Submitted to HPCA '95

The Impact of Message Traffic on Multicomputer Memory Hierarchy Performance

Scott Pakin Andrew Chien

Department of Computer Science
1304 W. Springfield Ave.
Urbana, IL 61801
{pakin,achien}@cs.uiuc.edu

July 24, 1994

Abstract

Multicomputer cache performance is highly sensitive to interprocessor message traffic. The widening gap between microprocessor speeds and primary memory latencies means slight increases in cache miss rate can have a severe impact on application performance. It is therefore critical to reduce cache misses. While there are a number of factors that may contribute to an increase in cache misses, of particular concern is multicomputer message traffic. In this paper, we examine the extent to which handling message traffic increases cache misses.

Using register-transfer-level trace-based simulators of two representative multicomputer nodes, we performed extensive tests (over 70 billion memory references) with a variety of system configurations. Our results indicate that message traffic has a tremendous impact on multicomputer node performance. A Paragon-style multicomputer with a 16 kilobyte (KB) cache that processes an average of one word of message data per 50 clock periods can be expected to observe up to 20% more misses on a given application than a similarly-configured uniprocessor. Furthermore, increasing the cache size increases the importance of messaging. For example, doubling the cache size from 16KB to 32KB decreases cache misses by 55% for one application on a uniprocessor but by only 35% on a CM-5-style multicomputer. Finally, we show that control and data messages (i.e. short and long messages) have different effects on the cache. We quantify the regions of message length space in which each multicomputer node architecture performs best.

Keywords: message traffic, memory hierarchy, cache, multicomputer, network

1 Introduction

Multicomputers—collections of independent, cooperating microprocessors that communicate by sending and receiving messages over a high-speed interconnection network—are a desirable platform for high-performance computing because of their scalability and comparatively low cost. However, multicomputing introduces new issues regarding performance. Multicomputer performance is largely determined by two factors: the sequential performance of the component microprocessors and the latency and bandwidth of the interconnection network. In light of the former, we investigate whether microprocessors perform equally well in multicomputers as they do in uniprocessor systems.

Cache miss rate is a critical factor in microprocessor performance, due to the large disparity between cache and primary memory access times. A cache miss idles the microprocessor until the data loads from primary memory.[1] Thus, satisfying a cache miss can be as expensive as executing tens or even hundreds of instructions, and so it is crucial to identify and remove potential sources of cache misses from a system or application design. In a multicomputer, interprocessor message traffic can increase the miss rate, reducing cache performance. This reduces the microprocessor's sequential performance and, ultimately, that of the multicomputer as a whole. The performance penalty is determined by how much message traffic affects the cache for various memory-network interfaces and applications.

There are two basic approaches to interfacing memory hierarchies and networks. Machines like the Thinking Machines CM-5 [16] and Fujitsu AP1000 [14] logically connect the network to the cache. That is, they directly involve the cache in message sends and/or receives. An alternative design is to logically connect the network to primary memory, using DMA transfers for interprocessor communication. Such an approach is utilized by a wealth of machines, for instance the Intel Paragon [10], Cray T3D [6], MIT Alewife [11], Bull/ECRC/ICL/Siemens EDS [17], Meiko CS-2 [9], and Caltech Mosaic-C [13], as well as systems designed around the Inmos T9000 Transputer [12].

The tradeoff between the two interfaces is—at the highest level—parallel performance versus sequential performance. Transmitting messages between the network and the cache benefits parallel performance because it reduces messaging latency. On the other hand, transmitting messages between the network and primary memory benefits sequential performance because it avoids displacing program data.

We performed trace-based simulations of CM-5- and Paragon-style multicomputers. The former, like the CM-5, uses the CPU to transport data between the network and the *cache*. The latter, like the Paragon, uses the DMA subsystem to transport data between the network and *primary memory*. The simulators are otherwise identical. We based our simulators on the CM-5 and Paragon because they are established commercial machines and because their designs are fairly straightforward. That is, they lack such features as the Alewife's integrated cache coherence mechanisms or the CS-2's ability to perform remote memory reads.

Our simulations show that with typical system and application parameters—a 16KB cache size and a message rate of one word per 50 CPU clock cycles—message traffic increases the number of microprocessor cache misses by an average of 9.84% on the Paragon-style system and 85.12% on the CM-5-style system.[2] In short, message traffic does make a difference. The large increase in cache misses implies that the logical connection of the network interface to the memory hierarchy is crucial to microprocessor—and hence, overall system—performance.

The contributions of this paper are:

- An architectural comparison of two, established memory–network interfaces performed without bias towards specific implementations
- A quantitative analysis of cache sensitivity to message traffic for a number of applications in a variety of scenarios
- A study of the effect of cache size and message length on cache sensitivity to message traffic

In the remainder of this paper, we present a detailed comparison of the CM-5- and Paragon-style node architectures in a variety of scenarios and provide evidence that multicomputer message

[1] Even with a non-blocking cache, a microprocessor will still stall if it requires that data in order to make progress.

[2] **Definition:** new # of misses = old # of misses × (100% + percent increase)

passing *does* significantly degrade sequential performance for both architectures. Section 2 discusses the differences and tradeoffs between the CM-5 and Paragon styles of logically connecting the network to the memory hierarchy. Section 3 details our simulators and experimental methodology. We present our results and analysis in Section 4 with an in-depth discussion of the ramifications in Section 5. Section 6 contains a summary and our conclusions. Finally, we suggest future directions for this research in Section 7.

2 The Problem

2.1 Interfacing Networks and Memory Hierarchies

We want to find the best place to logically connect a network to the memory hierarchy. When a message arrives at a node in a multicomputer, it is generally placed in the node's local memory hierarchy for processing. Similarly, when a node wants to send a message, the message must be read from the node's local memory hierarchy. Finding the best way to logically connect the memory hierarchy to the network is important because the logical connection can have a significant impact on performance. If messages are dealt with at too high a level in the memory hierarchy, they can displace frequently-referenced data. If messages are dealt with at too low a level in the hierarchy, the cost to access the message data will be high.

In short, the tradeoff is one of messaging latency versus sequential node performance. Fine-grained applications—those that execute few CPU instructions between message sends—generally require quick access to message data, even at the cost of displacing other data. In contrast, coarse-grained applications—those that execute many CPU instructions between message sends—can often tolerate a high access cost for message data to preserve data high in the memory hierarchy. In this paper, we do not focus directly on the access cost, but rather direct our attention to the issue of displacing heavily-referenced data at high levels of the memory hierarchy.

2.2 CM-5 System

Figure 1 shows the path that incoming messages take on the CM-5. First, messages enter the network interface from the interconnection network ($\boxed{1}$). The processor then reads the data from the network interface into registers[3] using ordinary memory read instructions ($\boxed{2}$). The CPU writes the data into memory (really the cache) in preparation for the user's code to access it ($\boxed{3}$). And, finally, the user's code reads the data into registers as it would any data ($\boxed{4}$). Note that there is no guarantee the line containing the message data will not be replaced in the interim, incurring cache misses when the message is processed (gray lines in figure).

Sending a message from CPU registers to the network (Figure 2) is the reverse process. The user's code writes the message into a buffer ($\boxed{1}$). (The gray lines in the figure represent the case where some or all of the data is displaced from the cache into memory.) Then, the microkernel reads the message from the buffer ($\boxed{2}$) and sends the data to the network interface ($\boxed{3}$), which forwards it into the network ($\boxed{4}$).

Clearly, the advantage of the CM-5's messaging scheme is that it is possible for message transmission and reception to occur entirely within cache memory, without accessing the slower primary memory. The disadvantage is that caching short-term message data displaces more long-term program data.

