# Impact of Cache Coherence Models on Performance of OpenMP Applications

Jie Tao and Wolfgang Karl

Institut für Rechnerentwurf und Fehlertoleranz
Universität Karlsruhe (TH)
76128 Karlsruhe, Germany
{tao,karl}@ira.uka.de

**Abstract.** OpenMP is becoming an important shared memory programming model due to its portability, scalability, and flexibility. However, as it is a fact with any programming paradigms, cache access behavior significantly influences the performance of OpenMP applications. Improving cache performance in order to reduce misses therfore becomes a critical issue for High Performance Computing. This can be achieved by optimizing the source code, but also gained through adequate coherence schemes.
This work studies the behavior of various cache coherence protocols, including both hardware based mechanisms and software based relaxed models. The goal is to examine how well individual schemes perform with different architectures and applications, in order to find general ways to support the cache design in shared memory systems. The study is based on a simulation environment capable of modeling the parallel execution of OpenMP programs. First experimental results show that relaxed models are scalable and can be used as efficient alternative for those hardware coherence mechanisms.

## 1 Introduction

OpenMP [3] is becoming increasingly popular in both research and commercial areas. In contrast to the message passing models, like PVM and MPI [8], OpenMP has the advantages of being capable of releasing the programmers from the burden of explicitly specifying the control and communication between processes and threads. OpenMP also enforces a nested structure in the parallel program allowing to exploit the parallelism to the full extent. In addition, OpenMP is portable, flexible, and scalable.

Traditionally, OpenMP is developed to program Symmetric Multiprocessor (SMP) machines that deploy a physically shared memory. With the endeavor of compiler developers, however, OpenMP applications can currently also run on top of cluster environment with software Distributed Shared Memory (DSM). Examples are the Nanos Compiler [4] and the Polaris parallelizing compiler [2]. These compilers enable to maintain the architectural properties, e.g. scalability and cost-effectiveness, of modern cluster systems, while at the same time use OpenMP for developing parallel programs.

A problem with parallel programming is that applications usually can not achieve expected performance. Among those various reasons cache utility is rather critical. Since data stored in caches can often not be reused, applications still suffer from large

M. Danelutto, D. Laforenza, M. Vanneschi (Eds.): Euro-Par 2004, LNCS 3149, pp. 149–154, 2004.

amount of cache misses and the thereby resulted long access time. This issue is specially critical for OpenMP programs on cluster architectures since OpenMP exploits fine-grained parallelism, which introduces more data transfers between processors.

A primary source causing cache misses is cache line invalidation. On a multiprocessor system, each processor cache can hold a copy of the same data and all these copies have to be kept consistent. Tightly coupled systems usually use hardware based consistency mechanisms, and clusters with software DSM often deploy relaxed consistency models. These coherence protocols vary significantly in techniques for selecting cache lines to invalidate and hence probably result in rather different behavior of invalidation.

This work investigates various cache coherence protocols, including those existing schemes and several novel ones. This study is based on an OpenMP cache simulator that models shared memory architectures and the parallel execution of OpenMP programs. First experimental results with standard benchmark applications show that hardware consistency mechanisms perform well on small systems, while relaxed models often behave better on larger machines.

The remainder of this paper is organized as follows. Section 2 briefly describes common used cache coherence protocols and several new ones. In Section 3 the OpenMP simulation environment is introduced. This is followed by the first experimental result in Section 4 that shows the different validation behavior of the investigated coherence models. The paper concludes in Section 5 with a short summary and a few future directions.

## 2 Cache Consistency Model

Shared memory systems are typically constructed from commodity processors with on-chip caches or cache hierarchies. Replications of data in caches have to be consistent giving the user a unified memory abstraction. This kind of consistency is maintained by cache coherence protocols.

