Memory Performance Profiling via Sampled Performance Monitor Event Traces

Diana Villa, Jaime Acosta, and Patricia Teller
The University of Texas at El Paso
Department of Computer Science
demarquez@utep.edu, jaime.acosta@wsmr.army.mil, and pteller@cs.utep.edu

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

Trevor Morgan
Exxon/Mobil
trevormorgan31@sbcglobal.net

Abstract

Memory performance can be studied, process behavior can be characterized, and application performance can be improved through the use of sampled performance monitor event traces. As an example, this paper demonstrates how sampled traces of the TPC-C benchmark executed on eight- and 32-processor configurations of the IBM eServer pSeries 690 (p690) are analyzed to identify the resolution sites of level-two (L2) cache data-load misses and study the heavily-hit resolution sites, i.e., level-three (L3) caches and main memory, with the goal of recognizing the heavily-hit regions of the application's address space, segments, pages, cache blocks, routines, instructions, and data structures. Preliminary data analysis of the traces, using a powerful and flexible performance evaluation framework, indicates that data-load hits at heavily-hit resolution sites have high locality of reference within regions of the address space, segments, and pages. Specifically, the buffer pool and heap regions of the TPC-C address space dominate as the effective address regions for data loads satisfied by local L3 caches and main memory. Furthermore, for the data loads satisfied by local L3 caches, the segments, pages, and cache blocks that comprise the buffer pool exhibit a dense distribution. Work continues to characterize related process behaviors as well as other workloads, and to define ways to remedy the performance degradation associated with L2-cache data-load misses serviced at high-penalty levels of the p690 memory hierarchy.

1. Introduction

In the long run, the research reported in this paper is being performed to answer the following question: "As processor frequency and memory size increase, can we generate the address traces and/or memory-hierarchy miss rate information needed to permit us to study how to optimize memory subsystem performance?" To begin to answer this question, we use sampled performance monitor event traces to profile the memory performance of large, complex applications. To facilitate the analysis of the traces, we developed a powerful and flexible performance evaluation framework, which can be used in many ways, e.g., to characterize process behavior and to understand what modifications to the application, operating system, and/or architecture will improve application performance. To demonstrate the usefulness of sampled event traces, this paper analyzes the traces of the TPC-C benchmark executed on eight- and 32-processor *IBM eServer pSeries 690 systems (p690s)*. The foci of the analysis are the sources of L2-cache data-load misses and their points of resolution

Why focus on just L2-cache data-load misses? While not the only source of memory subsystem activity, data cache misses are the most dominant, and for workloads like TPC-C, they are the most important. Other potentially interesting events include instruction cache misses, *translation-lookaside buffer (TLB)* misses, address-only coherence operations, and uncached memory accesses for I/O.

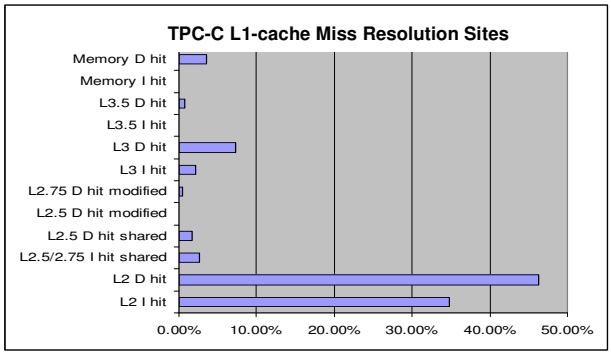

Figure 1. TPC-C L1-cache misses resolution sites for the 32-processor p690

The performance monitor unit of the POWER4 microprocessor, the basis of the p690, makes it possible to count instruction and data cache misses serviced at the different levels of the memory hierarchy. In addition, it is possible to sample the instruction and data addresses associated with data cache misses, but it is not possible to

sample the instruction addresses associated with instruction cache misses. In general, this is acceptable, since the instruction cache footprint of commercial workloads tends to be well cached in large *level-two (L2)* and *level-three (L3)* caches. As shown in Figure 1, which identifies the points of resolution for *level-one (L1)* cache misses, this is the case for TPC-C, i.e., data cache misses dominate the activity at the L2 and L3 caches, as well as memory. From a CPI viewpoint, the L2-cache data-load misses that hit at high-penalty resolution sites, i.e., L3 caches and memory, are most important given the load latencies presented in Table 1 and if decreased, could have a positive impact on performance. Note that the different cache events, i.e., L2, L2.5, L2.75, L3, L3.5, and main memory, are described in more detail in Section 4.3.

L2-cache Service 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. Load latencies of the eight-processor p690

With respect to TLB misses, the rate of TPC-C TLB misses is reduced by mapping the database buffer pool using 16MB "large pages" rather than standard 4KB pages. Additionally, unlike cache and TLB misses, address-only coherence operations do not involve movement of data. For example, if 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, their impact on the performance of POWER4-based systems tends to be small.

Finally, uncached memory accesses for I/O via loads and stores have very high latency, which 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.

Why TPC-C? TPC-C is a transaction-processing application that is understood fairly well and is representative of commercial workloads of interest to IBM customers. Additionally, preliminary data analysis of information concerning the data access streams generated on the p690 by TPC-C L2-cache data-load misses [7] indicates that there is opportunity for performance improvement. 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 improvement.

