A Pipelined Execution of Tiled Nested Loops on SMPs with Computation and Communication Overlapping

Maria Athanasaki, Aristidis Sotiropoulos, Georgios Tsoukalas and Nectarios Koziris
National Technical University of Athens
Dept. of Electrical and Computer Engineering
Computing Systems Laboratory
e-mail: {maria, sotirop, gtsouk, nkoziris}@cslab.ece.ntua.gr

Abstract

This paper proposes a novel approach for the parallel execution of tiled Iteration Spaces onto a cluster of SMP PC nodes. Each SMP node has multiple CPUs and a single memory mapped PCI-SCI Network Interface Card. We apply a hyperplane-based grouping transformation to the tiled space, so as to group together independent neighboring tiles and assign them to the same SMP node. In this way, intranode (intragroup) communication is annihilated. Groups are atomically executed inside each node. Nodes exchange data between successive group computations. We schedule groups much more efficiently by exploiting the inherent overlapping between communication and computation phases among successive atomic group executions. The applied non-blocking schedule resembles a pipelined datapath where group computation phases are overlapped with communication ones, instead of being interleaved with them. Our experimental results illustrate that the proposed method outperforms previous approaches involving blocking communication or conventional grouping schemes.

1 Introduction

One of the most difficult areas in the field of parallel computing is loop parallelization and efficient mapping onto different parallel architectures. In order to achieve maximum acceleration of the final program, one of the key issues that should be taken into account is minimization of the communication overhead, which considerably decelerates the system. As far as fine grain parallelism is concerned, in order to reduce the communication overhead, several methods have been proposed to group together neighboring chains iterations [10, 7], while preserving the optimal hyperplane schedule [13, 11, 3]. As far as coarse grain parallelism is concerned, researchers are dealing with the problem of alleviating the communication overhead by applying the supernode or tiling transformation. Supernode partitioning of the iteration space was proposed by Irigoin and Triolet in [6]. They introduced the initial model of loop tiling and gave conditions for a tiling transformation to be valid. Later, Ramanujam and Sadayappan in [9] showed the equivalence between the problem of finding a set of extreme vectors for a given set of dependence vectors and the problem of finding a tiling transformation H that produce valid, deadlock-free tiles. The problem of determining the optimal shape was surveyed, and more accurate conditions were also given by others as Xue in [14] and Boulet et al. in [2].

Nevertheless, all above approaches ignore the actual iteration space boundaries. Although tile shape is of great importance to communication reduction, the objective should be the overall tiled space completion time. Hodzic and Shang [5] proposed a method to correlate optimal tile size and shape, based on overall completion time reduction. Their approach considers a straightforward time schedule where each processor executes all tiles along a specific dimension, by interleaving computation and communication phases. All processors first receive data, then compute and finally send result data to neighbors in explicitly distinct phases, according to the hyperplane scheduling vector.

In [4] we proposed an alternative method for the problem of scheduling the tiles to single CPU nodes. Each atomic tile execution involves a communication and a computation phase and this is repeatedly done for all time planes. We are compacting this sequence of communication and computation phases, by overlapping them for the different processors. The proposed method acts like enhancing the performance of a processor's datapath with pipelining [8], because a processor computes its tile at k time step and concurrently receives data from all neighbors to use them at $k + 1$ time step and sends data produced at $k - 1$ time step. Since data communication involves some startup latencies, we adjust the computation grain to make room for this overhead and try to overlap with all communication, which can be done in parallel. Previous work in the field of UET-UCT scheduling of grid graphs in [1], has shown that this schedule is optimal when the computation to communication ratio is one.

In this paper we extend the method proposed in [4] for executing tiled iteration spaces in SMP nodes. We group together neighboring tiles along a hyperplane. Hyperplane-grouped tiles are concurrently executed by the CPUs of the same SMP node. In this way, we eliminate the need for tile synchronization and communication between intranode CPUs. As far as scheduling of groups is concerned, we take advantage of the overlapping schedule of [4] in order to "hide" each group communication volume within the respective computation volume. Under the above implementation scheme, the iteration space involves the overlapped execution of communication and computation phases between successive groups of tiles. We thus avoid most of the communication overhead by allowing for actual computation to communication overlapping.

We compare our method, using blocking schedules and vertical grouping of neighboring tiles along a specific dimension. Vertically grouped tiles are assigned to the same node, and an optimal hyperplane time schedule is applied. However, this imposes additional intranode synchronization delays. All experimental results show that when the hyperplane grouping of tiles together with the overlapping schedule are applied, the overall completion time is considerably reduced, under the condition of controlling the computation to communication grain.

The rest of this paper is organized as follows: Basic terminology used throughout the paper and definitions of loop tiling are introduced in Section 2. In Section 3 we supply an algorithm for the application of the overlapping scheme (proposed in [4]) on clusters of SMP nodes and we investigate the resulting time schedule. In Section 4 we describe the experiments executed on a cluster of SMPs using PCI-SCI Network Interface cards in order to verify our theory. Finally, in Section 5 we summarize our results and propose future work.

2 Models – Loop Tiling

In this paper we consider algorithms with perfectly nested FOR-loops and uniform data dependencies, as in [4]. Throughout the paper the following notation is used: N is the set of natural numbers and n is the number of nested FOR-loops of the algorithm. $J^n \subset Z^n$ is the set of indices: $J^n = \{j(j_1, ..., j_n) | j_i \in Z \wedge l_i \leq j_i \leq u_i, 1 \leq i \leq n\}$. Each point in this n-dimensional integer space is a distinct instantiation of the loop body. A dependence vector is denoted $d_i = (d_{i1}, ..., d_{in}), 1 \leq i \leq q$. The dependence set D of an algorithm is the set of all dependence vectors of this algorithm: $D = \{d_1, d_2, ..., d_q\}$.

In a supernode or tiling transformation the Iteration space J^n is partitioned into identical n-dimensional parallelepiped areas (tiles or supernodes) formed by n independent families of parallel hyperplanes. Tiling transformation is defined by the n-dimensional square matrix H. Each row vector of H is perpendicular to one family of hyperplanes forming the tiles. Dually, tiling transformation can be defined by n linearly independent vectors, which are the sides of the tiles. Similar to matrix H, matrix P contains the side-vectors of a tile as column vectors. It holds $P = H^{-1}$.