[3] But not into the cache; the addresses mapped to the network interface are never cached

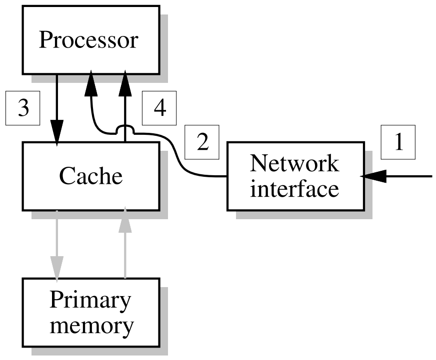

Figure 1: CM-5 incoming message path

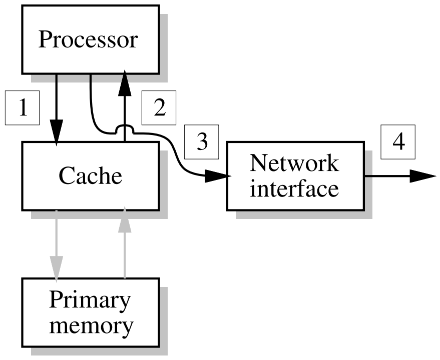

Figure 2: CM-5 outgoing message path

2.3 Paragon System

The Paragon takes a different approach to moving data to and from the network. Message reception is shown in Figure 3. After a message enters the network interface ($\boxed{1}$), it is written to primary memory ($\boxed{2}$). When the user's program reads the received data into registers, it will miss in the cache and have to load the data from primary memory ($\boxed{3}$ & $\boxed{4}$).

As on the CM-5, sending a message on the Paragon (Figure 4) is the reverse of receiving one. The message must first be written to primary memory ($\boxed{1}$ & $\boxed{2}$). From there, the message is sent to the network interface ($\boxed{3}$) and the network ($\boxed{4}$).

The advantage of the Paragon's approach is that interprocessor communication does not affect the cache until the data is actually used. The disadvantage is that primary memory must be accessed multiple times for each message transmitted. Because primary memory latency plus DMA overhead is fairly large relative to cache latency, the resulting end-to-end communication latency is high.

3 Experimental Framework

We pursue an experimental approach to evaluating memory hierarchy–network interfacing alternatives. System designers need to know what memory hierarchy–network interface has superior overall performance. Also, application writers could use such information to explain performance data. Unfortunately, analytic modelling of the memory–network interface is infeasible because of

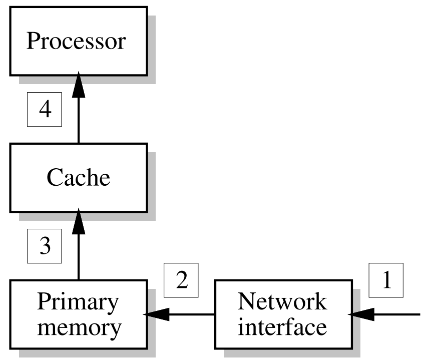

Figure 3: Paragon incoming message path

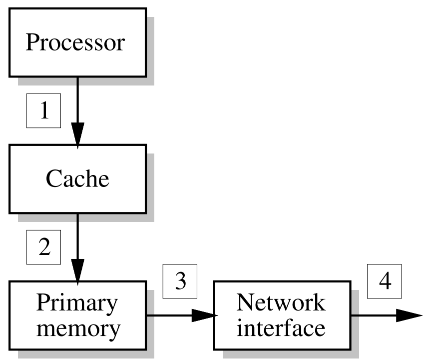

Figure 4: Paragon outgoing message path

the complexity of memory and network access patterns. Furthermore, it is meaningless merely to code the same application for both types of machines and compare execution times on the actual systems, as there are far too many variables, including different processors, clock speeds, network topologies, chip technologies, and run-time systems.

A better approach is to use *simulators* to compare systems that are identical except for the way they handle interprocessor communication. Simulators have the following advantages:

- Systems can be compared fairly and accurately.
- Results are repeatable.
- Parameters can be precisely controlled.

3.1 Simulator

Because the goal is to compare node architectures, not specific machines, we separate *architecture* and *implementation*. To do so, we constructed register-transfer-level trace-based simulators of a generic uniprocessor node and two multicomputer nodes—a CM-5-style node and a Paragon-style node—using the Parsim simulation engine [2] and the CARL simulation language [1].

The conceptual model for the simulator is that the processor continuously issues memory read and write requests, pausing occasionally to send a message to the network. In addition, a message will periodically arrive at the node from the network, at which point the network interface interrupts the processor to notify it of the message. In the CM-5-style simulator, the processor then reads the data from the network interface's message queues and writes it to memory, while in the Paragon-style simulator, the processor sets up registers in the direct memory access (DMA) controller, which then copies the message directly to primary memory.

The logical structure shared by all three versions of the simulator is shown in Figure 5 and contains the following modules, the first four of which are based specifically on the DECchip 21064-AA implementation [8] of the Alpha architecture:

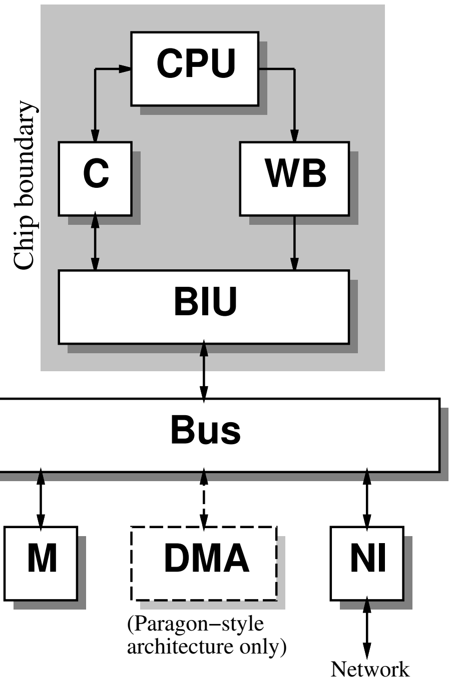

Figure 5: Logical structure of a simulated multicomputer node

CPU 64-bit CPU, assumed 100% hit rate for translation lookaside buffer (TLB) and instruction cache, continuously issues memory read/write requests taken from memory-reference traces of actual programs (i.e. *only* memory instructions are simulated)

C physically-addressed direct-mapped cache, four 64-bit words per line, read-allocate scheme for misses

WB four-line-deep write buffer, four-word lines, write merging of multiple writes to the same line—see [8] for further details

Bus 128 data + 32 address lines

BIU bus interface unit, arbitrates fairly between cache and write buffer for bus control

M primary memory

NI network interface, provides memory-mapped message first-in first-out buffers (FIFOs)

DMA direct memory access controller, bridges network interface and memory, used only in Paragon-style architecture

In addition, we simulate a *network fringe* (Figure 6). The network fringe embodies the actions of the remainder of the multicomputer (everything external to the node). Specifically, it sends messages to the network interface at a given rate and accepts any messages sent by the node.

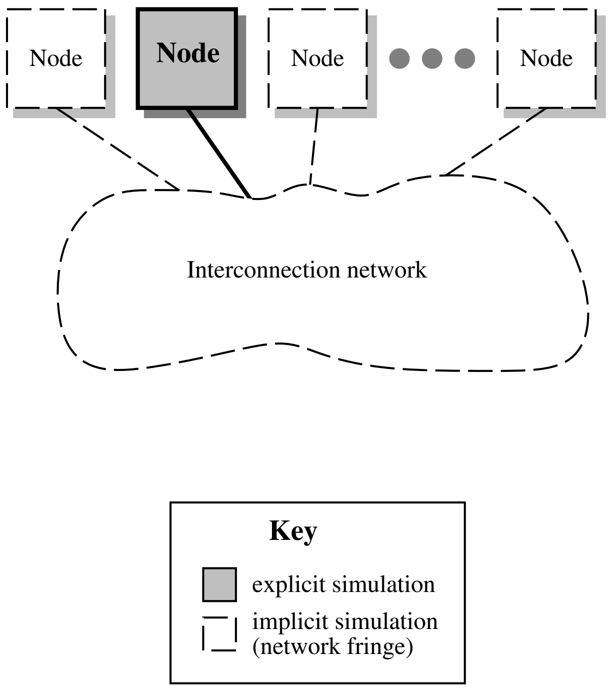

Figure 6: Network fringe

The simulators accurately model component and system functionality. But because all simulation-based research must make tradeoffs between accuracy and control/real-time performance, we made the following simplifications:

- Non-memory instructions execute in zero time.
- There is no operating system or run-time system.
- A single node surrounded by a network fringe possesses approximately the same characteristics as a set of intercommunicating nodes.

