Memory Systems and Memory Latency

Lecture #7: Wednesday, April 19, 2000
Lecturer: Brian Towles
Scribe: Shuaib Arshad

The research papers discussed prior to this session dealt with the problems related to the instruction fetch. Starting from this session the focus of discussion was shifted towards another very important part of the computer organization - Memory access and latency of memory components.

1 Introduction

There has been a remarkable increase in the speed of execution of instructions which is performed by the processor. But this speed is greatly dependent upon the speed at which memory components can supply instructions and data to the processor. Unfortunately the access time of memory components, which has been increasing at the rate of around 25% per year, hasn't been improving as fast as the processor performance whose rate of improvement has been 60% per year and hence the memory access penalty is considerably increasing with each enhancement in the processor architecture.

In this session we discussed the following three papers which address the issue of memory latency:

- Improved Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers
- Missing the Memory Wall : The Case for Processor/Memory Integration
- Memory Access Scheduling

2 Improved Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers [1]

Cache is an important part of the currently used memory hierarchy and this paper proposes new hardware techniques to improve its performance.

Before discussing the proposed techniques the author explains the baseline design used for the evaluation of the proposed enhancements which include a processor with 1,000

MIPS peak instruction issue rate, separate 4KB L1 instruction and data caches with 16B lines, and unified 1MB L2 cache with 128B lines. The miss penalties are assumed to be 24 instruction times for L1 and 320 instruction times for L2 cache. The author also mentions that off-chip cycle time is 3-8 times longer than the instruction issue rate.

As the title also mentions all the proposed enhancements are meant for improving the performance of direct-mapped cache and the reason for only considering direct-mapped cache is its faster access as compared to set-associative cache in which in addition to line access, time is spent on predicting the set to access.

During the discussion a reference was also made to an important paper by Harold Stone [4]. This paper presents a model for the estimation of the influence of the cache size on the reload transients. According to the author cache could be visualized as an array of sets with each set having a different number of elements depending upon the number of lines that can be placed at a particular location in the cache.

During a memory reference, a conflict miss occurs when it happens to map to a cache line already occupied by a different line. This results in replacement of the cache line. If a reference is made to the cache line which has just been evicted by the previous reference, the line is brought back form L2 cache or main memory incurring a penalty. According to the analysis of the data presented by the author, on average 39% of L1 D-cache misses and 29% of L1 I-cache are conflict misses.

Jouppi has proposed 3 different enhancements for the direct mapped caches to reduce these conflict misses:

2.1 Miss Cache

Miss cache is a fully associative memory with a capacity of 2-5 lines and is placed between L1 cache and the access port of the L2 cache. Each time L1 cache is probed, the Miss Cache is also probed. In case of miss both in L1 and the Miss Cache, the line is brought from L2 or main memory and is placed in L1 cache as well as in Miss Cache using LRU replacement policy. In case of a miss in L1 and a hit in Miss Cache, the line is replaced into L1.

Two problems were discussed regarding this organization of Miss Cache:

1. Miss Cache isn't considerably effective if the data being brought in after a miss is greater in size as compared to its capacity.

2. This technique introduces redundancy because the same data is being placed in L1 cache and the Miss Cache.

Analyzing the performance of the system with miss cache added to the baseline specs, it is clear that miss cache removes a large number of conflict misses even with only 2 entries. The major sources of conflict misses in case of I-cache are procedure calls which are not very frequent. For 4KB D-cache a miss cache of 2 entries can remove 25% of conflict misses on average, or 13% misses overall. If the number of entries is increased

to 4, 36% of conflict misses can be removed, or 18% of the misses overall. Increasing the number of entries further does provide any significant improvement.

2.2 Victim Cache

Considering the problems mentioned above, Victim Cache is a better memory structure. On a miss when data is brought into L1 cache, the line being evicted form L1 cache is placed in the Victim Cache instead of the line brought into L1. This removes the data redundancy and potentially provides higher hit rate as compared to miss cache.

Analysis of the resulting performance improvement provided by Miss Cache and Victim Cache using the graphs in Figure 3-3 and Figure 3-5 of the paper, shows that D-cache has a higher performance improvement as compared to I-cache. The reason, as discussed, is that due to the sequential nature of instruction access, the conflicts are spaced out more as compared to the data access conflicts.

Analysis of the results for the victim cache proves that Victim Cache is more effective in removing the conflict misses as compared to Miss Cache. Even with 1 entry, Victim Cache removes a significant number of conflict misses.

Victim Caches are not only being placed for improving L1 cache performance but also for L2 caches. A good example is HP PA-RISC architecture which include a Victim Cache for the large sized L2 cache.

