Coarse-grain Parallel Execution for 2-dimensional PDE Problems

Georgios Goumas, Nikolaos Drosinos, Vasileios Karakasis, and Nectarios Koziris

National Technical University of Athens
School of Electrical and Computer Engineering
Computing Systems Laboratory
{goumas,ndros,bkk,nkoziris@cslab.ece.ntua.gr}

Abstract

This paper presents a new approach for the execution of coarse-grain (tiled) parallel SPMD code for applications derived from the explicit discretization of 2-dimensional PDE problems with finite-differencing schemes. Tiling transformation is an efficient loop transformation to achieve coarse-grain parallelism in such algorithms, while rectangular tile shapes are the only feasible shapes that can be manually applied by program developers. However, rectangular tiling transformations are not always valid due to data dependencies, and thus requiring the application of an appropriate skewing transformation prior to tiling in order to enable rectangular tile shapes. We employ cyclic mapping of tiles to processes and propose a method to determine an efficient rectangular tiling transformation for a fixed number of processes for 2-dimensional, skewed PDE problems. Our experimental results confirm the merit of coarse-grain execution in this family of applications and indicate that the proposed method leads to the selection of highly efficient tiling transformations.

1 Introduction

A large variety of physical phenomena is described by partial differential equations (PDEs). A common strategy to numerically solve PDEs is to discretize the derivatives involved in the equation by finite-differences. At present, finite-difference methods provide a powerful approach to solve differential equations and are widely used in many field of applied sciences. Finite-differencing a partial derivative of function $u(i,j)$ involves the construction of a proper stencil employing discrete values of u at points $i \pm \kappa\Delta x$ and $j \pm \lambda\Delta y$ of a computational grid which in turn leads to the construction of an n-dimensional nested loop with n linearly independent data dependencies. This fact prohibits DOALL parallelism, which would enable efficient execution of the nested loop onto parallel architectures. Consequently, the parallel execution of discretized PDEs requires frequent communication that greatly degrades performance in parallel platforms. In order to alleviate this overhead, researchers have proposed the application of the tiling transformation [7] to algorithms with n linearly independent data dependencies. Tiling groups neighboring iterations into a single group (tile or supernode) which is formed by an n-dimensional parallelogram. The algorithm is then scheduled so that communication is interleaved with the parallel execution of *tiles* instead of single *iterations*, achieving in this way coarse-grain parallelism.

The problem of the application of general tiling transformations has been attacked in [1] and [2], where it is shown that the general case can be handled only by an automatic parallelizing compiler. However, the problem of efficiently mapping non-rectangular tiles to a fixed number of processes is still open. On the other hand, rectangular tiling transformation is valid only when the nested loop is fully permutable [9], i.e., when there is no negative element in the dependence vectors. Unfortunately, this is not the general case for discretized PDEs, where the employment of $u(i+\kappa\Delta x)$ terms in the construction of the stencil incurs negative coefficients in some data dependencies. Traditional loop theory recommends the application of the skewing transformation prior to applying tiling in order to transform the loop nest into a fully permutable one. This fact, however, generates a number of new problems since the shape of the original iteration space changes after skewing, making code development quite error-prone and, more importantly, greatly complicating the process of mapping and scheduling the tasks to a fixed number of processes. The above problems make parallel developers refrain from applying the tiling transformation to PDE codes.

This research is supported by the Pythagoras II Project (EPEAEK II), co-funded by the European Social Fund (75%) and National Resources (25%).

However, since tiling is a significant transformation to attain performance for DOACROSS loop nests executed in NUMA parallel architectures, we aim at providing the parallel programmer with a detailed method to develop coarse-grain (tiled) parallel code. In order to avoid great complexities imposed by the general case, we restrict our attention to 2-dimensional PDE problems. Note that, in principle, PDEs have a dimension of 4 at maximum, modelling physical problems with one, two or three spatial and one temporal dimension. With the approach presented in this paper, the developer will be able to cope with problems with the temporal and one spatial coordinate. However, in many cases it is possible to split a 4-dimensional problem into three 2-dimensional ones, as in atmospheric modelling [4]. Moreover, as stated in [5], several physical problems in one spatial dimension correspond to physical systems in three dimensions which have cylindrical or spherical symmetry.

This paper presents a detailed guideline to the programmer for the development of efficient SPMD MPI code for the execution of 2-dimensional PDE codes onto a parallel platform where remote access times are significantly larger than local access times. For this reason we summon previous knowledge from traditional loop transformation and scheduling theory and clarify the application of skewing and tiling transformations. In addition, since the parallel execution time of a tiled algorithm is extremely sensitive to the selection of the tiling parameters (size and shape), the execution times may result to be quite disappointing if these parameters are not properly fine-tuned. Overall, in this paper we provide detailed steps of coarse-grain SPMD code development, employ a cyclic mapping scheme and present an efficient method to select proper tiling transformations in the skewed and tiled space that maximize process concurrency. The rest of the paper is organized as follows: The next section presents the algorithmic model along with some background knowledge. Section 3 illustrates the straightforward, fine-grain execution of PDE applications while Section 4 proposes explicit guidelines for the development of coarse-grain SPMD code broken down into steps, together with our method to determine efficient tiling transformations. In Section 5 we investigate the efficiency of coarse-grain execution and compare our proposed tiling transformation to alternative ones. Finally, in Section 6 we close this paper with our concluding remarks.

