Optimal Orthogonal Tiling of 2-D Iterations

Rumen Andonov
LIMAV, Valenciennes, France
andonov@univ-valenciennes.fr

Sanjay Rajopadhye
IRISA, Rennes, France
rajopadh@irisa.fr

Abstract

Iteration space tiling is a common strategy used by parallelizing compilers and in performance tuning of parallel codes. We address the problem of determining the tile size that minimizes the total execution time. We restrict our attention to uniform dependency computations with two dimensional, parallelogram shaped iteration domain which can be tiled with lines parallel to the domain boundaries. The target architecture is a linear array (or a ring). Our model is developed in two steps. We first abstract each tile by two simple parameters, namely tile period, P_t and inter-tile latency, L_t. We formulate and partially resolve the corresponding optimization problem independent of the machine and program. Next, we refine the model with realistic machine and program parameters, yielding a discrete nonlinear optimization problem. We solve this analytically, yielding a closed form solution, which can be used by a compiler before code generation.

Keywords: coarse grain pipelining, SPMD programs, loop blocking, nonlinear optimization, communication-computation overlap, supernode partitioning, automatic parallelization, macro-systolic arrays.

1 Introduction

Tiling the iteration space [Wol87, IT88, RS91] is a common method for improving the performance of loop programs on distributed memory machines. It may be used as a technique in parallelizing compilers (see [HKT94, PSCB94] where it is also called coarse-grain pipelining), as well as in performance tuning of parallel codes by hand (see also [KCN90b, SD90]). A *tile* in the iteration space is a (hyper) parallelepiped shaped collection of iterations to be executed as a single unit with the following protocol—all the (non-local) data required for each tile is first obtained. The body of the tile is executed next, and this is repeated iteratively. The communications are performed by send/receive calls and the code for the body contains no communication calls.

The *tiling problem* can be broadly defined as the problem of choosing the tile parameters (notably the shape and size) in an optimal manner. It may be decomposed into two subproblems: *tile shape optimization* [BDRR94], and *tile size optimization* [AR96, HKT94, KCN90a, PSCB94] (some authors also attempt to resolve both problems under some simplifying assumptions [RS91, SD90]). By its very nature, such a two step approach is not globally optimal, but is necessary in order to make the problem tractable.

In this paper, we address the tile size problem, which, for a given tile shape, seeks to choose the size (length along each dimension of the hyper parallelepiped) so as to minimize the total execution time. In its most general formulation, this is a hard discrete non-linear optimization problem, and there is currently no solution. However, optimal solutions can be found analytically under certain restrictions.

We assume that the dependencies are uniform, the iteration space (domain) is two-dimensional and parallelogram shaped, and the tile boundaries are parallel to the domain boundary (we call this *orthogonal tiling*). Due to space limitations, we do not treat the *non* orthogonal case (this is discussed elsewhere [AR96] but a closed form solution is not currently available, as also noted independently by Ohta et al. [OSKO95]).

Our constraints may seem overly restrictive at first glance, but similar assumptions are tacitly made by most researchers treating the problem. For example, the Fortran 90 and Paradigm compilers automatically determine the block size for coarse grain pipelined loops [HKT94, PSCB94], assuming that the iteration space is rectangular. Moreover, although they treat arbitrarily nested loops, it is only the two innermost loops that are tiled. Thus, they also impose similar restrictions. Our main contributions are as follows.

- We propose a two step approach in which we first abstract each tile by two simple parameters: tile period, P_t and inter-tile latency, L_t. We formulate and partially resolve the corresponding abstract optimization problem.
- We then use a precise model of the machine and program behavior to obtain a non-linear function that gives the running time of the tiled program. We model the possibility of multiple passes (the so called *block cyclic* distribution).
- We analytically solve the optimization problem and obtain a closed form solution. In addition, we show which of the two standard projection directions to choose.

The remainder of this paper is organized as follows. Sections 2-4 are devoted to the above three points. In Section 5 we highlight the advantages of our two step approach by showing how our model deals easily with variations of the machine and program. In

Section 6 we discuss related work and show how our model subsumes many others in the literature. We present our conclusions in Section 7.

