Towards A Bandwidth Friendly Memory System

Jerry B. Backer
supervisor: Mohamed Zahran
Department of Electrical Engineering
The City College of the City University of New York
New York, NY 10031
{jbacker00,mzahran}@ccny.cuny.edu

ABSTRACT

In modern computer systems, the use and availability of memory bandwidth, particularly off-chip bandwidth, is becoming increasingly important. Current design techniques and technology trends such as the migration to the multicore technology may expose the bandwidth limitations by putting more pressure on the memory system. The lack of memory bandwidth results in processor stalls which negatively influence the overall performance. This situation becomes even more critical for memory sensitive applications. In this project, we explore how hardware and software tendencies have impacted memory bandwidth availability. In addition, we propose a new technique, Bandwidth Friendly (BF) writes, a low overhead write method aimed at leveling the off-chip bandwidth by managing it around a budgeted bandwidth. We focus mainly on the writes as they are they represent the bulk of traffic between processor and the off-chip memory. Using the SpecCPU 2000 benchmarks, the BF writes method improves the number of instructions per cycle (IPC) by as much as 12%.

1. BACKGROUND

Due to the disparity between memory and processor speeds, the memory system is not able to keep up with the demands of the processor. To reduce this gap, designers have used techniques to either reduce or hide large memory access latencies. The success of many of these techniques however may expose the bandwidth limitations of the system [2] by either increasing the overall traffic, causing pollution in the memory system, or by requiring more memory operands per unit of time, resulting in contention. Software prefetching[5] for example is aimed at predicting data before requested by the processor. However, if the predicted data is incorrect, the cache will be polluted with needless data. Moreover, in prefetching, data in the memory system may be evicted before their use to make space for the predicted block; this causes an extra cache miss, increases the overall traffic and puts more pressure on the memory system. In the case of a lockup-free cache, in order to hide memory latency, the processor does not stall when a cache miss occurs; instead it continues the execution of instructions independent of the requested data until it is again needed by another instruction. With the issue of multiple memory requests, this technique increases the queue delay of the memory system which also increases its bandwidth requirements. With the advancement in processor technology and the reduced size of transistor size, the design of multiple processor cores on one chip has gained popularity. With the movement to multicore, special attention needs to be paid to the use and management of off-chip bandwidth. Since the on-chip caches can't hold the working sets of the executing threads, DRAM bandwidth becomes critical [9]. With two or more processing cores using the same system bus and sharing the DRAM bandwidth, conflict for bus and memory access will lead to bus congestion. In the case that one of the processors is stalled, all other related processors will be influenced and the overall performance will decrease. Moreover, the use of external peripherals such as audio and video cards, graphics accelerators, and other I/O devices will put more pressure on the memory for access and exposes its bandwidth limitations. Furthermore, the development and use of complex 3-D graphics and multimedia applications, along with High Performance Computing (HPC) applications will likely increase the bandwidth requirements of computer systems as these programs require large sets of matrices of floating point data. These applications exploit the cache ineffectiveness in managing scientific- like data [1].

In Section 2, we present results obtained from analyzing the off-chip traffic distriubtion using the SpecCPU 2000 and its impact on the bandwidth. We also present related work accomplished to improve memory system bandwidth. Section 3 discusses our proposed technique, Bandwidth Friendly writes aimed at improving the traffic distribution and bandwidth

usage. This section also presents the results obtained using our technique. The last sections conclude with a summary of the research work, the progress made and the future works planned to improve on the overall efficiency and scalability of the technique

