Parallel Processing Letters,

MEMORY COST DUE TO ANTICIPATED BROADCAST

VINCENT LOECHNER and CATHERINE MONGENET
{loechner,mongenet}@icps.u-strasbg.fr
Laboratoire ICPS-LSIIT, Université Louis Pasteur de Strasbourg
Illkirch, F-67400, France

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

ABSTRACT

To get efficient solutions, parallelization techniques mainly focus on data alignment or on communication minimization. The efficiency of a parallel solution not only depends on the communication cost, but also on the memory cost. This paper mainly focus on a symbolic evaluation of the memory cost due to anticipated broadcast. This evaluation is conducted in the polytope model using Ehrhart polynomials, which express the number of integer points in a parameterized polytope.

1. Introduction

Over the last years many research works have focused on the parallelization of nested loops. To get an efficient parallelization, these techniques mainly focus on data alignment or on communication minimization [1,2,3,4,5,6,7,8,9,10]. The efficiency of a parallel solution not only depends on the communication cost, but also on the memory cost. This paper mainly focus on a symbolic evaluation of the memory cost.

The parallelization of a loop nest consists in determining a schedule function which states for any elementary computation the time when it is executed, and an allocation function which associates a virtual processor to each elementary computation. In order to be valid the parallelization of a loop nest relies on its dependences. Besides well studied uniform dependences, many scientific problems are expressed using loops with affine dependences. For instance, matrix computations such as triangulation use a diagonal element to compute all the elements in the same row or column. In a parallel implementation, this diagonal element is computed by a given processor (called *emission processor*), which has to send it to all the processors that use it (called *utilization processors*). This can be done using two communication schemes:

- pipeline: the data is sent from the emission processor to one of the utilization processors, and then propagated from neighbor to neighbor among the utilization processors ;

- broadcast: the data is simultaneously sent from the emission processor to all the utilization processors.

On a parallel architecture where broadcast is efficiently implemented, the second communication scheme should be chosen. Hence, it corresponds to a single communication primitive, whereas the first scheme requires as many communication primitive calls as utilization processors. This is in particular relevant when the network is a bus, as it occurs in a cluster of workstations connected by ethernet for example. On this type of architecture, since the cost of a broadcast communication is equal to the cost of one point-to-point communication, the broadcast scheme, as opposed to the pipeline scheme, drastically reduces the network load. This approach is of course irrelevant on any parallel architecture where the broadcast is simulated by a pipeline, such as on systolic-like architectures. In the following, we assume that the broadcast is efficiently implemented and we focus on techniques to derive parallel loops using as much broadcast as possible.

The ideal broadcast communication scheme is the following: a given data is computed, it is then sent to all its utilization processors which use it right away. We call it *true broadcast*. For a given data, a true broadcast can only be implemented if all the computations that use it are scheduled at the same time step. Therefore true broadcast imposes strong conditions to be satisfied by the schedule function. For a problem characterized by several dependences, these conditions can not always be satisfied simultaneously. When they are not satisfied for a given dependence, it means that the utilization processors use the data at successive time steps. Even though pipeline could be used in this case, we choose to still use broadcast to reduce the communication cost. We call it *anticipated broadcast*. Since each utilization processor does not use the data immediately after its reception, it must store it until it finally uses it. This memory cost should be controlled in order to fit the hardware.

This work presents a method, based on a symbolic evaluation, to compute for given schedule and allocation functions, the memory cost related to each dependence in case of an anticipated broadcast. This symbolic evaluation is conducted in the framework of systems of parameterized affine recurrence equations which abstract single assignment loop nests. Using the dependence modeling in terms of utilization and emission sets [11], classical affine per variable space-time transformations are computed [12]. In this approach, called the polytope model [13], the information can be represented by parameterized* convex polyhedra: utilization and emission sets, isotemporal spaces (i.e. polyhedra containing all the computation points scheduled at the same time step), etc. One of the techniques used to extract information from such polyhedra relies on counting the number of integer points it contains [14]. For instance, the potential parallelism at a given time step, i.e. the number of active processors at that time step, is the number of integer points in the corresponding isotemporal space. This symbolic evaluation is realized using Ehrhart

*The parameters can be: the problem size parameters, a time step, a processor coordinates, an emission point, ...

polynomials [15,14] which express this number as a function of its parameters.

The paper is organized as follows. It first summarizes the classical framework of systems of affine recurrence equations parallelization. It particularly recalls how to determine different space-time transformations. It then shows how to compute, for a given transformation, the memory cost using appropriate Ehrhart polynomials. It illustrates the approach on the Gaussian elimination problem. It finally gives some concluding remarks.

2. Parallelization framework

A single assignment loop nest can be formalized by a system of parameterized affine recurrence equations (PARE). Such an equation is defined by:

$$X[z] = f(..., Y[g(z,p)], ...) \qquad z \in \mathcal{D}_{Eq}(p) \qquad (Eq)$$

where X and Y are variable names. p is a vector of $\mathbb{Z}^q$ defining the problem parameters. $\mathcal{D}_{Eq}(p) \subset \mathbb{Z}^n$ is a parameterized convex bounded polyhedron called the *iteration domain*, defined by a set of linear constraints. $g(z,p)$ is an affine function of z and p.

