

Hardware-Software Trade-Offs in a Direct Rambus Implementation of the RAMpage Memory Hierarchy

Philip Machanick* Pierre Salverda Lance Pompe

Department of Computer Science

University of the Witwatersrand

Private Bag 3

2050 Wits

South Africa

{philip,psalverd,lpompe}@cs.wits.ac.za

Abstract

The RAMpage memory hierarchy is an alternative to the traditional division between cache and main memory: main memory is moved up a level and DRAM is used as a paging device. The idea behind RAMpage is to reduce hardware complexity, if at the cost of software complexity, with a view to allowing more flexible memory system design. This paper investigates some issues in choosing between RAMpage and a conventional cache architecture, with a view to illustrating trade-offs which can be made in choosing whether to place complexity in the memory system in hardware or in software. Performance results in this paper are based on a simple Rambus implementation of DRAM, with performance characteristics of Direct Rambus, which should be available in 1999. This paper explores the conditions under which it becomes feasible to perform a context switch on a miss in the RAMpage model, and the conditions under which RAMpage is a win over a conventional cache architecture: as the CPU-DRAM speed gap grows, RAMpage becomes more viable.

1 Introduction

The RAMpage memory hierarchy [Mac96, MS98] moves the main memory up a level to what is traditionally the lowest-level cache, and uses DRAM as a paging device. RAMpage is an attempt at exploiting the growing latency gap between faster levels of the system (SRAM and CPU) and DRAM. While this gap is a problem many are working to reduce, it also presents an opportunity to do more interesting work during a miss to DRAM.

Characteristics of DRAM are very different to those of disk: latency is orders of magnitude lower, and the cost in lost bandwidth of initiating a new, non-contiguous access is also considerably lower. Accordingly, the opportunities for interesting activities on a miss are not as great as with a traditional page fault to disk, but kinds of activities proposed in the 1980s for software-managed caches [CSB86, CGBG88] are now more feasible, and there is the potential to explore even more interesting possibilities, such as context switches on misses.

* At time of writing, the first author was on sabbatical at Advanced Computer Architecture Laboratory (ACAL), Electrical Engineering and Computer Science Department, University of Michigan.

Despite the big differences between DRAM and disk, it is interesting to observe that the latencies for page faults in the first commercial virtual memory system, the Atlas [KELS62], were of the order of a few hundred to over 1,000 instructions, which is roughly the cost of a miss, if a page-sized unit is used in the interface between DRAM and SRAM today.

This paper presents some data on performance of the RAMpage hierarchy, in which DRAM is modeled as a simplified version of the proposed Direct Rambus design [Cri97]. In order to illustrate the possibilities for doing interesting things on a miss, the paper also presents data on cases where taking a context switch on a miss is feasible.

The focus here is on demonstrating the way in which gains resulting from a more sophisticated software strategy can be traded against hardware complexity. The software-based approach obviously has to be slower than the hardware-based approach where like activities are compared, but the software strategy can gain by reducing the frequency of worst-case events.

To illustrate how the RAMpage model competes with a conventional design, it is compared first with a baseline design with a direct-mapped second-level (L2) cache, and then with a 2-way associative L2 cache. The goal is to compare first with like hardware, then show that RAMpage can be seen as an alternative to a more complex hardware strategy.

Through managing the lowest level of SRAM as a paged memory, RAMpage is able to achieve full associativity without a hit penalty and the resulting reduction in misses compensates for the extra time required for each miss. The trade-off of interest here is whether doing more in software (which gives the CPU and the upper levels of the memory hierarchy more work) results in any gains either in reducing hardware complexity, or in improving performance. Since the direct-mapped cache represents approximately the same hardware complexity as RAMpage, the goal versus this baseline design is to show a clear performance win. A 2-way associative cache requires more complex hardware particularly if hit time is not to be compromised, so the goal for RAMpage versus this more realistic design is to show that trading hardware complexity for software complexity (with its inherently greater flexibility) does not result in a performance loss.

A recent CPU design, the PowerPC 750, has on-chip L2 tags and logic, which take up a substantial amount of space on the CPU chip. In order to keep costs under control, the designers were forced to make compromises. The PowerPC 750 can have up to 1Mbyte of L2 cache, but a cache of this size is required to have a 128-byte line size (though coherency is maintained at the level of 32-byte blocks) [IBM98]. Moving tags off-chip could in principle allow the on-chip 32 Kbyte L1 data or instruction cache to be doubled

in size¹. Another useful benefit of removing the on-chip L2 tags is that L2 parameters need not be fixed in the design at an early stage. Of course, on-chip L2 tags were introduced for performance reasons, and it is therefore important that a strategy for moving the tags off-chip should not result in a performance penalty.

In addition, there is demand for chip real-estate for additional functionality such as additional functional units and branch prediction tables [KE97]. If an alternative memory management strategy can achieve at least the same performance as a conventional 2-level cache implementation without requiring on-chip tags, other performance enhancements could use this space. Alternatively, the same performance could be achieved at lower cost since less on-chip hardware is needed.

The Pentium II uses a different strategy: it has a specialized package containing the CPU, tags and L2 cache [Int98], which has other drawbacks, most specifically, that the cache size is fixed at time of manufacture. Both the Pentium II and PowerPC 750 illustrate the problems in providing flexibility in the design of the L2 cache, while ensuring a fast hit time.

An additional advantage of a software implementation is that it becomes possible to vary the page size dynamically, whereas the design constraints on caches typically require that line (or block) sizes be fixed (in the PowerPC 750, for example, the line size is a function of the total size of the cache and cannot be varied under program control [IBM98]).

Although not explored in this paper, other possibilities which could arise include addressing DRAM through higher-level abstractions (for example, a driver which makes it look more like a disk). Such an approach would make it easier to retrofit RAMpage to an existing operating system without major software changes. Another advantage of such an implementation is that DRAM access could be more tightly managed, with more sophisticated models for protection, sharing and resource management.

The remainder of this paper is organized as follows. Section 2 gives an overview of the major features of the RAMpage design. Section 3 goes on to explain the problem RAMpage addresses in more detail and summarizes other work on similar problems. Section 4 describes the simulated systems, including why specific parameters were chosen. Results of simulations are presented in Section 5. In conclusion, overall findings and future work are discussed in Section 6.

2 RAMpage Overview

2.1 Introduction

Figure 1 shows the major conceptual differences between a RAMpage and conventional hierarchy; only the first two levels of memory are shown, since DRAM and paging to disk are organized the same way (except the TLB is not used to cache DRAM page translations in the RAMpage model). Note that only two SRAM levels are shown here for simplicity, but — as with a conventional cache hierarchy — more levels are possible in principle. As with a cache, an important factor in sizing any SRAM level is balancing size and speed (“small is fast”: not only are faster memories more expensive [HP96], but they also tend to be lower in density [Han98], and propagation delays become more of a problem as size increases, all of which limits the size of faster memories).

