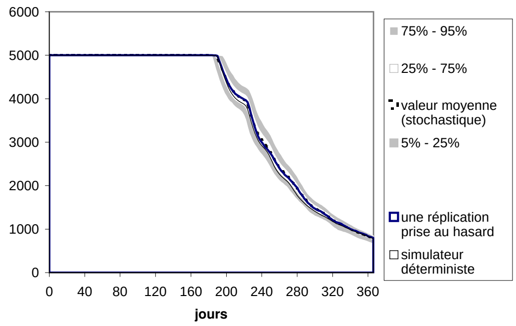


Figure 101. Spectre du nombre d'hôtes pour $\rho = 0.6$ et $\mu = 2\%$

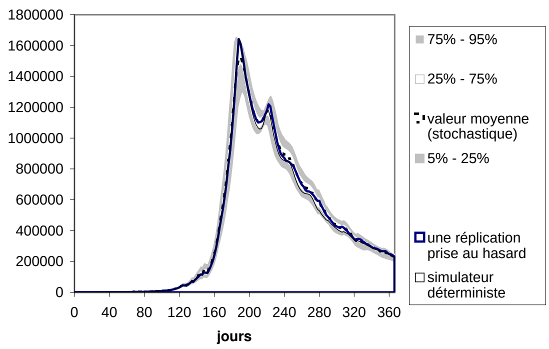


Figure 102. Spectre du nombre de parasites pour $\rho = 0.6$ et $\mu = 2\%$

Phénomène d'oscillation Pour illustrer la manière dont les parasites se répartissent parmi les hôtes disponibles à un temps t, et les conséquences que va avoir cette répartition sur le comportement démographique du système couplé, nous avons représenté sur la Figure 103 les variations du logarithme de l'espérance du nombre de parasites par hôte (abscisse) et du logarithme de la variance correspondante (ordonnée) pour la simulation de référence (voir aussi les Figures 99, 100 correspondant à la même simulation).

De $t = 0$ à $t = 284$, le recrutement est de type Poissonien en deçà du seuil *palier*. L'agrégation croît ensuite non linéairement avec l'intensité parasitaire : c'est la larve qui cherche son hôte, et elle le trouve d'autant plus facilement que ce dernier est affaibli par l'importance de son parasitisme. Après $t = 284$, l'intensité de certains hôtes devient

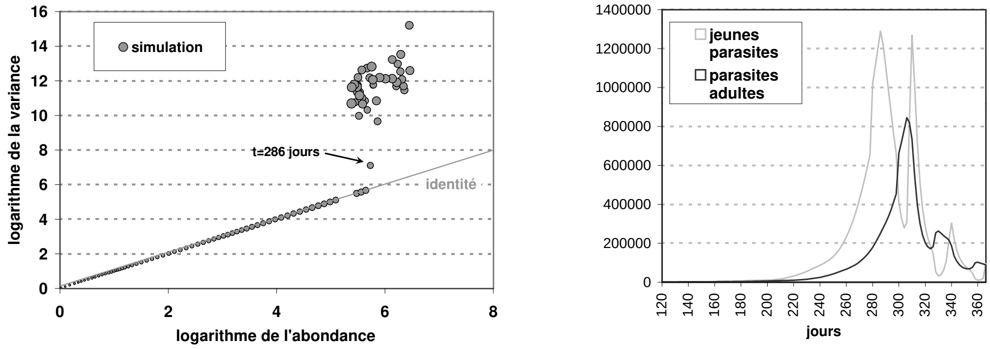


Figure 103. Variance fonction du nombre moyen de parasites par hôte pour la simulation de référence $\rho = 0.6$, et nombre de parasites jeunes et adultes

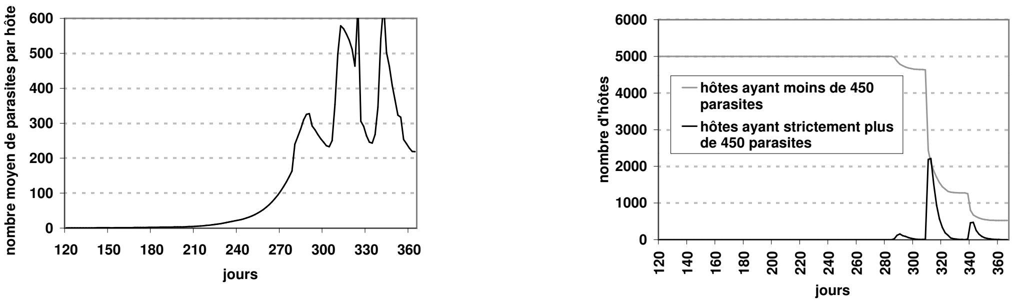


Figure 104. Nombre moyen de parasites par hôte pour la simulation de référence $\rho = 0.6$, puis nombre d'hôtes dans la queue de distribution de $N(., t)$

supérieure à $palier = 300$ parasites ; l'inégalité dans le recrutement débute, la variance s'accroît alors plus vite que la moyenne. L'abondance et le nombre de jeunes parasites chutent lorsque les hôtes sur-infestés commencent à disparaître. En effet, l'agrégation induit que les hôtes très infestés (voire mourant), recrutent majoritairement les larves. Lorsque $t = 302$, la moyenne (voir Figure 104) est réduite (236 parasites), le nombre de jeunes est faible (voir Figure 104), la mortalité des hôtes est très ralentie. Dès lors, les conditions sont propices au développement parasitaire. Le peu d'hôtes restant dans la queue de distribution N meurent (voir Figure 104). Le nombre de parasites moyen augmente à nouveau et la surdispersion introduit une hausse du nombre d'hôtes très infestés ayant plus de 450 parasites. Un nouveau cycle démarre en $t = 310$ ressemblant à celui initié en $t = 284$. Ces fluctuations sont l'illustration même des propos de C. Combes [15] “On peut dire de façon imagée que toute hétérogénéité crée de la surdispersion et que tout processus densité-dépendant crée de la sous-dispersion. ” ; dans notre étude le processus densité-dépendant consiste en la mort des hôtes sur-infestés.

Ecart-type fonction de μ On remarque sur les Figures 100 et 102 que l'espace interquartile augmente dans un premier temps jusqu'aux premières morts d'hôtes. On va étudier ce phénomène particulier qui semble dépendre de la mortalité μ. L'écart-type du

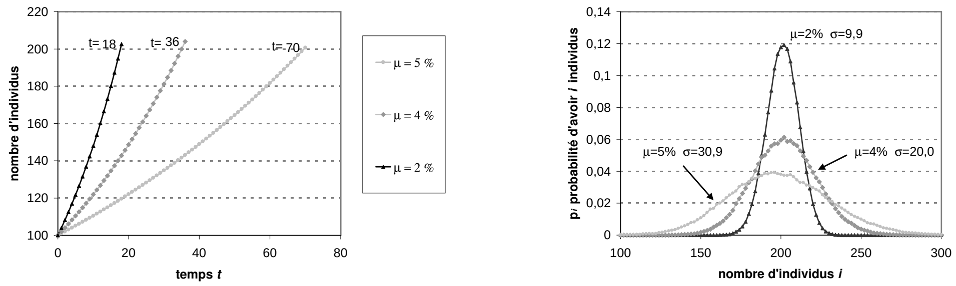
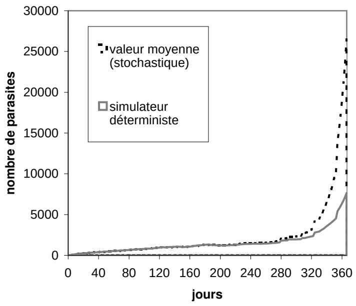


Figure 105. Evolution du nombre moyen de parasite $Y^{\diamond}(t)$ pour trois mortalité μ différentes, $Y(0) = 100$, $\gamma = 1.06$ (à gauche) ; fréquence du nombre de parasites lorsque le nombre moyen de parasites atteint 200 individus fonction de μ (à droite)

nombre de parasites lorsque celui-ci atteint 1600000 paraît corrélé positivement avec μ. Pour comprendre ce fait, on construit un méta-modèle : on considère que les parasites se reproduisent grâce à un processus déterministe (loi de croissance exponentielle), et qu'ils meurent selon un processus stochastique (loi binomiale de probabilité élémentaire μ). Ce méta-modèle décrit bien la partie initiale de croissance exponentielle des parasites, si l'on néglige les variations de l'environnement et des phénomènes retard. Soit $Y(t)$ le nombre de parasites au temps t, on a

$$Y(t+1) = g(Y(t))\,Y(t) - \mathcal{B}(\mu, Y(t)) \quad .$$

On considère tout d'abord que $g(Y(t)) = \gamma > \mu$. On calcule des réplications de ce méta-modèle pour différentes valeurs de μ. A l'initialisation on prend $Y(0) = 100$, on arrête les simulations lorsque la valeur moyenne des réplications $Y^{\diamond}$ atteint 200. Si l'on prend un modèle déterministe de cette croissance, on a $Y(t) = Y(0)\,(\gamma - \mu)^t$, ce qui correspond aux moyennes du méta-modèle présentées Figure 105. Le temps de doublement de la population est alors $t_2(\mu) = ln(\gamma - \mu)/ln(2)$ ($t_2(0.05) = 70$; $t_2(0.04) = 35$; $t_2(0.02) = 28$). La croissance déterministe est identique à celle de la moyenne stochastique. On explique que l'écart-type dépend de μ à cause du processus de mortalité qui suit une loi binomiale. Dans un nombre de cas restreints, on constate dans nos résultats une différence notable entre la simulation stochastique et déterministe. On présente sur la Figure 106 le nombre de parasites dans la simulation de référence pour $\rho = 0.40$. La moyenne stochastique augmente plus rapidement que la sortie déterministe. Dans la littérature, ce phénomène est souvent lié à l'action d'un mécanisme non-linéaire. On ajoute au méta-modèle précédant un mécanisme supplémentaire qui agit dans le système original : le fait que la ponte d'œufs dépend du nombre de parasites fixés sur l'hôte. Précédemment (p. 51), on a vu que le parasite adulte pond avec une probabilité de 0 à 0.5 puis 1 lorsque respectivement l'hôte a un parasite, 2 ou plus. Lors du développement de la population parasitaire, il y a donc un moment où le taux de croissance augmente. On modifie le méta-modèle en prenant une fonction g croissante en Y et en admettant une limite asymptotique en $+\infty$: $g(Y) = \alpha + \gamma Y^3/(c^3 + Y^3)$. Sur les courbes de droite de la Figure 106, on constate que le méta-modèle reproduit bien



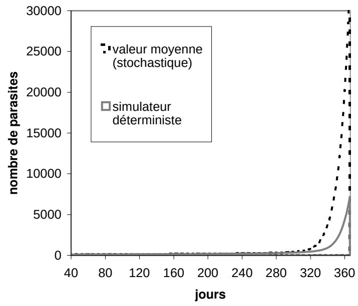


Figure 106. Evolution du nombre moyen de parasites pour la simulation de référence $\rho = 0.40$ (à gauche) ; Evolution du nombre moyen de parasites $Y^{\diamond}(t)$ avec le méta-modèle paramétré par $Y(5) = 100$, $\gamma = 0.18$, $c = 700$, $\alpha = 1.055$, $\mu = 0.05$ (à droite)

un écart entre courbes déterministe et stochastique.

Les résultats des deux simulateurs permettent ici de quantifier un phénomène intéressant : la croissance initiale du parasitisme s'effectue selon plusieurs modes successifs. Le taux de croissance grandit en fonction de l'environnement et aussi en fonction de la répartition des parasites sur les hôtes. La non-linéarité de la croissance du parasitisme explique que, d'une réplication stochastique à l'autre, le moment où l'accroissement devient très élevé est décalé dans le temps. La comparaison des résultats stochastique et déterministe est très profitable puisqu'elle nous amène à regarder les processus non-linéaires qui influent sur la dynamique.

Que ce soit pour cette version du modèle ou celles pour lesquelles les fonctions F et f étaient différentes, on constate une forte sensibilité des résultats du modèle déterministe à la perturbation des paramètres d'entrée. Ce fait est relié notamment aux deux phénomènes non-linéaires décrits ci-dessus : la croissance initiale de la population parasitaire, les oscillations témoignant d'une régulation de la population parasitaire. Ceci appuie *a posteriori* la nécessité de réaliser des calculs précis en double précision pour la simulation déterministe (*cf* p. 81).

10.2 Etude sur le paramètre *lethal*

Les simulations présentées par les Figures 107, 108, 109 correspondent à la simulation de référence ($\rho = 0.85$) pour laquelle on fait varier le paramètre *lethal*. Pour les trois valeurs de *lethal* : 800, 1200, 1600, le coût en calcul des simulations est respectivement de 72 TFLOP, 364 TFLOP, 1148 TFLOP.

La survie des hôtes est améliorée lorsque l'on augmente le seuil *lethal*. Plus précisément, la mortalité des hôtes lors des cycles est diminuée lorsque *lethal* croît. Au niveau du nombre de parasites, les courbes présentent moins de discontinuités pour un paramètre *lethal* élevé. Pour $lethal = 1600$, les premiers cycles apparaissent synchrones au niveau des simulations stochastique et déterministe.

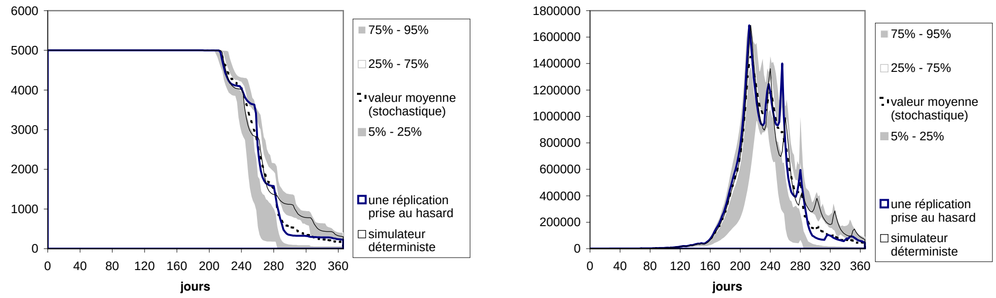


Figure 107. Evolution du nombre d'hôtes (à gauche) et du nombre de parasites (à droite) pour *lethal* égal à 800 ; les autres paramètres son ceux de la simulation de référence et $\rho = 0.85$

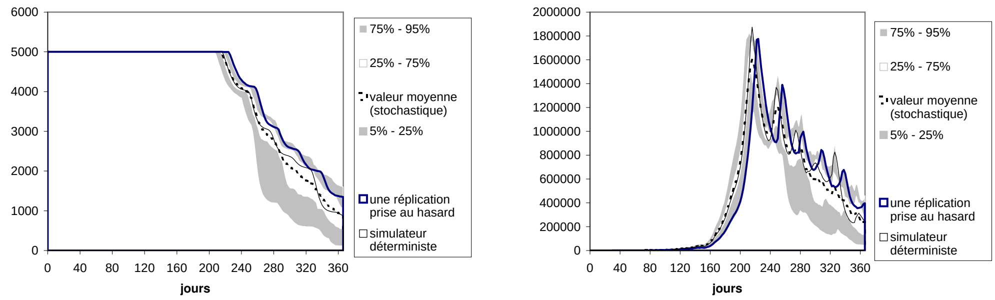


Figure 108. Evolution du nombre d'hôtes (à gauche) et du nombre de parasites (à droite) pour *lethal* égal à 1200 ; les autres paramètres son ceux de la simulation de référence et $\rho = 0.85$

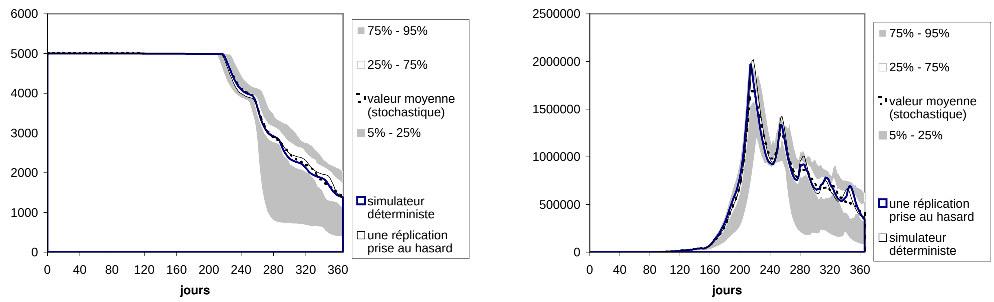


Figure 109. Evolution du nombre d'hôtes (à gauche) et du nombre de parasites (à droite) pour *lethal* égal à 1600 ; les autres paramètres son ceux de la simulation de référence et $\rho = 0.85$

10.3 Etude sur *palier*, seuil de la surdispersion

Etudions l'effet des modifications apportées au modèle sur les résultats des simulations lorsque *palier* et ρ varient.

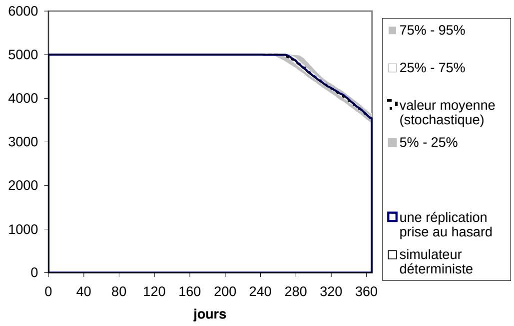


Figure 110. Spectre du nombre d'hôtes pour $\rho = 0.6$ et *palier* $= 50$

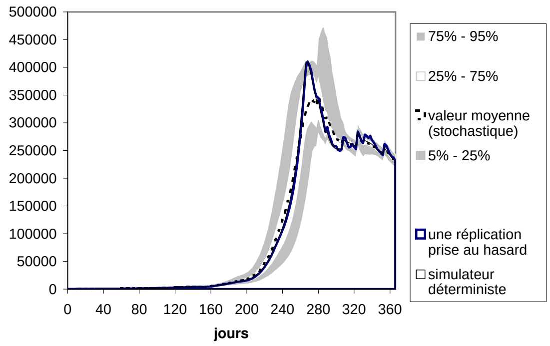


Figure 111. Spectre du nombre de parasites pour $\rho = 0.6$ et *palier* $= 50$

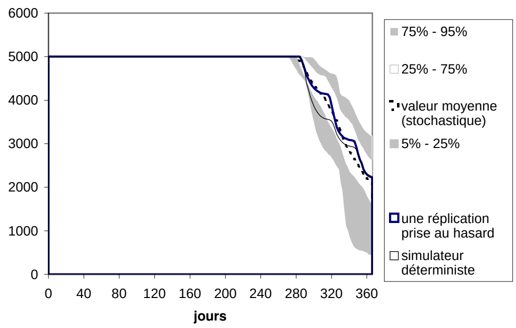


Figure 112. Spectre du nombre d'hôtes pour $\rho = 0.6$ et *palier* $= 200$

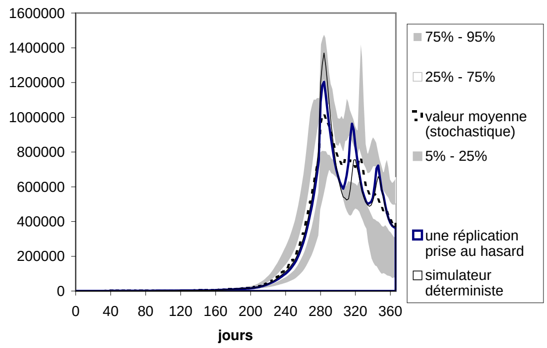


Figure 113. Spectre du nombre de parasites pour $\rho = 0.6$ et *palier* $= 200$

Pour un palier à 50 (Figures 110 et 111), l'étendue interquartile augmente jusqu'en $t = 268$ jours, puis diminue considérablement. La réplication prise au hasard se confond au résultat déterministe durant une très grande partie de la simulation. Lorsque *palier* $=$ 200, l'étendue interquartile reste importante jusqu'à la fin de la simulation. Les cycles de régulation de la population parasitaire par la mort des hôtes semblent être à l'origine de la variabilité des résultats dans les simulations stochastiques. Pour *palier* $= 50$, il n'y a pas de cycles. L'abondance est peu élevée et le nombre parasites adultes dans le système est moindre par rapport à un *palier* plus grand. Une fois que le croissance des parasites a été amortie par la mort des hôtes ($t = 300$), il n'y a plus assez de parasites adultes pour que la population parasitaire s'accroisse à nouveau. La régulation par mort des hôtes sur-infestés maintient un flux égal de parasites qui entrent et sortent du système.

10.4 Etude sur nf, a, paramétrant la surdispersion

Examinons les simulations pour lesquelles on fait varier les paramètres d'entrée ρ, nf et a.

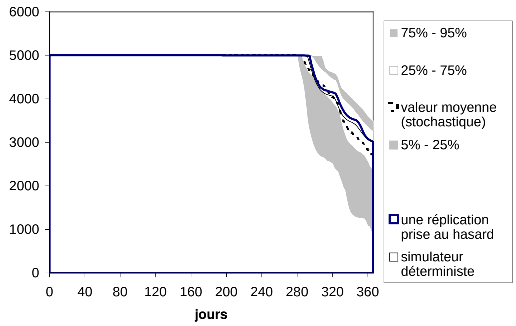


Figure 114. Spectre du nombre d'hôtes pour $\rho = 0.6$ et $nf = 5$

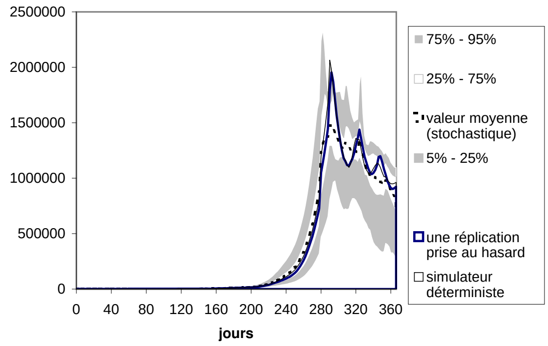


Figure 115. Spectre du nombre de parasites pour $\rho = 0.6$ et $nf = 5$

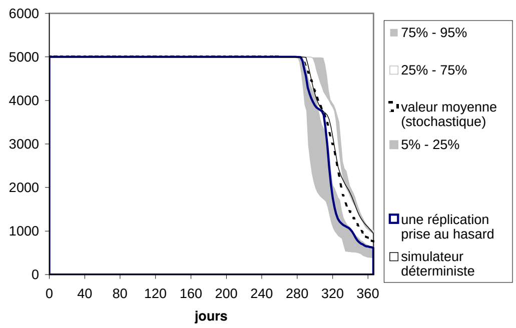


Figure 116. Spectre du nombre d'hôtes pour $\rho = 0.6$ et $a = 500$

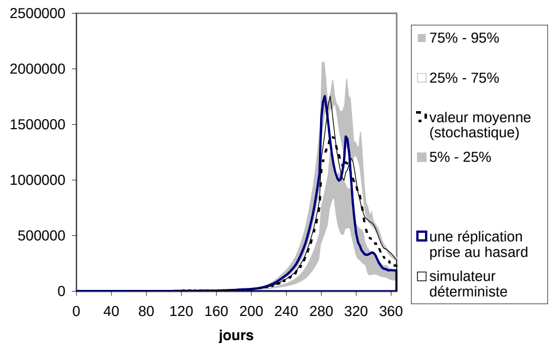
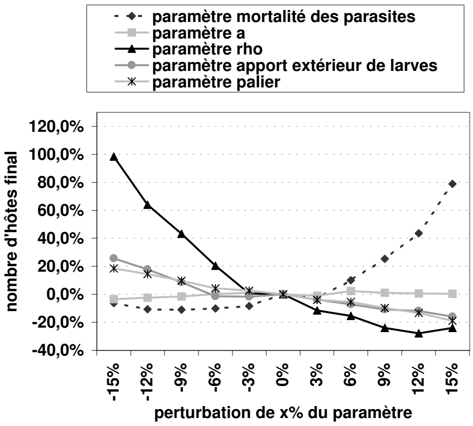


Figure 117. Spectre du nombre de parasites pour $\rho = 0.6$ et $a = 500$

Lorsque $nf = 5$ (Figures 114 et 115), la survie des hôtes augmente par rapport à la simulation de référence $nf = 3$ (Figure 99). Lorsque nf est élevé, les hôtes les plus infestés recrutent un nombre important de larves durant l'épizootie. Ce mécanisme permet au reste de la population hôte d'être moins touché par le parasitisme. Les Figures 116 et 117, illustrent l'effet d'une diminution du paramètre a par rapport à la simulation de référence. On constate que l'agrégation agit alors comme un frein moins puissant à la mort des hôtes. A l'instar de nf, quand le paramètre a est grand, cela favorise l'agrégation et améliore la survie des hôtes.

10.5 Etude de sensibilité (stochastique)

Les Figures 118, 119 illustrent une analyse de sensibilité de 50 simulations autour de la simulation de référence (pour laquelle on a néanmoins changé $palier = 200$).



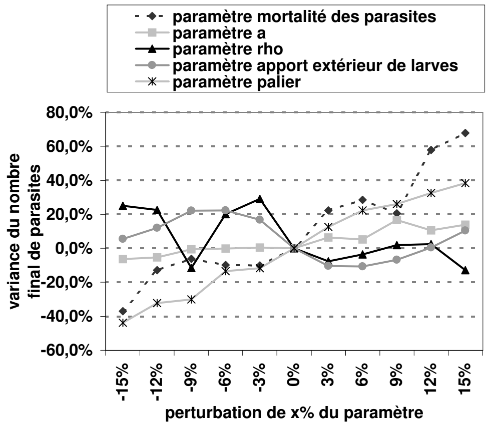


Figure 118. Variation du nombre d'hôte en $t = 366$ en fonction de la variation des paramètres d'entrée (simulation stochastique pour le jeu de paramètres de référence avec $palier = 200$ et $\rho = 0.6$), puis variation de la variance du nombre final de parasites (estimée pour $R = 512$)

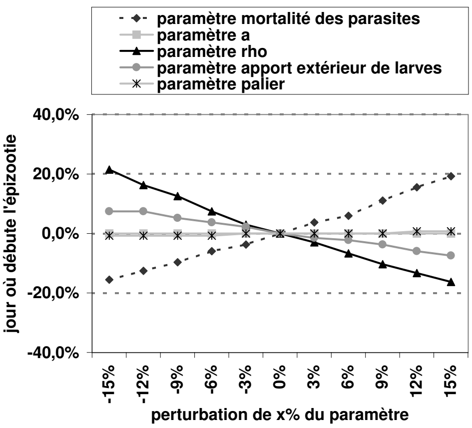


Figure 119. Variation du jour où débute l'épizootie en $t = 366$ en fonction de la variation des paramètres d'entrée (simulation stochastique avec jeu de paramètres de référence avec $palier = 200$ et $\rho = 0.6$)

On remarque que les deux paramètres qui influencent le plus le nombre final d'hôtes (et l'ensemble de la dynamique) sont μ et ρ. Une mortalité des parasites μ faible ou un paramètre ρ élevé conduisent à une survie moindre des hôtes, et un début d'épizootie plus tôt dans l'année. La sensibilité par rapport à l'apport extérieur de larves et à *palier* est moindre. La variance du nombre de parasites au temps $t = 366$ jours est une fonction croissante des paramètres μ, *palier* et a. Elle est très sensible à la variation de la mortalité des parasites et au *palier* de recrutement ; en effet, ils déterminent fortement l'allure des cycles de régulation de la population parasitaire. Les phénomènes non-linéaires issus de ces cycles sont à l'origine d'une variance importante (d'une réplication à l'autre les cycles sont décalés dans le temps).

