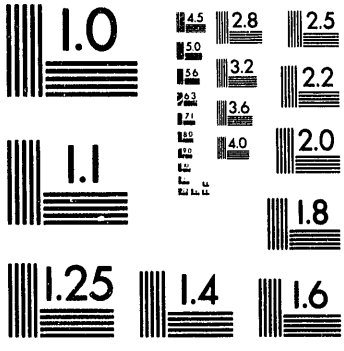
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0

1 of 1

Conf-9308129--1

UCRL-JC-113858
PREPRINT

Performance and Scalability Aspects of Directory-Based Cache Coherence in Shared-Memory Multiprocessors

Silvio Picano - Purdue University
David G. Meyer - Purdue University
Eugene D. Brooks III - LLNL
Joseph E. Hoag - LLNL

This paper was prepared for submittal to
22nd Annual Conference
1993 International Conference on Parallel Processing
The Pennsylvania State University - University Park, PA
August 16-20, 1993

May 1993

Lawrence Livermore National Laboratory

PERFORMANCE AND SCALABILITY ASPECTS OF DIRECTORY-BASED CACHE COHERENCE IN SHARED-MEMORY MULTIPROCESSORS *

Silvio Picano & David G. Meyer
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

Eugene D. Brooks III & Joseph E. Hoag
Massively Parallel Computing Initiative (MPCI)
Lawrence Livermore National Laboratory (LLNL)
Livermore, CA 94550

Abstract - *We present a study that accentuates the performance and scalability aspects of directory-based cache coherence in multiprocessor systems. Using a multiprocessor with a software-based coherence scheme, efficient implementations rely heavily on the programmer's ability to explicitly manage the memory system, which is typically handled by hardware support on other bus-based, shared memory multiprocessors. We describe a scalable, shared memory, cache coherent multiprocessor and present simulation results obtained on three parallel programs. This multiprocessor configuration exhibits high performance at no additional parallel programming cost.*

Keywords: directory-based cache coherence, parallel programming costs, Fortran D, multiprocessor simulation.

1 Introduction

General purpose machines, such as the BBN TC2000 [1], are achieving wide performance advantages (and cost advantages) over traditional mainframe technology. Yet these new parallel computers come with their own unique design and implementation problems with regards to shared and non-shared memory paradigms, scalability issues, parallel programming costs, and cache coherence support.

In the current parallel programming environment, programmers distinctly annotate programs with directives and instructions specifically aimed at a particular multiprocessor that will extract the desired performance. The resulting programs become difficult to understand because in addition to the underlying algorithm, details associated with the machine architecture are embedded in the programs.

In this paper, we analyze three parallel programs. We introduce a multiprocessor configuration with several important design features to enhance performance with little or no cost to an application programmer. We simulate the execution of the parallel programs on a multiprocessor simulator and show that high performance is achieved without additional programming costs and concerns.

*Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

2 The Parallel Programs

The parallel programs used in this study are described in the sections which follow.

2.1 A Network Simulation Program

The network simulation program [2] simulates and computes performance statistics of a proposed communications network for scalable multiprocessors. Simulated cpu ports interface to this network and communicate with other cpus via packet requests and responses. Multi-stage interconnection networks provide the communication vehicles for any number of simulated cpu and memory ports. The cpu and memory ports are connected to a request network, and a separate response network returns the requests back to the requesting cpu port.

2.2 An Iterative Relaxation Program

An iterative relaxation method is a stencil algorithm, which sets each element of a multi-dimensional grid array to the average of itself and its neighbors. The problem space is represented as a set of discrete elements, and at each iteration, each element is recalculated as a function of itself and its nearest neighbors. For our particular program, we average each element with its 8 nearest neighbors in a 2-dimensional, square grid. The domain decomposition is such that each processor will be responsible for the update of a fixed group of contiguous rows in the grid.

2.3 A Gaussian Elimination Program

Gaussian elimination is a method of solving a linear systems equation. The first part of the algorithm is referred to as the *reduction*, where the matrix is reduced to an upper triangular form. The second part of the algorithm, the *back-solve*, starts when the reduction is complete. Vector elements of the unknown solution are successively solved for and substituted into the remaining equations.

