L1 Cache and TLB Enhancements to the RAMpage Memory Hierarchy

Philip Machanick[1] and Zunaid Patel[2]

[1] School of ITEE, University of Queensland
Brisbane, Qld 4072, Australia
philip@itee.uq.edu.au
[2] School of Computer Science, University of the Witwatersrand,
Johannesburg, Private Bag 3, 2050 Wits, South Africa
zunaid@cs.wits.ac.za

Abstract. The RAMpage hierarchy moves main memory up a level to replace the lowest-level cache by an equivalent-sized SRAM main memory, with a TLB caching page translations for that main memory. This paper illustrates how more aggressive components higher in the hierarchy increase the fraction of total execution time spent waiting for DRAM. For an instruction issue rate of 1 GHz, the simulated standard hierarchy waited for DRAM 10% of the time, increasing to 40% at an instruction issue rate of 8 GHz. For a larger L1 cache, the fraction of time waiting for DRAM was even higher. RAMpage with context switches on misses was able to hide almost all DRAM latency. A larger TLB was shown to increase the viable range of RAMpage SRAM page sizes.

1 Introduction

The RAMpage memory hierarchy moves main memory up a level to replace the lowest-level cache with an SRAM main memory, while DRAM becomes a first-level paging device. Previous work has shown that RAMpage represents an alternative, viable design in terms of hardware-software trade-offs [22] and that it scales better as the CPU-DRAM speed gap grows, particularly by virtue of being able to take context switches on misses [21].

In previous work, it was hypothesized that RAMpage would be more competitive across a wider range of SRAM page sizes (equivalent to line size of the lowest-level cache) with a more aggressive TLB. Secondly, it was hypothesized that a more aggressive L1 cache would emphasize differences in lower levels of the hierarchy. In this paper, we report on investigation of both hypotheses as separate issues. Improving the TLB and L1 has different effects on performance. The intent in presenting both in the same paper is to add several data points to our case for RAMpage.

In some studies, TLB misses have accounted for as much as 40% of run time [13], with figures in the region of 20–30% common [6, 23]. RAMpage has the potential to reduce the significance of the TLB on performance for two reasons. Firstly, unless the reference which causes a TLB miss would also miss in the

SRAM main memory, no reference to update the TLB needs go to DRAM, with the page table organization used for RAMpage. Secondly, there is no mismatch between the size of page mapped by the TLB and the "line size" of the "lowest-level cache", as would be the case with a conventional hierarchy. Consequently, the TLB can more easily be designed to map a specific fraction of the SRAM main memory, than is the case for a conventional cache.

The role of increasingly aggressive on-chip caches also needs to be evaluated, against the view that such caches address the memory wall problem. Quadrupling the size of a cache may halve the number of misses [28], but such expansion may not always be practical. Increasing the size of caches in any case makes it harder to scale up their speed [11].

The approach in this paper is to compare RAMpage with a conventional 2-level cache hierarchy as the size of the TLB scales up, across different SRAM main memory page sizes, as well as a variety of L1 cache sizes, in separate experiments. The simulated L2 cache of 4Mbytes runs at a third of the issue rate excluding misses. The intent is to emphasize that even a very fast, large on-chip cache results in a large fraction of run-time being spent waiting for DRAM. Even so, given that DRAM references are the dominant effect being measured, a fast cache should not invalidate the general trends being studied.

TLB measurements show that both models see a reduction in TLB miss rates as the TLB size increases, but RAMpage becomes more viable with smaller SRAM main memory page sizes. Cache measurements show that as L1 size increases, the fraction of time spent waiting for DRAM increases (even if overall run time decreases), which makes the option in the RAMpage hierarchy of taking a context switch on a miss more attractive.

The remainder of this paper is structured as follows. Section 2 presents more detail of the RAMpage hierarchy and related research. Section 3 explains the experimental approach, while Section 4 presents experimental results. In conclusion, Section 5, summarizes the findings and outlines future work.

2 Background

The RAMpage model was proposed [20] in response to the memory wall [30, 16]. The key idea of the RAMpage model is to minimize hardware complexity, while moving more of the memory management intelligence into software. A RAMpage machine therefore looks very like a conventional model, except the lowest-level cache is replaced by a conventionally-addressed physical memory, though implemented in SRAM rather than DRAM.

A number of other approaches to addressing the memory wall have been proposed. This section summarizes the memory wall issue, followed by more detail of RAMpage. After presenting other alternatives, the options are discussed.

