Communication-aware Supernode Shape

Georgios Goumas, Nikolaos Drosinos and Nectarios Koziris
National Technical University of Athens
School of Electrical and Computer Engineering
Computing Systems Laboratory
Zografou Campus, Zografou 15780, Greece
{goumas, ndros, nkoziris}@cslab.ece.ntua.gr

***Abstract*— In this paper we revisit the supernode-shape selection problem, that has been widely discussed in bibliography. In general, the selection of the supernode transformation greatly affects the parallel execution time of the transformed algorithm. Since the minimization of the overall parallel execution time via an appropriate supernode transformation is very difficult to accomplish, researchers have focused on scheduling-aware supernode transformations that maximize parallelism during the execution. In this paper we argue that the communication volume of the transformed algorithm is an important criterion, and its minimization should be given high priority. For this reason we define the metric of the per process communication volume and propose a method to miminize this metric by selecting a communication-aware supernode shape. Our approach is equivalent to defining a proper Cartesian process grid with `MPI_Cart_Create`, which means that it can be incorporated in applications in a straightforward manner. Our experimental results illustrate that by selecting the tile shape with the proposed method, the total parallel execution time is significantly reduced due to the minimization of the communication volume, despite the fact that a few more parallel execution steps are required.**

***Index Terms*— Loop tiling, supernode transformation, tile shape, MPI, process grid, scheduling.**

I. Introduction

Tiling or supernode [1] transformation has been proposed as one of the most efficient methods to map applications based on stencils onto distributed-memory architectures with significant commucation latencies. Stencil computations are very frequently met in image processing and in simulation applications resulting from the discretization of PDEs using explicit finite-difference schemes [2]–[4]. Such applications are essentially *DOACCROSS* loop nests, i.e. n-dimensional loop nests with at least n linearly independent data dependencies. In order to execute *DOACCROSS* nested loops in the aforementioned parallel architectures, researchers have proposed the application of tiling transformation [5]. Tiling groups neighboring iterations into one computational unit, the tile or supernode. For the parallel execution of tiled iteration spaces, tiles are assigned to the available processes which are orchestrated to communicate before and after the computation within one tile. In this way, both the communication volume and frequency are reduced enabling *DOACCROSS* nested loops, that suffer from high communication needs, to efficiently execute onto parallel platforms where remote memory access times are crucially larger than local access times and communication startup latencies are an important hurdle to high performance.

Tiling for coarse-grain parallelism has attracted extensive scientific research [5]–[26] right after its presentation by Irigoin and Triolet in 1988 [1]. Tiling transformation provides flexibility concerning the number of iterations to be grouped together into a single tile (tile size), as well as the shape of the enclosing parallelogram. Since the selection of the tile size and shape greatly affects the properties of the transformed space, researchers have focused on defining criteria for an efficient tiling transformation. Ohta et. al. [17], Hodzic and Shang [11], and Andonov et. al. [27] focused on the selection of the optimal tile size based on the special characteristics of the application and the target architecture. Ramanujam and Sadayappan [5], Boulet et. al. [7] and Xue [21] worked on the selection of a tile shape that minimizes the *per tile* communication volume, i.e. their goal was to minimize the dependence vectors cutting the planes defining a tile. In this case, the optimal tile shape is formed by planes parallel to the algorithm's dependence cone. More importantly, for a given tile size, Hodzic and Shang [11], [14] and Högstedt et. al. [15], [28] determined the tile shape that minimizes the parallel execution steps of the tiled space. In this case, the *scheduling-aware* tile shape is obtained by: (a) deciding on an appropriate basic tile shape (in most cases, tile sides are again parallel to the dependence cone) and (b) properly scaling the sides of the tile, in order to minimize the maximum parallel execution path between the first and the last tile.

Determining the optimal tiling transformation, i.e. the one that minimizes parallel execution times, is very difficult, since various tiling transformations lead to different transformed (tiled) iteration spaces, memory access patterns, communication granularities, processor idle times and communication volumes. In order to evaluate the impact of each of the above factors to the overall execution time, one needs to devise a highly accurate parallel execution model that takes into consideration all the above factors. However, such an execution model is almost impossible to exist for the extremely complex modern parallel platforms. For this reason, the researchers simplify their approaches by discarding some of the above factors, paying the cost of suboptimal tiling transformations. For example, Högstedt et. al. in [15], [28] determine the shortest path between the first and the last tile without taking into consideration the communication overhead. Hodzic and Shang in [11], [14] propose a tile-shape selection technique assuming constant communication times for all message sizes. This approach disregards the communication data volume with the assumption that the volume-dependent message transmission is overlapped by useful computations and thus hidden. However,

[1]Throughout relevant research papers the terms *supernode* and *tiling* transformation have been used to describe the same transformation. Although we adopt the term tiling, we have used the term supernode in our title and in several sections of this manuscript as a reference to the seminal paper of Irigoin and Triolet [1].

as shown in [9] and [16], one needs to apply special scheduling strategies (that are not considered in [11], [14]) and employ sophisticated communication hardware in order to hide some part –and not all– of the transmission time.

In this paper we focus on a special but important class of problems that are frequently met in practice. We assume rectangular iteration spaces (as in [11]) and non-negative elements in dependence vectors. Note that, a tiling transformation can be uniquely defined by determining three parameters: (a) tile size, (b) basic tile shape and (c) scaling factors of tile sides. In our approach, we consider the tile size as an input parameter determined by the computation and communication costs of the algorithm and the hardware features of the target architecture. In addition, we consider rectangular basic tile shapes. A general, parallelogram tiling transformation can only be implemented by automatic parallelizing compilers due to the complexity of the code that traverses non-rectangular tiles [10], [12]. For the above problem class we propose a new criterion for the selection of an efficient tile shape. This criterion emphasizes the minimization of the *per process* communication volume. Note that minimizing the communication overhead is the primary goal of tiling transformation, therefore trying to further decrease the communication data by properly selecting the tile shape seems a good idea in the first place. The problem arises in the cases where the scheduling-aware tile shape differs from the *communication-aware* tile shape proposed here. In this paper we demonstrate that the criterion for communication minimization should be given the greatest priority when targeting distributed memory architectures, since it is the one that more drastically affects the overall execution time of the parallel algorithm.

Our method takes into consideration the boundaries of the initial iteration space and the dependencies of the original algorithm, and can be applied on a distributed memory architecture for a limited (fixed) number of processes. This selection of the tile scaling factors is equivalent to determining a virtual process topology, and thus can be easily incorporated in a message-passing programming environment like MPI, with a proper initialization to the parameters of the `MPI_Cart_Create` routine. Our experimental results indicate that the proposed communication-aware tile shape significantly reduces the overall execution time, compared to the one achieved by the scheduling-aware tile shape, although it requires a larger number of parallel execution phases.

The rest of the paper is organized as follows: Section II provides useful background knowledge and basic definitions concerning the program model, supernode (tiling) transformation, scheduling and mapping techniques. Section III discusses in more detail work concerning criterions and methods for the selection of efficient tiling transformations, while Section IV defines our problem and proposes a method to select tiling transformations that minimize the per process communication volume. Section V experimentally tests the efficiency of the proposed approach in terms of total parallel execution times and compares it to the selections proposed by previous research. Finally, Section VI provides the overall conclusions drawn from this paper.

II. Preliminaries

A. Algorithmic model

Our algorithmic model concerns stencil applications, which involve $(n+1)$-dimensional perfectly nested loops with constant flow dependencies. The iteration space J^{n+1} is rectangular, thus it holds $J^{n+1} = \{\vec{j}(j_1, j_2, \ldots, j_{n+1}) \in Z^{n+1} \wedge l_i \leq j_i \leq u_i,\ i = 1 \ldots n+1\}$, where $l_i, u_i \in Z$ are the lower and upper bounds of the i-th loop respectively. The dependencies of the problem are expressed with constant, $(n+1)$-dimensional dependence vectors $\vec{d_i}$, $i = 1 \ldots m$. We denote $\vec{d}_{ij}$ the j-th element of vector $\vec{d_i}$. In the class of problems under consideration it holds $\vec{d}_{ij} \geq 0$, $i = 1 \ldots m$ and $j = 1 \ldots n+1$. The dependence matrix of the algorithm, denoted D, is an $(n+1) \times m$ matrix containing as columns the dependence vectors of the algorithm. The reader is referred to [37] for more details on the properties of data dependencies. It holds $rank(D) = n+1$, which means that the algorithm has $n+1$ linearly independent data dependence vectors. Note that, if $rank(D) < n+1$, then the iteration space can be partitioned into independent subspaces and parallelized without the use of tiling [29]. We define the vector $\vec{d'} = (d'_1, d'_2, \ldots, d'_{n+1})$ with $d'_i = max(d_{il}), l = 1 \ldots m$, which expresses the maximum dependence length per dimension. Unlike [23], [30] we consider in-core computations, i.e. all data sets assigned to each process fit in main memory, thus we do not consider secondary storage access times. Overall, the algorithms have the general form of Algorithm 1, where U is an $(n+1)$-dimensional matrix and F is a linear function.