2 Abstract Problem Formulation

Our approach is an extension of systolic array synthesis techniques. The original computation is a uniform recurrence equation (URE, [KMW67]) whose domain is an $N_1 \times N_2$ parallelogram, $\mathcal{P}$. It is well known [RS91] that orthogonal tiling is possible if all dependencies lie in the cone formed by the edges of $\mathcal{P}$. Let the tiles be of size $x_1 \times x_2$. Next, we write a new recurrence that describes the tiled program (each index point in this recurrence corresponds to a complete tile and dependencies are "inter-tile" dependencies). It is well known [IT88] that this recurrence can be defined over an $n_1 \times n_2$ rectangle ($n_1 = \frac{N_1}{x_1}$ and $n_2 = \frac{N_2}{x_2}$), and has the canonical basis vectors (namely $[1,0]$ and $[0,1]$) as dependencies, provided the tiles are large relative to the original dependencies (this is a common assumption, and only introduces a small lower bound on the tile size). Additional dependencies, if any, will be *binary* linear combinations of these, and can be ignored for analysis purposes (their communication can be subsumed by the canonical dependencies). Thus, we need to analyze the following URE which specifies the computation of the tile graph, defined over the domain $\{i,j \mid 0 < i \leq n_1,\ 0 < j \leq n_2\}$.

$$X[i,j] = f(X[i-1,j], X[i,j-1]) \tag{1}$$

The next step is to apply systolic space-time transformations [QR89], yielding a virtual systolic array, and to implement this array on a fixed size array (or ring) of p processors (by performing multiple passes if necessary). The optimal schedule of URE (1) is given by the vector $[1,1]$, i.e., tile $[i,j]$ is computed at step $i+j$. Moreover, in order to have nearest neighbor interconnections, it is known [ZRW92] that there are only four possible projection directions (i.e., allocations of tiles to processors), namely, $[1,0]$, $[0,1]$ $[1,1]$ and $[1,-1]$. Of these, the $[1,-1]$ direction conflicts with the schedule and the $[1,1]$ direction yields an array with only 50% efficiency*. Hence, we really have only two choices: projection

*Although it seems obvious that such an inefficient systolic array is not optimal, it is not easy to prove formally. This was done recently by Calland et al. under some simplifying assumptions [CDR97].

along either $[1, 0]$ or $[0, 1]$. Both of these yield very similar arrays. We consider the $[1, 0]$ projection (visualized as a vertical projection): tile $[i, j]$ is computed on processor j (the other case can be analyzed similarly). The space-time image of the tile graph is an $n_1 \times n_2$ parallelogram, and corresponds to $n_1 + n_2$ "time steps" on n_2 virtual processors. Hence, we will need $\frac{n_2}{p}$ passes through our p-processor ring. We use the following terminology.

- Two tiles are said to be *successive* if one depends directly on the other.
- The *tile period*, P_t is the time between corresponding instructions in two successive tiles executing on the *same* processor.
- The *inter-tile latency*, L_t is the time between corresponding instructions in two successive tiles on *adjacent* processors.

We use the convention that subscripts t, m and p denote, respectively, a *single tile*, a *macro-column* (i.e., a column of the tile graph) and a *pass* through the ring. Since a macro-column contains n_1 tiles, the period and execution time of a macro-column, are both given by

$$P_m = T_m = n_1 P_t = \frac{N_1}{x_1} P_t \tag{2}$$

A single pass involves p communications (the last one is around the ring), and we define the latency L_p of a pass as

$$L_p = pL_t \tag{3}$$

Eqns. (2) and (3) give us two constraints that determine when the next pass can start. The first macro-column must be completely finished, and the last processor should have produced its first results. Hence, the pass-period of the ring is

$$P_p = \max(P_m, L_p) = \max\left(\frac{N_1}{x_1} P_t, pL_t\right) \tag{4}$$

In a single pass, the last macro-column can start only at $(p-1)L_t$, and hence, the execution time of a pass is given by

$$T_p = (p-1)L_t + T_m = (p-1)L_t + \frac{N_1}{x_1} P_t \tag{5}$$

The entire program is executed in n_2/p passes, of which, the last pass can only start at $(\frac{n_2}{p} - 1)P_p$. Thus, the total running time, $T(x_1, x_2)$, is $\left(\frac{n_2}{p} - 1\right) P_p + T_p$. Considering the two cases of Eqn. (4) and simplifying, our optimization problem is formulated as follows.

Prob. 1: Minimize

$$T(x_1,x_2) = \begin{cases} \frac{N_1}{x_1}P_t \geq pL_t \ \Rightarrow \ \dfrac{N_1N_2}{px_1x_2}P_t + (p-1)L_t = T_1(x_1,x_2) \quad \text{say} \\ \frac{N_1}{x_1}P_t \leq pL_t \ \Rightarrow \ \frac{N_1}{x_1}P_t + (\frac{N_2}{x_2}-1)L_t = T_2(x_1,x_2) \quad \text{say} \end{cases} \tag{6}$$

in the feasible space, $\mathcal{R}$ given below (the lower bounds on x_2 may be other than 1, based on the dependencies of the original recurrence).