2.1 Memory Wall

The memory wall is the situation where the effect of CPU improvements becomes insignificant as the speed improvement of DRAM becomes a limiting factor. Since

the mid-1980s, CPU speeds have improved at a rate of 50-100% per year, while DRAM latency has only improved at around 7% per year [12]. If predictions of the memory wall [30] are correct, DRAM latency will become a serious limiting factor in performance improvement. Attempts at working around the memory wall are becoming increasingly common [9], but the fundamental underlying DRAM and CPU latency trends continue [27].

2.2 The RAMpage Approach

RAMpage is based on the notion that DRAM, while still orders of magnitude faster than disk, is increasingly starting to display one attribute of a peripheral: there is time to do other work while waiting for it [24], particularly if relatively large units are moved between DRAM and SRAM level. In RAMpage, the lowest-level cache is managed as the main memory (i.e., as a paged virtually-addressed memory), with disk a secondary paging device. The RAMpage main memory page table is inverted, to minimize its size. An inverted page table has another benefit: no TLB miss can result in a DRAM reference, unless the reference causing the TLB lookup is not in any of the SRAM layers [22].

RAMpage is intended to have the following advantages:

- *fast hits* – a hit physically addresses an SRAM memory
- *full associativity* – full associativity through paging avoids the slower hits of hardware full associativity
- *software-managed paging* – replacement can be as sophisticated as needed
- *TLB missess to DRAM minimized* – as explained above
- *pinning in SRAM* – critical OS data and code can be pinned in SRAM
- *hardware simplicity* – the complexity of a cache controller is removed from the lowest level of SRAM
- *context switches on misses to DRAM* – the CPU can be kept busy

These advantages come at the cost of slower misses because of software miss-handling, and the need to make operating system changes. However, the latter problem could be avoided by adding hardware support for the model.

The RAMpage approach has in the past been shown to scale well in the face of the grown CPU-DRAM speed gap, particularly with context switches on misses. The effect of context switches on misses is that, provided there is work available for the CPU, waiting for DRAM can effectively be eliminated [21]. Context switches on misses have the most significant effect.

2.3 Alternatives

Approaches to addressing the memory wall can loosely (with some overlaps) be grouped into latency tolerance and miss reduction. Some approaches to latency tolerance include prefetch, critical word first, memory compression, write buffering, non-blocking caches, and simultaneous multithreading (SMT).

Prefetch requires loading a cache block before it is requested, either by hardware [5] or with compiler support [25]; predictive prefetch attempts to improve accuracy of prefetch for relatively varied memory access patterns [1]. In critical word first, the word containing the reference which caused the miss is fetched first, followed by the rest of the block [11]. Memory compression in effect reduces latency because a smaller amount of information must be moved on a miss. The overhead must be less than the time saved [18]. There are many variations on write miss strategy, but the most effective generally include write buffering [17]. A non-blocking cache (lockup-free) cache can allow an aggressive pipeline to continue with other instructions while waiting for a miss [4].

SMT is aimed at masking DRAM latency as well as other causes of pipeline stalls, by hardware support for more than one active thread [19]. SMT aims to solve a wider range of CPU performance problems than RAMpage.

These ideas have costs (e.g., prefetching can displace needed content, causing unnecessary misses). The biggest problem is that most of these approaches do not scale with the growing CPU-DRAM speed gap. Critical word first is less helpful as latency for one reference grows in relation to total time for a big DRAM transaction. Prefetch, memory compression and nonblocking caches have limits as to how much they can reduce effective latency. Write buffering can scale provided buffer size can be scaled, and references to buffered writes can be handled before they are written back. SMT could mask much of the time spent waiting for DRAM, but at the cost of a more complex CPU.

Reducing misses has been addressed by increasing cache size, associativity, or both. There are limits on how large a cache can be at a given speed, so the number of levels has increased. Full associativity can be achieved in hardware with less overhead for hits than a conventional fully-associative cache, in an indirect index cache (IIC), by what amounts to a hardware implementation of RAMpage's page table lookup [10]. A drawback of IIC is that all references incur overhead of an extra level of indirection. Earlier work on software-based cache management has not focused on replacement policy [7, 14].

The advantages of RAMpage over SMT and other hardware-based multithreading approaches are that the CPU can be kept simple, and software implementation of support for multiple processes is more flexible (the balance between multitasking and multithreading can be dynamically adjusted, according to workload). An advantage of IIC is that the OS need not be invoked to handle the equivalent of a TLB miss in RAMpage. As compared with RAMpage, an IIC has more overhead for a hit, and less for a miss.

