

Application Performance on the MIT Alewife Multiprocessor

Frederic T. Chong[†], Beng-Hong Lim[‡], Ricardo Bianchini^{*}, John Kubiawicz[◊], and Anant Agarwal[◊]

[†]Dept. of Computer Science, University of California at Davis

[‡]IBM T.J. Watson Research Center

^{*}COPPE Systems Engineering, UFRJ/Brazil

[◊]Lab. for Computer Science, Massachusetts Institute of Technology

Abstract

This study reports on the performance of several applications on the Alewife machine, focusing on emerging applications and evolving architectural mechanisms. It shows that low-latency cache miss handling mechanisms for *both* local and remote accesses in Alewife make these emerging applications viable candidates for shared-memory parallel processing. The results show that efficient shared memory is an excellent communication mechanism, even for fine-grain applications that do not re-use data. Such applications are thought to favor message-passing. As expected, traditional coarse-grain applications perform well with Alewife’s mechanisms. The results also confirm that hardware support for limited sharing is adequate for a broad range of applications, even on large numbers of processors. Additionally, modeling local cache-miss behavior is important for machines such as Alewife, where remote-miss latencies are only five times longer than local miss latencies. We introduce two novel performance metrics that account for the effect of local misses and are more accurate than previously proposed metrics. We conclude that most applications perform well on Alewife. In particular, fine-grain applications can take advantage of Alewife’s high integration and efficiency to achieve a new level of performance on scalable shared-memory machines.

Keywords: distributed shared memory, multiprocessor, performance metrics, applications, fine grain

1 Introduction

Developments in the architecture of parallel machines influence the evolution of the structure of parallel programs; emerging parallel applications, in turn, impact the future directions in parallel machine architecture. Benchmark suites and architectural mechanisms constantly evolve from the dynamics of the architecture-applications symbiosis.

This study reports on the performance of several applications on the Alewife machine [ABC⁺95] (see Sidebar A), focusing on fine-grain applications and evolving architectural mechanisms. The results show that low-latency miss-handling mechanisms for both local and remote accesses in Alewife make fine-grain applications viable candidates for shared-memory parallel processing. We discover that efficient shared memory is an excellent communication mechanism for fine-grain applications, even without data re-use. This is a very interesting result, given that such applications have long been thought to favor message passing over shared memory.

Not surprisingly, Alewife’s mechanisms allow traditional coarse-grain applications from the SPLASH and NAS benchmark suites to perform well. The results confirm that hardware support for limited sharing is adequate for a broad range of applications, even on large numbers of processors. Local cache miss behavior turns out to be important on multiprocessors with low remote miss latencies. To account for the effect of local misses, we introduce two performance metrics that provide more accurate and revealing results for Alewife than previously proposed metrics.

Coarse-Grain Program	Description	Input Data
APPBT	Solves multiple independent systems of equations	$20 \times 20 \times 20$ floats
BARNES	Simulates movement of bodies under gravitational forces	16K bodies, 4 iters
CGRID	Straightforward 2D successive over-relaxation	512x512 floats, 50 iters
CHOL	Cholesky factorization of a sparse matrix	BCSSTK15 (order 3948, 56934 floats)
FFT	1D Fast Fourier Transform	80000 floats
GAUSS	Unblocked Gaussian elimination	512x512 floats
LOCUS	Routes of wires in a standard cell circuit	3817 wires
MG	3D Poisson solver using multi-grid techniques	56x56x56 floats
MSORT	Sorts a list of integers	64K integers
WATER	Simulates movement of water molecules	125 molecules, 10 iterations
Fine-Grain Program	Description	Input Data
EM3D	Electromagnetic wave propagation through 3D objects	10000 nodes 20% remote neighbors
ICCG	Preconditioned conjugate gradient sparse solver	BCSSTK18 (order 11948, 149090 doubles)
MP3D	Simulates rarefied fluid flow	18000 particles, 6 iterations
MMP3D	Modified MP3D (reduced sharing)	18000 particles, 6 iterations

Table 1: Applications and Kernels

We conclude that both coarse and fine-grain applications can benefit significantly from efficient mechanisms such as those available in Alewife. Fine-grain applications can perform well on scalable shared memory multiprocessors, and they represent an important and emerging class that warrants further study.

The rest of this paper is organized as follows. Section 2 describes the applications in this paper. Section 3 analyzes the performance of the applications on Alewife using several novel performance metrics. Section 4 presents a detailed study of three fine-grain applications, and describes the mechanisms that enhance their performance. Section 5 summarizes the results and Section 6 presents the conclusions drawn from this study.

2 Applications

The applications in this study use the shared-memory programming model. They are written in the C programming language with parallel constructs based on the ANL `p4` library. Most of the programs begin with a master process that allocates and initializes a block of globally shared memory. After initialization, the master spawns a number of slave threads that persist until the end of the computation. Computation is usually partitioned and scheduled statically among the threads that synchronize with locks and barriers.

Table 1 provides a short description of each of the applications and their input parameters. MP3D, BARNES, LOCUS, CHOL, and WATER are from the SPLASH suite [SWG92]. APPBT and MG are part of the NAS parallel benchmarks [Bai94]. The rest of the applications are engineering-type kernels from the University of Rochester, MIT [CA96], and Berkeley [CDG⁺93]. We categorize the applications into traditional, coarse-grain applications and emerging, fine-grain applications.