La date où les premières morts d'hôtes sont recensées est visiblement reliée à trois des paramètres dont on ne connaît pas précisément la valeur : la mortalité des parasites, ρ et l'apport extérieur de larves. Ceci est intéressant car cette caractéristique donne une relation entre ces paramètres qui peut aider à les estimer. Si l'on suppose que l'on connaît la date des premières morts d'hôtes, et la mortalité des parasites avec suffisamment de précision, les simulations peuvent nous donner ρ en fonction de l'apport extérieur de larves.

Il serait intéressant de réaliser une analyse statistique des résultats produits par la simulation pour mieux caractériser et interpréter les deux modèles. Cette étude serait très enrichissante si elle était menée conjointement avec une validation expérimentale afin de comprendre au mieux les processus à l'œuvre dans un système hôte-macroparasite.

10.6 Conclusion

La multiplicité et la complexité des comportements démographiques observés dans nos simulations ont notamment pour origine l'ampleur du recrutement des stades infestants, la manière dont ces derniers se dispersent parmi les hôtes, la survie des parasites fixés, la variabilité de l'impact de l'intensité parasitaire, l'influence des paramètres environnementaux. Une simulation précise permet d'éclairer cas par cas les dynamiques dans un contexte d'agrégation variable, dans le temps et dans l'espace des hôtes. Cela serait impossible si l'on se contentait d'intégrer des lois de probabilité invariantes (de type binomiale négative ou Poisson) dans des systèmes d'équations différentielles. Les effets des processus générateurs de surdispersion et de sousdispersion, simultanés ou successifs, peuvent désormais être quantifiés et expliqués précisément, alors qu'ils relevaient de déductions hypothétiques auparavant [5, 67]. Nous avons illustré le rôle différent que peuvent jouer les cohortes d'adultes et de jeunes parasites dans la dynamique d'un système hôte-parasite.

La simulation permet aussi de relier des paramètres et des variables internes du modèle aux données recueillies sur le terrain. L'exploration des états démographiques potentiels de ces modèles déterministes et stochastiques est loin d'être achevée ; d'autant qu'ils ne s'appuient pour l'instant que sur une application : un système biologique Téléostéen-Monogènes. Néanmoins, les principes développés sont transposables à termes à de nombreux systèmes hôte-macroparasite régulés par la mort des hôtes.

Conclusion générale

Bilan des résultats obtenus dans cette thèse

Cette thèse présente deux modèles d'un système hôte-macroparasite et deux simulateurs hautes performances. La modélisation mathématique du système puis la simulation nous permettent d'étudier quantitativement la dynamique du système. Deux simulateurs numériques parallèles ont été développés, un premier ayant une approche déterministe, un second mettant en œuvre une méthode individu-centrée. Ces simulateurs constituent des applications de très grandes tailles. Nous avons réalisé une mise en œuvre optimisée et efficace des simulateurs pour des machines parallèles. Nous présentons une interprétation des résultats numériques de la simulation et nous analysons le rôle des paramètres du modèle hôte-macroparasite. Ce travail apporte donc des contributions dans plusieurs disciplines.

Modèle Bio-mathématique Le phénomène d'agrégation des macroparasites est classiquement pris en compte par une distribution des parasites sur les hôtes de type loi binomiale négative. Dans la première partie de cette thèse, nous exposons un modèle général de système hôte-macroparasite qui n'utilise pas une telle hypothèse simplificatrice. Les hétérogénéités de la population parasitaire y sont reproduites plus finement, ce qui constitue une approche originale. Une application du modèle est donnée pour un macroparasite constituant un élément pathogène en aquaculture : le système Bar-*Diplectanum Aequans*. La gestion de la distribution des parasites sur les hôtes finaux peut être utilisée pour modéliser différents systèmes hôte-parasite. De nombreux échanges inter-disciplinaires m'ont permis de faire des avancées sur le modèle bio-mathématique, ainsi que sur son paramétrage. J'ai notamment collaboré avec Patrick Silan en ce qui concerne la biologie et la dynamique des populations, et avec Michel Langlais dans le domaine de la modélisation bio-mathématique.

Algorithmique parallèle Nous avons abordé les problèmes de complexité algorithmique inhérents à la simulation du système hôte-macroparasite. Une solution parallèle du modèle déterministe a été développée puis affinée. Nous avons décrit la distribution des calculs et des données, et évalué les volumes des communications. Les performances sont données pour une application calculant des centaines de TFLOP. Une étude fine de l'algorithme a été conduite afin de démontrer formellement son extensibilité et sa capacité de passage à l'échelle. Nous avons pu constater plus de 77 % d'efficacité sur 448 processeurs. Une analyse en profondeur du code, ainsi qu'une modélisation mathématique de la répartition des charges conduisent à des performances remarquables, au cœur des calculs parallèles : 92 % d'efficacité est atteint sur 448 processeurs.

Accès optimisé aux caches Afin de pouvoir réaliser des simulations encore plus réalistes, il a fallu changer certains paramètres de la simulation qui la rendent très coûteuse en calcul (jusqu'à 1.45 PFLOP). Notre contribution dans ce domaine a consisté à optimiser l'accès aux caches de données afin d'atteindre une puissance maximale au niveau des calculs parallèles, à savoir 60 % de la puissance crête sur IBM SP3 NH2 et Origin 3800. La démarche utilisée pour faire apparaître des calculs par blocs est détaillée ; *in fine* les performances sont uniquement limitées par la bande passante mémoire.

Simulation parallèle stochastique D'autre part, on donne un équivalent stochastique du modèle déterministe qui permet de modéliser plus finement le système et donne accès à d'autres variables d'état (la variance par exemple). En outre, ce modèle stochastique permet d'introduire des hétérogénéités supplémentaires au niveau des hôtes et des parasites, propriété que nous pourrons mettre en œuvre dans le futur (mortalité des parasites différenciée en âge). Nous avons exhibé trois niveaux de parallélisme pour les simulations de type Monte-Carlo, ce qui nous a permis d'utiliser jusqu'à 256 processeurs très efficacement.

Interprétation des résultats Les deux simulateurs ont des temps d'exécution raisonnables et constituent des outils efficaces pour analyser le système. Ces simulateurs effectuent des calculs précis permettant d'obtenir des dynamiques de populations que l'on n'avait pas obtenues avec un précédant simulateur séquentiel qui utilisait trop d'approximations. Le modèle bio-mathématique qui sous-tend le simulateur contenait potentiellement ces nouvelles dynamiques ; la simulation parallèle a permis de les mettre en évidence.

La simulation parallèle permet d'ouvrir de manière conséquente l'éventail des dynamiques de populations accessibles. Les comportements qualitatifs obtenus donnent un éclairage nouveau sur l'interaction des mécanismes à l'œuvre dans les systèmes hôte-macroparasite. La simulation permet d'établir des relations entre les variables du système qui aident à identifier plus précisément les paramètres inconnus. L'amélioration de nombreuses fonctions mathématiques internes à ce modèle conduit à l'élimination de plusieurs artéfacts numériques et donc à plus de réalisme.

D'autre part, ces modifications et l'utilisation du simulateur stochastique font ressortir des propriétés du modèle et du système biologique. La gestion explicite de la distribution des parasites sur les hôtes et l'utilisation de paramètres environnementaux donnent l'accès à des dynamiques riches et complexes. Des liens entre les différentes variables du système hôte-parasite et leur rôle dans des cycles régulateurs sont décrits. Nous avons mis en évidence que la valeur de certains paramètres mal connus, contrôlant la force de la surdispersion, peut orienter la dynamique vers une mortalité des hôtes peu ou très importante. On constate pour de nombreux jeux de paramètres que la régulation par la mort des hôtes sur-infestés génère des oscillations de la population parasitaire ; nous donnons une interprétation quantitative à ce phénomène.

Perspectives et travaux Futurs

La modélisation d'épidémie sur un support non homogène, ou celle de dynamique complexe n'est pas toujours possible avec les seuls outils mathématiques. La simulation informatique s'avère être un complément essentiel pour analyser et comprendre ces modèles. Il serait intéressant d'adapter les techniques parallèles que nous avons développées à d'autres modèles déterministes qui incluent des éléments spatiaux.

La simulation stochastique présentée est d'autant plus actuelle qu'elle est particulièrement bien adaptée à une architecture de type grappe de nœuds multi processeurs ; or cette architecture connaît actuellement un fort développement. Ce type de simulation trouve un intérêt dans la modélisation de systèmes complexes et se trouve être un très bon candidat pour des plate-formes de type grille de calcul. En effet, les tâches élémentaires que constituent les réplications sont intrinsèquement très indépendantes et nécessitent peu de communications.

Le développement de hiérarchies mémoire qui comportent de nombreux niveaux (tel l'IBM Regatta) impose de nouvelles contraintes pour réaliser des applications hautes performances. Comme on l'a illustré au cours de cette thèse, la prise en compte de la localité mémoire est cruciale. La conception d'algorithmes paramétrables qui permettent d'utiliser efficacement les différents niveaux de caches pour différentes machines est intéressant pour la simulation numérique. Il me semblerait très profitable de développer et d'analyser en profondeur la prise en compte de la localité des données dans le cache avec les spécialistes de ce domaine.

On peut envisager de mettre au point un algorithme ne réalisant que partiellement les mises à jour du modèle déterministe de manière à en réduire les coûts. Il faudra regarder précisément s'il est possible de ne calculer qu'un sous-ensemble adaptatif des structures à mettre à jour et d'utiliser ensuite une interpolation pour les éléments qui n'auront pas été calculés. Ceci permettrait d'avoir des simulations moins coûteuses. Néanmoins, le simulateur déterministe réalise énormément de calculs et repose sur un modèle fondamentalement non-linéaire. Le système est sensible à de petites variations des structures de données utilisées (*cf* la différence constatée entre calculs en simple et double précision). En conséquence, cette voie de recherche sera donc à explorer avec prudence.

Le modèle hôte-macroparasite auquel nous avons contribué peut s'adapter à de nombreux systèmes réels. On peut envisager, par exemple, de s'en servir pour décrire la propagation de certaine pathologie dans les vignes. L'exploration du modèle est commencée ; elle devrait se poursuivre par une validation en vraie grandeur et sur le terrain pour le système Bar-*Diplectanum aequans*. Elle conduira à la mise au point de méthodes de prophylaxie visant à préserver partiellement les bars d'élevage du parasitisme. D'autre part, une analyse statistique des données observées et des sorties de la simulation serait très instructive pour approfondir notre compréhension des mécanismes actifs au sein de la dynamique.

La comparaison des méthodes stochastique et déterministe est informative car elle permet de cerner certains comportements non-linéaires d'un système. Dans notre cas, elle a souligné l'impact de la stochasticité démographique en exhibant des comportements stochastique et déterministe différents. Elle donne une information précise sur le rôle que joue la variabilité inter-individuelle. Il serait très intéressant d'effectuer des études quantitatives fines pour d'autres systèmes, et de caractériser dans quels cas cette double approche est nécessaire.

Table des symboles

Annexes

Annexe A

Preuve pour l'équilibrage des charges

Dans cette annexe, on démontre la Proposition suivante : “ *considérons un ensemble de tâches de calcul ayant une certaine fonction de coût ; on distribue ces tâches sur une grille de processeurs grâce à la méthode du serpentin en 2 dimensions. Avec certaines hypothèses concernant la convexité et la décroissance de la fonction de coût des calculs, on est capable de déterminer quel est le processeur de la grille qui a le plus de calculs à effectuer et quel est le coût en calculs de ce processeur.*”
La preuve qui est donnée dans cette annexe correspond à la Proposition 2 énoncée p. 88. On renvoie à la Section III-6.3 pour toutes les notations utilisées ici.

A.1 Complexité α associée à chaque processeur

A.1.1 Introduction

Soit $CM(c,i)$ la complexité d'une tâche élementaire $M(c,i)$ de calcul ($i \in [0..S]$, $c \in |i..S]$). La variable S est donnée, et elle correspond à la borne supérieure des indices i et c. On a une grille carrée de processeurs de largeur Y ; on a donc Y^2 processeurs notés $p_{e,u}$. Soit $\alpha(e,u)$ la complexité en nombre d'opérations des calculs réalisés sur le processeur $p_{e,u}$. Les processeurs $(p_{e0,u})_{u\in 1..Y}$ appartiennent à un groupe de processeurs que l'on appelle h_{e_0}.

Dans une première étape, on distribue selon la méthode du serpentin les indices c parmi les groupes de processeurs h_e. Soit $\kappa(e,i)$ la complexité en calcul pour traiter les tâches $M(c,i)$ effectuées au sein du groupe de processeurs h_e.

Dans un second temps, on distribue en serpentin les indices i aux processeurs au sein même d'un groupe h_e. La division euclidienne de S par $2Y+1$ a pour résultat $q+1$ et pour reste r (on suppose ici que $q > 0$). On utilise dans la suite le prédicat δ avec $\delta(cond)$ qui vaut 1 si *cond* est vrai, 0 sinon. Le coût des calculs par processeur est alors avec les deux distributions en serpentin emboîtées :

$$\alpha(e,u) = \left[\sum_{j=1}^{q} \underbrace{\kappa(e,2Y(j-1)+u)}_{\text{terme 1}} + \underbrace{\kappa(e,2Yj-u-1)}_{\text{terme 2}}\right] + \delta(r \geq u)\ \underbrace{\kappa(e,2Yq+u)}_{\text{terme 3}}$$
$$+\ \delta(r \geq 2Y-u-1)\ \underbrace{\kappa(e,2Y(q+1)-u-1)}_{\text{terme 4}}$$

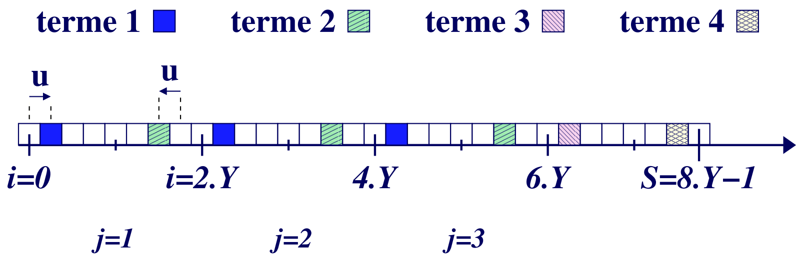


Figure 120. Les indices i distribués en serpentin

La Figure 120 illustre la deuxième distribution en serpentin des indices i, ainsi que les quatres catégories de termes numérotés de 1 à 4 que l'on trouve dans la formule. On va chercher à expliciter $\alpha(e,u)$ en fonction des coûts élémentaires $CM(c,i)$. Pour cela, on exprime $\kappa(e,i)$ en fonction des coûts $CM(c,i)$ dans la formule précédente. On souhaite réaliser le serpentin en prenant les indices c dans le sens décroissant en partant de S vers i (nous en avons vu la raison dans la sous Section III-6.2.2 p. 73). On effectue donc le serpentin sur la variable $c' = S-c$ qui varie de 0 à $S-i$ pour calculer les termes 1, 2, 3 et 4 :

- Terme 1 : Calculons $\kappa(e,2Y(j-1)+u)$, en posant tout d'abord $i_1 = 2Y(j-1)+u$; pour cela on commence par effectuer une division euclidienne de $S-i_1$ par $2Y$: $S-(2Y(j-1)+u) = 2Yq_1 + r_1$.

$$\kappa(e,i_1) = \left[\sum_{g=1}^{q_1} CM(S-(2Y(g-1)+e),i_1) + CM(S-(2Yg-e-1),i_1)\right]$$
$$+\ \delta(r_1 \geq e)\, CM(S-(2Yq_1+e),i_1)$$
$$+\ \delta(r_1 \geq 2Y-e-1)\, CM(S-(2Y(q_1+1)-e-1),i_1)\ .$$

  On a $S-i_1 = 2Yq+r-(2Y(j-1)+u) = 2Yq_1+r_1$, d'où $2Y(q-j+1)+(r-u) = 2Yq_1+r_1$. On peut distinguer deux sous cas :
  - $r \geq u$, alors $q_1 = q-j+1$ et $r_1 = r-u$;
  - $r < u$, alors $q_1 = q-j$ et $r_1 = 2Y-r+u$;

- Terme 2 : Calculons $\kappa(e,2Yj-u-1)$, avec $i_2 = 2Yj-u-1$; effectuons une division euclidienne de $S-i_2$ par $2Y$: $S-(2Yj-u-1) = 2Yq_2 + r_2$.

$$\begin{aligned}\kappa(e,i_2) = &\left[\sum_{g=1}^{q_2} CM(S-(2Y(g-1)+e),i_2) + CM(S-(2Yg-e-1),i_2)\right]\\ &+ \delta(r_2 \geq e)\, CM(S-(2Yq_2+e),i_2)\\ &+ \delta(r_2 \geq 2Y-e-1)\, CM(S-(2Y(q_2+1)-e-1),i_2)\ .\end{aligned}$$

On peut distinguer deux sous cas :

- $r+u+1 \geq 2Y$, alors $q_2 = q-j+1$ et $r_2 = r+u+1-2Y$;
- $r+u+1 < 2Y$, alors $q_2 = q-j$ et $r_2 = r+u+1$;

– Terme 3 : Calculons $\kappa(e, 2Yq+u)$, avec $i_3 = 2Yq+u$; effectuons une division euclidienne de $S-i_3$ par $2Y$: $S-(2Yq+u) = 2Yq_3 + r_3$. On en déduit que $q_3 = 0$ et $S-i_3 = r_3 = r-u$.

$$\begin{aligned}\kappa(e,i_3) =&\delta(r-u \geq e)\, CM(S-e,i_3)+\\ &\delta(r-u \geq 2Y-e-1)\, CM(S-(2Y-e-1),i_3)\ .\end{aligned}$$

– Terme 4 : Calculons $\kappa(e, 2Y(q+1)-u-1)$ avec $i_4 = 2Y(q+1)-u-1$; effectuons une division euclidienne de $S-i_4$ par $2Y$: $S-(2Y(q+1)-u-1) = 2Yq_4+r_4$. On en déduit que $q_4 = 0$ et $S-i_4 = r_4 = r+u+1-2Y$.

$$\begin{aligned}\kappa(e,i_4) =&\delta(r+u+1-2Y \geq e)\, CM(S-e,i_4)+\\ &\delta(r+u+1-2Y \geq 2Y-e-1)\, CM(S-(2Y-e-1),i_4)\ .\end{aligned}$$

A.1.2 Hypothèse simplificatrice

Réécrivons les 4 termes en tenant compte de l'hypothèse suivante : $r = 2Y-1$ ce qui revient à dire que $S+1$ est un multiple de $2Y$, ou bien encore que $S+1 = 2Y(q+1)$.

– Terme 1 : On a la propriété $r \geq u$ vraie donc $q_1 = q-j+1$ et $r_1 = r-u$:

$$\begin{aligned}\kappa(e,2Y(j-1)+u) = &\left[\sum_{g=1}^{q-j+1} CM(S-(2Y(g-1)+e),2Y(j-1)+u) + CM(S-(2Yg-e-1),2Y(j-1)+u)\right]\\ &+ \delta(r-u \geq e)\, CM(S-(2Y(q-j+1)+e),2Y(j-1)+u)\\ &+ \delta(r-u \geq 2Y-e-1)\, CM(S-(2Y(q-j+2)-e-1),2Y(j-1)+u)\ .\end{aligned}$$

– Terme 2 : On a la propriété $r+u+1 \geq 2Y$ vraie d'où $q_2 = q-j+1$ et $r_2 = r+u+1-2Y$.

$$\begin{aligned}\kappa(e,2Yj-u-1) = &\left[\sum_{g=1}^{q-j+1} CM(S-(2Y(g-1)+e),2Yj-u-1) + CM(S-(2Yg-e-1),2Yj-u-1)\right]\\ &+ \delta(r+u+1-2Y \geq e)\, CM(S-(2Y(q-j+1)+e),2Yj-u-1)\\ &+ \delta(r+u+1-2Y \geq 2Y-e-1)\, CM(S-(2Y(q-j+2)-e-1),2Yj-u-1)\ .\end{aligned}$$

– Terme 3 et Terme 4 :

$$\begin{aligned}
\kappa(e, 2Yq+u) = {} & \delta(r-u \geq e)\, CM(S-e, 2Yq+u) + \\
& \delta(r-u \geq 2Y-e-1)\, CM(S-(2Y-e-1), 2Yq+u) \\
\kappa(e, 2Y(q+1)-u-1) = {} & \delta(r+u+1-2Y \geq e)\, CM(S-e, 2Y(q+1)-u-1) + \\
& \delta(r+u+1-2Y \geq 2Y-e-1)\, CM(S-(2Y-e-1), 2Y(q+1)-u-1)\ .
\end{aligned}$$

On tire de l'ensemble de ces calculs l'expression suivante :

$$\begin{aligned}
\alpha(e,u) = \sum_{j=1}^{q} \Bigg[\sum_{g=1}^{q-j+1} \Bigg[& CM(S-(2Y(g-1)+e), 2Y(j-1)+u) + CM(S-(2Yg-e-1), 2Y(j-1)+u) \\
& CM(S-(2Y(g-1)+e), 2Yj-u-1) + CM(S-(2Yg-e-1), 2Yj-u-1) \Bigg] \\
& + \delta(r-u \geq e)\, CM(S-(2Y(q-j+1)+e), 2Y(j-1)+u) \\
& + \delta(r-u \geq 2Y-e-1)\, CM(S-(2Y(q-j+2)-e-1), 2Y(j-1)+u) \\
& + \delta(r+u+1-2Y \geq e)\, CM(S-(2Y(q-j+1)+e), 2Yj-u-1) \\
& + \delta(r+u+1-2Y \geq 2Y-e-1)\, CM(S-(2Y(q-j+2)-e-1), 2Yj-u-1) \Bigg] \\
& + \delta(r-u \geq e)\, CM(S-e, 2Yq+u) \\
& + \delta(r-u \geq 2Y-e-1)\, CM(S-(2Y-e-1), 2Yq+u) \\
& + \delta(r+u+1-2Y \geq e)\, CM(S-e, 2Y(q+1)-u-1) \\
& + \delta(r+u+1-2Y \geq 2Y-e-1)\, CM(S-(2Y-e-1), 2Y(q+1)-u-1)\ .
\end{aligned}$$

A.1.3 Simplification de la formule

On a une grille de processeurs de largeur et hauteur Y, indicée par les variables u et e. On a alors les relations

$$0 \leq u < Y-1 \tag{43}$$

$$0 \leq e < Y-1\ . \tag{44}$$

– On déduit de (43) l'inéquation $2Y-1-u \geq Y$. D'autre part on a la relation (44) et l'égalité $r = 2Y-1$. En combinant toutes ces relations, il vient $r-u \geq e$ et on a donc finalement $\delta(r-u \geq e) = 1$.

– On a $r+u+1-2Y = u \leq Y-1$ et l'on déduit de (44) $2Y-1-e \geq Y$. On tire $r+u+1-2Y < 2Y-1-e$, ce qui implique $\delta(r+u+1-2Y \geq 2Y-e-1) = 0$.

On a alors l'expression de α légèrement simplifiée suivante (on a aussi remplacé r par sa valeur $2Y-1$):

$$\begin{aligned}\alpha(e,u) = \sum_{j=1}^{q}\Bigg[\sum_{g=1}^{q-j+1}\Bigg[& \underbrace{CM(S-(2Y(g-1)+e),2Y(j-1)+u)}_{\mathsf{A}} + \underbrace{CM(S-(2Yg-e-1),2Y(j-1)+u)}_{\mathsf{B}} \\ & \underbrace{CM(S-(2Y(g-1)+e),2Yj-u-1)}_{\mathsf{C}} + \underbrace{CM(S-(2Yg-e-1),2Yj-u-1)}_{\mathsf{D}}\Bigg] \\ & + \underbrace{CM(S-(2Y(q-j+1)+e),2Y(j-1)+u)}_{\mathsf{E}} \\ & + \delta(e \geq u)\ \underbrace{CM(S-(2Y(q-j+2)-e-1),2Y(j-1)+u)}_{\mathsf{F}} \\ & + \delta(u \geq e)\ \underbrace{CM(S-(2Y(q-j+1)+e),2Yj-u-1)}_{\mathsf{G}}\Bigg] \\ & + \underbrace{CM(S-e,2Yq+u)}_{\mathsf{H}} \\ & + \delta(e \geq u)\ \underbrace{CM(S-(2Y-e-1),2Yq+u))}_{\mathsf{I}} \\ & + \delta(u \geq e)\ \underbrace{CM(S-e,2Y(q+1)-u-1)}_{\mathsf{J}}\ .\end{aligned}$$

Considérons cette charge du processeur $p_{e,u}$. Le coût $\alpha(e,u)$ se décompose en une somme de termes qui dépendent des indices g et j. On identifie chaque terme par une lettre comprise entre A et J. Chaque terme correspond au coût $CM(c,i)$ d'une tâche $M(c,i)$ affectée au processeur $p_{e,u}$. On a représenté chacun des termes $CM(.,.)$ de l'expression donnant $\alpha(e,u)$ sur la Figure 121 en reportant son identifiant lettre.

Dans la double somme de $\alpha(e,u)$ prenons un couple d'indices (g,j) possible ; si l'on considère maintenant les termes A, B, C, D pour chacun des P processeurs $p_{e,u}$ pour (g,j) fixés, on obtient un ensemble de termes $CM(c,i)$ pour lesquels :

$$(c,i) \in [S-(2Yg-1), S-2Y(g-1)] \times [2Y(j-1), 2Yj-1]\ .$$

On appellera *bloc extra-diagonal* cet ensemble de termes. Ce bloc contient exactement $2Y \times 2Y = 4P$ termes. Un bloc extra-diagonal pour $j \in [1,q]$ et $g \in [1, q-j+1]$ fixés est désigné par $ED(g,j)$. Au sein d'un bloc $ED(g,j)$, on note le groupement de termes A,B,C,D correspondant au processeur $p_{e,u}$ par $ED_{e,u}(g,j)$. On a donc

$$\begin{aligned}ED_{e,u}(g,j) = \ & \underbrace{CM(S-(2Y(g-1)+e),2Y(j-1)+u)}_{\mathsf{A}} + \underbrace{CM(S-(2Yg-e-1),2Y(j-1)+u)}_{\mathsf{B}} \\ + \ & \underbrace{CM(S-(2Y(g-1)+e),2Yj-u-1)}_{\mathsf{C}} + \underbrace{CM(S-(2Yg-e-1),2Yj-u-1)}_{\mathsf{D}}\ .\end{aligned}$$

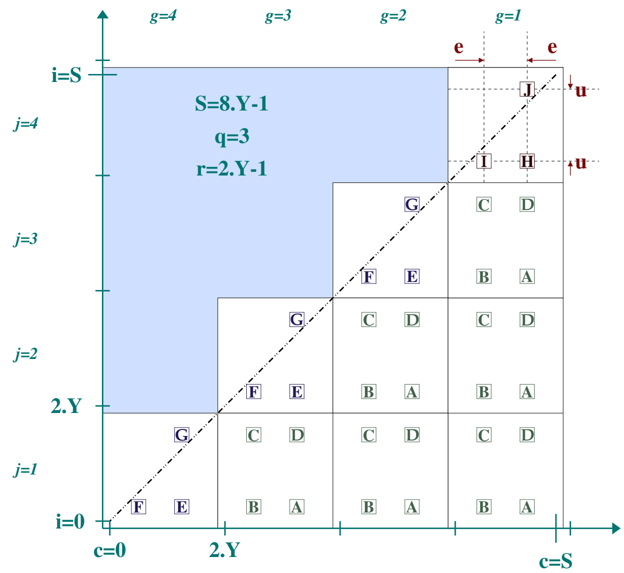


Figure 121. Termes $CM(c,i)$ correspondant aux tâches effectuées par le processeur $p_{e,u}$ et apparaissant dans la formule de $\alpha(e,u)$. Chaque terme $CM(.,.)$ est noté par une lettre comprise entre A et J

Dans chaque bloc extra-diagonal (représenté par un carré blanc de côté de taille $2Y$ sur le schéma), cela veut dire qu'il y a quatre tâches affectées au processeur $p_{e,u}$. Sur le schéma les termes A, B, C, D des groupements $ED_{e,u}(.,.)$ sont représentés pour un processeur donné $p_{e,u}$.

