Tiling with limited resources*

Pierre-Yves Calland[1], Jack Dongarra[2,3] and Yves Robert[2]

[1] LIP, Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France
[2] Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, USA
[3] Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
e-mail: pycallan@lip.ens-lyon.fr
e-mail: [dongarra, yrobert]@cs.utk.edu

February 1997

Abstract

In the framework of perfect loop nests with uniform dependences, tiling has been extensively studied as a source-to-source program transformation. Little work has been devoted to the mapping and scheduling of the tiles on to physical processors. We present several new results in the context of limited computational resources, and assuming communication-computation overlap. In particular, under some reasonable assumptions, we derive the optimal mapping and scheduling of tiles to physical processors.

Cover Page Information

Corresponding author: Yves Robert
Submitted as: Regular paper
Conference area: Design methodologies, Compilers

*This work was supported in part by the National Science Foundation Grant No. ASC-9005933; by the Defense Advanced Research Projects Agency under contract DAAH04-95-1-0077, administered by the Army Research Office; by the Office of Scientific Computing, U.S. Department of Energy, under Contract DE-AC05-84OR21400; by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615; by the CNRS–ENS Lyon–INRIA project *ReMaP*; and by the Eureka Project *EuroTOPS*. Yves Robert is on leave from Ecole Normale Supérieure de Lyon and is partly supported by DRET/DGA under contract ERE 96-1104/A000/DRET/DS/SR. Pierre-Yves Calland is supported by a grant from *Région Rhône-Alpes*.

1 Introduction

Tiling is a widely used technique to increase the granularity of computations and the locality of data references. This technique is restricted to perfect loop nests with uniform dependences, which we define as in Banerjee [2] (see also the examples below). The basic idea is to group elemental computation points into tiles that will be viewed as computational units. The larger the tiles, the more efficient the computations are performed using state-of-the-art processors with pipelined arithmetic units and a multi-level memory hierarchy (this is illustrated by recasting numerical linear algebra algorithms in terms of blocked Level 3 BLAS kernels [11, 6]). Also, another advantage of tiling is the decrease of the communication time (which is proportional to the surface of the tile) relatively to the computation time (which is proportional to the volume of the tile). The price to pay for tiling may be an increased latency (if there are data dependences, say, we need to wait for the first processor to complete the whole execution of the first tile before another processor can start the execution of the second one, and so on), as well as some load-imbalance problems (the larger the tile, the more difficult to distribute computations equally among the processors).

Tiling has been studied by several authors and in different contexts. We provide a short review of the existing literature in Section 2.1. Basically, most of the work amounts to partitioning the computation domain of a uniform loop nest into tiles whose shape and size are optimized according to some criteria. Little attention has been paid to distributing the tiles to physical processors and to computing the final scheduling. For example, if each physical processor is assigned several tiles, what should be the computation ordering of these tiles? An in-depth study has been presented by Ohta et al [14], who have extended results of Hiranandani et al. [12] on fine grain pipelining for `DOACROSS` loops. We survey their work in Section 2.2.

In this paper, we build upon the work of Ohta et al [14]. We reformulate the problem of tiling with limited resources using more realistic assumptions on data dependences and communication-computation overlap than those used in [14], and we provide new results on the "optimal" tile size. We also derive an optimal mapping to assign tiles to physical processors. All these results are presented in Section 3. Finally, we state some conclusions in Section 4.

2 Literature overview

2.1 Tiling as a loop transformation technique

When targeting a data-parallel or SPMD style of programming, classical constraints in the literature to define tiles are the following:

Tiles are bounded For scalability reasons, we want the number of points inside a tile to be bounded by a constant independent of the domain size.

Tiles are identical by translation This constraint is imposed to allow for automatic code generation: a tile must be the image by a translation of any other one unless it crosses the computation domain boundaries.

Tiles are "atomic" Each tile is a unit of computation: all synchronization points are beginnings and ends of tiles. The order on tiles must be compatible with the order on nodes: one must thus avoid that two distinct tiles depend upon each other.