The traditional, coarse-grain applications are applications that appear in most studies of shared-memory applications and architectures. These include APPBT, BARNES, CGRID, CHOL, FFT, GAUSS, LOCUS, MG, MSORT, and WATER. These applications are relatively coarse-grained and perform well on Alewife.

The emerging, fine-grain applications fail to achieve acceptable performance in most shared-memory multiprocessors because they communicate frequently and are sensitive to memory latencies.

These include EM3D, ICCG, MP3D, and MMP3D. For information on how our benchmarks relate to other benchmark studies, see Sidebar B.

3 Application Performance Characteristics

This section presents the major factors that affect the performance of the applications. First, it considers the processor utilization (the fraction of execution time spent doing useful computation) by each application. Next, to investigate why some of the applications achieve only low to moderate utilization, it presents data on cache hit ratios, computation grain sizes, load balance, and degree of data sharing. For each of the metrics, we plot the data as a series of bar graphs with applications sorted to illustrate overall relationships. Measurements were taken using hardware statistics counters on Alewife (see Sidebar A). Later, Section 5 presents numbers summarizing speedup and efficiency.

3.1 Processor Utilization

The top of Figure 1 presents processor utilization for each of the applications, ranked by average utilization. The results show that LOCUS, MG, MMP3D, APPBT, WATER, FFT, GAUSS, BARNES, and CGRID all attain good utilization of above 60% for 1 through 32 processors. These applications do not present extremely challenging loads to Alewife, and most of them achieve good performance on other multiprocessors.

The results also show that MSORT, EM3D, CHOL, MP3D, and ICCG present more challenging workloads. These applications interest us most in this study as they serve to expose hardware limitations of the multiprocessor. To determine the performance bottlenecks for these applications, we need to consider cache behavior, computation grain sizes, load-balance, and degree of data sharing.

3.2 Local and Remote Misses

Remote cache misses involve another processor’s cache or memory, and are an integral part of every multiprocessor application study. It turns out that local cache misses are also very important on the Alewife machine because an average remote miss is only five times as expensive as a local miss. Although the number of cycles for a cache miss will increase with higher processor-clock rates, the *ratio* of local to remote access times can be sustained (see Sidebar A).

Figure 1 presents the cache hit statistics. An important point is that remote misses in themselves are not indicative of application performance. In particular, note that many applications, such as EM3D, exhibit increasing local cache hit ratios as the number of processors increases. This is because the total available cache increases while the dataset size remains constant. On multiprocessors with low remote to local miss ratios, this local cache effect has a significant impact on performance. To capture the effect of local misses, the bottom graph in Figure 1 presents the hit ratio for a weighted total of memory references, computed by:

$$\frac{(\text{remote hits} + (\text{local hits})/5)}{(\text{remote accesses} + (\text{local accesses})/5)}.$$

where we assume that five local misses are equivalent to an average remote miss. This metric, which we call *weighted hit ratio*, will be instrumental in the next section in computing a *weighted computation granularity*.

Note that ICCG, an application whose algorithm does not significantly re-use data, exhibits a higher remote access hit ratio than might be expected. This is because it uses spin locks for fine-grain synchronization that are often referenced in-cache. Cache misses occur only when the status of the lock changes. The amount of “useful” data re-use is small.

Overall, EM3D, ICCG, and MP3D have lower hit ratios than the other applications. This partly explains the lower utilization of these applications. However, low hit ratios only lead to poor processor utilization when there is little actual computation between memory references. The next section analyzes the amount of computation between cache misses.

