A Framework for Profiling Multiprocessor Memory Performance

Diana Villa, Jaime Acosta, and Patricia J. Teller
The University of Texas at El Paso, Department of Computer Science
demarquez@utep.edu, jacosta@cs.utep.edu, pteller@cs.utep.edu

Bret Olszewski, breto@us.ibm.com
IBM Corporation-Austin

Trevor Morgan, trevormorgan31@hotmail.com
Exxon/Mobil

Abstract

Because of the increasing gap between processor frequency and Dynamic Random Access Memory (DRAM) speed, the performance of the memory subsystem typically governs that of the system as a whole. This is especially true for symmetric multiprocessor systems (SMPs). Therefore, performance evaluation methodologies that facilitate the analysis and optimization of the memory subsystem are essential. This paper describes such a methodology, a performance evaluation framework, and demonstrates its power, speed, and flexibility in the context of a study of the TPC-C benchmark, executed on eight- and 32-processor IBM ~pSeries 690 (p690) systems. The framework facilitates analysis of sampled performance monitor event traces that are collected in real time. The analyses are used to characterize the locality of reference exhibited by TPC-C data loads at the various levels of the memory hierarchy and evaluate the efficacy of design aspects of and policies associated with the p690 memory hierarchy w.r.t. workload demands.

1. Motivation

Because of the increasing gap between processor frequency and DRAM speed [4], one of the major architectural design considerations for any computer system is that of the memory subsystem. In most cases, the performance of the memory subsystem governs that of the system as a whole [14]. This is true especially for modern, as well as future, symmetric multiprocessor (SMP) systems. Consequently, performance evaluation methodologies that facilitate the analysis and optimization of the memory subsystem are essential to the development of next-generation computer systems.

Analysis of the memory access behavior of a workload executed on a particular computer system can reveal ways to improve performance, through modifications to the application, operating system, hardware, or combinations of these. However, for SMP systems this type of performance evaluation is becoming increasingly more challenging. For example, a traditional way to analyze cache performance is through the use of address traces generated by the hardware or by software architecture simulation, as is the case with SimOS [11]. As systems become faster and caches become increasingly larger, it is more difficult to collect traces that are long enough to accurately model the memory hierarchy of even a single processor. Furthermore, for large, complex workloads, such as on-line transaction processing (OLTP) workloads, executed on multiprocessors, system simulation requires as much disk space as the workload, typically multiple terabytes by today's standards, and usually more memory. Additionally, the effort to simulate a large multiprocessor system is intimidating. One alternative to tracing is a cache simulator built in hardware and connected to a running system [10].

Our approach to meeting the challenge of multiprocessor performance evaluation uses sampled performance monitor event traces collected in real time and a powerful, fast, and flexible performance evaluation framework. The traces are captured via on-chip performance counters that recognize events such as level-two (L2) cache misses, stored in databases, and post processed in conjunction with a partitioned address space map of the workload to gain insights into application behavior and performance. Report-generation software facilitates the analysis of the traces and provides graphical output that can be used to understand when the workload and architecture complement one another in terms of performance and identify performance bottlenecks, with the intent of identifying ways to alleviate them.

In this paper, the framework is described and its power and flexibility are demonstrated in the context a study of the memory access behavior and performance of

a large, complex OLTP workload, the TPC-C benchmark, executed on eight- and 32-processor IBM ~pSeries 690 (p690) systems. The study characterizes the locality of reference exhibited by TPC-C data loads at the various levels of the memory hierarchy and evaluates the efficacy of design aspects of and policies associated with the p690 memory hierarchy with respect to workload demands.

The paper continues with Section 2, which presents related research. Next, Section 3 explains why the study presented in the paper focuses on L2-cache data-load misses and the TPC-C benchmark. Section 4 describes our data collection methodology, i.e., the workload under study, the platform from which the data was collected, the events of interest, and the data collection tools. Data analysis is the focus of Section 5, which describes the tools and methodology used for data analysis, and discusses a sample of the results of the study. Finally, Section 6 summarizes the paper and presents conclusions and future work.

2. Related Research

Performance monitor event traces captured via performance counters have been used to characterize application behavior in the past. Barroso et al. [2] use event traces, captured by tools such as IPROBE and DCPI (Digital Continuous Profiling Infrastructure) [1, 3], to characterize applications, including OLTP workloads, executed on a four-processor AlphaServer 4100 using Oracle 7.3.2. And, Keeton et al. [6] use performance monitors to analyze the behavior of an OLTP workload executed on a four-processor Pentium Pro-based server. Both efforts explore the performance effects of architectural modifications. In [2] this is done by workload characterization, accomplished by source code instrumentation, coupled with simulation methodologies; in [6] this is accomplished by physically changing the hardware. Desikan et al., like Barroso et al., use the DCPI tool [1] but Desikan et al. use it to validate an Alpha 21264 simulator by sampling events that are used to derive performance measurements for the Compaq DS-10L workstation.