Why sampled performance event traces? Historically, cache analysis is done using traces generated by hardware or by software architecture simulation, for example, SimOS [10]. 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 by today's standards), and usually more memory. Also, the time to simulate a large multiprocessor system is intimidating. Alternatives to tracing are a cache simulator built in hardware and connected to a running system [8], and sampled event traces, the alternative that we adopted.

2. Motivation

In this paper, we demonstrate how analysis of sampled event traces, facilitated by a powerful and flexible performance evaluation framework, described in Section 5.1 and pictured in Figure 14 at the end of the paper, can be used to identify (1) the areas of the address space, down to a granularity of 128B cache lines, 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.

In the case of the TPC-C benchmark executed on eight- and 32-processor configurations of the p690, this analysis indicates that a fairly large number of L2-cache misses are resolved at local L3 caches and main memory, where latencies are relatively high in comparison to load-hit latencies at local and remote L2 caches and remote L3 caches, respectively. The resolution of these misses at high-penalty levels of the memory hierarchy does not seem intuitive for two reasons. First, the p690 architecture, further described in Section 4.2, 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 an 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-processor 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 [7]. Such clustering indicates locality of reference that, if exploited, should lead to hits in the upper-level caches.

Given the identification of the reasons why L2-cache misses are resolved at high-penalty levels of the p690 memory hierarchy, it may be possible to modify the application, operating system, and/or hardware to alleviate or at least decrease them. Research in progress, facilitated by our performance evaluation framework, is aimed at identifying the reasons. Information gathered thus far suggests the following: (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.

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

3. Related Research

Related research focuses on two aspects of this study: the use of event trace sampling and the performance of TPC-C on other multiprocessor platforms. 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. [5] use performance monitors to analyze the behavior of an OLTP workload executed on a four-processor Pentium Pro-based server. Both explore the performance effects of architectural modifications. In [2] this is done by workload characterization, accomplished by source code instrumentation coupled with simulation methodologies and in [5] this is accomplished by physically changing the hardware. Desikan et al., like Barroso et al., also use the DCPI tool [1] to check the reliability of an Alpha 21264 simulator by sampling certain 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. [11] 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, while Leutenegger and Dias [6] study it executed on an unidentified multiple-node distributed system. Both investigate TPC-C's buffer hit rate. Our initial results [7] and those presented in this paper, 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, which also investigates 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.

Unlike the research described above, Itzkowitz, et al. [4] 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 and provides 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 this paper but, of course, our performance evaluation framework and compute platform are used to perform the analysis.

The major differences between our work and the related research described above are the scale of the systems and the methodology used. Itzkowitz, et al. use a two-processor system, Barroso et al. and Keeton et al. each use a four-processor system, and Tsuei et al. use a 16-processor system, while 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. Our methodology does not require source code instrumentation and it is not restricted to memory access behavior analysis. Our performance evaluation framework provides numerous ways to analyze sampled event traces. A description of the type of analyses that can be performed is presented in Section 5.

4. 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 event traces of the data access streams produced by L2-cache data-load misses.

4.1. 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 [13] is a well-known benchmark that emulates read-only and update-intensive transactions found in complex *on-line transaction processing (OLTP)* application environments [11]. It has been used widely in the database server industry as a basis of server performance analysis and platform comparison.

4.2. Compute platform: IBM eServer pSeries 690

IBM's eServer pSeries 690 family of *symmetric multiprocessor (SMP)* architectures includes eight- and 32-processor configurations [14,15]. The operating system for these configurations is AIX version 5.2. The *MultiChip Module (MCM)*, the building block of the architecture, contains four chips. Each chip is comprised of two 1.3 GHz POWER4 processors and, thus, in general, each MCM contains eight processors. Accordingly, the eight- and 32-processor configurations normally are comprised of one and four MCMs, respectively. In contrast, the eight-processor p690 configuration used in this study is comprised of two MCMs – each MCM contains four *"single core good"* chips, i.e. only one functional processor per chip, instead of the typical eight-processor configuration that only contains one MCM with eight functional processors. Note that, as expected, the 32-processor p690 configuration used in this study consists of four MCMs, each with all eight functional processors.

For the p690s under study,

- each CPU has a 64KB L1 instruction cache and a 32KB L1 data cache;
- 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-)processor p690.

The L1 and L2 caches have 128B lines, while the L3 cache has 512B lines. Data private to and shared by processes are managed via the p690 cache coherence protocol. 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, 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.

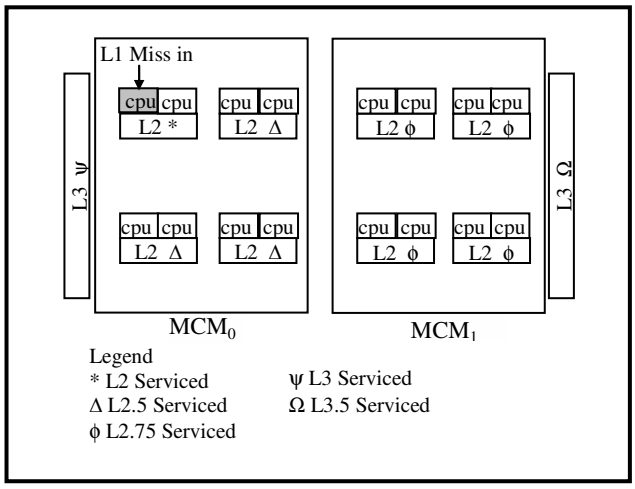

Figure 2. Two (double core good) MCMs of 32-processor p690

4.3. L2-cache miss events

L2-cache miss events are classified according to the level at which they are resolved and the state (with respect to the cache coherency protocol) of the block at the resolution site. Note that the load latencies associated with each of the L2-cache miss events described below is presented in Table 1.

