Optimal Orthogonal Tiling

Rumen Andonov[1], Sanjay Rajopadhye[2], and Nicola Yanev[3]

[1] LIMAV, University of Valenciennes, France, andonov@univ-valenciennes.fr
[2] Irisa, Rennes, France, Sanjay.Rajopadhye@irisa.fr
[3] University of Sofia, Bulgaria choby@math.acad.bg

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 *orthogonal tiling*—uniform dependency programs with (hyper) parallelepiped shaped iteration domains which can be tiled with hyperplanes parallel to the domain boundaries. Our formulation includes many machine and program models used in the literature, notably the BSP programming model. We resolve the optimization problem analytically, yielding a closed form solution.
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 [17, 7, 14] 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 as well as in performance tuning of parallel codes by hand (see also [6, 13, 10, 15]). A *tile* in the iteration space is a (hyper) parallelepiped shaped collection of iterations to be executed as a single unit, with the communication and synchronization being done only once per tile. Typically, communication are performed by send/receive calls and also serve as synchronization points. 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* [5], and *tile size optimization* [2, 6, 9, 13] (some authors also attempt to resolve both problems under some simplifying assumptions [14, 15]). 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. We assume that the dependencies are uniform, the iteration space (domain) is an n-dimensional hyper-rectangle, and the tile boundaries are parallel to the domain boundary (this is called *orthogonal tiling*). A sufficient condition for this that in all dependence vectors, all non-zero terms have the same sign. Whenever orthogonal tiling is possible, it leads to the simplest form of code; indeed most compilers do not even implement any other tiling strategy. We show here that when orthogonal tiling is possible the tile sizing problem is easily solvable even in its most general, n-dimensional case.

Our approach is based on the two step model proposed by Andonov and Rajopadhye for the 2-dimensional case in [2], and which was later extended to 3 dimensions in [3]. In this model we first abstract each tile by two simple parameters: tile period, $\mathcal{P}$ and inter-tile latency, $\mathcal{L}$. We then "instantiate" the abstract model by accounting for specific architectural and program features, and analytically solve the resulting non-linear optimization problem, yielding the desired tile size. In this paper we first extend and generalize our model to the case where an n-dimensional iteration space is implemented on a k-dimensional (hyper) toroid (for any $1 \leq k \leq n-1$). We also consider a more general form of the specific functions for $\mathcal{L}$ and $\mathcal{P}$. These functions are general enough to not only include a wide variety of machine and program models, but also be used with the BSP model [11, 16], which is gaining wide acceptance as a well founded theoretical model for developing architecture independent parallel programs.

Being an extension of previous results, the emphasis of this paper is on the new points in the mathematical framework and on the relations of our model to others proposed in the literature. More details about the definitions, notations and their interpretations, as well about some practical aspects of the experimental validations are available elsewhere [1–3].

The remainder of this paper is organized as follows. In the following section we develop the model and formulate the abstract optimization problem. In Section 3 we instantiate the model for specific machine and program parameters, and show how our model subsumes most of those used in the literature. In Section 4 we resolve the problem for the simpler HKT model which assumes that the communication cost is constant, independent of the message volume (and also a sub-case of the BSP model, where the network bandwidth is very high). Next, in Section 5 we resolve the more general optimization problem. We present our conclusions in Section 6.

2 Abstract Model Building

We first develop an analytical performance model for the running time of the tiled program. We introduce the notation required as we go along. The original iteration space is an $N_1 \times N_2 \times \ldots \times N_n$ hyper-rectangle, and it has (at least n linearly independent) dependency vectors, $d_1, \ldots d_m$. The nonzero elements of the dependency vectors all have the same sign (say positive). Hence, orthogonal tiling is possible (i.e., does not induce any cyclic dependencies between the tiles). Let the tiles be $x_1 \times x_2 \times \ldots \times x_n$ hyper-rectangles, and let $q_i = \frac{N_i}{x_i}$ be the number of tiles in the i-th dimension. The *tile graph* is the graph where each node represents a tile and each arc represents a dependency between tiles, and can be modeled by a uniform recurrence equation [8] over an $q_1 \times q_2 \times \ldots \times q_n$ hyper-rectangle. It is well known that if the x_i's are large as compared to the elements of the dependency vectors[1], then the dependencies between the tiles are *unit vectors* (or binary linear combinations thereof, which can be neglected for analysis purposes without any loss of generality). A tile can be identified by an index vector $\mathbf{z} = [z_1, \ldots, z_n]$. Two tiles, $\mathbf{z}$ and $\mathbf{z}'$, are said to be *successive*, if $\mathbf{z} - \mathbf{z}'$ is a unit vector. A simple analysis [7] shows that the earliest "time instant" (counting a tile execution as one time unit) that tile $\mathbf{z}$ can be executed is $t_z = z_1 + \ldots + z_n$.