2.4 Summary

RAMpage masks time which would otherwise be spent waiting for DRAM by taking context switches on misses. Other approachs either do not aim to mask time spent waiting for DRAM, but to reduce it, or require more complex hardware. RAMpage can potentially be combined with some of the other approaches (such as SMT), so it is not necessarily in conflict with other ideas.

3 Experimental Approach

This section outlines the approach to the reported experiments. The simulation strategy is explained, followed by some detail of simulation parameters; in conclusion, expected findings are discussed.

3.1 Simulation Strategy

A range of variations on a standard 2-level hierarchy is compared to similar variations on RAMpage, with and without context switches on misses. RAMpage without context switches on misses to conveys the effects of adding associativity (with an operating system-style replacement strategy). Adding context switches on misses shows the value of alternative work on a miss to DRAM. Simulations are trace-driven, and do not model the pipeline. Processor speed is in GHz, representing instruction issue rate without misses, not clock speed.

Ignoring the pipeline level neglects effects like branches and the potential for other improvements like non-blocking caches. However, the results being looked for here are relatively large improvements, so inaccuracies of this kind are unlikely to be significant. What is important is the effect as the CPU-DRAM speed gap increases, and the simulation is of sufficient accuracy to capture such effects, as has been demonstrated in previous work.

3.2 Simulation Parameters

Parameters are similar to previous published work to make results comparable. The following parameters are common across RAMpage and the conventional hierarchy. This represents the baseline before new L1 and TLB variations:

- L1 cache – 16Kbytes each of data and instruction cache, physically tagged and indexed, direct-mapped, 32-byte block size, 1-cycle read hit, 12-cycle penalty for misses to L2 (or RAMpage SRAM main memory); for data cache: perfect write buffering (zero effective hit time), writeback (12-cycle penalty; 9 cycles for RAMpage: no L2 tag to update), write allocate on miss
- TLB – 64 entries, fully associative, random replacement
- DRAM level – Direct Rambus [8] 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 for simplicity, infinite DRAM with no misses to disk
- TLB and L1 data hits fully pipelined – only time for L1d or TLB replacements or maintaining inclusion costed as "hits"

The same memory timing is used as in earlier simulations. Although faster DRAM has since become available, the timing can be seen as relative to a particular CPU-DRAM speed gap, and the figures can accordingly be rescaled.

Context switches are modelled by interleaving a trace of text-book code. A context switch is taken every 500,000 references, though RAMpage with context switches on misses also takes a context switch on a miss to DRAM. TLB misses are handled by a trace of page table lookup code, with variations on time for a lookup based on probable variations in probes into an inverted page table [22].

specific to conventional hierarchy

The "conventional" system has a 2-way associative 4Mbyte L2. The L2 cache and its bus to the CPU the are clocked at one third of the CPU issue rate (the cycle time is intended to represent a superscalar issue rate). The L2 cache-CPU bus is 128 bits wide and runs Hits on the L2 cache take 4 cycles including the tag check and transfer to L1. Inclusion between L1 and L2 is maintained [12]. The TLB caches virtual page translations to DRAM physical frames.

specific to RAMpage hierarchy

In RAMpage simulations, most parameters remain the same, except that the TLB maps the SRAM main memory, and full associativity is implemented in software, through a software miss handler. The OS keeps 6 pages pinned in the SRAM main memory when simulating a 4 Kbyte-SRAM page, i.e., 24 Kbytes, which increases to to 5336 pages for a 128 byte block size, a total of 667 Kbytes.

inputs and variations

Traces are from the Tracebase trace archive at New Mexico State University[1]. 1.1-billion references are used, with traces interleaved to create the effect of a multiprogramming workload.

To measure variations on L1 caches, the size of each of the instruction and data caches was varied from the original size of 16 KB to 32 KB, 64 KB, 128 KB and 256 KB. To explore more of the design space, L1 block size was measured at sizes of 32, 64 and 128 bytes. We did not vary L2 block sizes when varying L1: an optimum size was determined in previous work [21, 22]. However, while varying the TLB, we did vary L2 block size in the conventional hiearchy, for comparison with varying the RAMpage SRAM main memory page size. To measure the effect of increasing the TLB size, we varied it from the original 64 entries to 128, 256 and 512. Even larger TLBs exist (e.g., Power4 has a 1024-entry TLB [29]), but this range is sufficient to capture variations of interest.

3.3 Expected Findings

As L1 becomes larger, RAMpage without context switches on misses should see less of a gain. While improving L1 should not affect time spent in DRAM, RAMpage's extra overheads in managing DRAM may have a more significant effect on overall run time. However, as the fraction of references in upper levels increases without a decrease in references to DRAM, context switches on misses should become more favourable.