Formally, tiling transformation is defined as follows:

$$r : Z^n \longrightarrow Z^{2n}, r(j) = \begin{bmatrix} \lfloor Hj \rfloor \\ j - H^{-1} \lfloor Hj \rfloor \end{bmatrix},$$

where $\lfloor Hj \rfloor$ identifies the coordinates of the tile that index point $j(j_1, j_2, \ldots, j_n)$ is mapped to and $j - H^{-1} \lfloor Hj \rfloor$ gives the coordinates of j within that tile relative to the tile origin. The Tile space J^S and the Tile Dependence matrix D^S are defined as follows: $J^S = \{j^S | j^S = \lfloor Hj \rfloor, j \in J^n\}$, $D^S = \{d^S | d^S = \lfloor H(j_0 + d) \rfloor, d \in D, j_0 \in J^n | 0 \leq \lfloor Hj_0 \rfloor \leq 1\}$ where j_0 denotes the index points belonging to the first complete tile starting from the origin of the Iteration space J^n. The Tile space can be also written as $J^S = \{j^S(j_1^S, \ldots, j_n^S) | j_i^S \in Z \wedge l_i^S \leq j_i^S \leq u_i^S, 1 \leq i \leq n\}$. Each point j^S in this n-dimensional integer space J^S is a distinct tile with coordinates $(j_1^S, j_2^S, \ldots, j_n^S)$.

Given an algorithm with dependence matrix D, for a tiling to be legal, it must hold $HD \geq 0$ (see [6, 9]). In this paper, as in [4], we assume that all dependence vectors are smaller than the tile size, thus they are entirely contained in each tile's area, which means that $|HD| < 1$ [15], or, alternatively, that the tile dependence matrix D^S contains only 0's and 1's.

3 Application of the overlapping schedule to SMP nodes

In the rest of this paper, we shall consider that the non-overlapping and overlapping schedules, extensively described in [4] (sections 3,4), are known. In the sequel we shall generalize the proposed overlapping schedule and apply it on a cluster of Symmetric Multiprocessors (SMP nodes).

Let us consider the following scenario: A 2-dimensional nested loop to be executed onto a cluster of 3 single CPU nodes. We tile the Iteration Space of the algorithm and assign each row of tiles to a CPU node. Apparently, we should select the size and shape of tiles so that the Iteration Space is partitioned into 3 rows of tiles (since 3 CPU's are available). Then the tiles can be computed using either the overlapping or the non-overlapping schedule presented in [4].

If, instead of 3 single CPU nodes, we have 3 SMP nodes,

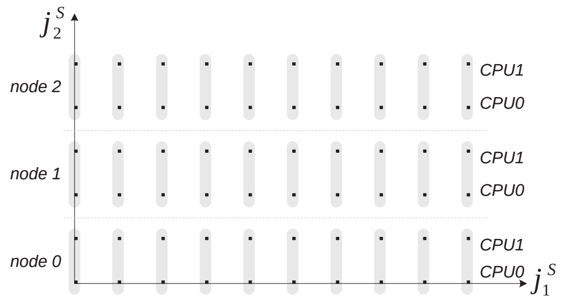

Figure 1. Vertical grouping

with 2 CPU's each, then we can split each tile into two subtiles and assign each subtile to one of the CPU's of the corresponding SMP node, as indicated in Fig. 1. Equivalently, we may tile the initial Iteration Space, selecting the size of tiles so as to get six rows of tiles. Then we assign a row of tiles to each CPU of the SMP nodes and group together neighboring tiles assigned to the same SMP node, as in Fig. 1. It is obvious that the tiles grouped together by this scheme cannot be simultaneously executed, unless they are split into subtiles. Thus, additional synchronization overhead is imposed due to subtile dependencies.

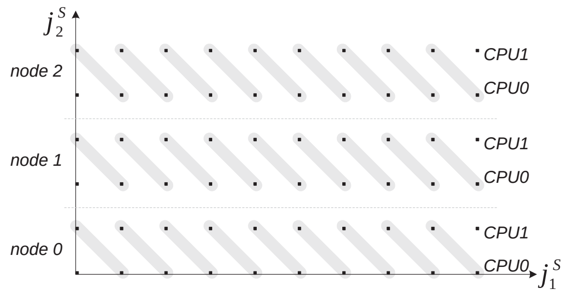

Figure 2. Hyperplane grouping

A more efficient scheme can be obtained if we group the tiles assigned to the same SMP nodes as indicated in Fig. 2. Then both tiles belonging to the same group can be simultaneously executed by the CPU's of an SMP node. We shall call this grouping scheme as *"hyperplane grouping"*. On the contrary, any other grouping scheme along a specific dimension, such as the one presented in Fig. 1, will be called *"vertical grouping"*.

3.1 Grouping Transformation

In order to generate an appropriate time schedule, we need to group together the tiles of J^S that will be concurrently executed by the CPU's of the same SMP node. We further apply an additional supernode transformation to the Tile Space J^S. Thus from the Tile Space J^S we produce the *Group Space* $J^G = \{j^G | j^G = \lfloor H^G j^S \rfloor, j^S \in J^S\}$. This *grouping transformation* is defined by the $n \times n$ non-singular matrix H^G. In correspondence to the tiling matrix H, we call the $n \times n$ matrix H^G as *grouping matrix*. The $n \times n$ matrix $P^G = (H^G)^{-1}$ is called *inverse grouping matrix*.

In order a grouping transformation to be valid, it should preserve the constraint of atomicity of groups ($H^G D^S \geq 0$ in correspondence to $HD \geq 0$ for tiling). In addition, since within a group all tiles are concurrently executed by the CPU's of an SMP node, in order to preserve data consistency, there should be no direct or indirect dependence among them.

