Global Tiling for Communication Minimal Parallelization on Distributed Memory Systems

Lei Liu[1,2], Li Chen[1], ChengYong Wu[1], and Xiao-bing Feng[1]

[1] Key Laboratory of Computer Architecture Institute of Computing Technology, Chinese Academy of Sciences, 100080 Beijing, China
[2] Graduate School of the Chinese Academy of Sciences, 100080 Beijing, China
{liulei,lchen,cwu,fxb}@ict.ac.cn

Abstract. Most previous studies on tiling focus on the division of iteration space. However, on distributed memory parallel systems, the decomposition of computation and the distribution of data must be handled at the same time, in order to attain load balancing and to minimize data migration. In this paper, we formulate a 0-1 integer linear programming for the problem of globally optimal tiling to minimize the total execution time. To simplify the selection of tiling parameters, we restrict the tile shape to semi-oblique shape, and present two effective approaches to decide the tile shape for multi-dimensional semi-oblique shaped tiling. Besides, we present a tile-to-processor mapping scheme based on hyperplanes, which can express diverse parallelism and gain better performance than traditional methods. The experimentations with NPB2.3-serial SP and LU on Qsnet connected cluster achieved the average parallel efficiency of 87% and 73% respectively.

Keywords: compiler, parallelization, tiling, distributed memory systems, tile-to-processor mapping, semi-oblique shaped tile.

1 Introduction and Related Work

Iteration space tiling is one of the most popular transformations used by compiler and automatic parallelizers for improving locality in multi-level memory hierarchies as well as achieving coarse-grain parallelism. In distributed memory parallel computers, iteration space tiling partitions the loop nest into tiles that may be concurrently executed on different processors with a reduced volume and frequency of inter-processor communication [13, 20].

A lot of discussion has been made concerning the selection of an optimal tiling transformation to minimize communication volume and overall execution time in distributed memory machines. Ramanujam and Sadayappan [17], Schreiber [19], Boulet et al [3], and also Xue [20], discussed the problem of the choice of the tiling parameters to solve the communication-minimal tiling optimally, and proved that the communication-minimal tiling can derive from the algorithm's tiling cone. Hodzic and Shang in [9] proposed a method to correlate optimal tile size and shape, and discussed the selection of scheduling-optimal tile shape and size to minimize execution time. Hogstedt et al.[10] studied the idle time associated with parallelepiped tiling.

E. Luque, T. Margalef, and D. Benítez (Eds.): Euro-Par 2008, LNCS 5168, pp. 382–391, 2008.

They defined the rise as a critical parameter to characterize the time processors wait for data from other processors, and gave analytic formulae for the idle time of the program, and also determined the optimal schedule using linear programming.

Despite all these methods for the selection of tiling parameters, general parallelepiped tiling is not applied by commercial and research compilers due to the complexity of implementation. In order to make the tiling optimal problem tractable and without too much performance fluctuating, we restrict the tile shape to semi-oblique, which are hyper-parallelograms with at least one side perpendicular to the coordinate axes.

Andonov et al. [2] addressed the problem of optimal semi-oblique tiling, and determined the optimal tile size, optimal number of processors and also the optimal slope of the oblique tile boundary, based on the BSP model. However, in their paper, the iteration space tiling was restricted to a two-dimensional semi-oblique shaped tile, which is a special case of our multi-dimensional semi-oblique shape tiling.

Previous work, however, addressed only tiling the iteration space for a single nested Do-loop, and did not consider the problem of data layout. But the problem of data layout is a crucial factor for gaining performance on distributed memory systems. If data and computation are not properly partitioned across processors, it may cause heavy inter-processor communication. Such excessive communication will offset the benefit of parallelization, even if each loop nest has selected optimal iteration space tiling. Similar to the iteration space tiling, we introduce the notion of data space tiling to describe the problem of data layout, and synthesize the iteration space tiling and data space tiling to minimize the inter-processor communication.

Traditional program decomposition frameworks use exhaustive search methods for guiding data and computation partitioning [14]. However, in order not to increase the search space, only row-wise and column-wise or other restricted data distribution schemes can be considered in their methods.

