RC 21835 (98276) 18 September 2000 **Computer Science**

IBM Research Report

MemorIES: A Programmable, Real-Time Hardware Emulation Tool for Multiprocessor Server Design

Ashwini Nanda, Kwok-Ken Mak, Krishnan Sugavanam, Ramendra K. Sahoo,
Vijaraghavan Soundararajan, T. Basil Smith
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

MemorIES: A Programmable, Real-Time Hardware Emulation Tool for Multiprocessor Server Design

Ashwini Nanda, Kwok-Ken Mak[1], Krishnan Sugavanam, Ramendra K. Sahoo, Vijayaraghavan Soundararajan, and T. Basil Smith

IBM T.J. Watson Research Center

Yorktown Heights, NY

{ashwini,sugavana,rsahoo,sound,tbsmith}@us.ibm.com

Abstract

Modern system design often requires multiple levels of simulation for design validation and performance debugging. However, while machines have gotten faster, and simulators have become more detailed, simulation speeds have not tracked machine speeds. As a result, it is difficult to simulate realistic problem sizes and hardware configurations for a target machine. Instead, researchers have focussed on developing scaling methodologies and running smaller problem sizes and configurations that attempt to represent the behavior of the real problem. Given the increasing size of problems today, it is unclear whether such an approach yields accurate results. Moreover, although commercial workloads are prevalent and important in today's marketplace, many simulation tools are unable to adequately profile such applications, let alone for realistic sizes.

In this paper we present a hardware-based emulation tool that can be used to aid memory system designers. Our focus is on the memory system because the ever-widening gap between processor and memory speeds means that optimizing the memory subsystem is critical for performance. We present the design of the ***Memor***y ***I***nstrumentation and ***E***mulation ***S***ystem (MemorIES). MemorIES is a programmable tool designed using FPGAs and SDRAMs. It plugs into an SMP bus to perform on-line emulation of several cache configurations, structures and protocols while the system is running real-life workloads in real-time, without any slowdown in application execution speed. We demonstrate its usefulness in several case studies, and find several important results. First, using traces to perform system evaluation can lead to incorrect results (off by 100% or more in some cases) if the trace size is not sufficiently large. Second, MemorIES is able to detect performance problems by profiling miss behavior over the entire course of a run, rather than relying on a small interval of time. Finally, we observe that previous studies of SPLASH2 applications using scaled application sizes can result in optimistic miss rates relative to real sizes on real machines, providing potentially misleading data when used for design evaluation.

1 Introduction

Modern computer system design often proceeds through a combination of simulation, mathematical modeling, and experience. Models are helpful but can be complex to derive and verify, and may not provide sufficient detail. Basing machine design on experience with previous systems can be misleading if the target applications have changed. As a result, simulation is often the tool of choice for design evaluation. While machines have proceeded to get faster, the simulation tools, unfortunately, have not really evolved at the same speed. Although the level of detail for simulators has increased[PRA+97][RHW+95], the systems being modeled are much more complex, so the simulators are not necessarily able to run larger problem sizes or realistic machine parameters because of prohibitive simulation time or storage requirements. For example, Table 1 shows the L2/L3 cache sizes that have been used in real systems over the last few years in comparison to the cache sizes simulated during the same period. As the table shows, although the machines use larger caches, the simulators are unable to keep up. The problem sizes are therefore often scaled to allow the simulations to complete in a reasonable time while still preserving the essential characteristics (e.g. miss ratio, communication patterns) of the applications. Because these applications are smaller than they would be in practice, the cache sizes must also be scaled so that the relevant working sets have the appropriate cache behavior. As a result of this scaling, design decisions are often made based on partial information.

In the case of high-end commercial servers, choice of system parameters can be extremely critical, both from a cost as well as a performance perspective. As the gap between processor speed and memory speed continues to increase, caches can play a particularly pivotal role in the performance of high end SMP servers and NUMA systems. The size of these caches increases from one sys-

[1]Kwok-Ken Mak is currently with Cisco Systems.

TABLE 1. Simulated Cache Sizes vs. Actual Cache Sizes in Previous Studies. (n/a means either not applicable or the data was unavailable. Sources: [WOT+95][FW97][MNL+97][BDH+99][FW99])

Year	Application	Problem Size	# simulated processors	Simulated L2 cache size	L2 size for machines of that year	L3 size for machines of that year
1995	FFT	64K points	16-64	8KB-1MB	512KB	n/a
	Barnes Hut	16K bodies	16-64	8KB-1MB	512KB	n/a
	Water	512 molecules	16-64	8KB-1MB	512KB	n/a
1997	FFT	64K points	32-64	8KB-1MB	4MB	32MB
	Barnes Hut	16K bodies	32-64	8KB-1MB	4MB	32MB
	Water	512 molecules	32-64	8KB-1MB	4MB	32MB
1999	FFT	64K points	32-64	128KB-512KB	8MB	32MB
	Barnes Hut	16K bodies	32-64	n/a	8MB	32MB
	Water	512 molecules	32-64	128KB-512KB	8MB	32MB

tem generation to another to keep up with the ever-increasing working set requirements of important applications, such as transaction processing. For example, TPC-C database sizes have grown from ~10GB in 1995 to ~100GB or more in 1999[TPC]. Similarly, Decision Support applications (TPC-D/TPC-H) have also grown from ~10GB in 1994 to ~300GB or more in 1999[TPC]. Partly in response to these changes, the cache sizes for such servers have increased. Figure 1 shows the approximate ranges of level 2 or level 3 cache sizes in current systems as well as projected ranges in the future assuming the current rate of increase in workload demands remains the same. As an example, the IBM RS/6000 S7A 12-way SMP system [IBM] available in the market today has 8MB off-chip level 2 caches per processor.

Designing an optimal structure for these caches is a challenging task due to the lack of a proper evaluation infrastructure. Clearly, *actual measurement* for a given cache configuration is not possible without building it. Trace-driven simulation or execution-driven simulation serves the purpose for relatively small caches of up to a few megabytes. However, accurate trace-driven simulation for caches as large as several tens or hundreds of megabytes is not feasible today due to the difficulty in obtaining, storing and running very large traces. Execution driven simulation such as COMPASS [NHO+98] or Augmint [NMS+96] would take several days to several months to run an application large enough (as for example a 100GB-1TB database) to appropriately exercise the large caches. The only alternative left today to the designer for evaluating large caches is to make analytical projections of cache statistics from earlier measurements of smaller cache configurations. Although projections could be reasonably accurate for one or two larger cache sizes than the measured cache size, the accuracy of such predictions would drastically decrease as we get into much larger sizes.