We map the tiles to $p_1 \times p_2 \times \ldots \times p_k$ processors, arranged in a k dimensional hyper-toroid[2] (for $k < n$). The mapping is by projection onto the subspace spanned by k of the n canonical axes. To visualize this mapping, first consider the case when $k = n - 1$, where tiles are allocated to processors by projection along (without loss of generality) the n-th dimension, i.e., tile $\mathbf{z}$ is executed by processor $[z_1, \ldots, z_{n-1}]$. This yields a (virtual) array of $q_1 \times q_2 \times \ldots \times q_{n-1}$ processors, each one executing all the tiles in the n-th dimension. In general $p_i \leq q_i$, so this array is emulated by our $p_1 \times p_2 \times \ldots \times p_{n-1}$ hyper-toroid by using multiple passes (there are $\frac{q_i}{p_i}$ passes in the i-th dimension). The case when $k < n - 1$ is modeled by simply letting $p_{k+1} = \ldots = p_{n-1} = 1$.

The *period*, $\mathcal{P}$ of a tile is defined as the time between executions of corresponding instructions in two successive tiles (of the same pass) mapped to the same processor. The *latency*, $\mathcal{L}$ between tiles is defined to be the time between executions of corresponding instructions in two successive tiles mapped to *different* processors. Depending on the volume of the data transmitted and the nature of the program dependencies, the latency may be different for different dimensions of the hyper-toroid, and we use $\mathcal{L}_i$ (for $i = 1 \ldots k$) to denote the latency in the i-th dimension.

We assume that by the time the first processor, $\mathbf{1} = [1 \ldots 1]$, finishes computing its macro column, at least one of the "last" processors (i.e., one of $[1, \ldots, p_i, \ldots, 1]$, for $1 \leq i \leq k$) has finished its first tile[3]. In this case, processor $\mathbf{1}$ can *immediately* start another pass. Let $W = \prod_{i=1}^{n} N_i$ denote the total computation volume, $v = \prod_{i=1}^{n} x_i$ be the tile volume and $P = \prod_{i=1}^{n} p_i$ be the total number of processors. The total number of tiles is $\prod_{i=1}^{n} q_i = \frac{W}{v}$. Let $\widetilde{p_i}$ denote $p_i - 1$, $\widetilde{P_k} = \sum_{i=1}^{k} \widetilde{p_i}$ and $v_{\max} = (\prod_{i=1}^{k} N_i)/P$.

Let us first analyze a single pass. Each processor must execute q_n tiles, and this takes $q_n \mathcal{P}$ time. However, the last processor (i.e., the one with coordinates $[p_1, p_2, \ldots p_k]$) cannot start because of the dependencies of the tile graph. Indeed, processor $[p_1, 1 \ldots, 1]$ can only start at time $(p_1 - 1)\mathcal{L}_1$, processor $[p_1, p_2 \ldots, 1]$,

---

[1] In general this implies that the feasible value of each x_i is bounded from below by some constant. For the sake of clarity, we assume that this is 1.

[2] This is just an abstract machine architecture. Our machine model is independent of the topology, since we will later assume that communication time is independent of distance and/or contention.

[3] There is no loss of generality in this assumption. Were it not true, the processors would be idle between passes, and this would lead to a sub-optimal solution. this has been proved for 2 and 3 dimensions [2, 3] and the arguments carry over.

at time $(p_1-1)\mathcal{L}_1+(p_2-1)\mathcal{L}_2$, and hence processor $[p_1, p_2 \ldots, p_k]$ can only start its first pass at time $\sum_{i=1}^{k} \widetilde{p}_i \mathcal{L}_i$.