For bus-based multiprocessors, like SMPs, usually a snooping protocol is deployed. A snoopy coherence protocol uses an invalidation strategy with this concept: when a processor writes to its cache, it sends the address in a special invalidation message through the bus to other processors; cache copies in these processors are correspondingly invalidated. This strategy does not result in much traffic due to the shorter invalidation message. Snooping protocols often combine a cache tag with different states and uses a simple state machine to adjust the status of cache tags. A well known representative is MESI [1], which uses four states: modified, exclusive modified, shared, and invalid, to identify the cache tags.

Another group of hardware coherence protocols are directory based mechanisms. In contrast to the snooping protocols, the directory based protocols maintain a distributed directory for information about which caches contain which memory entries. When an entry is modified, the directory invalidates the copies in other caches with that entry. Since no broadcast is required during the invalidation process, directory based protocols are usually applied on multiprocessors with hardware DSM like NUMAchine [5] and SGI Origin O2000/3000 [7].

However, both groups of coherence mechanisms can not be used on clusters with software DSM since they rely on hardware support. On these machines the consistency behavior is controlled in software and usually a relaxed coherence protocol is deployed.

Relaxed consistency models [12] formalize the use of inconsistent memory hence giving the programmer a consistent memory abstraction. They are generally combined with synchronization primitives, like locks and barriers, which are contained in shared memory codes. These primitives force the caches consistent by invalidating cache entries with an *Acquire* access. Existing systems in this area often deploy a full invalidation mechanism that invalidates the whole cache by a synchronization operation. This is easy to implement, but introduces significant overheads and cache line misses.

Actually, full invalidation is not necessary because often only partial entries in the caches are updated before a synchronization operation is required. Therefore, we propose two novel schemes based on partial invalidations: Selective I and Selective II. The former only invalidates cache lines holding remote memory contents, while the latter invalidates only those remote cache lines which require to be invalidated. In contrast to the former scheme, the second one is capable of leading to less invalidations. For example, locks usually protect specific regions. Cache lines that would need to be invalidated under a lock might not be updated outside the critical regions and need not to be invalidated during the next barrier operation.

## 3 Simulation Platform

The cache coherence mechanisms described above have been evaluated using an OpenMP simulation platform [10] based on Valgrind [11]. Valgrind is a trace-driven memory simulator and debugger for x86-GNU/Linux. It is capable of supervising the execution of a program and recording all memory reads and writes. More specially, Valgrind models the POSIX threads supporting therefore any shared memory programming models based on this thread library. Additionally, Valgrind uses a runtime instrumentation technique to mark each memory access instruction within the executables. This enables to build an independent OpenMP simulator allowing the use of any compiler including the commercial ones.

Valgrind itself contains a cache simulator, which supports two level cache hierarchy with fixed write back behavior and LRU replacement policy. However, this simulator does not model multiprocessors, but uniprocessor systems. In order to enable the study of the memory system on shared memory machines, we have integrated into Valgrind a backend, while using Valgrind as a frontend for establishing the basic simulation infrastructure and generating memory references.

The backend [9] is a self-developed package that models the target architectures. As we focus on the research work on the memory system, the backend contains mainly mechanisms for modeling the complete memory hierarchy in detail. This includes a flexible cache simulator which models caches of arbitrary levels and various cache coherence protocols, a memory control simulator which models the management of shared and distributed shared memories and a set of data allocation schemes, and a network mechanism simulating the interconnection traffic.

The cache simulator models a multilevel cache hierarchy and allows each cache to be organized as either write-through or write-back depending on the target architectures. All relevant parameters including cache size, cache line size, and associativity can be specified by the user. In addition to simulating the caches themselves, the cache simulator also models a set of cache coherence protocols. This includes those mechanisms described in Section 2 and an optimal scheme in order to evaluate them. All schemes maintain the caches on the system consistent in the following way:

- *MESI*: invalidates all valid cache copies on other processors at every shared write.
- *Optimal*: within this scheme, invalidations are not done by write operations, rather by read accesses. At each shared read, the accessed cache line is invalidated only when the content has been updated by other processors since the last loading. This scheme hence performs invalidations only when necessary.
- *Full Invalidation*: invalidates all valid cache lines on other processors whenever an application initiates a lock or a barrier.
- *Selective I*: similar to the *Full Invalidation*, but invalidates only those cache lines holding remote memory contents.
- *Selective II*: similar to *Selective I*, but invalidates only those remote cache lines which require to be invalidated.

