Mining Performance Data from Sampled Event Traces

Ricardo Portillo, Diana Villa, and Patricia J. Teller
University of Texas at El Paso, Department of Computer Science
{raportil,pteller}@cs.utep.edu; demarquez@utep.edu

Bret Olszewski
IBM Corporation
breto@us.ibm.com

Abstract

The prominent role of the memory hierarchy as one of the major bottlenecks in achieving good program performance has motivated the search for ways of capturing the memory performance of an application/machine pair that is both practical in terms of time and space, yet detailed enough to gain useful and relevant information. The strategy that we endorse periodically samples events during program execution, producing an event trace that is both manageable and informative. As demonstrated, adopting this strategy, a diverse set of performance issues can be studied using the same set of traces. For example, using one set of traces and our performance evaluation framework, memory access performance, process migration, compulsory and conflict misses, and false sharing can be characterized.

1. Introduction

As the gap between the speeds of processors and Dynamic Random Access Memory (DRAM) continues to increase [4], the memory subsystem continues to be a major computer design consideration. For many computer systems, in particular future, symmetric multiprocessor (SMP) systems, the performance of the memory subsystem governs that of the system as a whole [15].

Although the performance evaluation of SMP systems, in particular their memory subsystems, is of great import, it is becoming increasingly more challenging. For example, although address traces generated by the hardware or by software architecture simulation, as is the case with SimOS [10], have been used to analyze cache performance, 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, system simulation of large, complex workloads executed on multiprocessors requires as much disk space as the workload, typically multiple terabytes by today's standards, and usually more memory. In addition, the effort to simulate a large multiprocessor system is intimidating. An alternative is a cache simulator built in hardware and connected to a running system [9] or, as we suggest, sampled performance monitor event traces, i.e., sampled event traces.

Sampled event traces are collected in real time via on-chip performance counters that recognize events of interest. Most state-of-the-art processors have on-chip performance counters. For example, the Performance Monitor Unit (PMU) of the Power4 has eight that support the capture of over 200 events [6]. Supported events include, among others, different types of cache misses, TLB misses, memory operations, and fixed and floating-point operations.

Event traces can provide information that can be used to understand when the workload and architecture complement one another in terms of performance and to identify performance bottlenecks. This information can be attained by mining the traces and analyzing the resultant data, both of which are facilitated by our performance evaluation framework [14], described briefly herein. As demonstrated in this paper, one set of sampled event traces for a large, complex application, i.e., the TPC-C benchmark, executed on eight- and 32-processor IBM eServer pSeries 690 systems, can be used to study, among other things, the performance of the memory hierarchy, to analyze the performance implications of process migration and L2-cache sharing, and to identify false sharing. This paper presents insights regarding the performance issues mentioned above and, more importantly, demonstrates

that a large amount of information can be gleaned from one set of small and manageable sampled event traces.

The remainder of the paper is organized as follows: Section 2 presents related research. Section 3 describes the subject event traces, including the platform from which the data was collected, compute platform, workload under study, events of interest, and tools used to collect the data. Section 4 targets data analysis, describing the tools and methodology used, and presents a sample of our results. Finally, Section 5 presents a summary, conclusions, and future work.

2. Related Research

Several researchers have used event traces to characterize application behavior. For example, to explore the performance effects of architectural modifications, Barroso, et al. [2] use tools such as IPROBE and DCPI (Digital Continuous Profiling Infrastructure) [1,3] to capture event traces of applications executed on a four-processor AlphaServer 4100 using Oracle 7.3.2 and Keeton, et al. [8] use performance monitors on a four-processor Pentium Pro-based server. In [2] workload characterization, accomplished by source code instrumentation, is coupled with simulation methodologies; in [8] the hardware was physically modified. Desikan, et al. [3], like Barroso, et al., use the DCPI tool to check the reliability of an Alpha 21264 simulator by sampling events that are used to derive performance measurements for the Compaq DS-10L workstation.

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, which can be gathered either by clock or hardware-counter profiling, provides per-instruction details of memory accesses in the annotated disassembly and data aggregated and sorted by object structure types and elements. Compiler-generated padding introduces minor inaccuracies and collection perturbation can be controlled through configuration of the processors' counter overflow rates. Future work described by Itzkowitz, et al., i.e., analysis of event data addresses by machine entity, e.g., memory segment, page, etc., is presented in [13], in which the analysis is facilitated by our performance evaluation framework.

