The Case for SRAM Main Memory

Philip Machanick
Department of Computer Science
University of the Witwatersrand
2050 Wits
South Africa
phone 27(11)716-3309
philip@cs.wits.ac.za
http://www.cs.wits.ac.za/~philip/dram-page.html

abstract

The growing CPU-memory gap is resulting in increasingly large cache sizes. As cache sizes increase, associativity becomes less of a win. At the same time, since costs of going to DRAM increase, it becomes more valuable to be able to pin critical data in the cache—a problem if a cache is direct-mapped or has a low degree of associativity. Something else which is a problem for caches of low associativity is reducing misses by using a better replacement policy. This paper proposes that L2 cache sizes are now starting to reach the point where it makes more sense to manage them as the main memory of the computer, and relegate the traditional DRAM main memory to the role of a paging device. The paper details advantages of an SRAM main memory, as well as problems that need to be solved, in managing an extra level of virtual to physical translation.

1. Introduction

The gap between CPU speed and DRAM speed is growing steadily. Since the mid-1980s, performance of CPUs had tended to double every 12 to 18 months, while DRAM performance has only improved by about 7% per year [Hennessy and Patterson 1995]. Consequently, it takes 6.2 years for the cost of memory access to double in terms of clock cycles [Boland and Dollas 1994]. Cache sizes—especially second-level (L2) cache sizes are increasing in an attempt at reducing misses to DRAM. Recent designs have L2 caches of over 1Mbyte.

Increases in cache sizes and miss costs may invalidate some of the underlying assumptions in cache design. For example, a cache is designed on the assumption that replacement policy is not as important as in paged memory management. Furthermore, increases in miss costs impact other aspects of memory management, such as page translation. Even on some older architectures, TLB (a translation lookaside buffer caches recent page translations) misses may account for a relatively high fraction of execution time [Nagle *et al.* 1993; Cheriton *et al.* 1993].

In this paper, I propose treating the level of the memory hierarchy that has traditionally played the role of the L2 cache as main memory, while treating the DRAM that previously was the main memory as a first-level paging device. To avoid confusion with the L*n* notation used for an *n*-level cache, the DRAM paging device is called the P1 paging device, while the disk becomes the P2 paging device.

The benefits which result from this change in memory hierarchy are as follows:

- parts of the operating system including data structures such as parts of the page tables can be pinned in the SRAM main memory, which is not possible with a direct-mapped cache and will cause extra misses with a cache of low associativity
- the best-case hit on the SRAM main memory is less complex than for an L2 cache: if the page translation is in the TLB, the memory reference continues without further complication, whereas a cache hit would still require a tag comparison and extraction of the appropriate data within a block
- more efficient replacement strategies, borrowing from previous work on paging, can be used, potentially reducing misses

- there is the option of a context switch if the miss cannot be handled fast enough to justify stalling the processor

Potential problems which need to be addressed include:

- more complexity in page tables
- references to DRAM may require a more sophisticated asynchronous controller if context switches on faults to DRAM are allowed
- the need to rewrite parts of the operating system to support the new strategy

The remainder of this paper elaborates on the rationale for the proposed change, and provides more detail of both benefits and problems. It also proposes some solutions, which are the basis for future research. The paper concludes with a summary of the problem and how I propose it be dealt with.

2. Background and Rationale

In this section, I provide some background in the form of terminology and basics of memory systems, and examine the impact of performance trends in more detail.

First, some standard terminology [Smith 1982].

A cache is organized into blocks, or lines, each of which is the minimum unit of allocation in the cache. Each block has tags associated with it, indicating which actual block (usually in terms of its physical address, but possibly its virtual address) is in the cache, and the state of the block (e.g., dirty if it's been written and differs from a lower level of the hierarchy). If a block which is referenced (or more accurately, contains data or code which is referenced, since a block is usually bigger than a single machine word) is not in the cache, a cache miss results. Usually, a cache miss will stall the CPU, until the miss is handled. When a block is brought into the cache, another block is replaced (except in the event that the place in the cache to be used is not yet occupied).

A computer system may have several levels of cache (most recent designs have at least two). The level closest to the CPU is the first level or L1 cache, with numbers increasing as the cache is closer to the main memory (L2, L3, etc.). Another important aspect of the organization of caches is that most caches use the physical rather than virtual address in their tags, so address translation has to be done as fast as possible. A fast, small cache of recent page translations, called a translation lookaside buffer (TLB), is used to allow virtual address translation in parallel with cache lookup. (For example, in the MIPS R4000, the virtual address is used to look up a block in the cache, but the physical address is used in the tag comparison [Kane and Heinrich 1992]).