## 4 Experimental Results

Using this simulation platform described above, we could study the influence of various cache coherence policies on the cache performance of OpenMP applications. All applications for this study are chosen from the NAS parallel benchmark [6]. This benchmark is designed for comparing the performance of parallel computers and is widely recognized as a standard indicator of computer performance. Selected applications are FT, LU, MG, and CG.

FT contains the computational kernel of a three-dimensional FFT-based spectral method. It performs three one-dimensional fast Fourier transformations, one for each dimension. LU is a simulated CFD application that uses symmetric successive over-relaxation (SSOR) to solve a block lower triangular and block upper triangular system of equations resulting from an unfactored finite-difference discretization of the Navier-Stokes equations in three dimensions. MG uses a multigrid algorithm to compute the solution of the three-dimensional scalar Poisson equation. CG tests unstructured grid computation and communication using a matrix with randomly generated locations of entries. For the following experiment, the working set size is chosen: $64\times64\times32$ for FFT, $32\times32\times32$ for LU and MG, and 1400 for CG.

Firstly, we examine the invalidation behavior of these coherence protocols on DSM systems in order to give an initial comparison. For this, all tested codes are simulated on a loosely-coupled multiprocessor with 4 processor nodes, each deploying a 16KB, 2-way L1 cache and a 512KB, 4-way L2 cache. Access latency on memory locations is chosen with 1 cycle for L1, 5 cycles for L2, 50 cycles for the main memory, and 2000 cycles for inter-node communication.

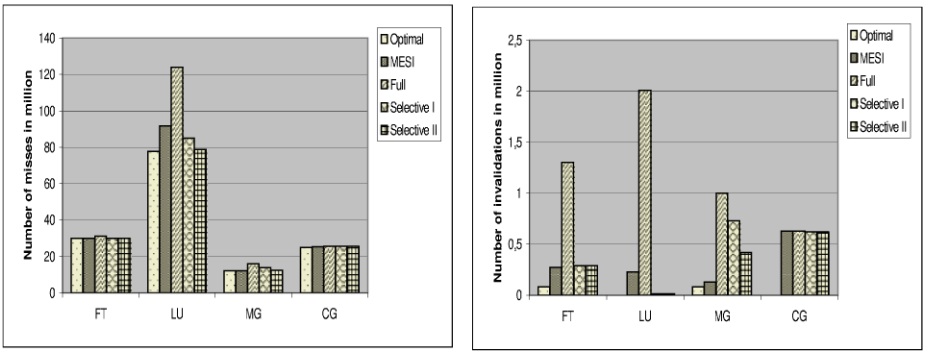


**Fig. 1.** Cache misses (right) and invalidations (left) on a 4-node system.

Figure 1 shows the experimental results with the left presenting the number of invalidations and the right the number of cache misses caused by different coherence protocols. It can be observed that applications vary significantly.

For FT similar behavior can be seen with all protocols in terms of miss number. A surprised case is the Full scheme, where even 1.3 million invalidations have been caused, which are nearly 5 factors more than the MESI protocol, the execution behavior of FT is not clearly influenced. This can be explained by the fact that the invalidated cache lines do not contain reused data and hence those additional invalidations result in no more cache misses.

The LU code, however, shows a quite different behavior. As expected, the Optimal protocol behaves better than MESI due to its ability of only performing invalidations as necessary. MESI behaves better than Full. This is caused by the full cache invalidation of the latter. Nevertheless, both Selective schemes performs well with even a better performance than MESI. This can be explained by the fact that LU contains many barriers and locks, where Full and the Selective protocols perform invalidations. Since both Selective schemes only invalidate partial cache lines, rather than the whole cache as Full does, frequently reused data can be maintained in the caches.

