Minimisation des communications dans une résolution distribuée des équations de Navier-Stokes

Frédérique Voisin-Demery

ICPS, Université Louis Pasteur
Pôle API, Bd S. Brant, F-67400 Illkirch
E-mail: voisin@icps.u-strasbg.fr

Résumé
On se propose de résoudre les équations de Navier-Stokes par une méthode se prêtant bien à la parallélisation par distribution des données. Cette distribution des données implique des communications, dont on va tenter de minimiser le volume.

Mots-clés : Navier-Stokes - Communications - Minimisation

1. Présentation du problème

On étudie un écoulement de fluide laminaire (ordonné), avec un sillage dans lequel se créent des instabilités de type de Hopf (voir [2]) présentant des oscillations: ceci permet par exemple de modéliser les tubes dans les échangeurs de chaleur d'une centrale électrique. Ce phénomène est représenté par les équations de Navier-Stokes dont on donne la formulation en vitesse-pression :

$$\frac{\partial v}{\partial t} + v\nabla v - \nu\nabla^2 v + \nabla p = 0 \text{ et } \nabla . v = 0$$

Pour tenir compte des oscillations qui apparaissent, on choisit une formulation en série de Fourier des variables v et p [1]:

$$v(X,t) = \sum_n c_n(X,t)e^{in\omega t} \text{ et } p(X,t) = \sum_n d_n(X,t)e^{in\omega t}$$

où X est un vecteur spatial à 2 ou 3 dimensions (dans la suite, on l'omettra).
Les oscillations ont une période qui varie un peu au cours du temps; le terme $c_n(t)$ permet de prendre en compte cette évolution. Pour résoudre les équations, on est tout d'abord amené à découper le domaine physique pour approximer les dérivées partielles par rapport aux variables spatiales. Cette discrétisation se fait par des éléments spectraux, c'est-à-dire qu'on découpe l'espace en domaines relativement grands à l'intérieur desquels on choisit un nombre important de points. Ensuite, afin d'approximer la dérivée partielle par rapport au temps, on est amené à faire une discrétisation temporelle. Elle se fait en trois temps, permettant d'isoler les 3 termes de la somme intervenant dans les équations de Navier-Stokes. La résolution du problème initial se traduit donc par la résolution des 3 problèmes suivants:

$$\frac{c_n^* - c_n^i}{\delta t} = \sum_{k=-\infty}^{k=\infty} (c_k^i . \nabla) c_{n-k}^i \tag{1}$$

$$\frac{\hat{c}_n - c_n^*}{\delta t} = -\nabla d_n \text{ et } \mathrm{div}\hat{c}_n = 0 \tag{2}$$

$$\frac{c_n^{i+1} - \hat{c}_n}{\delta t} = -(i\ n\omega - \nu\nabla^2)c_n^{i+1} \tag{3}$$

où on calcule tout d'abord c_n^* grâce à c_n^i , puis $\hat{c}_n$ grâce à c_n^*, et enfin c_n^{i+1} grâce à $\hat{c}_n$.
Ici apparaît l'intérêt de la décomposition en série de Fourier: on va être amené à faire moins d'itérations sur le temps que lors d'une modélisation plus classique qui oblige à prendre un pas de temps très petit afin de suivre les oscillations. D'autre part, pour des raisons de stabilité numérique, on est amené à avoir deux boucles sur le temps imbriquées [2]. Bien évidemment, pour effectuer les calculs, on tronque la série de Fourier, pour n'avoir plus que N harmoniques (ordre de grandeur $N \leq 7$).

2. Position du problème lié à la parallélisation

Il faut savoir que la résolution de (1) représente 1% du temps de calcul, la résolution de (2) et (3) utilisant les 99% restant. La parallélisation doit donc porter principalement sur ces 2 problèmes. La formulation en série de Fourier fait apparaître une parallélisation simple pour ces 2 équations, par distribution des données et des calculs : on place c_n et d_n sur le processeur n, tous les calculs sont alors locaux car ils ne font intervenir que des données d'indice n. On note qu'on a besoin de N processeurs, ce qui, vu la taille de N est raisonnable[1]. Par contre, on doit se pencher sur la résolution de (1), qui, dans ces conditions, nécessite des communications: notre objectif est d'en diminuer le nombre en restructurant le problème.

