Delivering High Performance to Parallel Applications Using Advanced Scheduling (Extended Abstract)

Nikolaos Drosinos, Georgios Goumas, Maria Athanasaki and Nectarios Koziris

National Technical University of Athens

School of Electrical and Computer Engineering

Computing Systems Laboratory

Zografou Campus, Zografou 15773, Athens, Greece

e-mail: {ndros, goumas, maria, nkoziris}@cslab.ece.ntua.gr

Abstract

This paper presents a complete framework for the parallelization of nested loops by applying tiling transformation and automatically generating MPI code allowing for an advanced scheduling scheme. In particular, under advanced scheduling scheme we consider two separate techniques: first, the application of a suitable tiling transformation, and second the overlapping of computation and communication when executing the parallel program. As far as the choice of a scheduling-efficient tiling transformation is concerned, the data dependencies of the initial algorithm are taken into account and an appropriate transformation matrix is automatically generated according to a well-established theory. On the other hand, overlapping computation with communication partly hides the communication overhead and allows for a more efficient processor utilization. We address all issues concerning automatic parallelization of the initial algorithm. More specifically, we have developed a tool that automatically generates MPI C code and supports arbitrary tiling transformations as well as both communication schemes, e.g. the conventional receive-compute-send scheme and the overlapping one. We investigate the performance benefits in the total execution time of the parallel program attained by the use of the advanced scheduling

scheme, and experimentally verify with the help of application-kernel benchmarks (SOR, ADI, Jacobi) that the obtained speedup can be significantly improved when overlapping computation with communication and at the same time applying an appropriate (generally non-rectangular) tiling transformation, as opposed to the combination of the standard receive-compute-send scheme with the usual rectangular tiling transformation.

1 Introduction-Background

Clusters are becoming the de-facto standard for parallel processing. The combination of commodity high performance computers with fast interconnection networks provides a very appealing solution for parallel processing. Nevertheless, the above configuration benefits are partly annulled by the message passing programming model adopted in clusters for the design of parallel applications. It is common belief that clusters are quite difficult to program, since they generally comply to the distributed memory model, and the performance achieved in parallel applications is directly associated with the efficient utilization of the required message passing library. The most commonly used message passing library, namely MPI ([3]), provides an interface for message passing functionality, but is generally more comfortably used by the scientific community in comparison to the commercial one.

Tiling or supernode transformation is one of the most common loop transformations discussed in bibliography, proposed to enhance locality in uni-processors and achieve coarse-grain parallelism in multiprocessors. Tiling groups a number of iterations into a unit (tile), which is executed uninterruptedly. From each processor's point of view, the communication required for the exchange of data between the particular processor and the other computing nodes concerns either the receipt of non-local data or the transmission of locally computed data. In the first case, data required for the computations associated with a tile assigned to a specific processor should be received from the owner processors, while in the second case data calculated locally during a tile computation of that processor should be sent to those processors that demand knowledge of their value. Obviously, tiling allows for the exchange of fewer communication messages between the processing nodes, since only one message is transmitted to each processor per tile, thus avoiding the costly startup latency of additional messages and consequently reducing the total communication time.

Traditionally, only rectangular tiling has been used for generating SPMD parallel code for distributed

memory machines. In [8], Tang and Xue provided a detailed methodology for generating efficient tiled code for perfectly nested loops, but only used rectangular tiles due to the simplicity of the parallel code, since only division and modulo operations are required in this case. However, recent scientific research has indicated that the performance of the parallel tiled code can be greatly affected by the tile size ([1], [5], [10]), as well as by the tile shape ([6], [7], [9]). The effect of the tile shape on the scheduling of the parallel program is depicted in Figure 1. It is obvious that non-rectangular tiling is more beneficial in this particular case than rectangular one, since it leads to fewer execution steps for the completion of the parallel algorithm. The main problem with arbitrary tile shapes appears to be the complexity of the respective parallel code and the performance overhead incurred by the enumeration of the internal points of a non-rectangular tile. Therefore, an efficient implementation of an arbitrary tiling method and its incorporation in a tool for automatic code generation would be desirable in order to achieve the optimal performance of parallel applications.

