A Study of Performance Impact of Memory Controller Features in Multi-Processor Server Environment

Chitra Natarajan
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052, USA.
(1) 408-765 8080
chitra.natarajan@intel.com

Bruce Christenson
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052, USA.
(1) 408-765 8080
bruce.christenson@intel.com

Fayé Briggs
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052, USA.
(1) 408-765 8080
faye.a.briggs@intel.com

ABSTRACT
With the growing imbalance between processor and memory performance it becomes more and more important to optimize the memory controller features to obtain the maximum possible performance out of the memory subsystem. This paper presents a study of the performance impact of several memory controller features in multi-processor (MP) server environments that use a DDR/DDR2 based memory subsystem. The results from our studies show that significant performance improvements can be obtained by carefully optimizing the memory controller features. For instance, one of our studies shows that in a system with an in-order shared bus connecting the CPUs and memory controller, an intelligent read-to-write switching memory controller feature can provide the same order of benefit as doubling the number of interleaved memory ranks. Another study shows that much lower average loaded read latency across a wider range of throughput can be obtained by a delayed write scheduling feature.

Categories and Subject Descriptors
B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; B.3.3 [**Memory Structures**]: Performance Analysis and Design Aids – *Simulation*; C.4 [**Performance of Systems**]:– *Performance Attributes.*

General Terms
Performance, Measurement, Design, Experimentation.

Keywords
Memory subsystem, Memory controller, Memory transaction scheduling, Multi-processors, Server systems, Performance impact.

WMPI '04, Munich, Germany.

1. INTRODUCTION
Over the last two decades processor & memory performance have been increasing, but at significantly different rates: processor performance has increased at the rate of roughly ~55% per year while DRAM latencies have decreased only at the rate of ~7% per year and DRAM bandwidths have only increased at the rate of ~20% per year ([6]). This has led to the well-known memory-wall problem: the ever-widening gap between processor & memory performance reducing the final delivered processor performance. Despite extensive research on processor techniques to tolerate long memory latencies such as pre-fetching, out-of-order execution, speculation, multi-threading, etc., memory latency continues to be an increasingly important factor of processor stall times ([8]). Moreover, many of these processor techniques to tolerate memory latencies result in increased bandwidth demand on the memory subsystem ([3]).

System performance depends not only on the peak bandwidth and idle latency but also on the actual maximum sustainable bandwidth and the queuing latency encountered by the application during execution and hence, the loaded latency (idle + queuing latency). For a given architecture and workload, the loaded latency and sustainable bandwidth can vary quite widely depending on the memory controller features. Previously a number of studies have looked at the impact of one or more memory controller features/policies in a desktop or workstation uni-processor environment ([1], [5], [4]) and typically using SPECint / SPECfp benchmarks. In this paper we present a study of the impact of several memory controller features on the memory performance in a shared-bus MP server environment.

We evaluate server system memory performance by comparing the loaded latency vs. delivered throughput/bandwidth characteristics with different memory controller features. The loaded latency vs. delivered throughput performance can be used as an input to project impact on server benchmarks such as TPC-C, SPECweb99 etc.; how-ever such benchmark impact projection is beyond the scope of this paper.

The remainder of this paper begins with a brief description of the typical shared-bus MP server system and its memory subsystem in Section 2. In Section 3 we provide a brief overview of our simulation models. Section 4 describes and presents the study of various memory controller features in MP server environments that use a DDR/DDR2 based memory subsystem. Section 5 concludes the paper.

2. SHARED-BUS MP SERVER SYSTEMS AND ITS MEMORY SUBSYSTEM

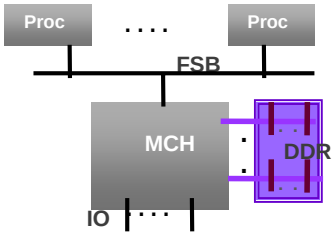


Figure 1 Single Shared-bus MP Server

Fig. 1 illustrates a typical shared-bus MP server system in which multiple processors are connected to a single shared coherent Front Side Bus (FSB), where the coherency is maintained by snoop mechanism in the processors. The shared FSB is also connected to a Memory Controller Hub (MCH). The MCH is the component that interfaces the FSB, memory channels, and IO buses, and the processors/devices connected to these buses. The FSB may be an in-order bus (where responses have to occur in the order in which requests arrive, and if responses are deferred they can be completed later in any order, but only by using a FSB deferred-reply transaction that utilize all the FSB resources again) such as the Intel IA32 FSB or an out-of-order (O-o-O) bus such as the Intel Itanium Processor Family (IPF) FSB (where responses can be deferred and can be completed later in a different order compared to the order in which requests arrived using an enhanced-defer mechanism that does not use all the FSB resources again). Requests to the memory subsystem come from processors via the FSB or from IO devices via the IO buses.

2.1 Memory Subsystem

The memory subsystem in these servers typically consists of one or more DDR or DDR2 channels that may be all used independently or ganged together in lockstep (typically as lockstep pairs). Each channel supports one or more DIMMs. A DIMM typically has 1 or 2 ranks. Each rank has multiple independent banks (4 & 8 banks/rank are expected to be common). Each bank has a number of memory cells organized as an array of rows and columns.

