IMPLICATIONS OF EMERGING DRAM TECHNOLOGIES FOR THE RAMPAGE MEMORY HIERARCHY

Philip Machanick Pierre Salverda
Department of Computer Science
University of the Witwatersrand
Private Bag 3
2050 Wits
South Africa
{philip,psalverd}@cs.wits.ac.za

Abstract

The RAMpage memory hierarchy is an attempt at devising a comprehensive strategy to address the growing DRAM-CPU speed gap. By moving the main memory up a level to the SRAM currently used to implement the lowest-level cache, a RAMpage system in effect implements a fully associative cache with no hit penalty (in the best case). Ordinary DRAM is relegated to a paging device. This paper shows that even with an aggressive SDRAM conventional main memory (or equivalently the new Direct Rambus design proposed for 1999), a RAMpage hierarchy is over 16% faster than a conventional 2-level cache design, with a high-end CPU of a speed likely to be delivered in 1998. Further optimizations of the RAMpage hierarchy, such as context switches on misses, are likely to further improve this result.

Introduction

There is a growing CPU-DRAM gap [12, 4, 13]. The RAMpage memory hierarchy is an attempt at arriving at a comprehensive solution to the problem which reduces the overall time spent on DRAM references by reducing the number of misses to DRAM (at the expense of spending more time on each miss).

The RAMpage memory hierarchy replaces the lowest-level cache by a similar-sized paged memory implemented in SRAM. In effect, the main memory moves up a level and DRAM becomes a paging device. The result is that what was previously the lowest level of cache is fully software-managed, and is fully associative, without the usual penalties associated with a fully associative cache.

Since the RAMpage hierarchy's lowest level of SRAM is fully software-managed, other benefits can result from managing what is kept in that level of SRAM, including operating system data and code.

Research into the RAMpage architecture has so far emphasized the conditions under which the architecture is a win over a conventional cache architecture, with emphasis on miss behaviour to DRAM [24, 25].

However, there are many competing strategies which aim to reduce the cost of DRAM access. One such competing strategy is faster DRAM technologies, such as SDRAM, which is emerging as a mass-market standard. Although our earlier results with conventional DRAM backing up the SRAM main memory were promising [24, 25], we considered it important to compare the RAMpage hierarchy to a more aggressive conventional architecture, as SDRAM is now moving into the mainstream (for example, on faster PCs and Power Macintosh G3 models).

Since the RAMpage memory hierarchy aims to be a comprehensive solution to the growing CPU-DRAM speed gap, it is necessary to evaluate a range of effects, including TLB and operating system performance, to arrive at an overall evaluation of the proposed new model. However, to make it possible to see where performance effects are coming from, it is useful to break the evaluation down so smaller numbers of variables are tested at a time.

In this paper, results are presented showing the performance improvement of a RAMpage architecture only taking into account differences in overall miss penalties. We have done preliminary work on other variables, such as context switch and TLB performance; effects of these variables will be more fully reported in future work, and not presented fully in this paper.

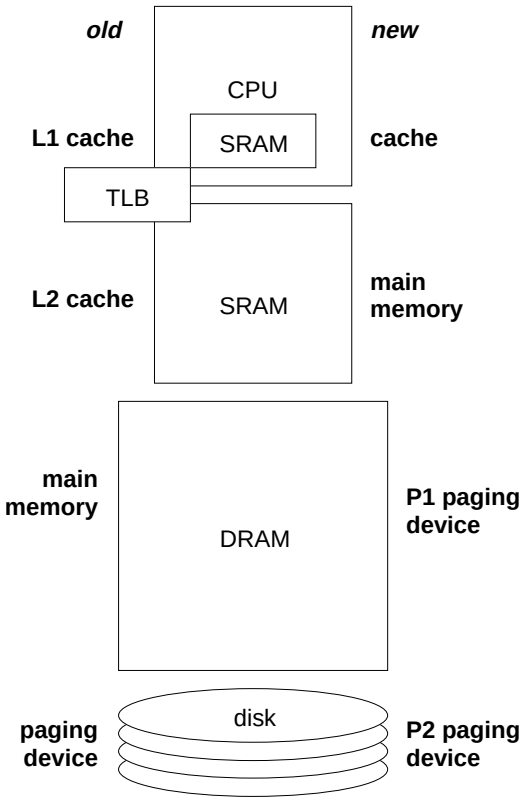

Figure 1: A Standard Hierarchy *vs.* a RAMpage Hierarchy

Figure 1 illustrates the key differences between the two hierarchies: a traditional two-level cache system, and a RAMpage system. Note that the major components are the same. Both systems use the same amount of SRAM, and both have a TLB to cache recent page translations; the difference is in the way they are managed.

The RAMpage hierarchy extends earlier work on software-managed caches [8, 7, 6, 19] by going all the way to implementing what was previously the lowest level of cache as a paged memory. There are two major differences between the RAMpage strategy and earlier work on software-menaged caches:

- hits can be handled immediately if the TLB hits, without the overhead of cache tag index and comparison, and extraction of bytes from a block (operating system code or hardware which works with physical addresses – such as snooping requests to ensure coherency in a multiprocessor implementation – can reference physical addresses in SRAM directly without translation)
- full associativity is implemented at no cost to best-case hit time

These changes are obvious improvements; the success of the RAMpage strategy has to be evaluated on the basis of the costs of making these changes, in particular, the trade-off between increased miss time required to handle a miss in a paged system versus the reduction in misses resulting from higher associativity.

