DiMEPACK — A Cache–Optimized Multigrid Library*

Markus Kowarschik
Lehrstuhl für Informatik 10 (Systemsimulation)
Institut für Informatik, Universität Erlangen–Nürnberg
Cauerstraße 6, D–91058 Erlangen, Germany
markus.kowarschik@cs.fau.de

Christian Weiß
Lehrstuhl für Rechnertechnik und Rechnerorganisation
Institut für Informatik, Technische Universität München
D–80290 München, Germany
weissc@cs.tum.edu

Abstract *The efficient execution of numerically intensive codes often suffers from high memory access times. Efficient execution can only be expected if the codes respect the hierarchical memory design of the underlying architecture, including its cache memories. Unfortunately, even modern compilers are not very successful in performing data locality optimizations to enhance the performance of the codes, so that most of this effort is left to the programmer.*

DiMEPACK is a combined C++/FORTRAN77 library of cache–optimized multigrid implementations for the numerical solution of elliptic partial differential equations (PDEs) in two dimensions. DiMEPACK can handle constant–coefficient problems on rectangular domains. It implements a set of highly tuned red–black Gauss–Seidel smoothers as well as cache–aware intergrid transfer operators. The number of cache conflict misses is reduced by the application of various array padding heuristics. Furthermore, we reduce the total number of floating–point operations by introducing several arithmetic optimizations.

In this paper, we explain the design of the DiMEPACK library and present performance results.

Keywords: cache optimization, performance, iterative methods, multigrid

*This work is presently supported in part by the *Deutsche Forschungsgemeinschaft* (DFG), grants Ru 422/7–1,2,3.

1 Introduction

Our research on cache–aware numerical methods in general and the development of the DiMEPACK [1] library have been motivated by two independent observations.

Firstly, it is well–known that the speed of processors has been increasing and will even continue to increase much faster than the speed of memory components. As a general consequence, current memory chips based on DRAM technology cannot provide the data to the CPUs as fast as necessary. This memory bottleneck often results in significant idle periods of the processors and thus in very poor code performance compared to the theoretically available peak performances of the machines under consideration. To mitigate this effect modern computer architectures use cache memories which keep data frequently used by the CPU. Caches are usually based on SRAM chips which on the one hand are much faster than DRAM components, but on the other hand have comparatively small capacities, for both technical and economical reasons.

Efficient execution can therefore be achieved only if the hierarchical structure of the memory subsystem (including main memory, caches and the processor registers) is respected by the

[1]DiME = **D**ata–local **i**terative **ME**thods

code, especially by the order of memory accesses. Unfortunately, even modern compilers are not very successful in performing data locality optimizations to enhance cache efficiency. As a consequence, most of this effort is left to the programmer.

Secondly, it has been demonstrated that — at least from a theoretical point of view — multigrid methods are among the most efficient algorithms for the solution of large systems of linear equations arising for instance in the context of numerical PDE solution [1]. Multigrid techniques belong to the class of iterative methods, which means that the underlying data set, which in general is very large, is repeatedly processed several times.

It is our principal goal to investigate in how far multigrid methods can be restructured in order to respect the hierarchical memory design of modern computer architectures and thus significantly speed up their execution. We have investigated various optimization techniques [2, 3, 4] and have integrated them into our DiMEPACK library [5].

Cache optimization techniques — including data access transformations [6] like *loop interchange, loop fusion, loop blocking,* and *prefetching* as well as data layout transformations [7] like *array padding* and *array merging* — have proven their efficiency to optimize cache hit rates for example for matrix multiplication algorithms [8], linear algebra packages for dense matrices [9], and FFT algorithms [10]. Compilers are already able to automatically generate codes for these problems which nearly run with peak performance on cache–based architectures. However, current compiler technology is apparently not able to successfully apply such kinds of optimization techniques to iterative algorithms.

Some research in the area of cache optimizations for iterative methods has been done by U. Rüde et al. [11], C.C. Douglas et al. [12], D.E. Keyes et al. [13], and D. Quinlan et al. [14].

Research to automatically perform data locality optimizations has been done in the SUIF project [15] and by Banerjee et al. [16], but it seems that the techniques are still far away from being applicable to more complex methods, like for example multigrid.

In addition to various cache optimization techniques, DiMEPACK uses sophisticated multigrid implementations combining smoothing steps and intergrid transfer operations, i.e. residual restriction and error prolongation. The idea behind the combination of the multigrid components is to save passes through the entire data set, i.e. to enhance the temporal locality of the code. In order to avoid the occurrence of severe cache conflict misses, we have implemented various *array padding* heuristics [7].

DiMEPACK provides a comfortable C++ user interface. This interface comprises data types for grid functions, operators, etc. However, all computationally intensive parts are implemented in FORTRAN77 in order to enable aggressive optimizations by the compiler. This multi–language approach is completely hidden from the user.