Misses serviced at the L2.5 level generate one of two types of events: an *L2.5-shared (L25_SHR)* or *L2.5-modified (L25_MOD)* hit event. An L25_SHR denotes that, although the requested block may reside simultaneously in more than one L2 cache, it is resolved by a *local L2 cache*, i.e., one associated with the MCM containing the processor that generated the miss. An L25_MOD denotes that the requested modified block is exclusively owned by and, thus, resides in only one L2 cache, a local L2 cache; this L2 cache contains a more recent version of the block 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 (L275_SHR)* or *L2.75-modified (L275_MOD)* hit event. The former denotes that the requested block resides in more than one L2 cache but not in a local L2 cache, rather in an L2 cache on another MCM, i.e., a *remote L2 cache*. The latter denotes that the requested block resides in only one L2 cache, a remote L2 cache.

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 may reside in more than one L3 cache and is resident in the *local L3 cache*, i.e., the one associated with the MCM of the processor that generated the miss. An *L3.5-shared* hit event denotes that the requested block resides in more than one L3 cache but not in the local L3 cache. An *L3-modified* hit event denotes that the requested block resides in only one L3 cache, the local L3 cache. An *L3.5-modified* hit event denotes that the requested block resides in only one L3 cache, a remote L3 cache.

For this study we monitored cache hit events as well as main *memory* hit events (*MEM*). However, instead of monitoring the four events associated with the L3 level of the memory hierarchy, only two events were monitored: L3 and L3.5 hits (L3 and L35). Additionally, only the L25_MOD and the L275_MOD hit events are analyzed due to the unmanageable size of the event traces associated with the L25_SHR and L275_SHR hit events.

4.4. 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 one event trace from the eight-processor p690 and one from the 32-processor system. The trace information for the events described in Section 4.3 was gathered during a 10-minute interval of the steady-state execution of TPC-C. As described below, sample information was recorded upon the periodic occurrence of the event being monitored. The information collected during each sample includes the timestamp which indicates when the event occurred, the effective instruction and data addresses associated with the event, and the CPU, process, and thread IDs of the entity that triggered the event.

To 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 POWER4 microprocessor includes performance monitoring facilities. The *performance monitor unit (PMU)* includes eight counters that permit up to eight concurrent events to be monitored. 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.

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 execution points at which to start and stop counters and at which software is to retrieve results.

Event-based sampling, which is important for long-running programs with extremely large numbers of events, like TPC-C, 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, *eprof*, which 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.

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

Table 2. Event sample counts

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.

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.

5. Data Analysis

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

5.1. Methodology – Performance Evaluation Framework

As mentioned in Section 4.4, the IBM tool trcrpt was used to post-process the sampled AIX event trace generated by eprof. The specified output of trcrpt includes for each sampled event the effective instruction and data addresses, the CPU, process, and thread IDs, and the timestamp. As part of our performance evaluation framework, depicted in Figure 14 at the end of the paper, a set of Java tools processes each sample and stores it in a MySQL database according to the workload being monitored, the number of processors used to execute the workload, and the event being sampled. For example, database *tpcc_32_g48c1* stores the sampled event trace for the TPC-C benchmark executed on a 32-processor system associated with the L2-cache data-load misses resolved in local L3 caches (g48c1 identifies a local L3 hit event). Each database consists of 12 tables that store information related to the experiment itself, e.g., a description of the workload and compute platform, and data contained within the samples themselves. Once the sampled events are loaded into their corresponding databases, a second set of tools in our framework is used to query the databases and produce results of the queries, i.e., default and customized reports, in the form of formatted text files. These text files are transformed into graphs via a spreadsheet application with built-in graphing capabilities. In this way, the performance evaluation framework facilitates the analysis of the sampled event traces.

Storing the sampled performance monitor event traces in databases facilitates data analysis and provides numerous ways to easily examine and explore the data. Accordingly, the analysis and results presented in this paper is only a sample of the kind of information that can be obtained using this methodology. The potential of our performance evaluation framework is exemplified in [9] and [12], which describe different kinds of analyses that can be conducted using our methodology and the same sampled event traces used in this study.

Even though the work presented in this paper studies the behavior of a commercial application, i.e., the TPC-C benchmark, the methodology can be applied to study the memory subsystem behavior of any kind of application. Furthermore, it is not restricted to Power-based systems; it can be applied to any system with the capability of producing sampled event traces.

5.2. Data Partitioning

The address space for TPC-C ranges from 0x0000000000000000 to 0xF10000B6FFFFFFFF and is partitioned as illustrated in Table 4, which appears at the end of the paper. The segment size is 256MB, while each page in a segment is 4KB. As can be seen from the table, the different memory regions are identified by address ranges, e.g., the 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 a private memory region per process, as well as a shared memory region that stores global database information, i.e., 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/Os, many transactions must be executing concurrently to maximize CPU utilization.