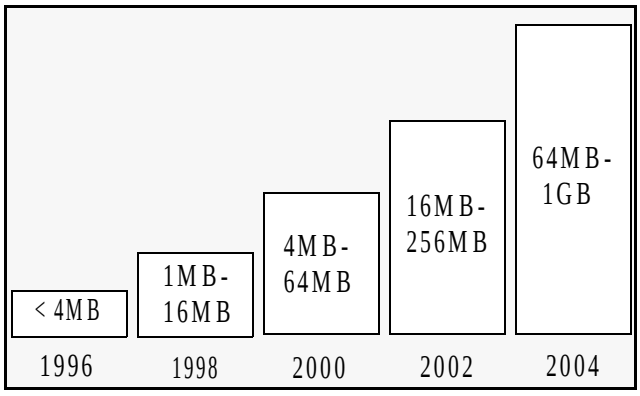

FIGURE 1. Level 2 and Level 3 Cache Sizes in High End Servers

In order to determine trade-offs for future cache designs in multiprocessor servers, we have designed a hardware emulation tool called the Memory Instrumentation and Emulation System (*MemorIES*). The MemorIES tool is capable of *emulating* caches ranging from 2MB-8GB with varying associativity and line size attributes, different replacement algorithms and cache protocols in real-time with no slow down in execution speed. The MemorIES board consists of several Field Programmable Gate Arrays (FPGAs) and up to 1GB of DRAM memory. The board sits on a conventional SMP bus (such as in a RS/6000 SMP server) and passively monitors all the bus transactions. It emulates several SMP nodes (smaller than the host SMP) each having a shared cache by keeping the state information on lines that would be shared in the caches of the emulated SMPs. The tool can be programmed either by its console software and/or by reconfiguring the on-board FPGAs to change the cache parameters, cache protocol, and other attributes.

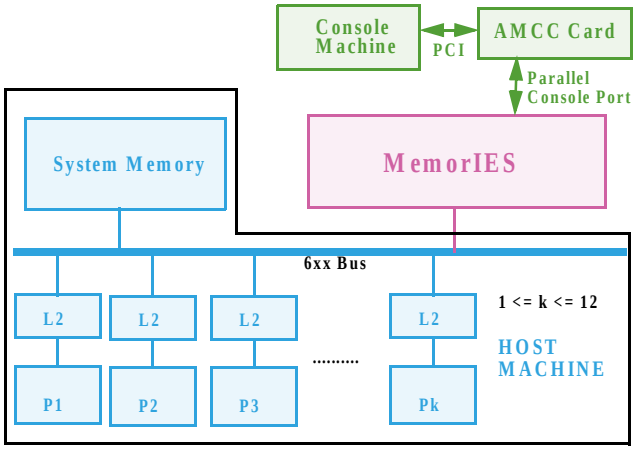

FIGURE 2. Operating Environment of MemorIES

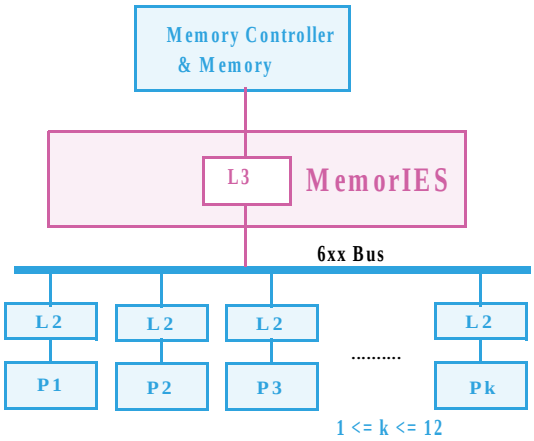

FIGURE 3. Logical Target Machine for Single-Node L3 Emulation

The *emulated shared caches* can behave as level 2 or level 3 caches depending on whether the actual level 2 cache is switched off or on. MemorIES emulates an L3 cache when the host machine uses both L1 and L2 caches, and emulates an L2 cache when the host machine turns off its L2 cache. The tool captures extensive memory and cache access behavior as well as bus transaction statistics on-line and non-intrusively while the machine is running real benchmarks in real-time. This mechanism allows the MemorIES boards to collect memory behavior of important benchmarks over a period of several hours compared to less than a minute of data that can be gathered using conventional tracing mechanisms. Although the MemorIES tool relies upon *on-line* trace analysis and real-time cache emulation, it also provides a mechanism to collect traces for finer and repeatable *off-line* analysis when necessary. While most of today's processors and cache controllers provide extensive instrumentation to capture program statistics at run time, they can collect statistics only on the one cache design they represent. On the other hand, the MemorIES tool is equivalent to implementing multiple programmable cache controllers in hardware for the purpose of performance measurements.

MemorIES provides a way to quickly determine bottlenecks in applications, providing enough information for a detailed simulation model to then be used to more precisely pinpoint the performance problems. By focussing on one critical piece of the system (the memory subsystem/caches), we are able to use the MemorIES board as an efficient complement to our already-existing simulators.

The rest of the paper is organized as follows. Section 2 describes the system environment in which MemorIES is used. Section 3 discusses the design of MemorIES. Section 4 compares MemorIES with trace- and execution-driven simulation and section 5 presents working examples of the MemorIES tool. Section 6 discusses prior work. Section 7 concludes the paper.

2 System Environment

The MemorIES tool is designed to help understand future SMP server performance issues, especially L2/L3 cache behavior, by providing emulated cache directories and protocol tables, and to study non-uniform memory access (NUMA) memory reference behavior using current generation SMPs. The operating environment of MemorIES is shown in Figure 2. The current generation MemorIES board plugs into the memory bus (called the 6xx bus) of IBM's S70 class RS/6000 or AS/400 servers that have a maximum capacity of 12 processors.

The CPU IDs on the memory bus of the *host machine* are partitioned to emulate a variety of *target machines*. A target machine has one or more emulated SMP nodes connected to a memory controller (or controllers) through a shared cache in each emulated node. Each emulated node has one or more processors connected to a 6xx bus through L2 caches.

The board captures all the memory references on the 6xx bus of the host system and maintains cache tag and state information for the emulated shared cache(s). The parameters that can be evaluated include cache capacity, line size, associativity, number of caches, cache protocols, and replacement algorithms. Various cache statistics such as hit ratio, read/write ratio, effect of I/O on hit ratio, and amount of cache-to-cache interventions, to name a few, are collected using counters in the emulation board.

The console machine is an IBM PC running Windows 95/98, which provides a programming interface to the MemorIES board using an AMCC parallel port control card. The console software is used for power-up initialization of the MemorIES board, cache parameter setting, and statistics extraction. A few example target systems that can be emulated using MemorIES are described next. All of these configurations are dynamically programmable using the console software.

2.1 Single and Multiple Node Emulation

Figure 3 shows how MemorIES emulates the L3 cache behavior of a single node target SMP. The target SMP has one or more processors connected to a 6xx bus through L2 caches. The

MemorIES board passively collects various statistics of the target system which can then be used to study cache behavior.

MemorIES can also emulate the L3 cache behavior of a multiple-node, hierarchically connected SMP or NUMA machine. Each of the multiple nodes in the target system is connected to the memory controller through an L3 cache. Each node has one or more processors connected to a 6xx bus through L2 caches. This configuration is similar to the single node configuration, except that there are multiple L3 emulators in use, each responding to requests from a different node (i.e. subset of processors) in the system.

2.2 Multiple Cache Configurations for Single/ Multiple Node Systems

In Figure 4, two of the L3 node directory emulators of MemorIES emulate two different cache configurations of the first node of the target system, while the remaining two L3 directory emulators emulate two other configurations of the second node of the target system. Configurations such as this are used to evaluate multiple cache structures for the same workload in parallel.

2.3 Other Applications of MemorIES

Although the primary use of the MemorIES board is to emulate large cache systems, the tool is very flexible and can be programmed to perform many other functions relatively easily by changing the FPGA firmware and recompiling.

Identification of hot spots. The FPGAs can be programmed to treat their private 256MB memory as a table of memory read/write frequency counters either on cache line basis or page basis. These counters help to identify hot spots in cache lines or in memory pages and provide useful insight into program behavior for OS and application tuning.

