Determining the Idle Time of a Tiling: New Results*

Frédéric Desprez[1], Jack Dongarra[2,3], Fabrice Rastello[1], 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: [desprez, frastell]@lip.ens-lyon.fr
e-mail: [dongarra, yrobert]@cs.utk.edu

April 1997

Abstract

In the framework of perfect loop nests with uniform dependencies, tiling has been studied extensively as a source-to-source program transformation. We build upon recent results by Högsted, Carter, and Ferrante [10], who aim at determining the cumulated idle time spent by all processors while executing the partitioned (tiled) computation domain. We propose new, much shorter proofs of all their results and extend these in several important directions. More precisely, we provide an accurate solution for all values of the *rise* parameter that relates the shape of the iteration space to that of the tiles, and for all possible distributions of the tiles to processors. In contrast, the authors in [10] deal only with a limited number of cases and provide upper bounds rather than exact formulas.

Corresponding Author : Yves Robert.

*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-AC05-96OR22464; 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.

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 dependencies, as defined by Banerjee [2]. 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 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 [9, 8]). Another advantage of tiling is the decrease in communication time (which is proportional to the surface of the tile) relative 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 dependencies, for example, 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 researchers and in different contexts [11, 15, 17, 14, 3, 16, 5, 13, 1, 7, 12, 4, 10][1]. Rather than providing a detailed motivation for tiling, we refer the reader to the papers by Calland, Dongarra, and Robert [4] and by Högsted, Carter and Ferrante [10], which provide a review of the existing literature. Most of the work amounts to partitioning the iteration space of a uniform loop nest into tiles whose shape and size are optimized according to some criteria (such as the communication-to-computation ratio). Once the tile shape and size are defined, there remains to distribute the tiles to physical processors and to compute the final scheduling.

In this paper, we build upon the work of Högsted, Carter, and Ferrante [10]. Given a tiled domain, they aim at determining the cumulated idle time spent by all processors. This cumulated idle time heavily depends upon the tile and domain shapes. A new parameter, called the tile *rise*, is introduced in [10] in order to relate the shape of the iteration domain to that of the tiles, and it is shown to have a significant impact on the idle time. Both parallelogram-shaped and trapezoidal-shaped iteration spaces are considered. We summarize the results of [10] in Section 2. Then we propose new, much shorter proofs of these results, and we extend them in several important directions. More precisely, we provide an accurate solution for all values of the *rise* parameter and for all possible distributions of the tiles to processors, while the authors in [10] deal only with a limited number of cases and provide upper bounds rather than exact formulas. These new results are presented in Section 3. In Section 4, we apply our results to the problem of *hierarchical tiling*, that is, when multiple levels of memory and parallelism hierarchy are involved. In Section 5, we state our conclusions and discuss directions for future research.

2 Determining the Idle Tile of a Tiling

In this section, we summarize the results of Högsted, Carter, and Ferrante [10], who make the following hypotheses:

(H1) There are P available processors interconnected as a ring. Processors are numbered from 0 to $P-1$.

(H2) Tiles are parallelograms with vertical left and right edges. The size and shape of the tiles are given, so that we deal only with a partitioned (already tiled) iteration space, as in Figure 1.

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

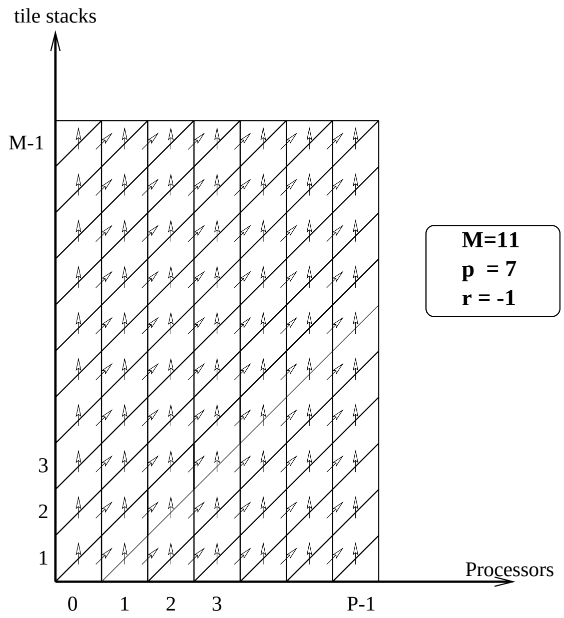

Figure 1: An example of parallelogram-shaped iteration space with parallelogram-shaped tiles. Arrows represent dependences between tiles.

(H3) The iteration space is a two-dimensional parallelogram or trapezoid, with vertical left and right boundaries. The first column (and all columns in case of a parallelogram-shaped iteration space) has M tiles.

