Scheduling of Tiled Nested Loops onto a Cluster with a Fixed Number of SMP Nodes

Maria Athanasaki, Evangelos Koukis and Nectarios Koziris
National Technical University of Athens
School of Electrical and Computer Engineering
Computing Systems Laboratory
e-mail: {maria, vkoukis, nkoziris}@cslab.ece.ntua.gr

Abstract

In this paper we propose several alternative methods for the compile time scheduling of Tiled Nested Loops onto a fixed size parallel architecture. We investigate the distribution of tiles among processors, provided that we have chosen either a non-overlapping communication mode, which involves successive computation and communication steps, or an overlapping communication mode, which supposes a pipelined, concurrent execution of communication and computations. In order to utilize the available processors as efficiently as possible, we can either adopt a cyclic assignment schedule, or assign neighboring tiles to the same CPU, or adapt the size and shape of tiles, so that the required number of processors is exactly equal to the number of the available ones. We theoretically and experimentally compare the proposed schedules, so as to design one which achieves the minimum total execution time, depending on the cluster configuration, (i.e. number and type of nodes, interconnect bandwidth, etc) the internal characteristics of the underlying architecture (i.e. NIC and DMA latencies, etc) and the iteration space size and shape.

1. Introduction

When optimizing a code segment, our main concern should be the minimization of run time of the computation intensive program structures, such as nested `for`-loops. Nested `for`-loops are very usual in scientific codes using numerical analysis (e.g. in maths, physics, image processing, biology, etc.) Thus, their efficient parallelization can vastly accelerate a great category of scientific programs. However, in order to achieve it, we should take care not to impose an excessive communication load. Several methods have been proposed and applied in the literature, aiming at this specific problem, among which the most general and widely used one has been "tiling". Tiling transformation was proposed by Irigoin and Triolet in [15]. They introduced the initial model of loop tiling and gave conditions for a tiling transformation to be valid. Later, the optimal tile shape has been precisely determined and more accurate conditions have been given in [9, 13, 14, 21, 24].

As far as the execution of tiles on a cluster of PCs is concerned, all conventional approaches [12, 21] consider that 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. Taking into account that modern network interfaces allow for concurrent communication and computation, in [11] we proposed an alternative method for the problem of scheduling the tiles to single CPU nodes. The proposed method acts like enhancing the performance of a processor's datapath with pipelining [20], 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. Such a pipelined execution scheme was proven [22] to nearly double the performance of the algorithms, provided that we use modern NICs (Network Interface Cards), capable of performing communication without annoying the CPU, and advanced communication protocols (i.e. VIA) with Zero-Copy [8], DMA support and User-Level [5] characteristics.

In [4] the method proposed in [11] has been extended for executing Tiled Iteration Spaces in SMP nodes (Symmetric MultiProcessors). We grouped together neighboring tiles along a hyperplane. Hyperplane-grouped tiles are concurrently executed by the CPUs of the same SMP node. In this way, we eliminated the need for tile synchronization and communication between intranode CPUs. As far as scheduling of groups is concerned, we took advantage of the overlapping execution scheme of [11] in order to "hide" each group communication volume within the respective computation volume.

However, the proposed schedule assumes the availability of an unlimited number of SMP nodes. In [3] Andronikos et al. have proposed an assignment scheme onto a fixed number of nodes, however the complexity of evaluating which tiles should be assigned to which node is too high. In [6, 7] Boulet et al. and Calland et al. have theoretically proven the optimality of a cyclic assignment of 2-dimensional tiles onto a fixed number of single CPU nodes. On the other hand, Manjikian and Abdelrahman have presented in [18] an alternative method for scheduling Tiled Iteration Spaces onto a fixed number of SMP nodes, without taking into account, however, that there is no need for communication among CPUs of the same SMP node, since the data required are located in the node's shared memory.

In this paper, we propose four different methods for scheduling tiled iteration spaces onto an existing clustered system with a fixed number of SMP nodes: the cyclic assignment schedule, the mirror assignment schedule, the cluster assignment schedule and the retiling schedule. Firstly, we adapt the method proposed in [4] for a cluster of SMPs with a fixed number of nodes. We discuss the approaches of [6, 7, 18] and generalize them for n-dimensional Spaces, taking into account the particularity of immediate exchange of data among CPUs of the same SMP node. In addition, we apply to all four schedules, two alternative execution schemes, the overlapping [11] and the non-overlapping [12] communication scheme and we discuss the merits and drawbacks of each combined approach.

The rest of this paper is organized as follows: In Section 2 we provide the mathematical background and terminology used throughout the paper and we briefly revise concepts, such as grouping transformation, described in our previous work. In Section 3 we adapt the theory proposed in [4] for a fixed number of SMP nodes, using four different mapping methods. In Section 4 we use some exemplary Iteration Spaces, so as to experimentally delve into the advantages of each schedule. We deduce that our experimental results strongly confirm our theory. Finally, in Section 5 we summarize our conclusions.

2. Algorithmic model - Notation

Our proposed method can be applied to any code segment which can be transformed into a Tiled Iteration Space. However, without lack of generality, in this paper our model consists of perfectly nested `for`-loops with uniform data dependencies, as in [4],[11].

Throughout this paper, the following notation is used: N is the set of natural numbers, n is the number of nested `for`-loops of the algorithm. $J^n \subset Z^n$ is the set of loop 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.

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.

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 resulting Tile Space J^S is defined as follows: $J^S = \{j^S | j^S = \lfloor Hj \rfloor, j \in J^n\}$. It can be also written as $J^S = \{j^S(j_1^S, \ldots, j_n^S) | j_i^S \in Z \wedge l_i^S \leq j_i^S \leq u_i^S, 1 \leq i \leq n\}$, where l_i^S, u_i^S can be directly computed from the functions $l_1, \ldots, l_n, u_1, \ldots, u_n$ and the tiling matrix H, as described in [1, 10].

In the rest of this paper we shall consider that the non-overlapping and overlapping execution schemes, extensively discussed in [11] (sections 3,4), [22] and the concept of grouping, introduced in [4] (section 4) are known.