$$\mathcal{R} = \left\{[x_1,x_2] \in \mathcal{Z}^2 \mid 1 \leq x_1 \leq N_1,\ 1 \leq x_2 \leq \frac{N_2}{p}\right\} \tag{7}$$

2.1 Solution of the Abstract Problem

We now try to resolve the optimization problem as far as possible, without entering into the details of the machine, and/or the program. The condition $\frac{N_1}{x_1}P_t = pL_t$ divides $\mathcal{R}$ into two regions. In **region I**, $\frac{N_1}{x_1}P_t \geq pL_t$ and the cost function is given by $T_1(x_1,x_2)$. In **region II**, $\frac{N_1}{x_1}P_t \leq pL_t$ and the cost function is given by $T_2(x_1,x_2)$. It is easy to verify that $T_1(x_1,x_2) = T_2(x_1,x_2)$ when $\frac{N_1}{x_1}P_t = pL_t$ i.e. the function is continuous across the region boundary. Also, when we have only one pass (i.e., $x_2 = \frac{N_2}{p}$), then $T_1(x_1,x_2) = T_2(x_1,x_2)$. We therefore consider the entire segment $1 \leq x_1 \leq N_1,\ x_2 = \frac{N_2}{p}$ as an "extended" region I.

We claim that only the (extended) region I is of interest. To see this, first note that P_t and L_t are functions of x_1 and x_2 but are *independent of* p (provided that the machine and program parameters do not depend on the *number* of processors). Hence, for any feasible tile size, $[x,y]$ in region II, as we decrease p, the non-linear constraint $\frac{N_1}{x_1}P_t = pL_t$ moves closer and closer to $[x,y]$, until ultimately $[x,y]$ is in region I. Second, the function $T_2(x_1,x_2)$ is *also independent of* p. This gives us the following general strategy.

1. Solve the optimization problem corresponding to Eqns. (6-7) as two sub-problems (minimize $T_1(x_1,x_2)$ in region I, and $T_2(x_1,x_2)$ in region II). Compare the two solutions. If the better one is in region I, we are done.

2. If the optimal solution is in region II, we reduce p (for this tile size) until we are back in region I, and we then re-solve the first sub-problem with this reduced number of

processors. This solution is guaranteed to be no worse than the optimal solution with p processors.

The intuitive explanation is that in region II, the latency of a pass is larger than the period of a macrocolumn, and hence the processors are idle between successive passes. This is a waste of resources, and we do not expect the optimal solution to be here. This seems obvious, but it is difficult to prove formally that the optimal solution can *never* occur in region II, for all possible variations of machine and program parameters. The following lemma gives us a sufficient condition.

Lemma 1 If the minimal value of $T_1(x_1, x_2)$ over the **entire** feasible space $\mathcal{R}$ is in the (extended) region I, then it is also the optimal solution of **Prob. 1**.

Proof: $T_2(x_1, x_2) - T_1(x_1, x_2)$ can be simplified to $(\frac{N_2}{px_2} - 1)(pL_t - \frac{N_1}{x_1}Pt)$. Hence in region II, $T_2(x_1, x_2) \geq T_1(x_1, x_2)$. The minimal value of $T_2(x_1, x_2)$ in region II is therefore no less than the minimal value of $T_1(x_1, x_2)$ over $\mathcal{R}$. ∎

3 Machine and Program Specific Model

We now "instantiate" **Prob. 1** for a specific program and machine architecture. Assume that the time to execute a single instance of the loop body is τ_a. The time to execute a tile body is then, $\tau_a x_1 x_2$ (we neglect the overhead of setting up the loop for each tile, as well as cache effects). Let us further assume that the program is such that the data that is communicated is the *result* of the computation of the iterations. This arises, for example, in dynamic programming for string comparison and the knapsack problem. In Section 5, we will consider a variant where the transmitted data is a tile *input*, and hence may be overlapped with the computation (as in matrix-vector multiplication).

We assume simple "blocking" communication calls, and use the standard communication model [Bok90]). The following sequence of events occurs when a processor sends a message. The sending processor first makes an OS call, an overhead of β_s (the time for switching contexts, allocating message buffers, and setting up DMA channels). The OS copies data from user to system memory, which takes $v\tau_c$ time (for a v-byte message). The main processor is now free to continue, while a communication coprocessor takes care of establishing a channel (a small overhead, which we ignore) and actually transmitting the data, which takes $v\tau_t$ time ($1/\tau_t$ is the network bandwidth). The receiving procedure is similar with a constant overhead of β_r (usually comparable to β_s). The processor blocks until the data arrives. Once the data arrives it is copied to user memory, and the system call returns. In general, the time spent waiting for data to arrive is unknown. However, for tiled programs all the computation is synchronized, and under steady state (i.e., except for the very first call) there will be no waiting.

