# Status report: HLRB project *h048z* — Data–local iterative methods (DiME)*


Markus Kowarschik[1], Ulrich Rüde[1], and Christian Weiß[2]

[1] Lehrstuhl für Systemsimulation (Informatik 10), Institut für Informatik, Universität Erlangen–Nürnberg, Germany

[2] Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR-TUM), Fakultät für Informatik, Technische Universität München, Germany



**Abstract.** In order to mitigate the effect of the growing gap between CPU speed and main memory performance current computer architectures use hierarchical memory designs comprising one or even more levels of cache memory. This is especially true for the Hitachi SR8000-F1, which is based on a Power3–like processor using a 128 kByte on–chip L1 cache. The general goal of our project is to investigate how numerical algorithms can be designed and/or restructured such that the hierarchical memory structure of the underlying architecture is respected. In particular we focus on the processing of algorithms on regular meshes.
Our research efforts concerning code performance optimizations on the Hitachi architecture yield techniques which are immediately applied by the KONWIHR projects *gridlib* as well as *ParEXPDE*.


## 1 Introduction

There is no doubt about the fact that the speed of computer 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 that is 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 [5, 4]. 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 code, especially by the order of memory accesses [9].

The Hitachi SR8000–F1 supercomputer is based on Power3–like CPUs, each of which uses a 128 kByte L1 cache which is located on–chip. This cache is


* This research is being supported in part by the *Deutsche Forschungsgemeinschaft* (German Science Foundation), projects Ru 422/7–1,2,3.

4–way set–associative. Despite the memory bandwidth of the machine being significantly higher than the memory bandwidths of current (RISC–based) workstations, cache effects can be observed.

The goal of this project is to investigate in how far our data locality optimization techniques for iterative numerical schemes — which have primarily been developed for standard cache–based architectures — carry over to the Hitachi machine [14, 15, 3, 17, 16]. More general surveys on our research activities can be found in [1, 2].

There are two KONWIHR projects dealing with iterative numerical schemes on highly regular grid structures and thus immediately benefit from our project's results. The first one is the *gridlib* project, the second one is the *ParEXPDE* project.

## 2 Preliminary results

### 2.1 Blocked Gauss–Seidel code on regular grids

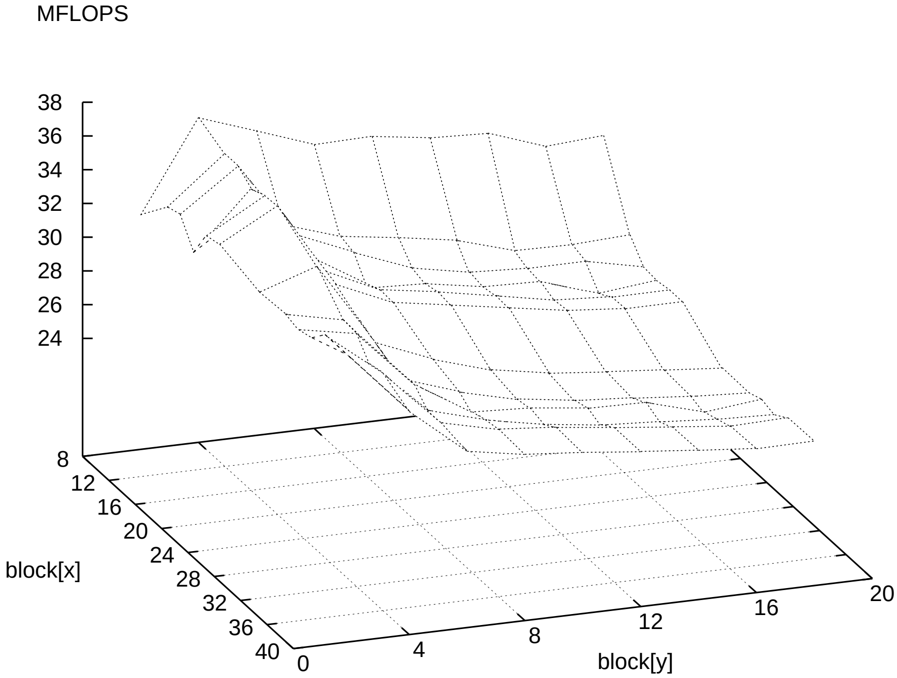
This section describes performance results for a 2D blocked Gauss–Seidel code with a red/black ordering of the unknowns on structured rectangular grids, using 5–point stencils to discretize the continuous operators. A description of our data layout optimization and data access optimization techniques can be found in [11] and in [8]. These experiments are based on the use of suitable array paddings [12].