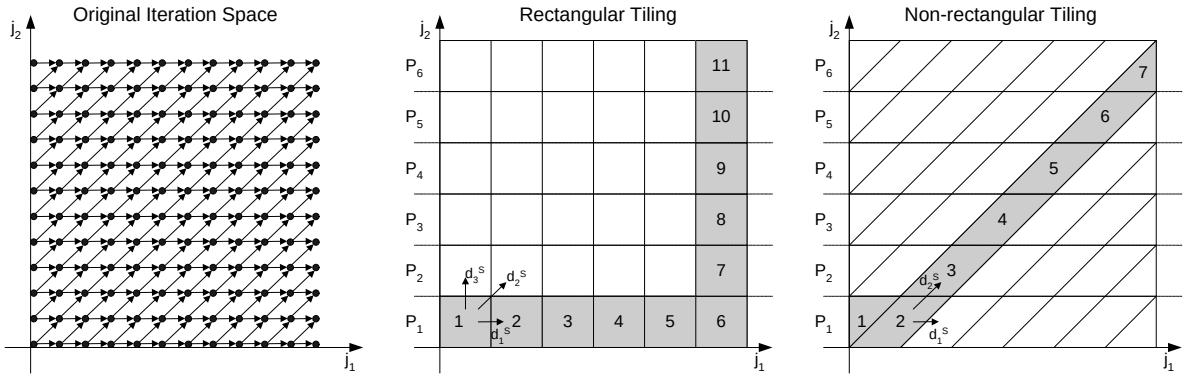

Figure 1. Effect of tile shape on overall completion time

Elaborating further more on scheduling, under conventional schemes, the required communication between different processors occurs just before the initiation and after the completion of the computations within a tile. That is, each processor first receives data, then computes all calculations involved with the current tile, and finally sends data produced by the previous calculations (see Figure 2). By providing support for an advanced scheduling scheme that uses non-blocking communication primitives and consequently allows the overlapping of useful computation with burdensome communication, it is expected that the performance of the parallel application will be further improved. This hypothesis is also established by recent scientific work ([4], [2]). More specifically, the blocking communication

primitives are substituted with non-blocking communication functions, which immediately return after being called, and can be tested for completion at a later part of the program. By doing so, after initializing non-blocking communication the processor can go on with useful computation directly related to the user application, and the communication completion can be tested as late as possible, when it will most likely have completed, and thus the processor will not have to stall idle prolonging the total execution time of the application (see Figure 3).

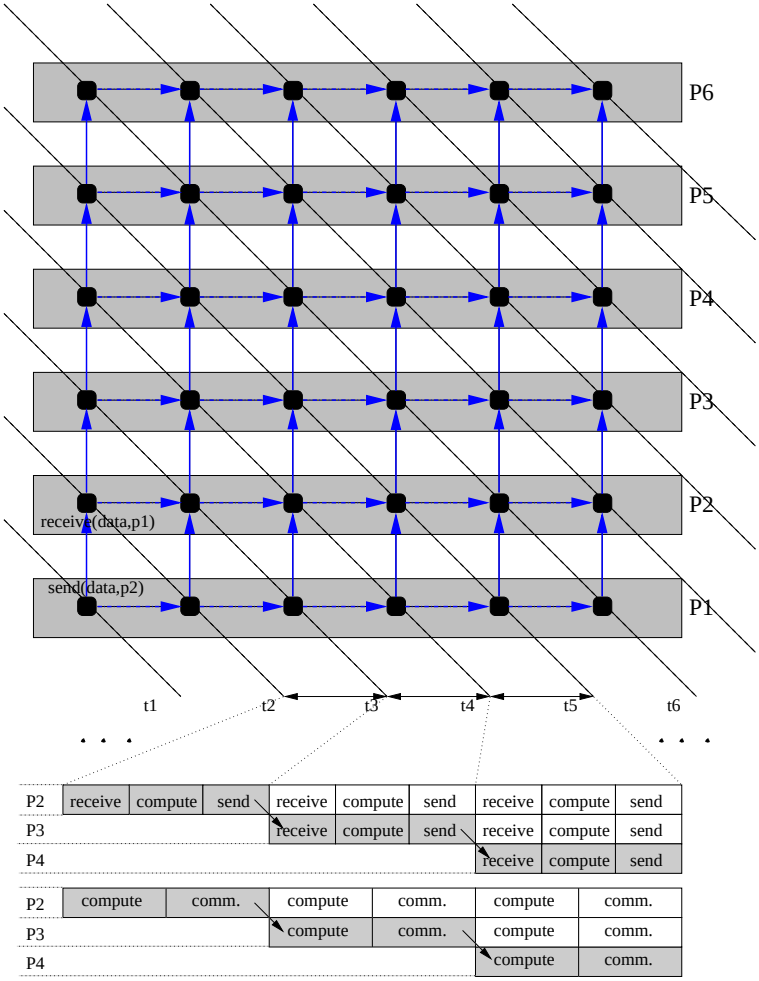

Figure 2. Non-overlapping computation and communication

2 Contribution

In this paper we combine the application of arbitrary tiling and the overlapping of computation and communication for the generation of parallel MPI code. Although, as explained above, there is active