Our work is differentiated from the related research described above by the scale of the systems and the methodology used. We analyze performance data obtained from both eight- and 32-processor systems, and 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. In addition, as described in Section 4, our methodology does not require source code instrumentation and it is not restricted to memory access behavior analysis.

3. Studied Sampled Event Traces

The sampled event traces studied in this paper were generated via the Performance Monitoring Units of POWER4 microprocessors of eight- and 32-processor IBM eServer pSeries 690s (p690s) executing TPC-C (Transaction Processing Performance Council). The PMUs were programmed to monitor events triggered by L2-cache data-load misses. Below we describe the compute platform, workload, monitored events, and methodology used to collect the sampled traces.

3.1. Compute Platform

The eServer pSeries 690 family of SMP architectures includes the eight- and 32-processor configurations used in this study [5], which run AIX version 5.2. The MultiChip Module (MCM), the building block of the architecture, contains four chips, each of which is comprised of two 1.3 GHz POWER4 processors. Usually an eight-processor system contains one MCM but the one used in this study contains two, each with four "single core good" chips, i.e., only one functional processor per chip.

For the configurations under study,

- each CPU has a 64KB L1 instruction cache (Icache) and a 32KB L1 data cache (Dcache) with 128B lines each;
- each chip has a 1.44MB L2 unified cache, with 128B lines;
- the four chips on an MCM share a 128MB L3 unified cache with 512B lines; and
- main memory is 128GB (256GB) for the eight(32)- processor p690.

Data private to and shared by processes are managed by a cache coherence protocol. As illustrated in Figure 1, 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. local, intra-MCM L2 cache, the *L2.5 level*;
2. remote, inter-MCM L2 cache, the *L2.75 level;*
3. local intra-MCM L3 cache, the *L3 level*;
4. remote inter-MCM L3 cache, the *L3.5 level*; and
5. main memory.

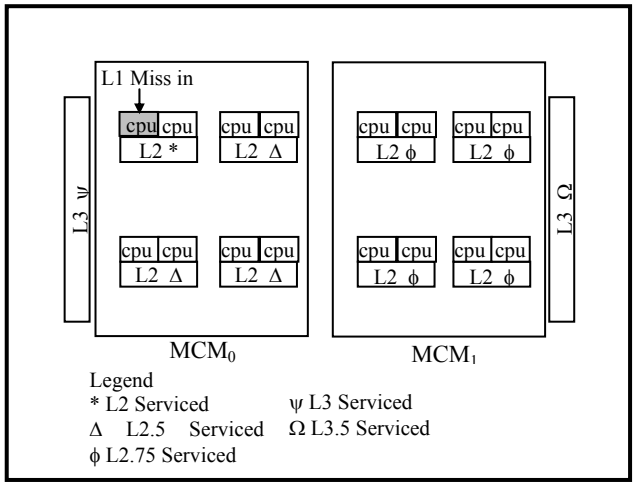

Figure 1: Two (double-core good) MCMs of the 32-processor p690

The segment and page sizes are 256MB and 4KB, respectively.

3.2. Workload

A fully implemented TPC-C benchmark drives a commercially available relational database compiled by the IBM C for AIX version 5 compiler. TPC-C is a well-known benchmark [11] that emulates read-only and update-intensive 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.

Based on a process model, TPC-C allows for per-process, private data address regions and a shared data address region that contains database state information and the database buffer pool, which 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.

3.3. Monitored Events

The sampled event traces used in this study are generated by the occurrence of events related to L2-cache data-load misses. As explained in [14], these events were chosen because, for the experimental platform and workload, L2-cache misses have a much higher performance impact than events such as instruction cache misses, translation-lookaside buffer misses, address-only coherence operations, and uncached memory accesses for I/O. In addition, as illustrated in Figure 2, from a CPI viewpoint, TPC-C's most dominant memory subsystem operation is L2-cache miss service. Figure 2 indicates where in the memory hierarchy level-one (L1) cache misses are satisfied. L1 data-cache misses that miss in the L2 cache are satisfied by either a level-three cache (L3 D hit) or main memory (Memory D hit). As described below, these are high-penalty misses, which if decreased, could have a positive impact on performance.