Figure 1 shows the MFLOPS rates for a grid with 1025 nodes in each direction, yielding about one million unknowns. This corresponds to a size of roughly 56 MByte. For the measurements a single node was reserved exclusively and the computations were ran sequentially. Four Gauss–Seidel iterations were blocked in this experiment. The idea behind this experiment was to quantify the performance of a single Hitachi SR8000 processor.

The upper figure shows the results for the matrix being stored in a *band–wise manner (compressed diagonal storage scheme)*, whereas the lower figure corresponds to our *access–oriented storage scheme*. See again [11] and [8] for details on data layout optimizations for iterative numerical methods on structured meshes.

The upper figure illustrates that the MFLOPS rate decreased with the block growing in x–direction. However, if the y–dimension of the block is chosen too large the MFLOPS rate also drops. Only for small blocks in x–direction the MFLOPS rate nearly remains constant when the y–dimension of the block is varied.

Comparing the two figures reveals that the access–oriented storage schemes yields higher MFLOPS rates in general. However, for large block sizes the performance drops dramatically for both data layouts. The reason for this is that the working set is too large to fit in cache. Thus, the cache cannot be used efficiently. Nevertheless the single–CPU performance is even low for the best block dimensions. This might be due to cache thrashing effects. Due to instabilities of the C compiler on the Hitachi SR8000 further experiments have not been done yet.



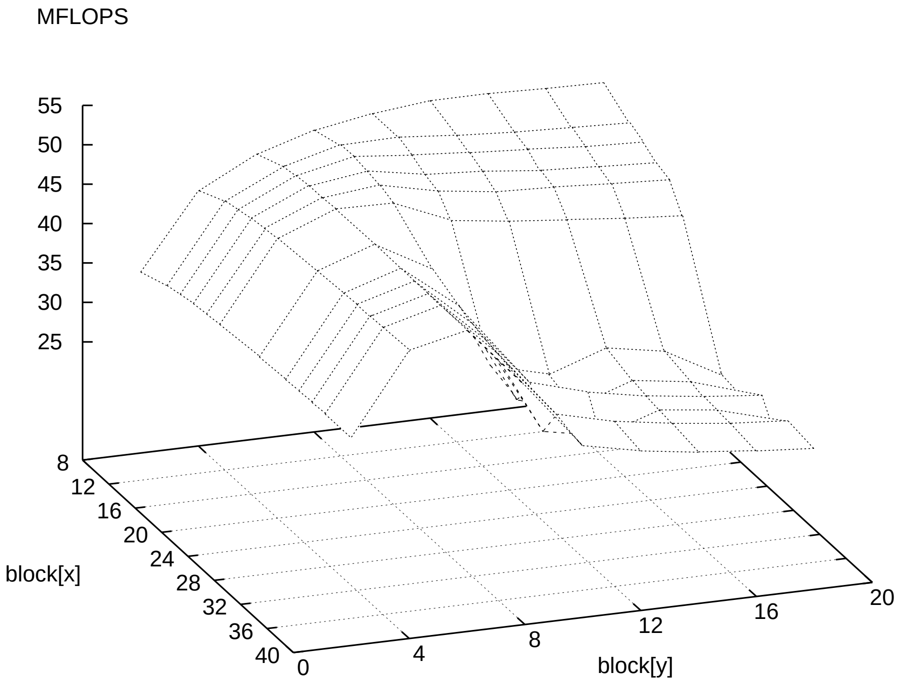
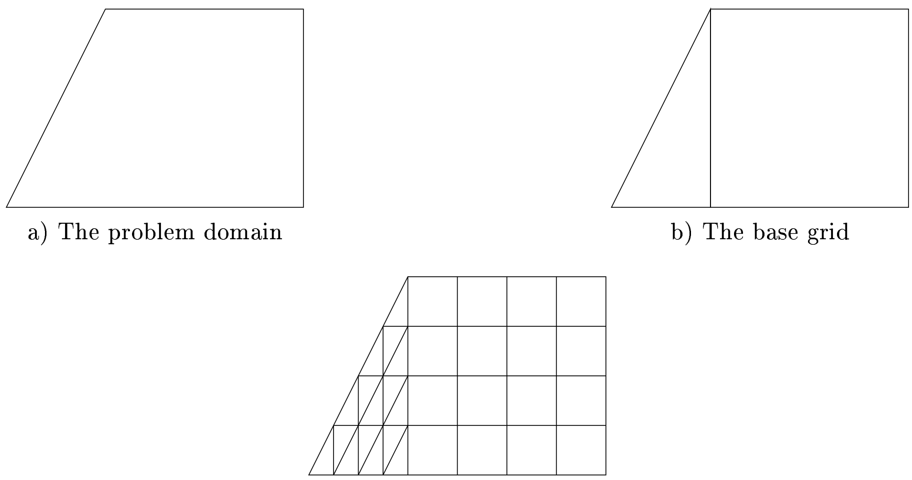