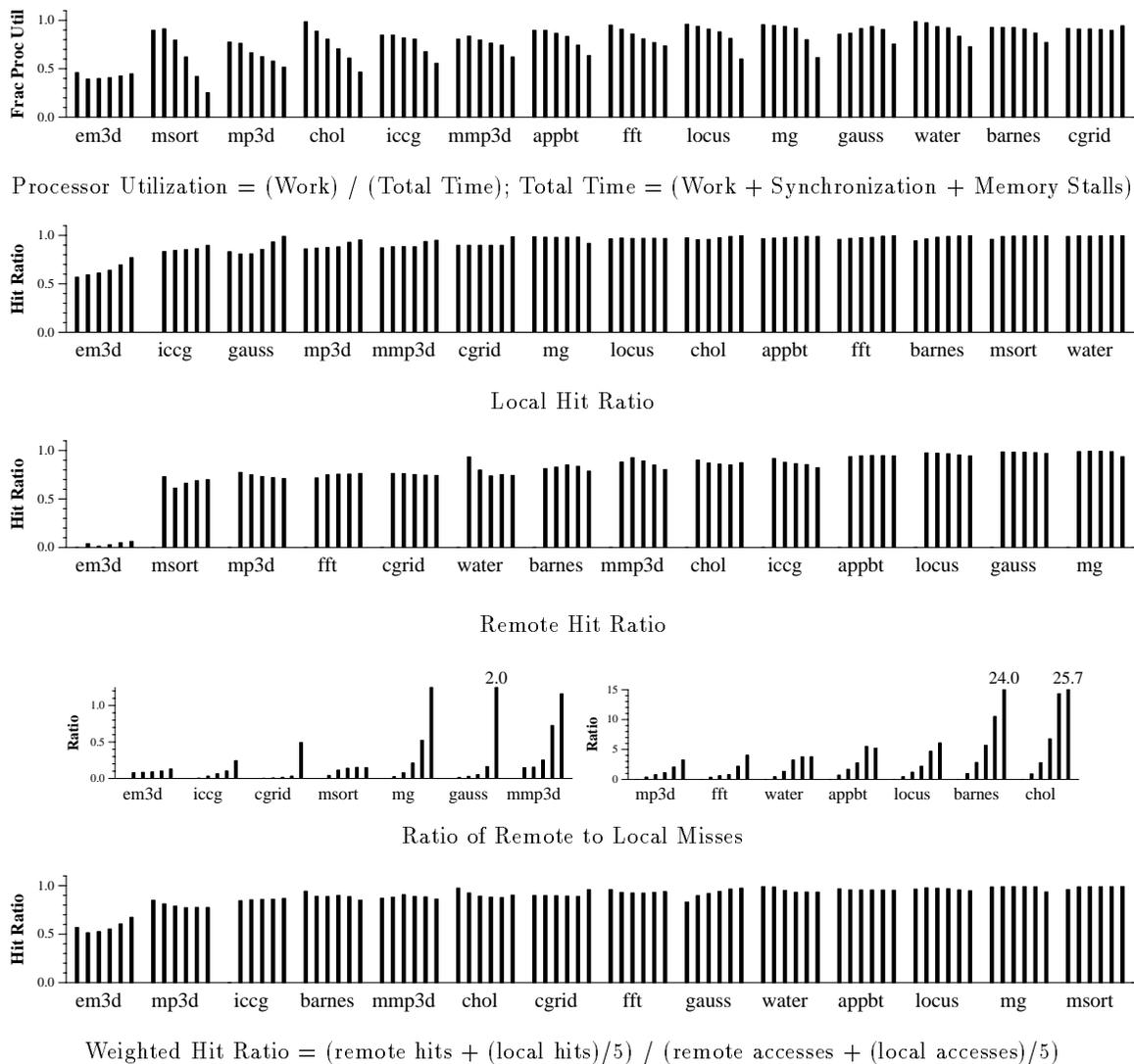
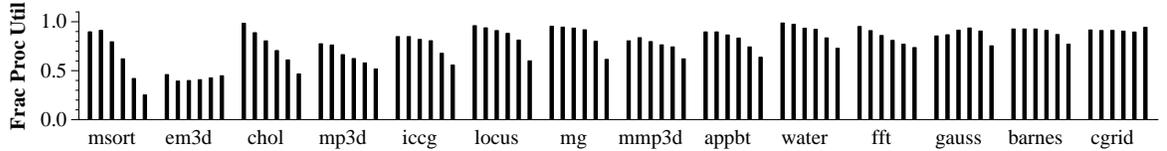
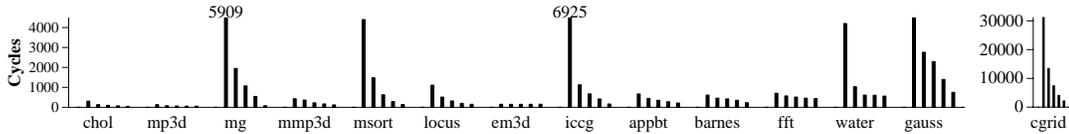


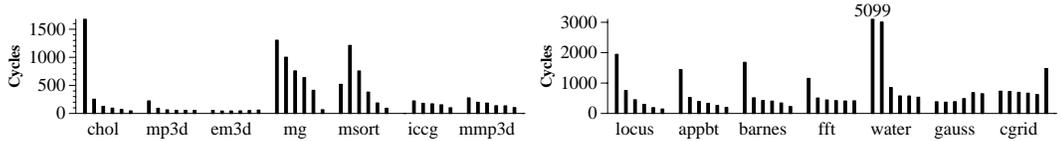
Figure 1: Application characteristics with applications sorted by average. Bars are for 1, 2, 4, 8, 16, and 32 processors for each application. ICCG is missing a 1-processor bar because its dataset does not fit on a single Alewife processor.



Processor Utilization = (Work) / (Total Time); Total Time = (Work + Synchronization + Memory Stalls)



Computation Grain = (Work) / (Remote Cache Misses)



Weighted Computation Grain = (Work) / (Remote Cache Misses + (Local Cache Misses / 5))

Metric	Processors					Avg
	2	4	8	16	32	
Comp / Remote Miss	0.956	0.978	0.993	0.998	0.995	0.984
Comp / Weighted Miss	1.000*	0.985	1.000*	1.000*	1.000*	0.997
* correlation greater than 0.9995						

Correlation of Application Ranking for each Metric Relative to Processor Utilization

Figure 2: Application ranking by processor utilization and granularity. Bars are for 1, 2, 4, 8, 16, and 32 processors for each application and are sorted by 32-processor value. Correlations are computed by comparing, via the *sum squared difference of ranks*, application rankings for each metric and number of processors.

