L1 Cache and TLB Enhancements to the RAMpage Memory Hierarchy

Philip Machanick
School of ITEE, University of Queensland
Brisbane, Qld 4068, Australia
philip@itee.uq.edu.au

Zunaid Patel
School of Computer Science, University of the Witwatersrand
Private Bag 3, Wits, 2050, 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, and uses the TLB to cache page translations in that main memory. Earlier RAMpage evaluation used a relatively small L1 cache and TLB. Given that TLB misses can take up a significant fraction of run time, better TLB management in general is worth pursuing. For the RAMpage hierarchy, the effect is clearer than with a conventional hierarchy, because it is more feasible to make a TLB which maps a high fraction of main memory pages. This paper illustrates how more aggressive components higher in the memory hierarchy make time spent waiting for DRAM more significant as a fraction of total execution time, and, hence, approaches to hide the latency of DRAM become more important. For an instruction issue rate of 1 GHz, the simulated standard hierarchy waited for DRAM 10% of the time; with the instruction issue rate increased to 8 GHz, the fraction of time spent waiting for DRAM increased to 40%, and was higher for a larger L1 cache. The RAMpage hierarchy with context switches on misses was able to hide almost all DRAM latency. Increasing the processor speed in a standard hierarchy by a factor of 8 and increasing L1 cache size by a factor of 16, with DRAM speed unchanged, resulted in a speedup of 6.12. Adding the RAMpage model and introducing context switches on misses, with similar processor speed and L1 improvements, resulted in a speedup of 10.7 over the slowest conventional hierarchy. A larger TLB was shown to increase the viable range of SRAM page sizes in the RAMpage hierarchy.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*memory hierarchy, memory wall, cache memories, virtual memory, translation lookaside buffer, TLB*; B.3.3 [**Memory Structures**]: Performance Analysis and Design Aids—*simulation*

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 [25] and that it scales better as the CPU-DRAM speed gap grows, particularly by virtue of being able to take context switches on misses [24].

One of the potential gains of the RAMpage hierarchy, which has been mentioned in previous work but not fully explored, is for the TLB to be less of a bottleneck in the RAMpage model than in a conventional physically-addressed cache. Further, it was hypothesized that the RAMpage model would be more competitive across a wider range of SRAM page sizes (equivalent to the line size of the lowest-level cache in a conventional hierarchy) with a more aggressive TLB design. Secondly, it was hypothesized that a more aggressive L1 cache would emphasize differences in lower levels of the hierarchy – particularly that DRAM latency would become a more significant factor.

In some studies, handling TLB misses has accounted for as much as 40% of run time [15], with figures in the region of 20–30% not uncommon [8, 26]. The RAMpage model 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 chosen 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 de 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. While it may be accepted that quadrupling the size of a cache halves the number of misses [31], such cache expansion may not always be practical, and increasing the size of caches in any case makes it harder to scale up their speed [13].

The approach taken in this paper is to compare the RAMpage model with a conventional 2-level cache hierarchy as the size of the TLB is scaled up, across a number of different SRAM main memory page sizes, as well as with a variety of L1 cache sizes.

The simulated second-level cache of 4Mbytes runs at a third of the peak issue rate, which is unrealistically fast, given that some current commodity designs are up to 9-way superscalar [2]. The intent

Technical Report submitted for potential publication 18 November 2002: look for published version in 2003.

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, an unrealistically 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. Further, since the SRAM page size is the unit mapped by the TLB, it becomes easier to find an optimum point in the design space for the RAMpage model. Finally, the possibility of taking context switches on misses makes relatively large SRAM page sizes viable, which means that a smaller TLB can achieve reasonable results in the RAMpage model than is the case for a conventional cache architecture.

Cache measurements show that as L1 cache 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

2.1 Introduction

The RAMpage model was proposed [23] in response to talk of the memory wall [33, 18]. 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 controller 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.

