Selecting the Tile Shape to Reduce the Total Communication Volume

Nikolaos Drosinos, Georgios Goumas and Nectarios Koziris

National Technical University of Athens
School of Electrical and Computer Engineering
Zografou Campus, Zografou 15780, Greece
{ndros, goumas, nkoziris}@cslab.ece.ntua.gr

Abstract

In this paper we revisit the tile-shape selection problem, that has been extensively discussed in bibliography. An efficient approach is proposed for the selection of a suitable tile shape, based on the minimization of the process communication volume. We consider the large family of applications that arise from the discretization of partial differential equations (PDEs). Practical experience has shown that for such applications and distributed memory architectures, minimizing the total communication volume is more important than minimizing the total number of parallel execution steps. We formulate a new method to determine an appropriate communication-aware tile shape, i.e. the one that reduces the communication volume for a fixed number of processes. 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.

1. Introduction

Tiling or supernode transformation has been proposed as the most efficient method to map onto distributed memory architectures nested loop algorithms, whose iteration spaces cannot be partitioned into independent subspaces. The vast majority of such algorithms are encountered in applications stemming from the field of the partial differential equations (PDEs). PDEs are discretized using an appropriate stencil, which calculates value u_i of function u at position i, using the values of its neighboring points $u_{i-1}, u_{i+1}, u_{i-2}, u_{i+2}$ etc. This computation imposes data dependencies along all dimensions of the iteration space, which in turn induce high communication needs when the application is executed in parallel. In order to alleviate this communication overhead, tiling transformation groups neighboring iterations together into tiles, thus reducing both the total communication volume and frequency (total number of messages).

Tiling has attracted extensive scientific research right after its presentation by Irigoin and Triolet in 1988 [12]. 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. [14], Hodzic and Shang [7], and Andonov et al. [1] focused on the selection of the optimal tile size based on the special characteristics of the application and the target architecture. Ramanujam and Sadayappan [15], Boulet et al. [2] and Xue [18] 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 [7], [8] and Högstedt et al. [9], [10] 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.

In this paper we propose a new criterion for the selection of an efficient tile shape. This criterion emphasizes the minimization of the per process communication volume, while at the same time attaining simple, rectangular basic tile shapes, thus ensuring simplicity and applicability of the proposed methodology. 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 significantly decreases the overall execution time of the parallel algorithm.

In general, 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 a parameter determined by the computation and communication costs of the algorithm and the hardware features of the target architecture. In addition, we consider only rectangular basic tile shapes. Rectangular tiling is employed because when it comes to programming the vast majority of tiled iteration spaces for parallel architectures, this is the only legal or practical basic tile shape that can be applied. 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 [5]. Given a specific algorithm and iteration space, we propose a method to properly scale the sides of a rectangular tiling transformation, in order to reduce the total communication volume. 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, as will be shown in Section 4. 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: in the next section we present some preliminary concepts, i.e. our algorithmic model, some basic information about tiling transformation and the parallelization and mapping strategy. In Section 3 we provide some intuition of the method that will be presented in Section 4. In Section 5 we present a thorough experimental comparison of the communication-aware and the scheduling-aware tile shapes for two popular micro-kernels, while Section 6 concludes the paper by summarizing our contribution.

2. Preliminaries

Prior to delving into the core of our work, let us begin our discussion by briefly presenting essential background knowledge. Obviously, the importance of all proposed high performance optimizations is directly associated with the type of the applications addressed. Thus, we will initially define the algorithmic model considered here, refer to the tiling transformation for the partitioning of the algorithm's iteration space into tiles, as well as to the columnwise allocation scheme of tiles to processes.

2.1. Algorithmic Model

Our algorithmic model concerns PDE applications, which involve $N+1$-dimensional perfectly nested loops with constant flow dependencies. If D is the dependence matrix of the algorithm, then $rank(D) = N+1$, which means that the algorithm has $N+1$ linearly independent data dependence vectors. If $rank(D) < N+1$, then the iteration space can be partitioned into independent subspaces and parallelized without the use of tiling [4]. Furthermore, we consider rectangular iteration spaces. If the physical PDE domain is not rectangular, then we may consider either the smallest enclosing rectangle, or a mapping of the physical domain consisting of blocks with curvilinear borders into a set of adjacent rectangular blocks, as described in [11]. In the latter case, the forthcoming analysis is applied to each rectangular block, after a number of processes has been assigned to it.

Overall, the algorithms have the general form of Alg. 1, where l_i, u_i are constants, U is an $N+1$-dimensional matrix, $\vec{j} = (j_1, \ldots, j_{N+1})$, $\vec{d}^{(1)}, \ldots, \vec{d}^{(m)}$ are the dependence vectors and F is a linear function. In practical cases $N+1$ equals 3 or 4, since common PDEs model physical phenomena involving two or three spatial coordinates and possibly an additional temporal coordinate.