With respect to the performance of TPC-C, Tsuei et al. [12] study TPC-C executed on an unidentified Sun Microsystems 16-processor shared-memory multiprocessor with 4GB of memory using IBM's DB2 for Solaris version 2.1.1. Leutenegger and Dias [8] study TPC-C executed on an unidentified multiple-node distributed system. Both efforts investigate TPC-C's buffer hit rate. Leutenegger and Dias also investigate the memory access characteristics of TPC-C and show that data access skew, i.e., non-uniform data memory access, exists at the tuple and page levels. Our initial results [9], which indicate that load accesses are concentrated in certain memory regions and within those regions smaller defined areas are heavily accessed, corroborate the study of Leutenegger and Dias.

Unlike the research described above, Itzkowitz, et al. [7] discuss and demonstrate the use, on a dual 900 MHz UltraSPARC-III Cu Sun Fire 280R™ system, of extensions to the Sun ONE Studio™ compilers and performance tools that provide information related to the data space of an application. This information, gathered either by clock or hardware-counter profiling, provides per-instruction details of memory accesses in the annotated disassembly as well as data aggregated and sorted by object structure types and elements. Compiler-generated padding introduces minor inaccuracies but collection perturbation can be controlled through configuration of the processors' counter overflow rates. Future work described by Itzkowitz, et al. includes analysis of event data addresses by machine entity, for example, memory segment, page, and cache line. Note that our methodology has already been used to perform analysis of this kind [9, 13].

The major differences between our work and the related research described above is the scale of the systems and the methodology used. Itzkowitz, et al. use a two-processor system; Barroso et al. and Keeton et al. use a system with four processors; and Tsuei et al. use one with 16 processors. In contrast, we analyze performance data obtained from both eight- and 32-processor systems. In addition, our work attempts to extract information about the dynamic behavior of a large, complex application with a considerably simpler, more powerful, faster, and, in some cases, more precise methodology. As described in Section 4, our methodology does not require source code instrumentation; data collection does not perturb workload execution behavior. And, as described in Sections 5 and 6, our methodology is not restricted to memory access behavior analysis; instead, it provides the capability to analyze workload execution behavior in a myriad of ways.

3. Why Study L2-cache Data-load Misses and TPC-C?

The sampled event traces studied in this paper were generated via the Performance Monitoring Units (PMUs) of POWER4 microprocessors of eight- and 32-processor, i.e., eight- and 32-way, p690s executing TPC-C. The PMUs were programmed to monitor events triggered by L2-cache misses.

TPC-C, an OLTP application, is used to demonstrate our methodology because it is understood fairly well, is representative of workloads of interest to IBM customers, and commercial workloads, such as TPC-C, are run on a vast majority of today's commercial servers [5]. Furthermore, our preliminary data analysis of information concerning the data access streams produced by L2-cache

data-load misses generated by TPC-C executing on a p690 indicates that there is opportunity for performance enhancement [9, 13]. In particular, it reveals that accesses to particular areas of the address space, e.g., working storage and the buffer pool, and components of the operating system, may be targets for performance improvement.

There are several events that could have been the focus of this study; these include instruction cache (Icache) misses, translation-lookaside buffer (TLB) misses, address-only coherence operations, and uncached memory accesses for I/O. However, as explained below, the performance impact of these events pale in comparison to high-penalty L2-cache misses.

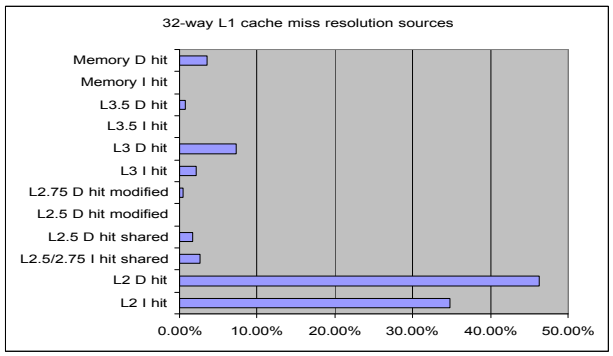

Figure 1. TPC-C instruction and data access sites for 32-way p690

From a CPI viewpoint, TPC-C's most dominant memory subsystem operation is L2-cache miss service. As shown in Figure 1, which indicates where in the memory hierarchy level-one (L1) cache misses are satisfied, over 50% of L1 data-cache (Dcache) misses miss in the L2 cache and are satisfied by either a level-three cache (L3 D hit) or main memory (Memory D hit). As shown in Table 1, these are high-penalty misses, which if decreased, could have a positive impact on performance.