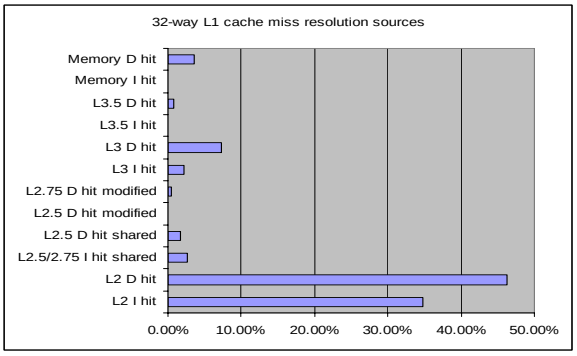

Figure 2: TPC-C instruction and data access sites for the 32-processor p690

The events related to an L2-cache data-load miss are classified according to the levels at which they are serviced and the state of the referenced cache lines. An L2-cache data-load miss serviced by the L2.5 level (at a cost of approximately 73 vs. 12 cycles for an L2 hit), generates either an L2.5-shared hit (L25_SHR) or L2.5-modified (L25_MOD) hit event. If the cache line is in the shared state, it resides simultaneously in more than one L2 cache. In this case, if one of the caches is a local, intra-MCM cache, it services the miss and an L25_SHR event occurs. If the line is in the modified state, it is exclusively owned by one L2 cache and contains more recent data than is in the backing physical memory. In this case, if it is a local, intra-MCM cache, an L25_MOD event occurs.

Similarly, L2-cache misses serviced by the L2.75 level (at a cost of approximately 96 cycles) generate either an L2.75-shared hit (L275_SHR) or L2.75-

modified hit (L275_MOD) event. In both cases the referenced cache line, whether in the shared or modified state, does not reside in any of the intra-MCM L2 caches and, thus, is serviced by a remote L2 cache.

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). For both L3-shared and L3.5-shared hit events, the requested cache line resides in more than one L3 cache. In the former case, the line resides in the local L3 cache, which services the miss (at a cost of approximately 112 cycles). In the latter case, the line does not reside in the local L3 cache and, thus, is serviced by a remote L3 cache (at a cost of approximately 143 cycles). For both the L3-modified and L3.5-modified hit events, the line resides in only one L3 cache. In the former case, it is the local L3 cache; otherwise, it is a remote L3 cache.

For this study we consider all of these events, in addition to main memory hit events (MEM); L2 cache misses serviced by memory cost approximately 320 cycles. 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).

3.4. Methodology

Valuable performance information for a section of code or an entire program can be collected, through the use of the POWER4 performance-monitoring facilities and tools, in the form of aggregate counts of and, for selected models of pSeries hardware, sampled traces of information associated with user-specified events. Up to eight events can be monitored concurrently by the eight PMU counters. The state of the counters is controlled by special-purpose registers that are accessible only via the operating system, through a programming interface that accesses them through a kernel extension. This interface permits, among other things, the specification of the events to be monitored, the execution points at which to start and stop counters, and the points at which software is to retrieve results.

Three tools were used to collect the sampled performance monitor event traces used in this paper:

1. *tprof*, a time-based profiling tool that is part of the AIX operating system,
2. *eprof*, 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, and
3. *trcrpt*, a utility that formats the file of events collected by an AIX trace and creates a time-stamped text file of events.

eprof is used to program the PMU to sample hardware countable events at a defined rate, the default, which is used in this study, being approximately 100 events per second per processor. When an event is sampled, the instruction address and data address (if applicable) are captured by the PMU and a PM interrupt is delivered. The interrupt causes the sample information to be extracted from the PMU and appended, as a trace record, to the AIX trace, along with an AIX trace hook, which describes the record. Using the AIX trace, samples can be written to disk or collected and summarized by a daemon. In order to correlate sampled events with processes/threads, the profiling also enables selected AIX trace hooks, such as those related to dispatching.

Some sampled events occur at a fixed rate, e.g., processor cycles; others, e.g., cache misses, occur at a variable rate. Time-based sampling is appropriate for the former, a sample is collected every 10 milliseconds. For the latter, if the rate of event occurrence is greater than the default, 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. Despite the adoption of the default sampling rate and a 10-minute workload, as shown in Table 1, this is the case for the studied events. 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.

Table 1: Event sample counts

Event	Sample Count	
	8-processor	**32-processor**
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

4. Data Analysis

Through the use of a set of tools implemented in Java, each sampled event is processed and stored in a MySQL database according to the workload being monitored, the number of processors used to execute the workload, and the event being sampled, e.g., 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 tools, also implemented in Java, is used to query the databases and process the results, producing as output default and customized reports in the form of formatted text files, which are transformed into graphs via a spreadsheet application with built-in graphing capabilities.