As the TLB size increases, we expect to see smaller SRAM page sizes become viable. If the TLB has 64 entries and the page size is 4 KB with a 4 MB SRAM

[1] From `ftp://tracebase.nmsu.edu/pub/traces/uni/r2000/utilities/` and `ftp://tracebase.nmsu.edu/pub/traces/uni/r2000/SPEC92/`.

main memory, 6.25% of the memory is mapped by the TLB. If the TLB has 512 entries, the TLB maps 50% of the memory. By comparison, with a 128 B page, a 64-entry TLB only maps about 0.2% of the memory, and a big increase in the size of the TLB is likely to have a significant effect.

The effect on a conventional architecture of increasing TLB size is not as significant because it maps DRAM pages (fixed at 4 KB), not SRAM pages. Further, variation across L2 block sizes should not be related to TLB size.

4 Results

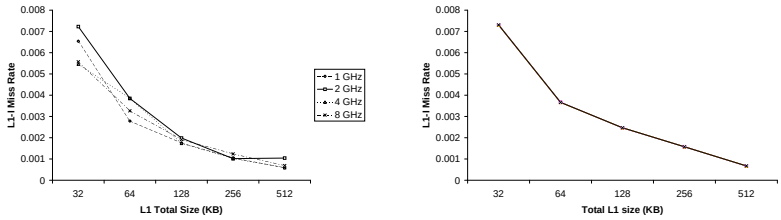
This section presents results of simulations, with some discussion. The main focus is on differences introduced by changes over previous simulations, but some advantages of RAMpage, as previously described, should be evident again from these new results. Presentation of results is broken down into effects of increasing L1 cache size, and effects of increasing TLB size, since these improvements have very different effects on the hierarchies modelled. Results are presented for 3 cases: the conventional 2-level cache with a DRAM main memory, and RAMpage with and without context switches on misses.

The remainder of this section presents the effects of L1 changes, then the effects of TLB changes, followed by a summary.

4.1 Increasing L1 Size

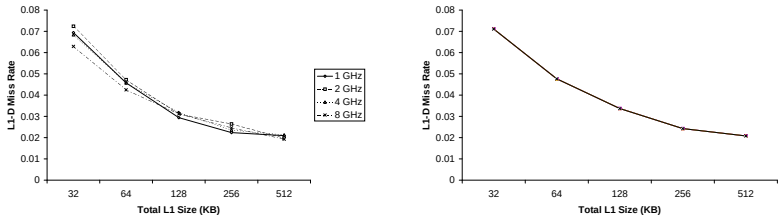
Fig. 1 shows how miss rates of the L1 instruction and data caches vary as their size increases for both RAMpage with context switches on misses and the standard hierarchy. (RAMpage without switches on misses follows the same trend as the standard hierarchy.) As cache sizes increase, the miss rate decreases, initially fairly rapidly. The trend is similar for all models.

Execution times are plotted in fig. 2, normalised to the best execution time at each CPU speed. As expected, larger caches decrease execution times by reducing capacity misses, as evident from the reduced miss rates – with limits to the benefits as L1 scales up. The best overall effect is from the combination of RAMpage with context switches on misses and increasing the size of L1. The execution time of the fastest variation speeds up 10.7 over the slowest configuration, as compared with the clock speedup of 8. Comparing a given hierarchy's slowest (1GHz, 32 KB L1) and fastest case (8GHz, 256 KB total L1) results in a speedup of 6.12 for the conventional hierarchy, 6.5 for RAMpage without switches on misses and 9.9 for switches on misses. For slowest CPU and smallest L1, RAMpage with switches on misses has a speedup of 1.08 over the conventional hierarchy, rising to 1.74 with the fastest CPU and biggest L1. For RAMpage without switches on misses, the scaling up of improvement over the conventional hierarchy is not as strong: for the slowest CPU with least aggressive L1, RAMpage has a speedup of 1.03, as opposed to 1.11 for the fastest CPU with largest L1. So, whether by comparison with a conventional architecture or by



(a) RAMpage with context switches on misses (Rsw) L1i

(b) Standard hierarchy (Std) L1i



(c) Rsw L1d

(d) Std L1d

Fig. 1. L1 miss rate *vs.* L1 size for varying issue rates. *L1d size = L1i size.*

