Domain Specific Parallel Programming Models for Numerical Computation

Gerhard Zumbusch

Friedrich-Schiller-Universität Jena,
Institut für Angewandte Mathematik,
Ernst-Abbe-Platz 2, 07743 Jena, Germany,
zumbusch@mathe.uni-jena.de,
WWW home page: http://cse.mathe.uni-jena.de

Abstract. Following a brief review of current programming models and languages for parallel numerical computations, an splitting into distributed data container and parallel data iterators is proposed. Data distribution is implemented in application specific libraries. Data iterators are directly analysed and compiled into parallel code. Target architectures include shared and distributed memory programming styles. Some model applications including grid and tree based hierarchical numerical methods, model implementations and an auto-parallelizing source-to-source compiler are introduced.

1 Parallel Programming

Development of parallel code for numerical computations is still an issue, for several reasons: On the one hand, any computer of sufficient computing power nowadays is a parallel computer, using multi-core processors, symmetric multiprocessing and/ or clustering of network interconnected nodes. On the other hand, parallelism is still very visible in the code development: Automatic parallelization of standard programming languages is not available. Parallel libraries can hide parallelism only for certain applications. Current efficient parallel programming models are either limited in their expressiveness or lead to the well known low level parallel programming.

Current standards of parallel programming models include shared memory thread libraries like Pthreads (Posix), Java threads and the like and OpenMP as code annotation basically for loop parallelism. Distributed memory programming models are the genuin two-sided message passing libraries like MPI, PVM and one-sided communication systems like in BSP [1], Shmem (Cray/SGI), MPI-2 and in UPC [2], which often incurrs additional overhead and sometimes even relies on additional hardware support. A different approach was taken with HPF and its predecessors [3], where distributed data is restricted to multi-dimensional arrays. In many cases of array based implementations, automatic parallelization can be performed and can be efficient, with a substantial amount of index analysis.

There is a long history of research projects for parallel programming models. Currently, some commercial research projects on parallel languages are under development like Chapel (Cray) and Fortress (Sun), roughly as sucessors of HPF, X10 (IBM) extending ideas of UPC and Concur (Microsoft) establishing thread parallelism in C++ similar to the Cilk [4] project.

2 The Container-Iterator Programming Model

In the domain of numerical software, good strategies to parallelize a given sequential implementation are often known. Larger sets of data can often be decomposed and mapped to coupled parallel tasks efficiently. Further, the sequential algorithm may not need to be changed, but can be re-aranged according to the data decomposition.

We assume that data is organized in a large data structure (container) of similar smaller units, called atoms. The sequential algorithms operate on all or parts of the atoms in a similar way and in a given order. Parallelization means to schedule operations on different atoms of an algorithmic step. This approach is called data parallel. Examples for the successfull application of the scheme can be found in numerical linear algebra algorithms, finite element algorithms on large grids [5] and fast summation algorithms on trees of particles [6, chap. 8].

We restrict ourselves to numerical algorithms and the data parallel programming model. Further, we assume that a good domain specific data decomposition scheme is available. Now, numerical algorithms can be written in as a sequence of iterators. We write an iterator as a specification of the iteration space and a code fragement for the algorithmic atom. The atom can be analysed using standard compilation techniques. This leads to automatic parallelization. The automatic parallelization can target shared memory and distributed memory systems with thier respective low level programming models. Further, hybrid architectures such as clusters of shared memory nodes or hierarchies of tightly and weakly copuled systems can be targeted.

Note that this programming model includes cases of loop parallelism like in OpenMP and array parallelism like in HPF.

3 A Sample Implementation

We show how such a programming style as domain specific extensions of C++ may look like, see figure 1. Further, we present a source-to-source compiler which translates the domain specific code into standard code either using pthreads or MPI or both. Finally we present some illustrative examples, how application codes using the parallel programming model look like and how they perform.

We chose a programming style based on C++. An application specific library provides data containers like uniform grids implemented as multi-dimensional

