PLISKA

STUDIA MATHEMATICA BULGARICA

ПЛИСКА

БЪЛГАРСКИ МАТЕМАТИЧЕСКИ СТУДИИ

Pliska Stud. Math. Bulgar. **12** (1998), 5-20

PLISKA
STUDIA MATHEMATICA
BULGARICA

n –DIMENSIONAL ORTHOGONAL TILE SIZING PROBLEM

Rumen Andonov, Nicola Yanev

We discuss in this paper the problem of generating highly efficient code when a $n+1$-dimensional nested loop program is executed on a n-dimensional torus/grid of distributed-memory general-purpose machines. We focus on a class of uniform recurrences with non-negative components of the dependency matrix. Using tiling the iteration space strategy we show that minimizing the total running time reduces to solving a non-trivial non-linear integer optimization problem. For the later we present a mathematical framework that enables us to derive an $O(n \log n)$ algorithm for finding a good approximate solution. The theoretical evaluations and the experimental results show that the obtained solution approximates the original minimum sufficiently well in the context of the considered problem. Such algorithm is real-time usable for very large values of n and can be used as optimization techniques in parallelizing compilers as well as in performance tuning of parallel codes by hand.

Keywords: coarse grain pipelining, dynamic programming, uniform recurrence equations, parallelizing compilers, integer non-linear optimization

AMS subject classification: 68Q22, 90C90

1 Introduction

A common approach to generate highly efficient parallel code when a nested loop program is executed in SPMD (Single Program Multiple Data) fashion on DMM (distributed memory multicomputers) is the iteration space tiling (also called "super-node partitioning") [1, 12, 20, 25]. It may be used as a technique in parallelizing compilers (see [11, 17]) where it is called coarse-grain pipelining) as well as in performance tuning of parallel codes by hand [14, 21, 22]. A *canonical tile* is an n-dimensional parallelepiped box in an n-dimensional iteration space, and denotes a block of computations that are performed atomically. *Tiling* refers to the process of paving the entire iteration space with translated copies of the canonical tile. The *tiling problem* can be broadly defined as the problem of

choosing the tile parameters (notably the tile shape and size) in an optimal manner. It may be decomposed into two subproblems: choosing a "good" tile shape [8], and finding the "best" tile size [15, 21]. For the former problem, the communication cost is approximated by the number of dependency vectors crossing a tile border. The latter problem assume the tile shape is first given and then seeks to minimize the total execution time. Successfully solving the both problems guarantees choosing the optimal task granularity that balances the available parallelism with the cost of communication. Exact, but time expensive solutions are not of significant practical interest taking in account the context of their use. In general, both problems are hard optimization problems and researchers use different heuristics to solve them. The emphasis of this study is on finding "best" tile size and we show that even in the simplest case this implies solving a non-trivial non-linear integer optimization problem.

We address the tile sizing problem in the case of $n+1$-dimensional perfect nested loop nest with constant loop bounds [6] and uniform recurrence equations (UREs) [13] with non-negative dependencies. More precisely, the problem is to compute, for all $\mathbf{j} = [j_1, j_2, \ldots, j_{n+1}]^T \in Dom \subset Z_+^{n+1}$, a function, $Y[\mathbf{j}]$ given by

$$Y[\mathbf{j}] = f(Y[\mathbf{j} - \mathbf{d_1}], Y[\mathbf{j} - \mathbf{d_2}], \ldots, Y[\mathbf{j} - \mathbf{d_l}]) \tag{1}$$

with appropriate boundary conditions in a domain $Dom = \{\mathbf{j} : 0 < j_1 \leq m_1, 0 < j_2 \leq m_2, \ldots, 0 < j_{n+1} \leq m_{n+1}\}$, also called the iteration space. We call $\mathbf{D} = [\mathbf{d_1}, \mathbf{d_2}, \ldots, \mathbf{d_l}]$ the dependency matrix of (1). We assume $\mathbf{D}$ to be a full rank matrix of size $(n+1) \times l$ where $l \geq n+1$. f is a single-valued function, computed at point $\mathbf{j}$ in a single unit of time and $Y[\mathbf{j}]$ is its value in this point. In addition to these common assumptions we suppose $\mathbf{D}$ to have *non-negative* components, i.e. $\mathbf{D} \geq 0$. Typical example for such recurrences can be found in dynamic programming algorithms for solving optimization problems as finding the edit distance between strings [23].

The above assumptions imply that the iteration space is a parallelepiped and that the dependencies admit *orthogonal tiling* (i.e. they are such that tiles whose boundaries are parallel to the domain boundary are valid). We assume in this case a rectangular parallepiped for tile shape and we focus only on the tile sizing problem. The goal is for any instance of the domain *Dom*, to determine in a reasonably acceptable time the optimal tile size and to generate in this way a highly efficient SPMD code for executing (1) on a virtual n-dimensional torus of $p_1 \times p_2 \times \ldots \times p_n$ general-purpose processors.

The two-dimensional version of (1) has been recently addressed [4] where the authors also show how the methods of systolic array synthesis can be used for optimal tiling. The main observation is that independent of the original dependencies, the tiled computation for *all* programs in the considered class can be abstracted by a single, specific URE. An analysis of this URE using a realistic model of the communication cost yields a non-linear integer minimization problem having the total running time as objective function and where *all the available resources* (size of iteration space as well size of the processors array) appear in the problem constraints. The results in [4, 5] give a closed form formula for the optimal tile size in the two and three-dimensional version of (1). Here we give an $O(n \log n)$ algorithm for solving the n-dimensional case and show in this way that the tile

sizing problem is easily solvable when orthogonal tiling is possible. The proposed model has been experimentally validated in 2 and 3-dimensional cases in [3, 5]. Let us point that each one of these generalizations is *not* a straightforward extension of our previous low-dimensional results but requires a development of a *new* technique for its solution. This is due to the hardness of underlying minimization problem which is complicated by the non-convexity of its continuous relaxation. This problem could be harder even in the two-dimensional case (as shown in [4] for the Bitz-Kung path planning algorithm [7] whose dependency vectors have *negative* components). The existence of elegant and fast tile sizing algorithms for larger class of recurrences is still an open question.

Let us mention again that the emphasis of this paper, being an extension of previous results is on the new points in the mathematical framework that enable deriving the algorithm for the n-dimensional case. More details about the used definitions, notations and their interpretations, as well about some practical aspects of the experimental validations the interested reader can find in [4, 3].