Trace collection. The on-board memory (which goes up to 8GB with higher density DRAMs) can be used to collect bus traces from the host machine and later dump to a disk in the console machine. The current revision of the MemorIES board is capable of collecting traces containing up to 1 billion 8-byte wide bus references at a time. In contrast to collecting a trace via a simulator, the MemorIES board collects traces in real-time, allowing it to collect large traces in a fraction of the time that conventional methods would take. While it also possible to collect traces using a logic analyzer connected to a running machine, the program that is running must be periodically stopped to allow the logic analyzer to dump the trace data to disk. MemorIES requires no such stoppage, allowing for the collection of large traces without gaps in the trace and without perturbing the program.

NUMA directory emulation. MemorIES can also emulate NUMA directory protocols, for example, a system with 4 NUMA nodes kept coherent using a sparse-directory cache coherence scheme[WEB93]. The memory address space can be partitioned so that one of the 4 nodes is the ‘home’ for that particular partition. The memory requests issued by a processor can now be separated into local or remote requests depending on whether the memory location to which the request is made belongs to the same node as the processor or not. The private 256MB memory present in each of the 4 nodes can be partitioned to hold both the L3 tag directory and the sparse directory belonging to the corresponding ‘home’. If an entry gets evicted out of the sparse directory, then the other L3 nodes can be informed about the eviction so that the entry can also be invalidated in the other L3 tag directories. Because MemorIES is a passive emulator, it is not able to invalidate L2 or L1 cache entries; however, the L2 cache can be turned off or reduced to a smaller size to get a good approximation of the sparse directory behavior using real-time workloads.

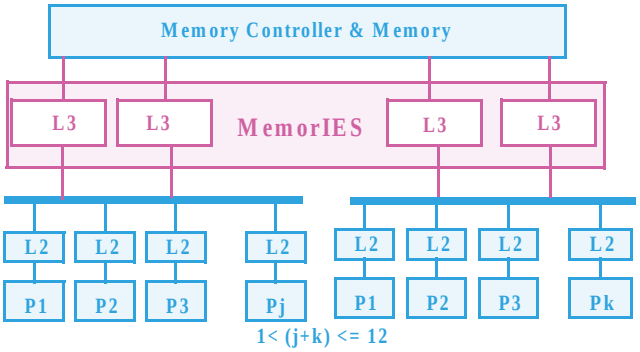

FIGURE 4. Logical Target Machine for Multiple Cache, Multiple Node Emulation

Remote cache emulation. In a similar vein, MemorIES can also model NUMA nodes with remote caches. The private 256MB memory belonging to each node can hold both the L3 tag directory as well as the remote cache tag directory.

3 Design of the MemorIES Board

MemorIES is a self contained board (Figure 5) with a parallel port interface to a console PC and a 6xx SMP memory bus interface to the host machine. The board uses seven Field Programmable Gate Arrays (FPGAs) to implement the programmable cache controller functions and 1GB of SDRAM memory to implement the cache tag and state tables. It has the ability to plug directly into the 6xx bus of the host machine at a maximum speed of 100MHz, or connect to an interposer card to take measurements from systems with a different bus architecture, such as an Intel X86 platform. Different bus architecture measurements require protocol conversion on the interposer card, reprogramming of the FPGA, or changing the command map file if the protocol is similar.

The current design can emulate up to four shared cache nodes. Each emulated shared cache node is implemented using one Altera 10K250 FPGA and four industry standard 64MB SDRAM DIMMs. Table 2 lists the various parameters for the cache emula-

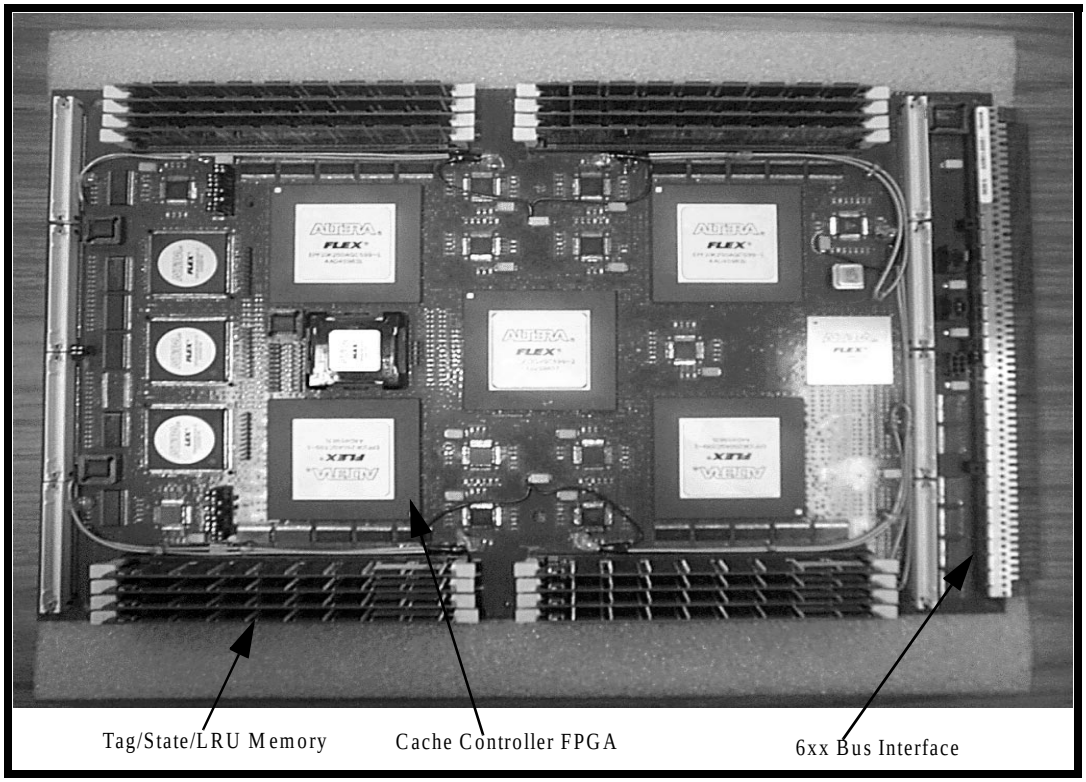

FIGURE 5. The MemorIES Board

tion function of MemorIES. The MemorIES board contains more than 400 counters to count various cache hit/miss events in detail. Each counter is 40-bit wide and can hold performance data for more than 30 hours of real time program execution at the typical 20% bus utilization level. Figure 6 shows the functional flow for cache emulation using MemorIES and Figure 7 shows the physical block diagram of the MemorIES board.

3.1 FPGA Functions

The seven FPGAs that incorporate the MemorIES functionality are the address filter, global events counter and buffer FPGA, the console FPGA and the 4 SMP node controllers (Nodes A through D), as illustrated in Figure 7. Each SMP Node controller can emulate a shared L2, L3 or remote cache. The address filter FPGA is responsible for interfacing with the 6xx bus, filtering out non-emulation related transactions (like retries on the bus), grouping the transactions based on the bus ids and forwarding the transactions to the global events counter FPGA. The global events counter and buffer FPGA has a set of global counters to keep track of events like number of bus cycles, number of reads issued on the bus etc. It also provides buffering to handle occasional bursts (see Section 3.3) and forwards the transactions to one of the node controller FPGAs based on the bus id of the requesting processor and which emulated SMP node that processors belongs to. The console FPGA controls the parallel port interface to the PC via the AMCC chip set and is necessary for all diagnostic activities.