There are $\frac{W}{v}$ tiles, of which Pq_n are executed in each pass, and hence there are $\frac{W/v}{Pq_n}$ passes. Because there is no idle time between passes, the last pass starts at time $\left(\frac{W}{Pvq_n}-1\right)\mathcal{P}q_n$. Thus, the *last* processor starts executing its *last* macro column at time instant $\left(\dfrac{W}{Pvq_n}-1\right)\mathcal{P}q_n+\displaystyle\sum_{i=1}^{k}\widetilde{p}_i\mathcal{L}_i$. It takes another $\mathcal{P}q_n$ time, and hence the total running time and the corresponding optimization problem are as follows.

$$T(x_1, \ldots x_n) = \frac{W\mathcal{P}}{Pv} + \sum_{i=1}^{k} \widetilde{p}_i \mathcal{L}_i \tag{1}$$

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

$$\mathcal{R} = \{[x_1 \ldots x_n] \in \mathcal{Z}^n \mid 1 \le x_i \le u_i\} \tag{2}$$

where $u_i = \frac{N_i}{p_i}$ for $i = 1 \ldots k$, and $u_i = N_i$ for $k < i \le n$.

3 Machine and Program Specific Model

We now “instantiate” **Prob. 1** for a specific program and machine architecture. The code executed for a tile is the standard loop:

```
repeat
  receive(v1); receive(v2), ...,receive(vk) ;
  compute(body);
  send(v1); send(v2), ..., send(vk) ;
end
```

where `vi` denotes the message transmitted in the i-th dimension. We will now determine $\mathcal{P}$ and $\mathcal{L}_i$. Our development is based on Andonov & Rajopadhye [1], and uses standard assumptions about low level behavior of the architecture and program [4]. The sole difference is that each tile now makes k systems calls to send (and receive) messages. A tile depends directly on its n neighbors in *each* dimension. The volume of data transfer along the i-th dimension is proportional to the (hyper) surface of the tile in that dimension, i.e., $\prod_{j \ne i} x_j = v/x_i$. In the first k dimensions, this corresponds to an inter-processor communication, whereas in the dimensions $k+1 \ldots n$, the transfer is achieved through local memory. Hence the period of a tile can be written as follows.

$$\mathcal{P} = k(\beta_r + \beta_s) + \left(2\tau_c v \sum_{i=1}^{k} \frac{1}{x_i}\right) + \alpha v \tag{3}$$

Here β_s (resp. β_r) is the overhead of the `send` (resp. `receive`) system call, τ_c is the time (per byte) to copy from user to system memory, α is the computation time for a single instance of the loop body (we neglect the overhead of setting up the loop for each tile, as well as cache effects). Similarly, if $1/\tau_t$ is the network bandwidth, the latency is as follows (see [2] for details).

$$\begin{aligned} \mathcal{L}_i &= \mathcal{P} + \tau_t \frac{v}{x_i} - k\beta_r \\ &= k\beta_s + \left(2\tau_c v \sum_{i=1}^{k} \frac{1}{x_i}\right) + \alpha v + \tau_t \frac{v}{x_i} \end{aligned} \tag{4}$$

Note that the β_r's are 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. (1) and simplifying, we obtain

$$\begin{aligned} T_k(\boldsymbol{x}) = \alpha v \widetilde{P_k} + 2\tau_c \widetilde{P_k} v \sum_{i=1}^{k} \frac{1}{x_i} + \tau_t v \sum_{i=1}^{k} \frac{\widetilde{p_i}}{x_i} + \frac{2\tau_c W}{P} \sum_{i=1}^{k} \frac{1}{x_i} + \\ + \ k(\beta_r + \beta_s) \frac{W}{Pv} + k\beta_s \widetilde{P_k} + \frac{\alpha W}{P} \end{aligned} \tag{5}$$

3.1 Simplifying assumptions and particular cases

The model of (5) is very general. One may want to specialize it for a number of reasons—say rendering the final optimization problem more tractable, or modeling a certain class of architectures or computations. It turns out that many of these simply consist of choosing the parameters appropriately in the above function.

The HKT model, first used by Hiranandani et al. [6], corresponds to setting $\beta_r = \tau_c = \tau_t = 0$ (a slightly more general version consists of letting β_r be nonzero). This model assumes that communication cost is independent of the message size, but is dominated by the startup time(s) β_s (and β_r).

