L2-Cache Miss Profiling on the p690 for a Large-scale Database Application

Trevor Morgan, Diana Villa, Patricia J. Teller, and Jaime Acosta
The University of Texas at El Paso
Department of Computer Science
trevormorgan31@hotmail.com, dvilla@utep.edu, pteller@cs.utep.edu, jacosta@cs.utep.edu

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

Abstract

This paper profiles L2-cache data-load misses generated by the TPC-C benchmark executed on 8- and 32-way configurations of the IBM eserver pSeries 690 (p690). Using sampled performance monitor event traces, the resolution sites of L2-cache data-load misses are identified. To determine ways to enhance performance, the heavily hit resolution sites, L3 caches and main memory, are studied with respect to associated memory regions, segments, pages, cache blocks, routines, and instructions. Collected data indicates that the related data-load hits have high concentration within regions of the address space, segments, and pages. Specifically, our results show that the buffer pool and heap regions of the TPC-C address space tend to dominate as the effective address regions for data loads satisfied in L3 caches and main memory. Furthermore, for those data loads satisfied in L3 caches, the segments, pages, and cache blocks that constitute the buffer pool exhibit a rather dense distribution. Future work will continue the analysis in an effort to define ways to remedy the performance degradation associated with L2-cache data-load misses being serviced at high-penalty levels of the p690 memory hierarchy.

1. Motivation

The research reported in this paper represents a first step in addressing the following question: "As processors get faster and memories get larger, can we generate the address traces and/or memory-hierarchy miss rate information that is needed to permit us to study how to optimize memory subsystem performance?" To answer this question, we are endeavoring to design and develop a general method that, given a workload, can be used to generate a characterization, i.e., a model, of the workload in terms of its memory access parameters. Such a workload model could be used to either generate traces or directly model cache miss rates. The model would allow a parametric exploration of the system and workload design spaces.

The initial workload under study is that of TPC-C, a transaction-processing application that is understood fairly well and is representative of workloads of interest to IBM customers. The initial data access streams under study are those produced by level-two cache (L2-cache) misses for data loads generated by the TPC-C benchmark executed on an 8-way and a 32-way IBM eserver pSeries 690. Data analysis of information concerning these access streams indicates that there is opportunity for performance enhancement. In addition, it indicates that accesses to particular areas of the address space, e.g., working storage, the buffer pool, and components of the operating system, may be targets for this performance enhancement.

As described below, sampled event traces were used in this study. Historically, cache analysis is done using traces generated from hardware measurement or software architecture simulation, for example, SimOS [4]. As systems become faster and caches become much larger, it is very difficult to collect traces that are long enough to accurately model the memory hierarchy. In addition, for workloads like TPC-C, system simulation requires as much disk space as the workload (multiple terabytes today) and usually more memory. Also, the time to simulate a large n-way system is intimidating. An alternative to tracing is a cache simulator built in hardware and connected to a running system [3] or sampled event traces, the alternative that we adopted.

Using sampled traces from an 8-way configuration of the IBM eserver pSeries 690, also referred to in the remainder of this paper as the p690, we identify (1) the areas of the address space, down to a granularity of 128-byte cache lines (a.k.a. cache blocks), that are referenced repeatedly and generate L2-cache data-load misses that are resolved in high-penalty areas of the memory hierarchy and (2) the addresses of instructions that access these "hot" data areas. Analysis of the 8-way

and 32-way sampled event traces indicates that a fairly large number of L2-cache misses are resolved principally at the level-three (L3) caches and main memory of the p690, where the latencies are relatively high. The resolution of these misses at these high-penalty areas of the memory hierarchy does not seem intuitive for two reasons. First, the p690 architecture allows L2-cache misses generated by a processor to be serviced by any other L2 cache in the system. Since each processor has its own defined memory hierarchy, including a L2 cache that it physically shares with only one other (chip co-resident) processor, the fraction of accesses going to the L3 cache, or beyond, should be small. In addition, note that in the 32-way system there are 256GB of physical memory in use and 44.8MB of L2 cache. Second, it has been demonstrated that the data-load addresses for these L2-cache misses are not distributed uniformly throughout the address space, but rather tend to cluster in relatively small regions of the address space. Such clustering indicates locality of reference that, if exploited, should lead to hits in the upper-level caches.