(H4) Tiles are assigned to processors using either a one-dimensional full block distribution or a one-dimensional cyclic distribution. In other words,

- for a block distribution, there are P columns in the iteration space, and all the tiles in column j, $0 \leq j \leq P-1$, are assigned to processor j; and
- for a cyclic distribution, there are bP columns in the iteration space, and all the tiles in column j, $0 \leq j \leq bP-1$, are assigned to processor $j \bmod P$.

(H5) The *rise* parameter relates the shape of the iteration space to that of the tiles. It is defined as follows:

- Let the slope (in reference to the horizontal axis) of the top and bottom edges of the tiles be r_{tile}.
- If the iteration domain is a parallelogram, let r_{iter} be the slope of the top and bottom boundaries. In this case, Högsted, Carter, and Ferrante [10] define the *rise* r as

$$r = r_{iter} - r_{tile}.$$

- If the iteration domain if a trapezoid, let r_{iter_top} and r_{iter_bottom} be the slopes of the top and bottom boundaries, respectively. In this case, Högsted, Carter, and Ferrante [10] let $r_t = r_{iter\_top} - r_{tile}$ be the rise at the top of the iteration space and $r_b = r_{iter\_bottom} - r_{tile}$ be the rise at the bottom of the iteration space.

See Figure 2 for an illustration.

(H6) Each tile depends upon both its left neighbor and its bottom neighbor (see Figure 1).

(H7) Because of hypotheses (H4) and (H6), the scheduling of the tiles is by column. Each processor starts executing its first column of tiles as soon as possible. After having executed a whole column of tiles, a processor moves on to its next column. The time needed to process a tile is T_{comp} (with the notations of [10], $T_{comp} = h \times w$, where h and w are the normalized height and width of a tile). The time needed to communicate data from a tile to its right neighbor is $T_{comm} = c \times T_{comp}$. As stated in [10], c may take any positive value (even though we expect $c < 1$ for large tiles, because the communication volume grows linearly with the tile perimeter, while the computation volume is proportional to the tile volume).

Communications can be overlapped with the computations of other (independent) tiles. Moreover, no communication cost is paid between a tile and its top neighbor, because both are assigned to the same processor.

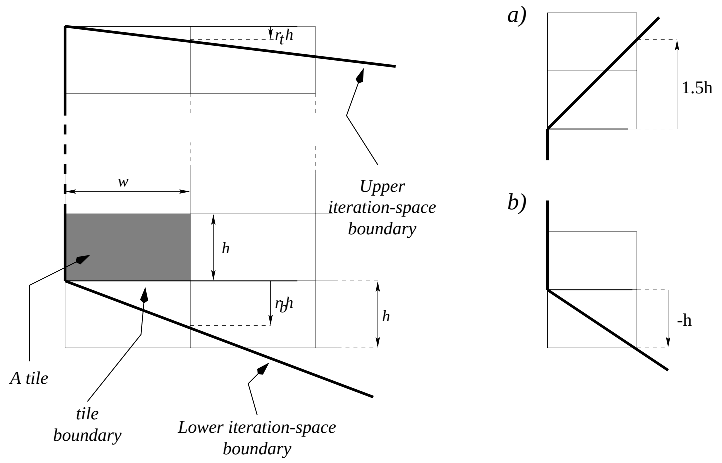

Figure 2: Shape of the iteration space; the rise is positive in (a) and negative in (b).

We summarize in Table 1 the results obtained in [10]. In this table, I_a denotes the cumulated idle time spent by the P processors while executing the tiled iteration space. As pointed out in [10], idle time can occur for two different reasons: (i) a processor may have to wait for data from another processor; or (ii) a processor may have finished all of the tiles assigned to it, and it is waiting for the last processor to terminates execution. In Table 1, condition (C) is a technical condition ($M \geq (1+c+r)P$) that states than no processor is kept idle when ending the processing of one column of tiles assigned to it; in other words, it can move on to its next column without waiting for any data to be communicated.

3 New Results

In this section we propose new proofs and extend the work of [10].

3.1 Task Graph Framework

The key to our approach is the following: rather than laboriously computing the idle time of each processor, and then summing up the results to get the total idle time I_a, we compute the parallel

Parallelogram shaped–Block distribution	
$r \geq -1$	$I_a = P(P-1)(1+r+c)T_{comp}$
$r \leq -2$	$I_a \leq \max(c - \frac{1}{2r}, \frac{-r}{2})PT_{comp}$

Parallelogram shaped–Cyclic distribution	
$r \geq -1$ and condition (C)	$I_a = P(P-1)(1+r+c)T_{comp}$
$r \leq -2$	$I_a \leq \max(c - \frac{1}{2r}, \frac{-r}{2})bPT_{comp}$

Trapezoidal shaped–Block distribution–$r_b < r_t$	
$r_b \geq -1$	$I_a = P(P-1)(1 + \frac{r_t+r_b}{2} + c)T_{comp}$
$r_b \leq -2 < -1 \leq r_t$	$I_a \leq (\max(c - \frac{1-r^2}{2r}, 0) + (P-1)\frac{r_t+r_b}{2})PT_{comp}$
$r_b < r_t \leq -1$, $r_b \leq -2$	$I_a \leq (\max(c - \frac{1-r^2}{2r}, 0) + (P-1)\frac{r_t+r_b}{2} - \frac{r_t}{2})PT_{comp}$