Algorithm 1: algorithmic model

```
1 for j_1 ← l_1 to u_1 do
2     ...
3     for j_n ← l_n to u_n do
4         for j_{n+1} ← l_{n+1} to u_{n+1} do
5             U[j⃗] = F(U[j⃗ − d⃗_1], ..., U[j⃗ − d⃗_m]);
```

B. Application example: advection equation

Advection is the physical process of transportation within a fluid, as is for example the transportation of polluted particles in the atmosphere. Advection phenomena are very commonly studied in meteorology. The advection equation is the partial differential equation (PDE) that governs the motion of a conserved scalar as it is advected by a known velocity field (the material in which advection occurs). The advection equation for a scalar v (e.g. particle density or temperature) is expressed mathematically as:

$$\frac{\partial v}{\partial t} = \vec{a} \bigtriangledown v$$

where $\vec{a}$ is the vector field, e.g. the velocity vector of the material. In two spatial dimensions the above equation is equivalent to:

$$\frac{\partial v}{\partial t} = a_x \frac{\partial v}{\partial x} + a_y \frac{\partial v}{\partial y} \quad (1)$$

If we need to study an advection process in a $X \times Y$ space for a time window T, we can discretize the initial domain into a uniform grid using a time step Δt and space steps Δx and Δy. Then, we can discretize the above PDE using a variety of finite differencing schemes. For example, if we employ the *Euler-Forward* scheme [31], the time derivative can be substituted by a fraction of differences as follows: $\frac{\partial v}{\partial t} = \frac{v_{ij}^{n+1} - v_{ij}^{n}}{\Delta t}$. The physics of the problem allows us to employ *upwind* [31] differencing schemes for the space derivatives, which involves computations with "previous" spatial grid points. This discretization strategy favors the direct application of rectangular tiling in the sequel.

Thus, in this case we can substitute the space partial derivatives as follows: $\frac{\partial v}{\partial x} = \frac{v^n_{ij} - v^n_{(i-1)j}}{\Delta x}$. If we substitute the above formulas in Equation (1) we get:

$$v^{n+1}_{ij} = \left(1 + 2a\frac{\Delta t}{\Delta x}\right) v^n_{ij} - a\frac{\Delta t}{\Delta x}\left(v^n_{(i-1)j} + v^n_{i(j-1)}\right) \qquad (2)$$

where for notational convenience we suppose a uniform computational grid in the two spatial dimensions ($\Delta x = \Delta y$) and $a_x = a_y = a$. Note that v^0_{ij}, v^n_{0j} and v^n_{i0} are known from the initial and boundary values of the PDE problem. Equation (2) can be easily solved for all points in the discretized computational grid $T' \times X' \times Y'$ where $T' = T/\ \Delta t$, $X' = X/\Delta x$ and $Y' = Y/\Delta y$ with the nested loop shown in Algorithm 2.

Algorithm 2: nested loop for 2-D advection equation

```
for j1 ← 0 to T' do
    for j2 ← 1 to X' do
        for j3 ← 1 to Y' do
            U[j1+1][j2][j3] = (1+2·a·dt/dx)·U[j1][j2][j3] −
            a · dt/dx · (U[j1][j2 − 1][j3] + U[j1][j2][j3 − 1]);
```

The dependence matrix of the above algorithm is $D = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and $\vec{d'} = (1, 1, 1)$. The discretization process followed leads to nonnegative elements in the dependence matrix. Note that alternative discretization schemes can lead to longer dependencies. The reader can find additional details on the equation and the discretization process in [31].

C. Supernode transformation

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

Formally supernode transformation is defined as follows:

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

where $\lfloor H\vec{j} \rfloor$ identifies the coordinates of the tile that index point $\vec{j}(j_1, j_2, \ldots, j_{n+1})$ is mapped to and $\vec{j} - H^{-1}\lfloor H\vec{j} \rfloor$ gives the coordinates of $\vec{j}$ within that tile relative to the tile origin. Thus the initial $(n+1)$-dimensional iteration space is transformed to a $(2n+2)$-dimensional one, the space of tiles and the space of indices within tiles. Indices within tiles have to be sequentially executed, while tiles themselves can be assigned to processes and executed in parallel according to a valid hyperplane schedule as we will see in Section II-D. The tiled space J^S and the supernode dependence matrix D^S are defined as follows: $J^S = \{\vec{j}^S(j^S_1, \ldots, j^S_{n+1}) | \vec{j}^S = \lfloor H\vec{j} \rfloor, \vec{j} \in J^{n+1}\}$, $D^S = \{\vec{d}^S | \vec{d}^S = \lfloor H(\vec{j}_0 + \vec{d}) \rfloor, \vec{d} \in D, \vec{j}_0 \in J^{n+1} | 0 \leq \lfloor H\vec{j}_0 \rfloor \leq 1\}$ where $\vec{j}_0$ denotes the index points belonging in the first complete tile starting from the origin of the iteration space J^{n+1}. The tiled space can be also written as $J^S = \{\vec{j}^S | j^S_i \in Z \wedge l^S_i \leq j^S_i \leq u^S_i, i = 1 \ldots n+1\}$. Each point $\vec{j}^S$ in this $(n+1)$-dimensional integer space J^S is a distinct tile with coordinates $(j^S_1, j^S_2, \ldots, j^S_{n+1})$.

Given an algorithm with dependence matrix D, for a tiling to be legal, it must hold $HD \geq 0$. This ensures that tiles are atomic and that the initial execution order is preserved [1], [5]. In the opposite case any execution order of tiles would result in a deadlock. In this paper we assume that all dependence vectors contain no negative element and are smaller than the tile size. This allows us to apply rectangular tiling transformations which are defined by the diagonal matrices $H = diag(h_1, h_2, \ldots, h_{n+1})$ and $P = diag(p_1, p_2, \ldots, p_{n+1})$. Figure 1 (left) shows an example of a rectangular tiling transformation.

Finally, we assume that all dependencies are entirely contained in each supernode's area, which means that $|HD| < 1$ [20] or alternatively that the supernode dependence matrix D^S contains only 0's and 1's. This assumption is quite reasonable since dependence vectors for common problems are relatively small, while tile sizes may result to be orders of magnitude greater in systems with very fast processors. In this case every tile needs to exchange data only with its nearest neighbors. The number of index points contained in a supernode expresses the respective computation cost of this supernode (tile), and is calculated by $det(P)$. Thus we have $V_{comp} = det(P)$ and for rectangular tiling transformations $V_{comp} = det(P) = \prod_{i=1}^{n+1} p_i$.

D. Scheduling, mapping and parallel execution time

We will schedule the problems under consideration with linear scheduling techniques [32], [33]. Central to linear scheduling is the notion of the scheduling vector Π. Intuitively, in simple cases, it suffices to calculate the inner product of a point $\vec{j} \in J^{n+1}$ with Π to derive the parallel time step at which $\vec{j}$ will be executed. In the general case [32], $\vec{j} \in J^{n+1}$ scheduled according to a linear scheduling vector Π, will be executed at $t_j = \lfloor \frac{\Pi\vec{j} + t_0}{disp\Pi} \rfloor$, where $t_0 = -min\Pi\vec{i} : \vec{i} \in J^{n+1}$ is the alignment constant or the initial time of execution of the first point in the iteration space, and $disp\Pi = min\Pi\vec{d_i} : \vec{d_i} \in D$ is the displacement constant expressing the time pace of computations. Thus, a tile $\vec{j}^S \in J^S$ will be executed at $t_{j^S} = \lfloor \frac{\Pi\vec{j}^S + t_0}{disp\Pi} \rfloor$. All points (or tiles) that lie within each n-dimensional surface perpendicular to the scheduling vector Π can execute in parallel, thus, one can employ an n-dimensional array of processes to maximize parallelism [34]. In our approach we will also consider the general case of an n-dimensional process grid to execute in parallel $(n+1)$-dimensional iteration (or tiled) spaces.

Suppose, for notational convenience, that $l_i = 0$, $i = 1 \ldots n+1$. Then the last point of the iteration space $\vec{j}_{last} = (u_1, u_2, \ldots, u_{n+1})$ will be transformed by a rectangular tiling transformation to the last tile $\vec{j}^S_{last} = (\frac{u_1}{p_i}, \frac{u_2}{p_2}, \ldots, \frac{u_{n+1}}{p_{n+1}})$. Using $\Pi = (1, 1, \ldots, 1)$ as scheduling vector, then the last tile will be scheduled at time step $t_{j^S} = \sum_{i=1}^{n+1} \frac{u_i}{p_i}$ ($t_0 = 0$ and supposing $disp\Pi = 1$), which clearly constitutes the total number of parallel time steps for the problems under consideration. In every parallel time step each process performs uninterrupted computation within a single tile and communicates with its n neighbors in order to exchange data. Note that, even if the dependencies of the problem lead to the need for data exchange with diagonal neighbors, one can apply indirect message passing techniques (discussed in [35]), in order to limit the neighboring processes to the n non-diagonal ones. If t_c is the time to compute one iteration, t_s is the

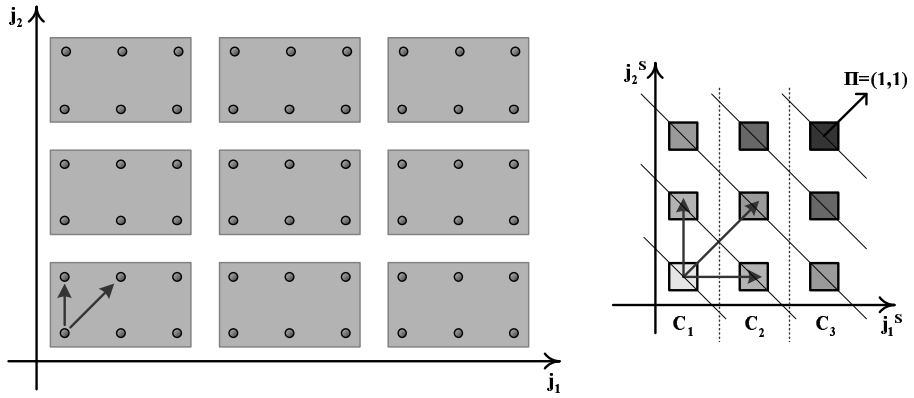


Fig. 1. Tiling, mapping and scheduling example for 1-D advection equation.