The four cache emulation FPGAs are designed to always run in lock step mode regardless of the cache parameter settings. This simplifies the design for emulating multiple node configurations. The lock step feature also simplifies the control of the transaction buffers in the Console Decode FPGA and the Address Filter FPGA.

3.2 Modeling Cache Protocols

MemorIES has the capability to program a wide range of cache protocols using state transition tables that are easily synthesized as FPGA logic. The cache state transitions are modeled as a lookup table which consists of the type of memory operation, the current state of the cache entry, and the resulting state from other cache nodes. The table lookup map file is loaded into each cache node controller FPGA during the initialization phase. Different state table files could be loaded to different node controller FPGAs to experiment with different coherence protocols during the same measurement.

3.3 Matching Bus and Memory Speeds

The throughput of the SDRAMs implementing state/Tag/LRU functions is roughly 42% of the maximum 6xx bus bandwidth. In order to handle occasional bursts exceeding 42% bus utilization, MemorIES provides transaction buffers between the 6xx bus and the cache control logic. The transaction buffer in the Address Filter FPGA can take in memory operations at a maximum rate of 100 MHz. Operations such as I/O register accesses, interrupts or memory operations that are rejected by other system

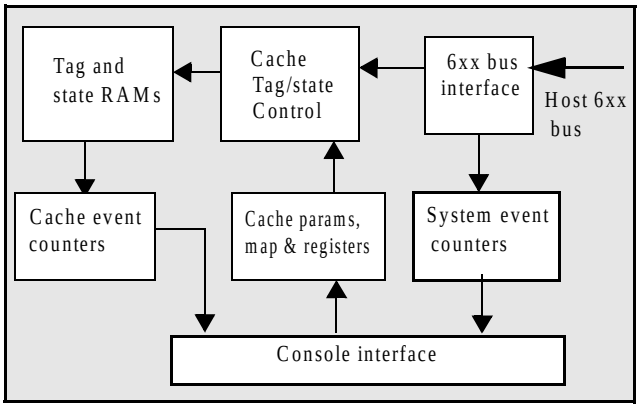

FIGURE 6. Cache Emulation Functional Flow

Feature	Parameters
Cache size	2MB - 8GB
Cache associativity	Direct mapped to 8-way set associative
Processors per shared cache node	1 - 8
Cache line size	128B - 16KB

TABLE 2. Summary of Cache Emulation Parameters

bus devices are filtered out and do not take up any transaction buffer space. The node controller FPGAs contain 512 transaction buffer entries to pace the SDRAM tag directory operations. In the rare event that all the entries in the transaction buffers get filled, the Address Filter rejects further transactions on the bus by posting a retry command. This is a case when the MemorIES board could alter system bus behavior, and hence memory access behavior to some extent. However, in the past few months of use in the labs in high-end database workload environments, the MemorIES board has never once posted a retry command (implying that MemorIES behaves passively). The maximum bus utilization with 8 CPUs always varied between 2% to 20% across 2 platforms, 2 OSes, and 2 benchmarks, indicating that 42% was a safe target for the MemorIES board.

3.4 Limitations

It is important to note that MemorIES is a passive emulator: it snoops on the 6xx bus, but is in general unable to stop and/or inject transactions on the bus.This limitation has two important ramifications. First, MemorIES is unable to simulate the latency of a reference by pausing the requesting processor for a pre-established period of time. Second, when a line gets replaced in the L3 cache, the line cannot be invalidated in the lower levels (L1 and L2). Therefore, it cannot emulate accurately a fully-inclusive L3 cache. It is to be noted that all trace driven simulations using bus traces also have the same limitation.

4 MemorIES vs. Simulation

The MemorIES tool is intended for use as a complement to different simulation technologies. It is much faster than either trace-driven or execution-driven simulation and is thus capable of quick profiling of applications so that bottlenecks can be rapidly identified. After identification, other types of simulation are employed to narrow down the sources of the bottleneck and enable fine-tuning of the system.

4.1 MemorIES vs. Trace-driven Simulation

Faster run time of the evaluation tool is critical to system design time, especially systems designed for large applications. A trace-driven C simulator (which was used as one of the methods to validate the MemorIES design) was used to run varying trace sizes and the resulting run times compared to that of the MemorIES board. Table 3 shows the significant savings in run times of the MemorIES board and clearly indicates that software simulation becomes prohibitive as trace sizes grow. The software simulation optimistically assumes that the entire trace is memory resident and the MemorIES board assumes a 6xx bus speed of 100 MHz with a bus utilization of 20%. In reality, MemorIES can emulate up to 4 different L3 configurations at the same time but this speedup is not indicated.

4.2 MemorIES vs. Execution-driven Simulation

In order to determine the time savings MemorIES provides in evaluating memory hierarchy performance, the SPLASH2 FFT application with 8 threads was run with varying data sizes on an execution-driven simulator, Augmint [NMS+96], as well as on an 8-way IBM S7A SMP. Table 4 shows the execution time of the Augmint simulator vs. the time taken by the MemorIES board to run the FFT program with different data sizes.

While numerous researchers have proposed varying the level of detail of simulation in order to speed up simulation time [RHW+95], and these simulators are quite fast, the slowdown with respect to MemorIES is still quite significant. For example, Embra[WR96], an execution-driven simulator for processors, caches and memory systems, uses dynamic binary translation and reports a slow down of a factor of 7 to 20 for uniprocessor workloads and a factor of 94 to 221 for multiprocessor workloads. Of course, the advantage of a simulator like Embra is that it runs a workload until it reaches a particular point in simulation, and then checkpoint the workload, so that a more detailed simulator can then run from the checkpointed time, rather than from the beginning of the application. MemorIES does not simulate an entire system, and does not allow the positioning of a workload.

5 Case Studies

In this section we present detailed case studies illustrating the use of our MemorIES board. Although MemorIES is a passive emulator, and not a system-level simulator, its ability to process