¹The PowerPC 750 has a two-way set-associative L2 cache with 4K tags per way, a total of 8K tags. To allow for the largest supported cache of 1 Mbyte, we calculate that there are 32K coherency tags — coherency is maintained on 32-byte blocks — each 3 bits (modified, exclusive, invalid) for a total of 12 Kbytes. After selecting one of 4K locations, 21 bits (out of 32) of the address identify the contents of a given byte, but the 4 bits for the offset within a 128-byte line need not be stored, so each address tag needs only to be 15 bits, for a total of 15 Kbytes, so the total needed to implement on-chip L2 tags is 27 Kbytes. In addition to this, space is needed for controller logic.

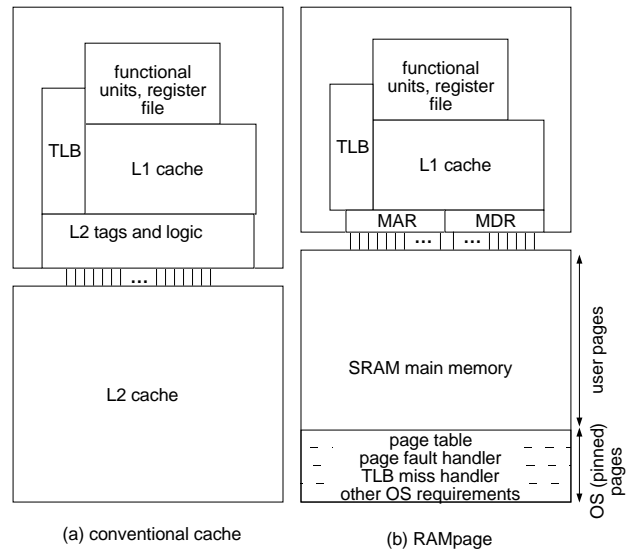


Figure 1: RAMpage versus a conventional L2 implementation. The RAMpage SRAM main memory is addressed as a conventional RAM, using a memory address register (MAR) and memory data register (MDR), and does not use tags.

This section provides an overview of the major design features of a RAMpage system: see 4.3–4.5 for more details of the simulated systems.

2.2 SRAM Main Memory

The main memory in the RAMpage hierarchy occupies the same position as the lowest-level cache in a conventional hierarchy: a relatively large (as compared with the L1 cache) off-chip SRAM memory. As with an off-chip cache, its size is constrained by the relatively high cost and low density of SRAM, so it is significantly smaller than the DRAM level. Unlike a cache, it is byte-addressed, using a conventional physical addressing mechanism. It can be physically addressed by the operating system, but access by user programs is through virtual addresses.

As with any other virtually-addressed hierarchy, user programs are not aware of the different levels — though a user program can of course be written with sensitivity to the underlying hierarchy [CGM91, CGHM93].

The basic unit of allocation in the SRAM main memory is a page. This paper leaves open the question of coherency and multi-processor implementation; strategies from distributed shared memory could apply [BCZ90].

Page translation in RAMpage is done using an inverted page table, a page table indexed on the physical instead of the virtual address [HH93] for several reasons:

- the SRAM main memory is relatively small, and an inverted page table is relatively well-suited to small physical memories (since otherwise the page table would be very large in relation to the physical memory)
- the size of the table can be fixed, making it relatively easy to pin the whole table in the SRAM main memory
- with the whole of the SRAM main memory mapped by a table which is pinned in the SRAM main memory, a TLB miss need never reference DRAM or disk, until there is a page fault from SRAM

An inverted page table is slower on lookup than a forward page table, since a hash function is used to find a virtual address. However, it is useful to be able to guarantee that a reference that is found in the SRAM main memory or L1 cache will never result in another reference, resulting from a TLB miss, going to a lower level. Programmers attempting to minimize DRAM references will find this property useful, as TLB miss behaviour can be a major complication when attempting to do memory hierarchy-sensitive application structuring [CGHM93].

2.3 Role of the TLB

The TLB (translation lookaside buffer) in a conventional hierarchy, particularly with a physically-indexed L1 cache, plays a vital role in performance [NUS⁺93], as every address in user-level programs has to be translated, including that of every instruction fetched.

In the RAMpage hierarchy, the TLB plays a slightly different role. It is still needed to translate addresses in the event that the L1 cache is physically indexed and tagged, but the address translations are to physical addresses in the SRAM main memory. The TLB does not need to translate addresses to DRAM, as these translations are not on the critical path for a hit to L1 or the SRAM main memory. A translation to a DRAM address is only needed when a page fault from the SRAM main memory occurs.

Part of the RAMpage SRAM main memory is reserved for the TLB miss handler. All TLB misses that are for references that hit in SRAM can be serviced without going to DRAM. If a page is replaced from the SRAM main memory, its entry (if it has one) in the TLB is flushed. Being able to service TLB misses without having to go to DRAM in most cases is potentially a big win, considering that some studies have shown that TLB misses can account for a large fraction of execution time [NUS⁺93, CGHM93].

As with a conventional hierarchy, it is possible in principle to address the L1 cache virtually, in which case the TLB would only be needed on a miss to the SRAM main memory. Similar problems to those found in virtually-addressed caches in a conventional hierarchy [IKWS92, WB92] would apply. This possibility is not explored in this paper.

2.4 DRAM Paging Device

The DRAM paging device is a conventional DRAM, which can be implemented using the same range of design choices as a conventional DRAM main memory. In the work reported on here, the same inverted page table strategy is used as for the SRAM main memory, for simplicity. The DRAM page table need not be implemented the same way as the RAMpage SRAM main memory. For example, in the results presented here, the DRAM page size is held constant, while the SRAM page size is varied.

In the work reported on here, the simulated DRAM is a Direct Rambus, but without exploiting the option of pipelining multiple references. This implementation has similar characteristics to an SDRAM implementation, as is explained in 3.3.

3 The Problem and Related Work

3.1 Introduction

Since the mid-1980s, CPU speeds have been improving at 50 to 100% per year, while DRAM latency has only been improving at 7% per year [HP96], resulting in a doubling of the time in instructions to access DRAM every 6.2 years [BD94].

Caches have traditionally been used to bridge the CPU-DRAM speed gap. However, the growing gap presents increasing challenges for cache designers. This section summarizes some improvements to the basic design of caches, as well as attempts at

hiding the basic latency of DRAM by more sophisticated organization. To place the RAMpage strategy in context, other software-based strategies are reviewed. To conclude the section, RAMpage is described.

3.2 Improvements to Caches

A cache presents designers with a number of trade-offs between reducing misses to the next level down, and both cost and speed of hits. A simple direct-mapped cache is easy and relatively cheap to design for fast hits. Adding associativity makes it more difficult to achieve fast hits, while reducing the number of misses. In general, as the penalty for a miss increases, adding complexity (and hence cost) to the hardware to achieve fast hits with more associativity becomes more worthwhile.

