Time-minimal tiling when rise is larger than zero

Jingling Xue
School of Computer Science and Engineering
University of New South Wales
Sydney, NSW 2052, Australia

Wentong Cai
School of Computer Engineering
Nanyang Technological University
Singapore 639798

Abstract. This paper presents a solution to the open problem of finding the optimal tile size to minimise the execution time of a parallelogram-shaped iteration space on a distributed memory machine when the rise of the tiled iteration space is larger than zero. Based on a new communication cost model, which accounts for computation and communication overlap for tiled programs, the problem is formulated as a discrete non-linear optimisation problem and the closed-form optimal tile size is derived. Our experimental results show that the execution times when optimal tile sizes are used are close to the experimentally best. The proposed technique can be used for hand tuning parallel codes and in optimising compilers.

Keywords: Loop tiling; loop nest; optimising compiler; SPMD

1. Introduction

Loop tiling is a useful compiler optimisation for achieving coarse-grain parallelism on distributed memory machines. *Tiling* aggregates small loop iterations into tiles [13, 21]. By executing tiles as atomic units of computation, communication takes place per tile instead of per iteration. While small tiles expose more parallelism, large tiles require less communication frequency. By adjusting the size of tiles, we can make the tradeoff between parallelism and communication. Thus, finding the optimal tile size to minimise the execution time of a program is imperative.

This paper is concerned with finding the optimal tile size to minimise the execution time of a double loop with a parallelogram-shaped iteration space on a distributed memory machine. We introduce our program model in Section 1.1 and give a statement of the problem in Section 1.2.

1.1. Program Model

Figure 1 illustrates our model, which is the most general one used in the literature when the execution time is the objective function to be minimised [1, 2, 7, 11, 14].

Iteration Space. The iteration space, which is a parallelogram with vertical left and right edges, is characterised by a triple $(W, H, s_{\mathsf{iter}})$, where s_{iter} denotes the slope of its top (or bottom) edge. The number of iterations in the iteration space is approximated by WH.

Tiling. The iteration space is divided into parallelogram-shaped tiles of the same shape and size. The left and right edges of a tile are chosen to be vertical. Thus, a tile is also specified by a triple $(w, h, s_{\mathsf{tile}})$, where s_{tile} denotes the slope of its top (or bottom) edge. The number of iterations inside a tile is approximated by wh. By definition, a tile includes its left and bottom edges but excludes its top and right.

Legality and Tile Dependences. Following [13, 23], a tiling can also be specified by a matrix:

$$\mathcal{H} = \begin{pmatrix} -\frac{s_{\mathsf{tile}}}{h} & \frac{1}{h} \\ \frac{1}{w} & 0 \end{pmatrix}$$

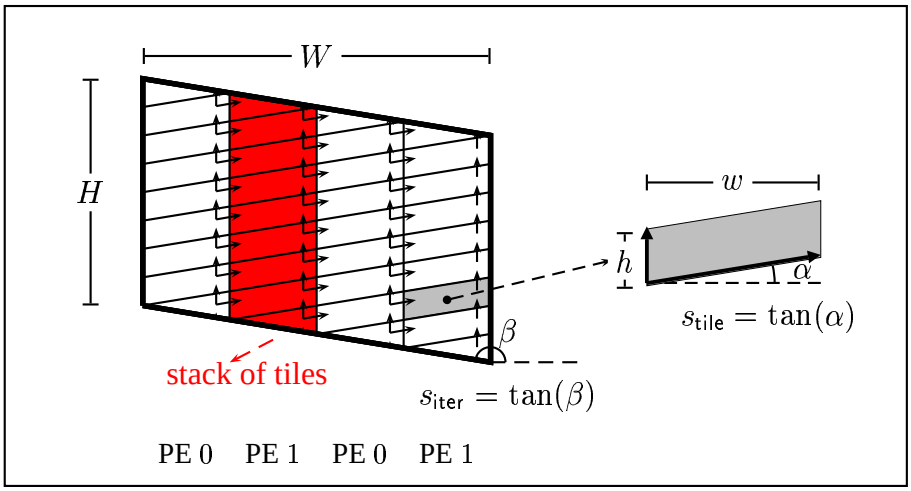


Figure 1: Tiling of a parallelogram-shaped iteration space.

It is well-known that $\mathcal{H}$ is *legal* if $\mathcal{H}\vec{d} \geqslant 0$ for every dependence vector $\vec{d} \in \mathbb{Z}^2$ in the program [13]. In a sequential program, every dependence vector $\vec{d} = (d_1, d_2)$ must be lexicographically positive, which implies $d_1 \geqslant 0$. Thus, for every $\vec{d} = (d_1, d_2)$, we have:

$$\mathcal{H}\begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \geqslant 0 \quad \Longleftrightarrow \quad s_{\text{tile}} \leqslant \frac{d_2}{d_1} \tag{1}$$

A tile is identified by a vector in $\mathbb{Z}^2$. If an iteration in a tile $\vec{t}$ depends on an iteration in another tile $\vec{t'}$, the *tile dependence* between the two tiles is the vector $\vec{t} - \vec{t'}$ [23]. In practice, the tile size is much larger than the entries of any constant distance vector. If the data dependences span a 2-D space, then the set of tile dependence vectors must include:

$$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$$

In a distributed memory machine, cross-processor tile dependences must be enforced by message-passing. The communication induced by other tile dependence vectors can be implemented by systolic pipelining along $(1, 0)$ and $(0, 1)$ via the intervening processors. For example, all messages associated with $(1, 1)$ can be sent first along $(1, 0)$ and then $(0, 1)$. Therefore, it is sufficient to assume the existence of the two elementary tile dependence vectors, which are depicted as arrows in Figure 1.

SPMD Code. The region between a pair of adjacent vertical lines is called a *stack of tiles* or a *tile stack*. All stacks are allocated cyclically to a linear arrangement of P processors numbered from 0 to $P - 1$. Figure 1 depicts a scenario where each of the two processors gets two stacks ($P = 2$). The SPMD code that a processor p executes is sketched in Figure 2 [19].

Due to our assumption made earlier on data dependences, only neighbouring communication is required. In addition, a processor will execute all the tiles in a given stack from bottom to top and

for *(every stack of tiles allocated to processor* p*)* **do**
  for *(every tile in the current stack)* **do**
    Receive one message, if any, from $(p-1) \bmod P$*;*
    Execute the current tile;
    Send one message to $(p+1) \bmod P$*, if necessary;*

Figure 2: The SPMD code at processor p, where $0 \leqslant p < P$, for a tiled program.

all its allocated stacks from left to right. A tile is an atomic unit of computation. Once scheduled, it runs to completion without preemption. After having executed a tile, a processor sends one single message to the right neighbouring processor unless the tile is close to the border of the iteration space. Before executing a tile, a processor must receive one single message from is left neighbouring processor unless the tile is close to the border of the iteration space.

1.2. Problem Statement

The concept of rise introduced in [11] is useful for two reasons. First, it enables rectangular and parallelogram-shaped iteration spaces to be treated in the same framework. Second, it is inherently related to the complexity of the problem of finding the optimal tile size.

Definition 1 The *rise* of a tiled iteration space is $r = s\frac{w}{h}$, where $s = s_{\mathsf{iter}} - s_{\mathsf{tile}}$.

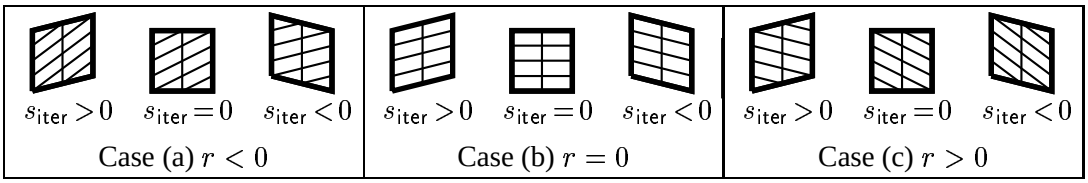


Figure 3: Three types of rise values.

Based on this concept, all tiled iteration spaces fall into the three categories illustrated in Figure 3. There have been several research efforts on finding optimal tile sizes to minimise the execution times of these iteration spaces on distributed memory machines. Cases (a) and (b) are solved in [2, 3, 14]. This paper provides a solution to Case (c).

