Towards Cache-Optimized Multigrid Using Patch-Adaptive Relaxation

Markus Kowarschik[1], Iris Christadler[1], and Ulrich Rüde[1]

System Simulation Group, Computer Science Department
Friedrich-Alexander-University Erlangen-Nuremberg, Germany
{Markus.Kowarschik,Iris.Christadler,Ulrich.Ruede}@cs.fau.de

Abstract. Most of today's computer architectures employ fast, yet relatively small cache memories in order to mitigate the effects of the constantly widening gap between CPU speed and main memory performance. Efficient execution of numerically intensive programs can only be expected if these hierarchical memory designs are respected. Our work targets the optimization of the cache performance of multigrid codes. The research efforts we will present in this paper first cover transformations that may be automized and then focus on fundamental algorithmic modifications which require careful mathematical analysis. We will present experimental results for the latter.

1 Introduction

In order to mitigate the effects of the constantly widening gap between CPU speed and main memory performance, today's computer architectures typically employ hierarchical memory designs involving several layers of fast, yet rather small cache memories (caches) [6]. Caches contain copies of frequently used data.

However, efficient code execution can only be expected if the codes respect the structure of the underlying memory subsystem. Unfortunately, current optimizing compilers are not able to synthesize chains of complicated cache-based code transformations. Hence, they rarely deliver the performance expected by the users and much of the tedious and error-prone work concerning the tuning of the memory efficiency is thus left to the software developer [3, 8].

The research we will present in this paper focuses on the optimization of the data cache performance of multigrid methods which have been shown to be among the fastest numerical solution methods for large sparse linear systems arising from the discretization of elliptic partial differential equations (PDEs) [14]. In particular, we will concentrate on the optimization of the smoother which is typically the most time-consuming part of a multigrid implementation [7, 15]. The smoother is employed in order to eliminate the highly oscillating components of the error on the corresponding grid levels. Conversely, the coarse-grid correction is responsible for the reduction of the slowly oscillating error frequencies.

Our techniques can be divided into two categories. On the one hand, we will briefly address "compiler-oriented" cache optimization approaches which do not modify the results of the numerical computations and may therefore be fully automized using a compiler or a source code restructuring tool; e.g., ROSE [11]. On the other hand, we will describe a more fundamental approach which is derived from the theory of the *fully adaptive multigrid method* [13] and based on a patch-adaptive processing strategy. This novel algorithm requires careful mathematical analysis and is therefore a long way from being automatically introduced by a compiler [7].

See [3, 8, 9] for comprehensive surveys on memory hierarchy optimizations. In particular, cache performance optimizations for computations on structured grids have further been presented in [7, 12, 15]. For the case of unstructured grids, we particularly refer to [2].

Our paper is structured as follows. Compiler-oriented cache-based transformations will be reviewed briefly in Section 2. Section 3 contains the description of the adaptive relaxation schemes. Afterwards, experimental results will be demonstrated in Section 4. In Section 5, we will draw some final conclusions.

2 Compiler-Oriented Data Locality Optimizations

2.1 Data Layout Optimizations

Data layout optimizations aim at enhancing code performance by improving the arrangement of the data in address space [7]. On the one hand, such techniques are applied to change the mapping of array data to the cache frames, thereby reducing the number of cache conflict misses and avoiding cache thrashing [3]. This is commonly achieved by a layout transformation called *array padding* which introduces additional array elements that are never referenced in order to modify the relative distances of array entries in address space [12].

On the other hand, data layout optimizations can be applied to increase spatial locality. In particular, they can be introduced in order to enhance the reuse of cache blocks once they have been loaded into cache. Since cache blocks contain several data items that are arranged next to each other in address space it is reasonable to aggregate those data items which are likely to be referenced within a short period of time. This technique is called *array merging* [6].

For a detailed discussion of data layout optimizations for multigrid codes, we again refer to [7, 15].

2.2 Data Access Optimizations

Data access optimizations are code transformations which change the order in which iterations in a loop nest are executed. These transformations strive to improve both spatial and temporal locality. Moreover, they can also expose parallelism and make loop iterations vectorizable.

Classical loop transformations cover *loop fusion* as well as *loop blocking (tiling)*, amongst others [1]. Loop fusion refers to a transformation which takes two adjacent loops that have the same iteration space traversal and combines their bodies into a single loop, thereby enhancing temporal locality. Furthermore, fusing two loops results in a single loop which contains more instructions in its body and thus offers increased instruction-level parallelism. Finally, only one loop is executed, thus reducing the total loop overhead by a factor of 0.5 [8].