A tile repeatedly executes a `receive-compute-send` loop, and we now specify the tile period and latency. P_t is as given below.

$$\begin{aligned} P_t &= (\beta_r + \tau_c x_1) + \tau_a x_1 x_2 + (\beta_s + \tau_c x_1) \\ &= \beta_r + \beta_s + 2\tau_c x_1 + \tau_a x_1 x_2 \end{aligned} \tag{8}$$

Now, assume that two (successive) tiles start simultaneously on adjacent processors. Because of the dependency between the tiles, the `receive` on the second tile will block until the compute first tile is *finished*, its results are *sent*, they *arrive* at the other, and are *copied* into user memory. Hence,

$$L_t = P_t + \tau_t x_1 - \beta_r \quad = \quad \beta_s + \tau_a x_1 x_2 + (2\tau_c + \tau_t) x_1 \tag{9}$$

Note that β_r is subtracted because the `receive` occurs on a *different* processor, and when the sender and receiver are properly synchronized, the calls are overlapped. Substituting in Eqn. (6) and simplifying,

$$\begin{aligned} T_1(x_1, x_2) &= \frac{N_1 N_2(\beta_r + \beta_s)}{p x_1 x_2} + (p-1)\tau_a x_1 x_2 + (p-1)(\tau_t + 2\tau_c) x_1 \\ &\quad + \frac{2 N_2 N_1 \tau_c}{p x_2} + (p-1)\beta_s + \frac{N_2 N_1 \tau_a}{p} \end{aligned} \tag{10}$$

We denote by **Prob. 2:** the problem of minimizing Eqn. 10 over $\mathcal{R}$.

4 Solution of the Optimization Problem

We assume that $(N_1 - p)P_t > \tau_t - \beta$. This holds in most cases, since $\tau_t - \beta$ is a constant, P_t is always positive, and since the problem size is usually (significantly) larger than the available number of processors. In this case, it is easy to verify that the $x_1 = 1$ boundary of $\mathcal{R}$ is within region I.

Theorem 1 If $(N_1 - p)P_t > \tau_t - \beta$, the optimal tile size for **Prob. 1** satisfies either $x_1 = 1$ or $x_2 = \frac{N_2}{p}$ boundary.

Proof: The result follows from Lemma 1, if we can show that the solution of **Prob. 2** satisfies either $x_1 = 1$ or $x_2 = \frac{N_2}{p}$.

Now, for any point in the feasible region, $T_1(x_1, x_2)$ strictly decreases if we reduce x_1, keeping $x_1 x_2$ fixed, i.e., "move left along a hyperbola": the first two terms in (10) remain unchanged, the third and the fourth terms are strictly smaller, and the remaining terms are constants. If we continue to move in this manner, we must eventually satisfy $x_1 = 1$ or $x_2 = \frac{N_2}{p}$, where any further movement takes us out of the feasible region. ∎

Substituting $x_1 = 1$ in Eqn. (10), we get a function of only x_2:

$$\widehat{T}(x_2) = \frac{N_1 N_2(\beta_r + \beta_s + 2\tau_c)}{p x_2} + (p-1)\tau_a x_2 + (p-1)(\beta_s + \tau_t + 2\tau_c) + \frac{N_1 N_2 \tau_a}{p} \quad (11)$$

Similarly, substituting $x_2 = \frac{N_2}{p}$ in (10) yields $\widetilde{T}(x_1)$, a function of only x_1,

$$\widetilde{T}(x_1) = \frac{N_1(\beta_r + \beta_s)}{x_1} + \frac{(p-1)}{p}(N_2\tau_a + p\tau_t + 2p\tau_c)x_1 + (p-1)\beta_s + \frac{N_1 N_2 \tau_a}{p} + 2N_1\tau_c \quad (12)$$

Hence, we separately minimize Eqn. (11) in the interval $1 \le x_2 \le \frac{N_2}{p}$ and Eqn. (12) in the interval $1 \le x_1 \le N_1$. The solution of **Prob. 1** is the smaller of the two.

We see that both $\widehat{T}(x_2)$ and $\widetilde{T}(x_1)$ have the form, $F(y) = \frac{A}{y} + By + C$, the sum of a hyperbolic term, a linear term and a constant. This is a classic, strongly convex function, which has a unique minimum (for positive y) at $y^* = \sqrt{A/B}$, whose value is $F(y^*) = C + 2\sqrt{AB}$. Moreover, the optimal solution of the *discrete* version occurs at one of the two integers $\lfloor y^* \rfloor$ or $\lceil y^* \rceil$.