This paper makes two contributions. First, we present a new communication cost model for quantifying the communication cost of tiled programs. Separate communication parameters are used for measuring the communication overhead incurred by a stack of tiles allocated to the same processor and the tiles allocated to different processors. This allows computation and communication overlap in a processor to be taken into account more precisely than otherwise. This model subsumes several existing ones and is equally powerful as some others [1, 2, 5, 10, 14]. Second, we give the closed-form optimal tile size for a tiled iteration space when $r > 0$. While approximate, our solution is very close to the exact optimal solution since only realistic simplifications are made (Table 2). Our experimental results show that the execution times when

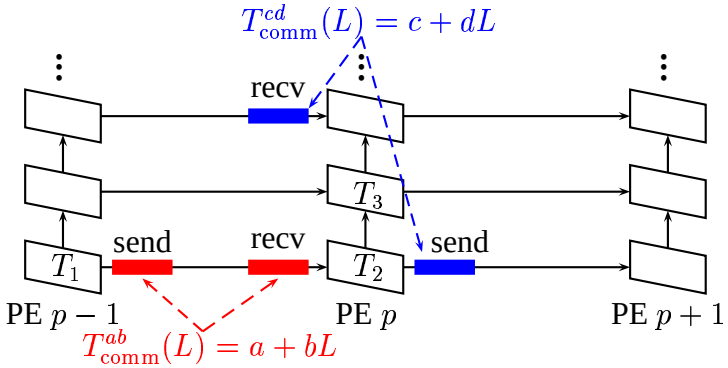


Figure 4: Communication cost model. Rectangles depict tiles and solid arrows tile dependences.

optimal tile sizes are used are close to the best observed in our experiments. This also implies the accuracy of our communication cost model for tiling purposes. Unlike Cases (a) and (b), where the distribution of tile stacks blockwise is often optimal [2, 3, 14], the optimal tiling in Case (c) often requires the tile stacks to be distributed cyclically. The tiling technique proposed in this paper is useful for hand tuning parallel codes and restructuring codes in an optimising compiler.

The rest of the paper is organised as follows. Section 2 introduces our computation and communication cost models. Section 3 derives the execution time formula and formulates the problem of finding the optimal tile size as a discrete non-linear optimisation problem. Section 4 finds the closed-form optimal tile size. Section 5 presents our experimental results for running a 2-D PDE program on a distributed memory machine Fujitsu AP1000. Section 6 discusses the related work. Section 7 concludes the paper.

2. Computation and communication cost models

In our computation model, the number of iterations in a tile is approximated by wh. Let t be the execution time for a single iteration. The time for executing a tile is given by:

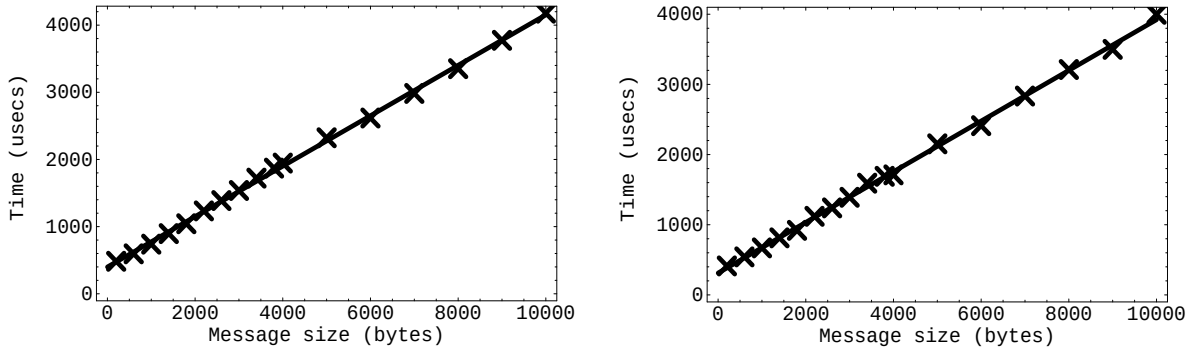
$$T_{\text{comp}} \quad = \quad wht$$

In quantifying the communication overhead, we propose to use a simple yet effective cost model designed specifically for tiled programs with the computation and communication patterns as specified in the SPMD code (Figure 2). As Figure 4 shows, different communication parameters are used to measure the communication overhead for the horizontal and vertical traversals.

Consider the two tiles T_1 and T_2 executed in two neighbouring processors as marked in Figure 4. The computations of both tiles and the associated send and receive calls must all be carried out sequentially. Here, the communication and computation cannot be overlapped. The cost of communicating a message of L bytes between two processors is modeled by:

$$T^{ab}_{\text{comm}}(L) \quad = \quad a + bL \tag{2}$$

where a is the startup cost and b is the transfer time per byte. The two parameters on Fujitsu AP1000 are shown in Figure 5(a). They are obtained using the standard ping-pong technique [8].



(a) $T^{ab}_{\text{comm}}(L) = a + bL = 398 + 0.375L$ (μsecs) (b) $T^{cd}_{\text{comm}}(L) = c + dL = 311 + 0.361L$ (μsecs)

Figure 5: Communication parameters for AP1000.

Consider the execution of a stack of tiles in the processor p as illustrated in Figure 4. In some distributed memory machines, some dedicated hardware is available for sending and receiving messages while the processor is processing a tile. For example, either the receive call or send call can be overlapped with the computation of T_3. In fact, all the three can be carried out simultaneously if enough hardware resources are available. To account for the potential computation and communication overlap that arises this way, the communication cost charged for the execution of a single tile is approximated by a one-degree polynomial function:

$$T^{cd}_{\text{comm}}(L) \quad = \quad c + dL \tag{3}$$

where c symbolises the startup cost and d is the cost paid for one byte of message. The two parameters on Fujitsu AP1000 are shown in Figure 5(b). They are obtained using the following experiment. Let K be the number of tiles in a tile stack allocated to a processor. Let T be the execution time of the entire tile stack. The communication overhead for the message size determined by a tile is given by $(T - KT_{\text{comp}})/K$. By fitting (3) to the data obtained this way for different message sizes (i.e., tile sizes), the values of c and d are obtained.

Due to computation and communication overlap, $a \geqslant c$ and $b \geqslant d$ are expected to hold. If no computation and communication overlap is supported, $a = c$ and $b = d$ are expected.

Our cost model does not directly incorporate other costs such as buffer management and system-specific overheads. Therefore, the values of the four communication parameters are simple to establish for a machine. Since tiled programs are highly regular, these additional costs are typically proportional to the number and size of messages communicated [8]. Our experimental results confirm that our model can be used effectively to predict optimal tile sizes.

The message size L for a tile, which is proportional to the tile height h, is approximated by:

$$L \quad = \quad gh$$

where g can be approximated based on the data dependences of the program [2, 14, 22].

Thus, the two communication cost formulas can be refined to:

$$\begin{aligned} T^{ab}_{\text{comm}}(L) &= T^{ab}_{\text{comm}}(gh) &= a + bgh \\ T^{cd}_{\text{comm}}(L) &= T^{cd}_{\text{comm}}(gh) &= c + dgh \end{aligned}$$

3. Non-linear optimisation problem

For a fixed P, we use $\mathcal{S}$ to denote the set of all possible tile sizes or tilings:

$$\mathcal{S} \;=\; \{(w,h) \mid 1 \leqslant w \leqslant \frac{W}{P}, 1 \leqslant h \leqslant H\} \tag{4}$$

In Section 4, we discuss how to use fewer than P processors, whenever possible, to achieve the same execution time. In the extreme case when one single processor is the best choice, then w is allowed to take effectively the values in the interval $[1, W]$, i.e., $1 \leqslant w \leqslant W$.

A tiling (w,h) yields a *single-pass* program if $w = \frac{W}{P}$, i.e., if every processor is assigned exactly one stack of tiles and a *multiple-pass* program otherwise. A tiling (w,h) is *pass-idle* if a processor idle waits between the execution of two consecutive stacks of tiles and *pass-free* otherwise. By definition, a tiling that yields a single-pass program is always pass-free.