2 Preliminaries

2.1 Algorithmic Model

In this paper we study nested loops resulting from the discretization of 2-dimensional PDEs. We consider rectangular computational domains, thus the iteration space J of the nested loops is defined as: $J = \{\vec{j} = (j_1, j_2) : l_1 \leq j_1 \leq u_1 \wedge l_2 \leq j_2 \leq u_2\}$. Finite-differencing discretization schemes lead to loop-carried data dependencies. In addition, if we consider explicit discretization schemes, the data dependencies are lexicographically positive [6]. The $2 \times m$ matrix D contains the data dependencies. The algorithms are of the form shown in Algorithm 1.

Algorithm 1: algorithmic model

```
1 for (j1 = l1; j1 < u1; j1++) do
2     for (j2 = l2; j2 < u2; j2++) do
3         U[j⃗] = F(U[j⃗ − d⃗(1)], ..., U[j⃗ − d⃗(m)]);
```

Example 1: Thermal diffusion in a bar of length X' for a time window T' is described by the equation $\frac{\partial u}{\partial t} - \theta \frac{\partial^2 u}{\partial x^2} = F(t,x)$. With finite-differences, the partial derivatives are approximated by $\frac{\partial u}{\partial t} = \frac{u_{t+1,x} - u_{t,x}}{\Delta t}$ and $\frac{\partial^2 u}{\partial x^2} = \frac{u_{t,x+1} - 2u_{t,x} + u_{t,x-1}}{\Delta x^2}$. Thus, we obtain $u_{t+1,x} = u_{t,x} + \theta \frac{\Delta t}{\Delta x^2}(u_{t,x+1} - 2u_{t,x} + u_{t,x-1}) + \Delta t F(t,x)$, which is implemented by the 2-dimensional nested loop of Algorithm 2 ($T = \frac{T'}{\Delta t}$, $X = \frac{X'}{\Delta x}$, $r = \frac{\Delta t}{\Delta x^2}$). The dependence matrix of the algorithm is: $D = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$.

Algorithm 2: nested loop for thermal diffusion

```
1 for (t = 0; t < T; t++) do
2     for (x = 1; x < X; x++) do
3         U[t+1][x] = (1 − 2θr)U[t][x] + θr(U[t][x −
          1] + U[t][x + 1]) + ΔtF(t, x);
```

2.2 Tiling and Skewing Transformations

In a *tiling* transformation the original iteration space of an algorithm is partitioned into identical n-dimensional parallelepiped areas (tiles or supernodes). Tiles are then assigned to processes and executed uninterruptedly according to a linear schedule [7], while communication occurs before and after the computations within a single tile. Tiling transformation is uniquely defined by the n-dimensional square matrix H. Each row vector of H is perpendicular to one family of hyperplanes forming the tiles. Dually, tiling transformation can be defined by n linearly independent vectors, which represent the sides of the tiles. Similar to matrix H, matrix H^{-1} contains the side vectors of a tile as column vectors. Note that a rectangular tiling transformation is defined by a diagonal matrix H. In the 2-dimensional problems under consideration we denote $H^{-1} = diag(c_t, c_x)$. Thus, if $\vec{j} = (j_1, j_2) \in J$ then $\vec{j}^S$ corresponds to the coordinates of the tile $\vec{j}$ belongs to, and it will hold: $\vec{j}^S = (j_1^S, j_2^S) = (\lfloor \frac{j_1}{c_t} \rfloor, \lfloor \frac{j_2}{c_x} \rfloor)$.

A tile dependence arises when a point in the interior of one tile is dependent on a point in the interior of another. The tile dependence matrix D^S is determined by: $D^S = \{\vec{d^S} : \vec{d^S} = \lfloor H(\vec{j_0} + \vec{d}) \rfloor,\ \vec{d} \in D,\ \vec{j_0} \in J \wedge 0 \leq$

$\lfloor H\vec{j_0} \rfloor \leq 1\}$, where $\vec{j_0}$ denotes the index points lying inside the first complete tile at origin of the original iteration space. For a dependence matrix D and a tiling transformation H, it must hold $HD \geq 0$ for the tiling to be legal. This condition ensures that tiles are atomic and that the initial execution order is preserved [7]. In the opposite case, any execution order of tiles would result in a deadlock, since the tile dependence matrix D^S would contain lexicographically negative vectors, consequently preventing the application of a valid linear schedule to the tiled space [8].

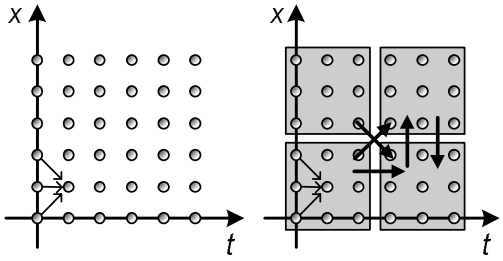


Figure 1. Application of rectangular tiling transformation to thermal diffusion.