**Fig. 1.** The MFLOPS rates for the Hitachi SR8000. The 2D blocking method is used with different block sizes in x– and y–direction. The upper figure corresponds to the band–wise storage scheme and the lower figure refers to the access–oriented storage scheme.

## 2.2 Hierarchical hybrid grids

The concept of hierarchical hybrid grids combines geometrical flexibility with high efficiency by using different grid levels. The task of the coarse level is to represent the problem geometry and topology adequately, hence we expect it to be unstructured in general. The computations are carried out on a finer level, the so-called *compute grid*. Consequently, the main requirement for the compute grid is to facilitate high performance computations, a condition which is met by construction. The compute grid is generated through regular subdivision of the elements of the geometry grid. This process yields a patch-wise structured grid on the fine level that inherits the quality of the geometry representation from the base grid. The idea is now to refine the patches so much that the overall performance is dominated by computations in structured parts.

In the following, we demonstrate the efficiency of the core concept. The appropriate way to exploit the structured patches depends on the algorithm and the hardware. We focus on the Gauss-Seidel iteration as it is a versatile smoother in a multigrid context and report our results on the Hitachi SR8000 at the Leibniz computing center in Munich. However, as mentioned before, not only supercomputers will benefit from hybrid grids. On workstations or clusters thereof, the implementation can draw from the cache-optimization techniques described in [17], which apply directly to the structured grids in the patches.



a) The problem domain

b) The base grid

c) The compute grid after two refinement steps

**Fig. 2.** The problem domain and the various grids for the Gauss-Seidel example.

Our model problem arises from the discretization of the Laplace operator with varying coefficients by (bi-)linear finite elements on the problem domain in figure 2 a). The unstructured base grid is shown in figure 2 b) from which the compute grid in figure 2 c) is derived with two regular subdivision steps. The differential operator is only important to determine the stencil at a vertex. In the triangle, the Gauss-Seidel update involves all six neighbors of a vertex. Likewise in the rectangle, all eight neighbors are accessed. The compute grid is derived by 10 uniform refinement steps from the base grid. For the rectangle, this leads to $1024 \times 1024$ interior vertices.

| Best performance in MFLOPS | | | |
|---|---|---|---|
| unstruct. | hybrid | | |
| | RB | Zebra 1D | Zebra 2D |
| 520 | 2000 | 2300 | 3900 |

**Table 1.** RB: Red-Black ordering, Zebra: Zebra smoother

The performance results for various implementations of the Gauss-Seidel method are summarized in Table 1. The MFLOPS values are the sum of the performance of the eight CPUs in one node. The unstructured version works on sets of unknowns that can be updated simultaneously which are then processed by the eight CPUs in parallel. Three patch-wise oriented versions were tested: a red-black smoother, a zebra smoother on one-dimensional data structures and a zebra smoother on two-dimensional data structures. The zebra smoothers are based on a red-black coloring of *lines* of unknowns instead of individual points as in the standard red-black smoother. First, all red lines are updated, before the black lines are processed. This numbering has the advantage that a complete line of 1024 unknowns can be assigned to one CPU, which obviously makes better use of the vector processing facilities of the Hitachi than the pointwise red-black ordering. The performance gain of the hybrid grid variants over the unstructured implementation is significant and demonstrates amply the potential of the framework. Further details can be found in [6].

### 2.3 Porting the DiMEPACK Library

The DiMEPACK library [7] is the result of a joint effort of the Lehrstuhl für Informatik 10, University Erlangen–Nuremberg, Germany, and the Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR–TUM), Technische Universität München, Germany.

DiMEPACK is a library of cache–aware multigrid routines. 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. So far, DiMEPACK can handle constant–coefficient problems based on discretizations using 5–point or 9–point stencils.

As part of the HLRB project *h048z* the DiMEPACK library has recently been ported to the Hitachi SR8000 machine. Performance evaluations of the DiMEPACK library on the SR8000 are part of ongoing work.

## 3 Conclusions and future work

Our research focuses on optimizations of iterative numerical methods on cache–based architectures. We have demonstrated that our techniques particularly effect the performance that is achieved on the Hitachi SR8000 machine. This is mainly due to the fact that the SR8000 is based on a Power3–like CPU with a 128 kByte L1 cache.