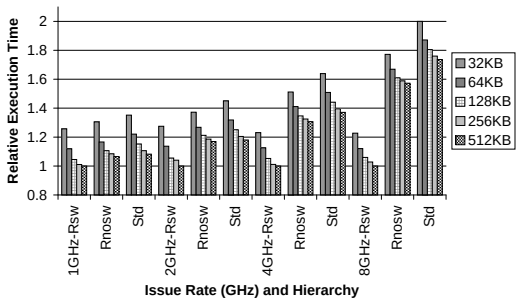
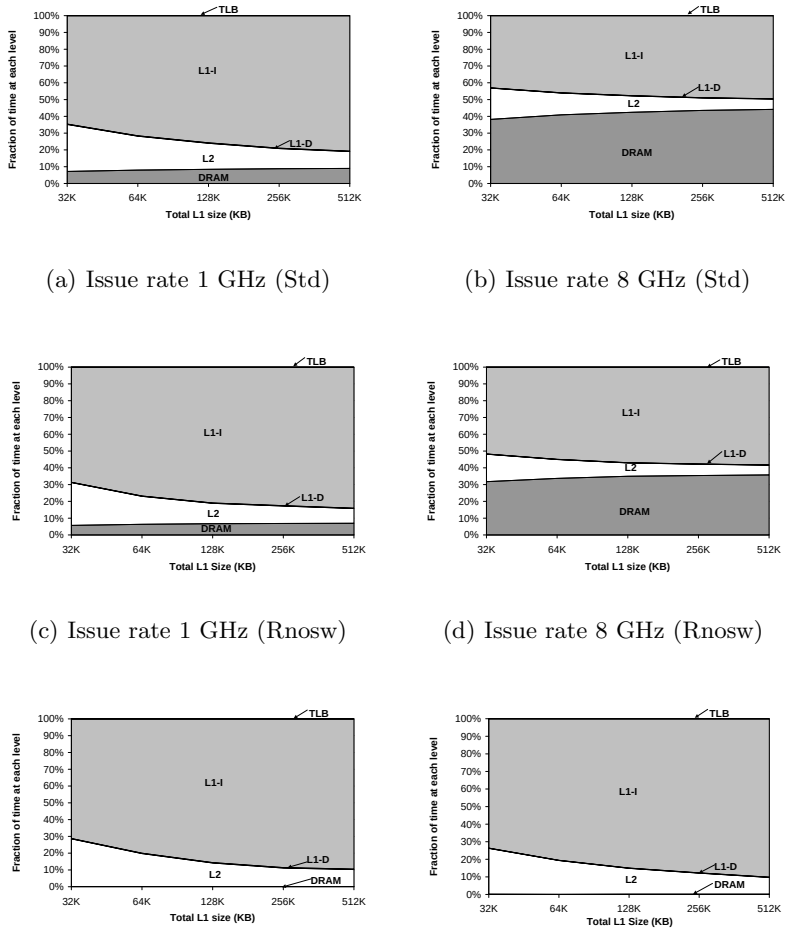


Fig. 2. Relative execution times (normalised: best at each issue rate = 1) as cache sizes vary with instruction issue rates.

comparison with a slower version of itself, RAMpage scales up well with more aggressive hardware, but more so with context switches on misses.



(a) Issue rate 1 GHz (Std)

(b) Issue rate 8 GHz (Std)

(c) Issue rate 1 GHz (Rnosw)

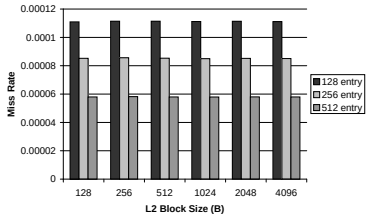
(d) Issue rate 8 GHz (Rnosw)

(e) Issue rate 1 GHz (Rsw)

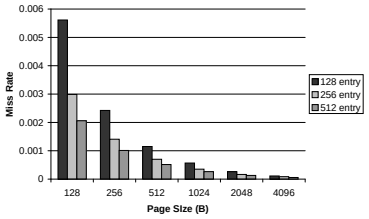
(f) Issue rate 8 GHz (Rsw)

Fig. 3. Fraction of time in each level of hierarchy.

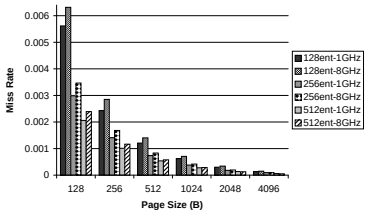
Fig. 3 shows relative times each variation of the slowest and fastest CPU spend waiting for each level of the various hierarchies, as L1 size increases. The 8 GHz issue rate for the conventional hierarchy spends over 40% of total execution time waiting for DRAM for the largest L1 cache – in line with measurements of the Pentium 4, which spends 35% of its time waiting for DRAM running SPECint2k on average at 2 GHz [28]. This Pentium 4 configuration corresponds roughly to a 6 GHz issue rate in this paper. The similarity of the time waiting for DRAM lends some credibility to our view that our results are reasonably in line with real systems.