2.2. Tiling Transformation

When applying tiling transformation, the iteration space of an algorithm is partitioned into atomic $N+1$-dimensional parallelepiped areas, formed by $N+1$ inde-

Algorithm 1: algorithmic model

```
for $j_1 \leftarrow l_1$ to $u_1$ do
    ...
    for $j_N \leftarrow l_N$ to $u_N$ do
        for $j_{N+1} \leftarrow l_{N+1}$ to $u_{N+1}$ do
            $U[\vec{j}] = F(U[\vec{j} - \vec{d}^{(1)}], \ldots, U[\vec{j} - \vec{d}^{(m)}]);$
```

pendent families of parallel hyperplanes. Tiling 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. Equivalently, tiling transformation can be defined by $N+1$ linearly independent vectors, which correspond to the tile sides. The inverse H^{-1} of matrix H contains the side vectors of a tile as column vectors. In this paper we consider rectangular tiling transformations, i.e. matrices H and H^{-1} are diagonal. The number of iterations contained within a tile, that is, the size g of the tile equals the determinant of matrix H^{-1}, thus $g = det(H^{-1})$.

2.3. Mapping of Tiles to Processes

In order to parallelize a tiled iteration space we need to assign tiles to processes and schedule their execution legally, i.e. without violating the data dependencies of the original algorithm. According to the columnwise allocation, we assign a linear sequence of tiles to the same process and employ a linear time schedule as in [6] and [7]. In order to effectively utilize hardware with potential for communication and computation overlapping, each process assumes the execution of a sequence of tiles that are successive along the longest iteration space dimension. For homogeneous platforms and fully permutable iterative algorithms, related scientific literature [3], [6] has proven the optimality of the columnwise allocation of tiles to processes, as long as sequential pipelined execution along the longest dimension is preserved.

3. An Intuitive Approach

We are now ready to proceed with the definition of the problem addressed in this paper. First, we will formally define the problem, and then we will develop some intuition regarding the solution of the problem.

3.1. Definition of the Problem

The input of our problem is an $N+1$-dimensional nested loop with a rectangular iteration space ($X_1 \times \ldots \times X_N \times Z$) and a dependence matrix D of nonnegative, constant, flow dependencies, a tile size g and a fixed number of processes P. Z is considered as the longest dimension of the algorithm iteration space and should be brought to the innermost position (e.g. through loop permutation), so as to enable the pipelined, columnwise parallel execution of the algorithm. We assume that D is a diagonal matrix, i.e. the algorithm imposes only dependencies parallel to the axes. Even if this is not the case, we can apply all proposed methods by simply defining a new data dependence matrix $D' = diag(d'_i)$, where $d'_i = max\{d_i^{(j)}, 1 \leq j \leq m\}$, and implementing the indirect message passing techniques discussed in [16]. The goal of this paper is to determine a rectangular tiling transformation, that minimizes the communication volume of a typical, non-boundary process during the parallel execution of the tiled iteration space.

Note that problems complying to the form considered in this paper can often arise when solving applications derived from PDEs. The rectangular iteration space represents either the minimum rectangle that encloses the computation domain of interest, or a single block from the decomposition of the computation domain into adjacent blocks, as discussed in Section 2.1. In addition, PDE problems impose constant data dependencies, which can be kept nonnegative as long as backward discretization schemes are employed. Therefore, the assumption for constant, nonnegative flow dependencies poses no restrictions in applying our approach to actual PDE algorithms. On the other hand, this assumption allows us to perform a tiling transformation directly on the initial iteration space, without the prior use of a unimodular transformation, since all loops are fully permutable [17]. Furthermore, such algorithms usually impose data dependencies parallel to all space unitary vectors, thus effectively permitting only rectangular tiling transformations. Even if this is not the case, rectangular tiling is the only tiling transformation that can be applied by a program developer, since the complex code associated with a non-rectangular tiling transformation can be practically generated only by a compiler. For all these reasons, the assumption for a rectangular tiling transformation constitutes only a minor restriction.

3.2. Intuition of the Solution

As an introductory example, suppose one needs to solve an initial value/initial boundary problem (IVP/IBP) in a two-dimensional rectangular domain $X \times Y$ for a time window T. Let the tile size be g, the available number of processes 16 and the data dependencies of the algorithm $[\vec{d^{(1)}}, \vec{d^{(2)}}, \vec{d^{(3)}}] = [(2,0,0)^T, (0,2,0)^T, (0,0,2)^T]$. The rectangular tiling transformation that will be applied can be defined by a 3-dimensional diagonal matrix $H^{-1} = diag(h_x, h_y, h_t)$. Suppose that $T > X, Y$, thus we will map tiles along dimension t to the same process. In this case, we choose to partition the $X \times Y$ space in 16 tiles and appropriately adjust the tile height to conform with the restriction of the tile size. Thus, we will first determine h_x, h_y and subsequently we will set $h_t = g/h_x h_y$. We will investigate two alternative feasible tile shapes, namely $(h_x = X/4, h_y = Y/4, h_t = 16g/XY)$ and $(h_x = X/8, h_y = Y/2, h_t = 16g/XY)$.

