Characterization, Monitoring and Evaluation of Operational Performance Trends on Server Processor Hardware[*]

Ernest Sithole
School of Computing and
Information Engineering
University of Ulster
Coleraine - BT52 1SA
Co. Londonderry, UK.
e.sithole@ulster.ac.uk

Sally McClean
School of Computing and
Information Engineering
University of Ulster
Coleraine - BT52 1SA
Co. Londonderry, UK.
si.mcclean@ulster.ac.uk

Bryan Scotney
School of Computing and
Information Engineering
University of Ulster
Coleraine - BT52 1SA
Co. Londonderry, UK.
bw.scotney@ulster.ac.uk

Gerard Parr
School of Computing and
Information Engineering
University of Ulster
Coleraine - BT52 1SA
Co. Londonderry, UK.
gp.parr@ulster.ac.uk

Adrian Moore
School of Computing and
Information Engineering
University of Ulster
Coleraine - BT52 1SA
Co. Londonderry, UK.
aa.moore@ulster.ac.uk

Stephen Dawson
SAP Research Belfast
Queens Island, Titanic Quarter
Belfast - BT3 9DT
stephen.dawson@sap.com

ABSTRACT

Enterprise IT environments have seen a sharp growth in content use due to the popularity of on-demand data-intensive applications. In turn, the huge demand in content has spawned off major developments such as growth and distribution of computing nodes as well as the adoption of various implementation technologies. Given the complexity brought to the makeup of business computing environments in addressing the above-mentioned factors, the critical planning task of determining the appropriate infrastructure sizes for supporting firm Quality of Service (QoS) guarantees becomes a very challenging undertaking to fulfil. Benchmarking methods are widely employed in calibrating attainable performance in IT solutions, but these have the drawback of presenting output performance metrics as composite measurements that only give an end-to-end perspective. As an enhancement to benchmarking approaches, we explore the use of Performance Monitoring Counters (PMCs) in obtaining detailed operational performance of CPU and memory hardware. Performance Monitoring Counters (PMCs) are on-chip registers found on most modern processor hardware. We use PMC-derived measurements to validate cache performance trends that have been derived analytically, and in the course of validations, PMC data is also used to investigate the nature and character of surges in cache miss events,

which emerge as the memory load generated by runtime processes increases.

Categories and Subject Descriptors

B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; C.4 [**Performance of Systems**]:

General Terms

Measurement, Performance

Keywords

Performance Monitoring Counters, Benchmarks, Performance Evaluations

1. BACKGROUND

In order to cope with the increase in resource needs that are associated with emerging user applications, business IT solutions today are largely characterised by expansions in infrastructure domains such as the ever-increasing numbers of computing nodes and the dispersion of resources over wide geographic locations. In addition, other major developments such as the proliferation of implementation technologies are having just as significant an impact on defining the shape of the IT landscape. Given the interplay of these dominant trends, the following tasks which are vital to the infrastructure planning exercise are becoming increasingly complex to carry out: (a) establishing performance trends emanating from IT implementations (b) calculating the Service Level Agreement (SLA) parameters for service provisioning and (c) making decisions regarding the sizes of infrastructure deployments required to ensure that IT services that are delivered to user environments fall within satisfactory QoS thresholds.

In order to quantify the performance levels that can be provided by IT solutions, benchmarking methods are generally employed and the output metrics associated with spe-

---

cific implementations established through physical measurement. Some benchmark approaches specifically target the performance of application routines thereby leading to metrics being captured and presented as user-level performance indicators. Example benchmarks which are application-related include the Transaction Processing Performance Council (TPC) metrics for transaction processing and database operations on server hardware [6], and the SAP benchmarks for various workload configurations associated with enterprise application routines such as Sales Distribution (SD), Assemble-to-Order (ATO), Cross Assemble Time Sheet (CATS), Material Management (MM), Production Planning (PP), Financial Accounting (FI) and Human Resources Payroll (HR) services [23]. However, a major drawback with application benchmarks is the software-based overheads emanating from the utility programs that are set up to calculate the output performance metrics derived from measurements. Furthermore, the output metrics are essentially compound performance indicators that reflect the overall application response as perceived at end user level. Thus, the application benchmarks provide very little information about the response patterns of low-level infrastructure operations in relation to applied user loads.

In contrast to application-based benchmarks, hardware-specific benchmarks such as the Standard Performance Evaluation Corporation (SPEC) CPU measurements provide metrics that are used in the calibration of the performance of processor and memory hardware [4]. Essentially, the SPEC CPU metrics for both floating point and integer operations are presented as ratios, which are calculated from the comparisons of throughput and response times that are obtained from CPU and memory hardware of various server hardware kits. The normalisation of results is achieved through comparisons conducted with respect to corresponding metrics obtained from processor and memory hardware of the Sun SPARC Ultra Ultra Enterprise 2 reference machine that uses a 296 MHz UltraSPARC II processor [3][5]. In other approaches that employ server hardware benchmarks, the performance of CPU and memory architectures is considered in the context of parallel processing implementations as in [1][16][27], while hardware benchmarks for disk drive systems are considered in [25]. Even though it is possible to gain improved understanding of the operational capabilities of respective server hardware kits from hardware-related benchmarks, the output metrics provided are still not detailed enough given that they are calculated as lumped parameters derived from measurements of combined operational stages inside the server hardware. As an example, the SPEC CPU benchmarks are based on grouping the processor, caching and system memory functions into a single measured operation. As a result the SPEC CPU data provide very little insight as to the inner workings of the various components of server processor architecture or the resource consumption patterns at each of the respective operational stages in response to the varying quantities of user requests being handled by the server.

In order to further explore and understand the CPU and memory QoS that is provided by server nodes, this paper focusses on hardware operational performance and we propose a performance monitoring and evaluation strategy that uses data gathered by the on-chip registers called Performance Monitoring Counters (PMCs), which form part of the processor chipsets of most modern CPU hardware architec-

tures [9][10][11][13][20]. Unlike the SPEC CPU benchmarks, which provide high-level data in a format that is largely opaque in so far as the internal performance and resource utilisation trends occurring within the CPU and memory architecture are quantified, the PMCs provide rich sets of information that can help illuminate performance issues associated with the main components of the memory and processor hardware. By targeting appropriate PMC events for collection during performance monitoring sessions, it is possible to obtain performance data relating to the hardware functional stages such as processor cores, thread instances, multilevel caches and system memory elements as well the support operations for address translations for both data and instruction fetch operations [7][8][14][16]. Another advantage of using PMCs is that the process of invoking monitoring registers is largely non-intrusive since the data is obtained from on-chip counters whose operations are ordinarily separate from CPU cores. As such, no significant additional workload is applied on processor and memory operations apart from the interrupt operations that track specified events of interest.

It is also worth taking note of the fact in order to speed up application responses in many enterprise IT implementations, there is a rising trend in the adoption of in-memory database strategies [2][12][24][26]. The in-memory database techniques ensure that user data is initially transferred from the backend database into the system-memory buffer, where it is kept for the almost the entire duration of the computational operations. Hence with most of the compute operational processes being confined to CPU and memory hardware due to increased use of the in-memory computing functionality, the merits of our proposal to initially consider server performance in terms the CPU and memory hardware operation is further strengthened.

