Cache Memory Behavior of Advanced PDE Solvers

D. Wallin[a] *, H. Johansson[a], S. Holmgren[a]

[a]Uppsala University, Department of Information Technology
P.O. Box 337, SE-751 05 Uppsala, Sweden

Three different partial differential equation (PDE) solver kernels are analyzed in respect to cache memory performance on a simulated shared memory computer. The kernels implement state-of-the-art solution algorithms for complex application problems and the simulations are performed for data sets of realistic size.

The performance of the studied applications benefits from much longer cache lines than normally found in commercially available computer systems. The reason for this is that numerical algorithms are carefully coded and have regular memory access patterns. These programs take advantage of spatial locality and the amount of false sharing is limited. A simple sequential hardware prefetch strategy, providing cache behavior similar to a large cache line, could potentially yield large performance gains for these applications. Unfortunately, such prefetchers often lead to additional address snoops in multiprocessor caches. However, applying a bundle technique, which lumps several read address transactions together, this large increase in address snoops can be avoided. For all studied algorithms, both the address snoops and cache misses are largely reduced in the bundled prefetch protocol.

1. INTRODUCTION

Designing the memory system of a parallel computer is a complex, multi-variable optimization problem. One parameter that must be determined is the cache line size. In a uniprocessor a large cache line size normally efficiently reduces the number of cache misses. However, in multiprocessors a longer cache line may lead to increased traffic as well as more cache misses if a large amount of false sharing occurs [1].

Another complication when designing the cache subsystems is that the preferred cache line size is very application dependent. Most available multiprocessors are optimized for running commercial software, e.g. databases and server-applications. In these applications, the data access pattern is very unstructured and a large amount of false sharing occurs [2,3]. The computer vendors usually build multiprocessors with cache line sizes ranging between 32 and 128 B. This cache line size range gives rather good performance trade-offs between cache misses and traffic for commercial applications, but might not be ideal for scientific applications.

Computing the solution of partial differential equations (PDEs) is a central issue in many fields of science and technology. Despite increases in computational power and advances in solution algorithms, PDEs arising from accurate models of complex, realistic problems still result in very time and memory consuming computations which must be performed using parallel computers. In this paper, we evaluate the behavior of three PDE solvers. The kernels are based on modern, efficient algorithms, and the settings and working sets are chosen to represent realistic application problems. These problems are much more demanding than commonly used scientific benchmarks, e.g. found in SPLASH-2 [1], which usually implement simplified solution algorithms. The codes are written in Fortran 90 and parallelized using OpenMP directives.

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 for possible future designs. Based on the simulations, we conclude that the spatial locality is much better

*This work is supported in part by Sun Microsystems, Inc. and the Parallel and Scientific Computing Institute, Sweden.

in these PDE kernels than in commercial benchmarks. Therefore, the optimal cache line size for these algorithms is larger than in most available multiprocessors. Spatial locality could be better explored also in a small cache line size multiprocessor using prefetching. A very simple sequential prefetch protocol, prefetching several consecutive addresses on each cache miss, gives a cache miss rate similar to a large cache line protocol. However, the coherence and data traffic on the interconnect increase heavily compared to a non-prefetching protocol. We show that by using the bundling technique, previously published in [3], the coherence traffic can be kept under control.

2. THE PDE SOLVERS

The kernels studied below represent three types of PDE solvers, used for compressible flow computations in computational fluid dynamics (CFD), radar cross-section computations in computational electromagnetics (CEM) and chemical reaction rate computations in quantum dynamics (QD). The properties of the kernels differ a lot with respect to the amount of computations performed per memory access, memory access patterns, amount of communication and communication 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 [4]. The implementation is described in detail by Nordén [5]. The data used in the computations are highly structured. Each smoothing operation in the multigrid scheme consists of a sweep through an array representing the grid. The values of the solution vector at six neighbor cells are used to update the values in each cell. After smoothing, the solution is restricted to a coarser grid, where the smoother is applied again recursively. The total work is dominated by the computations performed within each cell for determining the updates in the smoothing operations. These local computations are quite involved, but the parallelization of the smoothing step 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 [6]. 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 [7]. 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 first dimension are performed in parallel and locally [8]. For the FFTs in the second dimension, a parallel transpose operation is applied to the solution data, and the local FFTs are applied in parallel again. The communication within the kernel is concentrated to the transpose operation, which involves heavy all-to-all communication between the threads. However, the communication pattern is constant between the iterations in the time loop.