Example 2: Figure 1 displays the application of a rectangular tiling transformation defined by $H^{-1} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$ to the thermal diffusion algorithm of Example 1. Thin arrows depict the dependencies of the original algorithm, while bold arrows depict the dependencies between tiles, i.e. the column vectors of matrix D^S. It holds $D^S = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & -1 & -1 \end{bmatrix}$. Since $HD \not> 0$, D^S contains a lexicographically negative dependence $\vec{d_5^S} = (0, -1)$ that leads to a deadlock. Note in Figure 1, that there exist tiles which are interdependent.

Loop *skewing* is a linear and unimodular loop transformation described by a lower triangular matrix [9], that we denote W. Since in this paper we are dealing with 2-dimensional spaces, the skewing matrix W assumes the form $W = \begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix}$. We will call α the skewing factor. Thus, an iteration $\vec{j}$ of the original iteration space is mapped to $\vec{j'} = W\vec{j}$. After the application of skewing transformation—similar to any other linear loop transformation—the bounds of the transformed iteration space need to be determined. In the case of skewing the transformed iteration space J' is $J' = \{\vec{j'} = (j'_1, j'_2) : l_1 \leq j'_1 \leq u_1 \wedge l_2 + \alpha j'_1 \leq j'_2 \leq u_2 + \alpha j'_1\}$. Accordingly, the dependence matrix of the transformed space is $D' = WD$.

Example 3: If we apply the skewing transformation defined by $W = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ to the algorithm of the previous examples, the dependence matrix becomes $D' = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$. The transformed nested loop is listed in Algorithm 3. Figure 2 visualizes

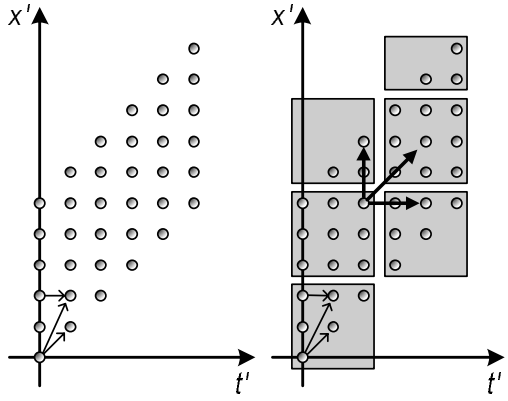


Figure 2. Successive application of skewing and tiling transformation.

the result of skewing and subsequent rectangular tiling transformation to the original iteration space. In this case, it holds that $HD' \geq 0$ and thus rectangular tiling transformation is valid.

Algorithm 3: nested loop for skewed thermal diffusion

```
1 for (t' = 0; t' < T; t'++) do
2     for (x' = 1 + t'; x' < X + t'; x'++) do
3         t = t'; x = x' − t';
          U[t + 1][x] = (1 − 2θr)U[t][x] + θr(U[t][x −
          1] + U[t][x + 1]) + ΔtF(t, x);
```

2.3 Linear Loop Schedules

We consider linear loop schedules for the parallel execution of iterations or tiles onto parallel architectures. A point $\vec{j} \in J$ scheduled according to a linear time schedule $\Pi = (\pi_1, \pi_2)$ will be executed at $t_j = \lfloor \frac{\Pi\vec{j} + t_0}{disp\Pi} \rfloor$, where $t_0 = -min\Pi\vec{j} : \vec{j} \in J$ and $disp\Pi = min\Pi\vec{d} : \vec{d} \in D$ [8]. In this paper we will apply wavefront scheduling defined by $\Pi = (1, 1)$ to schedule tiled iteration spaces.

3 Fine-grain Parallel PDE Execution

Prior to delving into the details of efficient, coarse-grain execution of PDEs in parallel architectures, we will examine the basic alternative: fine-grain parallel execution. The advantage of this approach is that it can be directly applied to the original algorithm, without the use of any loop transformations, leading to a simple code development process. However, due to the dependencies of discretized PDEs, there is a need for frequent process communication in order to exchange boundary data. The pseudo-code of the fine-grain parallel approach using MPI primitives is summarized in Algorithm 4. The development of this code is straightforward, but it is obvious that the processes need to

synchronize and exchange data at every time step t due to the problem's dependencies.