2. MOTIVATION

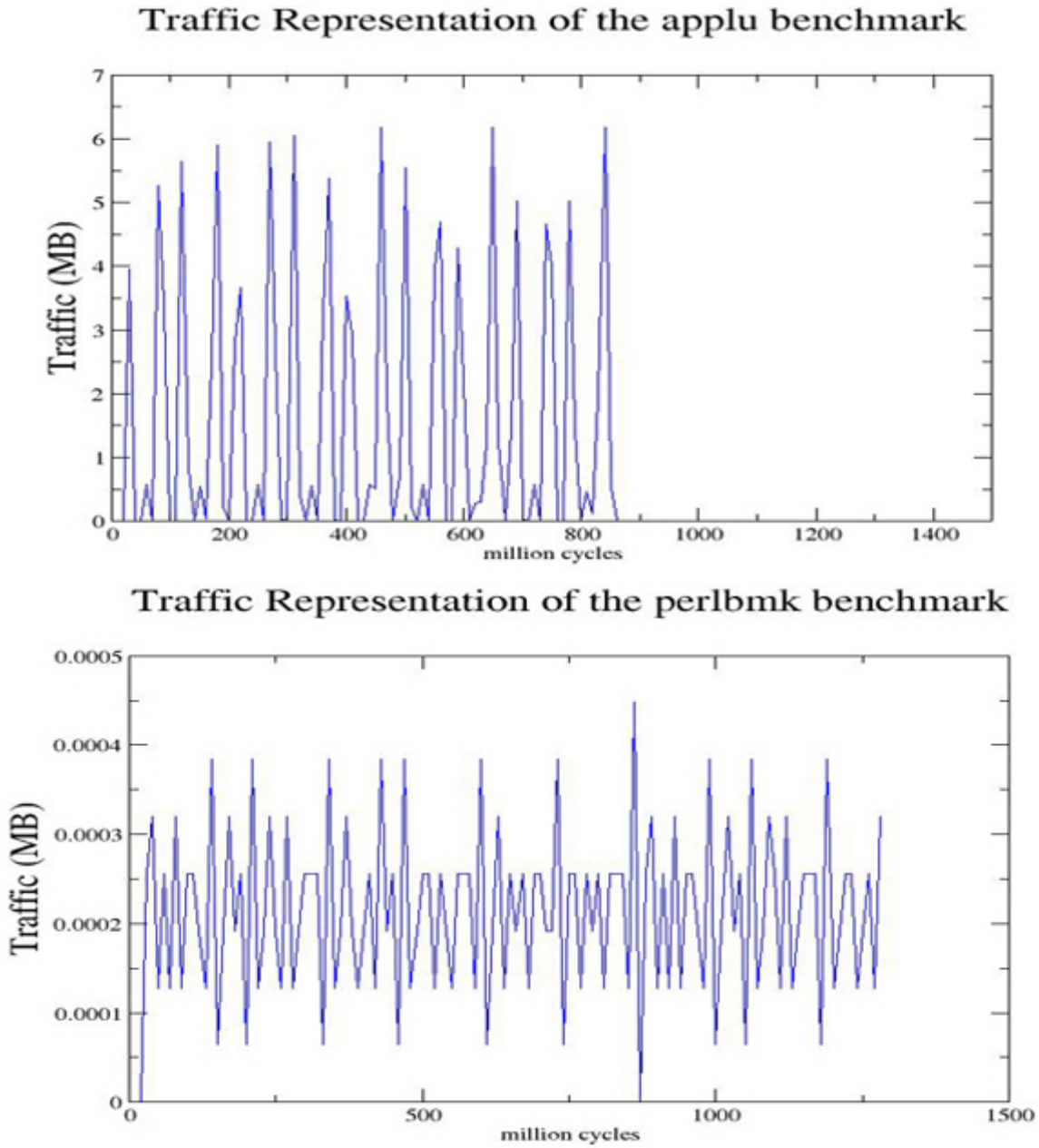

Figure 1: Off-chip Traffic Representation (from CPU chip to RAM) of apply and pelbmk benchmarks

Due to power consumption and packaging cost, the number of processor pins are not very scalable and off-chip memory bandwidth becomes a scarce resource. In addition, throughout the execution of a program, the memory traffic is not leveled and does not make good use of the available bandwidth. There are periods of intensive memory traffic, during which bursts of data are transferred and the memory bandwidth may be limited. During other phases, there is little traffic, the bus stays idle and the available bandwidth is not used. This uneven distribution of traffic does not use the memory bandwidth effectively and bandwidth is exposed.

2.1 Uneven Traffic Distriubtion

For our simulations, we use the Spec2000 benchmarks as they are representative of current applications. Using the sim-outorder simulator from the SimpleScalar, figure 1 shows the off-chip traffic from L2 to the main memory. As the program runs, its bandwidth requirements are periodic, where the application has idle bus periods and other periods of high off-chip traffic. The traffic peaks and spikes sections illustrated in figure 1 are usually due new data sets needed by the application, causing the write-back of old data sets. With the limited number of ports off chip, the high traffic bursts result in bus contention and processor stalls, hence performance loss. Throughout low traffic periods, the system bus is idle and the available bandwidth is wasted.