L2-cache Access Site	Load Latency
L2 cache	12 cycles
L2.5 cache	73 cycles
L2.75 cache	96 cycles
L3 cache	112 cycles
L3.5 cache	143 cycles
Main memory	320 cycles

Table 1. Eight-way p690 approximate load latencies

With respect to Icache misses, on the POWER4 microprocessor it is possible to count instruction and data cache miss rates, and to sample the instruction and data addresses associated with Dcache misses. However, it is not possible to sample the instruction addresses associated with Icache misses. This does not pose a problem since Icache misses tend to have a minimal effect on hardware performance due to the fact that the Icache footprint tends to be well cached in large L2 and L3 caches. As Figure 1 shows, this is the case for TPC-C, i.e., few Icache misses are resolved beyond the on-chip L2 cache.

The TPC-C TLB miss rate is reduced by mapping the database buffer pool using 16MB "large pages" rather than the standard 4KB pages. Additionally, unlike cache and TLB misses, address-only coherence operations do not involve movement of data. For example, consider the case where a cache line is held shared by two processes on different POWER4 chips. When one processor stores into the cache line, an address-only operation is initiated to ensure that other caches invalidate copies of the shared data. Because data is not transferred for these operations and the p690 has very high address bus bandwidth, their impact on POWER4-based system performance tends to be small.

Finally, uncached memory accesses for I/O via loads and stores have very high latency. The latency is driven by the fact that many of these operations must pass all the way through to a PCI adapter for acknowledgement. Luckily, most of the actual I/O traffic is handled by DMA, which is asynchronous to processor execution. Accordingly, the high-latency uncached accesses tend to be fairly infrequent.

4. Data Collection

The performance data collected for this study are traces of sampled events associated with the servicing of L2-cache data-load misses generated by TPC-C while executing on eight- and 32-way IBM p690 systems. Below we describe the workload and the compute platform, as well as the monitored events and the methodology used to collect the sampled traces.

4.1. Experimental Platform

4.1.1. Workload: TPC-C. To collect the data used in this study, a fully-implemented TPC-C benchmark drives a commercially-available relational database, which was compiled using the IBM C for AIX version 5 compiler. The TPC-C (Transaction Processing Performance Council Benchmark C) workload [16] is a well-known benchmark that emulates update-intensive and read-only transactions found in complex OLTP application environments [12]. It has been used widely in the database server industry as a basis of server performance analysis and platform comparison.

4.1.2. Compute Platform: IBM ~pSeries 690. The ~pSeries 690 family of SMP architectures includes the

eight- and 32-processor configurations used in this study [15, 17]. The operating system for these configurations is AIX version 5.2. The Multi Chip Module (MCM) is the building block of the architecture. An MCM contains four chips, each of which is comprised of two 1.3 GHz POWER4 processors, and, thus, eight processors. In general, the eight- (32-)processor configuration contains one (four) MCMs, but in this study a two-MCM eight-processor system is used – each MCM contains four "single core good" chips, each of which has only one functional processor.

For the configurations under study,

- each CPU has a 64KB L1 Icache and a 32KB L1 Dcache;
- each chip has a 1.44MB L2 unified cache shared by the two processors on the chip;
- the four chips/eight processors on an MCM share a 128MB L3 unified cache; and
- main memory is 128GB (256GB) for the eight-(32-)way p690.

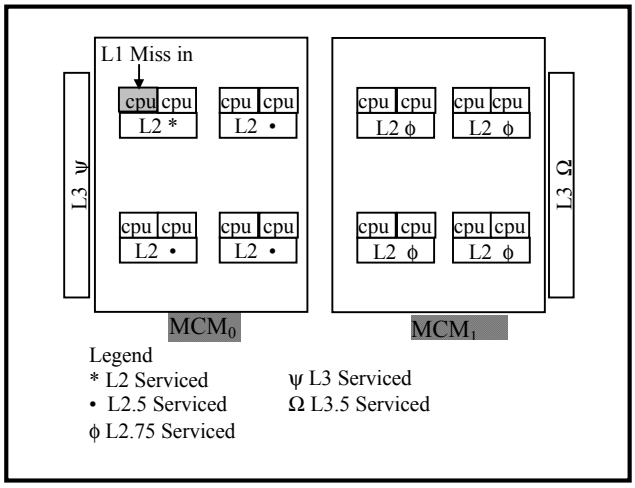

Figure 2. Two (double core good) MCMs of 32-way p690

The L1- and L2-cache line size is 128B, while the L3-cache line size is 512B. Each L3 cache line is partitioned into four 128B sectors. Data private to and shared by processes are managed via the cache coherence protocol implemented in the p690. As illustrated in Figure 2, an L2-cache miss for either type of data generated by a processor in an MCM can be serviced at five different levels of the memory hierarchy:

1. another L2 cache within the same MCM, i.e., a *local L2 cach*e, the *L2.5 level*;
2. an L2 cache in another MCM, i.e., a *remote L2 cache*, the *L2.75 level*;
3. the MCM's L3 cache, i.e., the *local L3 cache*, the *L3 level*;
4. an L3 cache in another MCM, i.e., a *remote L3 cache*, the *L3.5 level*; and
5. main memory.