As already said, tiling is restricted to perfectly nested loops with uniform dependences, such as the following simple example:

Example 1

for $i = 0$ to N_1 do
    for $j = 0$ to N_2 do
        $a(i,j) = a(i-3,j) + a(i,j-2)$
        $b(i,j) = a(i-2,j-3) + b(i-2,j-1)$
    enddo
enddo

This loop nest has depth 2. The dependences are uniform (intuitively, dependence vectors are translations), and they can be encapsulated into the dependence matrix

$$D = \begin{pmatrix} 0 & 3 & 2 & 2 \\ 2 & 0 & 3 & 1 \end{pmatrix}.$$

The atomicity constraint can be expressed by the analytical condition $HD \geq 0$, where H is the matrix of vectors normal to the edges of the tile. Irigoin and Triolet [13] were the first to derive the condition $HD \geq 0$. In the context of vector machines with a two-level memory hierarchy, they have introduced tiling as a much more powerful technique than strip-mining or rectangular partitioning [15]. In their paper, tiling consists in aggregating elemental computation points in *tiles* (or supernode) defined as multi-dimensional parallelepipeds so that each tile executes atomically with no intervening synchronization or communication. This approach has very close objectives to those of Schreiber and Dongarra [17] who aim at automatically designing block versions of nested loop kernels. Schreiber and Dongarra [17] have discussed how to determine the size and shape of the tiles so as to optimize the communication-to-computation ratio. Their work has been extended by Ramanujam and Sadayappan [16], and by Boulet et al. [3]. Several other papers have discussed the same framework, including [18, 4, 1, 5][1].

Back to Example 1

In Figure 1, we sketch a valid timing for Example 1. The matrix H is the one derived using the scalable communication-to-computation criteria of Boulet et al. [3]:

$$H = \frac{1}{16} \begin{pmatrix} 0 & \frac{1}{3} \\ \frac{1}{2} & 0 \end{pmatrix}.$$

We check that $HD \geq 0$. Note that the volume of the tile, which represents the number of computations per tile, is given by the determinant of H^{-1}: $V_{comp} = \det(H^{-1}) = 96$. The number of communications is the following: each tile sends

- 24 data items to its right neighbor,
- 34 data items to its lower neighbor,
- and 6 data items to its lower-right neighbor.

Note that the third message (the diagonal communication) may be routed horizontally and then vertically, or the other way round, and even may be combined with any of the first two messages.

[1] This small list is far from being exhaustive.

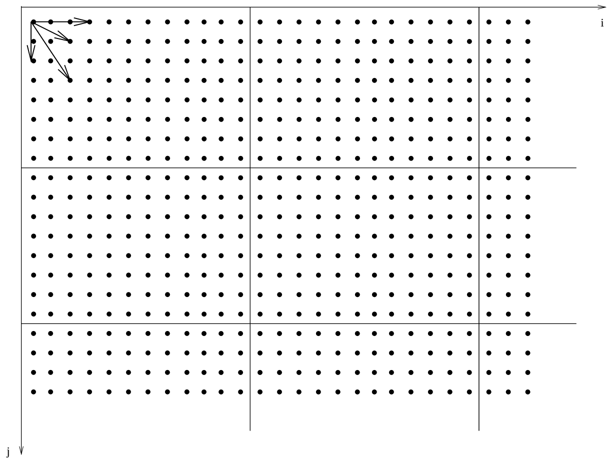

Figure 1: Optimal tiling for a computation volume $V_{comp} = 96$.

It is important to point out that the dependences between tiles are summarized by the vector pair

$$\{\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}\}.$$

In other words, the computation of a tile cannot be started before both its left and upper neighbor tiles have been executed.

As stated above, the atomicity constraint implies that inter-processor communications only take place at the end of the processing of each tile. Note that current architectures do allow for communications and computations to overlap, and it is important to point out that the atomicity constraint does not prevent a given processor from simultaneously communicating boundary data of one tile (whose execution it just completed) and starting the computation of another tile. Also, minimizing communication start-up overheads is a "sine-qua-non" condition towards achieving good performance. Even though sophisticated routing strategies are designed and implemented in hardware, communication start-up costs remain very expensive as opposed to the elemental time for communicating one data item (and even worse for performing a computation). Frequent exchanges of short messages should therefore be replaced by fewer sends and receives of longer messages. To summarize, in the context of distributed memory architectures, tiling is a technique that permits to optimize communications while increasing the granularity of computations.

While well-suited to a data-parallel approach, the atomicity constraint is unnecessarily restrictive when targeting VLSI processor arrays. In this context, tiling is more like a data partitioning approach, and tiles are entities that are mapped to processors. Communications can occur as soon as the data is available, while processing the last computational points of the current tile. Each tile still executes the same program, up to a translation in time. This approach has been chosen by several authors, including [9, 8, 7, 20, 10, 19]. In several of these papers, resource constraints on the number of communication links and the fixed topology of the communication network are introduced, leading to solving complex optimization problems via Integer Linear Programming (ILP) techniques. We do not present further details of these results, since we are targeting for distributed memory machines, and therefore we want to enforce the atomicity constraint.

2.2 Tiling with resource constraints

Ohta et al. [14] aim at determining the best tile size under the following hypotheses:

(H1) There are P available processors interconnected as a ring.