Griebl [7] presented an integrated framework for optimizing data placement and parallelism in the use of tiling. However, mainly due to tiling being modeled as a post-processing after space-time mapping, they can not guarantee optimal tiling and minimal remapping communication for the whole program. In technical report [18], the authors developed a tiling approach that can optimize parallelism and locality simultaneously, and used loop fusion to enhance the data locality between loop nests so as to reduce communication between loop nests. However, loop fusion may impact the parallelism, and their framework can not balance between the parallelism gain and benefit from data locality.

In this paper, we present some approaches to calculate multi-dimensional semi-oblique shaped tiling with polynomial time complexity, and develop a unified framework that can be used for computation partitioning and data layout optimization. The central contribution of this paper is a method called global tiling to select the optimal iteration space tiling and data space tiling, in order to improve parallelism as well as to minimize the volume of communication between loop nests for the whole program.

The rest of this paper is organized as follows. Section 2 covers the background of tiling and notation and assumption. In section 3, we describe the tile space selection algorithms and tile-to-processor mapping strategy. Section 4 formulates the 0-1 ILP for global tiling optimization problem. The experimentations with SP and LU applications are illustrated in the section 5, and conclusions are presented in Section 6.

2 Nomenclature and Definitions

2.1 Basic Definitions and Assumptions

The iteration space, denoted ***J***, is the set of all loop iterations bounded by loop limits, which can be viewed as a polytope constrained by loop bounds. Constrained by array bounds, a data space ***T*** can also be viewed as a set of polytopes in the same way. The relations between iteration spaces and data spaces can be built via array reference functions. An array reference function is a transformation from iteration space ***J*** into data space ***T***, or more exactly, from the iteration vectors to the array elements. We assume that the reference functions are affine. Under this assumption, we can write a reference to an array A as $f(\vec{j})=F^A\vec{j}+\alpha^A$, where F^A is a linear transformation matrix called access matrix and α^A is the offset vector.

In this paper, the programs considered are limited to a series of perfected nested loops with uniform dependencies, i.e., dependencies that can be described by a finite number of constant dependence distance vectors.

2.2 Iteration Space and Data Space Tiling Transformation

In k-dimensional space, a hyperplane can be defined as a set of tuples (a_1, a_2, …, a_k), such that $g_1a_1+g_2a_2+\ldots+g_ka_k=q$, where g_1, g_2 …, g_k are rational numbers called hyperplane coefficients. A hyperplane vector (g_1, g_2,…, g_k) defines a hyperplane family where each member hyperplane has the same hyperplane vector but different q.

Tiling transformation is defined as k independent families of parallel equidistant hyperplanes that partition the iteration index space into k-dimensional parallelepiped tiles. A square matrix H consists of the k normal vectors as rows, which is a vector orthogonal to the hyperplanes. Similar to matrix H, matrix P contains the side-vectors of a tile as column vector, and $P=H^{-1}$. The tiling hyperplanes defined by matrix H satisfy $HD\geq 0$ (D is the data dependence matrix), i.e., each entry in the product matrix is greater than or equal to zero, which is a sufficient condition for the legal tiling transformation [13, 17, 20].

The entire data space can also be tiled in the similar way that the iteration space is tiled. Therefore, we introduce the notion of data space tiling that defines the data elements accessed by an iteration space tile. An *m-by-m* nonsingular matrix M is defined as the data tile matrix, and each row of M is a vector perpendicular to a tile boundary. As in the iteration space tiling case, each column of M^{-1} gives the direction vector for a tile boundary. Assuming that F^A is the access matrix of Array A in loop nests x, we have $M=F^A\times H$, which define the relationship between iteration space tiling and data space tiling.

3 Tile Shape Selection and Tile-to-Processor Mapping

3.1 Semi-oblique Shaped Tile

We use the same way as [8] to factorize the tiling transformation matrix. Defining $P=P_uL$, where P_u is a matrix with unit determinant and column vectors in the directions of the corresponding column vectors of the matrix P, L is a $k\times k$ diagonal matrix which

defines the size of tile. $H_u=P_u^{-1}$ is a unit determinant with rows in the same direction as rows of matrix H. Matrix H_u and matrix P_u do not include any information about the size of the tile. We call H_u and P_u the tile shape matrix.