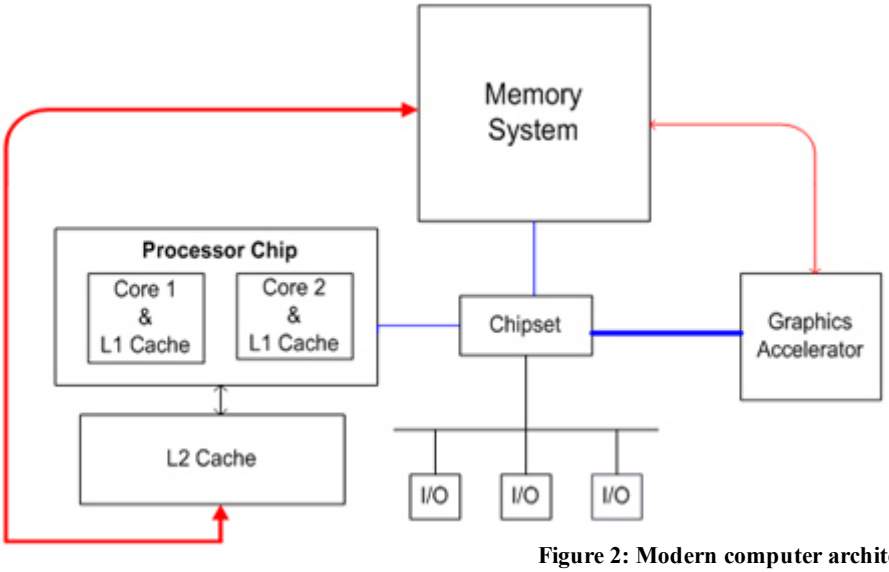

Figure 2: Modern computer architecture design

In modern computer architecture design (figure 2), the uneven traffic distribution becomes more significant. During high traffic periods, the processor cores are competing for bus access and the limitations of the memory bandwidth are exposed. This will lead to one or more of the cores being stalled, waiting for access.Furthermore, peripherals and I/O devices may also have memory requests, putting more pressure on the off-chip bandwidth.

2.2 Related Work

Previous work has been documented to improve off-chip memory bandwidth limitations by either increasing its availability or by reducing the bandwidth needed. Drost et al. [3] presented a technology, proximity communication, which allows wireless communication between chips. By replacing the standard copper wires with capacitive couplings, the transfer speed and the bandwidth availability increase. Patterson et al. propose the unification of the processor and the DRAM into a single chip, the IRAM. In such case, the memory will be able to operate at processor speed, increasing the bandwidth 100-fold [8]. Another technique proposed by Milenkovic and Milutinovic et al. [6] is cache injection which combines the good properties of data prefetching and data forwarding in order to reduce the miss ratio, which will decreases the overall traffic and indirectly reducing the bandwidth requirements of the system. A technique conducive to chip multiprocessors (CMP) is presented by Zang et al [11]. They propose to slice the L2 cache and give each processor core a private L2 instead of having the traditional shared L2. They also propose a new cache management policy, victim replication, which allows a private L2 cache to save an old block in its L2 neighbor instead of evicting that block to the lower level (usually main memory). In case that block is again requested, the penalty of accessing the L2 neighbor is less than a cache miss and off-chip memory access. Since this technique reduces off-chip access, bandwidth requirements decrease.

Other proposed techniques include wider bus widths to directly increase the bandwidth availability [10], data and address compression [4] None of the techniques above tackle the problem of leveling the unbalanced traffic distribution, which does not make good use of the bandwidth. Our study focuses on improving the off-chip bandwidth utilization by reducing the number of peaks and bursts of the unfriendly traffic. By leveling the traffic around the available bandwidth, we will in

effect reduce the number of bus contentions and processor stalls caused by bandwidth limitations during memory sensitive periods of the application..

3. PROPOSED TECHNIQUE

The irregular off-chip traffic does not utilize the available bandwidth efficiently. In order to reduce the traffic peaks and make it bandwidth friendly, we propose a technique that manages the traffic and balances it out throughout the execution of a program. Previous work in traffic re-distribution was done by Lee et al.[6] However, our work takes in consideration the available bandwidth (and its time-varying approach) to manage the total traffic around it.

3.1 Bandwidth Friendly Writes

The Bandwidth Friendly (BF) writes technique is aimed at redistributing the off-chip traffic and keeps it as leveled as possible around a budgeted bandwidth. The budgeted bandwidth is assigned from the average traffic throughout execution. In order to adapt to the time-varying nature of memory bandwidth usage, the BF writes technique uses a dynamic checkpoint (a position during execution) that when reached, flushes the dirty Least Recently Used (LRU) blocks from L2 cache to the memory. The current bandwidth is then compared to the budgeted bandwidth. The current bandwidth is calculated as the total traffic between L2 and memory over the total execution time. If the actual bandwidth is greater, the application is likely in a memory intensive period causing the peaks and bursts. In that case, the checkpoint for the next set of LRU blocks flushing is delayed, effectively postponing traffic which will bring down the peaks. In the case of the current bandwidth being lower than the budgeted bandwidth, the available bandwidth is being wasted. Therefore, the checkpoint for the following LRU blocks flushing is forwarded and the writes are down earlier than scheduled, making use of the available bandwidth and bringing up the low traffic periods around the budgeted bandwidth. With the low probability of LRU blocks being referenced before eviction, our technique does not induce additional cache misses.