For purposes of evaluation, we have not investigated the potential for a significant improvement in hit time, as hit time in conventional hierarchies can generally be improved by throwing more hardware at the problem (e.g. including the L2 tags on the CPU chip, as in the Pentium Pro and PowerPC 750). However, we do note that our solution potentially frees up chip real estate for other performance enhancements.

Simulations reported on here use a reasonably comprehensive memory hierarchy, including page management of DRAM. The DRAM level is managed in a similar fashion in both a convetional two-level cache system, and a RAMpage system. We assume an infinite DRAM so as to avoid the need to simulate the disk level, which is common to the two models.

Simulations are based on traces obtained from a trace archive at University of New Mexico and total approximately 1.1 billion references.

The rest of this paper is organized as follows. The next section describes the problem the RAMpage hierarchy is addressing in more detail, and relates the RAMpage approach to other solutions. Section presents the principles of the RAMpage model in more detail. Next comes Section , which defines the two competing architectures being simulated, and explains which aspects of the RAMpage model are included in measurements presented in this paper. Section contains results of simulations. The paper concludes with a summary of results and a discussion of their significance.

The Problem and Related Work

Introduction

The growing CPU-DRAM speed gap creates a need to work around the gap. Most solutions to the problem are piecemeal: they offer a one-shot improvement, after which the problem arises again with the next change in technology.

This section examines some alternative approaches to reducing the CPU-DRAM speed gap, and points out why they are not comprehensive solutions. Section builds on this background to explain why the RAMpage model goes further than other solutions.

First, this section considers trends which give rise to the CPU-DRAM speed gap. It goes on to examine proposed solutions, and concludes by comparing these solutions to the RAMpage model.

Trends

Since the mid-1980s, CPU speed has improved by 50 to 100% per year. At the same time, DRAM speed has only improved by 7% per year [13]. Consequently, every 6.2 years, the time for a DRAM access is in effect doubled with respect to CPU speed [4].

To see what effect this trend has, consider the IBM Power3 CPU [15], announced in 1997 for shipment in 1998. This processor can issue up to 8 instructions per clock cycle at speeds of 500MHz or more, a peak issue rate of one instruction every 0.25ns. With a simple DRAM main memory resulting in an L2 miss cost of 200ns, a Power3-based system would lose up to 800 instructions on a cache miss in the worst case (assuming no other memory hierarchy delays).

Obviously, by the time the Power3 CPU is shipped, DRAM speeds will have improved but, even so, a miss cost in high hundreds of lost instructions is a major penalty – not out of line with page fault penalties of early virtual memory systems of the 1960s [23].

If these trends persist, we are not far off a time when miss penalties will be over 1000 instructions.

Other Solutions

Since the major speed gap to be bridged is between the lowest level of cache and main memory, this discussion focusses on work in that area, although there has been a significant amount of work on reducing misses from the L1 cache[26].

One of the problematic issues with large L2 caches is implementing associativity without a significant performance penalty [13].

There are various approaches to reducing the cost (both in money and speed) of associativity, in which a direct-mapped cache is extended to allow an alternative location for a given block. Column associative caches are a representative example of this work [1]. The general idea is that a hit at the primary location of a block is as quick as for a direct-mapped cache, and the penalty for finding the block at its alternative location should not arise often enough to offset the saving from the faster best-case hits.

An alternative to cheaper forms of associativity is to reduce misses in direct-mapped caches. One approach

to reducing misses is a small additional buffer for storing victims of recent misses, a *victim cache*. A victim cache is likely to have most effect with a relatively small cache. However, the idea can in principle work with a large L2 cache [20].

There have been various approaches to mapping pages to physical memory to minimize conflict misses in direct-mapped caches [3, 22].

Since the underlying problem is DRAM speed, there has been work on better organizations of DRAM, including the SDRAM used in simulations in this paper [16].

There has also been work on reducing the impact of a miss on overall run time, including non-blocking caches and prefetch instructions [5, 11].

Finally, it is worth noting various approaches to software-managed caches, though these approaches are focussed at different problems than the RAMpage model.

Software-managed caches on VMP were designed to reduce memory traffic in a multiprocessor context, and most of the issues involved do not apply to a uniprocessor system[8, 7, 6]. Unlike RAMpage which implements full associativity in software, VMP used 4-way set associative caches [6].

Other more recent work on software-controlled caches was designed to do efficient address translation on a miss with a virtually addressed cache, and is therefore not closely related to the problem RAMpage is addressing (caches in this work were direct-mapped) [19].

RAMpage Versus Other Solutions

Other solutions to the CPU-DRAM speed gap generally at best hide the underlying trend, they do not provide a scalable way of working around it.

A modest reduction in the number of misses to DRAM (as in approaches to improve direct-mapped caches) simply delays the point where misses to DRAM will dominate performance.

Non-blocking caches are a one-shot fix in the sense that instruction-level parallelism limits the gains. The higher the miss penalty, the smaller the fraction of a miss can usefully be used up by other instructions [2].