2.1. Dependence modeling

The dependence information is extracted from such a PARE in terms of utilization and emission sets, as presented in [11,10].

The *utilization set* corresponding to occurrence $Y[z_0]$ in equation (Eq) is the set of points z whose computation $X[z]$ needs $Y[z_0]$. It is a convex polyhedron parameterized by z_0 and p: $\mathrm{Util}_{Eq,Y}(z_0,p) = \{z \in \mathcal{D}_{Eq}(p) \mid g(z,p) = z_0\}$. The geometric dimension of $\mathrm{Util}_{Eq,Y}(z_0,p)$ is denoted by $dim_{\mathrm{Util}_{Eq,Y}}$. This set expresses that there is a dependence between point z_0 and each of the points of $\mathrm{Util}_{Eq,Y}(z_0,p)$.

The set of all points z_0 which originate such a dependence is called the *emission set*: $\mathrm{Emit}_{Eq,Y}(p) = \{z_0 \in \mathbb{Z}^n \mid \exists z \in \mathcal{D}_{Eq}(p) \text{ such that } z_0 = g(z,p)\} = g(\mathcal{D}_{Eq}(p),p)$.

In the following, we simply denote $g(\mathcal{D}_{Eq}(p),p)$ by $g(\mathcal{D}_{Eq}(p))$, $\mathrm{Util}_{Eq,Y}$ by Util_Y and $\mathrm{Emit}_{Eq,Y}$ by Emit_Y.

2.2. Space-time transformation

In order to find a parallel solution to a set of PARE an affine space-time transformation has to be determined. Such a transformation is classically defined by two functions: an affine schedule function and an affine allocation function. In order to efficiently optimize the communications, we deal with *per variable* transformation. Such a transformation is expressed, for each variable X, by a full-column rank transformation matrix $T_X = \begin{pmatrix} \lambda_X \\ \sigma_X \end{pmatrix}$ and two constant vectors α_X and γ_X where:

- λ_X is a $d_\lambda \times n$ integer matrix defining the *multi-dimensional schedule*: $t_X(z) = \lambda_X z + \alpha_X$. The row vectors of λ_X are called the *schedule vectors*. They are orthogonal to a set of affine subspaces called *isotemporal spaces*. Each isotemporal space contains all the computation points of X executed at a given time step.

- σ_X is a $d_\sigma \times n$ integer matrix representing the *allocation* on a virtual processor space of dimension d_σ. The row vectors of σ_X generate the virtual processor space associated with X. The allocation function is defined by $alloc_X(z) = \sigma_X z + \gamma_X$.

Once a set of per variable transformations has been determined, the parallel code can be synthesized by applying these transformations to the PARE, as presented in [16,17] for a single transformation, and in [18,19] for per variable transformations.

For the transformations to be valid, matrices T_X must be full rank. Hence, it guarantees that no two computations are executed at the same time step by the same processor. Of course, not every full rank matrix is valid. Other conditions must or may be enforced. The first conditions that must always be satisfied are the *causal constraints*. These constraints state that a data has to be computed before its use. They are classically expressed for any dependence from Y to X of the form $X[z] \leftarrow Y[g(z,p)]$ by: $\quad t_X(z) \succ t_Y(z_0), \;\; \forall z \in \mathrm{Util}_Y(z_0,p)$.

2.3. Communication optimization

Other conditions may be added to optimize communications. As presented in earlier works [3,5,7,9,10], a *strong condition* can be computed so to map, for a given dependence, any utilization set on a single processor. It results into single communications from each emission processor to a unique utilization processor. This condition imposes constraints on the allocation matrices and on the schedule matrices. For many problems characterized by several dependences, this condition can not be satisfied for all of them. In such a case one approach is to apply the strong condition for as many dependences as possible, and then to optimize the residual communications [7]. Another approach is to combine the strong condition with other communication minimization conditions that do not impose such strong constraints on the transformation matrices. We propose two such conditions. The first one is a *weak condition* that enforces pipeline (also called *propagation*) of the data among the processors [12]. The second one takes advantage of broadcast communications, and in [10] various conditions on the schedule and the allocation functions are given to ensure local and/or normal broadcast. In the following, we focus on broadcast and we show how to communicate using either true or anticipated broadcast.

2.4. Symbolic evaluation

In order to evaluate the quality of a synthesized parallel solution, in particular its efficiency in term of latency, number of processors, communication cost and memory cost, symbolic evaluation must be conducted. The objects manipulated in the framework of PAREs are convex parameterized polyhedra. The parameters can be the problem parameters p, an emission point z_0, a time step τ, a processor reference a, or any combination of them. The evaluation consists in counting the number of points in such a polytope.