(H2) The computation domain is a two-dimensional rectangle of size $N_1 \times N_2$.

(H3) Tiles are rectangular and their edges are parallel to the axes (see Figure 2). The size of a tile is $n_1 \times n_2$, where n_1 and n_2 are unknowns.

(H4) Dependences between tiles are summarized by the vector pair $\{\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}\}$ (as in Example 1).

(H5) Tiles are assigned to processor using a one-dimensional cyclic distribution: in other words, tile (i, j) is allocated to processor $j \bmod P$.

(H6) The scheduling of the tiles is column-wise: at step 0, processor P_0 executes tile $(0, 0)$ and the computed value is communicated to the adjacent processor P_1 (more precisely, a rectangular slice of width w and height n_2 is sent). At step 1, processors P_1 and P_2 execute tiles $(0, 1)$ and $(1, 0)$ simultaneously. After having executed a whole column of tiles, a processor moves on to its next column.

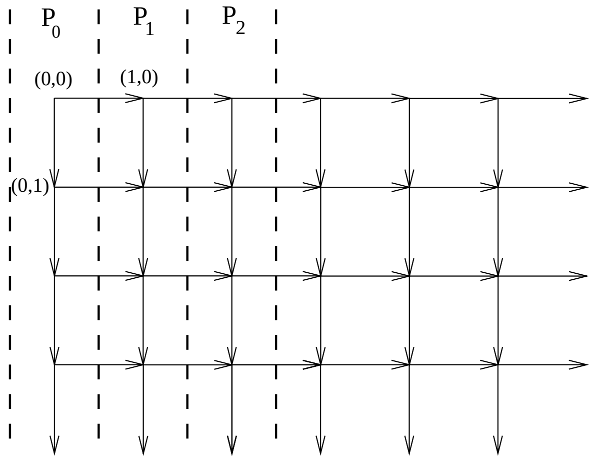

Figure 2: Mapping rectangular tiles onto a ring of processors.

A step is the time required to compute a tile and to communicate data. Ohta et al. [14] use the following expression:

$$T_{tile} = n_1 n_2 t + a + b n_2$$

where t is the elemental computation time, a is a communication start-up and b is the inverse of the communication bandwidth times the width w of the slice being communicated (the communication cost is a linear expression in the message size).

To compute the total execution time, Ohta et al. [14] use the formula $(M_l + M_p)T_{tile}$, where $M_l = P - 1$ is the latency (the step at which the last processor begins to work) and $M_p = \frac{N_1 \times N_2}{P \times n_1 \times n_2}$

is the number of tiles per processor (assumed to be an integer). Using the approximation $M_l = P$, they derive the total execution time T as

$$T = (P + \frac{N_1 N_2}{P n_1 n_2})(n_1 n_2 t + a + b n_2).$$

The execution time is found to be minimal when choosing

$$n_1 = \frac{N_1}{P} \text{ and } n_2 = \sqrt{\frac{N_2 a}{N_1 t}}.$$

2.3 Discussion

The objective of this paper is to discuss the hypotheses (H1) to (H6) of Ohta et al., and to reformulate their results using a more accurate modeling of current architectures. Indeed, their study is conducted while assuming that processors cannot simultaneously communicate bordering data items of the last tile and perform computations for the next tile. However, overlapping computations and communications is a facility provided by all distributed memory computers, so we relax this restriction. This simple modification has a tremendous effect on the determination of the best tile size.

3 Allowing for communication-computation overlap

3.1 On the model

Hypotheses (H2), (H3) and (H4) may appear very restricting. However, we point out the following justifications:

Tile shape We assume that the tiles are rectangular. This is to be understood as the outcome of previous program transformations. The first step in tiling amounts to determining the best size and size of the tiles, assuming an infinite grid of virtual processors. Because this step will lead to tiles whose edges are parallel to extremal dependence vectors in the convex hull of the dependence cone, we can perform a unimodular transformation and rewrite the original loop nest along the edge axes. The resulting domain may not be a rectangular, but we can approximate it using the smallest bounding box.

Dependence vectors We assume that dependences are summarized by the vector pair $\mathcal{V} = \{\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}\}$. Note that these are dependences between tiles, not between elementary computations. In Example 1, we had 5 dependence vectors in the original loop nest, but we ended up with $\mathcal{V}$ after tiling. This is a very general situation if the tiles are large enough. For instance, having a dependence vector $(0, a)$ with $a \geq 2$ between tiles, *instead of* having vector $(0, 1)$, would mean unusually long dependences in the loop nest (in Example 1, $a(i, j)$ would depend upon $a(i, j - 8)$ but not on $a(i, j - x)$ for $x \leq 7$). Note that having $(0, a)$ *in addition to* $(0, 1)$ as a dependence vector between tiles is simply redundant.