Given the identification of the reasons for this behavior, program modifications can be made to alleviate resolution of L2-cache data-load misses at high-penalty areas of the memory hierarchy. The information gathered thus far suggests that these costly L2-cache misses occur as a result of: (1) data sharing patterns, especially within the address space allocated to working storage, the buffer pool, and components of the operating system, (2) related cache invalidations initiated by the cache-coherence protocol, and (3) process migration. A subsequent study using a 32-way configuration, which is underway and for which partial results are presented, corroborates some of these findings.

This paper presents the data analysis described above but, since the research that will identify the reasons for the observed behavior is not yet complete, it does not present evidence of the causes of this behavior. Given the database, and associated tools, that are under development, identification of the reasons for this behavior will be straightforward. The database stores information obtained from sampled event traces of data-load hit and miss events at the various levels of the p690 memory hierarchy. An interface allows a user to query the database to view statistics derived from the stored information, which includes the data address at the byte level, the corresponding instruction address (also at the byte level), the thread id, the process id, the CPU id, and the timestamp.

The remainder of the paper, which discusses this work in more detail, is organized as follows. Section 2 presents related research. Section 3 focuses on data collection, describing the workload under study (TPC-C), the platform from which the data was collected, the events of interest, and the tools used to collect the data. Section 4 targets data analysis, describing the tools and methodology used and the results of the analysis. Sections 5 and 6 present the conclusions, the database design, and future work.

2. Related research

Related research focuses on two aspects of this study: the performance of TPC-C on other multiprocessor platforms and the use of event trace sampling. With respect to the performance of TPC-C, Tsuei, Packer, and Ko [5] study TPC-C executed on an unidentified Sun Microsystems 16-CPU shared-memory multiprocessor system with 4GB of memory using IBM's DB2 for Solaris Version 2.1.1, while Leutenegger and Dias [2] study TPC-C executed on an unidentified multiple-node distributed system.

[2] and [5] investigate the buffer hit rates for TPC-C. In contrast, we investigate the levels of the memory hierarchy at which most cycles are consumed in servicing load instructions, the levels of the memory hierarchy at which load hits are recorded, and the data-load access distribution across the address space by region, segment, page, and cache block. Our findings, i.e., that load accesses are dominant in certain regions and within those regions, smaller defined areas are heavily accessed, corroborates the study by [2], which investigates the memory access characteristics of the TPC-C benchmark. [2] shows that data access skew, i.e., non-uniform data memory access, exists at the tuple and page levels, citing that 84% of the accesses are directed at approximately 20% of the hottest tuples. [5] also notes this skew, making reference to [2].

The work of [5] uses IBM's DB2 database engine on a commercial shared-memory multiprocessor and investigates the performance impact of changes to the database size, the number of CPUs, the database buffer size, or some combination of the three. Performance is measured in terms of transactions per minute (TPM), or throughput, buffer hit rates, and index hit rates. The motivation behind this work is to show that simply increasing the size of the buffer in physical memory, which becomes an increasingly tempting option to improve performance as technology advances and memory becomes cheaper, does not necessarily improve the performance factors just mentioned. The authors demonstrate that in order to maintain the same rate of performance, an increase in buffer size requires an even greater increase in the size of physical memory. [5] quantifies relationships between data hit rate, buffer size, and database size; between TPM, data hit rate, and database size; and between TPM and the number of CPUs. From this data the authors developed algorithms that are able to predict throughput when the buffer size

and number of CPUs change. Our study compares results obtained for 8-way and 32-way systems, and the size of the system's physical memory is a consequence of the number of processors in the system. In our study, the percentage of memory used to store the database buffer is considerably larger than that recommended by [5] (about 66%); in addition, the size of physical memory is much larger than that with which the authors experimented. But again, our concern is not buffer hit rates, but rather access patterns for L2-cache misses. Note, however, that future work certainly will investigate the affect of different buffer sizes on locality of reference patterns.

With respect to using event trace sampling, Desikan, et al. [1] use the Compaq DCPI (DIGITAL Continuous Profiling Infrastructure) tool to sample certain events. The samples are used to derive for the Compaq DS-10L workstation performance measurements, such as IPC, which are used to check the reliability of an Alpha 21264 simulator. Care is taken in selecting a reasonable interval between samples. Decreasing the interval size ensures that collected data is more representative of the machine's actual performance, but increases execution time. Conversely, increasing the interval size has a smaller affect on execution time but the resulting data set is less representative and, consequently, errors may be introduced. In a range from 1000 to 64K cycles, the best results are obtained when samples are taken every 40,000 cycles. For the 1.3 GHz POWER4 used in this study, we take samples about every 13 million cycles.