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

4.1. Identifying Concentrated Areas of Locality of Reference

The goal of identifying concentrated areas of locality of reference is to pinpoint application-specific sources of memory performance degradation associated with data references, and to identify application, operating system, or architectural changes that could enhance performance. The analysis is done using a process of refinement. First, L2-cache data load misses are studied with respect to their resolution sites across the p690 memory hierarchy. Analysis of the traces of the monitored events, via queries to the associated databases that count the number of hits to the different levels of the memory hierarchy, shows that the majority of L2-cache data-load misses are resolved in L3 caches and main memory, both of which carry high load-hit latencies when compared to L2-cache load latencies.

Next the analysis focuses on these heavily hit resolution sites, querying the databases containing the L3, L35, and MEM event traces to identify concentrated areas of locality of reference within regions of the application address space. These queries report the number of hits to all regions, number of hits to each region, and number of unique cache lines referenced in each region (*Unique_cache_line* bar in Figure 3). Computing the percentage of hits to each region (*Hit_%* bar of Figure 3) and comparing it to the number of unique cache lines referenced in the region gives a measure of locality of reference. For TPC-C, this shows that the Buffer Pool and Data,BSS,Heap regions dominate as the effective address regions for data loads satisfied by local L3 caches (see Figure 3) and main memory. In addition, as shown in Figure 3, the analyses indicate that even though the Data,BSS,Heap region contains a higher *Hit_%* bar, i.e., accounts for a higher number of data references to local L3 caches, the Buffer Pool contains a smaller *Unique_cache_line* bar, i.e., data loads to the Buffer Pool reference a smaller set of unique cache lines and, therefore, exhibits greater locality of reference.

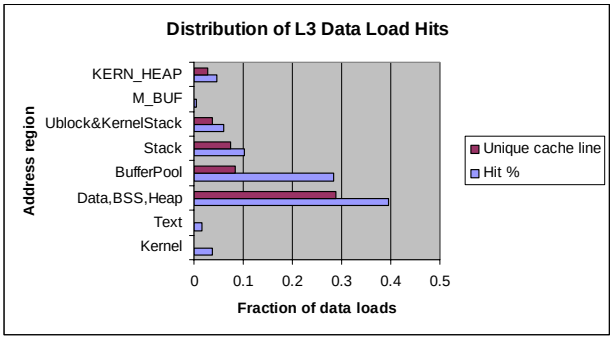

Figure 3: Distribution of TPC-C local L3-cache data-load hits across address regions of the 32-processor p690

Given this information, analysis of L3-cache and memory data-load hits to the Buffer Pool is refined to study the most heavily referenced segments, pages, and cache lines in this region. Querying the L3 database, we find that of the 570 unique Buffer Pool segments that are touched via local L3 caches, references to only six of the segments accounts for over 90% of the Buffer Pool activity in the L3 caches. Further querying of the database indicates that this concentrated locality of reference exists down to the cache line level, i.e., the L2-cache data load misses resolved in local L3 caches generally reference the same data within the Buffer Pool region of the TPC-C address space.

In [13] the analysis continues to endeavor to pinpoint instructions and data structures that are sources of performance degradation. Although the reported analysis falls short of identifying application sources of performance degradation, this analysis prompted modifications to the operating system's

(AIX's) management of the Buffer Pool region, which yielded observable performance improvements.

4.2. Characterizing Memory Behavior

The same set of sampled event traces can be used to study the effectiveness of design aspects and policies associated with the memory hierarchy with respect to workload demands. This is demonstrated in [14], which describes an analysis (via database queries) that characterizes the behavior of TPC-C shared and private data loads with respect to the p690 memory hierarchy. Since shared data, e.g., global variables and application code, are accessible by every TPC-C and database process in the system, for better performance this data should remain in the higher levels of the p690 memory hierarchy until it is no longer referenced. On the other hand, private data, e.g., a process's return stack and local variables, can be accessed only by the process that owns them and, therefore, for performance reasons should remain as close as possible to the executing processor.

The analysis focuses on the Buffer Pool and Data,BSS,Heap regions of the TPC-C address space because queries to the event trace databases indicate that the Buffer Pool is the most frequently accessed shared address region, Data,BSS,Heap is the most frequently accessed private region, and together these regions are referenced by 90% of the L2-cache data-load misses. Given the regions of interest, the next step in the analysis computes *Data_Loads_Hits* bars and *Unique_Cache_Line* bars for each address region across the levels of the p690 memory hierarchy. The distribution of the Buffer Pool (shared) data-load hits is depicted in Figure 4.