3.3 Computation Granularity

The amount of work per remote cache miss, the *computation grain* of an application, has traditionally been a good indicator of performance. Figure 2 shows that *computation grain* is well-correlated with processor utilization. However, the effect of local cache misses is also very important for machines with low remote-to-local memory latency ratios. Thus, we introduce another metric, called *weighted computation granularity*, the amount of work per weighted cache miss. As Figure 2 shows, this weighting produces a better metric which is more correlated to utilization than granularity derived from remote misses alone.

In particular, applications with high local miss rates and fine granularity, such as EM3D and ICCG, are in more appropriate rank order with weighted granularity than with unweighted granularity. Also note that for both granularity metrics, the ranking for MSORT is considerably different from the utilization ranking. This is because neither granularity metric incorporates any notion of load balance. Section 3.4 examines the effect of load balance.

Together, the granularity and cache hit data show that the low utilization for EM3D, MP3D, and ICCG are because these applications are naturally fine-grained. Their performance is primarily determined by *both* local and remote memory access latencies.

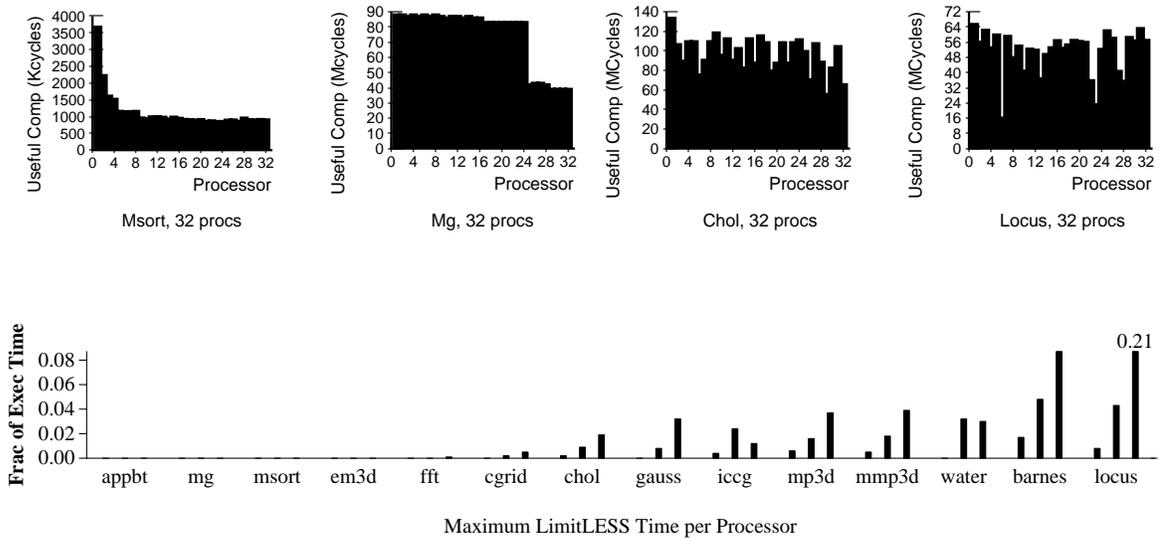


Figure 3: Load balance and LimitLESS software traps. Load imbalance is a factor in MSORT, MG, CHOL, and LOCUS. LimitLESS plays little role in performance except for LOCUS. Bars are for 8, 16, and 32 processors for each application.

3.4 Load Balance

As the number of processors increases, an application needs to spread its computation evenly among the processors in order to maintain good utilization. Of the applications, MSORT, MG, CHOL, and LOCUS are susceptible to load imbalance on large numbers of processors. Figure 3 plots the load on 32 processors for these applications.

The load imbalance in MSORT occurs because the number of lists to merge decreases as the computation progresses. The load imbalance in MG occurs for two reasons. First, the multigrid algorithm uses a hierarchy of grids. The smallest grids have fewer points than processors. Second, our $56 \times 56 \times 56$ dataset does not divide evenly into all processor sizes (this size was chosen to fit into the memory of a single processor and still be interesting).

CHOL and LOCUS are dynamically load-balanced applications, where idle threads request work from global task queue(s). Evidence from CHOL [WOT⁺95] indicates that larger input data sets can help load balance. Of the four applications, MSORT is the most severely affected by load imbalance.

3.5 Sharing

The results show that most applications exhibit limited read sharing and are well-supported by Alewife’s LimitLESS combination of hardware and software coherence. LimitLESS supports limited read-sharing in hardware and relies on software handling for widely-shared data items. The first five processors to request a data item are handled directly by the CMMU. If a sixth processor requests a copy of this data item, the CMMU generates an interrupt to the local processor, requesting software help to keep track of coherence information.

Figure 3 shows the maximum fraction of time spent executing coherence handlers on any single processor. Although some applications, such as ICCG, access the same data on many processors, frequent writes prevent high levels of read sharing. As a result, many of the applications do not invoke LimitLESS at all. Only LOCUS and BARNES spend a significant fraction of execution time handling coherence in software, which explains why utilization drops as the number of processors increase for these two applications.