Algorithm 4: pseudo-code for fine-grain parallel execution of thermal diffusion

```
1  for (t = 0; t < T; t++) do
2      if exists_up(p_id) then
3          MPI_Send(buf1, 1, MPI_DOUBLE, up, tag,
           MPI_COMM_WORLD);
4          MPI_Recv(buf3, 1, MPI_DOUBLE, up, tag,
           MPI_COMM_WORLD, &status);
5      if exists_down(p_id) then
6          MPI_Recv(buf4, 1, MPI_DOUBLE, down, tag,
           MPI_COMM_WORLD, &status);
7          MPI_Send(buf2, 1, MPI_DOUBLE, down, tag,
           MPI_COMM_WORLD);
8      for (x = 1; x < X/P; x++) do
9          Ul[t+1][x] = (1 - 2θr)Ul[t][x] + θr(Ul[t][x -
           1] + Ul[t][x + 1]) + ΔtF(t, x);
```

4 The Coarse-grain Approach

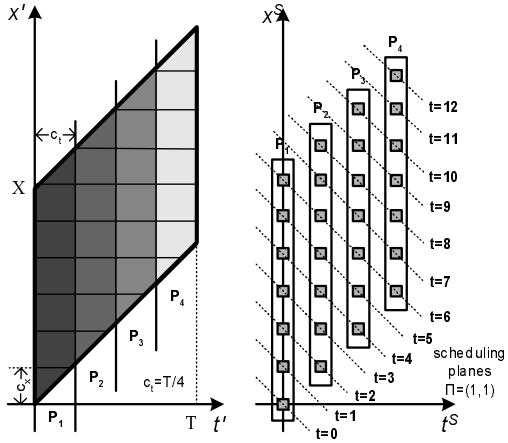
In this section we present a detailed approach for coarse-grain execution of 2-dimensional PDEs onto parallel architectures. Our problem input is a PDE with an initial domain $T' \times X'$, the discretization steps Δt, Δx, the dependence matrix D and the number of available processes P. The iteration space is $T \times X$ with $T = \frac{T'}{\Delta t}$ and $X = \frac{X'}{\Delta x}$. In order to achieve efficient parallel execution of the problems under consideration, three issues need to be addressed: the selection of an efficient rectangular tiling transformation, the mapping and scheduling strategy of tiles and the final SPMD code development. However, as mentioned before, prior to applying rectangular tiling transformation, we need to transform the dependence coordinates of our algorithms into nonnegative ones. Overall, the proposed approach can be decomposed into the following five discrete steps:

Step 1: Skew the Iteration Space
Our goal is to make all dependence coordinates nonnegative and prepare the iteration space for the application of rectangular tiling in the next step. Explicit discretization schemes lead to lexicographically positive dependence vectors that may contain negative coordinates. In our 2-dimensional case nonnegative coefficients are encountered in the second element d_2 of dependence vectors $\vec{d} = (d_1, d_2)$. After the application of a skewing transformation defined by $W = \begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix}$ we demand that $d'_2 \geq 0$ for all the transformed dependence vectors $\vec{d'} \in D' = WD$. Hence, we choose a skewing factor α such that $\alpha d_1 + d_2 \geq 0 \; \forall \; \vec{d} \in D : \; d_2 < 0$.

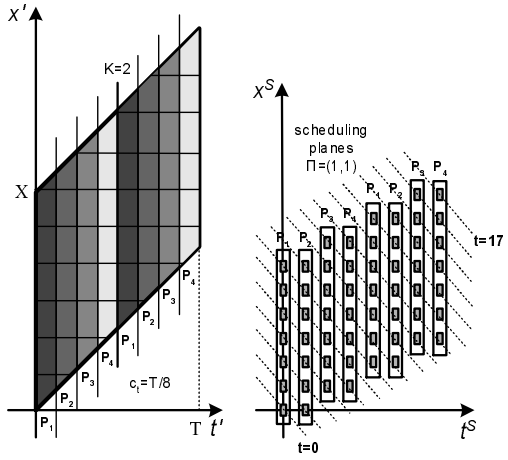
Step 2: Apply Rectangular Tiling Transformation Since all dependencies in the skewed space have been made nonnegative, we are now free to apply rectangular tiling transformation defined by $H^{-1} = \begin{bmatrix} c_t & 0 \\ 0 & c_x \end{bmatrix}$. Note, however, that since the selection of c_t and c_x affects the tile mapping and scheduling of our transformed application, we need to postpone this task until the next step.

Step 3: Map Tiles to Processes and Schedule Tasks
In this step we need to efficiently partition the skewed and tiled iteration space among a fixed number of processes P, and select the parameters of the tiling transformation (c_t and c_x) from the previous step. Since we are working in a 2-dimensional domain, we consider a chain of P processes. According to this approach, each process assumes the execution of a sequence of tiles successive along a particular dimension, as in [3]. We choose to assign tiles along the second dimension x to the same process in order to simplify code development. Thus, a first approach would be to distribute the t dimension of our algorithm to P processes by setting $c_t = T/P$ and assign one chain of tiles along the x dimension to each process (see Figure 3(a) for $P = 4$). However, in several cases this approach prohibits maximum parallelism since, due to the skewed shape of the iteration space, processes at the end of the process chain may start their execution close to (or even after) the completion of tile execution of processes at the head of the process chain. This reduction in process concurrency is also depicted on the right of Figure 3(a), where out of 13 parallel execution times, only one ($t = 6$) utilizes all four available processes.

