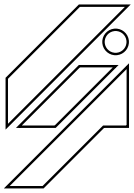
Loop Partitioning versus Tiling for Cache-based Multiprocessors.

Fabrice Rastello, Yves Robert



Laboratoire de l'Informatique du Parallélisme

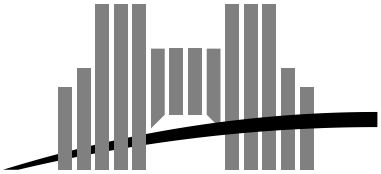
École Normale Supérieure de Lyon
Unité de recherche associée au CNRS n° 1398

Loop Partitioning versus Tiling for Cache-based Multiprocessors

Fabrice RASTELLO
Yves ROBERT

February 1998

Research Report N° 98-13



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.00
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

Loop Partitioning versus Tiling for Cache-based Multiprocessors

Fabrice RASTELLO
Yves ROBERT

February 1998

Abstract

In this paper, an efficient algorithm to implement *loop partitioning* is introduced and evaluated. We improve recent results of Agarwal, Kranz and Natarajan [1] in several directions. We give a more accurate estimation of the cumulative footprint, and we derive a much more powerful algorithm to determine the optimal tile shape. We illustrate the superiority of our algorithm on the same examples as in [1] to ensure the fairness of the comparisons.

Keywords: Compilation technique, hierarchical memory systems, loop partitioning, tiling, cache, data locality, footprint.

Résumé

Nous présentons dans ce papier une heuristique efficace permettant de faire de la distribution de boucles. Nous appuyons notre travail sur un papier récent de Agarwal, Kranz et Natarajan [1] que nous améliorons dans de nombreuses directions. Plus précisement, nous proposons une estimation des empreintes cumulées de tuiles plus précise ; nous proposons une heuristique puissante permettant de minimiser cette empreinte cumulée ; enfin, nous montrons la superiorité de notre algorithme en l'appliquant aux exemples donnés dans [1] afin d'assurer l' équité de notre comparaison.

Mots-clés: Techniques de compilation, systèmes à mémoire hiérarchisée, distribution de boucles, pavage, mémoire cache, localité de données, empreintes de tuiles.

Loop Partitioning versus Tiling for Cache-based Multiprocessors*

Fabrice Rastello and Yves Robert

LIP, Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France
e-mail: [Fabrice.Rastello, Yves.Robert]@ens-lyon.fr

February 1998

Abstract

In this paper, an efficient algorithm to implement *loop partitioning* is introduced and evaluated. We improve recent results of Agarwal, Kranz and Natarajan [1] in several directions. We give a more accurate estimation of the cumulative footprint, and we derive a much more powerful algorithm to determine the optimal tile shape. We illustrate the superiority of our algorithm on the same examples as in [1] to ensure the fairness of the comparisons.

Key words: compilation technique, hierarchical memory systems, loop partitioning, tiling, cache, data locality, footprint.

Corresponding author: Yves Robert
LIP, Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France
Phone: + 33 4 72 72 80 37, Fax: + 33 4 72 72 80 80
E-mail: Yves.Robert@ens-lyon.fr

*This work was supported by the CNRS–ENS Lyon–INRIA project *ReMaP* and by the Eureka Project *EuroTOPS*.

1 Introduction.

The aim of this paper is to derive an efficient algorithm to implement *loop partitioning*, a compilation technique to make the best use of hierarchical memory systems when dealing with loop nests computations. This technique clearly applies to cache-based multiprocessors, because data re-use and locality are crucial for such systems. Loop partitioning is also relevant for implementing out-of-core algorithms (the tandem 'cache–local memory" in the former sentence being replaced by the tandem "local memory–secondary storage").

Loop partitioning amounts to divide an iteration space into hyper-parallelepipeds, whose size and shape are optimized according to some criteria. It is closely related to *tiling* [11, 14, 2, 6, 13, 5, 10], a technique also known as loop blocking [15], whose objective is to increase the granularity of computations, the locality of data references, and the computation-to-communication ratio of fully permutable loop nests. In fact, loop partitioning and tiling have similar objectives: the basic idea of both techniques is to group elemental computation points into tiles that will be viewed as computational units. The larger the tiles, the more efficient the computations performed using state-of-the-art processors with pipelined arithmetic units and a multilevel memory hierarchy (illustrated by recasting numerical linear algebra algorithms in terms of blocked Level 3 BLAS kernels [7, 9]).

But loop partitioning and tiling operate in different contexts. Tiling is valid only if the loops are fully permutable [8, 12, 16], and the optimization criteria aim at minimizing the communication-to-computation ratio. Loop partitioning can be applied to any loop nest with affine dependences, and the optimization criteria is to minimize the number of accessed data. We explicit this difference in Section 4.1. Still, because tiling and loop partitioning share many characteristics, we will be able to make use of recent results on tiling [4] to derive our algorithm for loop partitioning.

Loop partitioning has been studied by Agarwal, Kranz and Natarajan [1]. The central contribution of [1] is a method for deriving an optimal hyper-parallelepiped tiling of iteration spaces, where the optimization criterion is the following: given a fixed tile size (typically the fraction of the cache that is available to store program data), determine the tile shape so that the number of accessed data (the so-called *cumulative footprint* in [1]) is kept minimal.

In this paper we build upon the results of [1], which we improve in several directions. We give a more accurate estimation of the cumulative footprint, and, more importantly, we derive a powerful algorithm to determine the optimal tile shape. While the search was limited to rectangular shapes (which corresponds to searching for diagonal matrices) in [1], we are able to deal with arbitrary parallelepipeds (which corresponds to searching for arbitrary non)singular matrices). We illustrate our algorithm on the same examples as in [1] to ensure the fairness of the comparisons.

The paper is organized as follows: we summarize the approach of Agarwal, Kranz and Natarajan [1] in Section 2. We introduce the better estimation of the cumulative footprint in Section 3, and we explain how to solve the optimization problem in Section 4. We show several examples in Section 5. We give some final remarks in Section 6.