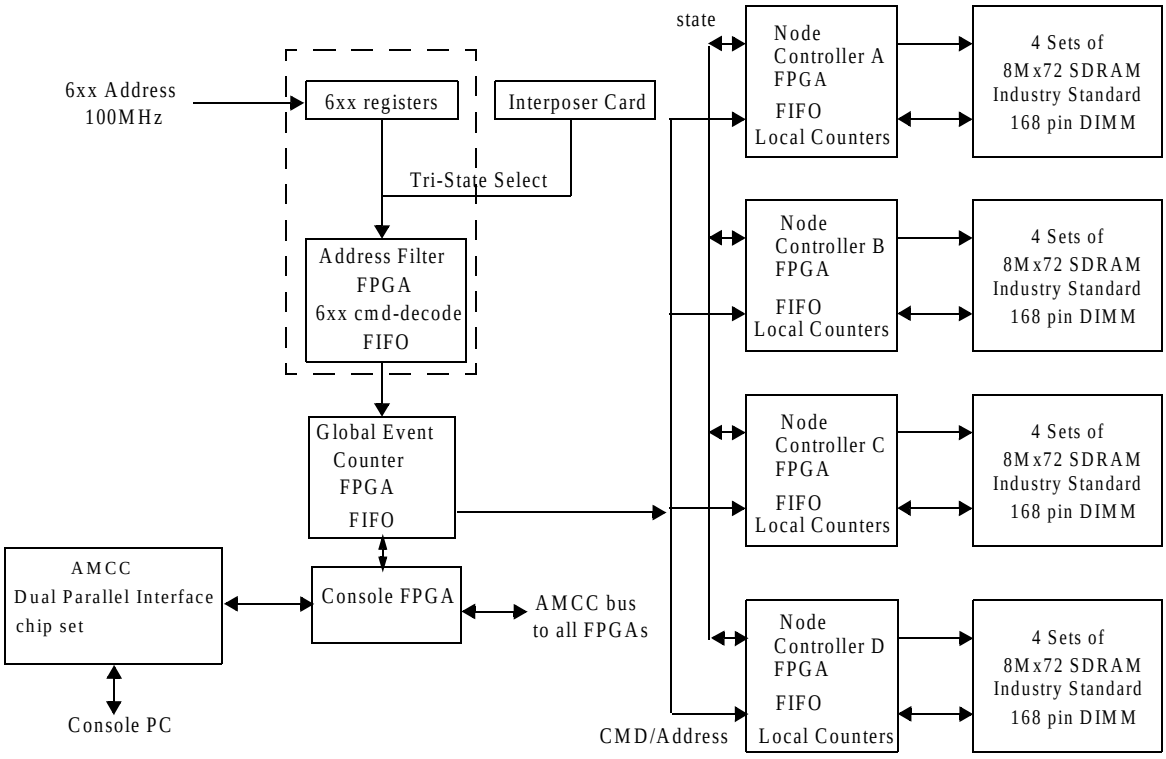

FIGURE 7. MemorIES Physical Block Diagram

large problem sizes and large cache sizes still enables us to perform several important validation and debugging exercises that are not possible with current simulation techniques.

For these experiments, the host machine is an 8-way IBM S7A SMP [IBM] that uses 262 MHz Northstar processors with 8MB individual L2 caches. The system has 16 GB of main memory and over a terabyte of disk space. The L2 cache size and associativity can be changed at boot time from 8MB 4-way set-associative to 1MB direct mapped.

5.1 Case Study 1: Impact of trace length on observed cache miss behavior

Our first case study involves the use of traces in machine evaluation. While trace-driven simulation is a common, low-cost technique used for evaluating memory system behavior in multiprocessors, one big drawback is that collecting and analyzing a large trace that comprises a complete run of a workload can be quite time consuming. Because MemorIES utilizes on-line emulation and real-time data processing, it speeds up trace analysis enormously. A long trace representing 8 hours of runtime, which would be nearly impossible to collect using a simulator due to excessive simulation time, can be processed and processed in 8 hours on MemorIES.

We used the MemorIES board in order to study cache miss behavior in future systems. Because of our ability to process long traces, we discovered an important relationship between trace length and cache misses. Figure 8 shows miss ratios for the TPC-C (150GB database) benchmark, for two trace lengths (10 billion references and 20 million references), and for TPC-H (100GB database), for 400 billion, 200 billion, and 10 billion reference traces. As seen in Figure 8, using inappropriate trace sizes dramatically changes the optimal parameters for performance. Using too small a trace may suggest that larger caches (for example, beyond 128MB in TPC-C) have no impact on miss rate, when in reality larger caches continue to reduce the miss rate. The overestimation of miss rate occurs because the trace is so small that startup effects, rather than steady-state effects, are measured. In particular, when the trace is too small, cold miss behavior is much more dominant

Trace size (number of vectors)	Execution time of C simulator	Execution time of MemorIES
32768	1 second	3.28 milliseconds
262144	8 seconds	26.21 milliseconds
10 million	5 minutes	1 second
10 billion	approx 3 days	16.67 minutes

TABLE 3. Execution Times of C Simulator vs. MemorIES. The C Simulator was run on a 133MHz machine. The host machine for MemorIES was 262MHz.

FFT data size parameter 'm'	Execution time of Augmint	Execution time of MemorIES (Host machine run time)
20	47 minutes	3 seconds
22	3.2 hours	13 seconds
24	13 hours	53 seconds
26	> 2 days	196 seconds

TABLE 4. Execution Time of Augmint vs. MemorIES. Augmint was run on a 133MHz machine, while the host machine for the MemorIES experiment was 262MHz.

an effect than in a real run. Clearly, the longer the trace, the more accurate the resulting simulation can become.

Another interesting result from our trace length studies involves choosing the number of processors per shared L3 cache. We chose to address the issue that we have a given L3 cache size (64MB) and 8 processors, and we wish to consider whether to construct an SMP with all 8 processors sharing the L3 cache, or split the system into several SMPs, each with their own 64MB L3 cache, with all the L3 caches connected on a bus. As shown in Figure 9, the trace length directly impacts this choice. The long trace results indicate that miss ratio increases with increasing number of processors per L3 cache, while the short trace results indicate an opposite trend. When too short a trace is used, the incremental impact of adding more processors per L3 cache is to reduce the cold miss ratio, since the additional processors effectively prefetch data for the other processors that share the L3 cache. With a short trace that is representative of trace lengths in use today (45 million L3 references, or about 600 million instructions), this impact is quite large. However, with a larger trace (over 10 billion L3 references), the effect of reducing cold misses is much smaller. Instead, each processor reaches its steady state. In this steady state, the processors all access their different data sets. These data sets do not overlap completely, and as a result, when multiple processors share an L3 cache, the L3 cache must support a larger aggregate working set (the working sets of each of individual processors). As a result, the miss ratio increases for that L3 cache. This data contradicts the previous data for short traces, and suggests using fewer processors per L3 cache.

5.2 Case Study 2: OS performance debugging

Our use of the MemorIES board for collecting traces is also useful in combination with other techniques for performance debugging. Figure 10 shows a run of the TPC-C transaction processing benchmark on an IBM SMP system over several hours. The important feature of the graph is that there are periodic spikes in the miss ratio around every 5 minutes, no matter what cache size is being modeled. This suggests some sort of software issue unrelated to cache design. Without such a long-running trace, this behavior might not have been observed: instead, it is possible that only the plateaus between the spikes would have been seen. For example, 5 minutes of trace in the system under consideration corresponds to about 2 billion memory accesses on the bus, whereas most traces collected in this time frame are in the range of 20-60 million bus accesses, which makes it very hard for the smaller traces to catch this kind of activity. Even an execution-driven simulator might not have seen this problem, if a sufficiently short-running input set were used, or if the wrong window of time were simulated. After collecting the traces and observing the problem, an OS monitoring tool was used in order to collect more detailed information. This detailed information established a correlation between the periodic spikes in cache misses and a software problem with the OS journaling activity in the file system. Even though the OS team was already aware of this problem from independent experimentation, the MemorIES board was able to quantify the severity of the problem. Upon fixing the problem in the OS the spikes were eliminated, resulting in significant improvements in the system performance.

5.3 Case Study 3: Scaling

Our final case study involves the use of MemorIES for scaling purposes. As an example, we use applications from the SPLASH2 [WOT+95] benchmark suite. SPLASH2 is widely used in execution-driven simulator studies, but because such simulation can be prohibitively expensive, often the problem sizes are scaled so that they can be run in reasonable time, while still maintaining the miss rates and communication patterns of the realistic sizes. Because variants of the problems in the SPLASH2 suite are run daily in research labs, and because the problem sizes are ever increasing, it is valuable to see if the scalings that have been proposed in the past accurately model the behavior of real-world problem sizes. If so, then such scalings can safely be used in place of the real sizes for debugging and performance evaluation. If not, then it is important to understand how the scalings are deficient, in order to suggest more accurate modeling methods.