For our problem, the value of x_2^* and x_1^* are given as follows.

$$x_2^* = \sqrt{\frac{N_1 N_2(\beta_r + \beta_s + 2\tau_c)}{(p-1)p\tau_a}} > \widehat{x_2} = \sqrt{\frac{N_1 N_2(\beta_r + \beta_s)}{(p-1)p\tau_a}} \quad (13)$$

$$x_1^* = \sqrt{\frac{pN_1(\beta_r + \beta_s)}{(p-1)(N_2\tau_a + p\tau_t + 2p\tau_c)}} \approx \widehat{x_1} = \sqrt{\frac{pN_1(\beta_r + \beta_s)}{(p-1)N_2\tau_a}} \quad (14)$$

The approximation in Eqn. 14 holds because N_1 and N_2, grow much faster than p, i.e., $N_2\tau_a \gg p\tau_t + 2p\tau_c$ asymptotically (but note that always, $x_1^* < \widehat{x_1}$). Let the solutions of the discrete versions of the respective optimization problems be x_1^o and x_2^o (either $\lfloor x_1^* \rfloor$ or $\lceil x_1^* \rceil$, and $\lfloor x_2^* \rfloor$ or $\lceil x_2^* \rceil$). Note that the feasible intervals of the two problems intersect at $[1, \frac{N_2}{p}]$, and that $\widehat{T}(\dfrac{N_2}{p}) = \widetilde{T}(1) = T_1(1, \dfrac{N_2}{p})$.

Now, consider whether x_2^* and x_1^* lie in the intervals, $1 \leq x_2 \leq \frac{N_2}{p}$, and $1 \leq x_1 \leq N_1$, respectively. Asymptotically, $x_2^* \geq 1$, and $x_1^* \leq N_1$ (since $N_1 N_2 \gg \frac{(p-1)p\tau_a}{p(\beta_r+\beta_s+2\tau_c)}$ and $\frac{p(\beta_r+\beta_s)}{(p-1)\tau_a}$). It is easy to see that $\widehat{x_2} \geq \frac{N_2}{p}$ iff $\widehat{x_1} \geq 1$. Then x_2^* is outside its feasible interval, and $x_2^o = \frac{N_2}{p}$. But this puts $[1, x_2^o]$, i.e., $[1, \frac{N_2}{p}]$ into the feasible interval of $\widetilde{T}(x_1)$, and the final solution is at $[x_1^o, \frac{N_2}{p}]$. Otherwise, $x_1^* < 1$, and the optimal solution is at $[1, x_2^o]$. Based on this discussion, we have the following result.

Theorem 2 The optimal tile size is unique and given by

$$\begin{cases} [x_1^o, \frac{N_2}{p}] & \text{if } pN_1(\beta_r + \beta_s) \geq (p-1)N_2\tau_a \\ [1, x_2^o] & \text{otherwise} \end{cases}$$

If x_2^o minimizes $\widehat{T}(x_2)$ in the interval $[1, \frac{N_2}{p}]$, and x_1^o minimizes $\widetilde{T}(x_1)$ in $[1, N_1]$.

We thus have a closed form expression, involving compile time parameters, that gives us the optimal tile size. The condition $pN_1(\beta_r + \beta_s) \geq (p-1)N_2\tau_a$ is simply a constraint on the aspect ratio of the iteration space, $\frac{N_1}{N_2}$, relative to a constant, $\alpha = \frac{(p-1)\tau_a}{p(\beta_r+\beta_s)}$. For certain aspect ratios, a single pass program with x_1^o rows in each tile is optimal, and for others, we should use a multi-pass program, with one only row per tile.

Comparison between the two projections

As mentioned before, the tile graph can yield two possible systolic arrays, corresponding to the projection $[1, 0]$, which we analyzed above, and along $[0, 1]$, which we now consider. First, observe (repeating the development of Section. 2) that for this projection, the running time is still given by Eqn. (6) with N_1 and N_2 interchanged, but that that the function $T_1(x_1, x_2)$ in region I is the same (albeit with a different boundary between the two regions). Nevertheless, Lemma 1 still holds and we will always be interested in the solution in region I. Now, when we instantiate the machine and program parameters, we see that P_t is the same except that x_1 and x_2 are interchanged (messages are now of length

x_2). This symmetry carries over to L_t, and the subsequent development of the running time, $T_1'(x_1, x_2)$, leading to the following observation of symmetry.