3 Performance Potential

3.1 Multiprocessor Simulation

A hardware enforced, cache coherent multiprocessor simulator system is described in [3, 4]. We detail the simulator's configuration of interest below:

1. RISC cpus with fully pipelined functional units, register scoreboarding, and compile-time branch prediction are used,
2. instruction pipeline stalls and flushes due to data hazards and wrong branch predictions are modeled exactly,
3. 2x2 switch nodes comprise a request and a response network,
4. each network is a worm-hole-routed, multi-buffered, multi-stage cube type with 64-bit data paths,
5. cache memory latency is 3 clock cycles with full pipelined access (1 access per clock cycle),
6. memory accesses and invalidations are faithfully modeled through the cache memories and networks,
7. a maximum of 5 outstanding shared memory requests per PE is allowed,
8. a *lock-up free*, write-back, shared data cache per PE (64 kilobyte, 2-way set associative, 16-byte line size) is used,
9. an invalidation-based protocol (fully documented in [4]) allows any number of read-only copies of a cache line and grants an exclusive copy of a cache line to a processor when a write operation occurs,
10. the presence flag technique [5] implemented at the memory controllers facilitates cache coherence, and
11. barrier synchronization is supported by software.

We do not model the effects of instruction cache misses or private data cache misses in the results presented below. Recall that 2x2 switch node elements comprise the networks. While a 4 PE system uses 4 network stages (i.e., 2 networks, each having 2 stages), a 32 PE system uses 10 network stages - which accounts for the increasing latency as the PE count increases. Invalidation distribution refers to the quantity of read/write sharing in the program. Before a write operation takes place, a processor must make sure that it has the only exclusive write access to this data. At the memory controller, invalidations are sent to every cache having a copy of this data before the write operation can continue.

3.2 Network Simulation Results

This version of the program ignores any logical, domain decomposition of the network simulation program's structure when scheduling parallel activity. We summarize some important run-time statistics in Table 1, and we show execution time results in Figure 1, for 2-ary 7-cube network simulations. [1]

Table 1: Network Simulation Execution Statistics

PEs	Cache Hit Ratio (%)	Latency (cycles)	Invalid. Dist. (%) Width-1	Width-2
2	85	10	96	4
4	89	13	93	6
8	92	15	92	6
16	92	19	92	7
32	90	28	91	7
64	88	33	89	8

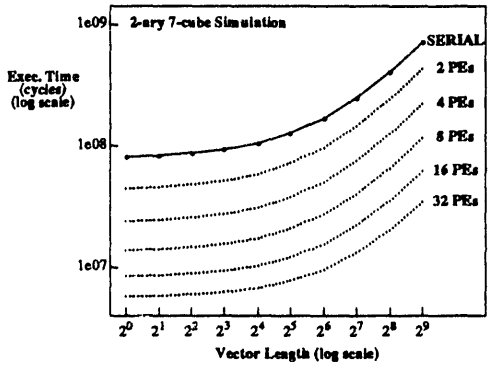

Figure 1: Network Simulation Execution Time

Analyzing the simulator's statistics more closely, we note that the shared data caches (64 kilobytes each) experience higher rates of both capacity misses and invalidation misses with respect to the rate of shared memory requests. For the 2-processor results, the capacity miss rate grows 1.08 times faster than the shared memory reference rate, while the invalidation miss rate grows 1.38 faster than the shared memory reference rate. In terms of total quantity of misses, capacity misses are 3 times greater than invalidation misses.

This situation reverses at higher processor counts. Analyzing the 32-processor results in particular, we note that capacity misses now grow 10 times slower than misses due to invalidations and 14 times slower than the shared memory reference rate. In absolute terms, capacity misses are only 5% of the total cache miss rate. Providing large, coherent caches is one key to achieving scalable performance for this application.

3.2.1 Iterative Relaxation Results

In Table 2, we summarize some of the important run-time statistics. In each category, the first column of numbers represents the statistics for the *128x128* problem size, while the second column of numbers represents the statistics for

[1] This problem size represents the operation of 7-stage networks having 2x2 switch nodes. [3]

Table 2: Iterative Relaxation Execution Statistics

PEs	Cache Hit Ratio (%)	Latency (cycles)	Invalid. Dist. (%)	
			Width-1	Width-2
2	96-93	10-10	95-98	5-1
4	96-93	12-13	93-98	6-2
8	98-93	17-15	83-97	15-1
16	98-95	25-19	75-96	22-2
32	96-95	30-28	71-98	25-1
64	90-97	33-33	53-82	42-17