On the other hand, hypotheses (H5) and (H6) are unnecessarily restrictive, because the mapping and scheduling of the tiles should be an output decision of the procedure of tiling with limited resources, rather than being given a priori. We overcome this restriction in Section 3.5.

3.2 Determining the total execution time

Proposition 1 *Under the hypotheses (H1) to (H6) of Section 2.2, and allowing for communication-computation overlap, the total computation time T is (assuming all fractions to be integer):*

$$T = \begin{cases} T_1 = (P-1)(n_1 n_2 t + a + bn_2) + \frac{N_1 N_2}{P} t & \text{if } N_2 n_1 t \geq P(n_1 n_2 t + a + bn_2) \\ T_2 = (\frac{N_1}{n_1} - 1)(n_1 n_2 t + a + bn_2) + N_2 n_1 t & \text{otherwise} \end{cases} \quad (1)$$

Proof According to hypothesis (H4), the computation goes column-wise. When a processor has completed the execution of a whole column of tiles, it starts the next column that has been assigned to it. The time to process a whole column of tiles is the number of tiles in the column, namely $\frac{N_2}{n_2}$, times the time to compute a tile, namely $T_{comp} = n_1 n_2 t$. We obtain the value $N_2 n_1 t$ for processing a whole tile column.

Now, according to hypothesis (H5), tile columns are distributed cyclically to processors. If a processor starts the execution of the first tile in a given column at time-step t, its right neighbor cannot start the execution of the first tile in the next column before time-step $t + T_{tile}$, where $T_{tile} = T_{comp} + T_{comm} = n_1 n_2 t + (a + bn_2)$ (this is due to the dependence vector $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$). Note that T_{tile} is the same as in Section 2.2, but we pay a communication cost only when the processors owning the tiles are not the same. Two cases can occur:

P_0	P_1	P_2	P_0
0	2	4	8
1	3	5	9
2	4	6	10
3	5	7	11
4	6	8	12
5	7	9	13
6	8	10	14
7	9	11	15

Figure 3: Scheduling tiles with $T_{comp} = 1$, $T_{comm} = 1$ and $P = 3$.

- Either there are enough tiles in each column so that when a processor has completed the execution of a whole tile column, it does not have to wait for its next tile column to be ready. This will happen if $\frac{N_2}{n_2} T_{comp} = N_2 n_1 t$ is greater than or equal to the delay imposed by horizontal constraints, i.e. if

$$\frac{N_2}{n_2} T_{comp} \geq P\ T_{tile}.$$

  If this condition holds, all processors remain active throughout the entire computation, once they have started execution. Since the last processor starts at time $(P-1)T_{tile}$ and has $\frac{N_1 N_2}{P n_1 n_2}$ tiles to execute (each in time $T_{comp} = n_1 n_2 t$), we obtain T_1, the first expression in Equation (1). See Figure 3 where $T_{comp} = T_{comm} = 1$, and $P = 3$. There are $\frac{N_2}{n_2} = 8$ tiles per column, and $PT_{tile} = 6$, hence the condition is satisfied.

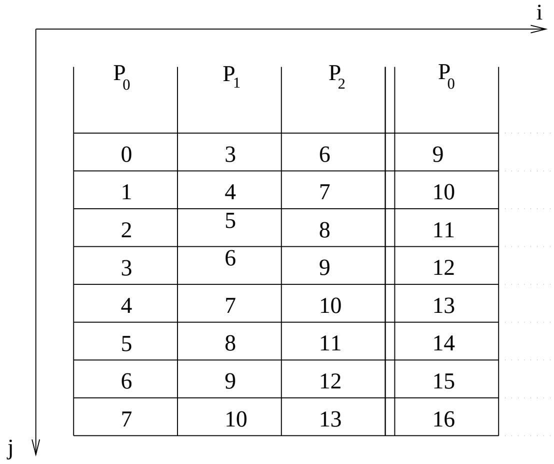

Figure 4: Scheduling tiles with $T_{comp} = 1$, $T_{comm} = 2$ and $P = 3$.