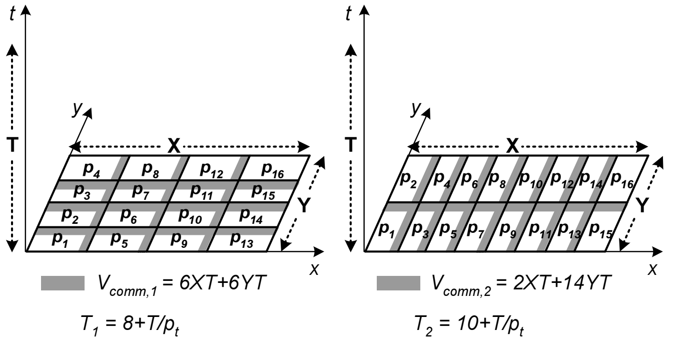

t
T
y
X
Y
x
p_4 p_8 p_{12} p_{16}
p_3 p_7 p_{11} p_{15}
p_2 p_6 p_{10} p_{14}
p_1 p_5 p_9 p_{13}
$V_{comm,1}$ = 6XT+6YT
T_1 = 8+T/p_t
t
T
y
X
Y
x
p_2 p_4 p_6 p_8 p_{10} p_{12} p_{14} p_{16}
p_1 p_3 p_5 p_7 p_9 p_{11} p_{13} p_{15}
$V_{comm,2}$ = 2XT+14YT
T_2 = 10+T/p_t

Figure 1. Total communication volume and parallel execution steps for two tiling transformations

Fig. 1 shows the projection of the tiled iteration space on the xy surface and its allocation to the 16 processes $(p_1 \cdots p_{16})$ for the two alternative tiling transformations. Note that each process is assigned a chain of tiles along the t dimension. The shaded parts of Fig. 1 represent the communication data for the two candidate tiling transformations. The communication volume derives from the boundary area between the processes ($3XT + 3YT$ for the first transformation and $XT + 7YT$ 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 $V_{comm,1} = 6XT + 6YT$ and $V_{comm,2} = 2XT + 14YT$.

If we apply linear scheduling defined by vector $\Pi = [1,1,1]$, then the tile $(4, 4, T/p_t)$ will be scheduled last according to the first tiling transformation, and will be executed at time step $T_1 = 8 + T/p_t$, while the respective last tile $(8, 2, T/p_t)$ of the second transformation will be executed at time step $T_2 = 10 + T/p_t$. 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 $X > 2Y$, then $V_{comm,1} > V_{comm,2}$, which means that the second transformation is superior in terms of total communication volume.

Consequently, when it comes to executing the above problem for $X > 2Y$, 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 $X = 4Y$, 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. In the following section we present a method to determine the communication-aware tiling transformation.

4. Communication-Aware Tile Shape

Contrary to related scientific work, we adopt a slightly different approach for the specification of the desirable communication-aware tile shape: Instead of defining a tiling transformation matrix H, we equivalently aim at determining an appropriate process topology $P = \prod_{i=1}^{N} P_i$ for the mapping of the parallel algorithm, according to the columnwise computation distribution scheme presented above. Indeed, the selection of the process topology implicitly enforces a particular tiling transformation: Determining a topology $P_1 \times \cdots \times P_N$ for the parallel mapping of an algorithm with iteration space $X_1 \times \cdots \times X_N \times Z$ effectively slices dimension X_i to P_i parts $(i = 1 \ldots N)$. This fact is equivalent to applying a rectangular tiling transformation described by the following matrix H^{-1}=

$$\begin{bmatrix} X_1/P_1 & 0 & \ldots & 0 & 0 \\ & & \ldots & & \\ 0 & 0 & \ldots & X_N/P_N & 0 \\ 0 & 0 & \ldots & 0 & (g\prod_{i=1}^{N} P_i)/(\prod_{i=1}^{N} X_i) \end{bmatrix}$$

where g is the tile size dictated by the underlying architecture (processor speed, interconnection bandwidth etc.) and affecting the grain of the parallelism.

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.

According to [7], [8], given P processes for the mapping of an $N + 1$-dimensional algorithm across the outermost N dimensions on an N-dimensional process grid, the scheduling-aware tiling transformation can be obtained as a feasible solution to the following optimization problem:

$$\left.\begin{array}{lcl} P_i & \rightarrow & \sqrt[N]{P}, P_i \in \mathbb{N}, 1 \leq i \leq N \\ P & = & \prod_{i=1}^{N} P_i \end{array}\right\} \tag{1}$$

A process topology complying to (1) minimizes 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.