For example, let us consider an n-dimensional rectangular Tile Space J^S, whose bounds are defined as follows: $0 \leq j_i^S < u_i^S, i = 1, \ldots, n$ and $u_1^S \geq u_i^S, i = 2, \ldots, n$. It is grouped according to the matrix

$$H^G = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 0 & \frac{1}{m_2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \frac{1}{m_n} \end{bmatrix}$$

Thus, a tile j^S belongs to group $j^G = (\sum_{i=1}^{n} j_i^S, \lfloor \frac{j_2^S}{m_2} \rfloor, \ldots, \lfloor \frac{j_n^S}{m_n} \rfloor)^T$. Following the overlapping execution scheme, if there are as many SMP nodes as required, it will be executed in the SMP node $(j_2^G, \ldots, j_n^G)$ during the time step $t = \sum_{i=1}^{n} j_i^G = \sum_{i=1}^{n} j_i^S + \sum_{i=2}^{n} \lfloor \frac{j_i^S}{m_i} \rfloor$ (according to the scheduling vector $\Pi^G = (1, 1, \ldots, 1)$). Thus, the makespan (number of time steps) of the algorithm will be:

$$P_{unlimited-overlap} = \sum_{i=1}^{n} u_i^S + \sum_{i=2}^{n} \lceil \frac{u_i^S}{m_i} \rceil - 2n + 2. \qquad (1)$$

Similarly, if we follow the non-overlapping execution scheme, then group $j^G = (\sum_{i=1}^{n} j_i^S, \lfloor \frac{j_2^S}{m_2} \rfloor, \ldots, \lfloor \frac{j_n^S}{m_n} \rfloor)^T$ will be executed during the time step $t = j_1^G = \sum_{i=1}^{n} j_i^S$ (according to the scheduling vector $\Pi^G = (1, 0, \ldots, 0)$). Thus, the makespan of the algorithm will be:

$$P_{unlimited-non-overlap} = \sum_{i=1}^{n} u_i^S - n + 1 \qquad (2)$$

In the sequel, we shall present some scheduling strategies of a Tile Space onto a fixed number of SMPs. All the formulas concerning the allocation of tiles or groups to SMP nodes and the time step of their execution can be applied to any convex space, as defined in §2. However, when calculating the makespan of an algorithm, we shall consider a rectangular Tile Space, such as the one aimed by formulas (1) and (2). We consider this simplification is necessary for expressing our ideas without too much complicating our mathematical formulas. In addition, this simplification does not constrain any of the advantages or disadvantages of the proposed methods, apart from those concerning load balancing, which can be more thoroughly investigated in our future work.

3. Scheduling onto a fixed number of SMPs

3.1. Cyclic assignment to SMPs

In [6, 7] the optimality of the cyclic assignment of 2-dimensional tiles onto a fixed number of processors was theoretically proven. However, the calculations in [6, 7] did not take into account the communication overhead involved. Generalizing this approach for n-dimensional tiles and for clusters of SMP nodes, we consider that the available SMP nodes form a virtual $(n-1)$-dimensional mesh of $p_2 \times \ldots \times p_n = p$ SMP nodes. We cyclically assign the groups to the SMP nodes. That is, we assign group j^G to the SMP node $(j_2^G \% p_2, \ldots, j_n^G \% p_n)$, as indicated in Fig. 1. Therefore, each SMP node will execute $\lceil \frac{\lceil \frac{u_2^S}{m_2} \rceil}{p_2} \rceil \times \ldots \times \lceil \frac{\lceil \frac{u_n^S}{m_n} \rceil}{p_n} \rceil = \lceil \frac{u_2^G}{p_2} \rceil \times \ldots \times \lceil \frac{u_n^G}{p_n} \rceil$ rows of groups (where $u_i^G = \lceil \frac{u_i^S}{m_i} \rceil$, $i = 2, \ldots, n$).

If the rows of groups assigned to an SMP node, are executed in lexicographic order, the row $(x, j_2^G, \ldots, j_n^G)$ will be executed in the SMP node $(j_2^G \% p_2, \ldots, j_n^G \% p_n)$ after $\sum_{i=2}^{n} \left[\lfloor \frac{j_i^G}{p_i} \rfloor \prod_{k=i+1}^{n} \lceil \frac{u_k^G}{p_k} \rceil \right]$ rows, imposing a latency of $u_1^S \sum_{i=2}^{n} \left[\lfloor \frac{j_i^G}{p_i} \rfloor \prod_{k=i+1}^{n} \lceil \frac{u_k^G}{p_k} \rceil \right]$ time steps. In addition, as deduced from Fig. 1, the location of a group, relatively to the corresponding chunk origin, is $(j_1^{G\prime}, j_2^G \% p_2, \ldots, j_n^G \% p_n)$, where $j_1^{G\prime} = j_1^S + \sum_{i=2}^{n} j_i^S \% m_i p_i$.

Therefore, if the underlying architecture allows for concurrent execution of computations and communication, following the overlapping execution scheme, group j^G will be

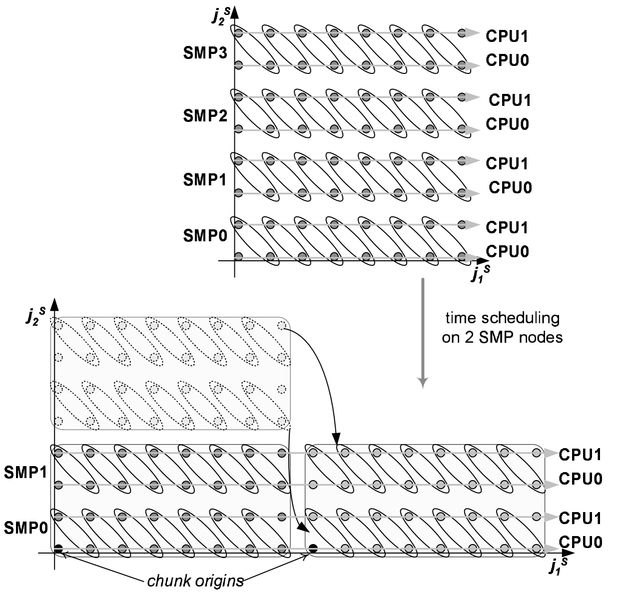

Figure 1. Cyclic assignment to SMPs

computed during the time step $t(j^G) = j_1^{G\prime} + \sum_{i=2}^{n} j_i^G \% p_i + u_1^S \sum_{i=2}^{n} \left[\lfloor \frac{j_i^G}{p_i} \rfloor \prod_{k=i+1}^{n} \lceil \frac{u_k^G}{p_k} \rceil \right]$. Thus, the makespan of the algorithm will be $P_{cyclic-overlap} = \max t(j^G) - \min t(j^G) + 1 \Rightarrow$

$$P_{cyclic-overlap} = \sum_{i=2}^{n} \left[(u_i^S - 1) \% m_i p_i + (\lceil \tfrac{u_i^S}{m_i} \rceil - 1) \% p_i \right] + u_1^S \prod_{i=2}^{n} \lceil \tfrac{u_i^S}{m_i p_i} \rceil . \qquad (3)$$

The first term of the right-hand part in formula (3) represents the time required for filling the pipeline, while the second term corresponds to the time each processor is busy executing calculations.