A further point to emphasize is that while the proposal to use hardware counters for performance monitoring offers greater insight into the trends associated with the internals of CPU and memory elements, our approach is meant to provide a complementary rather than an alternative strategy to the application and hardware benchmarks that have already been reviewed in this section. Thus, we consider the PMC-driven approach to be potentially useful as a follow-up strategy in those enterprise scenarios where the initial metrics from top-level measurements may indicate the need to address IT resource provisioning and configuration in the infrastructure if SLA and QoS parameters are to be protected. The PMC-based methods can be therefore employed to determine the specific operational elements in the hardware resource fabric where reconfigurations are required and can also indicate appropriate remedies based on the observed resource consumption patterns.

The rest of the paper is organised as follows: We present a brief overview of Performance Monitoring Counters by highlighting some the well-known packages for CPU performance profiling used in modern generations of processor hardware. In attempting to unpack and quantify the performance trends associated with the operational elements of the CPU and memory hardware, we consider the analytical models of Cache Miss Ratios and Processor Service Time. The Processor Service Time is the amount of time spent by a runtime process executing at CPU and interacting with the memory. The analytical characterisations of CPU Service Time trends and Cache Miss Ratios are derived from the

memory load imposed on the respective stages of multi-level cache hierarchy of the CPU architecture. The elements of the memory hierarchy are the sources, from which the run-time processes executing at the processor core requests obtain input data. The AMD CodeAnalyst profiler is then used to validate the assumptions and characterisations leading up to the models for performance trends. The paper concludes by considering additional investigations using PMCs to establish the source and character of sharp increases in cache miss events, which occur with a regular pattern as the memory load that is imposed on the CPU hardware architecture increases.

2. PMC-BASED STRATEGIES

As stated in the previous section, performance monitoring registers are becoming a standard feature of the CPU chipsets on motherboard kits from leading vendors of processor hardware [9][10][20]. Running as separate functional units from the processor core and memory elements on the computer hardware, the PMCs have the ability to track the occurrence of a comprehensive set of runtime events, which are associated with the executions of scheduled processes running on a machine. Key categories of hardware functionality, for which information can be obtained from monitoring registers, are as follows: (a) CPU cores and threads, (b) multi-level caches and system memory, (c) Input/Output operations, (d) Bus Transfers and (e) Special Instructions.

To take advantage of the functionality provided by PMCs, a number of software interfacing packages have been developed and these enable user-defined access to selected operational events of interest. The interfacing strategies involve working with two sets of performance Application Programming Interfaces (APIs), with one set targeting hardware-specific features of the performance monitoring registers and the other group of libraries enabling bindings to the operating system kernel. To ensure that the event monitoring procedure is non-intrusive, the performance APIs are of lightweight design and as such impose very little overhead whenever monitored events are counted, recorded, collated and presented to the user environment for interpretation. Some of the popular middleware profiling tools that have system-wide ability to access performance monitoring registers include the OProfile [17] and Performance Application Programming Interface (PAPI) [27][28] packages. Other performance profiling packages such the Intel VTune [16][18] and Advanced Micro Devices CodeAnalyst [9][14] are optimised to collect detailed event data on vendor-specific hardware, while the Hardware Performance Monitoring Counter (HWPMC) is designed for FreeBSD platforms [19].

In the initial set of experiments that are considered in this paper, our interest is primarily on establishing the trends in the processor service times as the loading levels that are presented on the CPU hardware vary. Given that the durations of the effective CPU service times whenever scheduled runtime processes execute on the CPU are dependent on memory access times, we focus on the events associated with CPU core and thread operations as well as cache and system memory accesses. We use the AMD CodeAnalyst as a profiling tool to collect the measurements data. Our choice of CodeAnalyst in the experiments was largely influenced by the need to obtain detailed results from the AMD Athlon server hardware on which the performance monitoring runs were carried out.

3. CACHE MISS TRENDS FOR FULLY ASSOCIATIVE CONFIGURATIONS

We consider the theoretical characterisations of Processor and Memory hardware performance by recalling basic expressions of runtime performance on CPU and memory hardware architectures of computing implementations:

$$T_{CPUService} = T_{CPUExec} + T_{MemStall}, \qquad (1)$$

where the CPU Service Time, $T_{CPUService}$ is the effective duration of CPU occupancy for an active runtime process using the CPU resource. The CPU Service Time, is made up of the Memory Stall Time, $T_{MemStall}$ that is consumed in fetching data from memory and the Processor Execution Time, $T_{CPUExec}$ which is spent in performing the computational operations within the CPU cores' functional elements, the Arithmetic Logic Unit (ALU), Floating Point Unit (FPU) and register assembly.

In term of processor cycle consumption, CPU Service Time can alternatively be expressed as:

$$CycleTime(ExecutionCycles + MemStallCycles) \quad (2)$$

The Average Memory Access Time, $AMAT$, for an N-level cache hierarchy is given by the general expression:

$$AMAT = T_{CacheHit1} + \sum_{K=1}^{N} (MissRatio_K)(T_{MissPenaltyK})$$
$$(3)$$

The Cache Miss Ratio is the number of cache miss events expressed as a fraction of total cache access operations. In terms of processor cycle consumption and cache performance, memory stall time can be expressed as follows:

$$MemStallCycles = MemAcesses(MissRatio)(T_{MissPenalty})$$
$$(4)$$

Next, we consider the cache miss ratio as a function of user load by making a number of assumptions about the key parameters having a bearing on the performance patterns on an L1 (D) caching system. Note that the cache performance trends that we consider in our analysis are derived from capacity misses in a fully-associative cache configuration. According to the fully-associative cache mapping approach, the main memory blocks occupied by a process image can be mapped and transferred to a cache slot or cache line *anywhere* in the cache memory. The lines or slots are uniform subdivisions in the cache and main memory stages, and the partitions are usually 32 or 64 bytes in size. Other contributory factors to cache miss ratios such as compulsory and conflict misses are considered as negligible offset parameters in the analytical treatment. Also noteworthy in our models is the assumption that all the user requests have homogeneous characteristics i.e. their memory requirements and processor execution time needs are considered uniform. Yet another simplification we make for the initial the model is that the available memory and CPU time on the server are unaffected by competition from background job routines that may also be running on the server. We define the following parameters:

- Average memory needs per user = m bytes

- Number of simultaneous users = N

- Maximum simultaneous users before L1 Cache Capacity Misses occur = $N1max$

- Cache Memory Capacity $= M$ bytes

Thus, the total user memory needs $= Nm$ bytes and the excess memory requirements $= (N - N1max)m$. Assuming that the risk of cache misses is spread across all the active N users whenever cache memory capacity is exceeded, then the following expression is derived:

$Cache\ Miss\ Ratio = \frac{N-N1max}{N}$.