3. SIMULATION ENVIRONMENT AND WORKING SETS

All experiments are carried out using execution-driven simulation in the Simics full-system simulator. The modeled system implements a snoop-based invalidation MOSI cache coherence protocol. 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. The only hardware parameter that is varied in the experiments is the cache line size, which is normally 64 B in the SunFire 6800. The second level caches of this computer system are subblocked. To isolate the effects of a varying cache line size and to avoid corner cases in the prefetch experiments, the figures presented are for simulated non-subblocked caches. A comparative study was performed also with subblocked second level caches with a small increase in cache misses as a consequence.

This article only presents results for the measured second level cache misses and the amount of traffic produced rather than to simulate the contention that may arise on the interconnect. This will not allow us to estimate the wall clock time for the execution of the benchmarks. 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.

We only perform a small number of iterative steps for each PDE kernel to limit the simulation time. Also a limited number of iterations gives appropriate results since the access pattern is most often regular within each iteration for the PDE kernels. The results were verified with longer simulations. Before starting the measurements, each solver completes a full iteration to warm-up the caches. The CFD solver runs for two iterations, the CEM solver runs until convergence (about 20 iterations) and the QD solver runs for three iterations. The CFD problem size has a grid size of 32×32×32 elements using four multigrid levels. The CEM problem represents a modeled generic aircraft with a problem coefficient matrix of about 175,000×175,000 elements; a little more than 300,000 of these are non-zero. The QD problem size is 256×256×20, i.e. a 256×256 2D FFT in the x-y plane followed by a 20 element FDM in the z-direction, a realistic size for problems of this type.

4. IMPACT OF VARYING CACHE LINE SIZE ON THE PDE SOLVERS

The cache miss characteristics, the number of snoop lookups and the data traffic have been measured for different cache line sizes for the three PDE solvers in Figure 1. The cache misses are categorized into five different cache miss types according to Eggers and Jeremiassen [9]. The *data traffic* is a

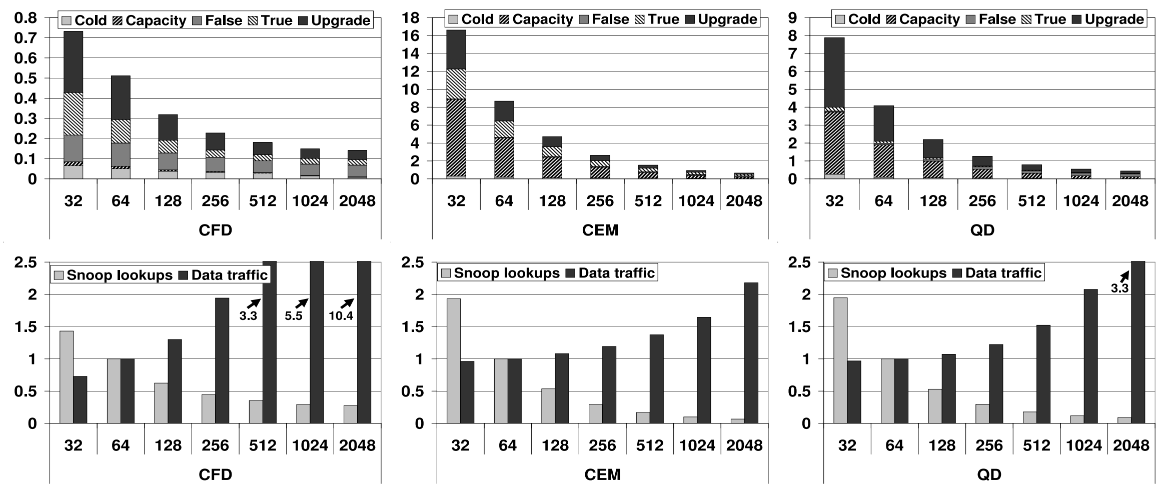

Figure 1. Influence of cache line size on second level cache misses, snoop lookups and data traffic in the three PDE kernels. The miss ratio in percent is indicated in the cache miss figures. The snoop lookups and the data traffic are normalized relative to the 64 B configuration.

measurement of the number of bytes transferred on the interconnect, while the term *snoop lookups* represents the number of snoops the caches have to perform.