For the MG code, it can be observed that all relaxed consistency models result in more cache misses than MESI. This poor behavior is caused by the larger number of invalidations which can be seen in the right figure. While only one barrier is used in this code, both selective schemes can not show their advantages.

The last code CG shows a close behavior with FT in terms of cache misses. While Optimal presents a slight improvement, the others behave similarly with only a 0.06% gain for MESI in comparison to Full and the Selective schemes.

Overall, the experimental results show that the relaxed consistency models are generally not worse than MESI on small systems. This leads us to further examine the invalidation behavior on larger system and using larger working set size. The experimental results show that relaxed consistency models are beneficial on larger systems and larger working set, and can provide an efficient alternative for hardware coherence mechanisms.

## 5 Conclusions

This work uses a simulation platform to investigate the impact of various cache coherence protocols on the execution behavior of OpenMP applications. The studied mechanisms include hardware based schemes and several relaxed models. It is found that the former, like the MESI protocol, works well on machines with small number of processors, while the latter often outperforms on larger systems and working set.

In the next step of this research work, we will examine more applications with complex access patterns, especially those realistic ones. The goal is to classify various applications and detect adequate coherence schemes for them. In addition, the same work would be done on actual cluster systems.

## References

1. J. Archibald. A Cache Coherence Approach for Large Multiprocessor Systems. In *Proceedings of the International Conference on Supercomputing*, pages 337–345, Vovember 1988.
2. A. Basumallik, S.-J. Min, and R. Eigenmann. Towards OpenMP Execution on Software Distributed Shared Memory Systems. In *Proceedings of the 4th International Symposium on High Performance Computing (ISHPC 2002)*, pages 457–468, 2002.
3. L. Dagum and R. Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Computational Science & Engineering*, 5(1):46–55, January 1998.
4. Marc Gonzàlez, Eduard Ayguadé, Xavier Martorell, Jesús Labarta, Nacho Navarro, and José Oliver. NanosCompiler: Supporting Flexible Multilevel Parallelism in OpenMP. *Concurrency: Practice and Experience*, 12(12):1205–1218, 2000.
5. T. S. Grbic, S. Brown, S. Caranci, G. Grindley, M. Gusat, G. Lemieux, K. Loveless, N. Manjikian, S. Srbljic, M. Stumm, Z. Vranesic, and Z. Zilic. Design and Implementation of the NUMAchine Multiprocessor. In *Proceedings of the 1998 Conference on Design Automation*, pages 66–69, Los Alamitos, CA, June 1998.
6. H. Jin, M. Frumkin, and J. Yan. The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. Technical Report NAS-99-011, NASA Ames Research Center, October 1999.
7. J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 241–251, May 1997.
8. I. Pramanick. MPI and PVM Programming. In R. Buyya, editor, *High Performance Cluster Computing*, volume 2, Programming and Applications, chapter 3, pages 48–86. Prentice Hall PTR, 1999.
9. J. Tao, M. Schulz, and W. Karl. A Simulation Tool for Evaluating Shared Memory Systems. In *Proceedings of the 36th Annual Simulation Symposium*, pages 335–342, Orlando, Florida, April 2003.
10. J. Tao and J. Weidendorfer. Cache Simulation Based on Runtime Instrumentation for OpenMP Applications. In *Proceedings of the 37th Annual Simulation Symposium*, Arlington, VA, April 2004. to appear.
11. WWW. Valgrind, an open-source memory debugger for x86-GNU/Linux, 1999. http://developer.kde.org/∼sewardj/
12. Y. Zhou, L. Iftode, J. P. Singh, K. Li, B. R. Toonen, I. Schoinas, M. D. Hill, and D. A. Wood. Relaxed Consistency and Coherence Granularity in DSM Systems: A Performance Evaluation. In *Proceedings of the Sixth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 193–205, June 1997.