2 Survey of previous work.

In this section, we summarize the approach of Agarwal, Kranz and Natarajan [1]. We formally state the problem to be solved. After giving some notations, we survey their main results.

2.1 Optimal tiling for minimizing communications.

We start with the following example:

Example 1

Consider the following loop nest:

```
Doall (i=0:99, j=0:99)
| A[i,j]=B[i,j]+B[i+1,j-2]+C[2i,2i+j]+C[2i+1,2i+j]
EndDo
```

In order to increase the granularity of computation, and the locality of data dependences, loop partitioning may be used. This method consists in grouping neighboring points of the iteration space into a single parallelepiped-shaped tile. Tiles are then considered as atomic and distributed over the processors. For example, the loop nest of Example 1 can be tiled with rectangles of sizes 10×5 as follows:

```
Doall (I=0:9, J=0:19)
| Do (i=0:9,j=0:4)
| | A[I*10+i,J*5+j]= B[I*10+i,J*5+j]+B[I*10+i+1,J*5+j-2]
| |                  +C[2(I*10+i),2(I*10+i)+J*5+j]
| |                  +C[2(I*10+i)+1,2(I*10+i)+J*5+j]
| EndDo
EndDo
```

Of course two different tilings do not lead to the same execution time: the volume and the shape of the tiles are important parameters that must be determined. Usually, the tile size is fixed: it is chosen so as to fully utilize the cache (or more precisely the fraction of the cache that is available to store data). Given a fixed tile size (or volume), the tile shape has a great impact on the amount of loaded data. Determining the best tile shape so as to minimize the number of loaded data is the optimization problem that is dealt with in [1].

2.2 Notations.

We need a few notations to formalize the problem of computing (maybe approximately) the number of loaded data during the computation of a tile:

1. *The computation* "`A[i,j]=...`" is represented by the column vector $\overrightarrow{\imath} = \begin{pmatrix} i \\ j \end{pmatrix}$
2. As data references are assumed to be affine, *the access* during computation $\overrightarrow{\imath}$ of the data $g(\overrightarrow{\imath})$ can be represented by the expression $\mathbf{G}\overrightarrow{\imath} + \overrightarrow{a}$. Hence, in Example 1, the reference to
   - `B[i,j]` can be represented by the couple $(\mathbf{G}_1, \overrightarrow{a_{1,1}}) = \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right)$
   - `B[i+1,j-2]` can be represented by the couple $(\mathbf{G}_1, \overrightarrow{a_{1,2}}) = \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -2 \end{pmatrix} \right)$
   - `C[2i,2i+j]` can be represented by the couple $(\mathbf{G}_2, \overrightarrow{a_{2,1}}) = \left(\begin{pmatrix} 2 & 2 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right)$

- `C[2i+1,2i+j]` can be represented by the couple $(\mathbf{G}_3, \overrightarrow{a_{3,1}}) = \left(\begin{pmatrix} 2 & 2 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right)$

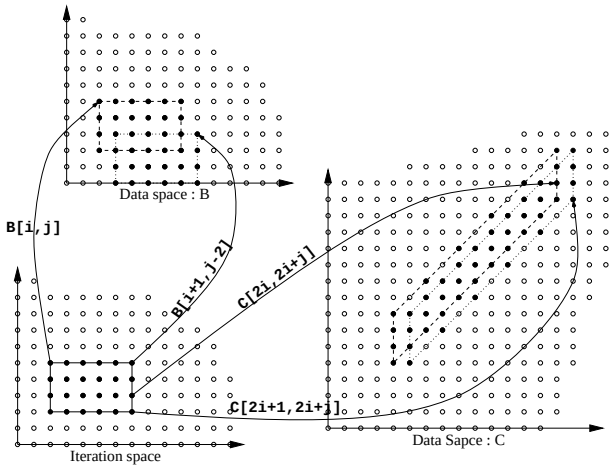


Figure 1: References `B[i,j]` and `B[i+1,j-2]` refer to some common data, which must be counted only once. However, references `C[2i,2i+j]` and `C[2i+1,2i+j]` do not intersect.

We point out that we gave the same name G_1 to the matrix representing reference `B[i,j]` and to the matrix representing reference `B[i+1,j-2]`. This is because the loaded data corresponding to these tow references overlap. On the other hand, we gave a different matrix name for references `C[2i,2i+j]` and `C[2i+1,2i+j]`, even though they deal with the same array C and the same matrix $\begin{pmatrix} 2 & 2 \\ 0 & 1 \end{pmatrix}$: this is because the loaded data do not intersect (see Figure 1).

3. The data loaded by a single reference during the execution of one tile is called *the footprint* of this reference, and the total number of loaded data is called the *cumulative footprint*. The previous example shows that the cumulative footprint is not the sum of all footprints. In the important case of the same data array being accessed twice, say, one time with the couple $(\mathbf{G}, \overrightarrow{a_1})$ and the other time with the with the couple $(\mathbf{G}, \overrightarrow{a_2})$, where $G^{-1}(\overrightarrow{a_2} - \overrightarrow{a_1})$ has integer components, then both footprints have a significantly large intersection, and the computation needs to be refined. Note this happens each time the same data array is accessed twice with the same unimodular matrix (because G^{-1} is integer, $G^{-1}(\overrightarrow{a_2} - \overrightarrow{a_1})$ always has integer components); this is the case for the first two references of Example 1. In all other cases, different footprints have an empty or negligible intersection [1].

4. A tile in a n-dimensional box determined by n free vectors $\overrightarrow{u_1}, ..., \overrightarrow{u_n}$, where n is the number of loops in the loop nest. See Figure 2 for an example with $n = 2$. Hence, a tile can be represented by a $n \times n$ non-singular matrix H, built up from the column vectors $\overrightarrow{u_1}, ..., \overrightarrow{u_n}$. The volume of the tile is $|detH|$. In fact this matrix H is exactly the inverse of the matrix defined by Irigoin and Triolet [11] from the normal vectors to the faces of the tile.