At the same time, some of the underlying ideas can be layered on top of RAMpage. A victim cache can easily be implemented in software, by the standard technique used in operating systems (recently replaced pages are kept on a *standby page list*; the page which was on the list longest is the one actually discarded [10]).

Although previous work on software-managed caches does not directly apply, there are some lessons to be learned from that work. For example, the VMP caches required a special memory to hold cache control code and data [8]. However, if we implement a paged memory in the lowest-level SRAM instead of a cache, we can use the standard operating system strategy of locking a given range of pages reserved for operating system use – a principle that can apply to other important operating system memory. Also, the previous work on software-managed caches where the cache is virtually addressed has provided some insights into how to do addres translation efficiently for a relatively small portion of the virtual address space [6, 19].

RAMpage Principles

Introduction

Given that other solutions do not scale up as well as one might like, it is useful to consider whether the RAMpage model offers more, or whether it is simply another solution no better than the others.

This section examines how the RAMpage model reduces the impact of the CPU-DRAM gap, and goes on to consider how the RAMpage model could scale up better than other approaches. Finally, the main principles of the model are summarized.

Reducing the Impact of the CPU-DRAM Gap

Designers of large L2 caches are faced with a dilemma. If they make the cache highly associative, hit time suffers. If they aim for speed and go for a simple direct-mapped cache, misses increase. The growing penalty for DRAM accesses favours reducing misses rather than emphasizing hit speed.

The RAMpage model offers the possibility of a simple (and therefore fast) hit mechanism, with full associativity. Provided that the address translation is available (which we expect will be the case, since the TLB generally runs in parallel with an L1 access), a hit is a simple memory reference, with no cache tag overheads (indexing, comparison), and no sub-block extraction. Misses are slower, since a software miss handler is required. However, full associativity plus the possibility of using a more sophisticated replacement strategy reduce the number of misses.

Once miss penalties become high enough, the RAMpage model offers the opportunity to take a context switch on a miss – just as happens on a conventional page fault. In this way, the higher latency of a RAMpage miss relative to a onrmal cache miss can be hidden, reducing the need for RAMpage to win purely by having fewer misses.

Scalability of the RAMpage Hierarchy

Assuming a point is reached where relative speeds of components favour the RAMpage model, as the CPU-DRAM speed gap grows, the RAMpage model will look increasingly attractive. Unlike with a cache, speed is unrelated to size (to the extent that static RAM is affordable, and its higher density allows it to fit on a low-cost logic board), so relatively large and potentially upgradeable RAMpage SRAM main memories can be implemented. By contrast, new designs which achieve associativity in hardware require on-chip L2 tags to achieve acceptable L2 hit times, which places a hard upper limit on L2 size (512KB for the Pentium II [18], and in the case of the PowerPC 750 [17], 1MB).

Also, the trade-off of fewer misses in exchange for spending time in a software miss handler will look increasingly attractive as miss penalties increase. A key point here is that the miss handler should be SRAM-resident, so its performance is not scaled down as the CPU-DRAM speed gap grows.

The possibility of locking parts of the operating system in SRAM can also potentially improve scalability, as time spent on activities such as TLB miss handling and context switching may be reduced.

Finally, taking a context switch on a miss – provided there are processes available to run – will be an increasingly attractive option as the CPU-DRAM speed gap increases.

Main Principles

In summary, the main principles of the RAMpage model are:

- the lowest level of cache is replaced by an equivalent-sized main memory implemented in SRAM
- the previous main memory becomes a first-level paging device, while the disk becomes a second-level paging device
- misses to DRAM are handled in software as page faults
- full associativity with a page replacement strategy borrowed from virtual memory practice is used to reduce misses
- the downside is slower miss-handling, and the key issue in assessing the viability of the RAMpage model is the trade-off between this extra miss penalty versus the reduction in misses

Simulation Parameters

Introduction

This section contains a summary of the common points between the RAMpage and the conventional cache architecture which are compared in this paper, followed by points of difference between the two models. Finally, the traces used for simulation are described.

Common Features of The Two Architectures

Both memory hierarchies use the same amount of memory. To avoid the need to simulate page faults to disk (which we expect to be the same in both hierarchies), we simulate an infinite DRAM which is preloaded with data and code. Both hierarchies have 16K each of instruction and data L1 cache, both physically tagged and indexed, with 32-byte blocks. The TLB has 64 entries shared between code and data, and is fully associative with a hardware-implemented random replacement policy.

DRAM is modelled as SDRAM with a latency of 50ns for an initial access and 10ns for subsequent accesses. The bus to DRAM is fixed at a speed of 100MHz, to match the time for burst-mode accesses. The bus is 128 bits wide. This DRAM system is an aggressive design by today's standards.

The speed of the SDRAM modelled here which will be available in 1998 or early 1999[1]. Alternatively, the DRAM level could be implemented using the proposed Direct Rambus [9], which will be available in 1999, and which has the same latency and bandwidth as the SDRAM modelled in this paper. Although some of the more aggressive features of Direct Rambus (particularly, pipelining multiple independent requests) are not modelled, the DRAM subsystem is a reasonable approximation to what will be available in the mass market in the 1998-1999 timeframe.