If we should do with a conventional communication architecture as node interconnect (i.e. without NIC support for relieving the CPU from the communication burden), following the non-overlapping execution scheme, group j^G will be computed during the time step $t(j^G) = j_1^{G\prime} + u_1^S \sum_{i=2}^{n} \left[\lfloor \frac{j_i^G}{p_i} \rfloor \prod_{k=i+1}^{n} \lceil \frac{u_k^G}{p_k} \rceil \right]$. Thus, the makespan of the algorithm will be $P_{cyclic-non-overlap} = \max t(j^G) - \min t(j^G) + 1 \Rightarrow$

$$P_{cyclic-non-overlap} = \sum_{i=2}^{n} \left[(u_i^S - 1) \% m_i p_i \right] + u_1^S \prod_{i=2}^{n} \lceil \tfrac{u_i^S}{m_i p_i} \rceil \qquad (4)$$

3.2. Mirror assignment to SMPs

Let us consider another schedule, if we assign the tiles to SMP nodes as indicated in Fig. 2. That is, we assign group

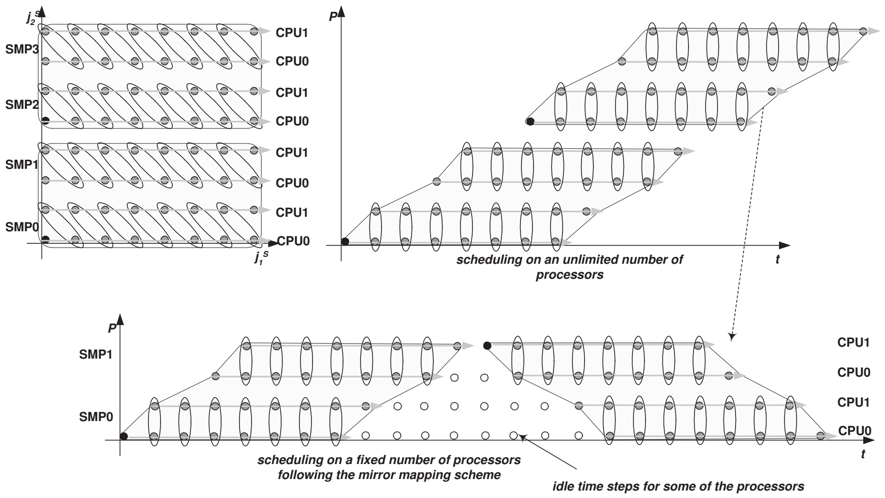

Figure 2. Mirror assignment to SMPs

j^G to the SMP node

$$\left(\begin{array}{c} j_2^G\%p_2 \texttt{ if } even(j_2^G/p_2) \\ (p_2-1)-j_2^G\%p_2 \texttt{ if } odd(j_2^G/p_2) \end{array}, \ldots, \begin{array}{c} j_n^G\%p_n \texttt{ if } even(j_n^G/p_n) \\ (p_n-1)-j_n^G\%p_n \texttt{ if } odd(j_n^G/p_n) \end{array} \right).$$

This schedule has the advantage that there is no need for data transfer along the boundaries of chunks of tiles, thus less time is wasted for communication.

Then, like the cyclic assignment schedule, if the chunks of groups are executed in lexicographic order, the chunk containing row $(x, j_2^G, \ldots, j_n^G)$ will be executed after $\sum_{i=2}^{n}\left[\lfloor\frac{j_i^G}{p_i}\rfloor \prod_{k=i+1}^{n}\lceil\frac{u_k^G}{p_k}\rceil\right]$ chunks. The latency imposed by each of the previous chunks, when combining the mirror assignment schedule with the overlapping execution scheme, is greater than the respective one when applying the cyclic assignment schedule. It, thus, equals to $u_1^S + \sum_{i=2}^{n}[(m_i+1)p_i] - 2n + 2$, as the computation of a whole chunk should be finished before the computation of the next chunk starts. In addition, as deduced from Fig. 2, the position of a group, relatively to the corresponding chunk origin, is $(j_1^{G\prime}, j_2^G\%p_2, \ldots, j_n^G\%p_n)$, where $j_1^{G\prime} = j_1^S + \sum_{i=2}^{n} j_i^S\%m_i p_i$. Therefore, group j^G will be computed during the time step $t(j^G) = j_1^{G\prime} + \sum_{i=2}^{n} j_i^G\%p_i + \left[u_1^S + \sum_{i=2}^{n}[(m_i+1)p_i] - 2n + 2\right]\sum_{i=2}^{n}\left[\lfloor\frac{j_i^G}{p_i}\rfloor \prod_{k=i+1}^{n}\lceil\frac{u_k^G}{p_k}\rceil\right]$. Thus, the makespan of the algorithm will be

$$P_{mirror-overlap} = \max t(j^G) - \min t(j^G) + 1 \Rightarrow$$

$$\begin{aligned} &P_{mirror-overlap} = \\ &= \sum_{i=2}^{n}\left[(u_i^S-1)\%m_i p_i + (\lceil\frac{u_i^S}{m_i}\rceil - 1)\%p_i\right] - \\ &- \sum_{i=2}^{n}[(m_i+1)p_i] + 2n - 2 + \\ &+ \left[u_1^S + \sum_{i=2}^{n}[(m_i+1)p_i] - 2n + 2\right]\prod_{i=2}^{n}\lceil\frac{u_i^S}{m_i p_i}\rceil \end{aligned} \tag{5}$$

If there is no shortage of processors ($u_i^S \leq m_i p_i$, $\forall i = 2, \ldots, n$), the proposed schedules are equivalent. Otherwise, it can be easily deduced from (3),(5) that $P_{cyclic-overlap} < P_{mirror-overlap}$. Their difference is due to the fact that, following the mirror assignment schedule, every time the computation of a chunk finishes and the computation of the next one starts, there are some idle time steps for some of the processors, as indicated in Fig. 2. The cyclic schedule is thus preferable to the mirror one.