communication startup latency, t_t is the time to transmit a unit of data and k is the mapping dimension (i.e. the dimension across which all tiles are assigned to the same process), then the total parallel execution time can be expressed by Equation (3).

$$T = \sum_{i=1}^{n+1} \frac{u_i}{p_i} \left(\prod_{i=1}^{n+1} p_i t_c + n t_s + \sum_{i=1, i\neq k}^{n+1} \left(\prod_{j=1, j\neq i}^{n+1} p_j \right) d'_i t_t \right) \quad (3)$$

The above equation multiplies the number of the parallel time steps ($\sum_{i=1}^{n+1} \frac{u_i}{p_i}$) with the total time o feach step. This in turn is decomposed in the computation time of a tile ($\prod_{i=1}^{n+1} p_i t_c$), the startup time for the communication along n dimensions (nt_s) and the transmission time of all messages ($\sum_{i=1, i\neq k}^{n+1} \left(\prod_{j=1, j\neq i}^{n+1} p_j \right) d'_i t_t$).

Example 1: *Figure 1 (left) shows the original iteration space for 1-D advection equation ($J^2 = \{\vec{j}(j_1, j_2) | 0 \leq j_1 \leq 8 \wedge 0 \leq j_2 \leq 5\}$) and the dependencies of the problem ($\vec{d_1} = (1,0)^T$ and $\vec{d_2} = (1,1)^T$, thus $\vec{d'} = (1,1)$) depicted by arrows. The grey boxes represent the grouping of neighboring iteration points imposed by the tiling transformation defined by* $P = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$. *Figure 1 (right) shows the tiled iteration space ($J^S = \{\vec{j}^S(j_1^S, j_2^S) | 0 \leq j_1^S \leq 2 \wedge 0 \leq j_2^S \leq 2\}$) and the tile dependencies ($\vec{d_1^S} = (1,0)^T$, $\vec{d_2^S} = (0,1)^T$ and $\vec{d_3^S} = (1,1)^T$). We map tiles along the innermost (second in this case) dimension to the same process (C_1, C_2, C_3). If we apply linear scheduling with vector $\Pi = (1,1)$, the last tile of the algorithm ($\vec{j}^S_{last} = (2,2)$) will be executed at time step $t_{j^S} = \Pi \vec{j}^S_{last} = 4$, thus the total number of parallel time steps will be 5. Tiles executed at the same time step are shaded in the same color. According to Equation (3), the total parallel execution time will be:*

$$T = 5(6t_c + t_s + 2t_t)$$

III. Related work – "optimal" supernode shapes

After the proposition of tiling tranformation by Irigoin and Triolet [1] researchers started elaborating on efficient or optimal tiling tranformations. Appart from the definition of legal tile shapes, i.e. tile shapes that respect the dependencies of the algorithm and allow uninterrupted execution of tiles, the seminal work of Ramanujam and Sadayappan [5] introduced the idea of tiling transformations *for minimal communication*. They formulated a *per tile* communication function which needs to be minimized respecting the legality criterions, in order to minimize the number of dependencies that crosscut the tile's surfaces. These cross-cutting dependencies are responsible for communication between tiles. Further elaborating on this optimization criterion, Xue [21] showed that the surfaces that define the tile should be selected to be parallel to the dependence cone of the algorithm.

A more general approach could be to search for a tiling transformation that minimizes the total parallel execution time of the algorithm as expressed in Equation (3). Note that the selection of various tiling transformations affects the number of parallel time steps and the time for computation and communication per step. Minimizing the above equation is quite an intricate task, since one cannot effectively model the factors t_c, t_s and t_t. t_c being the computation time of a single iteration within a tile, includes the necessary memory accesses for the computed data. The time to access data from memory is crucial for the computation process and is greatly affected by the tile size and shape. The interaction of various tiling transformations with the memory subsystem is difficult to model, since a number of factors such as the cache size and organization and hardware prefetching mechanisms need to be taken into consideration. In any case t_c cannot be considered as a constant but, on the contrary, as a proper function of the tiling parameters, thus $t_c = t_c(p_1, p_2, \ldots, p_{n+1})$. Accordingly, the t_s time is dependent on the implementation of the message-passing mechanisms employed (e.g. MPI library) and cannot be easily considered as constant since it includes communication tasks that are not deterministic, such as the management of unexpected messages (e.g. call to *receive* functions after the arrival of a message). Finally, even for t_t which is the simplest of the above parameters, since one needs to simply divide the message size with the effective bandwidth of the underlying communication network, one has to consider implementation-specific issues. For example, various MPI libraries employ different transmission mechanisms (eager, rendezvous, etc) according to the message size.

With a goal to minimize the total parallel execution time, Hodzic and Shang [14], [36] employ a simpler model than that of Equation (3). The authors disregard the communication transmission part, assuming that this task can be overlapped by useful computations. They consider t_c and t_s as constants and thus focus on the selection of tile shapes that minimize the total number of parallel execution steps. However, as shown in [9] and [16], one needs to apply special scheduling strategies that are not

considered in [14], [36] and employ sophisticated communication hardware in order to hide some part of the transmission time. Similarly, Högstedt et al. [15] split the communication time in startup and transmission times. The former is added to the computation time as a constant while the latter is again considered overlapped by computations. Again here the authors select a tile shape that minimizes the path to reach the last tile, or, in other words to minimize the processor idle times and maximize parallelism. We call the tile shapes selected by [14] and [15] as *scheduling-aware*. Note that both approaches ignore the total communication volume imposed by a tiling selection, by assuming that all of the transmission time is hidden underneath useful computation. However, it is not always possible to overlap communication transmission even if advanced scheduling schemes and sophisticated hardware are employed [16]. In our approach presented in the next section we show that the minimization of the total communication volume is an important issue that should be taken into consideration, especially when commodity interconnection networks with no overlapping capabilities are concerned.

Finally, Parsa and Lotfi [23] provide a genetic algorithm that minimizes an objective function encapsulating processing, communication and disk-access times. Although their approach targets general tiling transformations, their objective function does not represent the overall parallel execution time, while the selected transformations are not tested in real problems and platforms as far as their efficiency is concerned.

IV. Communication-aware supernode shape

In this section we propose a new criterion for the selection of an efficient tile shape, i.e. the minimization of the *per process* communication volume of the algorithm. In addition, we provide a method to select a tile shape based on this criterion, called the *communication-aware* tile shape. Prior to this, we formally define our problem and provide the *scheduling-aware* solution to this problem proposed by previous work.

A. Definition of the problem

The input of our problem is an algorithm following the model discussed in Section II-A. For notational convenience we assume $l_i = 0$, $i = 1 \dots n+1$, thus we consider an $(n+1)$ - dimensional nested loop with a rectangular iteration space $(u_1 \times u_2 \times \dots \times u_{n+1})$ and a dependence matrix D of nonnegative, constant, flow dependencies. The tile size g is also given as an input to our problem. Note that the definition of the optimal tile size is a very difficult problem. However, for a specific parallel architecture one can conduct a series of benchmarks taking into consideration parameters such as the cache size, the cpu power and the communication latency and bandwidth to experimentally approximate an efficient tile size. Finally, we consider a fixed number of available processes C.

Contrary to related scientific work, we adopt a different approach for the specification of the desirable communication-aware tile shape: Instead of defining a tiling transformation matrix H or P, we equivalently aim at determining an appropriate process topology $C = \prod_{i=1}^{n} C_i$ for the mapping of the parallel algorithm, according to the mapping scheme presented in Section II-D. Indeed, the selection of the process topology implicitly enforces a particular tiling transformation: Determining a topology $C_1 \times C_2 \times \dots \times C_n$ for the parallel mapping of an algorithm with iteration space $u_1 \times u_2 \times \dots \times u_{n+1}$ effectively slices dimension u_i to C_i parts ($i = 1 \dots n$). This fact is equivalent to applying a rectangular tiling transformation described by the following matrix

$$P = \begin{bmatrix} u_1/C_1 & 0 & \dots & 0 & 0 \\ & & \dots & & \\ 0 & 0 & \dots & u_n/C_n & 0 \\ 0 & 0 & \dots & 0 & (g\prod_{i=1}^{n} C_i)/(\prod_{i=1}^{n} u_i) \end{bmatrix}$$

where g is the tile size dictated by the underlying architecture (processor speed, interconnection bandwidth etc.) and affecting the grain of the parallelism. Thus, the problem of selecting an efficient tiling transformation, collapses to the definition of the terms $C_1, C_2, \dots, C_n$ of the above tiling transformation. Moreover, proposing an efficient Cartesian process topology can lead to the direct incorporation of the optimization technique in a message passing library like MPI, e.g. through the `MPI_Cart_create` library routine. Note that, in the above discussion it is assumed that all tiles along the inner dimension are mapped to the same process. Since the algorithm contains no negative dependence element, any permutation of the loop nest is legal [37], [38], that is, any loop can be selected to be the innermost one.