Misses to DRAM take varying amounts of time, according to the page or block size being measured.

In both hierarchies, page translations to DRAM is handled using an inverted page table (for simplicity, the same model as is used for the RAMpage SRAM main memory, though in the RAMpage case, the DRAM page table still uses a fixed page size when the SRAM page size is varied).

Features Specific to the Conventional Memory Hierarchy

The L2 cache on the conventional hierarchy is 4Mbytes, a size likely to become increasingly common on high-end designs (it was already a possibility on the MIPS R4000 [21]). It is direct-mapped, and connects to the CPU via a 128-bit bus running at one-third CPU speed[2].

In all simulations of the conventional hierarchy, we assume a miss to L2 cache takes 12 CPU cycles. We assume perfect write buffering so there is no penalty for a writeback on a dirty miss. Since RAMpage has significantly fewer writebacks than the conventional hierarchy, this assumption does not favour the new approach.

Features Specific to the RAMpage Hierarchy

The SRAM main memory on the RAMpage hierarchy has very similar basic attributes to the conventional L2 cache. The bus speed and width are the same, and read costs are the same. Writes, on the other hand, are 25% faster, since tag checks are elminated (we assume tag checks on a read can be done in parallel with an L1 access on the conventional hierarchy).

To compare like with like, we increase the size of the RAMpage SRAM main memory to 4.125Mbytes, to account for the space needed for L2 tags on the cache hierarchy (based on the space needed for tags for 128byte blocks).

[1] 100MHz buses appeared with the 400MHz Pentium II generation in April 1998, but it is not clear that SDRAM of the specificiation simulated here was available at the time.

[2] Although some recent CPUs such as the PowerPC 750 allow for an L2 bus cache at full CPU speeds, such CPUs are superscalar, so a one-third CPU-speed bus is not a conservative design.

For the RAMpage hierarchy, since the SRAM main memory is a relatively small part of the total address space, we have chosen to use an inverted page table [14]. An inverted page table has one entry per physical page, and is ordered on phyiscal page number, and uses hashing on virtual to look up replacements. The trade-off here is a smaller page table versus longer lookup time, which we considered acceptable in view of our desire to keep the entire page lookup process in SRAM wherever possible. The page replacement strategy uses a simple clock algorithm [10] which simulates least recently used by clearing a used bit on each clock sweep, and resetting it on each use of the page table entry. We simulate the varying overhead of the clock algorithm by interleaving traces of the algorithm of varying length in the simulation, whenever a page fault occurs.

The RAMpage hierarchy, although it uses the same page table implementation as the conventional model, does not use a TLB for translations to DRAM, since translations to DRAM addresses are not on the critical path for a cache hit.

Traces

To reduce the effects of cold misses on the overall miss rate, a trace which totals slightly less than 1.1 billion references has been used to drive the simulators.

References from 18 user-level programs were obtained from the trace archive at the University of New Mexico[3], and concatenated in units of 500 000-reference long sequences (to simulate context switching), yielding the final 1.1 billion-reference trace. Table 1 lists those 18 traces.

Table 1: Address traces used in the simulations.

Program	Description	Instruction fetches	Total references
alvinn	neural net training (from SPECfp92)	59 027 112	72 814 137
awk	unix text utility	62 834 833	86 435 124
cexp	from SPECint92	28 460 654	37 512 032
compress	file compression utility (SPECint92)	8 014 160	10 459 159
ear	human ear simulator (SPECfp92)	65 000 001	80 400 251
gcc	C compiler (SPECint92)	78 798 313	100 000 000
hydro2d	physics computations (SPECfp92)	8 248 427	10 985 700
mdljdp2	solves motion equations (SPECfp92)	65 000 000	84 233 871
mdljsp2	solves motion equations (SPECfp92)	65 000 003	76 954 695
nasa7	NASA applications (SPECfp92)	65 000 000	99 731 796
ora	ray tracing (SPECfp92)	65 000 009	82 942 488
sed	unix text utility	7 717 459	9 752 248
su2cor	physics computations (SPECfp92)	65 000 001	88 755 536
swm256	physics computations (SPECfp92)	65 000 001	87 416 474
tex	unix text utility	50 288 264	66 829 759
uncompress	file decompression utility (SPECint92)	5 689 5	7 458 670
wave5	solves particle equations	65 000 000	78 282 009
yacc	unix text utility	9 664 466	12 186 384

Note that overheads of context switching are not modelled, so we can focus here on differences in application miss behaviour. Pinning context switching code and scheduler data structures in SRAM is one of the goals of the RAMpage model. Hence, excluding context switching overheads from our simulations does not favour RAMpage over the conventional cache model.

[3]The traces are available from `ftp://tracebase.nmsu.edu/pub/.tb1/r2000/utilities/` and `ftp://tracebase.nmsu.edu/pub/.tb1/r2000/SPEC92/`.

Results

Introduction

Simulation measurement aims to assess the trade-off offered by the RAMpage hierarchy, as well as to assess its scalability. In this paper, the focus is on memory reference costs, so we have not included context switching costs, or investigated context switches on misses to DRAM.