(a) Standard Hierarchy (8 GHz)



(b) RAMpage no switches (8 GHz)



(c) RAMpage with switches on misses

Fig. 4. TLB miss rate vs L2 block/SRAM page size.

While cache size increases boost performance significantly, as CPU speed increases, a large L1 cannot save a conventional hierarchy from the high penalty of waiting for DRAM. In fig. 3(d), it can be seen that RAMpage only improves the situation marginally without context switches on misses.

With RAMpage with context switches on misses, time waiting for DRAM remains negligible as the CPU-DRAM speed gap increases by a factor of 8 (fig. 3(f)). The largest L1 (combined L1i and L1d size 512KB) results in only about 10% of execution time being spent waiting for SRAM main memory, while DRAM wait time remains negligible. By contrast, the other hierarchies, while seeing a significant reduction in time waiting for L2 (or SRAM main memory), do not see a similar reduction in time waiting for DRAM as L1 size increases.

4.2 TLB Variations

All TLB variations are measured with the L1 parameters fixed at the original RAMpage measurements – 16 KB each of instruction and data cache.

The TLB miss rate (fig. 4), even with increased TLB sizes, is significantly higher in all RAMpage cases than for the standard hierarchy, except for a 4 KB RAMpage page size. As SRAM main memory page size increases, TLB miss rates drop, as expected. Further, as TLB size increases, smaller pages' miss rates decrease. In the case of context switches on misses, the number of context

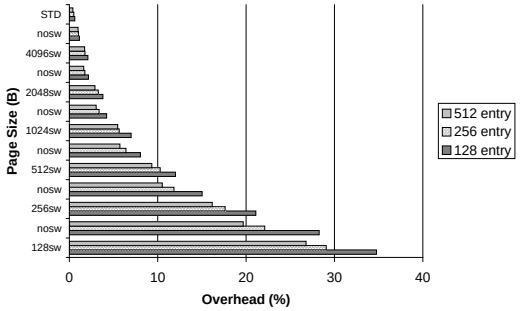
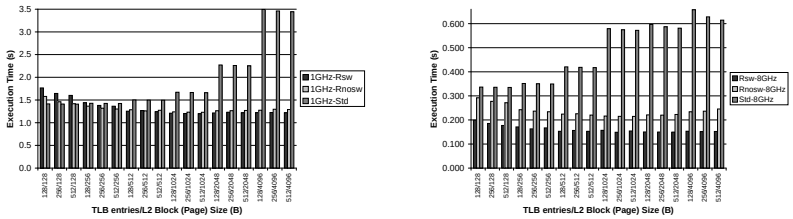


Fig. 5. TLB miss, page fault overhead (fraction of all references, 8GHz)



(a) 1 GHz issue rate (b) 8 GHz issue rate

Fig. 6. Comparison of execution times for each hierarchy with different TLB and page or L2 cache block sizes.

switches increases as the CPU-DRAM speed gap grows, since the effective time waiting for one DRAM reference grows. Consequently, the TLB miss rate is higher for a faster clock speed in this case (fig. 4(c)), whereas it does not change significantly for the other variations measured. Note also that L2 block size has little effect on TLB miss rate in the standard hierarchy (fig. 4(a)).

Fig. 5 shows how TLB miss and page fault handling overhead varies with page and TLB size for all hierarchies with an 8 GHz processor issue rate. Overhead here is measured as extra references generated, which is conservative, as the actual cost can be up to double, once memory hierarchy effects are taken into account [15]. From 1024 B pages upwards percentage differences in overhead between 256 and 512 entry TLBs are minor. Although overheads of TLB and page fault handling are still relatively high for small pages, with a 4 KB page, RAMpage without context switches on misses is within 50% of the overhead

of the standard hierarchy. RAMpage TLB misses do not result in references to DRAM, unless there is a page fault, so the additional references should not result in a similarly substantial performance hit.

Fig. 6 illustrates execution times for the hierarchies at 1 and 8 GHz, the speed gap which shows off differences most clearly. There are two competing effects: as L2 block (SRAM page size) increases, miss penalty to DRAM increases. In RAMpage, reduced TLB misses compensate for the higher DRAM miss penalty, but the performance of the standard hierarchy becomes worse as block size increases. TLB size variation makes little difference to performance of the standard hierarchy with the simulated workload. Performance of RAMpage with switches on misses does not vary much for pages of 512 B and greater even with TLB variations, while RAMpage without switches is best with 1024 B pages.