3. Data collection

This section provides information on the data collected for this study. First, we describe the workload, TPC-C, and the platform on which TPC-C was executed and monitored. Next, we discuss the events of interest and the methodology used to collect the sampled traces of the data access streams produced by L2-cache data-load misses generated by the TPC-C benchmark.

3.1. 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 [6] is a well-known benchmark that emulates read-only and update-intensive transactions found in complex on-line transaction processing (OLTP) application environments [5]. It has been used widely in the database server industry as a basis of server performance analysis and platform comparison.

3.2. Compute platform: IBM eserver pSeries 690 architecture

The eserver pSeries 690 family of symmetric multiprocessor (SMP) architectures includes the 8-way and 32-way configurations used in this study [7,8]. The operating system for these configurations is AIX version 5.2. The MultiChip 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; each MCM contains eight processors. Thus, the 32-way configuration contains four MCMs. Normally, the 8-way configuration contains one MCM. In contrast, the 8-way configuration used in this study contains two – each MCM contains four "single core good" chips, each of which has only one functional processor.

For the configurations under study,

- each CPU is accompanied by a 64KB L1 instruction cache and a 32KB L1 data cache;
- a 1.44MB L2 unified cache is associated with each chip, i.e., it is shared by the two processors on a chip;
- a 128MB L3 unified cache is shared by the four chips/eight processors on an MCM; and
- main memory is 128GB for the 8-way configuration and 256GB for the 32-way.

The L1 and L2 caches have a cache line size of 128 bytes, while the L3 cache has a 512B line size. Data private to and shared by processes are managed via the cache coherence protocol implemented in the studied architectures. 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, the L2.5 level;
2. an L2 cache in another MCM, the L2.75 level;
3. the MCM's L3 cache, the L3 level;
4. an L3 cache in another MCM, the L3.5 level; and
5. main memory.

3.3. L2-cache miss events

L2-cache miss events are classified according to the level at which they are resolved and the state of the block, with respect to other levels of cache and memory, at the resolution site. Misses serviced at the L2.5 level generate one of two types of events: an L2.5-shared or L2.5-modified hit. An L2.5-shared hit event (L25_SHR) denotes that although the requested block may reside simultaneously in more than one L2 cache, it is resolved by the MCM containing the processor experiencing the

miss. An L2.5-modified hit event (L25_MOD) denotes that the requested block resides in only one cache and that cache is on the MCM that contains the processor experiencing the miss. A modified block is exclusively owned by one cache and contains more recent data than is in the backing physical memory.

Similarly, L2-cache misses serviced at the L2.75 level of the memory hierarchy generate either an L2.75-shared or L2.75-modified hit event. The former (L275_SHR) denotes that the requested block resides in more than one L2 cache but not in any L2 cache on the MCM of the processor experiencing the miss. The latter (L275_MOD) denotes that the requested block resides in only one L2 cache and that cache is not on the processor's MCM.

At the L3 level, the cache-hit events are called L3-shared, L3-modified, L3.5-shared, and L3.5-modified. An L3-shared hit event denotes that the requested block resides in more than one L3 cache, including the one associated with the MCM of the processor experiencing the miss. An L3.5-shared hit event denotes that the requested block resides in more than one L3 cache, but not the one associated with the processor's MCM. An L3-modified hit event denotes that the requested block resides in only one cache, the one on the processor's MCM. An L3.5-modified hit event denotes that the requested block resides in only one cache and that cache is not the one on the processor's MCM.

Table 1. Load latencies of 8-way configuration

L2-Cache Access Resolution 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

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). Approximate load latencies associated with these events are given in Table 1. The latencies are about the same for the 32-way system.

3.4. Event Trace Sampling Methodology: PMU, eprof, and trcrpt

On selected pSeries hardware models, through the use of tools such as eprof and trcrpt, described in this section, trace information for specified events can be collected. These tools were used in this study to collect two traces, one for an 8-way and one for a 32-way configuration of the p690 multiprocessor. The trace information for the events described in Section 3.3 was gathered during the execution of TPC-C, which consumed 10-minutes of execution time. Regardless of the event being sampled, the default sampling rate of 100 events/second per CPU was used. Sample information was recorded upon the periodic occurrence of the event being monitored. The information collected during each sample includes the timestamp indicating when the event occurred, the effective instruction and data addresses associated with the event, as well as the CPU id, process id, and thread id of the entity that triggered the event. This information was then used to conduct the performance analysis described in Section 4.