For a single level cache system, Equation (3) can therefore be modified to take into account user load so that the equation for *Average Memory Access Time* becomes,

$$AMAT = CacheHitTime1 + (\frac{N - N1max}{N})(PenaltyTime1).$$
(5)

Since CPU Service Time is equal to the sum of CPU Execution Time and AMAT as in Equations (1) - (4), the processor service time can thus be expressed as a function of user load by the following expression:

$$ExecTime + CacheHitTime1 + (1 - \frac{N1max}{N})(PenaltyTime1).$$
(6)

In considering the data access trends for a level-two cache system on the AMD Athlon hardware, the CPU cycles for memory access per user process are derived from the summation of Cache Hit Times of the Level 1 and Level 2 stages and the Memory Stall Times associated with cache accesses in the main memory whenever consecutive cache misses occur in the L1 (D) and L2 stages.

Total CPU Service Time = Processor Execution Time + CacheHitTime1 + (MissRatio1)(MissPenalty1) + (MissRatio2)(MissPenalty2).

Also, the data access times can be expressed as follows:
Miss Penalty Time1 (for Level 1 Accesses) $=$ *Cache Hit Time2* (for data fetched at L2 cache stage) $+$ *Miss Penalty Time2* (for Level 2 Cache Accesses), where the Miss Penalty Time for Level 2 Cache accesses is the same as the Hit Time for main memory accesses associated with the data that is fetched on System Memory stage. Assuming cache capacity misses for the L1 (D) and L2 stages, where exclusive cache policies are coupled with fully-associative mapping of cache stages to main memory, then the cache miss ratio for the L2 stage are expressed as follows:

$Cache\ Miss\ Ratio$ (Level 2) $= \frac{N-N1max-N2max}{N-N1max}$.

Therefore, Average Memory Access Time, $AMAT$ is equal to:
$CacheHitTime1 + (MissRatio1)(PenaltyTime1) + (MissRatio2)(PenaltyTime2).$
In exclusive caching policies, only a single copy of a cached item can be kept in the L1 and L2 cache memory stages i.e. the cached copy is in either the L1 or L2 cache, not in both caching levels. Inclusive caching policies on the other hand permit both the L1 and L2 cache memories to contain same cached data.

The Average Memory Access Time for L1 and L2 Cache thus becomes:
$AMAT = CacheHitTime1 + (PenaltyTime1)\frac{(N-N1max)}{N} + (PenaltyTime2)\frac{(N-N1max-N2max)}{N-N1max}$.

It is worth emphasising that an important assumption made in the analytical derivations of the expressions for the Average Memory Access and CPU Service Times is that the application requests being responded to by the CPU and memory hardware require the same instruction to be executed at runtime.

4. CACHE MISS TRENDS FOR N-WAY SET ASSOCIATIVE CONFIGURATIONS

With N-Way set associative mapping, the space on each of the memory elements of the CPU hardware architecture is subdivided into slots or sets, which are then allocated or mapped according to specific ratios that are fixed between the Main Memory and L2 Cache, and between the L2 and L1 cache stages. While the proportional mechanism fixes the ratio of slot allocations between the memory stages for the N-Way set associativity approach, the multi-way mapping feature introduces a degree of flexibility by a factor of N to the number of slots where a cached copy can be placed in whenever it is obtained from higher caching levels. Thus, the N-way set associativity approach is a compromise between the direct mapped caching technique, which has a rigid allocation to a single cache block and the fully associative mapping strategy that permits all cache blocks to be used. Figure 1 shows the proportional mapping employed between L1 and L2 cache, between the L2 and main memory stages of CPU hardware architecture that is based on N-Way cache associativity.

We make the following assumptions for the executing processes and the configuration of the memory hardware. Let Nm be the required memory for the input data parameters to be used by the running processes that are generated by N users, with each user-generated process consuming m bytes at runtime. Let α be the ratio of the memory mapping of cache slots between L1 and L2 cache stages, and β the ratio of the cache slot mapping between the L2 Cache and Main Memory. Therefore, the allocations of memory in the L1 and L2 caches for N user initiated processes based on associative mapping are $\alpha\beta Nm$ and βNm respectively. Similarly, the Cache Miss Ratios for L1 and L2 cache stages are $\frac{(Nm-\alpha\beta Nm)}{Nm}$ and $\frac{(Nm-\alpha\beta Nm-\beta Nm)}{(Nm-\alpha\beta Nm)}$ respectively. Assuming perfect locality of reference for the data accesses on the L1 and L2 caches, the miss ratios therefore simplify to $(1-\alpha\beta)$ and $\frac{(1-\alpha\beta-\beta)}{(1-\alpha\beta)}$ respectively.

If we introduce the average locality factors for the data access patterns to L1 and L2 caches which are assumed to be λ and μ respectively, then the cache miss ratios for L1 and L2 stages are $(1-\lambda\alpha\beta)$ and $\frac{(1-\lambda\alpha\beta-\mu\beta)}{(1-\lambda\alpha\beta)}$.

The Average Memory Access Time for associative mapping becomes,
$AMAT = HitTime1 + (1-\lambda\alpha\beta)(PenaltyTime1)$
$+ \frac{(1-\lambda\alpha\beta-\mu\beta)}{(1-\lambda\alpha\beta)}(PenaltyTime2)$.

The Average Processor Service Time for associative mapping then becomes,
$T_{CPUService} = ExecTime + HitTime1 + (1-\lambda\alpha\beta)(PenaltyTime1)$
$+ \frac{(1-\lambda\alpha\beta-\mu\beta)}{(1-\lambda\alpha\beta)}(PenaltyTime2)$.

5. MAIN CONSIDERATIONS FOR EXPERIMENTAL SCENARIOS

The internal structure of the AMD Athlon 64X2 processor hardware that is used in the experimental studies is made up of the following functional components: (a) one dual core processor, (b) one 64-Kbyte Level 1 Data Cache 1 i.e. L1
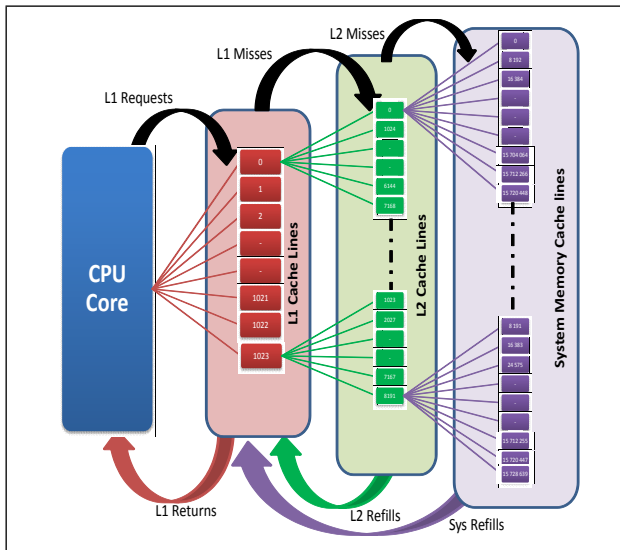
Figure 1: Proportional allocation of memory slots according to associative mapping.