The remainder of the paper is organized as follows. In the following section, we obtain analytic formula for the running time of a tiled program on a torus by combining the systolic synthesis design methodology with a realistic message passing communication model for general purpose DMM. We solve the associated minimization problem when the matrix $\mathbf{D}$ is the identity matrix $\mathbf{I}$ in section 3. We present experimental results in section 4 and we conclude in section 5. For the lack of space we shorten the explanations and we omit the proofs of few lemmata which are easy or very similar to the 3-dimensional case.

2 Orthogonal tiling in a rectangular parallelepiped

2.1 Analysis of the general case for a dependency matrix with non-negative components

Let us denote $N = \{1, 2, \ldots, n\}$, $\overline{N} = \{N \bigcup \{n+1\}\}$ and let us first suppose $\mathbf{D}$ to be a diagonal matrix with integer components $d_{ii}, i \in \overline{N}$ such that $d_{ii} \geq 1$. Under these assumptions (1) is a perfect loop nest of depth $n+1$ whose dependencies are positive multiples of the orthogonal basis vectors $\{\mathbf{e_i}\}_{i=1}^{n+1}$ and whose iteration space is a $n+1$ dimensional rectangular parallelepiped *Dom*. We are interested in executing Eqn. (1) on a n-dimensional grid of general-purpose processors. Following the approach from [4] applied in the $n+1$ dimensional space, we perform the following steps:

1. Choose the slopes of $n+1$ families of parallel hyperplanes in order to partition the iteration space into parallelogram shaped tiles of of size $x_1 \times x_2 \times \ldots \times x_{n+1}$. Leave $x_i, \forall i \in \overline{N}$, as free integer variables.

2. "Cluster" recurrence into tiles, yielding a new uniform recurrence over a new domain (say $\overline{Dom}$). Each tile is considered as an atomic computation and the obtained graph is called a *tile graph*.

3. Apply systolic space-time transformations, yielding a virtual n-dimensional systolic array. Implement this array on a torus/grid of size $p_1 \times p_2 \times \ldots \times p_n$ by performing multiple passes if necessary (the so called *locally parallel, globally sequential* partitioning [16]).

4. Relax the systolic timing model to account for practical machines, and obtain a formula for the total running time of the final implementation. This formula will be expressed as a function of $x_i, i = 1, \ldots, n+1$, as well as other parameters which are constants.

5. Choose the tile size that minimizes this running time by solving the corresponding optimization problem. This is a non-linear integer problem, and special techniques are used to solve it.

The first step in our procedure is to choose a convenient tile shape. For this particular case ($\mathbf{D}$ is square and non-singular) the result of Boulet et al. [8] shows that the optimal tile shape is a rectangular parallelepiped, the tile boundaries being parallel to the domain boundaries. We therefore, partition the domain in parallelepipeds of size $x_1 \times x_2 \times \ldots \times x_{n+1}$, specified by orthogonal planes. Now the tile graph is a parallelepiped of $\left\lceil \frac{m_1}{x_1} \right\rceil \times \left\lceil \frac{m_2}{x_2} \right\rceil \times \ldots \times \left\lceil \frac{m_{n+1}}{x_{n+1}} \right\rceil$ nodes (we will drop the ceilings from all subsequent formulæ). It is easily seen that the volume of the transmitted message between any two tiles in axis k (say) is the volume of the tile facet in this axis multiplied by d_{kk} i.e. $d_{kk} \prod_{\substack{i \in \overline{N} \\ i \neq k}} x_i$. Let us suppose now that the $n+1$-th axis is chosen for projection of the tile graph on the array of processors. To guarantee local communications between tile nodes we impose $d_{kk}, k \neq n+1$, to be lower bound for the variable x_k. For the variable x_{n+1} we can keep $x_{n+1} \geq 1$ since the non-local communications are in fact projected in the processors memory. As for the tile graph dependencies, it is known [12] that the tile graph will always have the standard basis vectors as dependencies, provided the tiles are large compared to the original dependencies.

In the more general case of dependency matrix they will be some additional (diagonal) dependencies. By assumption they are with non-negative components and they will lie in the cone of the basis vectors, and can be represented as positive linear combinations of these. The corresponding data can always be combined with the results of the orthogonal dependencies and their communication can be subsumed. The volume of the tile facet should be multiplied by an appropriate constant multiplier, but its value does not modify the nature of our analysis. Static diagonal dependencies can be easily ignored in this way for analysis purposes in our approach, and for these reasons we keep the orthogonal tiling.

Typical examples of n-dimensional orthogonal tiling can be found in Waterman's book [23]. They are often related with the application of dynamic programming to combinatorial optimization problems (as the alignment of protein sequences in [23]).

$$Y[\mathbf{j}] = \min_i (Y[\mathbf{j} - \mathbf{d_i}] + \rho(\ldots)) \quad \mathbf{d_i} \in \{0,1\}^{n+1}, \mathbf{d_i} \neq \mathbf{0}$$

$\mathbf{j} \in Dom \subset Z_+^{n+1}$, where $Dom = \{[\mathbf{j}] : 0 < j_1 \leq m_1, 0 < j_2 \leq m_2, \ldots, 0 < j_{n+1} \leq m_{n+1}\}$,

2.2 "Systolic" space-time transformation for the tile graph

As shown in the previous section the dependency matrix of the tile graph can be considered to be the identity matrix and the iteration space now is given by $\overline{Dom} = \{\mathbf{j} : 0 < j_1 \leq \frac{m_1}{x_1}, 0 < j_2 \leq \frac{m_2}{x_2}, \ldots, 0 < j_{n+1} \leq \frac{m_{n+1}}{x_{n+1}}\}$ where each point represents a tile. We then apply a space-time transformation [16, 18] by choosing $a(\mathbf{j}) = (\mathbf{j} : j_{n+1} = 0)$ as an allocation function and $t(\mathbf{j}) = j_1 + j_2 + \ldots + j_{n+1}$ as a timing function. This is an optimal linear schedule for this recurrence and this gives a virtual n-dimensional "systolic" array of $\frac{m_1}{x_1} \times \frac{m_2}{x_2} \times \ldots \times \frac{m_n}{x_n}$ nodes. We consider the P processors in the network as a logical $p_1 \times p_2 \times \ldots \times p_n$ n-dimensional torus, where $\prod_{i=1}^{n} p_i = P$. We reasonably assume that $P \leq \prod_{i=1}^{n+1} m_i$ and $p_i \leq m_i$ for $\forall i \in N$. We will denote the $[k_1, k_2, \ldots, k_n]$th processor in this mesh as $\mathbf{P}_{k_1,k_2,\ldots,k_n}$. The virtual array will be distributed to the physical array by using multiple passes, so that the $[j_1, j_2, \ldots, j_n]$th tile node is assigned to processor $\mathbf{P}_{(j_1-1) \bmod p_1,(j_2-1) \bmod p_2,\ldots,(j_n-1) \bmod p_n}$.