For example, the elementary computations of an iteration domain $\mathcal{D}(p)$, executed on a given processor a, are expressed by the following polytope: $\mathcal{E}(a,p) = alloc^{-1}(a) \cap \mathcal{D}(p)$, where $alloc^{-1}$ is the preimage of the allocation function. It defines the set of all the points whose image by function $alloc$ is a. Therefore the intersection of this set with $\mathcal{D}(p)$ characterizes all the elementary computations executed on processor a. Counting the number of points of this polytope parameterized by a and p will give the number of computations executed on processor a, as a function of a and p.

Such a counting is realized using Ehrhart polynomials [15]. Depending on the actual values of the parameters, the shape of the polytope may be different. Therefore an Ehrhart polynomial is defined on each *validity domain* of the parameters. The coefficients of an Ehrhart polynomial are periodic numbers: their value are defined by a `case` function of the parameters *modulo* a constant.

The computation of Ehrhart polynomials has been implement in the *PolyLib* library (`http://icps.u-strasbg.fr/PolyLib`) and the results presented in section are computed using this library.

3. Broadcast communication schemes

The ideal broadcast communication scheme occurs when all the utilization processors use a given data at the same time step. We call it a *true broadcast*. This imposes strong constraints on the schedule. In the simple case of a broadcast along a one-dimensional utilization set, all its points must be scheduled at the same time step [11]. In the general case of multi-dimensional utilization sets, the condition on the schedule function is given in [10].

The memory cost associated with true broadcast communication is minimal. For instance, in the case of a broadcast along a one-dimensional utilization set, each utilization processor needs only one memory location to hold the data.

However, due to the strong conditions it imposes on the schedule functions, in many cases true broadcast can not be enforced for all the dependences. In this case, for those dependences where true broadcast can not be applied, we propose to use *anticipated broadcast*. This means that even though broadcast is not necessary (relatively to the schedule) we still broadcast the data. We do so in order to have only one communication as opposed to a set of neighbor to neighbor communications if pipeline would be used. Notice that this type of broadcast can always be applied, without any constraints on the schedule.

For a given utilization processor, there is a delay between the time step it receives the data and the time step it first uses it. Therefore, the data must be stored from the time step it is received until the one corresponding to its last use. We call this interval the *lifetime* of the data. During this interval, the processor may receive other data to be used later. It therefore needs memory locations for all of these data. Of course, in order to optimize memory usage, one must not retain a memory

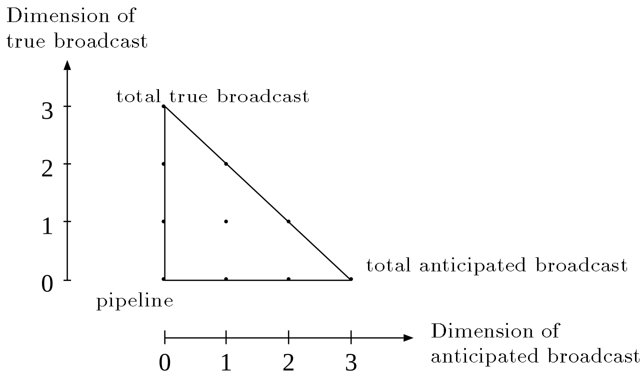

Fig. 1. Communications implementation: dimension of anticipated broadcast.

location for a data after its last use. This memory location should be freed and allocated to another data used by the processor at that time step [20].

For a problem with several dependences, there is an important degree of freedom in the choices of space-time transformations and communication schemes. In order to determine a large spectrum of efficient solutions one should apply a heuristic that proposes various space-time transformations and determines for each of them a communication cost and a memory cost, as presented in [10]. Such a heuristic analyses successively each of the dependences of the problem, and determines the constraints on the schedule and the allocation functions relatively to these various communication schemes.

The degree of freedom for a given dependence is pictured in figure 1 for a 3-dimensional utilization set. One end of the spectrum is characterized by a total true broadcast (3-dimensional true broadcast). This obviously corresponds to a strongly constrained schedule function: all the points of this 3-dimensional utilization set are scheduled at the same time step. This is the upper left point on the figure. If we relax the constraints on the schedule, the dimension of true broadcast decreases. A 2-dimensional true broadcast is represented by the points of ordinate 2. For the third dimension of the utilization set where there is no true broadcast, one may either choose to use pipeline (it corresponds to the left-hand side point), or to use anticipated broadcast (right-hand side point). The points of ordinate 0 correspond to space-time transformations avoiding true broadcast. In such a case there is still a large degree of freedom, from pipeline in all three dimensions (left-hand side point) to total (3-dimensional) anticipated broadcast (right-hand side point). The intermediate points correspond to partial anticipated broadcast communications associated with pipeline on the remaining dimensions.

In [10] the memory cost for anticipated broadcast was evaluated as a naive upper bound: the lifetime intervals were not taken into account, but considered as infinite. Our objective here is to symbolically compute the exact memory cost for a complete anticipated broadcast (right-hand side points of figure 1), as a function of the processor coordinates and the problem parameters.