(D) Cache and one 64-Kbyte Level 1 Instruction Cache 1 i.e. L1 (I) Cache per core (c) one 512-Kbyte unified Level 2 cache that contains both data and instructions and (d) 960 Mbyte Double Data Rate (DDR2) DRAM system memory. The access operations on the L1 D-cache and I-Cache units of each CPU core are supported by the page-mapping information that is contained in the data and instruction translation look-aside buffers (TLBs) respectively. The Victim Buffer (VB) unit also provides important support functionality for accelerated exclusive-cache operations by holding recently evicted data from the L1 cache as they await transfer to the L2 cache stage. The D-TLB and I-TLB are 24 and 32 entries in size and VB can hold up to 16 cache lines, where each cache line on the AMD Athlon 64X2 architecture is equal to 64 bytes.

To enable the study of the main CPU performance trends (such as Processor Service Times, Average Memory Access Times and the incidence of Cache Misses) with respect to user-generated loads being handled by the CPU and memory hardware, the CodeAnalyst's Event-based Profiling feature is invoked and the relevant hardware operations are selected for capture by the PMCs. The following event categories summarise the hardware operations of interest that are selected for collection in the measurement profiles defined in CodeAnalyst:

- **Data Cache Events** encompassing Total Data Cache Accesses and Misses, Level Cache 1 Refills from System and L2 Cache memory, Cache Lines Evictions to L2 cache and System memory and TLB Misses for L1 and L2 cache stages.

- **L2 Cache and System Interface Events** that include L2 Requests from L1 Misses and L2 Misses.

- **Execution Unit Events** that take into account Retired CPU Instructions and CPU Clocks that are not halted.

- **Load/Store Unit Events** that capture the number

of operational cycles which the Load and Store Queue Unit is filled to capacity with miss requests from and those waiting to refill the L1 caches.

It is worth emphasising that some of the events considered in the paper such as Refills from L2 and System provide compound metrics that will be investigated at sub-event level [15] when experimental runs are generated. Based on the selection of abovementioned operational events, key experimental scenarios were defined to investigate and study these phenomena:

- **Trends in L1 (D) and L2 cache miss ratios**. The results from the PMC-based event monitoring are interpreted and, comparisons are made with analytical models that were developed for estimating cache performance characteristics for both exclusive and inclusive caching policies.

- **Trends in the durations of CPU service times**. The results data obtained is used to calculate Clock cycles per Instruction (CPI) and IPC values and, comparisons are made with derived characteristics from theoretical models.

- **Performance penalties arising from worst-case caching scenarios**. The performance degradations that are considered, occur due to the enforcement of Exclusive cache policies between the L1 (D) and L2 stages as well as the internal cache-size settings of VB units.

In order to obtain the data points for the results that are presented in this paper, a C++ program is used in the experiments. The input load levels on the processor cores are varied according to the number of specified threads that send service requests to the CPU. Thus, each executing thread is used as the basic proxy for user load in our experiments and the memory consumptions levels, which are associated with each user thread are specified in terms of the number of input data parameters accessed for computational operations at the processor core. Detailed consideration of the experimental runs that were conducted for the study of main scenarios is featured in the succeeding sections.

6. EXPERIMENTAL INVESTIGATIONS FOR N-WAY ASSOCIATIVE CACHES

The definitions for the experimental scenarios we consider are based on generating workloads from a C++ application for execution on the processor and memory hardware of the AMD K8 Athlon architecture. The performance data pertaining to the operational events that arise from the runtime execution of application requests is captured by the AMD CodeAnalyst tool.

The structural makeup of the test program is largely based on the configuration of the SAP Sell-from-Stock routine, which is made up of Process Components that generate specific runtime requests to the server processor in terms of dialog steps [22]. Typically, six process components or transaction entities comprise the enterprise application routine and, within each transaction a number of dialog steps are initiated for execution at the server hardware. Figure 2 shows the basic implementation of the business process component in enterprise application solutions. The operations
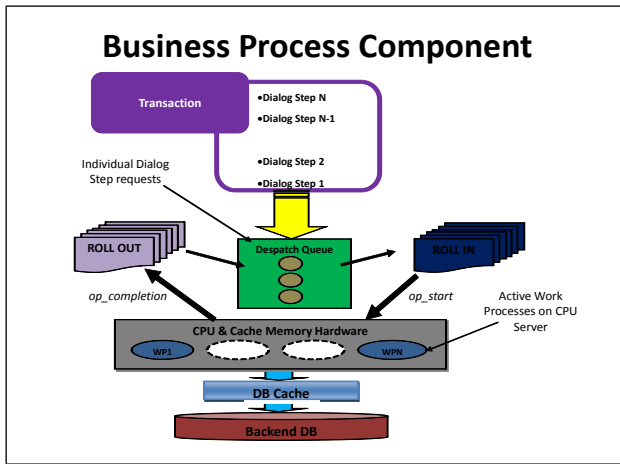
Figure 2: Basic makeup of Process Components in enterprise application implementation.

Figure 3: Data access across arrays of double precision parameters in 25 Matrices.

performed by the dialog step executions range from creating order items, retrieving and displaying order information and updating customer data. Since our objective is primarily focussed on studying the performance of processor and memory hardware, one departure we make from actual SAP Sell from Stock's configuration in our workload definitions is that the user requests are characterised as identical work items, all of which are CPU and memory intensive in their resource consumption characteristics. The Work Process (WP) routine, which is the runtime entity that actually executes the scheduled dialog steps received from multiple users in the real implementation, is represented by a multi-threaded program and each thread instance represents a single user. In order to represent the Sell from Stock configuration where each user has its own set of data [21], the matrix definitions that are specified for the test program for our experiments ensure that each user thread has a specific set of input data that it accesses during execution.

As an initial focus for the experimental measurement in CodeAnalyst, we attempt to validate the theoretical performance trends that have been derived for cache implementations which employ N-way associative mapping strategies in the allocation of memory space on the L1 and L2 caching stages. The specific trends that enable the validity of the analytical trends to be determined are: (a) Variability of Cache Miss Ratios and Clock cycles Per Instruction (CPI) rates with respect to the quantity of user generated workloads and (b) impact of data locality on the Cache Miss Ratios and CPI rates.