The first objective of Agarwal, Kranz and Natarajan [1] was to find a precise evaluation of the cumulative footprint. Then they fix the tile size, and they search for the tile shape that minimizes the expression of the cumulative footprint.

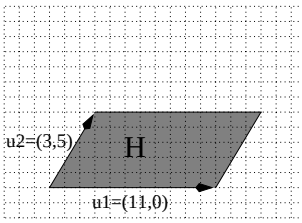


Figure 2: The tile can be represented by the matrix $H = \begin{pmatrix} 11 & 3 \\ 0 & 5 \end{pmatrix}$. Its size is $|detH| = 55$.

2.3 Results.

As already said, the difficulty is to correctly estimate the cumulative footprint for several references that make accesses to a common data (cf Figure 3).

The solution proposed by Agarwal, Kranz and Natarajan [1] is the following. Consider the references $(G, \overrightarrow{a_1}), ..., (G, \overrightarrow{a_k})$ where G is unimodular:

- Let $(\overrightarrow{x_1}, ..., \overrightarrow{x_n})$ denote the canonical basis of $\mathbb{R}^n$, and

$$\overrightarrow{a} = \left(\max_{1 \leq j \leq k} |\overrightarrow{x_i}.\overrightarrow{a_j}| \right)_{1 \leq i \leq n}$$

- If $D = (\overrightarrow{d_1} \cdots \overrightarrow{d_n})$ is a $n \times n$ matrix built up with column vectors $\overrightarrow{d_i}$, let $det(D_{j \rightarrow i})$ represent the determinant of the matrix D_j obtained by replacing $\overrightarrow{d}_j$ in D by $\overrightarrow{a}$.

Then, as intuitively explained by Figures 3 and 4, the cumulative footprint for the references $(G, \overrightarrow{a_1}), ..., (G, \overrightarrow{a_k})$ can be approximated by

$$|det(D)| + \sum_{j=1}^{n} |det(D_{j \rightarrow \overrightarrow{a}})|, \quad \text{where } D = GH.$$

Using the notations of Figure 3, $V_{calc} = |det(D)| = |det(H)|$ (because G is unimodular) is the tile size (or volume), and $V_{com} = \sum_{j=1}^{n} |det(D_{j \rightarrow \overrightarrow{a}})|$. We use the intuitive name V_{com} because the shadowed area in Figure 3 would correspond to communications in the context of tiling (while indeed they correspond to loads in the context of loop partitioning.

With this approximation of the cumulative footprint, Agarwal, Kranz and Natarajan [1] are able to analytically solve the optimization problem. However, they have the very restrictive assumption that the tiles are rectangular, i.e. they limit their search to diagonal matrices H. We extend their results in two directions: first, we give a more accurate estimation of the cumulative footprint. Second (and more importantly), we provide a general heuristic to solve the optimization problem for parallelepiped tiles, i.e. for arbitrary matrices H.

3 Estimating the cumulative footprint.

To motivate a more precise estimation of the cumulated footprint, consider the following example:

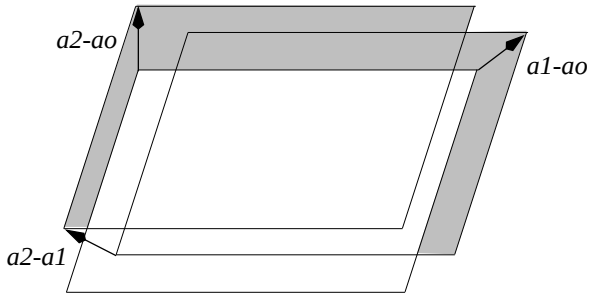


Figure 3: Cumulative footprint for the three references to a common array, $(G, \overrightarrow{a_0})$, $(G, \overrightarrow{a_1})$ and $(G, \overrightarrow{a_2})$. If V_{calc} is the volume of the tile, V_{com} the volume of the shadowed part, then the cumulative footprint of those three references is $V_{calc} + V_{com}$. One must find a good approximation for the expression of V_{com}.

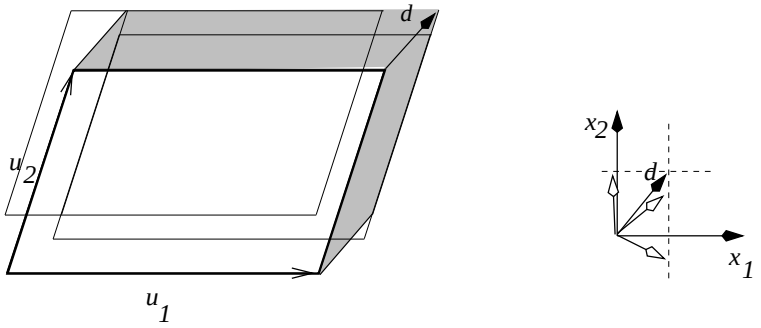


Figure 4: If $\overrightarrow{d} = (d_i)_{1 \leq i \leq n}$, d_i is obtained by taking the maximum value of $|(\overrightarrow{a_k} - \overrightarrow{a_l}).\overrightarrow{x_i}|$ where $\overrightarrow{x_i}$ is the column vector $(\delta_{ij})_{1 \leq j \leq n}$. With this notation, V_{com} can be approximated by $|det(\overrightarrow{u_1}, \overrightarrow{d})| + |det(\overrightarrow{u}_2, \overrightarrow{d})|$.