4. Memory cost evaluation for anticipated broadcast

An anticipated broadcast can be implemented using two symetric memory management schemes:

1. the data is sent by the emission processor immediately after its computation to all its utilization processors[†] In this case the memory cost is entirely carried by the utilization processors.

2. the data is stored on the emission processor until its first use by one of the utilization processors. At this time step it is simultaneously sent to all the utilization processors.

The memory required on a processor to store the data for a given dependence $X \leftarrow Y$ is obviously strongly related to the allocation and the schedule functions associated with both variables. When any processor receives a single item of variable Y, i.e. there is a single utilization set mapped on any processor, the memory cost is one. The interesting situation occurs when a given utilization processor receives successively different items of variable Y (i.e. when several utilization sets are mapped on this processor).

A given item $Y[z_0]$ must be stored on utilization processor a between the following time steps:

- $\tau_{min}(z_0, p, a)$: the time step when the anticipated broadcast is realized.
  - In the first memory management scheme, the broadcast is realized immediately after the data is computed. In this case $\tau_{min}(z_0, p, a) = t_Y(z_0) + \vec{1}$ where $\vec{1}$ is the smallest time unit, i.e. d_λ-vector $(0, ..., 0, 1)$ for a d_λ-dimensional schedule.
  - In the second memory management scheme, the broadcast is realized immediately before the first use of the data. Since the schedule is affine, the first use of the data occurs at a vertex of the utilization set. Let $v_i(z_0, p)$ be the vertices of $\text{Util}_Y(z_0, p)$. It follows that $\tau_{min}(z_0, p, a) = \underset{i}{min}(t_X(v_i(z_0, p)))$.

- $\tau_{max}(z_0, p, a)$: the time step corresponding to the last use on this utilization processor. Let $\text{UP}(z_0, p, a) = alloc_Y^{-1}(a) \cap \text{Util}_Y(z_0, p)$ be the set of utilization points mapped on processor a. It is a convex parameterized polyhedron characterized by a set of parameterized vertices $v_i'(z_0, p, a)$. It follows that $\tau_{max}(z_0, p, a) = \underset{i}{max}(t_X(v_i'(z_0, p, a)))$.

[†] Whatever the dimension of the utilization set, the broadcast will be realized over all the utilization processors. An alternative would be to cut this global broadcast into successive broadcasts of smaller dimension.

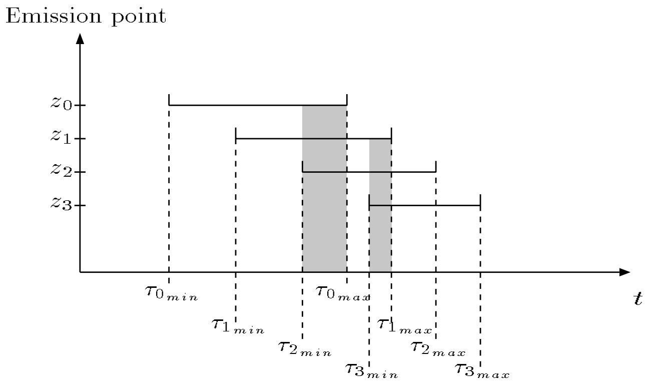

Fig. 2. Lifetime intervals for a given processor.

We call *lifetime* the time interval between $\tau_{min}(z_0, p, a)$ and $\tau_{max}(z_0, p, a)$. Notice that when the schedule function is multidimensional, we consider the lexicographic order on the time step vectors. Therefore, the lifetime is a union of parameterized polyhedra.

Definition 1 *For a given dependence from Y to X, the lifetime of $Y[z_0]$ on processor a is:*

$$\mathcal{L}(z_0, p, a) = \left\{ \tau \in \mathbb{Z}^{d_\lambda} | \tau_{min}(z_0, p, a) \preceq \tau \preceq \tau_{max}(z_0, p, a) \right\}.$$

When a given utilization processor receives successive data, they must be stored during their respective lifetimes. Two cases may occur:

- the various lifetimes are all disjoint, in which case the memory cost is one;
- the lifetimes do intersect, in which case the memory cost is the maximum at any time step of the number of intersecting sets. Figure 2 represents four lifetime intervals related to four emission points for a given processor. The maximal number of intersecting sets is 3, between $\tau_{2_{min}}$ and $\tau_{0_{max}}$ or between $\tau_{3_{min}}$ and $\tau_{1_{max}}$. Thus the maximal number of simultaneously alive data, that is the memory cost, is 3.

In the general case (when the lifetimes intersect) the amount of memory required on a processor is computed according to the following theorem.