This analysis identifies one of the key differences between TPC-C shared and private data loads: in general, shared data references are more costly since they are satisfied outside an MCM, while private data references are satisfied within an MCM. This and the fact that shared data-load hits display a more concentrated locality of reference than references to private data presents a possible source of performance degradation. Concentrated locality of reference is best illustrated by the L2.75 hit event in Figure 4. Note that this event has a Unique_cache_line bar that is only a very small portion of its Data_load_hits bar. This indicates significant sharing of a relatively small number of unique cache lines. This behavior may have resulted in premature L2-cache line evictions and could indicate a mismatch between the application and the architecture if the evictions were due to false sharing or process sharing that could be localized to an MCM.

In contrast, an application/architecture match that is common to both shared and private data is demonstrated by TPC-C main memory accesses. Although accesses to main memory carry the highest penalty, the data shows that they are primarily associated with compulsory misses, i.e., the majority of references are to unique data. In Figure 4, this is illustrated by the MEM bar having a Unique_cache_line bar that is a very large portion of its corresponding Data_load_hits bar.

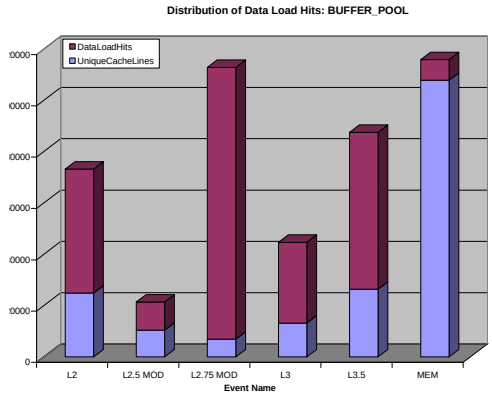

Figure 4: Distribution of Buffer Pool data-load hits across 32-processor p690 memory hierarchy

This type of analysis identifies both good and possibly bad matches between design aspects of the architecture and the workload's memory access behavior, and presents targets for potential performance improvement via modifications to the application, the operating system, the architecture, or a combination of these.

4.3. Compulsory Misses

Potential sources of performance degradation, such as L2-cache misses that are satisfied by main memory, are targets of further analysis. In the p690, when main memory is accessed to satisfy an L2-cache miss, the referenced block is placed in the L1 and L2 caches of the processor that generated the miss as well as in its MCM's L3 cache. The first miss to the referenced block is compulsory, i.e., it cannot be avoided. Accordingly, our assumption is that the first recorded L2-cache miss on a block that hits in main memory is compulsory. This is an assumption because we are using sampled event traces that do not contain all L2-cache miss events. But, since analysis of the traces indicates 80% of the sampled L2-cache miss events hit in memory only once, it is highly likely that the first hit in main memory is, indeed, a compulsory miss. Thus, at least with these sampled event traces, compulsory misses can be identified with a fairly high degree of probability.

Using the above-stated assumption, 80% (93%) of the recorded main memory hits of the eight(32)-processor p690 are due to L2-cache compulsory misses. This indicates that the majority of the main memory accesses on both systems are compulsory, which means that the application and architecture are well matched--the system is maintaining most of the workload in the upper levels of the memory hierarchy.

4.4. Process Migration

Since a sampled event trace record includes when and on which processor an event occurred, as well as the address of the requested data reference, intra- and inter-MCM process migration can be identified easily and, as described below, its overhead can be computed. This is of interest because process migration from one chip to another can degrade performance when all or part of the process' working set must follow, via L2-cache misses, the migrating process.

In terms of the p690, intra- and inter-MCM process migration, overhead can be identified in the following way. For intra-MCM process migration, the associated data migration overhead is quantified by the L2.5 hit events that occur after the time of process migration. For inter-MCM process migration, it is the L2.75 and L3.5 hit events that occur after process migration that quantify the overhead. In this paper we focus on intra-MCM migration effects on memory performance, but our methodology can be applied in the same way to quantify inter-MCM migration effects.

One way that sampled event traces can be used to observe if intra-MCM process migration is a source of memory performance degradation is to correlate, for each thread, the number of intra-MCM migrations and the number of L2.5 hit events. This is based on two assumptions: (1) the more a thread migrates the more it has to access the L2.5 level of the memory hierarchy to migrate its workload and (2) the workload for each thread is more or less static, i.e., the same addresses are referenced throughout the run.