The load on the CPU and memory architecture is defined in terms of the number of users and resource consumption that is associated with each user entity. The number of active users is determined by the number of threads that will request double-precision parameters at the CPU cores during runtime. Each user instance is defined in the C++ program such that it generates data requests to the memory and then performs arithmetic additions of the obtained double precision parameters, which will have been fetched from across 25 matrices. During the program execution each user thread is locked to a specific row of each of 25 matrices and the data fetch operation advances through the
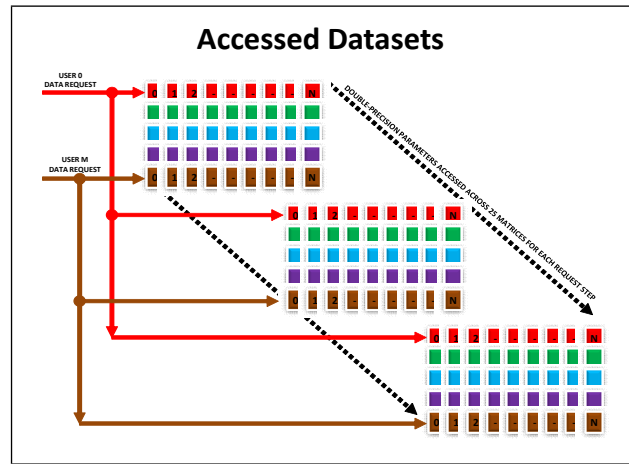
columns of double-precision data, which will then be added together before updating the running total. Figure 3 shows the basic access scheme that is employed in accessing the double-precision values by user threads from 25 matrices. The memory load on the Level 1 (Data) cache memory is varied by uniformly increasing the number of users between 1 and 20 in order to come up with scenarios that generate data points for validating the analytical characterisations that have been defined for LI (D) and L2 caches' request and miss trends. During the measurement epoch, each of the user instances cycles through the same set of operations (involving data accesses, additions and updating the running totals of double-precision parameters). The user request cycles are punctuated by delays that have been defined in the program by inserting a sleep time duration of 100 milliseconds.

The principal definitions in CodeAnalyst involve (a) making configurations for capturing specific CPU and memory operational events that are of relevance to the validation of L1 and L2 cache trends, (b) setting the duration of the measurement profile, (c) specifying the initial delay time from program launch before CodeAnalyst monitoring tool can begin recording the specified hardware event metrics and (d) specifying the launch of the C++ test program from CodeAnalyst so that its runtime execution along with the CPU hardware performance monitoring can take place concurrently. A delay time of 20 seconds and a profile measurement period of 25 seconds are specified in the CodeAnalyst's session settings facility. The following hardware events are selected for monitoring:

- Data Cache Access (for both Levels 1 and 2)

- Data Cache Misses (for both Levels 1 and 2)

- Requests to L2 (Cache)

- L2 Misses

- CPU Clocks (Not Halted)

- Retired Instructions

In order to isolate the data access events that only reflect L1 (D) operations in the measurements that are captured in CodeAnalyst, the sub-events relating **Instruction Cache (IC) Fills** and **Translation Lookaside Buffer (TLB)** fills are filtered out from the event specifications of L2 Requests and Misses. Additionally, in order to obtain a clearer indication of the number of data access and miss events related to L1 (D) operations, the following hardware events are selected to provide a complementary set of metrics to the ones mentioned above:

- Data Cache Refills from Level 2 or System Memory

- Data Cache Refills from System Memory
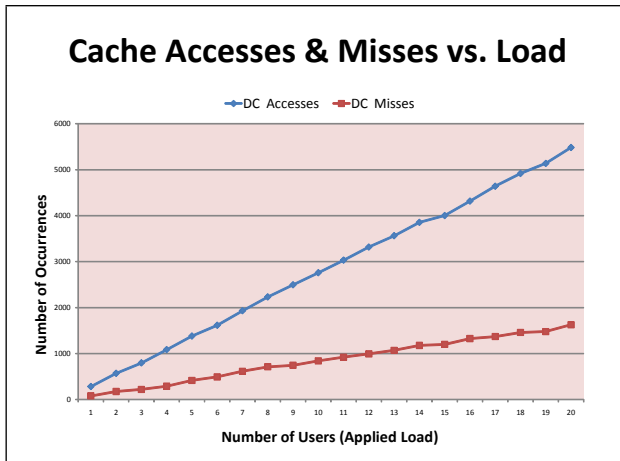
Figure 4: Data Cache Access and Miss trends with respect to User Load.

The trends in Figure 4 show that there is a linear increase in both Data Access and Miss Events at the L1 (D) as the number of user generated requests rises uniformly from 1 to 20. The linear increases are conformity with the analytical characterisations derived for miss ratios that occur in N-Way set associative caches as considered in Section 4. Similar trends for Level 2 cache requests and misses are presented in Figure 5, which is also in agreement with theoretical derivations of N-Way cache set associativity.

The actual cache miss ratios calculated for the L1 (D) and L2 stages are shown in Figure 6 and, in harmony the analytical expression developed in Section 4, it can also be established on the graphs that the average Instruction per Cycle ratio for all load scenarios remains at approximately the same level. While the cache miss ratio level for the L2 cache is fairly constant, the miss events are so high that on average they constitute around 45 % of L2 Requests. The poor cache performance at the L2 stage is accounted for by the exclusive caching scheme that is employed by the AMD Athlon hardware on which the monitored runtime events execute. Since the exclusive caching policy seeks to maximise the cache space on the hardware, only a single copy of data can be kept in the cache memory area; in other words a cached copy can only be in either the L1 or L2 memory. The enforcement of the exclusive caching policy involves data copies being exchanged between the L2 hits and L1 evictions, thereby resulting in the spatial locality
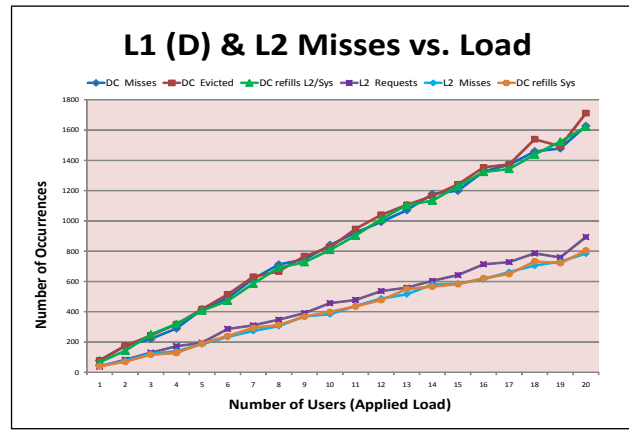

Figure 5: Data Cache Access and Miss trends with respect to User Load.

patterns that were in the L1 cache lines during the initial input transfers from System memory being lost whenever evicted data is moved up to the L2 memory.
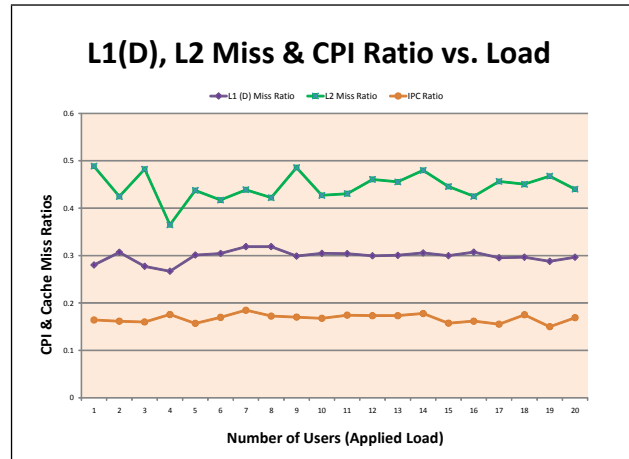

Figure 6: Data Cache Miss Ratio and IPC Rate trends with respect to User Load.