Those simplifications are reasonable because:

- It is unnecessary to simulate non-memory instructions in a study concerned exclusively with memory hierarchy performance.
- Operating systems and run-time systems add little insight to what is already present in our model. That is, it is more important to know that a cache miss occurred than that it occurred in either user space or operating system space.
- Surrounding a single node with a network fringe provides the flexibility to fine-tune the message rate and message length.

3.2 Trace Data

We selected application traces based on the following criteria:

- scientific nature
- programming language
- locality
- size

Information about the applications is given in Table 1. The applications are taken from well-known benchmark suites. They are written in different languages (C and Fortran). They exhibit a range of localities. And they each make enough memory references to provide meaningful data, but not so many as to take an inordinate amount of time to simulate. We use the same trace files for both the uniprocessor and multicomputer simulators.[4]

3.3 Simulation Parameters

Table 2 lists the four simulation parameters we varied in each experiment. Entries marked as "default" were used except where otherwise noted. The following sections detail the simulation parameters and the range of values.

3.3.1 Node Architecture

We simulated two memory-network interfaces. One, like the CM-5, logically connects the network to the cache. The other, like the Paragon, logically connects the network to primary memory. In addition, we simulated a uniprocessor system—a system lacking a network and network interface, yet otherwise identical to the multicomputer systems—for use as a base case.

3.3.2 Message Rate

In order to examine the relationship between message traffic and the cache miss rate, we varied the rate at which message data are sent between the processor and network. Messages are introduced in a Poisson distribution.

Data provided in [7] show that for the machines and applications examined in that paper, the message rate varies from 0.00004 to 0.00674 64-bit words per clock, with a mean of 0.00114. However, those numbers include the time spent executing non-memory instructions, while our simulations execute exclusively memory instructions. In other words, the numbers calculated from [7] are unrealistically low for a memory-only study. Hence, we adjusted our range of message rates upward. The message rates used emphasize "interesting" regions of the data—where the curves are not flat—and were selected based on preliminary results.

Preliminary results also show that there is a maximum message rate the two memory-network interfaces are capable of sustaining. At message rates upwards of approximately 0.1000 words/clock,

[4]To do otherwise would confuse effects caused by a node architecture and artifacts of an application's implementation.

[5]We ran our memory traces through dineroIII, using various-sized fully-associative caches with LRU replacement. We then plotted the miss rate against the cache size and took the knee in each curve as the corresponding application's working set size.

Application	Benchmark suite	Language	Memory references	Working set size[5]	Description
`barnes` ("Barnes-Hut")	SPLASH	C	128,299,383	32KB	Simulate evolution of galaxies using Barnes-Hut hierarchical N-body method [15]
`doduc`	CFP92	Fortran	113,957,528	64KB	Monte Carlo simulation of nuclear reactor component [4]; we used the "small" input set
`ear`	CFP92	C	256,352,000	8KB	Simulate propagation of sound in human cochlea [4]; we used the "short" input set
`fpppp`	CFP92	Fortran	294,101,082	16KB	Two electron integral derivative from GaussianXX quantum chemistry programs [4]; we traced a 5-atom run
`mdljsp2`	CFP92	Fortran	64,567,619	128KB	Molecular dynamics program modelling the interaction of 500 atoms in a Lennard-Jones potential [4]; we used the "short" input set
`pthor`	SPLASH	C	58,133,474	64KB	Event-driven logic circuit simulator [15]—uses Chandy-Misra's distributed-time algorithm
`xlisp` ("Li")	CINT92	C	63,666,825	64KB	LISP interpreter running a program to solve the N-queens problem [5]; we specified 7 queens

Table 1: Application traces

the CM-5 and—to a lesser extent—Paragon spend so much time handling messages, they can make no other progress. This is useful to know because it provides a lower bound on current node architectures communication granularity. Applications that attempt to transmit more frequently than 1 word for every 10 clocks spent in the memory hierarchy (a 1:1 communication–memory ratio in our model) are likely to spend an inordinate amount of time waiting on message-handling routines and the cache refills they cause.

3.3.3 Cache Size

We experimented with a variety of cache sizes in order to forecast performance for future microprocessors. We simulated an 8KB cache because the DECchip 21064-AA—on which most of the components in our simulated microprocessor are based—uses an 8KB cache. But because modern microprocessors have ever larger primary data caches, we also simulated 16KB, 32KB, and 64KB caches. We did not simulate a second-level cache.

Variable	Description	Values
Node architecture	Manner in which message traffic is handled	"CM-5" "Paragon" "uniprocessor"
Message rate	Number of words sent and received per processor clock	0.0010 0.0020 0.0100 $\boxed{0.0200}$ 0.0400 0.0600 0.0800
Cache size	Size of cache in bytes	8KB $\boxed{\text{16KB}}$ 32KB 64KB
Long message percentage	Percentage of messages (in number) considered "long" ("long"≡128 words, "short"≡4 words)	0% 1% 2% 3% 4% 10% 20% $\boxed{25\%}$ 50% 75% 100%

Table 2: Simulation parameters; defaults are boxed

3.3.4 Long Message Percentage

We experimented with both long and short messages to determine the effect of message length on the cache miss rate. Because they are allocated in contiguous memory locations, long messages give near-perfect spatial locality. In contrast, sequences of short messages generally do not. Because of its spatial locality, a single long message should displace fewer cache lines than an equivalent volume of short messages, and is therefore expected to be less harmful to cache performance.

To quantify the impact of short and long messages, we simulated different mixes of two message lengths (short and long). As with message rate, the specific long message percentages we chose emphasize "interesting" regions of data and were selected based on preliminary results.

3.4 Dependent Variables

Cache misses is the dependent variable used in all of the graphs in Section 4. Most of those graphs normalize the number of cache misses to that observed on the uniprocessor simulation. Where appropriate, absolute misses are also used. Normalized numbers represent how many times slower an application can be expected to run in the presence of message traffic.

For the purposes of this study, cache misses is a superior metric both to execution time and cache miss rate. Execution time is technology-dependent, tying findings to specific choices of component speeds. Cache miss rate, while commonly used in cache studies, is inappropriate for this study because the number of memory references is variable; message handling introduces additional memory references. Cache misses, unlike miss rate, does not depend on the total number of memory references and therefore provides better intuition about the data.

In Section 4, cache read misses that access primary memory are considered "misses," and all other cacheable memory operations are considered "hits." Most notably, cache write misses are considered hits because the write buffer and write no-allocate cache hide memory latency, thereby making write miss response time approximately equal to that of a write hit. Uncacheable operations (reads and writes to memory-mapped I/O locations) are deemed I/O and are therefore considered neither hits not misses. Note that references made to primary memory as a result of network traffic (viz. storing incoming messages to memory) are considered regular memory operations.

3.5 Component Speeds

For our experiments, we assigned the following delays—which we believe are reasonable for a 200MHz DEC Alpha-based system (5 ns cycle time)—to the components described in Section 3.1:

cache access	1 CPU cycle
write buffer access	1 CPU cycle
bus cycle	2 CPU cycles
memory access	10 CPU cycles
network cycle	$\frac{1}{2}$ CPU cycle
router in→out latency ...	2 CPU cycles

All other components have a latency of 1 cycle on their interfacing component's clock. Note that write buffer accesses are performed in parallel with cache accesses.