Using the 32-processor data and concentrating on intra-MCM migration, 885 unique threads are identified. For each of these 885 threads, the number of intra-MCM migrations and L2.5 hit events are computed. The next step in the analysis is to correlate these two sets of numbers, thus, identifying the threads that are potential sources of performance degradation. Figure 5 shows the correlation of intra-MCM migrations with only L2.5 modified cache hits (i.e., L2.5 cache hits where the data resides in only one cache); the sampled event traces for L2.5 shared cache hits were not collected.

Figure 5 shows that the eight threads in the circled cluster are the only ones that show any correlation between number of migrations between chips on an MCM and the number of L2.5 modified cache hits (L2.5mod hits). The other threads, in the lower left corner of the graph, either migrate and do not generate L2.5mod hits or do not migrate within their MCMs. This observation, itself, is very useful since it allows 877 of the 885 threads to be ignored and the analysis narrowed to the cluster of eight threads.

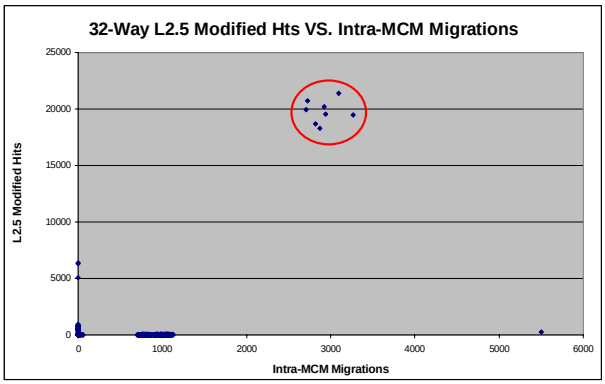

Figure 5: Correlation between L2.5 modified cache hits and intra-MCM migrations

A caveat to this correlation is that a very large percentage of the data references associated with the L2.5mod hits hit only once at the L2.5 level of the memory hierarchy. Thus, if a process migrates a lot and generates a large number of L2.5mod hits, it is possible that the associated accesses to the L2.5 level of the memory hierarchy are not associated with process migration. It may mean that a large portion of the "first time hits" are caused by *single specific* migrations and do not correlate with the total number of migrations. Therefore, for thoroughness, the analysis is redone without the L2.5mod hits that occur only once. Although this does not completely filter out all data references that may not have a correlation with the total number of migrations (e.g., data references that generate two L2.5mod hits) and since a very large percentage of L2.5mod hits are "first time hits," this may be sufficient to show whether the above-mentioned caveat affects the correlation analysis. Although not included due to space restrictions, the correlation analysis and associated graph are essentially the same, i.e., the same pattern is observed.

Since it may be coincidental that the eight threads clustered in Figure 5 have high values for both metrics, the sampled event traces are used to analyze the data for each of the threads to understand how migration affects L2.5mod hits across time. Visualizing the threads in this manner allows us to see if the frequency of L2.5mod hits increases directly after intra-MCM migration. This type of behavior would imply that

migration is, in fact, playing a role in the remote cache access activity and, therefore, is causing memory performance degradation. Note that this analysis would have been cumbersome without the correlation analysis presented above, which allowed this analysis to focus on eight, rather than 885, threads.

As can be seen in Figure 6, all eight threads, at around 50 seconds into the trace, generate L2.5mod hits only when they reside on specific processors. For example, although *thread 3* migrates constantly to several processors throughout the run, as do all the other threads, it only hits in L2.5mod when executing on processor 5. This indicates that, in general, migration does not affect L2.5mod hits. However, the demonstrated behavior may imply that these threads should not be scheduled on specific processors. This could reduce L2.5 accesses and, consequently, may reduce L2 misses.

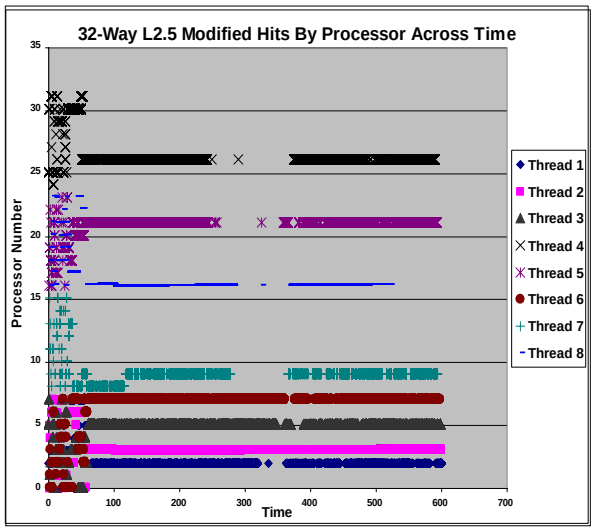

Figure 6: 32-processor L2.5 modified cache hits by processor across time

4.5. Resource Contention

Contention for available resources, such as memory, is another potential source of performance degradation. However, in large-scale systems in particular, it is often difficult to efficiently identify instances of resource contention and quantify their impact on performance.

