Cycle Accurate Memory Modelling: A Case-Study in Validation

Andrew Over, Peter Strazdins and Bill Clarke

Australian National University
sim-devel@ccnuma.anu.edu.au

Abstract

Simulation is an integral tool in performance analysis, however without some knowledge of a simulator's underlying accuracy and limitations, the results may prove wrong or misleading. Timing validation is one aspect of development which is easy to overlook, typically due to the lack of a comparison target at the time the simulator was written. This paper discusses the design and validation of an accurate timing model for an UltraSPARC IIICu–based system. An existing functional simulator was augmented with a cycle-accurate model of the memory hierarchy of a reference system. Key features of the model include the use of a 'bridge' for the processor / memory system interface, the use of event windows between the simulated backplane and processors, implementation of pipelined transactions, and the extension of the processor run loop to support this. The modelling of the store buffer and prefetch mechanisms proved both challenging and important for the model's accuracy.

Using a combination of documentation, microbenchmarks, and comparisons of the NAS parallel benchmarks between the simulator and a real machine, it was possible to uncover several undocumented architectural artifacts, and validate the simulator to a reasonable degree. Hardware performance counters and timing information were used to identify the source of discrepancies. Surprisingly, the overhead of introducing the model was within a factor of two, compared with the original functional simulator.

1. Introduction

Detailed modelling of computer systems can yield much deeper insights into the behaviour of both hardware and software than simple high-level metrics of application performance. One key advantage is the ability to extend a model to examine machines which do not exist, or to which the researchers do not have access. At every stage of simulator design, decisions are made regarding which system aspects are modelled in detail, and which are approximated, and here lies the key tradeoff of simulator design. Too much detail results in an extremely slow and complex simulator, while too little detail may fail to capture subtleties of system behaviour.

Validation is an essential consideration which is easy to overlook, however, without careful comparison to the target in question under a variety of workloads, it is impossible to have faith that a simulator correctly models its authors' intent.

Memory system effects are becoming increasingly important in modern multiprocessor design, and the effect of prefetching, cache behaviour, coherency protocols and the locality of memory access can have an important effect on memory performance, particularly for scientific and commercial applications with large memory footprints and non-regular memory access patterns. The accurate and detailed simulation of modern memory systems is important for understanding both the effectiveness of the design of a given memory system, and the performance of applications running on that system.

One project requiring such simulation is the ANU's CC-NUMA project [1]. Its goal is to evaluate high performance computational quantum chemistry algorithms on NUMA platforms. Through detailed modelling of existing systems, it is hoped that insight may be gained into either algorithmic or hardware modifications which may yield performance improvements, especially for 'fast' computational methods, which while parallelisable, have largely irregular memory access patterns with little temporal locality.

This paper documents the design, implementation and validation of a detailed memory simulation model embedded within the Sparc Sulima simulator framework [6], an (almost) complete machine simulator developed at the ANU. This simulation model provides a detailed timing simulation of the memory hierarchy of mid-range UltraSPARC III-based Sun servers, and is a key component in the performance evaluation framework of the CC-NUMA project.

In Section 2, general background information is provided on simulator validation, and on Sparc Sulima, while Section 3 presents the design changes required for a more accurate memory model. The validation methodology is ex-

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 IEEE

plained in Section 4, with results in Section 5. The performance of the model is briefly discussed in Section 6. Finally future directions for this work are discussed in Section 7, with conclusions summarised in Section 8.

2. Background

Validation remains an exceedingly important (but frequently overlooked) aspect of simulator development, for without an overall view of an implementation's shortcomings, it is not possible to draw meaningful conclusions.

Functional validation (as far as execution driven simulators are concerned) is so essential it is rarely mentioned; errors in functional behaviour typically manifest as a crashed application within the simulator.

Timing validation is far more subtle. Errors in the timing model merely produce incorrect results, without any obvious sign of error. The only practical way to discover such shortcomings is careful examination of every facet of simulator behaviour. Given the complexity of modern architectures, where behavioural characteristics of the processor and its memory subsystem can interact to give very different behaviour under varying workloads.

One underlying reason for a general lack of validation is the nature of computer architecture research. Typically researchers are far more interested in considering the performance of next generation of processors, memory hierarchies and interconnects rather than existing machines. The need to look forward constrains the ability to compare a simulator against an existing machine, since in the majority of cases there is no reference.

In such cases, the best that can be done is to attempt to verify the validity of the implementation, and compare behaviour of aspects of the model which closely resemble real systems (and may be appropriately isolated).

On the other hand, in cases where researchers do have access to a real machine, more direct methods of validation are available.

2.1. Previous Work

Simulators written without hardware validation (but with access to the hardware designers) have been compared to the final implementation of their target hardware. Gibson et al [7] reported a disappointing fidelity between the Stanford FLASH simulation toolsuite and the performance of the completed FLASH prototype. However, there were a number of interesting conclusions drawn from this comparison. In spite of their intimate knowledge of the architecture, their models' projections were inaccurate to varying degrees, but did predict performance trends as a result of scaling quite well. Perhaps most surprisingly the simpler pipeline models predicted performance somewhat more effectively than the more complex pipeline models.