4.1.3. Monitored Events. A hit event that is generated by an L2-cache miss is classified according to the level at which the miss is serviced and the state of the referenced cache line. An L2-cache miss serviced by the L2.5 level generates one of two types of events: an L2.5-shared hit (L25_SHR) or L2.5-modified hit (L25_MOD). L25_SHR denotes that, although the referenced cache line may reside simultaneously in more than one L2 cache, a local L2 cache services the miss. L25_MOD denotes that the referenced line resides in only one L2 cache, a local L2 cache, which services the miss. A modified block is owned exclusively by one cache and contains more recent data than the backing physical memory.

Similarly, L2-cache misses serviced by the L2.75 level generate either an L2.75-shared hit (L275_SHR) or L2.75-modified hit (L275_MOD) event. The former denotes that the referenced cache line resides in one or more L2 caches but not in a local L2 cache, and is, thus, serviced by a remote L2 cache. The latter denotes that the requested line resides in only one L2 cache, a remote L2 cache, which services the miss.

At the L3 level, the cache-hit events are called L3 shared (L3_SHR), L3 modified (L3_MOD), L3.5 shared (L35_SHR), and L3.5 modified (L3_MOD). L3_SHR denotes that the requested cache line resides in the local L3 cache, from where it will be accessed, but may reside in other L3 caches as well. L35_SHR denotes that the requested line resides in at least one remote L3 cache, but not in the local L3 cache. L3_MOD (L35_MOD) denotes that the requested line resides in only one cache, the local L3 cache (a remote L3 cache), where it will be accessed.

For this study we consider all these events in addition to main memory hit events (MEM). However, instead of monitoring the four events associated with the L3 level of the memory hierarchy, only two events are monitored: L3 and L3.5 hits (L3 and L35).

4.1.4. Event Trace Sampling Methodology: PMU, *eprof*, and *trcrpt*. On ~pSeries hardware, valuable performance information for a section of code or an entire program can be collected through the use of the POWER4 microprocessor performance-monitoring facilities and tools. This information is delivered in the form of aggregate counts and, for selected models, sampled traces of data associated with user-specified events. Up to eight events can be monitored concurrently by a POWER4 performance-monitoring unit (PMU), which has eight counters. Special-purpose registers, only accessible via the operating system through a programming interface that accesses the registers through a kernel extension, control the state of the counters. This interface permits,

among other things, the specification of the events to be monitored and the execution points at which to start and stop counters.

Sampled performance monitor event traces, used in this paper to study the events described in Section 3.3, are collected via two tools: tprof and eprof. *tprof* is a time-based profiling tool that is part of the AIX operating system. *eprof* is an in-house IBM tool that uses tprof functionality for data collection and reduction and is tied to the PMU on selected ~pSeries hardware models. *eprof* is used to program the PMU to sample hardware countable events at a defined rate; the default rate is approximately 100 events per second per processor. When an event is sampled, i.e., at each increment of the performance counter, the instruction address and data address (if applicable) are captured by the PMU, and a performance-monitor (PM) interrupt is delivered. The PM interrupt causes the sample information to be extracted from the PMU and an AIX trace hook to be generated and added to the trace. The AIX trace hook describes the associated trace record. Using the AIX trace allows samples to be either written to disk or collected via a daemon that can summarize the data. The profiling also enables selected AIX trace hooks, such as those related to dispatching, so that the sampled events can be correlated with the processes/threads. If AIX trace is used to collect events in a file, the file can be formatted with the *trcrpt* utility to create a time-stamped text file of events. For this study, we used trcrpt as well as a program that reads the formatted trace and extracts summary information. The specified output of trcrpt includes the effective instruction and data addresses, the process and thread ids, and the timestamp for each sampled event.

Event	Sample Count	
	8-way p690	32-way p690
L2	312,252	259,716
L25_MOD	313,431	197,592
L25_SHR	748,064	n/a
L275_MOD	126,376	167,485
L275_SHR	835,339	n/a
L3	301,791	170,910
L35	121,274	172,008
MEM	272,835	262,941

Table 2. Event sample counts

The sampled event traces used in this study were generated using the default sampling rate. If the sampled event is processor cycles, time-based sampling is accomplished and a sample is collected every 10 milliseconds. In contrast, if the event is one that occurs at a variable rate, e.g., cache misses, and if the rate of event occurrence is greater than the default sampling rate, then eprof adjusts the sampling rate to approximate the default rate. Accordingly, the interval between PM interrupts can be variable. Because some events occur more often than others, it follows that a different number of samples are collected for different types of events. As shown in Table 2, despite the adoption of the default sampling rate and a 10-minute workload, this is the case for the events studied in this research. Event-based sampling is important for long-running programs, like TPC-C, with extremely large numbers of events. Even using event-based sampling, the amount of samples collected for the L25_SHR and L275_SHR events was so large that reduced eight-processor sample counts are used and the 32-processor events are not analyzed.