Example 2

```
Do (i=0:N, j=0:M)
| A[i,j]=B[i,j]+B[i+1,j+1]
EndDo
```

Suppose we want V_{calc} to be equal to 100. Then the tile H that minimizes the expression $|detH| + |detH_{1\rightarrow\overrightarrow{a}}| + |detH_{2\rightarrow\overrightarrow{a}}|$ with $\overrightarrow{a} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ is the square tile $H = 10.I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. This tile (see Figure 5) leads to $V_{com} = 19$. However, the tile $H' = \frac{1}{\sqrt{2}} \begin{pmatrix} 100 & 1 \\ 100 & -1 \end{pmatrix}$ would lead to $V_{com} = 1$.

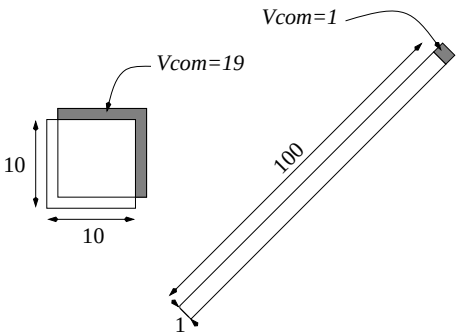


Figure 5: Comparison of the cumulative footprint for the tiles H and H' that have the same volume $V_{calc} = 100$.

In the light of this example, we propose a new, more accurate, expression of the cumulative footprint that takes into account the directions of the vectors $\overrightarrow{a_{i,j}}$.

Letting $\overrightarrow{b_i} = G^{-1}\overrightarrow{a_i}$, the expression for the cumulative footprint becomes

$$V = V_{calc} + \sum_{k=1}^{n} \max_{j<j'} \left| det\left(H_{k\rightarrow(\overrightarrow{b_j}-\overrightarrow{b_{j'}})}\right)\right|.$$

Moreover, since H is a non-singular matrix, let $E = H^{-1} = (\overrightarrow{e_1}\cdots\overrightarrow{e_n})^T$ be its inverse made up with the row vectors $\overrightarrow{e_1}, ..., \overrightarrow{e_n}$. Then $det(H_{k\rightarrow\overrightarrow{b}}) = det(H).det(\overrightarrow{e_k}^T.\overrightarrow{b})$. Hence, minimizing the above expression corresponds to finding a non-singular matrix E such that $|det(E^{-1})| = |det(H)| = V_{calc}$ and such that

$$V = V_{calc} + V_{calc}.\sum_{k=1}^{n} \max_{j<j'} \left|\overrightarrow{e_k}^T.(\overrightarrow{b_j} - \overrightarrow{b_{j'}})\right|$$

is minimized.

So far we have dealt with the same access matrix G. If we have m distinct access matrices G_i, let $\overrightarrow{b_{i,j}} = G_i^{-1}\overrightarrow{a_{i,j}}$ and $c_{i,j}$ the elements of C_i where $\begin{cases} C_1 = \{\overrightarrow{b_{1,j}} - \overrightarrow{b_{1,j'}},\ j < j'\} \\ C_i = \{\overrightarrow{b_{1,j}} - \overrightarrow{b_{1,j'}},\ j \neq j'\} \quad \text{for } i > 1 \end{cases}$. The

expression to be minimized becomes

$$\begin{aligned} V &= \sum_{i=1}^{m} \left(V_{calc} + V_{calc}. \sum_{k=1}^{n} \max_{j<j'} \left(|\overrightarrow{e_k}^T . (\overrightarrow{b_{i,j}} - \overrightarrow{b_{i,j'}})| \right) \right) \\ &= mV_{calc} + V_{calc}. \left(\max_{j<j'} \left| \overrightarrow{e_k}^T . (\overrightarrow{b_{1,j}} - \overrightarrow{b_{1,j'}}) \right| + \sum_{i=2}^{m} \max_{j \neq j'} \left(\overrightarrow{e_k}^T . (\overrightarrow{b_{i,j}} - \overrightarrow{b_{i,j'}}) \right) \right) \\ &= mV_{calc} + V_{calc}. \left(\max_{j<j'} \left| \overrightarrow{e_k}^T . \overrightarrow{c_{1,j}} \right| + \sum_{i=2}^{m} \max_{j} \left(\overrightarrow{e_k}^T . \overrightarrow{c_{i,j}} \right) \right) \end{aligned}$$

The sum can be shifted inside the max, at the price of an increase in the number of terms: for $i \in [1, m]$ assume there are d_i vectors $\overrightarrow{c_{i,j}}$, $1 \leq j \leq d_i$. Let

$$\sigma : \left(\begin{array}{rcl} \{1, .., m\} & \longrightarrow & \mathbb{N} \\ i & \longmapsto & 1 \leq j \leq d_i \end{array} \right)$$

and by

$$\overrightarrow{l_\sigma} = \sum_{i=1}^{m} \overrightarrow{c_{i,\sigma(i)}}$$

Then

$$V = mV_{calc} + V_{calc}. \sum_{k=1}^{n} \max_{\sigma} \left| \overrightarrow{e_k}^T . \overrightarrow{l_\sigma} \right| \tag{1}$$

4 Solving the optimization problem.

4.1 Related problems.

The problem of minimizing the expression (1) is difficult. In fact, we know how to minimize the two related expressions:

Problem 1 If $(\overrightarrow{a_1}, ..., \overrightarrow{a_m})$ are m free vectors, and $|detE| = \frac{1}{V_{calc}}$, Boulet et al. propose in [2] a solution for minimizing the following expression:

$$\sum_{i=1}^{m} \sum_{k=1}^{n} \overrightarrow{e_k}. \overrightarrow{a_i}$$

The solution is simply $E = A^{-1}$ if $m = n$ but gets very complex if $m \neq n$.

Note that the $\overrightarrow{a_i}$ represent dependences in the context of tiling fully permutable loop nests, hence their components are known to be nonnegative. We do not have this property in our loop partitioning problem.