The performance-optimal TLB and page size combination for RAMpage without context switches on misses, with a 512 entry TLB, is a 1024 B page for all issue rates. In previous work, with a 64-entry TLB, the optimal page size at 1 GHz was 2048 B, while other issue rates performed best with 1024 B pages. Thus, a larger TLB results in a smaller page size being optimal for the 1 GHz speed. While other page sizes are still slower than the 1024 B page size, for all cases with pages of 512 B and greater RAMpage without context switches on misses is faster than the standard hierarchy.

For RAMpage with context switches on misses, the performance-optimal page size has shifted to 1024 B with a larger TLB. Previously the best page size was 4096 B for 1, 2 and 4 GHz and 2048 B for 8 GHz. A TLB of 256 or even 128 entries combined with the 1024 B page will yield optimum or almost optimum performance. With a 1024 B page and 256 entries, a total of 256 KB, or 6.25% of the RAMpage main memory is mapped by the TLB, which appears to be sufficient for this workload (a 4 KB page with a 512-entry TLB maps half the SRAM main memory, overkill for any workload with reasonable locality of reference). Nonetheless, TLB performance is highly dependent on application code, so results presented here need to be considered in that light.

Contrasting the 1 Ghz and 8 GHz cases in fig. 6 makes it clear again how the differences between RAMpage and a conventional hierarchy scale as the CPU-DRAM speed gap increases. At 1 GHz, all variations are reasonably comparable across a range of parameters. At 8 GHz, RAMpage is clearly better in all variations, but even more so with context switches on misses. A larger TLB broadens the range of useful RAMpage configurations, without significantly altering the standard hierarchy's competitiveness.

4.3 Summary

In summary, the RAMpage model with context switches on misses gains most from L1 cache improvements, though the other hierarchies also reduce execution time. However, without taking context switches on misses, increasing the size of L1 has the effect of increasing the fraction of time spent waiting for DRAM, since the number of DRAM references is not reduced, nor is their latency hidden. As was shown by scaling up the CPU-DRAM speed gap, only RAMpage with

context switches on misses, of the variations presented here, is able to hide the increasing effective latency of DRAM. Increasing the size of the TLB, as predicted, increased the range of SRAM main memory page sizes over which RAMpage is viable, widening the range of choices for a designer.

5 Conclusion

This paper has examined enhancements to RAMpage, which measure its potential for further improvement, as opposed to similar improvements to a conventional hierarchy. As in previous work, RAMpage has been shown to scale better as the CPU-DRAM speed gap grows. In addition, it has been shown that context switches on misses can take advantage of a more aggressive core including a bigger L1 cache, and a bigger TLB. The remainder of this section summarizes results, outlines future work and sums up overall findings.

5.1 Summary of Results

Introducing significantly larger L1 caches – even if this could be done without problems with meeting clock cycle targets – has limited benefits. Scaling the clock speed up by a factor of 8 achieves only about 77% of this speedup in a conventional hierarchy measured here. RAMpage with context switches on misses is able to make effective use of a larger L1 cache, and achieves superlinear speedup with respect to a slower clock speed and smaller L1 cache. While this effect can only be expected in RAMpage with an unrealistically large L1, this result shows that increasingly aggressive L1 caches are not as important a solution to the memory wall problem as finding alternative work on a miss to DRAM.

That results for RAMpage without context switches on misses are an improvement but not as significant as results with context switches on misses suggests that attempts at improving associativity and replacement strategy will not be sufficient to bridge the growing CPU-DRAM speed gap.

Larger TLBs, as expected, increase the range of useful RAMpage SRAM main memory page sizes, though the performance benefit on the workload measured was not significant versus larger page sizes and a more modest-sized TLB.

5.2 Future Work

It would be interesting to match RAMpage with models for supporting more than one instruction stream. SMT, while adding hardware complexity, is an established approach [19], with existing implementations [3]. Another thing to explore is alternative interconnect architectures, so multiple requests for DRAM could be overlapped [24]. HyperTransport [2] is a candidate. A more detailed simulation modelling operating system effects accurately would be useful. SimOS [26], for example, could be used. Further variations to explore include virtually-addressed L1 and hardware TLB miss handling. Finally, it would be interesting to build a RAMpage machine.