5. Data Analysis

This section describes the tools used to perform the data analysis, the partitioning of the address space, and the results of the data analysis.

5.1. Methodology

Through the use of a set of tools implemented in Java, each sample is processed and stored in a MySQL database according to the following:

1. monitored workload,
2. number of processors used to execute the workload, and
3. sampled event.

For example, database tpcc_32_g48c1 stores the sampled event traces associated with the L2-cache data-load misses resolved in the L3 caches (event g48c1) for the TPC-C benchmark executed on a 32-processor system. Each database consists of 12 tables that store information related to the experiment itself, e.g., a description of the workload and machine being used, as well as the effective instruction and data addresses, process and thread ids, and timestamp of each sample. Once the sampled event traces are loaded into their corresponding databases, a second set of Java-based tools is used to query the database and generate default and customized reports in the form of formatted text files. The files are transformed into graphs via a spreadsheet application with built-in graphing capabilities.

Storing the sampled performance monitor event traces in databases facilitates data analysis, providing countless ways to easily examine and explore the data. Accordingly, the analysis and results presented in this paper are only a sample of the kind of information that can be obtained using this methodology.

5.2. Data Partitioning

The segment and page sizes are 256MB and 4KB, respectively. Based on a process model, TPC-C allows for per-process private data address regions and a shared data

address region. The latter contains database state information and the buffer pool. The buffer pool is the largest consumer of physical memory, containing unmodified data currently on disk and data modified by transactions but not yet updated on disk. Since the size of the database is much larger than physical memory and the pattern of access to disk data is unpredictable, disk I/O is continuous. Incoming database transactions are passed off to idle processes for service. The number of processes available for processing transactions is based on the number needed to achieve nearly 100% processor utilization. Because most transactions experience some number of disk I/Os, many transactions must be executing concurrently to maximize processor utilization.

5.3. Results

Although our performance evaluation framework facilitates various forms of analysis, the analyses presented in this paper have two main goals: (1) characterization of the locality of reference exhibited at the various levels of the memory hierarchy by TPC-C private and shared data loads and (2) evaluation of the effectiveness of design aspects of and policies associated with the p690 memory hierarchy w.r.t. workload demands. These analyses focus on regions of the TPC-C address space referenced by 90% of L2-cache data-load misses, i.e., the Buffer Pool and Data/BSS/Heap, tracking the missing L2-cache lines across the levels of the hierarchy. Even though we have thoroughly studied the memory references to all regions in the TPC-C address space, only analyses of the most-referenced shared and private data regions are presented below.

5.3.1. Shared Data. Shared data, e.g., global variables and application code, are accessible by every TPC-C and database process in the system. Ideally, shared data remains resident at the high levels of the memory hierarchy until it is no longer referenced. For the p690 this means that shared data remains in the extensive L2-cache level, accessible via L2, L2.5, and L2.75 hits, until it is no longer referenced. Hits at these levels of the hierarchy carry smaller access penalties than hits at the L3-cache level or memory and, thus, lead to better performance.

The Buffer Pool is a shared address region that stores data associated with the most heavily accessed database tables. It serves as a shared database cache between the application and the database management system and is the largest address region in the TPC-C address space. As such, during the 10-minute observation interval of TPC-C's execution on the 32-way p690, it is the target of approximately 36% of the sampled L1-cache data-load misses. Figure 3 shows the distribution of these data-load hits among the levels of the p690 memory hierarchy. For each level of the hierarchy, the dark-colored bar represents the number of data-load hits, while the light-colored bar represents the unique cache line count. The number of unique cache lines referenced gives an indication of locality of reference.

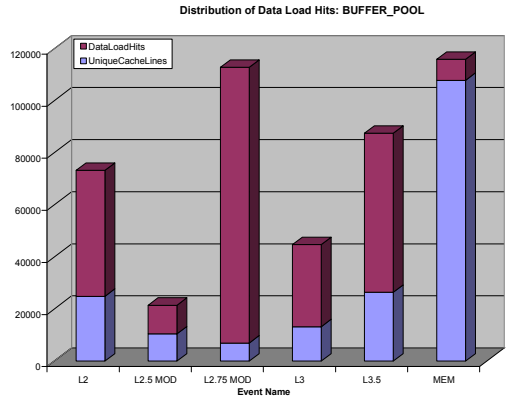

Figure 3. Distribution of Buffer Pool data-load hits across 32-way p690 memory hierarchy