Un *bloc diagonal* $BD(j)$ correspond par définition aux termes E,F,G, pour j fixé et lorsque l'on considère tous les processeur $p_{e,u}$. On définit $BD_{e,u}(j)$ de la manière suivante :

$$\begin{aligned} BD_{e,u}(j) = \quad & CM(S-(2Y(q-j+1)+e), 2Y(j-1)+u) \\ & + \delta(e \geq u)\, CM(S-(2Y(q-j+2)-e-1), 2Y(j-1)+u) \\ & + \delta(u \geq e)\, CM(S-(2Y(q-j+1)+e), 2Yj-u-1)\ . \end{aligned}$$

Les termes E,F,G du processeur $p_{e,u}$ et du bloc diagonal $BD(j)$ correspondent à $BD_{e,u}(j)$. On remarque que les termes H,I,J du processeur $p_{e,u}$ correspondent en fait à $BD_{e,u}(q+1)$.

Avec ces nouvelles notations, on a la formule simplifiée suivante :

$$\alpha(e,u) = \left[\sum_{j=1}^{q} \sum_{g=1}^{q-j+1} ED_{e,u}(g,j) \right] + \sum_{j=1}^{q+1} BD_{e,u}(j). \tag{45}$$

A.2 Charge plus importante sur le processeur $p_{0,0}$

Dans cette partie, on montre maintenant que le processeur $p_{0,0}$ est le processeur ayant le plus de calculs à réaliser. Ceci est prouver d'abord au niveau des blocs extra-diagonaux puis au niveau des blocs diagonaux introduits ci-dessus.

A.2.1 Preuve pour les blocs extra-diagonaux

Proposition sur les fonctions convexes décroissantes

Proposition 3. *Soit f une fonction positive décroissante et convexe sur $[0, 2Y-1]$. Soit $\beta(v)$ la fonction définit sur $[0, Y-1]$ par :*

$$\beta(v) = f(v) + f(2Y - 1 - v) \ .$$

La fonction β est décroissante.

Preuve : On souhaite prouver que β est décroissante sur son intervalle de définition $[0, Y-1]$. Soient les variables suivantes $(a, b, c, d) \in [0, 2Y-1]^4$ pour lesquelles on a la relation $0 \leq a < b < c < d < 2Y$. Comme f est convexe et $a < b < c$, on a

$$\frac{f(a) - f(b)}{a - b} \leq \frac{f(c) - f(b)}{c - b} \ .$$

D'autre part, comme $b < c < d$, on a aussi :

$$\frac{f(b) - f(c)}{b - c} \leq \frac{f(d) - f(c)}{d - c} \ .$$

De ces deux inégalités, on tire :

$$\frac{f(a) - f(b)}{a - b} \leq \frac{f(d) - f(c)}{d - c} \ . \tag{46}$$

On remplace $a = v$, $b = v + 1$, $c = 2Y - 2 - v$ et $d = 2Y - 1 - v$ avec $v \in [0, Y-2]$, dans l'inéquation (46) :

$$\frac{f(v) - f(v+1)}{-1} \leq \frac{f(2Y-1-v) - f(2Y-2-v)}{1} \ ,$$

$$f(v+1) - f(v) \leq f(2Y-1-v) - f(2Y-2-v) \ ,$$

$$f(v+1) + f(2Y-2-v) \leq f(2Y-1-v) + f(v) \ ,$$

Cela peut être réécrit sous la forme

$$\beta(v+1) \leq \beta(v) \ .$$

On obtient donc que $\beta(v)$ est décroissante sur $v \in [0, Y-1]$. □

Décroissance en e de $ED_{e,u}(g,j)$

Pour j,g donnés (Equation 45), soient

$$\begin{aligned}\beta_u^1(e) &= CM(S-(2Y(g-1)+e),2Y(j-1)+u)+CM(S-(2Yg-e-1),2Y(j-1)+u) \quad ,\\ \beta_u^2(e) &= CM(S-(2Y(g-1)+e),2Yj-u-1) + CM(S-(2Yg-e-1),2Yj-u-1) \quad .\end{aligned}$$

La fonction $ED_{e,u}(g,j)$ est la somme de ces deux fonctions $\beta_u^1(e)$ et $\beta_u^2(e)$. Le coût de calcul $CM(S-c,i)$ est décroissant en c sur $[0,S]$, et il est aussi convexe en c. De la même manière, la fonction $CM(S-2Y(g-1)-e,2Y(j-1)+u)$ est décroissante convexe pour $e\in[0,2Y-1]$. En appliquant la Proposition 3 à la fonction $\beta_u^1(e)$, on déduit que $\beta_u^1(e)$ est décroissante en $e\in[0,Y-1]$. De façon identique, $CM(S-2Y(g-1))-e,2Yj-u-1)$ est décroissante convexe sur $e\in[0,2Y-1]$. En appliquant à nouveau la Proposition 3 à la fonction $\beta_u^2(e)$, on déduit que $\beta_u^2(e)$ est décroissante en $e\in[0,Y-1]$. $ED_{e,u}(g,j)$ est la somme des deux fonctions $\beta_u^1(e)$ et $\beta_u^2(e)$ décroissantes sur $e\in[0,Y-1]$, elle est donc elle-même décroissante sur $e\in[0,Y-1]$.

Décroissance en u de $ED_{e,u}(g,j)$

On fixe j et g. Soient les fonctions définies comme suit

$$\begin{aligned}\beta_e^3(u) &= CM(S-(2Y(g-1)+e),2Y(j-1)+u)+CM(S-(2Y(g-1)+e),2Yj-u-1) \quad ,\\ \beta_e^4(u) &= CM(S-(2Yg-e-1),2Y(j-1)+u)+CM(S-(2Yg-e-1),2Yj-u-1) \quad .\end{aligned}$$

La fonction $ED_{e,u}(g,j)$ est la somme de ces deux fonctions $\beta_e^3(u)$ et $\beta_e^4(u)$. La fonction $CM(S-c,i)$ est décroissante en i sur $[0,S]$, et elle est aussi convexe en i. En refaisant une démonstration analogue à la précédente on obtient que $\beta_e^3(u)$ et $\beta_e^4(u)$ sont décroissantes en $u\in[0,Y-1]$. On en déduit que $ED_{e,u}(g,j)$ est somme de fonctions décroissantes en $u\in[0,Y-1]$; elle est donc décroissante sur $[0,Y-1]$.

Sens de variation de $ED_{e,u}(g,j)$

Si l'on fixe S,Y,g et j, on vient donc de montrer que $ED_{e,u}(g,j)$ est décroissante à la fois sur $u\in[0,Y-1]$ et sur $e\in[0,Y-1]$. Le terme présent dans l'expression de $\alpha(e,u)$ comportant $ED_{e,u}(g,j)$ ($\sum_{j=1}^{q}\sum_{g=1}^{q-j+1} ED_{e,u}(g,j)$) est donc lui aussi décroissant à la fois sur $u\in[0,Y-1]$ et sur $e\in[0,Y-1]$. Au niveau des blocs extra-diagonaux, le processeur ayant le plus d'opérations à réaliser est donc le processeur $p_{0,0}$, et celui en ayant le moins est $p_{Y-1,Y-1}$.

A.2.2 Preuve pour les blocs diagonaux

Dans cette Section, on construit des majorations $BD'_{e,u}(j)$ pour les termes $BD_{e,u}(j)$. On prouve ensuite que toutes ces majorations sont inférieures ou égales à $BD_{0,0}(j)$. On sera ainsi amener à conclure que le processeur $p_{0,0}$ a les coûts de calcul les plus importants en ce qui concerne les blocs diagonaux.

Majorant de $BD_{e,u}(g,j)$

Le terme $BD_{e,u}(j)$ est la somme de deux ou trois quantités positives. Il est majoré par $BD'_{e,u}(j)$ défini par :

$$\begin{aligned} BD'_{e,u}(j) = \quad & CM(S-(2Y(q-j+1)+e), 2Y(j-1)+u) \\ + \; & CM(S-(2Y(q-j+2)-e-1), 2Y(j-1)+u) \\ + \; & CM(S-(2Y(q-j+1)+e), 2Yj-u-1) \ . \end{aligned}$$

Décroissance en e de $BD_{e,u}(g,j)$

On fixe j, g, S, Y ; soient

$$\begin{aligned} \zeta_u^1(e) = \; & CM(S-(2Y(q-j+1)+e), 2Y(j-1)+u) \; + \\ & CM(S-(2Y(q-j+2)-e-1), 2Y(j-1)+u) \ , \\ \zeta_u^2(e) = \; & CM(S-(2Y(q-j+1)+e), 2Yj-u-1) \ . \end{aligned}$$

La fonction $BD'_{e,u}(j)$ est égale à la somme de ces deux fonctions $\zeta_u^1(e)$ et $\zeta_u^2(e)$. Pour $g = q-j+2$, on a en fait $\zeta_u^1(e) = \beta_u^1(e)$, d'où l'on tire que $\zeta_u^1(e)$ est décroissante en u. D'autre part, comme $CM(S-c,i)$ est décroissante en c, on a $\zeta_u^2(e)$ décroissante en $e \in [0, Y-1]$. La somme des deux fonctions $BD'_{e,u}(j)$ est donc elle-même décroissante sur $e \in [0, Y-1]$.

Décroissance en u de $BD_{e,u}(g,j)$

On fixe j, g, S, Y ; soient

$$\begin{aligned} \zeta_e^3(u) = \; & CM(S-(2Y(q-j+1)+e), 2Y(j-1)+u) \; + \\ & CM(S-(2Y(q-j+1)+e), 2Yj-u-1) \ , \\ \zeta_e^4(u) = \; & CM(S-(2Y(q-j+2)-e-1), 2Y(j-1)+u) \ . \end{aligned}$$

La fonction $BD'_{e,u}(j)$ est égale à la somme de ces deux fonctions $\zeta_e^3(u)$ et $\zeta_e^4(u)$. Pour $g = q-j+2$, on a en fait $\zeta_u^3(e) = \beta_e^3(u)$. D'où l'on tire que $\zeta_e^3(u)$ est décroissante en e. D'autre part, comme $CM(S-c,i)$ est décroissante en i, on a $\zeta_e^4(u)$ décroissante en $u \in [0, Y-1]$. La somme des deux fonctions $BD'_{e,u}(j)$ est donc elle-même décroissante sur $u \in [0, Y-1]$.

Sens de variation de $BD_{e,u}(j)$

La fonction $BD'_{e,u}(j)$ est donc décroissante sur $u \in [0, Y-1]$ ainsi que sur $e \in [0, Y-1]$. On en tire que pour toute combinaison $(u,e) \in [0, Y-1]^2$, on a l'inéquation $BD'_{e,u}(j) \leq BD'_{0,0}(j)$. Or l'on sait d'une part que $BD_{e,u}(j) \leq BD'_{e,u}(j)$; d'autre part, on vérifie aisément que $BD'_{0,0}(j) = BD_{0,0}(j)$. On en conclut que :

$$\forall (u,e) \in [0, Y-1]^2, \;\; BD_{e,u}(j) \leq BD_{0,0}(j) \ .$$

On a donc aussi

$$\forall (u,e) \in [0, Y-1]^2, \quad \sum_{j=1}^{q+1} BD_{e,u}(j) \leq \sum_{j=1}^{q+1} BD_{0,0}(j) \ .$$

Là encore, le processeur ayant le plus d'opérations à réaliser est le processeur $p_{0,0}$. On peut remarquer que la fonction $BD_{e,u}(j)$ est somme de termes $CM(S-c,i)$ avec c et i très proches l'un de l'autre. Comme on a l'inéquation $c \geq i \geq 0$ et que $CM(S-c,i)$ est décroissante en i et c, cela signifie que la fonction $CM(S-c,i)$ a des valeurs relativement faibles en approchant de la droite $c = i$. En effet, les blocs diagonaux correspondent à des blocs à cheval sur la diagonale d'équation $c = i$ (voir Figure 122). La contribution des termes diagonaux au coût de calcul par processeur $\alpha(e,u)$ sera donc relativement faible par rapport à celle des blocs extra-diagonaux.

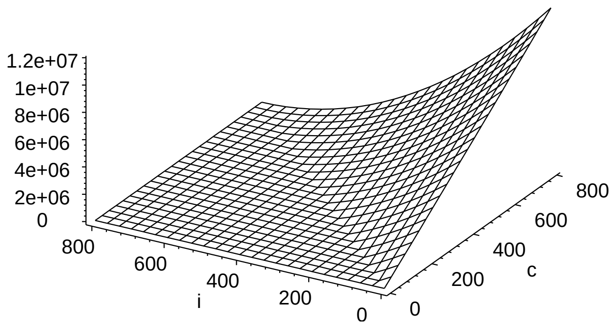


Figure 122. Coût $CM(c,i)$ (pour $S(t) = 800$) donnant le coût en nombre d'opérations pour chaque tâche $M(c,i)$

A.2.3 Conclusion

On a donc prouvé les deux inéquations suivantes :

$$\begin{aligned} &\forall (u,e) \in [0, Y-1]^2, \\ &\sum_{j=1}^{q+1} BD_{e,u}(j) \leq \sum_{j=1}^{q+1} BD_{0,0}(j) \ , \\ &\sum_{j=1}^{q} \sum_{g=1}^{q-j+1} ED_{e,u}(g,j) \leq \sum_{j=1}^{q} \sum_{g=1}^{q-j+1} ED_{0,0}(g,j) \ . \end{aligned}$$

En combinant ces résultats :

$$\begin{aligned} &\forall (u,e) \in [0, Y-1]^2, \\ &\left[\sum_{j=1}^{q} \sum_{g=1}^{q-j+1} ED_{e,u}(g,j)\right] + \sum_{j=1}^{q+1} BD_{e,u}(j) \leq \left[\sum_{j=1}^{q} \sum_{g=1}^{q-j+1} ED_{0,0}(g,j)\right] + \sum_{j=1}^{q+1} BD_{0,0}(j) \end{aligned}$$

d'où :

$$\forall (u,e) \in [0, Y-1]^2, \qquad \alpha(u,e) \leq \alpha(0,0)$$

Le processeur $p_{0,0}$ est donc celui qui le plus d'opérations à réaliser. □

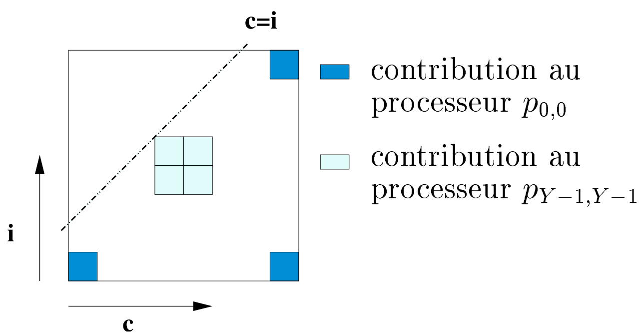
A.3 Extension du résultat pour $S+1 \neq 2\,Y\,q$

En pratique l'hypothèse $S+1 = 2\,Y\,q$ n'est pas toujours vérifiée. La preuve que nous avons développée n'est pas vraie dans de nombreux cas exotiques lorsque $S \ mod \ 2Y \neq 2Y-1$ (quand la racine carré du nombre de processeurs ne divise pas la variable $S+1 = psup(t)+1$).

Contre-exemple Bien que l'on puisse toujours prouver la décroissance en u et e sur les blocs extra-diagonaux, le problème se situe au niveau des blocs diagonaux. Dans ces blocs diagonaux, il existe des exemples pour lesquels le processeur 0 n'est pas celui qui possède le plus grand coût de calcul. Par exemple, soit des coûts en calcul pour le bloc $BD_{u,e}(j_a)$ tels que

$$\forall (u,e) \in [0, Y-1]^2 \quad CM(c_a - e, i_a + u) = z_a > 0 \ .$$

Le coût en calcul est donc constant sur ce bloc diagonal, donc l'hypothèse de décroissance et de convexité de la fonction $CM(S-c, i)$ en c et i est bien respectée. Représentons sur le bloc diagonal les contributions du processeur $p_{0,0}$ et celle du processeur $p_{Y-1,Y-1}$. Dans cet exemple, pour lequel $0 \leq S \ mod \ 2Y < Y$, on a



$$\begin{aligned} BD_{Y-1,Y-1}(j_a) &= 4\,z_a \\ BD_{0,0}(j_a) &= 3\,z_a \\ \text{donc} \quad BD_{0,0}(j_a) &< BD_{Y-1,Y-1}(j_a) \ . \end{aligned}$$

Pour ce contre-exemple, le processeur $p_{0,0}$ n'est donc pas celui qui est le plus chargé en calcul au niveau de ce bloc diagonal.

Généralisation Néanmoins même si pour les blocs diagonaux le processeur $p_{0,0}$ n'est pas celui qui a le plus de calcul :

- les blocs diagonaux correspondent à des tâches de calcul relativement faibles vue la décroissance de $CM(S-c, i)$. Si un processeur a plus de calculs sur ces blocs diagonaux que le processeur $p_{0,0}$, la différence sera relativement faible.

– le nombre de blocs extra-diagonaux est beaucoup plus important pour S grand ($\Theta(S^2)$) que le nombre de blocs diagonaux qui est en $\Theta(S)$. Or sur les blocs extra-diagonaux le processeur $p_{0,0}$ est celui qui a le plus de calculs à effectuer.

Ces deux éléments permettent de prévoir qu'en général le processeur $p_{0,0}$ est bien celui qui aura le plus d'opérations à réaliser. Il faudrait des hypothèses supplémentaires pour généraliser la Proposition, car la preuve faite pour $S \ mod \ 2Y = 2Y - 1$ ne tient pas sans cette condition.

Annexe B

Calculs de complexité

Dans la partie III, on traite une étude de la complexité de l'algorithme de mise à jour de N. Celle-ci requiert un certain nombre de calculs symboliques pour effectuer le comptage du nombre d'additions et de multiplications effectuées par différents algorithmes. Les calculs nécessaires pour cette étude ont été rassemblés dans cette annexe afin que le lecteur puisse comprendre ou reproduire les résultats donnés dans la Section III-6.1 p. 59. On donne une expression simple des polynômes E_*, F_* et D_* qui interviennent dans l'évaluation des complexités des algorithmes séquentiels.

B.1 Les polynômes E_*, F_*, D_*

On suppose que pour un instant t donné, on a $S = p_{sup}(t)$.

B.1.1 Les polynômes E

Calcul de E_1

$$E_1 = \sum_{l=0}^{S} \sum_{i=0}^{l} \sum_{m=i}^{l} \sum_{c=m}^{S} \sum_{d=i}^{c-m+i} 1$$

$$E_1 = \sum_{l=0}^{S} \sum_{i=0}^{l} \sum_{m=i}^{l} \sum_{c=m}^{S} (c - m + 1)$$

$$E_1 = \sum_{l=0}^{S} \sum_{i=0}^{l} \sum_{m=i}^{l} \frac{(S-m)(S-m+1)}{2}$$

$$E_1 = \sum_{l=0}^{S} \sum_{i=0}^{l} \sum_{m=i}^{l} \frac{1}{2}\, m^2 - (S + \frac{3}{2})\, m + (\frac{S^2}{2} + \frac{3\,S}{2} + 1)$$

$$E_1 = \sum_{l=0}^{S} \sum_{i=0}^{l} \frac{1}{2}\, [\frac{(l+1)^3 - i^3}{3} - \frac{(l+1)^2 - i^2}{2} + \frac{(l+1) - i}{6}] - (S + \frac{3}{2})\, [\frac{(l+1)^2 - i^2}{2} - \frac{(l+1) - i}{2}] + (\frac{S^2}{2} + \frac{3\,S}{2} + 1)\,(l - i + 1)$$

$$E_1 = \sum_{l=0}^{S} \sum_{i=0}^{l} \frac{(l+1)^3 - i^3}{6} - (S+2)\frac{(l+1)^2 - i^2}{2} + (\frac{1}{12} + \frac{S}{2} + \frac{3}{4} + \frac{S^2}{2} + \frac{3\,S}{2} + 1)((l+1) - i)$$

$$
\begin{aligned}
E_1 &= \sum_{l=0}^{S}\sum_{i=0}^{l} \frac{(l+1)^3}{6} - (S+2)\frac{(l+1)^2}{2} + (\frac{S^2}{2} + 2\,S + \frac{11}{6})(l+1) \\
&\quad - \sum_{l=0}^{S}\sum_{i=0}^{l} \frac{1}{6}i^3 - \frac{S+2}{2}i^2 + (\frac{S^2}{2} + 2\,S + \frac{11}{6})i
\end{aligned}
$$

$$
\begin{aligned}
E_1 &= \sum_{l=0}^{S} \frac{(l+1)^4}{6} - (S+2)\frac{(l+1)^3}{2} + (\frac{S^2}{2} + 2\,S + \frac{11}{6})(l+1)^2 \\
&- \sum_{l=0}^{S} \frac{1}{6}[\frac{(l+1)^4}{4} - \frac{(l+1)^3}{2} + \frac{(l+1)^2}{4}] - \frac{S+2}{2}[\frac{(l+1)^3}{3} - \frac{(l+1)^2}{2} + \frac{l+1}{6}] \\
&- \sum_{l=0}^{S}(\frac{S^2}{2} + 2\,S + \frac{11}{6})[\frac{(l+1)^2}{2} - \frac{l+1}{2}]
\end{aligned}
$$

$$
\begin{aligned}
E_1 = \sum_{l=0}^{S+1} \quad & \frac{3}{24}\,l^4 + [-\frac{S+2}{2} + \frac{1}{12} + \frac{S+2}{6}]\,l^3 \\
& +[(\frac{S^2}{2} + 2\,S + \frac{11}{6}) - \frac{1}{24} - \frac{S+2}{4} - \frac{1}{2}\,(\frac{S^2}{2} + 2\,S + \frac{11}{6})]\,l^2 \\
& + [\frac{S+2}{12} + \frac{1}{2}\,(\frac{S^2}{2} + 2\,S + \frac{11}{6})]\,l
\end{aligned}
$$

$$
E_1 = \sum_{l=0}^{S+1} \frac{3}{24}\,l^4 - [\frac{S}{3} + \frac{7}{12}]\,l^3 + [\frac{S^2}{4} + \frac{3\,S}{4} + \frac{3}{8}]\,l^2 + [\frac{S^2}{4} + \frac{13\,S}{12} + \frac{13}{12}]\,l
$$

$$
\begin{aligned}
E_1 = \quad & \frac{3}{24}\,[\frac{(S+2)^5}{5} - \frac{(S+2)^4}{2} + \frac{(S+2)^3}{3} - \frac{(S+2)}{30}] \\
& -[\frac{S}{3} + \frac{7}{12}]\,[\frac{(S+2)^4}{4} - \frac{(S+2)^3}{2} + \frac{(S+2)^2}{4}] \\
& +[\frac{S^2}{4} + \frac{3\,S}{4} + \frac{3}{8}]\,[\frac{(S+2)^3}{3} - \frac{(S+2)^2}{2} + \frac{S+2}{6}] \\
& +[\frac{S^2}{4} + \frac{13\,S}{12} + \frac{13}{12}]\,[\frac{(S+2)^2}{2} - \frac{(S+2)}{2}]
\end{aligned}
$$

$$
\begin{aligned}
E_1 = \quad & \frac{3}{120}(S+2)^5 - (\frac{3}{48} + \frac{S}{6} + \frac{7}{48})(S+2)^4 + (\frac{1}{24} + \frac{S}{6} + \frac{7}{24} + \frac{S^2}{12} + \frac{3\,S}{12} + \frac{3}{24})(S+2)^3 \\
& -(\frac{S}{12} + \frac{7}{48} + \frac{S^2}{8} + \frac{3\,S}{8} + \frac{3}{16} - \frac{S^2}{8} - \frac{13\,S}{24} - \frac{13}{24})(S+2)^2 \\
& +(-\frac{1}{240} + \frac{S^2}{24} + \frac{3\,S}{24} + \frac{3}{48} - \frac{S^2}{8} - \frac{13\,S}{24} - \frac{13}{24})(S+2)
\end{aligned}
$$

$$
\begin{aligned}
E_1 = \quad & \frac{1}{40}(S+2)^5 - (\frac{S}{12} + \frac{5}{24})(S+2)^4 + (\frac{S^2}{12} + \frac{5\,S}{12} + \frac{11}{24})(S+2)^3 \\
& +(\frac{S}{12} + \frac{5}{24})(S+2)^2 - (\frac{S^2}{12} + \frac{5\,S}{12} + \frac{29}{60})(S+2)
\end{aligned}
$$

$$
\begin{aligned}
E_1 = \quad & \frac{1}{40}\,(S^5 + 10\,S^4 + 40\,S^3 + 80\,S^2 + 80\,S + 32) \\
& -(\frac{S}{12} + \frac{5}{24})\,(S^4 + 8\,S^3 + 24\,S^2 + 32\,S + 16) \\
& +(\frac{S^2}{12} + \frac{5\,S}{12} + \frac{11}{24})\,(S^3 + 6\,S^2 + 12\,S + 8) \\
& +(\frac{S}{12} + \frac{5}{24})\,(S^2 + 4\,S + 4) \\
& -(\frac{S^2}{12} + \frac{5\,S}{12} + \frac{29}{60})\,(S+2)
\end{aligned}
$$

$$\boxed{E_1 = \frac{1}{40}\,S^5 + \frac{7}{24}\,S^4 + \frac{31}{24}\,S^3 + \frac{65}{24}\,S^2 + \frac{161}{60}\,S + 1}$$

