# Analyzing Advanced PDE Solvers Through Simulation

Henrik Johansson, Dan Wallin, and Sverker Holmgren

Department of Information Technology, Uppsala University
Box 337, S-751 05 Uppsala, Sweden
{henrik.johansson,dan.wallin,sverker.holmgren}@it.uu.se

**Abstract.** By simulating a real computer it is possible to gain a detailed knowledge of the cache memory utilization of an application, e.g., a partial differential equation (PDE) solver. Using this knowledge, we can discover regions with intricate cache memory performance. Furthermore, this information makes it possible to identify performance bottlenecks.
In this paper, we employ full system simulation of a shared memory computer to perform a case study of three different PDE solver kernels with respect to cache memory performance. The kernels implement state-of-the-art solution algorithms for complex application problems and the simulations are performed for data sets of realistic size. We discovered interesting properties in the solvers, which can help us to improve their performance in the future.

## 1 Introduction

The performance of PDE solvers depends on many computer architecture properties. The ability to efficiently use cache memory is one such property. To analyze cache behavior, hardware counters can be used to gather rudimentary data like cache hit rate. However, such experiments can not identify the reason for a cache miss. By software simulation of a computer, it is possible to derive, e.g., the type of a cache miss and the amount of address and data traffic. A further advantage is that the simulated computer can be a fictitious system.

In this paper, we present a case study of the cache memory behavior of three PDE solvers. The solvers are based on modern, efficient algorithms, and the settings and working sets are carefully chosen to represent realistic application problems. The study is based on full-system simulations of a shared memory multiprocessor, where the baseline computer model is set up to correspond to a commercially available system, the SunFire 6800 server. However, using a simulator, the model can easily be modified to resemble alternative design choices.

We perform simulations where both the cache size and the cache line size is varied, as two simulation case studies. Many other parameters can be varied as well, but the chosen parameters show some typical properties of the studied PDE solvers.

We begin this paper with a general discussion of simulation, coupled with a description of our simulation platform (section two). Next, we present the PDE solvers used for the simulations (section three). The fourth section holds the results of the simulations, divided into two case studies. Finally, our conclusions are presented in section five.

J. Dongarra, K. Madsen, and J. Waśniewski (Eds.): PARA 2004, LNCS 3732, pp. 893–900, 2006.
© Springer-Verlag Berlin Heidelberg 2006

## 2 Related Work

Early work in full system simulation was performed at Stanford University and resulted in the SimOS simulator [2]. The SimOS was, among others, used to study the memory usage of commericial workloads [3], data locality on CC-NUMA systems [4] and to characterize and optimize an auto-parallelizing compiler [5]. However, the work on SimOS was discontinued in 1998.

Most recent work in the field is based on the Simics full system simulator [1]. Simics has been used in a large number of publications. These include an examination of the memory system behavior for Java-based middleware [6] and a comparision of memory system behavior in java and non-java commercial workloads [7].

The work presented above is primarily concerned with hardware. In the papers real applications are simulated with the goal of improving the underlying hardware. Our paper uses the opposite approach, it uses simulation in an effort to improve the software.

## 3 Simulation Techniques

There are several ways to analyze algorithms on complex high performance computer systems, e.g., a high-end server. One alternative is to perform measurements on the actual physical machine, either using hardware or software. Another alternative is to develop analytical or statistical models of the system. Finally, there is simulation.

Simulation has several advantages over the other methods. First, the characteristics of the simulated computer can be set almost arbitrary. Second, the simulated machine is observable and controllable at all times. Third, a simulation is deterministic; it is always possible to replicate the results obtained from the simulation. Fourth, a high degree of automation is possible.

The ability to change parameters in the area of interest gives a researcher great flexibility. Looking at cache memory, we can easily change cache line size, cache size, replacement policy, associativity and use sub-blocked or unblocked caches. We can also determine which information to collect. In this case it can, e.g., be the number of memory accesses, cache miss rates, cache miss type and the amount of data or address traffic.

There are also a few drawbacks inherent in simulation. It is in practice impossible to build a simulator that mimics the exact behavior of the target computer in every possible situation. This might lead to some differences between the collected information and the performance of the real computer. However, comparisons between the individual simulations are valid since they all encounter the same discrepancies. Estimating execution time is difficult based on simulation and it is highly implementation dependent on the bandwidth assumptions for the memory hierarchy and the bandwidth of coherence broadcast and data switches. The complexity of an "exact" simulation is thus overwhelming, and performance and implementation issues make approximations necessary.