The CFD kernel performs most of the computations 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 and the amount of these misses is halved with each doubling of the line size. However, the true and false sharing misses decrease slower since they cannot be reduced below a certain level. The decrease in snoop lookups is approximately proportional to the decrease in miss ratio. Due to the large increase in data traffic, the ideal cache line size is shorter in this kernel than in the other kernels. The decrease in cache miss ratio is influenced by a remarkable property in this kernel: 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. When a processor is about to finish a smoothing operation, it requests data that previously have been modified by another processor and thus is invalidated. The invalidated data are placed consecutively in the remote cache. Larger pieces of this data are brought into the local cache when a longer cache line is used. If a shorter cache line is used, less invalidated data will be brought to the local cache in each access and therefore a larger number of accesses is required to bring all the requested data to the cache. Therefore, all these accesses will be categorized as false sharing misses.

The CEM kernel has a large problem size, causing a high miss ratio for small cache line sizes. Capacity misses are most 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 randomly accessed. False sharing is not a problem in this application, not even for very large cache line sizes. The data traffic exhibits nice properties for large cache lines because of the rapid decrease in the miss ratio. The snoop lookups are almost halved with each increase in cache line size. Therefore, the optimal cache line size for the CEM kernel is very long.

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 snoop lookups show the opposite behavior and decreases rapidly until the 256 B cache line size, where it levels out.

5. SEQUENTIAL PREFETCHING AND BUNDLED SEQUENTIAL PREFETCHING

The former results showed that a very long cache line would be preferred in a computer optimized for solving PDEs. A preferable line size would probably be between 256 and 512 B for these applications. Even larger cache lines lead to less efficiently used caches with a large increase in data traffic as a consequence. Unfortunately, no computer is built with such large line size. Instead, a similar behavior to having a large cache line can be obtained by using various prefetch techniques. These techniques try to reduce miss penalty by prefetching data to the cache before the data are being used.

A very simple hardware method to achieve cache characteristics similar to having a large cache line, while keeping a short cache line size, is to implement sequential prefetching. In such systems, a number of prefetches is issued for data having consecutive addresses each time a cache miss occurs. The amount of prefetching is governed by the prefetch degree, which decides how many additional prefetches to perform on each cache miss. Dahlgren has studied the behavior of sequential prefetching on the SPLASH benchmarks [10]. Sequential prefetching normally efficiently reduces the number of cache misses since more data is brought into the cache on each cache miss. Sequential prefetching requires only small changes to the cache controller and can easily be implemented without a large increase in coherence complexity. The main disadvantage with sequential prefetching schemes is that the data traffic and the snoop lookups usually increase heavily.

The snoop lookups can be largely reduced in a sequential prefetch protocol using the bundling technique [3]. Bundling lumps the original read request together with the prefetch requests to the consecutive addresses. The original read request is extended with a bit mask, which shows the address offset to the prefetch addresses. Bundling efficiently limits the number of address transactions on the interconnect. However, not only the address traffic can be reduced but also the number of snoop lookups required. This is done by requiring bundled read prefetch requests to only supply data if the requested prefetch cache line is also in the owner state in the remote cache. Data that are in other states will not be supplied to the requesting cache. Write and upgrade prefetches are not bundled and will not reduce the amount of address traffic generated or the snoop bandwidth consumed. Bundling gives a large performance advantage since the available snoop bandwidth is usually the main contention bottleneck in snoop-based multiprocessors. The available data bandwidth is a smaller problem since the data packets do not have to be ordered and can be returned on a separate network. For example, the Sunfire 6800 server has a data interconnect capable of transferring 14.4 GB/s while its snooping address network can support 9.6 GB/s worth of address snoops [11].