We chose to run SPLASH applications with sizes more appropriate for today's machines. These sizes are indicated in Table 5, along with their runtimes and memory footprints. The sizes that are run in practice are typically bounded by 2 factors: memory space and application speedup. The problems are typically sized so that they are the largest problem that still fits in the memory of the target machine. Moreover, they are sized so that the speedup is reasonable for that problem size[LEV00]. The sizes we have chosen fit these criteria, and the reasonableness of the selections was confirmed by searching for example runs of these applications in the literature[FQG+92][HLC+99][JS99][WLM+99]. These sizes are quite a bit larger than the sizes used in previous SPLASH studies (see Table 1). For our sizes, the data sets of these applications consume up to several GB of memory, in contrast to the several MB of memory that the data sets of the previous SPLASH studies occupy. Using the MemorIES board, we are able to collect miss rate, miss ratio, and communication statistics and

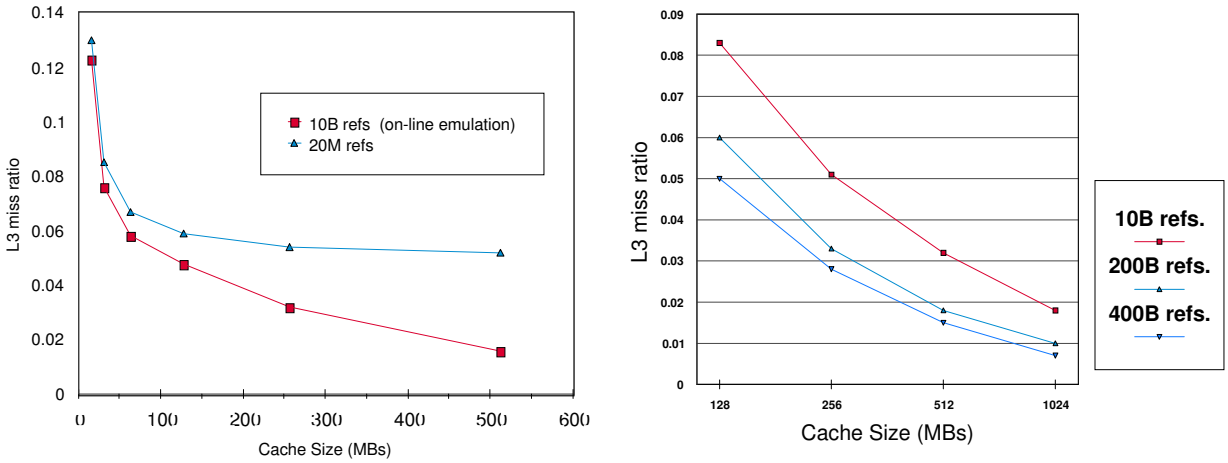

FIGURE 8. L3 Miss Ratio for Different Trace Lengths for TPC-C (left) and TPC-H (right). For TPC-C, a 10 billion reference trace is compared to a 20 million reference trace. For TPC-H, a 400 billion reference trace is compared to both a 200 billion reference trace and a 10 billion reference trace.

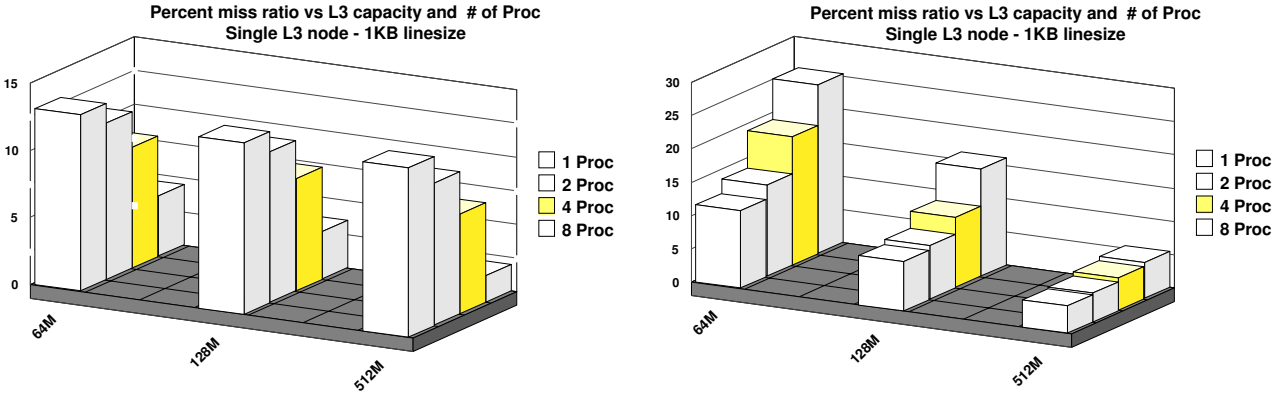

FIGURE 9. L3 Miss Ratio for Different Degrees of L3 Sharing, Short Traces and Long Traces. There are a total of 8 processors. 1 Proc is a configuration in which each processor has a separate 64MB L3 cache, and the L3 caches are connected over a bus. 8 Proc is a configuration in which all 8 processors share a single 64MB cache. The figure on the left represents a run with a trace containing 45 million references, while the figure on the right represents a run with 10 billion references.

profile these applications to see if the observed behavior is close to the typical simulated behavior.

In Table 6 we show the cache miss rates (in misses per thousand instructions) of our applications, both for some of the parameters from previous generation simulations, as well as for the MemorIES board with large caches and large problem sizes. The results from the original SPLASH2 paper were generated using only 1 level of caching, but our system has 2 levels of caching, so we use miss rate, rather than miss ratio, as the comparison metric. Because the L1 and L2 caches in our system are fully inclusive, the miss rate for our L2 can be compared to the miss rate in the SPLASH2 paper[1]. The miss rates were measured using the on-chip counters of the L2 controller of the S7A machine. There are several important points to consider. First, the miss rates from previous studies are vastly different from realistic problem sizes, suggesting that the scalings employed in the past are potentially inaccurate for much larger problem sizes. In addition, the miss rates for our sizes are not necessarily always larger than in the smaller sizes: for FFT, for example, our miss rate is much smaller. Clearly, this suggests that scaling becomes more difficult as the

1. The 1MB 4-way set-associative cache was chosen for the SPLASH2 results because data were available for all applications for that size, and because the level 1 working sets of the small sizes all fit in that cache size. The level 1 working sets for all of our sizes fit in our 8MB L2 cache, which is the actual cache size in our system.