Trapezoidal shaped–Block distribution–$r_t < r_b$	
$-1 \leq r_t \leq r_b$	$I_a = P(P-1)(1 + \frac{r_t+r_b}{2} + c)T_{comp}$
$-(1+c) \leq r_t < -1 \leq r_b$	$I_a \leq ((P-1)(1 + \frac{r_t+r_b}{2} + c) - \frac{r_t}{2})PT_{comp}$
$r_t \leq -(1+c) < -1 \leq r_b$	$I_a \leq ((P-1)\frac{r_b-r_t}{2} - \frac{r_t}{2})PT_{comp}$
$r_t < r_b \leq -2$, $r_b \leq -2$	$I_a \leq (\max(c - \frac{1-r^2}{2r}, 0) - (P-1)\frac{r_t-r_b}{2} - \frac{r_t}{2})PT_{comp}$

Table 1: Summary of the results of Högsted, Carter, and Ferrante [10]

execution time t_P with P processors, and we state that

$$P \times t_P = I_a + T_{seq},$$

where T_{seq} is the sequential time, that is, the sum of all tile weights (see Figure 3).

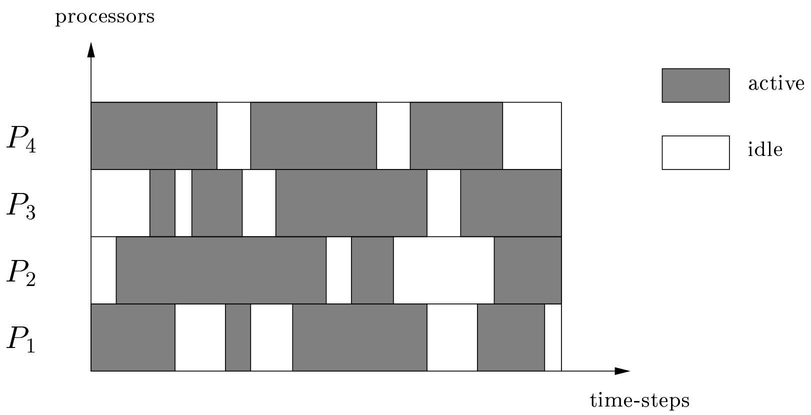

Figure 3: Active and idle processors during execution (illustrating the formula $P \times t_P = I_a + T_{seq}$).

We describe the tiled iteration space as a task graph $G = (V, E)$, where vertices represent the tiles and edges represent dependencies between tiles. A handy view of the graph is obtained by "rotating" the iteration space so that $r_{tile} = 0$. Dependencies between tiles are now summarized by the vector pair

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

See Figure 4, where we have rotated the iteration space of Figure 1.

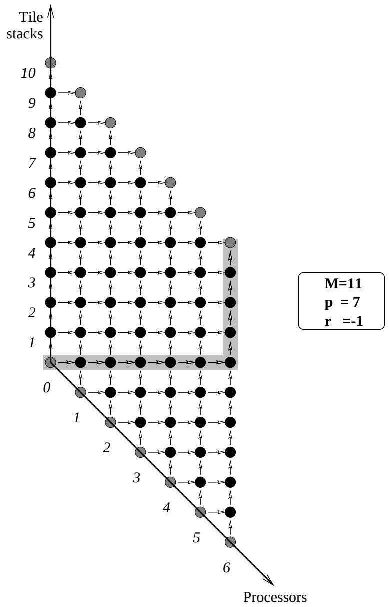

Figure 4: Another view of Figure 1 after rotation.

Computing the parallel execution time t_P is a well-known task graph scheduling problem. Since the allocation of tiles to processors is given, the task amounts to computing the longest path in the dependency graph, where the weight of a path is the sum of the weights of its vertices and edges. All vertices have same weight T_{comp}. Horizontal edges have weight T_{comm} (they imply a communication cost), while vertical edges have zero weight (no communication cost due to the allocation). The problem has complexity $O(|V| + |E|)$ (simply traverse the direct acyclic graph G), but we aim at finding a closed-form formula for t_P, specifically, an analytical expression in the problem parameters M, P, r, and c.

3.2 Preview of Results

A summary of our results is given in Table 2. A few comments are in order:

- In Table 2 we assume that M is sufficiently large (see Sections 3.3 to 3.6 for a more precise statement). This hypothesis was implicit in the results in Table 1 quoted from [10] (see Remark 2).

- We assume that all tiles have the same weight T_{comp}. This assumption is different from [10], where partial tiles are assigned a weight that is proportional to their area. Since partial tiles may only occur at the bottom and at the top of the iteration space, their weight has a little impact on the total execution time. Our motivation for handling partial tiles as regular tiles is code generation: it is simpler to have the same code (with guarded instructions) for all tiles than generating special code for boundary tiles.