The tile parameters selection is a hard, nonlinear optimization problem, and is currently no good solution. So we restrict the tile shape to semi-oblique ones to simplify the tiling parameters selection problem. The semi-oblique shaped tiles are hyper-parallelograms with at least one side that are perpendicular to the coordinate axes. That is, the iteration tiling matrix P has at least one row vector which has exact one non-zero element.

We assume that the data dependence matrix D is a $k\times m$ matrix, and define: $D=\begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,m} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ d_{k,1} & d_{k,2} & \cdots & d_{k,m} \end{bmatrix}$, a binary operator: $x \div y = \begin{cases} \infty & ,if\ x \neq 0 \wedge y = 0 \\ 1 & ,if\ x = 0 \wedge y = 0 \\ 0 & ,if\ y = \infty \\ \frac{x}{y} & otherwise \end{cases}$, and function: $Min(x,\infty) = x$, $Max(x,\infty) = \infty$. The following algorithms show how to construct a legal semi-oblique tile shape matrix.

Algorithm 1. For data dependence matrix D, a lower triangular matrix P_u is constructed as $P_u=\begin{bmatrix} 1 & 0 & \cdots & 0 \\ p_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ p_{k,1} & p_{k,2} & \cdots & 1 \end{bmatrix}$. $p_{i,j} = \begin{cases} Min_{l=1}^{m}(d_{i,l} \div d_{1,l}) & if\ j = 1 \\ Min_{l=1}^{m}(q_{i,j}^{l}) & if\ j \geq 2 \end{cases}$, where i>j, and $q_{i,j}^{l} = \begin{cases} \dfrac{d_{i,l}}{d_{j,l}}, & if\ d_{i,l} < 0 \wedge d_{j,l} \neq 0 \\ & \wedge d_{j-1,l} = \ldots = d_{1,l} = 0 \\ 0, & otherwise \end{cases}$

Theorem 1. Given a Dependence matrix D for loop nest x, the semi-oblique shaped tiling matrix P_u by *Algorithm 1* is a legal loop transformation.

The proof of *Theorem 1* is too long, so we omit this proof due to the space constraints.

When $D\geq 0$, in order to select a better scheduling tile shape, *algorithm 1* can be improved as following:

Algorithm 2. Assuming data dependence matrix D, we define a (k-1)×k matrix $P^{*}=\begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$. A legal tiling shape matrix $P_u = \begin{bmatrix} \vec{P}_1 \\ P^{*} \end{bmatrix}$ can be constructed, where: $\vec{P}_1 = \begin{bmatrix} 1 & 1 \div (Max_{j=1}^{m}((\sum_{i=2}^{k} d_{i,j}) \div d_{1,j})) & \cdots & 1 \div (Max_{j=1}^{m}((\sum_{i=2}^{k} d_{i,j}) \div d_{1,j})) \end{bmatrix}$.

Theorem 2. If dependence matrix $D \geq 0$ which means that every entry of D is non-negative, and the tile shape matrix P_u by *Algorithm 2* can guarantee a legal tiling transformation.

Proof: Define matrix $Q=H_uD$, for $H_u= P_u^{-1}$, the element of the first row vector in matrix Q is $Q_{1,j} = d_{1,j} - (\sum_{i=2}^{k} d_{i,j}) \div (Max_{j=1}^{m}((\sum_{i=2}^{k} d_{i,j}) \div d_{1,j}))$, the other rows vector in matrix Q is equal to the corresponding rows in matrix D. If $d_{1,j} \neq 0$, then $Q_{1,j} = d_{1,j} \times (1 - (\sum_{i=2}^{k} d_{i,j}) / Max_{j=1}^{m}(\sum_{i=2}^{k} d_{i,j}))$, $\because (\sum_{i=2}^{k} d_{i,j}) / Max_{j=1}^{m}(\sum_{i=2}^{k} d_{i,j}) \leq 1 \wedge d_{1,j} > 0$, $\therefore \Rightarrow hd_{1,j} \geq 0$; else $d_{1,j} = 0$, so that $(\sum_{i=2}^{k} d_{i,j}) \div d_{1,j} = \infty$, $\therefore hd_{1,j} = 0 - (\sum_{i=2}^{k} d_{i,j}) / \infty = 0$. Because $D \geq 0$, therefore, the elements of Q are non-negative, that is, $H_uD \geq 0$, P_u is a legal tiling transformation. ■