Another improvement which can reduce misses without adding to the complexity of achieving fast hits is a *victim cache*, a small additional cache which contains recently replaced blocks [Jou90].

Others have investigated alternatives to full associativity, including column-associative caches [AP93], which allow an alternative location for a block without the overhead of associativity in the best case for a hit. Another strategy is to use a direct-mapped cache, but to try to reduce conflict misses by choice of placement of virtual pages in physical memory [KH92b, BLRC94].

While a reduction in the number of misses appears to offer the best potential for speed improvement especially as the CPU-DRAM speed gap grows, hiding latency is also a common strategy. Non-blocking prefetch instructions can reduce the probability of a future miss [Kro81, CB92, MLG92, KK97, Che95], and non-blocking misses [BK96, CB92] in a superscalar CPU allow other instructions to complete while waiting for a miss. Speculative loads are another variant on prefetch: they do not result in an interrupt if there is a page fault or other problem (e.g. a null pointer), unless the result of the load is actually used [RL92, Dul98].

Some of these approaches can be applied to the RAMpage architecture, others compete. The victim cache concept can be implemented as an extension of the page replacement strategy, using a conventional operating system approach: when a page is replaced, it is moved to the *standby page list*; the page which is on the list longest is the one actually discarded [Cro97]). Prefetch could be added to RAMpage, as could speculative loads.

Approaches to implementing associativity cheaply can be considered to be competing alternatives. However, conventional limited associativity implemented in hardware has up to now been capable of meeting design goals for speed and cost and is therefore the standard against which RAMpage is judged in this paper. Taking a context switch on a miss could be thought of as similar to a non-blocking cache. However, a more directly comparable approach is a multithreaded architecture [WW93, HKT93, LEL⁺97], a concept which could in principle be implemented on top of RAMpage.

3.3 Improvements to DRAM

While the underlying trend in cycle time for DRAM remains unchanged, it is possible to hide some of the latency by exploiting typical reference patterns. For example, a cache miss – the most common way of referencing DRAM from the CPU – references multiple sequential bytes. Since it is easier to achieve high bandwidth than low latency, both in general and specifically for DRAM [HP96], DRAM designers have offered various modes which allow multiple sequential accesses to be faster on average than a single random access. These include fast page mode and EDO (extended data out). More recent approaches include synchronous DRAM (SDRAM) and Rambus [Cri97].

SDRAM clocks DRAM to the bus and after an initial delay (for example 50ns), subsequent transfers can occur at bus speed (e.g.,

10ns) [IBM97]. With a wide 128-bit bus, a 10ns SDRAM memory system can in principle deliver 1.5Gbyte/s, excluding the initial latency. Rambus aims to deliver similar latency more cheaply and in smaller increments, in terms of the minimum RAM upgrade, by using a narrow bus clocked at high speed. The original Rambus design used a byte-wide bus whereas the proposed Direct Rambus design for 1999 uses a 2-byte bus clocked at 1.25ns, giving the same 1.5Gbyte/s as a 128-bit wide 10ns SDRAM design. It is also possible to have multiple Rambus channels to increase bandwidth, though latency is not improved [Cri97].

Direct Rambus goes further than other latency-hiding DRAM designs (aside from memories in high-end supercomputers which use expensive strategies, like hundreds or thousands of banks of DRAM) [Fat90, SWL⁺92] in that it allows multiple independent references to be pipelined, allowing a theoretical 95% of peak bandwidth to be achieved on units as small as 2 bytes. Whether the reference pattern of a typical workload will support this theoretical peak is an issue still to be investigated.

The challenge to DRAM designers – as long as they are unable to break out of the current learning curve which emphasizes density over latency improvement – is to find creative strategies for hiding latency. Alternatively, the challenge for system designers is to work around the worsening CPU-DRAM speed gap.

3.4 Software-Based Approaches

Designers of both DRAM and systems have mostly focused efforts on hardware-based approaches for the obvious reason of supporting backward compatibility. While changes in cache architecture may require some operating system changes, those changes can usually be minimized since the cache is managed in hardware. The same is true of most variants on DRAM. As a result, while Rambus, for example, presents relatively major changes at the level of the bus and memory controller, it can be designed into a conventional system without requiring operating system changes.

Software-based approaches on the other hand may require operating system modification, which is clearly a harder sell.

In the 1980s, there was some work on software-based management of caches, with emphasis on reduction of misses in a shared-memory system [CSB86, CGBG88]. More recently, work on managing the interface between cache and DRAM in software has focused on address translation [JM97].

In neither case has the major focus been on achieving a higher degree of associativity than is common in caches; the space created by high miss costs has been exploited for other reasons. Also, the possibility of increasing block size to allow room in a miss for doing other useful work has not previously been explored. Finally, other related work has not gone as far as treating the lowest level of cache as a fully software-managed paged memory (in effect, an SRAM main memory).

3.5 The RAMpage Strategy in Context

The RAMpage approach recognizes that software management of misses imposes a penalty, and attempts to go as far as possible in eliminating misses to justify the extra penalty. While there are major differences between DRAM and disk, some principles applicable to handling page faults start to become applicable when miss costs increase to hundreds of missed instructions. While the comparison with disk should not be taken too far, given the clear differences, the following issues are similar:

- a disk's peak transfer rate is significantly higher than the rate for a small transfer because of its high latency; DRAM shares this property if with significantly different numbers

- the amount of time lost by a miss is large enough to create space for doing useful work if other processes are available

To quantify these points, Table 1 contains efficiency measures for a disk and for two variants on Direct Rambus, to show that, while the comparison cannot be taken too far, RAM does share disk's property of being more efficient at transferring large units. To give a specific example, with a 1GHz issue rate, a 4Kbyte disk transfer costs about 10-million instructions, whereas a 4Kbyte Direct Rambus transfer costs about 2,600 instructions.

Bytes Transferred	(% efficiency)		
	1 Channel 16 bits	4 Channels 64 bits	disk
16	16.7	4.8	0.0038
32	28.6	9.1	0.0076
64	44.4	16.7	0.015
128	61.5	28.6	0.031
256	76.2	44.4	0.061
512	86.9	61.5	0.12
1024	92.8	76.2	0.24
2048	96.2	86.5	0.49
4096	98.1	92.8	0.97

Table 1: Efficiency (% bandwidth utilized) of 2-byte-wide Direct Rambus, compared with disk with 10ms latency and 40MB/s transfer rate, assuming no pipelining of Rambus references. Measured as the percentage of available bandwidth actually used for each case of number of bytes transferred.

In summary, the characteristics of DRAM, although very different from disk, offer enough similarities to make it reasonable to investigate the option of managing DRAM as a paging device. The differences suggest that different compromises may be called for, but trends in DRAM performance suggest that the opportunities for a change towards treating DRAM as a paging device are improving.