- For trapezoidal shaped iteration spaces, the total idle time I_a is not reported in the table. However, it can be computed straightforwardly from the relation $Pt_P = I_a + T_{seq}$, where $T_{seq} = bP[M + \frac{bP-1}{2}(r_t - r_b)]T_{comp}$ (let $b = 1$ for a block distribution).

Parallelogram shaped–Block distribution	
$1+r+c \leq 0$	$t_P = MT_{comp}$, $I_a = 0$
$1+r+c \geq 0$	$t_P = [M + (P-1)(1+r+c)]T_{comp}$ $I_a = P(P-1)(1+r+c)T_{comp}$

Parallelogram shaped–Cyclic distribution	
$1+r+c \leq 0$	$t_P = bMT_{comp}$, $I_a = 0$
$1+r+c \geq 0$, $M \leq (1+r+c)P$	$t_P = [M + (1+r+c)(bP-1)]T_{comp}$ $I_a = P[(bP-1)(1+r+c) - (b-1)M]T_{comp}$
$1+r+c \geq 0$, $M \geq (1+r+c)P$	$t_P = [bM + (1+r+c)(P-1)]T_{comp}$ $I_a = P(P-1)(1+r+c)T_{comp}$

Trapezoidal shaped–Block distribution	
$1+r_b+c \leq 0$, $r_t \leq r_b$	$t_P = MT_{comp}$
$1+r_b+c \leq 0$, $r_t \geq r_b$	$t_P = [M + (P-1)(r_t - r_b)]T_{comp}$
$1+r_b+c \geq 0$, $1+r_t+c \leq 0$	$t_P = MT_{comp}$
$1+r_b+c \geq 0$, $1+r_t+c \geq 0$	$t_P = [M + (P-1)(1+r_t+c)]T_{comp}$

Trapezoidal shaped–Cyclic distribution	
$1+r_b+c \leq 0$, $r_t \leq r_b$	$t_P = [bM + \frac{b(b-1)}{2}P(r_t - r_b)]T_{comp}$
$1+r_b+c \leq 0$, $r_t \geq r_b$	$t_P = [bM + (\frac{b(b-1)}{2}P + b(P-1))(r_t - r_b)]T_{comp}$
$1+r_b+c \geq 0$	see Proposition 4

Table 2: Summary of the results of this paper

3.3 Parallelogram shaped–Block distribution

This is the simplest case, and we work it out in full detail. In the formula below, we use the notation a^+ to denote the positive part of a real number a:

$$a^+ = \begin{cases} a \text{ if } a \geq 0 \\ 0 \text{ if } a \leq 0 \end{cases}$$

Proposition 1 *Assume a parallelogram-shaped iteration space of size $M \times P$ (block distribution) and rise r. If $M \geq (P-1)|r|$. Then*

$$t_P = [M + (P-1)(1+r+c)^+]T_{comp}.$$

Equivalently,

$$I_a = \begin{cases} P(P-1)(1+r+c)T_{comp} \textit{ if } 1+r+c \geq 0 \\ 0 \textit{ otherwise} \end{cases}$$

Proof All processors have the same workload MT_{comp}. Because of the dependencies, processor $P-1$ is always the last one to terminate execution. We discuss separately the case $r \leq 0$ and the case $r \geq 0$.

If $r \leq 0$, processor $q = P-1$ can start processing its first $(P-1).(-r)$ tasks at time-step $t = 0$. Then, at time-step $t = -(P-1)rT_{comp}$, it can continue the processing of its column (i.e.,

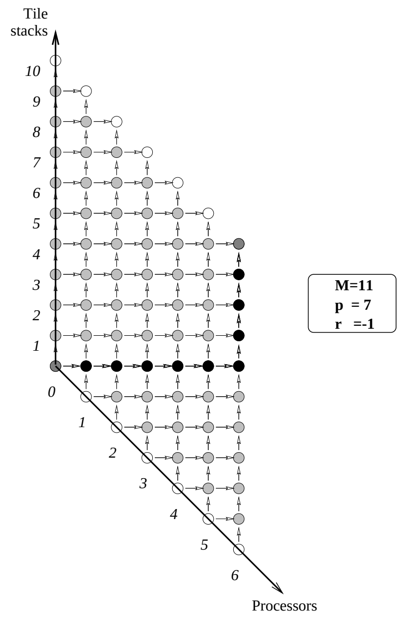

Figure 5: Longest path when $r \leq 0$ (and $M \geq P|r|$)

the remaining $M + (P-1)r$ tiles) *only if* data communicated along the horizontal axis is already available. Otherwise it must wait. To process (and communicate data from) the first $(P-1)$ tasks of the horizontal axis takes $(P-1)(1+c)T_{comp}$. Therefore, the longest path in the dependency graph has length

$$(P-1)\max(-r, 1+c)T_{comp} + (M + (P-1)r)T_{comp}.$$

This longest path is represented in Figure 5.