Also mentioned by Prof. Dally was the Inclusion Property of caches. According to the Inclusion Property, L2 cache should have at least the same size and number of lines as L1 cache and also same replacement policy. This property has a significant importance in case of multiprocessors.

2.3 Stream Buffers

If the memory access pattern is predictable, a number of compulsory misses can be reduced using a technique called Prefetching. This techniques utilizes different algorithms:

- Prefetch always - which brings in the next line whenever a memory access is made. This is not possible due to bandwidth limitations.
- Prefetch on miss - counts on the sequential nature of accesses and so brings in 2 lines instead of 1. This technique performs quite well on instructions and considerably on data.
- Tagged prefetch - brings in the next line whenever a line in L1 cache is accessed for the first time. This technique isn't very effective as the memory access latency is quite large.

When a miss occurs, the Stream Buffer begins prefetching successive lines starting at the miss target. The missed line is placed in L1 cache while the lines following it are

placed in the Stream Buffer. By doing this cache pollution is avoided. Stream Buffers are also used to fetch from L2 cache and this is done to hide the L2 access latency.

A limitation of stream buffers is encountered when dealing with arrays e.g. adding individual elements of array and storing them in another array. In this case flushing of stream buffer will be frequent and so performance improvement will be small. Multi-way stream buffers have been proposed as a solution to this problem. Multi-way stream buffers store cache lines belonging to different streams and can greatly reduce the mentioned problem. The usefulness can be seen from the fact that one way stream buffer removed 25% of the data cache misses on the simulated benchmarks while multi-way stream buffers removed 43% of the misses showing 2 times performance improvement.

This papers has proposed very practical techniques for improving memory system performance which are being utilized successfully in the current microprocessors.

3 Missing the Memory Wall: The case for Processor/Memory Integration [2]

This paper proposes a novel microprocessor architecture which uses a simpler and less powerful CPU, tightly integrated with advanced memory technology, instead of using complex, large superscalar CPU with multilevel memory hierarchy.

As discussed, the development of complex memory hierarchy which includes several levels of cache has led to a fallacy. This fallacy is related to the use of SPEC92 benchmarks for the performance evaluation of the memory system. If the cache is large so that the complete benchmark fits inside, then the evaluation done will not be applicable to the current applications e.g. CAD programs, databases or scientific applications which are much larger in size.

The above mentioned fact can be seen in Figure 2 of the paper. The SS-10/61, though has a superior SPEC92 performance shows a huge memory access latency in case of an array access with strides of 256 as compared to the latency shown by SS-5 which has an inferior SPEC92 performance.

The basic idea proposed by the paper is to place a simpler CPU inside a memory on the same die. The CPU proposed for this purpose is MIPS 4300i which has a 5 stage pipeline.

The cache line size for the proposed architecture is 512 bytes which is much larger as compared to the line sizes currently being used. It is important to note that this 512 byte cache line is fetched from DRAM in one cycle. This has been made possible by the memory arrangement of the DRAM into 16 banks each capable of transferring 32 bytes in one cycle. And as the processor is on the same die as the DRAM, there is no problem of limited number of pins for the transfer 512 byte cache line.

3.1 Uniprocessor Performance

The performance was evaluated using SPEC95 benchmark suite (despite the fallacy discussed earlier). Each benchmark program was compiled for SPARC V8 architecture using SunPro V4.0 compiler suite, according to the SPEC base-line rules, and then executed using a simulator derived from SHADE.

Analysis of I-cache miss rates was done by comparing it to the miss rates of a conventional direct mapped 8KB 32B line cache, 16KB direct mapped 32B line cache, 64KB direct mapped 32B line cache and 256KB direct mapped 32B line cache. The proposed cache had significant performance advantage over conventional L1 caches with 32B lines. For almost all the applications the proposed cache has a lower miss rate than conventional I-caches of over twice the size. This reduced miss rate can be directly attributed to the prefetching effect of the long cache line size, combined with the usually high degree of locality found in instruction streams. Conventional processor designs cannot utilize this approach because fetching a 512B line will take a lot of time and cause memory-contention.

The instruction stream possesses a high degree of spatial and temporal locality and so the prefetching can be quite beneficial. In case of data cache, the data references have much less locality. Additionally long cache lines can increase the number of conflict misses. This effect can be clearly seen in the analysis of data cache (Figure 8 of the paper). A large number of applications show a significant increase in miss rates. This effect can be considerably reduced by adding a victim cache.

3.2 Multiprocessor Performance

To measure the performance of the integrated design, an execution driven simulation, with a timing accurate architecture model was used to evaluate simple system. The performance figures were compared with results obtained from reference CC-NUMA design with 16KB direct-mapped L1 caches, and infinite sized L2 caches. To simulate the proposed integrated system, the Inter-Node Cache (INC) size was set at 1MB per memory/processor node, which is larger than the working set of the applications used and so comparable to infinite L2 caches. The INCs were configured as 7-way set associative. The simulation benchmarks were taken from SPLASH suite.