The above explanations could seem not enough explicit but let us point that we use known results from systolic synthesis from recurrences which is now a well developed research area [10, 13, 16, 18, 19]. We would like however to emphasize that the architecture is not really a conventional systolic array, but has the granularity of a tile which is imposed by the communication model on DMM machine.

2.3 Relaxing the systolic model: execution time on DMM

Using the standard communication model [9, 11, 15, 17], we now develop an expression for the running time of such a tiled program on a DMM machine, which is valid for the range of all possible values of the tile size $\mathbf{x} = (x_1, x_2, \ldots, x_{n+1})$, where x_i - integer, and $\mathbf{x} \in X$,

$$X = \left\{ \mathbf{x} \in R^{n+1} : 1 \leq x_i \leq u_i = \frac{m_i}{p_i}, i = 1, \ldots, n, 1 \leq x_{n+1} \leq u_{n+1} = m_{n+1} \right\} \tag{2}$$

The code executed for a tile is the standard loop (the receive call is a blocking one to ensure synchronization):

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

where we denote by `vi` the message transmitted in the ith axis. In the DMM communication model the time to transmit a message of v words between two processors is given by

$$\text{transfer}(v) = \beta + v\tau_t. \tag{3}$$

where β is the start-up latency, τ_t is the transmission rate for a specific architecture. We assume that the time to execute a single instance of the loop body in (1) is τ_a. The time

to execute a tile body is therefore $\tau_a v(\mathbf{x})$ where $v(\mathbf{x}) = \prod_{i=1}^{n+1} x_i$ is the volume of the tile associated to any point $\mathbf{x} \in R^{n+1}$.

We will use the convention that $\mathcal{P}$, Λ and T denote *period*, *latency* and *completion time*, respectively. Let us define the *tile period*, $\mathcal{P}_t$ as the time elapsed (in the steady state) between corresponding instructions of any two successive tiles (the tiles are considered successive if one depends directly on the other) that are mapped to the *same* processor. The *tile latency*, Λ is the time elapsed between corresponding instructions of two successive tiles mapped to *adjacent* processors.

Since $\mathcal{P}_t$ is at least as large as the time that the CPU is busy during each tile, ($2n$ OS calls, and the tile body) we can specify it as,

$$\mathcal{P}_t(\mathbf{x}) = 2n\beta + \tau_a v(\mathbf{x}) \tag{4}$$

Now consider the latency, Λ, between tiles. 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 first tile is finished, its results are sent, and they arrive into user memory. Taking in account that the volume of the message in a fixed axis (say $k, k \neq n+1$) is $\frac{v(\mathbf{x})}{x_k}$, we get for the latency Λ_k:

$$\Lambda_k(\mathbf{x}) = \mathcal{P}_t(\mathbf{x}) + \tau_t \frac{v(\mathbf{x})}{x_k} - n\beta \tag{5}$$

Observe that because the call to receive is overlapped, we subtract a factor of $n\beta$, and the $\tau_t \frac{v(\mathbf{x})}{x_k}$ accounts for the transmission time. When the axis coincides with the direction of projection we simply neglect the propagation time and therefore we obtain:

$$\Lambda_{n+1}(\mathbf{x}) = \mathcal{P}_t(\mathbf{x}) \tag{6}$$

We have a tile graph of $M_1 \times M_2 \times \ldots \times M_{n+1}$ tiles, where $M_i = \frac{m_i}{x_i}$. The total running time of the program is then obtained using a straightforward application of systolic like counting arguments. In general, we will need $\prod_{i=1}^{n} \frac{M_i}{p_i}$ passes through the torus of processors where each pass treats in a pipelined way $p_1 \times p_2 \times \ldots \times p_n \times M_{n+1}$ tiles. In fact, each pass consumes a parallelepiped of $(p_1 x_1) \times (p_2 x_2) \times \ldots \times (p_n x_n) \times m_{n+1}$ points of the original iteration space. Note that because the $(n+1)$th axis is chosen as direction for projection, we explore the iteration space in this axis completely (i.e, we perform M_{n+1} tiles in the $(n+1)$th axis), before to start a new pass in any other axis. Therefore, the period in the $(n+1)$th axis is given by:

$$\mathcal{P}_{n+1}(\mathbf{x}) = M_{n+1} \mathcal{P}_t(\mathbf{x}) \tag{7}$$

A single pass in kth axis, $(k \neq n+1)$, involves communications between p_k processors, and we define the latency of a pass in this axis as $p_k \Lambda_k(\mathbf{x})$. The pass-period of the torus is determined as the earliest instant when the next pass can start and is given by

$$\mathcal{P}_{torus}(\mathbf{x}) = \max(\mathcal{P}_{n+1}(\mathbf{x}), \min_{i=1,n} (p_i \Lambda_i(\mathbf{x}))) \tag{8}$$

In a single pass, the execution time of a pass is given by

$$T_{pass}(\mathbf{x}) = \mathcal{P}_{n+1}(\mathbf{x}) + \sum_{i=1}^{n}(p_i - 1)\Lambda_i(\mathbf{x}) \tag{9}$$

The entire program is executed in $\prod_{i=1}^{n} \frac{M_i}{p_i}$ passes through the torus, and hence the last pass can only start at $\left(\prod_{i=1}^{n} \frac{M_i}{p_i} - 1\right) \mathcal{P}_{torus}(\mathbf{x})$. Thus, the total running time is given by

$$T(\mathbf{x}) = \left(\prod_{i=1}^{n} \frac{M_i}{p_i} - 1\right) \mathcal{P}_{torus}(\mathbf{x}) + T_{pass}(\mathbf{x}) = \left(\frac{1}{P}\prod_{i=1}^{n} \frac{m_i}{x_i} - 1\right) \mathcal{P}_{torus}(\mathbf{x}) + T_{pass}(\mathbf{x}) \tag{10}$$