Let

$$\begin{aligned} T_{ab} &= T_{\text{comp}} + T^{ab}_{\text{comm}}(gh) \;=\; wht + a + bgh \\ T_{cd} &= T_{\text{comp}} + T^{cd}_{\text{comm}}(gh) \;=\; wht + c + dgh \end{aligned}$$

Figure 6 illustrates the derivation of the execution time formulas for these two cases. The critical path in each case represents the earliest completion time for the last processor $P-1$, which is the total execution time of the program when all processors are assigned the same number of tile stacks. Let k be the number of passes (i.e., cycles):

$$k \;=\; \frac{W}{wP}$$

Let us write the various segments depicted in Figure 6(a) as follows:

$$\begin{aligned} T_1 &= PrT_{cd} = Ps\frac{w}{h}T_{cd} \\ T'_1 &= (P-1)rT_{cd} \;\approx\; T_1 \\ T_2 &= PT_{ab} \\ T'_2 &= (P-1)T_{ab} \;\approx\; T_2 \\ T_3 &= \frac{H}{h}T_{cd} \end{aligned}$$

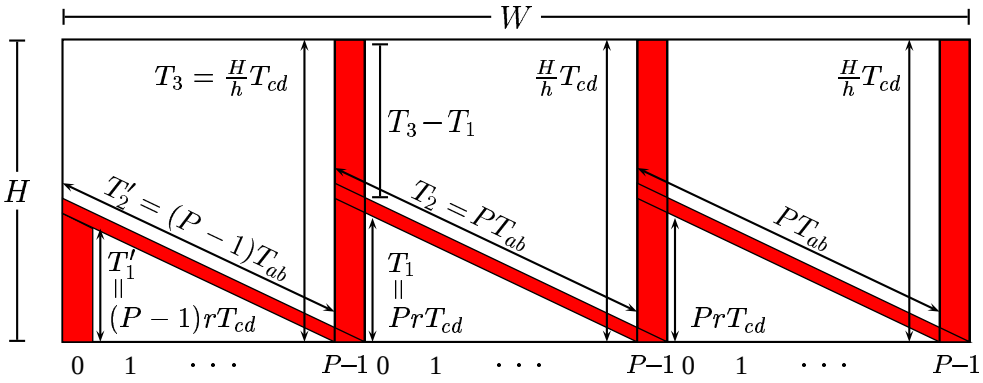
Let

$$C(w,h) \;=\; T_2 - (T_3 - T_1) \;=\; PT_{ab} - (\frac{H}{h} - Ps\frac{w}{h})T_{cd}$$

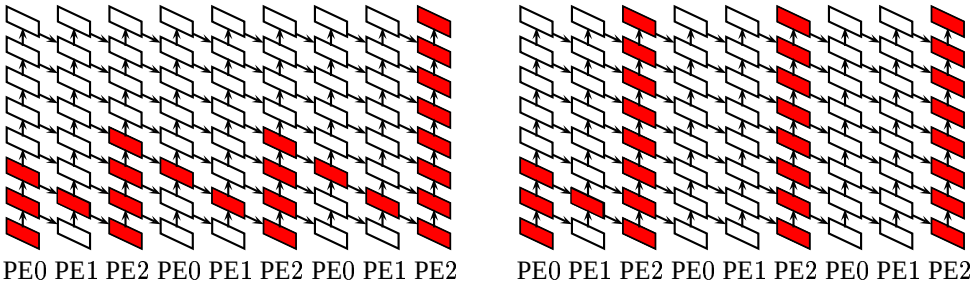
Consider the situation when a processor has just finished the bottom T_1 tiles in a tile stack. Clearly, a tiling is pass-idle if $C(w,h) > 0$ and pass-free if $C(w,h) \leqslant 0$.

The execution time corresponding to the critical path shown in Figure 6(b) is:

$$\begin{aligned} T_{\mathsf{idle}}(w,h) &= k(T_1 + T_2) + T_3 \\ &= \frac{W}{w}(T_{ab} + s\frac{w}{h}T_{cd}) + \frac{H}{h}T_{cd} \end{aligned} \tag{5}$$



(a) The critical path for the last processor $P-1$



(b) Pass-idle ($P = 3$)

(c) Pass-free ($P = 3$)

Figure 6: Derivation of execution time formulas.

The execution time corresponding to the critical path shown in Figure 6(c) is:

$$\begin{aligned} T_{\mathsf{free}}(w,h) &= T_1 + T_2 + kT_3 \\ &= P(T_{ab} + s\frac{w}{h}T_{cd}) + \frac{HW}{whP}T_{cd} \end{aligned} \tag{6}$$

The k (i.e., the number of passes), $c(w,h)$, $T_{\mathsf{idle}}(w,h)$ and $T_{\mathsf{free}}(w,h)$ are related as follows:

$$(k-1)C(w,h) = T_{\mathsf{idle}}(w,h) - T_{\mathsf{free}}(w,h) \tag{7}$$

which will be exploited in the proof of Theorem 4.

The problem of finding the optimal tile size is formulated as follows:

$$\begin{array}{ll} \textbf{Minimise} & T(w,h) \\ \textbf{Subject to} & (w,h) \in \mathcal{S} \end{array} \tag{8}$$

where

$$T(w,h) = \begin{cases} T_{\mathsf{idle}}(w,h) & \text{if } C(w,h) > 0 \\ T_{\mathsf{free}}(w,h) & \text{if } C(w,h) \leqslant 0 \end{cases} \tag{9}$$

The optimal tile slope s_{tile} (i.e., the one that minimises $T(w,h)$) is chosen to be the smallest of the slopes of all distance vectors. For a fixed (w,h), $T(w,h)$ is minimised if $s = s_{\mathsf{iter}} - s_{\mathsf{tile}}$ is the smallest possible, i.e., if s_{tile} is the largest possible such that (1) is satisfied.

4. Closed-form optimal tile size

As a discrete non-linear optimisation problem, (8) is difficult to solve directly. Our solution strategy is as follows. In Section 4.1, we show that the following problem

$$\begin{array}{rl} \textbf{Minimise} & T_{\mathsf{idle}}(w,h) \\ \textbf{Subject to} & w,h \geqslant 0 \end{array} \tag{10}$$

has a unique minimum point and provide an accurate approximation to the minimum point. In Section 4.2, we do the same for the following problem:

$$\begin{array}{rl} \textbf{Minimise} & T_{\mathsf{free}}(w,h) \\ \textbf{Subject to} & w,h \geqslant 0 \end{array} \tag{11}$$

Sometimes, the minimum point of (8) is located on the boundaries of its solution space. This is addressed in Section 4.3. In Section 4.4, we combine all these results to provide a solution to (8).

The proofs of all theorems except Theorem 4 are given in the appendix.

4.1. Solving (10)

Theorem 1 *(10) has a unique minimum point, denoted* (w_i^*, h_i^*).

An approximation to (w_i^*, h_i^*) is derived next. Our experimental results show that our approximation often differs from the real minimum point only in the fractional part.

The minimum point of (10) must satisfy the following two equations:

$$\frac{\partial T_{\mathsf{idle}}}{\partial w} = tH - \frac{(a + bg\,h - stw^2)\,W}{w^2} = 0 \tag{12}$$

$$\frac{\partial T_{\mathsf{idle}}}{\partial h} = \left(t + \frac{bg}{w}\right) W - \frac{c\,(H + sW)}{h^2} = 0 \tag{13}$$

Solving (13) for h yields its unique positive root:

$$h_i^* = \sqrt{\frac{cHw + cswW}{bgW + twW}} \tag{14}$$

Substituting this into (12) and simplifying, we get:

$$\mathcal{F}_{\mathsf{idle}}(w) = Ht + stW - \frac{aW}{w^2} - \frac{bg}{w^2}\sqrt{\frac{cw(H+sW)W}{bg+tw}} = 0 \tag{15}$$

By dropping the last complex but less dominant term in (15) since $a \gg bg$, we get:

$$\mathcal{F}_{\mathsf{idle}}(w) \;=\; Ht + stW - \frac{aW}{w^2} \;=\; 0$$

The optimal tile width w is *under* approximated as the root of the above equation:

$$w_i^* \;\approx\; \sqrt{\frac{aW}{Ht + stW}} \tag{16}$$

The corresponding optimal tile height becomes:

$$h_i^* \;=\; \sqrt{\frac{cHw_i^* + csw_i^*W}{bgW + tw_i^*W}} \tag{17}$$

4.2. Solving (11)

We rely on the following assumption to prove that (11) also has a unique minimum point.

$$HW \;\geqslant\; \frac{b^2g^2P^2s}{4t^2} \tag{18}$$

which should always be true since $WH \gg P$ and $\frac{b^2g^2s}{4t^2}$ is small for practical applications. For the PDE example in Section 5 running on AP1000, This inequality becomes $HW \geqslant 0.16P^2$.

Theorem 2 *If (18) holds, (11) has a unique minimum point, denoted* (w_f^*, h_f^*).

We derive the interval in which w_f^* lies and argue that the interval is extremely tight since the communication parameters b and d are close. We give an accurate approximation to (w_f^*, h_f^*).