Le problème qui se pose alors à nous n'est pas la parallélisation proprement dite - puisqu'elle est imposée - mais il s'agit plutôt de minimiser le temps d'exécution de la résolution de (1), de telle sorte que cette étape négligeable lors d'une exécution séquentielle ne devienne pas d'un coût trop important en parallèle. L'idée est donc de reformuler le problème pour optimiser les mouvements de données mis en jeu. En tenant compte de la propriété $c_n = -\bar{c}_n$ et en découpant la somme en 3 parties, la résolution de (1) se traduit par le nid de boucles suivant :

```
Do I = 0 , N
    Do K = 1 , N-I
        A[I] = A[I] + F1(C[K],C[I+K])
    Do K=0 , I
        A[I] = A[I] + F2(C[K],C[I-K])
    Do K = I+1 , N
        A[I] = A[I] + F3(C[K],C[K-I])
```

Les fonctions `F1`, `F2`, `F3` déterminent l'usage ou non de la forme conjuguée des variables `C`.
On remarque qu'on a à faire à un nid de boucles faisant appel à des dépendances affines.

3. Etude du nid de boucles et transformation

L'objectif est de minimiser les communications induites dans ce nid de boucles par la distribution des données en respectant certaines contraintes :

- La distribution des données et des calculs est donnée: en effet, ce problème s'insérant dans un bloc plus grand pour lequel l'allocation est fixée comme on l'a vu dans 2, on la respecte aussi dans cette partie. On a $N+1$ processeurs, $A[I]$ étant sur le processeur I, de même que $C[I]$.
- On suppose que la mémoire de chaque processeur ne peut pas contenir l'ensemble des $C[K]$. On élimine donc la solution qui serait de faire une communication globale All-To-All au départ : chaque processeur communiquant à tous les autres le $C[I]$ qu'il a en mémoire et effectuant ses calculs, ayant toutes les données dont il a besoin sur place.

3.1. Parallélisation du nid de boucles original

La première idée est de prendre le nid de boucles tel qu'il est et à chaque fois qu'un processeur a besoin d'une donnée, il y a une communication. Soit au total, pour l'ensemble des processeurs $3N^2$ communications. On constate que cette solution requiert la place mémoire de 3 variables `C` (la variable locale et les 2 nécessaires au calcul)

3.2. Première amélioration

En faisant un changement de variables $K = K - I$ dans la troisième boucle on se rend compte qu'elle a les mêmes bornes et utilise les mêmes données que la première : on peut donc les fondre en une seule boucle. De même pour la deuxième boucle : on la scinde, puis on applique le changement de variables à une des parties, et on les refond en une seule. On obtient :

```
Do I = 0 , N
    Do K = 1 , N-I
```

1. Si on dipose de plus de N processeurs, on peut combiner cette technique de parallélisation avec une méthode de décomposition de domaines: le code originel, qui ne prenait pas en compte cette formulation en série de Fourier le permet. Si on a moins de N processeurs, on peut regrouper les calculs correspondants à plusieurs modes sur un seul processeur

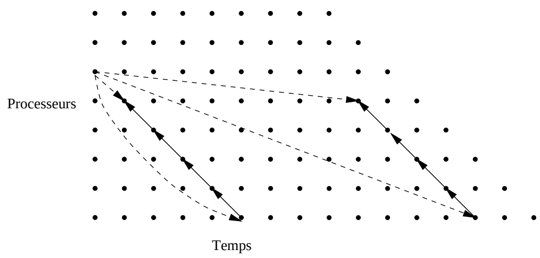

FIG. 1 - *Ensemble d'utilisation de 2 occurrences de* $C[5]$ *(*$N = 7$*)*

```
        A[I] = A[I] + F1(C[K],C[I+K])
        A[I] = A[I] + F3(C[I+K],C[I])
    Do K=0 , [I/2]-1
        A[I] = A[I] + F2(C[K],C[I-K])
        A[I] = A[I] + F2(C[I-K],C[K])
    If  (I Mod 2) = 0
        A[I] = A[I] + F2(C[I/2],C[I/2])
```