The overall conclusion of the FLASH comparison was that — provided all "important effects" are modelled — the simulator may be trusted to yield decent speedup predictions. The difficulty is identifying in advance which particular subsets of memory, processor, interconnect and operating system behaviour are important to the application of interest.

Talisman is a simulator of MIT's Meerkat architecture, and is discussed by Bedichek [2]. Its intent was to accurately model the existing prototype, to allow the exploration of design spaces, and to project the behaviour of systems larger than the prototype. As this simulator was written with the reference target available for comparison, the resulting validation is of particular interest.

An iterative comparison process was followed, with small benchmarks identifying specific performance regimes run on both simulator and the prototype. Underlying causes for performance discrepancy were identified, and appropriate compensation was added to Talisman's timing model. This process was repeated until the desired degree of accuracy had been attained. The careful enumeration and testing of the timing of system behaviours allowed the developers to develop timing models which provided the necessary degree of accuracy, without modelling too much detail, or compromising the simulator's performance.

2.2. Original Simulator Design

Sulima's original design goal was to provide a functional full-system simulator modelling an UltraSPARC I-based system. The simulator model employed a simple `fetch/decode/execute` loop, and simple fixed latencies for cache and memory accesses. This implementation is documented in some detail by Clarke et al [6].

Instructions were executed one at a time, and assumed to have a single cycle latency, potentially with additional delays due to memory latency. As only a single instruction could be in progress, the caches were effectively blocking. Multiprocessor support was handled through a round robin scheme which interleaved the execution of each processor.

A system-call emulation environment known as Solemn [5] was used to run unmodified Solaris applications (including dynamically linked). This also allowed the observation of page faults and TLB misses.

2.3. System Model

With the availability of a suitable machine for comparison, it was decided to extend Sulima to model machines of the same class as a Sun V1280, the basic specifications of which are shown in Table 1.

The UltraSPARC IIICu [12] is composed of a relatively simple processor microarchitecture coupled to a complex cache hierarchy. Instructions are executed strictly in-order using a 14 stage pipeline, with loads stalled on a cache miss blocking until data is available. Up to 4 instructions may be issued per cycle if sufficient functional units are available.

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 IEEE

Table 1. Sun V1280 configuration

Processor count	12
Processor	900 MHz Ultra IIICu
Interconnect	150 MHz Sun FirePlane
Coherence	Snooping MOESI
Coherence Line Size	64 B
I-Cache	4-way 32 KiB
D-Cache	write-through 4-way 64 KiB
E-Cache	2-way 8 MiB
E-Cache Block Size	512 B (64 B subblocks)
Store Buffer	8 entry

A small low-latency prefetch cache (henceforth referred to as the P-cache) is checked for data on floating point loads in parallel with the D-cache (and a P-cache load may be performed from an ALU pipeline rather than the memory pipeline). This cache may be filled either by speculative hardware prefetch or through `prefetch` instructions (though only one of software or hardware prefetch may be active).

The typical split instruction/data level one cache is employed, however neither are inclusive. A large level two cache (known as the E-cache) is used which stores tags on-chip, but data off-chip. Due to bandwidth limitations caused by storing data off chip, a write cache (W-cache) exists in parallel with the E-cache. The W-cache has per-byte valid bits and is used to coalesce dirty data prior to write out to the off-chip E-cache, thus reducing bandwidth requirements on the off-chip data. The W-cache is inclusive with respect to the E-cache.

Finally a store buffer is used for all stores. Although under certain specific circumstances stores may be merged, this does not happen in practice. Data from the store buffer may be forwarded to satisfy pending loads.

Due to the in-order execution of the processor, making good use of the memory hierarchy (in particular prefetch) is essential to obtaining good performance on this processor.

The coherency protocol used between processors is documented in great detail by Charlesworth [4]. Low and mid-range systems are composed of a single snooping coherency domain, employing the MOESI protocol.

The data bus is a multi-level bidirectional crossbar-switched network used only for point-to-point data transmission.

This complexity presents potential bottlenecks at a number of locations. Stores may be throttled either entering the store buffer, or allocating space within the W-cache. The need to access off-chip data in the E-cache restricts both loads and stores. Furthermore load behaviour is different depending upon the destination register (integer or floating point). This constitutes a source of substantial complexity.

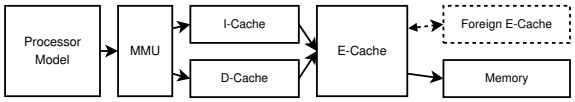

Figure 1. Original module layout

3. Design

In order to accurately examine workloads on the desired target system, a `fetch/decode/execute` functional simulator with a simple timing model and blocking caches needs to be modified into a simulation tool which can accurately analyse the behaviour of a complex interconnect, including a pipelined coherence protocol and non-blocking caches. Due to the in-order nature of the processor, correct support of prefetch is essential as it is the only means available to overlap read misses. A data cache miss will still stall the instruction stream (and is therefore effectively blocking).