3.2 Determining P^G according to the number of CPU's within a node

Consider now the general case, where we have an n-dimensional tiled Iteration Space and a cluster of SMP nodes, each with m processors inside. Our objective is to assign the tiles of J^S along of the 1-st dimension to the same CPU of an SMP node. Let us assume that the natural number m can be written as $m = m_2 \times m_3 \times \ldots \times m_n$, where $m_2, m_3, \ldots, m_n \in N$. Then, we select the grouping matrices to be

$$P^G = \begin{bmatrix} 1 & -m_2 & \ldots & -m_n \\ 0 & m_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & m_n \end{bmatrix},$$
$$H^G = (P^G)^{-1} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 0 & \frac{1}{m_2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \frac{1}{m_n} \end{bmatrix}. \quad (1)$$

The maximum number of tiles contained inside a group is $det(P^G) = m$, exactly equal to the number of CPU's inside each SMP node.

In order to prove that H^G defines a legal grouping transformation, it suffices to prove that $H^G D^S \geq 0$, where D^S is the dependence matrix of the Tile Space J^S and that any two tiles ($j^S, j^{S'} \in j^G$) within the same group are independent. We have assumed (see §2) that the dependence matrix D^S contains only 0's and 1's. Consequently, the first condition is apparently valid. In order to prove the second condition, we assume that the dependence matrix D^S is equal to the unitary matrix. Even if there is a dependence vector with more than one 1's, it is the sum of more than one unitary dependence vectors. So it will be included in the following proof as an indirect dependence:

If tiles $j^S, j^{S'} \in J^S$ belong to the same group j^G then it holds that: $\lfloor H^G j^S \rfloor = \lfloor H^G j^{S'} \rfloor \Rightarrow j_1^S + j_2^S + \ldots +$

$j_{n-1}^S + j_n^S = j_1^{S'} + j_2^{S'} + \ldots + {j_{n-1}^S}' + j_n^{S'}$ In addition, if there is a direct or an indirect dependence from j^S to $j^{S'}$ then it holds that $j^{S'} = j^S + \sum_{i=1}^n \lambda_i d_i$, where $\lambda_i \in N$ and d_i is a unitary dependence vector. The previous equality can be rewritten as follows: $j^{S'} = j^S + \lambda$, where $\lambda = (\lambda_1, \ldots, \lambda_n)$. Thus $j_i^{S'} = j_i^S + \lambda_i$, $i = 1, \ldots, n$. Therefore the equality $j_1^S + j_2^S + \ldots + j_{n-1}^S + j_n^S = j_1^{S'} + j_2^{S'} + \ldots + {j_{n-1}^S}' + j_n^{S'}$ can be rewritten as follows: $\lambda_1 + \lambda_2 + \ldots + \lambda_n = 0$. As $\lambda_1, \ldots, \lambda_n \in N$ it holds that $\lambda_1 = \ldots = \lambda_n = 0$. Consequently, there is no direct or indirect dependence between two iterations belonging to the same group $j^G \in J^G$ and all tiles of a group in J^G can be computed simultaneously by the CPU's of an SMP node. Thus the above grouping transformation is valid according to our algorithmic model.

Example 1: We have a cluster of SMP nodes with 2 CPU's and a NIC each. We assume a 2-dimensional rectangular Tile Space J^S. Let us assign the tiles along of the dimension j_1^S to the same CPU, as indicated in Fig. 3 by the grey arrows. The CPU's of the same SMP node will undertake two neighboring rows of tiles.

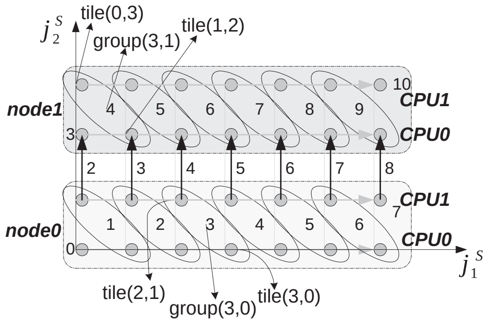

Figure 3. 2D example

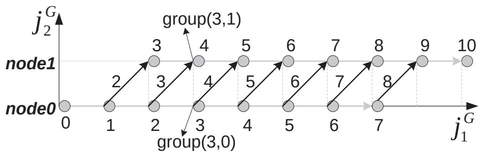

Figure 4. Group Space for the 2D example

Then, during the time step t=0, the CPU-0 of the SMP node0 computes tile $(0,0)$. During the time step $t = 1$ the CPU-0 of node0 computes tile $(1,0)$, while the CPU-1 of the same SMP node computes tile $(0,1)$. Similarly, during the time step $t = 2$ the CPU-0 computes tile $(2,0)$, while the CPU-1 computes tile $(1,1)$. At the same time, the data computed in tile $(0,1)$, which are necessary for the computation of tile $(0,2)$, can be sent to node1. During the time step t=3 the CPU's of node0 can continue the execution as above, while the CPU's of node1 start executing the same routine with the rows of tiles $(x,2)$ and $(x,3)$.

In order to construct a time schedule for this example, we group together the tiles that should be concurrently executed by the same SMP node. In particular, we perform *grouping* to the Tile Space J^S, as indicated in Fig. 3 and derive the Group Space J^G. The appropriate grouping matrices, according to the formula (1), for this case are $P^G = \begin{bmatrix} 1 & -2 \\ 0 & 2 \end{bmatrix}$ and $H^G = (P^G)^{-1} = \begin{bmatrix} 1 & 1 \\ 0 & \frac{1}{2} \end{bmatrix}$. In this way, tiles $(1,0)$ and $(0,1)$ which, as we have already mentioned, are simultaneously executed by the same SMP node, are grouped together in $j^G = \lfloor H^G(1,0)^T \rfloor = \lfloor H^G(0,1)^T \rfloor = (1,0)^T$. Similarly, tiles $(2,0)$ and $(1,1)$ are grouped together in $j^G = (2,0)^T$. In Fig. 3, the time step, when each group will be computed, is shown, together with the time step, where each data transfer will take place. In Fig. 4, the corresponding Group Space is also shown. It can be easily deduced that a group $j^G = (j_1^G, j_2^G) \in J^G$ will be executed during the time step $t(j^G) = j_1^G + j_2^G$ in the SMP node j_2^G. Therefore the linear time scheduling vector for this example is $\Pi^G = (1,1)$. $\diamond$