Let us set $\tilde{p_k} = p_k - 1$ and let us assume for the sake of simplicity that $\min_{i=1,n}(p_i\Lambda_i) = p_1\Lambda_1$. Our problem reduces therefore to the following minimization problem.

$$\min T(\mathbf{x}) = \begin{cases} T_1(\mathbf{x}) = \dfrac{\mathcal{P}_t(\mathbf{x})}{P}\displaystyle\prod_{i=1}^{n+1} \frac{m_i}{x_i} + \sum_{i=1}^{n} \tilde{p_i}\Lambda_i(\mathbf{x}) & \text{if } \mathcal{P}_{n+1}(\mathbf{x}) \geq p_1\Lambda_1(\mathbf{x}) \\ T_2(\mathbf{x}) = \left(\frac{m_1}{x_1}\displaystyle\prod_{i=2}^{n} \frac{m_i}{x_i p_i} - 1\right)\Lambda_1(\mathbf{x}) + \sum_{i=2}^{n} \tilde{p_i}\Lambda_i(\mathbf{x}) + \mathcal{P}_{n+1}(\mathbf{x}) & \text{otherwise} \end{cases} \tag{11}$$

where x_i are integers and $\mathbf{x} \in X$ given by (2).

3 Solution of the optimization problem

3.1 Basic properties of the objective function

We saw in the previous section that minimizing the total execution time for Eqn (1) is equivalent to finding the minimum of a non-linear objective function (11) in the parallelepiped (2). This domain is divided by the non-linear constraint $\mathcal{P}_{n+1}(\mathbf{x}) = p_1\Lambda_1(\mathbf{x})$ into two regions. Similar to the two(and three)-dimensional case [4, 5] it is easy to argue that region $\mathcal{P}_{n+1}(\mathbf{x}) < p_1\Lambda_1(\mathbf{x})$ is not an interesting one and our problem reduces to minimizing $T_1(\mathbf{x})$ in the region $\mathcal{P}_{n+1}(\mathbf{x}) \geq p_1\Lambda_1(\mathbf{x})$. Substituting and simplifying in (11), we obtain

$$\min\ T(\mathbf{x}) = \prod_{i=1}^{n+1} \frac{m_i}{x_i}\left(\frac{2n\beta}{P} + \frac{\tau_a v(\mathbf{x})}{P}\right) + \sum_{k=1}^{n} \tilde{p_k}\left(n\beta + \tau_t\frac{v(\mathbf{x})}{x_k} + \tau_a v(\mathbf{x})\right) \tag{12}$$

subject to $\mathbf{x} \in X \bigcap Z_+^{n+1}$, where by Z^{n+1} we denote the integer points in R^{n+1}, and

$$p_1\left(n\beta + \tau_t\frac{v(\mathbf{x})}{x_1} + \tau_a v(\mathbf{x})\right) \leq \frac{m_{n+1}}{x_{n+1}}\left(2n\beta + \tau_a v(\mathbf{x})\right) \tag{13}$$

Consider the set of $n+1$ edges of the parallelepiped X defined as follows

$$\mathcal{E}_k = \left\{ \mathbf{x} \in X : \bigcap_{\substack{i=1 \\ i \neq k}}^{n+1} (x_i = u_i) \right\}, k = 1, 2, \ldots, n+1 \tag{14}$$

It is convenient to denote by W the volume of the iteration space, i.e. $W = \prod_{i=1}^{n+1} m_i$. The following lemma describes the behavior of the objective function $T(\mathbf{x})$ for a fixed tile volume.

Lemma 3.1 *For any fixed feasible volume $\overline{v}$ (i.e. $v(\mathbf{x}) = \overline{v}$), where $\overline{v} \in [1, \prod_{i=1}^{n+1} u_i]$, $T(\mathbf{x})$ is an increasing function with respect to x_{n+1} and reaches its minimum for $x_{n+1} = \max(1, \overline{v}/\prod_{i=1}^{n} u_i)$ which is equivalent to saying that the solution of problem* (12–13) *lies either on the facet $x_{n+1} = 1$ or on the edge $\mathcal{E}_{n+1}$ of the parallelepiped X.*

This lemma is a natural extension of the two(and three)-dimensional cases and it reduces the dimension of the solution space for problem (12–13). The objective function in this case has the form, $T(u_1, u_2, \ldots, u_n, x_{n+1}) = \frac{A}{x_{n+1}} + Bx_{n+1} + C$, for some constants A, B, C and its minimum on the edge $\mathcal{E}_{n+1}$ is given by

$$x^*_{n+1} = \sqrt{A/B} = \sqrt{\frac{2n\beta W}{P \prod_{i=1}^{n} u_i^2 (\sum_{k=1}^{n} \tilde{p_k}(\tau_t/u_k + \tau_a))}} \tag{15}$$

3.2 Exploration of facet $x_{n+1} = 1$ of the parallelepiped solution space

Now we are looking for a solution when the minimum lies on the facet $x_{n+1} = 1$. We consider the space R^n_+ but for the sake of simplicity we keep the same notation $(\mathbf{x})$ for the points and we denote by $\mathcal{T}$ the restriction of the objective function T when $x_{n+1} = 1$. We have to solve the problem

$$\min\ \mathcal{T}(\mathbf{x}) = \frac{W}{v(\mathbf{x})} \left(\frac{2n\beta}{P} + \frac{\tau_a v(\mathbf{x})}{P} \right) + \sum_{k=1}^{n} \tilde{p_k} \left(n\beta + \tau_t \frac{v(\mathbf{x})}{x_k} + \tau_a v(\mathbf{x}) \right) \tag{16}$$

subject to $\mathbf{x} \in \left\{ X \bigcap \{x_{n+1} = 1\} \bigcap Z^{n+1}_+ \right\}$, and

$$p_1 \left(n\beta + \tau_t \frac{v(\mathbf{x})}{x_1} + \tau_a v(\mathbf{x}) \right) \quad \leq \quad m_{n+1} \left(2n\beta + \tau_a v(\mathbf{x}) \right) \tag{17}$$

Our strategy will be the following: in this section we will find the *global* minimum of the function (16) in R^n_+, and in the following section we discuss how to project in a convenient way this minimum in the feasible domain of the problem (16)–(17).