où `Mod` est la fonction modulo.
Par rapport à la formulation précédente, on divise le nombre de communications par 2, donc $\frac{3}{2}N^2$ communications au total. Et comme précedemment, chaque processeur a besoin de la place mémoire pour 3 valeurs de `C`.
On note que l'utilisation de techniques de transformation de nid de boucles ne permettent pas l'obtention directe de cette solution. En effet, l'analyse de dépendances telle qu'on peut la voir dans [5] par exemple ne permet pas d'appliquer des techniques de transformation de nids de boucles. Cette analyse ne tient pas compte du caractère commutatif de l'addition et ne permet donc pas de fusionner les boucles. Par contre, une analyse de dépendances réalisée grâce à un outil comme OPERA [4] permet de visualiser les régularités qui peuvent apparaître (voir la figure 1).

3.3. Parallélisation optimale

La boucle précédente fait encore trop de communications car le minimum de communications de chaque processeur est N. Pour obtenir ce nombre minimum de communications, il suffit une fois qu'on a une donnée distante dans la mémoire locale de l'utiliser dans tous les calculs où elle apparaît. La figure 2 montre comment on peut réutiliser $C[4]$ sur le processeur 3. On voit qu'on est amené à utiliser $C[1]$ puis $C[1+3]$, puis $C[1+3]$ et $C[1+2*3]$. Plus généralement, on obtient :

```
I=0
A[I] = A[I] + F2(C[0],C[0])
Do K=1 , N
  A[I] = A[I] + F2(C[K],C[K])
Do I = 1
  K =0
  A[I] = A[I] + F2(C[0],C[I])
  A[I] = A[I] + F2[C[I],C[0])
  Do J=1, [[(N-I)/I]]
    A[I] = A[I] + F1(C[JI],C[JI+I])
    A[I] = A[I] + F3(C[JI+I],C[JI])
  Do K = 1 , [I/2]-1
    A[I] = A[I] + F2(C[K],C[I-K])
    A[I] = A[I] + F2(C[I-K],C[K])
    Do J= 0, [[(N-I-K)/I]]
      A[I] = A[I] + F1(C[K+IJ],C[K+(J+1}I])
      A[I] = A[I] + F3(C[K+(J+1)I],C[K+IJ])
    Do J = 0, [[(N-2I+K)/I]]
      A[I] = A[I] + F1(C[I(J+1)-K],C[I(J+2)-K])
      A[I] = A[I] + F3(C[I(J+2)-K],C[I(J+1)-K])
  If (I Mod 2) = 0
    A[I] = A[I] + F2(C[I/2],C[I/2])
    Do J = 0, [[(2N-I)/I]]
      A[I] = A[I] + F1(C[I/2+JI],C[I/2+(J+1)I])
      A[I] = A[I] + F3(C[I/2+(J+1)I],C[I/2+JI])
```

où `[.]` représente la partie entière supérieure et `[[.]]` la partie entière inférieure. Le nombre maximum de `C` sur un processeur est 4. On augmente la taille mémoire nécessaire, mais on atteint le nombre minimal de communications nécessaires, et ceci sans passer par une communication All-To-All qui demanderait plus de mémoire.
On remarque qu'on a ajouté une dimension de plus par rapport au nid de boucles original, et que ceci fait intervenir des dépendances qui ne sont plus affines ($K + IJ$), d'où l'impossiblité d'une visualisation lisible ou d'un traitement classique des dépendances, tel qu'on le fait dans le modèle polyèdrique. Ceci suggère une extension intéressante de ce modèle.