The POWER4 microprocessor includes performance monitoring facilities that can collect data on various events that occur within the processor, such as the completion of a load instruction or an L2 instruction cache miss, and, thus, provide valuable performance information. The performance-monitoring unit (*PMU*) includes eight counters that permit up to eight concurrent events to be monitored. 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, the execution points at which to start and stop counters, and the points at which software is to retrieve results. In addition to recording aggregate counts for either a section of code or an entire program, the PMU is capable of capturing instruction and data addresses associated with events. This is of particular value when event-based sampling is desired. Event-based sampling, which is important for long-running programs with extremely large numbers of events, is provided by the PMU and associated software via user-selected trigger events and Performance Monitor (PM) interrupts. As is exemplified below, the former can be used to trigger the increment of a counter and the latter can be used to write PMU data to a file.

The AIX operating system contains a time-based profiling tool called *tprof*. In addition to tprof, there exists 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. This tool, *eprof*, programs the PMU to sample hardware countable events at a defined rate. For this research, we employed eprof and event-based sampling, using eprof's default sampling rate of approximately 100 events per second per CPU. In this way, using the default sampling rate, if the event sampled 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 rate at which samples are collected so

that the 100 samples per CPU per second collection rate is approximated. Accordingly, the interval between PM interrupts can be variable, and 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. The size of the collected data set for each event of interest is given in Table 2. Due to memory limitations of the data analysis tools used in this study and the immense amount of samples collected for the L25_SHR and L275_SHR events, the sample counts for the 8-way are reduced counts. For this same reason, the sample counts for L25_SHR and L275_SHR events for the 32-way configuration could not be analyzed and are, therefore, not presented.

Table 2. Event sample counts

Event	Sample Count	
	8-way	32-way
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

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 PM interrupt is delivered. The 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.

4. 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.

4.1. Methodology

As mentioned in Section 3.3, the IBM tool trcrpt was used to post-process the sampled AIX event trace generated by eprof. The specified output of trcrpt includes the effective instruction and data addresses, the process id, and the timestamp for each sampled event. This is input to a program written in C that

- sorts each sample, according to its effective data address, into its corresponding address space region (described in Section 4.2) and, within a region, into its corresponding segment, page, and cache block, and, in this way, acquires data-load hit counts for regions, segments, pages, and cache blocks
- stores process ids along with data addresses and, using this data, calculates the amount of private and shared memory "touched", i.e., accessed, by the cooperating processes — note that two samples having the same effective address but a unique process id denote two references to two distinct "per-process" or private regions
- utilizes the program's symbol table to determine the number of data-load hits associated with and the amount of memory touched by a set of routines that are suspected to be affecting performance
- filters out data that is insignificant — for example, the program can identify regions that are the target of 90-100% of the memory accesses, i.e., concentrated areas of locality of reference.

4.2. Data Partitioning

The address space for TPC-C ranges from 0x0000000000000000 to 0xF10000B6FFFFFFFF and is partitioned as illustrated in Table 3, which appears at the end of the paper. The segment size is 256MB, while each page in a segment is 4096KB in size. As can be seen from the table, the address range used for lock instrumentation begins at 0xF100009E00000000 and ends at 0xF100009E0fffffff.

The TPC-C application used in this study is based on a process model. The process model allows for private data per process, as well as sharing of global database information via a shared memory region. For this workload, the shared memory region contains the database's state information and buffer pool. The buffer pool is the largest consumer of physical memory. It contains unmodified data, currently on disk, as well as data that has been modified by transactions and is 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% CPU utilization. Because most transactions experience some number of disk I/O's, many transactions must be executing concurrently to maximize CPU utilization.

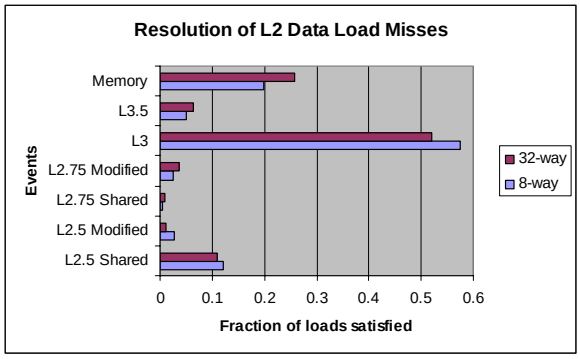

Figure 1. Distribution of TPC-C L2-cache data-load miss resolution sites of the p690 memory hierarchy

4.3. Results

The goal of this analysis is to pinpoint the application-specific sources of performance degradation associated with data references. This is done in three phases.