Example 2: *Suppose we need to solve in parallel a three dimensional problem with* $u_1 = u_2 = u_3 = 128$ *and have* 16 *processes available. Let also the appropriate tile size dictated by the parallel platform be* $g \approx 4096$*. If we map the first two dimensions to a process grid* $C_1 \times C_2$ *then clearly we have the five candidate topologies:* 1×16, 2×8, 4×4, 8×2 *and* 16×1*. The first topology (*1×16*) does not partition the first dimension of the iteration space leading to* $p_1 = 128$*, but slices the second dimension into* 16 *pieces leading to* $p_2 = 128/16 = 8$*. Thus we set* $p_3 = 4096 \times 16/128^2 = 4$ *to determine the tiling parameter for the third dimension. Thus, given the constraints on the number of available processes and the appropriate tile size the above five process topologies lead to the following tile shapes:* $128 \times 8 \times 4$, $64 \times 16 \times 4$, $32 \times 32 \times 4$, $16 \times 64 \times 4$ *and* $8 \times 128 \times 4$.

B. Previous work: Scheduling-aware supernode shape

According to [14], [15], given C processes for the mapping of an $(n+1)$-dimensional algorithm on an n-dimensional process grid, the scheduling-aware tiling transformation can be obtained as a feasible solution to the following optimization problem:

$$\left.\begin{array}{rcl} C_i & \rightarrow & \sqrt[n]{C}, C_i \in \mathbb{N}, i = 1 \dots n \\ C & = & \prod_{i=1}^{n} C_i \end{array}\right\} \quad (4)$$

A process topology complying to (4) tries to place equal number of processes in each dimension, minimizing in this way the required total number of parallel execution steps, but fails to consider both the algorithmic dependencies and the iteration space, in order to reduce the communication volume. The advantage of such a process topology is that it minimizes the latency of the parallel program; it ensures that the most distant process will start executing its work share at the earliest possible time step. Note that in this paper we do not compare against [14], [15] as a whole, since these papers propose the selection of a general tiling transformation (not necessarily rectangular) that minimizes the idle tiles of processes. On the contrary, we compare our approach presented in the next paragraph against the approach of [14], [15] to solve the problem defined in Section IV-A.

C. Communication-aware supernode shape

In this section we discuss that an important criterion for the selection of an efficient tiling transformation is the communication volume imposed by the transformation. We use the notion of the *per process* communication volume, i.e. the data that need to be sent by one process due to algorithmic dependences. For the problems under consideration, a process in the n-dimensional process grid needs to send n distinct messages (one per dimension). The size of each message is equal to the product of the maximum dependence across the dimension of communication by the size of the $n-1$-dimensional boundary surface. Lemma 1 provides an expression for this metric.

Lemma 1: If an $(n+1)$-dimensional rectangular iteration space $u_1 \times u_2 \times \ldots \times u_{n+1}$ is assigned to an n-dimensional process grid $C_1 \times C_2 \times \ldots \times C_n = C$, then the total communication volume of a non-boundary process is given by the expression:

$$\begin{aligned} V_{pcomm} &= d'_1 \prod_{\substack{i=1 \\ i\neq 1}}^{n} \frac{u_i}{C_i} u_{n+1} \cdots + d'_n \prod_{\substack{i=1 \\ i\neq n}}^{n} \frac{u_i}{C_i} u_{n+1} \\ &= \frac{u_{n+1} \prod_{i=1}^{n} u_i}{C} \left(\frac{d'_1 C_1}{u_1} + \cdots + \frac{d'_n C_n}{u_n} \right) \end{aligned} \tag{5}$$

where $d'_i = max(d_{il}), l = 1 \ldots m,\ i = 1 \ldots n$.

Proof: In an n-dimensional process grid, each non-boundary process needs to send communication data imposed by the algorithmic dependencies from n boundary surfaces to exactly n neighboring processes. The area of the boundary surface in the j-th dimension for this process is $\prod_{\substack{i=1 \\ i\neq j}}^{n} \frac{u_i}{C_i} u_{n+1}$. The communication data across the j-th dimension clearly derive from the product of the area of the boundary surface with the maximum dependence perpendicular to that dimension, i.e. $d'_j \prod_{\substack{i=1 \\ i\neq j}}^{n} \frac{u_i}{C_i} u_{n+1}$. If we sum the communication data for all dimensions we deduce (5). ■

Example 3: *As an example, suppose one needs to solve an advection problem in a two-dimensional rectangular domain $u_1 \times u_2$ for a time window u_3. Let the tile size be g, the available number of processes 16 and the data dependencies of the algorithm lead to the following dependencies per dimension $\vec{d'}_1 = (2,0,0)^T$, $\vec{d'}_2 = (0,2,0)^T$ and $\vec{d'}_3 = (0,0,2)^T$. The rectangular tiling transformation that will be applied can be defined by a 3-dimensional diagonal matrix $P = diag(p_1, p_2, p_3)$. We partition the $u_1 \times u_2$ space in 16 tiles and appropriately adjust the tile height to conform with the restriction of the tile size. Thus, we will first determine p_1, p_2 and subsequently we will set $p_3 = g/p_1 p_2$. We will investigate two alternative feasible tile shapes, namely ($p_1 = u_1/4, p_2 = u_2/4, p_3 = 16g/u_1u_2$) and ($p_1 = u_1/8, p_2 = u_2/2, p_3 = 16g/u_1u_2$).*

Figure 2 shows the projection of the tiled iteration space on the $j_1 j_2$ surface and its allocation to the 16 processes ($c_1 \cdots c_{16}$) for the two alternative tiling transformations. Note that each process is assigned a chain of tiles along the j_3 dimension. The shaded parts of Figure 2 represent the communication data for the two candidate tiling transformations. The total communication volume V_{tcomm} derives from the boundary area between the processes ($3u_1u_3 + 3u_2u_3$ for the first transformation and $u_1u_3 + 7u_2u_3$ for the second transformation), multiplied by the maximum coordinate of the dependence matrix in the corresponding dimension, which in our case is 2. Thus, we have $V_{tcomm,1} = 6u_1u_3 + 6u_2u_3$ and $V_{tcomm,2} = 2u_1u_3 + 14u_2u_3$. This difference in communication volume is depicted on the per process communication volume as well, derived from Lemma 1, where $V_{pcomm,1} = 2u_3(\frac{u_1}{4} + \frac{u_2}{4})$ and $V_{pcomm,2} = 2u_3(\frac{u_1}{8} + \frac{u_2}{2})$.

If we apply linear scheduling defined by vector $\Pi = (1,1,1)$, then the tile $(4,4,T/p_3)$ will be scheduled last according to the first tiling transformation, and will be executed at time step $T_1 = 8 + T/p_3$, while the respective last tile $(8,2,T/p_3)$ of the second transformation will be executed at time step $T_2 = 10 + T/p_3$. This implies that the first transformation is better as far as the total number of parallel execution steps is concerned, since $T_1 < T_2$. However, notice that if $u_1 > 2u_2$, then $V_{tcomm,1} > V_{tcomm,2}$, which means that the second transformation is superior in terms of total communication volume. Consequently, when it comes to executing the above problem for $u_1 > 2u_2$, we need to decide between scheduling-aware and communication-aware tiling. Intuitively, in our example one can see that the communication-aware transformation entails a moderate increase in the number of time steps, if we make the reasonable assumption that $T/p_t >> 2$. On the other hand, if we have $u_1 = 4u_2$, the communication-aware transformation leads to almost 27% less communication data. For this reason, we claim that the communication-aware transformation will lead to a significantly lower total execution time.

The following lemma provides the condition that must hold in order to minimize the communication of a non-boundary process. This is achieved by the even distribution of communication data in all process dimensions.

Lemma 2: Let $u_1 \times u_2 \times \ldots \times u_{n+1}$ be an $(n+1)$-dimensional rectangular iteration space, $d'_i = max(d_{il}), l = 1 \ldots m, i = 1 \ldots n$ be the maximum dependence per direction and C be the number of processes available for the parallel execution of the algorithm. If there exist $C_i \in \mathbb{N}$, such that

$$C = \prod_{i=1}^{n} C_i \tag{6}$$

and

$$\frac{d'_i C_i}{u_i} = \frac{d'_j C_j}{u_j}, i, j = 1 \ldots n \tag{7}$$

then process topology $C_1 \times \cdots \times C_n$ minimizes the per process communication for the tiled algorithm on C processes.