The additional complexity arising from these changes will impose a noticeable performance overhead. Due to the project's interest in examining multiprocessor systems, this slowdown is of particular concern, as the overhead will be scaled by the number of processors being modelled. The ability to run a simplified functional model without performance degradation until the application reaches an area of interest (this is an approach advocated by the SimOS developers [10]) should be preserved if at all possible. This approach requires a separation between the processor model and the memory hierarchy, so that multiple models may co-exist within the code base.

As parallelisation seemed a promising way of reducing the slowdown of multiprocessor simulation on a multiprocessor host, potential parallel simulation techniques were kept in mind throughout the design. Processor and cache objects should not blithely observe or change the state of other such entities within the system, as this situation would be unworkable should multiple target processors be simulated in parallel.

3.1. General Requirements

To effectively isolate the processor model from the memory model (of which there may exist more than one), a new component, known as the "bridge" was added to the simulator. In the original design (illustrated in Figure 1), caches could examine (and modify the contents of every other cache in the system (and indeed this is how coherence was managed). This approach is deficient in many respects, particularly when faced with multiple potential timing models, more complex coherence protocols, or a desire to experiment with parallelism.

In the new design (shown in Figure 2), the bridge component sits between the processor model and the caches, and

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 IEEE

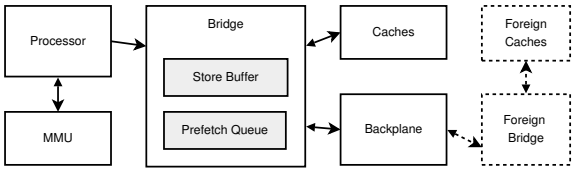

Figure 2. Revised module layout

provides the interface between the processor model and the memory model. By isolating these two components from one another, multiple memory timing models may be used with the same processor model.

Timing models of interest include a cacheless global memory system (the simplest model), non-pipelined and fixed latency cache/memory models (as in the original Sulima design), and the detailed pipelined model (described in this paper). These can be changed dynamically in order to permit high-speed functional emulation, or alternatively to permit detailed performance analysis when the simulated workload reaches areas of interest.

When simulating an interaction with other processors in more complex models, it is not possible to determine in advance the latency of a given memory transaction. It must be possible to maintain state for a given memory transaction until it can be determined that it has completed. Given the desire to maintain the existing model, it was decided to associate a structure with each memory request, and to ensure that the same instance of this structure was passed through various layers of the hierarchy.

Additional state tracking — which associated requests with their pending memory transactions — allowed the simulator to effectively track multiple simultaneous requests (and to determine what to do with the provided data once the request had completed).

Partitioning the simulation of processors into a form suitable for parallelisation required a careful split of communication and coherence logic. While the original implementation of Sulima directly manipulated all caches, doing so in a parallel implementation would expose concurrency issues. Instead a "backplane" object is used to manage communication between processors via their respective bridge objects.

In the more detailed timing models, this backplane is responsible for simulating the core interconnection network. Rather than a simple round-robin timing loop, a modified scheduler is run in which the backplane determines communication events before processors are run for their time slice. This is similar to the `accurate` model used in the WWT [3].

3.2. Processor Run Loop Modifications

The traditional `fetch/decode/execute` style of processor simulation requires a memory model which can immediately return the desired latency when a memory event occurs. This section describes how the run loop for the processor has to be modified in order to support a memory system modelling pipelined transactions.

As the completion time of a memory event is not known in advance, to avoid a complete restructure of the processor model, the in-built exception handling mechanism can be extended for this purpose. A special dummy "stall" exception is returned to indicate that the processor should idle the instruction stream for the given number of cycles before re-executing the current instruction. This number of cycles represents the minimum time that the operation could complete, given the current state of the memory model. This process is repeated until the instruction requesting the memory event succeeds, at which point normal execution resumes.

The use of the backplane component also requires each processor maintain an event queue, in order to process coherency events at the right time and in the right order. This must be checked upon every cycle at the top of the run loop, along with checking for processor interrupts.

The modified run loop then also checks whether it is in an 'idle' state, due to the stall exception; in which case, it simply advances its clock and performs no action apart from checking its event queue. After any stage that can involve a memory event (e.g., fetch, or execute load/store), the stall exception is checked for. A state variable is set to suppress the fetch/decode/execute stages on future iterations of the loop until the required idle time expires. Thus, the structure of the loop is only modestly more complex than the original.

3.3. Store Buffer

The store buffer masks store latency, by permitting a store instruction to retire immediately and allowing the buffer to perform the memory transaction. This implies some additional processing of memory events outside of the instruction stream.

A queue of pending stores must be maintained. If the queue is full, the store instruction must stall until there is room. Furthermore, this queue must be checked on the evaluation of each load instruction to determine whether data may be forwarded. On a store buffer hazard (partial conflict with data in the store buffer), the load must stall until the store buffer no longer represents a hazard.