An alternative approach would be to consider a more communication-aware process topology, as is the case with the one provided by the following lemma:

Lemma 1. *Let $X_1 \times \cdots \times X_N \times Z$ be the iteration space of an $N+1$-dimensional nested loop algorithm, that imposes data dependencies $[d_1, \ldots, 0]^T, \ldots, [0, \ldots, d_{N+1}]^T$. Let P be the number of processes available for the parallel execution of the algorithm. If there exist $P_i \in \mathbb{N}$, such that*

$$P = \prod_{i=1}^{N} P_i \tag{2}$$

and

$$\frac{d_i P_i}{X_i} = \frac{d_j P_j}{X_j}, 1 \leq i, j \leq N \tag{3}$$

then process topology $P_1 \times \cdots \times P_N$ minimizes inter-process communication for the tiled algorithm on P processes. Also, (3) is equivalent to

$$P_j = \frac{X_j}{d_j} \sqrt[N]{\frac{P \prod_{i=1}^{N} d_i}{\prod_{i=1}^{N} X_i}}, 1 \leq j \leq N \tag{4}$$

Proof. According to (2), it holds

$$P_N = \frac{P}{P_1 \times \cdots \times P_{N-1}} \tag{5}$$

Each process assumes $\lceil X_i/P_i \rceil$ iterations along direction i, where $1 \leq i \leq N$. For the sake of simplicity, we assume that $\lceil X_i/P_i \rceil \simeq X_i/P_i$. Due to the data dependencies of the algorithm, a non-boundary process is required to send $d_i \prod_{\substack{j=1 \\ j \neq i}}^{j=N} \frac{X_j}{P_j} Z$ data along direction i. Thus, the total communication volume V_{comm} of a process can be obtained by the following expression:

$$\begin{aligned} V_{comm} &= d_1 \prod_{\substack{i=1 \\ i \neq 1}}^{N} \frac{X_i}{P_i} Z + \cdots + d_N \prod_{\substack{i=1 \\ i \neq N}}^{N} \frac{X_i}{P_i} Z \\ &= \frac{Z \prod_{i=1}^{N} X_i}{P} \left(\frac{d_1 P_1}{X_1} + \cdots + \frac{d_N P_N}{X_N} \right) \end{aligned} \tag{6}$$

Using (5), (6) can be written by substituting P_N as follows:

$$V_{comm} = \frac{Z \prod_{i=1}^{N} X_i}{P} \sum_{i=1}^{N-1} \frac{d_i P_i}{X_i} + \frac{d_N Z \prod_{i=1}^{N} X_i}{X_N P_1 \ldots P_{N-1}} \tag{7}$$

Note that V_{comm} is substantially a function of $P_1, \ldots, P_{N-1}$ (formally: $V_{comm} : \mathbb{N}^{N-1} \rightarrow \mathbb{R}$). Let $\overline{V}_{comm}$ be the real extension of V_{comm}, defined by (7) for $P_j \in \mathbb{R}, 1 \leq j \leq N$ ($\overline{V}_{comm} : \mathbb{R}^{N-1} \rightarrow \mathbb{R}$). For a stationary point $(P_1, \ldots, P_{N-1})$ of $\overline{V}_{comm}$ and $1 \leq j \leq N-1$ it holds:

$$\frac{\partial \overline{V}_{comm}}{\partial P_j} = 0 \Rightarrow \frac{d_j P_j}{X_j} = \frac{d_N P_N}{X_N} \tag{8}$$

Also,

$$\frac{\partial^2 \overline{V}_{comm}}{\partial P_j{}^2} = \frac{2 d_N \prod_{i=1}^{N} X_i}{X_N P_1 \ldots P_j^3 \ldots P_{N-1}} > 0 \tag{9}$$

Because of (8) and (9), $\overline{V}_{comm}$ has a minimum at $(P_1, \ldots, P_{N-1})$, and as $P_i \in \mathbb{N}, 1 \leq i \leq N-1$, this will be the minimum of V_{comm}, as well. Therefore, the communication data is minimal when a topology $P_1 \times \cdots \times P_N$ satisfying (8) is assumed. Finally, it holds

$$\frac{P \prod_{i=1}^{N} d_i}{\prod_{i=1}^{N} X_i} = \frac{d_1 P_1}{X_1} \cdots \frac{d_N P_N}{X_N} \tag{10}$$

By combining (10) with (8), we can easily deduce (4). □

It should be noted that (4) does not always define a valid integer process topology: it is possible that $P_j \notin \mathbb{N}$ for some value j with $1 \leq j \leq N$. 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 (4). In practice, as $N+1$ does not exceed 3 or 4, and P 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 $(100 \ldots 10k) \times (100 \ldots 10k) \times (100 \ldots 10k) \times Z$, data dependencies $[(1 \ldots 3, 0, 0, d), (0, 1 \ldots 3, 0, d'), (0, 0, 1 \ldots 3, d'')]$ and for $100 \leq P \leq 1k$. The execution time equaled on average 21 msec, while under no circumstances did it exceed 0.9 sec.