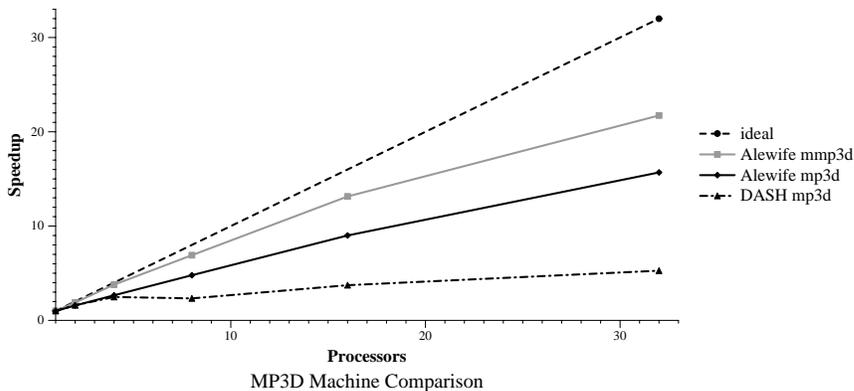


Figure 4: Comparison of MP3D speedup on Alewife versus on DASH.

4 Emerging Fine-Grain Applications

Applications such as MP3D are often considered too naive and fine-grained to be important benchmarks. This naiveté, however, is exactly what needs to be supported for the benefit of both compilers and users. After all, a major motivation for shared-memory is its ease of programming over message-passing. Furthermore, less naive codes such as EM3D and sophisticated codes such as ICCG are inescapably fine-grained. This section discusses these three applications in detail and demonstrates why they are now emerging as viable applications on shared-memory multiprocessors.

4.1 MP3D

MP3D needs efficient support for migratory data. While the particles in the simulation are represented by data that are statically assigned to processors, the wind tunnel through which the particles move is represented by data that migrate frequently. Figure 4 compares the performance of MP3D on Alewife with Stanford DASH [LLG⁺92].

Alewife achieves substantially higher speedups than DASH on MP3D. The primary reason is Alewife’s coherence protocol that is better suited for migratory data. On DASH, a remote dirty cache line owned by processor 1 is not invalidated, but is “downgraded” to remote clean/read-only copies at the time that processor 2 attempts to read it. When processor 2 subsequently attempts to write this line, it must first invalidate processor 1’s copy. Since Alewife performs the invalidation as part of the read transaction, the subsequent write transaction is faster. This savings becomes especially important when data is primarily read then written by different processors throughout an application.

MMP3D, with more optimized code and data mapping, performs even better. This suggests that this application needs either efficient mechanisms or intelligent data mapping to perform well. The next two applications need both.

4.2 EM3D

EM3D is one of the finest-grain applications in the literature. The input dataset consists of a randomly generated graph where 20 percent of the edges are between nodes mapped on different processors. EM3D is thought to favor message-passing, since it is a graph computation where nodes in the graph have a producer-consumer relationship. Alewife exhibits good performance on EM3D, in spite of the fact that invalidate protocols are sub-optimal for producer-consumer computation. As discussed earlier, local cache behavior accounts for much of the good speedup.

On 32 Alewife processors, each edge takes 107 cycles of computation. For comparison, only a few results are available in the literature. A Berkeley Split-C study [CDG⁺93] achieved 56 cycles

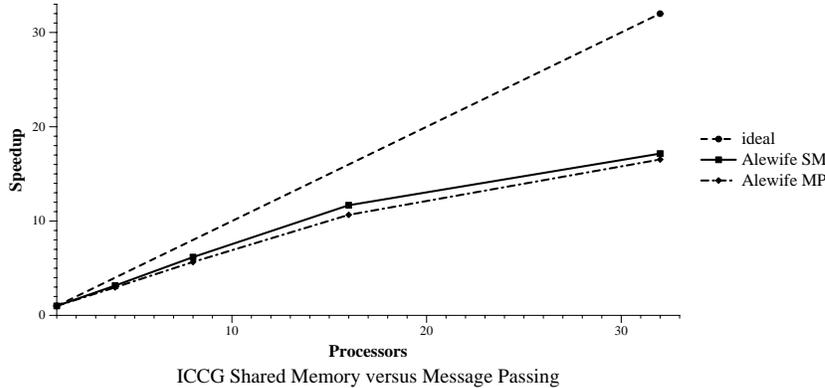


Figure 5: Comparison ICCG with shared memory versus message passing on Alewife.

per edge with message-passing implementation on the Thinking Machines CM-5 on 64 processors. A Wisconsin study [CLR94] simulates shared-memory and message-passing machines with hardware configurations based closely on the CM-5. It finds that an invalidation-based shared-memory implementation (172 cycles/edge) is twice as slow as a message-passing implementation (86 cycles/edge).

Alewife performs substantially better than previous shared-memory results and competitively with message-passing implementations. The 5-to-1 remote-to-local miss ratio allows performance to scale well as the number of processors increases, even with an invalidation-based protocol.

4.3 ICCG

ICCG is a difficult application to parallelize and has only recently attained reasonable performance on multiprocessors. With only 6 to 10 floating-point operations per double word of data communicated, ICCG has historically been unparallelizable. ICCG is similar to EM3D in that its kernel involves a producer-consumer graph computation. However, ICCG uses real scientific datasets mapped onto the machine with the best-known algorithms. More importantly, the sparse triangular solve kernel in ICCG requires finer-grained synchronization than the relaxation computation in EM3D. To synchronize, shared-memory ICCG uses read-modify-write operations, where the producer of a value can perform an accumulate to a variable on a remote processor. Consequently, the performance of ICCG is latency-critical: each read-modify-write operation requires three round-trip messages between processors upon a remote miss.