Proof: According to (6), it holds

$$C_n = \frac{C}{C_1 \times \cdots \times C_{n-1}} \tag{8}$$

Each process assumes $\lceil u_i/C_i \rceil$ iterations along direction i, where $1 \leq i \leq n$. For the sake of simplicity, we assume that $\lceil u_i/C_i \rceil \simeq u_i/C_i$. Using (8), (5) from Lemma 1 can be written by substituting C_n as follows:

$$V_{comm} = \frac{u_{n+1} \prod_{i=1}^{n} u_i}{C} \sum_{i=1}^{n-1} \frac{d'_i C_i}{u_i} + \frac{d'_n u_{n+1} \prod_{i=1}^{n} u_i}{u_n C_1 \ldots C_{n-1}} \tag{9}$$

Note that V_{comm} is substantially a function of $C_1, \ldots, C_{n-1}$ (formally: $V_{comm} : \mathbb{N}^{n-1} \rightarrow \mathbb{R}$). Let $\overline{V}_{comm}$ be the real extension of V_{comm}, defined by (9) for $C_j \in \mathbb{R}, 1 \leq j \leq n$ ($\overline{V}_{comm} : \mathbb{R}^{n-1} \rightarrow \mathbb{R}$). For a stationary point $(C_1, \ldots, C_{n-1})$ of $\overline{V}_{comm}$ and $1 \leq j \leq n-1$ it holds:

$$\frac{\partial \overline{V}_{comm}}{\partial C_j} = 0 \Rightarrow \frac{d_j C_j}{u_j} = \frac{d'_n C_n}{u_n} \tag{10}$$

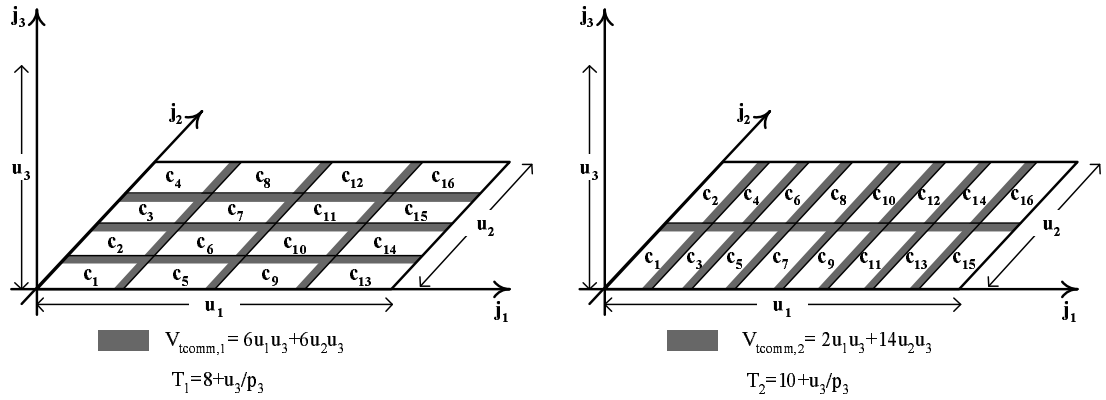


Fig. 2. Total communication volume and parallel execution steps for two tiling transformations

Also,

$$\frac{\partial^2 \overline{V}_{comm}}{\partial {C_j}^2} = \frac{2d'_n \prod_{i=1}^{n} u_i}{u_n C_1 \dots C_j^3 \dots C_{n-1}} > 0 \quad (11)$$

Because of (10) and (11), $\overline{V}_{comm}$ has a minimum at $(C_1, \dots, C_{n-1})$, and as $C_i \in \mathbb{N}, 1 \leq i \leq n-1$, this will be the minimum of V_{comm}, as well. Therefore, the communication data is minimal when a topology $C_1 \times \dots \times C_n$ satisfying (10) is assumed.

■

Finally, Theorem 1 makes use of Lemma 2 to derive an expression for the number of processes in each dimension.

Theorem 1: Let $u_1 \times u_2 \times \dots \times u_{n+1}$ be an $(n+1)$-dimensional rectangular iteration space, $d'_i = max(d_{il}), l = 1 \dots m, i = 1 \dots n$ be the maximum dependence per direction and C be the number of processes available for the parallel execution of the algorithm. In order to minimize the per-process communication volume, the number of processes in each dimension should be set by

$$C_j = \frac{u_j}{d'_j} \sqrt[n]{\frac{C \prod_{i=1}^{n} d'_i}{\prod_{i=1}^{n} u_i}}, j = 1 \dots n \quad (12)$$

Proof: Note that it holds

$$\frac{C \prod_{i=1}^{n} d'_i}{\prod_{i=1}^{n} u_i} = \frac{d'_1 C_1}{u_1} \times \dots \times \frac{d'_n C_n}{u_n} \quad (13)$$

By combining (13) with (10), we can easily deduce (12). ■

It should be noted that (12) does not always define a valid integer process topology: it is possible that $C_j \notin \mathbb{N}$ for some value j with $j = 1 \dots n$. However, when truncated to an integer, it can serve as a good starting value for an exhaustive algorithm searching for feasible process topologies in the close neighborhood of the minimum of $\overline{V}_{comm}$, as determined by (12). In practice, as $n+1$ does not exceed 3 or 4, and C ranges up to a few hundreds or maybe thousands of processes, the high complexity of the heuristic algorithm does not result in high execution times. Furthermore, the monotonicity of function $\overline{V}_{comm}$ allows immediate elimination of candidate process topologies, that lead to increased communication cost. In order to verify this claim, we measured on a PIII@800MHz the execution times for the specification of a feasible communication-aware 3D process topology, given all possible 4D iteration spaces $(100 \dots 10k) \times (100 \dots 10k) \times (100 \dots 10k) \times u_{n+1}$, data dependencies $[(1 \dots 3, 0, 0, d), (0, 1 \dots 3, 0, d'), (0, 0, 1 \dots 3, d'')]$ and for $100 \leq C \leq 1k$. The execution time equaled on average 21 msec, while under no circumstances did it exceed 0.9 sec. Note, finally, that the approach described above does not theoretically ensure the finding of the minimum communication volume in integer space. However, it greatly restricts the search space compared to an exhaustive search within all possible process topologies at the cost of possible unoptimality.

V. Experimental Results

In this section we will experimentally evaluate the parallel performance of the communication-aware tiling transformations or equivalently process topologies as derived from Equation (12). In addition, we will compare the proposed topologies against the scheduling-aware ones proposed in [14], [15] and derived from Equation (4). We consider the 2-D and 3-D advection equation and an artificial kernel with a 3-dimensional nested loop following the algorithmic model described in Section II-A. Our experimental platform is a 16-node Linux cluster (kernel 2.6.23.1). Each node includes two quad-core Xeon chips based on Intel's Core 2 microarchitecture (E5335@2GHz). Two cores per package share a 4MB L2 cache. The interconnection network is Gigabit Ethernet. We experimented with 100 processes running in the above cluster. We used MPICH v. 1.2.7 MPI implementation, configured with gcc v. 4.2.3 and applied the -O2 optimization flag to all programs. In order to reduce as much as possible the differences in memory access patterns induced by various tiling transformations, we applied cache blocking as described in [39].

A. 2-D advection equation

Recall from Section II-B that the 2-D advection equation problem results in a 3-dimensional iteration space which is mapped on a 2-dimensional process grid. We experimented with various iteration spaces ($X \times Y \times T$) and all possible process topologies for the 100 processes. We mapped the $X \times Y$ plane on the process grid and assigned tiles along T to the same process. In all cases the scheduling-aware process topology that derives from Equation 4 is 10×10. Figure 3 provides information on the scalability of our implementation for an increasing number of processing cores and four different problem sizes. As expected by the nearest-neighbor nature of the communication and the dimension of the problem, the algorithm scales better for larger problems. The hardware configuration of our platform enforces several cores to share the same network interface which is a bounding factor for the scalability of our implementation.

Figure 4 presents the first comparison between the scheduling-aware and the communication-aware strategies for the selection

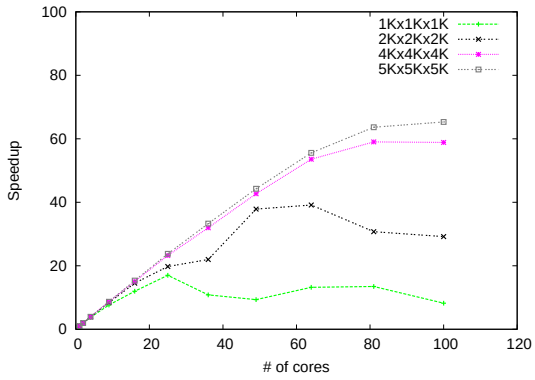


Fig. 3. Scalability of the 2D advection equation.

of a process topology for $X = 50K$ and $Y = 8K$. The communication-aware topology in this case is 25×4. We varied T and the tile size in order to assign a different number of tiles (denoted $Tiles$) to the same process across different runs. We observe that when the total number of tiles assigned to each process is small, then the scheduling-aware topology outperforms the communication-aware, since, as expected, for small number of total parallel time steps it is crucial to maximize the concurrency of processes, or in other words to minimize the steps before the last process starts its execution. The scheduling-aware topology enforces the last process to start its execution at parallel time step 20, while the communication-aware topology enforces the last process to start its execution at parallel time step 29. Observe also that this difference in concurrency diminishes as the number of tiles increases. In this case, the reduction of the communication volume imposed by the communication-aware topology leads to smaller total parallel execution times. Note that for meaningful parallelization scenarios we assign a number of tiles to each process significantly larger than C, thus for the forthcoming experiments we will assume $Tiles \gg 100$.

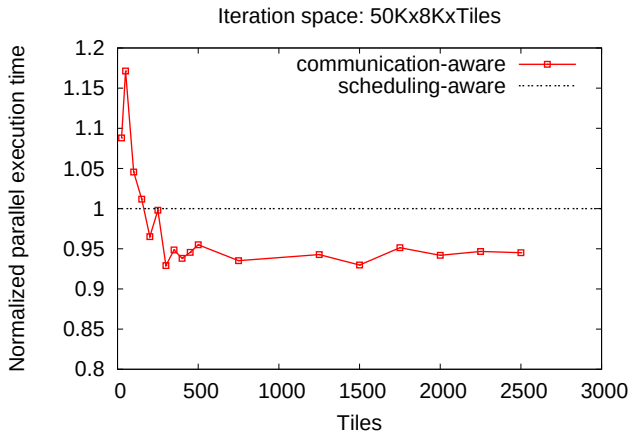


Fig. 4. Comparison of *scheduling-aware* and *communication-aware* process topologies for various numbers of tiles.