Sampled event traces can facilitate this process. For example, in this section we show how they can be used to identify contention in L2-caches shared by pairs of processors, an architectural characteristic of the 32-processor system used in this study, and false sharing. These events are of interest since contention for cache resources increases the number of conflict misses and may cause thrashing. False sharing, which also can result in an increase in cache misses and thrashing, occurs when two processors share a cache line but access different data words within the line.

The cache configuration and data addresses that are captured in the sampled traces permit mapping to L2-cache sets, lines, and words within lines. The caveat, however, is that the L2 cache is physically addressed and the data addresses are virtual addresses. Thus, in general, the investigation of L2-cache contention and false sharing requires dynamic load map information, which was not captured during data collection. However, since the (database) Buffer Pool as well as the Kernel, Text, and Shared Library regions are shared address spaces, and threads access them with the same virtual addresses, these performance issues can be explored in a limited fashion and, in doing so, demonstrate the methodology.

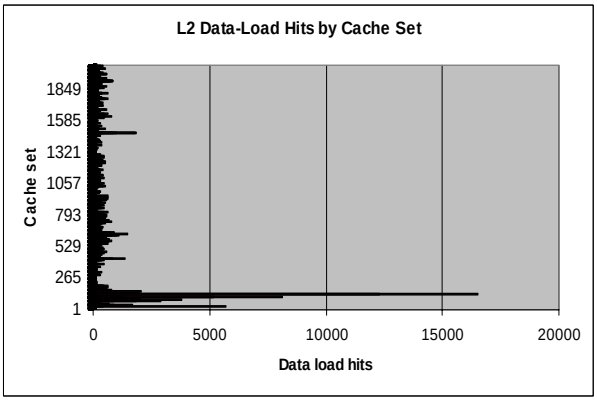

Figure 7: Distribution of TPC-C L2-cache data-load hits across L2-cache sets

The analysis begins with cache sets and is refined to examine lines and then the words within lines. First, as depicted in Figure 7, the distribution of Buffer Pool data-load hits across the 2048 sets in the L2-cache is visualized. As can be seen, the more frequently accessed sets, e.g., those in the 27-124 range, stand out and, thus, are the target of further analysis.

In an effort to identify L2-cache line contention, as depicted in Figure 8, for each of these "hot" cache lines, we study the access pattern of co-resident processor pairs, which share an L2 cache, over time. Although the sampled event traces correspond to a 10-minute observation interval of the execution of TPC-C, for readability the pictured data represents a 100-second snapshot. The data suggests that there are alternating intervals of references to this "hot" cache line by each of the chip co-resident processors. Thus, it appears that this processor pair is contending for this cache line.

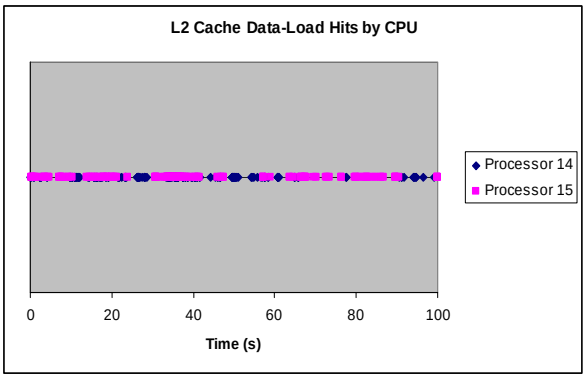

Figure 8: References to a "hot" cache line by a chip co-resident processor pair

In an effort to identify L2-cache line contention, as depicted in Figure 8, for each of these "hot" cache lines, we study the access pattern of co-resident processor pairs, which share an L2 cache, over time. Although the sampled event traces correspond to a 10-minute observation interval of the execution of TPC-C, for readability the pictured data represents a 100-second snapshot. The data suggests that there are alternating intervals of references to this "hot" cache line by each of the chip co-resident processors. Thus, it appears that this processor pair is contending for this cache line.