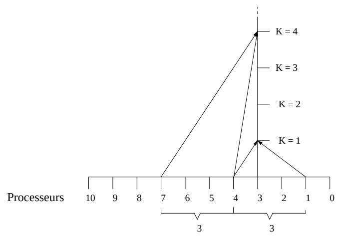

FIG. 2 - *Communications -représentées par les flèches - nécessaires à l'exécution de la première boucle sur K du nid de boucles de 3.2 sur le processeur 3, avec* $N = 10$

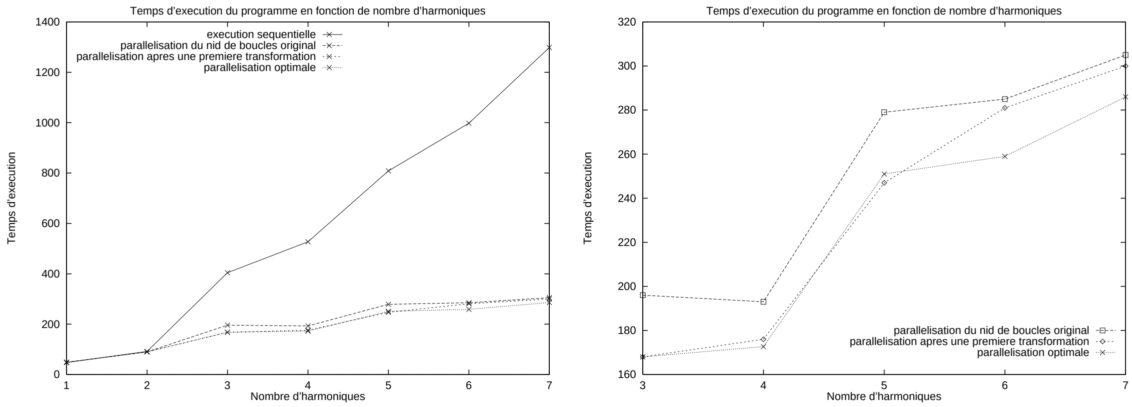

FIG. 3 - *Temps d'exécution de la résolution du problème de départ (100 itérations), en fonction de N*

4. Expérimentations

On a obtenu trois façons possibles de paralléliser l'équation (1), générant $3N^2$, $\frac{3}{2}N^2$ et N^2 communications. On veut alors voir si ces améliorations théoriques se retrouvent au niveau pratique. On a effectué la parallélisation en utilisant MPI [3], sur une Origin2000, en faisant 2 itérations externes et 50 internes. Le figure 3 montre clairement l'apport de la parallélisation pour le problème étudié même si on n'obtient pas une efficacité égale à 1 (on a au mieux environ 0.65). Ceci s'explique par le fait qu'il y a des synchronisations, pour mettre à jour la pulsation entre autres; or les temps de résolution des problèmes 2 et 3 diffèrent selon les harmoniques: convergence plus ou moins rapide de la résolution de systèmes linéaires par méthodes itératives. C'est donc le temps de résolution le plus long qui prévaut pour chaque pas de temps.

D'autre part, la comparaison des courbes concernant les exécutions parallèles confirment les gains apportés par chaque solution, même si ces gains sont faibles comparés à ceux apportés par la parallélisation du programme séquentiel. Les différences ne sont visibles qu'à partir de $N = 3$ car pour un N inférieur, les boucles sont très courtes, et génèrent donc peu de communication. Pour un nombre élevé de processseurs, on trouve des temps d'exécutions conformes à nos attentes, même si les différences sont assez faibles. Néanmoins, on a un gain de l'ordre de $1,5\%$ à $3,5\%$ lorsqu'on passe d'une courbe à l'autre, ce qui n'est pas complètement négligeable pour beaucoup plus d'itérations.

Bibliographie

1. Carte (Gilles), Dusek (Jan) et Fraunié (Phillipe). – A spectral time discretization for flows with dominant periodicity. *Journal of Computational Physics.*, vol. 120, 1995, pp. 171–183.
2. Dusek (Jan), Le Gal (Patrice) et Fraunié (Philippe). – A numerical and theoretical study of the first hopf bifurcation in a cylinder wake. *J. Fluid Mech.*, vol. 264, 1994, pp. 59–80.
3. MPI Forum. – *MPI: A Message-Passing Interface MPI Forum.* – Rapport technique n° CS/E 94-013, Department of Computer Science, Oregon Graduate Institute, mars 94.
4. Loechner (Vincent) et Mongenet (Catherine). – A toolbox for affine recurrence equations parallelization. *Lecture Notes in Computer Science*, vol. 919, 1995, pp. 263–268.
5. Wolfe (Michael). – *High Performance Compilers for Parallel Computing.* – Addison Wesley, 1986.