The integrated design out performs the traditional CC-NUMA designs because of the prefetching effect of the long cache lines. But the long line prefetching effect does not help for data accesses held on another node because in this case the data is transferred as 32B blocks. If there is a case where true sharing misses dominate, adding victim cache removes a lot of misses.

This proposed architecture can be used in applications requiring embedded control, for example is graphics controller for laptops and handheld computers (similar to the graphics controller chip by NeoMagic).

This paper has proposed an idea which is quite different from the conventional microprocessor designs. Now when chips containing a billion transistor are not far away,

complex superscalar processors can utilize this idea of processor-memory integration to achieve higher performance.

4 Memory Access Scheduling [3]

This paper proposes a new technique to improve performance of memory system by reordering memory references to exploit locality within the 3-D memory structure. This paper will be published in June 2000.

The major tradeoff in case of memory access is between memory bandwidth and latency of memory access. The proposed technique reduces memory access latency at the cost of increasing load on memory bandwidth. This is in contrast with the generally proposed techniques which reduce bandwidth load.

The paper presents a different memory access scheduler architecture. It also presents a number of scheduling policies including in-order, priority, open, closed, most-pending and fewest pending.

The evaluation of the proposed technique is done considering 0 cycle processing time of the CPU. As the modern microprocessors are getting faster and faster, this is an important assumption in order to visualize the impact of future high speed processors on the proposed technique.

It was also discussed that the memory accesses are usually done in the form of bursts and not very often because the cache lines brought in can possibly service some of the future memory references.

Also mentioned was the fact that the current DRAM design, though know as Dynamic Random Access Memory, isn't completely random access because the sequential access to different rows in the same bank has high latency while access to different banks or different words within a single row has low latency. This 3-D nature of memory devices (row, column and bank arrangement) can be utilized to reorder memory accesses exploiting the non-uniform access times. This results in the DRAM operations to possibly complete out of order.

The proposed technique is evaluated by comparing against a memory controller which performs no access reordering. According to the results Unit Load (benchmark) performs very well with no access scheduling with 97% of the peak bandwidth of the DRAM. Forcing intermixed load and store to complete in order causes the sustained bandwidth to drop considerably. Each time the reference switches between load and store, a cycle of high impedence must be left on the data pins which causes further decrease in sustainable bandwidth.

4.1 First Ready Scheduling

The use of simple first ready access scheduler improved performance by an average of over 25% on all of the benchmarks. It uses the ordered priority scheme to make all scheduling decisions. It considers all pending references and schedules the DARM operation for the

oldest pending reference the does not violate the timing and resource limitations of the DRAM. The benefit of this algorithm is that accesses targeting other banks can be made while waiting for a precharge or activate operation to complete for the oldest pending reference. The sustained bandwidth increases by 79% for the microbenchmarks, 17% for the applications and 40% for application traces, over in-order scheme.

4.2 Aggressive Reordering

There are 4 algorithm which were evaluated for aggressive reordering: col/open, col/closed, row/open, row/closed. These algorithms improved memory bandwidth of the microbenchmarks by 106-144%, the applications by 27-30% and the application traces by 85-93% over in-order scheduling.

Additionally the effect of varying the bank buffer size on sustained memory bandwidth was also studied. The row/closed scheduling algorithm was used with bank buffers varying in size from 4 to 64 entries. A 16 entry buffer allowed all of the applications to achieve their peak memory bandwidth, which is 7% higher than with a 4 entry buffer. For application memory traces, there is a slight advantage to further increasing the buffer size beyond 16, a 16 entry buffer improved bandwidth by 27% and a 64 entry buffer improved bandwidth by 30% over a 4 entry buffer.

Memory Access Scheduling is an important step towards maximizing the utilization of the increasingly scarce memory bandwidth resources. Media applications have huge bandwidth utilization requirement and so this technique can prove to be very useful.

References

[1] N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers", in the *Proceedings of the 17th International Symposium on Computer Architecture*, June 1990.

[2] A. Saulsbury, F. Pong, A. Nowatzyk, "Missing the Memory Wall: The Case for Processor/Memory Integration", in the *Proceedings of the 23rd International Symposium on Computer Architecture*, May 1996.

[3] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens, "Memory Access Scheduling", in the *Proceedings of the 27th International Symposium on Computer Architecture*, June 2000.

[4] D. Thiebaut and H. S. Stone, "Footprints in Cache", in the *Proceedings of the Joint Conference on Computer Performance Modelling, Measurement and Evaluation*, 1986.