- Or each processor has to wait upon finishing a tile column until the next one is ready. This translates into the condition $\frac{N_2}{n_2} T_{comp} \leq P T_{tile}$. In that case, the total computation time is equal to the time at which the last processor starts the execution of the first tile in the last column, namely $(\frac{N_1}{n_1} - 1) T_{tile}$ plus the time needed to process this column. We obtain the expression $(\frac{N_1}{n_1} - 1) T_{tile} + \frac{N_2}{n_2} T_{comp}$, as stated in the second formula of Equation (1). See Figure 4 where $T_{comp} = 1$, $T_{comm} = 2$, and $P = 3$. There are $\frac{N_2}{n_2} = 8$ tiles per column, and $P T_{tile} = 9$, hence $\frac{N_2}{n_2} T_{comp} \leq P T_{tile}$. Processors remain idle at the end of each tile column, waiting for their next column to be ready. ■

3.3 Optimizing the tile size

Equation 1 in Proposition 1 states that all processors remain active once started if

$$N_2 n_1 t \geq P(n_1 n_2 t + a + b n_2).$$

We rewrite this equation by introducing the following function f:

$$n_2 \geq f(n_1) = \frac{N_2 n_1 t - Pa}{P(n_1 t + b)} \tag{2}$$

The values of n_1 and n_2 that minimize the total execution time are given by the following theorem:

Theorem 1 *Under the hypotheses (H1) to (H6) of Section 2.2, and allowing for communication-computation overlap, the total execution time is minimum for*

- $n_1 = \sqrt{\frac{N_1(a+b)}{(N_2-1)t}}$ *and* $n_2 = 1$ *if* $f(\frac{N_1}{P}) \leq 1$
- $n_1 = \frac{P(a+b)}{t(N_2-P)}$ *and* $n_2 = 1$ *otherwise.*

Proof We break down the problem into two subcases depending on the values taken by the function f, whose argument n_1 ranges from 1 to $\frac{N_1}{P}$;

- $\forall n_1,\ f(n_1) \leq 1$. Since f is a nondecreasing function of n_1, this condition is equivalent to $f(\frac{N_1}{P}) \leq 1$. In this case, Equation (2) is always satisfied ($n_2 \geq 1$). Then the minimum of T is obtained by minimizing T_2 with $n_2 = 1$, namely

$$T = (\frac{N_1}{n_1} - 1)(n_1 t + a + b) + N_2 n_1 t$$

  This easily leads to $n_1 = \sqrt{\frac{N_1(a+b)}{(N_2-1)t}}$, as stated in the theorem

- $\exists n_1,\ f(n_1) \geq 1$. Since f is a nondecreasing function of n_1, we can safely take n_1^0 be such that $f(n_1^0) = 1$. Note that all values of $n_1 \geq n_1^0$ will lead to admissible values for n_2, because we always have $f(n_1) \leq \frac{N_2}{P}$ by definition of f. Now consider the expression of T for arbitrary n_1 and n_2:

  - if $n_2 \leq f(n_1)$, then $T = T_1$, T is a non-increasing function of both n_1 and n_2 decreases, then the minimum is obtained with $n_2 = 1$ and $n_1 = n_1^0$.
  - if $n_2 \geq f(n_1)$ then $T = T_2$ and is a non-increasing function of n_2. Then the minimum of T is reached if $n_2 = f(n_1)$. In that case $T_2 = T_1$, and again the minimum is reached when $n_2 = 1$ and $n_1 = n_1^0$.

■

This result is disappointing in that we end up with degenerate tiles in most practical situations. For instance if $P \ll N_2$ (which is very likely to happen in practice), $f(1) \geq 1$, and the optimal tile size is $n_1 = n_2 = 1$, not a very coarse-grain tiling indeed! The flaw is that the model is not accurate enough to take the impact of data locality and data reuse into account (which are the main objectives of tiling). A first solution is to model the computation cost of a tile by an affine expression like $T_{comp} = n_1 n_2 t + u$, where u represents some access overhead. It is not difficult to plug this expression into the formula given for the total execution time T, and to derive the optimal tile size. Another solution is to assume a fixed tile size that would be imposed by some a priori considerations (such as the cache size). Again, we can let $n_1 n_2 = S$ in Equation (1), and minimize T for n_1, say.

3.4 Optimizing the number of processors

Another interesting question is the following: since the size and shape of the tiles may be imposed by some a priori considerations (such as the cache size, and the results on the best communication-to-computation ratio that have been reviewed in Section 2.1), we may consider n_1 and n_2 as input parameters and try to determine what is the optimal number of processors that should be used so as to minimize the total execution time. The answer is given by the following proposition:

Theorem 2 *Under the hypotheses (H2) to (H6) of Section 2.2, and allowing for communication-computation overlap, let*

$$P_\alpha == \sqrt{\frac{N_1 N_2 t}{n_1 n_2 t + a + b n_2}} \quad \textit{and} \quad P_\beta = \frac{N_2 n_1 t}{n_1 n_2 t + a + b n_2}$$

The number of processors P that minimizes the total execution time is given by:

- *if* $P_\beta \leq 1$ *or* $P_\alpha \leq 1 \leq P_\beta$, *then* $P = 1$,
- *if* $1 \leq P_\beta \leq P_\alpha$ *then* $P = P_\beta$,
- *if* $1 \leq P_\alpha \leq P_\beta$ *then* $P = P_\alpha$.