Problem 2 Let $\mathcal{H}_n$ be the Hadamard matrix of size n, i.e. a square matrix of coefficients either 0 or 1 and whose determinant is maximal [3]. If $A = (\overrightarrow{a_1} \cdots \overrightarrow{a_n})$ is non-singular, then $E = \mathcal{H}_n A^{-1}$ minimizes the following expression (see [5]):

$$\sum_{k=1}^{n} \max_{1 \leq j \leq n} \overrightarrow{e_k}. \overrightarrow{a}_j.$$

Again, if A is not square, the problem becomes very difficult.

So to speak, our optimization problem lies somewhere between Problem 1 and Problem 2. We introduce an heuristic that is inspired by the solution of of Problem 2 given in [5]. First, we reduce the problem to the vector subspace generated by the set of vectors $\{\overrightarrow{l_\sigma}\}$. Second, we choose among those l_σ, n' of them, where n' is the dimension of the generated vector sub-space (usually n' is equal to the number of loops n but it can be smaller in degenerate cases). These selected n' vectors should be free vectors that give the most accurate representation of all the others. For that purpose, we propose to choose n' vectors that (almost) maximize the volume of the polytope that they generate. Then, we solve this problem using the solution of Problem 2 in the considered vector sub-space (we are in the simple case of a square matrix). Finally, for the remaining dimensions, we choose orthogonal vectors of length 1.

4.2 Our heuristic.

To formalize the previous discussion:

- $L = (\overrightarrow{l_1} \cdots \overrightarrow{l_m})$ is a rectangular matrix made up with m column vectors of size n.
- $\mathcal{H}_n$ is the Hadamard matrix of dimension n.
- If $D = (\overrightarrow{d_1} \cdots \overrightarrow{d_m})$ is a rectangular matrix made up with m column vectors of size n that generate a n-dimensional vector space of dimension n, then $C_{maxvol}(D)$ is a $n \times n$ sub-matrix of D such that $|detC|$ is maximized.
- If D is a matrix, then the rank of D is denoted by $rank(D)$.
- If D is a matrix made up with $m \geq n$ column vectors of dimension n that generate a n-dimensional vector space, then $O_{Schmitt}(D)$ is a $n \times n$ matrix made up with n column vectors obtained from the Schmitt orthonormalization of $(\overrightarrow{d_1}, ..., \overrightarrow{d_m})$.
- If C is a matrix, then $up_n(C)$ is the sub-matrix of C made of the n upper rows of C.

Then our solution to the optimization problem (1) is given by the following algorithm:!la

Procedure `Finds.H.that.minimizes.expression.1`$(Vcalc, L)$

$O = O_{Schmitt}(L, I_n)$
$r = rank(D)$
$C = up_r(O^T L)$
$P = C_{maxvol}(C)\mathcal{H}_r$
$P = \frac{V_{calc}}{|detP|^{\frac{1}{r}}} P$
$P = \begin{pmatrix} P & 0 \\ 0 & I_{n-r} \end{pmatrix}$
$H = OP$
Return(H)

End

4.3 An heuristic to find a subset of vectors of maximum volume.

Given a set of m vectors generating a n-dimensional vector space, finding a subset of n of them whose volume is maximum can be done by comparing the volume of every subset of n vectors. There are $\begin{pmatrix} m \\ n \end{pmatrix} = \frac{m!}{n!(m-n)!}$ such subsets, so this method is unacceptable if m is large. Hence we use a greedy algorithm: we begin with the largest vector. Then, iteratively, we add a vector to the constructed set, such that the corresponding volume in the generated subspace is maximized. When the constructed set contains n free vectors, we try to exchange one of those vectors by another one, if it increases the volume. We continue until no more exchange can increase the volume.
The volume of p vectors of dimension n when $p < n$ can be easily calculated using the Gram matrix: if D is a matrix made up with column vectors $(\overrightarrow{d_1}, ..., \overrightarrow{d_p})$ then $Gram(\overrightarrow{d_1}, ..., \overrightarrow{d_p}) = D^T D$. Then the volume of the polytope generated by $(\overrightarrow{d_1}, ..., \overrightarrow{d_m})$ is $\sqrt{det(D^T D)}$.

The algorithm is given in figure 6. In this procedure, we denote by B_i the ith column of a matrix B, and by $[B, C]$ the horizontal concatenation of two matrices B and C.

5 Examples.

To compare our approach with that of Agarwal, Kranz and Natarajan [1], we use two examples from [1]. Beforehand, we point out that our algorithm does return the optimal solution for Example 2 of Section 3. To evaluate the quality of our expression of cumulative footprint, we compare the values of the following expressions on each example:

- $\sum_{i=1}^{m} \sum_{j=1}^{n} |det(G_i H)_{j \to \overrightarrow{a}}|$ which is the approximation of V_{com} given by [1]
- $V_{calc} . \sum_{k=1}^{n} \max_{\sigma} \left| \overrightarrow{e_k}^T . \overrightarrow{l_\sigma} \right|$, which is our approximation of V_{com}.
- The exact value of V_{com}.

5.1 First example

This example is "Example 7" of [1]:

```
Doall (i=1:N, j=1:N, k=1:N)
A[i,j,k]=B[i-1,j,k+1]+B[i,j+1,k]+B[i+1,j-2,k-3]
EndDoall
```