Theorem 1 *For a given dependence from Y to X, the memory required on a utilization processor a is:*

$$mem(p, a) = \underset{z_0 \in \mathrm{Emit}_Y(p)}{Max} (EP(g(t_X^{-1}(\mathcal{L}(z_0, p, a)) \;\cap\; \mathcal{D}_{Eq}(p) \;\cap\; alloc_X^{-1}(a)))).$$

Proof. For a given point $z_0 \in \mathrm{Emit}_Y(p)$, the lifetime of $Y[z_0]$ on processor a is $\mathcal{L}(z_0, p, a)$. This set represents the time steps at which $Y[z_0]$ must be stored on utilization processor a. The inverse image by t_X intersected with the iteration

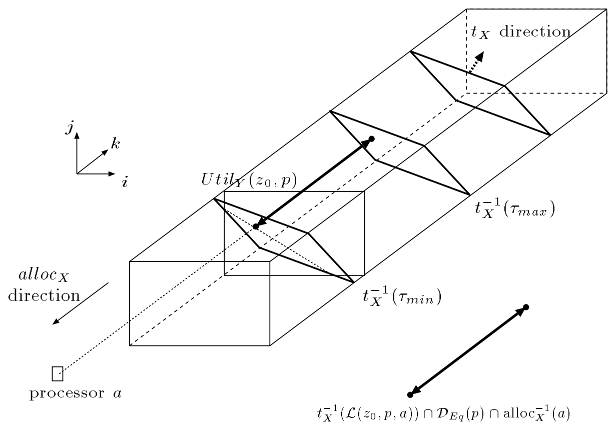

Figure 3: Active utilization points during the lifetime of data $Y[z_0]$ on processor a.

domain, $t_X^{-1}(\mathcal{L}(z_0,p,a)) \cap \mathcal{D}_{Eq}(p)$, is the set of all the points of the domain that are active during this lifetime. The intersection of this set with $alloc_X^{-1}(a)$ restricts it to the points mapped on processor a.

Figure 3 shows such a set of points. The main parallepiped represents the domain $\mathcal{D}_{Eq}(p)$. For a given emission point[‡] z_0, its corresponding utilization set $\mathrm{Util}_Y(z_0,p)$ is shown. The isotemporal spaces $t_X^{-1}(\tau_{min})$ and $t_X^{-1}(\tau_{max})$ are represented[§] Notice that τ_{min} depends on the memory management scheme as described above. The bold segment on the figure represents $\mathcal{A}(z_0,p,a) = t_X^{-1}(\mathcal{L}(z_0,p,a)) \cap \mathcal{D}_{Eq}(p) \cap alloc_X^{-1}(a)$. This segment, called the *activity set*, contains the utilization points whose corresponding data must be stored on processor a at the $Y[z_0]$ transmission time step.

In this example, there is only one point per utilization set belonging to the activity set. However, in the general case, there could be several of them in the same utilization set. Since they correspond to only one data, they require only one memory location. In order to compute the amount of memory required when $Y[z_0]$ is sent, we must count the number of utilization sets, or equivalently the number of corresponding emission points. To get these emission points we apply function g to the activity set. We compute the Ehrhart polynomial of the resulting set to get the memory cost related to z_0:

$$EP(g(\mathcal{A}(z_0,p,a)))$$

In order to get the final memory cost on processor a, we compute the maximum over $z_0 \in \mathrm{Emit}_Y(p)$ of this expression. □

[‡]Notice that this point belongs to the definition domain of variable Y, therefore it does not appear on the picture which focuses on variable X.

[§]For the sake of clarity, we represent only the part of the isotemporal spaces intersecting the domain.

In the simple case where the lifetimes do not intersect, theorem 1 simplifies as $mem(p,a) = 1$.

If the first memory management scheme is used, i.e. if the emission processor sends the data to the utilization processors immediately after its computation, then theorem 1 gives the total amount of memory required on any processor for this dependence. If the second memory management scheme is used, i.e. if the data is stored on the emission processor until its first use by one of the utilization processors, then one must compute the amount of memory required on an emission processor. This is expressed by the following theorem, whose proof is similar to the one of theorem 1, by substituting $\mathcal{L}$ by $\mathcal{L}'$.

Theorem 2 *For a given dependence from Y to X the memory required on an emission processor a, when using the second memory management scheme, is:*

$$mem(p,a) = \underset{z_0 \in \mathrm{Emit}_Y(p)}{Max} (EP(g(\ t_X^{-1}(\mathcal{L}'(z_0,p,a)) \cap\ \mathcal{D}_{Eq}(p)\ \cap\ alloc_X^{-1}(a)\ )))$$

where $\mathcal{L}'(z_0,p,a)) = \{\tau | t_Y(z_0) \prec \tau \preceq \tau_{min}(z_0,p,a)\}$ *is the time interval during which the data is stored on the emission processor.*

These two theorems characterize, for a given dependence, the memory required on both the emission and the utilization processors. For a problem, they have to be applied to all its dependences. This is illustrated in the next section on a example.