As the processor itself explicitly denies any possible conflicts between an in-progress store and a pending load, this does not present substantial complications.

Care must be taken to fully drain the store buffer on atomic instructions in order to avoid violating the processor's consistency model (total store order).

3.4. Prefetch

Accurate modelling of prefetch is essential for realistic performance estimates of this processor. It was decided not

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 IEEE

to model hardware prefetch, as the algorithm for determining which addresses to fetch is not documented. Instead, the `prefetch` instruction was modelled. This instruction allows the user manual control over whether a line is fetched into the P-cache (only), the E-cache (only), both the P-cache and the E-cache, or the E-cache with exclusive access.

There are two immediate complications. The need to process asynchronous transactions (as in the case of the store buffer), and the need to detect conflicts between later load instructions, pending stores (from the store buffer), and instruction fetches. Of particular concern are conflicting requests for exclusive and shared access (if an exclusive request arrives while a shared request is outstanding, it must stall). While a request that begins as a prefetch may be destined only for the prefetch cache, if a later load request overlaps, it needs to be "upgraded" to a real load as far as the cache is concerned.

After some consideration it seemed the simplest resolution to this problem was to implement the equivalent of MSHRs (Laudon [8, §5.1] provides a good overview). State tracking within the bridge was implemented to keep track of the origin and destination of all requests, and this logic merged compatible requests, while stalling incompatible requests.

Additional overhead is inflicted by the need to check the prefetch cache on each floating point load, the need to invalidate the prefetch cache on each snoop or store, and care must be given to ensure that prefetches that cause a TLB miss are silently dropped without faulting.

4. Methodology

Overall model validation is a critical concern in simulator development. Without having confidence in the accuracy of a simulator, it is not possible to draw conclusions from the provided data.

Part of the motivation for choosing this particular system configuration as a reference was the authors' access to the reference system. This permits direct comparison to a running system, and allows us to directly evaluate our model. Although we are modelling only user level applications (including system libraries, page faults and TLB misses, but not including kernel activity), careful selection of comparison applications allows verification of the simulation areas we hope to model accurately. The intended focus on scientific applications allows us to ignore kernel-level effects (aside from considerations of page coloring, memory placement and processor affinity).

A two-pronged approach was taken to validation. Firstly, the low-level processor model (including the memory timing of caches, and the behaviour of the store buffer) was examined using a system of carefully written microbenchmarks. Secondly, higher level benchmarks were examined using the NAS parallel benchmarks [9], in conjunction with the processor's performance counters. This enabled a direct comparison of a number of aspects of system behaviour (beyond simply overall run time). The NAS benchmarks made for particularly good baseline applications for validation, as they are scientific applications which (once running) do not interact with the kernel in any noticeable way.

On several critical measures, validation of the high-level benchmarks was made possible through an accurate CPU simulation module [11]. This provided a good estimate of the behaviour (particularly IPC) of the real system's CPU, and accurately timed requests issued to the memory subsystem.

Although some documentation was provided regarding the memory latencies of the system, it seemed judicious (and as it turns out, justified) not to place undue faith in this documentation, but rather to validate the latencies ourselves.

4.1. Microbenchmarks

The aim of microbenchmarking was to determine the latency of a full range of memory transactions. A simple framework was written to allow benchmarking on either single or multiple processors (which was necessary to force certain cache transitions).

When running these benchmarks care was taken to ensure that the following assumptions were satisfied:

- All threads within a benchmark are on processor simultaneously
- Each thread is assigned its own processor
- Threads did not migrate between processors
- No other application would access the memory or processors being used by the microbenchmark
- Nothing could perturb cache state beyond victimisation due to OS interference (or interrupts)
- All threads are run on the same processor board (and where possible the same processor pair) to minimise data transfer latency

Care was also taken to ensure that device related interrupts were not serviced on the processor used for benchmarking. All tests were run on the reference host using Solaris 9.

A typical microbenchmark is illustrated in Figure 3. Note that the assembly is slightly simplified, and branch delay slots are ignored for brevity. Once the cachelines were manipulated into the appropriate state, the processor's `%tick` register (which counts processor clocks) was used to time the instruction.

Flushing the cache from userspace proved surprisingly difficult as the UltraSPARC IIICu uses a pseudo-random

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 IEEE

```
Processor A                Processor B

st    %g0, [A]     call  _cache_flush
call  _barrier     call  _barrier
                   rd    %tick, %l0
                   ld    [A], %g0
                   rd    %tick, %l1
                   sub   %l1, %l0, %l0
call  _barrier     call  _barrier
```

Figure 3. Microbenchmark to measure cache-to-cache transfer latency

Table 2. Latencies on UltraSPARC IIICu

Benchmark	*CPU clocks*
`ld` (E$ **I**)	206–240
`ld` (D$ inv, E$ valid)	18–23
`ld` (D$ valid)	1
`st` (E$ **I**)	180–205+
`st` (E$ **S**)	126–134
`st` (E$ **E/M**)	1
`ld` following `st` (RAW bypass)	1
cache to cache transfer	269–273
`cas` **E**	47–55
`cas` **M**	42–50
`cas` **S** foreign **S**	171–183
`cas` **S** foreign **O**	171–183
`cas` **O**	171–183
`cas` **I**	243–264
`cas` **I** foreign **M**	301–321

LFSR-based replacement policy for the E-cache (a fact which is poorly documented), and the only reliable means of flushing is privileged. As a result, a range of pages with appropriate *physical* addresses must be acquired to flush the E-cache, while contiguous virtual addresses suffice for the D-cache. In each case, displacement flushes must be performed numerous times to ensure a flush.