Algorithm	*Iteration spaces*	*Data dependencies*	*#Processes*
ADI	$\{16, 32, 64, 128, 256\} \times 256 \times 16k$	$(d_1, d_2, d_3) = (1, 1, 1)$	16/12
DE$_{XYT}$	$\{16, 32, 64, 128, 256\} \times 256 \times 16k$	$(d_1, d_2, d_3) = (3, 3, 1)$	16
DE$_{XYT}$	$1k \times \{32, 64, 128, 256, 512\} \times 2k$	$(d_1, d_2, d_3) = (3, 3, 1)$	16
DE$_{TXY}$	$1k \times \{32, 64, 128, 256, 512\} \times 2k$	$(d_1, d_2, d_3) = (1, 3, 3)$	16

Table 1. Configuration layout of conducted experiments

5. Experimental Results

In order to evaluate the comparative performance of the proposed communication-aware tiling against the standard scheduling-aware alternative, we measured the total parallel execution time of two micro-kernel benchmarks, namely Alternating Direction Implicit (ADI) integration and the Diffusion Equation (DE), for various iteration spaces and tile grains. ADI integration is a three-dimensional nested loop stencil computation, used for solving partial differential equations [13]. On the other hand, the DE micro-kernel arises from the discretization of the diffusion PDE on a two-dimensional spatial domain using a second-order discretization scheme for the spatial derivatives. DE imposes higher communication to computation demands with respect to ADI, thus it facilitates the experimental evaluation of the efficiency of the communication-aware tiling in terms of scalability.

Our experimental platform is an 16-node Linux cluster (Pentium-III CPU at 800 MHz, 256 MB RAM, 16 KB L1 I cache, 16 KB L1 D cache, 256 KB L2 cache), interconnected with 100 Mbps FastEthernet. Each cluster node runs Linux kernel 2.4.26. We used MPICH v. 1.2.6 MPI implementation, configured with the Intel C++ compiler v. 8.1.

Table 1 displays the selected iteration spaces and data dependencies for the various experiments we conducted. DE$_{XYT}$ corresponds to a DE micro-kernel with two outermost spatial loops (XY) and one innermost temporal (T). Similarly, DE$_{TXY}$ denotes a DE algorithm with an outermost temporal and two innermost spatial loops. Due to the second-order accuracy of the discretization scheme employed in DE, the data dependence components along the x and y dimensions are equal to 3.

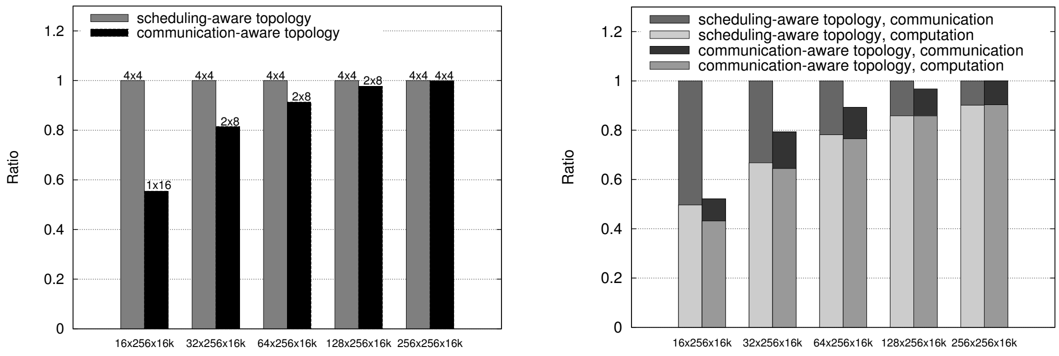
Recall that related theory requires the longest algorithm dimension at the innermost position, therefore both DE variations are possible, depending on the nature and complexity of the particular problem. Due to our underlying infrastructure, we considered 16 processes for the parallel execution of the tiled algorithms, and occasionally also 12, in order to evaluate the proposed theory for additional process topology combinations. All results are averaged over at least three independent executions. We shall refer to the iteration spaces $\{16, 32, 64, 128, 256\} \times 256 \times 16k$ as *set 1*, whereas the iteration spaces $1k \times \{32, 64, 128, 256, 512\} \times 2k$ will be addressed to as *set 2*.

5.1. ADI Integration

The results for ADI integration are presented in Fig. 2 (set 1, 16 processes) and Fig. 3 (set 1, 12 processes). All results have been normalized with respect to the ones obtained under the scheduling-aware tiling transformation, so as to allow for straightforward quantitative comparison: the height difference between the bars in each bar pair equals the percentage variation in the total execution time. Assuming 16 processes, we compare the scheduling-aware tiling transformation, that corresponds to a 4×4 process topology, with the communication-aware tiling approach, that implements the tile shape selection methodology described in Section 4. Similarly, when using 12 processes, we compare against the scheduling-aware 3×4 topology. Regarding the tile size, we tried all possible tile heights between 1 and 200, and in each case consider the tile size leading to the minimum overall execution time. However, in Fig. 4, we also present detailed granularity results for ADI on 16 processes, in order to illustrate the relative advantage of the communication-aware tiling over the scheduling-aware one for all tile sizes considered.

In both cases (16 and 12 processes), there is a significant performance improvement when applying the communication-aware topology. This improvement is particularly evident for relatively asymmetric iteration spaces. For example, when using 16 processes we observe an impressive performance improvement of 45% for the $16 \times 256 \times 16k$ iteration space, which gradually drops to 19% ($32 \times 256 \times 16k$), 9% ($64 \times 256 \times 16k$) and finally 2% ($256 \times 256 \times 16k$). Similarly, when using 12 processes, we have a performance improvement that ranges from 41% to 3%.