4 Simulation Results and Analysis

4.1 Base Misses

We measure the impact of message traffic on the cache by comparing multicomputer and uniprocessor cache misses.[6] Figure 7 plots the uniprocessor misses for each application. The graphs in Sections 4.2–4.4 are normalized to the misses shown in Figure 7 and therefore represent the increase in the number of multicomputer cache misses compared to the corresponding uniprocessor simulation. Note that uniprocessor simulations will always observe a lesser number of misses than multicomputer simulations because they perform an application's memory references uninterrupted by miss-causing message traffic.

4.2 Misses versus Message Rate

The first experiment shows how sensitive the cache is to the volume of message traffic per unit time. The average message rate was varied from 0.0010–0.0800 words sent/received per processor clock. Figure 8 shows the CM-5/Paragon's increase in misses for each application, normalized to

[6] Uniprocessor, in this context, means a node executing in the absence of message traffic. The simulated uniprocessor system contains all the modules described in Section 3.1 except for the DMA module, network interface, and network fringe.

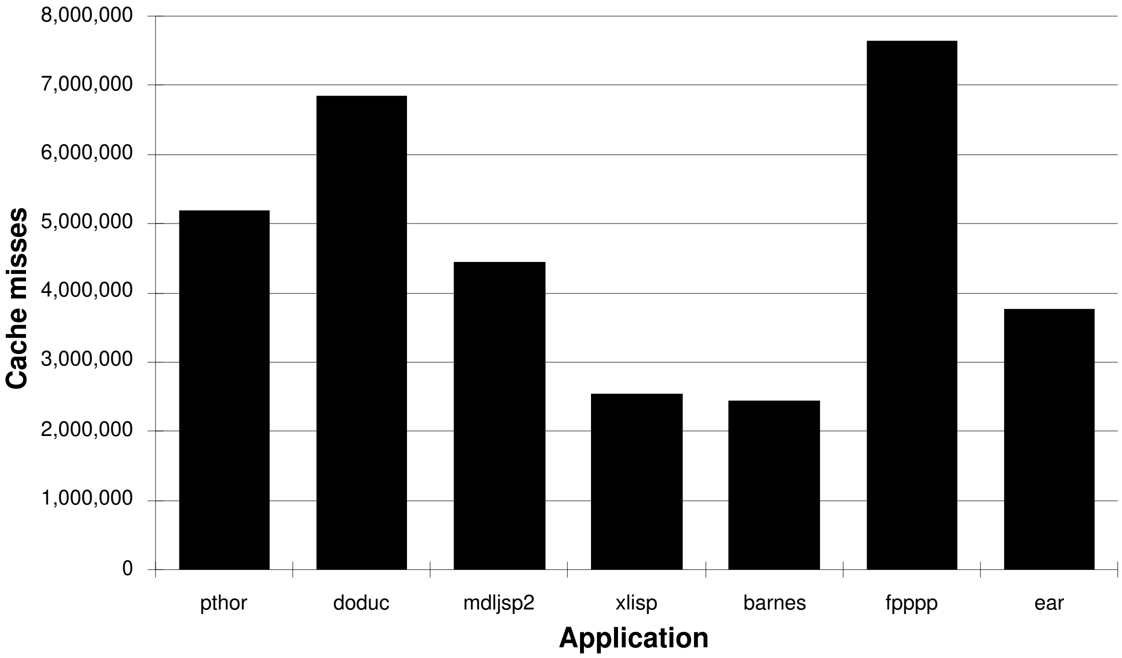

Figure 7: Uniprocessor cache misses

the number of misses observed on the uniprocessor system and sorted by that normalized number. Note that one of the abscissas is on a logarithmic scale.

For all seven applications, the effects of an increased message rate are far more pronounced on the CM-5-style system than on the Paragon-style system. That result is in accordance with our intuition, as the CM-5-style system utilizes the cache for message handling while the Paragon-style system offloads message handling to the DMA controller. The difference between the two multicomputer systems is significant at high message rates, with the CM-5-style system observing many times more misses than the Paragon-style system.

Sensitivity to cache pollution is related to memory reference locality. Compare the height of each line in Figure 8 to the height of the corresponding bar in Figure 9, a plot of each application's uniprocessor cache miss rate. While the heights are not strictly inversely proportional, they are close enough to imply that locality is a factor in cache sensitivity to message traffic. That makes sense because cache lines displaced in a high-locality application are apt to be accessed again, thereby introducing more cache misses than would occur from an application with less locality.

Data layout is another factor in cache sensitivity. The more an application's data is scattered throughout memory, the less likely it is that a single message will displace more than a few words used by the application.

4.3 Misses versus Cache Size

Future microprocessors will almost certainly contain larger caches than current ones. Therefore, predicting the performance of forthcoming message-passing systems depends on the sensitivity of these changes to cache size. To determine that relation, we varied the cache size from 8KB to 64KB and held all other simulation parameters at the default values (see Table 2). Figure 10 plots the normalized number of cache misses. Figures 11–13 plot the same data unnormalized to put Figure 10 in perspective. Note that for legibility, the normalized graph plots increasing cache sizes

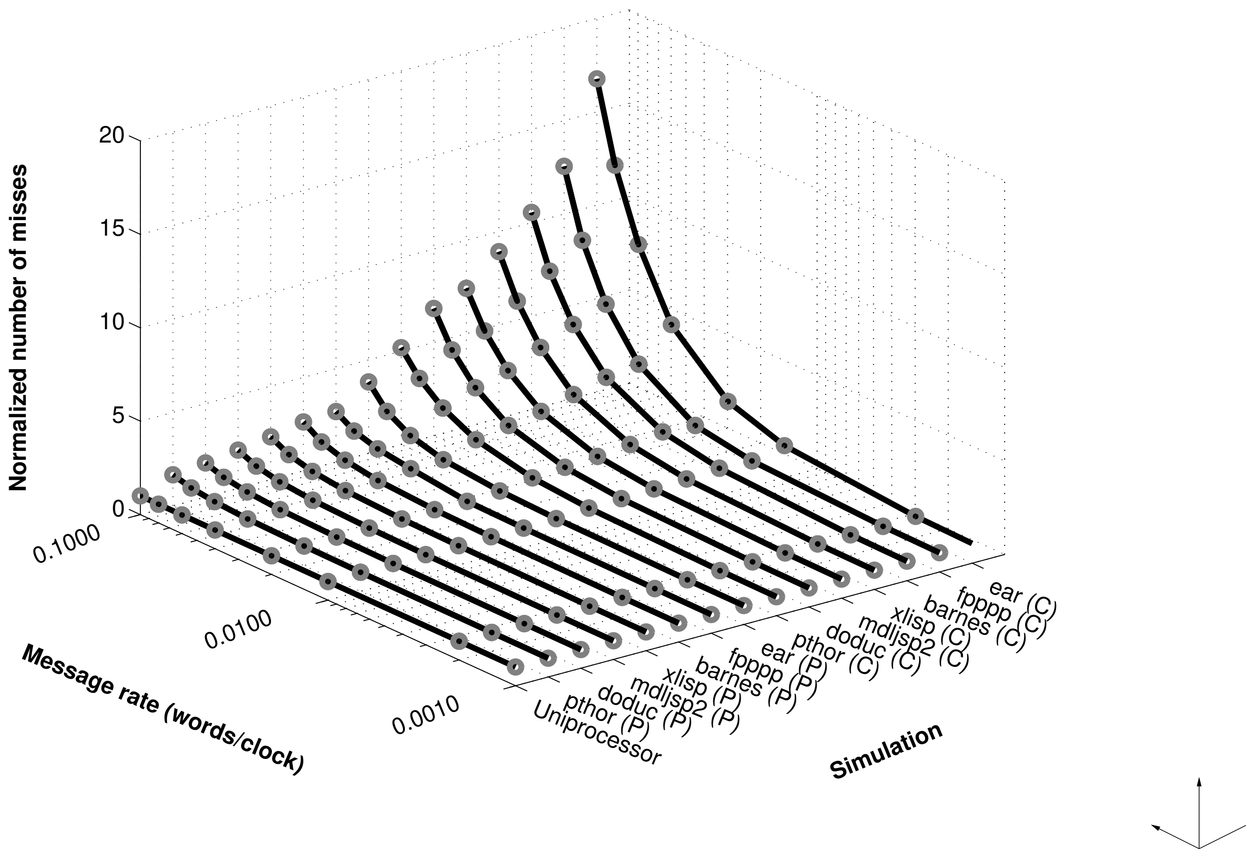

Figure 8: Normalized number of misses vs. message rate;
(C) indicates the CM-5-style system,
(P) the Paragon-style system

front-to-back, while the unnormalized graph plots them back-to-front.