Our previous work has led to several tutorials on cache–based algorithms:

- 10th Copper Mountain Conference on Multigrid Methods, Copper Mountain, CO, USA, Apr. 2001
- ACM/IEEE Supercomputing 2000, Denver, CO. USA, Nov. 2001
- LRZ Munich, Mar. 2002, see the corresponding LRZ web page: *http://www.lrz-muenchen.de/services/compute/hlrb/kurse*

More information about our research activities can be found on our project web page

$$\boxed{\textit{http://wwwbode/in.tum.de/Par/arch/cache}}$$

We will complete the performance evaluation of the DiMEPACK multigrid library. Our future work will then concentrate on cache performance optimizations for regular mesh structures in three dimensions.

Furthermore, we are currently investigating concepts for inherently data–local algorithms which — by construction — respect the hierarchical memory architecture of the machines under consideration. In this context, we are planning to investigate the behavior of patch-adaptive multigrid algorithms [10, 13] in larger–scale settings on the Hitachi SR8000 machine.

Besides, in order to further support the *gridlib* and the *ParEXPDE* projects it is inevitable that we keep our accounts on the Hitachi SR8000 machine.

## References

1. C. Douglas, G. Haase, J. Hu, M. Kowarschik, U. Rüde, and C. Weiss, *Portable Memory Hierarchy Techniques For PDE Solvers: Part I*, Siam News, 33 (2000).
2. ———, *Portable Memory Hierarchy Techniques For PDE Solvers: Part II*, Siam News, 33 (2000).
3. 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 (2000), pp. 21–40.
4. J. Handy, *The Cache Memory Book*, Academic Press, second ed., 1998.

5. J. L. Hennessy and D. A. Patterson, *Computer Architecture — A Quantitative Approach*, Morgan Kaufmann Publishers, second ed., 1996.
6. F. Hülsemann, P. Kipfer, U. Rüde, and G. Greiner, gridlib: *Flexible and Efficient Grid Management for Simulation and Visualization*, in Proceedings of the 2002 International Conference on Computational Science, Lecture Notes in Computer Science, Amsterdam, The Netherlands, April 21-24 2002, 2002, Springer. to appear.
7. M. Kowarschik and C. Weiss, *DiMEPACK — A Cache–Optimized Multigrid Library*, in Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001), H. R. Arabnia, ed., vol. I, Las Vegas, NV, USA, 2001, CSREA, CSREA Press.
8. M. Kowarschik, C. Weiss, and U. Rüde, *Data Layout Optimizations for Variable Coefficient Multigrid*, in Proceedings of the 2002 International Conference on Computational Science, Lecture Notes in Computer Science, Amsterdam, The Netherlands, April 21-24 2002, 2002, Springer. to appear.
9. D. Loshin, *Efficient Memory Programming*, McGraw–Hill, 1998.
10. H. Lötzbeyer and U. Rüde, *Patch-Adaptive Multilevel Iteration*, BIT, 37 (1997), pp. 739–758.
11. H. Pfänder, *Cache–optimierte Mehrgitterverfahren mit variablen Koeffizienten auf strukturierten Gittern*, Master's thesis, Department of Computer Science, University of Erlangen–Nuremberg Germany, 2000.
12. G. Rivera and C.-W. Tseng, *Data Transformations for Eliminating Conflict Misses*, in Proceedings of the 1998 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'98), Montreal, Canada, June 1998.
13. U. Rüde, *Fully Adaptive Multigrid Methods*, SIAM J. Numer. Analysis, 30 (1993), pp. 230–248.
14. U. Rüde, *Iterative Algorithms on High Performance Architectures*, in Proceedings of the EuroPar97 Conference, Lecture Notes in Computer Science, Springer, Aug. 1997, pp. 26–29.
15. U. Rüde, *Technological Trends and their Impact on the Future of Supercomputers*, in High Performance Scientific and Engineering Computing, Proceedings of the International FORTWIHR Conference on HPSEC, H.-J. Bungartz, F. Durst, and C. Zenger, eds., vol. 8 of Lecture Notes in Computational Science and Engineering, Springer, Mar. 1998, pp. 459–471.
16. 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, Dec. 2001.
17. C. Weiss, W. Karl, M. Kowarschik, and U. Rüde, *Memory Characteristics of Iterative Methods*, in Proceedings of the ACM/IEEE SC99 Conference, Portland, Oregon, Nov. 1999.