We have studied sequential hardware prefetching on the PDE solvers in Figure 2. We present results

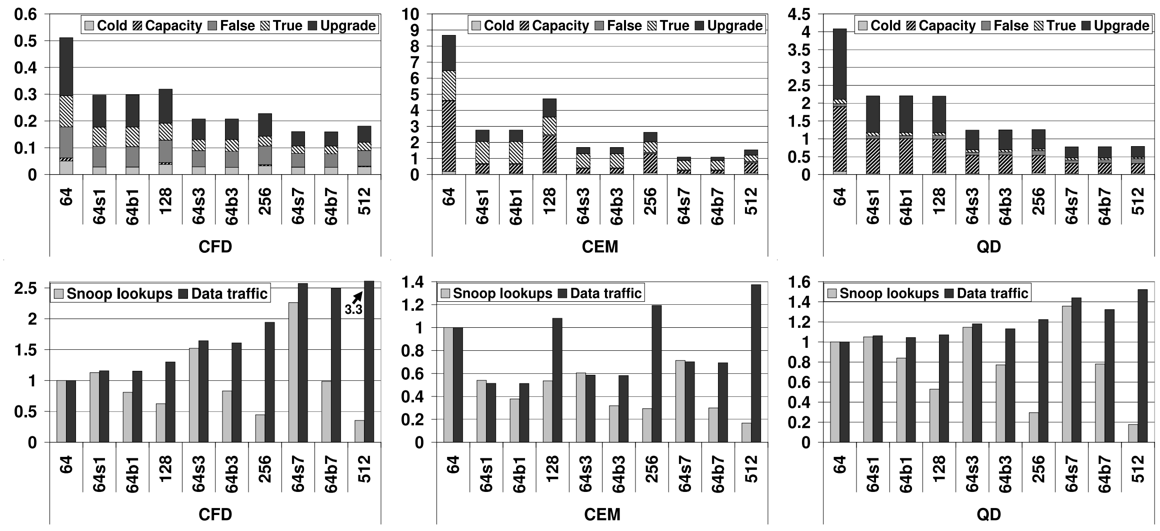

Figure 2. Influence of sequential and bundled sequential prefetching on second level cache misses, snoop lookups and data traffic. The miss ratio in percent is indicated in the cache miss figures. The snoop lookups and the data traffic are normalized relative to the 64 B configuration.

of non-prefetching configurations having different cache line sizes: *64*, *128*, *256* and *512* B. Several sequential prefetching configurations, *64s1*, *64s3* and *64s7*, are also studied which prefetch 1, 3 and 7 consecutive cache lines based on address on each cache miss. Finally the corresponding bundled configurations prefetching 1, 3 and 7 consecutive addresses, *64b1*, *64b3*and *64b7*, are evaluated.

Sequential prefetching works very well for the studied PDE solvers. If we compare the sequentially prefetching configurations, *64s1*, *64s3* and *64s7*, with the non-prefetching configuration *64*, we see that for all kernels the cache misses are largely reduced. Compared with the non-prefetching configuration with a comparable cache line size, e.g the *512* B configuration compared to the *64s7* configuration, the cache misses are lower or equal for the prefetching configuration. The main reason for the discrepancy is that with the sequential prefetching protocol, the consecutive addresses will always be fetched. If a protocol with a large cache line is used, a cache line aligned area around the requested address will be fetched, that is, both data before and after the desired address. Especially the CEM kernel takes advantage of this, where the cache misses are reduced about 30 percent in the sequential protocol compared to the large cache line non-prefetching protocol.