3.2 Tile-to-Processor Mapping

On distributed memory systems, we need to allocate both the iteration tiles and data tiles to processors. For k dimensional tiling transformation in loop nest x and k' dimensional processor grid S and $k' < k$, the iteration tile-to-processor mapping is defined as: $\vec{s} = G_\theta \times \vec{j}_t$, where G_θ, called iteration tile mapping matrix (mapping matrix for short), is a $k' \times k$ matrix with each row is a hyperplane vector g, $\vec{j}_t$ are the coordinates of tile, and $\vec{s}$ are the coordinates of processor space S. The mapping matrix G_θ also means k' degrees of parallelism in loop nest x.

Taking the semi-oblique shaped tile into account, we only assign the tiles along the hyperplanes that are perpendicular to the coordinate axis of iteration space. The following *algorithm 3* shows how to construct mapping matrix G_θ, giving the tile shape matrix P_u and data dependence D.

Algorithm 3. The tile shape matrix P_u is a $k \times k$ nonsingular matrix, for the data dependence matrix D in loop nest x. If P_u is not a unit matrix, the tile-to-processor mapping matrix G_θ is composed of the row vectors that have exact one non-zero element in P_u. Else if P_u is a unit matrix and the row rank of D is not one, then there are k different mapping matrixes, which are equal to a sub-matrix with k-1 rows of P_u. Otherwise, P_u is a unit matrix and the row rank of dependence matrix D is 1, besides the k different mapping matrix distributing the tile along the coordinate axis of iterations space, another mapping strategy called diagonal tile mapping is produced. Diagonal tile mapping arranges the tile mapping to processors along wrapped diagonals through the axis of iteration space.

For example, considering a two dimensional iteration space, the data dependence matrix $D = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, thus a semi-oblique shaped tiling $P_u = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ is constructed by *algorithm 2*. Because the row rank of D is 1, there are three different tile mapping schemes according to the *algorithm 3*: in Fig1.a, the mapping matrix $G_\theta = [1 \quad 0]$

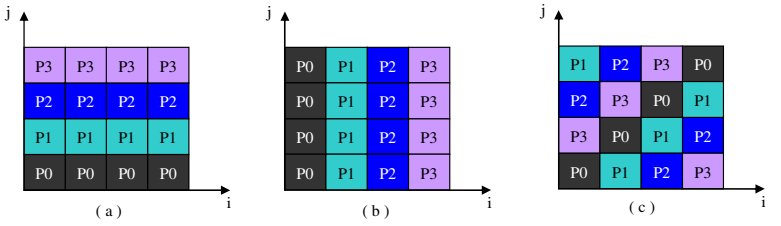

Fig. 1. Three kinds of tile-to-processor mapping strategies for two dimensional tiling

means of allocating the tiles along the axis i in iteration space; in Fig1.b, the mapping matrix $G_\theta = [0 \quad 1]$ means to allocate the tiles along the axis j, and in Fig1.c, a tile-to-processor mapping $G_\theta = [1 \quad -1]$ means to assign the tiles along the diagonal.

As for the data tile mapping to processors is considered, for array reference B, F^B is the array access matrix. If F^B is a non-singular matrix, then the data tile mapping matrix is equal to $G_\theta \times (F^B)^{-1}$, where G_θ is the mapping matrix for iteration space tiling. Otherwise we assume that the array is replicated on processors, denoted as $G_\theta = \perp$.