As shown in Figure 3, for the 32-way p690, approximately 93% of the L2-cache Buffer Pool data-load misses that are serviced by main memory (MEM hit events) reference data that maps to unique cache lines. (For the eight-way p690, the percentage is approximately 86%.) The large overlap of the MEM hit event's unique_cache_line bar with its data_load_hits bar indicates that the majority of L2-cache Buffer Pool data-load misses serviced by main memory are compulsory misses (misses associated with first-time data accesses). This is corroborated by analyses of the timestamps associated with these events. If this was not the case, i.e., if a large number of the data-load hits serviced by main memory were due to premature evictions from higher levels of the memory hierarchy (due to capacity or conflict misses), there would be a smaller overlap, indicating multiple data-load hits per cache line. This memory access behavior indicates efficient p690 main memory usage and represents a match between the architecture and application – ideally, each cache line would be accessed from main memory only once.

The data depicted in Figure 3 also shows that once Buffer Pool data is loaded from main memory, it migrates across and is accessed from all levels of the memory hierarchy. This distribution of data-load hits indicates that a processor, rather than exploiting the caches associated with its own MCM is forced to obtain this shared data from caches on other MCMs. This is illustrated by the fact that in the 32-way p690 approximately 44% of L2-cache Buffer Pool data-load misses are satisfied at the L2.75 and L3.5 levels of the memory hierarchy, while only 31% are satisfied at the L2, L2.5, and L3 levels.

Referring to Table 1, note that the penalty for an L2-cache miss satisfied at a remote L2 cache (L2.75 hit) is cheaper than one satisfied at the local L3 cache.

Ideally, all L2-cache data-load misses would be satisfied locally via L2.5 hits. In this case, we do not see this behavior. Instead, approximately 25% of Buffer Pool data references are satisfied by L2.75 hits, as opposed to only 5% being satisfied by L2.5 hits. The unique_cache_line bar for the L2.75 hit event is only a very small percentage of the corresponding data_load_hits bar, indicating good locality of reference w.r.t. the L2.75 level of the memory hierarchy and significant sharing of a relatively small number of cache lines. It appears that this sharing may have resulted in premature eviction of these lines from L2 caches. If the evictions were due to false sharing or process sharing that could be localized to an MCM, then this behavior would be considered a mismatch between the application and the architecture and would present a target for potential performance improvement.

In contrast, as depicted in Figure 4, the distribution of Buffer Pool data-load hits associated with the eight-way p690 shows that approximately 20% of references are satisfied by L2.5 hits, as opposed to 16% of references being satisfied by L2.75 hits. Furthermore, approximately 53% of L2-cache Buffer Pool data-load misses are serviced at the L2.5 and L3 levels of the memory hierarchy, while only 26% are serviced at the L2.75 and L3.5 levels. This distribution indicates that a processor in the eight-way p690 references data mainly from local L2 caches.

The difference in this memory behavior may be due to the fact that the eight-way p690 used in this study consists of "single core good" chips, i.e., chips with one functional processor, while the 32-way p690 is comprised of chips with two functional processors. As such, each MCM in the 32-processor system contains twice as many functional processors, and potentially twice as many processes, as the eight-processor system. Accordingly, in the 32-processor system, there is increased contention for both local and remote L2 caches. Intra-MCM L2-cache contention can cause premature eviction of cache lines from local L2 caches and can increase the need to access either the local L3 cache or remote L2 caches. Further analysis is necessary to validate whether or not this is the case.

5.3.2. Private Data. Private data, e.g., a process' return stack and local variables, can be accessed only by the process that owns them. Ideally, a process' private data remains as close as possible to the executing processor. In the case of the p690, this means that process-private data to be used in the future should be L1-cache resident. In addition, data-load hits to memory should occur only to satisfy compulsory misses. The only reason a process should access its private data from a remote cache is to re-load its working set as a result of process migration. However, in reality, contention between processes sharing an L2 cache can evict private data lines in a process' working set, forcing them to the L3 cache or memory.

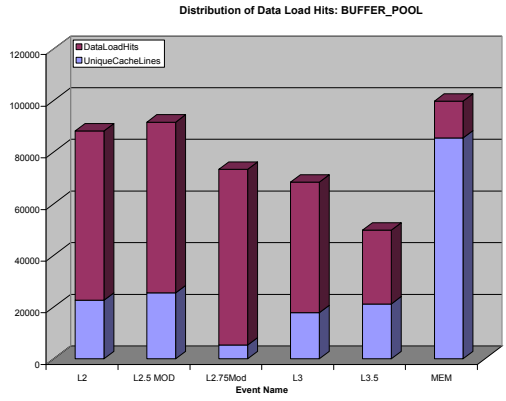

Figure 4. Distribution of Buffer Pool data-load hits across 8-way p690 memory hierarchy

The Data/BSS/Heap address region is a process-private region that contains data such as initialized and uninitialized variables referenced by a process, as well as the process' heap space. During the 10-minute observation of TPC-C executing on the 32-way p690, approximately 20% of sampled data-load hits target it. Figures 5 and 6 depict the distribution of data-load hits generated by L1-cache misses, to this region, arranged by levels of the eight- and 32-way p690 memory hierarchies, respectively.

