On the Parallelization of a Cache-Optimal Iterative Solver for PDEs based on Hierarchical Data Structures and Space-Filling Curves

Frank Günther, Andreas Krahnke, Markus Langlotz, Miriam Mehl, Markus Pögl, Christoph Zenger

Technische Universität München, Institut für Informatik,
{guenthef,krahnke,langlotz,mehl,poegl,zenger}@in.tum.de,
WWW home page: http://www5.in.tum.de

Abstract. Competitive numerical simulation codes solving partial differential equations have to tap the full potential of both modern numerical methods – like multi-grid and adaptive grid refinement – and available computing resources. In general, these two are rival tasks. Typically, hierarchical data structures resulting from multigrid and adaptive grid refinement impede efficient usage of modern memory architectures on the one hand and complicate the efficient parallelization on the other hand due to scattered data for coarse-level-points and unbalanced data trees. In our previous work, we managed to bring together high performance aspects in numerics as well as in hardware usage in a very satisfying way. The key to this success was to integrate space-filling curves consequently not only in the programs flow control but also in the construction of data structures which are processed linearly even for hierarchical multilevel data. In this paper, we present first results on the second challenge, namely the efficient parallelization of algorithms working on hierarchical data. It shows that with the same algorithms as desribed above, the two main demands on good parellel programs can be fulfilled in a natural way, too: The balanced data partitioning can be done quite easily and cheaply by cutting the queue of data linearized along the space-filling curve into equal pieces. Furtheron, this partitioning is quasi-optimal regarding the amount of communication. Thus, we will end up with a code that overcomes the quandary between hierarchical data and efficient memory usage and parallelization in a very natural way by a very deep integration of space-filling-curves in the underlying algorithm.

1 The Sequential Algorithm

In this section we give a very short description of the underlying sequential program, which is proven to combine numerical and hardware efficiency on a very high level. It was developed to show that modern numerical methods based on hierarchical data representation like multi-grid and adaptive grid refinement are no contradiction to a very efficient usage of modern hardware architectures like memory hierarchy[6].

The basic idea is to deeply integrate the concept of space-filling curves [10] in the flow control of a recursive cell-oriented iterative program for solving PDEs. It was known before that – due to locality properties of the curves – reordering grid cells according to the numbering induced by a space-filling curve improves chache-efficiency (see e.g. [1]). We go one step further and – in addition to the reordering of cells – construct appropriate data structures wich are processed strictly linear. The numerical tasks are to implement a Finite-Element-Method with adaptive grid refinement and an additive multi-grid method, which leads in a natural way to hierarchical generating systems for the FEM-spaces to be constructed. Bringing together these generating systems with space-filling curves, we could show that all data, e.g. coefficients in the generating systems, can be handled by a small number of stacks which are processed strictly linear. The number of stacks only depends on the dimensionality (2D or 3D) of the problem but not on the depth of the underlying spacetree. The space-filling curve defines a set of locally deterministic rules for the stack-access. Thus, all prerequisites for high efficiency of memory access in the memory hierarchy of modern processors as well as for a good exploitation of prefetching techniques are fulfilled: Our data access is highly local both in time and in space[7].

As presented in [6], [7] and [8] we obtain hit-rates on the L2-cache of modern processors beyond 99,0%. We can show that the measurable amount of cache-misses is in the same order of magnitude as the minimum for every imaginable iterative method. So the time spent in memory access is no longer the limitating factor to the runtime of the program. In addition, we also overcame the second aspect of memory-boundedness: Typically, the grid-resolution is limited by the size of main memory in a given computer. In our algorithm, the space-filling curve gives an absolutly precise prediction, at which point during an iteration a special part of the linear data is needed. So it is possible to load data stored on hard disk "just in time" to the main memory without losing efficiency. With a fixed number of relatively small buffers we can (in principle) process grids, where the maximal resolution is limited only by the size of hard disks. "In principle" means, that now the time needed for calculations is the limiting factor, which of course becomes drastic for resolutions far beyond the size of main memory.

In consequence, the next thing to do is to parallelize the code to break through the barriers resulting from the calculation time of large problems.

2 Parallelization of the Algorithm

It is well known that the concept of space-filling curves [10] fits the demands of data parallel approaches based on domain decomposition very well: Data are linearized along the space-filling curve and can therefore be partitioned in a balanced way very easily and cheaply (see figure 1 for a simple two-dimensional example). In particular, the balanced partitioning is much cheaper than via graph algorithms. Further on, the resulting domain decomposition can be shown to be quasi-optimal regarding the size of interfaces of subdomains and, thus, communication costs [1–5, 9, 11, 12]. Another advantage is the negligible amount

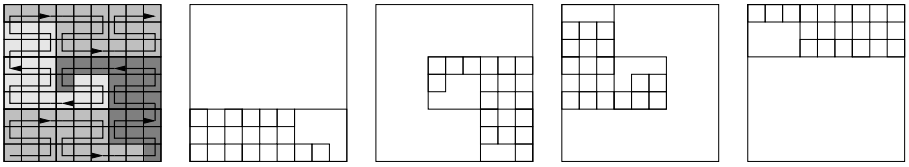

Fig. 1. Partitioning of a regular two-dimensional grid using the Peano-curve (left-hand side: partitioning, right-hand side: grids assigned to the resulting four processes)

of memory that is necessary to store the partitioning. Generally, a process has to know the location of its boundary. In the three dimensional case, we would have to store a two dimensional boundary. Using the space-filling curve linearisation we can represent the partitioning by the points, where the space-filling curve is cut into parts (*seperators*). Hence, we only need $p-1$ seperators to store all information on the domain decomposition. These seperators can be determined very cheaply by counting the cell numbers during one traversion of the grid along the Peano-curve (for a definition og the Peano-curve see [10]).