An alternative approach that copes with the above problem, is to divide the t dimension into K slices and map cyclically each slice to the P processes. Figure 3(b) displays the alternative mapping for $K = 2$. Process concurrency has been significantly improved. In order to determine efficient tiling parameters under this scheme, we should approximate the execution and communication times by a theoretical model and select those parameters that minimize the model's total execution time. However, accurate theoretical models for such parallel executions are extremely difficult to formulate since computation and communication times are affected by a large number of factors which may be difficult to determine. In our approach, we choose to simplify the tile selection process by determining scheduling-efficient and communication-efficient tiling parameters. Since unsuccessful mapping and scheduling may lead to significant performance degradation due to load imbalances and process idle times, we give the scheduling criterion greater priority. Thus, in order to efficiently schedule the above described cyclic mapping scheme, we first need to ensure that after the completion of the last tile of slice k ($1 \leq k < K$), each process will be able to proceed with the execution of the first tile of slice $k + 1$. Lemma 1 proves that we can meet this demand by a proper selection of c_x.



(a) Mapping of tiles to processes for $c_t = \frac{T}{P}$



(b) Cyclic mapping of tiles to processes for $c_t = \frac{T}{KP}$

Figure 3. Two mapping alternatives

Lemma 1. *If a 2-dimensional iteration space $T \times X$, skewed with a skewing factor α and tiled by a rectangular tiling transformation $H^{-1} = \begin{bmatrix} c_t & 0 \\ 0 & c_x \end{bmatrix}$, is cyclically mapped to P processes and scheduled with scheduling vector $\Pi = (1,1)$, then each process can execute all assigned tiles without stalling as long as $c_x < \frac{X - \alpha P c_t}{P-2}$.*

Proof. In order to allow the uninterrupted, stall-free execution of tiles in each process, we need to ensure that after the execution of the last tile of slice k, all dependencies of the first tile of slice $k+1$ are satisfied. In this way, processes proceed their execution between slices in successive parallel time steps. Without loss of generality, we consider the first process in the first slice ($k = 0$). The last point $\vec{j_1}$ of this process in slice 0 has coordinates $\vec{j} = (c_t - 1, X + \alpha c_t)$, will lie in tile $\vec{j_1}^S = (0, \lfloor \frac{X+\alpha c_t}{c_x} \rfloor)$ and will be scheduled at parallel time step $T_1 = \Pi \vec{j_1}^S = \lfloor \frac{X+\alpha c_t}{c_x} \rfloor$. The only intra-process dependence of the first tile of process p_0 at slice with $k = 1$ is created by the first tile $\vec{j_2}^S$ of process p_{P-1} at slice with $k = 0$. One point within this tile is clearly $\vec{j_2} = ((P-1)c_t, \alpha(P-1)c_t)$, thus $\vec{j_2}^S = (P-1, \lfloor \frac{\alpha(P-1)c_t}{c_x} \rfloor)$ and $T_2 = \Pi t_2 = P - 1 + \lfloor \frac{\alpha(P-1)c_t}{c_x} \rfloor$. Hence, in order for the dependence to be satisfied it must hold $T_2 \leq T_1 \Rightarrow P - 1 + \lfloor \frac{\alpha(P-1)c_t}{c_x} \rfloor \leq \lfloor \frac{X+\alpha c_t}{c_x} \rfloor \Rightarrow P - 2 + \frac{\alpha(P-1)c_t}{c_x} < P - 1 + \lfloor \frac{\alpha(P-1)c_t}{c_x} \rfloor \leq \lfloor \frac{X+\alpha c_t}{c_x} \rfloor \leq \frac{X+\alpha c_t}{c_x} \Rightarrow c_x < \frac{X - \alpha P c_t}{P-2}$. □

In tiling transformations there is a tradeoff between communication overhead and process concurrency. Tiling sacrifices concurrency to reduce the communication volume and the total number of messages. By fine-tuning the tile size and shape (parameters c_t and c_x in our case) one needs to minimize communication overhead without drastically increasing the idle times of processes. The following definition provides a metric for process concurrency that will be used to formulate an optimization problem for the selection of scheduling-efficient tiling parameters (Definition 2).

Definition 1. *The concurrency factor (*cf*) is defined as the ratio of the parallel time steps during which not all processes participate in the execution (non-concurrent parallel time steps) to the total number of parallel time steps, thus*

$$\mathrm{cf} = \frac{\textit{Number of non-concurrent parallel time steps}}{\textit{Total number of parallel time steps}}$$

Lemma 2. *In the family of problems described in Lemma 1 the concurrency factor (*cf*) is*

$$\mathrm{cf} = \frac{2P + \lfloor \frac{2\alpha(P-1)c_t}{c_x} \rfloor}{KP + \lfloor \frac{\alpha(KP-1)c_t + X)}{c_x} \rfloor}$$