Loop blocking refers to a loop transformation which increases the depth of a loop nest with depth d by adding additional loops. The depth of the resulting loop nest will be anything from $d+1$ to $2d$. Loop blocking is primarily used to improve temporal locality by enhancing the reuse of data in cache and reducing the number of cache capacity misses. Tiling a single loop replaces it by a pair of loops. The inner loop of the new loop nest traverses a *block (tile)* of the original iteration space with the same increment as the original loop. The outer loop traverses the original iteration space with an increment equal to the size of the block which is traversed by the inner loop. Thus, the outer loop feeds blocks of the whole iteration space to the inner loop which then executes them step by step [8].

As is the case for data layout transformations, a comprehensive overview of data access optimizations for multigrid codes can be found in [7, 15]. See [2] for blocking approaches for unstructured grids.

3 Enhancing Data Locality Through Adaptive Relaxation

3.1 Overview: Fully Adaptive Multigrid

Our novel patch-adaptive multigrid scheme is derived from the theory of the fully adaptive multigrid method. The latter combines two different kinds of *adaptivity* into a single algorithmic framework [13]. Firstly, *adaptive mesh refinement* is taken into account in order to reduce the number of degrees of freedom and, as a consequence, the computational work as well as the memory consumption of the implementation. Secondly, the fully adaptive multigrid method employs the principle of *adaptive relaxation* [13]. With this update strategy, computational work can be focused on where it can efficiently improve the quality of the numerical approximation.

In this paper, we will concentrate on the principle of adaptive relaxation only and demonstrate how it can be exploited to enhance cache efficiency. For the sake of simplicity, we will restrict the following presentation of the algorithms to a single grid level and thus omit all level indices. However, due to the uniform reduction of the algebraic error, this scheme reveals its full potential when used as a smoother in a multilevel structure [13].

3.2 Preliminaries

We consider the system

$$Ax = b \ , \ \ A = (a_{i,j})_{1 \leq i,j \leq n} \ \ , \tag{1}$$

of linear equations and assume that the matrix A is sparse and symmetric positive definite. As a common example, standard second-order finite difference discretizations of the negative Laplacian yield such matrices. The exact solution of (1) shall be denoted as x^*.

We assume that x stands for an approximation to x^*. The *scaled residual* $\bar{r}$ corresponding to x is then defined as $\bar{r} := D^{-1}(b - Ax)$, where D denotes the diagonal part of A. With e_i representing the i-th unit vector, the components θ_i, of the scaled residual $\bar{r}$ are given by $\theta_i := e_i^T \bar{r}$, $1 \leq i \leq n$.

An *elementary relaxation step* for equation i of the linear system (1), which updates the approximation x and yields the new approximation x', is given by $x' = x + \theta_i e_i$, and the Gauss-Seidel update scheme [5] can be written as follows:

$$x_i^{(k+1)} = a_{i,i}^{-1} \left(b_i - \sum_{j<i} a_{i,j} x_j^{(k+1)} - \sum_{j>i} a_{i,j} x_j^{(k)} \right) = x_i^{(k)} + \theta_i \ .$$

Here, θ_i represents the current scaled residual of equation i; i.e., the scaled residual of equation i *immediately* before this elementary update operation. As a consequence of this elementary relaxation step, the i-th component of $\bar{r}$ vanishes.

The motivation of the development of the adaptive relaxation method is based on the following relation, which states how the algebraic error $x^* - x$ is reduced by performing a single elementary relaxation step:

$$||x^* - x||_A^2 - ||x^* - x'||_A^2 = a_{i,i}\theta_i^2 \ , \tag{2}$$

see [13]. As usual, $||v||_A$ stands for the energy norm of v.

Equation (2) implies that, with every elementary relaxation step, the approximation is either improved (if $\theta_i \neq 0$) or remains unchanged (if $\theta_i = 0$). Note that, since A is positive definite, all diagonal entries $a_{i,i}$ of A are positive. Hence, it is the positive definiteness of A which guarantees that the solution x will never become "worse" by applying elementary relaxation operations.