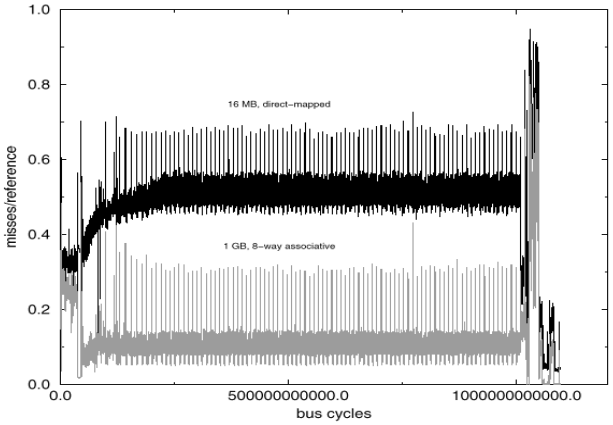

FIGURE 10. TPC-C Miss Ratio Profile. The periodic spikes are due to a bug in the journaling activity in the OS. The top curve is for a 16MB direct-mapped L3 cache, and the bottom curve is for a 1GB 8-way set-associative cache.

Application	Memory footprint (GB)	Runtime, 8MB 4-way set-assoc. L2 (seconds)	Runtime, 1MB direct-mapped L2 (seconds)
FMM (4M particles)	8.34	633	653
FFT -m28 -l7	12.58	777	853
OCEAN -n8194	14.5	860	971
WATER (spatial, 125^3 molecules	1.38	1794	2008
BARNES-HUT (16M bodies)	3.1	2021	2082

TABLE 5. SPLASH2 Application Characteristics. All applications are run with 8 processors.

problem sizes get larger. Moreover, the miss rates are rather large (0.2-8.2), as are the miss ratios (misses/reference), which range from 5-36% (not shown here), even though the L2 caches are large, suggesting that future systems can benefit from large caches, despite the possible increase in cache-to-cache sharing. In fact, if we add L3 caches, we see a significant number of misses being satisfied (Figure 11). In addition, we see that the miss ratios and miss rates are monotonically decreasing, further suggesting an incentive for large L3 caches. Although this does result in an increase in cache-to-cache sharing, preliminary calculations based on latencies and miss ratios suggest that performance improves from 2-25% for these applications, and for no L3 cache size do we see performance degradation.

To analyze the communication patterns, we configure MemorIES in a NUMA configuration, in which several SMPs are connected together and each SMP has an L3 cache. The two multiple L3 configurations studied are a) Two SMP nodes with 4 processors sharing each L3 and b) Four SMP nodes with 2 processors sharing each L3. Figure 12 shows how references (misses from L2) are satisfied on the SMP memory bus for FFT, Ocean and FMM. This kind of breakdown is useful for understanding the characteristics of an application for tuning its performance as well as choosing the optimum L3 size for a given set of price and performance criteria. The general trends reinforce previous results, although a precise comparison cannot be made since this type of traffic was not published in [WOT+95]. Some key observations from these results are:

Application	Miss rate for sizes in SPLASH2 paper, 1MB cache, 4-way set-associative	Miss rate for sizes in this paper, 8MB L2 cache, 2-way set-associative
FMM	0.33	0.7
FFT	5.5	0.3
Ocean	3.7	8.2
Water	0.073	0.2
Barnes	0.11	0.3

TABLE 6. Miss Rates (misses per thousand instructions) for Various SPLASH2 Applications

- FFT and Ocean have relatively small modified or shared interventions[1], indicating less data sharing among threads. For these kinds of applications more attention should be paid to memory data placement in a NUMA system, and tertiary caches are definitely useful.
- FMM has a significant amount of modified and shared intervention traffic relative to the other applications, indicating more data sharing. Applications like this will gain from efficient cache-to-cache transfer implementations in an SMP or NUMA environment.

We can also use the MemorIES board for scaling studies involving transaction processing, decision support, and web server workloads.

6 Prior Work

While the majority of systems studies have involved various methods of simulation, in this section we focus on related work in hardware-based emulation. The RPM project at USC [DGJ+98]

1. A shared intervention occurs when a processor misses on a line in its L2 cache and must retrieve a shared copy from another L2 cache, rather than going to memory or the L3 cache to fetch the line.

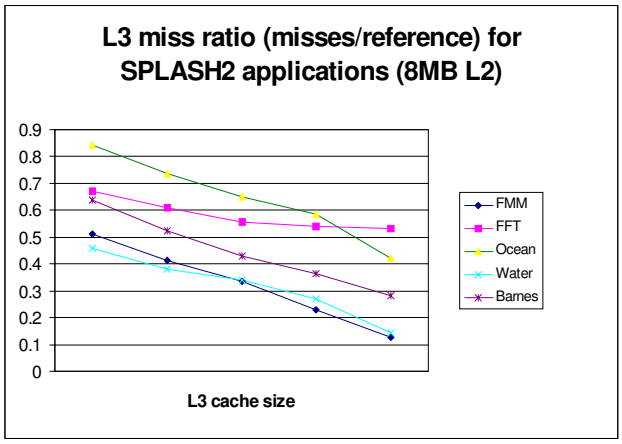

FIGURE 11. L3 Miss Ratio with 8MB 4-way set-associative L2. 8 processors are used, all sharing one L3 cache. The line sizes of the L2 and L3 cache are 128B.

has done extensive work on hardware prototyping using FPGAs. The MemorIES work, however differs from the RPM work in many ways. First, RPM emulates a complete system including several levels of caches, network and IO control, and a variety of NUMA and message passing multiprocessors. The scope of MemorIES is rather limited. We concentrate only on the cache and cache protocol emulation. However, given the systems we are considering and the focus of our design efforts (i.e., the memory system), this trade-off is appropriate. A second difference between RPM and Memories is that MemorIES plugs into a commercial SMP and logically partitions the host SMP into smaller target SMPs using CPU IDs on the bus. In contrast, RPM emulates each individual processor separately and builds a complete system from scratch. It would require a great deal of work to make a realistic commercial application (such as databases) run on RPM. Since MemorIES plugs into a real, state-of-the-art SMP, it can collect data on important real world applications including databases that run on that machine without having to worry about porting a huge application to a new machine. Finally, RPM operated at a smaller frequency (5-20MHz) than the contemporary memory bus speeds prevalent in its time. In contrast, MemorIES operates at the full speed of the system's memory bus (100MHz). As a result, it does not alter the program behavior.

Another type of emulation common in commercial system design involves synthesizing a board full of FPGAs to mimic the behavior of a particular chip in a target system [QUI]. The FPGAs are synthesized using the actual hardware description language that is used to describe the target chip, so that any bugs in the chip can be discovered quickly. This type of system is not generally meant for experimentation and performance debugging, but is used primarily for functional validation. In addition, it is expensive to synthesize and also runs slower than the target chip. MemorIES, in contrast, runs at the speed of the host system and is configurable to represent numerous memory systems, allowing performance debugging, but is not meant to be a tool for functional validation.