Calcul de E_2

$$E_2 = \sum_{l=0}^{S}\sum_{i=0}^{l}\sum_{m=i}^{l}\sum_{c=m}^{S} 1$$

$$E_2 = \sum_{l=0}^{S}\sum_{i=0}^{l}\sum_{m=i}^{l}(S+1-m)$$

$$E_2 = \sum_{l=0}^{S}\sum_{i=0}^{l}(S+1)\,(l-i+1) - \frac{l(l+1)-i(i-1)}{2}$$

$$E_2 = \sum_{l=0}^{S}\sum_{i=0}^{l} -\frac{l^2}{2} + (S+\frac{1}{2})\,l + S + 1 + \frac{i^2}{2} + (-S-\frac{3}{2})\,i$$

$$E_2 = \sum_{l=0}^{S}(l+1)\,[-\frac{l^2}{2} + (S+\frac{1}{2})\,l + S + 1] + \frac{1}{2}\,\frac{l(l+1)(2l+1)}{6} + (-S-\frac{3}{2})\,\frac{l(l+1)}{2}$$

$$E_2 = \sum_{l=0}^{S}[-\frac{l^3}{2} + S\,l^2 + (2S+\frac{3}{2})\,l + S + 1] + [\frac{1}{6}\,l^3 + \frac{1}{4}\,l^2 + \frac{1}{12}\,l] + [(-\frac{S}{2}-\frac{3}{4})\,l^2 + (-\frac{S}{2}-\frac{3}{4})\,l]$$

$$E_2 = \sum_{l=0}^{S} -\frac{l^3}{3} + (-\frac{1}{2}+\frac{S}{2})\,l^2 + (\frac{5}{6}+\frac{3}{2}\,S)\,l + S + 1$$

$$\begin{aligned} E_2 = \quad & -\frac{1}{3}\,\frac{S^2\,(S+1)^2}{4} + (-\frac{1}{2}+\frac{S}{2})\,\frac{S\,(S+1)\,(2S+1)}{6} \\ & +(\frac{5}{6}+\frac{3}{2}\,S)\,\frac{S\,(S+1)}{2} + (S+1)\,(S+1) \end{aligned}$$

$$\begin{aligned} E_2 = \quad & -\frac{1}{3}\,[\frac{S^4}{4} + \frac{S^3}{2} + \frac{S^2}{4}] + (-\frac{1}{2}+\frac{S}{2})\,[\frac{S^3}{3} + \frac{S^2}{2} + \frac{S}{6}] \\ & +(\frac{5}{6}+\frac{3}{2}\,S)\,[\frac{S^2}{2}+\frac{S}{2}] + (S^2 + 2\,S + 1) \end{aligned}$$

$$\boxed{E_2 = \frac{1}{12}\,S^4 + \frac{2}{3}\,S^3 + \frac{23}{12}\,S^2 + \frac{7}{3}\,S + 1}$$

Calcul de E_3

$$E_3 = \sum_{l=0}^{S}\sum_{i=0}^{l}\sum_{m=i}^{l} 1$$

$$E_3 = \sum_{l=0}^{S}\sum_{i=0}^{l} l - i + 1$$

$$E_3 = \sum_{l=0}^{S}(l+1)(l+1) - \frac{l(l+1)}{2}$$

$$E_3 = \sum_{l=0}^{S}\frac{l^2}{2} + \frac{3\,l}{2} + 1$$

$$E_3 = \frac{1}{2}\,\frac{S\,(S+1)\,(2S+1)}{6} + \frac{3}{2}\,\frac{S\,(S+1)}{2} + S + 1$$

$$E_3 = \frac{1}{2}\,[\frac{S^3}{3} + \frac{S^2}{2} + \frac{S}{6}] + \frac{3}{2}\,[\frac{S^2}{2} + \frac{S}{2}] + S + 1$$

$$\boxed{E_3 = \frac{1}{6}\,S^3 + S^2 + \frac{11}{6}\,S + 1}$$

Calcul de E'_3

$$E'_3 = \sum_{l=0}^{S}\sum_{i=0}^{l}\sum_{c=i}^{S} 1$$

$$E'_3 = \sum_{l=0}^{S}\sum_{i=0}^{l} S - i + 1$$

$$E'_3 = \sum_{l=0}^{S}(S+1)l + (S+1) - \frac{l(l+1)}{2}$$

$$E'_3 = \sum_{l=0}^{S} -\frac{l^2}{2} + (S - \frac{l}{2})\,l + (S+1)$$

$$E'_3 = -\frac{1}{2}\,[\frac{1}{3}\,S^3 + \frac{1}{2}\,S^2 + \frac{1}{6}\,S] + (S + \frac{l}{2})\,(\frac{1}{2}\,S^2 + \frac{1}{2}\,S) + (S+1)\,(S+1)$$

$$E'_3 = -\frac{1}{2}\,[\frac{1}{3}\,S^3 + \frac{1}{2}\,S^2 + \frac{1}{6}\,S] + [\frac{1}{2}\,S^3 + \frac{3}{4}\,S^2 + \frac{1}{4}\,S] + [S^2 + 2\,S + 1]$$

$$\boxed{E'_3 = \frac{1}{3}\,S^3 + \frac{3}{2}\,S^2 + \frac{13}{6}\,S + 1}$$

Calcul de E_4

$$E_4 = \sum_{l=0}^{S}\sum_{i=0}^{l} 1$$

$$E_4 = \sum_{l=0}^{S} l + 1$$

$$E_4 = \sum_{l=0}^{S+1} l$$

$$E_4 = \frac{(S+1)(S+2)}{2}$$

$$\boxed{E_4 = \frac{1}{2}\,S^2 + \frac{3}{2}\,S + 1}$$

B.1.2 Les polynômes F

Calcul de F_1

$$F_1 = \sum_{k=2}^{K} \sum_{l=0}^{S} \sum_{i=0}^{l} \sum_{m=i}^{l} \sum_{c=m}^{S} \sum_{d=i}^{c-m+i} 1$$

$$F_1 = \sum_{k=2}^{K} E_1$$

$$\boxed{F_1 = (K-1)\,[\frac{1}{40}\,S^5 + \frac{7}{24}\,S^4 + \frac{31}{24}\,S^3 + \frac{65}{24}\,S^2 + \frac{161}{60}\,S + 1]}$$

Calcul de F_2

$$F_2 = \sum_{k=2}^{K} \sum_{l=0}^{S} \sum_{i=0}^{l} \sum_{m=i}^{l} \sum_{c=m}^{S} 1$$

$$F_2 = \sum_{k=2}^{K} E_2$$

$$\boxed{F_2 = (K-1)\,[\frac{1}{12}\,S^4 + \frac{2}{3}\,S^3 + \frac{23}{12}\,S^2 + \frac{7}{3}\,S + 1]}$$

Calcul de F_3

$$F_3 = \sum_{k=2}^{K} \sum_{l=0}^{S} \sum_{i=0}^{l} \sum_{m=i}^{l} 1$$

$$F_3 = \sum_{k=2}^{K} E_3$$

$$\boxed{F_3 = (K-1)\,[\frac{1}{6}\,S^3 + S^2 + \frac{11}{6}\,S + 1]}$$

Calcul de F_3'

$$F_3' = \sum_{k=2}^{K} \sum_{l=0}^{S} \sum_{i=0}^{l} \sum_{c=i}^{S} 1$$

$$F_3' = \sum_{k=2}^{K} E_3'$$

$$\boxed{F_3' = (K-1)\,[\frac{1}{3}\,S^3 + \frac{3}{2}\,S^2 + \frac{13}{6}\,S + 1]}$$

Calcul de F_4

$$F_4 = \sum_{k=2}^{K} \sum_{l=0}^{S} \sum_{i=0}^{l} 1$$

$$F_4 = \sum_{k=2}^{K} E_4$$

$$\boxed{F_4 = (K-1)\,[\frac{1}{2}\,S^2 + \frac{3}{2}\,S + 1]}$$

B.1.3 Les polynômes D

Calcul de D_1

$$D_1 = \sum_{i=0}^{S} \sum_{c=i}^{S} \sum_{m=i}^{c} \sum_{d=i}^{c-m+i} 1$$

$$D_1 = \sum_{i=0}^{S} \sum_{c=i}^{S} \sum_{m=i}^{c} (c-m+1)$$

$$D_1 = \sum_{i=0}^{S} \sum_{c=i}^{S} \frac{(c-i+1)(c-i+2)}{2}$$

$$D_1 = \sum_{i=0}^{S} \sum_{c'=0}^{S-i} \frac{1}{2}\,c'^2 + \frac{3}{2}\,c' + 1$$

$$D_1 = \sum_{i=0}^{S} \frac{1}{2}\,(\frac{1}{3}\,(S-i+1)^3 - \frac{1}{2}\,(S-i+1)^2 + \frac{1}{6}\,(S-i+1)) + \frac{3}{2}\,(\frac{1}{2}\,(S-i+1)^2 - \frac{1}{2}\,(S-i+1)) + (S-i+1)$$

$$D_1 = \sum_{i=0}^{S} \frac{1}{6}\,(S-i+1)^3 + \frac{1}{2}\,(S-i+1)^2 + \frac{1}{3}\,(S-i+1)$$

$$D_1 = \sum_{i'=0}^{S+1} \frac{1}{6}\,i'^3 + \frac{1}{2}\,i'^2 + \frac{1}{3}\,i'$$

$$D_1 = \frac{1}{6}\,[\frac{1}{4}\,(S+2)^4 - \frac{1}{2}\,(S+2)^3 + \frac{1}{4}\,(S+2)^2] + \frac{1}{2}\,[\frac{1}{3}\,(S+2)^3 - \frac{1}{2}\,(S+2)^2 + \frac{1}{6}\,(S+2)] + \frac{1}{3}\,[\frac{1}{2}\,(S+2)^2 - \frac{1}{2}\,(S+2)]$$

$$D_1 = \frac{1}{24}\,(S+2)^4 + \frac{1}{12}\,(S+2)^3 - \frac{1}{24}\,(S+2)^2 - \frac{1}{12}\,(S+2)$$

$$D_1 = \frac{1}{24}\,(S+2)\,[(S+2)^3 + 2\,(S+2)^2 - (S+2) - 2]$$

$$D_1 = \frac{1}{24}\,(S+2)\,[(S+2)^3 + 2\,(S+2)^2 - (S+2) - 2]$$

$$D_1 = \frac{1}{24}\,(S+4)\,(S+3)\,(S+2)\,(S+1)$$

$$\boxed{D_1 = \frac{1}{24}\,S^4 + \frac{5}{12}\,S^3 + \frac{35}{24}\,S^2 + \frac{25}{12}\,S + 1}$$

Calcul de D_2

$$D_2 = \sum_{i=0}^{S} \sum_{c=i}^{S} \sum_{m=i}^{c} \sum_{l=m}^{S} 1$$

$$\dots$$

$$\boxed{D_2 = \frac{1}{12}\, S^4 + \frac{2}{3}\, S^3 + \frac{23}{12}\, S^2 + \frac{7}{3}\, S + 1}$$

Calcul de D_3

$$D_3 = \sum_{i=0}^{S} \sum_{c=i}^{S} \sum_{m=i}^{c} 1$$

$$\dots$$

$$\boxed{D_3 = \frac{1}{6}\, S^3 + S^2 + \frac{11}{6}\, S + 1}$$

Calcul de D_4

$$D_4 = \sum_{i=0}^{S} \sum_{c=i}^{S} \sum_{l=i}^{S} 1$$

$$\dots$$

$$\boxed{D_4 = \frac{1}{3}\, S^3 + \frac{3}{2}\, S^2 + \frac{13}{6}\, S + 1}$$

Annexe C

Routines GEMM et TRMM sur architecture superscalaire

On décrit ici l'architecture du processeur Power 3-II intégré dans les IBM SP3 WH2 et NH2. On étudie ensuite un exemple d'optimisation de code (multiplication de matrices pour lequel une des matrices a une petite dimension) sur cette architecture. On donne des implémentations performantes des algorithmes TRMM et GEMM pour l'architecture cible choisie. Cette annexe permet de réaliser des routines dédiées au simulateur déterministe. En effet des problèmes de localité des données dans les caches rendent nécessaires ce type d'approche (voir p. 101). Comme on le verra, la méthode employée permet de réaliser des routines qui sont portables sur les machines superscalaires en général.

C.1 Architecture du Power 3

C.1.1 Traitements des instructions sur le Power 3

Le Power 3-II est un processeur superscalaire qui comprend huit unités d'exécution séparées. Le terme superscalaire désigne un processeur scalaire [42] qui est capable d'exécuter plus d'une instruction par cycle. L'exécution superscalaire est d'abord réalisable grâce à une unité de décodage d'instructions qui peut prendre en compte plusieurs instructions. Ces instructions peuvent ensuite être exécutées simultanément si les unités d'exécution correspondantes sont disponibles. Notons que les unités d'exécution peuvent être pipelinées : plusieurs unités sont chaînées entre elles, les résultats générés par l'une servent d'entrée à l'unité suivante. Si l'on effectue des calculs identiques sur de nombreuses données, après une phase initiale de remplissage du pipeline, la chaîne de traitements peut fournir un

42. Par opposition à un processeur vectoriel, il ne permet pas de traiter des vecteurs de données et ne peut qu'exécuter des instructions dont l'effet se limite à des variables scalaires.

résultat par cycle horloge. Pour utiliser efficacement une telle architecture, on doit disposer d'instructions indépendantes. Il est alors possible de faire travailler différentes unités fonctionnelles en même temps : unité de recherche des instructions, unité de décodage d'instruction, unité de gestion de l'ordonnancement des instructions, unité de chargement/écriture en mémoire, unités de calcul (unité logique, entière ou flottante), unité de branchement. Jusqu'à quatre instructions peuvent se terminer à la fin d'un cycle d'horloge sur le Power 3, grâce au parallélisme d'instructions.

Unités de calcul

Il a trois unités de calcul entier, dont deux permettent l'exécution en un cycle d'une opération arithmétique, logique, de comparaison, ou de décalage. La troisième unité exécute des instructions multi-cycles telle que la multiplication et la division (voir Tableau C.5).

Instruction	nombre de cycles	
	32 bits	64 bits
multiplication entière	3-4	3-9
division entière	21	37
multiplication + addition flottante	3-4	3-4
division flottante	14-21	18-25
racine carré	14-23	22-31

Tableau 5. Nombre de cycles pour plusieurs types d'instructions

Les deux unités flottantes du processeur sont indépendantes et multi-cycles. Elles peuvent être utilisées en mode pipeline et réaliser des opérations d'addition, de multiplication, ou deux opérations d'addition et multiplication du type `fmadd` : $c \leftarrow c + a \times b$. Si le pipeline est plein, ce mode donne par exemple deux résultats d'opérations de type `fmadd` à chaque cycle. Le temps d'initialisation du pipeline est de 3 à 4 cycles pour ce type d'opération (voir Tableau C.5). Le processeur peut fournir les résultats de deux `fmadd` par cycle après initialisation du pipeline ; la performance crête en calcul flottant est alors atteinte et consiste en quatre opérations élémentaires par cycle (deux additions et deux multiplications). La performance en MFLOPS (1500 MFLOPS) est donc égale à quatre fois la fréquence horloge en Mhz du processeur qui est de 375 Mhz.

Autres unités

Deux unités de chargement/enregistrement permettent le transfert des données depuis la mémoire vers les registres. Une file d'enregistrement de 16 entrées fait office de mémoire tampon pour empêcher les écritures en attente de bloquer le processeur. Des chargements peuvent ainsi avoir lieu en attendant l'écriture effective d'un résultat en mémoire. Le Power 3 dispose d'instructions de chargement/mise à jour, qui permettent de charger une donnée et de mettre un pointeur à jour simultanément (instruction `lfdu`). On dispose d'instructions qui permettent donc à la fois de charger et d'écrire des données.

Une autre instruction permet d'écrire une donnée en mettant à jour un pointeur sur la donnée (instruction `stfdu`). Le compilateur a donc la possibilité de générer du code concis. Voyons un exemple pratique, celui de la multiplication matrice-vecteur (dimension de la matrice $dn \times dn$, du vecteur dn). Le nombre de références mémoire est en $o(dn^2)$ et la complexité en $o(2dn^2)$ opérations. En moyenne pour cet algorithme[43], on effectue un chargement pour deux opérations flottantes. Dans un code optimisé, la boucle interne comporte deux instructions indépendantes de chargement/mise à jour (`lfdu`) pour charger deux données et remettre à jour les pointeurs sur ces données et deux instructions `fmadd`. Les quatre instructions peuvent être exécutées simultanément par le processeur. Si les références mémoire sont déjà présentes dans le cache L1 et sont chargées efficacement, cela conduit à une performance proche de la crête pour dn grand (il faut prendre en compte le coût de gestion de la boucle). En effet, les deux unités flottantes sont occupées en permanence et n'attendent jamais de données. Enfin, le Power 3 autorise en un cycle le chargement de deux références mémoire de huit octets vers deux registres, ou bien une écriture d'un registre vers le cache L1. On notera donc l'importance de réaliser des opérations utilisant au maximum les registres, car les échanges avec le cache L1 sont coûteux, surtout les écritures.

Une unité de contrôle est responsable de la prise en charge des instructions pour allocation aux unités de calcul et à l'unité de branchement. Pour les branchements, la technique de prédiction dynamique est utilisée à l'aide d'un prédicteur à deux bits. Le processeur conserve ainsi un historique du succès des branchements, indexé par l'adresse des branchements. Deux bits sont associés à une adresse de branchement pour effectuer la prédiction, la valeur 00 pour ces deux bits correspond à branchement peu pris et 11 à branchement très utilisé. Ce compteur est incrémenté *a posteriori* si le branchement est pris, et décrémenté dans le cas inverse.

Le Power 3 est un processeur spéculatif ; l'ordonnancement des instructions y est dynamique. Le processeur peut organiser les instructions à exécuter pour maximiser le nombre d'opérations réalisées en parallèle par les unités fonctionnelles. Le processeur peut aussi minimiser les délais d'attente aux données en exécutant en premier les instructions qui ont des données prêtes. Jusqu'à quatre instructions peuvent se terminer à chaque cycle. Les instructions en cours de traitement ont la possibilité de s'exécuter dans le désordre (*out of order*), mais le processeur garantit que le résultat de ces instructions est identique à celui de l'ordre initial. Des registres de renommage sont utilisés au cours de l'exécution spéculative pour éliminer certaines dépendances entre instructions. L'exécution dans le désordre[44] permet au processeur de continuer à calculer même s'il y a des défauts de cache pour certaines instructions et qu'il attend des références mémoire. Le processeur comporte 32 registres entiers (64 bits) et 32 registres flottants, ainsi que 40 registres de renommage (16 entiers, 24 flottants). En définitive, huit

43. Correspond à l'algorithme de GEMV des BLAS 2.

44. L'éxécution dans le désordre est très employée pour les processeurs superscalaires actuels, tels que les : Compaq Alpha EV67/68, IBM Power 3/4, MIPS R14000, AMD Athlon XP, Intel Pentium 4, etc

unités d'exécution peuvent travailler en parallèle pendant un cycle horloge : 3 unités de calcul entier, 2 unités flottantes, 2 unités de chargement/écriture, une unité de contrôle.

Un mécanisme qui est apparu sur plusieurs architectures[45] consiste en un préchargement automatique (*prefetch* matériel). Il est effectué pour quatre flots de données au maximum sur le Power 3. Un flot est défini comme une suite d'accès séquentiels en mémoire, détectée par le processeur. Sur le Power 3, ce mécanisme peut maintenir une avance de deux lignes de cache disponibles (256 octets) pour chaque flot. La première de ces lignes est préchargée dans le cache L1, la seconde ligne est stockée dans une zone tampon spéciale. Lorsque le flot atteint une nouvelle ligne de cache, la zone tampon spéciale est copiée dans le L1, puis la ligne suivante est chargée, etc. Lorsque l'on effectue des accès séquentiels aux données dans une routine, ce préchargement matériel au niveau du processeur est très efficace. Ce mécanisme automatique permet de recouvrir notoirement des latences d'accès au cache L2 et à la mémoire. On verra un peu plus loin dans le document que le recouvrement de la latence peut éventuellement être réalisé par la technique de préchargement logiciel ou de pipeline logiciel, moyennant un coût de programmation supplémentaire.

C.1.2 La mémoire

Le cache L1

Pour le cache, on utilise des mémoires très rapides dont la fréquence de fonctionnement est si possible peu éloignée de celle du processeur. Elles sont de faible capacité, pour des questions de coût économique et de faisabilité technique. On accède aux données contenues dans le cache par ligne ; lorsque la ligne est déjà présente on parle de *cache hit*, lorsqu'elle est absente de *cache miss* (défaut de cache). Lors d'un défaut de cache, il faut un certain temps (latence) pour que la ligne soit reçue par le cache. Un des intérêts architecturaux d'une exécution dans le désordre est de ne pas bloquer le processeur lors d'un défaut de cache, c'est-à-dire que le processeur peut exécuter d'autres instructions indépendantes de celle qui a provoquée le défaut en attendant que les données soient disponibles. Lorsqu'une nouvelle ligne arrive dans le cache, où la place-t-on ? La politique de remplacement des lignes varie selon l'architecture. Avec un cache à correspondance directe, il n'y a qu'une seule place possible pour charger une ligne mémoire demandée dans le cache que l'on obtient en tronquant l'adresse. Les caches totalement associatifs autorisent le chargement d'une ligne à n'importe quelle place. Un cache est dit "n-way set associative" si le cache contient x ensembles de n lignes ; l'adresse d'une ligne détermine l'ensemble où elle est affectée, elle peut ensuite être placée parmi les n places de cet ensemble. Le remplacement au sein de l'ensemble pourra se faire avec un algorithme LRU (*least recently used*) ou FIFO ou encore au hasard. Le taux d'échec (*miss ratio*) du cache correspond au pourcentage de références non trouvées dans le cache lors d'une demande de

45. Le *hardware prefetch* est utilisé par exemple dans les processeurs AMD Athlon XP, Intel Pentium 4

donnée. Le taux d'échec diminue lorsque la taille du cache augmente, ou lorsque le degré d'associativité n augmente.

L'IBM SP3 dispose de deux niveaux de cache L1 et L2. Le cache L1 est de 64 KO sur WH2 et 128 KO sur NH2. Il est de type 128-way set associative, le remplacement au sein d'une ligne se faisant avec l'algorithme FIFO, qui est beaucoup moins efficace que l'algorithme LRU. Une ligne fait 128 octets, un ensemble est composé de 128 lignes (soit 16 KO) ; il y a donc 4 ensembles sur WH2 et 8 ensembles sur NH2. Le cache peut traiter simultanément quatres demandes de lignes sans blocage, la cinquième est donc bloquante. Les demandes concernant des données présentes dans le cache L1 peuvent être satisfaites en un cycle horloge. Deux fois 8 octets peuvent être transférés en un cycle du L1 vers deux registres du processeur. La bande passante depuis le L1 est donc de 16 O $\times$ 375 Mhz $=$ 6 GO/s. Tenir compte de cette limitation est essentiel dans les applications haute-performance. En calcul flottant par exemple, le processeur peut réaliser deux additions/multiplications (`fmadd`) en moyenne par cycle. En utilisant uniquement deux chargements de flottants par cycle (2×8 octets), il est très difficile de fournir des données assez rapidement pour les deux `fmadd` (nécessitant 8 opérandes). Néanmoins, dans certain cas, la routine BLAS DGEMM atteint pratiquement les performances crête du processeur. Dans ce cas particulier, le processeur réalise en moyenne et en un cycle, 2 opérations `fmadd`: $r_1 = a_1\, x_1 + r_1$ et $r_2 = a_2\, x_2 + r_2$. Les variables r_1, r_2, a_1, a_2 sont déjà dans des registres, seules x_1 et x_2 devant être chargées, ce qui est réalisable en un cycle. En général, l'unité de contrôle, les deux unités de chargement/écriture, les deux unités de calcul flottant, seront alors actives simultanément durant un cycle d'horloge.

Le cache L2

Le cache L2 est de 4 MO sur WH2 et de 8 MO sur NH2. Il est de type *2-way set associative* et sa latence moyenne est de 12 cycles pour une fréquence de 250 MHz. Il peut délivrer en mode *burst* une ligne de cache (128 octets) en quatre cycles (après 12 cycles de latence). En mode *burst*, la bande passante est donc de $(128/4) \times 250$ Mhz $=$ 8 GO/s, soit un peu plus que le cache L1. Si les données présentes dans le cache L2 peuvent être envoyées dans les registres suffisamment à l'avance (mécanisme de préchargement matériel ou logiciel), il est possible de réaliser 2 chargements par cycle. Si ce n'est la latence, cette performance correspond donc à celle du cache L1. Les caractéristiques des caches sont en accord pour assister le processeur grâce à une bande passante équivalente.

L'entrelacement

L'entrelacement (*interleaving*) permet d'accéder selon la donnée demandée à des bancs mémoire différents. Supposons que la mémoire soit entrelacée au niveau mot. Un exemple d'entrelacement consiste à dire que les adresses mémoires impaires seront traitées par le banc mémoire 0 et les paires par le banc mémoire 1. Un entrelacement de n bancs permet *a priori* de multiplier le débit de la mémoire jusque n. Pour un accès séquentiel au données, on utilise les n bancs les uns à la suite des autres sans discontinuer. Si le nombre de cycles α pour accéder à un mot est égal à n, on peut réussir à charger un mot par cycle en moyenne après une certaine latence. Si $2\,\alpha = n$, on peut espérer charger deux mots par cycle en moyenne. Pour les accès séquentiels, l'entrelacement augmente visiblement la bande passante mémoire.

Le cache de l'IBM SP3 WH2 (64 KO) est entrelacé par 4 au niveau des lignes de cache. En effet, il dispose de 4 ensembles (banques) de 128 lignes (128-way set associative). Dans chaque ensemble, il y a deux sous-banques, une pour les doubles-mots (16 octets) pairs, et une pour les doubles-mots impairs. Le cache de données peut effectuer deux chargements dans le même cycle, à condition que les adresses demandées correspondent à deux banques différentes et à deux sous-banques différentes. Si l'on accède à des séquences de flottants double précision (8 octets), il faut éviter absolument les pas (*strides*) de 2 ou 4. Dans cette dernière configuration, on accéderait uniquement aux doubles-mots pairs ou uniquement au doubles-mots impairs, réduisant ainsi la bande passante.

Les ports mémoires pour multiprocesseurs

Le concept de bancs mémoire peut-être étendu au niveau du contrôleur mémoire. On peut permettre à plusieurs *ports* mémoire d'accéder à la mémoire de manière indépendante. Il est possible d'augmenter la bande passante de multiprocesseurs qui ont une mémoire partagée en utilisant plusieurs contrôleurs en concurrence. Le IBM SP3 NH2 dispose de 2 contrôleurs mémoires indépendants, chacun est connecté à 2 cartes mémoires. Chaque carte constitue un port différent dont la bande passante est de 3.9 GO/s d'où un débit total de 15.6 GO/s. Sur le IBM SP3 WH2 il n'y a qu'un seul port à 1.5 GO/s. L'utilisation de caches associés à chaque processeur introduit un problème de cohérence des données sur les opérations de lecture/écriture vers la mémoire partagée (les nœuds SMP comportent 4 processeurs sur le WH2 et 16 processeurs sur le NH2 dans notre configuration). Le contenu de la mémoire peut être différent de la ligne correspondante présente dans le cache et en cours de modification par un processeur. Les techniques assurant la cohérence des caches sur une machine à mémoire partagée ne seront pas décrites ici.