It is also important to avoid undesired artifacts of the microarchitecture. When benchmarking stores, store buffer behaviour must be considered (and overcome), while when benchmarking loads, consideration must be given to the difference between pipeline recirculation stopping (and the next instruction commencing), and the time at which the loaded value is usable.

The basic results of microbenchmarking are illustrated in Table 2. This table considers the latency of a variety of memory related operations, with the MOESI state of the cacheline in question given in bold.

Observation of the load/store results yields several basic latencies. The D-cache and E-cache hit latencies are fixed at 1 and approximately 20 cycles respectively. Cache to cache transfer latency is of the order of 270 cycles, while upgrade latency is 125. Finally the basic overhead for performing an atomic operation is around 40 cycles.

Table 3. Store buffer latencies on Ultra IIICu

Storebuf Benchmark	*CPU clocks*
RAW (D$, E$ hit)	1
RAW (D$ miss, E$ hit)	1
RAW (D$, E$ miss)	13
RAW overlap (D$, E$ hit)	25
RAW overlap (D$ miss, E$ hit)	43
RAW overlap (D$, E$ miss)	220–250
RAW same line (D$, E$ hit)	1
RAW same line (D$ miss, E$ hit)	40–43
RAW same line (D$, E$ miss)	220–250
RAW past hazard (D$, E$ hit)	36–43
RAW no hazard (D$, E$ hit)	1

These measurements can be compared to the observed results for more complex scenarios (the atomic operations). An atomic operation which requires the update should be expected to consist of the upgrade latency and the atomic latency (i.e., 165 cycles) which compares well to the observed 171 cycles. Similarly, an atomic requiring a cache to cache transfer would be expected to take 310 cycles (compared to an observed 300–320).

A sufficiently detailed series of microbenchmarks allows us to determine some internal latencies which could not otherwise be observed directly.

The store buffer behaviour (shown in Table 3) proved quite interesting, and demonstrated several undocumented peculiarities. Read after write bypassing does not forward data if there is a miss on the E-cache. Instead there is a 13 cycle penalty. This is unusual behaviour, but is unlikely to influence timing projections of typical workloads.

Overlapped accesses (for example writing 8 bytes, then reading the lower or upper 4) inflicts quite a severe penalty (and in all cases it is about 20 cycles slower than an equivalent load without the preceding store). This result is quite surprising, as the requisite data is resident in the store buffer. Attempting to access a non-overlapping region of a cacheline with a pending store also imposes the same penalty on a D-cache miss.

When a series of stores are performed to a given cache line, an attempt to read back anything except the final store will inflict a 30 cycle penalty. This final latency is particularly troubling, as this access pattern is quite common when transferring data between integer and floating point registers via the stack (as there is no way to do this directly).

These penalties came as something of a surprise, and are not documented in the processor literature. Although the majority of the microbenchmarks closely matched documented behaviour, the store buffer proved to defy expec-

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 **IEEE**

Table 4. NAS Benchmark validation (1 processor)

Metric		bt.S	ft.S	is.S	lu.S	lu-hp.S	mg.S	sp.S
Cycle_cnt	real	598580922	625336080	10076197	185462563	183257234	20072195	251648809
	sim	604697080	492572873	10841070	184472044	176643478	17664892	230838376
	norm	1.01	0.791	1.08	0.995	0.964	0.881	0.9173
DC_rd_miss	real	1535717	2697685	90791	1414765	1164821	302577	2550524
	sim	807333	1034429	83300	1415994	1162283	233134	1876017
	norm	0.526	0.384	0.918	1.01	0.998	0.770	0.736
Re_DC_miss	real	33441391	73342393	1873771	31024520	24648855	6334474	57440254
	sim	16158660	21416846	1666400	28325080	23483020	4668760	37556340
	norm	0.483	0.292	0.889	0.913	0.957	0.737	0.653
Re_RAW_miss	real	3558661	2654169	89542	686327	1271659	101407	1138007
	sim	1098114	1562686	4880	131901	519532	199588	2425778
	norm	0.309	0.589	0.0545	0.192	0.409	1.96	2.13
Rstall_storeQ	real	161642730	257997086	5423461	28103945	26136530	514236	38924631
	sim	173557526	213641772	6099283	29472842	18891858	502118	28057554
	norm	1.07	0.828	1.12	1.05	0.723	0.976	0.721

tations, and was a significant source of inaccuracy prior to validation.

