Improving the Performance of Distributed Shared Memory Environments on Grid Multiprocessors[1]

Dimitris Dimitrelos and Constantine Halatsis

Athens High Performance Computing Laboratory and
Department of Informatics, University of Athens
Panepistimiopolis, Athens 15771, Greece
e-mail: ddimitr@hpcl.uoa.gr, halatsis@di.uoa.gr

Abstract. Distributed Shared Memory (DSM) is a good solution to the scalability, complexity and high cost problems of large scale Shared Memory Multiprocessors, as well as to the difficulty of the programming model problem of the message passing Distributed Memory Multiprocessors. We present a method for improving the performance of Distributed Shared Memory Environments running on grid multiprocessors. The method is based on removing the inherent centralism imposed by X-Y routing that causes congestion in the centre of the grid. Simulation, as well as implementation results on a 1024 processor machine show an improvement of up to 24%.

1. The Environment

Distributed Shared Memory (DSM) allows the execution of programs assuming the Shared Variable programming paradigm in Distributed Memory architecture multiprocessor systems.

All DSM systems must use a coherence protocol, similar to those used by Shared Memory multiprocessors for cache coherence, to keep the shared data consistent at all times. The underlying message passing system makes it unrealistic to use any protocol that assumes broadcast, so directory based protocols are used.

According to directory coherence schemes, each shared object has a directory entry associated with it. In this directory entry the identity of the object's owners, its status (exclusively owned/shared/invalidated) and possibly other information are maintained and updated.

The directory entry of each shared object is kept in some processor. A processor that keeps at least one directory entry is called a *directory handler* (DH). The directory handler maintaining the directory entry of each shared object must be at all times known to every processor in the DSM system, since interaction with the DH is necessary for non-local accesses to shared data.

[1] This work was partly supported by the Greek General Secretariat of Research and Technology under the YPER-94 program.

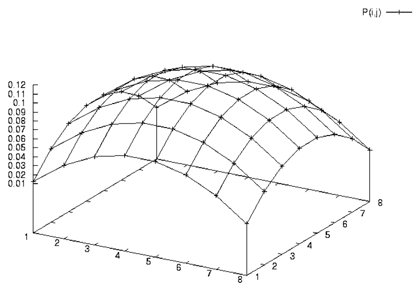

Fig. 1. P(i, j) values for an 8x8 processor grid

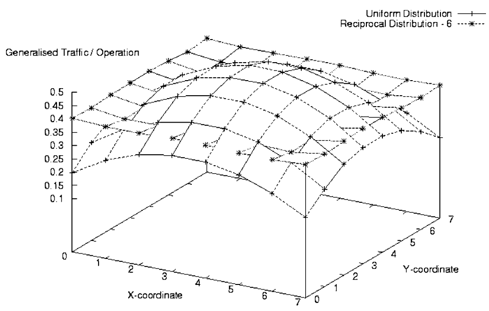

Fig. 2. Simulated traffic for uniform vs. reciprocal directory distribution.

The distribution of the directory data is a factor that plays an important role to the performance of the system. *Centralised Directory* schemes use a single processor for maintaining all directory data. According to *Distributed Directory* schemes, the directory data are distributed to more than one processors. According to the *Uniform Distribution* scheme (first introduced by Li in [1] as *Distributed Directory*) the directory entries are *uniformly* distributed among the processors. Even though the Uniform Distribution Directory scheme seems very attractive, its effect on grid multiprocessors using the X-Y routing algorithm for message passing has not been investigated.

2. Reciprocal Directory Distribution

According to the X-Y message routing, a message is propagated from the sender to the receiver first along the X axis until it reaches the column of the receiver, and then along the Y axis. The weak point of X-Y routing is the fact that more messages pass through the centre than through the 'edges' of the grid, on their way to their destination, causing more 'intermediate' traffic to the central processors.

We denote as *P(i ,j)* of processor (i, j) in a MxN processor grid the probability that a random message (not sent from, neither sent to (i, j)) passes through (i, j).

In [2] we formally proved the following theorem:

$$P(i,j) = \frac{-2(i^2 M + j^2 N) + 2iM(N+1) + 2jN(M+1) - 3MN - M - N + 1}{(MN-2)(MN-1)} \quad (1)$$

The value of P(i, j) among the processors on a 8x8 processor grid is depicted in Figure 1. The X and Y axes correspond to the processor's position in the grid and the Z axis to its P(i, j) value. It is easy to observe the mountain with the peak at the central (or central pair or quadruple) processor.