In the case of data tiling mapping strategies, it is possible that two different data tile-to-processor mapping strategies are conflicting, that is, the data tiles are allocated to processors by two ways. If both of the array access matrix F^{B1} and F^{B2} are nonsingular matrixes and $G_\theta^1 \times (F^{B_1})^{-1} = G_\theta^2 \times (F^{B_2})^{-1}$, or F^{B1} and F^{B2} are singular matrixes and $G_\theta{}^1 = G_\theta{}^2 = \perp$, then the data tile mapping strategies for array reference B_1 and B_2 are consistent; otherwise, these two data tile mapping strategy are conflicting, where $G_\theta{}^1$ and $G_\theta{}^2$ are the corresponding mapping matrixes.

4 Global Tiling Problem

On distributed memory systems, the iteration space tiling and data space tiling should be considered simultaneously. And the optimization of data space tiling is a global problem, because the layout of an array that is determined by a kind of data tiling and data tile mapping strategy should affect other references of that array in the entire program. Conflicting data tile mapping strategies will produce data remapping communications.

In this section, we concentrate on the global tiling optimization problem: that is, optimizing a number of consecutive loop nests simultaneously; and formulate an integer linear programming (ILP) method to solve this problem.

4.1 Local Tiling Candidates

In previous sections we introduced two methods to select a legal semi-oblique tile shape matrix according to the data dependence matrix given in a loop nest. Then we can get the tiling matrix after selecting the corresponding tile mapping matrix through *algorithm 3* and the tile size according to this tile shape. There are a large body of work on determining optimal tile size [1,2,3,5,8,10,16,20], so we will not address the

tile size problem in this paper. When working on a global setting which comprises a number of loop nests, however, a local suboptimal solution may be globally optimal or vice versa. This implies that it may not be a good idea to consider only local optimal tiling transformation during global optimization process. In other words, we need to consider a number of local alternatives of tiling transformation per nest in the global optimization. The following algorithms show how to achieve all local tiling candidates for a loop nest.

Algorithm 4. For a loop nest x, the data dependence matrix D, if $D \geq 0$, the corresponding tile shape matrix P_u is constructed by *algorithm 2*; otherwise P_u is constructed by *algorithm 1*. If loop interchange transformation is legal, then we get another data dependence matrix D' and define variable v as the loop depth of the original outermost loop in the new interchanged loop nest. Then for each data dependence matrix D' with different v in loop nest x, we can produce a tiling shape matrix P_u' using *algorithm 1* or *algorithm 2*. Let us interchange the v^{th} row vector of the matrix P_u' with the first row vector, then we get a corresponding candidate tiling matrix P_u^v. Choosing the corresponding tile size and tile mapping matrix for each tile shape P_u^v, then we can get all the tiling candidates for loop nest x.

The following example explains how to get all tiling candidates.

Example 1

```
X1:  do i =1 to 100
        do j =1 to 200
           a(i,j)=a(i-1,j-1)+a(i-1,j-2)*b(i,j)
```

For loop nest X_1, the data dependence matrix $D = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}$, and $P_1 = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{bmatrix}$ is the corresponding tiling matrix constructed through *algorithm 2*. If we interchange the loop i with loop j, then we get a new loop nest X_1'. The dependence matrix of X_1' $D' = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$, and tiling matrix $P_2' = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, then $P_2 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ is the corresponding tiling shape matrix to the original loop nest X_1. Therefore, the tiling matrix P_1 and P_2 are two local tiling candidates for loop next X_1.

4.2 Tile Layout Graph

The basic data structure for the formulation of the ILP model is a graph called Tile Layout Graph (TLG). The nodes in TLG correspond to a local iteration space tiling candidates with different tile mapping strategies. Each node denotes the tile mapping matrix G_θ, the tiling matrix P, and the array access matrixes for all array references in this loop nest. We use the notation $V^x[i]$ to denote the i^{th} iteration tiling candidate of a nest loop x in graph TLG, $E^{xx'}[i,i']$ to denote the remapping edge between the i^{th} iteration tiling candidate of a nest loop x and the i'^{th} iteration tiling candidate of a nest loop x′.