With our notations, there is a single access matrix $G_1 = I_2$, $a_{1,1} = (-1, 0, 1)^T$, $a_{1,2} = (0, 1, 0)^T$ and $a_{1,3} = (1, -2, -3)^T$. As $G_1 = I_2$, $b_{1,j} = a_{1,j}$. Consequently, for a given value of V_{calc}, the expression to be minimized is

$$V_{calc} \left(1 + \sum_{k=1}^{3} \max \left(|\overrightarrow{e_k} . (-1, -1, 1)^T|, |\overrightarrow{e_k} . (-2, 2, 4)^T|, |\overrightarrow{e_k} . (-1, 3, 3)^T| \right) \right)$$

over all matrices E such that $|detE| = \frac{1}{V_{calc}}$. Our algorithm leads to the solution

$$H = E^{-1} = \sqrt[3]{V_{calc}} \begin{pmatrix} -0.7331 & -0.3656 & -0.8018 \\ 0.7331 & 1.0967 & 0.2673 \\ 1.4622 & 1.0967 & -0.5345 \end{pmatrix}$$

```
Procedure C_maxvol(U)
    B ← 0
    For i=1:n
        B ← [B, U_1]
        j = 1
        V_max ← Volume(B)
        h ← m − i + 1
        For k=2:h
            B_i ← U_k
            V ← Volume(B)
            If V > V_max
                V_max ← V
                j ← k

            EndIf
        EndFor
        B_i ← U_j
        U_j ↔ U_h
        h ← h − 1
    EndFor
    may.increase ← True
    While may.increase do
        E ← B^{-1}
        V_max = 1
        may.increase ← False
        For k=1:h,i=1:n
            V = |E_i^T.U_k|
            If V > V_max
                V_max ← V
                j ← k
                l ← i
                may.increase ← True
            EndIf
        EndFor
        If may.increase
            B_l ← U_j
            U_j ↔ U_{m−l+1}
        EndIf
    EndWhile
    Return(B)
End
```

Figure 6: An heuristic to find a subset of vectors of maximum volume.

In that case, if we take $V_{calc} = 1000$ (say), we obtain

$$\begin{aligned} V_{com} &= V_{calc}.\sum_{k=1}^{3} \max\left(|\overrightarrow{e_k}.(-1,-1,1)^T|, |\overrightarrow{e_k}.(-2,2,4)^T|, |\overrightarrow{e_k}.(-1,3,3)^T|\right) \\ &= 173 \end{aligned}$$

The solution given in [1] is $H = \sqrt[3]{\frac{V_{calc}}{24}} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix}$.

	Agarwal et al. approx.	Our approx.	Exact value
Our tiling	2147	173	173
Agarwal et al. tiling	865	865	756

Table 1: Comparing results for Example 7 of [1].

In Table 1, both approximations of the communication volume of the solution of [1] are equal, because [1] always returns a diagonal matrix. The large difference in Table 1 between our algorithm and that of [1] is due to the fact that, in this example, the vector space generated by the vectors $(b_{i,j})_{i,j}$ is a two-dimensional sub-space of $\mathbb{R}^3$. Our algorithm takes that information into account by first solving the problem in this sub-space and then generalizing the solution to the entire vector space $\mathbb{R}^3$.

5.2 Second example

This example is "Example 8" in [1]:

```
Doall (i=1:N, j=1:N, k=1:N)
A[i,j,k]=B[i-2,j]+B[i,j-1]+C[i+j-1,j]+C[i+j+1,j+3]
EndDoall
```

With our notations, there are two access matrices $G_1 = I_2$ and $G_2 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. We have $a_{1,1} = (-2,0)^T$, $a_{1,2} = (0,-1)^T$, $a_{2,1} = (-1,0)^T$ and $a_{2,2} = (1,3)^T$. Hence, $b_{1,j} = a_{1,j}$, $b_{2,1} = (-1,0)^T$ and $b_{2,2} = (-2,3)^T$. Next, $c_{1,1} = (-2,1)^T$, $c_{2,1} = (1,-3)^T$ and $c_{2,2} = (-1,3)^T$. Then $l_1 = (-1,-2)$ and $l_2 = (-3,4)$. The expression to be minimized is

$$V_{calc}\left(1 + \sum_{k=1}^{2} \max\left(|\overrightarrow{e_k}.(-1,-2)^T|, |\overrightarrow{e_k}.(-3,4)^T|\right)\right)$$

To that problem, our algorithm gives the solution:

$$H = \sqrt[2]{V_{calc}}.\begin{pmatrix} -0.9487 & -0.3162 \\ 1.2649 & -0.6325 \end{pmatrix}$$

In that case, if we take $V_{calc} = 1000$, then $V_{com} = 195$. The solution given by Agarwal et al. is

$$H = \sqrt[2]{\frac{V_{calc}}{12}}.\begin{pmatrix} 3 & 0 \\ 0 & 4 \end{pmatrix}$$

which leads to $V_{com} = 214$ for the same value of $V_{calc} = 1000$.

	Agarwal et al. approx.	Our approx.	Exact value
Our tiling	240	200	195
Agarwal et al. tiling	219	219	214

Table 2: Comparing results for Example 8 of [1].

6 Conclusion

In this paper, we have improved the results of Agarwal, Kranz and Natarajan [1] on loop partitioning. We have refined their estimation of the cumulative footprint, and we have proposed a heuristic to solve the optimization problem without drastically reducing the search space. This heuristic is inspired from recent result in the context of tiling.

Several ameliorations can be made to our heuristic, and we need further experimental results (in addition to the examples dealt with in this paper) to fully assess the usefulness of our approach. Still, we believe our new approach to be much more powerful and efficient than previously published strategies.

7 Appendix

In this section, we present the **Matlab** programs that we wrote to check the validity of our algorithm and to compute our solution for the same exemples as of [1].

All notations used here are the same as those used in Section 3. For a better understanding, the reader may want to compare the programs of Section 7.3 with the corresponding examples guven in Section 5.