4 Simulated Systems

4.1 Introduction

This section describes and justifies the parameters of the simulated systems.

The approach used is to measure RAMpage against a baseline system with a simple direct-mapped cache, to compare it against a system of similar hardware complexity. The only major hardware difference (in terms of added components) is that the baseline system has tags and associated logic.

Once a baseline comparison is established, a more realistic set of measurements is presented, in which context switches are more accurately modeled, and a 2-way associative L2 cache is simulated for comparison with RAMpage. In this way, it can be seen how RAMpage's software-based approach for achieving full associativity compares against a more traditional hardware-based approach to achieving more limited associativity.

The remainder of this section starts by describing benchmark data used to drive the simulations, then itemizes configurations of the various simulated systems.

4.2 Benchmarks

Measurements were done with traces containing a total of 1.1-billion references, which were obtained from the Tracebase trace archive at New Mexico State University². Table 2 lists the traces.

²The traces used in this paper can be found at <ftp://tracebase.nmsu.edu/pub/.tbl/r2000/utilities/> and <ftp://tracebase.nmsu.edu/pub/.tbl/r2000/SPEC92/>.

Program	Description	Instr. fetches	Total refs
alvinn	neural net training (fp92)	59.0	72.8
awk	unix text utility	62.8	86.4
cexp	from SPECint92	28.5	37.5
compress	file compression (int92)	8.0	10.5
ear	human ear simulator (fp92)	65.0	80.4
gcc	C compiler (int92)	78.8	100.0
hydro2d	physics computation (fp92)	8.2	11.0
mdljdp2	solves motion eqns (fp92)	65.0	84.2
mdljsp2	solves motion eqns (fp92)	65.0	77.0
nasa7	NASA applications (fp92)	65.0	99.7
ora	ray tracing (fp92)	65.0	82.9
sed	unix text utility	7.7	9.8
su2cor	physics computation (fp92)	65.0	88.8
swm256	physics computation (fp92)	65.0	87.4
tex	unix text utility	50.3	66.8
uncompress	file decompression (int92)	5.7	7.5
wave5	solves particle equations	65.0	78.3
yacc	unix text utility	9.7	12.1

Table 2: Address traces used in the simulations. *Millions of instruction fetches and references (“fp92” or “int92” means from SPEC92).*

The traces were interleaved, switching to a different trace every 500,000 references, to simulate a multiprogramming workload. Although each individual trace is not very long, the overall effect of the combined traces’ 1.1-billion-references appears to be sufficient to warm up the memory hierarchy. For 128-byte SRAM pages, it takes about 50-million references before every page in the RAMpage SRAM main memory is occupied; this figure drops off with page size to about 25-million references before all pages in the 4 Kbyte pagesize simulation have been occupied at least once. Work is currently in progress on implementing a trace engine using Simplescalar [BA97], which will make it possible to use a substantial portion of Spec95 to experiment with longer traces.

4.3 Common Features

Both systems are configured as follows:

- CPU – single cycle non-superscalar, pipeline not modeled
- L1 cache – 16Kbytes each of data and instruction cache, physically tagged and indexed, direct-mapped, 32-byte block size, 1-cycle read hit time, 12-cycle penalty for misses to L2 (or SRAM main memory in the RAMpage case); for the data cache: perfect write buffering with zero (effective) hit time, writeback (12-cycle penalty; 9 cycles for RAMpage since there is no L2 tag to update), write allocate on miss
- TLB – 64 entries, fully associative, random replacement, 1-cycle hit time, misses modeled by interleaving a trace of page lookup software
- DRAM level – Direct Rambus without pipelining: 50ns before first reference started, thereafter 2 bytes every 1.25ns
- paging of DRAM – inverted page table: same organization as RAMpage main memory (see 2.2) for simplicity, but infinite DRAM modeled with no misses to disk
- TLB and L1 data hits are fully pipelined, i.e., they do not add to the execution time; where there are no misses, only instruction fetches add to simulated run time – the given hit times are however used when replacements or maintaining inclusion are simulated

Note that detail of the L1 cache is similar across all variations.

A superscalar CPU is not explicitly modeled. The CPU cycle time used is intended to approximate to the effect of a superscalar design, i.e., it is really meant to model the instruction issue rate rather than the actual CPU cycle time. Issue rates of 200MHz to 4GHz are simulated to model the growing CPU-DRAM speed gap (cache and SRAM main memory speed are scaled up but DRAM speed is not).

4.4 Baseline System Features

The baseline system has a direct-mapped 4Mbyte L2 cache, a size likely to become increasingly common. A direct-mapped cache is used in this case to keep the hardware as similar as possible to the RAMpage hardware, since a fast hit time is hard to achieve on an associative cache unless the tags and logic are on-chip.

Block (line) size is varied in experiments from 128 bytes to 4Kbytes.

The bus connecting the L2 cache to the CPU is 128 bits wide and runs at one third of the CPU clock rate (i.e., 3 times the cycle time), reflecting the fact that the modeled CPU cycle time is intended to represent a superscalar issue rate. The L2 cache is clocked at the speed of the bus to the CPU. Hits on the L2 cache take 4 cycles including the tag check and transfer to L1.

Inclusion between L1 and L2 is maintained [HP96], so L1 is always a subset of L2, except that some blocks in L1 may be dirty with respect to L2 (writebacks occur on replacement).

The TLB caches translations from virtual addresses to DRAM physical addresses.

4.5 RAMpage System Features

To compare like with like, the simulated RAMpage SRAM main memory is 128 Kbytes larger (since it does not need tags), for a total of 4.125Mbytes. The extra amount is scaled down for larger page sizes, since the number of tags in the comparable cache also scales down with block size. In our simulations, the operating system uses 6 pages of the SRAM main memory when simulating a 4 Kbyte-SRAM page, i.e., 24 Kbytes, up to 5336 pages for a 128 byte block size, a total of 667 Kbytes. These numbers cannot be compared directly with the conventional hierarchy as they not only replace the L2 tags, but also some operating system instructions or data (including page tables) which may have found their way into the L2 cache, in the conventional hierarchy.

The RAMpage SRAM main memory uses an inverted page table, as is explained in 2.2.

Replacements use a standard clock algorithm [Cro97]: a clock hand advances through the page table, marking each page that has previously been marked as “in use” as “unused”, until an “unused” page is found. This “unused” page becomes the victim and is replaced.

The TLB in the RAMpage hierarchy caches address translations of SRAM main memory addresses and not DRAM physical addresses. Otherwise, the major difference from the baseline hierarchy is that a TLB miss never results in a reference below the SRAM main memory, unless the reference itself results in a page fault from the SRAM main memory, as is explained in 2.3.