5.3. Results

As mentioned above, the goal of this analysis is to pinpoint the 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 local L3 caches or main memory, rather than by other, local or remote, 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 data structures 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.)

5.3.1. Phase 1. Figure 3 presents, for both the eight-processor, i.e., 8-way, and 32-processor, i.e., 32-way, p690, sampled performance monitor event counts that are associated with L2-cache data-load misses. These hit event counts show the distribution of L2-cache data-load misses across the resolution sites of the p690 memory hierarchy. Recall that in this architecture, L2-cache misses can be resolved by a local (on-MCM) L2 cache (L2.5 hit), a remote (off-MCM) L2 cache (L2.75 hit), the local L3 cache (L3 hit), a remote L3 cache (L3.5 hit), or main memory (MEM hit). This data, which is similar for the eight-processor, two-MCM p690 and 32-processor, four-MCM system, identifies the platform-specific causes of performance degradation associated with L2-cache data-load misses, i.e., local L3 caches and main memory dominate as the levels of the p690 memory hierarchy where L2-cache data-load misses are resolved.

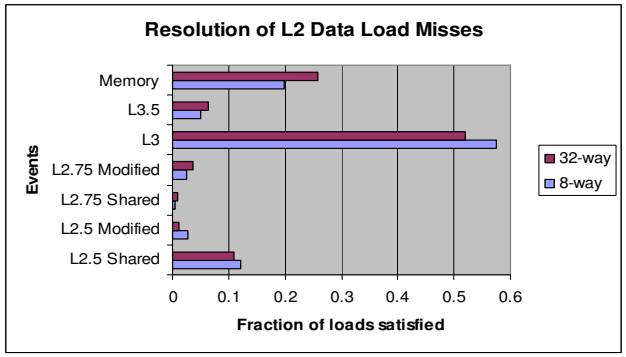

Figure 3. Distribution of sampled hit events among TPC-C L2-cache data-load miss resolution sites of the p690 memory hierarchy

5.3.2. 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, data structures, processes, CPUs, etc.

Regions of the Address Space – Local L3 Caches: As shown in Figure 3, data-load hits in local L3 caches, rather than hits in either local or remote L2 caches, appear to be one of the main factors affecting the performance of TPC-C running on the p690. Thus, we first explore the reason for this.

Figures 4 and 5 depict, for the eight- and 32-processor systems, respectively, the L3-cache hit percentages for the eight most-referenced regions of the TPC-C address space; note that the eight regions are the same for both data sets. Due to errors during the collection of the sampled event traces for the L2-cache data load misses that are resolved in the L3 cache for the 32-processor p690 configuration, the event traces for only 19, rather than 32, processors were successfully collected. As such, the 32-processor data set is based on only 19, rather than 32, event traces. Despite this, the two data sets expose similar data-load behavior in the L3 caches.

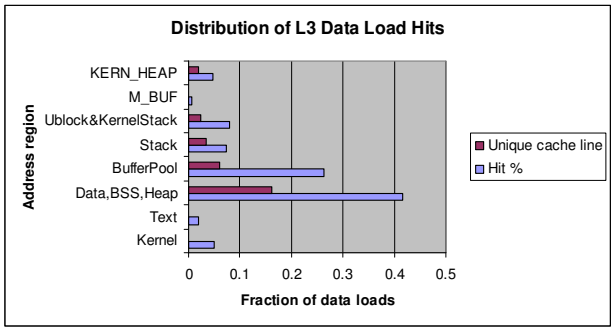

Figure 4. Distribution of TPC-C local L3-cache data-load hits across address regions of the eight-processor p690

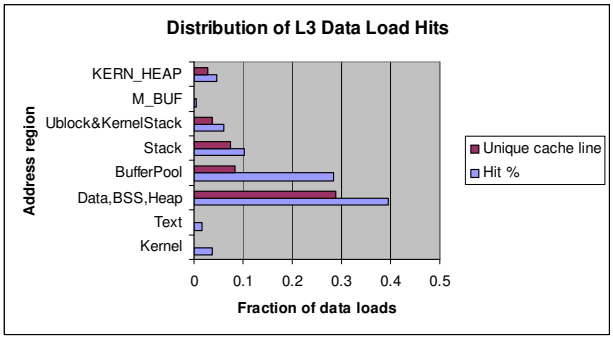

Figure 5. Distribution of TPC-C local L3-cache data-load hits across address regions of the 32-processor p690 (traces of only 19 CPUs are represented)

The light-colored *Hit_% bar* for a region is calculated by dividing the number of references to the region by the total number of references to the level of the memory hierarchy under study, in this case, local L3 caches. By examining the region Hit_% bars, we see that for both the eight- and 32-processor systems the Data,BSS,Heap and buffer pool regions clearly are the hardest hit at this level of the hierarchy.

A region's dark-colored bar, the *Unique_cache_line bar*, indicates the number of unique cache lines referenced in the region; it gives an idea of the density of the data loads, i.e., the locality of reference, for the region. In order to determine the cache block that is accessed by a particular data address, the address is partitioned into a tag, index, and offset using the L3 cache configuration, i.e., a 128MB eight-way associative cache with 512B blocks/lines, i.e., four 128B sectors. (Note that the L2-cache line size is 128B.)