Similarly, following the non-overlapping execution scheme, group j^G will be computed during the time step $t(j^G) = j_1^{G\prime} + (u_1^S + \sum_{i=2}^{n} m_i p_i - n + 1)\sum_{i=2}^{n}\left[\lfloor\frac{j_i^G}{p_i}\rfloor \prod_{k=i+1}^{n}\lceil\frac{u_k^G}{p_k}\rceil\right]$. Thus, the makespan of the algorithm will be $P_{mirror-non-overlap} = \max t(j^G) - \min t(j^G) + 1 \Rightarrow$

$$\begin{aligned} &P_{mirror-non-overlap} = \\ &= \sum_{i=2}^{n}\left[(u_i^S-1)\%m_i p_i\right] - \sum_{i=2}^{n} m_i p_i + n - 1 + \\ &+ \left[u_1^S + \sum_{i=2}^{n} m_i p_i - n + 1\right]\prod_{i=2}^{n}\lceil\frac{u_i^S}{m_i p_i}\rceil \end{aligned} \tag{6}$$

It can be deduced from (4),(6) that $P_{cyclic-non-overlap} \leq P_{mirror-non-overlap}$. (They are equivalent only in case there is no lack of processors.) However, since the communication overhead is not hidden under the computation time, this schedule may sometimes result in a shorter total execution time, due to better exploitation of the available bandwidth. In particular, if there are only two SMP nodes along a dimension, no SMP node should both send and receive data along that dimension. Thus, the communication overhead will be halved.

3.3. Cluster assignment to SMPs

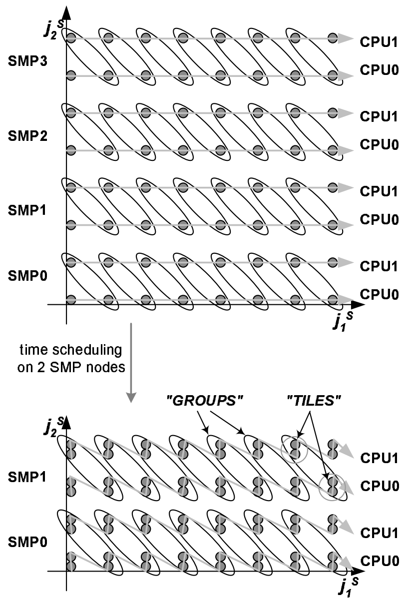

Figure 3. Cluster assignment to SMPs

Alternatively, following the approach of [18], generalizing it for n-dimensional spaces and taking into account that there is no need for communication among processors of the same SMP node, we may assign neighboring rows of tiles to the same CPU, as indicated in Fig. 3. In order to achieve this schedule, we cluster together neighboring tiles $(j_1^S, j_2^S, \ldots, j_n^S)$, mapping them to a supertile or "TILE" labeled as $(j_1^S, \lfloor \frac{j_2^S}{\lceil \frac{u_2}{m_2 p_2} \rceil} \rfloor, \ldots, \lfloor \frac{j_n^S}{\lceil \frac{u_n}{m_n p_n} \rceil} \rfloor)$. Thus, the corresponding "GROUP" will be $j^G = (j_1 + \sum_{i=2}^{n} \lfloor \frac{j_i^S}{\lceil \frac{u_i^S}{m_i p_i} \rceil} \rfloor, \lfloor \frac{j_2^S}{m_2 \lceil \frac{u_2^S}{m_2 p_2} \rceil} \rfloor, \ldots, \lfloor \frac{j_n^S}{m_n \lceil \frac{u_n^S}{m_n p_n} \rceil} \rfloor)$ and, following the overlapping execution scheme, it will be executed during the time "STEP" $t(j^S) = j_1 + \sum_{i=2}^{n} \lfloor \frac{j_i^S}{\lceil \frac{u_i^S}{m_i p_i} \rceil} \rfloor + \sum_{i=2}^{n} \lfloor \frac{j_i^S}{m_i \lceil \frac{u_i^S}{m_i p_i} \rceil} \rfloor$. As a "TILE" consists of $\prod_{i=2}^{n} \lceil \frac{u_i^S}{m_i p_i} \rceil$ tiles, a "STEP" will be equivalent to $\prod_{i=2}^{n} \lceil \frac{u_i^S}{m_i p_i} \rceil$ time steps (excluding the DMA initialization and synchronization time). Thus, the makespan of the algorithm will be $P_{cluster-overlap} = \prod_{i=2}^{n} \lceil \frac{u_i^S}{m_i p_i} \rceil (\max t(j^S) - \min t(j^S) + 1) \Rightarrow$

$$P_{cluster-overlap} = \prod_{i=2}^{n} \lceil \frac{u_i^S}{m_i p_i} \rceil (u_1^S - 2n + 2 + \\ + \sum_{i=2}^{n} \lceil \frac{u_i^S}{\lceil \frac{u_i^S}{m_i p_i} \rceil} \rceil + \sum_{i=2}^{n} \lceil \frac{u_i^S}{m_i \lceil \frac{u_i^S}{m_i p_i} \rceil} \rceil) \tag{7}$$

It can be proven that it always holds $P_{cyclic-overlap} \leq P_{cluster-overlap}$. The proof of this formula is omitted due to lack of space. Thus, this schedule results to a worse number of execution steps than the previous one. Their difference is due to the fact that, in this schedule, the filling of the pipeline is slower. However, the previous mathematical formula has not taken into consideration the time required for the initialization of messages and for synchronization. Since the cluster assignment schedule requires less messages to be sent and less synchronization, in some cases it may be practically proven more efficient.