At first sight this may seem an oversimplification. However, in addition to making the mathematical problem more tractable, it is not far from the truth, as corroborated by other authors [12, 13]. Indeed, experimental as well as analytic evidence [1] shows that on machines such as the Intel Paragon the more accurate models yield *no observable difference* in the predictions. With $\tau_c = \tau_t = 0$, we obtain the ***HKT cost function***:

$$T_k(\boldsymbol{x}) = \frac{\alpha W}{P} + k(\beta_r + \beta_s) \frac{W}{Pv} + (\alpha v + k\beta_s) \widetilde{P_k} \tag{6}$$

The BSP model [11, 16] has been proposed as a formal model for developing architecture independent parallel programs. It is a bridge between the PRAM model which is general but somewhat unrealistic, and machine-specific models which lead to lack of portability and predictability of performance. Essentially, the computation is described in terms of a sequence of "super-steps" executed in parallel by all the processors. A super-step consists of (i) some local computation, (ii) some communication events *launched* during the super-step, and (iii) a synchronization which ensures that the communication events are *completed*. The time for a super-step is the sum of the times for each of the above activities. This is very similar to our tile model: indeed, if we simply set $\tau_t = \beta_r = 0$ and $\beta_s = \frac{\beta}{k}$ (β is the BSP synchronization cost) we obtain the running time of the program under the BSP model. With this simplification, we obtain the ***BSP cost function*** as follows.

$$T_k(\boldsymbol{x}) = \frac{\alpha W}{P} + \frac{\beta W}{Pv} + (\alpha v + \beta)\widetilde{P_k} + 2\tau_c \left(\widetilde{P_k} v + \frac{W}{P}\right) \sum_{i=1}^{k} \frac{1}{x_i} \tag{7}$$

In the BSP model, the communication startup cost is replaced by the synchronization cost. However, in our general cost function (5) we incur the startup cost k times. As a result, if we take a particular case of the BSP model where the network bandwidth is extremely high (the communication time is dominated by the synchronization cost), then this ***high bandwidth BSP cost function*** is given as follows (it is similar but not identical to the HKT model).

$$T_k(\boldsymbol{x}) = \frac{\alpha W}{P} + \frac{\beta W}{Pv} + (\alpha v + \beta)\widetilde{P_k} \tag{8}$$

With some other simple modifications, our cost function can also model the overlap of communication and computation, which is often used as a performance tuning strategy. This is not detailed here due to space constraints.

4 Solution for HKT and high bandwidth BSP models

In this section, we will focus only on the simple models (6, 8). Our main results are that the optimal tile volume and the dimension of optimal virtual architecture can be determined as a closed form solution. These results serve two important purposes, in spite of the apparent simplicity of the model. First, they are valid for a number of current machines where the communication latency and network bandwidth are both relatively high (such as the Intel Paragon, and a number of similar machines as well as networks of workstations). Second, they give a good indication of our solution method for the more general results. We first solve the problem for the HKT model, and then show how the high bandwidth BSP model follows almost directly.

4.1 Optimal tile volume

It is easy to see that the running time (6) depends only on the *tile volume* and is a strictly convex function of the form $T_k(v) = \frac{A}{v} + Bv + C$, which attains its optimal value of $C + 2\sqrt{AB}$ at $\widetilde{v} = \sqrt{\frac{A}{B}}$. By substituting we obtain

$$\widetilde{v} = \sqrt{\frac{k(\beta_r + \beta_s)W}{P\alpha\widetilde{P_k}}} \tag{9}$$

$$\widetilde{T_k} = \frac{\alpha W}{P} + k\beta_s \widetilde{P_k} + 2\sqrt{\frac{k\alpha(\beta_r + \beta_s)W\widetilde{P_k}}{P}} \tag{10}$$

The optimal solution will be as given above if $\widetilde{v}$ is a feasible tile volume. Now, observe that each x_i is bounded from above by $\frac{N_i}{p_i}$, and hence $v \leq \frac{W}{P}$. Since W grows much faster than P, we have $1 \leq \widetilde{v} \leq \frac{W}{P}$ asymptotically. Hence we have the following result.

Theorem 1. *The optimal tile volume and the corresponding running time for the* HKT *model are given by (9-10).*

4.2 Optimal architecture

So far, we have assumed that k is a fixed constant as are each of the p_i's. In practice, we typically have P processors, and the values of each p_i are not specified, and neither is k, and we now solve this problem. Note that the dominant term in (10) is $\frac{\alpha W}{P}$, the "ideal" parallel time with no overhead. The ideal architecture will seek to minimize the overhead whose dominant term is $O(\sqrt{\frac{W}{P}})$.