4.2. NAS Parallel Benchmarks

The NAS parallel benchmarks are a set of benchmarks from a variety of scientific applications which include a number of different problem sizes, and may be scaled from uniprocessor up to 256 or more processors.

The OMP variants of these benchmarks were modified using Solaris's `libcpc` to collect performance counts around the computation phase of each benchmark (setup and verification was ignored due to the probability of greater operating system involvement). Initially the OMP instances were run on a single processor to minimise sources of error, while later in the validation cycle, these benchmarks were tested on multiprocessor runs.

The counts returned from running these applications natively were then compared with those from a simulation run with the identical region of the application instrumented. The "S" class was used as the basis of comparison against the most detailed timing model in the simulator. In each case they were run within a processor set on a quiesced processor board of our target machine, to minimise external disturbances. Results were reproducible to within 1–2%.

Initial attempts at validation revealed a number of model inaccuracies. The store buffer proved a particular source of trouble, as it is only capable of draining a store once every second cycle. Repeated stores to cachelines which are in memory (such as those performed by certain high-density floating point codes) may quickly fill the store buffer and experience unexpected stalls. This behaviour was not documented.

The W-cache also proved troublesome. Given the nature of the cache (inclusive with the E-cache, operates in parallel, per byte validity, data merged from E-cache and W-cache on a load), it was assumed that it could be effectively ignored. Unfortunately the bandwidth limitation between the W-cache and the E-cache's off-chip data is something which is hit all too easily. Experimentation reveals that it takes approximately 30 clock cycles to shift a cacheline off chip, and it is not possible to shift multiple cachelines in parallel. Benchmarks which stride cachelines (such as IS), were over twice as slow as projected by the simulator.

A bug in the prefetch implementation involving victimisation of dirty E-cache lines when requesting prefetch with exclusive access manifested as a massive inaccuracy in the runtime of the FT benchmark (which makes particularly heavy use of this form of the prefetch instruction).

In the first two cases, the inaccuracies were caused by the (what seemed relatively minor) failure to accurately model a bandwidth limitation inherent to the processor. However, given the nature of the benchmarks, and the way they stress hardware in a variety of different ways, this bandwidth limitations proved to be responsible for much of the observed performance.

Calibration against the NAS benchmarks tested a range of system behaviour, and demonstrated numerous inconsistencies in our model. At various points, the store buffer, read after write forwarding, and integer memory behaviour were tested, and all benchmarks (except IS) make extensive use of the `prefetch` instruction. Together they yielded a very good set of test cases for the memory behaviour of this system.

5. Results

Table 4 shows the final event counts provided by the detailed memory simulation model for uniprocessor runs of several of the benchmarks, normalised against the real

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 **IEEE**

Table 5. NAS Benchmark validation (OMP, 2 processors)

Metric		bt.S	ft.S	lu.S	lu-hp.S	mg.S	sp.S
Cycle_cnt	real	315607317	340967363	172221912	175478383	13483880	160194823
	sim	324684948	256938597	168825732	168049551	11490780	145957236
	norm	1.03	0.754	0.980	0.958	0.852	0.911
EC_snoop_cb	real	57259	236861	105057	229644	8561	120150
	sim	69737	253383	95727	280864	10630	143183
	norm	1.22	1.07	0.91	1.22	1.24	1.19
EC_snoop_inv	real	55591	193448	91122	101858	5541	100152
	sim	65786	237247	98053	190609	8029	139308
	norm	1.18	1.23	1.08	1.87	1.45	1.39

machine. The metrics describe UltraSPARC IIICu performance counters [12]. While Cycle_cnt represents the overall projected execution time, the D-cache read miss count is provided as a reference, and the remaining three counts are the primary source of pipeline stalls aside from register and functional unit dependencies.

Re_DC_miss indicates the total number of cycles stalled due to misses on the D-cache, and includes all memory latency as a result of the miss (such as time to retrieve data from the E-cache).

Re_RAW_miss indicates the time spent stalled due to read after write hazards between loads and stores pending in the store buffer. This occurs any time that it is not possible to bypass the data directly to a load.

Rstall_storeQ indicates time spent waiting while the store buffer is full. Benchmarks which write a large amount of data in sudden bursts are prone to stalling in this fashion. There is a fixed limit at which stores may be dispatched from the store buffer to the cache hierarchy (as discussed above).

While very good accuracy was achieved for the overall results of some benchmarks, others were well off the mark (almost 12% in the case of MG). The typical culprit for large inaccuracies was the total time spent on cache misses (which in extreme cases was underestimated by a factor of five).

The store buffer behaviour was somewhat more consistent, however, due to the sheer magnitude of time spent stalled on the buffer in FT, it contributed almost half of the difference between projected and real execution time.

Although substantial effort has been expended to mimic both the documented and the measured behaviour of the UltraSPARC IIICu, in most cases the simulator remains quite optimistic. Table 4 demonstrates that the benchmarks for which predicted performance deviates most significantly are those for which the predicted D-cache hit rate deviates.