In Figure 5 we present normalized parallel execution times decomposed into communication and computation times for six iteration spaces and all feasible process topologies. We experimented with various tile sizes per iteration space and present the best attained performance. The communication-aware topology is denoted with a dot over the corresponding bar. The first observation that can be made is that the total parallel execution time greatly differentiates between different topologies (the difference reaches even to a factor of 5). These large differences are due to the extreme increases in communication times that can be imposed by an unfortunate selection of process grid. This fact justifies our argument that the minimization of the communication volume via the selection of a proper tiling transformation should be given high priority. Observe also that the communication-aware topology outperforms the scheduling-aware topology in the first five iteration spaces, while for the sixth iteration space both strategies lead to the selection of a 10×10 topology. Note finally that the communication-aware topology leads to the lowest total parallel execution times among all topologies in five out of six iteration spaces.

Table I provides a direct comparison between the two strategies for the iteration spaces of Figure 5. The communication-aware topology exhibits an improvement in performance compared to the scheduling-aware topology in five out of six iteration spaces that ranges from 2.5% (in iteration space $40K \times 10K \times 5K$) to 32.8% (in iteration space $200K \times 2K \times 5K$). In iteration space $40K \times 20K \times 5K$ both strategies propose the 10×10 topology thus leading to the same execution and communication times. Note that we present the maximum computation and maximum communication time, reduced over all processes and normalized to the $maximum\ computation\ time + maximum\ communication\ time$ under the scheduling-aware tiling transformation. The sum of these partial times is not necessarily equal to the total execution time, as we depict the worst case scenario for both the communication and the computation times (this holds for the 3-D advection equation and the artificial kernel in the next sections). However, despite the relatively small differences in the computation times, that can be attributed to data locality effects, this profiling confirms that the relative advantage of the communication-aware tiling transformation can be directly attributed to the respective reduction of the communication times. The communication-aware topology takes into consideration the bounds of the iteration space and adjusts the placement of processes per dimension in order to reduce the communication volume. This, as shown from the experimental results, has a significant positive impact on the overall performance of the algorithm.

B. 3-D advection equation

In our second set of experiments we applied all feasible process topologies in various iteration spaces for the 3-D advection equation. The iteration space in this case is 4-dimensional ($X \times Y \times Z \times T$) thus, we map the plane $X \times Y \times Z$ on a 3-dimensional process grid and assign tiles across T to the same process. The scheduling-aware process topology in this case is $5 \times 5 \times 4$, or $5 \times 4 \times 5$ or $4 \times 5 \times 5$. We present the best result from the above three. In each iteration space and process topology we experimented with various tile sizes and present the best attained results. Figure 7 provides information on the scalability of the three-dimensional algorithm. In this case the performance and scalability are rather poor due to the more intense communication needs of the the three-dimensional process grid and the consequent bottlenecks created at the shared network interaces. Communication dominates in this application and platform, which

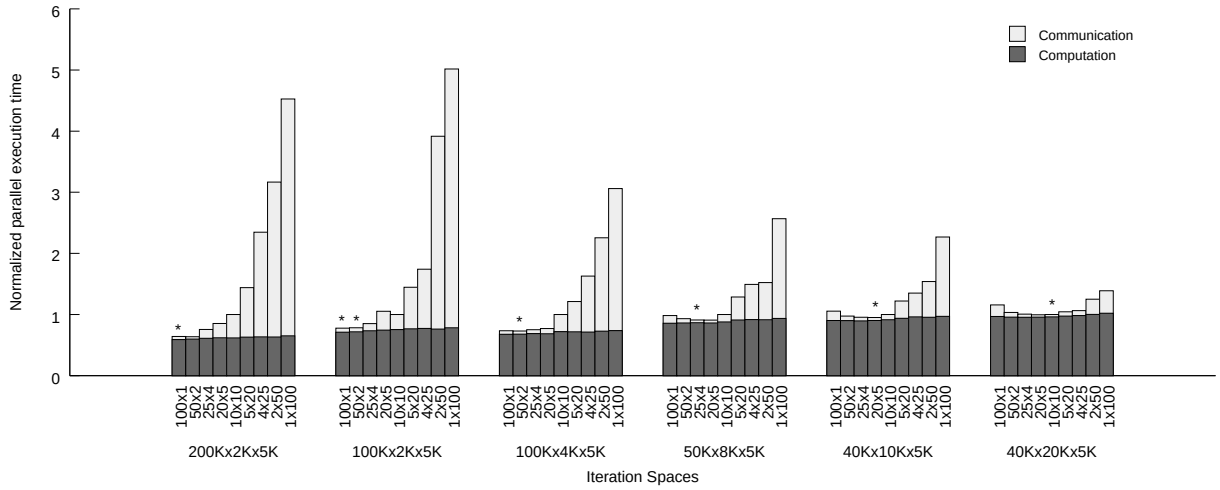


Fig. 5. Normalized total parallel execution time for various iteration spaces. The *scheduling-aware* topology is 10×10. The *communication-aware* is denoted with a dot over the bar.

Iteration space	Scheduling-aware topology		Communication-aware topology	
	Total time	Comm. time	Total time	Comm. time
$200K \times 2K \times 5K$	112.89	28.44	85.01	4.89
$100K \times 2K \times 5K$	53.33	10.90	42.68	2.74
$100K \times 4K \times 5K$	99.67	15.48	83.67	3.80
$50K \times 8K \times 5K$	88.52	7.33	83.57	3.71
$40K \times 10K \times 5K$	84.95	4.20	82.89	2.96
$40K \times 20K \times 5K$	165.96	6.62	165.96	6.62

TABLE I

COMPARISON OF *scheduling-aware* AND *communication-aware* PROCESS TOPOLOGIES FOR VARIOUS ITERATION SPACES IN 2-D ADVECTION EQUATION. ALL TIMES OF THE TABLE ARE IN SECONDS.

makes our approach to alleviate the communication overheads even more relevant.

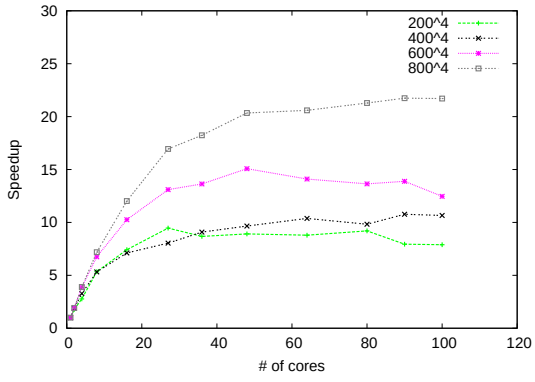


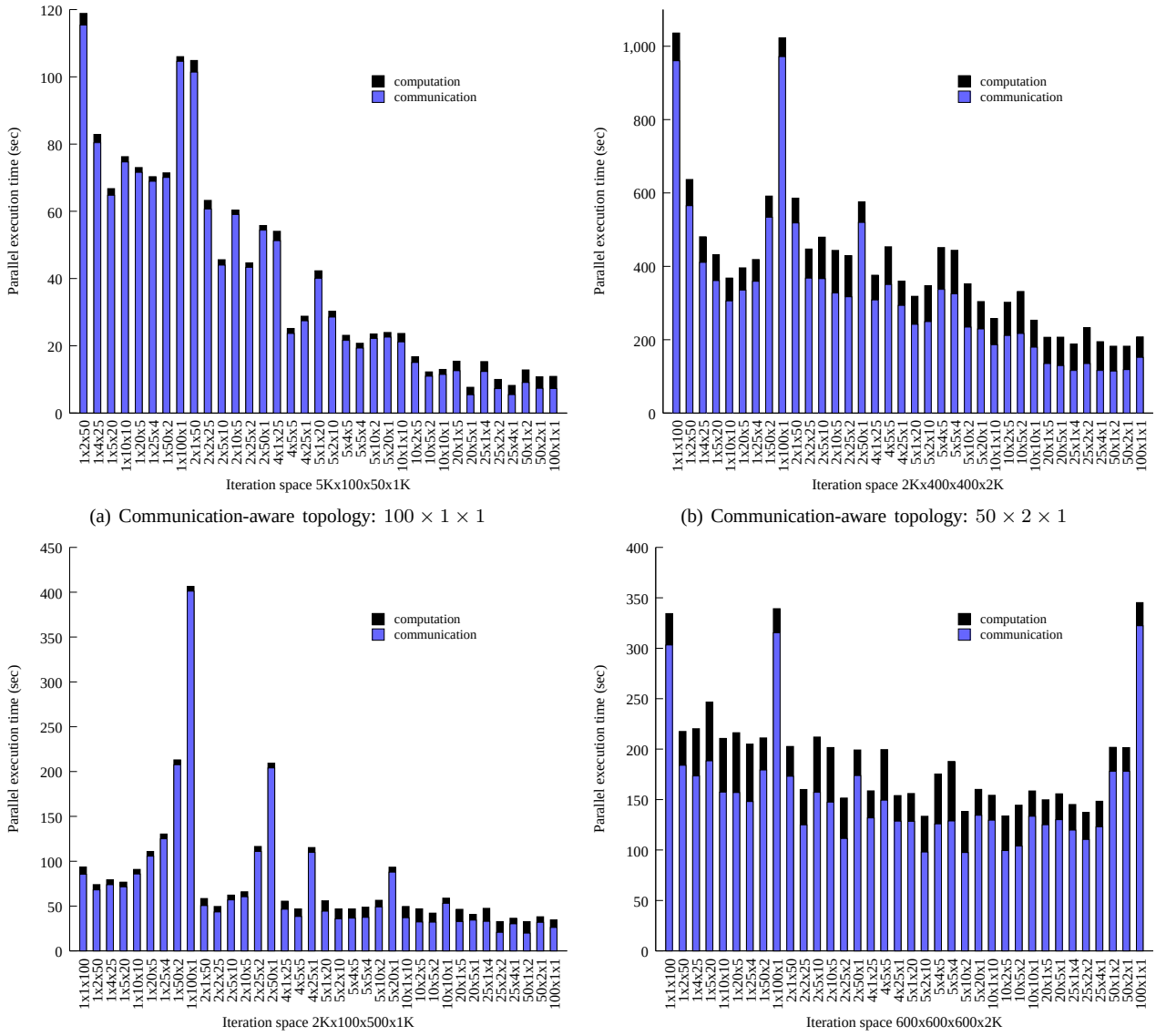
Fig. 7. Scalability of the 3D advection equation.