The experiments are carried out using execution-driven simulation in the Simics full-system simulator [1]. Simics can simulate computer architectures at the level of individual instructions. The target computer runs unmodified programs and even a complete operating system. Usually an operating system is booted inside Simics and programs, benchmarks and tests are run on this operating system in a normal fashion. Simics

collects detailed statistics about the target computer and the software it runs. The user determines what information to collect, and how to do it. For example, the user can initially collect a wide range of data about the cache memory system as a whole. Later, in order to isolate a certain behavior, it is possible to restrict the collection to a small region of the code.

The modeled system implements a snoop-based invalidation MOSI cache coherence protocol [8]. We set up the baseline cache hierarchy to resemble a SunFire 6800 server with 16 processors. The server uses UltraSPARC III processors, each equipped with two levels of caches. The processors have two separate first level caches, a 4-way associative 32 KB instruction cache and a 4-way associative 64 KB data cache. The second level cache is a shared 2-way associative cache of 8 MB and the cache line size is 64 B.

## 4 The PDE Solvers

The kernels studied represent three types of PDE solvers, used for compressible flow computations in computational fluid dynamics (CFD), radar cross-section computations in computational electro-magnetics (CEM) and chemical reaction rate computations in quantum dynamics (QD). The properties of the kernels differ with respect to the amount of computations performed per memory access, memory access patterns and the amount of communication and its patterns.

The CFD kernel implements the advanced algorithm described by Jameson and Caughey for solving the compressible Euler equations on a 3D structured grid using a Gauss-Seidel-Newton technique combined with multi-grid acceleration [9]. The implementation is described in detail by Nordén [10]. The data is highly structured. The total work is dominated by local computations needed to update each cell. These computations are quite involved, but their parallelization is trivial. Each of the threads computes the updates for a slice of each grid in the multi-grid scheme. The amount of communication is small and the threads only communicate pair-wise.

The CEM kernel is part of an industrial solver for determining the radar cross section of an object [11]. The solver utilizes an unstructured grid in three dimensions in combination with a finite element discretization. The resulting large system of equations is solved with a version of the conjugate gradient method. In each conjugate gradient iteration, the dominating operation is a matrix-vector multiplication with the very sparse and unstructured coefficient matrix. Here, the parallelization is performed such that each thread computes a block of entries in the result vector. The effect is that the data vector is accessed in a seemingly random way. However, the memory access pattern does not change between the iterations.

The QD kernel is derived from an application where the dynamics of chemical reactions is studied using a quantum mechanical model with three degrees of freedom [12]. The solver utilizes a pseudo-spectral discretization in the two first dimensions and a finite difference scheme in the third direction. In time, an explicit ODE-solver is used. For computing the derivatives in the pseudo-spectral discretization, a standard convolution technique involving 2D fast Fourier transforms (FFTs) is applied. The parallelization is performed such that the FFTs in the x-y dimensions are performed in parallel and locally [13]. The communication within the kernel is concentrated to a transpose opera-

tion, which involves heavy all-to-all communication between the threads. However, the communication pattern is constant between the iterations in the time loop.

# 5 Results

We will now present two case studies performed on the PDE solvers. In the first study we simulate the solvers with different sizes of the cache memory, varied between 512 KB and 16 MB. The cache line size is held constant at 64 B. This gives a good overview of the different characteristics exhibited by the solvers. In the second study, we fix the cache memory size and vary the cache line size between 32 B and 1024 B. These simulations provide us with detailed knowledge of the cache memory utilization for each of the three solvers. The case studies should be seen as possible studies to perform in a simulated environment. Other interesting studies could for example be to vary the cache associativity or the replacement policy [14,15].

We only perform a small number of iterative steps for each PDE kernel since the access pattern is regular within each iteration. Before starting the measurements, each solver completes a full iteration to warm-up the caches. The CFD problem has a grid size of $32\times32\times32$ elements using four multi-grid levels. The CEM problem represents a modeled generic aircraft with a problem coefficient matrix of about $175{,}000\times175{,}000$ elements; a little more than 300,000 of these are non-zero. The QD problem size is $256\times256\times20$, i.e. a $256\times256$ 2D FFT in the x-y plane followed by a FDM with 20 grid points in the z-direction. The wall clock time needed to perform the the simulations ranged from six up to twelve hours.