Similarly, following the non-overlapping execution scheme, tile $(j_1^S, j_2^S, \ldots, j_n^S)$, corresponding to "GROUP" $j^G = (j_1 + \sum_{i=2}^{n} \lfloor \frac{j_i^S}{\lceil \frac{u_i^S}{m_i p_i} \rceil} \rfloor, \lfloor \frac{j_2^S}{m_2 \lceil \frac{u_2^S}{m_2 p_2} \rceil} \rfloor, \ldots, \lfloor \frac{j_n^S}{m_n \lceil \frac{u_n^S}{m_n p_n} \rceil} \rfloor)$ is executed during the time "STEP" $t(j^S) = j_1 + \sum_{i=2}^{n} \lfloor \frac{j_i^S}{\lceil \frac{u_i^S}{m_i p_i} \rceil} \rfloor$. A computation "subSTEP" is equivalent to $\prod_{i=2}^{n} \lceil \frac{u_i^S}{m_i p_i} \rceil$ computation substeps, but a communication "subSTEP" is equivalent to less than $\prod_{i=2}^{n} \lceil \frac{u_i^S}{m_i p_i} \rceil$ communication substeps. In particular, if the communication load is equal along all communication dimensions (as resulted by the method proposed in [24]), the amount of data to be transferred, as indicated in Fig. 4, is $\prod_{i=2}^{n} \lceil \frac{u_i^S}{m_i p_i} \rceil \sum_{i=2}^{n} \frac{1}{(n-1) \lceil \frac{u_i^S}{m_i p_i} \rceil} \leq \prod_{i=2}^{n} \lceil \frac{u_i^S}{m_i p_i} \rceil$ times the communication load of a tile. Thus, the makespan of the algorithm will be $P_{cluster-non-overlap} = C \left(\max t(j^S) - \min t(j^S) + 1 \right)$ (where $1 \leq C \leq \prod_{i=2}^{n} \lceil \frac{u_i^S}{m_i p_i} \rceil$) $\Rightarrow$

$$P_{cluster-non-overlap} = \\ = C \left(u_1^S - n + 1 + \sum_{i=2}^{n} \lceil \frac{u_i^S}{\lceil \frac{u_i^S}{m_i p_i} \rceil} \rceil \right) \tag{8}$$

In conclusion, comparing to the cyclic assignment schedule, this method has the drawback of slower pipeline filling. However, it results to less communication overhead, which significantly reduces the total execution time, especially when the non-overlapping execution scheme is applied.

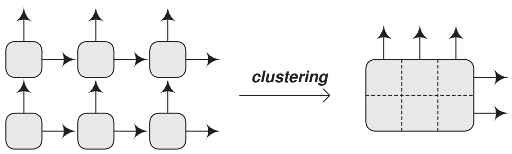

Figure 4. Clustering communication

3.4. Retiling

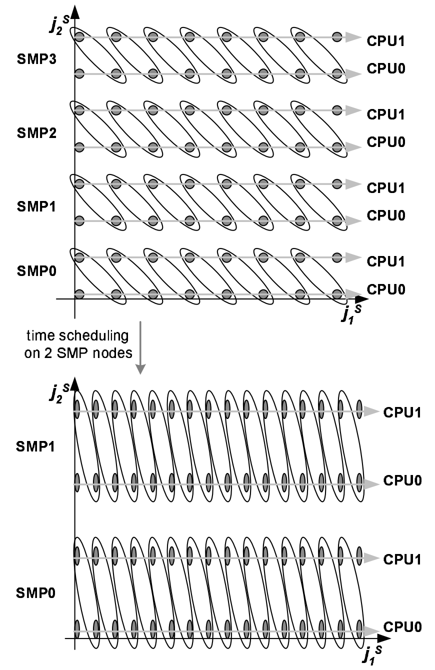

Figure 5. Retiling

A more efficient schedule can be obtained, if we adapt the size of tiles to the available number of SMPs (Fig. 5). That is, we retile the initial Iteration Space, so as to get $u_i^{S'} = m_i p_i$, $(i = 2, \ldots, n)$ and $u_1^{S'} = u_1^S \prod_{i=2}^{n} \frac{u_i^S}{m_i p_i}$. Then, the size of a "new" tile will be equal to the size of an "old" tile and, consequently, a "new" computation step will be equivalent to an "old" computation step. Following the overlapping execution scheme, the makespan of the algorithm, according to the formula (1), will be $P_{retile-overlap} = \sum_{i=1}^{n} u_i^{S'} + \sum_{i=2}^{n} \lceil \frac{u_i^{S'}}{m_i} \rceil - 2n + 2 \Rightarrow$

$$P_{retile-overlap} = \\ = \sum_{i=2}^{n} \left[(m_i + 1) p_i\right] - 2n + 2 + u_1^S \prod_{i=2}^{n} \frac{u_i^S}{m_i p_i}. \quad (9)$$

In case $u_i^S \% m_i p_i = 0$ $(i = 2, \ldots, n)$, it holds that $P_{retile-overlap} = P_{cyclic-overlap}$. Otherwise, $P_{retile-overlap} < P_{cyclic-overlap}$. Their difference is due to the fact that the cyclic schedule does not assign exactly the same number of tiles to each processor, resulting to a load imbalance.

Using the non-overlapping execution scheme, the makespan of the algorithm, according to the formula (2), will be $P_{retile-non-overlap} = \sum_{i=1}^{n} u_i^{S'} - n + 1 \Rightarrow$

$$P_{retile-non-overlap} = \\ = \sum_{i=2}^{n} m_i p_i - n + 1 + u_1^S \prod_{i=2}^{n} \frac{u_i^S}{m_i p_i}. \quad (10)$$

From (4),(10), we can deduce that $P_{retile-non-overlap} \leq P_{cyclic-non-overlap}$. In addition, a "new" computation substep is equivalent to an "old" computation substep, but a "new" communication substep is equivalent to less than an "old" communication substep. In particular, as in the cluster assignment schedule, if the communication load is equal along all communication dimensions, the amount of data to be transferred is $\sum_{i=2}^{n} \frac{1}{(n-1)\frac{u_i^S}{m_i p_i}} \leq 1$ times the communication load of an "old" tile.

In conclusion, in every case this schedule is preferable to previously proposed ones, assuming that there are no factors constraining the tile shape, such as false sharing, or cache locality [16, 17, 23]. It can fully exploit the computational power of all the SMP nodes and it achieves a perfect load balance, without imposing any additional complexity to the initial schedule. But if, apart from parallel scheduling, there are other factors constraining the tile size and shape, this schedule will be proven to be inefficient, since it totally reorganizes the execution order of iterations.

4. Experimental Results

4.1. Experimental platform

In order to evaluate the proposed methods, we use a Linux SMP cluster with 2 identical nodes. Each node has 1GB of RAM and 2 Pentium III @ 1266 MHz CPUs. The cluster nodes communicate through a Myrinet high performance interconnect, using the GM low level message passing system.