For all seven applications, each multicomputer's normalized number of misses *increases* with cache size (with the exception of `ear` from a 32KB–64KB cache—most likely a threshold effect). Of course, increasing the cache size decreases the absolute number of misses for all applications and node architectures. The intuition, then, is that the increased nunber of cache misses is magnified by the lower number of misses.

As Figure 10 shows, the Paragon-style system observes essentially the same number of misses as the uniprocessor system. In the worst case (i.e. the greatest increase in misses)—`ear` with a 32KB cache—36% more misses were observed on the Paragon-style system than on the uniprocessor. In contrast, the CM-5 system is extremely sensitive to message traffic for large cache sizes. In that system's worst case—also `ear` with a 32KB cache—290% more misses are observed than on the uniprocessor! Even the CM-5-style system's *best* case (`pthor` with an 8KB cache) shows a 40% increase in number of misses—even worse than the Paragon-style system's *worst* case mentioned above.

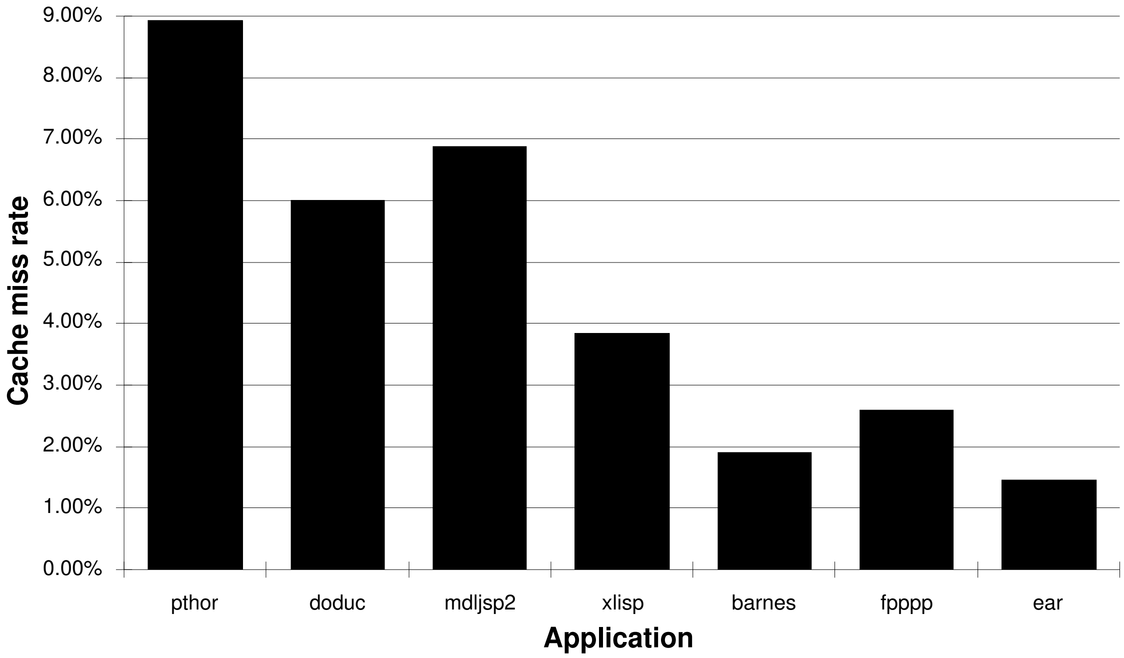

Figure 9: Uniprocessor cache miss rate

The reason the CM-5-style system consistently exhibits a greater increase in misses than the Paragon-style system is that the processor reads into the cache all words to be transmitted. While the read itself might miss in the cache, it may also displace cache lines used by the application for an additional performance penalty. Because the Paragon-style system uses DMA to transport messages directly from memory to the network interface, no additional cache misses are generated for message sends.

4.4 Misses versus Message Size

Message-passing applications employ a large variety of message lengths [7]. Therefore, we investigated the effect of message size on cache performance. We expect message size to impact the cache for two reasons: First, a small number of long messages exhibits more spatial locality than a large number of short messages. Second, long and short messages tend to be used differently in message-passing applications: short messages for program control (e.g. for synchronization and broadcasting state information) and long messages for transporting program data structures. The difference in usage is that control message are generally read upon receipt, while data messages are read piecewise, with intermittent computation.

Our simulators represent that difference in usage—both in this section and all the others—by immediately processing control messages. To that effect, the Paragon-style system's CPU reads control messages upon arrival, but leaves data messages in memory. That is, it ignores steps 3 and 4 in Figure 3. On the CM-5-style system, however, both control and data messages follow exactly the path detailed in Section 2.2. Because the CM-5-style system's CPU immediately processes all messages by default, control and data messages need not be differentiated.

To better isolate the effects of different message sizes, we used two sizes of messages in our experiments—"long" (data) and "short" (control)—and varied their relative volume while maintaining a constant total volume (Figures 14–15).