The minimum point of (11) must satisfy the following two equations:

$$\frac{\partial T_{\mathsf{free}}}{\partial w} \;=\; \frac{P^2\left(cs + h\left(dgs + t\left(h + 2sw\right)\right)\right) - \frac{(c+dgh)HW}{w^2}}{hP} \;=\; 0 \tag{19}$$

$$\frac{\partial T_{\mathsf{free}}}{\partial h} \;=\; bgP + Ptw - \frac{c\left(P^2sw^2 + HW\right)}{h^2Pw} \;=\; 0 \tag{20}$$

Solving (20) for h yields its unique positive solution:

$$h_f^* \;=\; \sqrt{\frac{cP^2sw^2 + cHW}{bgP^2w + P^2tw^2}} \tag{21}$$

Substituting this into (19) and simplifying, we get:

$$\mathcal{F}_{\mathsf{free}}(w) \;=\; dgPs + 2Pstw - \frac{dgHW}{Pw^2} + \frac{\sqrt{cP^2w}\left(bgPs + 2Pstw - \frac{bgHW}{Pw^2}\right)}{\sqrt{bg + tw}\sqrt{P^2sw^2 + HW}} \;=\; 0 \tag{22}$$

Rearranging this equation yields:

$$\mathcal{F}_{\mathsf{free}}(w) \;=\; f_d(w) + \frac{\sqrt{cP^2w}f_b(w)}{\sqrt{bg+tw}\sqrt{P^2sw^2+HW}}$$

where

$$f_d(w) \;=\; dgPs + 2Pstw - \frac{dgHW}{Pw^2} \tag{23}$$

$$f_b(w) \;=\; bgPs + 2Pstw - \frac{bgHW}{Pw^2} \tag{24}$$

By Descartes' rule of signs [9], both $f_d(w) = 0$ and $f_b(w) = 0$ each have a unique positive root. Let w_d^* be the positive root of $f_d(w) = 0$ and w_b^* the positive root of $f_b(w) = 0$. Clearly, w_f^* must lie within the interval defined by w_d^* and w_b^*. We discuss how to find w_f^* by assuming $d \leqslant b$. The situation is reversed when $d \geqslant b$ and is thus omitted. If $d \leqslant b$, the following condition holds:

$$w_d^* \leqslant w_b^*$$

where the sign of the inequality holds iff $d = b$. This is simply because

$$f_b(w) - f_d(w) \;=\; (b-d)gPs - \frac{(b-d)gHW}{Pw^2} \leqslant 0 \tag{25}$$

on the interval $(0, \sqrt{\frac{HW}{sP^2}}]$ and

$$f_b(\sqrt{\frac{HW}{sP^2}}) \;=\; f_d(\sqrt{\frac{HW}{sP^2}}) \;=\; \frac{2t}{P}\sqrt{\frac{(HW)^3}{s}} \;>\; 0$$

Thus,

$$w_d^* \leqslant w_f^* \leqslant w_b^* \tag{26}$$

The closed-form expressions for the two bounds can be obtained by solving algebraically $f_d(w) = 0$ and $f_b(w) = 0$, which are essentially cubic polynomial equations [9]. Since d and b are close, these two bounds are extremely tight. Thus, w_f^* can be found from the interval by a simple search.

For practical applications, even such a simple search does not seem to be necessary. When HW is large, the term $dgPs$ in $f_d(w) = 0$ can be dropped. This gives rises to:

$$w_f^* \;\approx\; \overline{w_f^*} \;=\; \sqrt[3]{\frac{dgHW}{2P^2st}} \tag{27}$$

Of course, w_f^* can also be approximated equally well from the equation $f_b(w) = 0$. By substituting (27) into (21), the corresponding optimal tile height becomes:

$$h_f^* \;=\; \sqrt{\frac{cP^2s(w_f^*)^2 + cHW}{bgP^2w_f^* + P^2t(w_f^*)^2}}$$

P \ $W \times H$	$5K \times 10K$	$10K \times 10K$	$10K \times 20K$
10	$[72.88, 73.88]/73.01$	$[91.86, 93.11]/91.99$	$[57.82, 58.61]/57.85$
50	$[24.84, 25.18]/24.96$	$[31.33, 31.76]/31.46$	$[19.69, 19.96]/19.82$
100	$[15.60, 15.81]/15.72$	$[19.69, 19.96]/19.82$	$[12.36, 12.52]/12.48$
200	$[\ 9.78,\ 9.91]/\ 9.91$	$[12.36, 12.52]/12.48$	$[\ 7.75,\ 7.84]/\ 7.86$

Table 1: Some representative values for $[w_d^*, w_b^*]/\overline{w_f^*}$ (when $d \leqslant b$). The four communication parameters are from Figure 4. The other three are taken from the PDE example in Section 5.

Table 1 shows that the bounds given in (26) are tight. There are few points in the interval $[w_d^*, w_b^*]$. In addition, our approximation $\overline{w_f^*}$ to w_f^* is accurate – $\overline{w_f^*}$ is within $[w_d^*, w_b^*]$ except in one case.

4.3. Solving (8) at Its Boundaries

The boundaries of (8) consist of the four sides of its solution space $\mathcal{S}$ and the curve $C(w, h) = 0$. We rely on the following two assumptions to find out how $C(w, h) = 0$ intersects $\mathcal{S}$:

$$P \geqslant \frac{c + dgH}{a + bgH} \tag{28}$$

$$H > \frac{a + bg}{c + dg} P \tag{29}$$

where $\frac{c+dgH}{a+bgH} < 1$ and $\frac{a+bg}{c+dg} P$ is slightly larger than P for practical applications.

Depending on the values of various parameters used, $C(w, h) = 0$ may or may not intersect the four sides of $\mathcal{S}$. The following theorem identifies all such intersection points, if any.

Theorem 3 *Suppose that both (28) and (29) are true. Let*

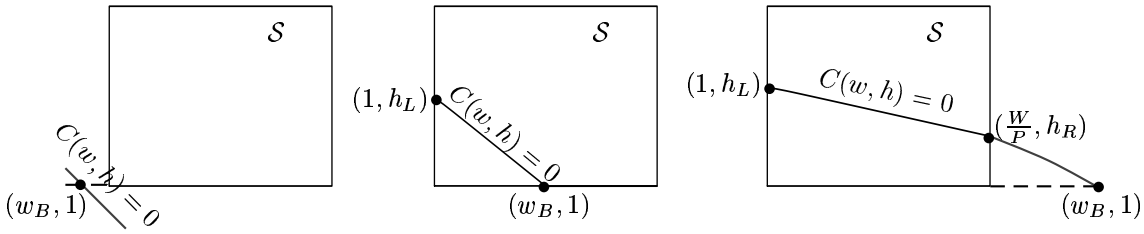
$$w_B = \frac{-cPs - dgPs + Ht - Pt + \sqrt{-4P(-cH - dgH + aP + bgP)st + (cPs + dgPs - Ht + Pt)^2}}{2Pst} \tag{30}$$

$$h_L = \frac{dgH - aP - dgPs + Ht - Pst + \sqrt{-4(-cH + cPs)(bgP + Pt) + (-dgH + aP + dgPs - Ht + sPt)^2}}{2(bgP + Pt)} \tag{31}$$

$$h_R = \frac{-aP^2 + dgP(H - sW) + tW(H - sW) + \sqrt{4cP^2(H - sW)(bgP + tW) + (aP^2 + dgP(-H + sW) + tW(-H + sW))^2}}{2P(bgP + tW)} \tag{32}$$

which are unique positive roots to $C(w, 1) = 0$, $C(1, h) = 0$ and $C(\frac{W}{P}, h) = 0$, respectively. There are three cases, as illustrated in Figure 7, regarding whether and how $C(w, h) = 0$ intersects the solution space $\mathcal{S}$ of (8): (a) If $w_B < 1$, $C(w, h)$ does not intersect $\mathcal{S}$, (b) If $1 \leqslant w_B \leqslant \frac{W}{P}$, $C(w, h) = 0$ intersects the left of $\mathcal{S}$ at $(1, h_L)$ and the bottom at $(w_B, 1)$, and (c) If $w_B > \frac{W}{P}$, $C(w, h) = 0$ intersects the left side of $\mathcal{S}$ at $(1, h_L)$ and the right at $(\frac{W}{P}, h_R)$.

Based on this theorem, (8) with its solution space being restricted to its boundaries is solved as follows. There are four straight line segments in Figure 7(a), six straight line segments and one curve in Figure 7(b) and six straight line segments and one curve in Figure 7(c). The minimum point of (8) in each of these segments is found individually. The minimum point of (8) in all these segments is simply the one such that $T(w, h)$ is minimised. All these segments are regarded as being closed since $T_{\mathsf{idle}}(w, h) = T_{\mathsf{free}}(w, h)$ on the curve $C(w, h) = 0$ according to (7).



(a) $w_B < 1$ (b) $1 \leqslant w_B \leqslant \frac{W}{P}$ (c) $w_B > \frac{W}{P}$

Figure 7: Three scenarios regarding whether and how $C(w,h)=0$ intersects $\mathcal{S}$.

