On Tiling Space-Time Mapped Loop Nests

[Extended Abstract]

Martin Griebl
University of Passau, FMI
griebl@fmi.uni-passau.de

ABSTRACT

Automatic parallelization of loop programs based on space-time mapping typically aims at maximal parallelism. In order to obtain reasonable performance, the granularity of the parallelism must be coarsened, e.g., by applying tiling techniques.

In contrast to others, we suggest to apply tiling *after* the computation of a space-time mapping. This extends the applicability of existing tiling methods and significantly improves efficiency of parallelized loop programs.

1. INTRODUCTION

Tiling is a well-known loop transformation technique for compiler optimization. In loop programs, multiple instances of all statements in a loop body are executed. We call them *operations*. The idea of tiling is to coalesce sets of operations. Technically, we restrict ourselves to the case that these sets, the *tiles*, are (multidimensional) *parallelepipeds*.

The problem area addressed in this paper is tiling for automatic parallelization, especially loop parallelization in the polytope model [3, 8], which is based on space-time mapping, and extensions thereof (the polyhedron model [5]).

The methods of the polyhedron model typically aim at maximal parallelism, not maximal performance, since the latter is too complex to be computed during compilation. Hence, the resulting fine-grained parallelism must be coarsened in order to reduce communication cost and obtain good performance. This is our interest in tiling.

2. RELATED WORK

So far, to the best of our knowledge, tiling is always applied to the source program, from the seminal paper of Irigoin and Triolet [7] up to most recent publications [2]. Unfortunately, the applicability of nearly all of these methods (and, hence, any method which parallelizes the tiled program) is very limited: most of them only deal with perfectly nested loops and uniform dependences, some even with limited dimensionality. These restrictions allow for provable optimality within the methods' frameworks. In contrast, our goal is to extend the applicability of tiling.

Recently, Song and Li [9] and Ahmed, Mateev and Pingali [1] developed special tiling methods for imperfectly nested loop programs in the framework of cache optimization. However, tiling methods for cache optimization cannot be used directly in tiling for coarse-grain parallelism, even if both try to improve locality. One of the main differences is the (missing) existence of distribution (i.e., parallelism) as a conflicting goal of locality. Our application domain is tiling for parallelization for distributed memory systems. (Cache optimization on every processor is a possible *subsequent* step.)

3. THE FRAMEWORK OF SPACE-TIME MAPPING

Input: Methods based on the polytope model take as input an arbitrarily nested sequential loop program. Only arrays are allowed as data structures, and only loops and conditional branches as control structures.

Output: The result of any space-time mapping is a pair of two functions: a *schedule* and an *allocation*. For every operation in the program, they determine at which logical time and on which virtual processor the operation is executed, respectively.

In the case of the polytope model we obtain multidimensional and (piecewise) affine functions as schedule and allocation – one pair per statement.

4. TILING AFTER SPACE-TIME MAPPING

Let us discuss some of the main advantages of tiling after space-time mapping.

Unique coordinates. After the space-time mapping, all operations of the parallelized loop nest are expressed in one unified coordinate system, consisting of space and time dimensions. Hence, we may reuse all tiling techniques which have already proved useful for tiling perfectly nested loops.

Flexibility of parallelization. If we apply tiling to the source program, we coalesce operations before we have any idea of how the operations should be distributed in space and time; this leads to an unnecessary loss of flexibility of the parallelization phase and, hence, to a potential loss of performance.

Affine dependences. Non-uniform dependences are easily integrated: after space-time mapping, at least the (always fully permutable) *space* dimensions may be tiled. On the other hand, it can be shown that, for general affine dependences (e.g., broadcasts), tiles must have unit extent in the *time* dimensions.

Precision. Space-time mapping gives us direct access to execution data such as (virtual) processor number and execution time, which are not available beforehand. Hence, we can use more concrete cost functions which take these data into account, e.g., in order to minimize the number of communication startups [4].

The price for these advantages is an increased compilation time due to more general scheduling, allocation, and code generation techniques. The applicability is determined by the input constraints of the polytope model (cf. Section 3).