Let us set $\tilde{P} = \prod_{i=1}^{n} \tilde{p_i}$. The properties of the objective function $\mathcal{T}(\mathbf{x})$ are given by the following theorem.

Theorem 3.2 *For the function $\mathcal{T}(\mathbf{x})$ over the domain $R^n_+ = \{\mathbf{x} \ : x_i > 0, i = 1, \ldots, n\}$ holds:*

(i) *The tile volume $\hat{v}$ which minimizes $\mathcal{T}(\mathbf{x})$, belongs to the interval $[\mu, \nu]$ where*

$$\mu = \sqrt{\frac{2n\beta W}{(n-1)\tau_t P \sqrt[n]{\tilde{P}} + P\tau_a \sum_{k=1}^n \tilde{p_k}}} \quad \text{and} \quad \nu = \sqrt{\frac{-2n\beta W + (n-1)\tau_t P \sqrt[n]{\tilde{P}}}{-P\tau_a \sum_{k=1}^n \tilde{p_k}}} \tag{18}$$

(ii) *The global minimum of $\mathcal{T}(\mathbf{x})$ is attained at the point $\tilde{\mathbf{x}}$ with components*

$$\tilde{x_i} = \tilde{p_i} \sqrt[n]{\frac{\hat{v}}{\tilde{P}}} \tag{19}$$

Proof. It is convenient to partition the points of R^n_+ into classes $H_v = \{\mathbf{x} \in R^n_+ \ : \prod_{i=1}^n x_i = v\}$. Then from $R^n_+ = \bigcup_v H_v$ we have

$$\min_{R^n_+} \mathcal{T}(\mathbf{x}) = \min_v \min_{H_v} \mathcal{T}(\mathbf{x}) \tag{20}$$

Let us set $A = \frac{2n\beta W}{P}$, $B = \tau_a \sum_{i=1}^n \tilde{p_i}$ and $g(v) = \frac{A}{v} + Bv$. $\mathcal{T}(\mathbf{x})$ can be represented as

$$\mathcal{T}(\mathbf{x}) \quad = \quad g(v) + \tau_t \sum_{k=1}^n \tilde{p_k} \frac{v}{x_k} + \text{const} = g(v) + v\tau_t \sum_{k=1}^n y_k + \text{const} \tag{21}$$

where we set $y_k = \frac{\tilde{p_k}}{x_k}$ and const $= \frac{\tau_a W}{P} + n\beta \sum_{k=1}^n \tilde{p_k}$. Using the inequality

$$\sum_{k=1}^n y_k \geq n \sqrt[n]{\prod_{k=1}^n \frac{\tilde{p_k}}{x_k}} = n \sqrt[n]{\frac{\tilde{P}}{v}} \tag{22}$$

and setting $C = \tau_t \sqrt[n]{\tilde{P}}$ and $f(v) = \frac{A}{v} + Bv + nCv^{1-\frac{1}{n}}$, we obtain

$$T(\mathbf{x}) \geq \frac{A}{v} + Bv + nCv^{1-\frac{1}{n}} + \text{const} = f(v) + \text{const} \tag{23}$$

Hence, for a fixed v the minimum of $\mathcal{T}(\mathbf{x})$ is achieved when $y_1 = y_2 = \ldots = y_n$ in which case (22) is reduced to equality. Hence, we have $\arg\min_{H_v} \mathcal{T}(\mathbf{x}) = \mathbf{x}^v$, where $\mathbf{x}^v$ is the intersection point of the hyperbola

$$\prod_{k=1}^n x_k = v \tag{24}$$

with the line

$$\frac{p_1-1}{x_1}=\frac{p_2-1}{x_2}=\ldots=\frac{p_n-1}{x_n} \tag{25}$$

From (24) and (25), for any $i, 1 \le i \le n$, we obtain the equality $\left(\frac{\tilde{p}_i}{x_i}\right)^n = \frac{\tilde{P}}{v}$ which gives the solution (19) for the optimal tile volume $\hat{v}$. Therefore, (20) is reduced to

$$\min_{R_+^n} \mathcal{T}(\mathbf{x}) = \min_{v>0} f(v) + \text{const} \tag{26}$$

The function $f(v)$ is in fact, the function $\mathcal{T}(\mathbf{x})$ along the optimal line (25) and is suitably expressed in terms of the tile volume v. The minimal value of $f(v)$ can be find through the equation $f'(v) = 0$ whose positive zero is bounded from above and from below by the roots of the polynomials $h(v)$ and $r(v)$ defined below. We have $f'(v) = -\frac{A}{v^2} + B + (n-1)Cv^{-\frac{1}{n}}$ and

$$h(v) = -\frac{A}{v^2} + B + (n-1)\frac{C}{v^2} \le f'(v) \le -\frac{A}{v^2} + B + (n-1)C = r(v).$$

Since $r(v) = 0 \Leftrightarrow v = \sqrt{\frac{A}{B+(n-1)C}}$ and $h(v) = 0 \Leftrightarrow v = \sqrt{\frac{-A+(n-1)C}{-B}}$ we obtain for the bounds

$$\mu = \sqrt{\frac{A}{B+(n-1)C}} \le v \le \sqrt{\frac{-A+(n-1)C}{-B}} = \nu \tag{27}$$

More precised bounds can be obtained but our observation is that the upper bound ν gives sufficiently good approximation in practice and we use it because of its simplicity. □

Example 1

Let us take the nested loop: $n = 4, m_1 = 200, m_2 = 700, m_3 = 100, m_4 = 150, m_5 = 500$ and suppose we have 1050 PE configured in a grid where $p_1 = 5, p_2 = 2, p_3 = 7, p_4 = 15$. For the architectural parameters values we use $\beta = 60, \tau_a = 1, \tau_t = 0.01$ μsec which we have from our experiments on the Intel Paragon [3]. Substituting in (23) we obtain $a = 135538.8711, b = 135873.2441$. Solving numerically $f'(v) = 0$ gives the exact value for the optimal volume $\hat{v} = 135856$ and the root of $s(v)$ provides the value 135873.2416. We observe the exact value is significantly closer to its upper bounds and also the gap between the roots of the polynoms $s(v)$ and $h(v)$ is really insignificant. This behavior has been confirmed in all our experiments. Furthermore, applying the formula (19) we obtain the point of global minimum $\tilde{x_1} = 17.9368, \tilde{x_2} = 4.4829, \tilde{x_3} = 26.8978, \tilde{x_4} = 62.7614$ using 135856 for value of $\hat{v}$ and the point $\tilde{x_1} = 17.9368, \tilde{x_2} = 4.4829, \tilde{x_3} = 26.8978, \tilde{x_4} = 62.7614$ using 135873.2441 for $\hat{v}$.

