Finding Free Schedules for Non-uniform Loops

Volodymyr Beletskyy and Krzysztof Siedlecki

Faculty of Computer Science, Technical University of Szczecin, Zolnierska 49 st., 71-210 Szczecin, Poland, fax. (+4891) 487-64-39
{vbeletskyy, ksiedlecki}@wi.ps.pl

Abstract. An algorithm, permitting us to build free schedules for arbitrary nested non–uniform loops, is presented. The operations of each time schedule can be executed as soon as their operands are available. The algorithm requires exact dependence analysis. To describe and implement the algorithm and to carry out experiences, the dependence analysis by Pugh and Wonnacott was chosen where dependences are found in the form of tuple relations. The algorithm can be applied for both non-parameterized and parameterized loops. The algorithm proposed has been implemented and verified by means of the Omega project software.

1 Introduction

The larger number of transformations has been developed to expose parallelism in loops, minimize synchronization, and improve memory locality in the past, for example, [3]-[8], [11]. Most of those transformations permit us to extract parallelism from both uniform and non-uniform loops. But the question is how much loop parallelism these approaches extract.

The general problem of the loop parallelism detection is as follows. Dependences in loops can be represented in different ways, with approximations in general, or with an exact representation when possible. For each dependence representation based on approximations (level of dependences, uniform dependences, polyhedral representation of dependences), optimal algorithms exist, but for exact affine dependences, it is not known what are the loop transformations that extract maximal parallelism [5].

Following Vivien [14], we imply that an algorithm extracting parallelism is optimal if it finds all parallelism: 1) that can be extracted in its framework; 2) that is contained in the representation of the dependences it handles; 3) that is contained in the program to be parallelized (not taking into account neither the dependence representation used nor the transformations allowed).

This paper presents an algorithm permitting us to find free schedules for arbitrary nested non-uniform loops. This algorithm is optimal and find all parallelism in a loop represented by the instances of statements.

Our approach is based on non-linear schedules that can be implemented with procedures using a set of routines for manipulating linear constraints over integer variables, Presburger formulas, integer tuple relations and sets, and the code generation routines for generating code to scan the points in the union of a number of convex polihedra.

2 Background and Definitions

In this section, we in brief attach well-known knowledge to better explain our algorithm for finding free schedules for non-uniform loops.

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

Following work [14], we refer to a particular execution of a statement for a certain iteration of the loops, that surround this statement, as an operation.

Two operations J and I are dependent if both access the same memory location and if at least one access is a write. We refer to I and J as the source and destination of the dependence, respectively, provided that I accesses the same memory location earlier than J $(I \prec J)$.

Definition 1. An affine loop nest is non-uniform if it originates non-uniform dependence relations represented by an affine function f that expresses the dependence sources I in terms of the dependence destinations J (I=f(J)) or the converse.

Definition 2[13]. A dependence anlisys is exact if for any affine dependence it detects a dependence if and only if one exists.

To describe the algorithm and carry out experiences, we chose the dependence analysis proposed by Pugh and Wonnacott [13] where dependences are presented with dependence relations.

Definition 3[13]. A dependence relation is a mapping from one iteration space to another, and is represented by a set of linear constraints on variables that stand for the values of the loop indices at the source and destination of the dependence and the values of the symbolic constants.

Definition 4[5]. A free schedule assigns operations as soon as their operands are available, that is, mapping $\sigma: I \rightarrow Z$ such that

$$\sigma(p) = \begin{cases} 0 \text{ } if \text{ } there \text{ } is \text{ } no \text{ } p' \in I \text{ } s.t. \text{ } p' \prec p \\ 1 + \max(\sigma(p'), p' \in I, p' \prec p) \end{cases} . \tag{1}$$

The free schedule is the "fastest" schedule possible. Its total execution time is

$$T_{free} = 1 + \max(q_{free}(p), p \in I) . \tag{2}$$

The algorithm proposed in this paper is applicable for loops that meet the requirements of the dependence analysis by Pugh and Wonnacott [13].

To follow the material of this paper, the reader should be familiar with the operations on tuple relations such as: union, difference, range, domain, application as well as existentially quantified variables. This knowledge is comprised in [9].

3 Free Schedules for Arbitrary Nested Loops

We will refer to the source of a dependence as the fair dependence source if it is not a destination of any other dependence.

We define the length of a chain of synchronization between a pair of dependent operations as N-1, where N is the maximal number of the operations which this chain connects, that is, it is the maximal number of the direct value based dependences [13] originated by this pair of the dependent operations.

The idea of the algorithm presented in this section is as follows. We divide all operations for each statement into two sets containing the independent and dependent operations (sources and destinations of dependences), respectively. For the second set, we firstly find those operations for which all operands are available. They form the operations of layer Lay[0] to be executed firstly. Next, we eliminate from the second set the operations of Lay[0] and find again those operations for which all operands are available. We repeat this process until there are no operations in the second set. The operations of the first set can be combined with the operations of arbitrary levels.