Remark 1 For the $[0,1]$ projection, the running time $T_1'(x_1, x_2)$ is the same as $T_1(x_2, x_1)$

We now have to minimize the function $T_1'(x_1, x_2)$ over the new feasible space defined by, $\left\{[x_1, x_2] \in \mathcal{Z}^2 \mid 1 \le x_1 \le \frac{N_1}{p},\ 1 \le x_2 \le N_2\right\}$. By interchanging the free variables, this is the same as minimizing $T_1(x_1, x_2)$ over $\mathcal{R}' = \left\{[x_1, x_2] \in \mathcal{Z}^2 \mid 1 \le x_1 \le N_2,\ 1 \le x_2 \le \frac{N_1}{p}\right\}$. Now, the "top-right" vertices of $\mathcal{R}$ and $\mathcal{R}'$ are, respectively, $[N_1, \frac{N_2}{p}]$ and $[N_2, \frac{N_1}{p}]$. They both lie on the hyperbola, $x_1 x_2 = \frac{N_1 N_2}{p}$. Assume, without loss of generality, that $N_1 \le N_2$, and consider the region, $\mathcal{R}'' = \left\{[x_1, x_2] \in \mathcal{Z}^2 \mid 1 \le x_1 \le N_2,\ 1 \le x_2 \le \frac{N_2}{p},\ x_1 x_2 \le \frac{N_1 N_2}{p}\right\}$, which includes both rectangles, $\mathcal{R}$ and $\mathcal{R}'$.

Using the same "hyperbolic" argument as in the proof of Theorem 1, we can now show that the minimum value of $T_1(x_1, x_2)$ over the entire region $\mathcal{R}''$ must lie in $\mathcal{R}$, and since $\mathcal{R}' \subseteq \mathcal{R}''$, this leads to the following strong result.

Theorem 3 The optimal direction of projection is always along the **smaller** of the sides of the original iteration space. This holds regardless of whether the ultimate optimal solution requires single or multiple passes.

5 Extensions

We now illustrate the advantages of our two-phase approach by showing how variations in code such as overlap of communication and computation can be easily modeled. Additional variations have been modeled and experimentally verified elsewhere [ABR96].

Consider a program (such as matrix vector multiplication) where the data circulated is not the *result* of each tile's computation, but one of it *inputs*. In this case, we can overlap communication and computation, i.e., the tile code is a `receive-send-compute` loop, rather than `receive-compute-send`. We see that that this reordering does not affect the tile period, which remains as in Eqn. 8, but the latency is now as follows.

$$L_t = (\beta_r + \tau_c x_1) + \beta_s + (\tau_c + \tau_t) x_1 - \beta_r \quad = \quad \beta_s + (2\tau_c + \tau_t) x_1 \tag{15}$$

Substituting in Eqn. (6) and simplifying, we obtain an equation identical to Eqn. 10 except that the term proportional to $x_1 x_2$ disappears. This is a strictly decreasing function of x_2 (regardless of x_1) and is minimized for the largest value of this variable which is $\frac{N_2}{p}$. Hence the single pass strategy is *always* optimal. Substituting $x_2 = \frac{N_2}{p}$, we then obtain

$$\widetilde{T'}(x_1) = \frac{N_1(\beta_r + \beta_s)}{x_1} + (p-1)(\tau_t + 2\tau_c)x_1 + (p-1)\beta_s 2N_1\tau_c + \frac{N_1 N_2 \tau_a}{p} \tag{16}$$

This is also a strongly convex function of the same form as $\widetilde{T}(x_1)$, and can be minimized using the same techniques. Hence we have the following result (the optimal tile size is now proportional to $\sqrt{N_1}$).

Theorem 4 The optimal tile size for overlapable computations is either $[\lfloor x^* \rfloor, \frac{N_2}{p}]$ or $[\lceil x^* \rceil, \frac{N_2}{p}]$, where $x^* = \sqrt{\dfrac{N_1(\beta_r + \beta_s)}{(p-1)(\tau_t + 2p\tau_c)}}$

Here too, we can show that the optimal direction of projection is along the shorter of N_1 and N_2.

6 Related Work

We discuss the related work by showing how other models in the literature are easily subsumed by our approach. We also summarize experimental results and other extensions reported elsewhere [ABR96].

Hiranandani et al. [HKT94] present one of the earliest results on incorporating grain size analysis into a parallelizing compiler. They use a simple model in which all communication costs are modeled by a fixed overhead, β (independent of the message volume). They also assume that the iteration space is square ($N_1 = N_2$). This corresponds to a particular case in our model where $\tau_c = \tau_t = \beta_r = 0$, and $\beta_s = \beta$. Then, Equation 14 becomes $x_1^* = \sqrt{\frac{pN_1\beta}{(p-1)N_2\tau_a}}$, which is precisely their result (extended to rectangular iteration spaces).