Proof. Since the last point in the skewed iteration space has coordinates $\vec{j_1} = ((KP-1)c_t, \alpha(KP-1)c_t + X)$, the last tile $\vec{j_1}^S$ will have coordinates $\vec{j_1}^S = (KP-1, \lfloor \frac{\alpha(KP-1)c_t+X)}{c_x} \rfloor)$ and thus will be scheduled at parallel time step $\Pi \vec{j_1}^S = KP - 1 + \lfloor \frac{\alpha(KP-1)c_t+X)}{c_x} \rfloor$. Thus, the total number of parallel time steps is $KP + \lfloor \frac{\alpha(KP-1)c_t+X)}{c_x} \rfloor$ since the first tile is executed at time step 0. The non-concurrent parallel time steps occur at the beginning of the execution until the last process starts its execution, and at the end of the execution after the first process ends its execution. These two cases clearly have equal duration. As shown in the proof of Lemma 1, the first tile of the last process starts its execution at parallel time step $P - 1 + \lfloor \frac{\alpha(P-1)c_t}{c_x} \rfloor$. Thus, the total number of non-concurrent parallel time steps is $2P + \lfloor \frac{2\alpha(P-1)c_t}{c_x} \rfloor$. From Definition 1 we have $\mathrm{cf} = \frac{2P + \lfloor \frac{2\alpha(P-1)c_t}{c_x} \rfloor}{KP + \lfloor \frac{\alpha(KP-1)c_t+X)}{c_x} \rfloor}$. □

Definition 2. *Optimization problem*
Select c_t, c_x: $\mathrm{cf}_{\min} \leq \mathrm{cf} \leq \mathrm{cf}_{\max} \wedge 1 \leq c_x < \frac{X - \alpha P c_t}{P-2}$, $c_x \in \mathbb{Z} \wedge 1 \leq K \leq \frac{T}{P}$, $K \in \mathbb{Z} \wedge KPc_t = T$.

The second requirement is taken from Lemma 1. Factors $\mathrm{cf}_{\min}$ and $\mathrm{cf}_{\max}$ are provided by the user and depend on both the features of the algorithm and the underlying architecture. In this way, we have reduced the problem of finding an efficient tiling transformation to the problem of providing appropriate concurrency factors to the optimization problem of Definition 2. However, the second problem is much more comprehensible, since it is clear that intuitive selections (e.g. $\mathrm{cf}_{\min} = 0.15$ and $\mathrm{cf}_{\max} = 0.2$) can lead to very efficient selections, as will be shown in Section 5.

After selecting a set of scheduling-efficient tiling transformations (pairs of c_t and c_x) we employ communication criterions to select communication-efficient members of the above set. Such criterions are the minimization of the total number of messages and the minimization of the overall communication volume. Lemma 3 quantifies the above metrics for the problems under consideration.

Lemma 3. *In the family of problems described in Lemma 1, the total number of messages is* $N_{comm} = (KP-1)\lceil \frac{X}{c_x} \rceil$ *and the communication volume is* $V_{comm} = (KP-1)Xmax(d'_1)$.

Proof. Clearly, the number of surfaces that impose communication between processes is $KP-1$ with length X. Thus, the number of messages per surface is $\lceil \frac{X}{c_x} \rceil$ and the total number of messages $N_{comm} = (KP-1)\lceil \frac{X}{c_x} \rceil$. Similarly, each surface accounts for communication data that equal the product of the length surface with the maximum first coordinate of the dependencies in D', i.e., $max(d'_1)$. Thus, the overall communication volume is $V_{comm} = (KP-1)Xmax(d'_1)$. □

Summarizing, in order to select an efficient tiling transformation determined by c_t and c_x, we first solve the optimization problem defined in Definition 2 and obtain a set of scheduling-efficient pairs of c_t, c_x. In the sequel, we investigate the efficiency of each pair in terms of communication using Lemma 3 and result to the final scheduling and communication-efficient tiling transformations.

Step 4: Local Data Spaces and Communication Sets
On a message-passing platform, each process allocates its local data space and exchanges communication data with neighboring ones. These data spaces are used to hold the data computed in the chains of the tiles assigned to the process, as well as the data received by the neighbors. Communication data are determined by the dependencies. A communication set $CS(k, p_{id})$ contains all iteration points that need to be sent to the neighboring process, where k is the current slice and p_{id} the serial number of the executing process. The communication set is given in Definition 3.

Definition 3. *The communication set* $CS(k, p_{id})$ *is defined as* $CS(k, p_{id}) : \vec{j'} = (t', x') : \vec{j'} \in J' \wedge \exists \vec{d'} = (d'_1, d'_2) \in D' : t' + d'_1 > (kP + p_{id} + 1)c_t \wedge \vec{j'} + \vec{d'} \in J'$