In imperfectly nested loops, statements may have different domains, and, in general, to find free schedules, we should deal with each statement independently. To generate resulting code, we should find free schedules for all statements independently, and next combine the same layers (including operations belonging to the same schedule time), originated by different statements, into one resulting layer.

Algorithm 1. Find the free schedule for an arbitrary nested loop

1. Find all cross-iteration dependences as well as the dependences originated by statements within the loop body. Build two sets of the dependences for each statement j, j=1,2,...,n. The first one, $S1_j$ includes the dependence relations whose destinations are the instances of statement j. Let the relations of $S1_j$ be $R1_{kj}$, kj=1,2,...,n_j, n_j is the number of the dependence relations in $S1_j$. The second set, $S2_j$ includes the dependence relations whose sources are originated with statement j.

For each statement j do:

2. Find the sources of the dependences as the domains of the relations, belonging to set $S2_j$, and unite them into one set I_j.
3. Find the destinations of the dependences as the ranges of the relations, belonging to set $S1_j$, and unite them into one set J_j.
4. Find the independent statement instances IND_j, that is, those that do not belong to any pair of the dependences as follows $IND_j:=IS_j-I_j-J_j$, where IS_j is the set representing the iteration space of statement j.
5. Find all fair dependence sources FS_j as the difference between I_j and J_j. Sets FS_j form the layer of the operations ($Lay[0]_j$) which should be executed firstly.
6. Find the dependence destinations that are linked with the fair dependence sources by a chain of synchronization of length one or more as follows $L1_{kj}:=R1_{kj}(FS(R1_{kj}))$, kj=1,2,...,$n_j$, where $R1_{kj}$ are found in step 1, $FS(R1_{kj})$ are the sets of the fair dependence sources that are the sources of the dependences represented with relations $R1_{kj}$. Unite sets $L1_{kj}$ into one set $L1_j$.
7. Find the dependence destinations that are linked with the fair dependence sources by a chain of synchronization of length two or more as follows $D1_{kj}:=R1_{kj}(J(R1_{kj}))$, k_j=1,2,...,n_j, where $J(R1_{kj})$ are the sets of the dependence destinations that are the

```
   dependence sources represented with relations $R1_{kj}$. Unite sets $D1_{kj}$ into one set
   $D1_j$.
8. Find the operations belonging to the first layer as $Lay[1]_j:=L1_j-D1_j$.
End for each
9. Find the second and remaining layers of the dependence destinations as follows:
   i=2;
   Loop:
   For each j
      $Jj:=Jj-Lay[i-1]_j$;                    – elimination of the dependence
                                              destinations belonging to layer i-1
                                              and originated by the instances of
                                              statement j;
      $Li_{kj}:=R1_{kj}(Lay[i]_j(R1_{kj}))$,  – finding the dependence destinations
      $kj=1,2,...,n_j$;                       that are linked with the fair
                                              dependence sources by a chain of
                                              synchronization of length i or more,
                                              $Lay[i]_i(R1_{ki})$ are the sets representing
                                              the operations of layer i and are the
                                              sources of the dependences
                                              represented with relations $R1_{kj;}$
      unite sets $Li_{kj}$ into one set $L_{ij}$;
      $Di_{kj}:=R1_{kj}(J(R1_{kj}))$,         – finding the dependence destinations
      $kj=1,2,...,n_j$;                       that are linked with the fair
                                              dependence sources by a chain of
                                              synchronization of length i+1 or
                                              more;
      unite sets $Di_{kj}$ into one set $D_{ij}$;
      $Lay[i]_j:=L_{ij}-D_{ij}$;              – finding the dependence destinations
                                              that are linked with the fair
                                              dependence sources by a chain of
                                              synchronization of length i;
   End for each
   if each $Lay[i]_j$ ==False then the end; else i=i+1; goto Loop;
```

The independent operations INDj can be combined with arbitrary layers. In our implementation, we unite them with the fair dependence sources to build $Lay[0]_j$.

4 Parameterized Loops

The algorithm presented permits us to find free schedules not only for the loops with the known loop bounds before compilation but also for parameterized loops.

Following the algorithm presented, we can try to find the given number m of parameterized layers for a loop. If there are no operations in a certain layer $i \le m$, this means that the loop is characterized by the constant number of layers and each of them can be presented in a parameterized form. The procedure described permits us to reveal whether the number of layers is irrelevant to the size of a loop. If such a loop is detected, code may be generated with a complexity that does not depend on the loop size.

But the main problem that requires further research is how to find the number of layers for parameterized loops in the general case.

If the number of layers does not depend on the size of a loop, we can present the free schedule in a symbolic way and generate corresponding code with a complexity that depends on the loop but not on the volume of the computation it describes.

When there exist only two layers under a free schedule, this means that there are no operations that simultaneously are the sources and destinations of dependences. Such a case can be very easily reviled. We should form the set of the dependence sources and the set of the dependence destinations originated with all loop statements and next find the intersection of these sets. If the result is FALSE, this means that there exist only two layers of operations under the free schedule.