In order to utilize the available processors in each SMP node as efficiently as possible, our implementation uses one multi-threaded process per SMP, with the number of threads equal to the number of CPUs. Multithreading support is based on the LinuxThreads library. Threads executing on the same SMP communicate using shared memory, eliminating the need for message passing. For the data exchange between processes executing on different SMPs, Myricom's

GM version 1.6.3 is used [19]. GM is a low-level message passing library for Myrinet. It comprises a library used by userspace programs, an OS driver (in our case, a Linux kernel module) and a Myrinet Control Program (MCP), which is executed on the LANai, the embedded RISC microprocessor on the Myrinet NIC. The GM driver is used during the execution of a userspace process to open and close *ports* and to allocate and free memory suitable for DMA transfers. A port is a communication endpoint, used as the interface between a userspace process and the NIC. Having opened a port, a process can communicate directly with the NIC, without the need for system calls, bypassing the operating system. Thus, all data exchange is performed directly to and from userspace buffers.

To provide flow control between the host and the NIC, sending and receiving messages is regulated by tokens. Initially, a process possesses a finite number of send and receive tokens. To be able to receive a message, the process must provide GM with a buffer in DMAable memory, relinquishing a receive token. When a message is received, the DMA engine on the Myrinet NIC places it directly into the userspace buffer. The process polls for new messages and retrieves the receive token when a message arrives. The same applies to sending messages: The process relinquishes a send token by requesting the transmission of a message from a userspace buffer, then retrieves it when the send operation completes and an appropriate send completion callback function is executed by GM. As the data exchange between the host memory and the NIC is undertaken by the DMA engine on the NIC, without involving the CPU, overlapping of communication with computation is possible.

4.2. Experimental Data

We performed several series of experiments in order to evaluate and compare the practical speedups obtained using each one of the four alternative schedules, combined with both the alternative execution schemes. Our test application code was the following:

```
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 consider, 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 [2, 11]. Each tile has i, j dimensions equal to x and k dimension equal to z. Thus, there are $\frac{X}{x}$ tiles along dimensions i, $\frac{X}{x}$ tiles along dimensions j and $\frac{Z}{z}$ tiles along dimension k.