A major organizational distinction between caches is their degree of associativity. A fully associative cache allows a particular block to be placed anywhere. The advantage is in potentially reducing misses caused by implementing a more effective replacement policy; the cost is in complexity of looking up a specific address to see if the block containing that address is in the cache. A direct-mapped cache always places a specific block in the same position in the cache, and is much easier to implement than a fully associative cache. A compromise is an *n*-way set associative cache, where a given block can be placed in any of *n* locations. In general, smaller caches tend to have higher degrees of associativity, since more associativity tends to reduce misses, but is hard to scale up to large caches (both in terms of cost and speed).

Figure 1 shows the CPU performance trend which has been fairly consistent since the mid-1980s. A consequence of this trend is the growing dominance of microprocessor-based systems. Another consequence is the growing importance of caches in memory systems, as DRAM lags increasingly behind the speed of even relatively low-end CPUs. As an example (here taken from high-end systems), a Silicon Graphics 4D/380 shared-memory multiprocessor system from the late 1980s had a cache miss cost of 1μsec. Its successor, the Challenge range, about 5 years later, had a similar miss cost. Although DRAM had become faster over that period, other changes in the memory system including large cache blocks, support for more processors and larger L2 caches, prevented an improvement in cache miss cost. In terms of missed

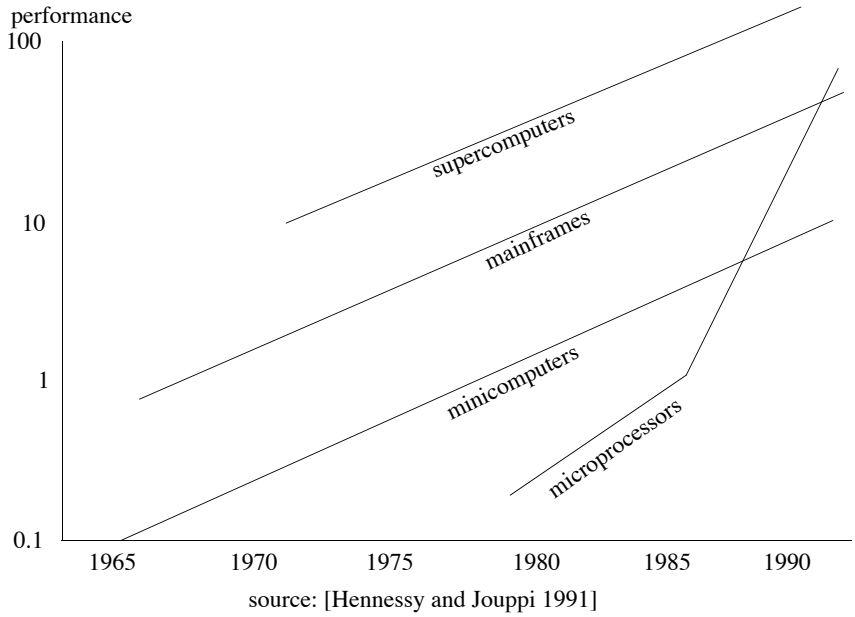

Figure 1 CPU Performance Trends

instruction executions, the 4D/380 lost approximately 30 instructions per L2 miss at its peak instruction execution rate. When the Challenge range was first introduced, the fastest processor available was a 75MHz superscalar R8000, with a peak execution rate of 4 instructions per clock cycle [Hsu 1994]—so a miss cost up to 300 instructions.

In more recent designs, a fast L2 miss cost is about 200ns; while the high end of current microprocessors dispatches 4 instructions per cycle at a clock speed of 500MHz, or one instruction every 0.5ns. Even with an aggressive memory system, therefore in current designs, an L2 miss costs up to 400 instructions.

A consequence of the growing CPU-memory speed gap is increasing sizes of L2 caches. Caches are generally made using static RAM (SRAM), which is more expensive and less dense than DRAM, but is generally faster. As cache sizes increase, on the other hand, it becomes less profitable both in terms of speed and cost to use associative caches. A large direct-mapped cache, while having more misses than an equivalent-sized 2-way set associative cache, may still result in better performance because it's faster. However, a direct-mapped cache can have unexpectedly high misses with some reference patterns, so the trend is towards 2-way associativity or additions to direct mapped caches such as Jouppi's *victim cache*, which holds recently replaced blocks, so they can be quickly retrieved [Jouppi 1990].

As the gap between L2 cache and DRAM grows and L2 cache speed increases, it starts to be increasingly attractive to pin critical data into the L2 cache.