If $r \geq 0$, processor $q = P-1$ must wait $(P-1)rT_{comp}$ time-steps for processor $q = 0$ to complete its first $(P-1)r$ tasks. Then processor $q = P-1$ must wait another $(P-1)(1+c)T_{comp}$ time-steps for executing tiles and communicating data along the horizontal dependence path that leads to its first task. Only then, at time-step $t = (P-1)(1+r+c)T_{comp}$, can processor $q = P-1$ start the execution of its M tasks, and it will not be further delayed during this processing. The longest path in the dependency graph is represented in Figure 6.

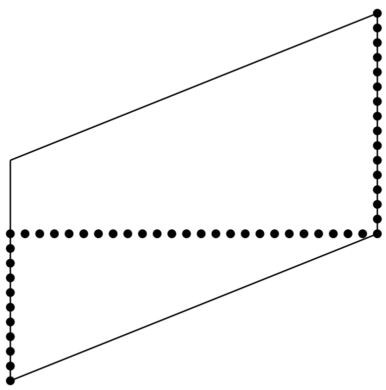

Figure 6: Longest path when $r \geq 0$ (and $M \geq P|r|$)

We summarize both cases with the single formula

$$t_P = [M + (P-1)(1+r+c)^+]T_{comp}.$$

The formula for I_a is derived from the equation $Pt_P = I_a + T_{seq}$, with $T_{seq} = MPT_{comp}$. ■

Remark 1 We see that the results in [10], as reported in Table 1, are inaccurate. A small rise does not prevent from a quadratic idle time; the precise condition is $1 + c + r \leq 0$, which makes good sense because the communication-to-computation ratio of the target architecture has to play a role. In a word, when $1 + r + c \leq 0$, the rise is so small ($r \leq -(1 + c)$) that all tile columns can be processed independently. On the other hand, when $1 + r + c \geq 0$ (which is always true when $r \geq -1$), the total idle time grows quadratically with the number of processors.

Remark 2 The assumption $M \geq (P-1)|r|$ is needed to ensure the validity of the formulas. See Figure 7: (one of) the longest path is given by the processor $q = Q$, where $M = (Q-1)|r|$, and the parallel time is

$$t_P = [M + (Q-1)(1+r+c)^+]T_{comp}.$$

Note that Q may be much smaller than P in this case. The idle time becomes

$$I_a = \begin{cases} P(\frac{M}{|r|} - 1)(1 + r + c)T_{comp} \text{ if } 1 + r + c \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

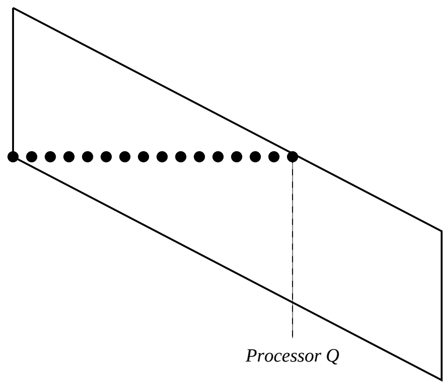

Figure 7: $r \leq 0$ and $M < P|r|$

3.4 Trapezoidal Shaped–Block Distribution

Proposition 2 *Assume a trapezoidal-shaped iteration space of size $M \times P$ (block distribution) and rises r_b (bottom) and r_t (top). If $M \geq (P-1)(|r_t| + |r_b|)$, then*

$$t_P = [M + (P-1)((1 + r_b + c)^+ + r_t - r_b)^+]T_{comp}.$$

Proof Let $C(j) = M + j(r_t - r_b)$ for $0 \leq j \leq P-1$ be the workload of processor j, that is, the total weight of column j. If $1 + r_b + c \leq 0$, then all columns can be processed independently. The total time is given by the largest processor workload: $t_P = C(0) = M$ if $r_t \leq r_b$, and $t_P = C(P-1) = M + (P-1)(r_t - r_b)$ otherwise.

If $1 + r_b + c \geq 0$, all processors spend some idle time due to horizontal communications. The discussion is similar to that in the proof of Proposition 1. If $r \leq 0$, processor q, $0 \leq q \leq P-1$, can start processing its first $j.(-r_b)$ tiles at time-step $t = 0$, but then needs to wait until time-step $t = (1+c)jT_{comp}$ before processing the rest of its column, that is, the remaining $C(j) + r_b j$ tiles. If $r \geq 0$, processor j has to wait until time-step $t = (1 + c + r)jT_{comp}$ before starting to work. In both cases, processor j terminates the execution of its column at time-step

$$t = [(1 + c + r_b)j + M + j(r_t - r_b)]T_{comp}.$$

Depending upon the sign of $(1 + c + r_b) + (r_t - r_b) = 1 + c + r_t$, this quantity is maximum either for $j = 0$ or for $j = P - 1$.