3.3 Linear time schedule - CPU Tile Assignement

Applying the above grouping transformation, the 1-st column-vector of the dependence matrix $D^S = I$ is transformed to the vector $d_n^{G'} = H^G d_n^S = (1, 0, \ldots, 0)^T$. In addition, the j-th column-vector of the dependence matrix $D^S = I$, $j = 2, \ldots, n$, is transformed to the vector $H^G d_j^S = (1, 0, \ldots, 0, \frac{1}{m_j}, 0, \ldots, 0)^T$. It imposes the dependencies $(1, 0, \ldots, 0, \lfloor \frac{1}{m_i} \rfloor, 0, \ldots, 0)^T = (1, 0, \ldots, 0, 0, 0, \ldots, 0)^T$ and $(1, 0, \ldots, 0, \lceil \frac{1}{m_j} \rceil, 0, \ldots, 0)^T = (1, 0, \ldots, 0, 1, 0, \ldots, 0)^T$ in the Group Space. Thus, the dependence matrix of the Group Space can be written as:

$$D^G = \begin{pmatrix} 1 & 1 & \ldots & 1 & 1 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix}.$$

We are searching for an appropriate linear time scheduling vector $\Pi^G = (\pi_1^G, \ldots, \pi_n^G)$ such that each group $j^G \in J^G$ is computed during the time step $t = \Pi^G j^G$. Consider the last $(n-1)$ coordinates of a group indicating which SMP node of the cluster will execute this group. Then, the groups $j^G = (j_1^G, \ldots, j_n^G)$ and $j^{G'} = (j_1^G + 1, j_2^G, \ldots, j_n^G)$ will be successively computed within the same SMP node. There is a dependence between them, as indicated by the first column of D^G, but there is no need for a communication step between their successive computation steps, because the necessary data are already located in the local shared memory of the SMP node. Consequently, their time distance $\Pi^G j^{G'} - \Pi^G j^G = \pi_1^G$ may be equal to 1. Thus $\pi_1^G = 1$. In addition, the i-th column of D^G $(i = 2, \ldots n)$ imposes a dependence between the groups $j^G = (j_1^G, \ldots, j_n^G)$ and $j^{G'} = (j_1^G + 1, j_2^G, \ldots, j_{i-1}^G, j_i^G + 1, j_{i+1}^G, \ldots, j_n^G)$. These groups are executed in neighboring SMP nodes, thus a communication step is required between their computation steps. It means that their time distance $\Pi^G j^{G'} - \Pi^G j^G = \pi_1^G + \pi_i^G$ must be equal to 2. Consequently $\pi_i^G = 1$, $i = 2, \ldots, n$. So the vector $\Pi^G = (1, 1, \ldots, 1)$ is selected for the linear time scheduling of our Group Space J^G.

Notice that in [4, 12], for the single CPU pipelined schedule, Π was $(1, 2, \ldots, 2)$ according to the UET-UCT theory. In other words, the optimal overlapping schedule could be achieved when we had equal computation to communication times, so that all communication could be hidden (overlapped) with the computation phase. Nevertheless, in the SMP case, presented here, the labeling of coordinates of groups, that is the grouping transformation P^G slightly skews the space (see Fig. 3 and the resulting Group Space in Fig. 4 the relative positions of groups $(3, 0)$ and $(3, 1)$), so the optimal overlapping schedule is achieved by $(1, 1, \ldots, 1)$.

For node labelling reasons, consider that the available SMP nodes form a virtual $(n-1)$-dimensional mesh. Thus, each node is identified by a $(n-1)$-dimensional vector. Then the last $(n-1)$ coordinates of a group indicate the SMP into which it will be executed. The first coordinate affects only the time of its execution. Thus, a tile $j^S = (j_1^S, \ldots, j_n^S)$, belonging to group $j^G = (j_1^G, \ldots, j_n^G)$, will be executed in node $(j_2^G, \ldots, j_n^G) = (\lfloor \frac{j_2^S}{m_2} \rfloor, \ldots, \lfloor \frac{j_n^S}{m_n} \rfloor)$.

Similarly, inside each SMP we consider a $(n-1)$-dimensional CPU virtual mesh containing labels $\{\vec{cpu} \in Z^{n-1} | 0 \le cpu_x < m_{x+1} - 1, 1 \le x \le n-1\}$. Then a tile $j^S = (j_1^S, \ldots, j_n^S)$ will be executed by CPU $(j_2^S \% m_2, \ldots, j_n^S \% m_n)$ of SMP node $(\lfloor \frac{j_2^S}{m_2} \rfloor, \ldots, \lfloor \frac{j_n^S}{m_n} \rfloor)$. So, apparently, only tiles with the same coordinate j_1^S will be assigned to the same CPU of the same node.

3.4 Generalization: Grouping along an arbitrary dimension of J^S

If we want to assign the iterations along the i-th dimension of J^S to the same CPU of an SMP node, then it can be similarly proven that the appropriate grouping matrices are

$$P^G = \begin{bmatrix} m_1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & m_{i-1} & 0 & 0 & \cdots & 0 \\ -m_1 & \cdots & -m_{i-1} & 1 & -m_{i+1} & \cdots & -m_n \\ 0 & \cdots & 0 & 0 & m_{i+1} & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & m_n \end{bmatrix},$$

$$H^G = \begin{bmatrix} \frac{1}{m_1} & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & \frac{1}{m_{i-1}} & 0 & 0 & \cdots & 0 \\ 1 & \cdots & 1 & 1 & 1 & \cdots & 1 \\ 0 & \cdots & 0 & 0 & \frac{1}{m_{i+1}} & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & \frac{1}{m_n} \end{bmatrix}, \quad (2)$$

respectively, where $m_1 \times \ldots \times m_{i-1} \times m_{i+1} \times \ldots \times m_n = m$. As previously, the time scheduling vector is $\Pi^G = (1, \ldots, 1)$. In addition, a tile $j^S = (j_1^S, \ldots, j_n^S)$ belonging to group $j^G = (j_1^G, \ldots, j_n^G)$, will be executed within node $(j_1^G, \ldots, j_{i-1}^G, j_{i+1}^G, \ldots, j_n^G)$ by CPU $(j_1^S \% m_1, \ldots, j_{i-1}^S \% m_{i-1}, j_{i-1}^S \% m_{i-1}, \ldots, j_n^S \% m_n)$.