7.1 Auxiliary routines

inverse_set.m

```
function [B] = inverse_set(G,A)
% function [B] = inverse_set(G,A)
% G : Set of non-singular matrix.
% A : Set of associated offsets.

p=size(A,1);
B=A;
for i=1:p
   B{i}=inv(G{i})*A{i};
end
```

differences.m

```
function [D] = differences(N)
% function [D] = differences(N)
% N : Offsets.
% D : all possible differences of offsets.

n=size(N,2);
D=[];
for i=1:n-1,
  for j=i+1:n,
    D=[D,N(:,i)-N(:,j)];
  end
end
```

differences_set.m

```
function [C] = differences_set(B)
% function [C] = differences_set(B)
% B : set of offsets.
% C : set of the "differences" for each offsets B{i}.

p=size(B,1);
C={differences(B{1})};
for i=2:p
   m=size(B{i},2);
   C={C{1:size(C,1)},differences(B{i})}';
   C{i}=[C{i},-C{i}];
end
```

sums.m

```
function [L] = sums(C)
% function [L] = sums(C)
% C : set of offsets.
% L : offsets made from all possible sums.

p=size(C,1);
if p==1
  L=C{1};
else
  K=sums({C{2:p}}');
  m=size(C{1},2);
  l=size(K,2);
  L=[];
  for j=1:m
     for k=1:l
         L=[L,C{1}(:,j)+K(:,k)];
     end
  end
end
```

7.2 The algorithm itself

gram.m

```
function [G] = gram(U)
% function [G] = gram(U)

G=U'*U;
```

volume.m

```
function [V] = volume(U)
% function [V] = volume(U)

V=sqrt(det(gram(U)));
```

hadamardPY.m

The general algorithm for $n > 3$ is not implemented here. The reader can find more information in [3].

```
function [H] = hadamardPY(n)
% function [H] = hadamardPY(n)

if n==1
  H=eye(1);
elseif n==2
  H=eye(2);
elseif n==3
  H=[[1,0,1];[1,1,0];[0,1,1]];
end;
```

maxvolume.m

```
function [B] = maxvolume1(U)
% function [B] = maxvolume1(U)
% U : offsets.
% B : offsets bases that maximizes the corresponding volume.

m=size(U,1);
n=size(U,2);
h=n;

B=[];
for i=1:m,
  B=[[B],[U(:,1)]];
  j=1;
  vmax=volume(B);
  h=n-i+1;
  for k=2:h,
    B(:,i)=U(:,k);
    v=volume(B);
    if v>vmax
      vmax=v;
      j=k;
    end
  end
  B(:,i)=U(:,j);
  U(:,j)=U(:,h);
  U(:,h)=B(:,i);
  h=h-1;
end
mayincrease=1;
while mayincrease
  E=inv(B)';
  vmax=1;
  mayincrease=0;
  for k=1:h
     for i=1:m
        V=abs(dot(E(:,i),U(:,k)));
        if V>vmax
           vmax=V;
           j=k;
           l=i;
           mayincrease=1;
        end
     end
  end
  if mayincrease
     B(:,l)=U(:,j);
     U(:,j)=U(:,n-l+1);
     U(:,n-l+1)=B(:,l);
  end
end
```

homothetize.m

```
function [B] = homothetize(A,vol)
% function [B] = homothetize(A,vol)
% A must be non-singular!
% B=u.A such that det(B)=vol

B=(vol/abs(det(A)))^(1/size(A,1))*A;
```

solution.m

```
function [H] = solution(D,vol)
% function [H] = solution(D,vol)
% D : offsets.
% H : matrix of determinant vol  that minimizes function
% eval1(H,{D})

O=orth([D,eye(size(D,1))]);
C=O'*D;
C=C(1:rank(D),:);
H=eye(size(D,1));
P=homothetize(maxvolume(C)*inv(hadamardPY(size(C,1))),vol);
H(1:rank(D),1:rank(D))=P;
H=O*H;
```

7.3 Implementation of the examples

ours7.m

```
function [sol]=ours7(vol)
% function [sol]=ours7(vol)
% Our solution to the problem of example 7 for a given volume
% of tile equal to vol
%
% B=[[-1;0;1],[0;1;0],[1;-2;-3]];

B={[[-1;0;1],[0;1;0],[1;-2;-3]]};
C=differences_set(B);
L=sums(C);
sol=solution(L,vol);
```

theirs7.m

```
function [sol]= theirs7(vol)
% function [sol]= theirs7(vol)
% Agarwal et al.'s solution to the problem of example 7
% for a given volume of tile equal to vol.

I=[[2,0,0];[0,3,0];[0,0,4]];
sol=homothetize((1/det(I))^(1/3)*I,vol);
```

ours8.m

```
function [H] = ours8(vol)
% function [H] = ours8(vol)
% Our solution to the problem of example 8 for a given volume
% of tile equal to vol
%
% A={[[-2;0],[0;-1]];[[-1;0],[1;3]]};
% G={eye(2),[[1,1];[0,1]]};
% B=inverse_set(G,A);

A={[[-2;0],[0;-1]];[[-1;0],[1;3]]};
G={eye(2),[[1,1];[0,1]]};
B=inverse_set(G,A);
C=differences_set(B);
L=sums(C);
H=solution(L,vol);
```

theirs8.m

```
function [sol]= theirs8(vol)
% function [sol]= theirs8(vol)
% Agarwal et al.'s solution to the problem of example 8
% for a given volume of tile equal to vol.

I=[[3,0];[0,4]];
sol=homothetize((1/det(I))^(1/2)*I,vol);
```

7.4 Evaluation of the solutions

eval1.m

Following is the implementation of our approximation of V (see the end of Section 3 and the beginning of Section 5).

```
function [vol] = eval1(H,B)
% function [vol] = eval1(H,B)
% H : tile.
% B : Set of references offsets.

m=size(B,1);
E=inv(H);
vol=0;
for i=1:m
  vol=vol+1+sum(max(abs(E*differences(B{i})),[],2));
end
vol=vol*abs(det(H));
```

eval3.m

Following is the implementation of the approximation given in [1] (see the beginning of Section 5).

```
function [vol] = eval3(H,G,A)
% function [vol] = eval3(H,G,A)
% H : tile.
% G,A : set of references,offsets.
% vol : approximated volume  ;  Agarwal's expression.

p=size(A,1);
vol=0;
for i=1:p
  D=G{i}*H;
  E=inv(D);
  a=max(abs(differences(A{i})),[],2);
  vol=vol+1/abs(det(G{i}))*abs(det(D))*(1+sum(abs(E*a)));
end
```