Altogether, we assemble the results of our case analysis in the above formula. ∎

Again, we point out that the condition $1 + r_b + c \leq 0$ is the key to minimizing idle time: If this condition holds, the only idle time that remains is due to the unbalanced workload (with a trapezoidal iteration space, processors have different workloads), but no overhead is due to data dependencies (and to the communications they incur).

3.5 Parallelogram Shaped–Cyclic Distribution

Proposition 3 *Assume a parallelogram-shaped iteration space of size $M \times Pb$ (cyclic distribution) and rise r. If $M \geq (P-1)br$, then*

$$t_P = \begin{cases} bMT_{comp} & \text{if } 1 + r + c \leq 0 \\ [(1 + r + c)(bP - 1) + M]T_{comp} & \text{if } 1 + r + c \geq 0, M \leq (1 + r + c)P \\ [(1 + r + c)(P - 1) + bM]T_{comp} & \text{if } 1 + r + c \geq 0, M \geq (1 + r + c)P \end{cases}$$

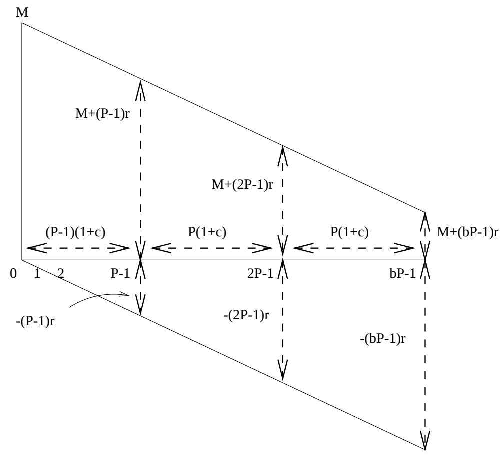

Figure 8: Sketch of the proof with $r_b \leq 0$.

Proof All processors have the same workload bMT_{comp}. If $1+r+c \leq 0$, all tile columns can be processed independently, and the first part of the result follows.

If $1+r+c \geq 0$, processor $P-1$ is always the latest one to terminate execution. We discuss the two cases $r \leq 0$ and $r \geq 0$ separately. If $r \leq 0$, the work of processor $P-1$ (which is assigned columns $P-1, 2P-1, \ldots, bP-1$), can be decomposed as follows (see Figure 8):

$$\begin{array}{lll} t_P = & \max(1+c, -r)(P-1) & \textit{max of propagating data along horizontal axis} \\ & & \quad \textit{and of computing tiles below axis in column } P-1 \\ & +\sum_{k=1}^{b-1} \max((1+c)P, M-Pr) & \textit{remaining tiles in column } kP-1 \\ & & \quad \textit{and tiles below axis in column } (k+1)P-1 \\ & +M+(bP-1)r & \textit{remaining tiles in column } bP-1 \end{array}$$

For $r \geq 0$, the same decomposition leads to (see Figure 9):

$$\begin{array}{lll} t_P = & (1+r+c)(P-1) & \textit{start-up time} \\ & +\sum_{k=0}^{b-2} \max((1+c+r)P, M) & \textit{tiles in column } j+kP \\ & +M & \textit{tiles in column } j+(b-1)P \end{array}$$

It turns out that, because $1+r+c \geq 0$, the two expressions for t_P coincide: in other words, the last expression is valid for both $r \leq 0$ and $r \geq 0$. This directly leads to the result. ∎

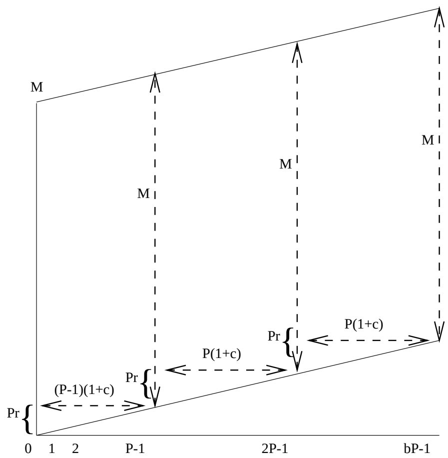

Figure 9: Sketch of the proof with $r_b \geq 0$.

3.6 Trapezoidal Shaped–Cyclic Distribution

The trapezoidal shaped–cyclic distribution is the most difficult case. We have the following result:

Proposition 4 *Assume a trapezoidal-shaped iteration space of size $M \times bP$ (cyclic distribution) and rises r_b (bottom) and r_t (top). If $M \geq (P-1)b(|r_t|+|r_b|)$, then*

$$t_P = \begin{cases} [bM + \frac{b(b-1)}{2}P(r_t - r_b)]T_{comp} & \textit{if } 1+r+c \leq 0, r_t \leq r_b \\ [bM + (\frac{b(b-1)}{2}P + b(P-1))(r_t - r_b)]T_{comp} & \textit{if } 1+r+c \leq 0, r_t \geq r_b \\ max(t(j), 0 \leq j \leq P-1) & \textit{if } 1+r+c \geq 0 \end{cases},$$

where $t(j) = [(1+c)j + \sum_{k=0}^{b-2} \max((1+c)P, M - Pr_b + (j+kP)(r_t - r_b)) + M + (j+(b-1)P)r_t]T_{comp}$.