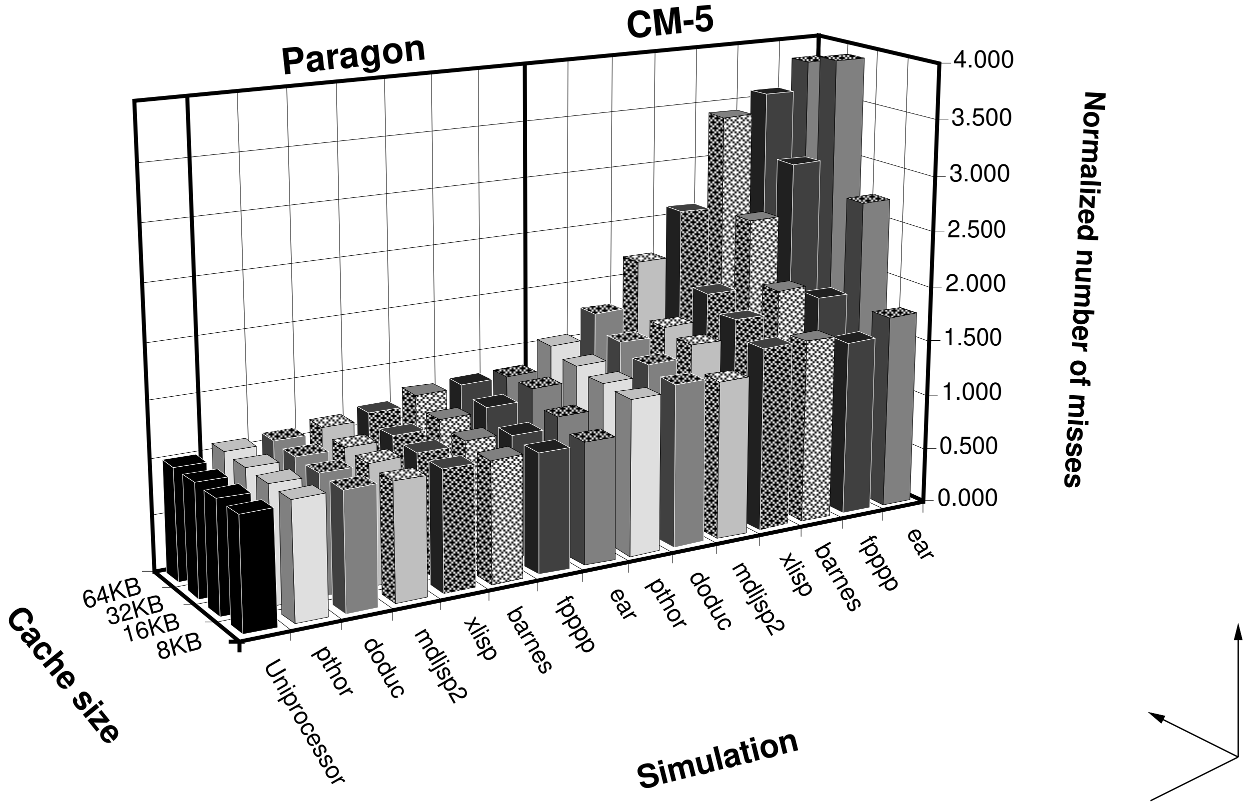

Figure 10: Normalized misses vs. cache size

The insight granted by Figures 14–15 is that short messages increase cache misses more than long messages on both the CM-5- and Paragon-style systems. As the long message fraction is varied from 0%–100%, the number of misses decreases an average of 38% on the CM-5-style system and an average of 64% on the Paragon-style system. The much higher number of cache misses for short messages is due the lack of temporal and spatial locality inherent in control messages. Control messages are read into the cache, where they displace program data and are then used only briefly.

Observe that the Paragon and CM-5 curves representing some of the applications (`pthor`, `mdljsp2`, and `ear`), would intersect if placed side by side. For the rest of the applications, the CM-5 curve is higher than the Paragon curve at all points. From this, we gain a second insight: In terms of cache performance, the CM-5-style system is *sometimes* superior to the Paragon-style system at handling short messages, while the Paragon-style system is *always* superior to the CM-5-style system at handling long messages. We do not know what aspect of an application's structure would make an application more resilient to short messages on the CM-5-style system than on the Paragon-style system. We can rule out locality, however, because Figures 14–15 show a crossover point for both `pthor` and `ear`, yet those applications have vastly different cache miss rates. A more meaningful classification of application structure is complex and beyond the scope of this study.

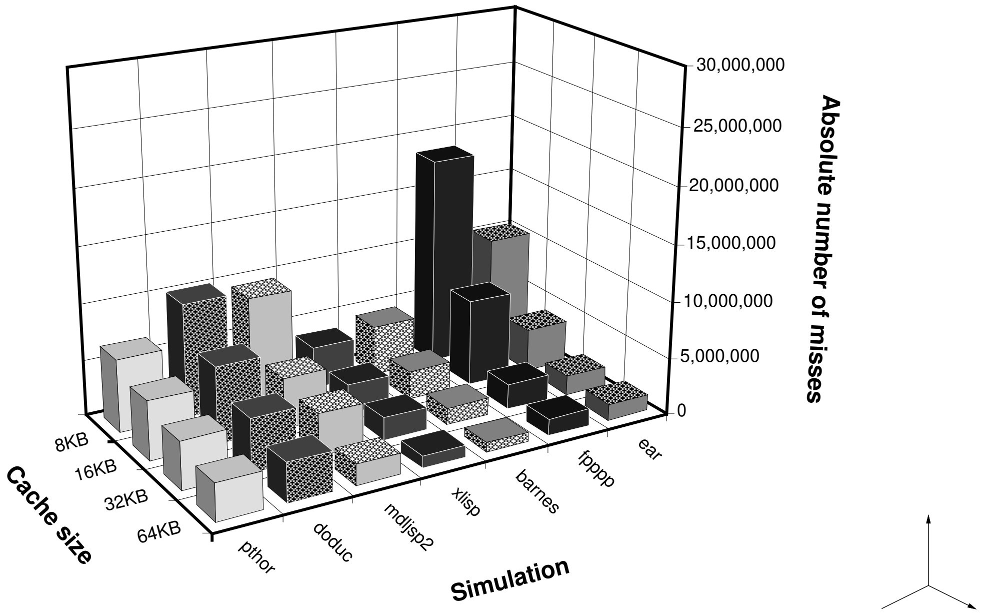

Figure 11: Absolute misses vs. cache size (uniprocessor)

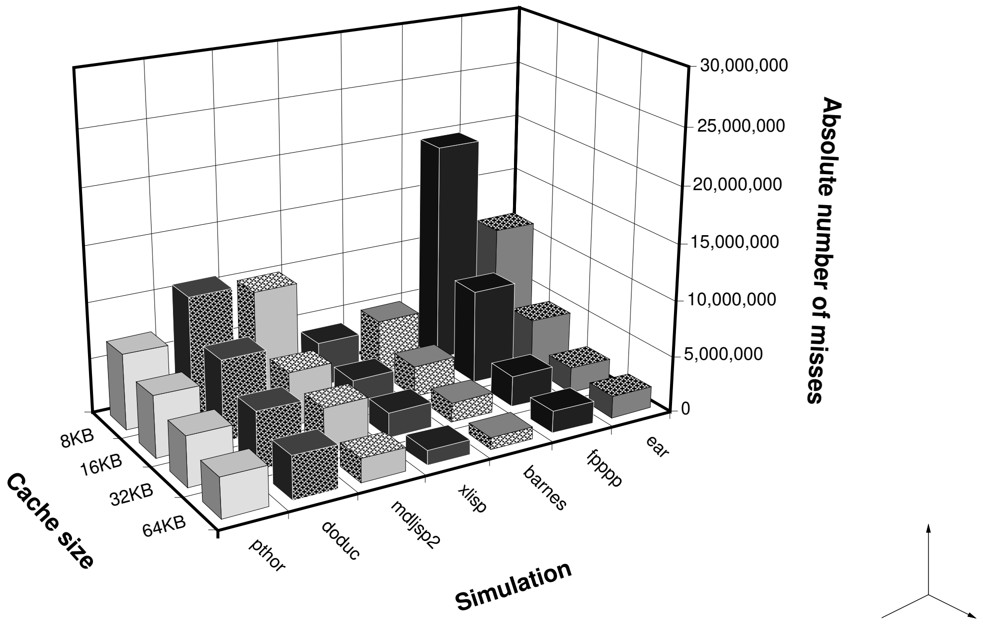

Figure 12: Absolute misses vs. cache size (Paragon)