Palermo et al. [PSCB94] present a more refined model which accounts for the transmission rate but they do not make a distinction between time spent on the main processor from the time by the communication co-processor (i.e., they do not separate the τ_t and τ_c

terms). Their model is also more general in that they consider the possibility of an outer loop (their iteration space may have more than two dimensions, but they tile only the innermost two loops). They do not allow a multiple pass solution (for the innermost two loops). For the case when there are only two loops to pipeline, their running time can be simplified as follows.

$$T(x_1) = \frac{N_1\beta}{x_1} + \frac{p-1}{p}(N_2\tau_a + p\tau)x_1 + (p-2)\beta + \frac{N_1N_2\tau_a}{p} \tag{17}$$

Note that if we neglect one β term above, this is identical to Equation 12, if we substitute $\tau_c = \beta_r = 0$, $\beta_s = \beta$, and $\tau_t = \tau$. Hence their model is a particular case of ours, where the cost of copying data to/from system memory is ignored, and only the single pass case is considered.

Ohta et al. [OSKO95] do not *a priori* rule out multiple passes, and use a model similar to Palermo et al. Their formula for the running time is as given below

$$T(x_1, x_2) = \frac{N_1N_2\beta}{px_1x_2} + p\tau_a x_1 x_2 + p\tau x_1 + \frac{N_2N_1\tau}{px_2} + p\beta + \frac{N_2N_1t}{p} \tag{18}$$

It is easy to check that this is identical to Eqn. 10 if $p = p - 1$ (an approximation that they make), and if $\beta = \beta_s$, $\beta_r = 0$ $\tau = \frac{\tau_c}{2}$ and $\tau_t = 0$. However, although they claim to consider multiple passes, they do not check that $\frac{N_1}{x_1}P_t \geq pL_t$. In effect, they simply minimize $T_1(x_1, x_2)$ in the *entire* feasible space $\mathcal{R}$, and assume that the solution will be valid. It is fortuitous for them that the values of L_t and P_t based on actual machine and program parameters are such that the premise of Lemma 1, holds. Second, while solving the optimization problem, they state that if x_1x_2 is fixed, the function is monotonically decreasing with x_1 (the same "hyperbolic" argument we used in the proof of Theorem 1). However, they do not consider that in proceeding this way, it is possible to exit the feasible region $\mathcal{R}$ by the $x_1 = 1$ boundary. This leads them to conclude incorrectly that the single pass is *always* optimal, and they miss the multiple pass solution. As discussed below, it has been experimentally confirmed by us that such cases can occur.

Experimental Validation

We conducted a number of experiments to validate the model and also to compare our results with other models (see [ABR96] for details). All theoretical predictions were shown to hold. One important observation is that the predictions are very sensitive to β, the startup overhead (for send *and* receive calls) but are insensitive to τ_c and/or τ_t (on machines like the Intel paragon, which we used for our experiments). We also observe that it is difficult to measure these parameters accurately on the Intel paragon. The standard and widely used "ping-pong" experiments [Bok90] measure β_s and not β_r, and it is this, more than anything else that enables our model to make better predictions than the others. Indeed, without this, even the approximate model of Hiranandani et al. gave almost identical predictions as all the other models (including ours).

7 Conclusions

We addressed the problem of finding the tile size that minimizes the running time of SPMD programs. Our source programs have a two dimensional, parallelogram shaped iteration space. Their dependencies are such that tiles parallel to the domain boundaries are legal. Although apparently restrictive, such hypotheses are common in the literature.

We use a two-step approach to the problem. First we define and partially solve an "abstract" optimization problem using two characteristic functions: the tile period, P_t and the inter-tile latency, L_t. Next, we specify these two functions for a given machine and program, using a detailed model for message startup, transmission and copying costs. We then analytically solve the resulting discrete nonlinear optimization problem, yielding the desired solution. We proved that for certain values of the parameters the minimal running time could be obtained by multiple passes (this has been overlooked till now by other researchers), and showed which direction of projection (allocation of tiles to processors) yielded a lower running time. The power of our two-step approach, as illustrated by numerous examples, is that several architectural models and program variants can be easily taken into account by appropriately defining P_t and L_t. We also showed that our model subsumes those proposed by other authors.

Our experimental results (reported in [ABR96]) confirm the validity of our model, and highlight the importance of accurately measuring the machine parameters. They also indicate that on machines like the intel Paragon our model is "too accurate": predictions of a simpler model [HKT94] are indistinguishable from those of ours (and from all others discussed here). Nevertheless, the effort of going through the complete development of a more general model is useful, since the relative importance of the parameters is not known beforehand. It is only after carrying the analysis through to the end that one can decide unequivocally which parameters are important. Moreover, the model is also stable if the relative importance of the parameters changes. For example, the dominance of message startup time is valid for current generation machines, but on machines like the Intel iWarp, communication channels can be allocated at compile time, and there is *no* run-time overhead for starting messages.