When a bank's row (sometimes called a page) is accessed (row activation operation) the entire row/page is brought into the row buffer of the bank. When a row/page is active/open, any number of individual elements of the row/page can be read or written via column access operations. Such read/write accesses to an open page requiring only a column access operation are typically referred to as page hits (RD). If an access needs to happen to a different page in the bank other than the one already open (bank conflict), the open page must be written back to the bank (pre-charge operation), the new page then activated, and the column access performed to access the desired element. Such accesses are typically referred to as page misses (PRE-ACT-RD). If the bank is already pre-charged i.e., no page is in open condition, accessing an element from the bank requires only opening the desired page and then performing the read/write operation. Such accesses are typically referred to as page empty (ACT-RD). In addition to an explicit pre-charge operation, these memory technologies have an auto pre-charge mechanism that can be used to automatically pre-charge the bank and close the page with every access.

We can easily observe from the above description that accesses have the lowest latency with page hits and the highest latency with page misses, and also deduce that a series of page misses would result in lowered bandwidth due to holes created in the data transfer waiting for the pre-charge operation to complete. The page hit/miss/empty probability for memory accesses depend on a number of factors such as the workload traffic characteristics (high locality vs. random), and memory controller features, which include page policy (open page policy: keep pages open either until it needs to be closed or closed adaptively using timers etc and closed page policy: use auto pre-charge to close pages with every access), and address mapping/memory interleaving. Evaluation of these and other memory controller features in a MP server environment is the subject of Section 4.

3. SIMULATION MODELS

We have built near cycle accurate micro-architecture simulation models of the FSB, the MCH, the memory channels, and the IO buses in a C++ environment with configurable knobs to vary a number of parameters. Traffic on the FSB and IO bus is driven using traffic generator models that can either read a trace file and generate the specified traffic or generate the traffic given a set of parameters such as read/write mix probability, request inter-arrival time distributions, and request size. We briefly describe the simulation models in this section.

3.1 FSB Model

The FSB model implements all the essential details of the FSB protocol such as symmetric bus arbitration among multiple FSB agents, priority arbitration between the MCH and FSB agents, the various bus phases & stalls, and data bus arbitration between the FSB agents & MCH. The model supports all the key FSB transactions generated by the FSB agents as well as snoop transactions generated by the MCH for coherent memory accesses from IO devices. The model supports both in-order and O-o-O FSB. Other configurable knobs include the data bus width and cycles per address. By configuring the knobs appropriately we can simulate both IA32 FSB & IPF FSB.

3.2 Memory Channel Model

The memory channel model is configurable to simulate DDR or DDR2 DRAM devices. A single memory channel module can simulate one independent channel or many channels operated in lockstep. The channel burst length and a knob indicating the number of bytes transferred per cycle determine if the channel is one independent channel or many channels operated in lockstep.

The core of the memory channel model is contained in a special function that is called whenever the memory controller wants to determine if a given transaction can be scheduled starting in the current DRAM clock cycle. This function allows the memory controller model to check a number of transactions for schedule readiness, and then use that data to determine which transaction is the best one to schedule in the current clock cycle using its policies/features. The special function in the memory channel model goes through a test process to determine if a transaction can be scheduled starting in the current clock cycle. First, the model checks to see if a page hit operation is possible.

If not, it checks to see if a page empty operation is possible. If not, it checks to see if a page miss operation is possible. If none of these operations are possible, the model returns false. If one of the operations is possible, the model returns true and also indicates the operation type (hit, empty, or miss), in case the memory controller model can make use of the extra information in its policies.

The functions that check to see if a page hit, empty, or miss operation is possible first determine which DRAM commands (ACT, RD/WR, and PRE) are required for the operation, and then determine if the DRAM commands for the transaction can be scheduled starting in the current clock cycle, obeying all required timing constraints of the memory channel protocol, including DRAM device timing constraints such as tCL (RD-to-data latency), tRCD (ACT-to-RD latency), tRP (PRE-to-ACT latency), etc. A table style record is kept of all on-going transactions on the memory channel, and this information is used for checking the schedule readiness of the current transaction.

3.3 MCH Model

The FSB interface unit, IO interface unit, the memory interface unit with memory controller, and internal data path, buffers & queues are modeled in the MCH model.

The memory controller model within the MCH model dynamically (based on a con-figuration knob) instantiates one or more memory channel models, based on how many independent individual channels or independent lockstep pairs of channels need to be simulated. The memory controller model has a read request queue, a write request queue, and an arbiter. Many different scheduling policies have been modeled and a given policy can be chosen by appropriately setting various configuration knobs. For example, the memory controller request queues can be fully out-of-order, or fully in-order, or partially out-of-order (by specifying the number of entries in the queue to examine during scheduling).

3.4 IO Bus Model

The IO bus model implements the basic features of the PCI-Express bus and is also highly configurable with knobs for number of channels, channel widths, buffers, credits, latency, arbitration policies etc.

.