Example 2: We have a cluster of SMP nodes with 2

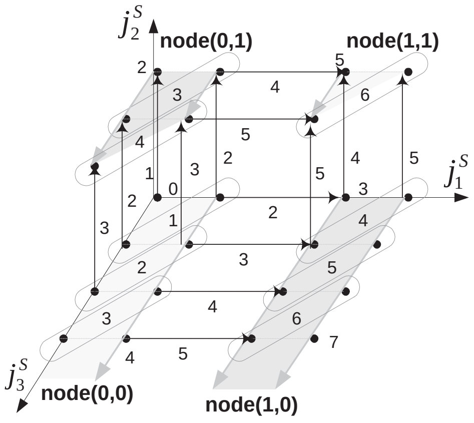

Figure 5. 3D example

CPU's and a NIC each. We assume a 3-dimensional rectangular Tile Space J^S. Let us assign the tiles along of the dimension j_3^S to the same CPU, as indicated in Fig. 5 by the grey arrows. The CPU's of the same SMP node will execute two neighboring rows of tiles which belong to the same $j_1^S - j_3^S$ plane. In respect to the formula (2), we choose the grouping matrices to be:

$$P^G = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -1 & 1 \end{bmatrix} \text{ and } H^G = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

In Fig. 5 we show the grouping of tiles and when each computation step and each communication step will be executed. It can be easily deduced that a group $(j_1^G, j_2^G, j_3^G) \in J^G$ will be executed in node (j_1^G, j_2^G) during the time step $t(j^G) = j_1^G + j_2^G + j_3^G$. Therefore the linear time scheduling vector for this example is $\Pi^G = (1, 1, 1)$.

Let us assume that the rectangular Tile Space has bounds: $0 \le j_1^S < j_{1max}^S$, $0 \le j_2^S < j_{2max}^S$, $0 \le j_3^S < j_{3max}^S$, where j_{1max}^S is an even number. Then, the bounds of the corresponding Group Space will be $0 \le$

$j_1^G \le \lfloor \frac{j_{1max}^S - 1}{2} \rfloor = \frac{j_{1max}^S}{2} - 1$, $0 \le j_2^G \le j_{2max}^S - 1$, $0 \le j_3^G \le j_{1max}^S + j_{2max}^S + j_{3max}^S - 3$. During the first time step $t = 0$, the group $(0, 0, 0)$ will be computed. During the last time step $t = \frac{3j_{1max}^S}{2} + 2j_{2max}^S + j_{3max}^S - 5$, the group $(\frac{j_{1max}^S}{2} - 1, j_{2max}^S - 1, j_{1max}^S + j_{2max}^S + j_{3max}^S - 3)$ will be computed. Thus the total execution steps will be $P = \frac{3j_{1max}^S}{2} + 2j_{2max}^S + j_{3max}^S - 4$. $\diamond$

3.5 Comparison

In this section we shall compare vertical grouping, which is indicated in Fig. 1, with the proposed scheme of hyperplane grouping, which is shown in Fig. 2, 3 in the case of a 2-dimensional algorithm and a cluster of SMP's with 2 CPU's each.

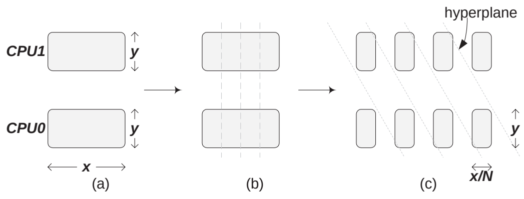

Figure 6. Splitting tiles in vertical scheme

As we have already mentioned, vertical grouping cannot exploit the computational power of both CPU's of our SMP's unless we split each tile into smaller subtiles and compute some of them in parallel, as shown in Fig. 6. Let us assume that a CPU needs time α for the computation of a tile with dimensions x, y (Fig. 6a). Consequently, it will need time $\frac{\alpha}{N}$ for the computation of a respective subtile with dimensions $\frac{x}{N}, y$ (Fig. 6c). The subtiles which are created can be computed by 2 CPU's in $N + 1$ computational steps, interleaved with N synchronization steps, following an optimal linear time schedule $(1, 1)$ as in Fig. 6c. If the average time consumed for the synchronization of 2 CPU's of an SMP node is t_{synch_in}, then the total time required for the computation of a pair of initial tiles is:

$$\beta = \alpha \frac{N+1}{N} + N t_{synch\_in}. \quad (3)$$

The time required for the computation of a pair of tiles is minimized when

$$N = \sqrt{\frac{\alpha}{t_{synch\_in}}}. \quad (4)$$

Therefore, the minimum value of β is $\beta_{min} = \alpha + 2\sqrt{\alpha t_{synch\_in}} > \alpha$.

If we consider an Iteration Space with size $X \times Y$, tiled with rectangular tiles with size $x \times y$, (for example in Fig. 1,2 we have $\frac{X}{x} = 10, \frac{Y}{y} = 6$), then we have the following options:

1. Following the **non-overlapping** scheme (which can be implemented using blocking calls) in combination with **vertical grouping**, the number of time steps required for the completion of the algorithm is $P = \frac{X}{x} + \frac{Y}{2y} - 1$. The minimum duration of a time step is $\beta_{min} + t_{comm}$, where t_{comm} is the time required for the communication between two SMP nodes. Thus the total time required is $T_{blocking,vertical} = P(\beta_{min} + t_{comm}) \simeq (\frac{X}{x} + \frac{Y}{2y})(\beta_{min} + t_{comm})$.

2. Following the **overlapping** scheme (which can be implemented using non-blocking calls) in combination with **vertical grouping**, the number of time steps required for the completion of the algorithm is $P = \frac{X}{x} + \frac{Y}{y} - 2$. According to the formula $T_{overlap} = P(t_{start\_dma} + \max(t_{comp}, t_{comm\_dma}) + t_{synchro})$ (explained in [4]), if we set $t_{comp} = \beta_{min}$, the minimum duration of a time step is $t_{start\_dma} + max(\beta_{min}, t_{comm\_dma}) + t_{synchro}$. Thus the total time required is $T_{non-blocking,vertical} = P(t_{start\_dma} + max(\beta_{min}, t_{comm\_dma}) + t_{synchro}) \simeq (\frac{X}{x} + \frac{Y}{y})(t_{start\_dma} + max(\beta_{min}, t_{comm\_dma}) + t_{synchro})$. If $\beta_{min} \ge t_{comm\_dma}$, then $T_{non-blocking,vertical} \simeq (\frac{X}{x} + \frac{Y}{y})(t_{start\_dma} + \beta_{min} + t_{synchro})$.