From (10) it can be easily deduced that for a given k and P, the optimal architecture is the one that minimizes $\widetilde{P_k}$, i.e., the torus with $p_i = \sqrt[k]{P}$. Substituting this in (10), the coefficient of the overhead term is proportional to the square root of the function $f(k, P) = k^2 P^{\frac{1}{k}} - k^2$. Thus $\widetilde{T_k}$ is minimized for the value of k that minimizes $f(k, P)$, i.e., $k^* = 0.625 \ln P$. Since $f(k, P)$ is monotonically decreasing up to k^* (for each P) and monotonically increasing thereafter, we have the following result.

Corollary 1. *If $n \leq \lceil k^* \rceil$ the optimal architecture is a balanced $n-1$ dimensional torus, and for $n > \lceil k^* \rceil$ it is either a balanced $\lceil k^* \rceil$ or a $\lfloor k^* \rfloor$ dimensional torus depending respectively on whether $f(\lceil k^* \rceil, P)$ is smaller than $f(\lfloor k^* \rfloor, P)$ or not.*

It is thus clear that as the number of processors grows, the optimal architecture tends towards an $n-1$ dimensional hyper-torus. However, k^* grows logarithmically with P, and for a limited number of processors (most practical cases), the optimal may not be $n-1$ dimensional. Indeed, the sensitivity of k^* with respect to number of processors P is illustrated by the following: for up to 25 processors, the optimal architecture is a linear array, from 25–130 it is 2-dimensional, for 130–650 processors it is a 3-dimensional, etc.

The extension of these results to the high bandwidth BSP model (Eqn 8) is straightforward, and indeed the mathematical treatment is a little simpler. The solution is the same as that given by (9-10), but with k, β_r and β_s respectively equal to $1, 0$ and β. The sole subtle difference is that the function $f(k, P)$ is $kP^{1/k} - k$, and hence $k^* = +\infty$, i.e., $f(k, P)$ is monotonically decreasing. Hence, the optimal architecture is *always* a balanced $n-1$ dimensional torus.

Finally, note that the optimization of the processor architecture yields a second order improvement—it does not affect the dominant term $\frac{\alpha W}{P}$.

5 Solution for the BSP cost function

We now solve our optimization problem for the BSP cost function (7) in the feasible space specified by (2). We will show that our problem can be decomposed into two special cases. The first case is very similar to the HKT model, but the second one is more complicated, for which we first solve the corresponding *unconstrained optimization problem* and then determine where the constrained solution lies. Finally, we show that the second solution is globally optimal asymptotically.

Lemma 1. *The minimum of (7) over conv($\mathcal{R}$) is attained at:*

$$\begin{array}{ll} \textit{either} & x_i = \frac{N_i}{p_i},\ \textit{for } i = 1 \ldots k \\ \textit{or} & x_i = 1,\ \textit{for } i = k+1 \ldots n \end{array}$$

Proof. Let v be an arbitrary feasible volume and let there exist two indices l, m for $l \le k$, $k+1 \le m \le n$ such that $x_l < N_l/p_l$ and $x_m > 1$. Then by increasing x_l, decreasing x_m and keeping their product constant, the function (7) strictly decreases and we obtain the needed. ∎

Based on this, we have to look for the solution in the two regions of $\mathcal{R}$ corresponding to the above two conditions. Let $\mathcal{R}_1 = \mathcal{R} \cap x_i = \frac{N_i}{p_i}$ for $i = 1 \ldots k$, and $\mathcal{R}_2 = \mathcal{R} \cap x_i = 1$ for $i = k+1 \ldots n$.

5.1 Case I

The cost function in region $\mathcal{R}_1$ can be simplified to

$$T_k(\boldsymbol{x}) = (\alpha + 2\tau_c \widetilde{N_k})\frac{W}{P} + \beta\widetilde{P_k} + \frac{\beta W}{Pv} + (\alpha + 2\tau_c\widetilde{N_k})\widetilde{P_k}v \qquad (11)$$

where $\widetilde{N_k} = \sum_{i=1}^{k} \frac{p_i}{N_i}$. We can now use the same reasoning as in Section 4, and obtain the optimal volume and running time as follows:

$$\widetilde{v} = \sqrt{\frac{\beta W}{P\omega}} \quad \text{and} \quad \widetilde{T_k} = (\alpha + 2\tau_c\widetilde{N_k})\frac{W}{P} + \beta\widetilde{P_k} + 2\sqrt{\frac{\beta W\omega}{P}} \qquad (12)$$

where $\omega = (\alpha + 2\tau_c\widetilde{N_k})\widetilde{P_k}$. As before, appropriate values for $x_{k+1} \ldots x_n$ in $\mathcal{R}_1$ that yield the optimal volume are all equivalent. Observe that in (12) we have a factor $2\tau_c\widetilde{N_k}$ in the dominant term, and this turns out to be important as we shall see later.