4. PERFORMANCE IMPACT OF MEMORY CONTROLLER FEATURES IN MULTI-PROCESSOR SERVER SYSTEMS

4.1 Study #1

We began our study of the performance impact of memory controller features in MP server systems by first examining a simple memory controller from a desktop system that was used in a dual-processor (DP) server system with an in-order FSB. In a uni-processor desktop/workstation environment, and for many of the applications in the SPECint / SPECfp benchmark suite that is representative of the workload in such environments, significant locality is found in the memory address stream ([1], [5]), especially with the relatively smaller cache sizes in the desktop processors (compared to server processor cache sizes). To exploit this locality with a simple memory controller this desktop design implemented the following features:

Page-open policy: Open-policy1: the pages are left open until they need to be closed (say due to an access to a different page on the same bank) or Open-policy2: all the open pages are closed after "n" idle cycles, where n is configurable to 0, 8, 16 or 64 clocks.

Non-overlapped scheduling: Since accesses were expected to be largely page-hits (RD-NOP-RD-NOP-RD type of command sequence on the memory channel), to simplify the implementation a non-overlapped scheduling was implemented. But, with this non-overlapped scheduling, if we had a series of page empty accesses instead of page hits, these would appear as (ACT1-NOP-RD1-NOP-ACT2-NOP-RD2-NOP-ACT3-NOP-RD3…) on the channel while with overlapped scheduling we could have done a more efficient (ACT1-ACT2-RD1-ACT3-RD2-ACT4-RD3…) sequence.

No memory rank address interleaving: Highest order address bits were used to map to different ranks. Since significant locality was expected, to facilitate page hits, successive addresses were mapped to the same page. Once the end of the page was reached the successive addresses were then mapped to a page on a different bank on the same rank. This process was continued to cover all the memory on a given rank before going to the next rank. This address mapping simplifies address decoding in a system that supports variable number of ranks.

In-order request processing: As requests arrived from the in-order FSB to the memory controller, the read requests were placed in a read request queue and the write requests were placed into the posted write request queue. Preference was given to service read requests. If the write request queue reached a threshold to indicate the approach of queue full condition or if there was no read requests, write requests were serviced in a burst of four writes. Since the read data response completion on the FSB had to happen in-order, the requests were serviced in-order.

When server benchmarks were run on the real DP server system that used this memory controller design, it was quickly determined that using Open-policy1 resulted in very poor performance due to excessive page misses. It was also observed that Open-policy2 with n=0 lowered page misses significantly and yielded the best results for this memory controller for server benchmarks.

Our simulation study compares this memory controller with Open-policy2 and other features described above (we will refer to it as MC-A) with other memory controller features described below:

MC-A with memory rank interleaving: Rather than use the highest order bits to map to different ranks, significantly lower order bits (such as bits 12 & 13) were used, thereby distributing the addresses in even a small region of memory over multiple banks on multiple ranks rather than on just one rank. Other features of MC-A remain the same.

MC-A with page-close & overlapped command scheduling: We next changed the policy to page-close with auto pre-charge (always automatically close the page on every access,

no explicit pre-charge command used) and also overlapped the row activate (ACT) & column (RD/WR) command scheduling to independent banks. Other features of MC-A remain unchanged.

MC-B: The MC-B memory controller uses a page-close policy with auto pre-charge, overlapped command scheduling, and memory rank interleaving. The request processing remained in-order as in MC-A.