Proof The "steady-state" condition $N_2 n_1 t \geq P(n_1 n_2 t + a + b n_2)$ in Equation (1) can be rewritten as

$$P \leq P_\beta.$$

Consider $T_1 = (P-1)(n_1 n_2 t + a + b n_2) + \frac{N_1 N_2}{P} t$ (see Equation (1)). The minimum of T_1 is obtained for $P = P_\alpha$. The expression of T_1 shows that is a non-increasing function of P when $P \leq P_\alpha$, and then a non-decreasing function of P when $P \geq P_\alpha$. Also, note that T_2 does not depend on P (except than through the condition $P \geq P_\beta$). Then the result follows according to a simple case analysis. ■

For large domains, we will have $1 \leq P_\beta \leq P_\alpha$, and it is no surprise that the optimal number of processors is the one required to ensure steady-state execution.

3.5 Optimal mapping and scheduling

Hypotheses (H5) and (H6) are very restrictive in that they impose the mapping of tiles to processors as well as their scheduling. The intuitive motivation for (H5) is that a cyclic distribution of tiles is quite natural to load-balance computations. Once the distribution of tiles to processors is fixed, there are several possible schedulings (any wavefront execution that goes along a left-to-right diagonal is valid). Specifying a column-wise execution may lead to the simplest code generation.

It turns out that (H5) and (H6) provide the best solution among all possible distributions of tiles to processors, which is a very strong result. This result holds true under the assumption that the communication cost for a tile is not larger than its computation cost. Since the communication cost for a tile grows linearly with its size, while the computation costs grows quadratically, this hypothesis will be satisfied if the tile is large enough[2]. This result is formally stated in the theorem below. Beforehand, we need to refine the communication cost as follows:

- $T_{comm\_horiz} = a + b n_2$ is the cost of communicating data from (the processor owning) tile (i,j) to (the processor owning) its right neighbor tile $(i+1, j)$,
- $T_{comm\_vert} = a' + b' n_1$ is the cost of communicating data from (the processor owning) tile (i,j) to (the processor owning) its bottom neighbor tile $(i, j+1)$.

We pay a communication cost only when the two processors that own the neighboring tiles are not the same. So far we never paid any cost for vertical communications, and we always did for horizontal communications, because of hypothesis (H5). We had to refine the communication cost because in this section, we do not make any assumption on the mapping of tiles to processors. Depending upon the values of T_{comm_horiz} and T_{comm_vert}, the best mapping will be column-wise or row-wise:

Theorem 3 *Under the hypotheses (H2) to (H4) of Section 2.2, and allowing for communication-computation overlap, let* n_1 *and* n_2 *be chosen so that*

$$\max\{T_{comm\_horiz}, T_{comm\_vert}\} = \max\{a + b n_2, a' + b' n_1\} \leq T_{comp} = n_1 n_2 t.$$

[2]Of course, we can imagine theoretical situations where the communication cost is so large that a sequential execution would lead to the best result.

1. *If $T_{comm\_horiz} \leq T_{comm\_vert}$, assume that the steady state equation holds: $N_2 n_1 t \geq P(n_1 n_2 t + a + b n_2)$. Then the absolute minimum for the total execution time is*

$$T_1 = (P-1)(T_{comp} + T_{comm\_horiz}) + \frac{N_1 N_2}{P} t$$

and it is achieved by mapping and scheduling tiles according to hypotheses (H5) and (H6),

2. *If $T_{comm\_vert} \leq T_{comm\_horiz}$, assume that the steady state equation holds: $N_1 n_2 t \geq P(n_1 n_2 t + a' + b' n_1)$. Then the absolute minimum for the total execution time is*

$$T_1' = (P-1)(T_{comp} + T_{comm\_vert}) + \frac{N_1 N_2}{P} t$$

and it is achieved by mapping rows *of tiles using a one-dimensional cyclic distribution (tile (i,j) is allocated to processor $i \bmod P$), and by scheduling the tiles row-wise.*

Proof Without loss of generality, assume that $T_{comm\_vert} \leq T_{comm\_horiz}$ (the result is symmetric in the rows and columns), and let $T_{comm} = T_{comm\_vert}$. We begin the proof with the following preliminary result, where σ denotes any valid scheduling of the tiles ($\sigma(I)$ is the time-step at which the execution of I begins):