In order to further establish the internal consistency of the measurements of cache memory operations, additional metrics were captured for comparisons with the principal cache events that are used in our experimental studies. Figure 5 shows that L1 Data Cache misses are approximately equal to L1 evictions of cache lines into L2 cache stage. The number of cache refills from L2 that execute in response L1 Data Cache misses is almost equal to the number of cache line evictions from the L1 memory. From the Level 2 cache requests following L1 misses, requests are triggered to the System Memory in event of L2 cache misses and as shown in Figure 5, the System Memory Refills, which are in response to unfulfilled data requests in the L2 cache are nearly equal to L2 Misses.

6.1 Impact of Data Locality

Another factor that was considered in the theoretical derivation of cache miss trends is the impact of locality distance

between the data parameters that are successively fetched the executing program which is monitored. We study the impact of spatial locality over the cache memory by considering the same basic configuration for data access shown in Figure 3 but the respective scenarios involve data parameters being fetched across 25, 50 and 100 matrices. For
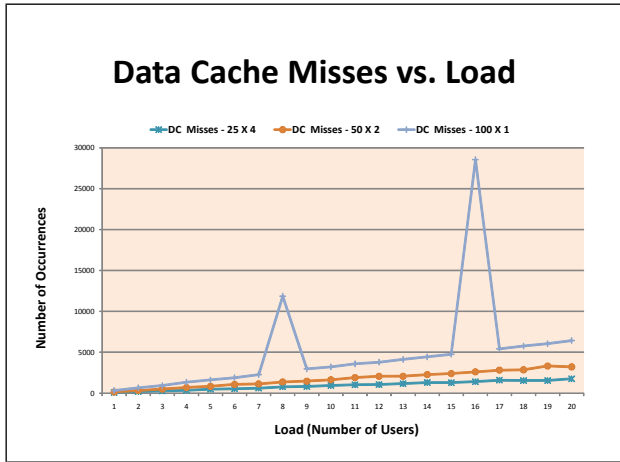

Figure 7: Data Cache Miss Trends for accessing 100 double-precision parameters.

each monitored program operation the runtime execution is made up of arithmetic additions of 100 double-precision values that are obtained from the same row on each matrix. Thus, data access of 100 double-precision parameters for the 3 scenarios featured in the study of data locality involves obtaining input parameters in the following ways per request step: (a) 4 consecutive values on each matrix row of the 25 matrices, (b) 2 consecutive values on each matrix row of the 50 matrices and (c) a single value on each matrix row of the 100 matrices. The results shown in Figure 7 present linear trends in cache misses that emanate from the uniform rise in the number of users and the corresponding increase in the allocated cache memory space that is fixed by the N-Way set associative mapping of the assigned size of the Data Segment (DS) in system memory. The manner of accessing the input data for the three featured scenarios determines the respective magnitudes of spatial locality associated with obtained data. In the case of 25 matrices the data accesses have the lowest incidence of cache misses since input operations take advantage of successive data parameter on each row before launching external requests to L2 or System memory in the event of cache misses. The cache misses get progressively worse as the number of accessed consecutive values is lowered to two and one for 50 and 100 matrices respectively.

The complementary graphs in Figure 8 also confirm low data locality compromises cache performance. The two featured scenarios are based on accessing 75 parameters from 25 matrices by obtaining 3 consecutive values and accessing 75 matrices by getting a single value from each row. Also shown in both Figures 7 and 8 is the regular occurrence of huge surges in cache misses that even exceeds the data cache accesses for certain load scenarios as number of users continues to rise. Detailed consideration of the phenomenon is provided in the following subsection.
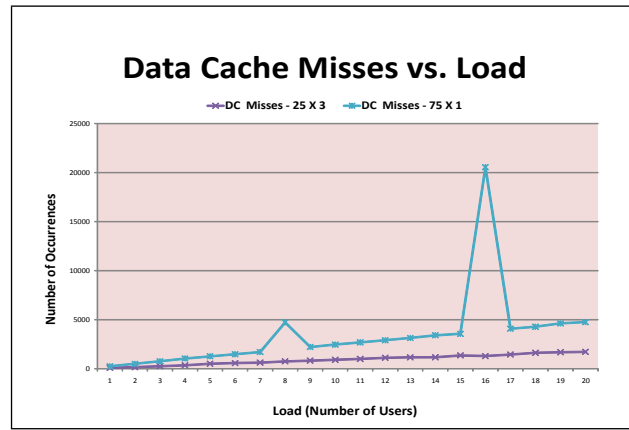

Figure 8: Data Cache Miss Trends for accessing 75 double-precision parameters.

6.2   Cache Miss Spikes in Detail

The following definitions were made for detailed experimental investigations of the spike phenomenon in cache misses as they occur during data access operations at L1 and L2 stages:

- Number of users - 160 (Fixed).

- Number of accessed double precision parameters - 4 to 200 (Increasing uniformly in steps 4).

- Number of matrices across which the double precision parameters are fetched - 50.

On Figure 9 one of the featured trends is the linear increase in the number of L1 Data Cache misses as the number of requested data parameters by each user rises uniformly. In the case of L1 (D) cache miss events, it can be seen that two component characteristics contribute in shaping the pattern shown on Figure 9, one of which component trend being the linear rise in cache misses as the number of request data parameters is increased. As presented in Figure 4, the linear trend follows the analytical relationship that is defined by basic expression, $(1 - \lambda\alpha\beta)$, for quantifying L1 cache miss ratios associated with homogeneous memory load in CPU cache systems that employ N-way set associative mapping strategies for allocating cache memory space. The other significant trend characterising the L1 Data Cache misses is the regular occurrence of sharp rises in cache miss events at intervals of 8, 16 and 32 parameters. These observed sets of cache miss spikes are a superimposition to the linear trend of L1 (D) misses and they thus combine to give the total L1 Data Cache misses shown by Figure 9. The number of total L1 Data Cache refills from the L2 Cache and System memory levels are nearly equal to the cache miss event in the L1 (D) cache and this relationship is in harmony with the configuration shown in Figures 1 and 11, where as a result of employing exclusive cache policies on the AMD hardware, the L1 (D) cache misses are satisfied by the cache refills from both the L2 cache and system memory.