3. Following the **overlapping** scheme in combination with **hyperplane grouping**, the number of time steps required for the completion of the algorithm is $P = \frac{X}{x} + \frac{3Y}{2y} - 2$. According to the formula $T_{overlap} = P(t_{start\_dma} + \max(t_{comp}, t_{comm\_dma}) + t_{synchro})$, if we set $t_{comp} = \alpha$, the minimum duration of a time step is $t_{start\_dma} + max(\alpha, t_{comm\_dma}) + t_{synchro}$. Thus the total time required is $T_{non-blocking,hyperplane} = P(t_{start\_dma} + max(\alpha, t_{comm\_dma}) + t_{synchro}) \simeq (\frac{X}{x} + \frac{3Y}{2y})(t_{start\_dma} + max(\alpha, t_{comm\_dma}) + t_{synchro})$. If $\alpha \ge t_{comm\_dma}$, then $T_{non-blocking,hyperplane} \simeq (\frac{X}{x} + \frac{3Y}{2y})(t_{start\_dma} + \alpha + t_{synchro})$.

In most real problems it holds that $\frac{Y/y}{X/x} = \lambda \ll 1$. Therefore, the overlapping scheme in combination with vertical grouping is more efficient than the non-overlapping scheme, in case that $\beta_{min} \ge t_{comm}$, when $t_{comm} > \frac{\lambda}{2}(t_{start\_dma} + \beta_{min} + t_{synchro})$. In addition, the overlapping scheme in combination with hyperplane grouping is more efficient than the overlapping scheme in combination with vertical grouping when $(\frac{X}{x} + \frac{3Y}{2y})(t_{start\_dma} + \alpha + t_{synchro}) < (\frac{X}{x} + \frac{Y}{y})(t_{start\_dma} + \alpha + 2\sqrt{\alpha t_{synch\_in}} + t_{synchro})$. If we consider $t_{start\_dma} + t_{synchro} \ll \alpha$ then we get $2\sqrt{\frac{t_{synch\_in}}{\alpha}} > \frac{\lambda/2}{1+\lambda} \simeq \frac{\lambda}{2} \Rightarrow t_{synch\_in} > \alpha\left(\frac{\lambda}{4}\right)^2$. But in any case the hyperplane grouping has the advantage that it needs no extra tiling inside each tile in order to exploit the computational force of the CPU's.

4 Experimental results

In [12] we applied the pipelined schedule proposed in [4], using a cluster of single CPU nodes with PCI-SCI NICs. In this paper, in order to evaluate the proposed methods, we ran our experiments on a Linux SMP cluster with 8 identical nodes. Each node had 128M of RAM and 2 Pentium III 800 MHz CPUs. The cluster nodes were interconnected with an SCI ring, using SCI Dolphin's PCI-SCI D330 cards. SCI NICs support shared memory programming, either through PIO (Programmed-IO) messaging, or through DMA. We are using their kernel-level DMA support for messaging. Invoking kernel system calls, causes extra CPU cycles overhead. However, we can avoid extra copying from user space to kernel space (physical memory) when using DMA. We allocate user level pages, which correspond to physically contiguous pre-reserved memory regions, for DMA communications.

Our test application was the following code:

```
for(i=1; i<=X; i++)
  for(j=1; j<=Y; j++)
    for(k=1; k<=Z; k++)
      A[i][j][k]=func(A[i-1][j][k],
             A[i][j-1][k],A[i][j][k-1]);
```

where A is an array of $X \times Y \times Z$ floats and $X = Y << Z$. Without lack of generality, we select as a tile a rectangle with ij, ik and jk sides. The dimension k is the largest one, so all tiles along the k-axis are mapped onto the same processor, as proposed in [4]. Each tile has i, j dimensions equal to x and the tile's "height" along k-axis equal to z. There are $\frac{X}{x}$ tiles along of the dimensions i and j and $\frac{Z}{z}$ tiles along of the dimension k. Tile's volume is equal to $g = x^2 z$, and since the number of available processors is initially known, the only unknown parameter is z.

We applied both vertical and hyperplane grouping, using both blocking and non-blocking communication primitives. For each exemplary Iteration Space and each possible tile height, we calculated the total execution time for the above schemes. In order to implement these schemes we used Linux POSIX threads with semaphores for the synchronization among the processors of an SMP node and the SISCI driver and libraries for the communication among the SMP nodes.

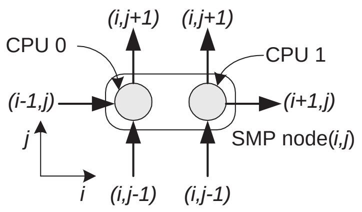

Figure 7. CPU communication directions

Table 1. Non-overlapping scheme Implementation

Thread 0	Thread 1
foreach group assigned	foreach group assigned
to node(i,j) do{	to node(i,j) do{
receive from node(i-1,j)	
receive from node(i,j-1)	receive from node(i,j-1)
compute_tile(i,j,k,CPU0)	compute_tile(i,j,k,CPU1)
	send to node(i+1,j)
send to node(i,j+1)	send to node(i,j+1)
semaphore_post(sem_s1)	semaphore_post(sem_s2)
semaphore_wait(sem_s2)	semaphore_wait(sem_s1)
}	}

First of all, as far as the implementation of vertical grouping is concerned, we experimentally verified formula (4), in order to find the optimal execution time for a couple of tiles by an SMP node. Once vertical grouping was implemented and precisely approximated with a theoretical formula, we implemented both blocking and non-blocking communication schemes. As far as the blocking communication scheme is concerned, it was implemented using the pseudo-code of Table 1. On the other hand, the non-blocking scheme was implemented using the pseudo-code of Table 2, because during each time step, every SMP node in the ij plane with coordinates (i, j) receives from neighboring nodes $(i-1, j)$ and $(i, j-1)$, computes and sends to nodes $(i+1, j)$,$(i, j+1)$ (Fig. 7). Since the `send_dma()` call is not blocking, the computation of the tiles will be performed concurrently with the transferring of data among the SMP nodes. After the execution of `wait_dma()`, it is assured that both computation and communication are already completed.