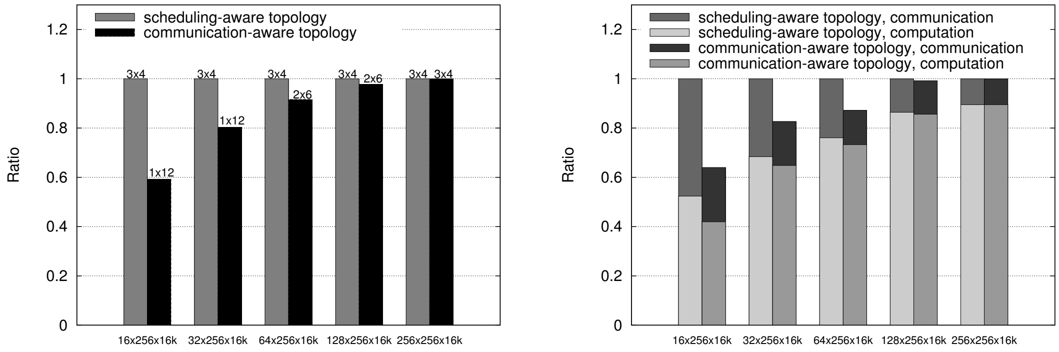
In order to further investigate the performance improvement of the communication-aware tiling transformation over the scheduling-aware alternative, we performed profiling of the per tile communication and computation times. We timed the computation time of



(a) total execution time

(b) execution time profiling

Figure 2. ADI integration, set 1, 16 processes



(a) total execution time

(b) execution time profiling

Figure 3. ADI integration, set 1, 12 processes

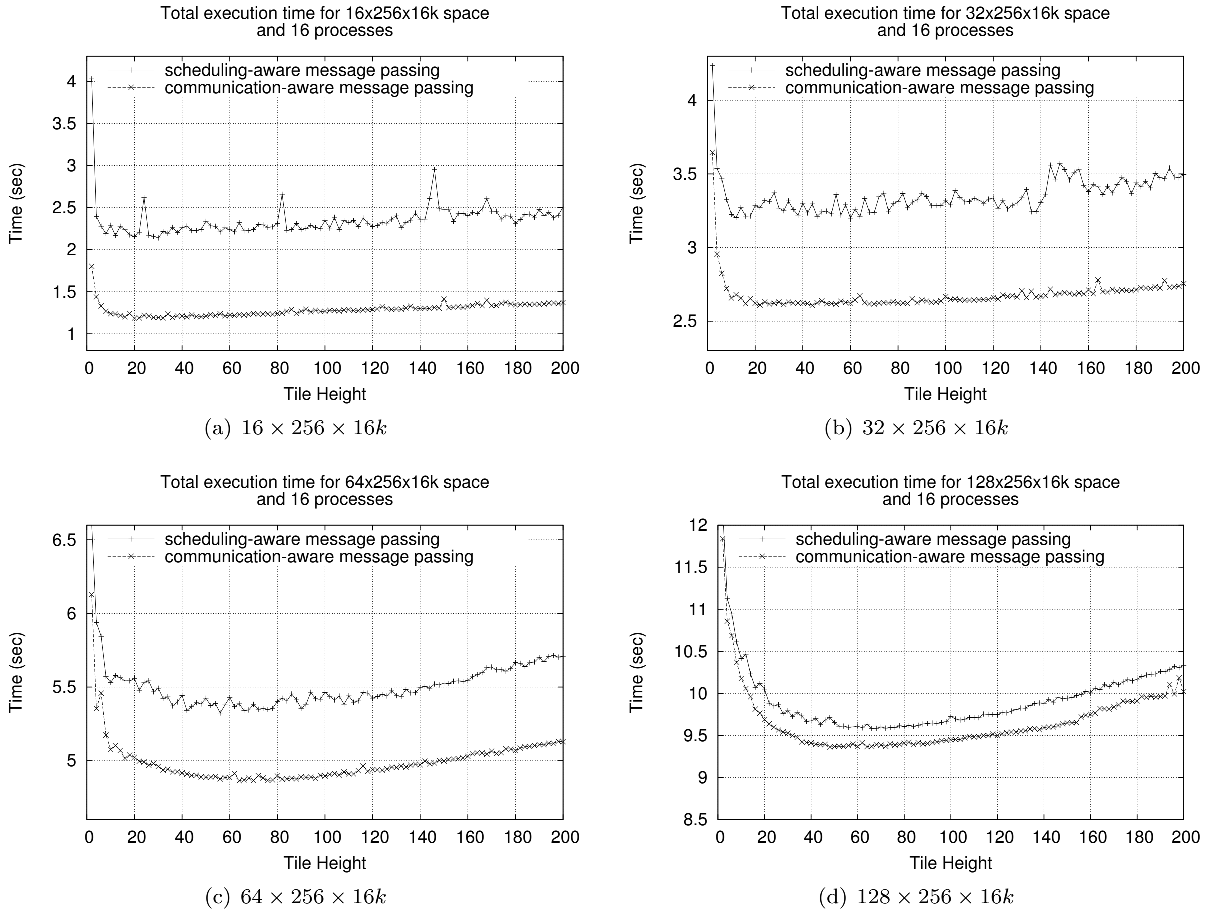
all processes on a per tile basis, as well as the communication time associated both with MPI primitives and packing/unpacking communication data. We present the maximum computation and maximum communication time, reduced over all processes and normalized to the {maximum computation time+maximum communication time} under the scheduling-aware tiling transformation. Note that 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. 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. Wherever this reduction is relatively high, we also obtain a significant reduction of the total execution time. Finally, Fig. 4 confirms that communication-aware tiling outperforms the scheduling-aware alternative for all possible tile grains.