The number of L2 cache requests follows the same pattern associated with L1 Data Cache misses of regularly occurring spikes that are superimposed onto the linear growth to the number of request events to the L2 cache. It can also be
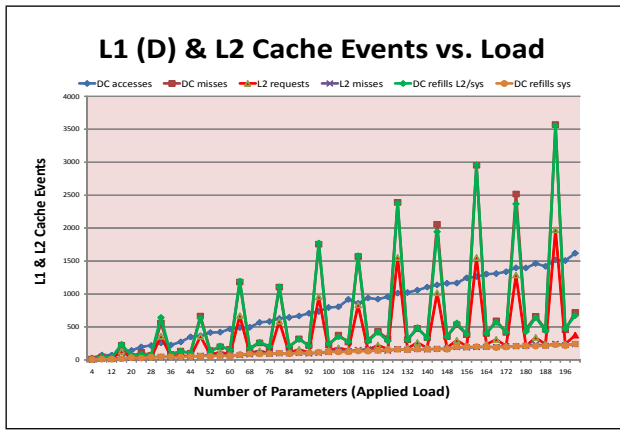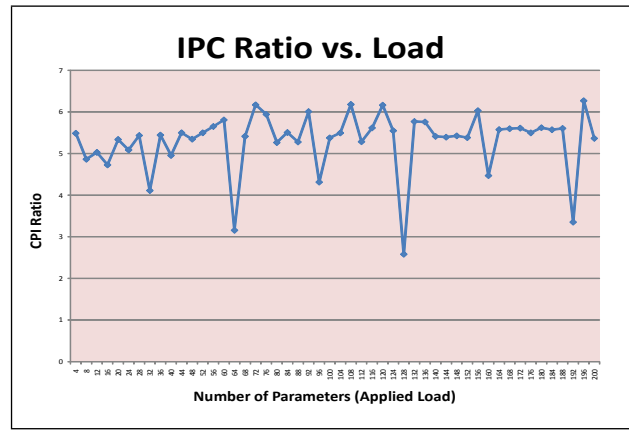
Figure 9: Trends in L1(D) and L2 Cache Events vs Load.

Figure 10: Impact of Cache Miss Surges on Performance.

observed that L2 requests are significantly lower that the L1 Data Cache misses. The difference can be attributed to the fact that some of the L1 (D) misses are satisfied in the Victim Buffer entity, a small cache memory unit, that sits between the L1 and L2 cache stages so that in line with exclusive cache policies, the exchange of data copies between the two cache levels is facilitated whenever an L1 miss results in a cache hit at the L2 stage. Reference can be made to Figure 11 to establish the functionality offered by the VB. Thus, it is only in the case of VB miss events that L1 (D) misses are eventually directed as requests to the L2 cache.

The L2 cache misses to System memory result in refills to the L1 (D) cache that are nearly equivalent to the number of cache misses that occurred at L2 stage as shown in Figure 9. It can also be observed that no spikes in cache miss events occur at the L2 cache stage since there is no attached VB between the L2 and System memory elements. Just as in the case presented in Figures 5 and 6, there is a linear trend in L2 cache misses as shown in Figure 9. This linear rise in cache miss events follows the analytical relationship, $\frac{(1-\lambda\alpha\beta-\mu\beta)}{(1-\lambda\alpha\beta)}$, derived for cache miss ratios associated with homogeneous loads on memory hardware that employs N-Way set associativity techniques for allocating cache memory space at L1 and L2 stages.

The impact on program performance of surges in L1 (D) cache misses are shown in Figure 10, where the retired Instructions per Cycle (IPC) ratio falls from the average of 5.5 to around 4.3 when the number of data parameters requested per user thread is either 32, 96 or 160. The IPC ratio further dips to an average of 2.9 whenever the requested data parameters are equal to 64, 128 or 192. The impact of cache miss surges shown in Figure 10 is in contrast to the trend presented in Figure 6, where the average CPI ratio of 0.18 (or IPC of 5.5) is maintained for all the featured data points. The difference is due to the fact the experimental scenarios, from which data points of Figure 6 were obtained, are unaffected by sharp rises in cache miss ratios.

In order to understand the spike phenomenon in DC miss events more fully, the component operational events at the L1 data cache are isolated and analysed individually. Since the components of L1 Data cache miss events cannot be broken up and traced directly from the CodeAnalyst results
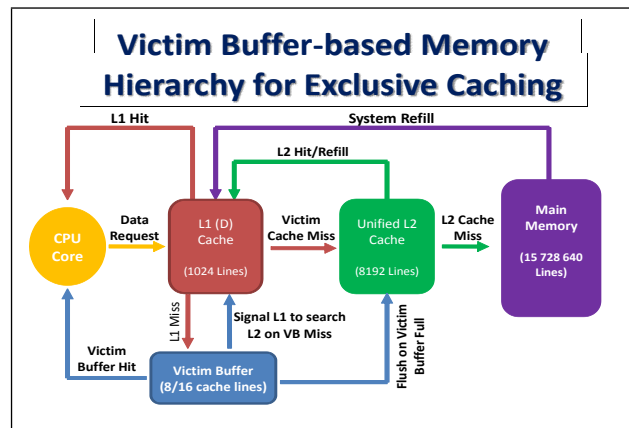

Figure 11: Exclusive caching using the Victim Buffer.

data, use is made of the measurements data of L1 Data cache refills from the L2 Cache and System memory, which consists of sub events that can be filtered for individual analysis. As has been stated in connection with the graphs in Figures 5 and 9, L1 DC Misses are equivalent to Cache Refills from L2 cache and System Memory; hence the components of L1 cache misses can be determined from the sub events of total cache refills into the L1 (D) memory. The sub events that make up the L2/System refills are follows:

- Modified-State Lines in L2
- Owned-State Lines in L2
- Exclusive-State Lines in L2
- Shared-State Line in L2
- DC refills from System Memory

As already stated in connection with Figure 9, the magnitudes of cache miss spikes follow linear pattern, increasing at regular intervals of 8, 16, and 32 parameters. From the additional measurements to determine the components of the L1 Data cache refills, which are associated with the cache

miss surges, the results in Figure 12 show that it is the refills of Exclusive-State Lines from the L2 cache for MOES cache coherency protocols that account for spikes. Although not shown in the graphs, the measurements conducted in Code-Analyst confirm that the refills into L1 (D) associated with the other MOES cache coherency states (i.e. the Modified, Owned and Shared states) bear very little or no connection to the steep cache miss rises, and as already highlighted from considering Figures 5 and 9, the refills into L1 (D) cache from System memory show a steadily increasing trend with applied user load.
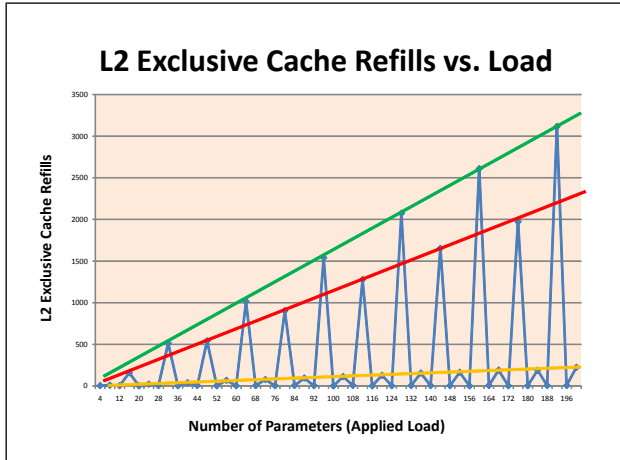

Figure 12: L2 Exclusive Cache Refill Events vs Load.