Equation (2) further represents the mathematical justification for the selection of the (numerically) most efficient equation update order. It states that the fastest error reduction is obtained by relaxing those equations i whose scaled residuals θ_i have relatively large absolute values. This is particularly exploited in the *method of Gauss-Southwell* where always the equation i with the largest value $|a_{i,i}\theta_i|$ is selected for the next elementary relaxation step. Yet, if no suitable update strategy is used, this algorithm will be rather inefficient since determining the equation whose residual has the largest absolute value is as expensive as relaxing each equation once [13].

This observation motivates the principle of the adaptive relaxation method which is based on an *active set* of equation indices and can be interpreted as an algorithmically efficient approximation to the method of Gauss-Southwell. For a concise description of the operations on the active set and the method of adaptive relaxation, it is convenient to introduce the *set of connections* of grid node i, $1 \leq i \leq n$, as

$$\mathrm{Conn}(i) := \{j; 1 \leq j \leq n, \ j \neq i, \ a_{j,i} \neq 0\} \ .$$

Intuitively, the set $\mathrm{Conn}(i)$ contains the indices of those unknowns u_j that directly depend on u_i, except for u_i itself. Note that, due to the symmetry of A, these are exactly the unknowns u_j, $j \neq i$, that u_i directly depends on.

Before we can provide an algorithmic formulation of the adaptive relaxation method, it is further necessary to introduce the *strictly active set* $S(\theta, x)$ of equation indices:

$$S(\theta, x) := \{i;\ 1 \leq i \leq n,\ |\theta_i| > \theta\} \ .$$

Hence, the set $S(\theta, x)$ contains the indices i of all equations whose current scaled residuals θ_i have absolute values greater than the prescribed threshold $\theta > 0$. Possible alternatives of this definition are discussed in [13].

Finally, an *active set* $\tilde{S}(\theta, x)$ is a superset of the strictly active set $S(\theta, x)$:

$$S(\theta, x) \subseteq \tilde{S}(\theta, x) \subseteq \{i;\ 1 \leq i \leq n\} \ .$$

Note that $\tilde{S}(\theta, x)$ may also contain indices of equations whose scaled residuals have relatively small absolute values. The idea behind the introduction of such an extended active set is that it can be maintained efficiently. This is exploited by the adaptive relaxation algorithms that we will present next.

3.3 Point-Based Adaptive Relaxation

The core of the *sequential (Gauss-Seidel-type) adaptive relaxation method*[1] is presented in Algorithm 1. Note that it is essential to initialize the active set $\tilde{S}$ appropriately. Unless problem-specific information is known a priori, $\tilde{S}$ is initialized with all indices i, $1 \leq i \leq n$.

The algorithm proceeds as follows. Elementary relaxation steps are performed until the active set $\tilde{S}$ is empty, which implies that none of the absolute values of the scaled residuals θ_i exceeds the given tolerance θ anymore. In Steps 2 and 3, an equation index $i \in \tilde{S}$ is selected and removed from $\tilde{S}$. Only if the absolute value of the corresponding scaled residual θ_i is larger than θ (Step 4), equation i will be relaxed. The relaxation of equation i implies that all scaled residuals θ_j,

[1] A *simultaneous (Jacobi-type)* adaptive relaxation scheme is also discussed in [13].

Algorithm 1 Sequential adaptive relaxation.

Require: tolerance $\theta > 0$, initial guess x, initial active set $\tilde{S} \subseteq \{i;\ 1 \leq i \leq n\}$
1: **while** $\tilde{S} \neq \emptyset$ **do**
2:    Pick $i \in \tilde{S}$ {Nondeterministic choice}
3:    $\tilde{S} \leftarrow \tilde{S} \setminus \{i\}$
4:    **if** $|\theta_i| > \theta$ **then**
5:       $x \leftarrow x + \theta_i e_i$ {Elementary relaxation step}
6:       $\tilde{S} \leftarrow \tilde{S} \cup \mathrm{Conn}(i)$
7:    **end if**
8: **end while**
Ensure: $S = \emptyset$ {The strictly active set S is empty}

$j \in \mathrm{Conn}(i)$, are changed, and it must therefore be checked subsequently if their absolute values $|\theta_j|$ now exceed the tolerance θ. Hence, if equation i is relaxed (Step 5), the indices $j \in \mathrm{Conn}(i)$ will be put into the active set $\tilde{S}$ (Step 6), unless they are already contained in $\tilde{S}$.