Figure 6 presents comparison results for four iteration spaces. In this case the differences in the total parallel execution time between different topologies are even greater. The communication overhead imposed by an unfortunate selection of process topology can kill performance. Thus it is clear that one needs a criterion to effectively select between the 36 feasible topologies for this problem. Again here, the communication-aware strategy succeeds well in this selection.

Table II performs a comparison between the scheduling-aware and the communication-aware process topology for 12 iteration spaces. For the first two iteration spaces both strategies lead to the proposal of the same topology. The third iteration space ($800 \times 200 \times 400 \times 1K$) is the only one in which the scheduling-aware topology outperforms the communication-aware one by a factor of 1.7%. For the rest of the iteration spaces the communication-aware topology outperforms the scheduling-aware one by a factor that ranges between 3.3% to 213.8%. It is clear that for 4-dimensional iteration spaces mapped on 3-dimensional process grids the selection of a communication-aware process topology is even more crucial, since the communication in this case occurs in three dimensions and thus the relevant overhead severely affects performance.

C. Artificial kernel

In this last set of experiments we implemented an artificial kernel expressed by a 3-dimensional nested loop following the model of Section II-A in order to compare the two topology selection strategies when, apart from the iteration space, the dependencies of the problems vary. Note that in 2-D and 3-D advection problems the dependencies in the communication dimensions were always the same. However, since the communication-aware topology takes this factor into consideration as well, we varied d'_1 and d'_2 in order to check their impact on performance. Table III presents results (total parallel execution times and communication times) for three iteration spaces and various combinations of



(a) Communication-aware topology: $100 \times 1 \times 1$

(b) Communication-aware topology: $50 \times 2 \times 1$

(c) Communication-aware topology: $20 \times 1 \times 5$

(d) Communication-aware topology: $5 \times 4 \times 5$

Fig. 6. Total parallel execution time (sec) for four iteration spaces in 3-D advection equation. The scheduling-aware topology in all cases is either $5 \times 5 \times 4$ or $5 \times 4 \times 5$ or $4 \times 5 \times 5$.

$d'_1 \times d'_2$. From this table we observe four important issues: (a) The communication-aware topology adjusts to the iteration space shape and to the dependencies of the problem, (b) In 6 out of 23 cases both strategies selected the same (10×10) topology, (c) In 3 cases the scheduling-aware topology outperformed the communication-aware topology since the latter caused a slow-down ranging from 2.7% to 23.5% and (d) In 14 cases the communication-aware topology led to lower parallel execution times providing an improvement that ranged from 0.4% to 96.9%.

D. Overall conclusions on the experiments

The experimental results presented in the previous paragraphs lead us to the following conclusions:

- When the total number of tiles assigned to each process is "small", then the minimization of the processor idle times with the scheduling-aware process topology is important (see Figure 4), thus the scheduling criterion should be given higher priority. As a rule of thumb, one can prioritize scheduling-aware tiling transformations when $Tiles \approx C$. However, we claim that for the majority of real-life problems it will hold $Tiles \gg C$.
- In clusters with commodity interconnection networks, such as the one used in our experiments, it is crucial to reduce the communication volume as much as possible. If $Tiles \gg C$ then the communication-aware process topologies were able to drastically reduce the communication times with an important positive impact on total parallel execution times compared to the scheduling-aware process topologies. The reduction in total parallel execution time reached up to 213%. (see Tables I, II and III). The greater differences were observed in 3-D advection which used a 3-dimensional process grid. In this case the communication overhead increases, both in terms of communication volume and in terms of number of messages (see Figure 6).
- The proposed communication-aware tiling is particularly efficient when the algorithm exhibits asymmetric data de-

Iteration space	Scheduling-aware			Communication-aware		
	Topology	Total time	Comm. time	Topology	Total time	Comm. time
$600 \times 600 \times 600 \times 2K$	$5 \times 4 \times 5$	175.36	125.86	$5 \times 4 \times 5$	175.36	125.86
$800 \times 400 \times 400 \times 1K$	$5 \times 5 \times 4$	61.48	42.65	$5 \times 5 \times 4$	61.48	42.65
$800 \times 200 \times 400 \times 1K$	$5 \times 4 \times 5$	30.75	21.37	$10 \times 2 \times 5$	31.28	21.70
$1K \times 200 \times 1K \times 1K$	$5 \times 5 \times 4$	49.99	21.76	$10 \times 1 \times 10$	43.89	10.54
$2K \times 200 \times 500 \times 1K$	$5 \times 4 \times 5$	67.97	25.89	$20 \times 1 \times 5$	43.74	14.96
$1K \times 500 \times 100 \times 1K$	$5 \times 5 \times 4$	27.85	20.91	$20 \times 5 \times 1$	23.22	15.46
$1K \times 200 \times 200 \times 1K$	$4 \times 5 \times 5$	19.18	14.21	$25 \times 2 \times 2$	18.56	13.10
$1.5K \times 200 \times 400 \times 1K$	$5 \times 5 \times 4$	56.61	40.94	$25 \times 1 \times 4$	41.43	28.45
$2K \times 100 \times 500 \times 1K$	$4 \times 5 \times 5$	49.77	38.33	$25 \times 1 \times 4$	47.47	32.87
$5K \times 100 \times 50 \times 1K$	$5 \times 5 \times 4$	20.75	19.33	$50 \times 2 \times 1$	10.80	7.35
$2K \times 200 \times 200 \times 2K$	$5 \times 5 \times 4$	77.99	64.167	$100 \times 1 \times 1$	61.29	46.27
$3K \times 400 \times 400 \times 2K$	$5 \times 5 \times 4$	443.78	325.19	$100 \times 1 \times 1$	207.58	151.93

TABLE II

COMPARISON OF *scheduling-aware* AND *communication-aware* PROCESS TOPOLOGIES FOR VARIOUS ITERATION SPACES IN 3-D ADVECTION EQUATION. ALL TIMES ARE IN SECONDS.

It. space	$d'_1 \times d'_2$	Sched.-aware topology		Comm.-aware topology			% diff.
		Total time	Comm. time	Topology	Total time	Comm. time	
$5K \times 5K \times 2K$	1×1	2.54	0.43	10×10	2.54	0.43	0.0
$5K \times 5K \times 2K$	2×1	2.60	0.45	5×20	2.41	0.28	-7.8
$5K \times 5K \times 2K$	3×1	2.55	0.44	5×20	2.54	0.42	-0.4
$5K \times 5K \times 2K$	4×1	2.77	0.54	5×20	2.63	0.51	-5.3
$5K \times 5K \times 2K$	5×1	3.27	1.03	4×25	2.49	0.33	-31.3
$2K \times 4K \times 2K$	1×1	1.02	0.32	5×20	0.84	0.13	-21.4
$2K \times 4K \times 2K$	1×2	0.85	0.14	10×10	0.85	0.14	0.0
$2K \times 4K \times 2K$	1×3	0.86	0.16	10×10	0.97	0.26	0.0
$2K \times 4K \times 2K$	1×4	1.20	0.48	20×5	1.57	0.77	+23.5
$2K \times 4K \times 2K$	1×5	1.44	0.71	20×5	1.48	0.72	+2.7
$2K \times 4K \times 2K$	2×1	1.07	0.37	5×20	0.86	0.15	-24.4
$2K \times 4K \times 2K$	3×1	0.93	0.22	4×25	0.98	0.46	+5.1
$2K \times 4K \times 2K$	4×1	1.81	1.06	4×25	1.23	0.53	-47.2
$2K \times 4K \times 2K$	5×1	2.58	1.82	4×25	1.31	0.70	-96.9
$2K \times 8K \times 2K$	1×1	1.80	0.45	5×20	1.67	0.28	-7.7
$2K \times 8K \times 2K$	1×2	2.08	0.84	5×20	1.96	0.57	-6.1
$2K \times 8K \times 2K$	1×3	1.74	0.40	10×10	1.74	0.40	0.0
$2K \times 8K \times 2K$	1×4	2.05	0.80	10×10	2.05	0.80	0.0
$2K \times 8K \times 2K$	1×5	2.13	0.78	10×10	2.13	0.78	0.0
$2K \times 8K \times 2K$	2×1	1.81	0.45	4×25	1.72	0.30	-5.2
$2K \times 8K \times 2K$	3×1	2.25	1.05	4×25	2.09	0.64	-7.6
$2K \times 8K \times 2K$	4×1	2.61	1.21	2×50	1.85	0.48	-41.1
$2K \times 8K \times 2K$	5×1	3.10	1.64	2×50	1.69	0.30	-83.4

TABLE III

COMPARISON OF *scheduling-aware* AND *communication-aware* PROCESS TOPOLOGIES FOR VARIOUS ITERATION SPACES AND DEPENDENCIES IN THE ARTIFICIAL KERNEL. THE *scheduling-aware* TOPOLOGY IS 10×10. ALL TIMES ARE IN SECONDS.

pendencies and/or iteration space dimensions.