For example, parts of the operating system, including parts of the page table and scheduler, could be kept in the L2 cache.

Keeping part of the page table in the L2 cache could reduce the average cost of a TLB miss, which can be very high [Nagle *et al.* 1993; Cheriton *et al.* 1993].

As the speed gap between L2 cache and DRAM grows, it increasingly becomes possible that a context switch on an L2 miss would be viable. If such context switches are to make sense, the cache would also have to be large enough to hold the working sets of more than one process. It would also make sense to pin enough of the scheduler into the L2 cache that there wouldn't be a miss on attempting to perform the context switch.

But a direct-mapped cache makes it problematic to pin anything into the cache. In principle, page translation could be modified to prevent any virtual addresses from translating to physical addresses that mapped onto the pinned blocks. Such an approach has been proposed to reduce conflict misses [Bershad *et al.* 1992]. But this strategy

would leave holes in physical memory—other parts of physical memory that would map onto the pinned blocks would be left unused.

With 2-way associativity, pinning is more feasible, but the effect is that half of a set is fixed while the other half becomes in effect direct mapped—with the potential for an increase in misses.

While there are work arounds for the problems arising from pinning specific blocks in a cache of low associativity (such as victim caches), it would be useful if a simpler overall strategy could be found.

Given that miss costs from L2 caches are increasing, it also is unfortunate that a less sophisticated replacement strategy is becoming the norm. While software managed caches have previously been proposed as a way of reducing misses [Cheriton *et al.* 1986], low associativity would limit the potential for improvement resulting from using software miss handling.

In summary, the problems of adjusting to larger L2 caches are:

- trend towards less associative caches contradicts the increasing need to pin critical resources in the L2 cache (e.g. to support context switches on L2 misses and to reduce TLB miss costs)
- miss costs from L2 caches are increasing, yet the trend to lower associativity precludes more effective replacement strategies

However, another option is open: moving the label "main memory" from the DRAM part of the hierarchy to the lowest level of SRAM. DRAM would then become a paging device, with disk a second-level paging device. Pinning critical resources into the SRAM main memory would be no problem, and context switches on misses would be a natural consequence of treating DRAM as a paging device. This change is contingent on two major factors: that the SRAM main memory is big enough to hold more than one working set (at least to the extent to support context switching on page faults to DRAM) and the miss cost to DRAM is high enough to justify a more sophisticated, software-based (hence slower) replacement strategy.

The following section contains more detail of the benefits of the proposed change; figure 2 summarizes the differences between a conventional and the new organization.

3. Benefits

The most obvious improvement is in simplification of the hardware. Whereas an L2 cache needs a cache controller including tags, indexing, comparison logic for the address and logic to interpret the tag bits (dirty, present, etc.), an SRAM main memory would only require a straightforward virtual addressing mechanism.

While there is some potential for parallelism in the cache organization, including starting the L2 indexing operation before an L1 miss is detected. In a virtually indexed cache such as the recent MIPS designs, the indexing can be done at the same time as TLB lookup. With an SRAM main memory, the addressing operation would have to wait for the TLB lookup. But in general it should be easier to make an SRAM main memory fast than is the case for a cache. The tag indexing and comparison operations would not be needed. Once a physical address was available from the TLB, it would immediately be known that the memory reference was a hit in the main memory. In other words, assuming the usual memory hierarchy property of hits being the common case (both in the TLB and the main memory), the common case is easier to make fast.

Given the much simpler organization of the interface between the L1 cache (including TLB) and the SRAM main memory compared with an L2 cache, more effort can in principle be put into making the data path between the L1 cache and the SRAM main memory as fast as possible. For example, the bus could be relatively wide, the SRAM could be interleaved, and the SRAM could be integrated more closely into the timing of the CPU pipeline.

By contrast, in some recent designs, effort has been put into making L2 hits faster, by moving more of the logic (in some cases, the tags included) onto the CPU chip. In such designs, an L2 hit is access in parallel with an L1 access. If the L1 access misses, the L2 access can happen without further delay to start the L2 indexing operation. The