4.6 Context Switches

Measurement is done by adding a trace of simulated context switch code to the baseline system (approximately 400 references per context switch). In the RAMpage system, context switches are also taken on misses to DRAM, and the speed difference measured. In the RAMpage model, the context switching code and data structures are pinned in the RAMpage SRAM main memory, so that

switches on misses do not result in further misses by the operating system code or data.

Code traced to simulate context switches is based on a standard textbook algorithm.

4.7 More Realistic Implementation

In addition to the baseline cache, results are presented using a 2-way set associative L2 cache using a random replacement policy, which otherwise has the same parameters as the baseline system. The intention here is to show the benefits of hardware-implemented associativity.

The trade-off being examined here is between the extra hardware needed to implement 2-way associativity (on-chip logic and tags) versus the software complexity of the RAMpage model (resulting in fewer misses but a higher miss penalty).

5 Results

5.1 Introduction

The major data presented is simulated run times, though memory system measurements are also presented. Results are presented following the breakdown of the previous section. The baseline system and RAMpage are presented first, followed by data illustrating where hardware and software can be traded, in alternative strategies for arriving at a more efficient memory hierarchy. Then, measurement of context switches on misses for RAMpage is presented, and finally, a more realistic L2 cache implementation is compared with RAMpage.

In conclusion, major points of the results are summarized, in preparation for the discussion contained in the Conclusion.

5.2 RAMpage vs. Baseline

Simulated run times appear in Table 3.

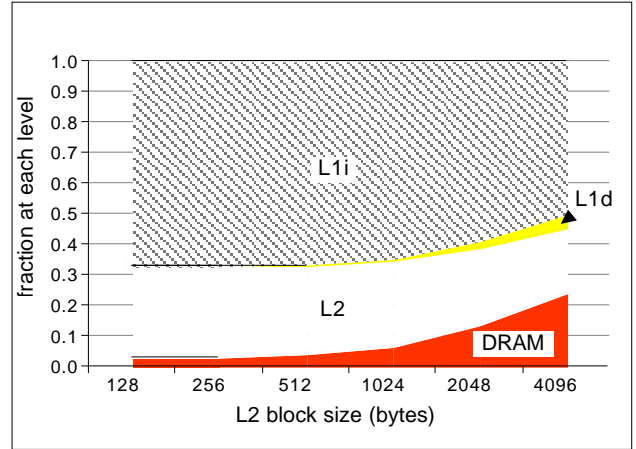
instruction issue rate	SRAM block/page size (bytes)					
	128	256	512	1024	2048	4096
200MHz	6.38	6.39	6.48	6.71	7.56	9.42
	8.48	7.11	6.41	5.99	6.03	6.09
500MHz	2.66	2.67	2.75	2.94	3.62	5.14
	3.46	2.88	2.61	2.44	2.45	2.49
1GHz	1.43	1.44	1.52	1.69	2.32	3.73
	1.80	1.49	1.35	1.26	1.27	1.29
2GHz	0.80	0.82	0.89	1.06	1.65	3.01
	0.95	0.78	0.70	0.66	0.67	0.68
4GHz	0.50	0.51	0.59	0.75	1.33	1.59
	0.53	0.44	0.39	0.37	0.37	0.39

Table 3: Elapsed simulated time (s) for 1.1 billion-reference combined traces. Each row contains cache-based hierarchy at the top, and RAMpage hierarchy below.

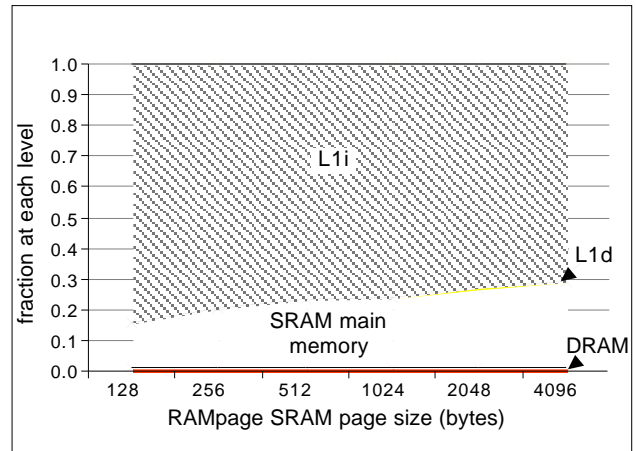
Since the baseline cache hierarchy is not realistic, there is a limit as to how much can be read into these figures. It is worth noting however that the RAMpage hierarchy performs better with larger page sizes in SRAM in part because of extra TLB and page fault handling overheads from small pages. Also note that the improvement resulting from fewer misses becomes greater as the CPU-DRAM speed gap increases, as would be expected. For a 200MHz issue rate, the best run time for RAMpage of 5.99s (page size 1Kbytes) is 6% faster than the best baseline run time of 6.38s (128-byte block). For a 4GHz issue rate, the best run times are achieved with the same page and block sizes as for 200MHz, but the best RAMpage time is 26% faster than the baseline hierarchy.

5.3 RAMpage Hardware-Software Trade-Offs

The use of software-based memory management results in a significantly lower fraction of execution time being spent in the DRAM level of the hierarchy, as can be seen in Figures 2 and 3, which illustrate the time spent in each level of the hierarchy, as a fraction of overall run time. The differences between the two figures illustrate the effect of scaling CPU speed up without improving DRAM speed: the RAMpage system is more tolerant of the increased DRAM latency.



(a) direct-mapped L2

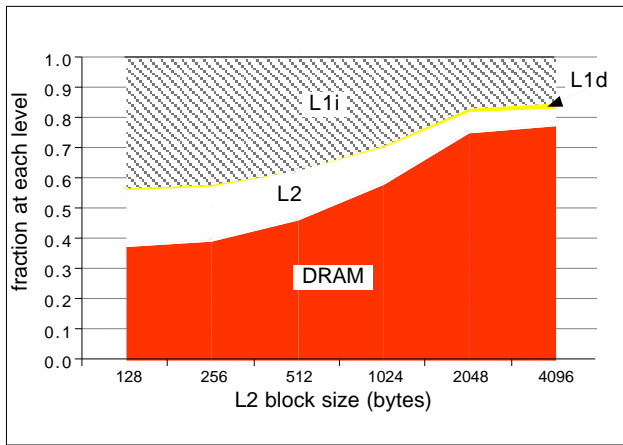


(b) RAMpage

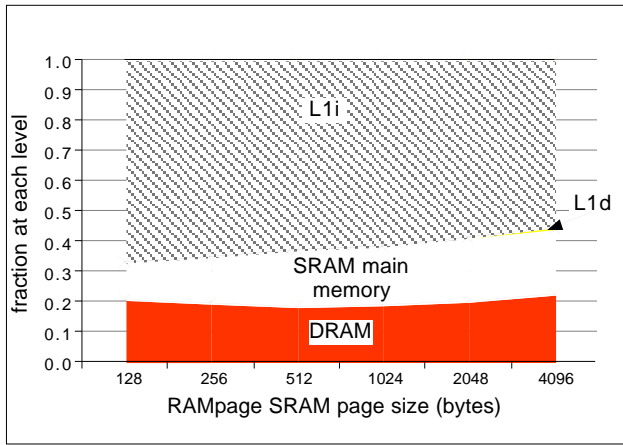
Figure 2: Fraction of simulated run time in each level of the hierarchy. 200MHz issue rate. Measured as a fraction of overall run time. Note that L1 data traffic is a very low fraction because hits are assumed to be fully pipelined; the “L1d” time accounted for is purely that taken to maintain inclusion. “L1i” time includes hits (instruction fetches) and time to maintain inclusion.