3.2 Experimental Methodology

Using the sim-outorder simulator, we implemented a synthesized traffic generator to imitate the requests of external peripherals to the main memory. In addition a write-buffer was implemented. Table 1 describes the baseline configuration used as the CPU and memory system. This model is used to imitate current computer trends. As a proof of concept, some of the benchmarks from the Spec2000 suite were used. 1 billion instructions were skipped throughout the simulation.

Table 1: Baseline computer system configuration

IL1 Cache	32 KB, 32 B block size, 1 cycle access latency; LRU replacement, write-through
IL2 Cache	32 KB, 32 B block size, 1 cycle access latency; LRU replacement, write-through
UL2 Cache	512 KB, 64 B block size, 4 cyclse access latency; LRU replacement, write-back
Memory	500 cycles access latency
Bus	16 B width, 1:1 processor: bus cycle differential
CPU	Single core, out-order, 2 INT ALUs, 1 INT multiplier/divider, 2 FP ALUs, 1 FP multiplier/divider

3.3 Results

In figure 3 below, the off-chip traffic of the mcf benchmark is presented for both the baseline simulation and that of using the BF writes method. Using the BF writes, the number of peaks and bursts is considerably less. Moreover, the overall traffic is more leveled and the number of instructions per cycle (IPC) has increased by 12%, resulting in faster execution of the application.