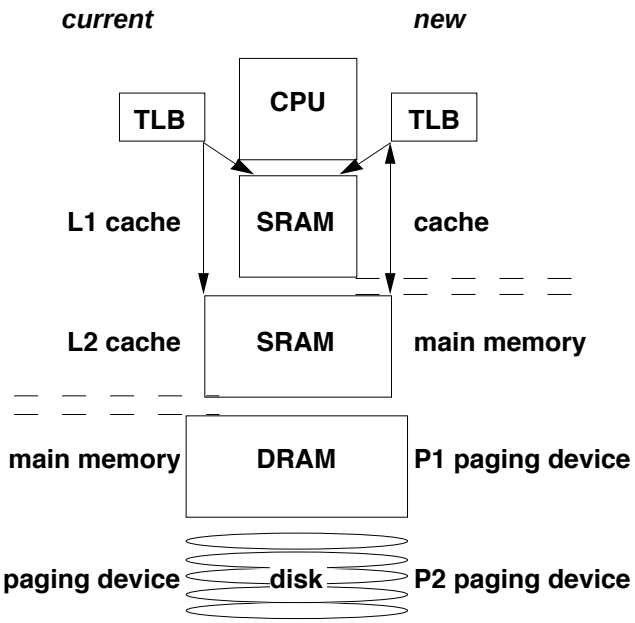

Figure 2 Old *vs.* New Memory Hierarchy
the dashed line is the dividing line between cache and paged memory; the bi-directional arrow between the TLB and the SRAM main memory in the new organization symbolizes the tighter relationship between the TLB and the RAM main memory than between a TLB and an L2 cache

proposed SRAM main memory organization can be seen as an alternative to putting the L2 tags and logic on the chip: the freed chip space could be used to make something else faster, or to increase the size of the TLB. Alternatively, as the SRAM-DRAM speed gap increases, the memory hierarchy could consist of more than one level of cache between the CPU and the SRAM main memory, in which case the on-chip L2 cache logic would still be of value.

Another potential win is in pinning the page table needed to cover the physical address space in the main memory. As discussed in the previous section, pinning in a large (typically direct-mapped) cache is problematic. If the page table were organized as an inverted page table, sufficient entries could be pinned in the main memory to cover the entire physical SRAM address space. If the code needed to reload the TLB were also pinned in the main memory, TLB misses in the case where a page fault does not occur could be kept relatively cheap. Clearly, efficient implementation of the page table is critical to performance of TLB misses in the case where there is no page fault.

4. Problems

The biggest single problem with the proposed new memory architecture is efficient management of page tables. There is an extra memory of the paging hierarchy, and the SRAM main memory would be relatively small in relation to the overall address space. An efficient representation of all levels of paging would be needed to minimize the cost of misses between the faster (higher) levels of the hierarchy.

Another potential problem is that a page size which is optimal for DRAM-disk interaction may be too big for DRAM-SRAM interaction. As a first cut at choosing the best size, the transfer unit at any level of a memory system (cache block or page) should be as small as possible to reduce the cost of handling a miss, while being big enough to amortize the cost of the latency of the slower device which is servicing the miss. There are other issues to take into account as well, such as locality issues (spatial locality implies larger units are better, while temporal locality is better served by smaller units

especially in a small memory, where bringing in a large page or block displaces a large amount of memory contents which may be referenced later).

A short-term problem is the question of when the proposed change will become feasible. Although an SRAM main memory of 1Mbyte would be big compared with a mainframe of 20 or 30 years ago, programmers today are used to assuming much larger main memories. With today's speeds and costs which determine how big an L2 cache is used, the advantages of the proposed new hierarchy may not be evident.

Another issue is how DMA and other memory accesses out of the control of the CPU should be managed. Should such memory references go to DRAM, or to SRAM? A related issue is how to organize a shared-memory multiprocessor system. If the SRAM is partitioned between processors, with a single DRAM, a multiprocessor system starts to look like a distributed shared memory system, along with all the problems that implementers of such systems have to solve. On the other hand, if the SRAM is shared, the interconnect and the SRAM itself could become major bottlenecks, compared with having large L2 caches.

A final problem is that the change requires not only hardware but software changes. Page management needs to be done differently, and enough of the kernel needs to be SRAM-resident for an acceptable context switch cost. The former requirement can potentially be met with any reasonably cleanly designed operating system in which page management is an independent module; the latter requirement would be easier to meet with a small microkernel, in which the scheduler was already tightly coded, with a small memory footprint. Furthermore, since the change today would only be useful at the high end, the number of systems that would benefit from the change may be small.

5. Solutions

Some of the problems raised in the previous section are short-term. If the current trend of a speed improvement continues, I argue that it will only be a matter of time before treating DRAM as a paging device becomes inevitable—even for mass-market systems. The exact timing is the only issue.

Compared with paging to disk, a miss cost of several hundred to a thousand instructions may seem low, but early virtual memory designs (1940s and 1950s) had page fault costs roughly of this order.