The implementation of vertical and hyperplane grouping was achieved by a proper `compute_tile(i,j,k,CPUx)` procedure. In order to implement vertical grouping we used the pseudocode of

Table 3. Vertical vs. Hyperplane Grouping

Vertical grouping	
compute_tile(i,j,k,CPU0):	compute_tile(i,j,k,CPU1):
foreach subtile of this tile do{	foreach subtile of this tile do{
compute each iteration	
of this subtile	
semaphore_post(sem1)	semaphore_post(sem2)
semaphore_wait(sem2)	semaphore_wait(sem1)
	compute each iteration
	of this subtile
}	}

Hyperplane grouping	
compute_tile(i,j,k,CPU0):	compute_tile(i,j,k,CPU1):
compute each iteration of this tile	compute each iteration of this tile

Table 2. Overlapping scheme Implementation

Thread 0	Thread 1	Explanation
foreach group assigned to node(i,j) do{	foreach group assigned to node(i,j) do{	
trigger_interrupt to node(i-1,j)		Inform "previous" nodes:
trigger_interrupt to node(i,j-1)	trigger_interrupt to node(i,j-1)	"I am ready to accept data"
	wait_interrupt from node(i+1,j)	Wait until "next" nodes
wait_interrupt from node(i,j+1)	wait_interrupt from node(i,j+1)	are ready to accept data
	send_dma(node(i+1,j),data)	Initialization of DMA transfer
send_dma(node(i,j+1),data)	send_dma(node(i,j+1),data)	to neighboring nodes
compute_tile(i,j,k,CPU0)	compute_tile(i,j,k,CPU1)	
	wait_dma()	Wait for DMA to complete
wait_dma()	wait_dma()	
	trigger_interrupt to node(i+1,j)	Inform "next" nodes:
trigger_interrupt to node(i,j+1)	trigger_interrupt to node(i,j+1)	"Your data has arrived"
wait_interrupt from node(i-1,j)		Wait until "previous" nodes
wait_interrupt from node(i,j-1)	wait_interrupt from node(i,j-1)	have finished sending data
semaphore_post(sem_s1)	semaphore_post(sem_s2)	Implementation of a barrier
semaphore_wait(sem_s2)	semaphore_wait(sem_s1)	
}	}	

Table 3. The number of subtiles inside a tile was selected according to formula (4). Notice that, the implementation of hyperplane grouping was much simpler as it is shown in Table 3.

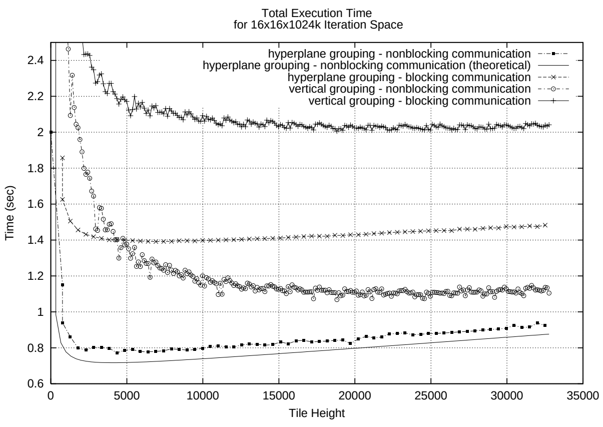

Figure 8. Experimental Results

The problem was solved using various values of $X = Y$ and Z. For each schedule, we are interested in the overall minimum execution time achieved at an optimally selected tile height (see [4, 12, 5]). The experimental results, shown in Figs 8-9, illustrate that in every case non-blocking communication is preferable to blocking communication and hyperplane grouping is preferable to vertical grouping. The lowest minimum is clearly achieved when using hyperplane grouping in combination with non-blocking communication, in all cases.

As far as hyperplane grouping in combination with non-blocking communication is concerned, according to our scheduling theory, as in Example 2, the number of time steps required for the completion of an experiment is $P(x,y,z) = \frac{3X}{2x} + \frac{2Y}{y} + \frac{Z}{z} - 4$. The minimum duration of a time step, as mentioned in §3.5, is $(t_{start\_dma} + t_{comp} + t_{synchro})$. Thus, $T_{non-blocking,hyperplane} = (\frac{3X}{2x} + \frac{2Y}{y} + \frac{Z}{z} - 4)(t_{start\_dma} + t_{comp} + t_{synchro})$. This formula was used to produce the theoretical curves of Figs 8-9 with values $t_{start\_dma} + t_{synchro} = 100\mu sec$ and $t_{tile\_comp} = x^2 z t_{comp1}$, where t_{comp1} is the execution time of a single iteration and it was measured equal to $39,6nsec$.

One can easily verify from Figs 8-9 that the graphs of the theoretical model are very close to the corresponding experimental graphs not only at the desired minimum, but along the whole graph. Thus, the theoretical model of scheduling is strongly verified by the experimental results.

5 Conclusions – Future Work

In this paper we presented a novel approach for the time scheduling of tiled nested loops on a cluster of SMP nodes. We minimized the total execution time by overlapping the computation with communication (as in [4, 12]). In addition, we achieved the maximum CPU's utilization with a proper grouping transformation. What remains open is an analytical computation of the parameters $m_1, \ldots, m_n$ of our grouping matrix according to the initial space shape and communication minimization criteria and an adaption of our theory for a cluster with a fixed number of SMP nodes.

References

[1] T. Andronikos, N. Koziris, G. Papakonstantinou, and P. Tsanakas. Optimal Scheduling for UET/UET-UCT Generalized N-Dimensional Grid Task Graphs. *Journal of Parallel and Distributed Computing*, 57(2):140–165, May 1999.