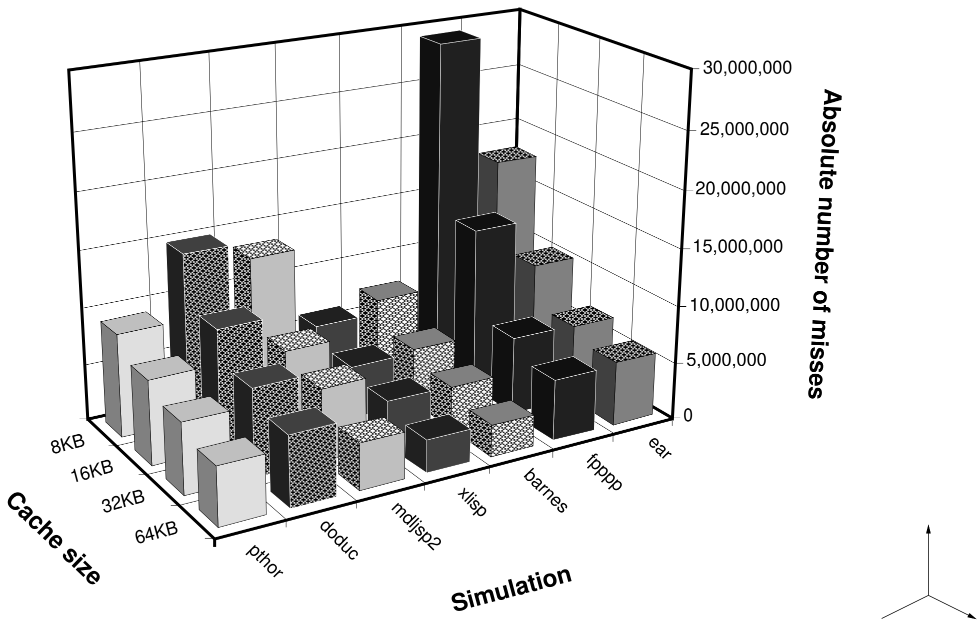

Figure 13: Absolute misses vs. cache size (CM-5)

5 Discussion

Our first set of experiments (Section 4.2) demonstrate a positive correlation between the message rate and the cache miss rate. Not surprisingly, the CM-5 system's cache (which is directly involved in message sends) is far more sensitive to message traffic than the Paragon system's. Figures 14–15 quantify that sensitivity for `pthor`, `doduc`, `mdljsp2`, `xlisp`, `barnes`, `fpppp`, and `ear`. The data imply that message traffic significantly increases cache misses and hence greatly influences multicomputer node performance. It therefore cannot be assumed that the cache miss rate exhibited by a parallel implementation of an application will approximately equal that exhibited by a sequential implementation of the same application, even with equal amounts of application-related work per node.

In Section 4.3, we quantified the extent to which message traffic's impact increases with cache size. Figure 10 shows that not only is the CM-5-style system more sensitive to message traffic than the Paragon-style system, but its sensitivity increases with increased cache size. The implication is that, all other factors being equal, the use of larger caches in multicomputer microprocessors will decrease parallel speedup. This does not mean that large caches are detrimental to performance. On the contrary, large caches typically improve performance, but they cannot be expected to improve performance as much for microprocessors operating in a multicomputer environment as in a standalone environment.

In our final set of experiments, detailed in Section 4.4, we varied the volume of long messages using a constant total message volume. Figures 14–15 show that for all seven applications, the Paragon-style system's cache is virtually immune to long messages, while both the CM-5- and Paragon-style systems' caches are heavily polluted by short messages. Finally, our data highlight the range of long message percentages within which the two node architectures might observe the

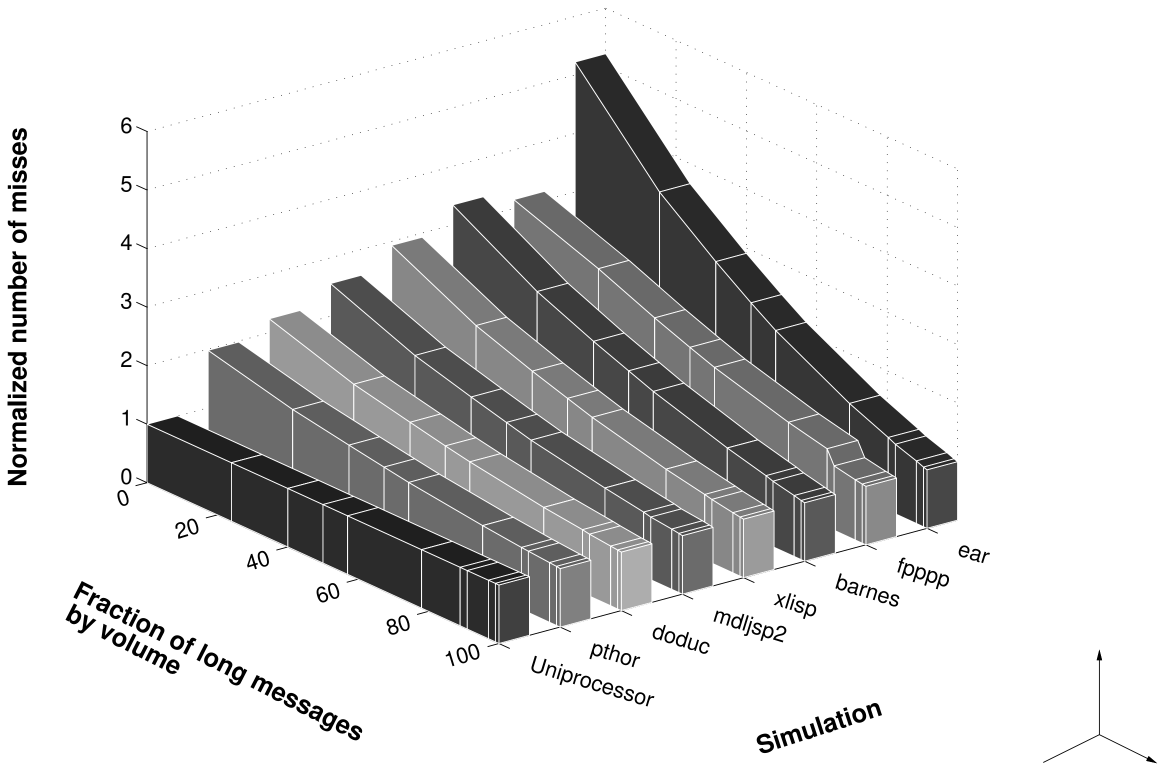

Figure 14: Normalized misses vs. fraction of long messages by volume (Paragon)

same cache miss rate and quantify the maximum expected differences in miss rate (i.e. at the 0% and 100% long message percentages).

Low-latency communication is the intended benefit of logically connecting the network interface and the cache. However, our studies show that this trades off latency for an increase in cache misses. Even though the Paragon is notorious for its high-latency communication, in some cases it may actually benefit by keeping the cache unpolluted by message data.

6 Summary

High cache miss rates detract significantly from microprocessor performance. Consequently, it is important to know which factors in a computer system contribute to cache misses. In this paper, we focused exclusively on one potential factor: interprocessor communication. We quantified how much message traffic affects cache performance by running a number of experiments on register-transfer-level simulators of two representative multicomputer nodes. One node's architecture, like the CM-5's, involves the CPU and cache in interprocessor communication. The other node's architecture, like the Paragon's, uses DMA to move message data between the node and the network.

We presented data showing the impact on two nodes' caches from a variety of message rates, cache sizes, and message lengths. What we discovered is that processing messages significantly impacts cache on both systems. As a result, sequential performance degrades noticeably in the