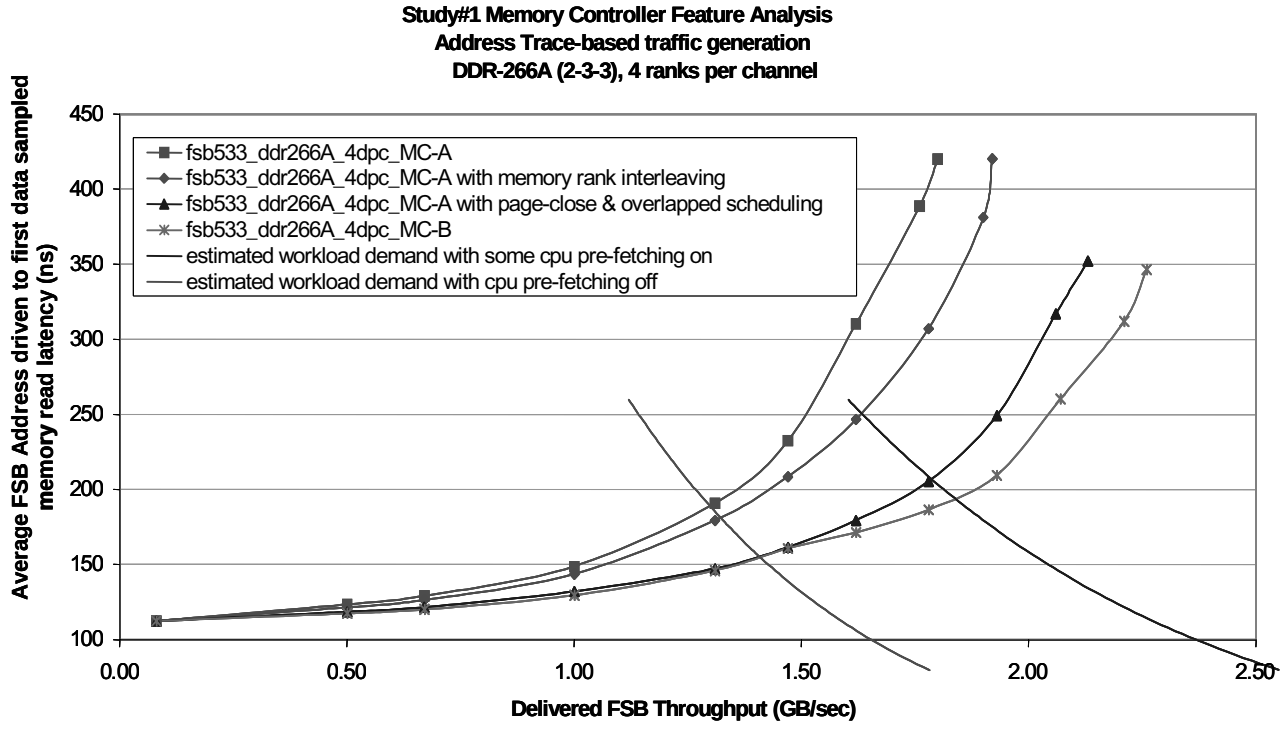
We used addresses and transaction types from FSB traces obtained from a MP server system running a popular web-server benchmark in our simulations for this study. The configuration for this study is an MP server system with in-order IA32 FSB at 133MHz / 533 MT/s and two DDR266A (2-3-3 tCL–tRCD–tRP DRAM device timings) channels operated in lockstep, each with four single rank DIMMs. For each memory controller feature combinations studied, we generated an average read latency vs. throughput curve. Each curve requires multiple simulations, each with a different level of demanded throughput, specified by the transaction inter-arrival rate (we used an exponential distribution for inter-arrival times). As the throughput demand increases from one simulation to the next, the queuing latency experienced by the requests increase, and eventually the maximum sustained throughput capability of the simulated system is reached. Fig. 2 presents the results of this study.

Fig. 2 shows that for this server workload, page-close policy with overlapped scheduling provides huge improvements in terms of latency-vs.-delivered-throughput performance compared to page-open policy with non-overlapped scheduling. Memory rank interleaving also provides significant performance gains (lowers loaded latency and improves max delivered throughput), and from Fig. 2 we see that its gain is additive to that provided by page-close policy with overlapped command scheduling.

With requests coming from multiple sources (multiple processors, IO) and the resulting "localized random access pattern" (random over different small regions of memory) at the memory, page-open policy with non-overlapped command scheduling suffers. For this MP server workload traffic pattern, with MC-A we observed a mix of 53% page-hits, 43% page-misses, and 4% page-empty. For such random workloads marked by a high fraction (43%) of page-miss accesses, page-open policy with non-overlapped scheduling is particularly inefficient resulting in lot of holes / unused cycles on the memory channel.

Rank interleaving distributes even small regions of memory over many more independent banks compared to no rank interleaving and helps reduce page miss probability for such localized random access patterns. For the simulated trace based server workload traffic pattern, on MC-A with rank interleaving we observed a mix of 53% page-hits, 29% page-misses, and 18% page-empty – this shift of a significant fraction of the accesses from being a page-miss access to a page-empty access causes the observed improvement in performance.

Fig. 2 also includes workload demand curves. The nature of these server benchmarks/workload is that when latency is decreased and performance increases the demanded bandwidth increases as well. The workload demand curve captures this effect and shows what the new workload operating point would be if the memory controller moved from features in MC-A to MC-B. Two workload demand curves are shown – one with CPU pre-fetching and the other without. We can see that pre-fetching adds significant load (as observed in [3]) and drives up loaded latency. In this case, we found that the resulting operating latency was too high that benchmark performance with pre-fetching was worse than without. To get the benefits of CPU pre-fetching, memory controller with advanced features as in MC-B that improves the latency-vs.-throughput characteristics was needed.



4.2 Study #2

For our next study we began with MC-B and looked at improving the in-order request processing with a more **intelligent read-to-write switching** memory controller feature. As described earlier, MC-B switches from reads to writes either when the write queue is approaching queue full condition or opportunistically when the read queue is empty. Since the idle cycles in the memory channel for bank conflicts is much larger than the channel turnaround cycle for switching between reads & writes, the idea with intelligent read-to-write switching is to opportunistically switch from reads to writes when the head of the read queue is blocked on a bank conflict, in addition to when the read queue is empty.

The simulation configuration for this study is an MP server system with an in-order IA32 FSB at 167MHz / 667 MT/s and two lockstep DDR266 (2-2-2 DRAM device timings) channels each with 1, 2, or 4 single rank DIMMs per channel. We conducted the study with statistically generated traffic with 2:1 read-to-write mix (representative of many server workloads) and uniformly randomly distributed addresses. We found that if the system had good memory rank interleaving, which distributes even relatively small regions of memory over multiple banks on multiple ranks available in the system rather than on just one rank, and used auto pre-charge page-close policy, similar relative performance impact was observed with this manner of statistically generated traffic compared to using MP server FSB trace-driven traffic such as the one used in the earlier study. The advantage with statistical traffic generation driven by parameters obtained from trace analysis is that it required far less simulation time & disk space.