Our paper is organized as follows. In Section 2 we describe the DiMEPACK library, briefly explain our locality optimization techniques, and present a survey on performance data. Finally, we summarize our results in Section 3.

2 The DiMEPACK Library

2.1 Functionality of DiMEPACK

DiMEPACK is a library of cache–aware multigrid routines [1]. It implements both V–cycles and full multigrid (FMG, nested iteration) based on a coarse–grid correction scheme. Full–weighting as well as half–weighting are implemented as restriction operators. The prolongation of the coarse–grid corrections is done using bilinear interpolation. DiMEPACK uses a Gauss–Seidel/SOR smoother based on a red–black ordering of the unknowns. It can be shown that, even in the case of certain anisotropic operators, the introduction of suitable relaxation parameters can help to achieve good smoothing properties with standard smoothing algorithms [17].

So far, DiMEPACK can handle constant–coefficient problems based on discretizations using 5–point or 9–point stencils. However, we have also investigated the case of variable coefficients [18, 19]. DiMEPACK is applicable to problems on rectangular domains where different mesh widths in both space dimensions are permitted. It can handle both Dirichlet and Neumann boundary conditions. Equations for Neumann boundary nodes are obtained by introducing second–order central difference formulae for approximating the external normal derivatives. The arrays containing the boundary data have to be passed as parameters to the DiMEPACK library functions.

The user may specify the total number of grid levels to be used. The linear systems on the coarsest grid are solved using LAPACK library routines [9]. For this purpose the corresponding matrix is split into two triangular factors in the beginning of the computation. If this matrix is symmetric and positive definite, a Cholesky factorization is computed. Otherwise an LU factorization is determined. In the course of the iterative process the coarsest systems are then solved using forward–backward substitution steps.

We have implemented various stopping criteria for the multigrid iterations which can be specified by the user. In general, the computation stops as soon as either a maximum number of multigrid cycles have been performed or as soon as the discrete L2 norm or the maximum norm of the residual has dropped below a prescribed threshold value.

Before the compilation of the DiMEPACK library the user may set an environment variable in order to specify either single precision or double precision as the type of floating–point representation to be used. The choice of single precision arithmetic can significantly speed up code execution.

DiMEPACK can handle certain problem classes yielding singular matrices, e.g., Poisson problems with pure Neumann boundary conditions. In this case, individual solutions of the coarsest–grid problems are selected by arbitrarily choosing the value of the unknown in a corner of the computational domain [20].

See the DiMEPACK User Manual [5] for a more detailed description of its functionality and features.

2.2 Optimization Techniques

2.2.1 Arithmetic Optimizations

Arithmetic optimizations do not aim at enhancing the cache performance, but at minimizing the number of floating–point operations to be executed, without losing the identity of the numerical results[2]. The following arithmetic optimizations are implemented in DiMEPACK :

- DiMEPACK respects that, if both a 5–point stencil and a Gauss–Seidel smoother are used, the residuals vanish at the black grid nodes. This drastically simplifies the implementation of the restriction operators.

- In order to save multiply and divide operations, both the relaxation parameter of the smoothing routine (if different from 1) and the diagonal entries of the matrices are respected during a preprocessing step for the finest grid and in the course of the residual restrictions for the coarser grids, respectively.

- If the problem is homogeneous, i.e. if the right–hand side of the linear system corresponding to the finest grid equals 0, DiMEPACK uses dedicated smoothing functions in order to avoid unnecessary memory accesses.

- If the matrix to be factorized is symmetric and positive definite, we apply Cholesky's method instead of computing an LU decomposition. This approximately saves 50% of the corresponding floating–point operations.

[2]Marginal differences in the numerical results may occur due to a reordering of the arithmetic instructions by the compiler. Note that certain arithmetic rules, like for example the law of associativity, do not hold for finite–precision floating–point arithmetic.

- DiMEPACK tries to save multiply operations by factoring out identical stencil coefficients.

2.2.2 Locality Optimizations

Data access optimizations In general, these techniques change the order in which the data are referred in the course of the computation. The idea is to enhance both the spatial and the temporal locality of the codes, yielding higher cache hit rates and, as a consequence, lower execution times. We have implemented the optimization techniques *loop fusion* as well as one– and two–dimensional *loop blocking (tiling)*. The basic idea behind these data access optimizations is to use a working set which is small enough to be kept in cache. As a consequence, the cache hit rates increase.

Since we have previously reported in detail on how these techniques work, we refer the reader to [2, 3].