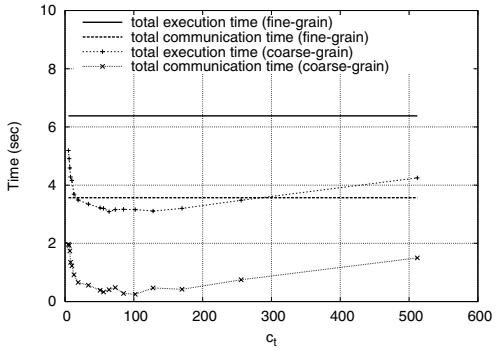
Algorithm 5: pseudo-code for coarse-grain parallel execution of thermal diffusion

```
1  for (k = 0; k < K; k++) do
2      t_0 = (kP + p_id)c_t;
3      T_start = (αt_0/c_x)c_x;
4      T_stop = α(t_0 + c_t) + X;
5      for (Tile = T_start; Tile < T_stop; Tile+=c_x) do
6          if valid_left_tile(p_id, k,Tile) then
7              MPI_Recv(rbuf, c_x, MPI_DOUBLE, prev, 0,
               MPI_COMM_WORLD, &status);
8              unpack(rbuf, Ul);
9          for (t' = 1; t' < c_t and t_0 + t' < T; t'++) do
10             for (x' =max(Tile, α(t_0 + t') + 1);
               x' <min(Tile + c_x, α(t_0 + t') + X − 1);
               x'++) do
11                 t = t';
12                 x = x' − t_0 − t';
13                 Ul[k][t + 1][x] = (1 − 2θr)Ul[t][x] +
                   θr(Ul[t][x−1]+Ul[t][x+1])+ΔtF(t, x);
14         if valid_right_tile(p_id, k,Tile) then
15             pack(Ul, sbuf);
16             MPI_Send(sbuf, c_x, MPI_DOUBLE, next, 0,
               MPI_COMM_WORLD);
```

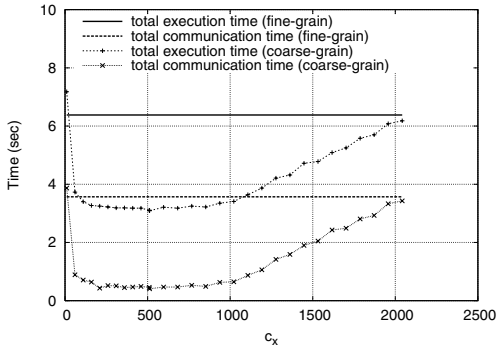
Step 5: Final SPMD Message-Passing Code
Summarizing, the coarse-grain parallel code for the thermal diffusion example is shown in Algorithm 5. K (line 1) is the number of slices. Functions `valid_left_tile` (line 6) and `valid_right_tile` (line 14) check whether there is a valid tile containing at least one iteration point in the tiled iteration space on the left and on the right of the current tile, respectively. Function `unpack` (line 8) copies the received data from the communication buffer $rbuf$ to the local array Ul and function `pack` (line 15) copies the communication data from Ul into the communication buffer $sbuf$. Both functions have trivial implementations based on the definition of the communication sets CS.

5 Experimental Evaluation

In order to evaluate the performance of the fine-grain parallel execution against the proposed coarse-grain alternative, we measured the total execution time for the thermal diffusion algorithm, which is a typical representative of this application family, in five different domains. In addition, we need to evaluate the efficiency of our proposed approach concerning the selection of the tiling parameters. Our experimental platform is a 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 2.4.29. We used MPICH v. 1.2.6 MPI implementation, configured with the Intel C++ compiler v. 8.1.

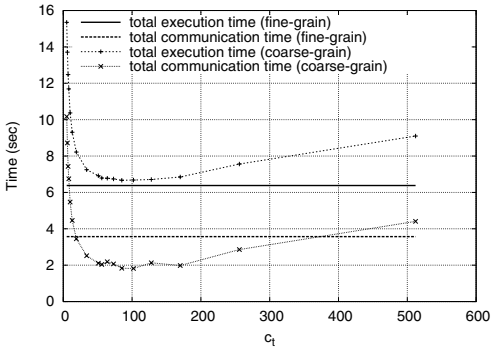


(a) parallel exec. time as a function of c_t ($c_x = 512$).

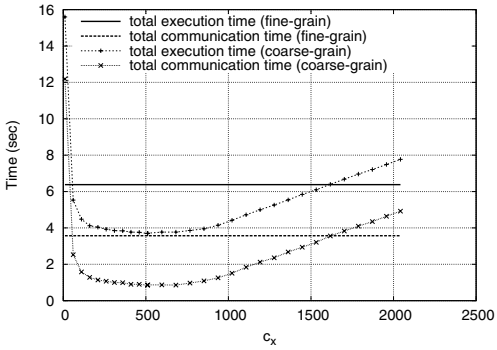


(b) parallel exec. time as a function of c_x ($c_t = 64$).

Figure 4. Iteration space $16K \times 16K$.



(a) parallel exec. time as a function of c_t (avg(c_x)).



(b) parallel exec. time as a function of c_x (avg(c_t)).

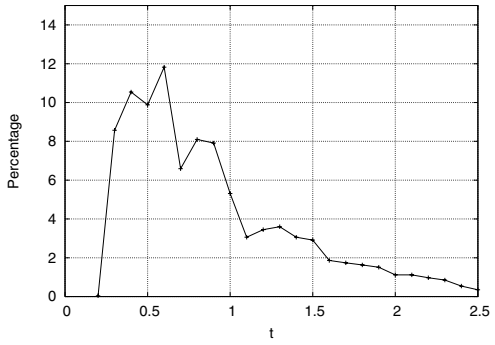
Figure 5. Iteration space $16K \times 16K$.