Results are presented in the following order. First, a count of references to DRAM for both hierarchies is presented with varying block (page) sizes. Second, resulting simulated run times are presented, with a low-end processor running at 200MHz. Next, memory management overhead (TLB misses and page management penalties) are presented. Finally results are discussed.

References to DRAM

The data in Table 2 verifies the projected benefits of the RAMpage hierarchy. The rightmost column contains the percentage improvement in total DRAM references (both reads and writes) achieved by the new hierarchy over the standard. This may be read as an improvement in miss rate.

Table 2: Total number of DRAM references incurred by each hierarchy after simulation of approximately 1.1 billion trace file references (one reference is one block or page).

SRAM block size (B)	Standard hierarchy		New hierarchy		Percent improvement
	Reads	Writes	Reads	Writes	
128	915 796	404 268	510 540	263 697	41
256	648 173	261 608	244 563	131 203	59
512	522 491	191 386	123 440	67 124	73
1024	462 148	159 287	64 597	35 472	84
2048	605 749	140 944	35 312	19 294	93
4096	719 170	158 512	21 623	11 339	96

The extremely low number of references for a 4K page size on the RAMpage hierarchy is very promising. Although the cost of transferring such a large page means that this page size may not result in the best run time, the low number of page faults suggests the possibility of context switches on misses may be viable in this case as an alternative to a faster miss time without taking a context switch.

Also of interest is the fact that large page sizes in the RAMpage hierachy do not increase misses as a result of increased conflict misses, as one would expect in a cache of limited associativity [13].

Simulated Run Times

As a baseline comparison, we include figures for a 200MHz (5ns) clock single-instruction per cycle CPU.

That data appears in Table 3, and is plotted in Figure 2.

Table 3: CPU cycles consumed by each hierarchy in processing 1.1 billion trace file references.

SRAM block size (B)	Elapsed simulated time (CPU cycles)		Percent improvement
	Standard hierarchy	New hierarchy	
128	2.3691	2.7902	-18
256	2.3710	2.5158	-6
512	2.3904	2.3767	1
1024	2.4360	2.2914	6
2048	2.6049	2.2999	12
4096	2.9774	2.3122	22

Note that the best run time occurs for a block size of 128bytes in the conventional architecture, but for a

page size of 1K for the RAMpage model. While RAMpage is slower for small block sizes (indicated by negative percentage improvement), it is faster for large block (or page) sizes. The best case of RAMpage, even for a modest-speed CPU (obsolete by 1998 standards: a low-end processor early in 1998 runs at over 200MHz and is superscalar), is 3% faster than for the conventional hierarchy.

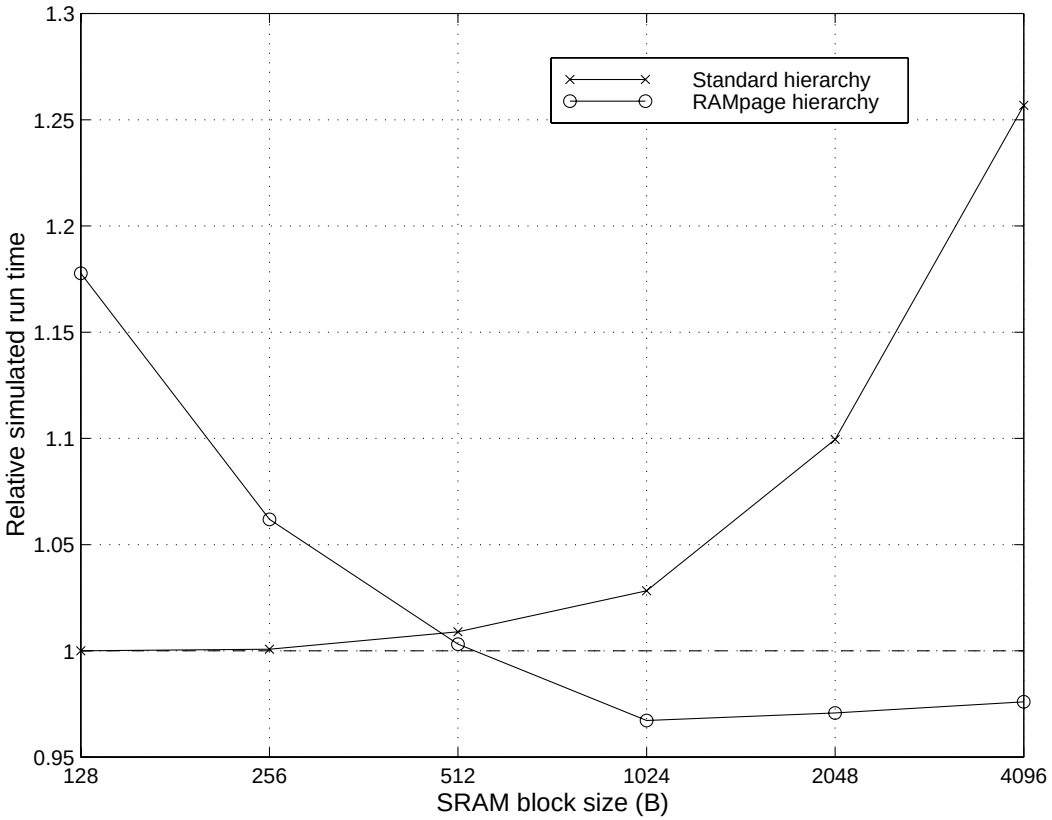

Figure 2: Relative performance of the new and standard hierarchies. Each curve plots the total elapsed simulated run time for each hierarchy relative to that of the 128B-block configuration of the standard hierarchy.