The tiling candidate cost in our problem is the estimation of the total execution time for this tiling candidate and tile-to-processor mapping strategy. This execution time cost is given by $Cost(V^x[i])= \zeta(T_{comp}+cT_{comm})$, where ζ is the number of synchronization, c is the number of processors that each processor needs to send to, T_{comp} the execution time of a tile, and T_{comm} the communication time. The parameter c depends on the data tile to processor mapping strategy.

If there are conflicting data tile mapping between the i^{th} tiling candidate in loop nest x, and the i'^{th} candidate in loop nest x′, then we add a remapping edge $E^{xx'}[i,i']$ to this two nodes in TLG. The value of this edge denoted as $Cost(E^{xx'}[i,i'])$, is the sum of cost of data remapping communications for all conflicting array references between these two loop nests.

4.3 0-1 Integer Programming Problem

After tiling candidates generation for each loop nests, we use the 0-1 integer linear programming (ILP) to solve the global tiling problem optimally. $V^x[i]$ and $E^{xx'}[i,i']$ are the 0-1 integer variables, and if the corresponding node or remapping edge are selected they have a value of 1, otherwise their values are 0.

The objective of the global tiling optimization problem now is to select a tiling candidate from each loop nest in a given TLG and communication edges between pairs of nests that are tile-mapping-conflicting,such that:

$$\sum_{x}\sum_{j=1}^{Num(V^x)}(V^x[j]\times Cost'(V^x[j]))+\sum_{x\,x'}\sum_{i=1}^{Num(V^x)}\sum_{j=1}^{Num(V^{x'})}(E^{xx'}[i,j]\times Cost(E^{xx'}[i,j]))$$

is minimized, where $Num(V^x)$ is the number of iteration space tiling candidates in loop nest x, and x, x′ denotes two nests with remapping edges. This objective is constrained by the following conditions:

For a loop nest x, only one tiling candidate should be selected:

$$\sum_{Num(V^x)}V^x[i]=1 \tag{c1}$$

An edge $E^{xx'}[i,i']$ will be selected if and only if both $V^x[i]$ and $V^{x'}[i']$ are selected. This condition can be stated as:

$$\begin{aligned} V^x[i]+V^{x'}[i']-E^{xx'}[i,i']\le 1\\ E^{xx'}[i,i']\le V^x[i]\\ E^{xx'}[i,i']\le V^{x'}[i']. \end{aligned} \tag{c2}$$

5 Experimental Results

We have implemented a prototype parallelizing compiler based on Open64, except for the code generation part, so that we generate parallelized codes manually, for SP and LU benchmark codes developed by NASA Ames (NPB2.3-serial release). Our experiments were performed on a high performance cluster—DeepComp6800, which is a 4-way SMP cluster with 1.3 GHz Itanium2 processors, connected by Qsnet. In our

experiment, we use a noted automatic program decomposition method presented by Kennedy and Kremer [14] to compare with our work.

For the NAS SP class 'B', we compare the parallel efficiency of our global tiling method with two different code versions. One code version is called Doall+Pipelined, which exploits Wavefront parallelism within each sweep in the ADI integration, and the data layout is static. Another code version uses dynamic data remapping communications between different loop nests, and each loop nest can be fully parallelized. The average efficiency of global tiling is 0.87, and that the average efficiency of Doall+Pipelined and doall+Remapping are 0.55 and 0.53 respectively.

And for the experiment with NAS LU class 'B' problem sizes, we compare the parallel efficiency of our global tiling with the method presented by Kremer. The average efficiency of global tiling code version is 0.73, and the average efficiency of the traditional method is 0.62.

6 Conclusion

In this paper, we formulate a 0-1 integer linear programming to solve the global tiling optimization problem, and illustrate how to calculate multi-dimensional semi-oblique shaped tile. Our method differs from previous methods in two aspects: Firstly, our global tiling framework considers the iteration space tiling and data space tiling simultaneously to improve the parallelism and reduce the inter-loop remapping communications. Secondly, our tile mapping scheme can implement different tile distribution strategies, which can yield more efficient parallelism as well as less communication frequency.