As was mentioned above, the full advantage of the adaptive relaxation scheme becomes evident as soon as it is introduced into a multigrid structure. The idea is to employ this method as a smoother in a multigrid setting, where it is not necessary to completely eliminate the errors on the finer grids. Instead, it is enough to smooth the algebraic errors such that reasonably accurate corrections can be computed efficiently on the respective coarser grids [14].

The application of the adaptive relaxation method in the multigrid context corresponds to the approach of *local relaxation*. The principle of local relaxation states that the robustness of multigrid algorithms can be improved significantly by performing additional relaxation steps in the vicinity of singularities or perturbations from boundary conditions, for example. The purpose of these additional elementary relaxation steps is to enforce a sufficiently smooth error such that the subsequent coarse-grid correction performs well. We refer to [13] for further details and especially to the references provided therein.

At this point, we eventually have everything in place to introduce our novel patch-adaptive relaxation scheme which can be considered cache-aware by construction.

3.4 Cache-Optimized Patch-Based Adaptive Relaxation

A *patch* can be interpreted as a frame (window) that specifies a region of a computational grid. For ease of presentation, we only consider non-overlapping patches. Our *sequential patch-based adaptive relaxation scheme* parallels the point-oriented version from Section 3.3. This implies that, when introduced into a multigrid structure, it also follows the aforementioned principle of local relaxation and, in general, behaves in a more robust fashion than standard multigrid algorithms.

In the case of patch-adaptive relaxation, however, the active set $\tilde{S}$ does not contain the indices of individual equations (i.e., the indices of individual grid nodes). Instead, $\tilde{S}$ contains the indices of patches. The set of all patches is denoted as $\mathcal{P}$. Since the number of patches is usually much smaller than the number of unknowns, the granularity of the adaptive relaxation scheme and, in addition, the overhead of maintaining the active set during the iteration are both reduced significantly.

Furthermore, the patch-adaptive relaxation is characterized by its inherent data locality. Consequently, it leads to a high utilization of the cache and therefore to fast code execution. This is accomplished through the use of appropriately sized patches as well as through the repeated relaxation of a patch once it has been loaded into cache. It has been demonstrated that, once an appropriately sized patch has been loaded into the cache and updated for the first time, the costs of subsequently relaxing it a few more times are relatively low [10]. See [7] for details.

The core of the sequential patch-adaptive relaxation scheme is presented in Algorithm 2[2]. Upon termination, the strictly active set S is empty and, therefore, each scaled residual θ_i is less or equal than the prescribed threshold θ, $1 \leq i \leq n$. In Step 4, $\bar{r}_P$ is used to represent the scaled residual corresponding to patch P (i.e., to the nodes covered by P) and $||\bar{r}_P||_\infty$ denotes its maximum norm.

Relaxing a patch (Step 5) means performing Gauss-Seidel steps on its nodes until each of its own scaled residuals θ_i has fallen below the (reduced) tolerance $\theta - \delta$. As a consequence, after a patch P has been relaxed any of its neighbor patches P' will only be inserted into the active set if the absolute values of the scaled residuals of P' have been increased "significantly". This allows for the application of sophisticated activation strategies in Step 6, which aim at avoiding unnecessary patch relaxations. A comprehensive discussion of such strategies is beyond the scope of this paper. Instead, we again point to [7].

4 Experimental Results

Performance results that demonstrate the effectiveness of our data layout transformations as well as our data access transformations have extensively been studied in [15] for the 2D case and in [7] for the 3D case. Therefore, this section will concentrate on the efficiency of the patch-adaptive multigrid scheme.

4.1 Model Problem

We define an L-shaped domain $\Omega := \{(x, y) \in (-0.5, 0.5)^2;\ x < 0 \text{ or } y > 0\} \subset \mathbf{R}^2$ and introduce 2D polar coordinates (r, φ), $r \geq 0$, $0 \leq \varphi \leq 2\pi$, as usual; i.e., we have $x = r\cos\varphi$ and $y = r\sin\varphi$. We consider the Dirichlet boundary value problem

$$-\Delta u = 0 \quad \text{in} \quad \Omega \ , \tag{3}$$

[2] As is the case for the point-based method, a simultaneous version of the patch-based scheme is also conceivable.

Algorithm 2 Sequential patch-adaptive relaxation.