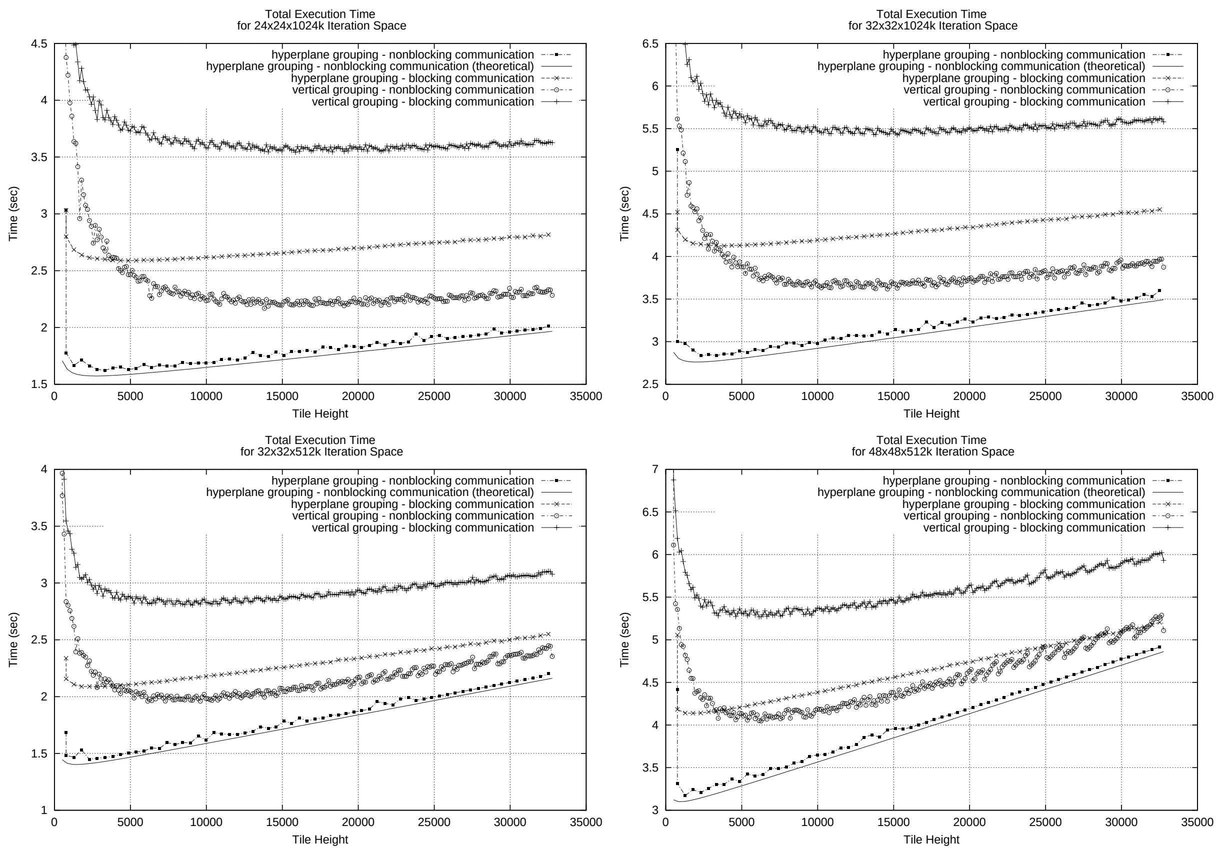

Figure 9. Experimental Results

[2] P. Boulet, A. Darte, T. Risset, and Y. Robert. (Pen)-ultimate tiling? *INTEGRATION, The VLSI Jounal*, 17:33–51, 1994.

[3] I. Drossitis, G. Goumas, N. Koziris, G. Papakonstantinou, and P. Tsanakas. Evaluation of Loop Grouping Methods based on Orthogonal Projection Spaces. In *Proceedings of the International Conference on Parallel Processing*, pages 469–476, Toronto, Canada, Aug 2000.

[4] G. Goumas, A. Sotiropoulos, and N. Koziris. Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping. In *Proceedings of IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS'01)*, San Francisco, Apr 2001. (best paper award).

[5] E. Hodzic and W. Shang. On Supernode Transformation with Minimized Total Running Time. *IEEE Trans. on Parallel and Distributed Systems*, 9(5):417–428, May 1998.

[6] F. Irigoin and R. Triolet. Supernode Partitioning. In *Proc. 15th Ann. ACM SIGACT-SIGPLAN Symp. Principles of Programming Languages*, pages 319–329, San Diego, California, Jan 1988.

[7] C.-T. King, W.-H. Chou, and L. Ni. Pipelined Data-Parallel Algorithms: Part ii Design. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):430–439, Oct 1991.

[8] D. Patterson and J.Hennessy. *Computer Organization & Design. The Hardware/Software Interface*, pages 364–367. Morgan Kaufmann Publishers, San Francisco, CA, 1994.

[9] J. Ramanujam and P. Sadayappan. Tiling Multidimensional Iteration Spaces for Multicomputers. *Journal of Parallel and Distributed Computing*, 16:108–120, 1992.

[10] J.-P. Sheu and T.-S. Chen. Partitioning and Mapping Nested Loops for Linear Array Multicomputers. *Journal of Supercomputing*, 9:183–202, 1995.

[11] J.-P. Sheu and T.-H. Tai. Partitioning and Mapping Nested Loops on Multiprocessor Systems. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):430–439, Oct 1991.

[12] A. Sotiropoulos, G. Tsoukalas, and N. Koziris. Enhancing the Performance of Tiled Loop Execution onto Clusters using Memory Mapped Network Interfaces and Pipelined Schedules. In *to appear in Proceedings of the 2002 Workshop on Communication Architecture for Clusters (CAC'02), Int'l Parallel and Distributed Processing Symposium (IPDPS'02)*, Fort Lauderdale, Florida, Apr 2002.

[13] P. Tsanakas, N. Koziris, and G. Papakonstantinou. Chain Grouping: A Method for Partitioning Loops onto Mesh-Connected Processor Arrays. *IEEE Trans. on Parallel and Distributed Systems*, 11(9):941–955, Sep 2000.

[14] J. Xue. Communication-Minimal Tiling of Uniform Dependence Loops. *Journal of Parallel and Distributed Computing*, 42(1):42–59, 1997.

[15] J. Xue. On Tiling as a Loop Transformation. *Parallel Processing Letters*, 7(4):409–424, 1997.