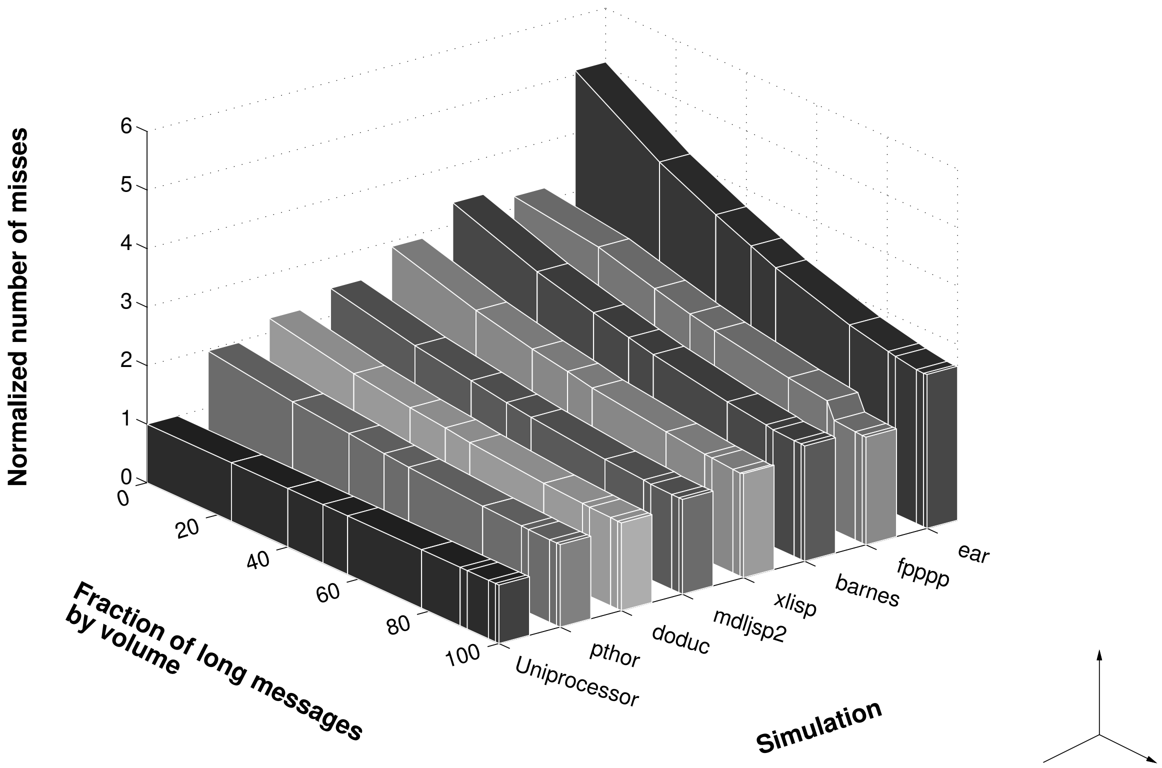

Figure 15: Normalized misses vs. fraction of long messages by volume (CM-5)

presence of message traffic. While the Paragon-style system observes up to four times as many cache misses as the uniprocessor under heavy message loads, the CM-5-style system observes up to 17 times the uniprocessor amount. The large increase is primarily because the cache is involved in message handling and, hence, is frequently polluted.

Control messages (short messages with little locality) cause greater cache pollution—and hence, a greater degradation in system performance—than data messages (long messages with much locality). However, our simulated CM-5-style system outperforms the corresponding Paragon-style system when handling short control messages on certain applications (`pthor`, `mdljsp2`, and `ear`), although the simulated Paragon-style system is always superior at handling long data messages.

The impact of message traffic on the cache is likely to increase in the future. Our results show that larger caches widen the gap between a node's sequential and parallel performance. Although increasing the cache size improves performance for both the uniprocessor and multicomputer nodes we simulated, the performance gain is smaller when a node must additionally handle interprocessor communication. For example, `fpppp` running on a Paragon-style system with an 8KB cache and handling an average of one word of message data every 50 clocks, 25% long messages by volume, observes fewer than 7% more misses than a uniprocessor with an 8KB cache. But when both systems are equipped with a 64KB cache, the Paragon-style system observes over 31% more misses than the uniprocessor.

The implications of our study are that one cannot assume that a microprocessor will perform equally well on message-passing applications as on sequential applications. Furthermore, there is a downside to cache-based message handling. While it provides for low-latency communication, sending and receiving messages through the cache results in significant cache pollution. The increased number of cache misses is definitely an important, yet underestimated, factor contributing to poor speedups of parallel programs over their sequential counterparts.

7 Future Work

The CM-5 and the Paragon do not represent the only ways to interface a processor's memory hierarchy and its network communication system. A worthy subject of future research might examine the effects of message passing on other architectures, especially those that connect the network interface and other levels of the memory hierarchy or multiple levels of the memory hierarchy.

We have already begun one such effort by starting to examine the relationship of network traffic to cache misses in the proposed DI-multicomputer [3], in which messages can be routed either directly from registers or memory, and the programmer or compiler chooses which is more appropriate on a per-message basis. Because of the additional flexibility in the DI-multicomputer scheme over the CM-5 and Paragon-style architectures, one would expect that machine to perform quite well relative to the architectures discussed in this paper.

Another useful study would be to take a radically different approach to studying message traffic's effect on the cache. Instead of simulating a single multicomputer node running pseudo-parallel programs (sequential programs with message passing grafted on), it would be interesting to simulate multiple nodes running true message-passing programs. While such a scheme would lack the precise control over message length and rate our simulators supported, it would benefit from added "realism."

References

[1] Carl J. Beckmann. CARL: An architecture simulation language. Report 1066, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 305 Talbot, 104 South Wright St.; Urbana, IL 61801-2932, December 1990.

[2] John Bruner. Parsim user interface reference manual. Report 1002, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 305 Talbot, 104 South Wright St.; Urbana, IL 61801-2932, September 1990.

[3] Lynn Choi and Andrew A. Chien. Integrating networks and memory hierarchies in a multicomputer node architecture. In *Proceedings of the International Parallel Processing Symposium*, April 1994.

[4] SPEC Steering Committee. Introduction to SPEC CPU floating point benchmark suite: CFP92. `INTRO` file in CFP92 distribution, January 1992.

[5] SPEC Steering Committee. Introduction to SPEC CPU integer benchmark suite: CINT92. `INTRO` file in CINT92 distribution, January 1992.

[6] Cray Research, Inc., Eagan, Minnesota 55121. *CRAY T3D Software Overview Technical Note*, 1992.

[7] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural requirements of parallel scientific applications with explicit communication. In *Proceedings of the International Symposium on Computer Architecture*, pages 2–13. IEEE Computer Society, 1993.

[8] Digital Equipment Corporation, Maynard, MA. *DECchip 21064-AA Microprocessor Hardware Reference Manual*, 1st edition, October 1992. Order number EC-N0079-72.

[9] M. Homewood and M. McLaren. Meiko CS-2 interconnect Elan – Elite design. In *Proceedings of the IEEE Hot Interconnects Symposium*. IEEE TCMM, August 1993.

[10] Intel Corporation. *Paragon XP/S Product Overview*, 1991.

[11] John Kubiatowicz and Anant Agarwal. The anatomy of a message in the Alewife multiprocessor. In *Proceedings of the International Conference on Supercomputing*, July 1993. Available from `cag.lcs.mit.edu/pub/papers/anatomy.ps.Z`.

[12] David May, Roger Shepherd, and Peter Thompson. The T9000 Transputer. Technical report, Inmos Limited, 1000 Aztec West, Almondsbury, Bristol BS12 4SQ, UK, 1993. Available from `ftp.inmos.co.uk:/inmos/info/T9000/T9000.ps.Z`.

[13] C. Seitz, N. Boden, J. Seizovic, and W. Su. The design of the Caltech Mosaic C multicomputer. In *Proceedings of the University of Washington Symposium on Integrated Systems*, 1993.

[14] Toshiyuki Shimizu, Takeshi Horie, and Hiroaki Ishihata. Low-latency message communication support for the AP1000. In *International Symposium on Computer Architecture*, pages 288–297, 1992.

[15] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. SPLASH: Stanford parallel applications for shared-memory. Technical report CSL-TR-91-469, Computer Systems Laboratory, Stanford University, Stanford University, CA 94305, 1991.

[16] Thinking Machines Corporation, Cambridge, Massachusetts. *CM-5 Technical Summary*, October 1991.

[17] Günter Watzlawik and Franz Hutner. A pipelined network interface for a parallel computer. Technical report, Bull/ECRC/ICL/Siemens, ZFE ST SN 22, Siemens AG, Otto-Hahn-Ring 6, 81730 München, Germany, 1993. e-mail: `watz@zfe.siemens.de`.