Figure 4 plots the overheads incurred by memory management software in each hierarchy, which is as high as 60% of the number of references in the benchmark trace files for small RAMpage SRAM page sizes, reflecting the relatively small 64-entry TLB used in these simulations.

The penalties incurred in the RAMpage model in reducing the time spent going to DRAM are in the higher number of TLB references for small page sizes, and in execution of a higher number of instructions, needed to manage misses from SRAM. Measurement



(a) direct-mapped L2



(b) RAMpage

Figure 3: Fraction of simulated run time in each level of the hierarchy. 4GHz issue rate. See Figure 2 for a detailed explanation.

of a competing solution, using more hardware, is presented after the following improvement to RAMpage.

5.4 RAMpage Context Switches on Misses

Performance of the baseline system is not significantly impacted when context switching costs are added: the major cost of context switching, extra misses caused by a working set change, is already accounted for in measurements in the previous subsection. For this reason, revised figures on the baseline system are not presented here.

Table 4 summarizes measurements of run times for the RAMpage architecture with context switches on misses. The most important result to note here, aside from a modest speed improvement (up to 16% in the 4GHz case over the best RAMpage time without context switches on misses), is that as the CPU speed increases, larger page sizes become more viable. It is also worth noting that the value of taking a context switch on a miss increases as CPU speed increases.

5.5 RAMpage with Context Switches vs. Better L2

Results presented here, in Table 5, are for a 2-way associative L2 with the same parameters otherwise as the baseline system. The

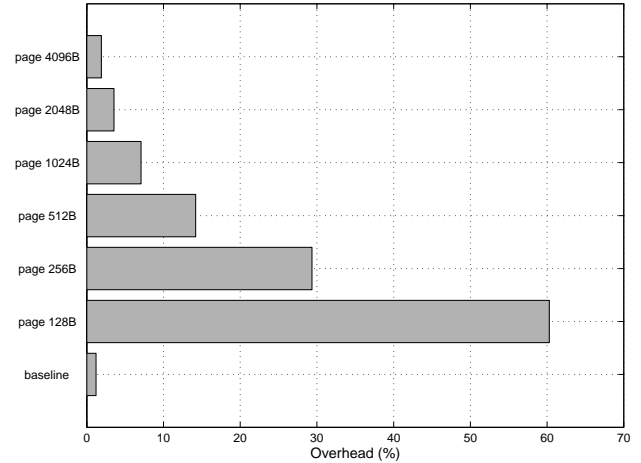


Figure 4: TLB miss and page fault handling overheads. Overhead is the ratio of additional TLB miss and page fault handling references to the total number of references in the benchmark trace files. The baseline hierarchy data is the same across all block sizes.

instruction issue rate	SRAM page size (bytes)						vs. no switch
	128	256	512	1024	2048	4096	
200MHz	9.92	7.67	6.72	6.16	6.05	6.11	0.99
500MHz	3.96	3.08	2.68	2.56	2.52	2.43	1.00
1GHz	1.93	1.54	1.39	1.26	1.23	1.25	1.02
2GHz	0.99	0.75	0.67	0.63	0.62	0.62	1.06
4GHz	0.47	0.38	0.33	0.33	0.31	0.31	1.16

Table 4: Run times (s) for RAMpage with context switches on misses. The “vs. no switch” numbers are speedup over RAMpage without context switches.

measurements here do include a trace of the execution of context switching code, though as indicated before, the difference made by adding a trace of context switching code and data is insignificant (under 1%).

instruction issue rate	L2 block size (bytes)					
	128	256	512	1024	2048	4096
200MHz	6.27	6.20	6.20	6.17	6.16	6.17
500MHz	2.57	2.52	2.52	2.49	2.48	2.47
1GHz	1.35	1.29	1.29	1.27	1.25	1.25
2GHz	0.73	0.67	0.67	0.65	0.63	0.62
4GHz	0.43	0.37	0.37	0.35	0.33	0.32

Table 5: Run times (s) for a 2-way associative L2 cache with context switches. A context switch trace is inserted between switches from one benchmark to another; context switches are not taken on misses.

Of interest is the closeness of the RAMpage and 2-way associative times. Figure 5 shows the differences between the two as a relative speed measure. Note the extent to which larger block sizes become favourable for the 2-way associative hierarchy as the CPU-DRAM speed gap grows. It is possible that this is an artifact of the context switch interval used in simulations; in a real system it would be based on a real-time clock rather than a fixed number of references and would therefore correspond to a higher number of references as the CPU was sped up. A short time slice favours larger blocks because larger blocks support spatial locality at the expense of temporal locality. If the time slice is made short enough,

temporal locality becomes less important than spatial locality.

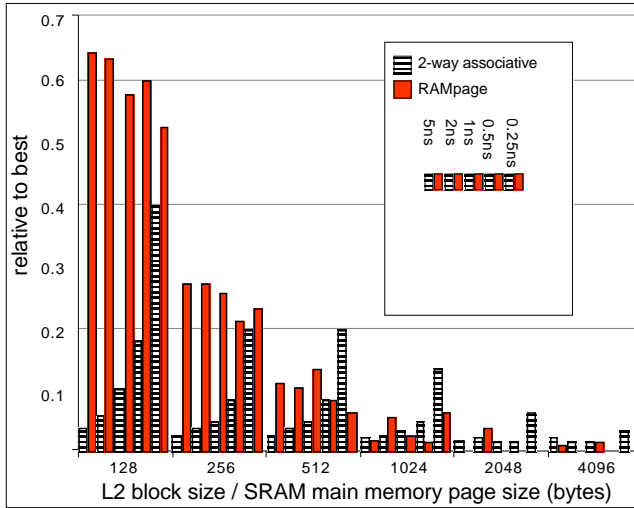


Figure 5: RAMpage (context switches on misses) speed vs. 2-way associative L2 cache for a range of CPU speeds. *The relative measure is n , where n means $1/n$ times slower than the best time for each CPU speed.*

5.6 Summary of Results

The baseline system is clearly slower than RAMpage except in the cases of smaller page sizes, where TLB overheads outweigh RAMpage’s advantages. RAMpage’s best-case performance is significantly better than the baseline, especially as CPU speed improves. Adding in context switches on misses increases RAMpage’s advantage over the baseline system, and makes larger page sizes attractive (at least as measured here). As the CPU-DRAM speed gap grows, the results show that context switches on misses become a bigger win, as is to be expected.