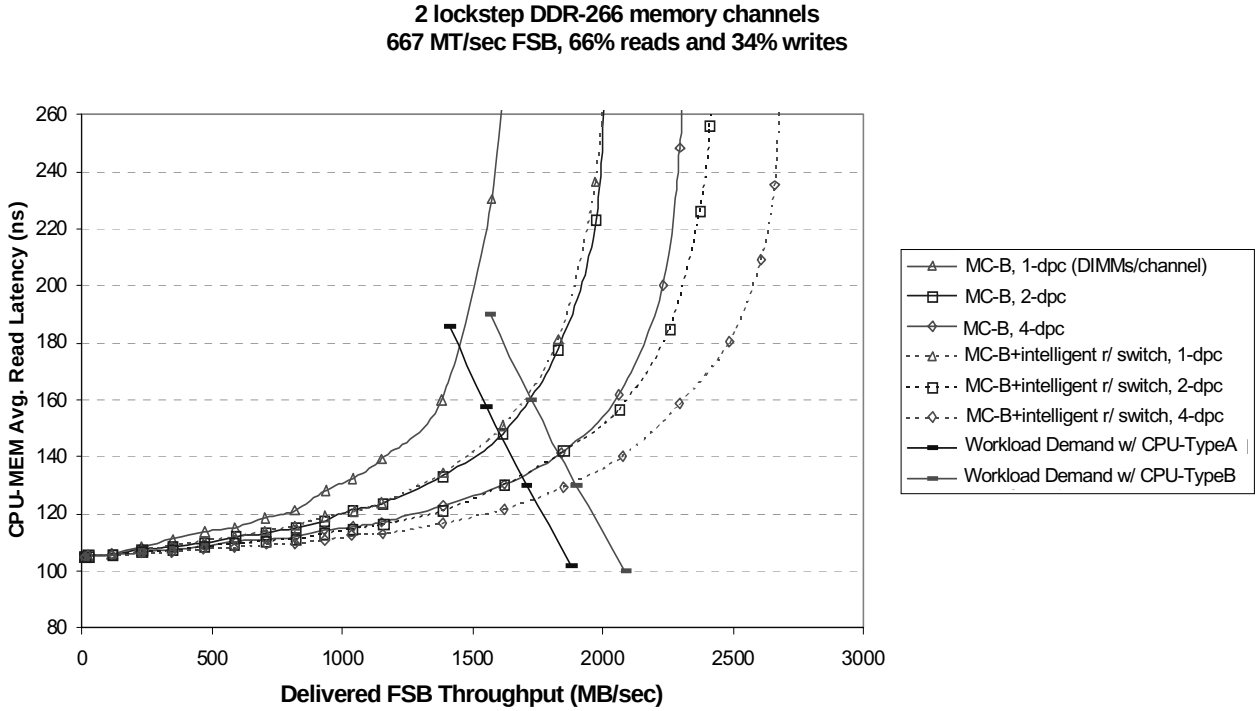
Fig. 3 illustrates the performance benefits from the intelligent read-to-write switching feature. We observe from the results that the intelligent read-to-write switching feature provides significant performance improvements – of the same order as doubling the number of interleaved ranks. The figures also include workload demand curves for two different types of CPU (CPU micro-architecture, cache size) that illustrates how the workload operating point moves with different number of ranks and with intelligent read-to-write switching.

4.3 Study #3

In the previous two studies we evaluated systems with in-order IA32 FSB. In this study we investigate a system with an out-of-order IPF FSB, specifically the single-bus version of the Intel 870 ([2]). This system consists of an IPF FSB at 200MHz / 400MT/s (6.4GB/s peak BW) and four lock-step 1.6GB/s memory channels connected to the MCH component (named SNC, Scalable Node Controller, in Intel 870). A DDR Memory Hub (DMH) on each memory channel is connected to two DDR200 channels that are operated independently. Each DDR channel can support up to 4 DIMMs.

In this study, we compare the performance of the system described above that has an O-o-O FSB with an in-order memory controller (as in MC-B described earlier with overlapped command scheduling, auto pre-charge page-close policy, memory rank interleaving, and in-order request processing) vs**. a memory controller that implements request re-ordering feature, i.e, an out-of-order memory controller** (MC-B but with out-of-order request processing).