In other cases, where the hit rate is predicted successfully, the predicted results are much closer to measured performance. The cause for the underlying discrepancy in miss rates is the pseudo-random replacement algorithm discussed briefly in Section 4.1. This algorithm is not consistently documented, and has not yet been modelled successfully. Instead an LRU replacement policy was used, yielding (in some cases) vastly improved miss rates. Although in some cases memory use patterns and effective use of prefetch masked this difference, in other cases it is all too apparent.

Benchmark results for the master thread of a two processor OMP run of the NAS benchmarks are illustrated in Figure 5. In all cases the benchmarks were run on a processor set within a quiesced system. Two additional performance counters are compared:

EC_snoop_cb indicates the number of cachelines copied back to another processor. These copybacks arise as a result of both shared and exclusive requests for the cacheline in question.

EC_snoop_inv indicates cachelines invalidated for coherency reasons. Typically this is due to modification by another processor.

Multiprocessor validation introduces additional inaccuracy due to shared memory. The precise timing (and even counts) of events arising from coherency cannot be reproduced precisely without also reproducing the exact timing of the target system.

Overall the simulator projected approximately 20% more copybacks and 25% more invalidations due to coherence than were observed in a real system for the two processor case.

5.1. Discussion

With such coarse-grained information about what was responsible for a stall, it is difficult to pinpoint the exact inaccuracy responsible for the discrepancy. Furthermore, implementation details may end up masking the severity of a problem. During testing, a bug was discovered in the implementation of prefetch, and upon fixing it, cache miss rates dropped further, making the projected execution time even more optimistic.

The combination of using a series of microbenchmarks

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 IEEE

(generated both from documentation and from observed behaviour) to begin calibration, coupled with a more high-level benchmark to pinpoint areas to examine for further inaccuracies, proved reasonably successful.

Through an examination of the performance counters, we were able to focus our testing efforts on the regions furthest from the observed results. It should be observed that this frequently had unforeseen consequences. The memory hierarchy of a processor such as the UltraSPARC II-ICu has complex interactions between its components, and tweaking one component can have serious consequences elsewhere. Uniprocessor FT was initially pessimistic and demonstrated extremely poor store buffer behaviour (excessive stalls). Upon discovery of a bug in the `prefetch`, it became the most optimistic of the benchmarks due to its extremely heavy use of "prefetch write many" (which acquires exclusive access, and reduces the time spent draining the store buffer).

The projected uniprocessor performance of IS was initially optimistic by a factor of 2 due to the aforementioned undocumented bandwidth limitation of the W-cache. In many cases reducing the latency of D-cache accesses had almost no effect on the projected execution time, and merely increased the time spent stalled draining the store buffer, or resolving read after write conflicts.

The location of the underlying model deficiency was not always apparent from the breakdown of stalls, and although this may provide a hint as to the problem, it does not always help.

Contingent upon the cache replacement policy, this simplified model has proven relatively accurate in comparison to a real system. As long as the cache miss counts are similar between the simulated model and the real system, reasonable confidence may be had in the accuracy of at least this family of scientific workloads. Validation against a real system has demonstrated that by modelling only some of the complexity (prefetch, multiple outstanding requests, store buffer) and coupling it with a lightweight processor timing model, it is still possible to retain a degree of accuracy.

Multiprocessor results indicate that the coherence behaviour matches the real system to within 25%, a sizeable error. Further examination of the cache replacement algorithm, and comparison of multiprocessor benchmarks in more finely-grained sections (e.g. within each OMP parallel region) may provide traction on this discrepancy. Some error is expected due to timing variation, however, it may prove possible to further reduce this with additional tuning.

6. Performance

Table 6 gives the overhead of the model, compared with the original Sulima speed [6]. The column 'plain' refers to the blocking non-pipelined memory model (see Section 3.1); this indicates the overhead introduced by the bridge and modifying the original run loop. 'Detailed' refers to non-blocking pipelined model described in this paper; surprisingly, it introduces very little extra overhead. The remaining columns indicate the more significant effect of enabling the store buffer and prefetch components.

Table 6. Normalized Performance of Memory Model

NPB	plain	detailed	+ store buffer	+ prefetch
bt.S	1.19	1.21	1.59	1.94
cg.S	1.21	1.32	1.40	1.98
lu-hp.S	1.17	1.24	1.52	1.95

7. Future Work

The validation completed thus far primarily focuses on the upper regions of the memory hierarchy. Validation of the interconnection network and the shared memory behaviour of the simulator may prove to be a challenging problem, but is the logical next step in the validation of the simulator. Preliminary two processor results indicate that a finer-grained examination of synchronisation regions may be required to fully validate shared memory interactions, and multiprocessor validation must be scaled to cover more processors.

Speed and accuracy are two conflicting goals. Once analysis of large shared-memory applications begins, avoiding a serious slowdown will become a matter of great interest. Given the manner in which the detailed memory model has been designed, it is hoped that the system as a whole may be parallelised on a shared-memory machine with a relatively minor slowdown for multiprocessor simulation runs. This will permit the examination of larger systems in a workable time frame.