References

1. Andonov, R., Rajopadhye, S.: Optimal orthogonal tiling of 2-d iterations. Journal of Parallel and Distributed Computing 45(2), 159–165 (1997)
2. Andonov, R., Balev, S., Rajopadhye, S., Yanev, N.: Optimal semi-oblique tiling. IEEE Trans. Par: & Dist. Sys. 14(9), 944–960 (2003)
3. Boulet, P., Darte, A., Risset, T., Robert, Y. (pen)-ultimate tiling? Intergration. The VLSI journal 17, 33–51 (1994)
4. Desprez, F., Dongarra, J., Rastello, F., Robert, Y.: Determining the idle time of a tiling: new results. In: PACT 1997: Proceedings of the 1997 International Conference on Parallel architectures Compilation Techniques, Washington, DC, USA, p. 307. IEEE Computer Society, Los Alamitos (1997)
5. Desprez, F., Dongarra, J., Rastello, F., Rober, Y.: Determining the idle time of a tiling: new results. Journal of Information Science and Engineering 14, 164–190 (1998)
6. Griebl, M.: On tiling space-time mapped loop nests. In: Proceedings of SPAA 2001, pp. 322–323 (2001)
7. Grieble, M.: Automatic Parallelization of Loop Programs for Distributed Memory Architecture. Univercity of Passau, 2004. Habilitation Thesis (2004)
8. Hodzic, E., Shang, W.: On Supernode Transformation with Minimized Total Running Time. IEEE Trans. Parallel and Distributed Systems 9(5), 417–428 (1998)

9. Hodzic, E., Shang, W.: On time optimal supernode shape. IEEE Trans. On Parallel and Distributed Systems. 13(12), 1220–1233 (2002)
10. Hogstedt, K., Carter, L., Ferrante, J.: Determining the Idle Time of a Tiling. Principles of Programming Languages (January 1997)
11. Hogstedt, K., Carter, L., Ferrante, J.: Selecting Tile Shape for Minial Exectution Time. In: Proc. 11th ACM Symp. Parallel Algorithms and Architectures, pp. 201–211 (June 1999)
12. Högestedt, K., Carter, L., Ferrante, J.: On the parallel execution time of tiled loops. IEEE Trans. Parallel Distrib. Syst. 14(3), 307–321 (2003)
13. Irigoin, F., Troilet, R.: Supernode Partitioning. In: At proc. 15th Ann. ACM Symp. Principles of Programming Languages, pp. 319–329 (1988)
14. Kennedy, K., Kremer, U.: Automatic data layout for distributed memory machines. ACM Trans. Program. Lang. Syst. 20(4), 869–916 (1998)
15. Krishnamoorthy, S., Baskaran, M., Bondhugula, U., Ramanujam, J., Rountev, A., Sadayappan, P.: Effective automatic Parallelization of stencil computations. In: PLDI 2007, pp. 235–244 (2007)
16. Ohta, H., Saito, Y., Kainaga, M., Ono, H.: Optimal tile size adjustment in compiling general doacross loop nests. In: ICS 1995: Proceedings of the 9th international conference on Supercomputing, pp. 270–279. ACM Press, New York (1995)
17. Ramanujam, J., Sadayappan, P.: Tiling Multidimensional Iteration Spaces for Non Shared-Memory Machines. Supercomputing, 111–120 (1991)
18. Bondhuagula, U., Baskaran, M., Krishnamoorthy, S., Ramanujam, J., Rountev, A., Sadayappan, P.: Affine Transformations for Communication Minimal Parallelization and Locality Optimization of Arbitrarily Nested Loop Sequences. OSU CSE Technical Report (2007)
19. Schreiber, R., Dongarra, J.: Automatic blocking of nested loops. Technical report, University of Tennessee, Knoxvile. TN (August 1990)
20. Xue, J.: Communication-minimal tiling of uniform dependence loops. J. Parallel Distrib. Comput. 42(1), 42–59 (1997)