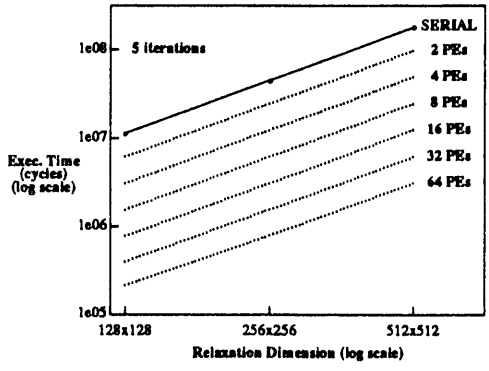

Figure 2: Iterative Relaxation Execution Time

Table 3: Gaussian Elimination Execution Statistics

PEs	Cache Hit Ratio (%)	Latency (cycles)	Invalid. Dist. (%)	
			Width-1	Width-2
2	51-50	16-15	99-99	1-0
4	51-50	20-19	99-99	1-0
8	52-51	24-22	99-99	1-0
16	56-52	30-26	97-99	2-0
32	62-54	38-31	97-99	3-1
64	71-58	50-38	95-99	5-1

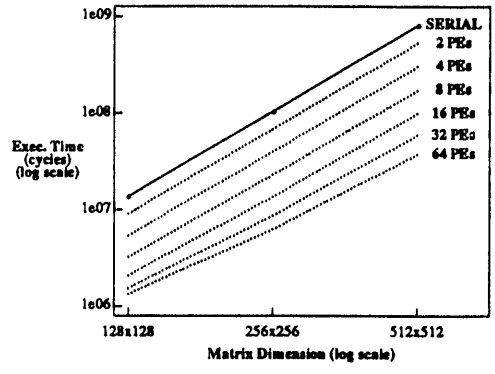

Figure 3: Gaussian Elimination Execution Time

the *512x512* problem size. We show the results of the simulations on this parallel program in Figure 2. In additional to the very high cache hit ratios, we see that width-1 and width-2 invalidation distributions combined make up over 95% of all invalidations across all numbers of processors. This distribution is a direct result of the decision to tile the 2-dimensional data in a blocked-row fashion to minimize invalidation requests (i.e., data sharing) from other PEs.

3.2.2 Gaussian Elimination Results

We summarize some of the important run-time parameters in Table 3, and we show serial and parallel execution time results in Figure 3. In each category, the first column of numbers represents the statistics for the *128x128* problem size. while the second column of numbers represents the statistics for the *512x512* problem size. From the table, we see some unexpected behavior in the cache hit ratio statistics. The increasing number of PEs actually increases the individual cache hit ratios, which is unlike many parallel applications. Investigating this further, we note that this program has a high quantity of barrier synchronization, which is actually implemented as an efficient software routine. The program's load imbalance and the execution time spent in barrier routines artificially inflates the cache hit ratio statistics. In terms of absolute performance, this program does not perform well because of the high level of invalidation traffic resulting from exclusively caching data that a PE will be unlikely to use again in a subsequent iteration. This performance degradation is evident through the increasing shared memory request latency at higher PE counts.

4 Related Work

4.1 Directory Limitations

As well documented in the literature [5, 6, 7], one major design constraint is the large amount of additional memory needed to implement the presence flag coherence technique. However, recent studies with Cerberus [6, 4] and Stanford's DASH project [7] show that *partial* directory states can reduce the memory requirements while maintaining a high level of desired performance. Work in [8] provides another method of reducing the directory requirements by using a combined hardware/software scheme to dynamically allocate directory state as shared data is referenced. Software-assisted cache coherence schemes [9] are alternatives to directory-based schemes. These schemes promote the use of compile-time reference marking combined with hardware-based coherence mechanisms.

4.2 Manual Data Distribution

Using simple and natural parallel program decompositions, each of the three programs required close to 8 parallel processors to regain the serial program's performance

on the BBN TC2000 [10, 2]. [2] This performance degradation is due to 1) the disabling of the data caches, and 2) the frequent use of the remote shared memory. Program optimizations to selectively enable data caching and to explicitly place shared data in proper regions of the memory hierarchy resulted in significant increases in parallel performance. However, programming costs and complexity to perform these optimizations are significant.