It is worth to note that in the case when a loop exposes the constant number of layers, its free schedule may be much faster than the best affine schedule. Consider the following example [7]:

Table 1. Example of parallelizing the parametrized loop

Sequential code	**Parallel code, our algorithm**
`for (i=0; i≤2*n; i++)`	`par for (i=0;  i≤n-1; i++){`
`  a[i]=a[2*n-i];`	`  a[i]=a[2*n-i];`
	`  a[2*n-i]=a[i]; }`
	`if (n ≥ 1) a[n] = a[2*n-n];`

The best affine schedule for this loop is i/2 [7], that is, the number of the layers yielded with this schedule equals to n. The algorithm presented finds only two layers for this loop. The first and second statements in the parallel code presented in Table 1 originate the fair dependence sources (Lay[0]) and the first layer (Lay[1]), respectively; the condition statement represents the independent operations.

5 Related Work and Conclusion

The approaches presented in [1], [12] build an explicit graph of a subset of the iteration space, with each node representing the instance of a statement. Free schedules can be found by searching the graph, but the problem regarding boundary cases exists. We use tuple relations as an abstraction for data dependences. This permits us to reach all the same merits that discussed in [10]: handle non-uniform dependences and multidimensional loops without having to make special checks in boundary conditions. But, in contrast to work [10], our technique does not require the calculation of the transitive closure of relations.

Concerning parallelism detection, the following facts are known. If the level of dependences is the only available representation, then Allen and Kennedy's algorithm is known to be optimal [2]. For the case of a single statement with uniform dependences, linear scheduling (optimized Lamport's hyperplane method) is asymptotically optimal [4]. For the case of several statements with uniform dependences, the previous result has been extended in work [8] to show that a linear schedule plus shifts leads to finding optimal parallelism. For the case of the polyhedral approximations of dependences, the method described in [3] is optimal. For affine dependences, the most powerful algorithm is Feautrier's one [6]. But as mentioned by Feautrier, it is not optimal for all codes with affine dependences. However, among all possible affine schedules, it is optimal [14]. For affine dependences, the index splitting approach is known [7]. But it is a heuristic procedure, and it is not known how much index splitting is necessary, hence it is not known how many different code structures must be generated to reach optimality.

The technique presented in this paper is a first step to understand the problem of free schedules for loops with affine dependences. It permits us to find free schedules for both non-parameterized and parameterized loops but it does not answer what in general is the number of layers under the free schedule for a parameterized loop.

References

[1] Chen, D.-K.: Compiler optimizations for parallel loops with fine- grained synchronization. Technical report TR-1863, Department of Computer Science, University of Illinois at Urbana-Champaign, (1994)

[2] Darte and Vivien, F.: On the optimality of Allen and Kennedy's algorithm for parallelism extraction in nested loops. Journal of Parallel Algorithms and Applications, Special issue on Optimizing Compilers for Parallel Languages, (1997) 12:83–112

[3] Darte and Vivien, F.: Optimal Fine and Medium Grain Parallelism Detection in Polyhedral Reduced Dependence Graphs. International Journal of Parallel Programming, 25(6), (1997) 447–496

[4] Darte and Khachiyan, L. and Robert, Y.: Linear Scheduling is Nearly Optimal. Parallel Processing Letters, 1(2), (1991) 73–81

[5] Darte, A., Robert, Y., Vivien, F.: Scheduling and Automatic Parallelization. Birkhäuser Boston, (2000)

[6] Feautrier, P.: Some efficient solutions to the affne scheduling problem, part II, multidimensional time. Int. J. of Parallel Programming, 21(6), (December 1992)

[7] Feautrier, P., Griebl, M. and Lengauer, C.: Index Set Splitting. International Journal of Parallel Programming, 28(6), (2000) 607–631

[8] Patrick Le Gouëslier d'Argence: Affine Scheduling on Bounded Convex Polyhedric Domains is Asymptotically Optimal. TCS 196(1–2), (1998) 395–415

[9] Kelly, W., Maslov V., Pugh, W., Rosser, E., Shpeisman, T. and Wonnacott, D.: The Omega Library Interface Guide. Technical Report CS-TR-3445, Dept. of Computer Science, University of Maryland, College Park, (March 1995)

[10] Kelly, W., Pugh, W., Rosser, E. and Shpeisman, T.: Transitive Closure of Infinite Graphs and its Applications, International Journal of Parallel Programming, v. 24, n. 6, (December 1996) 579–598

[11] Lim, W., Lam, M. S.: Maximizing parallelism and minimizing synchronization with affine transforms. In Conference Record of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, (January 1997)

[12] Midkiff, S.P. and Padua, D.A.: A comparison of four synchronization optimization techniques. In Proc. 1991 IEEE International Conf. on Parallel Processing, (August 1991) II-9–II-16

[13] Pugh, W., Wonnacott D.: An Exact Method for Analysis of Value-based Array Data Dependences. Workshop on Languages and Compilers for Parallel Computing, (1993)

[14] Vivien F.: On the optimality of Feautrier's scheduling algorithm. In Proceedings of the EUROPAR'2002, (2002)