2.2 Memory Wall

The memory wall is the situation where the effect of CPU improvements start to become 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 [14].

If predictions of the memory wall situation are correct [33], DRAM latency in future will be a serious limiting factor in performance improvement. Clearly, attempts at working around the memory wall are becoming increasingly common, including workshops at major architecture conferences [11]. But the fundamental underlying DRAM and CPU latency trends continue [30].

2.3 The RAMpage Approach

The RAMpage model is based on the notion that DRAM, while still orders of magnitude faster than disk, is increasingly starting to display the attributes of a peripheral – that there is sufficient time to do other work while waiting for it [27] – particularly if relatively large units are moved between DRAM and the lowest SRAM level, a trend encouraged by the general trend towards higher-bandwidth interconnects [10, 3] and larger cache block sizes (e.g., 512 bytes on the Power4 L3 cache [32]).

The RAMpage hierarchy makes as few changes from a traditional hierarchy as possible. The lowest-level cache is managed as the main memory (i.e., as a paged virtually-addressed memory), with disk used as a secondary paging device. The RAMpage main memory page table is inverted, to minimize its size. Further, 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 [25].

The RAMpage model has the following advantages:

- *fast hits* – provided there is not TLB miss, a hit is a simple matter of physically addressing an SRAM memory
- *full associativity* – hardware full associativity has a penalty in slower hits, but achieving full associativity by implementing a paged memory avoids that problem
- *software managed replacement* – page replacement can be as sophisticated as needed, drawing on operating system practice
- *TLB missess to DRAM minimized* – as explained above
- *pinning in SRAM* – critical operating system data and code can be pinned in SRAM, which is hard to do without full associativity
- *hardware simplicity* – the complexity of a cache controller is removed from the design of the lowest level of SRAM
- *context switches on misses to DRAM* – the processor 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 (e.g., in the form of an intelligent memory controller which made page faults to DRAM look like disk references, or the software may be hidden in architecture extensions such as PALcode in the Alpha processor [4]).

The RAMpage approach has in the past been shown to scale well in the face of the grown CPU-DRAM speed gap, particularly when context switches are taken on misses. The effect of taking context switches on misses is that, provided their is work available for the CPU, waiting for DRAM can effectively be eliminated [24]. While the other advantages of RAMpage may result in a small performance win, context switches on misses have the most significant effect, as time which would otherwise be spent waiting for DRAM increases.

2.4 Alternatives

Alternative 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 [7, 20] or with compiler support [28]; 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 [13]. The Power4 has a variation of this, in which large cache blocks are divided into sectors, and the critical sector is fetched first [32]. Memory compression in effect reduces latency because a smaller amount of information must be moved on a miss. Of course, the compression and decompression overhead should be less than the time saved on memory transfers [21]. There are many variations on write strategy when the write causes a miss, but the most effective generally include write buffering, because completion of a write is not necessary to continue with other operations [19]. A non-blocking cache (or lockup-free) cache can allow an aggressive pipeline to continue processing other instructions while waiting for a miss [6].

SMT is aimed at masking not only DRAM latency, but also other causes of pipeline stalls, by having hardware support for more than one active thread [22].

These ideas come with various costs, for example, prefetching can displace needed cache content, causing unnecessary misses. However, the biggest problem is that most of these approaches do not scale with the growing CPU-DRAM speed gap. Critical word first is not as helpful as the latency for the first reference grows in relation to the total time for a big DRAM transaction. Prefetch, memory compression and nonblocking caches have limits as to the extent to which they can reduce effective latency because of limits to predictability of future behaviour. Write buffering can scale provided buffer size can be scaled, and references to buffered writes can be handled before they are written back. SMT has the potential to mask much of the time spent waiting for DRAM, but at the cost of more complex hardware in the CPU.

Reducing misses has generally been addressed by increasing cache size, associativity, or both. Given the limits on how large a cache can be made at a given speed, the number of cache levels has also increased over time.