Proof If $1+r_b+c \leq 0$, all tile columns can be processed independently. In this case, the processor that has the largest workload is processor 0 if $r_t \leq r_b$ and processor $P-1$ otherwise. The workload of processor j is $\sum_{k=0}^{b-1} C(j+kP)$, where $C(j+kP) = M + (j+kP)(r_t - r_b)$ is the weight of column $j+kP$, $0 \leq j \leq P-1$, $0 \leq k \leq b-1$. This leads to the first part of the result: if $r_t \leq r_b$, the maximum is achieved for processor 0 (and $\sum_{k=0}^{b-1} C(kP) = bM + \frac{b(b-1)}{2}P(r_t - r_b)$), while if $r_t \geq r_b$, the maximum is achieved for processor $P-1$ (and $\sum_{k=0}^{b-1} C(P-1+kP) = bM + (\frac{b(b-1)}{2}P + b(P-1))(r_t - r_b)$).

If $1 + r_b + c \geq 0$, it is more difficult to determine the longest path. If $r_b \leq 0$, we use the same decomposition as in the proof of Proposition 3 to decompose the work of processor j, $0 \leq j \leq P-1$:

$$
\begin{array}{ll}
t(j) = \max(1+c, -r_b)j & \textit{max of propagating data along horizontal axis} \\
 & \quad \textit{and of computing tiles below axis in column } j \\
+\sum_{k=0}^{b-2} \max((1+c)P, M - Pr_b + (j+kP)(r_t - r_b)) & \textit{remaining tiles in column } j+kP \\
 & \quad \textit{and tiles below axis in column } j+(k+1)P \\
+M + (j+(b-1)P)r_t & \textit{remaining tiles in column } j+(b-1)P
\end{array}
$$

We have to take the maximum value of these quantities to obtain the parallel execution time:

$$t_P = max(t(j), 0 \leq j \leq P-1).$$

Now if $r_b \geq 0$, the same decomposition leads to the expression (for processor j):

$$
\begin{array}{ll}
t(j) = (1+c+r_b)j & \textit{start-up time} \\
+\sum_{k=0}^{b-2} \max((1+c+r_b)P, M + (j+kP)(r_t - r_b)) & \textit{tiles in column } j+kP \\
+M + (j+(b-1)P)(r_t - r_b) & \textit{tiles in column } j+(b-1)P
\end{array}
$$

Again, this last expression for $t(j)$ coincides with the one when $r_b \leq 0$, hence the result. ■

Remark 3 It is not difficult to analytically compute the value of j, $0 \leq j \leq P-1$, that maximizes $t(j)$ in Proposition 4. This is a simple but tedious case analysis depending upon the problem parameters P, M, c, r_b and r_t.

4 Hierarchical Tiling

As pointed out by Högsted, Carter, and Ferrante [10], tiling may be used for multiple levels of memory and parallelism hierarchy. One important motivation for determining the idle tile of a timing in [10] was, in fact, to demonstrate that such an idle time can have a significant impact on real performance for a large application.

We reuse the example in [10] to illustrate this point. A large rectangular iteration space with horizontal and vertical dependencies is partitioned into supertiles. In turn, each supertile is partitioned into second-level tiles that are assigned to processors. See Figure 10, where supertiles and second-level tiles are rectangular, as opposed to the situation in Figure 11, where supertiles and second-level tiles have a parallelogram shape. To motivate this example, think of a large out-of-core problem, where data is stored on disk. Supertiles are brought in from disk and distributed among the processor main memories (there is an implicit synchronization between two consecutive supertiles). Which is the best strategy, rectangular tiles as in Figure 10, or parallelogram-shaped tiles as in Figure 11? It is stated in [10] that rectangular tiles incur a substantial idle time penalty, whereas parallelogram-shaped tiles do not (at least in steady state—partial tiles do incur a penalty, too).

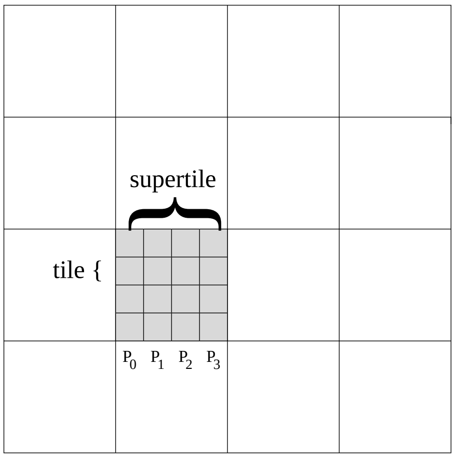

Figure 10: Partitioning the iteration space into rectangular supertiles and second-level tiles (the rise is $r = 0$).