Phase 1. The platform-specific causes of performance degradation are identified. For example, as is true in this study, it may be the case that a high number of L2-cache misses are satisfied by the L3 caches or main memory, rather than by other L2 caches.

Phase 2. The concentrated areas of locality of reference are identified. For example, references may be concentrated in the buffer pool.

Phase 3. The subroutines, instructions, and/or variables associated with these areas of locality of reference are identified. For example, a lock variable may be the target of a significant number of these references. (Note that Phase 3 is in progress.)

4.3.1. Phase 1. Figure 1 presents, for both the 8-way and 32-way configurations of the p690, performance monitor event counts that are associated with L2-cache data-load misses. These event counts show the distribution of these misses across the resolution sites of the p690 memory hierarchy. Recall that in this architecture, L2-cache misses can be resolved by another L2 cache within the MCM of the processor experiencing the miss (L2.5), an L2 cache outside the processor's MCM (L2.75), the L3 cache of the processor's MCM (L3), an L3 cache outside the processor's MCM (L3.5), or main memory. This data identifies the platform-specific causes of performance degradation associated with L2-cache data-load misses, i.e.,

- *L3 caches and main memory dominate as the levels of the p690 memory hierarchy where L2-cache data-load misses are resolved.*
- The distribution of L2-cache data-load misses across resolution sites is similar for the 8-way, 2-MCM configuration and the full 32-way, 4-MCM configuration.

4.3.2. Phase 2. During Phase 2, the analysis hones in on the concentrated areas of locality of reference. The analysis progresses from a level of the memory hierarchy to a region of the address space, then to segments, pages, and, finally, cache blocks. From cache blocks, the analysis can continue to instructions, processes, CPUs, etc.

Regions of the Address Space – L3 Caches: As shown in Figure 1, data-load hits in the *L3 caches appear to be one of the main factors affecting the performance of the TPC-C benchmark running on the p690*. Thus, we first explore the reason for this. Figure 2 depicts, for the 8-way configuration of the p690, the hit percentages for the eight most-referenced regions of the TPC-C address space. A hit percentage for a region is calculated by dividing the number of references to the region by the total number of memory references. By examining the light-colored hit bars, we see that *the data/bss/heap and buffer pool regions clearly are the hardest hit*. The dark-colored bar, called the unique_cache_line bar, indicates the number of unique cache lines (in the associated address region) referenced; it gives us an idea of the density of the data loads for this region. For example, the kernel region shows a unique_cache_line bar that is only a small portion of the size of its corresponding hit bar. This indicates that *all the L3-cache hits associated with the kernel region reference a relatively small number of cache lines*. Conversely, the stack section of the graph indicates that the *data references to the stack region of the address space are much more dispersed*. That is, there are probably many cache lines in the stack region associated with L3 data-load hits that only get referenced once or twice. This is not surprising for a per-process or private region of the address space, such as the stack region. Another per-process region, such as the data/bss/heap region, illustrates this as well.

Figure 3 depicts this same information for the 32-way configuration of the p690. Note that the eight most-

referenced regions of the TPC-C address space are the same for both the 8-way and 32-way configurations and that the hit percentages are very similar as well. Again, we see that *the data/bss/heap and buffer pool regions clearly are the hardest hit.*

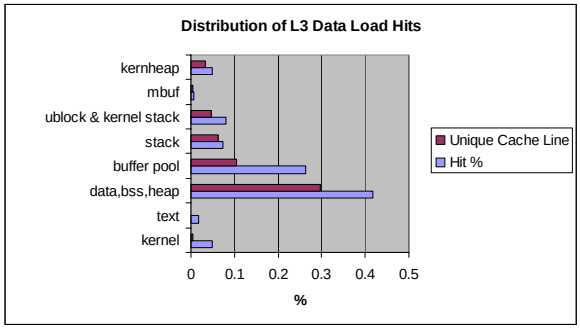

Figure 2. Distribution of TPC-C L3-cache data-load hits across memory regions for the 8-way p690 configuration