5.2 Case II

Let us now consider region $\mathcal{R}_2$. Here, $v = \prod_{i=1}^{k} x_i$, but the cost function remains the same as (7), except that we have only k variables to solve for. We obtain the solution in two steps.

Unconstrained optimization We first solve the problem in the entire positive orthant, without any constraints. This can be formulated as follows.
Prob. 2: Minimize (7) in the feasible space $\mathcal{R}_+^k = \left\{[x_1 \ldots x_k]^{\mathrm{T}} \mid x_i \geq 0\right\}$.

Let H_v be the hyperboloid defined by $\{\boldsymbol{x} \in \mathcal{R}_+^k \mid \prod_{i=1}^k x_i = v\}$. Observe that the set of families H_v is a partition of $\mathcal{R}_+^k$, i.e., $\mathcal{R}_+^k = \bigcup_v H_v$, and hence $\min_{R_+^k} T_k(\boldsymbol{x}) = \min_v \min_{H_v} T_k(\boldsymbol{x})$. Thus, we first minimize (7) for a given tile volume (i.e., over a given hyperboloid H_v) and then choose the volume. Now, observe that for a fixed v, (7) is of the form $A + B\sum_i \frac{1}{x_i}$ whose minimum is attained at $\boldsymbol{x} \in H_v, x_1 = x_2 = \ldots = x_k$. Hence, for a given tile volume, v, the optimal tile shape is (hyper) cubic with $x_i = \sqrt[k]{v}$, for each $i = 1 \ldots k$. Thus, $\sum_{i=1}^{k} \frac{1}{x_i} = \frac{k}{\sqrt[k]{v}}$, and we can define,

$$f(v) = \min_{H_v} T(\boldsymbol{x}) = \frac{A}{v} + Bv + 2k\tau_c\widetilde{P_k}v^{1-\frac{1}{k}} + kDv^{-\frac{1}{k}} + \frac{\alpha W}{P} + \beta\widetilde{P_k} \qquad (13)$$

where $A = \frac{\beta W}{P}$, $B = \alpha\widetilde{P_k}$, and $D = \frac{2\tau_c W}{P}$. Now we have to determine the optimal tile volume by minimizing $f(v)$ in the feasible space, $1 \leq v \leq v_{\max}$. It can easily be shown that $f(v)$ is an unimodal function and attains its minimum at the root of $f'(v) = 0$. Unfortunately this is not easily obtained in closed form but it is quite well approximated by the root of the function $h(v) = \frac{-A}{v^2} + B + 2\tau_c(k-1)\widetilde{P_k} - \frac{D}{v^2}$, whose zero is at $\sqrt{\frac{A+D}{2\tau_c(k-1)\widetilde{P_k}+B}}$. Thus, we obtain the following (approximate) optimal solution of the unconstrained problem.

$$\widetilde{v} \approx \sqrt{\frac{\gamma W}{P}} \text{ and } \widetilde{T_k} \approx \frac{\alpha W}{P} + 2k\tau_c\left(\frac{1}{\gamma}\right)^{\frac{1}{2k}}\left(\frac{W}{P}\right)^{\frac{2k-1}{2k}} + O\left(\sqrt{\frac{W}{P}}\right) \qquad (14)$$

where $\gamma = (\beta+2\tau_c)/((2\tau_c(k-1)+\alpha)\widetilde{P_k})$. Of course we could always determine an exact solution if needed, but we have found the approximate solution to be reasonable in practice. Moreover, as we shall see later, it illustrates some interesting points. Also recall that in the problem as we have resolved so far, we assume that k and the p_i's are fixed, as also the choice of which of the N_i's to map to the processor space.