5. Example: Gaussian elimination

Let us consider the gaussian elimination problem, which triangulates an $N \times (N+1)$ matrix m. The resulting upper-triangular matrix is denoted by M. It is defined by the following system of affine recurrence equations:

$$\begin{aligned}
A[i,j,0] &= m_{i,j} & (1)\\
& \mathcal{D}_1(N) = \{(i,j) \in \mathbb{Z}^2 | 1 \leq i \leq N; 1 \leq j \leq N+1\} &\\
A[i,j,k] &= A[i,j,k-1] - A[i,k,k-1] * A[k,j,k-1]/A[k,k,k-1] & (2)\\
& \mathcal{D}_2(N) = \left\{ (i,j,k) \in \mathbb{Z}^3 \;\middle|\; \begin{array}{l} k+1 \leq i \leq N; k+1 \leq j; \\ j \leq N+1; 1 \leq k \leq N-1 \end{array} \right\} &\\
M[i,j] &= A[i,j,i-1] & (3)\\
& \mathcal{D}_3(N) = \{(i,j) \in \mathbb{Z}^2 | 1 <= i <= N; i <= j <= N+1\} &
\end{aligned}$$

In the following, we focus on computation equation (2) defined over iteration domain $\mathcal{D}_2(N)$. In this equation, the first dependence is uniform and is therefore of no interest in the discussion. It always requires one memory location which will be added to the results presented hereunder. The last three dependences are numbered (2) to (4).

In order to find a schedule and an allocation function a dependence analysis has first to be conducted. The utilization sets related to each dependence are computed table 1. Using the communication minimization algorithm presented in [10] one can determine efficient space-time transformations. When restricting to

Table 1. Utilization sets.

Dependence		$\mathrm{Util}(i_0, j_0, k_0, N)$
(2)	$A[i,k,k-1]$	$\left\{ \begin{pmatrix} i_0 \\ j \\ k_0+1 \end{pmatrix} \middle\vert\, k_0+1 \le j \le N+1 \right\}$
(3)	$A[k,j,k-1]$	$\left\{ \begin{pmatrix} i \\ j_0 \\ k_0+1 \end{pmatrix} \middle\vert\, k_0+1 \le i \le N \right\}$
(4)	$A[k,k,k-1]$	$\left\{ \begin{pmatrix} i \\ j \\ k_0+1 \end{pmatrix} \middle\vert\, k_0+1 \le i \le N, k_0+1 \le j \le N+1 \right\}$

one-dimensional schedules, the schedule matrix λ_A can be chosen in the following interval:

- $\lambda_A = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$: this is the fastest affine schedule relatively to the dependences. It enforces true broadcast for all the dependences. When applying the results presented in this paper, one of course obtains a total amount of memory equal to 4.
- $\lambda_A = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$: this schedule is the fastest affine schedule without any true broadcast.

In the following, we focus on the first memory management scheme described above, in which the broadcast is realized immediately after the data is computed. We consider schedule matrix $\begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$, where true broadcast is partially required, and anticipated broadcast can be used. In [21], the same example using schedule matrix $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ is presented ; all three dependences may be realized using anticipated broadcast.

We choose the allocation function defined by matrix $\sigma_A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$. We denote by $\begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$ the coordinates of a given processor. In order to compute the lifetime, the sets UP of utilization points mapped on a given processor a are computed in table 2. Notice that each set $UP(i_0, j_0, k_0, N, a_0, a_1)$ is reduced to a single point which is therefore its unique vertex, denoted by $v'_1(i_0, j_0, k_0, N, a_0, a_1)$.

Using the first memory management scheme, the extremal values τ_{min} and τ_{max} are, for all three dependences:

$$\tau_{min}(i_0, j_0, k_0, N, a_0, a_1) = t_A(i_0, j_0, k_0) + 1 = j_0 + k_0 + 1$$

$$\tau_{max}(i_0, j_0, k_0, N, a_0, a_1) = t_A(v'_1(i_0, j_0, k_0, N, a_0, a_1)) = a_1 + k_0 + 1$$

The lifetime of $A[i_0, j_0, k_0]$ on processor $\begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$ for the three dependences is:

$$\mathcal{L}(i_0, j_0, k_0, N, a_0, a_1) = \{\tau \in \mathbb{Z} | j_0 + k_0 + 1 \le \tau \le a_1 + k_0 + 1\}$$

Table 2. Utilization points mapped on processor a.

Dependence		$\mathrm{UP}(i_0, j_0, k_0, N, a_0, a_1)$
(2)	$A[i,k,k-1]$	$\left\{ \begin{pmatrix} a_0 \\ a_1 \\ k_0+1 \end{pmatrix} \middle\vert\, i_0 = a_0, k_0+1 \leq a_1 \leq N+1 \right\}$
(3)	$A[k,j,k-1]$	$\left\{ \begin{pmatrix} a_0 \\ a_1 \\ k_0+1 \end{pmatrix} \middle\vert\, a_1 = j_0, k_0+1 \leq a_0 \leq N \right\}$
(4)	$A[k,k,k-1]$	$\left\{ \begin{pmatrix} a_0 \\ a_1 \\ k_0+1 \end{pmatrix} \middle\vert\, k_0+1 \leq a_0 \leq N, k_0+1 \leq a_1 \leq N+1 \right\}$