The resulting source programs are difficult to understand if the reader has little knowledge of the BBN TC2000 multiprocessor. The code modifications are necessary to explicitly place data in the multiprocessor's memory hierarchy to provide the fastest access possible.

4.3 Automatic Data Distribution

Several semi-automatic parallelization tools exist to simplify data distribution. One such tool aimed at architectures like the BBN TC2000 is the Parallel Data Distribution Preprocessor (PDDP) [10]. PDDP accepts a dialect of Fortran 77, Fortran 90 array syntax, and data layout directives recommended by the High Performance Fortran Forum (HPFF) committee. Much of the PDDP data distribution syntax and semantics have been borrowed from the Fortran D language [11].

The Gauss elimination program has been rewritten using PDDP primitives and has been benchmarked on the BBN multiprocessor. The initial PDDP program introduced some performance problems due to hot spot accesses when all processors attempt to read the pivot row (stored in one local memory). Additional code is necessary to minimize the performance penalty of this hot spot. This new PDDP program yielded from 40-50% performance loss over the best efforts reported on the BBN machine. From this experience, we see that automatic data distribution tools have some side-effects which still complicate performance and programming concerns [10].

5 Summary and Open Issues

Parallel programs make critical demands on non-uniform memory access multiprocessors because of the high communication requirements they pose. The resulting performance is strongly dependent on data placement, movement, and allocation strategies. Using incremental program modifications, we obtain significant performance gains on the BBN TC2000 multiprocessor through the effective use of the memory hierarchy. While automatic data distribution models may alleviate some of the programming costs incurred, resulting performance may not approach the multiprocessor's capabilities.

We described a multiprocessor configuration that has several important design features to enhance multiprocessor performance. Detailed execution time results of the parallel programs on this multiprocessor showed high performance at no additional parallel programming cost. We also believe the multiprocessor configuration used in this study is a conservative one, and that more aggressive designs can be implemented with current technology.

[2] The BBN TC2000 has facilities allowing programmers to maintain cache coherence in their software.

References

[1] BBN Advanced Computers Inc., Cambridge, MA. *Inside the TC2000*, 1989.

[2] S. Picano, E.D. Brooks III, and J.E. Hoag. Programming Costs of Explicit Memory Localization on a Large Scale Shared Memory Multiprocessor. In *Proc. of Supercomputing'91*, pages 36–45, 1991.

[3] E.D. Brooks III, T.S. Axelrod, and G.A. Darmohray. The Cerberus Multiprocessor Simulator. In G. Rodrigue, editor, *Parallel Processing for Scientific Computing*, pages 384–390. SIAM, 1989.

[4] J.E. Hoag. The Cache Group Scheme for Hardware-Controlled Cache Coherence and the General Need for Hardware Coherence Control in Large-Scale Multiprocessors. Technical Report UCRL-LR-106975, Lawrence Livermore National Laboratory, Livermore, CA, March 1991.

[5] L.M. Censier and P. Feautrier. A New Solution to Coherence Problems in Multicache Systems. *IEEE Transactions on Computers*, C-27(12):1112–1118, Dec. 1978.

[6] E.D. Brooks III and J.E. Hoag. A Scalable Coherent Cache System With Incomplete Directory State. In *Proc. of the International Conference on Parallel Processing*, pages 553–554, August 1990.

[7] A. Gupta, W.-D. Weber, and T. Mowry. Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes. In *Proc. of the International Conference on Parallel Processing*, pages 312–321, August 1990.

[8] D.J. Lilja and P.C. Yew. Combining Hardware and Software Cache Coherence Strategies. In *ACM International Conference on Supercomputing*, 1991.

[9] S.L. Min and J.-L. Baer. Design and Analysis of a Scalable Cache Coherence Scheme Based on Clocks and Timestamps. *IEEE Transactions on Parallel and Distributed Systems*, 3(1):25–44, Jan. 1992.

[10] E.D. Brooks III, B.J. Heston, K.H. Warren, and L.J. Woods. The 1992 MPCI Yearly Report: Harnessing the Killer Micros. Technical Report UCRL-ID-107022-92, Lawrence Livermore National Laboratory, Livermore, CA, Aug. 1992.

[11] S. Hiranandani, K. Kennedy, and C. Tseng. Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines. In *Proc. of Supercomputing'92*, pages 86–100, 1992.

DATE
FILMED
9/29/93

END