Require: tolerance $\theta > 0$, tolerance decrement $\delta = \delta(\theta)$, $0 < \delta < \theta$, initial guess x, initial active set $\tilde{S} \subseteq \mathcal{P}$
1: **while** $\tilde{S} \neq \emptyset$ **do**
2:   Pick $P \in \tilde{S}$ {Nondeterministic choice}
3:   $\tilde{S} \leftarrow \tilde{S} \setminus \{P\}$
4:   **if** $||\bar{r}_P||_\infty > \theta - \delta$ **then**
5:     relax(P)
6:     activateNeighbors(P)
7:   **end if**
8: **end while**
Ensure: $S = \emptyset$ {The strictly active set S is empty}

$$u(r,\varphi) = \sin\left(\frac{2}{3}\varphi\right) r^{\frac{2}{3}} \quad \text{on} \quad \partial\Omega \ . \tag{4}$$

It is straightforward to verify that the analytical solution of (3), (4) is given by $u(r,\varphi) := \sin(\frac{2}{3}\varphi)r^{2/3}$ and that we have $u \in C^2(\Omega) \cap C^0(\bar{\Omega})$, but $u \notin C^1(\bar{\Omega})$. This singular behavior of u results from the fact that the first derivatives of u are not bounded in $r = 0$, cf. [4].

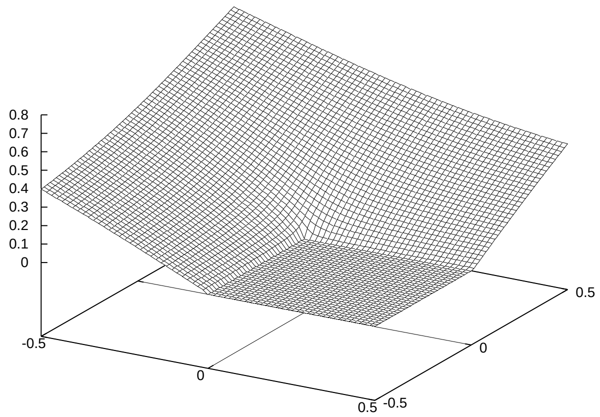

Fig. 1. Analytical solution of the model problem.

Figure 1 shows a plot of u using a grid with a mesh width of 64^{-1}. In the following, we will demonstrate how this model problem can be solved efficiently using our patch-adaptive multigrid scheme.

4.2 Numerical Tests

We have employed a hierarchy of nine regular grids with a finest grid of mesh width of 1024^{-1}. Standard coarsening has been used. The Laplacian has been discretized anew on each level using bilinear basis functions on the quadrilateral finite elements. This discretization approach leads to constant 9-pont stencils on each grid level.

We have determined the total numbers of elementary relaxations steps per grid level for a multigrid V(0,4)-cycle involving a red-black Gauss-Seidel (RBGS) smoother. Additionally, we have determined the corresponding numbers for the patch-adaptive multigrid scheme. The patch-adaptive relaxation scheme from Section 3.4 has been applied to smooth the errors on the five finest levels, using a patch size of approximately 32×32 nodes. On the four coarser levels, a standard red-black Gauss-Seidel smoother has been employed. This is because it is only reasonable to apply the patch-adaptive scheme if the number of unknowns on the corresponding level is large enough such that it pays off to implement a smoother based on an active set strategy.

Level	♯unknowns	RBGS-based MG	Patch-adapt. MG	Patch-adapt. MG
				♯residual comp.
		♯elem. rel. steps	♯elem. rel. steps	+♯elem. rel. steps
0	5	20	20	20
1	33	132	132	132
2	161	644	644	644
3	705	2 820	2 820	2 820
4	2 945	11 780	54 932	57 877
5	12 033	48 132	44 672	56 705
6	48 641	194 564	44 672	93 313
7	195 585	782 340	48 640	244 225
8	784 385	3 137 540	56 832	841 217
Total	1 044 493	4 177 972	253 364	1 296 953

Fig. 2. Comparison of standard and patch-adaptive multigrid methods.

The results of the comparison are summarized in Table 2. They clearly emphasize the numerical efficiency of the multigrid method involving our patch-adaptive smoother. It is obvious that the patch-adaptive multigrid algorithm requires far less computational work than the standard one, even when counting all necessary residual calculations in addition to the actual number of elementary relaxation steps, cf. again Algorithm 2 from Section 3.4. This is due to the fact that numerical schemes based on adaptive relaxation generally represent good candidates for solving such problems that exhibit singularities [13]. Their applicability to "smoother" problems, however, is subject to future research.