Memory Management Overhead

The reason for RAMpage's slower performance for smaller page sizes can be seen from examining memory management overhead. TLB misses for page sizes below 1K become significant compared with the conventional hierarchy, whereas the difference is smaller for larger page sizes – and can be compensated for by lower misses as well ensuring that memory management references are all in SRAM. Note that TLB hit rates in the conventional hierarchy are independent of cache block size.

Figure 3 plots the overheads incurred by memory management software in each hierarchy; we have not separated out times contributed by TLB misses and other page translation overhead. The overheads are represented in the figure as the percentage of additional references incurred by memory management in both hierarchies (the conventional hierarchy's overheads do not vary with cache block size).

Increasing the CPU-DRAM speed gap

Since the RAMpage model aims to be scalable in the face of the growing CPU-DRAM speed gap, it is useful to consider the effect of scaling up the CPU speed.

We have taken the CPU speed up to 4GHz, with a total of 5 speeds modelled. We are only modelling a

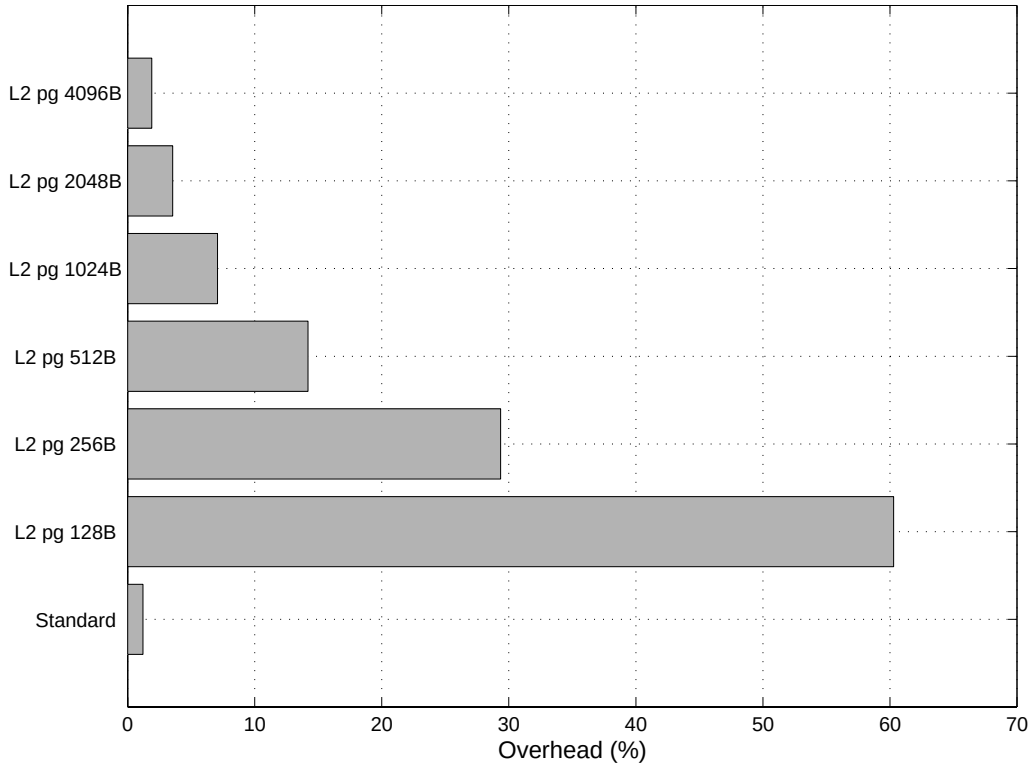

Figure 3: TLB miss and page fault handling overheads as a percentage of references additional to those in the original traces.

single cycle per instruction CPU for simplicity. A superscalar design such as the IBM Power 3 will be capable of issuing instructions at 4GHz (8 instructions per cycle at 500MHz).

Simulated run times appear in Table 4.

Table 4: Elapsed simulated time (seconds) for 1.1 billion-reference trace. Each row contains standard hierarchy at the top, new hierarchy below, with percentage improvement of new hierarchy over standard underneath.

CPU clock speed (ns)	SRAM block size					
	128B	256B	512B	1024B	2048B	4096B
5	11.8456	11.8550	11.9520	12.1802	13.0246	14.8871
	13.9508	12.5792	11.8833	11.4572	11.4994	11.5611
	-18%	-6%	1%	6%	12%	22%
2	4.8492	4.8621	4.9436	5.1331	5.8102	7.3346
	5.6454	5.0693	4.7971	4.6250	4.6437	4.6762
	-16%	-4%	3%	10%	20%	36%
1	2.5172	2.5311	2.6074	2.7840	3.4054	4.8170
	2.8905	2.5780	2.4359	2.3515	2.3584	2.3813
	-15%	-2%	7%	16%	31%	51%
0.5	1.3510	1.3656	1.4393	1.6095	2.2030	3.5583
	1.4995	1.3294	1.2546	1.2108	1.2158	1.2338
	-11%	3%	13%	25%	45%	65%
0.25	0.7680	0.7829	0.8553	1.0221	1.6018	1.8552
	0.8034	0.7060	0.6640	0.6408	0.6445	0.6601
	-5%	10%	22%	37%	60%	64%

The best-case run times occur as before with a block size of 128 with the standard hierarchy, and a page size of 1K for the RAMpage hierarchy.