Figure 4 shows the total execution time of the fine-grain and the coarse-grain execution of our benchmark as a function of c_t for $c_x = 512$ (Figure 4(a)) and as a function of c_x for $c_t = 64$ (Figure 4(b)), for an iteration space of size $16K \times 16K$. For $c_t = 64$ and $c_x = 512$, the tiled algorithm attained the best performance, since it achieved almost 50% reduction in the execution time. This reduction can be attributed to the minimization of the communication overhead in the parallel execution. In addition, we can also notice that the performance of the tiled algorithm is greatly affected by the selection of the parameters c_t and c_x. This is also predictable, since small values for c_t increase the communication volume and large values of c_t increase process idle times (e.g. if $K = 1$). Similarly, small values for c_x increase the total number of communication messages, while large values for c_x also increase process idle times.

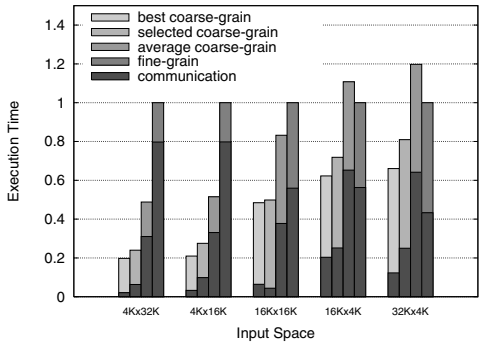
However, the experimental results are not so prominent for all combinations of c_t and c_x. Figure 5 presents the total execution time as a function of c_t (Figure 5(b)) and c_x (Figure 5(a)), where all values have been averaged over various c_x-and-c_t-ranges, respectively. Especially in Figure 5(a), a large number of selections for c_t and c_x lead to larger total execution times than the fine-grain approach. The average parallel execution time of 540 different selections for c_t and c_x is only 16% smaller than the fine-grain case, while 48% of the selections achieve worse performance than that of the fine-grain execution. This fact is summarized in Figure 6(a) for all iteration spaces. In the Y axis we represent the percentage of execution trials and in the X axis we represent the total parallel execution time normalized to the fine-grain execution time. We observe that a large percentage of selections for c_t and c_x perform worse than the fine-grain approach (the area of the plot for $t > 1$). This fact accentuates the need for a method to select efficient tiling parameters for the coarse-grain execution.

Figure 6(b) summarizes the performance across all five iteration spaces of the "best" tiling transformation, the tiling transformation selected according to the method proposed in Section 4 (for $\mathrm{cf}_{\min} = 0.15$ and $\mathrm{cf}_{\max} = 0.2$), the average of all candidate tiling transformations used (500–600 transformations in each iteration space), and the fine-grain approach. The results are normalized to the parallel execution time of the fine-grain case. The selected tiling transformation performs remarkably well, since it achieves a reduction of the parallel execution time compared to fine-grain execution that varies from 22% to 76% depending on the particular iteration space, while the deviation from the best tiling transformation varies from 3% to $23,5\%$.

Concluding, our experimental results confirmed that the application of skewing and tiling transformation to the tar-



(a)



(b)

Figure 6. Summary of results in all five iteration spaces.

get applications can lead to a drastical reduction of the total parallel execution time, due to the minimization of the communication overhead. However, since the parallel execution time is very sensitive to the selection of the tiling parameters, several transformations may even lead to significant degradation of performance. The selected parameters efficiently cope with the above problem, since they attain parallel execution times significantly lower than the fine-grain case and quite close to those achieved by the best tiling transformations.

6 Conclusions

In this paper we presented an alternative approach to execute 2-dimensional iteration spaces that result from the discretization of PDEs. Since the straightforward, fine-grain execution of such applications incurs significant communication overheads in scalable parallel architectures, we applied the skewing transformation to make all dependences nonnegative and in the sequel tiling transformation to implement coarse-grain parallelism. We employed an efficient cyclic scheme to map tiles to processes and provided a method to select scheduling-efficient and communication-efficient tiling transformations. Our experimental results led to the following three conclusions: (a) the coarse-grain approach can indeed lead to impressive reductions in overall parallel execution times of such applications due to the reduction of the communication overhead, (b) the performance of the tiled algorithm is very sensitive to the tiling parameters and (c) the proposed method to select efficient tiling transformation performs remarkably well, since it provides significant reduction in parallel execution time compared to the fine-grain approach, without largely deviating from the performance attained by the best tiling transformations.

References

[1] 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.

[2] G. Goumas, N. Drosinos, M. Athanasaki, and N. Koziris. Message-Passing Code Generation for Non-rectangular Tiling Transformations. *Parallel Computing*, 32(11):711–732, Nov 2006.

[3] 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.

[4] M. Jacobson. *Fundamentals of Atmospheric Modeling*. Cambridge University Press, 1999.

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

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

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

[8] W. Shang and J. Fortes. Time Optimal Linear Schedules for Algorithms with Uniform Dependencies. *IEEE Trans. Comput.*, 40(6):723–742, 1991.

[9] 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.