The Uniform Distribution Scheme for the shared object directory is the natural choice of a DSM system designer, since it provides uniformity in the distribution of data and avoids bottlenecks and hot spots. However, as all DSM operations are

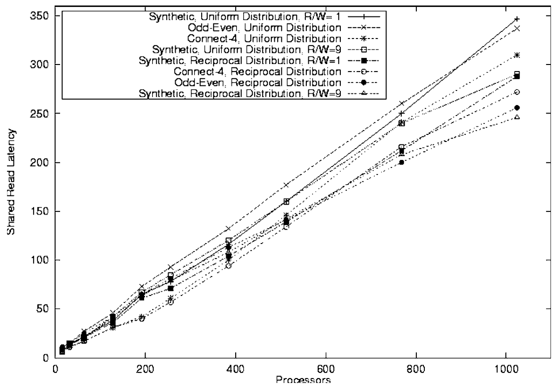

Fig. 3. Benchmark Results: Average Shared Read Times

implemented by explicit message passing between the processors, the centralisation observed at X-Y routing will be passed on to the system and therefore have an impact on the system's performance. It will cause easier congestion to the central parts of the network (that are accepting more traffic), reduce the message bandwidth and effectively reduce the performance of the DSM system.

The idea behind reversing the effects of X-Y centralism is simple. Our simulations revealed that 70%-90% of all messages exchanged in a DSM system start from or end to the Directory Handler. Therefore if we alter the directory distribution in order to move some traffic from the central processors to processors closer to the edges of the grid, we can create an anti-force to the X-Y centrality and equalise the communication load throughout the grid. Intuitively, the distribution should be done inverse proportionately to the 'centrality' of each processor, i.e. the processors residing on the edges of the grid will get a bigger part of the directory (more directory entries) than the processors closer to the centre of the grid.

According to our distribution scheme (*Reciprocal Directory Distribution*) the directory data are distributed proportionately to the sixth power of the average distance between the processor and any other processor. The simulated traffic per operation on an 16x16 processor grid is presented in Figure 2, along with the traffic resulting using the normal uniform directory distribution. The suppression of the centralism caused by X-Y is obvious.

The price that we have to pay for decentralisation is the extra distance that the messages have to travel in order for an operation to complete, as a result from the placement of more directory data in the outer parts of the grid. The increase varies from 3% in the 4x4 case to 9.3% in the 64x64 case.

4. Implementation Results

MaDCoWS[3] (MAssively Distributed COnfigurable Variable Validation System) is a scalable multi-parametric DSM system for 2D grid multiprocessors developed at the Athens High Performance Computing Laboratory and the Dpt. of Informatics of the University of Athens. It implements arbitrary-size variable sharing as well as global semaphore, barrier and read-modify-write operations. MaDCoWS has been developed as a runtime system under the parallel operating environment PARIX[4],

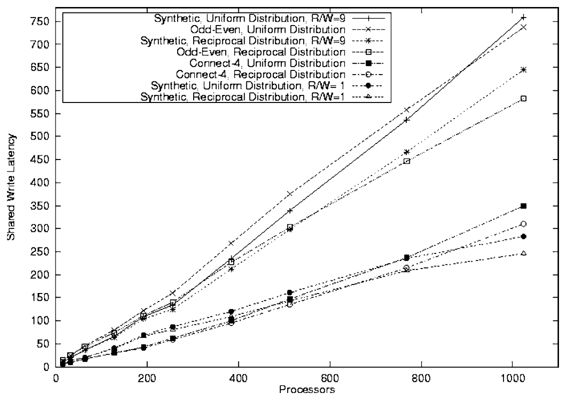

Fig. 4. Benchmark Result: Average Shared Write Times

and has been used, tested and benchmarked on the Parsytec GCel-1024, a 1024 processor machine. The system is currently being ported to the Intel Paragon and to the Parsytec GC-PP.

In order to evaluate the performance and scalability of the reciprocal directory distribution, we run several synthetic benchmarks and two real life applications -two games. The first is a virtual odd-even game played by the processors that exhibits a high degree of sharing, both temporally and spatially. The second application is a connect-4 game played by the system vs. a human.

The average shared read and shared write times for the 3 benchmarks are presented in Figures 3 and 4. In all cases, the Reciprocal Directory Distribution scheme performs better than the Uniform Directory Distribution, improving the DSM system's performance by up to 24% for shared read operations and up to 21% for shared write operations. The scheme performs better in the cases of dense sharing (as in the case of the Odd-Even benchmark), and for large grids. In the case of sparse sharing and very small numbers of processors (<64), a up to 10% increase in the write latency was observed.

Acknowledgments

The authors would like to thank Dr.Alexander Reinefeld and the PC2 centre of the University of Paderborn for allowing us to use their parallel computers for benchmark runs.

References

1. Li, K. '*Shared Virtual Memory on Loosely Coupled Multiprocessors.*' PhD thesis, Department of Computer Science, Yale University, September 1986.
2. Dimitrelos, D. and Halatsis, C. '*On the Distribution of Directory Information in a Software Controlled Distributed Shared Memory System*'. In Proc. of the Workshop on Parallel Programming and Computation (ZEUS'95), pages 75-89, Linkoping, May 1995.
3. Dimitrelos, D and Halatsis, C. '*MaDCoWS: A Scalable Distributed Shared Memory Environment for Massively Parallel Multiprocessors*', in Proc of HPCN'99, pp 784-793
4. Parsytec Computer GmbH '*PARIX 1.2. Software Documentation*', March 1993.