As can be seen from Table 4, the speed gap between the RAMpage and conventional hierarchies grows as CPU speed increases in general.

This effect is shown more clearly in Figure 4, which plots the relative performance of the two hierarchies across all variations in processor speed (for the best block or page size: 128 bytes for the conventioinal hierarchy, 1K for RAMpage).

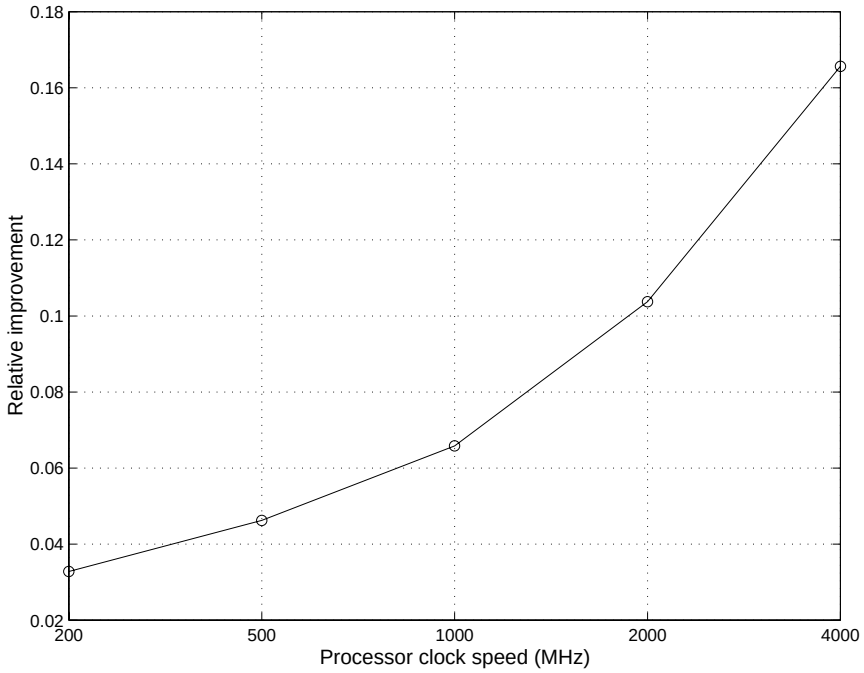

Figure 4: Relative improvements in performance of best-case block or page sizes of the RAMpage hierarchy over those for the standard hierarchy.

Discussion of Results

The measurements we have presented in this paper clearly make the case for the RAMpage hierarchy, only taking into account memory hierarchy effects. Our figures on number of references to DRAM provide a strong basis for supporting the claim that RAMpage is a win even on fairly modest hardware, a claim which is further backed up by our data on run times and memory management overheads.

Our scalability measures also indicate the potential of RAMpage to be a significant win on machines of the near future, with a performance improvement (with the best-case block or page size) of over 16% a conventional hierarchy.

Conclusions

Introduction

This section summarizes future work, and concludes with a final summary of the findings of this paper and the goals of the RAMpage project.

Future Work

We are working on simulation of context switch costs. Although our data is not ready to report on in detail, preliminary results indicate that maintaining operating system data and code for handling context switches is a win, especially under relatively high load. Context switches on misses appear to be viable with relatively large page sizes, which may make it viable to use page sizes greater than 1K.

We have also started to investigate the impact of a more aggressive L2 cache design in a conventional architecture. Recent designs with associative L2 caches have used on-chip L2 tags and logic to achieve acceptable hit times [17, 18]. While the speed difference between a more aggressive L2 design and RAMpage is not going to be as great as with a direct-mapped L2, the more aggressive L2 needs more hardware than RAMpage and the comparison becomes an example of hardware-software trade-offs.

Our current page handling code is not particularly optimized. Given that RAMpage has higher memory management overheads than a conventional hierarchy, attention to optimization of page management would be a win.

More detailed simulation of operating system effects in general would be useful. Also, a more detailed simulation of the overall machine including other traffic on the bus to DRAM (graphics, DMA, etc.) would give a clearer picture as to the benefits of the considerable reduction in memory traffic in the RAMpage model.

We have also not paid attention to alternative uses of chip real estate which is freed up by eliminating on-chip L2 tags, which are becoming increasingly common in recent CPU designs.

Final Summary

Overall, results look promising. While more detailed measurement is needed in some areas, the RAMpage model, on data currently available, looks worth pursuing. A performance win of over 16% on a system of the near future is a useful start; if more detailed simulation can support this result or even show a stronger performance improvement is possible, RAMpage will be worth considering for real systems. Alternatively, a more aggressive conventional hierarchy may narrow the gap, but it then becomes interesting to consider what else could be done with the extra hardware required for the more aggressive L2 cache.