7.5 Computing the exact value of the volumes

In this section, we explain how to computes the exact volume of a union of several parallelepipeds. Here are a few remarks that will lead to the algorithm:

- Let $\{\epsilon_i.T_i\}_{1\leq i\leq m}$ be a set of non intersecting signed tiles ($\epsilon_i \in \{-1,1\}$), and T be a new tile. Then $\bigoplus_{i=1}^{m}(\epsilon_i.T_i) \cup T = \bigoplus_{i=1}^{m}(\epsilon_i.T_i) + T - \bigoplus_{i=1}^{m}(\epsilon_i.T_i \cap T)$.
- Consider a set of tiles $\{(D, a_i)\}_i$ where D is non-singular. Then the cumulative footprint is $V = |det(G)|\,.V'$ where V' is the cumulative footprint for the references $\{(I, D^{-1}a_i)\}_i$.
- Since a rectangular tile is represented by a couple $\left(\begin{pmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & d_n \end{pmatrix}, \overrightarrow{a}\right)$, it can be represented by a couple of vectors $((d_1, d_2, ..., d_n)^T, \overrightarrow{a})$.

Hence, to compute the exact value of V, we first need to compute the intersection of two tiles. This is done by the following program:

intersection.m

```
function [I] = intersection(A,B)
% function [I] = intersection(A,B)
% A,B : Tile ie {Diagonal,Offset}
% I : Tile.

n=size(A{1},1);
O=max(A{2},B{2});
G=min(A{1}+A{2},B{1}+B{2});
I={max(zeros(n,1),G-O),O};
```

eval2.m
Finally, the following algorithm gives the exact value of V.

```
function [vol] = eval2(H,B)
% function [vol] = eval2(H,B)
% H : tile.
% B : Set of references offsets.

m=size(B,1);
n=size(H,1);
E=inv(H);
vol=0;

for i=1:m
  F=E*B{i};

  D=[]; O=[]; s=[];
  p=size(B{i},2);
  for j=1:p
    d=ones(n,1);   o=F(:,j);
    l=size(D,2);
    D=[D,d];  O=[O,o];  s=[s,1];
    for k=1:l
       I=intersection({D(:,k),O(:,k)},{d,o});
       if prod(I{1},1)~=0
          D=[D,I{1}]; O=[O,I{2}]; s=[s,-s(k)];
       end
    end
  end

  vol=vol+prod(D,1)*s';
end

vol=vol*abs(det(H));
```

References

[1] A. Agarwal, D.A. Kranz, and V. Natarajan. Automatic partitioning of parallel loops and data arrays for distributed shared-memory multiprocessors. *IEEE Trans. Parallel Distributed Systems*, 6(9):943–962, 1995.

[2] Pierre Boulet, Alain Darte, Tanguy Risset, and Yves Robert. (pen)-ultimate tiling? *Integration, the VLSI Journal*, 17:33–51, 1994.

[3] J. Brenner and L. Cummings. The Hadamard maximum determinant problem. *Amer. Math. Monthly*, 79:626–630, 1972.

[4] Pierre-Yves Calland and Tanguy Risset. Precise tiling for uniform loop nests. In P. Cappello et al., editors, *Application Specific Array Processors ASAP 95*, pages 330–337. IEEE Computer Society Press, 1995.

[5] P.Y. Calland, J. Dongarra, and Y. Robert. Tiling with limited resources. In L. Thiele, J. Fortes, K. Vissers, V. Taylor, T. Noll, and J. Teich, editors, *Application Specific Systems, Achitectures, and Processors, ASAP'97*, pages 229–238. IEEE Computer Society Press, 1997. Extended version available on the WEB at http://www.ens-lyon.fr/~yrobert.

[6] Y-S. Chen, S-D. Wang, and C-M. Wang. Tiling nested loops into maximal rectangular blocks. *Journal of Parallel and Distributed Computing*, 35(2):123–32, 1996.

[7] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: A portable linear algebra library for distributed memory computers - design issues and performance. *Computer Physics Communications*, 97:1–15, 1996. (also LAPACK Working Note #95).

[8] Alain Darte, Georges-André Silber, and Frédéric Vivien. Combining retiming and scheduling techniques for loop parallelization and loop tiling. *Parallel Processing Letters*, 7(4):379–392, 1997.

[9] J. J. Dongarra and D. W. Walker. Software libraries for linear algebra computations on high performance computers. *SIAM Review*, 37(2):151–180, 1995.

[10] K. Högstedt, L. Carter, and J. Ferrante. Determining the idle time of a tiling. In *Principles of Programming Languages*, pages 160–173. ACM Press, 1997. Extended version available as Technical Report UCSD-CS96-489, and on the WEB at http://www.cse.ucsd.edu/~carter.

[11] François Irigoin and Rémy Triolet. Supernode partitioning. In *Proc. 15th Annual ACM Symp. Principles of Programming Languages*, pages 319–329, San Diego, CA, January 1988.

[12] Amy W. Lim and Monica S. Lam. Maximizing parallelism and minimizing synchronization with affine transforms. In *Proceedings of the 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 201–214. ACM Press, January 1997.

[13] Naraig Manjikian and Tarek S. Abdelrahman. Scheduling of wavefront parallelism on scalable shared memory multiprocessor. In *Proceedings of the International Conference on Parallel Processing ICPP 96*. CRC Press, 1996.

[14] J. Ramanujam and P. Sadayappan. Tiling multidimensional iteration spaces for multicomputers. *Journal of Parallel and Distributed Computing*, 16(2):108–120, 1992.

[15] Robert Schreiber and Jack J. Dongarra. Automatic blocking of nested loops. Technical Report 90-38, The University of Tennessee, Knoxville, TN, August 1990.

[16] Michael E. Wolf and Monica S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. Parallel Distributed Systems*, 2(4):452–471, October 1991.