Constrained Optimization We now address the question of the restrictions on the optimal solution imposed by the feasibility constraints (2) namely $1 \leq x_i \leq u_i$. Note that the unconstrained (global) optimal solution is on the intersection of the line defined by the vector $\mathbf{1}$, and the hyperboloid $H_{\widetilde{v}}$, where $\widetilde{v}$ is the optimal tile volume (14). If this intersection is outside the feasible space (2) we will need to solve the constrained optimization problem. We have observed that the optimal running time is extremely sensitive to the tile volume, and much less dependent on the particular values of x_i (indeed this is predicted by the HKT model). We have the following cases

Case A: $\widetilde{v} > v_{\max}$ In this case, the optimal volume hyperboloid does not intersect the feasible space, and according to the properties of the function $f(v)$ the optimal tile size is given by $x_i = \frac{N_i}{p_i}$, for $i = 1 \ldots k$. However, note that since $\widetilde{v} = O(\sqrt{W/P})$ while $v_{\max} = O(W/P)$, this case is unlikely.

Case B: $\widetilde{v} \leq v_{\max}$ Now, $H_{\widetilde{v}}$ has a non-empty intersection with (2) and so our heuristic is to choose a solution by moving one of the x_i's such that we move within the feasible region.

5.3 Where is the global optimum?

The optimal solutions for the two cases are given, respectively by (12) for region $\mathcal{R}_1$ and by (14) for region $\mathcal{R}_2$. Most authors [6, 13, 12] have only considered region $\mathcal{R}_1$ either implicitly by not posing the problem in full generality, or by erroneously claiming that the solution is always in $\mathcal{R}_1$. This is incorrect because the final solution will always asymptotically be in $\mathcal{R}_2$, due to the additional factor, $2\tau_c\widetilde{N_i}$ in the dominant term in (12) as compared to (14). The following simple example illustrates the prediction of lemma 1: for $n = 3, k = 2, p_1 = p_2 = 4, \alpha = \beta = \tau_c = 10^{-6}, N_1 = 50, N_2 = N_3 = 1000$, the optimal time in $\mathcal{R}_1$ is 1.8 while that in $\mathcal{R}_2$ is 1.5. More illustrative instances with real data can be found in [3].

6 Conclusions

We addressed the problem of finding the tile size that minimizes the running time of SPMD programs. We formulated a discrete non-linear optimization problem using first an abstract model and then specific machine model. The resulting cost function is general enough to subsume most of those in the literature, including the BSP model. We then analytically solved the resulting discrete nonlinear optimization problem, yielding the desired solution.

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 (for the 2-dimensional case) and formulated a non-linear optimization problem [2], but a closed form solution is not available. Finally, experimental validation on a number of target machines is the subject of our ongoing work.

References

1. R. Andonov, H. Bourzoufi, and S. Rajopadhye. Two-dimensional orthogonal tiling: from theory to practice. In *International Conference on High Performance Computing*, pages 225–231, Tiruvananthapuram, India, December 1996. IEEE.
2. R. Andonov and S. Rajopadhye. Optimal orthogonal tiling of 2-d iterations. *Journal of Parallel and Distributed Computing*, 45(2):159–165, September 1997.

3. 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*, pages 209–218, Zurich, Switzerland, July 1997. IEEE, Computer Society Press.
4. S. Bokhari. Communication overheads on the Intel iPSC-860 Hypercube. Technical Report Interim Report 10, NASA ICASE, May 1990.
5. P. Boulet, A. Darte, T. Risset, and Y. Robert. (pen)-ultimate tiling? *INTEGRATION, the VLSI journal*, 17:33–51, Nov? 1994.
6. S. Hiranandani, K. Kennedy, and C-W. Tseng. Evaluating compiler optimizations for Fortran D. *Journal Of Parallel and Distributed Computing*, 21:27–45, 1994.
7. F. Irigoin and R. Triolet. Supernode partitioning. In *15th ACM Symposium on Principles of Programming Languages*, pages 319–328. ACM, Jan 1988.
8. 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.
9. 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.
10. 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.
11. W. F. McColl. Scalable computing. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000, pages 46–61. Springer Verlag, 1995.
12. 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.
13. 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.
14. J. Ramanujam and P. Sadayappan. Tiling multidimensional iteration spaces for non shared-memory machines. In *Supercomputing 91*, pages 111–120, 1991.
15. R. Schreiber and J. Dongarra. Automatic blocking of nested loops. Technical Report 90.38, RIACS, NASA Ames Research Center, Aug 1990.
16. L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.
17. M. J. Wolfe. Iteration space tiling for memory hierarchies. *Parallel Processing for Scientific Computing (SIAM)*, pages 357–361, 1987.