It may seem counter-intuitive to introduce a simpler hardware model when the trend is towards greater complexity. However, we are motivated by the same logic as modern CPU design [13], as exemplified by the RISC movement:

- make the common case fast
- simplify the hardware to make it easier to make the common case fast
- rely on software to reduce the impact of the less common case

Once maximum mileage has been extracted from simplicity, then one can make a case for adding complication – as has indeed happened with RISC processors.

We believe that the results presented in this paper make a strong case for further investigation of the RAMpage model, including more detailed measurement and refinements. The appealing thing about our work to date is that the most obvious refinements are software-related; the basic hardware model can remain simple.

Acknowledgements

Lance Pompe did the work on context switching which has been reported on briefly in this paper.

References

1. A Agarwal and S Pudar. ‘Column associative caches: A technique for reducing the miss rate of direct mapped caches’. In *Proceedings of the 20th International Symposium on Computer Architecture*, pp. 179–190, (May 1993).

2. S Belayneh and D Kaeli. ‘A discussion of non-blocking/lockup-free caches’. *Computer Architecture News*, **24**(3):18–25, (June 1996).

3. B Bershad, D Lee, T Romer, and J Chen. ‘Avoiding conflict misses dynamically in large direct-mapped caches’. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 158–170, (October 1994).

4. K Boland and A Dollas. ‘Predicting and precluding problems with memory latency’. *IEEE Micro*, **14**(4):59–67, (August 1994).

5. T Chen and J Baer. ‘Reducing memory latency via non-blocking and prefetching caches’. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 51–61, (September 1992).

6. D Cheriton, H Goosen, and P Boyle. ‘Multi-level shared caching techniques for scalability VMP-MC’. In *Proceedings of the 16th International Symposium on Computer Architecture*, pp. 16–24, Jerusalem, (May/June 1989).

7. D Cheriton, A Gupta, P Boyle, and H Goosen. ‘The VMP multiprocessor: Initial experience, refinements and performance evaluation’. In *Proceedings of the 15th International Symposium on Computer Architecture*, pp. 410–421, Honolulu, (May/June 1988).

8. D Cheriton, G Slavenburg, and P Boyle. ‘Software-controlled caches in the VMP multiprocessor’. In *Proceedings of the 13th International Symposium on Computer Architecture*, pp. 366–374, Tokyo, (June 1986).

9. R Crisp. ‘Direct Rambus tecnology: The new main memory standard’. *IEEE Micro*, **17**(6):18–28, (November/December 1997).

10. C Crowley. *Operating Systems: A Design-Oriented Approach*. Irwin Publishing, 1997.

11. L D., L J., G K., W W-D., G A., H J., H M., and M Lam. ‘The Stanford DASH multiprocessor’. *Computer*, **25**(3):63–79, (March 1996).

12. J Hennessy and N Jouppi. ‘Computer technology and architecture: An evolving interaction’. *Computer*, **24**(9):18–29, (September 1991).

13. J Hennessy and D Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kauffmann Publishers, San Mateo, CA, 2nd edition, 1996.

14. J Huck and J Hays. ‘Architectural support for translation table management in large address space machines’. In *Proceedings of the 20th International Symposium on Computer Architecture*, pp. 39–50, (May 1993).

15. IBM. ‘Next-generation high-performance 64-bit microprocessor family unveiled by IBM’. `http://www.chips.ibm.com/news/power3.html`, October 1997.

16. IBM. ‘Synchronous drams: The dram of the future’. `http://www.chips.ibm.com/products/memory/sdramart/sdramart.html`, 1997.

17. IBM. ‘PowerPC 740 and PowerPC 750 SCM RISC microprocessor: Hardware specifications’. `http://www.chips.ibm.com/products/ppc/documents/datasheets/750/740_750_%hwspec_rev10.gen.pdf`, January 1998.

18. Intel. ‘Pentium II processor product overview’. `http://developer.intel.com/design/PentiumII/prodbref/index.htm`, 1998.

19. B Jacob and T Mudge. ‘Software-managed address translation’. In *Proceedings of the Third International Symposium on High-Performance Computer Architecture*, San Antonio, Texas, (February 1997).

20. N Jouppi. ‘Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers’. In *Proceedings of the 17th International Symposium on Computer Architecture*, pp. 364–373, (May 1990).

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

22. R Kessler and M Hill. ‘Page placement algorithms for large real-indexed caches’. *ACM Transactions on Computer Systems*, **10**(4):338–359, (November 1992).

23. T Kilburn, D Edwards, M Lanigan, and F Summer. One-level storage system. Appears in *Computer Structures: Principles and Examples*, McGraw-Hill, New York, 1982.

24. P Machanick. ‘The case for SRAM main memory’. *Computer Architecture News*, **24**(5):23–30, (December 1996).

25. P Machanick and P Salverda. ‘Preliminary investigation of the rampage memory hierarchy’. *South African Computer Journal*, (21):16–25, (August 1998).

26. K Wilson and K Olukotun. ‘Designing high bandwidth on-chip caches’. In *Proceedings of the 24th International Symposium on Computer Architecture*, pp. 121–132, Denver, (June 1997).