Moreover, the RBGS-based multigrid method exhibits an average *residual reduction factor* of 0.22 per V-cycle[3]. Therefore, we have prescribed the tolerance $\theta := 0.22$ for the patch-adaptive smoother on each respective grid level, including the finest one which is typically considered solely for convergence rate measurements. The tolerance decrement δ has been chosen empirically as $\delta := 0.2\theta$. The patch-adaptive multigrid scheme has actually exhibited a significantly better residual reduction factor of 0.063. This observation again emphasizes the enhanced numerical efficiency of the patch-adaptive scheme, cf. Section 3.3.

A detailed analysis of the behavior of the patch-adaptive multigrid scheme reveals that, on the coarsest level on which the patch-adaptive smoother is used, more elementary relaxation steps are performed than in the case of the standard smoother. However, this additional work pays off on the finer levels, where only those patches in the vicinity of the singularity at $r = 0$ are relaxed more intensively. We again refer to [7] for a comprehensive discussion.

Preliminary performance experiments have indicated the improved cache efficiency of the patch-adaptive relaxation scheme. However, the development of highly tuned implementations which exhibit high floating-point performance (in terms of MFLOPS rates) is still in progress.

[3] This means that we have divided the maximum norms of the scaled residuals on the finest grid level before and after each V-cycle.

5 Conclusions

The continued improvements in processor performance are generally placing increasing pressure on the memory hierarchy. Therefore, we have developed and examined optimization techniques in order to enhance the data locality exhibited by grid-based numerical computations, particularly the time-consuming smoothers in multigrid algorithms.

We have presented a promising approach towards the design of novel, inherently cache-aware algorithms and, so far, we have demonstrated its enhanced numerical robustness. The development of highly tuned hardware-aware implementations, however, is the focus of ongoing and future work.

References

1. R. Allen and K. Kennedy. *Optimizing Compilers for Modern Architectures.* Morgan Kaufmann, 2001.
2. 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, 2000.
3. S. Goedecker and A. Hoisie. *Performance Optimization of Numerically Intensive Codes.* SIAM, 2001.
4. W. Hackbusch. *Elliptic Differential Equations — Theory and Numerical Treatment*, volume 18 of *Springer Series in Computational Mathematics.* Springer, 1992.
5. W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*, volume 95 of *Applied Mathematical Sciences.* Springer, 1993.
6. J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 3. edition, 2003.
7. M. Kowarschik. *Data Locality Optimizations for Iterative Numerical Algorithms and Cellular Automata on Hierarchical Memory Architectures.* PhD thesis, Lehrstuhl für Informatik 10 (Systemsimulation), Institut für Informatik, Universität Erlangen-Nürnberg, Erlangen, Germany, July 2004. SCS Publishing House.
8. M. Kowarschik and C. Weiß. An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms. In U. Meyer, P. Sanders, and J. Sibeyn, editors, *Algorithms for Memory Hierarchies — Advanced Lectures*, volume 2625 of *Lecture Notes in Computer Science*, pages 213–232. Springer, 2003.
9. D. Loshin. *Efficient Memory Programming.* McGraw-Hill, 1998.
10. H. Lötzbeyer and U. Rüde. Patch-Adaptive Multilevel Iteration. *BIT*, 37(3):739–758, 1997.
11. D. Quinlan, M. Schordan, B. Miller, and M. Kowarschik. Parallel Object-Oriented Framework Optimization. *Concurrency and Computation: Practice and Experience*, 16(2–3):293–302, 2004. Special Issue: Compilers for Parallel Computers.
12. G. Rivera and C.-W. Tseng. Tiling Optimizations for 3D Scientific Computations. In *Proc. of the ACM/IEEE Supercomputing Conf.*, Dallas, Texas, USA, 2000.
13. U. Rüde. *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, volume 13 of *Frontiers in Applied Mathematics.* SIAM, 1993.
14. U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid.* Academic Press, 2001.
15. C. Weiß. *Data Locality Optimizations for Multigrid Methods on Structured Grids.* PhD thesis, Lehrstuhl für Rechnertechnik und Rechnerorganisation, Institut für Informatik, Technische Universität München, Munich, Germany, December 2001.