In contrast with three messages, an active message implementation needs only a single message. Figure 5 makes this comparison. Surprisingly, the shared-memory implementation performs slightly better, since shared memory references are handled directly in hardware and incur minimal overhead, as compared to active messages that have to be handled in software. Even when added to round-trip network latencies, shared-memory latencies are much lower than those for message passing.

5 Summary

We summarize by presenting overall speedup results and revisiting the major factors leading to those results. Figure 6 presents the familiar overlapping speedup curves for the applications. It also presents a more digestible picture of the fraction of ideal speedup sorted by average.

Most of the traditional coarse-grain applications perform well: APPBT, BARNES, CGRID, FFT, GAUSS, MG, and WATER. LOCUS causes a large number of LimitLESS traps which results in high memory latencies. CHOL suffers from limited parallelism due to a small dataset size. MSORT suffers from load imbalance.

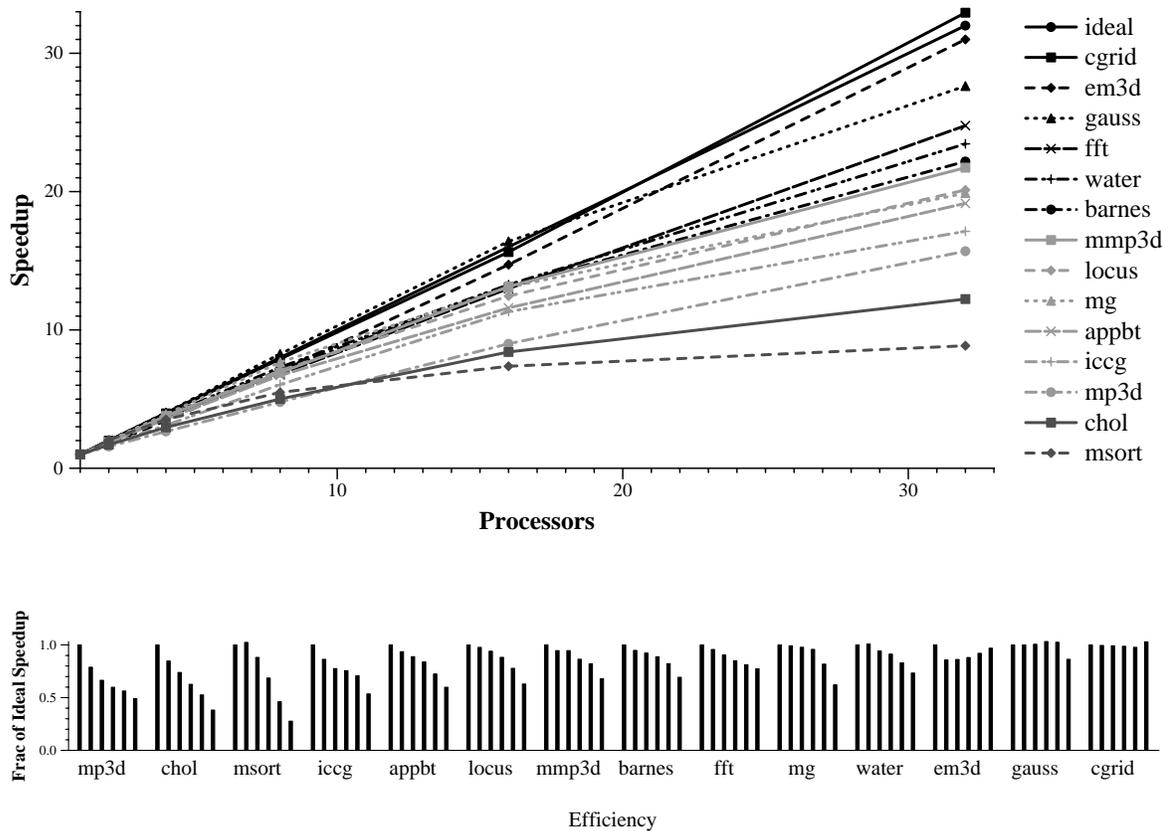


Figure 6: Application Speedups and Efficiency. Speedups are relative to the parallel code running on a single processor. Since the dataset does not fit on a single Alewife node, ICCG speedups are based upon single-processor times from the CM-5, a SPARC-based architecture with nearly identical single-node performance. Efficiencies are sorted by average efficiency. Bars are for 1, 2, 4, 8, 16, and 32 processors for each application.

The fine-grain applications exhibit new levels of performance. EM3D performs surprisingly well due to local cache behavior and fast communication. MP3D has the worst efficiency, but Alewife speedups are high when compared to other machines. MMP3D performs even better. ICCG also performs unexpectedly well for a shared-memory multiprocessor.

6 Conclusions

This paper presents the performance of several applications on the Alewife machine, focusing on emerging applications and the architectural mechanisms that allow them to perform well on the machine.