Gestion de la mémoire virtuelle

La mémoire virtuelle du SP3 est gérée par segmentation et pagination. Pour accéder à la mémoire principale, les adresses virtuelles des processus sont traduites en adresses physiques (sur 40 bits) à l'aide d'une arborescence de tables de pages. Pour chaque accès à la mémoire, on devrait *a priori* chercher systématiquement dans la table des pages (qui se trouve en mémoire) l'adresse physique correspondante (ce qui coûte des accès mémoire supplémentaires). Pour éviter d'avoir à faire plusieurs accès mémoire pour chaque chargement (sans compter la gestion de la segmentation), les processeurs utilisent un cache de la table des pages. Ce tampon de traduction anticipée (TLB) utilise le principe de localité des données pour n'avoir qu'un accès mémoire pour chaque référence demandée. Le TLB comporte des entrées, où l'étiquette contient des parties de l'adresse virtuelle, et la partie donnée contient un numéro de page physique et d'autres informations. Pour la segmentation, il y a un SLB (*segment lookaside buffer)* qui est l'équivalent du TLB pour les segments. Dans le multiplexeur, le déplacement dans une page (poids faible de l'adresse virtuelle) est combiné avec le numéro de page physique (issu des SLB et TLB si la page a été récemment accédée) pour donner l'adresse physique complète. Le SLB de chaque processeur Power 3 comporte 16 entrées et le TLB 256 références de page. La taille des pages étant de 4 KO, le TLB couvre 1 MO de donnée. Dans certaines applications, cela peut constituer un goulot d'étranglement, car un défaut de référence du TLB occasionne de 10 à plusieurs centaines de cycles d'attente. Le TLB est un mécanisme très largement utilisé dans les architectures actuelles de processeur.

C.2 Algorithme GEMM

Sur l'architecture présentée, on va constater sur un algorithme de multiplication de matrices, comment certaines techniques conduisent à des gains de performance notables.

C.2.1 Cadre simplifié

On considère la multiplication des matrices : $C \leftarrow A^T.B+C$. Les routines présentées ci-après étant en FORTRAN 90, leur stockage se fait par colonne. Le nombre de lignes des matrices A^T, B, C seront respectivement h_A, h_B, h_C. La multiplication aura lieu entre des matrices de dimension : $A(dm \times dk)$, $B(dk \times dn)$ et $C(dm \times dn)$. La routine simple correspondant à la multiplication en double précision (les variables sont typées `REAL*8`) est :

```
SUBROUTINE multi_1(dm,dk,dn,BB,hb,AA,ha,CC,hc)
  INTEGER hb,ha,hc,dm,dk,dn,nn1,kk1,mm1
  REAL*8 :: BB(1:hb,0:*), AA(1:ha,0:*), CC(1:hc,0:*)
  do mm1=1,dm,1
     do kk1=1,dk,1
        do nn1=1,dn,1
           CC(mm1,nn1-1)=CC(mm1,nn1-1)+ AA(kk1,mm1-1)*BB(kk1,nn1-1)
        enddo
     enddo
  enddo
END SUBROUTINE multi_1
```

L'appel à la bibliothèque BLAS qui correspond à cette multiplication est le suivant :
`call dgemm('T','N',dm,dn,dk,1.0_8,AA,ha,BB,hb,1.0_8,CC,hc)` .

C.2.2 Inversion et blocage des boucles

L'utilisation de mémoires cache dans les processeurs repose sur les principes de localité spatiale et temporelle des références mémoire. La localité spatiale correspond au fait que lorsque l'on accède à une certaine adresse mémoire, la probabilité d'accéder par la suite à une adresse proche est forte. La localité temporelle consiste à dire que lorsque l'on charge une certaine adresse mémoire, il y a une probabilité élevée que l'on charge à nouveau la même adresse dans un délai court. Les méthodes pouvant renforcer ces localités permettent d'améliorer les performances en augmentant l'utilisation des caches (*i.e.* les défauts de cache sont évités). L'un des premiers objectifs est donc de faire apparaître dans l'algorithme des réutilisations de données. Un autre élément est de trouver un schéma d'accès aux données en mémoire performant. Lors du calcul, on a intérêt à accéder le plus séquentiellement possible aux données (localité spatiale). La séquentialité des accès est d'autant plus favorable sur l'architecture du Power 3 qu'elle conduit à un préchargement automatique dans le cache L1 diminuant les latences d'accès à la mémoire et au cache L2. En `FORTRAN` pour lequel le stockage est fait par colonne, la séquentialité s'exprime bien par l'incrémentation de l'indice de la première dimension des tableaux utilisés. Dans la routine `multi_1`, comment faire pour accéder en lecture à des adresses mémoire qui se suivent pour les matrices A et B ? Il faut faire en sorte de parcourir $kk1$ éléments en séquence, car il s'agit de l'indice de la première dimension des matrices. Pour cela, on intervertit la boucle sur l'indice $kk1$ et celle sur l'indice $nn1$, pour accéder séquentiellement à A et B.

L'idée est de minimiser le pas qui sépare les éléments des tableaux référencés lors des itérations successives. On "mesure" la localité avec les éléments présentés dans le Tableau qui suit.

Boucles	**Nombre de flottants accédés entre deux références à la matrice**		
	A	B	C
$mm1$	ha	0	1
$kk1$	1	1	0
$nn1$	0	hb	hc

L'imbrication des boucles doit suivre les règles suivantes : celles qui offrent le plus de séquentialité (la distance entre deux références est alors 1) ou de localité temporelle (distance de 0) seront placées à l'intérieur, les autres à l'extérieur. Selon le Tableau, l'ordre d'imbrication le meilleur est donc de faire de $kk1$ la boucle la plus interne (localité pour les trois matrices A, B, C), puis $mm1$ (localité pour B et C), enfin $nn1$ en boucle la plus externe. Cet ordre maintenant fixé, regardons dans le Tableau qui suit le volume de données réutilisées d'une itération à l'autre de chaque boucle, afin de détecter le localité temporelle potentielle.

Boucles	**Taille des données réutilisables [nombre de réutilisations possibles pour les matrices]**		
	A	B	C
$nn1$	$dm \times dk$ [dn]	–	–
$mm1$	–	dk [dm]	–
$kk1$	–	–	1 [dk]

Un problème qui se pose à la lecture du Tableau est de savoir comment faire pour que le plus de données possible soient réutilisées. En effet, si les données de A de taille $dm \times dk$ ne restent pas dans le cache entre deux itérations de $nn1$, on devra les recharger en mémoire cache à chaque itération de $nn1$. Il est même possible que ces données de A ne puissent pas être contenues dans le cache. Dans ce dernier cas, on deva recharger A un nombre dn de fois. Il est possible d'obtenir plus de localité des données en considérant un algorithme par blocs qui multiplie des sous-matrices de A, B, C entre elles. On peut alors réutiliser des sous-matrices qui restent dans le cache et éviter par exemple de recharger A autant de fois. On gagne ainsi le temps qu'il aurait fallu pour charger à nouveau les sous-matrices de la mémoire vers le cache. L'algorithme avec blocage des boucles est le suivant (les facteurs de blocage sur les boucles $kk1$, $mm1$, $nn1$ sont $pask$, $pasm$, $pasn$) :

```
SUBROUTINE multi_2(dm,dk,dn,BB,hb,AA,ha,CC,hc,pasm,pask,pasn)
    INTEGER :: hb,ha,hc,dm,dk,dn,pask,pasn,pasm
    REAL*8  :: BB(1:*), AA(1:*), CC(1:*)
    INTEGER :: nn1,kk1,mm1,nn2,kk2,mm2
    do nn2=1, dn, pasn
      do mm2=1, dm, pasm
        do kk2=1, dk, pask
          do nn1=nn2,MIN(nn2+pasn-1,dn)
            do mm1=mm2,MIN(mm2+pasm-1,dm)
              do kk1=kk2,MIN(kk2+pask-1,dk)
                CC(mm1+(nn1-1)*hc)=CC(mm1+(nn1-1)*hc)+AA(kk1+(mm1-1)*ha)*BB(kk1+(nn1-1)*hb)
              enddo
            enddo
          enddo
        enddo
      enddo
    enddo
 END SUBROUTINE multi_2
```

L'amélioration de la localité peut se constater sur le Tableau qui suit. En effet, certains blocs de données réutilisables ont des tailles adaptables qui peuvent être contenues dans le cache L1.

Boucles	**Pas entre éléments accédés**			**Taille des données réutilisables [nombre de réutilisations possibles]**		
	A	B	C	A	B	C
$nn2$	0	hb	hc	$dn \times dk$ $[\frac{dn}{pasn}]$	–	–
$mm2$	ha	0	pasm	–	$pasm \times dk$ $[\frac{dm}{pasm}]$	–
$kk2$	pask	pask	0	–	–	$pasm \times pasn$ $[\frac{dk}{pask}]$
$nn1$	0	hb	hc	$pasn \times pask$ $[pasn]$	–	–
$mm1$	ha	0	1	–	$pask$ $[pasm]$	–
$kk1$	1	1	0	–	–	1 $[pask]$

Par exemple, pour les quatre boucles intérieures de l'algorithme ($kk1$, $mm1$, $nn1$, $kk2$), toutes les données réutilisables ont des tailles qui sont paramétrables par les dimensions des blocs : $pask, pasn, pasm$. Cela signifie que, pour des dimensions telles que $(pasm \times pasn) + (pasn \times pask) + pask <$ taille du cache L1, on peut théoriquement faire une réutilisation efficace de l'ensemble des sous-matrices. En conséquence, si la réutilisation des sous-matrices de A est effective (boucle $nn1$), alors au niveau de la boucle extérieure ($nn2$) on constate que le nombre de rechargements nécessaires de la matrice A n'est plus que $\frac{dn}{pasn}$ (dans le pire des cas, au lieu de dn avec l'algorithme précédent).

C.2.3 Déroulage des boucles et blocage de registres

Avec un processeur superscalaire comportant plusieurs unités flottantes, il est possible de réaliser simultanément plusieurs instructions de calcul flottant et de les *pipeliner*. Pour réaliser cela, une condition nécessaire est d'avoir des séquences d'instructions indépendantes. Le déroulage de boucle est une technique qui permet d'exhiber une quantité plus importante de parallélisme entre les instructions. Cela consiste à transformer une boucle en répliquant le corps de celle-ci sur plusieurs itérations successives. On cherche ainsi à proposer au compilateur plusieurs itérations réalisables indépendamment au cœur de la boucle. Ceci permet au compilateur puis au processeur d'utiliser avec un meilleur rendement les unités fonctionnelles. D'autre part, le déroulage conduit à un coût de gestion de la boucle moindre lorsqu'elle comporte beaucoup d'itérations, puisque le test de boucle est réalisé moins souvent. Sur l'algorithme précédant, déroulons les boucles $nn1$ et $mm1$ sur 4 itérations successives.

```
SUBROUTINE multi_3(dm,dk,dn,BB,hb,AA,ha,CC,hc,pasm,pask,pasn)
   INTEGER :: hb,ha,hc,dm,dk,dn,pask,pasn,pasm
   REAL*8  :: BB(1:*), AA(1:*), CC(1:*)
   INTEGER :: nn1,kk1,mm1,nn2,kk2,mm2,DEEP,i,j
   PARAMETER (DEEP=4)
   REAL*8 :: reg(0:DEEP-1,0:DEEP-1)
   do nn2=1, dn, pasn
     do mm2=1, dm, pasm
       do kk2=1, dk, pask
         do nn1=nn2,MIN(nn2+pasn,dn+1)-DEEP,DEEP
           do mm1=mm2,MIN(mm2+pasm,dm+1)-DEEP,DEEP
             reg=0.0_8
             do kk1=kk2,MIN(kk2+pask-1,dk)
                 reg(0,0)=reg(0,0)+AA(kk1+(mm1-1)*ha+ha*0)  * BB(kk1+(nn1-1)*hb+hb*0)
                 reg(1,0)=reg(1,0)+AA(kk1+(mm1-1)*ha+ha*1)  * BB(kk1+(nn1-1)*hb+hb*0)
                 reg(2,0)=reg(2,0)+AA(kk1+(mm1-1)*ha+ha*2)  * BB(kk1+(nn1-1)*hb+hb*0)
                 reg(3,0)=reg(3,0)+AA(kk1+(mm1-1)*ha+ha*3)  * BB(kk1+(nn1-1)*hb+hb*0)
                 reg(0,1)=reg(0,1)+AA(kk1+(mm1-1)*ha+ha*0)  * BB(kk1+(nn1-1)*hb+hb*1)
                 reg(1,1)=reg(1,1)+AA(kk1+(mm1-1)*ha+ha*1)  * BB(kk1+(nn1-1)*hb+hb*1)
                 reg(2,1)=reg(2,1)+AA(kk1+(mm1-1)*ha+ha*2)  * BB(kk1+(nn1-1)*hb+hb*1)
                 reg(3,1)=reg(3,1)+AA(kk1+(mm1-1)*ha+ha*3)  * BB(kk1+(nn1-1)*hb+hb*1)
                 reg(0,2)=reg(0,2)+AA(kk1+(mm1-1)*ha+ha*0)  * BB(kk1+(nn1-1)*hb+hb*2)
                 reg(1,2)=reg(1,2)+AA(kk1+(mm1-1)*ha+ha*1)  * BB(kk1+(nn1-1)*hb+hb*2)
                 reg(2,2)=reg(2,2)+AA(kk1+(mm1-1)*ha+ha*2)  * BB(kk1+(nn1-1)*hb+hb*2)
                 reg(3,2)=reg(3,2)+AA(kk1+(mm1-1)*ha+ha*3)  * BB(kk1+(nn1-1)*hb+hb*2)
                 reg(0,3)=reg(0,3)+AA(kk1+(mm1-1)*ha+ha*0)  * BB(kk1+(nn1-1)*hb+hb*3)
                 reg(1,3)=reg(1,3)+AA(kk1+(mm1-1)*ha+ha*1)  * BB(kk1+(nn1-1)*hb+hb*3)
                 reg(2,3)=reg(2,3)+AA(kk1+(mm1-1)*ha+ha*2)  * BB(kk1+(nn1-1)*hb+hb*3)
                 reg(3,3)=reg(3,3)+AA(kk1+(mm1-1)*ha+ha*3)  * BB(kk1+(nn1-1)*hb+hb*3)
              enddo
             FORALL(i=0:DEEP-1,j=0:DEEP-1) &
                  CC(mm1+i+(nn1+j-1)*hc)= &
                  CC(mm1+i+(nn1+j-1)*hc) + reg(i,j)
           enddo
         enddo
       enddo
     enddo
   enddo
END SUBROUTINE multi_3
```

Au niveau de la boucle interne $kk1$, il y a maintenant 16 instructions indépendantes. L'algorithme propose l'utilisation des registres architecturaux `reg(i,j)` pour stocker les sommes partielles de la matrice C. A chaque itération de $kk1$, on réutilise les 16 variables. On ajoute finalement le contenu de ces sommes partielles dans la matrice C à la fin de la boucle en $kk1$. S'il y a un nombre suffisant de registres pour contenir ces variables, et si le compilateur utilise effectivement ces registres, quelles en sont les conséquences ? Le nombre d'accès à la mémoire (accès à la matrice C) est diminué, car dans la boucle $kk1$ on travaille sur uniquement 16 registres `reg` en lecture/écriture et non sur la matrice C originale. Ceci est très intéressant (aucun coût lié à l'écriture), puisque cela libère de la bande passante pour faire d'autres accès mémoire. Cette technique s'appelle le blocage de registre (*register blocking*). Elle s'apparente au blocage de boucles (qui permet de réutiliser des données stockées dans le cache) car elle conserve au maximum les données manipulées au niveau des registres, qui peuvent être considérées ensemble comme un tout petit cache. En fait, on va aussi chercher à attribuer des registres pour les données en lecture de A et

B. Dans la boucle en $kk1$, il y a 4 lectures sur A et 4 lectures sur B, et chaque donnée lue est utilisée 4 fois. En définitive, pour une itération de la boucle $kk1$, il y a 8 lectures de flottants en mémoire, 16 lectures/écritures sur les registres *reg*, pour 16 `fmadd`. Pour le déroulage de boucle proposé, on a donc besoin au minimum de 24 registres flottants, alors que le Power 3 en possède 32. On a finalement réussi à améliorer l'utilisation des ressources suivantes : réutilisation des registres, utilisation en parallèle des deux unités de calcul flottant et des unités de chargement/mise à jour, réduction de la bande passante nécessaire vers les mémoires cache et conventionnelle.

C.2.4 Prise en compte du TLB et du cache L2

Au niveau de la boucle en $kk1$, on effectue 8 chargements pour 16 instructions `fmadd`. Le processeur Power3 peut *a priori* supporter une bande passante deux fois plus importante depuis la mémoire cache. Dans ce contexte, le point critique est l'accès à la mémoire conventionnelle, plus lente que la mémoire cache. Au delà de certaines tailles de données manipulées, le cache L1, le cache L2, le TLB provoquent des défauts de cache. Sur le SP3 WH2, le cache L2 contient 4 MO de données et le TLB adresse 1 MO de données. Pour éviter de générer trop de défauts de TLB et de cache L2, on réalise un nouveau blocage de boucles. Pour simplifier la gestion du blocage, le nouveau facteur de blocage sera un multiple de celui du cache L1. Dans l'algorithme précédant `multi_3`, on peut mesurer sur un exemple le nombre de défauts de cache et le comparer avec les résultats du nouvel algorithme avec blocage `multi_4`.

```
SUBROUTINE multi_4(dm,dk,dn,BB,hb,AA,ha,CC,hc,pasm,pask,pasn,wm,wk,wn)
   INTEGER :: hb,ha,hc,dm,dk,dn,pask,pasn,pasm,wm,wk,wn
   REAL*8  :: BB(1:*), AA(1:*), CC(1:*)
   INTEGER :: nn1,kk1,mm1,nn2,kk2,mm2,nn3,kk3,mm3,DEEP,i,j
   PARAMETER (DEEP=4)
   REAL*8 :: reg(0:DEEP-1,0:DEEP-1)
   do nn3=1, dn, pasn*wn
    do mm3=1, dm, pasm*wm
     do kk3=1, dk, pask*wk
      do nn2=nn3, MIN(nn3+pasn*(wn-1),dn), pasn
       do mm2=mm3, MIN(mm3+pasm*(wm-1),dm), pasm
        do kk2=kk3, MIN(kk3+pask*(wk-1),dk), pask
         do nn1=nn2,MIN(nn2+pasn,dn+1)-DEEP,DEEP
          do mm1=mm2,MIN(mm2+pasm,dm+1)-DEEP,DEEP
             reg=0.0_8
             do kk1=kk2,MIN(kk2+pask-1,dk)
              ⋮ (identique à multi_3)
             enddo
             FORALL(i=0:DEEP-1,j=0:DEEP-1) &
                  CC(mm1+i+(nn1+j-1)*hc)= &
                  CC(mm1+i+(nn1+j-1)*hc) + reg(i,j)
          enddo
         enddo
        enddo
       enddo
      enddo
     enddo
```

```
      enddo
     enddo
END SUBROUTINE multi_4
```

On utilise l'outil HPM (Hardware Performance Monitor Toolkit) pour évaluer l'utilisation du TLB et du cache L2 sur le SP3 WH2. On obtient les résultats du Tableau suivant (`multi_4(x,x,x,BB,x,AA,x,CC,x,8,288,16,8,100,16)` avec $x = 800,\ 1500$ ou 2000) :

algorithme	`multi_3`			`multi_4`		
mesure	nombre de défauts TLB	nombre moyen de chargements sans défaut L2	MFLOPS	nombre de défauts TLB	nombre moyen de chargements sans défaut L2	MFLOPS
matrice 800×800	$7.1\ 10^4$	151	1160	$4.4\ 10^4$	670	1286
matrice 1500×1500	$5.7\ 10^5$	118	1070	$5.3\ 10^5$	445	1274
matrice 2000×2000	$1.4\ 10^6$	119	1129	$1.6\ 10^6$	345	1264

On constate que le nombre moyen de chargements sans défaut de cache L2 augmente avec le nouvel algorithme. Ce qui est essentiel, c'est que le blocage permette d'avoir de meilleures performances en MFLOPS. Par contre on constate que le nombre de défauts pour le TLB augmente très légèrement pour `multi_4` pour la matrice 2000×2000, alors que l'on s'attendait plutôt à une baisse. Il ne faut pas négliger le fait que, d'une exécution à l'autre de la routine, les zones mémoire allouées pour les matrices peuvent avoir des adresses initiales différentes, ce qui peut changer le nombre de défauts de cache. D'autre part, il peut survenir un phénomène de conflit d'accès aux caches ou au TLB dû à leur degré d'associativité limité. Par exemple, lorsque l'on accède à trois zones mémoire pour les matrices A, B, C cela peut provoquer un accès concurrent au cache L2. En effet, ce cache est de type 2-way associatif, donc si l'on fait référence à trois zones mémoire qui se recouvrent (modulo la taille du cache L2), le processeur va en permanence remplacer des lignes du cache par des données provenant de ces trois zones mémoire (appelé en anglais *cache trashing*). Une solution pour solliciter raisonnablement les caches et éviter le problème est proposée dans la prochaine Section.

C.2.5 Accès mémoire, rôle de l'entrelacement

Au niveau de la boucle la plus interne en $kk1$, on accède aux données de A et B via 8 flots mémoire pour $i \in [0,3], j \in [0,3]$. Cela signifie que l'on dispose de huit pointeurs sur des zones mémoires différentes. Or, comme on l'a évoqué précédemment, le mécanisme de *prefetch* automatique gère quatre flots au maximum en simultané, ce qui empêche un chargement efficacement anticipé des 8 flots. D'autre part, on ne tient pas compte de l'entrelacement des bancs mémoire dans l'algorithme `multi_4` ; cela pourrait accélérer les temps d'accès. Lorsque l'on incrémente $kk1$, il n'est pas sûr que les données accédées soient dans des bancs différents, ce qui engendre un temps d'attente supplémentaire lorsqu'il y a des conflits d'accès aux bancs. Pour remédier à ces problèmes, l'idée est de recopier les données de A et B dans un tampon, de manière à ce que l'algorithme ait ensuite un accès optimisé, du point de vue de l'entrelacement, vers ce tampon.

Dans l'algorithme `multi_5` [46], le stockage de A et B dans tmp permet de réorganiser les données. En effet, les 4 flots de données que l'on avait sur la matrice A sont multiplexés dans le tampon tmp (de même pour B). Lorsque l'on considère quatre éléments successifs dans tmp, on a en fait des réels qui appartiennent à quatres colonnes différentes de B. Ce multiplexage permet à l'exécution d'accéder uniquement à 2 flots de données (correspondant aux matrices A et B stockés dans tmp) au niveau de la boucle interne $kk1$. L'organisation des données en deux flots de données augmente la localité spatiale, permet d'éviter les conflits de bancs lors des calculs (on organise les données pour cela), et utilise efficacement le mécanisme de `prefetch` automatique (moins de 4 flots de données à gérer). La routine `multi_5` qui intègre les modifications décrites est présentée ci-après :

```
SUBROUTINE multi_5(dm,dk,dn,BB,hb,AA,ha,CC,hc,pasm,pask,pasn,wm,wk,wn)
   INTEGER :: hb,ha,hc,dm,dk,dn,pask,pasn,pasm,wm,wk,wn
   REAL*8  :: BB(1:*), AA(1:*), CC(1:*)
   INTEGER :: nn1,kk1,mm1,nn2,kk2,mm2,nn3,kk3,mm3,DEEP,i,j,p,q
   PARAMETER (DEEP=4)
   REAL*8 :: reg(0:DEEP-1,0:DEEP-1)
   INTEGER :: HLONG,HLARG,HDEC,HDEB,HK
   PARAMETER (HLONG=512) ! HLONG multiple de 64
   PARAMETER (HLARG=512) !
   REAL*8 :: tmp(1:HLONG*HLARG)

   HDEC=HLONG*wm*pasm+34
   HDEB=1+(32768-MOD(LOC(tmp),32768))/8
   do kk2=1, dk, pask
   do nn3=1, dn, pasn*wn
      HK=32*(1+(MIN(pask,dk-kk2+1)-1)/32)
      do i=0,MIN(pasn*wn,dn-nn3+1)-DEEP,DEEP
       do j=0,MIN(pask-1,dk-kk2),1
         FORALL (p=0:DEEP-1) tmp(HDEB+HDEC+4*j+p+HK*i) = BB(kk2+(nn3-1+i+p)*hb+j)
       enddo
      enddo
    do mm3=1, dm, pasm*wm
       do i=0,MIN(pasm*wm,dm-mm3+1)-DEEP,DEEP
        do j=0,MIN(pask-1,dk-kk2),1
          FORALL (p=0:DEEP-1) tmp(HDEB+4*j+p+HK*i) = AA(kk2+(mm3-1+i+p)*ha+j)
        enddo
       enddo

      do nn2=nn3, MIN(nn3+pasn*(wn-1),dn), pasn
       do mm2=mm3, MIN(mm3+pasm*(wm-1),dm), pasm
         do nn1=nn2,MIN(nn2+pasn,dn+1)-DEEP,DEEP
          do mm1=mm2,MIN(mm2+pasm,dm+1)-DEEP,DEEP
             reg(0:DEEP-1,0:DEEP-1)=0.0_8
             p=HDEB+HDEC+(nn1-nn3)*HK
             q=HDEB+(mm1-mm3)*HK
             do kk1=0,4*MIN(pask-1,dk-kk2),4
               FORALL(i=0:DEEP-1,j=0:DEEP-1) &
                     reg(i,j)=reg(i,j)+tmp(q+kk1+i) * tmp(p+kk1+j)
```

46. Pour éviter des recopies trop volumineuses dans le tampon tmp, l'algorithme `multi_5` supprime la boucle en $kk3$ et place la boucle en $kk2$ à l'extérieur du nid de boucles. De cette manière, des sous-matrices de taille $pas_n\, w_n \times pas_k$ pour A sont copiées, et $pas_m\, w_m \times pas_k$ pour B.

```
                enddo
                FORALL(i=0:DEEP-1,j=0:DEEP-1) &
                     CC(mm1+i+(nn1+j-1)*hc)= &
                     CC(mm1+i+(nn1+j-1)*hc) + reg(i,j)
            enddo
           enddo
          enddo
         enddo
        enddo
       enddo
      enddo
END SUBROUTINE multi_5
```

Cette méthode permet de tirer un grand profit de l'entrelacement des bancs mémoire. En effet, on organise le placement des données pour que les premiers éléments de A stockés provisoirement dans *tmp* soient au début du banc 0, et ceux de B au début du banc 2. Lorsque l'algorithme aura consommé tous les éléments des bancs, il va simultanément passer sur le banc 1 pour A et sur le banc 3 pour B. Cette manière d'organiser le stockage dans *tmp* permet de ne jamais avoir de conflits d'accès aux bancs lors des calculs, donc de maximiser la bande passante vers la mémoire. Pour décaler A et B dans *tmp*, on introduit en pratique une variable `HDEC` qui est l'espace (en nombre de flottants) qui sépare le début de A du début de B. Cette variable est l'addition d'un multiple de 4 fois la taille d'une ligne de cache (*i.e.* 64 flottants), plus la taille de deux lignes de cache (32 flottants, auquel on ajoute 2 flottants pour l'entrelacement des sous-banques).