The activity set used in theorem 1 is defined by:

$$\begin{aligned}
&\mathcal{A}(i_0, j_0, k_0, N, a_0, a_1) \\
&= t_A^{-1}(\mathcal{L}(i_0, j_0, k_0, N, a_0, a_1)) \cap \mathcal{D}_2(N) \cap alloc_A^{-1}(a_0, a_1) \\
&= \left\{ \begin{pmatrix} i \\ j \\ k \end{pmatrix} \middle\vert \begin{array}{l} j_0 + k_0 + 1 \leq j + k \leq a_1 + k_0 + 1, k+1 \leq i \leq N, \\ k+1 \leq j \leq N+1, 1 \leq k \leq N-1, i = a_0, j = a_1 \end{array} \right\} \\
&= \left\{ \begin{pmatrix} a_0 \\ a_1 \\ k \end{pmatrix} \middle\vert\, j_0 + k_0 + 1 \leq a_1 + k, k \leq k_0 + 1, 1 \leq k \leq N-1 \right\}
\end{aligned}$$

The memory cost related to each dependence, given by theorem 1, is summarized table 3. Notice that the memory required is reduced to 1 location for dependence (3). This result is not surprising since this dependence is implemented as a true broadcast due to the schedule.

Table 3. Memory cost related to each dependence.

Dep.	$g(\mathcal{A}(i_0, j_0, k_0, N, a_0, a_1))$	$mem(N, a_0, a_1)$
(2)	$\left\{ \begin{pmatrix} a_0 \\ k \\ k-1 \end{pmatrix} \middle\vert\, \begin{array}{l} j_0+k_0+1 \leq a_1+k, k \leq \\ k_0+1 \end{array} \right\}$	$\left\lfloor \frac{a_1+1}{2} \right\rfloor$
(3)	$\left\{ \begin{pmatrix} k \\ a_1 \\ k-1 \end{pmatrix} \middle\vert\, \begin{array}{l} j_0+k_0+1 \leq a_1+k, k \leq \\ k_0+1 \end{array} \right\}$	1
(4)	$\left\{ \begin{pmatrix} k \\ k \\ k-1 \end{pmatrix} \middle\vert\, \begin{array}{l} j_0+k_0+1 \leq a_1+k, k \leq \\ k_0+1 \end{array} \right\}$	$\left\lfloor \frac{a_1+1}{2} \right\rfloor$

Memory required on any processor.

The total amount of memory on a given processor is the sum on all four dependences of $mem(N, a_0, a_1)$:

$$mem(N, a_0, a_1) = 1 + \left\lfloor \frac{a_1+1}{2} \right\rfloor + 1 + \left\lfloor \frac{a_1+1}{2} \right\rfloor = 2 \left\lfloor \frac{a_1+1}{2} \right\rfloor + 2$$

Applying the allocation function to the iteration domain $\mathcal{D}_2(N)$, the active virtual processor space is:

$$alloc(\mathcal{D}_2(N)) = \{(a_0, a_1) \in \mathbb{Z}^2 | 2 \leq a_0 \leq N, 2 \leq a_1 \leq N+1\}.$$

According to the above formula the smallest amount of memory is 4 and it is obtained for processor of ordinate 2. The largest amount of memory is upper bounded by $N+4$ for the processors of ordinate $N+1$.

This is the total amount of memory required when all dependences are realized as total anticipated broadcast. If this memory cost is too expensive one can of course realize some of these communications as propagation.

6. Conclusion

The work presented in this paper evaluates the memory cost related to dependences whose related communications are implemented using anticipated broadcast. For a given set of per variable space-time transformations, the exact amount of memory required on any processor is computed as a function of the program parameters and of the processor coordinates. This computation is realized using Ehrhart polynomials, which give the number of integer points contained in a parameterized convex polytope. The amount of memory we compute for a given processor is optimal: it corresponds to the minimum number of memory locations required and therefore relies on memory reuse. Hence, as soon as a data is no longer used by a processor, its memory location is released and allocated to another data. In [20] Wilde and Rajopadhye also deal with memory reuse based on lifetime information, but do not take advantage of the usage information (called usage table in their paper) to integrate broadcast communications. They define a memory allocation function which optimizes memory reuse, but do not give any method to compute the actual memory cost.

In this paper we only deal with total anticipated broadcast, that is for a multi-dimensional utilization set, we only use broadcast communications, either true or anticipated. An alternative could be to use, for dimensions where true broadcast does not apply, anticipated broadcast on some of the dimensions and propagation on the remaining ones. A first extension of this work will be to deal with partial anticipated broadcast: use simultaneously anticipated broadcast (in some of the dimensions) and propagation (in the other ones).

This work is an element of a global algorithm for efficient loop nests parallelization. The whole process consists in finding an optimal space-time transformation, according to various optimization criteria such as latency, communication cost, or memory cost. Latency optimization criteria are well known, as presented by Feautrier in [22,23]. An algorithm dealing with communication optimization has been presented in [12,10]. We propose here an approach to evaluate the memory cost related to anticipated broadcast. These results should of course be integrated in the whole process. Notice that, at that stage, memory cost is evaluated for a given space-time transformation. A further extension of this work will consist in finding appropriate conditions to determine memory-optimal space-time transformations, as it is done in [12,10] for communication optimization.