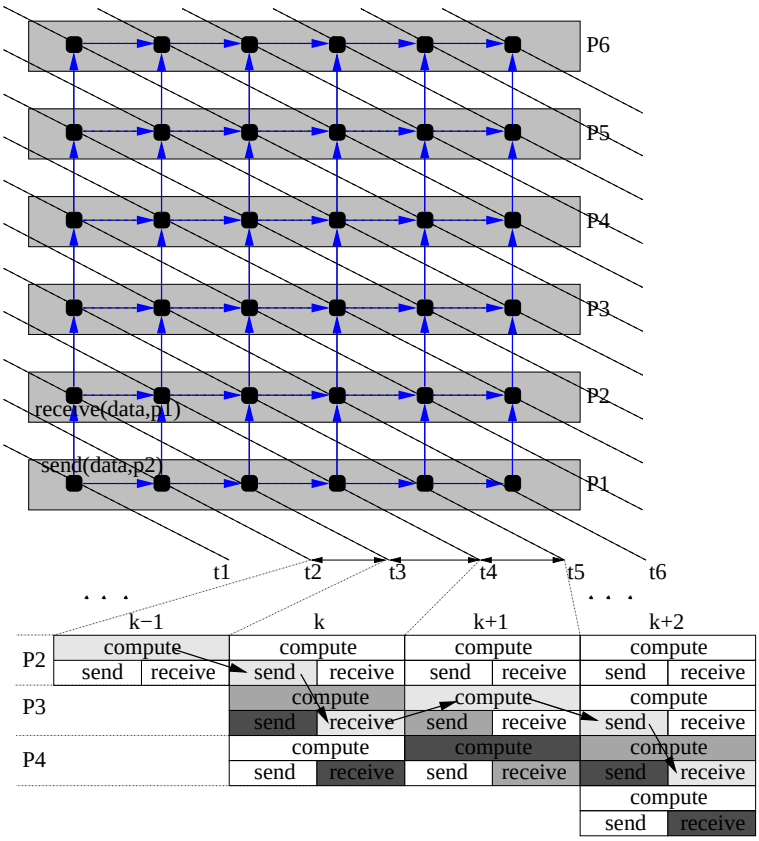

Figure 3. Overlapping computation and communication

research interest in both the determination of the optimal tile shape and size, as well as the overlapping of computation and communication, to our knowledge there is no complete framework for the automatic generation of MPI portable code that efficiently addresses all the aforementioned issues. We achieve to generate parallel SPMD code by applying tiling transformation to an algorithm of perfectly nested loops, and effectively address all issues concerning the transformation of the initial sequential code into an equivalent parallel one, such as task allocation, efficient sweeping of both the tile space and the internal of each tile, simple and straightforward partitioning of arbitrary-shaped data spaces into rectangular local data spaces, and finally MPI-based communication facilities. For a given algorithm, a suitable tiling transformation matrix is proposed, but the user is also allowed to apply different tiling transformations in order to carry out performance comparisons. Finally, apart from the standard implementation of the computation and the communication in distinct stages, the generated code optionally supports an alternative scheme with non-blocking communication primitives, enabling overlapping of computation and communication and thus achieving better performance.

According to the above, we have implemented a tool that operates as a parallel compiler for our model of target applications, and generates MPI C code. We have used a number of application-kernel benchmarks (SOR, ADI, Jacobi etc.) to test the efficiency of the parallel code on a 16-node Pentium III Linux cluster. Thus, we were able to experimentally calculate the obtained speedups for different combinations of tiling transformation matrices and both communication schemes (blocking and non-blocking). Our experimental evaluation allowed us to conclude that for our target applications the use of appropriate tiling transformation with non-blocking communication primitives is the most beneficial with respect to the total execution time of the parallel program, which is the decisive factor for delivering high performance from the user's point of view.

3 Conclusions

Summarizing, we have combined the notions of arbitrary tiling transformation and overlapping communication and computation and incorporated these aspects into a complete framework to automatically generate parallel MPI code. We have addressed all issues regarding parallelization, such as task allocation, transition from global to local data spaces, sweeping arbitrary shaped tiles and implementation of appropriate communication primitives. We have experimentally evaluated our work, and supplied results for a series of application-kernel benchmarks to verify the high performance gain obtained by the advanced scheduling scheme.

References

[1] R. Andonov, P. Calland, S. Niar, S. Rajopadhye, and N. Yanev. First Steps Towards Optimal Oblique Tile Sizing. In *8th International Workshop on Compilers for Parallel Computers*, pages 351–366, Aussois, Jan 2000.

[2] T. Andronikos, N. Koziris, G. Papakonstantinou, and P. Tsanakas. Optimal Scheduling for UET/UET-UCT Generalized N-Dimensional Grid Task Graphs. *Journal of Parallel and Distributed Computing*, 57(2):140–165, May 1999.

[3] Message Passing Interface Forum. MPI: A message passing interface standard. *International Journal of Supercomputing Applications*, 8(3/4), 1994.

[4] G. Goumas, A. Sotiropoulos, and N. Koziris. Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS'01)*, San Francisco, Apr 2001.

[5] 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.

[6] E. Hodzic and W. Shang. On Time Optimal Supernode Shape. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, CA, Jun 1999.

[7] K. Hogstedt, L. Carter, and J. Ferrante. Selecting Tile Shape for Minimal Execution time. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 201–211, 1999.

[8] P. Tang and J. Xue. Generating Efficient Tiled Code for Distributed Memory Machines. *Parallel Computing*, 26(11):1369–1410, 2000.

[9] J. Xue. Communication-Minimal Tiling of Uniform Dependence Loops. *Journal of Parallel and Distributed Computing*, 42(1):42–59, 1997.

[10] J. Xue and W.Cai. Time-minimal Tiling when Rise is Larger than Zero. *Parallel Computing*, 28(6):915–939, 2002.