5. TILING SPACE AND TIME

First, we do not allow tiling to arbitrarily modify, but only to coarsen a given allocation, i.e., if two operations are allocated on the same virtual processor, tiling must allocate them on the same physical processor. Hence, the mapping of virtual to physical processors must be independent of time, i.e., there are as many pairs of tile boundaries orthogonal to the time dimensions as there are space dimensions. Furthermore, one can show that the communication cost cannot be reduced by skewing the remaining boundary directions in space and time, compared to using boundaries orthogonal to the space dimensions. Thus, we may tile space and time dimensions separately. Note that this decision of keeping the allocator responsible for the allocation is a restriction and, thus, may lead us away from a global optimum. This aspect is left for future work – especially since an ad hoc solution is to apply the existing tiling techniques (including their optimality results) to the (perfectly nested) space-time mapped program, as already mentioned. Instead, we explore in more detail the obtained separation of tiling in space and time.

Tiling space dimensions means coalescing a set of virtual processors to one physical processor. This point is treated in more detail elsewhere [4].

Tiling time dimensions seems less intuitive at first glance. However, it may be necessary if the amount of work per physical processor and logical time step is too small. It can be shown that there is always at most one time loop that limits tiling; if it exists, all of its surrounding time loops cannot be tiled. An algorithm that identifies this loop (and the maximal allowed tile width thereof, based on a sufficient criterion for tiling time) is presented elsewhere [6]. Also some first experimental results on the importance of tiling time dimensions can be found there.

6. CONCLUSIONS AND FUTURE WORK

We have shown that tiling after space-time mapping extends the applicability of existing tiling methods in the context of loop parallelization.

Furthermore, tiling space dimensions can be performed with well-known techniques or with dedicated methods [4]. In most cases, tiling of time dimensions is applicable, and sometimes even necessary to obtain reasonable performance [6].

Future work will have to

- explore the interaction of allocation and tiling methods;
- deal with load balancing aspects (in the polytope model, processor number 0 has often more work to do than the others);

7. ACKNOWLEDGMENTS

I would like to thank Christian Lengauer, Nils Ellmenreich, Peter Faber, and Max Geigl for their fruitful comments on earlier versions of this paper, as well as Paul Feautrier for intensive discussions on this topic. Also thanks to Sven Anders, Armin Größlinger, Axel Krauth, and Stefan Schuster for their technical assistance. Financial Support was granted by the DAAD through PROCOPE.

8. REFERENCES

[1] N. Ahmed, N. Mateev, and K. Pingali. Synthesizing transformations for locality enhancement of imperfectly-nested loop nests. In *Conference proceedings of the 2000 International Conference on Supercomputing*, 2000. To appear.

[2] R. Andonov, S. Balev, S. Rajopadhye, and N. Yanev. Optimal semi-oblique tiling. In *Thirteenth ACM Symposium on Parallel Algorithms and Architectures*. ACM Press, July 2001. To appear.

[3] P. Feautrier. Automatic parallelization in the polytope model. In G.-R. Perrin and A. Darte, editors, *The Data Parallel Programming Model*, LNCS 1132, pages 79–103. Springer-Verlag, 1996.

[4] M. Griebl. Minimal number of communication startups by tiling space-time mapped programs. Submitted. Preliminary version available at `http://www.fmi.uni-passau.de/cl/papers/minstart.html`.

[5] M. Griebl. *The Mechanical Parallelization of Loop Nests Containing* while *Loops*. PhD thesis, Fakultät für Mathematik und Informatik, Universität Passau, Jan. 1997. Technical Report MIP-9701.

[6] M. Griebl. On the mechanical tiling of space-time mapped loop nests. Technical Report MIP-0009, Fakultät für Mathematik und Informatik, Universität Passau, Aug. 2000.

[7] F. Irigoin and R. Triolet. Supernode partitioning. In *Proc. 15th Ann. ACM Symp. on Principles of Programming Languages (POPL'88)*, pages 319–329. IEEECS, Jan. 1988.

[8] C. Lengauer. Loop parallelization in the polytope model. In E. Best, editor, *CONCUR'93*, LNCS 715, pages 398–416. Springer-Verlag, 1993.

[9] Y. Song and Z. Li. A compiler framework for tiling imperfectly-nested loops. In L. Carter and J. Ferrante, editors, *Languages and Compilers for Parallel Computing, 12th International Workshop, LCPC'99*, LNCS 1863, pages 185–200. Springer-Verlag, 1999.