The cache misses are categorized into five different cache miss types according to Eggers and Jeremiassen [16]. *Cold* cache misses occur when data is accessed for the first time. A*capacity* cache miss is encountered when the cache memory is full. We also treat *conflict* misses, when data is mapped to an already occupied location in the cache, as capacity misses. When shared data is modified, it can result in a *true* sharing miss when another processor later needs the modified data. A *false* sharing cache miss results when a cache line is invalidated in order to maintain coherency, but no data was actually shared. *Updates* are not cache misses, but the messages generated in order to invalidate a shared cache line. An update can result in a sharing miss. The *data traffic* is a measurement of the number of bytes transferred on the interconnect, while the term *address traffic* represents the number of snoops the caches have to perform. All results presented are from the second level cache.

## 5.1 Case Study 1: Varying the Cache Size

The CFD-problem has the lowest miss rate of all the kernels. There is a relatively large amount of capacity misses when the cache size is small. This is expected, as the problem sizes for all of the applications are chosen to represent realistic work loads. There is a drop in the miss rate at a cache size of 1 MB. The drop corresponds to that all of the data used to interpolate the solution to the finest grid fits in the cache.

The CEM solver has a higher miss rate than the other applications. The miss rate decreases rapidly with larger cache sizes, especially between 2 MB and 4 MB. This

indicates that the vector used in the matrix-vector multiplication now fits into the cache. The large drop in miss rate between 8 MB and 16 MB tells us that cache now is large enough to hold all the data. Some of the capacity misses become true sharing misses when the size is increased.

The QD-solver has a miss rate in-between the two other solvers. The miss rate is stable and constant until the whole problem fits into the cache. Most of the capacity misses have then been transformed into true sharing misses because of the classification scheme; the classification of capacity misses take precedence over the true sharing misses.

The distribution of the different miss types varies between the three solvers. All of the solvers have a lot of capacity misses when the cache size is small. The capacity misses tend to disappear when the cache size is large enough to hold the full problem. The CFD-kernel exhibits large percentage of false sharing misses. The CEM-solver is dominated by true sharing misses, while the QD-solver has equal quantities of upgrades and true sharing misses for large cache sizes. For all solvers, both data and address traffic follow the general trends seen in the miss rate.

## 5.2 Case Study 2: Varying the Cache Line Size

The CFD kernel performs most of the computations locally within each cell in the grid, leading to a low overall miss ratio. The data causing the true sharing misses and the upgrades exhibit good spatial locality. However, the true and false sharing misses decrease more slowly since they cannot be reduced below a certain level. The reduction in address traffic is proportional to the decrease in miss ratio. The decrease in cache miss ratio is influenced by a remarkable property: the false sharing misses are reduced when the line size is increased. The behavior of false sharing is normally the opposite; false sharing misses increase with larger line size. Data shared over processor boundaries are always, but incorrectly, invalidated. If a shorter cache line is used, less invalidated data will be brought into the local cache after a cache miss and accordingly, a larger number of accesses is required to bring all the requested data to the cache [14]. It is probable that this property can be exploited to improve the cache memory utilization.

The CEM kernel has a large problem size, causing a high miss ratio for small cache line sizes. Capacity misses are common and can be avoided using a large cache line size. The true sharing misses and the upgrades are also reduced with a larger cache line size, but at a slower rate since the data vector is being irregular accessed. False sharing is never a problem in this application. Both the data and address traffic exhibit nice properties for large cache lines because of the rapid decrease in the miss ratio with increased line size.

The QD kernel is heavily dominated by two miss types, capacity misses and upgrades, which both decrease with enlarged cache line size. The large number of upgrades is caused by the all-to-all communication pattern during the transpose operation where every element is modified after being read. This should result in an equal number of true sharing misses and upgrades, but the kernel problem size is very large and replacements take place before the true sharing misses occur. Therefore, a roughly equal amount of capacity misses and upgrades are recorded. The data traffic increases rather slowly for cache lines below 512 B. The address traffic shows the opposite behavior and decreases rapidly until the 256 B cache line size is reached, where it levels out.