5.3 Overall Conclusion

RAMpage has been simulated in a variety of forms. In this latest study, enhancing L1 and the TLB have shown that it gains significantly more from such improvements than a conventional architecture in some cases. The most important finding generally from RAMpage work is that finding other work on a miss to DRAM is becoming increasingly viable. While RAMpage is not the only approach to finding such alternative work, it is a potential solution. As compared with hardware multithreading approaches, its main advantage is the flexibility of a software solution, though this needs to be compared to hardware solutions to establish the performance cost of extra flexibility.

References

1. Thomas Alexander and Gershon Kedem. Distributed prefetch-buffer/cache design for high-performance memory systems. In *Proc. 2nd IEEE Symp. on High-Performance Computer Architecture*, pages 254–263, San Jose, CA, February 1996.
2. AMD. HyperTransport technology: Simplifying system design [online]. October 2002. `http://www.hypertransport.org/docs/26635A_HT_System_Design.pdf`.
3. J. M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla, and S. R. Kunkel. A multithreaded PowerPC processor for commercial servers. *IBM J. Research and Development*, 44(6):885–898, November 2000.
4. 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.
5. 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.
6. 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.
7. 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.
8. R. Crisp. Direct Rambus technology: The new main memory standard. *IEEE Micro*, 17(6):18–28, November/December 1997.
9. B. Davis, T. Mudge, B. Jacob, and V. Cuppu. DDR2 and low latency variants. In *Solving the Memory Wall Problem Workshop*, Vancouver, Canada, June 2000. In conjunction with 26th Annual lnt. Symp. on Computer Architecture.
10. Erik G. Hallnor and Steven K. Reinhardt. A fully associative software-managed cache design. In *Proc. 27th Annual Int. Symp. on Computer Architecture*, pages 107–116, Vancouver, BC, 2000.
11. J. Handy. *The Cache Memory Book*. Academic Press, San Diego, CA, 2 ed., 1998.
12. J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kauffmann, San Francisco, CA, 2 ed., 1996.
13. 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.

14. B. Jacob and T. Mudge. Software-managed address translation. In *Proc. Third Int. Symp. on High-Performance Computer Architecture*, pages 156–167, San Antonio, TX, February 1997.
15. Bruce L. Jacob and Trevor N. Mudge. A look at several memory management units, TLB-refill mechanisms, and page table organizations. In *Proc. 8th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, pages 295–306, San Jose, CA, 1998.
16. E.E. Johnson. Graffiti on the memory wall. *Computer Architecture News*, 23(4):7–8, September 1995.
17. Norman P. Jouppi. Cache write policies and performance. In *Proc. 20th annual Int. Symp. on Computer Architecture*, pages 191–201, San Diego, CA, 1993.
18. Jang-Soo Lee, Won-Kee Hong, and Shin-Dug Kim. Design and evaluation of a selective compressed memory system. In *Proc. IEEE Int. Conf. on Computer Design*, pages 184–191, Austin, TX, 10–13 October 1999.
19. 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 Trans. on Computer Systems*, 15(3):322–354, August 1997.
20. P. Machanick. The case for SRAM main memory. *Computer Architecture News*, 24(5):23–30, December 1996.
21. P. Machanick. Scalability of the RAMpage memory hierarchy. *South African Computer Journal*, (25):68–73, August 2000.
22. P. Machanick, P. Salverda, and L. Pompe. Hardware-software trade-offs in a Direct Rambus implementation of the RAMpage memory hierarchy. In *Proc. 8th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, pages 105–114, San Jose, CA, October 1998.
23. Philip Machanick. *An Object-Oriented Library for Shared-Memory Parallel Simulations*. PhD Thesis, Dept. of Computer Science, University of Cape Town, 1996.
24. Philip Machanick. What if DRAM is a slow peripheral? *Computer Architecture News*, 30(6):16–19 December 2002.
25. 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.
26. 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.
27. Ashley Saulsbury, Fong Pong, and Andreas Nowatzyk. Missing the memory wall: the case for processor/memory integration. In *Proc. 23rd annual Int. Symp. on Computer architecture*, pages 90–101, Philadelphia, PA, 1996.
28. Eric Sprangle and Doug Carmean. Increasing processor performance by implementing deeper pipelines. In *Proc. 29th Annual Int. Symp. on Computer architecture*, pages 25–34, Anchorage, Alaska, 2002.
29. J. M. Tendler, J. S. Dodson, Jr. J. S. Fields, H. Le, and B. Sinharoy. POWER4 system microarchitecture. *IBM J. Research and Development*, 46(1):5–25, 2002.
30. W.A. Wulf and S.A. McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(1):20–24, March 1995.

Acknowledgements

Financial support for this work has been received from Universities of Queensland and Witwatersrand, and South African National Research Foundation. We would like to thank the referees for helpful suggestions.