The more realistic L2 implementation narrows the gap, and has very similar performance characteristics to RAMpage for larger block sizes. For smaller block sizes, RAMpage again does not perform well, because of TLB overhead.

Overall, RAMpage is competitive with a simple 2-way associative L2 implementation as measured here. The reduction in misses in RAMpage is sufficient to compensate for the extra software overhead (as measured in Figure 4).

6 Conclusions

6.1 Introduction

This section discusses the significance of the results, as well as future work. The focus in future work is in filling gaps in the data presented here, as well as in further exploring the design space. To conclude, the paper ends with a final summary.

6.2 Discussion of Results

The data presents here makes a case for RAMpage as an alternative to hardware strategies for achieving associativity. Performance is shown to be essentially the same as for a 2-way associative L2 cache implemented in hardware. Context switches on misses are shown to be viable at least for page sizes larger than traditional cache block sizes, particularly as the CPU-DRAM speed gap grows. The possibility that such relatively large transfer units

could be optimal even for a conventional cache also comes out of the data presented here, though this result could be an artifact of the context switch interval used.

If it can be shown in future work that the optimal block size depends on factors like the context switch interval, RAMpage offers another potential win: the ability to change block size dynamically. The only hardware support needed for this is a TLB capable of managing variable page sizes (already an option on some architectures such as MIPS [KH92a]).

6.3 Future Work

There are several issues coming out of the results reported on here which need to be investigated further:

- interaction with operating system – issues like the impact of the time slice on optimal block or SRAM page size should be investigated further
- multithreading – it would be interesting to combine RAMpage with a hardware or software implementation of threads: a cheaper mechanism for context switching than that measured here would make better use of the relatively small miss cost of a page fault to DRAM
- more sophisticated CPU simulation – it would be worth simulating a superscalar CPU to investigate the impact of superscalar execution
- more sophisticated Direct Rambus simulation – the effect of pipelined memory references would be worth investigating, particularly to see if smaller block or page sizes become viable in this case
- variation of TLB size – a larger TLB would likely make RAMpage more competitive, with smaller SRAM page sizes
- more aggressive L1 cache – a more realistic L1 cache would make differences between L2 or SRAM main memory implementations clearer, as a higher fraction of execution time would result from misses to DRAM, if a lower fraction of references miss from L1
- coherence – for both interactions with I/O and for multiprocessor implementations

A start has been made on doing measurement with a much larger TLB (1K entries, 2-way associative) and more aggressive L1 caches (64 Kbytes each of instruction and data cache, 8-way associative). These figures are representative of designs anticipated in 1998 from Digital and IBM, for example. Results are incomplete, but indications are that with this improved hierarchy, RAMpage does become competitive under a wider range of conditions (for example, faster than a 2-way associative L2 cache with a 128-byte SRAM page).

Other work in progress includes more detailed evaluation of differences in individual application behaviour, to explore the value of a variable SRAM page size; initial results show that variation can make a difference in individual programs but that a single page size may be optimal for most programs under given assumptions about the memory system.

6.4 Final Summary

The RAMpage hierarchy provides designers with an alternative in choosing between relative complexity of hardware as opposed to software. A RAMpage memory system frees up chip space which would otherwise have been needed for L2 tags, and fast hits are relatively easy to implement. While SRAM size is constrained by cost

and the relatively low density of SRAM as compared with DRAM, RAMpage allows the system designer to choose a size for SRAM based on cost constraints without the additional constraint of a fixed upper limit on the number of L2 caches tags supported on the CPU chip.

The potential for dynamic tuning of a RAMpage hierarchy – choosing the SRAM page size on the fly, choosing whether to take context switches on misses and varying the complexity of the replacement strategy – adds additional flexibility which would be hard to achieve with a purely hardware strategy.

Another useful property of the RAMpage model which helps to offset the performance loss of doing more in software is that it becomes possible to pin critical operating system code and data in the lowest SRAM level.

The results presented here make a case for taking the RAMpage concept further. Once a more complete simulation is implemented, ideally including full operating system execution (an idea shown to be feasible for example by SimOS [RHWG95]), it will become clearer whether the idea can be practically deployed. However, given the growing complexity of hardware strategies for working around the underlying latency of DRAM technology, it is worth investigating software alternatives, if such alternatives can result in simplification of the hardware.

Acknowledgements

Trevor Mudge hosted the first author for a sabbatical, which help to make this work possible. Financial support for the sabbatical was received from the University of the Witwatersrand and the South African Foundation for Research Development. We would also like to thank the anonymous referees for some suggestions for improvements to this paper. Matt Postiff and Steven Reinhardt gave useful suggestions for improvement of the final draft of the paper.