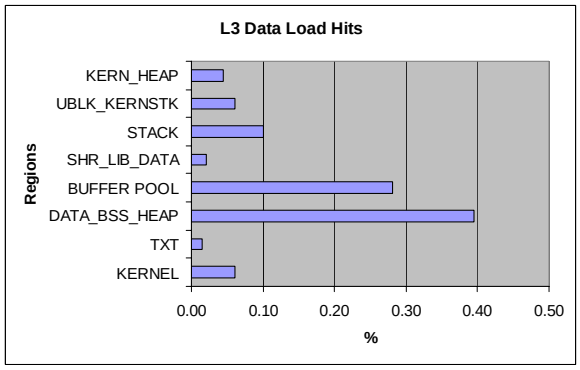

Figure 3. Distribution of TPC-C L3-cache data-load hits across memory regions for the 32-way p690 configuration (traces of only 19 CPUs are represented)

Regions of the Address Space – Main Memory: Since data-load hits in main memory also appear to be a main factor affecting the performance of the TPC-C benchmark running on the p690, we compare the distribution of data-load memory hits among the eight most-referenced regions with that of L3-cache data-load hits. Figure 4 depicts the distribution of data-load hits in memory for the 8-way configuration, while Figure 5 represents the data-load hits in memory for the 32-way configuration. Comparing this distribution with that of the L3-cache data-load hits, we see contrasts in locality of reference for the various regions of the address space. For example, *the data loads that are targeted at the buffer pool and miss the local and remote L3 caches no longer exhibit the same tight reference pattern, i.e., these loads exhibit a larger footprint – they are distributed across a relatively large number of cache lines. The other regions exhibit this same behavior.*

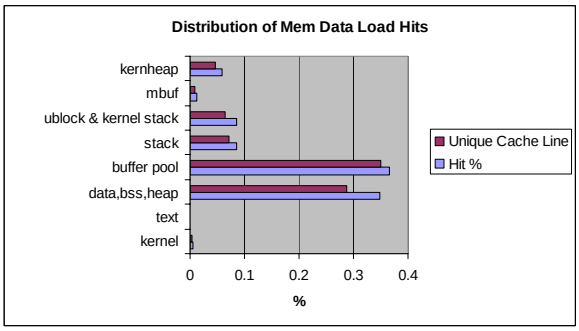

Figure 4. Distribution of TPC-C main memory data-load hits across memory regions for the 8-way p690 configuration

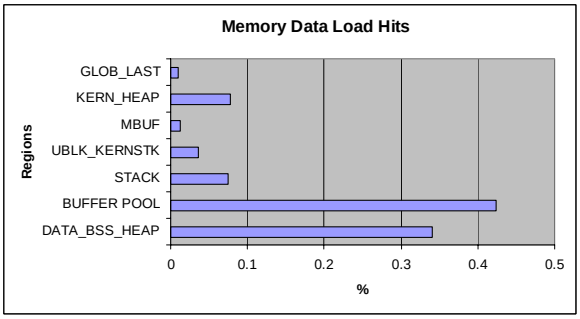

Figure 5. Distribution of TPC-C main memory data-load hits across memory regions for the 32-way p690 configuration

***Regions of the Address Space – Segments*:** For the ten-minute duration in which samples were collected, 109 unique segments were touched in the buffer pool. Of these 109 segments, four of them account for over 90% of the data-load activity in L3 caches. Figure 6 shows the four segments and their respective hit and unique_cache_line bars. In this figure, we can see the continuing pattern first seen in Figure 3, i.e., the majority of the hits reference a relatively small number of cache lines. One of the segments, however, appears to have been referenced in a much more uniform manner.

Regions of the Address Space – Pages: Continuing to hone in on the suspect causes of performance degradation, we next take a closer look at segment 0x070000004 of the buffer pool for data loads that hit in L3 caches. Examining Figure 7, which plots the distribution of L3-cache data-load hits across the pages of the TPC-C buffer pool segment for the 8-way p690 configuration, we see a very dense reference pattern. It shows that greater than 70% of the hits for this segment are located within a range

of approximately 200, out of 65,536, pages. In addition to this clustering of hot pages, we see that each page exhibits, as did the segments in the buffer pool region, a very dense reference pattern.