Lemma 1 *Let $I = (i,j)$ be a tile index, and let $I' = (i+1, j)$ and $I'' = (i, j+1)$ be its successor tiles. We have*

$$\max\{\sigma(I'') - \sigma(I), \sigma(I') - \sigma(I)\} \geq T_{comm} + T_{comp}.$$

Proof Let $proc(I)$ be the processor that executes tile I. We have three cases to consider, depending upon whether $proc(I)$ also executes both successors I' and I'', or exactly one of them, or none of them:

both successors: $proc(I) = proc(I') = proc(I'')$
The same processor executes both successors. They are executed sequentially and the last one being executed cannot begin execution before time-step $\sigma(I) + 2T_{comp}$. As $T_{comm} \leq T_{comp}$ the result is proven.

one successor: $proc(I) = proc(I')$ and $proc(I) \neq proc(I'')$
(respectively $proc(I) = proc(I'')$ and $proc(I) \neq proc(I')$). A communication is needed between I and I'' (respectively I and I'), hence $\sigma(I'') - \sigma(I) \geq T_{comm} + T_{comp}$ (respectively $\sigma(I') - \sigma(I) \geq T_{comm} + T_{comp}$)

no successor: $proc(I) \neq proc(I')$ and $proc(I) \neq proc(I'')$
This case is similar to the previous one. ■

Back to the proof of the theorem, let $T_{//}$ the total execution time using P processors. Let $Idle$ be the cumulated idle time of all processors during execution. Finally, let $T_{seq} = N_1 N_2 t$ be the sequential execution time. Clearly,

$$PT_{//} = Idle + T_{seq}.$$

Hence, to show that $T_{//} \geq T_1 = (P-1)(T_{comp} + T_{comm}) + \frac{T_{seq}}{P}$, we need to show that

$$Idle \geq P(P-1)(T_{comp} + T_{comm}).$$

The structure of the dependence graph does impose that some processors are idle at the beginning of the computation, which will lead to a lower bound for $Idle$. For instance, during the execution of tile $(0,0)$, there are necessarily $P-1$ idle processors. To go on, we recursively define $pivot\_tile(k)$ as follows (see Figure 5):

- $pivot\_tile(0) = (0,0)$, and
- for $k \geq 1$, $pivot\_tile(k)$ is the one of the two successors of $pivot\_tile(k-1)$ which is executed last: if $pivot\_tile(k-1) = I = (i,j)$, let $I' = (i+1,j)$ and $I'' = (i,j+1)$ be the successors of tile I:
  - If $\sigma(I') \geq \sigma(I'')$, then $pivot\_tile(k) = I'$, and we define $S(k)$ as the remaining tiles in column j: $S(k) = \{(i, j+l), l \geq 1\})$,
  - If $\sigma(I'') \geq \sigma(I')$, then $pivot\_tile(k) = I''$, and we define $S(k)$ as the remaining tiles in row i: $S(k) = \{(i+l, j), l \geq 1\}$,

We know from Lemma 1 that for all $k \geq 1$,

$$\sigma(pivot\_tile(k)) - \sigma(pivot\_tile(k-1)) \geq T_{comm} + T_{comp}.$$

We prove by induction that for $1 \leq k \leq P-1$, at least $P-k$ processors are kept idle between the beginning of the execution of $pivot\_tile(k-1)$ and that of $pivot\_tile(k)$. This will lead to be result that

$$Idle \geq ((P-1) + (P-2) + \ldots + 1)(T_{comm} + T_{comp}) = \frac{P(P-1)}{2}(T_{comm} + T_{comp}).$$

This will prove the desired result, because the same amount of idleness, so to speak, will be spent at the end of the computation (by symmetry of the dependence graph). Now, for the induction:

- Let $k = 1$: $pivot\_tile(1)$ is either $(0,1)$ or $(1,0)$. See Figure 5 where $pivot\_tile(1) = (1,0)$ and $S(1) = \{(0, 0+l), l \geq 1\}$. Between the the beginning of the execution of $pivot\_tile(0)$ and that of $pivot\_tile(1)$, the only successors of $pivot\_tile(0)$ that can be executed are in $S(1)$. But all tasks in $S(1)$ must be executed sequentially, hence between the beginning of the execution of $pivot\_tile(0)$ and that of $pivot\_tile(1)$, at least $(P-1)$ processors are kept idle.
- Assume that the hypothesis is true until step k. Between the beginning of the execution of $pivot\_tile(k)$ and that of $pivot\_tile(k+1)$, at most one processor can be active in $S(1)$, at most another one in $S(2)$, ..., and at most one processor in $S(k+1)$, so that at most $k+1$ processors can be active, or equivalently, at least $P-(k+1)$ processors remain idle.