As shown in Figures 5 and 6, approximately 79% (25%) of the L2-cache Data/BSS/Heap data-load misses generated by the eight-(32-)processor system that result in MEM hit events reference data that maps to unique cache lines. As is the case with the Buffer Pool, for Data/BSS/Heap, the unique_cache_line bar for the MEM hit event overlaps significantly with the associated data_load_hits bar, indicating that the majority of references satisfied by main memory are generated by compulsory misses. Unlike the Buffer Pool, for Data/BSS/Heap, once data is loaded from main memory, for the most part, it is accessed locally. This is illustrated by the fact that for TPC-C executed on the eight- (32-) way p690 approximately 73% (62%) of the L1-cache Data/BSS/Heap data-load misses are serviced by local caches, while less than 2% (1%) are serviced by remote caches. The remaining 25% (37%) of cache misses are satisfied by main memory. This indicates a good match between application and architecture.

The major difference between the distributions of Data/BSS/Heap data-load hits across the memory hierarchy of the eight- and 32-processor systems is the increased number of L2.5 events for the eight-way p690.

In fact, this is characteristic of all private address regions in the TPC-C address space. One possible explanation is that processes migrate more frequently in the eight-processor system. Again, the different number of processors/processes per MCM can account for this behavior. With fewer processors on which to execute, processes are forced to migrate more often in order to accomplish their work. And, on each migration a process is forced to reload its working set from the L2 cache of the processor on which it was previously executing. This is illustrated in Figure 6 by the L2.5 hit unique_cache_line bar being a fairly small percentage of its corresponding data-load hit bar, indicating repeated accesses to a small number of unique cache lines. Analyses of process migration, using the timestamp and processor and thread ids associated with every sample, corroborate these findings.

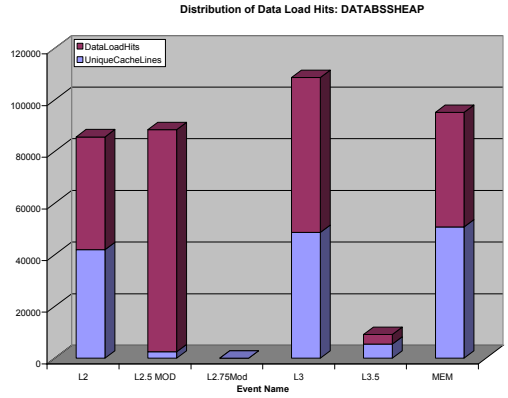

Figure 5. Distribution of Data/BSS/Heap data-load hits across 32-way p690 memory hierarchy

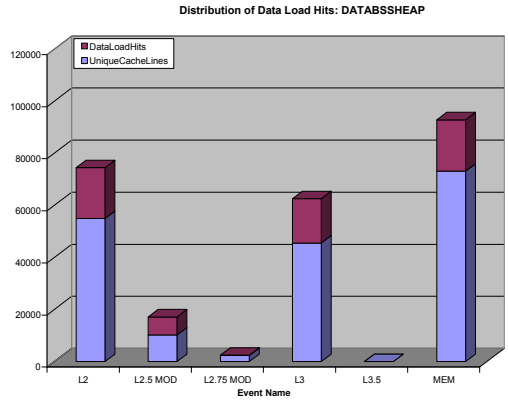

Figure 6. Distribution of Data/BSS/Heap data-load hits across 8-way p690 memory hierarchy

5.3.3. Summary. These results point out one similarity and two key differences between the memory access behavior of TPC-C private and shared data loads.

- In common is an application/architecture match: main memory accesses, the highest-penalty hit, primarily are associated with compulsory misses.
- In general, references to private data are serviced within an MCM—an application/architecture match, while references to shared data are satisfied outside an MCM. This makes shared data references, in general, more costly and identifies a potential source of performance degradation.
- Furthermore, as compared to shared data-load hits, private data-load hits have decreased locality of reference with respect to the memory hierarchy, i.e., references to shared data are much more localized. With respect to shared data, this identifies a target for further analysis to determine if processes that access common cache lines can be scheduled on the same MCM.

Some of the differences observed between the eight- and 32-processor data may be attributable to the fact that the 32-processsor event traces capture a subset of the sampled events. For example, the 32-processor L3 hit event traces capture the sampled events from 17 of the 32 processors, while the eight-processor traces capture the sampled events from all eight processors. This, however, does not affect the insights gained from the memory access behavior of the individual systems and the demonstration of the power and flexibility of our multiprocessor performance evaluation framework, which, as shown, can be used to characterize and evaluate application/architecture memory access behavior and performance.