With in-order request processing, if the head of the read request queue is blocked on a bank conflict the whole read queue is stalled increasing the latency of all the read requests behind as well. When the FSB is in-order, the later read requests, even if



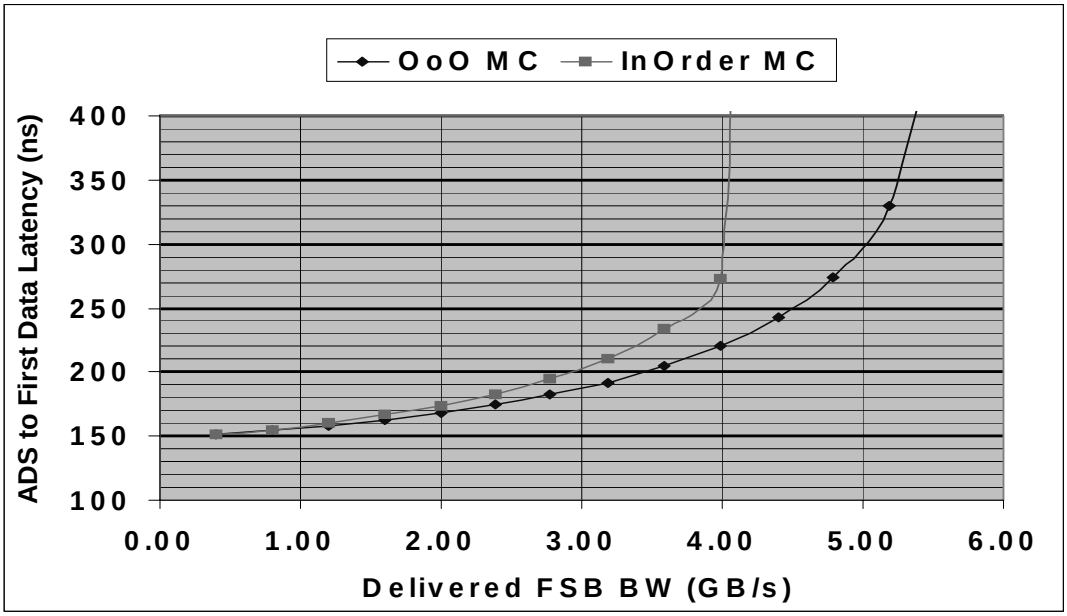
they have their data ready, could not be completed on the FSB until the earlier blocked request is completed; however, with an O-o-O FSB this constraint is removed. The O-o-O memory controller in Intel 870 exploits the O-o-O FSB by looking at up to 4 read requests that target independent banks to find a conflict-free request to schedule. As in MC-B, preference is given to service read requests. The memory controller switches to servicing writes: (1) opportunistically when the read queue is empty and there are at least 4 outstanding write requests, and in this case, bursts at least 4 writes before switching back to processing reads if read requests have become available, or (2) when the write queue approaches queue full condition, and in this case, bursts at least 16 writes before switching back to processing reads if read requests have become available. Since with request re-ordering in the read queue, the probability of the entire read queue being blocked on bank conflicts is small (only happens when all available read requests all have bank conflicts with previously scheduled requests), intelligent read-to-write switching feature was not needed. The O-o-O memory controller does request re-ordering when processing the write queue as well to minimize holes on the memory channel created by bank conflicts, there by improving the efficiency and the maximum sustainable bandwidth.

For this study, we simulated the Intel 870 single-bus configuration with 4 DIMMs on each DDR channel. Fig. 4 presents the benefits of the O-o-O memory controller in terms of latency-vs.-throughput curves. The simulations for this study were also done with statistically generated random address traffic with a 2-to-1 read-to-write mix that is representative of many server workloads. The results show that the request re-ordering feature results in significant loaded latency reduction (for example, >10ns when the delivered FSB BW is >2GB/s).

For any read-to-write mix, the request re-ordering also improves the system's maximum delivered / sustainable bandwidth. To illustrate this we show in Fig. 5 the results of an experiment where we had FSB bus agents requesting at the rate of 4GB/s and IO traffic (directly injected into the MCH to introduce the load as we do not have the Intel 870 specific IO subsystem modeled in this environment) requesting at the rate of 2.4GB/s with various read-to-write transaction mix. On this system we observed 12% to 30% maximum sustained bandwidth improvements with the O-o-O memory controller compared to the in-order memory controller. Similar order of bandwidth improvement has been reported in a memory request re-ordering study for the Imagine Stream Processor used in media processing systems ([7]).

4.4 Study #4

In this final study we investigated a **delayed write scheduling feature** to further optimize loaded read latency with an O-o-O memory controller. When the read queue is empty, the memory controllers described in earlier sections immediately switch to processing the write queue opportunistically and start sending row activate (ACT) operation immediately on the memory channel with the intent of keeping the memory channel busy. However, what this does is that if one or more read requests come immediately after the memory controller makes the switch to processing write requests, these read requests are blocked when they could have been issued, there by increasing their latency. With the delayed write scheduling feature, the memory controller determines the last possible cycle from the current cycle when the write request's row activate operation must be started to avoid unnecessary idle cycles in the memory data channel and waits until then to make the switch to processing writes. If read requests arrive during this waiting period, the memory controller does not switch to processing writes but processes the reads instead, lowering the average loaded system read latency which helps improve many server workload performance. If no read requests arrive during this waiting period, the memory controller switches to scheduling writes and starts a row activate operation at the last possible cycle that avoids any unnecessary idle cycles in the memory data channel.