Still to be confronted is the issue of memory placement. Memory placement and processor affinity are two aspects of operating system behaviour which may be expected to have a large impact on the behaviour of computation chemistry codes on a NUMA system. Providing some means of tuning this behaviour will be essential to achieving an accurate characterisation, and its influence on algorithm behaviour remains of some interest.

8. Conclusions

This paper has presented the design of a detailed memory model of an existing system (a Sun V1280), and the validation of the resulting timing model against this system, through a combination of microbenchmarks and small-scale scientific applications (a workload appropriate for our eventual simulation goals).

Key features of the model included the use of a 'bridge'

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 IEEE

to provide a flexible interface between the processors and the memory system, permitting the use of simpler and faster models for workload positioning. The use of event windows between the simulated backplane and processors enables an efficient but accurate means of modelling interactions, and will permit the model to be parallelized for improved performance. Other important features included the implementation of pipelined memory and cache coherence transactions, with the transaction state being modelled and updated throughout this process, and the extension of the processor run loop to support asynchronous event processing. The modelling of the store buffer and prefetch mechanisms proved both challenging and important for the model's accuracy.

Surprisingly, the overall overhead introduced by the cycle accurate model is within a factor of two, as compared with the original functional simulator model. Also surprisingly, the pipelined backplane itself introduced little overhead, the bulk being introduced by store buffer and prefetch modelling.

Even with documentation and access to a real system with reasonable performance monitoring infrastructure, this has proven a remarkably difficult task. Primarily due to the complexity of modern processors and memory hierarchies there are a large number of potential performance bottlenecks. While upon initial examination these may seem to play no role, under certain workload regimes they dominate performance.

Through a careful attempt to examine processor/memory behaviour using a series of both low and high level benchmarks, the processor's performance counters and an iterative approach, it is possible to obtain a good characterisation of a system, and incorporate this model into a simulator. However, without absolute certainty that all behavioural aspects have been accounted for, accuracy on untested workloads remains in doubt.

The methodology presented above is fairly generic, and relies on microbenchmarks tuned to the architecture, hardware performance counters, and operating system support for process pinning. It should be possible to validate in this manner on most modern systems.

With detailed and accurate documentation, validation is hard. Without documentation, validation may prove daunting. This work underscores the need to understand precisely how a processor behaves at a low-level in order to derive an accurate timing model.

Acknowledgements

The authors wish to thank the Australian Research Council, Sun Microsystems Inc. and Gaussian Inc. This research has been funded under ARC Linkage Grant LP0347178.

Yan Zhang and Nicolas Jean assisted instrumenting the NAS parallel benchmarks, while David Hearnden and the reviewers provided helpful feedback on drafts.

References

[1] Australian National University. The CC-NUMA project: Computational Chemistry on Non-Uniform Memory-access Architectures. `http://cs.anu.edu.au/CC-NUMA`.

[2] R. C. Bedichek. Talisman: Fast and Accurate Multicomputer Simulation. In *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modelling of Computer Systems*, pages 14–24. ACM Press, 1995.

[3] D. C. Burger and D. A. Wood. Accuracy vs. performance in parallel simulation of interconnection networks. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 22–31, 1995.

[4] A. Charlesworth. The Sun Fireplane System Interconnect. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*. ACM Press, New York, New York, USA, November 2001.

[5] B. Clarke. Solemn: Solaris emulation mode for Sparc Sulima. In *Proceedings of the 37th Annual Symposium on Simulation*, pages 64–71. IEEE Computer Society, 2004.

[6] B. Clarke, A. Czezowski, and P. Strazdins. Implementation aspects of a SPARC V9 complete machine simulator. In *CRPITS '02: Proceedings of the twenty-fifth Australasian conference on Computer Science*, volume 4, pages 23–32, 2002.

[7] J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich. FLASH vs. (Simulated) FLASH: Closing the Simulation Loop. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 49–58. ACM Press, 2000.

[8] J. P. Laudon. *Architectural and Implementation Tradeoffs for Multiple-Context Processors*. PhD thesis, Computer Systems Laboratory, Department of Electrical Engineering and Computer Science, Stanford University, September 1994. Available as Technical Report CSL-TR 94-634.

[9] NASA Advanced Supercomputing. NAS Parallel Benchmarks. `http://www.nas.nasa.gov/Software/NPB/`. Version 3.1.

[10] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer system simulation: the SimOS approach. *IEEE Parallel and Distributed Technology: Systems and Applications*, 3(4):34–43, 1995.

[11] P. Strazdins. CycleCounter: an Efficient and Accurate UltraSPARC III CPU Simulation Module. Technical Report TR-CS-05-01, Department of Computer Science, Australian National University, May 2005.

[12] *UltraSPARC III Cu User's Manual*. Sun Microsystems, Santa Clara, California, USA, January 2004. Version 2.2.1.

Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)
1526-7539/05 $20.00 © 2005 **IEEE**