Włodzimierz BIELECKI, Marek PAŁKOWSKI

West Pomeranian University of Technology Szczecin

doi:doi:10.15199/48.2015.11.39

On the maximal dimensionality of tiles in tiled code generated by means of Affine Transformations

Abstract. *Tiling(blocking) is a very important iteration reordering transformation for both improving data locality and extracting loop nest parallelism. Affine transformations are one of the most power approach to generate tiled code. Tile dimensionality has a strong impact on tiled code performance. This paper presents a way allowing one to discover before tiling what is the maximal dimensionality of tiles in code generated by means of affine transformations.*

Streszczenie. *Blokowanie jest bardzo ważną transformacją reorganizacji iteracji zarówno dla poprawy lokalności pętli jak i dla ekstrakcji równoległości w gniezdzie pętli programowej. Przekształcenia afiniczne są jednym z najbardziej mocnych podejść do implementacji techniki blokowania. W artykule przedstawiono sposób, za pomocą którego można odkryć przed zastosowaniem blokowania jaki jest maksymalny wymiar bloków w kodzie generowanym za pomocą przekształceń afinicznych, który ma silny wpływ na wydajność kodu.*
(On the maximal dimensionality of tiles in tiled code generated by means of Affine Transformations)

Keywords: optimizing compilers, tiling, affine transformations, code locality, parallelism
Słowa kluczowe: kompilatory optymalizujące, transformacje afiniczne, lokalność kodu, równoległość

Introduction

Tiling [1, 2, 3, 4, 6] is a very important iteration reordering transformation for both improving data locality and extracting loop nest parallelism.

One of the most power approaches to generate tiled code is affine transformations of program loops [1, 2, 3, 4, 6, 7, 8, 9, 10]. However, this approach does not guarantee generation of tiled code. Tiling is valid when there exist bands of fully permutable loops [7, 9].

Papers [1, 2, 7, 8, 9] are a seminal work presenting the theory of tiling techniques based on affine transformations. These publications present techniques consisting of the two steps: the first one transforms an original loop nest into a fully permutable loop nest, the second one converts the fully permutable loop nest into tiled code. A loop nest is fully permutable if its loops can be permuted arbitrarily without altering the semantics of the original program. If a loop nest is fully permutable, it is sufficient to apply a tiling transformation to this loop nest [7]. Many papers demonstrate how to apply interchange, reversal, skewing to get a fully permutable loop nest.

The dimensionality of tiles in tiled code has a strong influence on code performance: in general, the higher tile dimensionality is, the more performance of tiled code is. There is the following question: given a loop nest, can it be tiled or not; if so, what is the maximal dimensionality of tiles in tiled code? Well-known techniques allow us to answer this question by means of forming time-partition constraints for a given loop nest and next finding the maximal number of linearly independent solutions to those constraints. In general, this procedure is time-consuming.

The goal of this paper is to present a simple way allowing us to answer what is the possible maximal dimensionality of tiles without constructing time-partition constraints, i.e., before applying a technique to form affine transformations. We formulate a theorem allowing us to discover what is the maximal dimensionality of tiles in tiled code on the basis of dependence distance vectors. The application of the theorem is illustrated by means of several real benchmarks.

Background

In this paper, we deal with affine loop nests where, for given loop indices, lower and upper bounds as well as array subscripts and conditionals are affine functions of surrounding loop indices and possibly of structure parameters (defining loop index bounds), and the loop steps are known constants.

A dependence analysis is required to carry out a valid loop transformation. Two statement instances *I* and *J* are dependent if both access the same memory location and if at least one access is a write. *I* and *J* are called the source and target of a dependence, respectively, provided that *I* is lexicographically less than *J* ($I \prec J$, i.e., *I* is executed before *J*). Different forms of dependence representations are used in optimizing compiler techniques. In this paper, we use dependence relations and distance vectors.

A dependence relation is a tuple relation of the form [*input list*]→[*output list*]: *formula*, where *input list* and *output list* are the lists of variables and/or expressions used to describe input and output tuples, and *formula* describes the constraints imposed upon input and output lists and it is a Presburger formula built of constraints represented by algebraic expressions and using logical and existential operators [14].

Suppose that there is a dependence from statement *S1* on iteration *i* of a loop nest of depth *d* and statement *S2* on iteration *j*; then the dependence distance vector *d(i,j)* is defined as a vector of length *d* such that its *k-th* component is formed as $d(i,j)_k = j_k - i_k$.