5.2. Diffusion Equation

We performed analogous measurements for the DE micro-kernel. Fig. 5 depicts the total parallel execution times for 16 processes, XYT-loop form and iteration spaces belonging to set 1, Fig. 6 displays the times obtained for 16 processes, XYT-form and set 2, while Fig. 7 shows the respective times for 16 processes, TXY-form and set 2.

The selection between the XYT- and TXY-form of the algorithm depends upon the relative temporal vs spatial complexity of the particular physical problem. Moreover, the selection between the various iteration spaces of sets 1, 2 models the possibility of both "square" and "rectangular" surfaces. Note that the asymmetry of some iteration spaces used in the experimental series is not an artificial selection; on the contrary, asymmetric computational domains and iteration spaces are very common in PDEs (i.e. thermal diffusion in a bar).

In all cases, the experimental results indicate a clear



(a) $16 \times 256 \times 16k$

(b) $32 \times 256 \times 16k$

(c) $64 \times 256 \times 16k$

(d) $128 \times 256 \times 16k$

Figure 4. ADI integration, set 1, 16 processes, granularity results

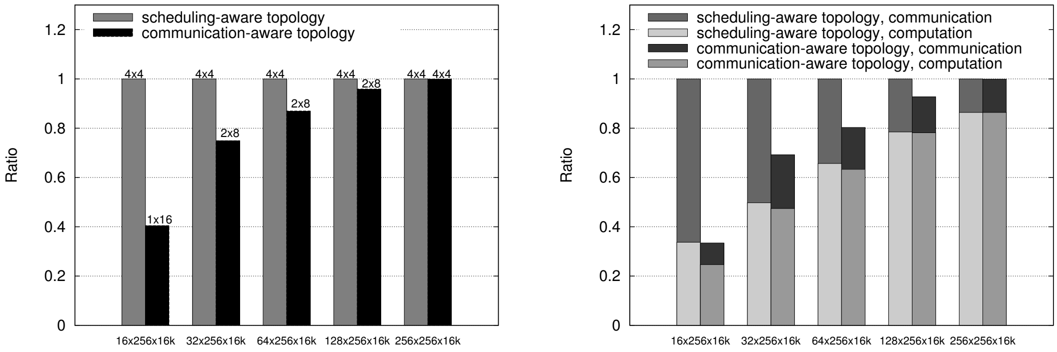
advantage of the communication-aware tiling transformation. By comparing the ADI and the DE results obtained for set 1, we can further conclude that the benefit attained when resorting to a communication-aware tiling transformation is even more profound for algorithms with intrinsic high communication demands. Indeed, the performance improvement of the XYT-form DE under the communication-aware tiling ranges from 60% for the $16 \times 256 \times 16k$ iteration space to 5% ($128 \times 256 \times 16k$). This can be attributed to the fact that DE requires substantially more communication than ADI, whereas both algorithms exhibit similar computational complexity. Based on that observation, we can reasonably assume that for even larger dependence factors d_i (e.g. if more previous time or space values are employed during the discretization process, in order to increase the accuracy), the relative performance benefit of the communication-aware tiling transformation will be more significant.

6. Conclusions

This paper presents a novel technique for the selection of an efficient and feasible tile shape for the message passing parallelization of fully permutable nested loop 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 and a specific algorithm (iteration space, data dependencies). The presented technique can be easily combined with the `MPI_Cart_create` primitive, to deliver efficient Cartesian process topologies.