With respect to locality of reference, four of the eight regions, M-BUF, Buffer Pool, Text, and Kernel exhibit the same behavior in both systems and all exhibit good locality of reference. For example, for both systems the Unique_cache_line bar for the buffer pool region is a relatively small portion (less than one-third) of the size of its corresponding Hit_% bar. This indicates that a majority of the local L3-cache hits associated with the buffer pool reference a relatively small number of cache lines and, thus, exhibit relatively good locality of reference.

In general, for the other four regions, the eight-processor system exhibits better locality of reference than the 32-processor system. In fact, the eight-processor system exhibits this behavior for the entire address space comprised of the eight regions depicted in Figure 4. That is, if the Unique_cache_line bars for the eight regions are aggregated and compared to an aggregated Hit_% bar, the aggregated Unique_cache_line bar is less than one-third the size of the aggregated Hit_% bar. This indicates that in the eight-processor system the majority of L2-cache data-load misses resolved in local L3 caches are to data previously referenced and pre-maturely evicted from L2 caches. If the evictions are due to false sharing or process sharing that can 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, in the 32-processor system, data-load hits to the "other" four regions, i.e., KERN_HEAP, Ublock & Kernel Stack, Stack, and Data,BSS,Heap, are more dispersed. This is illustrated by the large overlap of their Unique_cache_line bars and Hit_% bars, which indicates that a relatively large number of the referenced cache lines are only referenced once or twice. Thus, as is exemplified by the Data,BSS,Heap region, references to this region in the local L3 caches of the 32-processor system display worse locality of reference than in the eight-processor system.

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, next we compare the distribution of memory data-load hits among the eight most-referenced regions with that of L3-cache data-load hits. Figures 6 and 7 depict the distribution of memory data-load hits in the eight-and 32-processor systems, respectively. Comparing the distributions of memory and L3-cache data-load hits, we see contrasts in locality of reference for most of the 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 exhibited by the data loads that hit in local L3 caches, i.e., the memory hits exhibit a larger footprint than the local L3-cache hits. This is illustrated in Figures 6 and 7 by the buffer pool Unique_cache_line bar being a large percentage of its Hit_% bar, meaning the data-load hits to the buffer pool are distributed across a relatively large number of cache lines. The same behavior is exhibited by most of the other address regions of both the eight- and 32-processor systems. If the memory hits are the result of compulsory misses, then this indicates that the application is well matched to the architecture.

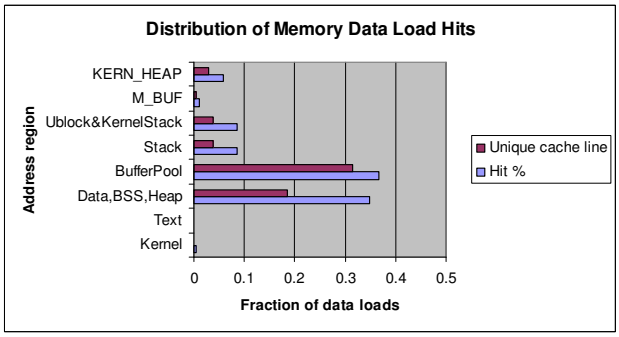

Figure 6. Distribution of TPC-C main memory data-load hits across memory regions of the eight-processor p690

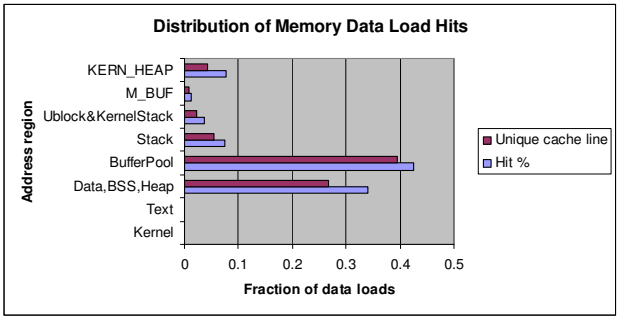

Figure 7. Distribution of TPC-C main memory data-load hits across memory regions of the 32-processor p690

Because the data-load hits in local L3 caches display better locality of reference when compared to those that hit in main memory, we now refine the analysis and study the references associated with L2-cache data-load misses that are resolved in local L3 caches, i.e., L3 hit events. Considering that Buffer Pool and Data,BSS,Heap are the

two most frequently referenced address regions in the L3 caches and the buffer pool region displays a more concentrated locality of reference than Data,BSS,Heap, the analysis now focuses on it.

Regions of the Address Space – Segments: For the 10-minute duration during which samples were collected, 302 and 570 unique segments in the buffer pool region were touched in the eight- and 32-processor p690s, respectively. Of the 302 (570) segments, four (six) account for over 90% of the buffer pool data-load activity in local L3 caches. Figure 8 (9) shows the four (six) segments and their respective Hit_% and Unique_cache_line bars.