Sequential prefetching generally leads to more data traffic than the baseline *64* B non-prefetching configuration and less data traffic than the corresponding larger non-prefetching configuration. The snoop lookups increase rather heavily for the CFD and QD kernels compared with both a small and a large cache line size non-prefetching configuration. Bundling efficiently reduces the snoop lookup overhead. For all kernels, bundling yields an equal amount of cache misses as the corresponding sequential prefetch protocol. However, both the snoop lookups and the data traffic are reduced in the bundled protocols for all kernels. The reason for the decrease in data traffic is that the bundled protocol is more restrictive at providing data than the sequential protocol. Data are only sent if the owner of the original cache line is also the owner of the prefetch cache lines. Bundling makes it possible to use a large amount of prefetching at a much smaller cost, especially in address snoops, than normal sequential prefetching. There is still more address snoops generated in the bundled prefetch protocol than in a non-prefetching large cache line size protocol. This is almost entirely caused by upgrades generating prefetch messages, which cannot be eliminated since only reads are bundled.

The overall performance for the bundled sequential protocols compared with the baseline 64 B non-prefetching is excellent especially for the CEM and QD-kernels. For example the *64b7* configuration has about 88 percent less cache misses, 70 percent less address snoops and 30 percent less data traffic than the 64 B non-prefetching protocol for the CEM kernel. The performance of the bundled protocols in the CFD kernel is somewhat worse since the data traffic increases rather heavily compared with the non-prefetching protocol. However, since the total ratio of cache misses is small in this kernel, contention on the interconnect will most likely not be a performance bottleneck. Also, the snoop bandwidth is often more limited than the data bandwidth in snoop-based systems.

Compared with previous studies [3], the PDE kernels gain more from bundling than the SPLASH-2 benchmarks and commercial JAVA-servers. This is an effect of more carefully coded applications with better spatial locality. Sequential prefetching can simply be implemented in a multiprocessor without a large increase in hardware complexity. Some extra hardware is needed in the cache controller to fetch more data on each cache miss. The cost of implementing a bundled protocol is rather small even though some extra corner cases can be introduced in the cache coherence protocol [3].

6. CONCLUSION

From full-system simulation of large sized state-of-the-art PDE solvers we have learned that these applications do not experience large problems with false sharing as is the case in many benchmark programs and commercial applications. Also, the spatial locality is good in the PDE solvers and therefore it is beneficial to use large cache line sizes in these applications.

Sequential prefetching is one of the simplest forms of hardware prefetching and can easily be implemented in hardware. For the studied PDE solvers, it is beneficial to use sequential prefetching. To further improve the performance of the PDE solvers, sequential prefetching can be used together with the bundling technique to largely reduce the amount of address snoops required by the caches in shared-memory multiprocessors. Using this technique, both the number of cache misses and the address snoops become lower than in a non-prefetching configuration.

REFERENCES

[1] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta, The SPLASH-2 Programs: Characterization and Methodological Considerations, Proceedings of the ISCA, 1995.
[2] M. Karlsson, K. Moore, E. Hagersten and D. A. Wood, Memory System Behavior of Java-Based Middleware, Proc. of HPCA, 2003.
[3] D. Wallin and E. Hagersten, Miss Penalty Reduction Using Bundled Capacity Prefetching in Multiprocessors, Proc. of the IPDPS, 2003.
[4] 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, Proc. of the CFD, 2001.
[5] M. Nordén, M. Silva, S. Holmgren, M. Thuné and R. Wait, Implementation Issues for High Performance CFD, Proc. of International IT Conf., Colombo, Sri Lanka, 2002.
[6] F. Edelvik, Hybrid Solvers for the Maxwell Equations in Time-Domain. PhD thesis, Department of Information Technology, Uppsala University, 2002.
[7] 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 Phys. Chemistry, 2002.
[8] D. Wallin, Performance of a High-Accuracy PDE Solver on a Self-Optimizing NUMA Architecture. Master's thesis, Department of Information Technology, Uppsala University, 2001.
[9] S. J. Eggers and T. E. Jeremiassen, Eliminating False Sharing, Proc. of the ICPP, 1991.
[10] F. Dahlgren, M. Dubois and P. Stenström, Sequential Hardware Prefetching in Shared-Memory Multiprocessors, IEEE Transactions on Parallel and Distributed Systems, 6(7):733-746, 1995.
[11] A. Charlesworth, The Sun Fireplane System Interconnect, Proc. of Supercomputing, 2001.