```
Non Overlapping Execution Scheme
Pre-computation Part of Communication in a tile:
   gm_provide_receive_buffer()
   do
     poll the GM event queue
     process the event
   until data received
Post-computation Part of Communication in a tile:
   gm_send_with_callback()
   do
     poll the GM event queue
     process the event
   until data sent
   Barrier for Threads in SMP
Overlapping Execution Scheme
Pre-computation Part of Communication in a tile:
   If on first tile
     Execute a non-overlapping receive
   gm_provide_receive_buffer() for tile (t1+1,t2,t3)
   gm_send_with_callback() for tile (t1-1,t2,t3)
Post-computation Part of Communication in a tile:
   do
     poll the GM event queue
     process the event
   until send & receive completed
   Barrier for Threads in SMP
   If on last tile
     Execute a non-overlapping send
```

Table 1. Execution Schemes Implementation

After implementing all four schedules in combination with both execution schemes, as described by the pseudocode of Table 1, we measured the performance of all schedules and compared it with their theoretically expected performance. For various tile sizes, we have conducted a series of experiments for each schedule+execution scheme combination, varying the iteration space size. In Fig. 6 we have plotted our experimental results along with the respective theoretical curves. In fact, we have experimented with many more configurations and tile sizes, but, since the graphical results were similar, we chose to show a representative set of measurements. As a measure of performance, we have used the ratio of the speedup obtained to the best possible speedup. That is, we have depicted the ratio of the speedup obtained to the number of processors used. Thus, the closer a plot is to 1, the more efficient a schedule is. As can be seen in Fig. 6, the practical completion times of our experiments differ to our theoretical predictions by at most 3%. For the overlapping communication schedules, this can be attributed to both the DMA engine on the Myrinet NIC and the CPU trying to access data in memory simultaneously.

One can easily deduce that in almost all cases, the retiling schedule achieves the best performance, both theoretically and experimentally. This result was expected, since the retiling schedule absolutely adjusts tiles to the existing configuration of a cluster. However, in our experiments we have eliminated the effect of cache miss penalties by using

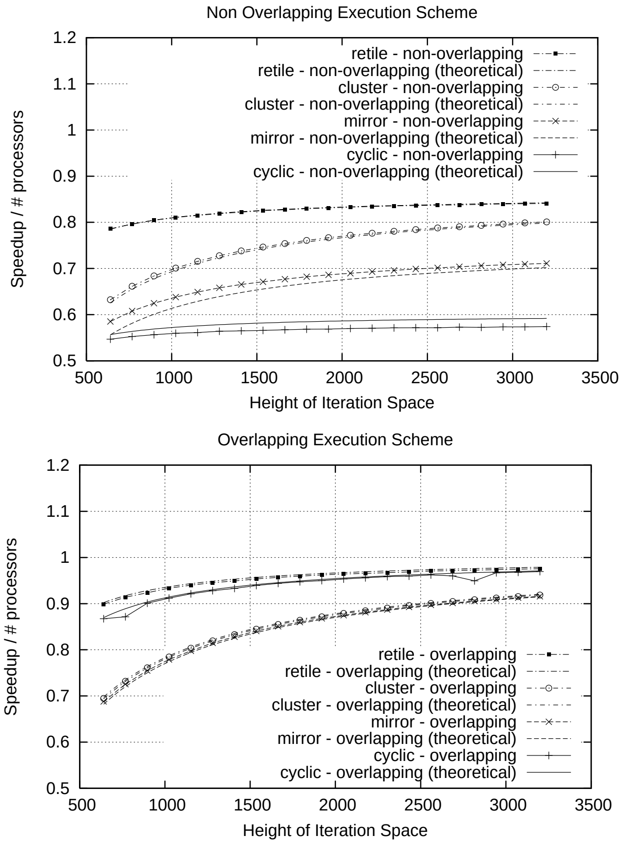

Figure 6. Experimental Data: Tile Size $128 \times 32 \times 32$

small iteration space widths. If our iteration space dimensions which are not assigned to the same processor were too long, the retiling schedule could have destroyed the data locality achieved by optimally selected small tiles.

Note also that the cluster assignment schedule using tile size x is equivalent to the retiling schedule using tile size $4x$. This was expected, considering that by construction the iterations executed and the data sent in these two cases are the same. What differs is the execution order of iterations but here we have eliminated the cache misses overhead, in order to test our schedules' optimality and not data locality.

When following the non-overlapping execution scheme, the difference among the performance of the four schedules is mainly due to the volume of the data to be transferred. As depicted in Fig. 7, the mirror assignment schedule involves double the communication of retiling and cluster assignment schedule, while the cyclic assignment schedule involves 6 times the same communication volume.

When following the overlapping execution scheme, since the communication volume is hidden under computation, their difference is due to the time steps that

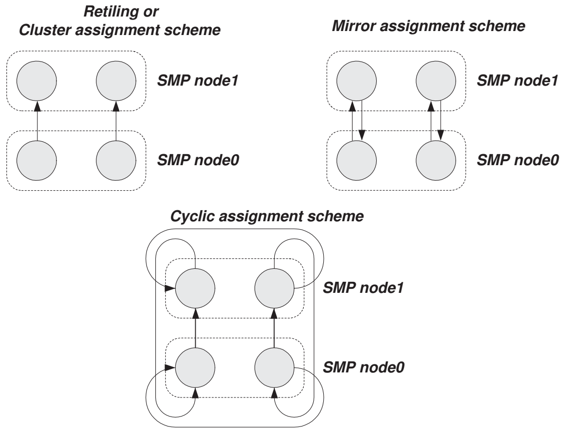

Figure 7. Communication among SMPs

each SMP has to stall waiting for the required data to arrive. The number of these time steps are equal regarding the retiling and the cyclic assignment schedules. However, using the cluster or the mirror assignment schedule, the idle time steps, as depicted in Figs 1, 2 are double.

In addition, note that all schedules achieve better performance for long Iteration Spaces. This is due to the fact that when the mapping dimension of the Iteration Space is comparatively short, the time required for the last processor to start computing after the first data have arrived, is not minor in comparison to the total execution time.

4.3. Simulation Data

The previous experimental data have been obtained on a cluster of 2 SMP nodes with 2 CPUs each. Note in Fig. 7 that in the retiling and the cluster assignment schedule there is no SMP node that should both send and receive data. Thus, we expect that the relative performance of the four schedules would change when scaling up our underlying architecture. In order to evaluate the merits of the proposed schedules, using bigger clusters than the one we had available, we performed a number of simulations, whose results are depicted in Figs 8-9. The performance of all four schedules has been simulated assuming that the initialization of DMA and synchronization overhead is negligible, as deduced from microbenchmarking in our platform. Similar to Fig. 6, the values plotted in Figs 8-9 express, for each proposed schedule, the speedup obtained, divided by the number of CPUs used: $\frac{Speedup}{Number\ \ of\ \ Processors\ \ Used}$.

It can be easily seen that when we are not interested in possible cache miss penalties imposed by reorganizing the Tile Space, the retiling schedule is again the most efficient one, due to the fact that it can fully exploit the com-

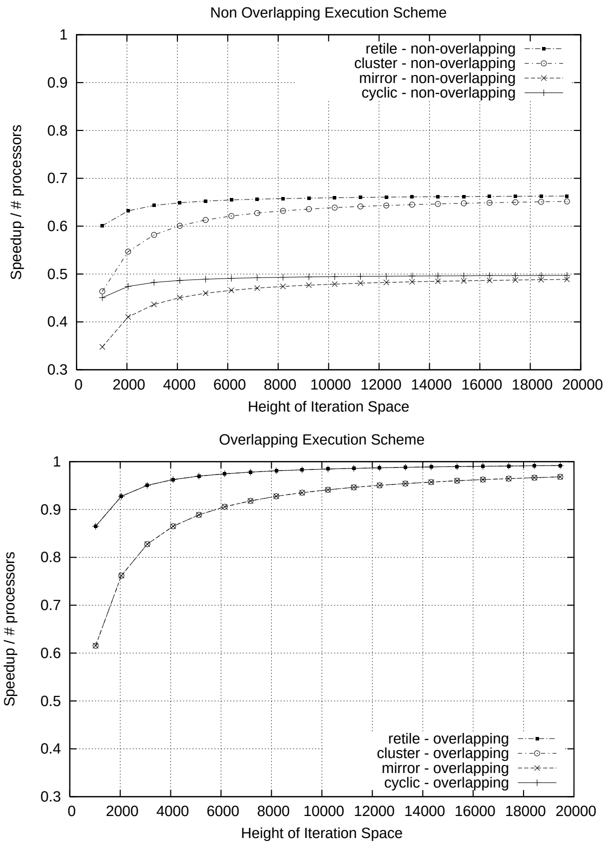

Figure 8. Simulation Data: Tile Space $\ldots \times 16 \times 16$ **on a grid of** 4×4 **nodes with** 2×2 **CPUs each**

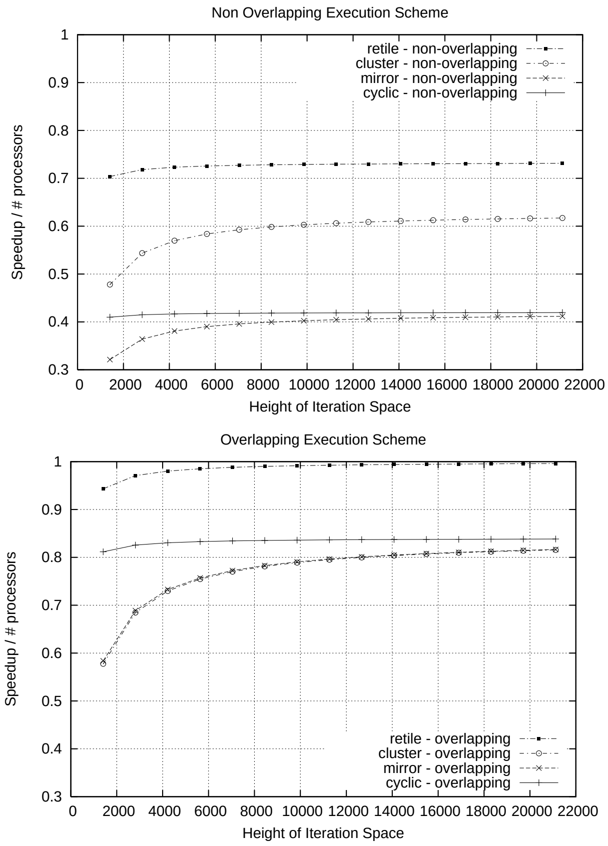

Figure 9. Simulation Data: Tile Space $\ldots \times 22 \times 22$ **on a grid of** 4×4 **nodes with** 2×2 **CPUs each**

putational power of all the SMP nodes and by definition it achieves a perfect load balance.