Another noteworthy feature that is presented in Figure 12 is that three separate spike trains can be traced. The magnitude of cache misses for each train approximate to a straight-line graph that can be produced through the origin as shown by the three lines obtained from joining the cache miss peaks shown in Figure 11. The magnitudes, $f(n)$, of the exclusive cache cache miss events as well as the regularity of cache miss occurrences with respect to number of accessed parameters, n, is summarised by the relationship(s) shown below.

$$f(n) = \begin{cases} 0 & \text{if } n = 4r \text{ with } r \text{ a positive odd integer} \\ M_1 n & \text{if } n = 8r \text{ with } r \text{ a positive odd integer} \\ M_2 n & \text{if } n = 16r \text{ with } r \text{ a positive odd integer} \\ M_3 n & \text{if } n = 32r \text{ with } r \text{ a positive integer} \end{cases}$$

The respective gradients, M_1, M_2, and M_3 are 1.12, 11.20 and 16.23 for the data access patterns considered in the particular scenarios whose results are shown in Figure 11. The peaks in cache misses associated with M_3 are responsible for the most significant performance knocks as shown by lowest IPC ratios in Figure 9. Given the deterministic nature of the cache miss spikes, it thus possible to determine the points where a severe fall in performance can occur. However, further study is required in order to fully account for the magnitude of the surges that punctuate cache miss trends as the load on the cache memory increases.

7. CONCLUSION AND FURTHER WORK

We started by highlighting the challenge of characterising performance in on-demand computing environments given the complexity of the current makeup of IT infrastructures.

As an enhancement to current approaches that predominantly employ benchmarking methods, we proposed the use of PMC-based measurements, which can monitor and capture detailed low-level operational events in the CPU and memory hardware. We used PMC data to validate key trends for L1 and L2 Cache Misses and Processor Service Times that were derived analytically for homogeneous loads executing on cache hardware that employs N-Way mapping for allocating memory space. While the measurements are consistent with main trends derived for L1 and L2 cache performance, specific cache miss rates and CPU Service Times can only be determined if the exact locality factors associated with the data access patterns for each runtime scenario are known.

Another aspect that was considered in the experiments was outbreak of spikes in the cache misses as the load on cache memory levels increases. Although there is consistency to the sharp increases in the cache miss events in terms of regularity and magnitudes of cache misses, further event monitoring is required to gain more clarity on the inner workings of the cache memory hardware so that the spikes can be theoretically characterised in terms both specific points of memory loading and the number of cache misses where steep rises occur.

On the basis of the above observations, further research on this work will therefore entail the following: (1) Quantifying or estimating the spatial locality factors associated with specific data access schemes for homogeneous loads. (2) Quantifying parameters, M_1, M_2 and M_3 in order to predict and head off worst-case scenarios for application performance that emanate from cache miss spikes. (3) Extending the characterisations of cache miss ratios and AMAT to scenarios with mixed workloads. (4) Establishing the overall CPU Response Time by considering CPU Waiting Times at the scheduler and using queuing theory to estimate variability of OS scheduling times with load. (5) Carrying out more performance investigations on other CPU hardware architectures and platforms using PMC profiling tools.

8. ACKNOWLEDGMENTS

9. ADDITIONAL AUTHORS

Additional author(s): Dave Bustard (University of Ulster - Coleraine, BT52 1SA. email: dw.bustard@ulster.ac.uk).

10. REFERENCES

[1] R. Azimi, M. Stumm, and R. W. Wisniewski. *Online Performance Analysis by Statistical Sampling of Microprocessor Performance Counters. 19th International Conference on Supercomputing (ICS 05), Boston, USA*, June 2005.

[2] O. Corporation. *Using Oracle In-Memory Database Cache to Accelerate the Oracle Databases. Oracle Technology Network*, July 2009.

[3] S. P. E. Corporation. *SPEC CPU 2006: Readme First. http://www.spec.org/cpu2006/Docs/readme1st.html.*

[4] S. P. E. Corporation. *SPEC CPU 2006 Results.* *http://www.spec.org/cpu2006/results/.*

[5] S. P. E. Corporation. *SPEC CPU2006 Benchmark Descriptions. http://www.spec.org/cpu2006.*

[6] T. P. P. Council. Tpc results listing. *http://www.tpc.org/information/benchmarks.asp.*

[7] J. Dongarra, S. Moore, P. Mucci, K. Seymour, and H. You. *Accurate Cache and TLB Characterization Using Hardware Counters. Computational Science - ICCS, Lecture Notes in Computer Science,* 3038:432–439, 2004.

[8] P. Drongowski. *Instruction-Based Sampling and AMD CodeAnalyst. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), White Plains,* March 2010.

[9] P. J. Drongowski. *Basic Performance Measurements for AMD Athlon 64, AMD Opteron and AMD Phenom Processors.* September 2008.

[10] D. Gove. *Using Performance Counters on UltraSPARC T1 and T2 Processors - Finding Core Load as a Means to Improving Performance. System News.*

[11] D. Gove. *Calculating Processor Utilisation From the UltraSPARC T1 and UltraSPARC T2 Performance Counters. Sun Developer Network (SDN) Technical Article,* September 2007.

[12] IBM. *IBM solidDB v6.5. IBM Data Management Solutions, http://www-01.ibm.com/software/data/soliddb/,* October 2009.

[13] IBM. *Pmcount for Linux on Power Architecture - A Hardware Performance Counter Tool for the IBM POWER4, POWER4+, POWER5, POWER5+, POWER6, and PowerPC 970 Processors. IBM alphaWorks - Emerging Technologies,* September 2009.

[14] A. M. D. Inc. *CodeAnalyst User Manual.*

[15] A. M. D. Inc. *BIOS and Kernel Developer's Guide for AMD NPT Family 0Fh Processors. http://developer.amd.com/documentation,* November 2009.

[16] D. Levinthal. *Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 Processors. http://software.intel.com/sites/products.*

[17] J. Levon. *OProfile Manual. http://oprofile.sourceforge.net/doc/index.html.*

[18] I. S. Network. *Using Intel VTune Performance Analyzer to Optimize Software on Intel Core i7 Processors. http://software.intel.com/en-us/intel-vtune/.*

[19] G. Neville-Neil. *Understanding Performance with HWPMC. BSD Conference (DCBSDCon), Washington D.C., USA,* February 2009.

[20] I. S. Products. *Performance Counters on Intel(R) Processors. http://www.intel.com/software/products/documentation/.*

[21] SAP. Published benchmark results. *http://www.sap.com/solutions/benchmark/sd1results.htm.*

[22] SAP. Published benchmark results. *http://www.sap.com/solutions/benchmark/sd2tier.epx.*

[23] SAP. Sap standard application benchmarks. *http://www.sap.com/solutions/benchmark/index.epx.*

[24] SAP. *SAP MaxDB Components. SAP Community Network: SAP NetWeaver Releases, http://www.sdn.sap.com/irj/sdn,* July 2009.

[25] P. Software. *Hardware Drive Benchmark. http://www.harddrivebenchmark.net,* August 2010.

[26] S. Systems. *StreamSQL Overview. http://www.streambase.com/developers-docs.htm.*

[27] D. Terpstra. *PAPI-C: What Can Performance Components Do For You? http://www.cs.utk.edu/ terpstra,* August 2010.

[28] D. Terpstra and H. Jakonde. *Introduction to PAPI the Performance Application Programming Interface. http://www.cs.utk.edu/ terpstra/,* February 2009.