The minimisation of (8) in a straight line segment is straightforward since the objective function $T(w,h)$, which simplifies to either $T_{\mathsf{idle}}(w,h)$ or $T_{\mathsf{free}}(w,h)$, is a strictly convex single-variable function over the line segment. As an example, let us consider the line segment $[1,h_L]$. $T_{\mathsf{free}}(w,h)$, i.e., $T_{\mathsf{free}}(1,h)$ is the objective function for the line segment. The unique solution to $\frac{\partial T_{\mathsf{free}}(1,h)}{\partial h}=0$ is $\sqrt{\frac{cP^2s+cHW}{bgP^2+P^2t}}$. Thus, $T_{\mathsf{free}}(1,h)$ attains its minimum on $[1,h_L]$ at:

$$(w^*,h^*) \;=\; \max(1,\min(\sqrt{\frac{cP^2s+cHW}{bgP^2+P^2t}},h_L)$$

The minimum points on the other straight line segments can be found in a similar manner.

The minimisation of (8) on the curve $C(w,h)=0$ is hard to solve exactly. By assuming $a=c$ and $b=d$ on our four communication parameters, an approximate solution is found below. In this case, the objective function is $T_{\mathsf{idle}}(w,h)$. In fact, $T_{\mathsf{idle}}(w,h)=T_{\mathsf{free}}(w,h)$ on the curve $C(w,h)=0$ according to (7). We note that $C(w,h)$ can be simplified to:

$$C(w,h) \;=\; P+sP\frac{w}{h}-\frac{H}{h}$$

Solving $C(w,h)=0$ for h, we can express h in terms of w as follows:

$$h_C^* \;=\; \frac{H-Psw}{P} \tag{33}$$

Substituting this into the objective function $T_{\mathsf{idle}}(w,h)$ yields:

$$\mathcal{F}_C(w) \;=\; \frac{H\left(cP+dg\left(H-sPw\right)+tw\left(H-Psw\right)\right)\left(Pw+W\right)}{Pw\left(H-sPw\right)}$$

To find the minimum point on $C(w,h)=0$, we need to solve:

$$\begin{array}{rl}\textbf{Minimise} & \mathcal{F}_C(w)\\ \textbf{Subject to} & 1\leqslant w\leqslant \min(w_B,\frac{W}{P})\end{array}$$

Differentiating $\mathcal{F}_C(w)$ with respect to w yields:

$$\begin{aligned}\frac{\partial\mathcal{F}_C}{\partial w} \;=\;& s^2tP^3w^4-2stHP^2w^3+\left(asP^3+tH^2P-bgs^2P^2W\right)w^2\\ &+\left(2bgsHPW+2asP^2W\right)w-bgH^2W-aHPW\end{aligned}$$

The signs of the coefficients for the equation

$$\frac{\partial \mathcal{F}_C}{\partial w} = 0 \tag{34}$$

are $+ - * + -$, where $*$ is either $+$ or $-$. By Descartes' rule [9], this equation has either one or three positive roots. Since the equation is quartic, all its roots can be found algebraically. Thus, $T_{\mathsf{idle}}(w,h)$ on $C(w,h)=0$ is minimised at (w_C^*, h_C^*), where w_C^* is one of the positive roots of (34) or one of the two end points of the interval $[1, \min(w_B, \frac{W}{P})]$ and h_C^* is given in (33).

4.4. Solving (8)

We define:

$$\begin{aligned} \mathcal{S}_{\mathsf{idle}} &= \mathcal{S} \cap \{(w,h) \mid C(w,h) > 0\} \\ \mathcal{S}_{\mathsf{free}} &= \mathcal{S} \cap \{(w,h) \mid C(w,h) \leqslant 0\} \end{aligned}$$

The objective function $T(w,h)$ given in (9) is made up of $T_{\mathsf{idle}}(w,h)$ and $T_{\mathsf{free}}(w,h)$, where $T_{\mathsf{idle}}(w,h)$ is defined over S_{idle} and $T_{\mathsf{free}}(w,h)$ over S_{free}.

By combining the results presented in Sections 4.1 – 4.3, we obtain the following theorem.

Theorem 4 *The minimum point of (8), denoted $(w_{\mathsf{opt}}, h_{\mathsf{opt}})$, is obtained as follows. If $(w_i^*, h_i^*) \in S_{\mathsf{idle}}$, then $(w_{\mathsf{opt}}, h_{\mathsf{opt}}) = (w_i^*, h_i^*)$. If $(w_f^*, h_f^*) \in S_{\mathsf{free}}$, then $(w_{\mathsf{opt}}, h_{\mathsf{opt}}) = (w_f^*, h_f^*)$. Otherwise, $(w_{\mathsf{opt}}, h_{\mathsf{opt}})$ is the minimum point of (8) solved at its boundaries as described in Section 4.3.*

Proof. Due to (7), $T_{\mathsf{idle}}(w,h) \leqslant T(w,h)$ over $\mathcal{S}$. In addition, $T_{\mathsf{idle}}(w,h) = T(w,h)$ over $\mathcal{S}_{\mathsf{idle}}$. If $(w_i^*, h_i^*) \in S_{\mathsf{idle}}$, then (8) attains its minimum at (w_i^*, h_i^*), i.e., $(w_{\mathsf{opt}}, h_{\mathsf{opt}}) = (w_i^*, h_i^*)$. For reasons of symmetry, the proof when $(w_f^*, h_f^*) \in S_{\mathsf{free}}$ is similar. That is, if $(w_f^*, h_f^*) \in S_{\mathsf{free}}$, then $(w_{\mathsf{opt}}, h_{\mathsf{opt}}) = (w_f^*, h_f^*)$. If $(w_i^*, h_i^*) \notin S_{\mathsf{idle}}$ and $(w_f^*, h_f^*) \notin S_{\mathsf{free}}$, an application of Theorems 1 and 2 shows that (8) must attain its minimum at its boundaries. ■

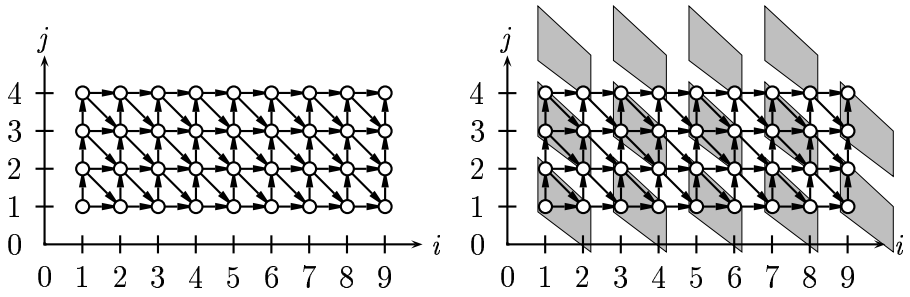
$T_{\mathsf{idle}}(w,h)$ and the pass-idle optimal tile size (w_i^*, h_i^*) are independent of P. The same execution time can be obtained if $P^* < P$ is used, where P^* is the solution to $C(w_i^*, h_i^*) = 0$:

$$P^* = \frac{H\left(c + dgh_i^* + w_i h_i^* t\right)}{h_i^* \left(a + bgh_i^* + w_i^* h_i^* t + \frac{sw_i^*\left(c + dgh_i^* + w_i^* h_i^* t\right)}{h_i^*}\right)} \tag{35}$$

The pass-free optimal tile size (w_f^*, h_f^*) is a function of WH, i.e., the product of W and H. This makes a perfect sense: the same optimal tile size will be found regardless whether the iteration space size is $W \times H$ or $H \times W$ as long as all processors are not idle!

5. Experiments

In this section, we present some experimental results of running a PDE program on a Fujitsu AP1000 with 128 processors. The AP1000 used consists of 128 SPARC cells each consisting of a SPARC processor running at 25MHz with 16MB RAM. All processors are connected by a 2-D torus network called the *T-net*. The T-net provides the interprocessor communication using the wormhole routing with a 25MB/sec of bandwidth. All I/O is performed on the host.



(a) Iteration space (b) 2×2 tiling

Figure 8: Parallelogram-shaped tiling of the PDE program.

Our example is the following PDE program modified from [20]:

$$\begin{array}{l} \mathsf{do}\ i = 1,\ W \\ \quad \mathsf{do}\ j = 1,\ H \\ \quad\quad a(j) = \frac{1}{3}(a(j-1) + a(j) + a(j+1)) \end{array}$$

where the outer loop serves as a time step. The data dependences can be captured by three distance vectors $(1, 0)$, $(0, 1)$ and $(1, -1)$. As discussed in Section 3, the optimal tile slope is $s = 1$. Thus, the parallelogram-shaped tiles as depicted in Figure 8 are used.