There are a number of open questions. The first one is the direct extension to the non orthogonal case (when the tiles boundaries cannot be parallel to the domain boundaries). We have addressed this elsewhere and formulated a non-linear optimization problem [AR96], but a closed form solution is not available. It would also be interesting to extend the work to higher dimensional recurrences (as done for the 3-d case by Andonov et al. [AYB97]). Finally, since machines are evolving towards shared memory, it is more and more important to adapt the model for caches.

References

[ABR96] R. Andonov, H. Bourzoufi, and S. Rajopadhye. Two-dimensional orthogonal tiling: from theory to practice. In *International Conference on High Performance Computing*. IEEE, December 1996. (to appear).

[AR96] R. Andonov and S. Rajopadhye. Optimal Tiling of Two-Dimensional Uniform Recurrences. Technical Report PI-999, IRISA, Rennes, France, April 1996. This report supersedes IRISA PI-792, January 1994 ("Optimal Tiling"). Preliminary results presented at CONPAR '94.

[AYB97] R. Andonov, N. Yanev, and H. Bourzoufi. Three-dimensional orthogonal tile sizing problem: Mathematical programming approach. In *ASAP 97: International Conference on Application-Specific Systems, Architectures and Processors*, Zurich, Switzerland, July 1997. IEEE, Computer Society Press. (to appear).

[BDRR94] P. Boulet, A. Darte, T. Risset, and Y. Robert. (pen)-ultimate tiling? *INTEGRATION, the VLSI journal*, 17:33–51, Nov? 1994.

[Bok90] S. Bokhari. Communication overheads on the Intel iPSC-860 Hypercube. Technical Report Interim Report 10, NASA ICASE, May 1990.

[CDR97] P-Y. Calland, J. Dongarra, and Y. Robert. Tiling with limited resources. In *ASAP 97: IEEE Conference on Application Specific Systems Architectures and Processors*, page (to appear), Zurich, Switzerland, July 1997. IEEE.

[HKT94] S. Hiranandani, K. Kennedy, and C-W. Tseng. Evaluating compiler optimizations for Fortran D. *Journal Of Parallel and Distributed Computing*, 21:27–45, 1994.

[IT88] F. Irigoin and R. Triolet. Supernode partitioning. In *15th ACM Symposium on Principles of Programming Languages*, pages 319–328. ACM, Jan 1988.

[KCN90a] C-T. King, W-H. Chou, and L. Ni. Pipelined data-parallel algorithms: Part I–concept and modelling. *IEEE Transactions on Parallel and Distributed Systems*, 1(4):470–485, October 1990.

[KCN90b] C-T. King, W-H. Chou, and L. Ni. Pipelined data-parallel algorithms: Part II–design. *IEEE Transactions on Parallel and Distributed Systems*, 1(4):486–499, October 1990.

[KMW67] R. M. Karp, R. E. Miller, and S. V. Winograd. The organization of computations for uniform recurrence equations. *JACM*, 14(3):563–590, July 1967.

[OSKO95] H. Ohta, Y. Saito, M. Kainaga, and H. Ono. Optimal tile size adjsutment in compiling general DOACROSS loop nests. In *International Conference on Supercomputing*, pages 270–279, Barcelona, Spain, July 1995. ACM.

[PSCB94] D. Palermo, E. Su, J. Chandy, and P. Banerjee. Communication optimizations used in the PARADIGM compiler for distributed memory multicomputers. In *International Conference on Parallel Processing*, pages xx–yy, St. Charles, IL, August 1994. IEEE.

[QR89] P. Quinton and Y. Robert. *Systolic Algorithms and Architectures.* Prentice Hall–Masson, 1989.

[RS91] J. Ramanujam and P. Sadayappan. Tiling multidimensional iteration spaces for non shared-memory machines. In *Supercomputing 91*, pages 111–120, 1991.

[SD90] R. Schreiber and J. Dongarra. Automatic blocking of nested loops. Technical Report 90.38, RIACS, NASA Ames Research Center, Aug 1990.

[Wol87] M. J. Wolfe. Iteration space tiling for memory hierarchies. *Parallel Processing for Scientific Computing (SIAM)*, pages 357–361, 1987.

[ZRW92] Xiaoxiong Zhong, S. V. Rajopadhye, and Ivan Wong. Systematic generation of linear allocation functions in systolic array design. *Journal of VLSI Signal Processing*, 4(4):279–293, Nov 1992.