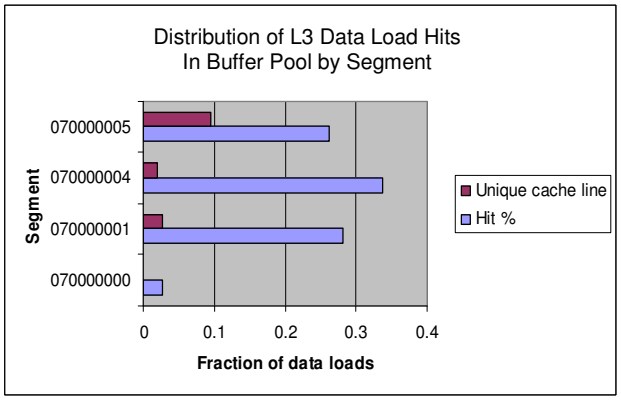

Figure 8. Distribution of TPC-C local L3-cache data-load hits across segments of Buffer Pool of the eight-processor p690

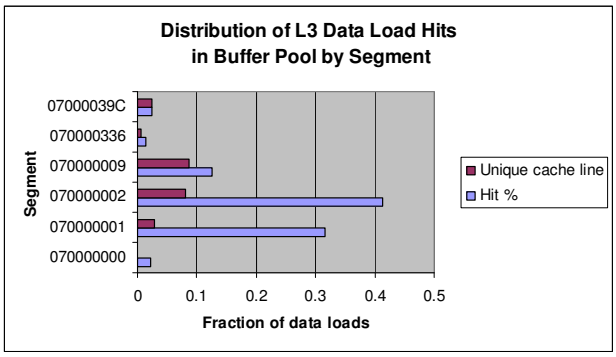

Figure 9. Distribution of TPC-C local L3-cache data-load hits across segments of Buffer Pool of the 32-processor p690

In these figures, we see that the majority of the hits reference a relatively small number of cache lines. In contrast, in the eight-processor system segment 0x070000005 and in the 32-processor system segments 0x07000039C, 0x070000336, and 0x070000009 appear to have been referenced in a much more uniform manner, i.e., the Unique_cache_line bar is a larger percentage of the corresponding Hit_% bar.

Note that because the 32-processor p690 consists of four MCMs, while the eight-processor system consists of two MCMs, the amount of physical memory available in the memory hierarchy of the 32-processor system is larger than that of the eight-processor system. As such, the amount of physical memory allocated to the buffer pool address region differs. Consequently, the number of segments touched, as well as those accounting for the majority of data-load references, during the 10-minute observation period is significantly larger in the 32-processor system.

Regions of the Address Space – Pages: Continuing to hone in on the suspect causes of performance degradation, we next take a closer look at a buffer pool segment frequently referenced in the local L3 caches of both the eight-and 32-processor p690s. Examining Figures 10 and 11, which plot the distribution of local L3-cache data-load hits across the pages of a TPC-C buffer pool segment for the eight- and 32-processor systems, respectively, we see very dense reference patterns. The dark-colored *Total_Loads bar* for a page represents the number of references to the page, while the *Unique_Cache_Line bar* indicates the number of unique cache lines referenced within the page.

Figures 10 and 11 show the pages of the 65,536-page segment that 70% of the L3-cache data loads reference. Although the range of targeted pages are different for the eight- and 32-processor p690s, in both cases approximately 200 pages are the source of these data loads. 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.

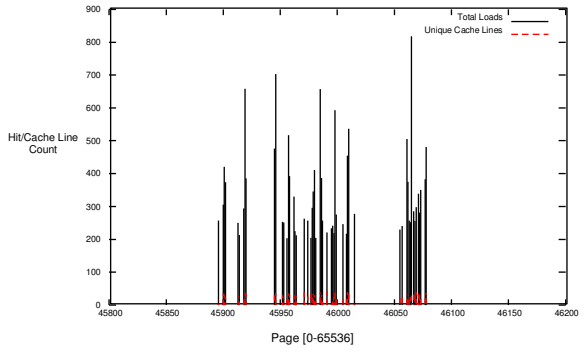

Figure 10. Distribution of TPC-C local L3-cache data-load hits across pages of a Buffer Pool segment of the eight-processor p690

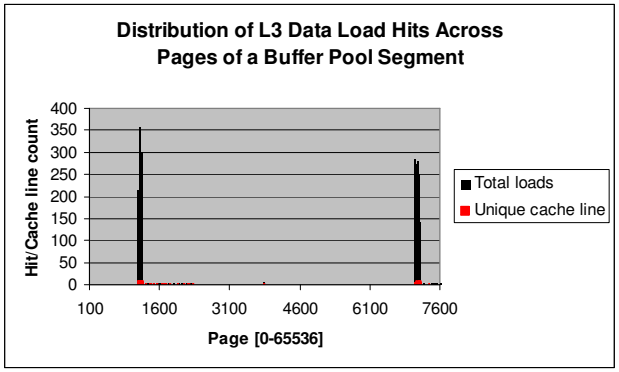

Figure 11. Distribution of TPC-C local L3-cache data-load hits across pages of a Buffer Pool segment of the 32-processor p690

Regions of the Address Space - L3 Caches: From the page-related data just presented, one would expect that within a page of the studied buffer pool segment we would see heavily-referenced cache lines. Figures 12 and 13, which illustrate the distribution of buffer pool data-load hits across cache lines of a hard hit page of the studied segment in the eight- and 32-processor p690s, respectively, show just that.