■

4 Conclusion

In this paper, we have studied tiling techniques aimed at adapting the granularity of uniform loop nest algorithms towards execution on distributed-memory machines. We view tiling as a two-step process: the first step amounts to determining the best shape and size of the tiles (assuming an infinite grid of virtual processors), while the second step consists in mapping and scheduling the tiles to physical processors. We have concentrated on the second step, assuming a realistic model

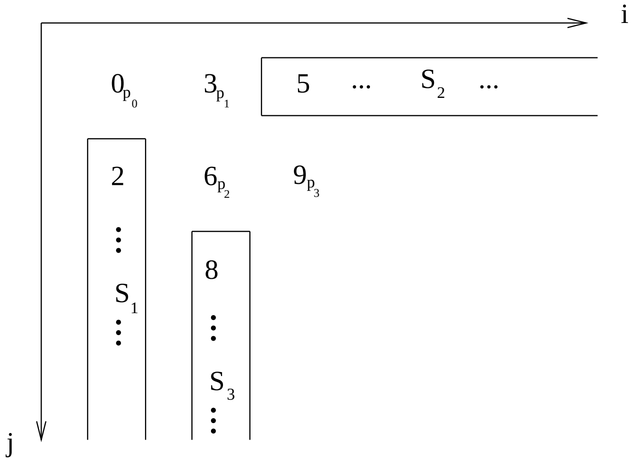

Figure 5: A schedule when $T_{comm} = 1$ and $T_{comp} = 2$. Pivot tiles are labeled, and sets $S(k)$ are framed.

where (independent) communication and computation may overlap. We have obtained several new results, including a strong result on the optimal mapping and scheduling. However, much remains to be done to extend these results to arbitrary dimensions and domain shapes.

More generally, the relationship between tiling, scheduling and mapping is not yet well understood, and the two-step approach may not prove too complicated for practical problems. Yet, such a two-step approach is typical in the field of parallelizing compilers (other examples are general task graph scheduling, software pipelining and loop parallelization algorithms).

Finally, the recent development of heterogeneous computing platforms may well lead to using tiles whose size and shape will depend upon the characteristics of the processors they are assigned to ... a truly challenging problem!

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] Utpal Banerjee. An introduction to a formal theory of dependence analysis. *The Journal of Supercomputing*, 2:133–149, 1988.

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

[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] 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):108–120, 1996.

[6] 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).

[7] W.H. Chou and S.Y. Kung. Scheduling partitioned algorithms with limited communication supports. In Luigi Dadda and Benjamin Wah, editors, *Application Specific Array Processors ASAP 93*, pages 53–64. IEEE Computer Society Press, 1993.

[8] Ed F. Deprettere. Example of combined algorithm development and architecture design. *INTEGRATION, the VLSI journal*, 16:199–220, 1993.

[9] Ed F. Deprettere, Peter Held, and Paul Wielage. Model and methods for regular array design. *International Journal of High Speed Electronics and Systems*, 4(2):133–201, 1993.

[10] Michèle Dion, Tanguy Risset, and Yves Robert. Resource-constrained scheduling of partitioned algorithms on processor arrays. *Integration, the VLSI Journal*, 20:139–159, 1996.

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

[12] Seema Hiranandani, Ken Kennedy, and Chau-Wen Tseng. Evaluating compiler optimizations for Fortran D. *Journal of Parallel and Distributed Computing*, 21(1):27–45, 1992.

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

[14] H. Ohta, Y. Saito, M. Kainaga, and H. Ono. Optimal tile size adjustment in compiling general DOACROSS loop nests. In *1995 International Conference on Supercomputing*, pages 270–279. ACM Press, 1995.

[15] J. K. Peir. *Program partitioning and synchronization on multiprocessor systems*. PhD thesis, University of Illinois at Urbana-Champaign, March 1986. Report UIUC-DCS-R-86-1259.

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

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

[18] S. Sharma, C.-H. Huang, and P. Sadayappan. On data dependence analysis for compiling programs on distributed-memory machines. *ACM Sigplan Notices*, 28(1), January 1993. Extended Abstract.

[19] J. Teich, L. Thiele, and L. Zhang. Scheduling of partitioned regular algorithms on processor arrays with constrained resources. In J. Fortes et al., editor, *Application-Specific Systems, Architectures, and Processors ASAP 96*. IEEE Computer Society Press, 1996.

[20] L. Thiele. Resource constrained scheduling of uniform algorithms. In Luigi Dadda and Benjamin Wah, editors, *Application Specific Array Processors ASAP 93*, pages 29–40. IEEE Computer Society Press, 1993.