A loop nest is uniform if all components of each dependence distance vector are constants, otherwise it is non-uniform. If some component of a dependence distance vector is non-constant, then its range of integers can be finite or infinite. In the former case, we can split a dependence distance vector into a finite set of simpler vectors whose all components are constants and a corresponding loop nest is still uniform.

For components with an infinite range of integers, we will use the following notations: *n+, m-, m-n+* meaning that a component value belongs to the integer interval [1, *n*], [-*m*, -1], [-*m*, *n*], respectively, *n*>0, *m*>0 are parameters. For example, if dependences are represented with the following dependence relation

```
R:={[i,j] -> [j,j-i] : 1 <= i < j <= n},
```

then vector $d = (j-i, -i)^T$. Taking into account the constraints $1 \leqslant i < j \leqslant n$ of relation *R*, we convert vector *d* to the form $((n-1)+, (n-1)-)^T$, where $(n-1)+$ and $(n-1)-$ denote the upper and lower bounds of the integer range of the first and second components of vector *d*, respectively.

A matrix whose columns are all dependence distance vectors is called a dependence matrix, *D*. For vectors $d1$= (1, n+, m-n+)^T and $d2$=(2,m-n+,-1)^T, matrix *D* is as follows

$$\begin{bmatrix} 1 & 2 \\ n+ & m\text{-}n+ \\ m\text{-}n+ & -1 \end{bmatrix}.$$

In the following section, we present a theorem which allows us to discover on the basis of a dependence matrix what is the maximal tile dimensionality in tiled code generated by means of affine transformations.

Discovering the maximal tile dimensionality

Let us remind that loops are fully permutable if they can be permuted arbitrarily without changing the semantics of the original program [7]. It is also well-known that a *k*-deep fully permutable loop nest can be always tiled in *k*-dimensions [7].

The features of tiled code generated by means of affine transformations are as follows: tile dimensionality is defined by the number of fully permutable loops in a band; the structure of tiles is the same (except from boundary tiles) and defined by affine transformations applied; the number of iterations within a tile is the same (except from boundary tiles) and defined by input constants denoting a tile size.

If an original loop nest is not fully permutable, we can try to convert it to a fully permutable one. With this purpose, we have to set up the time-partition constraints of a loop nest [5, 7, 9] that represent the condition that if one iteration is depend upon the other, then the first must be assigned to a time that is no earlier than that of the second; if they are assigned to the same time, then the first has to be executed after the second.

For a perfect loop nest, the time-partition constraints can be written as follows [2]

$$D^T * C \geqslant \mathbf{0}$$

where *C* is the unknown integer vector, D^T is the transposed dependence matrix for the loop nest, and **0** is the zero vector.

It is well-known that if there exist *k* linearly independent solutions to the time-partition constraints of a loop nest, then it is possible to transform this loop nest to have *k* outermost fully permutable loops [7, 9]. That is the maximal number of linearly independent solutions to the time-partition constraints of a loop nest defines the maximal dimensionality of tiles.

Theorem. Let *D* be a $k \times r$ dependence matrix for a *k*-deep loop nest whose dependences are represented with *r* dependence distance vectors. Then this loop nest can be converted by means of affine transformations to another loop nest with the maximal number of fully permutable loops in a band, *p*, defined by the following formula

$$p = k - s$$

where *s* is the number of the rows of matrix *D* such that for each of them there exists element *m*-*n*+ or there exist at least two elements such that one of them is *n*+ and the other is *m*-.

Proof. Let us rewrite the time-partition constraints of a loop nest

$$D^T * C \geqslant \mathbf{0}$$

as follows

$$\begin{aligned} &c_1 * d_{11} + c_2 * d_{12} + ... + c_k * d_{1k} \geqslant 0 \\ &c_1 * d_{21} + c_2 * d_{22} + ... + c_k * d_{2k} \geqslant 0 \\ &..... \\ &c_1 * d_{r1} + c_2 * d_{r2} + ... + c_k * d_{rk} \geqslant 0 \end{aligned}$$

where $c_1, c_2, ..., c_k$ are the unknown constants; $d_{ij}, 1 \leqslant i \leqslant r, 1 \leqslant j \leqslant k$ are the elements of matrix D^T.

In the constraints above, if some component, say d_{ij}, is equal to *m*-*n*+, then to satisfy inequality *i*, unknown constant c_j has to be 0 because the lower and upper bounds of this component value are the negative and positive symbolic parameters, respectively, so there do not exist any constants $c_1, c_2, ..., c_{j-1}, c_{j+1}, ..., c_k$ allowing for the satisfaction of inequality *i* when c_j is not equal to 0.