6. Summary, Conclusions, and Future Work

This paper describes a powerful, fast, and flexible performance evaluation framework to study the behavior of large, complex applications executed on multiprocessor systems. Sampled event traces are captured in real time via on-chip performance counters and stored in databases via Java tools. Java report-generation software facilitates the analysis of the event traces by querying the database and processing the results. This methodology was used to study L1- and L2-cache data-load misses generated by the TPC-C benchmark executed on eight- and 32-way IBM ~pSeries 690 systems. The framework facilitates the characterization of the locality of reference exhibited by TPC-C data loads at the various levels of the memory hierarchy and the evaluation of the efficacy of design aspects of and policies associated with the p690 memory hierarchy with respect to workload demands.

The study demonstrates good matches between design aspects of the architecture and the workload's memory access behavior. For example, main memory accesses primarily are associated with compulsory misses and memory references to private data are generally

satisfied within the MCM, while references to shared data are satisfied outside the MCM.

Future work will use other insights presented in this paper, e.g., observations that indicate contention between private and shared data at L1 and L2 caches, and our performance evaluation framework, to explore ways to improve performance, either through modifications to the application, the operating system, the hardware, or a combination of these.

In addition, future work will study the degree to which sampled event traces represent actual dynamic application behavior. And, it will track potential TPC-C performance impediments to actual code and/or data structures in the application or operating system, and study related application, operating system, and/or hardware modifications that may improve performance. Improvements to the performance evaluation framework will continue; in particular, user-friendly graphical user interfaces (GUIs) and automatic graph creation will be added.

Acknowledgements

We thank Robert Acosta, Robert Amezcua, Carole Gottlieb, Cathy Nunez, and Bret Olszewski, IBM-Austin, for their help in defining a research area of mutual interest and establishing a research partnership that has proven to be very effective. Also, we thank The Austin Center for Advanced Studies and Carole Gottlieb for the faculty research award and IBM/NPSC Ph.D. Fellowship that made this research possible; Cathy Nunez for arranging Trevor Morgan's summer 2002 internship, which kicked-off this research; and Carole Gottlieb for arranging Diana Villa's summer 2003 internship.

References

[1] J. Anderson, L. Berg, et al., "Continuous Profiling: Where have all the cycles gone?," *ACM Transactions on Computer Systems*, **15**:4, Nov. 1997, pp. 357-390.

[2] L. Barroso, K. Gharachorloo, and E. Bugnion., "Memory System Characterization of Commercial Workloads," *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998, pp. 3-14.

[3] R. Desikan, D. Burger, and S. Keckler, "Measuring Experimental Error in Microprocessor Simulation," *Proceedings of the 28th Annual International Symposium on Computer Architecture*, July 2001, pp. 266-277.

[4] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufman, CA, 1996.

[5] K. Keeton, R. Clapp, and A. Nanda, "Evaluating Servers with Commercial Workloads," *Computer*, **36**:2, Feb. 2003, pp. 29-32.

[6] K. Keeton, D. Patterson, et al., "Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads," *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998, pp. 15-26.

[7] M. Itzkowitz, B. Wylie, et al., "Memory Profiling Using Hardware Counters," CD *Proceedings of SC 2003*, Nov. 2004**.**

[8] S. Leutenegger and D. Dias, "A Modeling Study of the TPC-C Benchmark," *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, June 1993, pp. 22-31.

[9] T. Morgan, D. Villa, et al., "L2 Miss Profiling on the p690 for a Large-scale Database Application," *Proceedings of the 4th Annual IBM Austin Center for Advanced Studies Conference*, Feb. 2003.

[10] A. Nanda, K. Mak, et al., "MemorIES: a Programmable, Real-time Hardware Emulation Tool for Multiprocessor Server Design," *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000, pp. 37-48.

[11] M. Rosenblum, S. Herrod, et al., "Complete Computer Simulation: The SimOS Approach," *IEEE Parallel and Distributed Technology: Systems and Applications,* Winter 1995, pp. 34-43.

[12] T-F Tsuei, A. Packer, and K-T Ko, "Database Buffer Size Investigation for OLTP Workloads," *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, June 1997, pp. 112-122.

[13] D. Villa, J. Acosta, et al., "Memory Performance Profiling via Sampled Performance Monitor Event Traces," *Proceedings of the 5th Annual IBM Austin Center for Advanced Studies Conference*, Feb. 2004.

[14] W. Wulf and S. McKee. "Hitting the Memory Wall: Implications of the Obvious," *Computer Architecture News*, **23**:1, March 1995, pp. 20-24.

[15] The POWER4 Processor Introduction and Tuning Guide, IBM, ibm.com/redbooks.

[16] TPC Benchmark C Standard Specification Revision 3.0, Transaction Processing Performance Council, Feb. 15, 1995.

[17]http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4_4.html#hier