Data layout optimizations These techniques address the storage schemes which are used to arrange the data in memory. One of the most prominent layout optimization techniques is *array padding* [7]. Padding is based on the introduction of additional array elements, which are never accessed, but in fact modify the relative distances of array elements in memory. This can significantly reduce the occurrence of cache conflict misses. DiMEPACK uses appropriate paddings for all arrays involved in the computation. Our padding techniques have also been presented in [2, 3].

2.3 Performance Results

In this section we present performance results showing the efficiency of our optimization techniques for the Gauss–Seidel smoother, which usually is the most time–consuming part of a multigrid implementation and takes more than 70% of the overall computing time.

Figure 1 shows detailed performance results of our different locality optimizations on an A21164–based Digital PWS 500au (500 MHz)

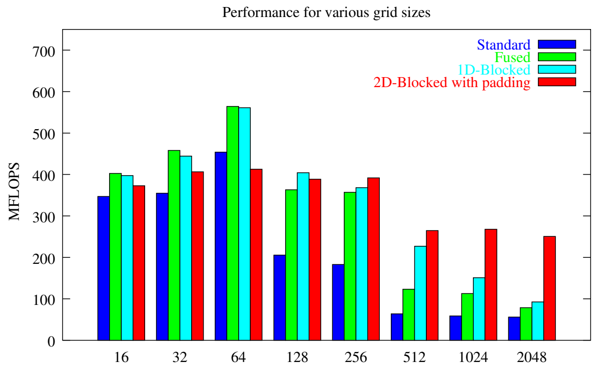

Figure 1: Performance results for the various optimization techniques of the Gauss–Seidel smoother on an Alpha 21164 machine.

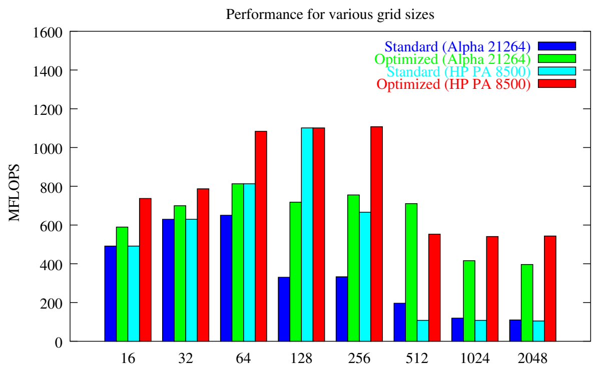

Figure 2: Performance results for the Gauss–Seidel smoother on an Alpha 21264 and on an HP PA8500 machine.

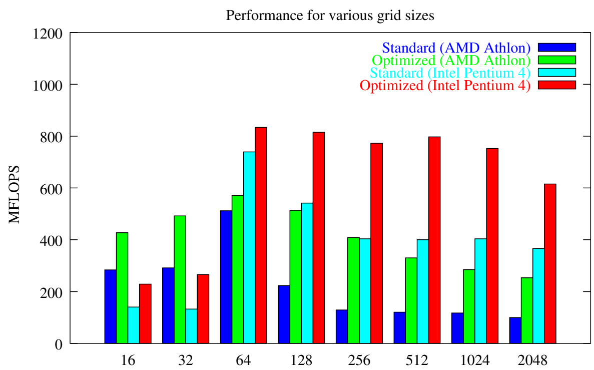

Figure 3: Performance results for the Gauss–Seidel smoother on an AMD Athlon based PC and an Intel Pentium 4 based PC.

running under Compaq Tru64 Unix. The results of the 2D–blocked codes are based on the use of appropriate paddings. It can be observed that the application of our loop fusion and one–dimensional loop blocking techniques significantly speeds up the execution for the smaller grid sizes. However, since both techniques assume that a certain number of grid lines simultaneously fit into the cache, they fail to sustain high performance for the 1024×1024 and 2048×2048 grids. In contrast, the two–dimensional blocking technique even improves the performance for the largest grid sizes. This is due to the fact that this approach uses a working set which does not depend on the grid size and which is chosen small enough to fit in cache.

Figures 2 and 3 provide a survey of speedups for four other architectures. The performance data shown in Figure 2 correspond to a Compaq XP1000 machine (A21264, 500 MHz, Compaq Tru64 Unix, Compaq f77) and an HP Convex Exemplar (PA8500, 400 MHz, HP–UX, HP f77), whereas Figure 2 shows performance data for an AMD Athlon based PC (800 MHz, Linux, GNU g77) and an Intel Pentium 4 based PC (1.5 GHz, Linux, GNU g77). In these figures we merely show the MFLOPS rates for the standard codes and the highest MFLOPS rates which can be achieved by applying our techniques.

Generally, it can be observed that our optimizations perform well on all architectures characterized by hierarchical memory structures. More detailed investigations using various profiling tools reveal that, in general, the hit rates can significantly be increased for all levels of the memory hierarchies.