The SPMD code for the example is written in C. Each array element consumes four bytes in memory. Based on the tile dependences in Figure 8, the message size for a tile is deduced to be $L = gh = 4h$. The execution time of a single loop iteration is measured to be $t = 1.85\mu$secs

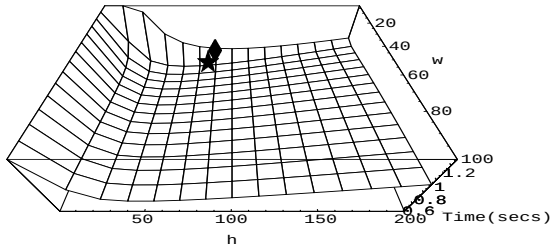
This paper is concerned with finding the optimal tile size that yields the smallest execution time rather than predicting the execution time for a given tile size. Therefore, the objective of our experiments is to evaluate how close the execution times of the optimal tile sizes found by our theory and experiments are. A series of five experiments were conducted to analyse and validate various aspects of our theory. The experimental results are presented in Figures 9 – 13. In each experiment, the theoretically optimal tile size $(w_{\mathsf{opt}}, h_{\mathsf{opt}})$ is found by Theorem 4 and the experimentally optimal tile size is the best observed in all test cases. If required, $(w_{\mathsf{opt}}, h_{\mathsf{opt}})$ is taken as the best of $(\lceil w_{\mathsf{opt}} \rceil, \lceil h_{\mathsf{opt}} \rceil)$, $(\lceil w_{\mathsf{opt}} \rceil, \lfloor h_{\mathsf{opt}} \rfloor)$, $(\lfloor w_{\mathsf{opt}} \rfloor, \lceil h_{\mathsf{opt}} \rceil)$ and $(\lfloor w_{\mathsf{opt}} \rfloor, \lfloor h_{\mathsf{opt}} \rfloor)$.

In the first four experiments, the iteration space size used is $W = 1000$ and $H = 2000$. In the first experiment illustrated in Figure 9, $P = 10$ processors are used. By Theorem 4, $(w_{\mathsf{opt}}, h_{\mathsf{opt}}) = (w_f^*, h_f^*) = (19.82, 91.57) \approx (20, 92)$. In the second experiment illustrated in Figure 10, $P = 50$, the optimal tile size is found to be on the curve $C(w, h) = 0$: $(w_{\mathsf{opt}}, h_{\mathsf{opt}}) = (8.89, 31.53) \approx (9, 31)$. In the third experiment illustrated in Figure 11, $P = 100$ processors are available. We have $(w_{\mathsf{opt}}, h_{\mathsf{opt}}) = (w_i^*, h_i^*) = (8.47, 21.45) \approx (9, 22)$. Instead of using $P = 100$ processors, we can obtain the same execution time by using $P^* = 61.13 \approx 61$ processors according to (35). Thus, $P = 61$ processors are used in the fourth experiment as illustrated in Figure 12. As before, $(w_{\mathsf{opt}}, h_{\mathsf{opt}}) = (9, 22)$. By comparing Figures 11 and 12, we find that the execution times of the theoretically optimal tile sizes in both cases are close and the execution times of the experimentally optimal tile sizes in both cases are also close. In the last experiment shown in

★ Theoretical ♦ Experimental

	5	50	70	90	92	96	100	200
5	3.2166	0.6238	0.5483	0.5085	0.5019	0.4953	0.4963	0.8083
10	2.3253	0.5455	0.4966	0.4726	0.4682	0.4644♦	0.4649	0.7764
20	1.8944	0.5210	0.4897	0.4778	0.4752★	0.4737	0.4751	0.7823
50	1.6540	0.5538	0.5466	0.5558	0.5555	0.5582	0.5621	0.8491
100	1.5990	0.6427	0.6668	0.7063	0.7089	0.7176	0.7263	0.9846

(a) Execution times in seconds



(b) Execution times of (a) in curves

Figure 9: Performance on 10 processors ($W = 1000$ and $H = 2000$).

Experiment	$(w_{\mathsf{opt}}, h_{\mathsf{opt}})$ by Theorem 4	Optimum by Mathematica
Figure 9	$(19.82, 91.57)$	$(19.71, 92.05)$
Figure 10	$(8.89, 31.53)$	$(8.89, 31.53)$
Figure 11	$(8.47, 21.45)$	$(8.80, 21.49)$
Figure 12	$(8.47, 21.45)$	$(8.80, 21.49)$
Figure 13	$(20, 114.36)$	$(20, 114.36)$

Table 2: Approximate and real optimal solutions for the PDE example.

Figure 13, we consider a scenario in which the optimal tile size is located on the right side of $\mathcal{S}$. With $W = 1000$, $H = 80000$ and $P = 50$, we obtain $(w_{\mathsf{opt}}, h_{\mathsf{opt}}) = (20, 114.36) \approx (20, 115)$.

The optimal tile size $(w_{\mathsf{opt}}, h_{\mathsf{opt}})$ found by Theorem 4 is approximate. However, all the experiments we have performed show that our approximate solution is very close to the exact optimal solution found by numerical simulation using Mathematica. Table 2 compares our approximate solutions with the optimal ones found by Mathematica for the PDE example.

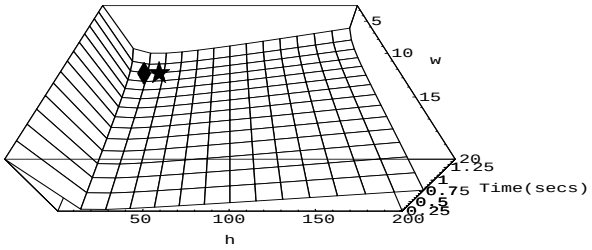
Our experimental results show that the execution times of the optimal tile sizes found by theory and experiments are very close. Therefore, our theory can be used, for example, in an optimising compiler to choose the best tile size to minimise the execution time of a tiled program.

Some observations are in order here. First, if $r > 0$, the cyclic distribution of tile stacks is often the best choice. If $r \leqslant 0$, block distribution is preferred unless $W \gg H$ [2, 3, 14]. Second, the curves around the minimum point in each of our experiments are flat, implying that

★ Theoretical ♦ Experimental

	5	10	20	22	31	50	100	200
5	0.6889	0.4036	0.2730	0.2767	0.2933	0.3428	0.4936	0.8083
8	0.6381	0.3746	0.2527	0.2557	0.2711	0.3197	0.4697	0.7826
9	0.6647	0.3835	0.2522♦	0.2548	0.2698★	0.3171	0.4674	0.7789
10	0.5301	0.3358	0.2522	0.2542	0.2689	0.3160	0.4645	0.7762
20	0.4755	0.3336	0.2864	0.2849	0.2934	0.3331	0.4757	0.7826

(a) Execution times in seconds



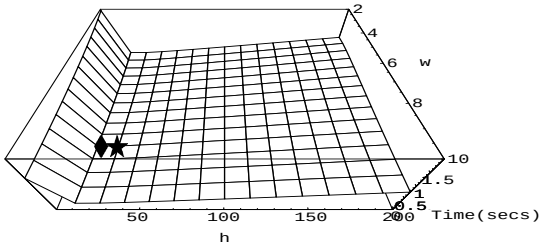
(b) Execution times of (a) in curves

Figure 10: Performance on 50 processors ($W = 1000$ and $H = 2000$).

★ Theoretical ♦ Experimental

	5	10	20	22	31	50	100	200
2	0.6546	0.2786	0.4074	0.4112	0.4346	0.4841	0.6557	.9445
4	0.4712	0.2953	0.2933	0.2965	0.3187	0.3662	0.5221	1.1246
5	0.4040	0.2784	0.2737	0.2766	0.2959	0.3486	0.5013	0.8121
9	0.4406	0.2669	0.2522♦	0.2546★	0.2710	0.3191	0.4693	0.7814
10	0.3531	0.2700	0.2522♦	0.0.2540	0.2695	0.3174	0.4662	0.7784

(a) Execution times in seconds



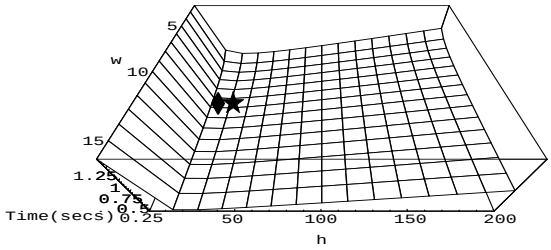
(b) Execution times of (a) in curves

Figure 11: Performance on 100 processors ($W = 1000$ and $H = 2000$).