The main contributions of this paper are:

- It determines the mechanisms that allow fine-grained applications to perform well on Alewife: low local to remote memory access latency ratios; hardware support for shared-memory computing; and a coherence protocol optimized for migratory sharing.
- It characterizes the performance of a large number of applications in terms of several metrics: processor utilization, local and remote miss ratios, computation granularity, load balance, and sharing behavior. In addition to these well-known metrics, it introduces two new metrics that account for local and global misses and their overheads. Previous metrics mispredict performance on machines such as Alewife.

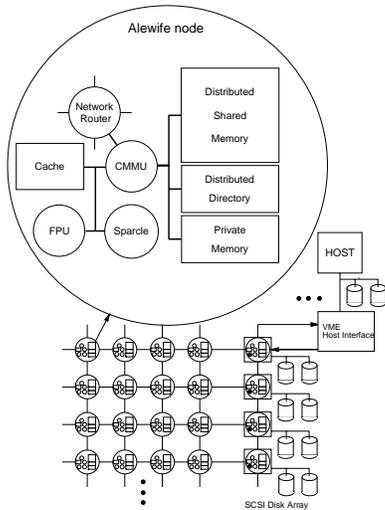
Fine-grain applications can perform well on scalable shared-memory multiprocessors. These applications are an important emerging class that warrants further study. Overall, both coarse and fine-grain applications can benefit significantly from the efficient mechanisms such as those available in Alewife.

7 Acknowledgements

Thanks to Fredrik Dahlgren and our anonymous referees.

References

- [ABC⁺95] Anant Agarwal, Ricardo Bianchini, David Chaiken, Kirk Johnson, David Kranz, John Kubiatowicz, Beng-Hong Lim, Ken Mackenzie, and Donald Yeung. The MIT Alewife machine: Architecture and performance. In *Proc. 22nd Annual International Symposium on Computer Architecture*, June 1995.
- [AG88] Anant Agarwal and Anoop Gupta. Memory-Reference Characteristics of Multiprocessor Applications under MACH. In *Proceedings of ACM SIGMETRICS 1988*, pages 215–225, May 1988.
- [Bai94] D. Bailey et al. The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, March 1994.
- [CA96] Frederic T. Chong and Anant Agarwal. Shared memory versus message passing for iterative solution of sparse, irregular problems. Technical Report MIT/LCS/TR-697, MIT Laboratory for Computer Science, September 1996.
- [CDG⁺93] David E. Culler, Andrea Dusseau, Seth Copen Goldstein, Arvind Krishnamurthy, Steven Lumetta, Thorsten von Eicken, and Katherine Yelick. Parallel programming in Split-C. In *Supercomputing*, November 1993.
- [CLR94] Satish Chandra, James R. Larus, and Anne Rogers. Where is time spent in message-passing and shared-memory programs. In *ASPLOS VI*, pages 61–73, San Jose, California, 1994.
- [DRPS87] F. Darema-Rogers, G. F. Pfister, and K. So. Memory Access Patterns of Parallel Scientific Programs. In *Proceedings of ACM SIGMETRICS 1987*, pages 46–58, May 1987.
- [EK88] S. J. Eggers and R. H. Katz. A Characterization of Sharing in Parallel Programs and Its Application to Coherency Protocol Evaluation. In *Proceedings of the 15th International Symposium on Computer Architecture*, New York, June 1988. IEEE.
- [LLG⁺92] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica S. Lam. The stanford Dash multiprocessor. *Computer*, 25(3):63–80, March 1992.
- [SWG92] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. SPLASH: Stanford parallel applications for shared-memory. *Computer Architecture News*, 20(1):5–44, March 1992.
- [WOT⁺95] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, , and Anoop Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.



Miss Type	Home Location	# Inv. Msgs	hw/sw	Miss Penalty	
				Cycles	μ sec
Load	local	0	hw	11	0.55
	remote	0	hw	38	1.90
	remote (2-party)	1	hw	42	2.10
	remote (3-party)	1	hw	63	3.15
	remote	–	sw [†]	425	21.25
Store	local	0	hw	12	0.60
	local	1	hw	40	2.00
	remote	0	hw	38	1.90
	remote (2-party)	1	hw	43	2.15
	remote (3-party)	1	hw	66	3.30
	remote	5	hw	84	4.20
	remote	6	sw	707	35.35

[†] This sw read time represents the throughput seen by a single node that invokes LimitLESS handling at a sw-limited rate.

Typical shared memory miss penalties.

Figure 7: The MIT Alewife multiprocessor

Sidebar A: The Alewife Multiprocessor

Figure 7 shows an overview of the architecture[ABC⁺95]. The nodes in an Alewife machine can communicate via either shared memory or message passing. Each node consists of a Sparcle processor (a modified SPARC), a floating point unit, 64K bytes of direct-mapped cache, 8M bytes of DRAM, an Elko-series 2D-mesh routing chip (EMRC) from Caltech, and a custom-designed Communication and Memory Management Unit (CMMU). This paper uses a 32-node version of the Alewife machine with a 20MHz Sparcle processor and EMRC network routers operating with a per-link bandwidth of 40M bytes/second.