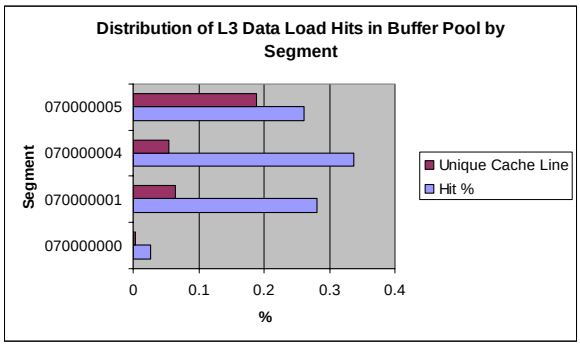

Figure 6. Distribution of TPC-C L3-cache data-load hits across segments of buffer pool for the 8-way p690 configuration

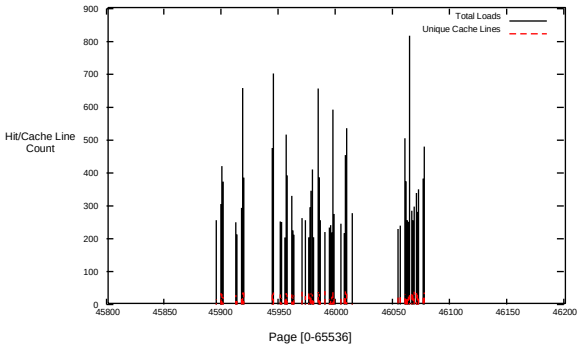

Figure 7. Distribution of TPC-C L3-cache data-load hits across pages of buffer pool segment for the 8-way p690 configuration

Regions of the Address Space - L3 Caches: From the page-related data, we would expect that within a page we would see heavily-referenced cache lines. Figure 8, shows just that. It is quite clear that only a handful of cache lines are responsible for the majority of the references, i.e., greater than 70% of the references.

***Regions of the Address Space – Instructions*:** As we indicated in Section 4.1, our data analysis tools allow a user to specify a list of routines and obtain a report that displays data-load hit percentages and the amount of memory touched for regions of the address space referenced by the routines. For this study, the lock routines and atomic operations of Table 4 were targeted as being potentially responsible for data loads resolved in the lower levels of the memory hierarchy. Our results indicate that only two routines from the ones listed had any notable impact on performance. The data shows that the disable_lock and simple_lock routines, whose data address is retrieved from an L3.5 cache, make up the biggest portion of data-load hits that are associated with lock and atomic operations. However, these percentages are insignificant, 1.1% and 2.2% respectively, and, therefore, do not contribute greatly to performance degradation with respect to L2-cache misses.

Table 4. List of routines under analysis

Lock routines	Atomic operations
simple_lock	fetch_and_add
simple_lock_ppc	fetch_and_add_h
simple_unlock	fetch_and_addlp
disable_lock	fetch_and_or
unlock_enable	fetch_and_orlp
simple_unlock_mem	fetch_and_and
unlock_enable_mem	fetch_and_andlp

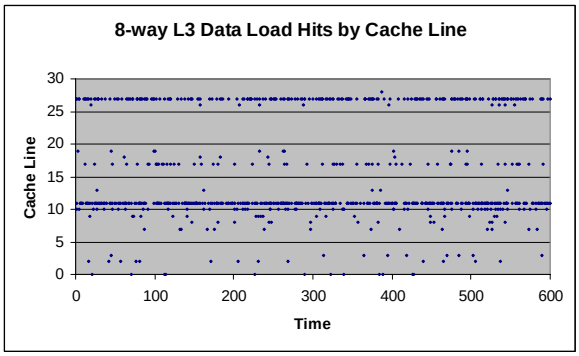

Figure 8. Distribution of TPC-C L3-cache data-load hits across the cache lines of a hard-hit page for the 8-way p690 configuration

5. Conclusions

Data collected from the 8-way and 32-way configurations of the p690 indicate that L2-cache data-load misses are often resolved in L3 caches or main memory. This information, coupled with the fact that load hit latencies for L3 caches and main memory are very high in comparison to L2-cache load latencies (see Table 1), presents an obvious target for performance enhancement. Through analysis, we will endeavor to attain this enhancement by (1) uncovering the application-specific causes for the L2-cache data-load misses that are resolved in L3 caches and main memory and (2) applying appropriate remedies that shift the resolution site of these expensive data-load misses to the L2 level of the memory hierarchy. The work presented in this paper provides the foundation for accomplishing this by honing in on the areas of memory, down to the cache block level, that

demonstrate a high concentration of data-load hits. Specifically, our results show that the buffer pool and heap regions of the TPC-C address space tend to dominate as the effective data address regions for data loads satisfied in L3 caches and main memory. Furthermore, for those data loads satisfied in L3 caches, the segments, pages, and cache blocks that constitute the buffer pool exhibit a rather dense distribution.

We were able to confirm through the use of our tools that routines associated with lock variables and atomic operations do not play a dominant role in the cause of L2-cache data-load misses. In fact, the percentage of these functions that are attributed to L2-cache data-load misses is so small that we did not even present the distribution of these misses across the address space as was done with the entire sample set.