If some component, say d_{ij}, is equal to n+ and there exists in the same column of matrix D^T (in the same row of matrix D) another element, say d_{qj}, which is equal to m- then to satisfy both inequalities i and q, c_j has to be 0 because the upper bound of d_{ij} is represented with the positive parameter while the lower bound of d_{qj} is represented with the negative parameter, hence there do not exist any constants $c_1, c_2, ..., c_{j-1}, c_{j+1}, ..., c_k$ allowing for the satisfaction of inequalities i and q simultaneously when c_j is not equal to 0.

So, we may conclude that the numbers of zero and non-zero components in vector *C* are defined by the value of *s* and the value of $p = k - s$, respectively. This means that the dimension of the space where we have to find non-zero solutions to the time-partition constraints is equal to *p*. It is known that the maximal number of linearly independent solutions to integer linear constraints is equal to the dimension of the space they occupy [11], i.e., for the time-partition constraints there exist maximum *p* linearly independent solutions.

Finally we conclude that in a transformed loop nest generated on the basis of linearly independent solutions to the time partition constraints of a loop nest, the maximal number of fully permutable loops in a band equals to *p*. □

Theorem application

Below we illustrate the application of the theorem to the following loop nest.

Example 1.

```
for(i=0; i<=N; i++){
  for(j=0; j<=N; j++){
    b[i][j] = b[i+1][N-j];
  }
}
```

The following relation describes dependences in this loop nest

```
R:={[i,j] -> [i+1,N-j] : 0 <= i < N and
0 <= j <= N}.
```

The corresponding dependence distance vector is of the form $d = (1, N - 2j)^T$, where $0 \leqslant j \leqslant N$.

Hence, we can define vector *d* and matrix *D* as follows $d = (1, N\text{-}N+)^T$,

$$D = \begin{bmatrix} 1 \\ N\text{-}N+ \end{bmatrix}$$

and conclude that k=2, s=1, p=1, i.e., there does not exist any affine transformation allowing for producing a band of fully permutable loops.

Components $n+m$-, n+, and m- can appear in dependence distance vectors in the following cases of loop nests:

i) affine array expressions include two or more loop indices whose lower and/or upper bounds are parameters, for example, as in the expression $i+j$, $0 \leqslant i, j \leqslant n$;
ii) affine array expressions comprise parameters, for example, as in the expression $n - j$;
iii) the dimensions of arrays within a loop nest are less than

the depth of this loop nest, for example, as in the below loop nest

```
for(i=1; i<=n; i++)
  for(j=1; j<=n; j++)
     a[i] = a[i]+ a[i-1]*b[i][j];
```

Let us note that time-partition constraints can be set up for different loop bands of an original loop nest of depth *d*: for all the loops of the nest or for only some d_1 inner loops, d_1 <d, provided that the first outer $d - d_1$ loops are serial. In the latter case, dependence distance vectors have to be extracted under the condition that the first $d - d_1$ outer loop indices are constant parameters, i.e., dependences carried by the first outer $d - d_1$ loops are ignored because they are always respected due to the serial execution of these loops.

We start to discover whether the original loop nest can be transformed to an outermost fully permutable one. For this purpose, we set up time-partition constraints taking into account dependences carried by all the loops of the nest. If there do not exist at least two independent solutions to such constraints, we have to set up new constraints taking into account dependences carried by only some inner loops.

Next, we apply the theorem for two real benchmarks, to discover what is the maximal dimensionality of tiles in tiled code generated by means of affine transformations.

Consider the following loop.

Example 2. Polybench: floyd-warshall, (a graph analysis algorithm to find shortest paths in a weighted graph), [12].

```
for (k = 0; k < N; k++)
{
 for(i = 0; i < N; i++)
  for (j = 0; j < N; j++)
   path[i][j] = path[i][j] < path[i][k] +
   path[k][j] ?
   path[i][j] : path[i][k] + path[k][j];
}
```

Further on for brevity (to skip parameter names), we will use the following notations: + , -, * that mean that a component value belongs to a parametric integer positive interval, negative interval, and both positive and negative intervals, respectively. The dependence matrix for this loop nest is as follows

$$D = \begin{bmatrix} + & 0 & + & 0 & + \\ 0 & 0 & 0 & + & * \\ 0 & + & * & 0 & 0 \end{bmatrix}.$$

There is the component "*" in the two rows of the matrix above, so we may conclude that there does not exist any affine transformation allowing for converting the original loop nest to an outermost fully permutable one by means of affine transformations.

