A Study of Multi-Grain Shared Memory Systems*

Donald Yeung†
Alewife Research Group, MIT LCS
piano@lcs.mit.edu

1 Introduction

Many shared memory multiprocessors incorporate hierarchy into their networks and memory systems. For example, the DASH multiprocessor [1] puts 4 processors on a bus in each cluster, and connects clusters via a 2-D mesh network. The KSR1 multiprocessor [2] places up to 32 processors on a ring network, and connects first level rings via second level rings. One advantage of these hierarchical architectures is that they exploit locality. A reference to a memory module in the same level of hierarchy only needs to consume network bandwidth at that level. The reference is not visible to processors outside the cluster. Also, processors in the same cluster that have overlapping working sets can sympathetically share remote data.

The benefits of clustering have been previously studied [3, 4]. What has not been explored is the interplay between the different levels of hierarchy and the granularity of sharing at each of these levels. The aforementioned systems all transport data at the same granularity irrespective of which levels of hierarchy are being traversed. Changing the granularity of sharing at the different levels can be advantageous for two reasons. First, larger per-transaction overheads are incurred as more levels of hierarchy are traversed. Therefore, employing larger granularities when communicating across more levels amortizes larger per-transaction overheads over more data. Granted, the success of this approach relies on applications exhibiting spatial locality. Second, when crossing a level in the hierarchy, a processor is requesting data both for itself, and potentially for all other processors in its cluster assuming there is overlap in the working sets of the processors in the cluster. This may warrant an increase in communication grain to match a larger computation grain.

While the work presented in this abstract applies to all shared memory systems with hierarchy, the immediate target is for DSM systems where each DSM node is itself a multiprocessor. A network of Fugus [5] would fall into this class of system. In such systems, the performance discrepancy between the hierarchy levels can be dramatic since there is hardware shared memory support in each DSM node, but only software DSM between nodes.

2 Methodology

A complete study of multi-grain shared memory will occur on the Fugu platform. Until this platform comes online, a preliminary study is being carried out on the 32-node Alewife Machine [6].

The Alewife Machine is logically partitioned into clusters. Within a cluster, processors replicate and migrate data at cache-line granularity using Alewife hardware support for shared memory and cache coherence. Shared memory accesses that cross cluster boundaries trap and run a page grain software protocol. The trapping mechanism and renaming of replicated and migrated page grain data is facilitated by virtual memory. Currently, the page grain software protocol is a sequentially consistent write-invalidate protocol.

Because Alewife does not provide hardware support for address translation, we implement virtual memory in software. In the current implementation, software translation adds 17 cycles to every load/store issued to shared memory. This overhead will disappear once the Fugu platform comes online since Fugu provides hardware support for address translation.

3 Results

Using the system described in Section 2, we studied the performance of multi-grain shared memory on the Water application from the SPLASH benchmark suite [7]. The problem size is 4 iterations on 343 molecules. Cluster size was varied from 1 to 32 processors, and page size was varied from 128 bytes to 2 kilobytes. The total number of processors is always 32 independent of the cluster size.

Figure 1 presents the measured data. Execution time is plotted against cluster size, and data points that share a common page size are plotted on the same curve. Notice that each end of the x-axis represents an extreme point in the shared memory design space. At the left-most end, the cluster size is 1 processor. All

*This work was supervised by Anant Agarwal.

†Supported in part by NSF grant # MIP-9012773, in part by ARPA contract # N00014-94-I-09885, in part by an NSF Presidential Young Investigator Award, and in part by LSI Logic and IBM.

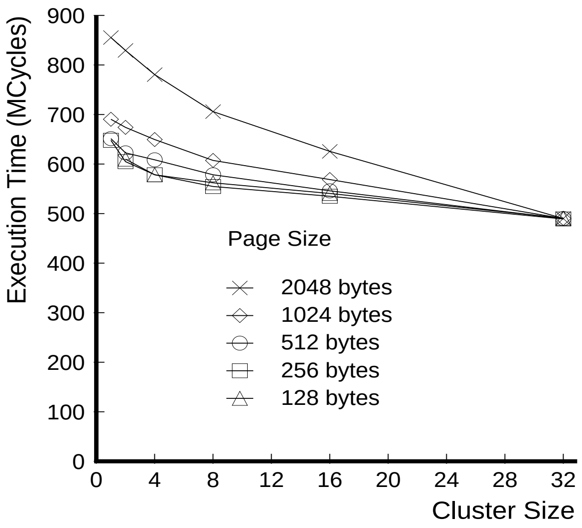

Figure 1: Execution time versus cluster size on the Water application. 4 iterations using 343 molecules were run on 32 processors total. Page size is varied from 128 bytes to 2Kbytes.