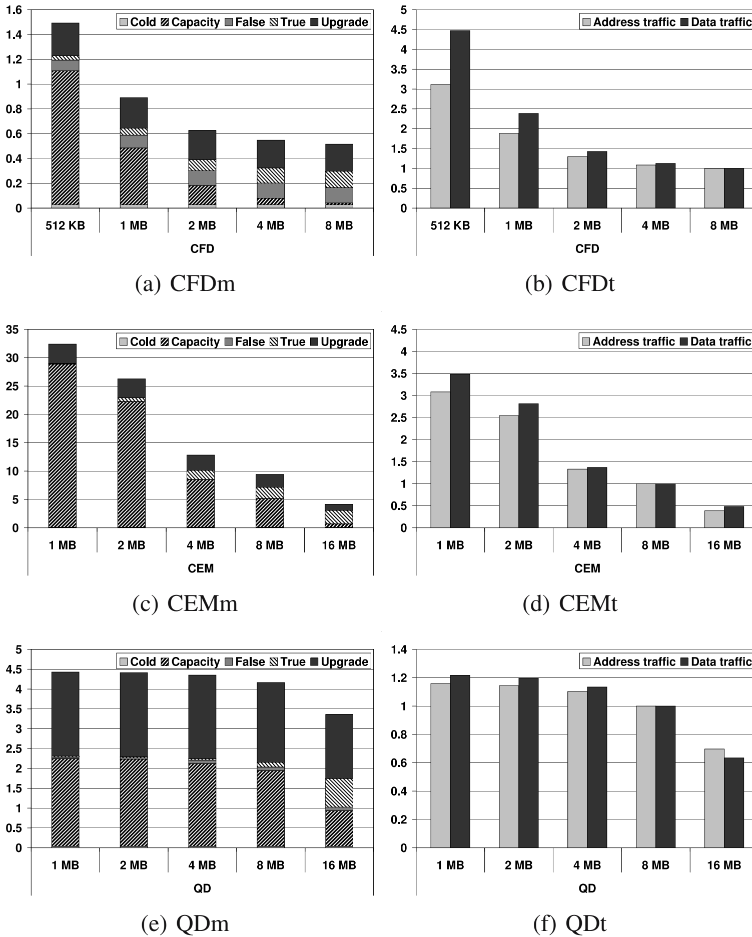


**Fig. 1.** Influence of cache size on cache misses, address and data traffic. The miss ratio in percent is indicated in the cache miss figures. The address and data traffic are normalized relative to the 8 MB configuration

## 6 Conclusion

We have shown that full system simulation of PDE solvers with realistic problem sizes is possible. By using simulation, we can obtain important information about the run time behavior of the solvers. The information is difficult to collect using traditional methods such as hardware counters. Simulation also allows us to run the solvers on non existing computer systems and configurations.

Our three PDE-solvers were found to have vastly different cache memory characteristics. The collected data in combination with a good knowledge of the algorithms made it possible to understand the behavior of each solver in great detail. We are certain that this Understanding will make it easier to find ways to better utilize the cache memory.