Next, we try to find affine transformations allowing for tiling of the two inner loops of the original nest. For this purpose, we extract dependence distance vectors for the original loop nest provided that *k* is the constant symbolic parameter. For such a case, the dependence matrix is as follows

$$D = \begin{bmatrix} 0 & + \\ + & 0 \end{bmatrix}.$$

Now, the maximal number of linearly independent solutions is equal to 2, hence the two inner loops can be tiled. PLUTO, implementing affine transformations techniques [9], generates the following code with tiles of the 32x32 size, where only the two inner loops are tiled (i.e., there are not any 3d tiles in tiled code).

```
if (N >= 1) {
for (t1=0;t1<=N-1;t1++) {
for (t2=0;t2<=floord(N-1,32);t2++) {
for (t3=0;t3<=floord(N-1,32);t3++) {
for (t4=32*t2;t4<=min(N-1,32*t2+31);t4++) {
for (t5=32*t3;t5<=min(N-1,32*t3+31);t5++) {
path[t4][t5] = path[t4][t5] < path[t4][t1]
 + path[t1][t5] ?
path[t4][t5] : path[t4][t1] + path[t1][t5];
}
}
}
}
}
}
```

Next we analyze the following loop nest.

Example 3. Livermore benchmark: Kernel 6 (general linear recurrence equations), [13].

```
for(l=1; l<=loop; l++){
 for(i=1; i<n; i++){
  for(k=0; k<i; k++){
    w[i] += b[k][i] * w[(i-k)-1];
  }
 }
}
```

For the loop nest above, Petit(the Omega project dependence analyzer) generates the dependence distance vectors presented by the following matrix

$$D = \begin{bmatrix} + & 0 & + & 0 & + \\ * & 0 & 0 & + & * \\ * & + & * & * & * \end{bmatrix}.$$

There is the component "*" in the two rows of the matrix above, so we get k=3, s=2, p=1, i.e., there does not exist any affine transformation allowing for converting the original loop nest to an outermost fully permutable one by means of affine transformations.

Next, we try to find affine transformations allowing for a tiling of the two inner loops of the original loop nest. For this purpose, we extract dependence distance vectors for this nest provided that index l is the constant symbolic parameter. For such a case, the dependence matrix is as follow

$$D = \begin{bmatrix} + & + & 0 \\ * & * & + \end{bmatrix}$$

and we conclude that the two inner loops also cannot be tiled because the maximal number of solutions to the corresponding constraint is equal to 1, i.e., for the considered loop nest, there is not any possibility to generate tiled code by means of affine transformations. Applying PLUTO to this benchmark, we do not get any tiled code.

Summing up, we may conclude that when there exist at least two linearly independent solutions to time partition constraints, affine transformations allow for the generation of tiled code with fixed size tiles, but their dimensionality can be less than the depth of a loop nest. The presented theorem allows us to discover in a very simple way how many linearly independent solutions to time partition constraints exist for a

given loop nest.

Related work

Wolfe [1] discusses a technique referred to as iteration space tiling which divides the iteration space of loop computations into tiles of some size and shape so that traversing the tiles results in covering the whole space. Irigoin and Triolet in [2] proposed supernode partitioning and discussed the constraints on the choice of supernode partitioning hyperplanes; they proved that satisfying those constraints leads to legal tiling. King and Ni [3] discuss the grouping of iterations for execution on multicomputers; they present a number of conditions for valid tile formation in the case of two-dimensional iteration spaces. Wolf and Lam [4] showed that the maximum degree of parallelism can be achieved by transforming the loops into a nest of coarsest fully permutable loop nests and wavefronting the fully permutable nests.

Ramanujam and Sadayappan in [6] showed the equivalence of the problem of finding a cone of dependence vectors and the problem of finding the supernode partitioning hyperplanes and proposed a method of optimizing grain size of tiles for multicomputers. Xue in book[8] provides mathematical foundations, investigates loop permutability in the framework of nonsingular loop transformations.

The “time-partition constraints” introduced in papers [5, 7, 9] represent the condition that if one iteration is depend upon the other, then the first must be assigned to a time that is no earlier than that of the second; if they are assigned to the same time, then the first has to be executed after the second. If there exist more than one linearly independent solutions to the time-partition constraints of a loop nest, then it is possible to apply a tiling transformation to this loop nest [7]; the maximal number of such solutions defines the maximal dimension of tiles in tiled code.