As shown in Figure 7, the single-chip CMMU is the heart of an Alewife node. It is responsible for coordinating message passing and shared memory communication as well as handling more mundane tasks such as DRAM refresh and control. It implements Alewife’s scalable LimitLESS cache coherence protocol, and provides Sparcle with a low-latency interface to the network. To communicate via shared-memory, users simply read/write from the shared address space; the CMMU takes care of the details of acquiring remote data and caching the results locally. Similarly, users send and receive messages by accessing hardware network queues directly through the CMMU. To aid experiments, the CMMU contains hardware statistics counters that allow non-intrusive monitoring of a wide-array of application performance characteristics.

Although Alewife’s clock-rate and network bandwidth are slower than the current state of the art, its architecture is balanced in a way that allows conclusions about current and future machines. There are three reasons for this: First, Alewife has a low *ratio* of remote to local cache miss times (on average this ratio is approximately five). Current technological trends argue that this ratio will stay constant or even decrease. Assuming memory controllers that pipeline accesses between DRAM and the network, and a reasonable matching of DRAM and network bandwidth, the dominant factors in the remote/local ratio are DRAM access time and network latency. Over the last decade, DRAM access times have stayed roughly constant while processor speeds have increased, and network latency times have decreased. The result is that Alewife’s average ratio of a 5-to-1 remote to local access time is conservative relative to future architectures.

Second, Alewife’s network has a per-node latency of 2 processor cycles/network hop which is not unreasonable with a direct point-to-point network such as a mesh, even for high processor clock rates. Finally, the ratio of network bandwidth to processor clock rate of from 1.5 to 2 is also in line with modern technology.

Sidebar B: Application Studies

The cycle of influence between shared-memory multiprocessor architecture and applications has been in motion for at least a decade, and continues to this day. The early studies, based on trace-driven simulation of shared-memory applications, focused on the memory reference characteristics [AG88][DRPS87] and coherence-protocol behaviors [EK88] for small-scale shared-memory multiprocessor architectures. Later studies [SWG92][WOT⁺95], based on execution-driven simulators, characterize in more detail the behavior of shared-memory applications on larger-scale shared-memory architectures. They include the effect of load imbalance, synchronization overhead and parallelization overhead in addition to cache and memory behavior. An important result of research on shared-memory applications is a set of SPLASH and NAS benchmarks that has driven much research into shared-memory architectures and cache-coherence protocols.

During the same period, several experimental, shared memory multiprocessors were designed and built. Stanford DASH was the first such machine, followed by MIT Alewife, and most recently by the Wisconsin Typhoon prototypes. Researchers have used these machines as a vehicle to characterize the behavior of shared-memory applications [LLG⁺92][ABC⁺95]. This study uses the built-in statistics hardware in Alewife to characterize the performance of shared-memory applications in this paper. It considers emerging fine-grain applications as well as the SPLASH and NAS applications. The analysis focuses on computation granularity, cache miss ratios, sharing patterns, and load balance as primary determinants of application performance. Unlike previous studies that focused on remote cache misses, it considers *both* local and remote cache misses. It does not provide details on the cause of cache misses in each of the applications; Woo *et al.* [WOT⁺95] already provides such details. This study is most closely related to [LLG⁺92], due to the similarities between DASH and Alewife. A comparison of the results from both studies shows that Alewife is more successful at fine-grain or irregular applications, primarily due to its shorter memory latencies. Both machines perform comparably well for coarse-grain applications.

With the availability of both scalable shared-memory machines and simulators, there is a debate over whether to use machines or simulators to characterize shared-memory applications. Real machines can execute more realistic programs and inputs, and their measurements capture all the nuances of a real hardware implementation. The disadvantage is that it is hard to compare different architectures, and some observed effects may be due to an artifact of the machine rather than some fundamental principle. This study uses the machine-based approach. The ideal approach is to use both machines and simulators for application studies, and cross-check the results from one with the other.

Fred Chong is an Assistant Professor of Computer Science at the University of California at Davis. He received his BS, MS, and PhD degrees in electrical engineering and computer science from MIT. His current work focuses on architectures and applications for multigrain parallel systems. His research interests include communication, applications, theory, and VLSI for parallel and distributed systems.

Beng-Hong Lim is a research staff member at the IBM T.J. Watson Research Center. He is part of a research team that is designing IBM's next-generation scalable multiprocessor systems. His current research interests are in the architecture, operating systems, and programming models for scalable and reliable high-performance computing. He received BS, MS and PhD degrees in electrical engineering and computer science from MIT, where he served as one of the principal members of the Alewife project.

Ricardo Bianchini is a research associate at COPPE Systems Engineering/UFRJ, where he co-leads the NCP₂ project, a large research effort focused on building hardware support for software-coherent distributed shared-memory systems. He received his Ph.D. degree in Computer Science from the University of Rochester in 1995. His research interests include parallel and distributed computing, advanced operating systems, and parallel computer architecture.

John Kubiawicz is a doctoral candidate in the Department of Electrical Engineering and Computer Science at MIT. His current research interests include parallel computer architecture, high-performance microprocessor design, artificial life, and high-energy particle physics. He received BS degrees in electrical engineering and physics and an MS in electrical engineering from MIT.

Anant Agarwal received his B.Tech at the Indian Institute of Technology in Madras, India, in 1982, and his M.S. and Ph.D. at Stanford University in 1987. Currently, he is an Associate Professor of Computer Science and Electrical Engineering at MIT, where he leads the Alewife and RAW Projects.