shared memory accesses are fulfilled in software, and the unit of coherence is a page. Traditional network of workstation (NOW) systems are represented at this end. At the right-most end, the cluster size is 32 processors. Shared memory accesses are always handled in hardware at cache-line grain. This corresponds to tightly coupled multiprocessor systems. The points in between represent the hybrid shared memory approach that use a mixture of hardware and software shared memory, and a mixture of granularities.

Some interesting results are apparent. First, in all cases, performance improves with increasing cluster size. As cluster size increases, the fraction of shared memory accesses fulfilled in hardware increases as well. This provides higher performance for the Water application. Note however that this will not always be the case. A coarser grained application may benefit from a coarser coherence grain even if the coherence is done in software at higher overheads per coherence operation. Another result is that reducing page size down to 256 bytes improves performance for the Water application, but page sizes below 256 bytes degrade performance. Reducing page size has two effects. First, it reduces false sharing on shared data. False sharing causes unnecessary invalidations and extra coherence overhead. Second, it increases the overhead per unit of data for each coherence operation. In the Water application, false sharing is the bottleneck preventing higher performance at large page sizes. At a page size of 256 bytes, false sharing is no longer significant and coherence overhead becomes the bottleneck. In general, the right tradeoff and thus the optimal page size is application specific and may vary even within the same application. Finally, we note that the page sizes that are best suited for Water are ridiculously small. The performance of the current system is highly sensitive to false sharing because the memory model is sequentially consistent. We expect that a release consistent memory model will allow for much larger page sizes.

4 Future Work

A release consistent version of the system currently running on Alewife is being developed to eliminate the effects of false sharing at larger page grains. In addition, being able to vary the page size is an important mechanism in this system. TLBs in modern processors already support multiple page sizes–super pages [8] and page subblocking [9] are popular techniques. What's needed is a way to choose the appropriate page size based on compiler information or runtime profiling. Finally, more applications need to be studied to understand the behavior of multi-grain shared memory under different workloads.

[1] Daniel Lenoski et. al. *Design of the Stanford DASH Multiprocessor*, Stanford CSL TR 89-403, December 1989.

[2] *Kendall Square Research*, Technical Summary, 1992.

[3] Timothy Mark Pinkston. *Parallel Processor Memory Reference Analysis: Examining Locality and Clustering Potential*, IBM T. J. Watson Research Center RC-15801, May 1990.

[4] Andrew Erlichson, Basem A. Nayfeh, Jaswinder P. Singh, and Kunle Olukotun. *The Benefits of Clustering in Shared Address Space Multiprocessors: An Applications-Driven Investigation.*

[5] Ken Mackenzie, John Kubiatowicz, and Anant Agarwal. *FUGU: Implementing Protection and Virtual Memory in a Multiuser, Multimodel Multiprocessor*, MIT Lab for Computer Science TM-503, October, 1994.

[6] Anant Agarwal et. al. *The MIT Alewife Machine: Architecture and Performance*, Proceedings of the 22nd Annual International Symposium on Computer Architecture, June 1995.

[7] J. P. Singh, W. -D. Weber, and Anoop Gupta. *SPLASH: Stanford Parallel Applications for Shared-Memory*, Stanford CSL TR 92-526, June 1992.

[8] Theodore H. Romer, Wayne H. Ohlrich, Anna R. Karlin, and Brian N. Bershad. *Reducing TLB and Memory Overhead Using Online Superpage Promotion*, Proceedings of the 17th ACM Symposium on Operating System Principles, 1995.

[9] Madhusudhan Talluri and Mark D. Hill. *Surpassing the TLB Performance of Superpages with Less Operating System Support*, Proceedings of the 6th Symposium on Architectural Support for Programming Languages and Operating Systems, October 1994.