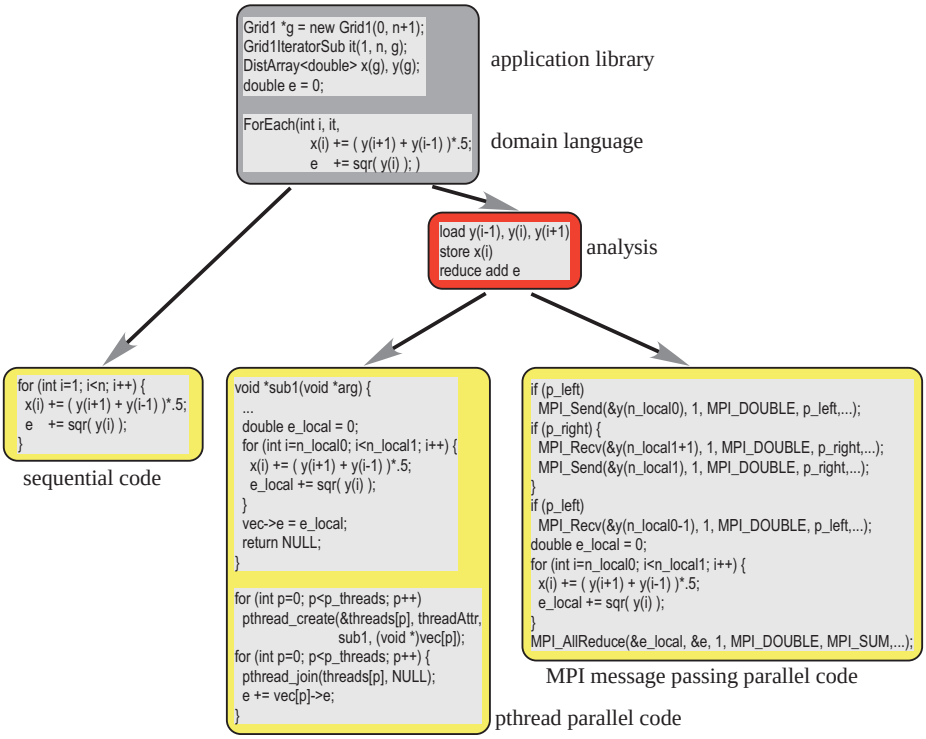

Fig. 1. Source-to-source transformation of the domain language to several target programming models. Vector/ one dimensional grid example.

arrays or a hierarchical decomposition of a set of particles implemented as quadtrees [7]. In each case a geometric domain decomposition works well for parallelization. The applications in mind based on the containers include the solution of partial differential equations, fast summation technqiues and integral equations.

The sample compilation system translates the extended C++ code using the conatiner-iterator parallel programming model into plain C++ code using standard parallel programming models. The source-to-source translation enables more flexibility than a direct single pass-compiler and can be considered as program generation [8]. Currently it is implemented using a general code dependence analysis tool and container/ iterator specific scripts. The tool uses the front end and some of the optimization stages of the back end of the Gnu gcc compiler.

The source-to-source transformation process does create the same number of communication operations than hand coded examples. Hence message passing parallel code on distributed memory computers is expected to perform comparable. The performance of multi-threaded code on shared memory computers heavily depends on the data and memory layout. This is given by the application

library, not the code transformation system. Hence, even in this case a competetive performance is expected. Experimental results to be presented support this conclusion.

4 Conclusions

A parallel programming model is presented, which allows for efficient automatic parallelization of numerical computations. However, it is designed as a set of domain dependent language extensions. An application specific libray provides a (distibuted) data container. Further, iterators are defined on the data containers. An application, which is written as a sequence of iterators can be parallelized: The iteration code is compiled into source code using standard parallel programming models. Different numerical examples demonstrate the broad applicability of the approach. Performance numbers of parallel sample codes demonstrate the efficieny of the parallelization.

References

1. Valiant, L.G.: A bridging model for parallel computation. Comm. ACM **33**(8) (1990) 103–111
2. Coarfa, C., Dotsenko, Y., Mellor-Crummey, J., Cantonnet, F., El-Ghazawi, T., Mohanti, A., Yao, Y., Chavarria-Miranda, D.: An evaluation of global address space languages: Co-array Fortran and unified parallel C. In: Proc. 10th ACM SIGPLAN symp. Principles & practice of parallel program., ACM (2005) 36–47
3. Lin, C., Snyder, L., Anderson, R., Chamberlain, B., Choi, S., Forman, G., Lewis, E., Weathersby, W.D.: ZPL vs. HPF: A comparison of performance and programming style. Technical Report UW-CSE-95-11-05, Univ. of Washington, Dept. of Computer Science and Engineering (1994)
4. Blumofe, R.D., Joerg, C.F., Leiserson, C.E., Randall, K.H., Zhou, Y.: Cilk: An efficient multithreaded runtime system. In: 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPoPP '95, ACM (1995) 207–216
5. Zumbusch, G.: Parallel Multilevel Methods. Adaptive Mesh Refinement and Loadbalancing. Teubner (2003)
6. Griebel, M., Knapek, S., Zumbusch, G., Caglar, A.: Numerische Simulation in der Moleküldynamik. Springer Lehrbuch (2004)
7. Zumbusch, G.: Data parallel iterators for hierarchical grid and tree algorithms. In Nagel, W., Walter, W., Lehner, W., eds.: Euro-Par 2006 Parallel Processing. Volume 4128 of LNCS., Springer (2006) 625–634
8. Lengauer, C.: Program optimization in the domain of high-performance parallelism. In Lengauer, C., Batory, D., Consel, C., Odersky, M., eds.: Domain-Specific Program Generation. Volume 3016 of LNCS., Springer (2004) 73–91