Difficulties for the parallelization are caused by the fact, that the grid data are logically associated to the vertices of the grid cells (this is nessecary to allow a cell-oriented operator evaluation) but stored pointwise on several stacks. Usually, a data parallel approach divides the domain and the corresponding data structures in the same way. Our approach uses more sophisticated techniques to realize the data partitioning in order to preserve the data structures of the sequential algorithm and its linear processing, and, thus, the high cache efficiency. The main difficulty is to build up the stack structure, which is needed to start the traversion of the grid along the Peano's curve at any point in the domain. This can be done by introducing some extra information for the coarse boundary nodes, which is stored on the stacks. As the number of coarse boundary nodes is small in comparision to the total number of nodes, the additional need in memory is small, too. As a result, the parallel implementation still passes through the whole domain in each process, but stays as coarse as possible within these parts of the domain, which do not belong to the process. Together with the fact that no compuations are performed outside the subdomain assigned to the process, the additional work ist very small.

In the context of multigrid, the additional storage of coarse level cells in each process even turns into an advantage as (cell-parts) of the residual can be restricted locally on one process and will be exchanged between processes like other 'boundary' data. Thus, for the additive multigrid algorithm, we can do completely without any master process collecting and updating coarse grid data.

Finally, we need an efficient method for exchanging the data of the boundary nodes. We implemented an efficient algorithm to determine (once per (re-)partitioning), which boundary node has to be sent to which processor. This leads to a minimal communication effort as each boundary node is only

exchanged between the two associated processes and the number of boundary nodes is quasi-minimal.

3 Results

The above mentioned changes were implemented and tested against the original sequential version. The parallel implementation executed within one process led to a cache efficiency of above 99,0%. Running the same example on multiple processors showed no deterioration. All processes still have hitrates of more than 99,0%. Hence, one can see the linear access to the data, which is responsible for the cache optimality still works fine in the parallel case. Furthermore, the parallelization overhead doesn't influence the overall L2 cache efficiency.

As the needed effort for data exchange is proportional to the number of boundary nodes one can expect, that the time used for communication is small in comparision to the time needed for the calculation. This is the basic requirement to reach a linear speedup. Up to now, we couldn't ascertain the speedup, as the efficient version of the algorithm, which is responsible for the data exchange of the boundary values has not been debugged yet.

4 Conclusion

We could show, that the parallel implementation has the following properties:

- It extends the cache-optimal sequential core to the possibility of parallel execution without decaying the cache-optimality for an individual instance of the program on a single processor.
- Due to our concept based on space-filling curves, both the overhead for partitioning the domain and for data communication is quasi-optimal.
- Coarse level data require no extra handling by a master process.

With the final (not yet debugged) version we expect a linear speedup-property because of minimized communication-work. We are sure that we are able to present the complete transfer of the sequential code's advantages to a cluster in combination with the optimal parallelization strategy at the time of the conference. In addition, we will present more detailed results on runtime, scalability and parallelization overhead.

References

1. M. J. Aftosmis, M. J. Berger, and G. Adomavivius, *A Parallel Multilevel Method for adaptively Refined Cartesian Grids with Embedded Boundaries*, AIAA Paper, 2000.
2. W. Clarke, *Key-based parallel adaptive refinement for FEM*, bachelor thesis, Australian National Univ., Dept. of Engineering, 1996.

3. M. Griebel, S. Knapek, G. Zumbusch, and A. Caglar, *Numerische Simulation in der Moleküldynamik. Numerik, Algorithmen, Parallelisierung, Anwendungen.* Springer, Berlin, Heidelberg, 2004.
4. M. Griebel and G. W. Zumbusch, *Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves*, Parallel Computing, 25:827-843, 1999.
5. M. Griebel and G. Zumbusch, *Hash based adaptive parallel multilevel methods with space-filling curves*, in Horst Rollnik and Dietrich Wolf, editors, NIC Symposium 2001, volume 9 of NIC Series, ISBN 3-00-009055-X, pages 479-492, Germany, 2002. Forschungszentrum Jülich.
6. F. Günther, M. Mehl, M. Pögl, C. Zenger *A Cache-Aware Algorithm for PDEs on hierarchical data structures based on space-filling curves*, SIAM Journal on Scientific Computing, submitted.
7. F. Günther *Eine cache-optimale Implementierung der Finite-Elemente-Methode*, Dissertation, TU München, 2004.
8. M. Pögl, *Entwicklung eines ccheoptimalen 3D Finite-Element-Verfahrens für große Probleme*, Dissertation, TU München, 2004.
9. S. Roberts, S. Klyanasundaram, M. Cardew-Hall, and W. Clarke, *A key based parallel adaptive refinement technique for finite element methods*, in Proc. Computational Techniques and Applications: CTAC '97, B. J. Noye, M. D. Teubner, and A. W. Gill, eds. World Scientific, Singapore, 1998, p. 577-584.
10. H. Sagan, *Space-Filling Curves*, Springer-Verlag, New York, 1994.
11. G. Zumbusch, *Adaptive Parallel Multilevel Methods for Partial Differential Equations*, Habilitationsschrift, Universität Bonn, 2001.
12. G. W. Zumbusch, *On the quality of space-filling curve induced partitions*, Z. Angew. Math. Mech., 81:25-28, 2001. Suppl. 1, also as report SFB 256, University Bonn, no. 674, 2000.