★ Theoretical ♦ Experimental

	5	10	20	22	31	50	100	200
2	1.0549	0.5541	0.4078	0.4112	0.4351	0.4842	0.6563	0.9443
5	0.6442	0.3603	0.2739	0.2768	0.2958	0.3491	0.5013	0.8133
9	0.4870	0.3078	0.2525	0.2548★	0.2710	0.3194	0.4692	0.7814
10	0.5051	0.3142	0.2522♦	0.2540	0.2696	0.3173	0.4663	0.7784
17	0.4310	0.3086	0.2722	0.2723	0.2827	0.3270	0.4784	0.7772

(a) Execution times in seconds



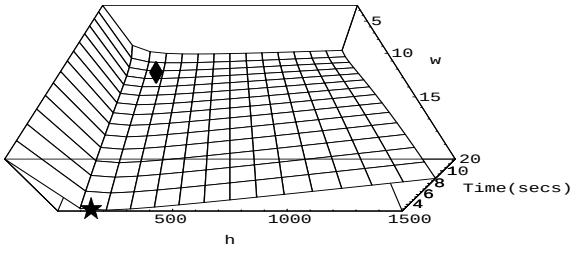
(b) Execution times of (a) in curves

Figure 12: Performance of on 61 processors ($W = 1000$ and $H = 2000$).

★ Theoretical ♦ Experimental

	5	50	115	240	241	242	300	1500
5	8.9536	5.0325	3.6567	3.1811	3.5639	3.5662	3.6825	5.9003
8	9.0037	5.3693	4.0541	3.5870	3.5842	3.5837	3.5246	5.6748
9	9.7364	5.8647	4.4390	3.8960	3.8934	3.8919	3.8085	5.7846
10	6.3230	4.4017	3.4654	3.2380	3.2376♦	3.2382	3.2533	5.9006
20	6.3231	4.1725	3.5240★	3.5629	3.5640	3.5662	3.6825	7.0644

(a) Execution times in seconds



(b) Execution times of (a) in curves

Figure 13: Performance on 50 processors ($W = 1000$ and $H = 80000$).

the execution times for those tile sizes are close.

6. Related work

In their seminal work, Irigoin and Triolet [13] and Wolfe [21] introduce loop tiling as a compiler optimisation to improve data locality and parallelism in a program. There has also been an interesting thread of research on finding the best tile shape for minimising communication overheads once the tile size is given [4, 6, 15, 17, 22]. These research efforts are compared in [22] and not repeated here. Some recent efforts on locality optimisations can be found in [16, 18].

Given a parallelogram-shaped iteration space to be tiled and executed in the SPMD mode on a distributed memory machine, the problem of finding the optimal tile size to minimise the execution time of the program has been solved when $r < 0$ or $r = 0$ [2, 5, 14]. In [2], the optimal tile size is presented for the case $r = 0$. Their communication cost model consists of four parameters β, τ_c, β_m and τ_t, where $\beta_m + \tau_t L$ predicts the cost of transferring L bytes between two processors, $\tau_c L$ the cost for copying a message between the user and system memories, and β the overhead of a send (or receive) OS call. Based on their assumption that the matching send and receive in two communicating processors may be properly synchronised, they manipulate these terms to quantify the communication cost for a stack of tiles (corresponding to our $T^{cd}_{\text{comm}}(L)$) and for the tiles across the processors ($T^{ab}_{\text{comm}}(L)$). In [14], the optimal tile sizes for both cases $r < 0$ and $r = 0$ are given. The execution time formula for the pass-free case when $r > 0$ is derived but the optimal tile size is not. In their communication cost model, $a = c$ and $b = d$ are assumed. In [5], it is assumed that computation and communication completely overlap with each other, i.e., $c = d = 0$. The optimal tile size degenerates into $(1, 1)$.

In [1], the problem of this paper is tackled independently. Based on the communication cost model from [2], their versions of our optimisation problems (10) and (11), called the $\mathcal{P}_2$ and $\mathcal{P}_1$ problems, are formulated as follows:

$$\begin{array}{lrl} \mathcal{P}_2\text{:} & \textbf{Minimise} & T_{\mathsf{idle}}(w,h) \\ & \textbf{Subject to} & (w,h) \in \mathcal{S}_{\mathsf{idle}} \\ \mathcal{P}_1\text{:} & \textbf{Minimise} & T_{\mathsf{free}}(w,h) \\ & \textbf{Subject to} & (w,h) \in \mathcal{S}_{\mathsf{free}} \end{array}$$

They present an approximate solution to the $\mathcal{P}_2$ problem. They adopt a two-step approach to solving the $\mathcal{P}_1$ problem. In the first step, they find an approximate optimal solution (w^*, h^*) for a fixed but arbitrary tile volume $wh = v$, i.e., over the hyperbola $wh = v$. Thus, w^* and h^* are each a function of v, say, $w^* = f_1(v)$ and $h^* = f_2(v)$. In the second step, they substitute w^* and h^* back into the objective function and obtain a new function of v only. They then find an (approximate) optimal tile volume v^* to minimise this new function. The problem with this two-step approach is that there is no guarantee that $(f_1(v^*), f_2(v^*))$ will be inside $\mathcal{S}_{\mathsf{free}}$, i.e., the domain of the $\mathcal{P}_1$ problem. If it is, the optimal tile size is $(w_{\mathsf{opt}}, h_{opt}) = (f_1(v^*), f_2(v^*))$. Otherwise, they distinguish two cases to find *by heuristics* a point in $\mathcal{S}_{\mathsf{free}}$ as a substitute for the optimal solution, depending on whether the hyperbola $wv = v^*$ intersects $\mathcal{S}_{\mathsf{free}}$ or not. In comparison with this work, we solve the problem by finding an approximate solution (w^*_i, h^*_i) to (10) and an approximate solution (w^*_f, h^*_f) to (11). By Theorem 4, (w^*_i, h^*_i) is optimal if

$(w_i^*, h_i^*) \in \mathcal{S}_{\mathsf{idle}}$ and (w_f^*, h_f^*) is optimal if $(w_f^*, h_f^*) \in \mathcal{S}_{\mathsf{free}}$. Otherwise, the optimal solution at a boundary point of $\mathcal{S}$ is found by a careful analysis of how the curve $C(w, h) = 0$ intersects $\mathcal{S}$.

In the past few years, some authors [3, 14] have also attempted to find the optimal tile sizes for higher-dimensional iteration spaces. The optimal solution is attainable for the so-called orthogonal tiling problem, assuming that all dependences are uniform (or nearly so), the iteration space is an n-dimensional parallelepiped, and its facets are parallel to the tile facets.

In [11], the authors investigate the idle time associated with a tiling, i.e., the time that processors are idle because they are either waiting for data from other processors or waiting to synchronise with other processors. They introduce the concept of rise to relate the shape of the iteration space with that of the tiles. Based on this concept, they present the idle time and execution time formulas for parallelogram-shaped and trapezoidal iteration spaces. Later in [7], simpler formulas with simpler proofs are presented. In addition, the execution time formulas for all three types of rises are also given. Recently, this line of research has been extended by considering block distribution of parallelepiped tiles to processors for n-dimensional convex iteration spaces [12]. In their model, determining the execution time of a tiling reduces to a linear programming problem. They discuss the impact of tile shape on the execution time of a tiling. In addition, closed-form execution time formulas for a subclass of convex iteration spaces, called the rectilinear iteration spaces, are also presented. In their communication cost model, the startup overhead is included in the execution time of a tile. The time for moving bytes from one processor to another is modeled to allow explicit computation and communication overlap.

In [10], the authors minimise the execution time by exploiting the DOALL parallelism at the level of tiles. They assume that an unbounded number of processors is available so that all tiles in the same wavefront can be executed simultaneously. Based on the simplified cost model, where $a = c$ and $b = d = 0$, they give closed-form optimal tile sizes for some special class of programs.

7. Conclusion

This paper provides a solution to the open problem of finding the optimal tile size to minimise the execution time of a tiled parallelogram-shaped iteration space on a distributed memory machine. To quantify the communication cost for tiled programs, a communication cost model that accounts for computation and communication overlap is presented. Based on this model, the problem of finding the optimal tile size is formulated as a discrete non-linear optimisation problem and the closed-form optimal tile size is derived. Our experimental results show that our cost model can predict accurately the communication cost for tiled programs and that our approximate optimal tile size yields the execution time close to the best observed in experiments.

For practical programs with W and H much larger than P, (w_f^*, h_f^*) is expected to be the optimal tile size. Then a processor will not be idle waiting between processing any two consecutive stacks of tiles. In this commonly occurring case, the exact optimal tile size can be found from the segment $([w_d^*, w_b^*], h_f^*)$, where the two bounds for w_f^* is extremely tight. Our experimental results show that the approximate optimal tile size found by Theorem 4 is accurate in practice.

8. Acknowledgements

The authors are grateful to the referees for their constructive and helpful comments, which have greatly improved the presentation of the paper. In particular, the formula given in (7) is from

one of the referees. This work is supported by an Australian Research Council Grant A10007149.

9. References

[1] R. Andonov, P. Y. Calland, S. Niar, S. Rajopadhye, and N. Yanev. First steps towards optimal oblique tile sizing. In *8th International Workshop on Compilers for parallel computers*, pages 351–366, Aussois, Jan. 2000.