The page table issue is difficult and will require careful attention. Inverted page tables, to be efficient, should still allow close to constant time access (for example, by hashing). Even so, there is another level of paging to support. Should the DRAM be treated as another level of physical memory, or be organized as a disk cache (or buffer)? If the former, there will be two translations for a virtual address to physical memory (as well as a disk location). If the latter, there would be a DRAM-based disk directory. Detailed implementation issues are not fundamental to evaluating the new hierarchy, so for purposes of initial research, either approach can be used. As a starting point, DRAM will be treated as another level of physical memory.

A compromise on page sizes could be made by using a small page size, to suit DRAM-SRAM transactions, but transferring multiple pages to and from disk.

The problem of DMA could be disposed of in the short term by having DMA write to DRAM. Transfer from DRAM to SRAM could be initiated by a process which would be suspended on a page fault when it attempted to access the DRAM. In the longer term, once SRAM main memories became large enough, DMA references could go directly to the SRAM—but at the cost of either preventing SRAM-cache traffic at the same time as DMA traffic, or dual-porting the SRAM.

6. Conclusions

The proposed change in memory hierarchy presents some interesting challenges and opportunities.

Measurement using parameters of today's memory systems and CPUs, and predicted future systems, will reveal whether the time is right for this change.

Work which I am proposing includes simulation of the new memory system (using MINT [Veenstra 1993], more detailed design of the page table strategy and consideration of whether any new hardware is needed.

On the whole, the simplifications and potential for speed improvement at first sight appear to outweigh the problems and new complications.

Whether further research shows the SRAM main memory approach to be an acceptable solution to the growing CPU-DRAM speed gap, if current trends continue, some change in strategy appears to be inevitable. While many other alternatives are possible, the SRAM main memory idea has the attraction of being relatively simple to implement and evaluate.

Acknowledgments

Pierre Salverda has helped with clarifying some of the ideas in this paper, and with proof reading. I would also like to thank the following for taking the time to listen to my ideas, or for offering me the opportunity to talk to their research group: John Mashey, Mark Smotherman, David Cheriton, Keith Diefendorff and Emil Sarpa.

References

Bershad *et al*. [1992] BN Bershad, D Lee, TH Romer and JB Chen. Avoiding Conflict Misses Dynamically in Large Direct-Mapped Caches, *Proc. 6th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, October 1992, pp 158–170.

Boland and A Dollas [1994] K Boland and A Dollas. Predicting and Precluding Problems with Memory Latency, *IEEE Micro*, vol. 14 no. 4 August 1994, pp 59–67.

Cheriton *et al*. [1986]. DR Cheriton, G Slavenburg and P Boyle. Software-Controlled Caches in the VMP Multiprocessor, *Proc. 13th Int. Symp. on Computer Architecture*, June 1986, pp 266–274.

Cheriton *et al*. [1993]. DR Cheriton, HA Goosen, H Holbrook and P Machanick. Restructuring a Parallel Simulation to Improve Cache Behavior in a Shared-Memory Multiprocessor: The Value of Distributed Synchronization, *Proc. 7th Workshop on Parallel and Distributed Simulation*, San Diego, May 1993, pp 159–162.

Hennessy and Jouppi [1991]. JL Hennessy and P Jouppi. Computer Technology and Architecture: An Evolving Interaction, *Computer*, vol. 24 no. 9, September, 1991, pp 18–29.

Hennessy and Patterson [1995]. JL Hennessy and DA Patterson. *Computer Architecture: A Quantitative Approach* (2nd edition), Morgan Kaufman, San Mateo, 1995.

Jouppi [1990]. NP Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers, *Proc. 17th Int. Symp. on Computer Architecture*, Seattle, WA, May 1990, pp 364–373.

Hsu [1994]. P Y-T Hsu. *Design of the R8000 Microprocessor*, MIPS Technologies, Inc., Mountain View, CA, 1994.

Kane and Heinrich [1992]. G Kane and J Heinrich. *MIPS RISC Architecture*, Prentice-Hall, Englewood Cliffs, NJ, 1992.

Nagle *et al*. [1993]. D Nagle, R Uhlig, T Stanley, S Sechrest, T Mudge and R Brown. Design Trade-Offs for Software Managed TLBs, *Proc. Int. Symp. on Computer Architecture*, May 1993, pp 27–38.

Smith [1982]. AJ Smith. Cache Memories, *ACM Computing Surveys*, vol. 14 no. 3 September 1982, pp 473–530.

Veenstra [1993]. JE Veenstra. *Mint Tutorial and User Manual*, Technical Report 452, Computer Science Department, University of Rochester, June 1993.