More importantly, we extensively evaluate the effectiveness of the proposed communication-aware tiling with the aid of typical micro-kernel benchmarks, namely ADI integration and the discretized Diffusion Equation. Summarizing the experimental results, we conclude that:

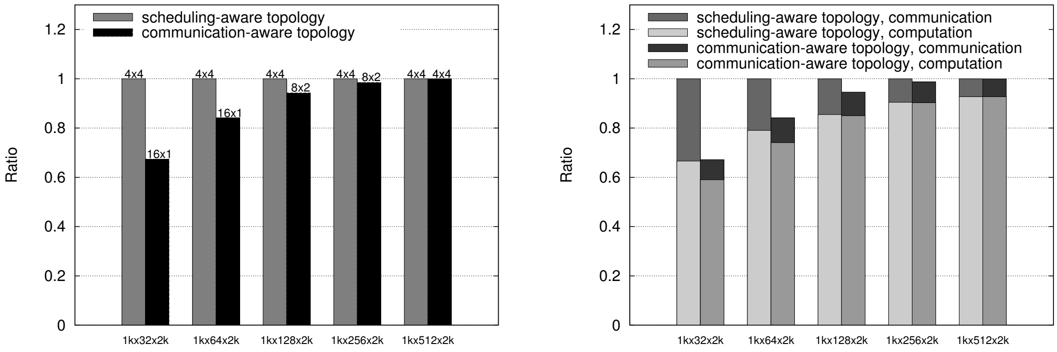
- the proposed communication-aware tiling is par-



(a) total execution time

(b) execution time profiling

Figure 5. Diffusion Equation, XYT-form, set 1, 16 processes



(a) total execution time

(b) execution time profiling

Figure 6. Diffusion Equation, XYT-form, set 2, 16 processes

ticularly efficient for the parallelization of nested loop algorithms on distributed memory architectures when the algorithm exhibits asymmetric data dependencies and/or iteration space dimensions

- the same observation holds for algorithms imposing relatively high communication-to-computation demands for the underlying hardware/network parallel infrastructure and lower level message passing software
- in any case, communication-aware tiling outperformed the scheduling-aware alternative for all benchmark applications, iteration spaces, data dependencies and tile sizes considered here

Acknowledgment

The Project is co-funded by the European Social Fund (75%) and National Resources (25%).

References

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

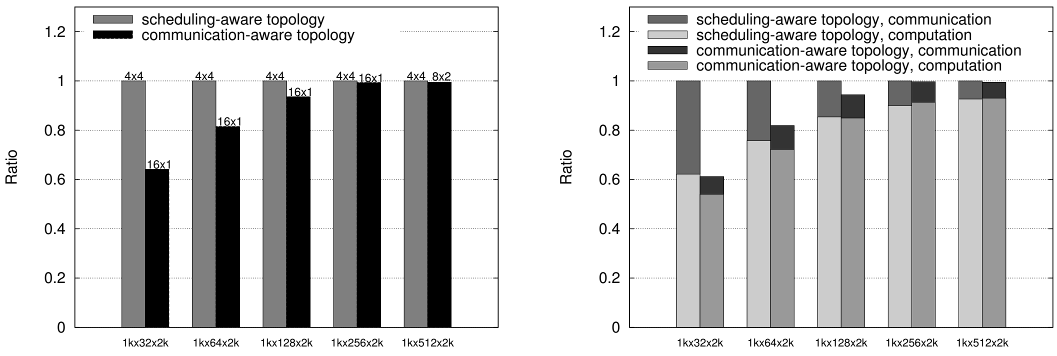
[2] P. Boulet, A. Darte, T. Risset, and Y. Robert. (Pen)-ultimate Tiling? *INTEGRATION, The VLSI Jounal*, 17:33–51, 1994.

[3] P. Calland, J. Dongarra, and Y. Robert. Tiling with Limited Resources. In *Application Specific Systems, Architectures and Processors*, pages 229–238, Jun 1997.

[4] E. D'Hollander. Partitioning and Labeling of Loops by Unimodular Transformations. *IEEE Trans. on Parallel and Distributed Systems*, 3(4):465–476, Jul. 1992.

[5] 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.

[6] G. Goumas, A. Sotiropoulos, and N. Koziris. Minimizing Completion Time for Loop Tiling with Computa-



(a) total execution time

(b) execution time profiling

Figure 7. Diffusion Equation, TXY-form, set 2, 16 processes

tion and Communication Overlapping. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, San Francisco, USA, Apr 2001.

[7] 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.

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

[9] 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*, pages 201–211, Saint Malo, France, 1999.

[10] K. Högstedt, L. Carter, and J. Ferrante. On the Parallel Execution Time of Tiled Loops. *IEEE Trans. on Parallel and Distributed Systems*, 14(3):307–321, Mar 2003.

[11] S. Ichikawa and S. Yamashita. Static Load Balancing of Parallel PDE Solver for Distributed Computing Environment. In *Proceedings of Int'l Conf. Parallel and Distributed Computing Systems*, pages 399–405, Las Vegas, Nevada, USA, 2000.

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

[13] 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.

[14] H. Ohta, Y. Saito, M. Kainaga, and H. Ono. Optimal Tile Size Adjustment in Compiling General DOACROSS Loop Nests. In *Proceedings of the International Conference on Supercomputing*, pages 270–279, Barcelona, Spain, Jul 1995.

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

[16] P. Tang and J. Zigman. Reducing Data Communication Overhead for DOACROSS Loop Nests. In *Proceedings of the International Conference on Supercomputing*, pages 44–53, Manchester, UK, Jul 1994.

[17] M. Wolf and M. Lam. A Loop Transformation Theory and an Algorithm to Maximize Parallelism. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):452–471, Oct 1991.

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