3.3 The projection of the global minimum over the parallelepiped domain

In this section we suppose the optimal tile volume $\hat{v}$ and the point of the global minimum $\tilde{\mathbf{x}}$ are found by theorem 3.2. For the lake of space we do not present here the proof that

the non-linear constraint (17) is non-active, whenever $(m_{n+1} - p_1)\tau_a/(p_1\tau_t) > 1$, which is true for the majority of the contemporary parallel machines. The goal in this section is to find the point which minimizes the total time (16) in the parallelepiped X. As shown in theorem 3.2 the global minimum point $\tilde{\mathbf{x}}$ lies on the intersection of the optimal tile volume hyperbola (24) with the line (25). If this intersection is not in the feasible domain, we need to "project" this point in the parallelepiped X. Our strategy will be to keep as close as possible to the optimal tile volume hyperbola (24). This is justified by the observation that the volume participates in the dominant term in the objective function (16) and small movements from the optimal tile volume may seriously degrade the optimal function value. Our strategy leads to the problem

$$\text{Minimize} \quad \left\{ \sum_{k=1}^{n} (x_k - \tilde{x_k})^2 \quad : \quad \prod_{k=1}^{n} x_k = \hat{v}, \mathbf{x} \in X \bigcap \{x_{n+1} = 1\} \right\} \tag{28}$$

If $\hat{v} > \prod_{k=1}^{n} u_k$ then we set $x_k = u_k, k = 1, \ldots, n$.

Lemma 3.3 *Let* $1 \leq \hat{v} \leq \prod_{k=1}^{n} u_k$. *The set* $\Delta = \{\mathbf{x} \in R_+^n \ : \ 1 \leq x_k \ \leq \ u_k, \bigcap_{k=1}^{n} x_k = \hat{v}\}$ *is non-empty.*

Proof. Let us suppose the variables x_i are indexed by the following order of the upper bounds $u_0 = 1 < u_1 \leq u_2 \leq u_3 \leq \ldots \leq u_n$. Let us consider the products $u_0 u_1, u_0 u_1 u_2, \ldots \prod_{i=0}^{n} u_i$. Our assumption implies that, for some k, the following relationship $\prod_{i=0}^{k-1} u_i \leq \hat{v} \leq \prod_{i=0}^{k} u_i$ holds. If $\hat{v} < u_1$ then $\mathbf{x} = (\hat{v}, 1, 1, \ldots, 1) \in \Delta$, else the point

$$x_i = \begin{cases} u_i & i = 1, \ldots, k-1 \\ \frac{\hat{v}}{\prod_{l=0}^{k-1} u_l} & i = k \\ 1 & i = k+1, \ldots, n \end{cases} \tag{29}$$

belongs to the domain Δ. $\square$

This lemma suggests the following algorithm of $O(n \log n)$ complexity for approximately solving problem (28).

Algorithm 1 (An algorithm of $O(n \log n)$ complexity)

input : *The global solution* $\tilde{\mathbf{x}}$ *given by* (19)
output : *solution* $\overline{\mathbf{x}}$ *of the problem* (28)
begin
$N := \{1, 2, \ldots, n\}$;
$L := \{i \ : \tilde{x_i} < 1\}$;
$U := \{i \ : \tilde{x_i} > u_i\}$;

$$r_i := \begin{cases} \tilde{x_i} & i \in N \backslash \left(L \bigcup U\right) \\ 1 & i \in L \\ u_i & i \in U \end{cases}$$

assume r_i *are ordered (and re-indexed if necessary) s.t.* $r_1 \leq r_2 \leq \ldots \leq u_n$.
If $\exists k$ *s.t.* $\prod_{i=1}^{k-1} r_i \leq \hat{v} \leq \prod_{i=1}^{k} r_i$ **then**
stop *(***output** $\overline{x_i} := r_i, i < k, x_k := \frac{\hat{v}}{\prod_{l=1}^{k-1} r_l}, \overline{x_i} := 1, i > k$*)*
else *order the set* $N \backslash U$ *s.t.* $u_i - r_i \geq u_{i+1} - r_{i+1}$.
for $i = 1$ **to** $|N \backslash U|$ **do**
$\overline{r_i} := u_i$
if $\prod_{j=1}^{n} r_j \geq \hat{v}$ **then stop** *(* **output** $\overline{x_j} = r_j, j \neq i, \overline{x_i} := \frac{\hat{v}}{\prod_{\substack{l=1 \\ l \neq i}}^{n} \overline{x_l}}$*)*
endfor
end

Example 1 – continuation

Let us apply the algorithm 1 to the considered example. In the previous section we obtained that the point of the global minimum is $\tilde{x_1} = 17.9368, \tilde{x_2} = 4.4842, \tilde{x_3} = 26.9052, \tilde{x_4} = 62.7788$ with optimal volume $\hat{v} = 135856$. The upper bounds $u_i = m_i/p_i$ are $u_1 = 40, u_2 = 350, u_3 = 14, u_4 = 10$. We use therefore for initial approximation of our solution the point $x_1 = 17.9368, x_2 = 4.4842, x_3 = 14, x_4 = 10$ which lies in the domain but not on the optimal volume hyperbola. Since $\prod_{j=1}^{4} x_j = 11260.5 < \hat{v}$ we need to increase the values of some variables. According to the above discussion the best candidate is x_2 since $u_2 - x_2$ reaches the maximum between the gaps $u_i - x_i$ and this variable should take the value $\frac{\hat{v}}{x_1 x_2 x_3} = 54.10$. In this way the proposed algorithm yields the point $x_1 = 17.9368, x_2 = 54.10, x_3 = 14, x_4 = 10$ which lies on the intersection of the domain with the optimal volume hyperbola.

3.4 Toward an integer solution

In the previous section we obtain a real values point lying on the intersection of the domain with the optimal volume hyperbola. When this intersection is empty we take the

point $x_k = u_k$, for $\forall k \in N$. In this section we discuss how to minimize the error due to the integer rounding at the very last step. More precisely the problem is as follows.