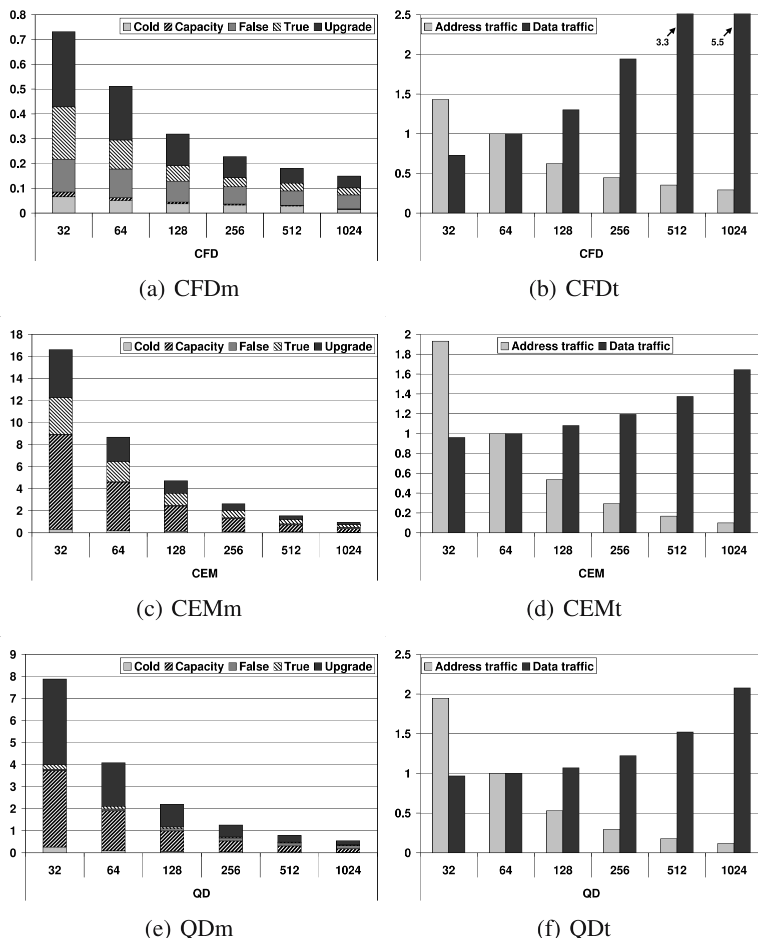


**Fig. 2.** Influence of cache line size on cache misses, address and data traffic. The miss ratio in percent is indicated in the cache miss figures. The address and data traffic are normalized relative to the 64 B configuration

## References

1. P. S. Magnusson et al, Simics: A Full System Simulation Platform, IEEE Computer, Vol. 35, No. 2, pp. 50-58, 2002.
2. M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod,Using the SimOS Machine Simulator to Study Complex Computer Systems, ACM TOMACS Special Issue on Computer Simulation,1997.
3. L. A. Barroso, K. Gharachorloo and E. Bugnion, Memory System Characterization of Commercial Workloads, In Proceedings of the 25th International Symposium on Computer Architecture, June 1998.
4. B. Verghese, S. Devine, A. Gupta, and M. Rosenblum, Operating System Support for Improving Data Locality on CC-NUMA Compute servers, In ASPLOS VII, Cambridge, MA, 1996.

5. E. Bugnion, J. M. Anderson and M. Rosenblum, Using SimOS to characterize and optimize auto-parallelized SUIF applications, First SUIF Compiler Workshop, Stanford University, Jan. 11-13, 1996.
6. M. Karlsson, K. Moore, E. Hagersten, and D. Wood,Memory System Behavior of Java-Based Middleware, n Proceedings of the Ninth International Symposium on High Performance Computer Architecture (HPCA-9), Anaheim, California, USA, February 2003.
7. M. Marden, S. Lu, K. Lai, M. Lipasti, Comparison of Memory System Behavior in Java and Non-Java Commercial Workloads, In the proceedings of the Computer Architecture Evaluation using Commercial Workloads (CAECW-02), February 2, 2002.
8. J. Hennessy and D Patterson, Computer Architechture, A Quantitative Approach, Morgan Kaufmann.
9. A. Jameson and D. A. Caughey, How Many Steps are Required to Solve the Euler Equations of Steady Compressible Flow: In Search of a Fast Solution Algorithm, AIAA 2001-2673, 15th AIAA Computational Fluid Dynamics Conference, June 11-14, 2001, Anaheim, CA.
10. M. Nordén, M. Silva, S. Holmgren, M. Thuné and R. Wait, Implementation Issues for High Performance CFD, Proceedings of International Information Technology Conference, Colombo, 2002.
11. F. Edelvik, Hybrid Solvers for the Maxwell Equations in Time-Domain, PhD thesis, Dep. of Information Technology, Uppsala University, 2002.
12. Å. Petersson, H. Karlsson and S. Holmgren, Predissociation of the Ar-12 van der Waals Molecule, a 3D Study Performed Using Parallel Computers, Submitted to Journal of Physical Chemistry, 2002.
13. D. Wallin, Performance of a High-Accuracy PDE Solver on a Self-Optimizing NUMA Architecture, Master's thesis, Dep. of Information Technology, Uppsala University, 2001.
14. H. Johansson, An Analysis of Three Different PDE-Solvers With Respect to Cache Performance, Master's thesis, Dep. of Information Technology, Uppsala University, 2003.
15. D. Wallin, H. Johansson and S. Holmgren, Cache Memory Behavior of Advanced PDE Solvers, Parallel Computing: Software Technology, Algorithms, Architectures and Applications 13, Proceedings of the International Conference ParCo2003, pp. 475-482,Dresden, Germany, 2003.
16. S. J. Eggers and T. E. Jeremiassen, Eliminating False Sharing, Proceedings of International Conference on Parallel Processing, pp. 377-381, 1991.