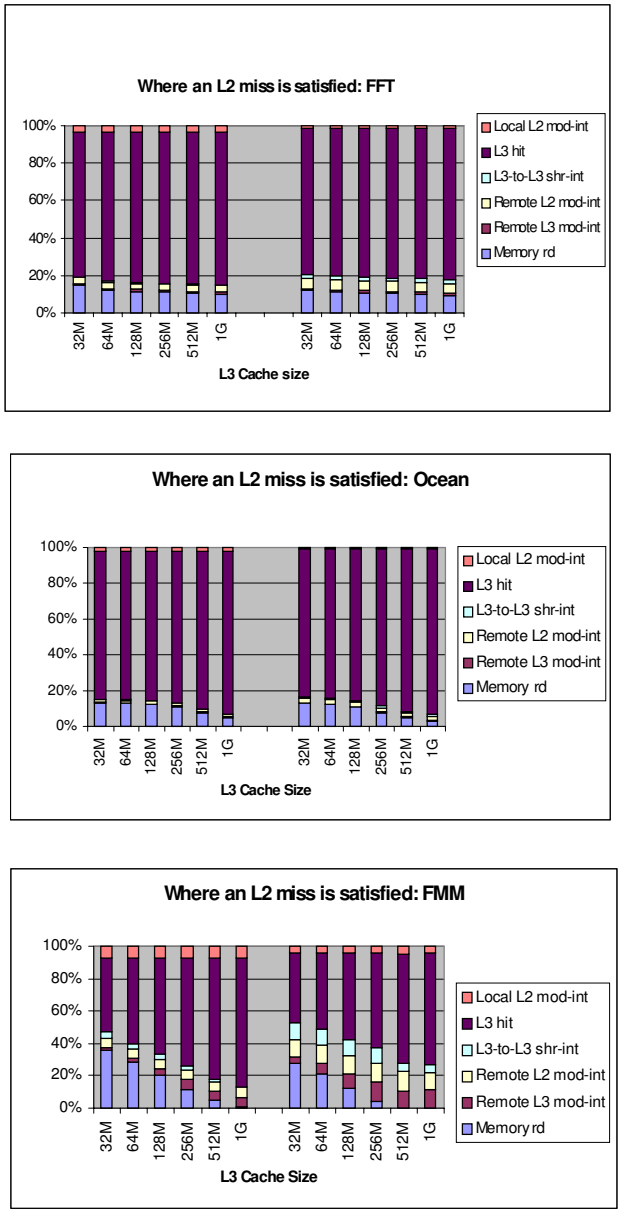

FIGURE 12. Where an L2 Miss is Satisfied in FFT, Ocean, and FMM. The leftmost bars in each graph are 2-node systems with 4 processors per L3. The rightmost bars are 4-node systems with 2 processors per L3. Both L2 and L3 caches are 4-way set-associative. The L2 is 8MB with a 128B line size, and the L3 has a 1KB line size. mod-int and shr-int are modified- and shared-interventions, respectively.

7 Summary

In this paper we presented the design of MemorIES, a real-time programmable hardware emulation tool used for evaluating large caches and SMP cache protocols for future server systems. MemorIES produces memory access and cache performance statistics on line while real life applications are running on the host machine, without slowing down the host system or degrading its

performance in any manner. MemorIES complements system simulation techniques by providing fast results for a wide design space: once the design space has been pruned or once performance bottlenecks have been discovered, detailed simulation can then be used to pinpoint the proper parameter choice or determine the specific source of the bottleneck. We demonstrate its usefulness in several case studies, and find several important results. First, using traces to perform system evaluation can lead to incorrect results (off by 100% or more in some cases) if the trace size is not sufficiently large. For example, a small trace may suggest that a there are only marginal gains when increasing the cache size beyond a certain point, while a larger trace suggests that caches continue to reduce the miss rate significantly beyond that cache size. Second, MemorIES is able to detect performance problems by profiling miss behavior over the entire course of a run, rather than relying on a small interval of time. Finally, we observe that previous studies of SPLASH2 applications using scaled application sizes can result in optimistic miss ratios relative to real sizes on real machines, providing potentially misleading data when used for design evaluation.

8 Acknowledgments

We are thankful to our management for their support, as well as to the numerous contributors who helped with the project definition, software development, design and validation of MemorIES. We are especially thankful to Wayne Nation, Michael Rosenfield, Lorraine Herger, Mickey Tsao, Nathan Lee, Luke Wong, Steven VanderWiel, Steve Kunkel, Phil Vitale, Marius Pirvu, and Anthony Nguyen for their contributions at various phases of the project.

References

[ALT] Altera Corporation, Flex10K Embedded Programmable Logic Family Data Sheet. http://www.altera.com.

[BDH+99] E. Bilir, R. Dickson, Y. Hu, M. Plakal, D. Sorin, M. Hill, and D. Wood. Multicast Snooping: A New Coherence Method using a Multicast Address Network. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*. May 1999.

[DGJ+98] M.Dubois, A. Gefflaut, J. Jeong, A. Moga, and K. Oner, “Rapid prototyping on RPM-Methodology and Experience,” IEEE Design and Test of Computers, pp 112-118, July-Sep. 1998.

[FW97] B. Falsafi and D. Wood. Reactive NUMA: A Design for Unifying S-COMA with CC-NUMA. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*. June 1997.

[FW99] B. Falsafi and D. Wood. Parallel Dispatch Queue: A Queue-Based Parallel Programming Abstraction to Parallelize Fine-Grain Communication Protocols. In *Proceedings of the 5th International Conference on High-Performance Computing.* January, 1999.

[FQG+92] D. Fullagar, P. Quinn, C. Grillmair, J. Salmon, and M. Warren. N-body Methods on MIMD Supercomputers: Astrophysics on the Intel Touchstone Delta. In *Proceedings of the Fifth Australian Supercomputing Conference.* December 1992.

[HLC+99] Y. Hu, H. Lu, A. Cox, and W. Zwaenepoel. OpenMP for Networks of SMPs. In *Proceedings of the Thirteenth International Parallel Processing Symposium*. April 1999.

[IBM] IBM Corp., RS/6000 Enterprise Server S7A Users’ Guide, Oct. 1998

[LEV00] J. Levesque. Personal Communication. April 2000.

[MNL+97] M. Michael, A. Nanda, B.-H. Lim, and M. Scott. Coherence Controller Architectures for SMP-Based CC-NUMA Multiprocessors. In *Proceedings of the 24th International Symposium on Computer Architecture*. June 1997.

[NHO+98] A.K. Nanda, Y. Hu, M. Ohara, M. Giampapa, C. Benveniste and M. Michael. The Design of COMPASS: An Execution Driven Simulator for Commercial Applications Running on Shared Memory Multiprocessors. In *Proceedings of International Parallel Processing Symposium*, April 1998.

[NMS+96] A.-T. Nguyen, M. Michael, A. Sharma and J. Torrellas. The Augmint Multiprocessor Simulation Toolkit for Intel x86 Architectures. In *Proceedings of the International Conference on Computer Design*, pp. 486-490, Oct.1996.

[PRA+97] V. S. Pai, P. Ranganathan, and S. Adve. RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors. In *Proceedings of the Third Workshop on Computer Architecture Education*. Feb. 1997.

[QUI] Quickturn Corporation. http://www.quickturn.com

[RHW+95] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta. Complete Computer Simulation: The SimOS Approach. In *IEEE Parallel and Distributed Technology.* Fall 1995.

[JS99] D. Jiang and J. P. Singh. Scaling Application Performance on Cache-coherent Multiprocessors. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*. May 1999.

[TPC] Transaction Processing Council: http://www.tpc.org

[WEB93] W.-D. Weber. Scalable Directories for Cache-Coherent Shared-Memory Multiprocessors. Stanford University Technical Report CSL-TR-93-557. Jan. 1993.

[WLM+99] Z. Wang, J. Lupo, A. McKenney, and R. Pachter. Large Scale Molecular Dynamics Simulations with Fast Multipole Implementations. In *Proceedings of SC99*. Nov. 1999.

[WOT+95] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, June 1995.

[WR96] E. Witchel and M. Rosenblum. Embra: Fast and Flexible Machine Simulation. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems* (SIGMETRICS). 1996.