To understand if false sharing is occurring, the traces can be used to study the references to words within the cache line. Figure 9 is the product of such a study–it depicts the accesses to words within the "hot" L2-cache line by the chip co-resident processor pair for the same 100-second interval. As shown, each processor uses the cache line to access different data; this supports the hypothesis that false sharing exists between these two processors.

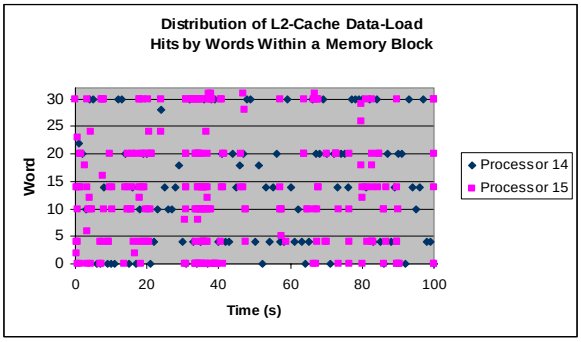

Figure 9: References to individual data words within a "hot" cache line by a chip co-resident processor pair

5. Conclusions and Future Work

This paper shows the potential of small and manageable sampled event traces by using them to analyze, with the help of our performance evaluation framework, a disparate set of performance issues for a large, complex application running on a multiprocessor system. It should be noted that these performance issues were investigated using one set of sampled event traces.

Future work will study the degree to which sampled event traces represent the actual dynamic application behavior and will use sampled event traces to explore performance issues related to other applications. In addition, the framework will continue to be enhanced; in particular, user-friendly graphical user interfaces (GUIs) and automatic graph creation will be added.

6. Acknowledgments

We wish to 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 would like to thank them along with The Austin Center for Advanced Studies (ACAS) for the faculty research awards, Ph.D. Fellowship, and summer internships that made this research possible. Finally, we thank the National Science Foundation for support of Ricardo Portillo under grant no. EIA-0080940.

7. References

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

[2] Barroso, L., Gharachorloo, K., & Bugnion, E. (1998). Memory System Characterization of Commercial Workloads *Proc 25th Intl. Symp. on Comp. Arch.* (pp. 3-14).

[3] Desikan, R., Burger, D., & Keckler, S. (2001). Measuring Experimental Error in Microprocessor Simulation *Proc. 28th Ann. Intl. Symp. on Comp. Arch.* (pp. 266-277).

[4] Hennessy, J., &, Patterson, D. (1996). *Computer Architecture: A Quantitative Approach.* California: Morgan Kaufman.

[5] IBM Corp. (Oct. 2001). POWER4 System Architecture http://www1.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html

[6] IBM Corp. (Nov. 2001). The POWER4 Processor Introduction and Tuning Guide http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg247041.pdf

[7] Itzkowitz, M., Wylie, B., et al. (2004). Memory Profiling Using Hardware Counters *CD Proc. SC 2003*.

[8] Keeton, K., Patterson, D., et al. (1998). Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads *Proc.25th Ann. Intl. Symp. on Comp. Arch.* (pp. 15-26).

[9] Nanda, A., Mak, K., et al. (2000). MemorIES: a Programmable, Real-time Hardware Emulation Tool for Multiprocessor Server Design *Proc. 9th Intl. Conf. on Architectural Support for Programming Languages and O. S.* (pp. 37-48).

[10] Rosenblum, M., Herrod, S., et al. (1995). Complete Computer Simulation: The SimOS Approach *IEEE Parallel and Distr. Tech.: Systems and Applications* (pp. 34-43).

[11] Transaction Processing Performance Council (1995). TPC Benchmark C Standard Specification Revision 3.0.

[12] Tsuei, T-F, Packer, A., & Ko, K-T (1997). Database Buffer Size Investigation for OLTP Workloads *Proc. 1997 ACM SIGMOD Intl. Conf. on Management of Data* (pp. 112-122).

[13] Villa, D., Acosta, J., Teller, P., Olszewski, B., & Morgan, T. (Feb. 2004). Memory Performance Profiling via Sampled Performance Monitor Event Traces *5th Ann. Austin CAS Conf.*

[14] Villa, D., Acosta, J., Teller, P., Olszewski, B., &, Morgan, T. (July 2004). A Framework for Profiling Multiprocessor Memory Performance *to appear in 10th Intl. Conf on Parallel and Distributed Systems*.

[15] Wulf, W., &, McKee, S. (1995). Hitting the Memory Wall: Implications of the Obvious *Computer Architecture News*, *23*(1), 20-24.