6. Future Work

After reviewing the data that is presented here, the authors began discussing the approaches one could take to take advantage of the ability to determine locality of reference. It was realized that ideally one would want to not only isolate a concentrated region of L3-cache data-load hits, for example, but also be able to analyze this region by determining if this repeated occurrence of L2-cache data-load misses could be related to certain CPUs, processes, threads, or routines. Having this kind of information would almost certainly provide more insight into the p690 memory hierarchy behavior and help determine the causes of L2-cache data-load misses generated by TPC-C as well as other applications.

Due to limitations associated with the data structures that are currently being used in the data analysis tools, this type of information cannot be generated efficiently, especially for the 32-way traces. Therefore, a database is under development. Using this database, repeated occurrences of L2-cache misses can be easily correlated with the threads, processes, and CPUs that generated them.

The database accepts as input the event information generated by trcrpt. It also accepts user input regarding information necessary to document the experiment including the following:

- specific address regions, such as the address range of the segments,
- event names,
- machine configuration, e.g., whether it is an 8-way or 32-way configuration,
- run information, such as the run duration and date the run was conducted, and
- workload information, e.g., the workload used to generate the event information.

Through the use of a script, the user-specified information, and input file are used to populate the database. The script also stores in the database a low-level count of the number of accesses to a specific word; this is maintained in an effort to minimize the time required to calculate the more complicated queries, such as deriving the segment and page counts. As a result, the user will be able to query the database and view the event information in a number of different ways. For example, the user will be able to view information such as the CPU that is associated with the majority of L2-cache misses. In this way, the database will allow us to better understand the relationship between the L2-cache misses and their related causes.

Acknowledgements

We wish to thank Robert Acosta, Robert Amezcua, 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 The Austin Center for Advanced Studies (ACAS) for the faculty research award that made this research possible, and Cathy Nunez for arranging Trevor Morgan's summer 2002 internship, which kicked-off this research.

References

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

[2] 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*, Washington, D.C., USA, June 1993, pp. 22-31.

[3] A. Nanda, K. Mak, K. Sugavanam, R. K. Sahoo, V. Soundararajan, and T. Basil Smith, "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*, Cambridge, MA, USA, November 2000, pp. 37-48.

[4] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta. "Complete Computer Simulation: The SimOS Approach", *IEEE Parallel and Distributed Technology: Systems and Applications*, Winter 1995, pp. 34-43.

[5] 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*, Tucson, AZ, USA, June 1997, pp. 112-122.

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

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

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

Table 3. TPC-C address space

Address Space	Range
Kernel	0x000000000 - 0x000000001
Proc. Priv., shmat/mmap & Loader Use	0x000000002 - 0x00000000F
Text	0x000000010 - 0x000000010
Data,BSS,Heap	0x000000011 - 0x06FFFFFF
Buffer Pool	0x070000000 - 0x07FFFFFF
Private Load	0x080000000 - 0x08FFFFFF
Shared Library Text	0x090000000 - 0x090010009
Shared Data	0x09001000A - 0x09001000A
Reserved	0x0A0000000 - 0xEFFFFFFFF
Stack	0x0F0000000 - 0x0FFFFFFFF
U-Block and Kernel Stack	0xF00000002 - 0xF00000002
DATA	0xF10000004 - 0xF10000004
PTA	0xF10000005 - 0xF10000005
DMAP	0xF10000006 - 0xF10000006
AME	0xF10000007 - 0xF1000000A
SCB	0xF1000000B - 0xF100000BA
SWHAT	0xF100000BB - 0xF1000013A
SWPFT	0xF1000013B - 0xF1000083B
Reserved	0xF1000083C - 0xF10000877
PROC_THRD	0xF10000878 - 0xF1000089B
M_BUF	0xF1000089C - 0xF1000099F
LDR_LIB	0xF100009A0 - 0xF100009BF
JFS_SEG	0xF100009C0 - 0xF100009C0
JFS_LKW	0xF100009C1 - 0xF100009CF
LFS_SEG	0xF100009D0 - 0xF100009DF
LOCK_INSTR	0xF100009E0 - 0xF100009E0
KERN_HEAP	0xF100009E1 - 0xF10000AE0
MP_DATA	0xF10000AE1 - 0xF10000AF0
GLOB_EXTREG	0xF10000AF1 - 0xF10000B6F