Paper [10] presents a convex characterization of all distinct, semantics-preserving, multidimensional affine transformations; then the authors bring together algebraic, algorithmic, and performance analysis results to optimize tiling. This paper also provides a detailed analysis of related work on tiling.

Summing up, we may conclude that all well-known papers about tiling discuss the conditions of loop nest permutability and tiling legality, suggest how to form "the best time-partition constraints" and generate optimal tiled code. To discover what is the maximal tile dimensionality, one should form time-partition constraints for a given loop nest and next find the maximal number of linearly independent solutions to those constraints.

In this paper, we prosed a simpler way to discover what is the maximal tile dimensionality. We suggest to form a dependence matrix (without forming and resolving time-partition constraints) and next apply the presented theorem to find the maximal number of linearly independent solutions to time-partition constraints for a given loop nest. The theorem takes into account only the content of the dependence matrix. The time complexity of the proposed way to discover the maximal tile dimensionality is much less than that based on forming and resolving time partition constraints.

Conclusion

We presented a simple way allowing us to discover the maximal dimensionality of tiles in tiled code generated by means affine transformations. It does not require constructing time-partition constraints for a given loop nest, i.e., before tiled code generation, we are able to predict whether a loop nest can be tiled and if so, the maximal dimensionality of tiles can be determined on the basis of dependence distance vectors. The application of the suggested way was illustrated by means of two real benchmarks. In our future work, we plan to implement the presented theorem and compare the maximal dimensionality of tiles generated by means of affine transformations with that of tiles generated by means of the transitive closure of dependence graphs [15].

REFERENCES

[1] Wolf, M.: Iteration Space Tiling for Memory Hierarchies, Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing, pp. 357–361, 1987.

[2] Irigoin, F. and Triolet, R.: Supernode partitioning, Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 319–329,1988.

[3] C King, C., Ni, L.: Grouping in nested loops for parallel execution on multicomputers, In Proc. of Int. Conf. on Parallel Processing, 1989.

[4] Wolf, M., Lam, S.: Loop Transformation Theory and an Algorithm to Maximize Parallelism, IEEE Trans. Parallel Distrib. Syst., 2(4), pp. 452–471, 1991.

[5] , Feautrier, P.: Some efficient solutions to the affine scheduling problem: II. Multidimensional time, Int. J. Parallel Program., 21(5), pp. 389–420, 1992.

[6] Ramanujam, J., Sadayappan, P.: Tiling Multidimensional Iteration Spaces for Multicomputer, Proceedings of the 1990 International Conference on Parallel Processing 1992.

[7] Lim, A., and Gerald I. Cheong,G., Lam, M.: An Affine Partitioning Algorithm to Maximize Parallelism and Minimize Communication, In Proceedings of the 13th ACM SIGARCH International Conference on Supercomputing, pp. 228–237, 1999.

[8] Xue, J.: Loop Tiling for Parallelism, Kluwer Academic Publishers, 43, pp. 101–113, 2000.

[9] Bondhugula, U., Hartono, A., Ramanujam, J., Sadayappan, P.: A practical automatic polyhedral parallelizer and locality optimizer, SIGPLAN, 43, pp. 101–113, 2008.

[10] Pouchet, L-N., Bondhugula, U.,Bastoul,C., Cohen A., Ramanujam, J., Sadayappan, P., Vasilache, N.: Loop transformations: convexity, pruning and optimization, Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 43, pp. 549–562, 2011.

[11] Lenstra, H.W.: Integer programming with a fixed number of variables., Mathematics of Operations Research, 8(4), pp. 538–548, 1983.

[12] The Polyhedral Benchmark suite, http://www.cse.ohio-state.edu/ pouchet/software/polybench/, 2012.

[13] "The Livermore Fortran Kernels: A Computer Test Of The Numerical Performance Range", Techn. report, UCRL-53745, 1986.

[14] Pugh W., Wonnacott D.: An exact method for analysis of value-based array data dependences, Sixth Annual Workshop on Programming Languages and Compilers for Parallel Computing, 1983.

[15] Bielecki W., Palkowski M.: Perfectly nested loop tiling transformations based on the transitive closure of the program dependence graph, Soft Computing in Computer and Information Science, 342, 2015.

***Authors**: Prof. Włodzimierz Bielecki, Ph.D. Marek Pałkowski, West Pomeranian University of Technology Szczecin, Faculty of Computer Science, 71-210 Szczecin, ul. Żolnierska 49, email: wbielecki@wi.zut.edu.pl, mpalkowski@wi.zut.edu.pl,*