3 Conclusions

Applying cache optimization techniques to numerically intensive codes can significantly speed up their execution on all architectures which are characterized by hierarchical memory designs. DiMEPACK is a multigrid library comprising various kinds of cache performance and arithmetic optimizations. We have given performance results showing the effectiveness of our efforts.

Our research has also focused on the development of cache optimization techniques for more general problem classes, e.g., involving variable coefficients. In these cases, finding cache–aware data layouts turns out to be much more complicated than in the case of constant coefficients [19]. Cache optimizations for multigrid on unstructured grids have also been investigated [4].

Presently, we are concentrating on algorithmic extensions involving combinations of patch–based multigrid approaches and adaptive processing strategies. These techniques are characterized by a high inherent potential of locality.

References

[1] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid.* Academic Press, 2001.

[2] M. Kowarschik, U. Rüde, C. Weiß, and W. Karl. Cache–Aware Multigrid Methods for Solving Poisson's Equation in Two Dimensions. *Computing*, 64(4):381–399, 2000.

[3] C. Weiß, W. Karl, M. Kowarschik, and U. Rüde. Memory Characteristics of Iterative Methods. In *Proceedings of the ACM/IEEE SC99 Conference*, Portland, Oregon, November 1999.

[4] C.C. Douglas, J. Hu, M. Kowarschik, U. Rüde, and C. Weiß. Cache Optimization for Structured and Unstructured Grid Multigrid. *Electronic Transactions on Numerical Analysis*, 10:21–40, February 2000.

[5] W. Karl, M. Kowarschik, U. Rüde, and C. Weiß. DiMEPACK — A Cache–Aware Multigrid Library: User Manual. Technical Report 01–1, Lehrstuhl für Informatik 10 (Systemsimulation), University

of Erlangen–Nuremberg, Germany, 2001. *www10.informatik.uni-erlangen.de.*

[6] D.F. Bacon, S.L. Graham, and O.J. Sharp. Compiler Transformations for High-Performance Computing. *ACM Computing Surveys*, 26(4):345ff, December 1994.

[7] G. Rivera and C.-W. Tseng. Data Transformations for Eliminating Conflict Misses. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'98)*, Montreal, Canada, June 1998.

[8] J. Bilmes, K. Asanovic, C.-W. Chin, and J. Demmel. Optimizing Matrix Multiply using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology. In *Proceedings of International Conference on Supercomputing*, July 1997.

[9] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide.* SIAM, 3rd edition, 1999.

[10] M. Frigo and S. G. Johnson. The Fastest Fourier Transform in the West. Technical Report MIT-LCS-TR-728, Massachusetts Institute of Technology, September 1997.

[11] U. Rüde. Iterative Algorithms on High Performance Architectures. In *Proceedings of the EuroPar97 Conference*, Lecture Notes in Computer Science, pages 26–29. Springer, August 1997.

[12] C. C. Douglas. Caching in With Multigrid Algorithms: Problems in Two Dimensions. *Parallel Algorithms and Applications*, 9:195–204, 1996.

[13] D.K. Kaushik, D.E. Keyes, and B.F. Smith. On the Interaction of Architecture and Algorithm in the Domain-Based Parallelization of an Unstructured Grid Incompressible Flow Code. In J. Mandel et al., editor, *Proceedings of the 10th Intl. Conf. on Domain Decomposition Methods*, pages 311–319, 1998.

[14] D. Quinlan F. Bassetti, K. Davis. Temporal Locality Optimizations for Stencil Operations within Parallel Object-Oriented Scientific Frameworks on Cache-Based Architectures. In *Proceedings of the PDCS'98 Conference*, Las Vegas, Nevada, July 1998.

[15] M. E. Wolf and M. S. Lam. A Data Locality Optimizing Algorithm. In *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation*, June 1991.

[16] M. Kandemir, A. Choudhary, J. Ramanujam, and P. Banerjee. Improving Locality Using Loop and Data Transformations in an Integrated Framework. In *Proc. 31st International Symposium on Micro-Architecture (MICRO-31)*, Dallas, Texas, December 1998.

[17] I. Yavneh. On Red–Black SOR Smoothing in Multigrid. *SIAM J. Sci. Comp.*, 17(1):180–192, January 1996.

[18] H. Pfänder. Cache–optimierte Mehrgitterverfahren mit variablen Koeffizienten auf strukturierten Gittern. Master's thesis, Department of Computer Science, University of Erlangen–Nuremberg, Germany, 2000.

[19] M. Kowarschik. Enhancing the Cache Performance of Multigrid Codes on Structured Grids. *Electronic Transactions on Numerical Analysis*, 2002. to appear.

[20] A. Berman and R.J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences.* SIAM, 1994.