[2] R. Andonov and S. Rajopadhye. Optimal orthogonal tiling of 2-D iterations. *Journal of Parallel and Distributed Computing*, 45(2):159–165, Sept. 1997.

[3] R. Andonov, S. Rajopadhye, and N. Yanev. Optimal orthogonal tiling. In David Pritchard and Jeff Reeve, editors, *1998 European Parallel Processing Conference*, Lecture Notes in Computer Science 1470, pages 480–490, Sept. 1998.

[4] P. Boulet, A. Darte, T. Risset, and Y. Robert. (Pen)-ultimate tiling. *Integration, the VLSI Journal*, 17:33–51, 1994.

[5] P. Y. Calland, J. Dongarra, and Y. Robert. Tiling on systems with communication/computation overlap. *Concurrency: Practice and Experience*, 11(3):139–153, 1999.

[6] P. Y. Calland and T. Risset. Precise tiling for uniform loop nests. In P. Cappello, editor, *International Conference on Application Specific Array Processors*, pages 330–337. IEEE Computer Society Press, 1995.

[7] F. Desprez, J. Dongarra, F. Rastello, and Y. Robert. Determining the idle time of a tiling: New results. *Journal of Information Science and Engineering (Special Issue on Compiler Techniques for High-Performance Computing)*, 14(1):167–190, Mar. 1998.

[8] I. Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley, 1994.

[9] L. Griffiths. *Introduction to the Theory of Equations*. Jon Wiley & Sons, Inc., 1947.

[10] E. Hodžić and W. Shang. On supernode transformation with minimized total running time. *IEEE Trans. on Parallel and Distributed Systems*, 9(5):417–428, May 1998.

[11] K. Högstedt, L. Carter, and J. Ferrante. Determining the idle time of a tiling. In *24th Annual ACM Symposium on Principles of Programming Languages*, pages 160–173, Paris, Jan. 1997.

[12] K. Högstedt, L. Carter, and J. Ferrante. Selecting tile shape for minimal execution time. In *11th ACM Symposium on Parallel Algorithms and Architectures*, pages 201–211, Jun. 1999.

[13] F. Irigoin and R. Triolet. Supernode partitioning. In *15th Annual ACM Symposium on Principles of Programming Languages*, pages 319–329, San Diego, California., Jan. 1988.

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

[15] J. Ramanujam and P. Sadayappan. Tiling multidimensional iteration spaces for multicomputers. *J. of Parallel and Distributed Computing*, 16(2):108–230, Oct. 1992.

[16] G. Rivera and C.-W. Tseng. A comparison of compiler tiling algorithms. In *8th International Conference on Compiler Construction*, Armsterdam, the Netherlands, March 1999.

[17] R. Schreiber and J. J. Dongarra. Automatic blocking of nested loops. Technical Report 90.38, RIACS, May 1990.

[18] Y. Song and Z. Li. New tiling techniques to improve cache temporal locality. In *ACM SIGPLAN'99 Conf. on Programming Language Design and Implementation*, pages 215–228, May 1999.

[19] P. Tang and J. Xue. Generating efficient tiled code for distributed memory machines. *Parallel Computing*, 26(11):1369–1410, 2000.

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

[21] M. J. Wolfe. More iteration space tiling. In *Supercomputing '88*, pages 655–664, Nov. 1989.

[22] J. Xue. Communication-minimal tiling of uniform dependence loops. *Journal of Parallel and Distributed Computing*, 42(1):42–59, 1997.

[23] J. Xue. On tiling as a loop transformation. *Parallel Processing Letters*, 7(4):409–424, 1997.

Appendix A

Proof of Theorem 1: Consider $\mathcal{F}_{\mathsf{idle}}(w)$ given in (15). First, $\mathcal{F}_{\mathsf{idle}}(w) > 0$ when $w \to \infty$ and $\mathcal{F}_{\mathsf{idle}}(w) \to -\infty$ when $w \to 0$. Second, we have:

$$\frac{\partial \mathcal{F}_{\mathsf{idle}}}{\partial w} = \frac{2aW}{w^3} + \frac{bg\left(3bg+4tw\right)W^2\sqrt{cw\left(H+sW\right)}}{2w^3\sqrt{\left(bg+tw\right)^3W^3}} > 0$$

so that $\mathcal{F}_{\mathsf{idle}}(w)$ is strictly increasing on the interval $(0,\infty)$. Thus, (10) has a unique solution. ■

Proof of Theorem 2: Consider $\mathcal{F}_{\mathsf{free}}(w)$ given in (22). First, $\mathcal{F}_{\mathsf{free}}(w) \to \infty$ when $w \to \infty$ and $\mathcal{F}_{\mathsf{free}}(w) \to -\infty$ when $w \to 0$. Second, if (18) holds, we have:

$$\frac{\partial \mathcal{F}_{\mathsf{free}}}{\partial w} = f_1(w) + \frac{f_2(w)+f_3(w)}{f_4(w)}$$

where

$$\begin{aligned}
f_1(w) &= 2Pst + \frac{2dgHW}{Pw^3} \\
f_2(w) &= bc^2gHP^3W\left(3bg\left(2P^2sw^2+HW\right)+4tw\left(3P^2sw^2+HW\right)\right) \\
f_3(w) &= c^2P^5sw^4\left(4Ht^2W-b^2g^2P^2s\right) \\
f_4(w) &= 2w\sqrt{\left(bgP^2w+P^2tw^2\right)^3}\sqrt{\left(cP^2sw^2+cHW\right)^3}
\end{aligned}$$

such that $f_1(w), f_2(w), f_4(w) > 0$ on $(0, \infty)$. If (18) holds, $f_3(w) \geqslant 0$ in $(0, \infty)$. Thus, $\mathcal{F}_{\mathsf{free}}(w)$ is strictly increasing in $(0, \infty)$. Thus, (11) has a unique solution. ■

The proof of Theorem 3 relies on Lemmas 1 and 2 introduced below.

Lemma 1 *If (28) holds, $C(w, h) = 0$ does not intersect the half line $\{(w, H) \mid w \geqslant 0\}$.*

Proof. Setting $h = H$ and simplifying the equation $C(w, H) = 0$, we get:

$$C(w, H) \;=\; HPstw^2 + \big(cPs + dgHPs - H^2t + H^2Pt\big)\, w - cH - dgH^2 + aHP + bgH^2P \;=\; 0$$

The coefficients of w^2 and w are both positive. If (28) holds, the constant term is nonnegative. Thus, the equation does not have a positive root. ■

Lemma 2 *If (29) holds, $C(w, h) = 0$ intersects the half line $\{(w, 1) \mid w \geqslant 0\}$ at only one point, denoted $(w_B, 1)$, where w_B given in (30) is the unique positive root to the equation $C(w, 1) = 0$. In addition, if w is on $(0, w_B)$, then $C(w, 1) < 0$. If w is on (w_B, ∞), then $C(w, 1) > 0$.*

Proof. Setting $h = 1$ and simplifying the equation $C(w, 1) = 0$, we get:

$$C(w, 1) \;=\; Pstw^2 + (cPs + dgPs - Ht + Pt)w - cH - dgH + aP + bgP \;=\; 0$$

The coefficient of w^2 is positive. If (29) holds, the constant term is negative. Thus, the above equation has a positive root w_B as given in (30) such that $C(w_B, 1) = 0$, and a negative root. The rest of the lemma follows from the fact that $C(w, 1)$ is a strictly convex function. ■

Proof of Theorem 3: The three cases as illustrated in Figure 7 are proved below:

(a) $w_B < 1$. By Lemma 2, $C(w, 1) > 0$ for every $(w, 1) \in \mathcal{S}$. This implies that $C(w, h) > 0$ for every $(w, h) \in \mathcal{S}$. Thus, $C(w, h) = 0$ does not intersect $\mathcal{S}$ as illustrated in Figure 7(a).

(b) $1 \leqslant w_B \leqslant \frac{W}{P}$. By Lemma 2, $C(w, 1) < 0$ when w is on $(0, w_B)$ and $C(w, 1) > 0$ when w is on (w_B, ∞). By further applying Lemma 1, $C(w, h) = 0$ intersects the left of $\mathcal{S}$ at $(1, h_L)$ and its bottom at $(w_B, 1)$ as illustrated in Figure 7(b), where h_L given in (31) is the unique positive root of the quadratic equation $C(1, h) = 0$.

(c) $w_B > \frac{W}{P}$. By Lemmas 1 and 2, $C(w, h) = 0$ intersects the left of $\mathcal{S}$ at $(1, h_L)$ and the right at $(\frac{W}{P}, h_R)$ as illustrated in Figure 7(c), where h_R given in (32) is the unique positive root of the quadratic equation $C(\frac{W}{P}, h) = 0$. ■