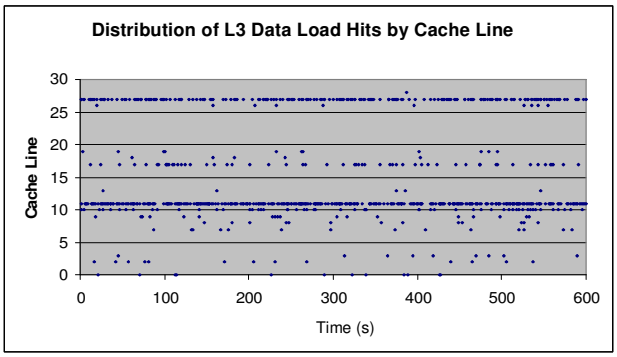

Figure 12. Distribution of TPC-C local L3-cache data-load hits across the cache lines of a hard-hit Buffer Pool page of the eight-processor p690

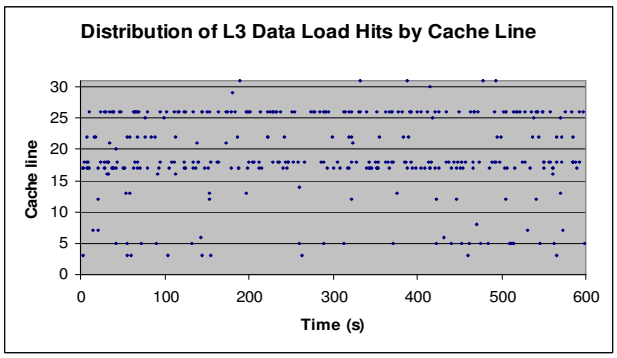

Figure 13. Distribution of TPC-C local L3-cache data-load hits across the cache lines of a hard-hit Buffer Pool page of the 32-processor p690

Referring to Figures 12 and 13, it is quite clear that in both the eight- and 32-processor systems only a handful of cache lines are the target of a majority (greater than 70%) of the local L3-cache data-load hits recorded during the 10-minute monitoring interval. Note that the y-axes of Figures 12 and 13 denote the 32 128-byte sectors (there are four sectors per 512-byte cache line) that comprise a 4KB section of a 16MB page of the buffer pool. In this way, using our performance evaluation framework, we can identify down to the cache-line level, sources of performance degradation in p690 systems.

Regions of the Address Space – Instructions: Additionally, our performance evaluation framework allows 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 3 were specified since they were potentially responsible for data loads resolved in the lower levels of the memory hierarchy.

The data retrieved from analyzing the event traces indicate that only two routines from the ones listed above had any notable impact on performance: disable_lock and simple_lock. The data referenced by these routines was retrieved from a remote cache (via an L3.5 hit) and 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.

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

Table 3. List of routines under analysis

6. Conclusions and Future Work

The work presented in this paper demonstrates the usefulness of p690 sampled performance monitor event traces and the power and flexibility of the performance evaluation framework that we developed to analyze them. Since the traces are stored in databases, they can be analyzed easily via queries to the database management system, producing reports and graphs that allow large amounts of information to be gleaned from the traces.

In the paper sampled event traces collected from eight- and 32-processor configurations of IBM's p690

executing TPC-C are analyzed to identify targets for performance improvement. The analysis focuses on memory subsystem performance, in particular, high-penalty L2-cache data-load misses, and shows how the framework can be used to ascertain reasons why a majority of these misses are resolved in p690 local L3 caches and main memory, which carry high load-hit latencies in comparison to L2-cache load latencies. The analysis is continually refined, identifying first the address regions most heavily referenced by L2-cache data-load misses, and then the most heavily referenced segments, pages, and cache lines.

Specifically, the analysis presented in the paper shows that the eight- and 32-processor data is very similar. Both indicate that the buffer pool and Data,BSS,Heap regions of the TPC-C address space dominate as the effective data address regions for data loads satisfied in local L3 caches and main memory. Furthermore, for those data loads satisfied in local L3 caches, the segments, pages, and cache blocks that comprise the buffer pool exhibit a rather dense distribution. In addition, using our framework, we are able to confirm 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.

Although the presented analysis falls short of identifying application sources of performance degradation, this analysis prompted modifications to the operating system's management of the buffer pool region, which yielded observable performance improvements. Due to the proprietary nature of the application and operating system used in this research, the specific sources of performance degradation and associated modifications that were implemented are not described in this paper. Currently, we are working on obtaining sampled event traces of publicly available benchmarks, such as RUBiS and Stock-Online [16], which will allow us to not only publicly identify targets for possible performance improvement, but also uncover and present the application-specific sources of performance problems and associated solutions via modifications to the application source code that enhance performance.

Future research also will attempt to quantify the accuracy of sampled event traces and will enhance the performance evaluation framework.

Acknowledgements

We want 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. In addition, we want to thank The Austin Center for Advanced Studies (ACAS), Carole Gottlieb, and Bret Olszewski for the faculty research awards 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 and 2004 internships which have helped further this research.

References

[1] J. Anderson, L. Berg, J. Dean, S. Ghermawat, M. Henzinger, S-T. Leung, R. Sites, M. Vandevoorde, C. Waldspurger, and W. Weihl, "Continuous profiling: Where have all the cycles gone?," *ACM Transaction on Computer Systems*, Vol 15, No. 4, November 1997, pp. 357-390.

[2] L. Barroso, K. Gharachorloo, and E. Bugnion., "Memory System Characterization of Commercial Workloads," *Proceedings of the 25th International Symposium on Computer Architecture*, Barcelona, Spain, 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*, Goteborg, Sweden, July 2001, pp. 266-277.