For a given problem, the various optimization criteria may be antinomical: using anticipated broadcast reduces the communication cost¶ but increases memory cost. Therefore, a trade-off between these optimizations must be found in order to find globally efficient transformations. The approach developped in [10] should be extended to take advantage of anticipated broadcast. The cost function should then integrate latency and required amount of memory, besides the number of communications.

References

[1] J. Ramanujam and P. Sadayappan. Compile-time techniques for data distribution in distributed memory machines. *IEEE Transactions on parallel and distributed systems*, 2(4):472–482, October 1991.

[2] J. Li and M. Chen. The data alignment phase in compiling programs for distributed-memory machines. *Journal of Parallel and Distributed Computing*, 13(2), 1991.

[3] J.M. Anderson and M.S. Lam. Global optimizations for parallelism and locality on scalable parallel machines. *ACM Sigplan Notices*, 28(6):112–125, 1993.

[4] A. Darte and Y. Robert. On the alignment problem. *Parallel Processing Letters*, 4(3):259–270, 1994.

[5] P. Feautrier. Towards automatic distribution. *Parallel Processing Letters*, 4(3):233–244, 1994.

[6] D. Bau, I. Kodukula, V. Kotlyar, K. Pingali, and P. Stodghill. Solving alignment using elementary linear algebra. In *Languages and Compilers for Parallel Computing*. LNCS 892, Springer-Verlag, 1995.

[7] M. Dion and Y. Robert. Mapping affine loop nests : new results. In B. Hertzberger and G. Serazzi, editors, *Int. Conf. on High-Performance Computing and Networking*, pages 184–189. Springer Verlag, May 1995. Milan, Italy.

[8] S. Chatterjee, J.R. Gilbert, L. Oliker, R. Schreiber, and T.J. Sheffler. Algorithms for automatic alignment of arrays. *Journal of Parallel and Distributed Computing*, 38(2):145–157, November 1996.

[9] C. Mongenet. Affine dependence classification for communications minimization. *International Journal of Parallel Programming*, 25(6), December 1997.

[10] V. Loechner and C. Mongenet. Communication optimization for affine recurrence equations using broadcast and locality. *Int. J. of Parallel Programming*, February 2000.

[11] C. Mongenet, Ph. Clauss, and G.R. Perrin. Geometrical tools to map systems of affine recurrence equations on regular arrays. *Acta Informatica*, 31:137–160, 1994.

[12] V. Loechner and C. Mongenet. Solutions to the communication minimization problem for affine recurrence equations. In Lengauer, Griebl, and Gorlatch, editors, *Europar'97*, pages 328–337. Springer-Verlag, August 1997. LNCS 1300.

[13] C. Lengauer. Loop parallelization in the polytope model. In *CONCUR 93*, 1993.

[14] Ph. Clauss and V. Loechner. Parametric analysis of polyhedral iteration spaces. *Journal of VLSI Signal Processing*, 19(2):179–194, July 1998. http://icps.u-strasbg.fr/pub-98.

[15] Ph. Clauss. Counting solutions to linear and nonlinear constraints through Ehrhart polynomials: Applications to analyze and transform scientific programs. In *10th ACM*

¶on appropriate architectures where broadcast is efficiently implemented.

Int. Conf. on Supercomputing, Philadelphia, 1996.

[16] J. Ramanujam. Non-unimodular transformations of nested loops. In *Supercomputing 92*, pages 214–223. IEEE Computer Society Press, November 1992.

[17] J.-F. Collard, P. Feautrier, and T. Risset. Construction of DO loops from systems of affine constraints. *Parallel Processing Letters*, 5(3):421–436, 1995.

[18] O. Albiez. Parcours d'une superposition de domaines. In D. Mery and G.R. Perrin, editors, *RenPar'10*, 1998.

[19] F. Quilleré, S. Rajopadhye, and D. Wilde. Generation of efficient nested loops from polyhedra. Research report, IRISA, Rennes, France, 1999.

[20] D. Wilde and S. Rajopadhye. Memory reuse analysis in the polyhedral model. *Parallel Processing Letter*, 7(2):203–215, 1997.

[21] V. Loechner and C. Mongenet. Memory cost due to anticipated broadcast. In *8th Workshop on Compilers for Parallel Computers, CPC*, January 2000. Available as ICPS RR 99-03, http://icps.u-strasbg.fr/pub-99/.

[22] P. Feautrier. Some efficient solutions to the affine scheduling problem, part 1 : one dimensional time. *Int. J. of Parallel Programming*, 21(5):313–348, October 1992.

[23] P. Feautrier. Some efficient solutions to the affine scheduling problem, part 2 : multi-dimensional time. *Int. J. of Parallel Programming*, 21(6), December 1992.