Le seul bémol à la technique décrite est le nombre de recopies mémoire nécessaires. Pour des petites matrices à multiplier, les performances seront meilleures sans utiliser la méthode. Pour des grandes matrices, les bénéfices se cumulent et apportent un gain significatif en performance. On observe ceci dans le Tableau suivant tant au niveau du nombre d'opérations réalisées par seconde qu'au niveau du nombre moyen de chargements mémoire sans défaut de cache L2.

algorithme	`multi_4`			`multi_5`		
mesure	nombre de défauts TLB	nombre moyen de chargements sans défaut L2	MFLOPS	nombre de défauts TLB	nombre moyen de chargements sans défaut L2	MFLOPS
matrice 800×800	$4.4\ 10^4$	670	1286	$7.4\ 10^4$	735	1293
matrice 1500×1500	$5.3\ 10^5$	445	1274	$4.8\ 10^5$	784	1288
matrice 2000×2000	$1.6\ 10^6$	345	1264	$1.0\ 10^6$	816	1321

C.2.6 Parallélisme d'instructions

Dans la Section C.1.1, on a vu que plusieurs instructions peuvent être exécutées en même temps. Notamment, il est possible en un cycle de finir deux `fmadd` pipelinés et de réaliser un `lfdu` pour charger une donnée en décalant le pointeur de cette donnée. En huit cycles exactement, le processeur peut réaliser l'ensemble des instructions au niveau de la boucle interne en $kk1$ si les données sont disponibles dans le cache L1. Pour constater le pseudo-code assembleur généré, le compilateur FORTRAN d'IBM (`xlf90`) dispose des options `-qsource -qlist -qreport`. Ainsi, l'utilisateur peut constater certaines des optimisations faites. Examinons ce code pour la boucle la plus interne en $kk1$ de la routine `multi_5`.

```
636|                            CL.371:
638| 001AB8 lfdx        LFL     fp13=tmp(gr15,gr5,0)
638| 001ABC fmadd       FMA     fp26=fp26,fp6,fp20,fcr
638| 001AC0 lfd         LFL     fp27=tmp(gr17,384)
638| 001AC4 fmadd       FMA     fp30=fp30,fp3,fp20,fcr
638| 001AC8 lfd         LFL     fp24=tmp(gr17,408)
638| 001ACC fmadd       FMA     fp1=fp1,fp9,fp20,fcr
  0| 001AD0 addi        AI      gr18=gr18,32,ca"
638| 001AD4 fmadd       FMA     fp8=fp8,fp13,fp20,fcr
638| 001AD8 add         A       gr17=gr5,gr18
638| 001ADC fmadd       FMA     fp28=fp28,fp27,fp9,fcr
  0| 001AE0 addi        AI      gr20=gr20,32,ca"
638| 001AE4 fmadd       FMA     fp21=fp21,fp13,fp15,fcr
  0| 001AE8 addi        AI      gr16=gr16,32,ca"
638| 001AEC fmadd       FMA     fp5=fp5,fp9,fp15,fcr
  0| 001AF0 addi        AI      gr15=gr15,32,ca"
638| 001AF4 fmadd       FMA     fp10=fp10,fp9,fp24,fcr
  0| 001AF8 addi        AI      gr19=gr19,32,ca"
638| 001AFC fmadd       FMA     fp25=fp25,fp6,fp15,fcr
638| 001B00 lfd         LFL     fp20=tmp(gr17,392)
638| 001B04 fmadd       FMA     fp29=fp29,fp3,fp15,fcr
638| 001B08 lfd         LFL     fp15=tmp(gr17,400)
638| 001B0C fmadd       FMA     fp2=fp2,fp27,fp3,fcr
638| 001B10 lfdx        LFL     fp9=tmp(gr20,gr5,0)
638| 001B14 fmadd       FMA     fp4=fp4,fp3,fp24,fcr
638| 001B18 lfdx        LFL     fp3=tmp(gr16,gr5,0)
638| 001B1C fmadd       FMA     fp12=fp12,fp27,fp6,fcr
638| 001B20 fmadd       FMA     fp7=fp7,fp6,fp24,fcr
638| 001B24 lfdx        LFL     fp6=tmp(gr19,gr5,0)
```

```
638| 001B28 fmadd       FMA       fp11=fp11,fp13,fp24,fcr
638| 001B2C fmadd       FMA       fp31=fp31,fp27,fp13,fcr
  0| 001B30 bc          BCT       ctr=CL.371,
```

On compte 16 `fmadd` comme dans l'algorithme dans la boucle en $kk1$, ainsi que 8 chargements (`lfd` et `lfdx`). En revanche, on constate aussi 8 autres instructions qui ne sont pas nécessaires *a priori*. Ces instructions supplémentaires sont des `add` et `addi` qui mettent à jour des pointeurs sur les matrices A et B. Ces instructions concernant des opérations entières peuvent tout à fait s'exécuter en parallèle des instructions flottantes. Néanmoins, elles peuvent générer des retards dans la disponibilité des flottants servant aux calculs. En effet, le compilateur aurait pû placer en lieu et place des opérations de chargements `lfd`, `lfdx`, des opérations de chargement/mise à jour qui permettent de charger une donnée et mettre un pointeur à jour en même temps (instruction `lfdu`). On peut réécrire l'algorithme en spécifiant la gestion explicite des pointeurs sur les matrices A et B. Cela est fait dans la routine `multi_6`.

```
SUBROUTINE multi_6(dm,dk,dn,BB,hb,AA,ha,CC,hc,pasm,pask,pasn,wm,wk,wn)
   INTEGER :: hb,ha,hc,dm,dk,dn,pask,pasn,pasm,wm,wk,wn
   REAL*8  :: BB(1:*), AA(1:*), CC(1:*)
   INTEGER :: nn1,kk1,mm1,nn2,kk2,mm2,nn3,kk3,mm3,DEEP,i,j,p,q
   PARAMETER (DEEP=4)
   REAL*8 :: reg00,reg01,reg02,reg03,reg10,reg11,reg12,reg13
   REAL*8 :: reg20,reg21,reg22,reg23,reg30,reg31,reg32,reg33
   REAL*8 :: a1,a2,a3,a0,b1,b2,b3,b0
   POINTER (pa0,a0)
   POINTER (pa1,a1)
   POINTER (pa2,a2)
   POINTER (pa3,a3)
   POINTER (pb0,b0)
   POINTER (pb1,b1)
   POINTER (pb2,b2)
   POINTER (pb3,b3)
   INTEGER :: HLONG,HLARG,HDEC,HDEB,HM,HN,HK
   PARAMETER (HLONG=1280) !
   PARAMETER (HLARG=513)  !
   REAL*8 :: tmp(1:HLONG*HLARG)

   HDEC=HLONG*wm*pasm+34
   HDEB=1+(32768-MOD(LOC(tmp),32768))/8
   do kk2=1, dk, pask
   do nn3=1, dn, pasn*wn
      HK=32*(1+(MIN(pask,dk-kk2+1)-1)/32)
      do i=0,MIN(pasn*wn,dn-nn3+1)-DEEP,DEEP
       do j=0,MIN(pask-1,dk-kk2),1
         FORALL (p=0:DEEP-1) tmp(HDEB+HDEC+4*j+p+HK*i) = BB(kk2+(nn3-1+i+p)*hb+j)
       enddo
      enddo
    do mm3=1, dm, pasm*wm
       do i=0,MIN(pasm*wm,dm-mm3+1)-DEEP,DEEP
        do j=0,MIN(pask-1,dk-kk2),1
          FORALL (p=0:DEEP-1) tmp(HDEB+4*j+p+HK*i) = AA(kk2+(mm3-1+i+p)*ha+j)
        enddo
       enddo

      do nn2=nn3, MIN(nn3+pasn*(wn-1),dn), pasn
       do mm2=mm3, MIN(mm3+pasm*(wm-1),dm), pasm
```

```
  do nn1=nn2,MIN(nn2+pasn,dn+1)-DEEP,DEEP
   do mm1=mm2,MIN(mm2+pasm,dm+1)-DEEP,DEEP
      reg00=0;reg01=0;reg02=0;reg03=0;
      reg10=0;reg11=0;reg12=0;reg13=0;
      reg20=0;reg21=0;reg22=0;reg23=0;
      reg30=0;reg31=0;reg32=0;reg33=0;
      pb0=LOC(tmp(HDEB+HDEC+(nn1-nn3)*HK+0))
      pb1=LOC(tmp(HDEB+HDEC+(nn1-nn3)*HK+1))
      pb2=LOC(tmp(HDEB+HDEC+(nn1-nn3)*HK+2))
      pb3=LOC(tmp(HDEB+HDEC+(nn1-nn3)*HK+3))
      pa0=LOC(tmp(HDEB+(mm1-mm3)*HK+0))
      pa1=LOC(tmp(HDEB+(mm1-mm3)*HK+1))
      pa2=LOC(tmp(HDEB+(mm1-mm3)*HK+2))
      pa3=LOC(tmp(HDEB+(mm1-mm3)*HK+3))
      do kk1=0,MIN(pask-1,dk-kk2),1
         reg00=reg00+a0 * b0
         reg02=reg02+a0 * b2
         reg01=reg01+a0 * b1
         reg03=reg03+a0 * b3
         reg20=reg20+a2 * b0
         reg22=reg22+a2 * b2
         reg21=reg21+a2 * b1
         reg23=reg23+a2 * b3
         pa0=pa0+32
         pa2=pa2+32
         reg10=reg10+a1 * b0
         reg12=reg12+a1 * b2
         reg30=reg30+a3 * b0
         reg32=reg32+a3 * b2
         pb0=pb0+32
         pb2=pb2+32
         reg11=reg11+a1 * b1
         reg13=reg13+a1 * b3
         reg31=reg31+a3 * b1
         reg33=reg33+a3 * b3
         pb1=pb1+32
         pb3=pb3+32
         pa1=pa1+32
         pa3=pa3+32
      enddo
      CC(mm1+0+(nn1+0-1)*hc)=CC(mm1+0+(nn1+0-1)*hc)+reg00
      CC(mm1+0+(nn1+1-1)*hc)=CC(mm1+0+(nn1+1-1)*hc)+reg01
      CC(mm1+1+(nn1+0-1)*hc)=CC(mm1+1+(nn1+0-1)*hc)+reg10
      CC(mm1+2+(nn1+0-1)*hc)=CC(mm1+2+(nn1+0-1)*hc)+reg20
      CC(mm1+3+(nn1+0-1)*hc)=CC(mm1+3+(nn1+0-1)*hc)+reg30
      CC(mm1+1+(nn1+1-1)*hc)=CC(mm1+1+(nn1+1-1)*hc)+reg11
      CC(mm1+2+(nn1+1-1)*hc)=CC(mm1+2+(nn1+1-1)*hc)+reg21
      CC(mm1+3+(nn1+1-1)*hc)=CC(mm1+3+(nn1+1-1)*hc)+reg31
      CC(mm1+0+(nn1+2-1)*hc)=CC(mm1+0+(nn1+2-1)*hc)+reg02
      CC(mm1+0+(nn1+3-1)*hc)=CC(mm1+0+(nn1+3-1)*hc)+reg03
      CC(mm1+1+(nn1+2-1)*hc)=CC(mm1+1+(nn1+2-1)*hc)+reg12
      CC(mm1+2+(nn1+2-1)*hc)=CC(mm1+2+(nn1+2-1)*hc)+reg22
      CC(mm1+3+(nn1+2-1)*hc)=CC(mm1+3+(nn1+2-1)*hc)+reg32
      CC(mm1+1+(nn1+3-1)*hc)=CC(mm1+1+(nn1+3-1)*hc)+reg13
      CC(mm1+2+(nn1+3-1)*hc)=CC(mm1+2+(nn1+3-1)*hc)+reg23
      CC(mm1+3+(nn1+3-1)*hc)=CC(mm1+3+(nn1+3-1)*hc)+reg33
  enddo
 enddo
enddo
```

```
          enddo
         enddo
        enddo
       enddo
END SUBROUTINE multi_6
```

Lorsque l'on affiche en pseudo-code la boucle interne (en $kk1$) de l'algorithme `multi_6`, on obtient alors l'utilisation des instructions `lfdu`. Au niveau de cette boucle, le jeu minimal d'instructions pour effectuer les opérations de l'algorithme est utilisé, ce qui permet d'augmenter le parallélisme d'instructions :

```
714|                              CL.110:
718| 000A54 fmadd     FMA      fp22=fp22,fp10,fp12,fcr
717| 000A58 fmadd     FMA      fp4=fp4,fp10,fp23,fcr
719| 000A5C fmadd     FMA      fp13=fp13,fp10,fp24,fcr
716| 000A60 lfdu      LFDU     fp10,gr23=b0.rns10.(gr23,32)
721| 000A64 fmadd     FMA      fp2=fp2,fp7,fp26,fcr
729| 000A68 fmadd     FMA      fp21=fp21,fp7,fp27,fcr
731| 000A6C fmadd     FMA      fp3=fp3,fp7,fp25,fcr
716| 000A70 lfdu      LFDU     fp7,gr25=b0.rns10.(gr25,32)
722| 000A74 fmadd     FMA      fp11=fp11,fp23,fp26,fcr
723| 000A78 fmadd     FMA      fp9=fp9,fp12,fp26,fcr
724| 000A7C fmadd     FMA      fp30=fp30,fp24,fp26,fcr
721| 000A80 lfdu      LFDU     fp26,gr20=b0.rns10.(gr20,32)
730| 000A84 fmadd     FMA      fp8=fp8,fp23,fp27,fcr
737| 000A88 fmadd     FMA      fp5=fp5,fp12,fp27,fcr
738| 000A8C fmadd     FMA      fp29=fp29,fp24,fp27,fcr
729| 000A90 lfdu      LFDU     fp27,gr22=b0.rns10.(gr22,32)
732| 000A94 fmadd     FMA      fp31=fp31,fp23,fp25,fcr
717| 000A98 lfdu      LFDU     fp23,gr19=b0.rns10.(gr19,32)
739| 000A9C fmadd     FMA      fp6=fp6,fp12,fp25,fcr
718| 000AA0 lfdu      LFDU     fp12,gr24=b0.rns10.(gr24,32)
740| 000AA4 fmadd     FMA      fp28=fp28,fp24,fp25,fcr
731| 000AA8 lfdu      LFDU     fp25,gr21=b0.rns10.(gr21,32)
719| 000AAC lfdu      LFDU     fp24,gr18=b0.rns10.(gr18,32)
716| 000AB0 fmadd     FMA      fp1=fp1,fp10,fp7,fcr
  0| 000AB4 bc        BCT      ctr=CL.110,
```

Les performances sont améliorées, comme on peut le voir sur le tableau suivant :

algorithme	`multi_5`			`multi_6`		
mesure	nombre de défauts TLB	nombre moyen de chargements sans défaut L2	MFLOPS	nombre de défauts TLB	nombre moyen de chargements sans défaut L2	MFLOPS
matrice 800×800	$7.4\ 10^4$	735	1293	$7.3\ 10^4$	691	1306
matrice 1500×1500	$4.8\ 10^5$	784	1288	$4.7\ 10^5$	733	1306
matrice 2000×2000	$1.0\ 10^6$	816	1321	$1.0\ 10^6$	770	1335

C.2.7 Ajustement des paramètres

En ajustant les tailles des blocs de l'algorithme, il est possible d'obtenir une amélioration significative des performances. Après avoir exploré l'espace des paramètres, les tailles suivantes sont les meilleures en moyenne : `multi_6'(x,x,x,BB,x,AA,x,CC,x,8,544,16,8,100,16)`. Cela signifie en ce qui concerne la matrice A des données de taille $8 \times 8 \times 544$ flottants $=$ 272 KO dans *tmp* et, pour B, des données de taille $16 \times 16 \times 544$ flottants $=$ 1088 KO dans *tmp*. Les éléments de *tmp* exploités dans les boucles internes (1088+272=1360 KO) peuvent donc rester dans le cache L2 et dépasse seulement de peu la capacité du TLB. D'autre part, la taille des sous-matrices manipulées dans les boucles $nn2$ et $mm2$ sont de taille 8×544 flottants $=$ 34 KO pour A, et 16×544 flottants $=$ 68 KO pour B. Le cache L1 de 128 KO contient donc aisément ces deux sous-matrices. Ainsi on vérifie *a posteriori* que les paramètres trouvés sont bien adaptés à l'architecture sous-jacente. Les performances de `multi_6'` sont :

algorithme	`multi_6`			`multi_6'`		
mesure	nombre de défauts TLB	nombre moyen de chargements sans défaut L2	MFLOPS	nombre de défauts TLB	nombre moyen de chargements sans défaut L2	MFLOPS
matrice 800×800	$7.3\ 10^4$	691	1306	$6.1\ 10^4$	699	1312
matrice 1500×1500	$4.7\ 10^5$	733	1306	$3.6\ 10^5$	811	1352
matrice 2000×2000	$1.0\ 10^6$	770	1335	$8.5\ 10^5$	816	1362

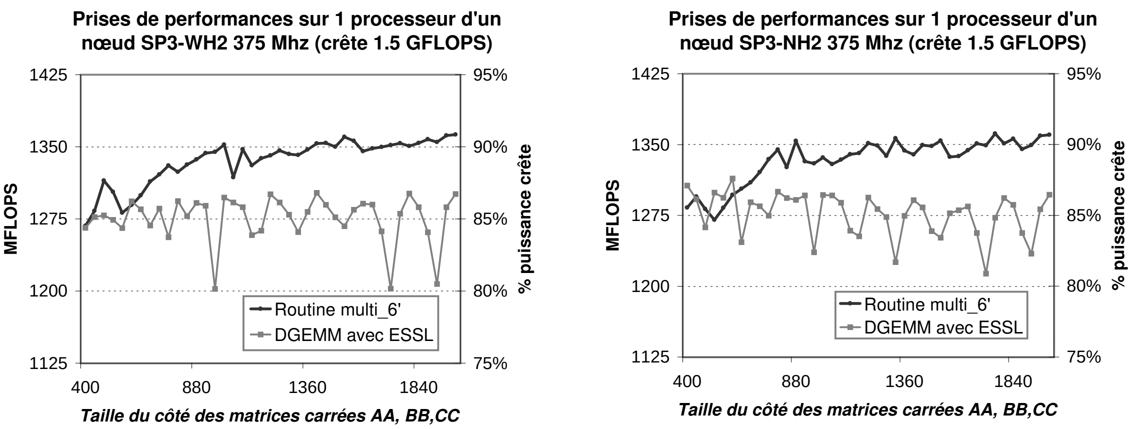


Figure 123. Comparaison de performances des routines `multi_6'` et DGEMM constructeur sur deux processeurs IBM Power 3 : WH2 et NH2

Comparons le nombre d'opérations à la seconde que réalisent l'algorithme `multi_6'` et celui de la fonction `DGEMM` de la bibliothèque IBM ESSL. Dans la Figure 123, on a en abcisse la taille du côté des matrices carrés A, B et C. Sur des matrices de taille supérieure à 400×400, l'algorithme `multi_6'` va plus vite que le `DGEMM` d'ESSL. De bons résultats ont aussi été obtenus avec la même routine sur la SGI Origin 3800. Des mesures de performances sont présentées sur la Figure 124 pour le paramétrage `multi_6'(x,x,x,-BB,x,AA,x,CC,x,4,544,4,4,100,192)`. On dispose donc de routines GEMM dont les performances sont comparables à celles fournies par les constructeurs. Cet objectif étant atteint, on va utiliser le code pour réaliser des routines GEMM et TRMM adaptées au cas particulier où la matrice A a un petit nombre de lignes.

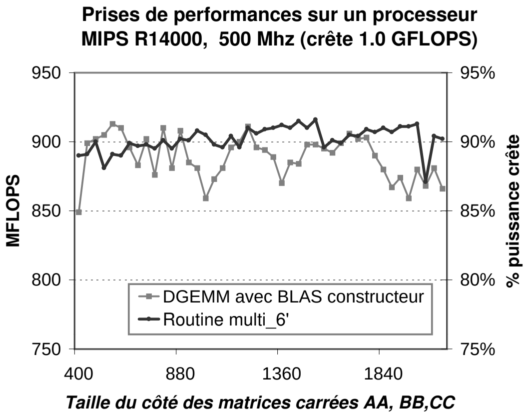


Figure 124. Comparaison de performances des routines `multi_6'` et DGEMM constructeur sur un processeur R14000 MIPS d'une SGI Origin 3800

C.3 GEMM avec une petite dimension

C.3.1 Déroulage et blocage des boucles

On considère encore la multiplication des matrices : $A^T.B = C$. La dimension dm de A est petite et égale à 9. Les tailles des matrices sont $A(9 \times dk)$, $B(dk \times dn)$, $C(9 \times dn)$. Ce cas correspond aux multiplications que l'on trouve au niveau du noyau de calcul pour un nombre de classes d'âge de parasites $K = 10$ (voir sous Section III-6.1.5.6 p. 69). L'algorithme précédent doit être adapté pour ce produit de matrices. En effet, si l'on utilise la technique du déroulage de boucle 4×4 (*c.f.* C.2.3) sur les dimensions n et m, on a un problème avec la petite dimension égale à 9 qui n'est pas multiple de 4. On réalise donc un déroulage 9×2 sur les dimensions $m \times n$. Dans la boucle interne en $kk1$, cela signifie $9+2=11$ chargements pour $2 \times 9 = 18$ instructions `fmadd`. L'algorithme `pmulti_1` (comparable à `multi_1`) suit :

```
SUBROUTINE pmulti_1(dk,dn,BB,hb,AA,ha,CC,hc,pask,pasn)
   INTEGER :: hb,ha,hc,dk,dn,pask,pasn,pasm
   REAL*8  :: BB(1:*), AA(1:*), CC(1:*)
   INTEGER :: nn1,kk1,mm1,nn2,kk2,mm2,i,j
   INTEGER :: DEEPN,DEEPM
   PARAMETER (DEEPN=2)
   PARAMETER (DEEPM=9)
   REAL*8 :: reg(0:DEEPM-1,0:DEEPN-1)

   do nn2=1, dn, pasn
      do kk2=1, dk, pask
         do nn1=nn2,MIN(nn2+pasn,dn+1)-DEEPN,DEEPN
             reg=0.0_8
             do kk1=kk2,MIN(kk2+pask-1,dk)
             FORALL(i=0:DEEPM-1,j=0:DEEPN-1) &
                     reg(i,j)=reg(i,j)+ &
                     AA(kk1+ha*i) * BB(kk1+(nn1-1)*hb+hb*j)
             enddo
             FORALL(i=0:DEEPM-1,j=0:DEEPN-1) &
                  CC(1+i+(nn1+j-1)*hc)= &
                  CC(1+i+(nn1+j-1)*hc) + reg(i,j)
       enddo
     enddo
   enddo
END SUBROUTINE pmulti_1
```

On peut appliquer les méthodes que l'on a vu dans la Section précédente pour dériver de `pmulti_1` une routine optimisée `pmulti_6`.

C.3.2 Ecriture avec pointeur et choix des paramètres

Comme dans la Section précédente, on réécrit donc l'algorithme en faisant apparaître explicitement les pointeurs. En revanche, on n'utilise pas ici la recopie des matrices dans un tampon *tmp*, car la "petite" dimension *dm* rend cette opération trop coûteuse et non rentable. L'algorithme obtenu est le suivant :

```
SUBROUTINE pmulti_6(dk,dn,BB,hb,AA,ha,CC,hc,pask,pasn,wk,wn)
   INTEGER :: hb,ha,hc,dm,dk,dn,pask,pasn,wk,wn
   REAL*8  :: BB(1:*), AA(1:*), CC(1:*)
   REAL*8 :: reg00,reg01,reg10,reg11,reg20,reg21,reg30,reg31,reg40,reg41
   REAL*8 :: reg50,reg51,reg60,reg61,reg70,reg71,reg80,reg81
   REAL*8 :: a1,a2,a3,a0,a4,a5,a6,a7,a8,b1,b0
   POINTER (pa0,a0)
   POINTER (pa1,a1)
   POINTER (pa2,a2)
   POINTER (pa3,a3)
   POINTER (pa4,a4)
   POINTER (pa5,a5)
   POINTER (pa6,a6)
   POINTER (pa7,a7)
   POINTER (pa8,a8)
   POINTER (pb0,b0)
   POINTER (pb1,b1)
   INTEGER :: nn1,kk1,nn2,kk2,nn3,kk3,i,j,p
   INTEGER :: DEEPN,DEEPM
   PARAMETER (DEEPN=2)
   PARAMETER (DEEPM=9)
```

```
do nn3=1, dn, pasn*wn
  do kk3=1, dk, pask*wk
   do nn2=nn3, MIN(nn3+pasn*(wn-1),dn), pasn
     do kk2=kk3, MIN(kk3+pask*(wk-1),dk), pask
      do nn1=nn2,MIN(nn2+pasn,dn+1)-DEEPN,DEEPN
          reg00=0;reg01=0;reg10=0;reg11=0;
          reg20=0;reg21=0;reg30=0;reg31=0;
          reg40=0;reg41=0;reg50=0;reg51=0;
          reg60=0;reg61=0;reg70=0;reg71=0;reg80=0;reg81=0;
          pb0=LOC(BB(kk2+(nn1-1)*hb))
          pb1=LOC(BB(kk2+(nn1)*hb))
          pa0=LOC(AA(kk2+ha*0))
          pa1=LOC(AA(kk2+ha*1))
          pa2=LOC(AA(kk2+ha*2))
          pa3=LOC(AA(kk2+ha*3))
          pa4=LOC(AA(kk2+ha*4))
          pa5=LOC(AA(kk2+ha*5))
          pa6=LOC(AA(kk2+ha*6))
          pa7=LOC(AA(kk2+ha*7))
          pa8=LOC(AA(kk2+ha*8))
          do kk1=0,MIN(pask-1,dk-kk2),1
             reg00=reg00+a0 * b0
             reg01=reg01+a0 * b1
             reg10=reg10+a1 * b0
             reg11=reg11+a1 * b1
             reg20=reg20+a2 * b0
             reg21=reg21+a2 * b1
             reg30=reg30+a3 * b0
             reg31=reg31+a3 * b1
             reg40=reg40+a4 * b0
             reg41=reg41+a4 * b1
             reg50=reg50+a5 * b0
             reg51=reg51+a5 * b1
             reg60=reg60+a6 * b0
             reg61=reg61+a6 * b1
             reg70=reg70+a7 * b0
             reg71=reg71+a7 * b1
             reg80=reg80+a8 * b0
             reg81=reg81+a8 * b1
             pb0=pb0+8
             pb1=pb1+8
             pa0=pa0+8
             pa1=pa1+8
             pa2=pa2+8
             pa3=pa3+8
             pa4=pa4+8
             pa5=pa5+8
             pa6=pa6+8
             pa7=pa7+8
             pa8=pa8+8
          enddo
          CC(1+0+(nn1+0-1)*hc)=CC(1+0+(nn1+0-1)*hc)+reg00
          CC(1+0+(nn1+1-1)*hc)=CC(1+0+(nn1+1-1)*hc)+reg01
          CC(1+1+(nn1+0-1)*hc)=CC(1+1+(nn1+0-1)*hc)+reg10
          CC(1+1+(nn1+1-1)*hc)=CC(1+1+(nn1+1-1)*hc)+reg11
          CC(1+2+(nn1+0-1)*hc)=CC(1+2+(nn1+0-1)*hc)+reg20
          CC(1+2+(nn1+1-1)*hc)=CC(1+2+(nn1+1-1)*hc)+reg21
          CC(1+3+(nn1+0-1)*hc)=CC(1+3+(nn1+0-1)*hc)+reg30
```

```
                  CC(1+3+(nn1+1-1)*hc)=CC(1+3+(nn1+1-1)*hc)+reg31
                  CC(1+4+(nn1+0-1)*hc)=CC(1+4+(nn1+0-1)*hc)+reg40
                  CC(1+4+(nn1+1-1)*hc)=CC(1+4+(nn1+1-1)*hc)+reg41
                  CC(1+5+(nn1+0-1)*hc)=CC(1+5+(nn1+0-1)*hc)+reg50
                  CC(1+5+(nn1+1-1)*hc)=CC(1+5+(nn1+1-1)*hc)+reg51
                  CC(1+6+(nn1+0-1)*hc)=CC(1+6+(nn1+0-1)*hc)+reg60
                  CC(1+6+(nn1+1-1)*hc)=CC(1+6+(nn1+1-1)*hc)+reg61
                  CC(1+7+(nn1+0-1)*hc)=CC(1+7+(nn1+0-1)*hc)+reg70
                  CC(1+7+(nn1+1-1)*hc)=CC(1+7+(nn1+1-1)*hc)+reg71
                  CC(1+8+(nn1+0-1)*hc)=CC(1+8+(nn1+0-1)*hc)+reg80
                  CC(1+8+(nn1+1-1)*hc)=CC(1+8+(nn1+1-1)*hc)+reg81
               enddo
            enddo
         enddo
         enddo
        enddo
 END SUBROUTINE pmulti_6
```

Pour cet algorithme, on a cherché le jeu de paramètres qui donnent les meilleures performances pour des grandes valeurs (> 1000) de $dn = dk$. Celui que nous avons pris est : `pmulti_6(x,x,BB,x,AA,x,CC,9,320,64,100,12)`. Les résultats sont donnés dans les figures 125 et 126 pour les IBM SP3. A la lecture de ces courbes, la conclusion est claire : le DGEMM n'est pas optimisé pour cette forme de matrice sur les IBM SP3. Sur l'ORIGIN 3800 (Figure 127), on remarque que les résultats sont de peu à l'avantage de la routine constructeur.