[4] M. Itzkowitz, B. Wylie, C. Aoki, and N. Kosche, "Memory Profiling Using Hardware Counters," CD *Proceedings of SC 2003*, Phoenix, AZ, November 2003.

[5] 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.

[6] 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, DC, USA, June 1993, pp. 22-31.

[7] T. Morgan, D. Villa, P. Teller, B. Olszewski, and J. Acosta, "L2 Miss Profiling on the p690 for a Large-scale Database Application," *Proceedings of the 4th Annual Austin CAS Conference*, February 2003.

[8] A. Nanda, K. Mak, K. Sugavanam, R. Sahoo, V. Soundararajan, and T. 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, November 2000, pp. 37-48.

[9] R. Portillo, D. Villa, P.J. Teller, and B. Olszewski, "Mining Performance Data From Sampled Event Traces", *To appear in the Proceedings of the 12th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Volendam, The Netherlands, October 2004.

[10] 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.

[11] 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, June 1997, pp. 112-122.

[12] D. Villa, J. Acosta, P.J. Teller, B. Olszewski, and T. Morgan, "A Framework for Profiling Multiprocessor Memory Performance", *Proceedings of the 10th International Conference on Parallel and Distributed Systems*, Newport Beach, CA, July, 2004, pp. 530-538.

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

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

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

[16] ObjectWeb, Open Source Widdleware. http://jmob.objectweb.org.

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

Table 2. TPC-C address space

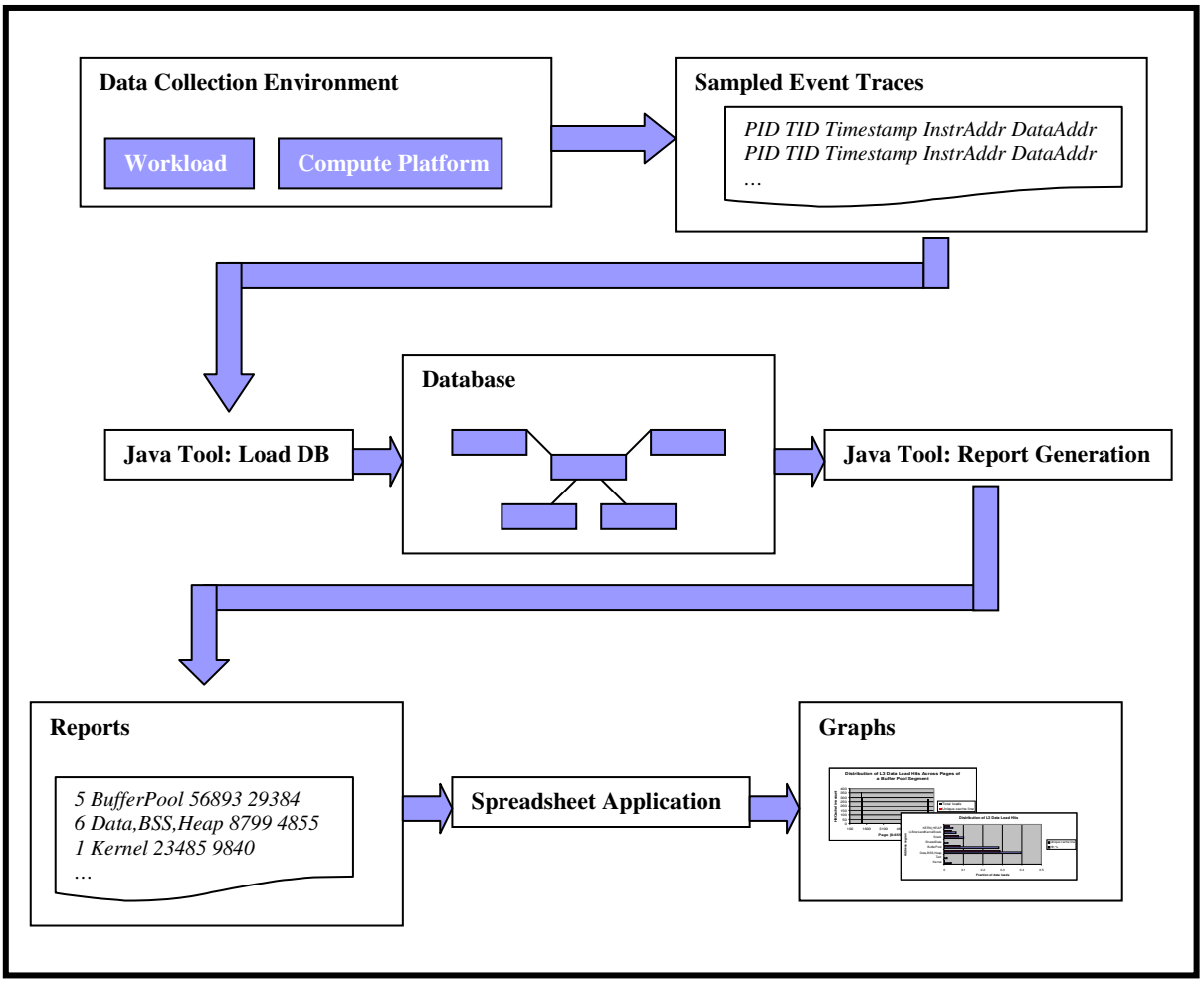

Figure 14. Performance evaluation framework