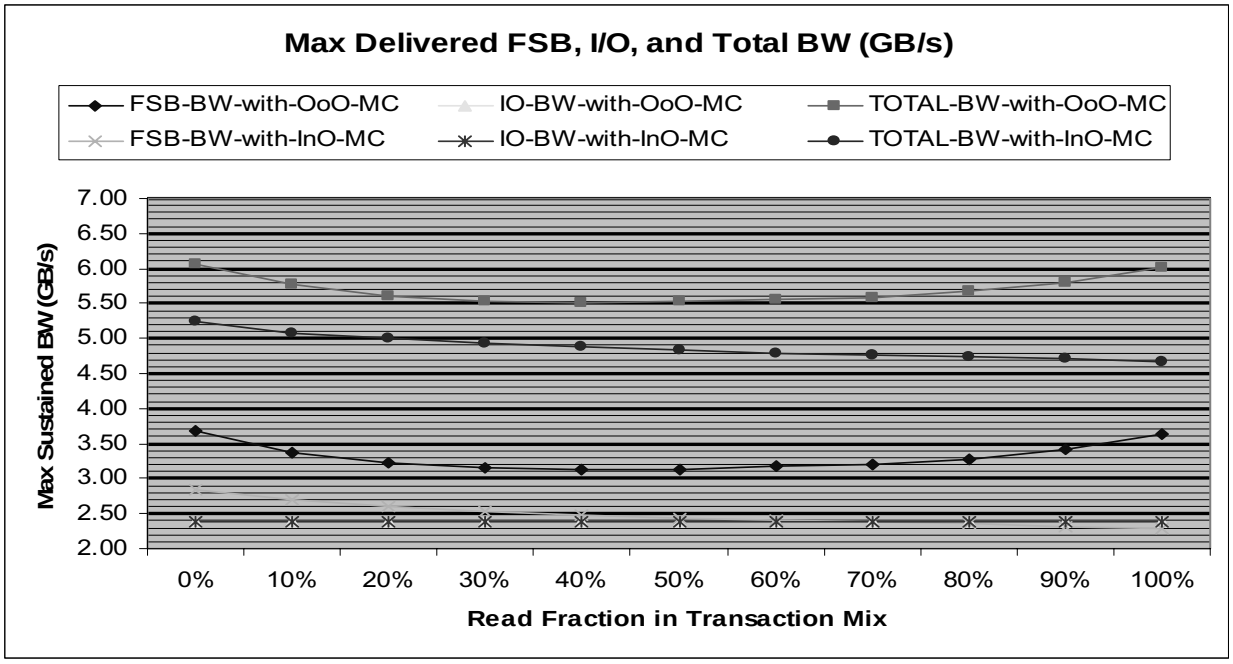
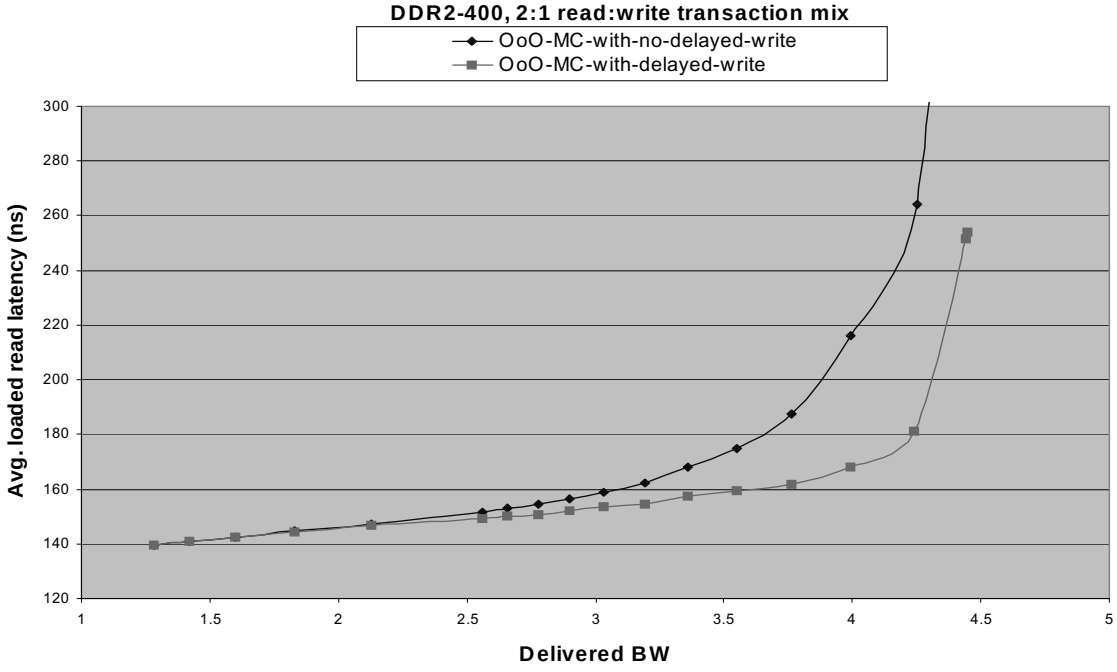


Figure 5 Memory Controller Feature Impact Study#3 Maximum Sustained BW Results

The simulation configuration for this study was an O-o-O server system with a pair of lockstep DDR2-400 channels with 4 ranks per channel. This study also used statistically generated random address traffic with 2-to-1 read-to-write transaction mix. The results are presented in Fig. 6. The results show that the delayed write scheduling feature provides significant read latency improvements and keeps the loaded read latency lower and relatively more flat even as the delivered bandwidth approaches maximum sustainable bandwidth.