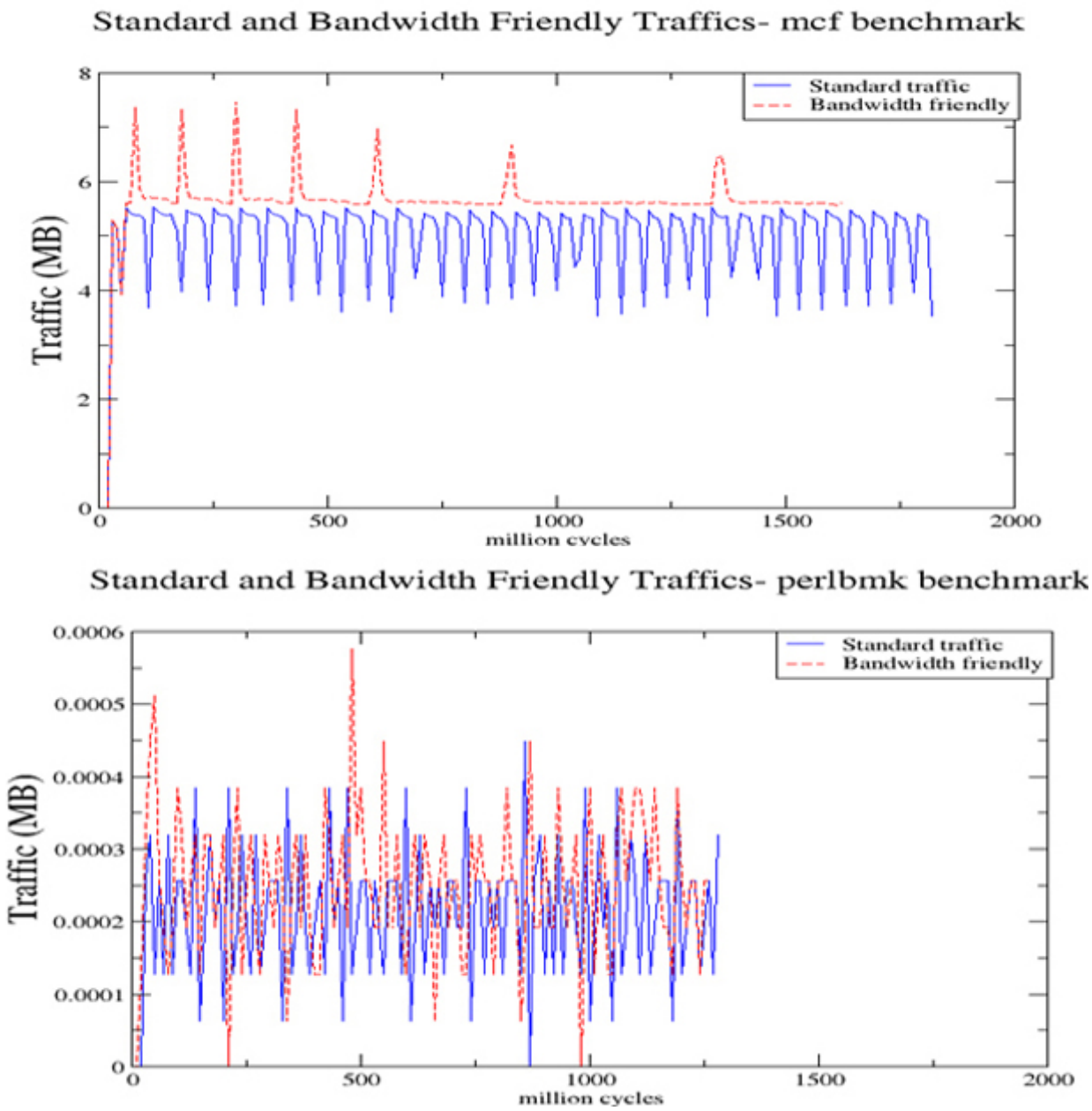

Figure 3: Comparison of baseline traffic and bandwidth friendly traffic - mcf and perlbmk benchmarks

Using the BF writes technique in the perlbmk benchmark, there is not much change in the overall traffic. However, the fraction of time the load-store queue (LSQ) was full has decreased by 19.98%. Therefore, the queueing delay decreases and blocks are written down at a faster rate. This is due to a decrease in bus contention and overall increase in bandwidth availability. In general, the average occupancy rate for the LSQ decreased by 5.77% for the perlbmk benchmark.

4. CONCLUSION

- The uneven off-chip traffic does not utilize memory bandwidth efficiently
- During low traffic periods of an application, the bandwidth is not used, which puts more pressure of the bus and memory system during memory critical stages
- We propose a new technique aimed at leveling the traffic around a budget bandwidth

- Our technique periodically flushes the dirty LRU blocks to the main memory during low traffic periods and delays the flushing during traffic peaks and bursts.

5. FUTURE WORK

For future work, we plan on exporting this technique to the multicore technology and adhere it to more current hardware and software trends. We will implement and simulate the BF writes using the SuperEScalar (SESC), a superscalar, out of order simulator with multicore implementation. This also involves using benchmarks known to stress off-chip traffic. We will also conduct more sensitivity analysis to come up with the optimal method for the checkpoint.

6. REFERENCES

[1] D. C. Burger, A. Kägi, and J. R. Goodman. The Declining Effectiveness of Dynamic Caching for General-Purpose Microprocessors. In Technical Report 1261, Computer Sciences

[2] D. Burger, J.R. Goodman and A. Kagi. Memory Bandwidth Limitations of Future Microprocessors. In Proceedings of the 23rd Annual International Symposium on Computer Architecture, pages 79-90, May 1996.

[3] R. J. Drost, R. D. Hopkins, and I. E. Sutherland, Proximity Communication, Sun Microsystems, Inc. Mountain View, California, U.S.A.

[4] M. Farrens and A. Park. Dynamic Base Register Caching: A Technique for Reducing Address Bus Width. Proceedings of the 18th Annual International Symposium on Computer Architecture, 19(3); pages 128-137, May 1991

[5] J. W. C. Fu and J. H. Patel. Data Prefetching in Multiprocessor Vector Cache Memories. In Proceedings of the 18th Annual International Symposium on Computer Architecture, pages 54–63, May 1991.

[6] H.S. Lee, G.S. Tyson and M.K. Farrens. Eager Writeback- a Technique for Improving Bandwidth Utilization. In Proceedings of the 33rd ACM/IEEE International Symposium on Microarchitecture, 2000.

[7] A. Milenkovic and V.M. Milutinovic. Cache Injection: A Novel Technique for Toleration Memory Latency in Bus-Based SMPs. In Proceedings from the 6th International Euro-Par Conference on Parallel Processing, pages 558-566, 2000.

[8] D. Patterson, T. Anderson, N. Cardwell, R. Fromn, K. Keeton, C. Kozyarakis, R. Thomas, and K. Yelik. A Case For Intelligent RAM: IRAM. In IEEE Micro. April 1997

[9] N. Ra_que, W. T. Lim, and M.Thottethodi. Effective Management of DRAM Bandwidth in Multicore Processor. In the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007), September 2007

[10] L. Rudolph and D. Criton. Creating a Wider Bus using Caching Techniques. Proc. 1st Int'l Symposium on. High Performance Computer Architecture. IEEE Comp Society Press, pages 90-99. 1995.

[11] M. Zhang and K. Asanovi´. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. In the Proceedings of the 32nd International Symposium on Computer Architecture, June 2005.