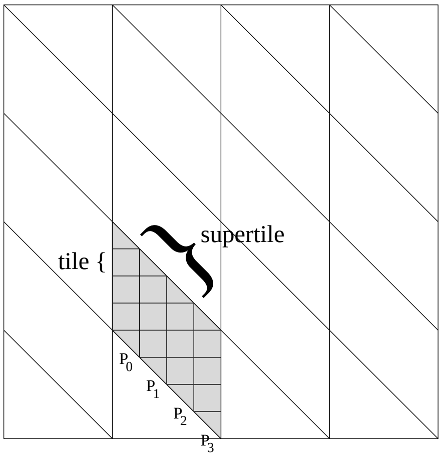

Figure 11: Partitioning the iteration space into parallelogram-shaped supertiles and second-level tiles (the rise is $r = -1$).

The results of the preceding sections enable us to answer the problem: we analytically compute the best partition shape as a function of the iteration space parameters and of the target machine characteristics.

Let h and w be the normalized height and width of second-level tiles whose processing requires $T_{comp} = hw$ time-steps. Assume a block distribution of tiles to processors so that each supertile is of size $Mh \times Pw$: in other words, in a supertile there are P columns of M tiles each. We have $P = 4$ and $M = 5$ in Figures 10 and 11. Let the size of the whole iteration space be $D_1h \times D_2w$, where $D_1 = d_1M$ and $D_2 = d_2P$. With the rectangular partitioning, there are $d_1 \times d_2$ supertiles. With the parallelogram-shaped partitioning, there are $(d_1+1) \times d_2$ supertiles, and the first and last supertiles in each column are partial. The following lemma is a direct consequence of the results in Table 2:

Lemma 1 *With the previous notations, assume that $M \geq (P-1)|r|$. The total execution time to* *process the iteration space is*

$$\begin{cases} T_{rect} = [M + (P-1)(1+c)]d_1 d_2 T_{comp} & \textit{for rectangular tiles} \\ T_{rise}(r) = [M + (P-1)(1+c+r)^+](d_1+1)d_2 T_{comp} & \textit{for parallelogram tiles with rise } r \end{cases}$$

For rectangular tiles, we rewrite T_{rect} as

$$T_{rect} = [M + (P-1)(1+c)]\frac{D_1}{M}\frac{D_2}{P}T_{comp},$$

to show that it is a decreasing function of M. In other words, M should be chosen as large as possible, namely, $M = M_{max}$, where M_{max} is such that M_{max} tiles (i.e., $M_{max}hw$ computational points of the iteration space) fit in the (cache) memory of a single processor. For the same value of M, we choose for r the smallest value such that $(1+r+c)^+ = 0$ (i.e., $r = -(1+c)$), and we derive that

$$T_{rise}(-(1+c)) = M(\frac{D_1}{M} + 1)\frac{D_2}{P}T_{comp}.$$

We formulate the following proposition:

Proposition 5 *If* $M_{max} \leq (P-1)(1+c)\frac{D_1}{M_{max}}$, *then* $T_{rise}(-(1+c)) \leq T_{rect}$.

The condition in Proposition 5 will always be true for large enough domains. In other words, parallelogram-shaped supertiles will lead to the best performance.

5 Conclusion

In this paper, we have extended results by Högsted, Carter, and Ferrante [10], and we have been able to accurately determine the idle time of a tiling for both parallelogram-shaped and trapezoidal-shaped iteration spaces. We have provided a closed-form expression of the idle time for all values of the rise parameter, for a block distribution as well as for a cyclic distribution.

Furthermore, we have used our new results in the context of hierarchical tiling. Although we have dealt only with a particular instance of the multilevel tiling problem, we believe our approach is general enough to be applied in several situations (such as those described in [6]).

Finally, we point out that 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. An interesting research direction would be to extend our approach so as to incorporate processor speed as a new parameter of the tiling 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, Jack Dongarra, and Yves Robert. Tiling with limited resources. Technical Report UT-CS-97-350, University of Tennessee, Knoxville, TN, 1997. Available at `http://www.cs.utk.edu/ yrobert`.

[5] 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.

[6] L. Carter, J. Ferrante, S. F. Hummel, B. Alpern, and K.S. Gatlin. Hierarchical tiling: a methodology for high performance. Technical Report CS-96-508, University of California at San Diego, San Diego, CA, 1996. Available at `http://www.cse.ucsd.edu/ carter`.

[7] 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.

[8] 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).

[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. Hg̈stedt, 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.

[11] 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.

[12] N. Manjikian and T.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.

[13] 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.

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

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

[16] 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.

[17] M. E. Wolf and M. S. Lam. A data locality optimizing algorithm. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 30–44. ACM Press, 1991.