Assuming a solution $\overline{\mathbf{x}}$ of the problem (28) is known, find an integer point solving the problem

$$\text{(30)} \quad \text{Minimize} \quad \left\{ |\prod_{k=1}^{n} x_k - \hat{v}| \quad : \quad \prod_{k=1}^{n} x_k = \hat{v}, \quad \mathbf{x} \in X \bigcap \{x_{n+1} = 1\}, \quad x_i \in Z \right\}$$

For solving this problem we propose the algorithm 2 which is a heuristic algorithm with linear complexity.

Example 2

Let suppose we know that the solution $\overline{\mathbf{x}}$ for a nested loop with data $n = 4, u_1 = 40, u_2 = 350, u_3 = 14, u_4 = 10$ is $\overline{x}_1 = 2.77, \overline{x}_2 = 350, \overline{x}_3 = 14, \overline{x}_4 = 10$. The volume for this tile is $\hat{V} = \prod_{i=1}^{4} \overline{x}_i = 135730$. However, after moving to the closest integer point $\hat{\mathbf{x}}' = (3, 350, 14, 10)$ the tile volume gets the value 147000 and the gap is 11270. Following the algorithm (2) we have to update the second component of this point. Since $\left\lceil \frac{\hat{v} - \prod_{k=1}^{n} \hat{x}'_k}{\prod_{\substack{k=1 \\ k \neq i^*}}^{n} \hat{x}'_k} \right\rceil = -26$ we can get closer to the optimal volume hyperbola by moving from point $\hat{\mathbf{x}}'$ to point $\hat{\mathbf{x}} = (3, 324, 14, 10)$ which has tile volume equal to 136080 with gap only 350.

Algorithm 2 (An algorithm of $O(n)$ complexity)

input : *solution* $\overline{\mathbf{x}}$ *of the problem* (28)
output : *solution* $\hat{\mathbf{x}}$ *of the problem* (30)
begin

$NZ := \{i : \overline{x}_i \neq 0 \pmod 1\}$
$\hat{x}_i := \overline{x}_i, i \in N \backslash NZ$
$\hat{x}_i := \lceil \overline{x}_i \rceil, i \in NZ$
find i^* *s.t.* $\hat{x}_{i^*} = \max_{i \in N} \{\hat{x}_i\}$
$\hat{x}_{i^*} := \hat{x}_{i^*} + \left\lceil \frac{\hat{v} - \prod_{k=1}^{n} \hat{x}_k}{\prod_{\substack{k=1 \\ k \neq i^*}}^{n} \hat{x}_k} \right\rceil$

end

4 Experimental results, discussions and illustrations

In this section, we present few results from our computational experiments on the Intel Paragon at IRISA[1] performed in the case of 3-dimensional iteration space. The Paragon's processing nodes are arranged in a two-dimensional rectangular grid (56 Intel

[1] Campus de Beaulieu, 35042 Rennes Cedex, France

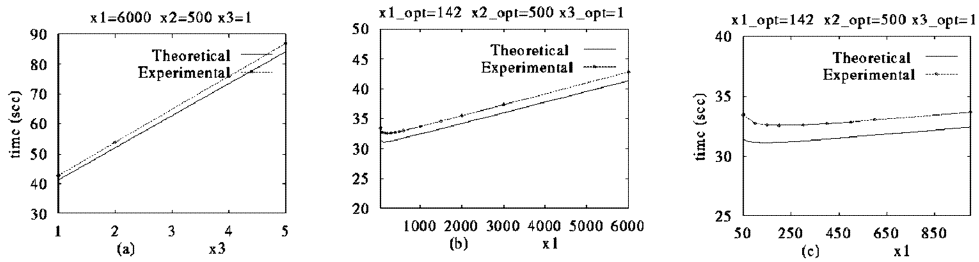

Figure 1: Plots of time vs tile size when the optimal solution needs several passes. The considered instance is $m_1 = 30000, m_2 = 2000, m_3 = 20, p_1 = 5, p_2 = 4$

i860 processors). Each node contains a message processor acting in parallel with the application processor, and access to memory by DMA.

To validate the theoretical results, we created special purpose parallel code which models recurrence (1). First of all, we designed a series of experiments to precisely measure the constants β, τ_t and τ_a. We obtained $\beta = 90\mu$sec, $\tau_t = 0.011\mu$sec per byte and $\tau_a = 0.51\mu$sec. Let us point that the purpose of this section is only to illustrate the peculiarities of the model in the considered (three-dimensional) case. Questions related to the way we measure these constants, some aspects of the impact of the cache on the precision of the results, comparisons with other models ..., are discussed in details in our experimentally oriented paper [3] which concerns the two-dimensional case.

Next, we performed experiments to validate the value of the optimal tile size. We show a series of such experiments in Fig. 1 which illustrates the case when the minimum lies on the plane $x_3 = 1$. We consider an asymmetric domain $m_1 = 30000, m_2 = 2000, m_3 = 20$. Fig. 1 (a) illustrates how faster the running time increases by moving away from the facet where the minimum is situated. On Fig. 1 (b) and Fig. 1 (c) (which is a magnified part of 1 (b)) we compare the theoretical with the experimental values on the edge $\mathcal{E}_1 = \{(x_1, x_2, x_3) \ : 1 \leq x_1 \leq m_1/p_1, x_2 = m_2/p_2, x_3 = 1\}$. In all these cases[2] we observe that the theoretical predictions matched the experimental results very closely. The minimum is attained in the point $x_1^{opt} = 142, x_2^{opt} = 500, x_3^{opt} = 1$. The optimal tile size requires 42 passes along the edge $\mathcal{E}_1$ in order to compute the whole iteration space of size $[30000, 2000, 20]$. We observe for this instance a gain of 10 sec. provided by multiple passes strategy versus the optimal one pass strategy requiring 40 sec. running time (Fig. 1 (a)). The good agreement between the theoretical and experimental time especially in this case is a good criterium both for our theoretical model and for our parallel code. When multiples passes are required, the last processor can produce the result before the first one is ready to accept it, hence a message buffering is needed. A deadlock could appear (when all the processors buffers are full) if the communications are not correctly synchronized between the processors on the border of the torus. This synchronization can be avoided by using a threads (light processes) to ensure a communications between boundary processors. Although, this strategy may seem complicated, our results shows that it is worth providing it. This is especially true when assymetric iteration domains are treated and some restrictive factors could impose fixing of a particular direction of

[2]more experiments are presented in [3, 5]

projection, which could be not the most convenient.