References

- [AP93] A. Agarwal and S.D. Pudar. Column associative caches: A technique for reducing the miss rate of direct mapped caches. In *Proc. 20th Int. Symp. on Computer Architecture (ISCA '93)*, pages 179–190, May 1993.
- [BA97] D. Burger and T. M. Austin. *The SimpleScalar Tool Set*. Version 2.0, Tech. Report No. 1342, Computer Sciences Department, University of Wisconsin-Madison, June 1997.
ftp://ftp.cs.wisc.edu/galileo/dburger/papers/TR_1342.ps.
- [BCZ90] J.K. Bennet, J.B. Carter, and W. Zwaenepoel. Adaptive software cache management for distributed shared memory architectures. In *Proc. 17th Int. Symp. on Computer Architecture (ISCA '90)*, pages 125–134, Seattle, WA, May 1990.
- [BD94] K. Boland and A. Dollas. Predicting and precluding problems with memory latency. *IEEE Micro*, 14(4):59–67, August 1994.
- [BK96] S. Belayneh and D.R. Kaeli. A discussion of non-blocking/lockup-free caches. *Computer Architecture News*, 24(3):18–25, June 1996.
- [BLRC94] B.N. Bershad, D. Lee, T.H. Romer, and J.B. Chen. Avoiding conflict misses dynamically in large direct-mapped caches. In *Proc. 6th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-6)*, pages 158–170, October 1994.
- [CB92] T. Chen and J. Baer. Reducing memory latency via non-blocking and prefetching caches. In *Proc. 5th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-5)*, pages 51–61, September 1992.
- [CGBG88] D.R. Cheriton, A. Gupta, P.D. Boyle, and H.A. Goosen. The VMP multiprocessor: Initial experience, refinements and performance evaluation. In *Proc. 15th Int. Symp. on Computer Architecture (ISCA '88)*, pages 410–421, Honolulu, May/June 1988.
- [CGHM93] D.R. Cheriton, H.A. Goosen, H. Holbrook, and P. Machanick. Restructuring a parallel simulation to improve cache behavior in a shared-memory multiprocessor: The value of distributed synchronization. In *Proc. 7th Workshop on Parallel and Distributed Simulation*, pages 159–162, San Diego, May 1993.
- [CGM91] D.R. Cheriton, H.A. Goosen, and P. Machanick. Restructuring a parallel simulation to improve cache behavior in a shared-memory multiprocessor: A first experience. In *Proc. Int. Symp. on Shared Memory Multiprocessing*, pages 109–118, Tokyo, April 1991.
- [Che95] T-F. Chen. An effective programmable prefetch engine for on-chip caches. In *Proc. 28th Int. Symp. on Microarchitecture (MICRO-28)*, pages 237–242, Ann Arbor, MI, 29 November – 1 December 1995.
- [Cri97] Richard Crisp. Direct Rambus technology: The new main memory standard. *IEEE Micro*, 17(6):18–28, November/December 1997.
- [Cro97] C. Crowley. *Operating Systems: A Design-Oriented Approach*. Irwin Publishing, 1997.
- [CSB86] D.R. Cheriton, G. Slavenburg, and P. Boyle. Software-controlled caches in the VMP multiprocessor. In *Proc. 13th Int. Symp. on Computer Architecture (ISCA '86)*, pages 366–374, Tokyo, June 1986.
- [Dul98] C. Dulong. The IA-64 architecture at work. *Computer*, 31(7):24–32, July 1998.
- [Fat90] R.A. Fatoohi. Vector performance analysis of the NEC SX-2. In *Proc. Int. Conf. on Supercomputing*, pages 389–400, 1990.
- [Han98] J. Handy. *The Cache Memory Book*. Academic Press, San Diego, CA, 2nd edition, 1998.
- [HH93] J. Huck and J. Hays. Architectural support for translation table management in large address space machines. In *Proc. 20th Int. Symp. on Computer Architecture (ISCA '93)*, pages 39–50, San Diego, CA, May 1993.
- [HKT93] Y. Hidaka, H. Koike, and H. Tanaka. Multiple threads in cyclic register windows. In *Proc. 20th Annual Int. Symp. on Computer architecture (ISCA '93)*, pages 131–142, San Diego, CA, May 1993.
- [HP96] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Francisco, CA, 2nd edition, 1996.

- [IBM97] IBM. *Synchronous DRAMs: The DRAM of the Future*. <http://www.chips.ibm.com/products/memory/sdramart/sdramart.html>, 1997.
- [IBM98] IBM. *PowerPC 750 RISC Microprocessor Technical Summary*. http://www.chips.ibm.com/products/ppc/documents/datasheets/750/750_TS_R%0.pdf, January 1998.
- [IKWS92] J. Inouye, R. Konuru, J. Walpole, and B. Sears. *The Effects of Virtually Addressed Caches on Virtual Memory Design and Performance*. Tech. Report No. CS/E 92-010, Department of Computer Science and Engineering, Oregon Graduate Institute of Science and Engineering, March 1992.
- [Int98] Intel. *Pentium II Processor Product Overview*. <http://developer.intel.com/design/PentiumII/prodbref/index.htm>, 1998.
- [JM97] B. Jacob and T. Mudge. Software-managed address translation. In *Proc. Third Int. Symp. on High-Performance Computer Architecture*, San Antonio, Texas, February 1997.
- [Jou90] N.P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. 17th Int. Symp. on Computer Architecture (ISCA '90)*, pages 364–373, May 1990.
- [KE97] D.R. Kaeli and P.G. Emma. Improving the accuracy of history-based branch prediction. *IEEE Transactions on Computers*, 46(4):469–472, April 1997.
- [KELS62] T. Kilburn, D.B.J. Edwards, M.J. Lanigan, and F.H. Sumner. One-level storage system. *IRE Transactions on Electronic Computers*, EC-11(2):223–35, April 1962.
- [KH92a] G. Kane and J. Heinrich. *MIPS RISC Architecture*. Prentice Hall, Englewood Cliffs, NJ, 1992.
- [KH92b] R.E. Kessler and M.D. Hill. Page placement algorithms for large real-indexed caches. *ACM Transactions on Computer Systems*, 10(4):338–359, November 1992.
- [KK97] A. Ki and A. E. Knowles. Adaptive data prefetching using cache information. In *Proc. 1997 Int. Conf. on Supercomputing*, pages 204–212, Vienna, 1997.
- [Kro81] D. Kroft. Lockup-free instruction fetch/prefetch cache organisation. In *Proc. 8th Int. Symp. on Computer Architecture (ISCA '81)*, pages 81–84, May 1981.
- [LEL⁺97] J.L. Lo, J.S. Emer, H.M. Levy, R.L. Stamm, and D.M. Tullsen. Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. *ACM Transactions on Computer Systems*, 15(3):322–354, August 1997.
- [Mac96] P. Machanick. The case for SRAM main memory. *Computer Architecture News*, 24(5):23–30, December 1996.
- [MLG92] T.C. Mowry, M.S. Lam, and A. Gupta. Design and evaluation of a compiler algorithm for prefetching. In *Proc. 5th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 62–73, September 1992.
- [MS98] P. Machanick and P. Salverda. Preliminary investigation of the RAMpage memory hierarchy. *South African Computer Journal*, 1998. In press. <http://www.cs.wits.ac.za/~philip/papers/rampage.html>.
- [NUS⁺93] D. Nagle, R. Uhlig, T. Stanley, S. Sechrest, T. Mudge, and R. Brown. Design tradeoffs for software-managed TLBs. In *Proc. 20th Int. Symp. on Computer Architecture (ISCA '93)*, pages 27–38, San Diego, CA, May 1993.
- [RHWG95] M. Rosenblum, S.A. Herrod, E. Witchel, and A. Gupta. Complete computer system simulation: The SimOS approach. *IEEE Parallel and Distributed Technology*, 3(4):34–43, Winter 1995.
- [RL92] A. Rogers and K. Li. Software support for speculative loads. In *Proc. 5th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-5)*, pages 38–50, September 1992.
- [SWL⁺92] M.L. Simmons, H.J. Wasserman, O.A. Lubeck, C. Eoyang, R. Mendez, H. Harada, and M. Ishiguru. A performance comparison of four supercomputers. *Comm. ACM*, 35(8):116–124, August 1992.
- [WB92] B. Wheeler and B.N. Bershad. Consistency management for virtually indexed caches. In *Proc. 5th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-5)*, pages 124–136, September 1992.
- [WW93] C.A. Waldspurger and W.E. Weihl. Register relocation: flexible contexts for multithreading. In *Proc. 20th Annual Int. Symp. on Computer Architecture (ISCA '93)*, pages 120–130, San Diego, CA, May 1993.