- The proposed communication-aware tiling exhibits very good performance even when compared to the best possible total parallel execution time achieved by any topology (see Figures 5 and 6), since in several cases it succeeds the minimum time. However, there exist cases where alternative topologies minimize the execution time, which is reasonable since, as discussed in Sections II and III, the minimization of the total parallel execution time is a problem involving numerous parameters such as cpu power, memory organization, communication bandwidth and latency. The communication-aware strategy takes into consideration only the communication volume of the problem. However, the fact that based on this sole criterion we were able to mininize the execution time is several cases and approach close to the minimum in several others, leads us to the conclusion that

the minimization of the communication volume should be given a very high priority.

VI. Conclusions

This paper presented a novel approach for the selection of an efficient and feasible tile shape for the parallelization of stencil algorithms. We formulate a simple and applicable method for the specification of an appropriate tile shape, that minimizes the communication volume of a non-boundary process, assuming a fixed total number of processes. Compared to alternative tile shapes that aim at minimizing the processor idle times thus maximizing parallelism, the communication-aware tile shapes proposed here exhibit significantly lower parallel execution times for real-life problems. This improvement in performance is due to the drastic reduction in the communication volume imposed by the proposed tile shapes, that take into consideration the bounds of the iteration space and the problem dependencies in order to reduce communication data. The presented technique can be easily combined with the `MPI_Cart_create` primitive, to deliver efficient Cartesian process topologies.

References

[1] F. Irigoin and R. Triolet, "Supernode Partitioning," in *Proceedings of the 15th Ann. ACM SIGACT-SIGPLAN Symp. Principles of Programming Languages (POPL'85)*, San Diego, California, USA, Jan 1988, pp. 319–329.

[2] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1992.

[3] B. D. Acunto, *Computational Methods for PDE in Mechanics*. World Scientific Pub., 2004.

[4] K. Morton and D. Mayers, *Numerical Solution of Partial Differential Equations*. Cambridge, UK: Cambridge University Press, 2005.

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

[6] R. Andonov, S. Balev, S. Rajopadhye, and N. Yanev, "Optimal Semi-Oblique Tiling," *IEEE Trans. on Parallel and Distributed Systems*, vol. 14, no. 9, pp. 944–960, Sep 2003.

[7] P. Boulet, A. Darte, T. Risset, and Y. Robert, "(Pen)-ultimate Tiling?" *INTEGRATION, The VLSI Jounal*, vol. 17, pp. 33–51, 1994.

[8] P. Boulet, J. Dongarra, Y. Robert, and F. Vivien, "Static Tiling for Heterogeneous Computing Platforms," *Journal of Parallel Computing*, vol. 25, no. 5, pp. 547–568, May 1999.

[9] G. Goumas, A. Sotiropoulos, and N. Koziris, "Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'01)*, San Francisco, USA, Apr 2001.

[10] G. Goumas, M. Athanasaki, and N. Koziris, "An Efficient Code Generation Technique for Tiled Iteration Spaces," *IEEE Trans. on Parallel and Distributed Systems*, vol. 14, no. 10, pp. 1021–1034, Oct 2003.

[11] E. Hodzic and W. Shang, "On Supernode Transformation with Minimized Total Running Time," *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, no. 5, pp. 417–428, May 1998.

[12] G. Goumas, N. Drosinos, M. Athanasaki, and N. Koziris, "Message-Passing Code Generation for Non-rectangular Tiling Transformations," *Journal of Parallel Computing*, vol. 32, no. 10, pp. 711–732, Nov 2006.

[13] N. Drosinos and N. Koziris, "Performance Comparison of Pure MPI vs Hybrid MPI-OpenMP Parallelization Models on SMP Clusters," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium 2004*, Santa Fe, New Mexico, Apr 2004, p. 10.

[14] E. Hodzic and W. Shang, "On Time Optimal Supernode Shape," *IEEE Trans. on Parallel and Distributed Systems*, vol. 13, no. 12, pp. 1220–1233, Dec 2002.

[15] K. Högstedt, L. Carter, and J. Ferrante, "On the Parallel Execution Time of Tiled Loops," *IEEE Trans. on Parallel and Distributed Systems*, vol. 14, no. 3, pp. 307–321, Mar 2003.

[16] N. Koziris, A. Sotiropoulos, and G. Goumas, "A Pipelined Schedule to Minimize Completion Time for Loop Tiling with Computation and Communication Overlapping," *Journal of Parallel and Distributed Computing*, vol. 63, no. 11, pp. 1138–1151, Nov 2003.

[17] H. Ohta, Y. Saito, M. Kainaga, and H. Ono, "Optimal Tile Size Adjustment in Compiling General DOACROSS Loop Nests," in *Proceedings of the 9th International Conference on Supercomputing (ICS'95)*, Barcelona, Spain, Jul 1995, pp. 270–279.

[18] Y. Song and Z. Li, "Impact of Tile-Size Selection for Skewed Tiling," in *Proceedings of the 5-th Workshop on Interaction between Compilers and Architectures (INTERACT'01)*, Monterrey, Mexico, Jan 2001.

[19] P. Tang and J. Xue, "Generating Efficient Tiled Code for Distributed Memory Machines," *Journal of Parallel Computing*, vol. 26, no. 11, pp. 1369–1410, 2000.

[20] J. Xue, "On Tiling as a Loop Transformation," *Parallel Processing Letters*, vol. 7, no. 4, pp. 409–424, 1997.

[21] ——, "Communication-Minimal Tiling of Uniform Dependence Loops," *Journal of Parallel and Distributed Computing*, vol. 42, no. 1, pp. 42–59, 1997.

[22] J. Xue and W. Cai, "Time-minimal Tiling when Rise is Larger than Zero," *Journal of Parallel Computing*, vol. 28, no. 6, pp. 915–939, 2002.

[23] S. Parsa and S. Lotfi, "A New Genetic Algorithm for Loop Tiling," *Journal of Supercomputing*, vol. 37, no. 3, pp. 249–269, 2006.

[24] L. Renganarayanan, D. Kim, S. Rajopadhye, and M. M. Strout, "Parameterized Tiled Loops for Free," in *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, NY, USA: ACM, 2007, pp. 405–414.

[25] S. Krishnamoorthy, M. Baskaran, U. Bondhugula, J. Ramanujam, A. Rountev, and P. Sadayappan, "Effective Automatic Parallelization of Stencil Computations," in *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, NY, USA: ACM, 2007, pp. 235–244.

[26] N. Ahmed, N. Mateev, and K. Pingali, "Tiling Imperfectly-nested Loop Nests," in *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2000, p. 31.

[27] R. Andonov, P. Calland, S. Niar, S. Rajopadhye, and N. Yanev, "First Steps Towards Optimal Oblique Tile Sizing," in *Proceedings of the 8th International Workshop on Compilers for Parallel Computers*, Aussois, France, Jan 2000, pp. 351–366.

[28] K. Högstedt, L. Carter, and J. Ferrante, "Selecting Tile Shape for Minimal Execution time," in *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, Saint Malo, France, 1999, pp. 201–211.

[29] E. D'Hollander, "Partitioning and Labeling of Loops by Unimodular Transformations," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 4, pp. 465–476, Jul. 1992.

[30] M. Kandemir, R. Bordawekar, A. Choudhary, and J. Ramanujam, "A Unified Tiling Approach for Out-of-Core Computations," in *Sixth Workshop on Compilers for Parallel Computers*. Aachen, Germany: Forschungzentrum Julich GmbH, 1996, pp. 323–334.

[31] G. E. Karniadakis and R. M. Kirby, *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and their Implementation*. Cambridge University Press, 2002.

[32] W. Shang and J. Fortes, "Time Optimal Linear Schedules for Algorithms with Uniform Dependencies," *IEEE Trans. on Computers*, vol. 40, no. 6, pp. 723–742, 1991.

[33] A. Darte, L. Khachiyan, and Y. Robert, "Linear Scheduling is Nearly Optimal," *Parallel Processing Letters*, vol. 1, no. 2, pp. 73–81, 1991.

[34] W. Shang and J. Fortes, "On Time Mapping of Uniform Dependence Algorithms into Lower Dimensional Processor Arrays," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 3, pp. 350–363, 1992.

[35] P. Tang and J. Zigman, "Reducing Data Communication Overhead for DOACROSS Loop Nests," in *Proceedings of the 8th International Conference on Supercomputing (ICS'94)*, Manchester, UK, Jul 1994, pp. 44–53.

[36] E. Hodzic and W. Shang, "On Time Optimal Supernode Shape," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, USA, Jun 1999, pp. 2019–2026.

[37] K. Kennedy and J. R. Allen, *Optimizing compilers for modern architectures: a dependence-based approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[38] M. Wolf and M. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism," *IEEE Trans. on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 452–471, Oct 1991.

[39] G. Rivera and C.-W. Tseng, "Tiling Optimizations for 3D Scientific Computations," in *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2000, p. 32.