5 Conclusion

We have addressed the problem of the optimal tile size in the case of $n+1$-dimensional perfect nested loop nest with constant loop bounds [6] and uniform recurrences with non-negative components of the dependency matrix. In the most general case of this particular class and we derive an $O(n \log n)$ algorithm for finding an approximate solution. We also reveal new and useful relationships between the objective function and the optimal tile volume. Although, that from mathematical view this is an approximate algorithm, the loss of precision is so insignificant that the results can be considered as exact in the context of the considered application. The validity of the proposed approach has been confirmed by our experimentation for the 3-dimensional case on the Intel Paragon.

Although the considered class may seem very restrictive, similar assumptions are often made by most researchers treating the problem. For example, the Fortran 90 and Paradigm compiler automatically determine the block size for coarse grain pipelined loops [17, 11], assuming that the iteration space is rectangular. Moreover, regardless of the fact that they treat arbitrarily nested loops, it is only the two innermost loops that are tiled. Our approach guarantees exploration of the entire solution domain in the $n+1$-dimensional space and for any size of the parallelepiped iteration space and for any torus configuration this ensures choosing the optimal task granularity. Such global search could provide better solutions, especially when the iteration space is very assymetric which is often the case in dynamic programming algorithms. Let us also mention here the algorithm developed by Wolf and Lam [2, 24] for applying unimodular transformations which leave the original loop nests (even with negative dependencies) in a canonical form consisting of a nest of fully permutable loop nests. What prevents us from a direct application of the described here solution is that the iteration space could be not any more a rectangular parallelepiped. It would be interesting to see how our result could be integrated in such "normalizing" algorithms which is automatically a subject of our ongoing research.

REFERENCES

[1] C. Ancourt, F. Irigoin. Scanning polyhedra with DO loops. In: Third Symposium on Principles and Practice of Parallel Programming (PPoPP). ACM SIGPLAN, ACM Press, 1991, 39-50.

[2] J. Anderson, M. Lam. Global optimisation for parallelism and locality on scalable parallel machines. *ACM Sigplan Notices* **28**, *6* (1993), 112-125.

[3] R. Andonov, H. Bourzoufi, S. Rajopadhye. Two-dimensional orthogonal tiling: from theory to practice. In: International Conference on High Performance Computing (HiPC). India, IEEE Computer Society Press, 1996, 225-231.

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

[5] R. Andonov, N. Yanev, H. Bourzoufi. Three-dimensional orthogonal tile sizing problem: mathematical programming approach. In: International Conf. on Application Specific Systems, Architectures and Processors ASAP'97, Zurich, Switzerland (eds. L. Thiele, J. Fortes, K. Visser, V. Taylor, T. Noll, J. Teich) IEEE, 1997, 209-218.

[6] U. Banerjee. Dependence Analysis for Supercomputing. Kluwer Academic Publishers, 1988.

[7] F. Bitz, H. T. Kung. Path planning on the WARP computer: using a linear systolic array in dynamic programming. *Int. J. Computer Math.* **25** (1988) 173-188.

[8] P. Boulet, A. Darte, T. Risset, Y. Robert. (Pen)-Ultimate Tiling? *INTEGRATION, the VLSI journal* **17** (1944), 33-51.

[9] J. J. Choi, J. Dongarra, D. W. Walker. PUMMA: parallel universal matrix multiplication algorithms on distributed memory concurrent computers. *Concurrency: Practice and Experience* **6**, *7* (1994), 543-570.

[10] A. L. Darte, Y. Robert. Affine-by-statement scheduling of uniform and affine loop nests over parametric domains. *Journal of Parallel and Distributed Computing* **29** (1995), 43-59.

[11] S. Hiranandani, K. Kennedy, C. Tseng. Evaluating compiler optimizations for Fortran D. *Journal of Parallel and Distributed Computing* **21** (1994), 27-45.

[12] F. Irigoin, R. Triolet. Supernode partitioning. In: 15th ACM Symposium on Principles of Programming Languages, ACM, Jan 1988, 319-328.

[13] R. M. Karp, R. E. Miller, S. Winograd. The organization of computations for uniform recurrence equations. *JACM* **14**, *3* (1967), 563-590.

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

[15] S. Miguet, Y. Robert. Path planning on a ring of processors. *Intern. J. Computer Math.* **32** (1990), 61-74.

[16] D. I. Moldovan, J. A. B. Fortes. Partitioning and mapping algorithms into fixed size systolic arrays. *IEEE Transaction on Computers* **C-35**, *1* (1986), 1-12.

[17] D. Palermo, E. Su, A. Chandy, P. Banerjee. Communication optimizations used in the PARADIGM compiler for distributed-memory multicomputers. In: International Conference on Parallel Processing, St. Charles, IL, August 1994.

[18] P. Quinton, Y. Robert. Algorithmes et architectures systoliques. Masson, 1989. (English translation by Prentice Hall, Systolic Algorithms and Architectures, Sept. 1991).

[19] S. V. Rajopadhye, R. M. Fujimoto. Synthesizing systolic arrays from recurrence equations. *Parallel Computing* **14** (1990), 163-189.

[20] J. Ramanujam, P. Sadayappan. Tiling multidimensional iteration spaces for non shared-memory machines. In: Supercomputing 91, 1991, 111-120.

[21] R. Schreiber, J. Dongarra. Automatic Blocking of Nested Loops. Technical Report 38, RIACS, NASA Ames Research Center, Aug 1990.

[22] V. Van Dongen. Loop parallelization on distributed memory machines: problem statement. In: Proceedings of EPPP, 1993.

[23] M. S. Waterman. Mathematical Methods for DNA Sequences. CRC Press, Inc. Boca Raton, Florida, 1989.

[24] M. Wolf, M. S. Lam. A Loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems* **2**, *4* (1991), 452-471.

[25] M. Wolfe. Iteration space tiling for memory hierarchies. In: Parallel Processing for Scientific Computing (SIAM), 1987, 357-361.

Rumen Andonov
LIMAV, Université de Valenciennes
Le Mont Houy, B.P.311
59304 Valenciennes Cedex
France
e-mail: `Rumen.Andonov@univ-valenciennes.fr`

Nicola Yanev
University of Sofia
Faculty of Mathematics and Informatics
5, J. Baucher str., Sofia, Bulgaria
e-mail: `choby@math.bas.bg`