As far as the cluster assignment schedule is concerned, for small Tile Spaces, it is inefficient due to its slow pipeline filling. However, when the mapping dimension of the Tile Space is long enough, this schedule achieves high speedups, due to the fact that it minimizes the volume of data to be transferred. In fact, as explained in §4.2, the plot representing the cluster assignment schedule will fall onto the plot representing the retiling schedule if we shift it parallely to the x-axis (see Fig. 8). The cluster assignment schedule is less efficient than the retiling schedule, only in case u_i^S is not a multiple of $m_i p_i$ (see Fig. 9), due to load imbalance.

We also deduce that the cyclic assignment schedule is equivalent to the retiling schedule, when the number of tiles along each dimension i is a multiple of $m_i p_i$ and the overlapping execution scheme is used. Otherwise, if u_i^S is not a multiple of $m_i p_i$, their difference is due to the fact that the cyclic schedule does not achieve a perfect load balance. Using the non-overlapping execution scheme, the difference is due to the fact that, as analyzed in Fig. 4 and §3.4, the cyclic schedule results to more communication load, which is not hidden under the computation load. In addition, it can be more efficient than the cluster assignment schedule, only in case we use the overlapping communication scheme. This is due to the fact that in this case the extra communication overhead of the cyclic schedule is hidden under the computation load.

5. Conclusions

In this paper, we presented and experimentally compared four different methods for scheduling Tiled Iteration Spaces onto a cluster with a fixed number of SMPs. We concluded that the most efficient schedule is in most cases obtained when we adapt the size and shape of tiles to the size of the underlying architecture (retiling schedule). However, in case it is not possible, or it is not desired, since tiles are already optimally selected considering data locality [16, 17, 23], we propose either a cyclic assignment schedule, or clustering together neighboring tiles and handling them as a super-tile. The cyclic approach is preferable when the communication and computation substeps can be overlapped. In the opposite case, we propose the cluster assignment schedule, which considerably reduces the volume of data to be transferred.

Acknowlegement

Maria Athanasaki is partially supported by a research student scholarship, awarded by the A.S. Onassis public benefit foundation.

References

[1] C. Ancourt and F. Irigoin. Scanning Polyhedra with DO Loops. In *Proceedings of the Third ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPoPP)*, pages 39–50, Williamsburg, VA, Apr 1991.

[2] 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.

[3] T. Andronikos, N. Koziris, G. Papakonstantinou, and P. Tsanakas. Optimal Scheduling for UET-UCT Grids Into Fixed Number of Processors. In *Proceedings of 8th Euromicro Workshop on Parallel and Distributed Processing (PDP2000), IEEE Press*, pages 237–243, Rhodes, Greece, Jan 2000.

[4] M. Athanasaki, A. Sotiropoulos, G. Tsoukalas, and N. Koziris. Pipelined Scheduling of Tiled Nested Loops onto Clusters of SMPs using Memory Mapped Network Interfaces. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing (SC2002)*, Baltimore, Maryland, Nov 2002. IEEE Computer Society Press.

[5] M. Blumrich. *Network Interface for Protected, User-Level Communication*. PhD thesis, Princeton University, Apr 1996.

[6] P. Boulet, J. Dongarra, Y. Robert, and F. Vivien. Tiling for Heterogeneous Computing Platforms. Technical Report UT-CS-97-373, University of Tennessee, Knoxville, 1997.

[7] P. Y. Calland, J. Dongarra, and Y. Robert. Tiling with Limited Resources. In L. Thiele, J. Fortes, K. Vissers, V. Taylor, T. Noll, and J. Teich, editors, *Application Specific Systems, Architectures, and Processors, ASAP'97, IEEE Computer Society Press*, pages 229–238, Jul 1997. Extended version available on the web at http://www.ens-lyon.fr/∼yrobert.

[8] F. O. Carroll, H. Tezuka, A. Hori, and Y. Ishikawa. The Design and Implementation of Zero Copy MPI Using Commodity Hardware with a High Performance Network. In *Proceedings of the International Conference on Supercomputing*, pages 243–249, Melbourne, Australia, 1998.

[9] F. Desprez, J. Dongarra, and Y. Robert. Determining the Idle Time of a Tiling: New Results. *Journal of Information Science and Engineering*, 14:167–190, Mar 1997.

[10] G. Goumas, M. Athanasaki, and N. Koziris. An Efficient Code Generation Technique for Tiled Iteration Spaces. *IEEE Trans. on Parallel and Distributed Systems*, 14(10):1021–1034, Oct 2003.

[11] 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.

[12] 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.

[13] E. Hodzic and W. Shang. On Time Optimal Supernode Shape. *IEEE Trans. on Parallel and Distributed Systems*, 13(12):1220–1233, Dec 2002.

[14] K. Hogstedt, L. Carter, and J. Ferrante. Determining the Idle Time of a Tiling. In *Principles of Programming Languages (POPL)*, pages 319–323, Jan 1997.

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

[16] M. Kandemir, J. Ramanujam, and A. Choudary. Improving Cache Locality by a Combination of Loop and Data Transformations. *IEEE Trans. on Parallel and Distributed Systems*, 48(2):159–167, Feb 1999.

[17] M. Lam, E. Rothberg, and M. Wolf. The Cache Performance and Optimizations of Blocked algorithms. In *Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 63–74, Santa Clara, California, Apr 1991.

[18] N. Manjikian and T. S. Abdelrahman. Exploiting Wavefront Parallelism on Large-Scale Shared-Memory Multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 12(3):259–271, Mar 2001.

[19] Myricom. GM: A Message-Passing System for Myrinet Networks, 2002. http://www.myri.com/scs/GM/doc/html.

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

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

[22] 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 *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.

[23] M. Wolf and M. Lam. A Data Locality Optimizing Algorithm. In *ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI)*, Toronto, Ontario, Jun 1991.

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