La différence de performance que l'on constate par rapport à un DGEMM constructeur est dû à deux raisons principales. Premièrement, par rapport au déroulage 4×4 fréquemment utilisé, le déroulage 9×2 que l'on emploie implique deux références à B qui est une matrice de plus grande taille que A. Si l'on considère qu'en moyenne la plus petite matrice A se trouve dans le cache, ceci permet de réduire l'utilisation de la bande passante mémoire en accédant à seulement deux réels de B par itération de $kk1$. Deuxièmement, la matrice A ayant des colonnes de hauteur 9, ce déroulage permet d'être certain de n'accéder qu'une seule et unique fois à un élément de B. Il y a donc économie de bande passante mémoire par rapport à des accès multiples à chaque élément de B ; détaillons ce point.

La complexité de l'algorithme est de l'ordre de $2 \times 9 \times dn^2$. Le nombre d'opérations est asymptotiquement 18 fois plus grand que le nombre de lecture/écriture ; il n'y a donc pas beaucoup plus d'opérations que de chargements mémoire, comme c'est le cas pour un DGEMM sur des matrices de plus grandes tailles. En fait, dans le cas qui nous intéresse, la puissance en MFLOPS est limité par la bande passante mémoire. En effet sur le WH2, la bande passante de la mémoire vers les caches et les 4 processeurs est de 1.5 GB/s (soit en moyenne 0.375 GB/s par processeur). En 16 cycles d'horloge, on peut donc amener au maximum deux flottants double précision (16 octets) de la mémoire vers deux registres. Supposons que A soit dans le cache, avec **deux** éléments de B, on peut effectuer 18 additions et 18 multiplications dans la boucle la plus interne de l'algorithme, soit 9 cycles horloge au mieux (deux `fmadd` par cycle). En résumé, pour deux éléments de B, on a besoin de 16 cycles de chargement pour 9 cycles de calcul. On ne peut pas recouvrir des chargements par les calculs ! Le processeur a un taux d'utilisation maximal de $\frac{9}{16}$ de la puissance crête, c'est-à-dire 843 MFLOPS. On constate les performances sur la Figure 125 (à droite).

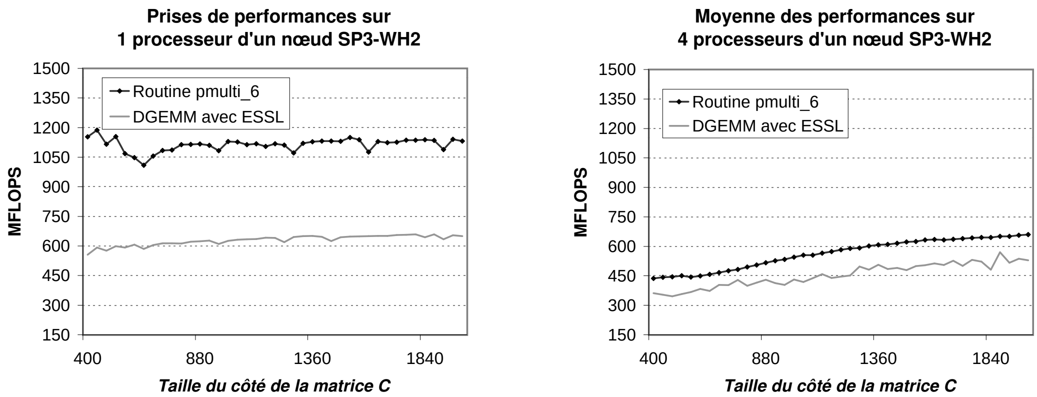


Figure 125. Comparaison des performances des routines `pmulti_6` et DGEMM constructeur sur un seul processeur IBM Power 3 d'un nœud WH2, avec la moyenne des performances lorsque tous les processeurs du nœuds WH2 (375 Mhz) calculent simul-

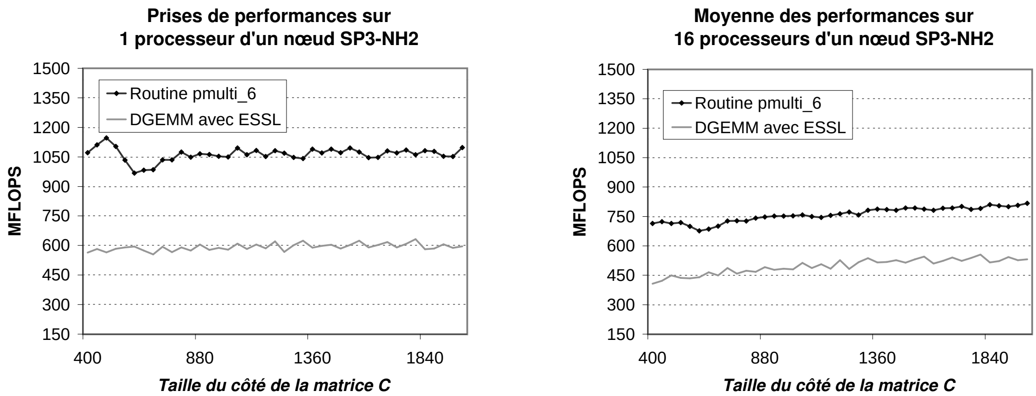


Figure 126. Comparaison des performances des routines `pmulti_6` et DGEMM constructeur sur un processeur IBM Power 3, avec la moyenne des performances lorsque tous les processeurs du nœuds NH2 (375 Mhz) calculent simultanément

Considérons maintenant un seul processeur actif par nœud effectuant une multiplication matricielle. Sur le WH2, la bande passante de la mémoire vers le processeur actif est maintenant de 1.5 GB/s. En quatre cycles d'horloge, on peut charger au maximum deux flottants double précision (16 octets) depuis la mémoire. Avec deux éléments de B (A dans le cache), on peut effectuer 18 `fmadd`, soit 9 cycles d'horloge. Il est donc maintenant possible de recouvrir le temps et les latences de chargement (4 cycles) par des calculs (9 cycles). Si l'on ne fait pas de recopie ou de rechargements de matrices et que les opérations sont bien ordonnancées et optimisées, on peut être proche de la performance de crête. On le constate d'ailleurs sur la Figure 125 (à gauche). Le temps critique est donc clairement le temps de chargement des matrices. Lorsque dn est grand, les algorithmes `pmulti` évitent de recharger les données de A ; les performances augmentent sensiblement par rapport aux BLAS 3 constructeur. Cette optimisation est réalisée grâce au blocage des boucles et le déroulage 2×9.

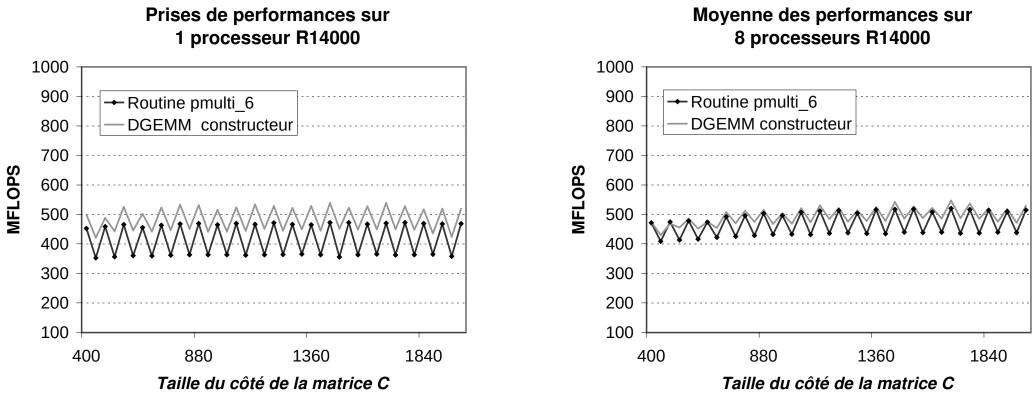


Figure 127. Comparaison des performances des routines `pmulti_6` et DGEMM constructeur sur un seul processeur R14000 (500 Mhz), avec la moyenne des performances lorsque plusieurs processeurs de l'Origin 3800 calculent simultanément

Sur un nœud NightHawk II, la bande passante vers la mémoire et par processeur est multipliée par 2.6. Le problème de saturation rencontré sur le WinterHawk est moindre (voir Figure 126) mais encore présent, contrairement à ce que l'on pouvait attendre compte tenue de la bande passante plus élevée.

C.4 TRMM avec une petite dimension

L'algorithme du TRMM effectue la multiplication d'une matrice A de dimension $dm \times dn$ par une matrice triangulaire B de dimension $dn \times dn$. Ici, on considèrera B comme triangulaire supérieure. Le résultat de la multiplication est stocké dans la matrice A par *écrasement* des données initiales. L'un des algorithmes possibles est présenté ci-dessous. L'ordre des opérations est important, car on ne peut écraser un élément $a_{.,.}$ que lorsqu'il a contribué aux calculs qui en dépendent. Par exemple, pour $dn = 8$, on ne peut pas écraser $a_{2,5}$ avant d'avoir fait les calculs où apparaissent la variable $a_{2,5}$: $a_{2,6} := a_{2,6} + a_{2,5}\, b_{5,6}$, $a_{2,7} := a_{2,7} + a_{2,5}\, b_{5,7}$, $a_{2,8} := a_{2,8} + a_{2,5}\, b_{5,8}$. On en conclut que l'ordonnancement des calculs doit respecter des dépendances. Notons $(a) \mapsto (b)$ le fait que les opérations de (a) doivent être effectuées avant celles de (b). Nous avons pour cet algorithme TRMM $(\forall k,\ \forall m)$:

$$(\forall n > k,\ a_{m,n} := a_{m,n} + a_{m,k}\, b_{k,n}) \mapsto a_{m,k} := a_{m,k}\, b_{k,k}$$

Procédure `algo_trmm(dm,dn,A,B)` {
  Pour $m := 1$ **à** dm **par pas de** `1` **faire** {
  . **Pour** $n := dn$ **à** `1` **par pas de** `-1` **faire** {
  . . $a_{m,n} := a_{m,n}\, b_{n,n}$;
  . . **Pour** $k := n-1$ **à** `1` **par pas de** `-1` **faire** {
  . . . $a_{m,n} := a_{m,n} + a_{m,k}\, b_{k,n}$;
  . . }
  . }
  }
}

Figure 128. Algorithme du produit matrice$\times$matrice triangulaire supérieure

Un algorithme par blocs correspondant au TRMM respecte ces contraintes de dépendance. Supposons que l'on pave de blocs rectangulaires de taille $pas_k \times pas_n$ la matrice B. Pour simplifier, on suppose que le nombre de lignes de B est un multiple de pas_k, et que celui des colonnes est multiple de pas_n. On nomme les sous-matrices de B, $\beta_{k',n'}$ et celles de A, $\alpha_{k'}^{pas_k}$. La sous-matrice $\beta_{k',n'}$ correspond aux éléments $b_{k,n}$ de la matrice B avec $(k,n) \in [k'\,pas_k, (k'+1)\,pas_k - 1] \times [n'\,pas_k, (n'+1)\,pas_n - 1]$, et $\alpha_{k'}^{pas_k}$ correspond aux éléments $a_{m,k}$ de la matrice A avec $(a,m) \in [1,dm] \times [k'\,pas_k, (k'+1)\,pas_k - 1]$. Pour les sous-matrices $\alpha_{k'}^{pas}$, un pas est mis en paramètre car l'algorithme va lire des sous-matrices de pas pas_k et générer des sous-matrices de pas pas_n.

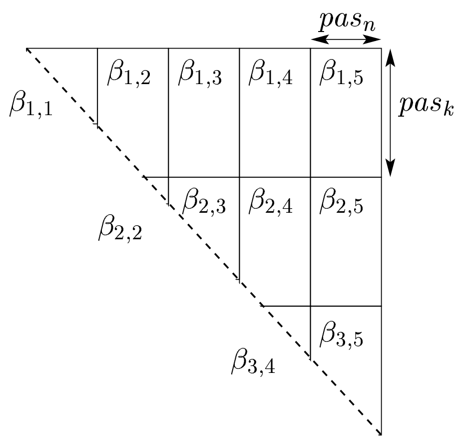


Figure 129. matrice B avec ses sous-matrices $\beta_{.,.}$

Les sous-matrices $\beta_{k',n'}$ que l'on va manipuler (voir Figure 129) sont rectangulaires mais peuvent ne pas être pleines. L'algorithme par blocs va calculer la matrice A résultat à l'aide des données des sous-matrices $\alpha_{k'}^{pas_k}$ et $\beta_{k',n'}$. Pour que celui-ci soit valide, il doit respecter les dépendances des calculs que l'on a énoncées. L'algorithme du TRMM dans lequel on utilise maintenant les sous-matrices α et β est présenté en Figure 130.

Le calcul de $maxk'$ permet de trouver la colonne k' correspondant à la sous-matrice diagonale pour un n' donné. Ce détail mis à part, l'algorithme par blocs ressemble à l'algorithme scalaire précédent. On peut montrer que cet algorithme respecte les dépendances

Procédure `algo_trmm(dn',A,B,`pas_k`,`pas_n`)` {
  Pour $n' := dn'$ **à** 1 **par pas de** -1 **faire** {
  . $maxk' := n'\,pas_n - \mathrm{mod}(n'\,pas_n - 1, pas_k)$
  . $\alpha_{n'}^{pas_n} := \alpha_{maxk'}^{pas_k}\,\beta_{maxk',n'}$;
  . **Pour** $k' := maxk'$ **à** 1 **par pas de** -1 **faire** {
  . . $\alpha_{n'}^{pas_n} := \alpha_{n'}^{pas_n} + \alpha_{k'}^{pas_k}\,\beta_{k',n'}$;
  . }
  }
}

Figure 130. Algorithme du produit matrice×matrice triangulaire supérieure par décomposition en sous-matrices

entre les calculs.

Ci-après, l'algorithme du TRMM par blocs qui en est issu. La routine `tmulti_1` permet de multiplier une matrice A de taille $9 \times dn$ par une matrice triangulaire supérieure B de dimension $dn \times dn$. Le résultat est stocké dans A par écrasement de la matrice initiale. On réalise un déroulage 9×2 sur les dimensions $m \times n$ et un blocage sur les deux dimensions de B de taille $pas_k \times pas_n$.

```
SUBROUTINE tmulti_1(dn,BB,hb,AA,ha,pask,pasn)
   INTEGER :: ha,hb,dn,pask,pasn,DEEPM
   REAL*8  :: AA(1:*), BB(1:*)
   INTEGER :: nn1,kk1,mm1,nn2,kk2,mm2,i,j,maxn2,maxk2
   PARAMETER (DEEPM=9)
   REAL*8 :: reg(0:DEEPM-1,0:1)
   do nn2 = dn-MOD(dn-1,pasn), 1, -pasn
      maxn2 = MIN(nn2+pasn-1,dn)
      do kk2 = maxn2-MOD(maxn1,pask), 1, -pask
         do nn1 = maxn1, nn2, -2
            maxk2 = MIN(nn1,kk2+pask-2)
            IF (maxk2 .GE. kk2) THEN
               reg=0.0_8
               do kk1=kk2,maxk2,1
                  FORALL(i=0:DEEPM-1,j=0:1) &
                       reg(i,j)=reg(i,j)+ &
                       AA(1+(kk1-1)*ha+i) * BB(kk1+(nn1-1)*hb+hb*j)
               enddo
               IF (maxk2 .eq. nn1) THEN
                  FORALL(i=0:DEEPM-1) &
                       reg(i,1)=reg(i,1)+ AA(1+maxk2*ha+i) * BB(maxk2+1+nn1*hb)
                  FORALL(i=0:DEEPM-1,j=0:1) &
                       AA(1+i+(nn1+j-1)*ha)= reg(i,j)
               ELSE
                  FORALL(i=0:DEEPM-1,j=0:1) &
                       reg(i,j)=reg(i,j)+ &
                       AA(1+maxk2*ha+i) * BB(maxk2+1+(nn1-1+j)*hb)
                  FORALL(i=0:DEEPM-1,j=0:1) &
                       AA(1+i+(nn1+j-1)*ha)= AA(1+i+(nn1+j-1)*ha) + reg(i,j)
               ENDIF
            ENDIF
         enddo
      enddo
   enddo
END SUBROUTINE tmulti_1
```

Comme dans l'étude précédente pour le GEMM, il est possible de faire un second blocage des boucles pour tenir compte des différents niveaux de cache. D'autre part, l'écriture à l'aide de pointeurs permet un léger gain de performance. La routine correspondant à ces modifications se nomme `tmulti_6` ; son code n'est pas décrit ici. On a recherché, dans l'espace des tailles de blocage, les paramètres qui conduisent aux meilleures performances pour `tmulti_6`. On donne ci-après les mesures de performances de `DTRMM` sur différentes machines.
On conclut donc que l'utilisation d'un algorithme optimisé améliore les performances par rapport au DTRMM de la bibliothèque ESSL dans le cadre que nous nous sommes fixés. D'autre part, on remarque que le type de calculs effectués peut engendrer une saturation de la bande passante mémoire lorsqu'il est employé simultanément sur tous les processeurs

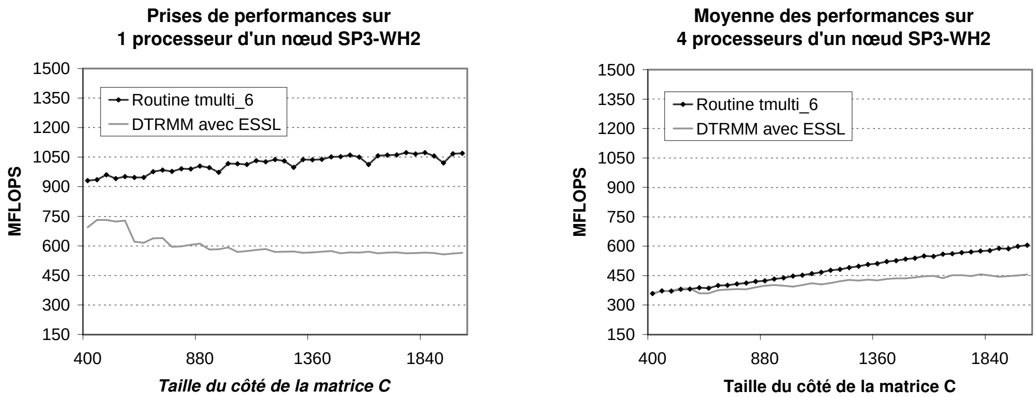


Figure 131. Comparaison des performances des routines `tmulti_6` et TRMM constructeur sur un seul processeur IBM Power 3 d'un nœud, avec la moyenne des performances lorsque tous les processeurs du nœud WH2 (375 Mhz) calculent simultanément

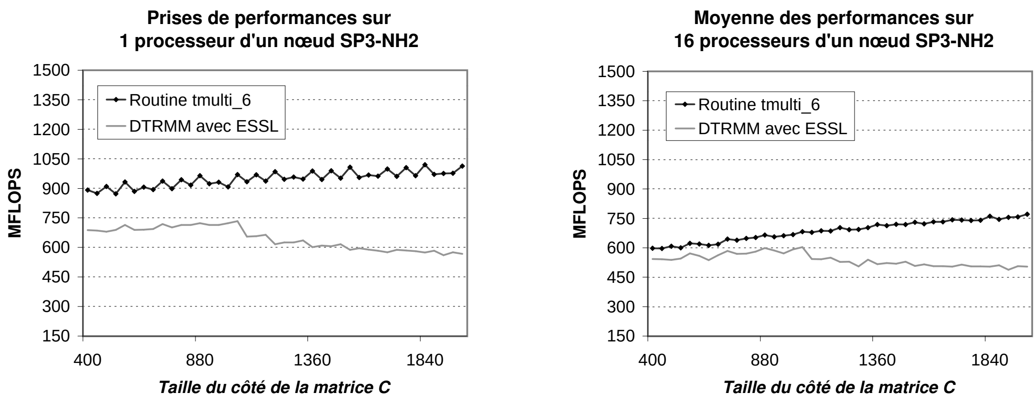


Figure 132. Comparaison des performances des routines `tmulti_6` et TRMM constructeur sur un seul processeur IBM Power 3 d'un nœud, avec la moyenne des performances lorsque tous les processeurs du nœud NH2 (375 Mhz) calculent simultanément

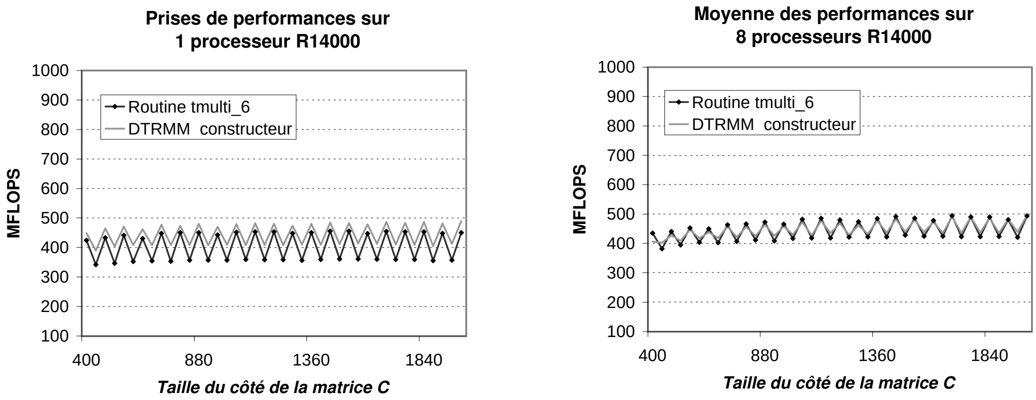


Figure 133. Comparaison des performances des routines `tmulti_6` et TRMM constructeur sur un seul processeur R14000, avec la moyenne des performances lorsque plusieurs processeurs de l'Origin 3800 calculent simultanément

Annexe D

Valeurs des paramètres de la fonction F

Il s'agit ici de calculer la valeur des inconnues α, β, δ et γ de la Section 4.3.4 sur la fonction F. Les contraintes énoncées alors s'écrivent de la manière suivante (on suppose $\gamma \neq 0$) :

$$\begin{cases} a_0 + \gamma\, log(\beta+\delta) = 1, \\ a_0 + \gamma\, log(\beta + \delta\, xi) = a_0, \\ e^{\alpha\, xi} - 1 = a_0, \\ \gamma\,\delta \,/\, (\beta + \delta\, xi) = \alpha\, e^{\alpha\, xi}, \end{cases} \quad \Leftrightarrow \quad \begin{cases} \gamma\, log(\beta+\delta) = 1 - a_0, \\ \gamma\, log(\beta + \delta\, xi) = 0, \\ \alpha\, xi = log(1 + a_0), \\ \gamma\,\delta \,/\, (\beta + \delta\, xi) = \alpha\,(1 + a_0), \end{cases}$$

$$\Rightarrow \begin{cases} \gamma\, log(\beta+\delta) = 1 - a_0, \\ \beta + \delta\, xi = 1, \\ \alpha = log(1 + a_0)/xi, \\ \gamma\,\delta \,/\, (\beta + \delta\, xi) = \alpha\,(1 + a_0), \end{cases} \quad \text{soit} \quad \begin{cases} \gamma\, log(\beta+\delta) = 1 - a_0, \\ \beta + \delta\, xi = 1, \\ \alpha = log(1 + a_0)/xi, \\ \gamma\,\delta = \alpha\,(1 + a_0). \end{cases}$$

On peut déduire directement $\alpha = log(1 + a_0)/xi$. En revanche, pour trouver δ, on doit résoudre l'équation suivante :

$$\gamma\, log(1 - \delta\, xi + \delta) = 1 - a_0, \text{ soit}$$
$$log(1 + \delta(1 - xi))/(\delta(1 - xi)) = (1 - a_0)/(\alpha(1 + a_0)(1 - xi)).$$

Posons $z = \delta(1 - xi)$ et $y = xi(1 - a_0)/((1 + a_0)(1 - xi)log(1 + a_0))$, on a alors

$$log(z+1)/z = y \;. \tag{47}$$

On connaît xi, a_0 et α, donc y et l'on recherche δ, ce qui revient à chercher z. On a les inégalités $0 < xi < 1$ et $0 < a_0 < 1$; le système d'équations posé permet de déduire que α, δ puis γ, z, y sont des réels strictement positifs. D'autre part, l'équation $log(z+1) = z\,y$

permet de conclure que y est nécessairement inférieure à 1 ($0 < y < 1$). De plus, cette dernière équation a deux solutions pour z, dont l'une d'elle est $z_0 = 0$ et l'autre $z_1 > 0$.