5. SUMMARY

With processor performance improvements continuing to far outpace the memory performance improvements, memory controller designs are under pressure to extract every possible ounce of performance from the memory subsystem to minimize processor stall times. Given this scenario, this paper studied and presented a number of memory controller features and their performance impact in MP server environments. We found that for MP systems with an in-order FSB running server workloads,



page-close policy with overlapped scheduling provides much lower loaded read latency than a page-open policy with non-overlapped scheduling. Our studies showed that memory rank interleaving can provide considerable loaded read latency gains and it is additive to the gains obtained with overlapped scheduling and page-close policy. We also demonstrated that intelligently switching from reads to write opportunistically when the read queue is blocked by bank conflicts results in improving the performance to about the same order as doubling the number of interleaved ranks. An O-o-O memory controller used with an O-o-O FSB was shown to have significantly reduced loaded read latencies compared to an in-order memory controller. Moreover, without memory request re-ordering the maximum sustainable system bandwidth was also shown to be considerably limited in such a system. The paper also presents a delayed write scheduling feature with an O-o-O memory controller that helps in considerably lowering average loaded read latency making the latency-vs.-throughput curve more flat over a wider throughput range.

In the future, we plan to model and analyze memory subsystems based on the newly introduced FB-DIMM technology ([9]), compare its performance with native-DDR/DDR2 based systems, and explore memory controller feature optimization opportunities with the FB-DIMM technology. We also plan to investigate adaptive hybrid page-open/close policies in MP server environments.

6. ACKNOWLEDGMENTS

We would like to thank a number of people whose contributions made this paper possible. First, we acknowledge Mark Heap and David C. Lee for their contributions to the development and study of the delayed write scheduling memory controller feature. Steve Kulick, Suneeta Sah, and Varin (Joe) Udompanyanan were key members of the Intel 870 memory controller definition team and contributed to its performance analysis in that capacity. We also thank Brian Bennett and Eric Dahlen for their review and feedback for the first two studies presented in this paper. We thank Olivier Maquelin and several others who have contributed to our model development infrastructure over the years. Jeff Wilder, Sundaram Chinthamani, Sivakumar Radhakrishnan, David C. Lee, and Jason Fung developed the models beyond the memory subsystem model to make up the full system model used in this work. Sundaram Chinthamani also provided the data for some of the workload demand curves illustrated in a few charts presented in this paper. We also thank Howard David for providing us DDR/DDR2 timing information and other memory channel protocol details. Finally, we would also like to thank our tracing team for providing the traces we used in the study.

7. REFERENCES

[1] Alakarhu, J.; Niittylahti, J., “A comparison of pre-charge policies with modern DRAM architectures”. In Proceedings of the 9th International Conference on Electronics, Circuits and Systems (Sept. 2002), vol. 2, pp. 823 – 826.

[2] Briggs, F.; Cekleov, M.; Creta, K.; Khare, M.; Kulick, S.; Kumar, A.; Lily Pao Looi; Natarajan, C.; Radhakrishnan, S.; Rankin, L., “Intel 870: a building block for cost-effective, scalable servers”. IEEE Micro (March -- April 2002), vol. 22, issue. 2, pp. 36 – 47.

[3] Burger, D.; Goodman, J.R.; Kagi, A., “Limited bandwidth to affect processor design”. IEEE Micro (Nov. -- Dec. 1997), vol. 17, issue. 6, pp. 55 – 62.

[4] Cuppu, V.; Jacob, B., “Concurrency, latency, or system overhead: Which has the largest impact on uni-processor DRAM-system performance?”. In Proceedings of the 28th Annual International Symposium on Computer Architecture (June -- July 2001), pp. 62 – 71.

[5] Cuppu, V.; Jacob, B.; Davis, B.; Mudge, T., “High-performance DRAMs in workstation environments”. IEEE Transactions on Computers (Nov. 2001), vol. 50, issue. 11, pp.1133 – 1153.

[6] Hennessy, H.; Patterson, D.A., “Computer Architecture: A Quantitative Approach”. Second edition. Morgan Kaufman, 1996.

[7] Rixner, S.; Dally, W.J.; Kapasi, U.J.; Mattson, P.; Owens, J.D., “Memory access scheduling”. In Proceedings of the 27th International Symposium on Computer Architecture (June 2000), pp.128 – 138.

[8] Wei-Fen Lin; Reinhardt, S.K.; Burger, D., “Reducing DRAM latencies with an integrated memory hierarchy design”. In Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (Jan. 2001), pp. 301 – 312.

[9] Vogt, P., “Fully Buffered DIMM (FB-DIMM) Server Memory Architecture: Capacity, Performance, Reliability, and Longevity”. Intel Developer Forum (Feb. 2004), Session OSAS008.