Pour inverser la fonction $log(z+1)/z = y$ et trouver z en fonction de y, on ne peut pas utiliser seulement les fonctions mathématiques classiques. Considérons l'emploi de la fonction W de Lambert, définie par $W_c(x)\, e^{W_c(x)} = x$. Une présentation de cette fonction et de ses utilisations est donnée dans [17]. Pour des valeurs de x dans l'intervalle réel $[-1/e, 0[$, la valeur de c peut être égale soit à 0, soit à –1. Dans ces conditions, $W_0(x)$ et $W_{-1}(x)$ donnent les deux solutions réelles de l'équation $W_c(x)\, e^{W_c(x)} = x$. Ces fonctions sont définies sur les intervalles $W_0(x) : [-1/e, +\infty[\rightarrow [-1, +\infty[$ et $W_{-1}(x) : [-1/e, 0[\rightarrow] - \infty, -1]$. Soit la fonction $g_c(y)$ susceptible d'être la fonction inverse recherchée $z(y)$:

$$g_c(y) = \frac{1}{y} log(-\frac{1}{y}\, W_c(-y\, e^{-y})) \ .$$

On vérifie aisément que $-y\, e^{-y}$ est dans l'intervalle $[-1/e, 0[$ donc *a priori* que la variable c a pour valeur 0 ou -1. Par définition, on a :

$$W_c(-y\, e^{-y})\, e^{W_c(-y\, e^{-y})} = -y\, e^{y}, \quad \text{donc} \quad W_c(-y\, e^{-y})/(-y\, e^{y}) = e^{-W_c(-y\, e^{-y})} \ . \tag{48}$$

Calculons la valeur de $q = log(z+1)/z$ avec $z = g_c(y)$. Avec l'hypothèse que $\exists c, g_c(y) > 0$ pour vérifier que $q = y$:

$$\begin{aligned}
q &= log(g_c(y) + 1)/g_c(y), \\
q &= log(\frac{1}{y}\, log(-\frac{1}{y}\, W_c(-y\, e^{-y})) + 1)/g_c(y) \\
q &= log(\frac{1}{y}\, log(-\frac{1}{y}\, W_c(-y\, e^{-y})) + \frac{1}{y} log(\frac{1}{e^{-y}}))/g_c(y), \\
q &= log(\frac{1}{y}\, log(-\frac{1}{y}\, W_c(-y\, e^{-y})\frac{1}{e^{-y}}))/g_c(y), \\
q &= log(\frac{1}{y}\, log(\frac{W_c(-y\, e^{-y})}{-y\, e^{-y}}))/g_c(y).
\end{aligned}$$

En utilisant l'égalité (47), on obtient :

$$q = log(\tfrac{1}{-y} W_c(-y\, e^{-y}))/g_c(y), \quad \text{puis avec l'expression de } g_c(y),$$
$$q = (y\, g_c(y))/g_c(y) = y \ .$$

La fonction $g_c(y)$ est donc bien la fonction inverse de l'équation (47) que l'on recherchait avec la condition $\exists c, g_c(y) > 0$. Pour examiner cette dernière condition, posons $x = -ye^{-y}$. Comme $-1 < -y < 0$, on a (par définition de W_0) $W_0(x) = -y$. Calculons $g_0(y)$ et

$g_{-1}(y)$ en utilisant la variable x :

$$g_0(y) = \frac{1}{y}\,log(\frac{1}{-y}\,W_0(-y\,e^{-y})) = \frac{1}{y}\,log(\frac{W_0(x)}{-y}) = \frac{1}{y}\,log(\frac{-y}{-y}),$$
$$\text{donc } g_0(y) = 0\ ;$$
$$g_{-1}(y) = \frac{1}{y}\,log(\frac{1}{-y}\,W_{-1}(-y\,e^{-y})) = \frac{1}{y}\,log(\frac{W_{-1}(x)}{-y})\ ;$$
$$\text{comme } \ x \in]1/e, 0[\Rightarrow W_{-1}(x) < -1, \text{ et que } -y \in]-1, 0[$$
$$\text{on a } \frac{W_{-1}(x)}{-y} > 0,$$
$$\text{et } g_{-1}(y) > 0.$$

La fonction inverse recherchée est en conséquence $z = g_{-1}(y)$. On en déduit immédiatement δ par la formule :

$$\delta = z/(1-xi) = \frac{1}{y}ln(-\frac{1}{y}W_{-1}(-y\,e^{-y}))/(1-xi),$$
$$\text{puis } \gamma = \alpha(1+a_0)/\delta \text{ et } \beta = 1 - \delta\,xi.$$

En pratique pour calculer la fonction $W_{-1}(x)$ sachant x, on peut utiliser un des algorithmes présents dans la littérature[47][17], une dichotomie, ou alors un logiciel tel que `Maple`.

47. Le résultat est obtenu après quelques itérations de l'algorithme qui converge vers la solution.

Annexe E

Liste des publications de l'auteur

Dans cette partie, deux symboles sont utilisés afin de distinguer les deux domaines principaux aborder par les travaux interdisplinaires présentés. Ce qui relève du domaine de l'algorithmique parallèle et de la simulation haute-performance est suivi du signe ⋆ dans la liste qui suit, et ce qui correspond au domaine de la modélisation mathématique de dynamique de populations comporte le signe †.

Publications

Congrès internationaux avec comité de sélection et avec actes :

[1] Langlais (M.), Latu (G.), Roman (J.) et Silan (P.). Parallel numerical simulation of a marine host-parasite system. *In : Europar'99 Parallel Processing*, éd. par Amestoy (P.), Berger (P.), Daydé (M.), Duff (I.), Frayssé (V.), Giraud (L.) et Ruiz (D.). pp. 677–685. LNCS 1685 - Springer Verlag. 1999. ⋆

[2] Langlais (M.), Latu (G.), Roman (J.) et Silan (P.). Stochastic simulation of a marine host-parasite system using a hybrid OpenMP/MPI programming. *In : Europar'02 Parallel Processing*. pp. 436–446. LNCS 2400 - Springer Verlag. 2002. ⋆

Revues nationales avec comité de lecture :

[3] Silan (P.), Langlais (M.) et Latu (G.). Dynamique des populations de monogènes, ectoparasites de téléostéens : stratégies démographiques et implications mathématiques. *Ecologie*, vol. 30, 1999, pp. 247–260. †

[4] Latu (G.). Solution parallèle pour un problème de dynamique de population. *Technique et Science Informatiques*, vol. 19, juin 2000, pp. 767–790. ⋆

Congrès nationaux avec comité de sélection et avec actes :

[5] Latu (G.). Solution parallèle pour un problème de dynamique des populations. *In : RenPar'11, 11ème Rencontres francophones du parallélisme des architectures et des systèmes (Rennes)*, pp. 121–126. 1999. ⋆

[6] Latu (G.). Simulation stochastique parallèle d'un système hôte-parasite par programmation hybride MPI+OpenMP. *In : RenPar'14, 14ème Rencontres francophones du parallélisme des architectures et des systèmes (Hammamet)*, pp. 83–90. 2002. ⋆

Article soumis en revue internationale :

[7] Langlais (M.), Latu (G.), Roman (J.) et Silan (P.). Performance analysis and qualitative results of an efficient parallel stochastic simulator for a marine host-parasite system. *Version étendue de l'article d'Europar'02 dans un numéro spécial de la revue " Concurrency & Computation: Practice and Experience ".* ⋆ †

Chapitre de livre :

[8] Silan (P.), Caltran (H.) et Latu (G.). Ecologie et dynamique des populations de monogènes, ectoparasites de téléostéens marins : approche et contribution montpelliéraine. *Contribution à l'ouvrage " Taxonomy, ecology and evolution of metazoan parasites " en hommage à Louis Euzet (Ed. C. Combes & J. Jourdane) Presses Universitaires de Perpignan (sous presse).* †

Article de vulgarisation :

[9] Langlais (M.), Latu (G.), Roman (J.) et Silan (P.). Simulation numérique parallèle d'un système hôte-parasite. *Journal du Centre National Universitaire Sud de Calcul (nouvellement renommé Journal du CINES)*, vol. 69, 1999, pp. 2–3. ⋆

Atelier avec actes :

[10] Latu (G.) et Langlais (M.). Simulation numérique parallèle d'un système hôte-parasite, deux modèles : déterministe et stochastique. *In : actes de l'atelier " Dynamique des Processus Epidémiques "*. 2001. Col de Porte - Chartreuse. †

Mémoire de DEA :

[11] Latu (G.). *Solution parallèle pour un simulateur d'un système hôte-parasite en environnement marin.* 1998. DEA d'informatique, Université Bordeaux 1. ⋆ †

Articles en cours de rédaction :

[12] Langlais (M.), Latu (G.), Roman (J.) et Silan (P.). Links between aggregation and regulation in a deterministic model of a host-parasite system. *A soumettre dans la revue Ecological Modelling.* †

[13] Langlais (M.), Latu (G.), Roman (J.) et Silan (P.). Highly scalable parallel simulator for a host-parasite system. *A soumettre dans la revue Multiscale Modeling and Simulation du SIAM.* ⋆

Communications

Exposés à des rencontres internationales :

[14] Latu (G.). Numerical simulation of a complex biological system. mai 2000. *European ACTC Workshop 2000, Paris.* ⋆

[15] Latu (G.). Contribution to a host-parasite system model by high-performance computing. august 2000. *DESTOBIO 2000, Second International Conference on Deterministic and Stochastic Modeling of Biointeraction, West Lafayette, Indiana, US.* †

[16] Latu (G.) et Silan (P.). Comparative study of a deterministic and an individual-based model for a host-parasite system simulation. august 2002. *Eureco'02, IX European Ecological Congress, Lund, Sweden.* †

Exposés à des rencontres nationales :

[17] Latu (G.). Modélisation du système Hôte-Parasite "Bar-Diplectanum" et calculs parallèles associés. décembre 1998. *Atelier intitulé " Epidémiologie et Modélisation ", Aussois.* † ⋆

[18] Latu (G.). Simulation individu-centrée d'un système hôte-parasite. mai 2001. *COREV Spring School Lalonde les Maures, (Var, France).* †

[19] Latu (G.). Comparaison qualitative de modèles stochastique et déterministe d'un système hôte-parasite. mai 2002. *COREV Spring School, Ile de Berder (Morbihan, France).* †

[20] Latu (G.) et Silan (P.). Modélisation déterministe ou individu centré de la dynamique d'un système hôte-parasite : application au Bar (Téléostéen) et à ses Monogènes. juin 2002. *Exposé au colloque écologie des populations et des communautés animales en afrique du nord et atelier scientifique 2002 de la Société Française d'Ecologie. Université Paul Sabatier, Toulouse.* †

Exposés à des séminaires :

[21] Latu (G.). Solution parallèle pour un simulateur d'un système hôte-parasite. septembre 1998. *Exposé au groupe de travail ALiENor (Talence, France).* ⋆

[22] Silan (P.) et Latu (G.). Biomathématique et parallélisme : application à la dynamique de populations de parasites. juin 2001. *Exposé au CINES (Montpellier, France).* † ⋆

[23] Langlais (M.), Latu (G.), Roman (J.). Algorithmique parallèle et performances de deux simulations déterministe et stochastique pour un système hôte-parasite. mars 2002. *Deux exposés au groupe de travail ScAlApplix (Talence, France).* ⋆

[24] Latu (G.). Simulation parallèle haute-performance d'un système hôte-parasite. mai 2002. *Exposé à la réunion de l'INRIA UR Futurs (Talence, France).* ⋆

[25] Latu (G.). Algorithmique parallèle dédiée à la simulation déterministe ou individu-centré d'un système hôte-macroparasite. octobre 2002. *Invité au séminaire de biologie quantitative de l'Institut Pasteur (Paris, France).* †

Bibliographie

[1] Adamson (M.). – Population models of parasites. – september 2001. course notes in Parasitology : `http://www.zoology.ubc.ca/~adamson/biol328`.

[2] Akçakaya (H.R.), Burgman (M.A.) et Ginzburg (L.R.). – *Applied Population Ecology.* – Sinauer Associates, Inc., 1999.

[3] Al-Battran (S.), Field (A.J.), Wiley (R.L.) et Woods (J.). – Parallel simulation of plankton ecology. *In : Proceedings of the IASTED International Conference Modelling And Simulation, Philadelphia, USA.* – pp. 259–263.

[4] Anderson (E.), Bai (Z.), Bischof (J.), Demmel (J.), Dongarra (J.J.), DuCroz (J.), Greenbaum (A.), Hammarling (S.), McKenney (A.), Ostrouchov (S.) et Sorensen (D.). – *LAPACK User's Guide*, 1995, SIAM Publications, 2nd édition.

[5] Anderson (R.M.) et Gordon (D.M.). – Processes influencing the distribution of parasite numbers within host populations with special emphasis on parasite-induced host mortalities. *Parasitology*, vol. 85, 1982, pp. 373–398.

[6] Anderson (R.M.) et May (R.M.). – *Infectious Diseases of Humans Dynamic and Control.* – Oxford University Press, 1992.

[7] Baveco (J.M.) et Lingeman (R.). – An object-oriented tool for individual-oriented simulation: Host-parasitoid system application. *Ecological Modelling*, vol. 61, 1992, pp. 267–286.

[8] Bedau (Mark). – Can unrealistic computer models illuminate theoretical biology? *In : Genetic and Evolutionary Conference Workshop Program, Orlando, Florida*, éd. par Wu (A.).

[9] Bernaschi (M.), Castiglione (F.) et Succi (S.). – A parallel algorithm for the simulation of the immune response. *In : WAE '97 Workshop on Algorithm Engineering, Venice, Italy*, pp. 198–208.

[10] Berry (M.W.) et Minser (K.S.). – Distributed Land-Cover Change Simulation Using PVM and MPI. *In : Proc. of the Land Use Modeling Workshop, 1997.*

[11] Booth (G.). – Gecko: A continuous 2-d world for ecological modeling. *Artificial Life Journal*, vol. 3, n° 3, 1997, pp. 147–163.

[12] Bouloux (C.). – *Modélisation, simulations et analyse mathématique de systèmes hôte-parasite.* – Thèse de doctorat, Université Bordeaux I, décembre 1997.

[13] Bouloux (C.), Langlais (M.) et Silan (P.). – A marine host-parasite model with direct biological cycle and age structure. *Ecological Modelling*, vol. 107, 1998, pp. 73–86.

[14] Caswell (H.). – *Matrix population models: construction, analysis, and interpretation.* – Second edition. Sunderland, Mass. Sinauer Associates Inc., 2001.

[15] Combes (C.). – *Interactions durables. Écologie et évolution du parasitisme.* – Éditions Masson, 1995.

[16] Coquillard (P.) et Hill (D.R.C.). – *Modélisation et simulation d'écosystèmes - des modèles déterministes aux simulations à événements discrets.* – Masson, 1997.

[17] Corless (R. M.), Gonnet (G. H.), Hare (D. E. G.), Jeffrey (D. J.) et Knuth (D. E.). – On the Lambert W Function. *Advances in Computational Mathematics*, vol. 5, 1996, pp. 329–359. – `ftp://ftp.inria.fr/lang/maple/5.3/share/LambertW.ps`.

[18] Costanza (R.) et Maxwell (T.). – Distributed modular spatial ecosystem modelling. *International Journal of Computer Simulation. Special Issue on Advanced Simulation Methodologies*, vol. 5, n° 3, 1995, pp. 247–262. – `http://iee.umces.edu/SME3/`.

[19] Crofton (H.D.). – A model of host-parasite relationships. *Parasitology*, vol. 63, 1971, pp. 343–364.

[20] Crofton (H.D.). – A quantitative approach to parasitism. *Parasitology*, vol. 62, 1971, pp. 179–193.

[21] Deelman (E.), Caraco (T.) et Szymanski (B. K.). – Parallel discrete event simulation of lyme disease. *In : 1996 Pacific Symposium, Hawaii*, éd. par Hunter (L.) et Klein (T.). pp. 191–202. – World Scientific Publishing Corp.

[22] Flechsig (M.). – *Strictly Parallelized Regional Integarted Numeric Tool For Simulation.* – Rapport technique, Telegrafenberg, D-14473 Postdam, Postdam Institue for Climate Impact Reasearch, 1999.

[23] Fujimoto (R.M.). – Parallel discrete event simulation. *Communications of the ACM*, vol. 33, n° 10, October 1990, pp. 31–53.

[24] Grama (A.), Gupta (A.), Kumar (V.) et Karypis (G.). – *Introduction to Parallel Computing: Design and Analysis of Algorithms.* – Benjamin/Cummings Publishing Company, Redwood City, 1994.

[25] Herbert (J.) et Isham (V.). – Stochastic host-parasite interaction models. *Journal of Mathematical Biology*, vol. 40, 2000, pp. 343–371.

[26] Herbert (J.) et Isham (V.). – Stochastic host-parsite interaction models. *Journal of Mathematical Biology*, vol. 40, 2000, pp. 343–371.

[27] Hochbaum (D.S.). – *Approximation algorithms for NP-hard problems.* – Edited by Dorit S. Hochbaum, PWS publishing company, 1997. ISBN 053494968-1.

[28] Hraber (P.T.), Jones (T.) et Forrest (S.). – The ecology of echo. *Artificial Life*, vol. 3, n° 3, 1997, pp. 165–190.

[29] Hudson (P. J.), Newborn (D.) et Dobson (A. P.). – Regulation and stability of a free-living host-parasites system : Trichostrongylus tenuis in red grouse ii population model. *Journal of Animal Ecology*, vol. 61, 1992, pp. 477–486.

[30] Kot (M.). – *Elements of Mathematical Ecology.* – Cambridge University Press, 2001.

[31] Kotstitzin (V.A. 1934). – Symbiosis, parasitism and evolution. *In : The Golden Age of Theoretical Ecology*, éd. par Scudo (F.) et Ziegler (J.), pp. 369–408.

[32] Langlais (M.), Latu (G.), Roman (J.) et Silan (P.). – Parallel numerical simulation of a marine host-parasite system. *In : Europar'99 Parallel Processing*, éd. par Amestoy (P.), Berger (P.), Daydé (M.), Duff (I.), Frayssé (V.), Giraud (L.) et Ruiz (D.). pp. 677–685. – LNCS 1685 - Springer Verlag. 1999.

[33] Langlais (M.), Latu (G.), Roman (J.) et Silan (P.). – Stochastic simulation of a marine host-parasite system using a hybrid OpenMP/MPI programming. *In : Europar'02 Parallel Processing.* pp. 436–446. – LNCS 2400 - Springer Verlag. 2002.

[34] Langlais (M.) et Silan (P.). – Theoretical and mathematical approach of some regulation mechanisms in a marine host-parasite system. *Journal of Biological Systems*, vol. 3, n° 2, 1995, pp. 559–568.

[35] Latu (G.). – *Solution parallèle pour un simulateur d'un système hôte-parasite en environnement marin.* – 1998. DEA d'informatique, Université Bordeaux 1.

[36] Latu (G.). – Solution parallèle pour un problème de dynamique de population. *Technique et Science Informatiques*, vol. 19, juin 2000, pp. 767–790.

[37] Latu (G.). – Simulation stochastique parallèle d'un système hôte-parasite par programmation hybride MPI+OpenMP. *In : RenPar'14, 14ème Rencontres francophones du parallélisme des architectures et des systèmes (Hammamet)*, pp. 83–90. – 2002.

[38] Lau (Lawrence). – *Implementation of Scientific Applications on Heterogeneous Parallel Architectures.* – Thèse de PhD, Department of Mathematics, University of Queensland, Australia, November 1996. `http://www.acmc.uq.edu.au/~ll/thesis/`.

[39] Levin (S. A.) (édité par). – *Mathematics and Biology: The Interface - Challenges and Opportunities.* Lawrence Berkeley Lab PUB-701. – 1992. `http://www.bis.med.jhmi.edu/Dan/mathbio/intro.html`.

[40] Levin (Simon) (édité par). – *On growth and form, Spatio-temporal pattern formation in biology*, pp. 225–246. – Wiley series in Mathematical and computational biology, 1999, chaplain, singh, maclachlan édition.

[41] Lorek (H.) et Sonnenschein (M.). – Using parallel computers to simulate individual-oriented models in ecology: A case study. *In : ESM'95 European Simulation Multiconference: Modelling and Simulation.* – SCS International. pp. 526–531.

[42] Maniatty (B.), Szymans ki (B.) et Caraco (T.). – Tempest: A fast spatially explicit model of epidemics on parallel machines. *In : Proc. High Performance Computing Symposium, San Diego*, éd. par Tentner (A.). pp. 114–119. – SCS.

[43] Maniatty (B.), Szymanski (B.) et Caraco (T.). – High-performance computing tools for modeling evolution in epidemics. *In : Proc. of the 32nd Hawaii International Conference on System Sciences.*

[44] Maniatty (W. A.), Szymanski (B.K.) et Caraco (T.). – Parallel computing with generalized cellular automata. *Parallel and Distributed Computing Practices*, vol. 1, 1998, pp. 85–104.

[45] Mascagni (M.), Ceperley (D.) et Srinivasan (A.). – Sprng: A scalable library for pseudorandom number generation. – In Proceedings of the Third International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing, June 1998.

[46] Masuda (Norio) et Zimmermann (Frank). – *PRNGlib : A Parallel Random Number Generator Library*, 1996. TR-96-08, *ftp://ftp.cscs.ch/pub/CSCS/libraries/PRNGlib/.*

[47] Matis (J.H) et Kiffe (T.R.). – *Stochastic Population Models : A Compartmental Perspective.* – Springer, 2000, *Lecture Notes in Statistics*, volume 145.

[48] May (R.M.). – Dynamical aspects of host-parasite associations : Crofton's model revisited. *Parasitology*, vol. 75, 1977, pp. 259–276.

[49] May (R.M.) et Anderson (R.M.). – Regulation and stability of host-parasite population interactions. I. regulatory processes. *Journal of Animal Ecology*, vol. 47, 1978, pp. 219–247.

[50] May (R.M.) et Anderson (R.M.). – Regulation and stability of host-parasite population interactions. II. destabilizing processes. *Journal of Animal Ecology*, vol. 47, 1978, pp. 249–267.

[51] Message Passing Interface Forum. – *MPI: A Message-Passing Interface Standard.* – version 1.1 June 12, 1995. Available by anonymous ftp from *ftp.mcs.anl.gov.*

[52] Message Passing Interface Forum. – MPI: A Message-Passing Interface Standard. *International Journal of Supercomputer Applications*, vol. 8, n° 3-4, 1994. – Also available by anonymous ftp from *ftp.netlib.org.*

[53] Minar (N.), Burkhart (R.), Langton (C.) et Askenazi (M.). – *The Swarm simulation system: A toolkit for building multi-agent simulations.* – Rapport technique, Santa Fe, Santa Fe Institute, June 1996. `http://www.santafe.edu/projects/swarm/`.

[54] Murray (J.D.). – *Mathematical Biology.* – Springer, 1989.

[55] Nicol (D.M.) et Fujimoto (R.M.). – Parallel simulation today. *Annals of Operations Research*, vol. 53, December 1994, pp. 249–286. – `http://www.cc.gatech.edu/computing/pads/papers.html`.

[56] Nåsell (I.). – Hybrid models of tropical infections. *Lecture Notes in Biomathematics*, no59, 1985.

[57] OpenMP. – A Proposed Industry Standard API for Shared Memory Programming. October 1997, OpenMP Forum, `http://www.openmp.org/`.

[58] Pavé (A.). – *Modélisation en biologie et en écologie.* – Aléas Editeur, Lyon, 1994.

[59] Perko (L.). – *Differential Equations and Dynamical systems.* – Springer, New-York, 1991.

[60] Shaw (D.J.) et Dobson (A.P.). – Patterns of macroparasites abundance and aggregation in wildlife populations: a quantitative review. *Parsitology*, vol. 111, 1995.

[61] Silan (P.). – *Biologie comparée des populations de Diplectanum aequans et Diplectanum laubieri , Monogènes branchiaux de Dicentrarchus labrax.* – Thèse de PhD, Université Montpellier II, 1984. 275 pp.

[62] Silan (P.), Birgi (E.), Louis (C.), Clota (F.), Mathieu (A.) et Giral (L.). – Aquaculture et ichtyoparasitologie : action *in vitro* du nitroxinil© (Antihelminthique) sur *Diplectanum Aequans*, ectoparasite branchial du bar *Dichentrachus labrax. Recl. Méd. Vét.*, vol. 172, n° 7-8, 1996, pp. 401–407.

[63] Silan (P.), Langlais (M.) et Bouloux (C.). – Dynamique des populations et modélisation : Application aux systèmes hôte-macroparasite et à l'épidémiologie en environnement marin. *In : Tendances nouvelles en modélisation pour l'environnement*, éd. par eds (C.N.R.S). – Elsevier, 1997.

[64] Silan (P.), Langlais (M.) et Latu (G.). – Dynamique des populations de monogènes, ectoparasites de téléostéens : stratégies démographiques et implications mathématiques. *Ecologie*, vol. 30, 1999, pp. 247–260.

[65] Silan (P.) et Maillard (C.). – Modalités de l'infestation par *Diplectanum Aequans*, Monogène ectoparasite de *Dichentrarchus labrax*, en aquiculture. Eléments d'épidémiologie et de prophylaxie. *Pathology in Marine Aquaculture.* Vivarès C.P., Bonami J.R., Jasper E. (Eds.). European Aquaculture Society, Special Publication No. 9, Bredene, Belgium, 1986, pp. 139–152.

[66] Silan (P.) et Maillard (C.). – Biologie comparée du développement et discrimination des Diplectanidae ectoparasites du Bar (Teleostei). *Ann. Sci. Nat. Zool. Paris 13*, vol. 10, 1989, pp. 31–45.

[67] Silan (P.) et Maillard (C.). – Comparative structures and dynamics of some populations of helminths, parasites of fishes: the sea bass-*Diplectanum* model. *Acta oecologica*, vol. 11, n° 6, 1990, pp. 857–874.

[68] Taylor (L. R.). – Aggregation, variance and the mean. *Nature*, vol. 189, 1961, pp. 732–735.

[69] Wilson (K.), Bjørnstad (O.N.), Dobson (A.P.), Merler (S.), Poglayen (G.), Randolph (S.E.), Read (A.F.) et Skorping (A.). – *The Ecology of Wildlife Diseases*, chap. 2: Heterogeneities in macroparasite infections – patterns and processes, pp. 6–44. – éd. par Hudson (P.J.), Rizzoli (A.), Grenfell (B.T.), Heesterbeek (H.), Dobson (A.P.), 2001. `http://asi23.ent.psu.edu/publ/misc/abstract2.html`.

Résumé

Ce travail contribue à un modèle déterministe discret d'un système hôte-macroparasite et propose un modèle individu-centré équivalent. Une application du modèle consiste en l'étude quantitative du système Bar-*Diplectanum aequans* à l'aide de deux simulateurs parallèles. Une étude algorithmique détaillée est donnée pour le simulateur déterministe. L'extensibilité de très bonne qualité est évaluée théoriquement et testée. Une utilisation optimisée des mémoires caches permet d'atteindre 60 % de la puissance crête au coeur des calculs. Les temps d'exécution sont réduits et la précision des calculs améliorée, ce qui permet de reproduire des dynamiques observées sur le terrain.

Le second simulateur utilise une méthode de type Monte Carlo. On donne les performances associées à une programmation hybride sur une grappe de noeuds SMP. L'étude quantitative effectuée sur les résultats des simulateurs et leur comparaison donne un éclairage nouveau sur l'interaction des mécanismes des systèmes hôte-macroparasite.

Mots clés : Algorithmique parallèle, simulation numérique, modèle individu-centré, modèle individu-centré, système hôte-parasite, dynamique de populations, *Monogenea.*

Discipline: Informatique

Abstract

We are interested in a host-macroparasite system. We have contributed to a deterministic model of this system and developed a new individual-based model. Our work focuses on two simulators using the sea bass-*Diplectanum Aequans* system. The algorithmic study of the deterministic simulator and the performance analysis establish the efficiency and scalability of our parallel algorithm. A study of memory accesses and cache utilization lead to an implementation reaching 60 % of peak performance in the computation kernel. Execution times are reduced and the parallel solution provides more accurate computations. This simulator gives realistic qualitative behaviors of the system.

We describes the Monte Carlo algorithm of the stochastic simulator. Analysis and performance of a hybrid MPI/OpenMP code are then presented for a cluster of SMP nodes. The detailed quantitative study of both simulators and their comparison gives some insights to the interaction of mechanisms taking place in a host-macroparasite system.

Keywords : parallel algorithmics, numerical simulation, discrete model, individual-based model, host-parasite system, population dynamics, *Monogenea.*

Discipline: Computer Science