Given the extra overheads on hits introduced by increasing associativity, there have been various attempts at supporting higher associativity in hardware by alternative means.

Full associativity can be achieved in hardware without the overheads for hits associated with a conventional fully-associative cache, in an indirect index cache (IIC), by what amounts to a hardware implementation of the page table lookup aspect of the RAMpage model. An inverted page table is in effect implemented in hardware, to allow a block to be placed anywhere in a memory organized as a direct-mapped cache [12]. The drawback of this approach is that all references incur some overhead of an extra level of indirection. The advantage is that the operating system need not be invoked to handle the equivalent of a TLB miss in the RAMpage model.

In the 1980s, there was some work on software-based cache management, with emphasis on managing cache coherence in a shared-memory system [9]. More recent work on managing the interface between cache and DRAM in software has focused on address translation [16]. These approaches, however, were not aimed specifically at managing replacement.

2.5 Summary

The memory wall problem needs to be addressed by minimizing time spent waiting for DRAM. The RAMpage approach allows time which would otherwise be spent waiting for DRAM to be masked by taking context switches on misses. Other approaches which have been proposed generally either do not aim to mask the time spent waiting for DRAM, but to reduce it, or require more complex hardware.

RAMpage can, however, 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

3.1 Introduction

This section outlines the approach to the reported experiments. Results are designed to be comparable to previously reported results as far as possible.

The simulation strategy is explained, followed by some detail of simulation parameters; in conclusion, expected findings are discussed.

3.2 Simulation Strategy

The approach followed here is similar to that used in previously reported work. A range of variations on a standard 2-level hierarchy is compared to similar variations on a RAMpage hierarchy, with and without context switches on misses.

Simulations are trace-driven, and do not model the pipeline. It is assumed that pipeline timing is less significant than variations in DRAM referencing behaviour. Processor speed is given in GHz, representing a peak instruction issue rate, rather than a clock speed.

It could be argued that over-simplification of 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.

3.3 Simulation Parameters

The parameters used are similar to previous published work to make results comparable. However, the range of instruction issue rates is higher than in earliest work [25], to take into account advances in CPU speed.

The following parameters are unchanged from previous simulations, and are common across RAMpage and the conventional hierarchy. This represents the baseline before 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 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 for simplicity, but infinite DRAM modeled with no misses to disk
- TLB and L1 data hits are fully pipelined: they do not add to execution time; only instruction fetch bits add to simulated run time; time for replacements or maintaining inclusion are costed as L1d or TLB "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.

3.3.1 specific to conventional hierarchy

The "conventional" system has a 2-way associative 4Mbyte L2 cache.

The bus connecting the L2 cache to the CPU is 128 bits wide and runs at one third of the CPU issue rate 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 [14], 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.

3.3.2 specific to RAMpage hierarchy

In the 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 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.

The RAMpage SRAM main memory uses an inverted page table, and TLB misses will not incur a DRAM reference, as long as the required reference can be found in an SRAM level.

3.3.3 inputs and variations

Traces used are from the Tracebase trace archive at New Mexico State University[1]. Although these traces are from the obsolete SPEC92 benchmarks, they are sufficient to warm up the size of cache used here, because 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 RAMpage size of 16 KB to 32 KB, 64 KB, 128 KB and 256 KB (i.e., the total of L1 cache varied from 32 KB to 512 KB). Further, to explore more of the design space, L1 block size was measured at sizes of 32, 64 and 128 bytes.

To measure the effect of increasing the TLB size we varied it from 64 entries (the original size) to 128, 256 and 512 entries. An even larger TLB could be modelled (e.g., the Power4 has a 1024-entry TLB [32]), but this range is sufficient to capture variations of interest.

[1]The traces used in this paper can be found at `ftp://tracebase.nmsu.edu/pub/traces/uni/r2000/utilities/` and `ftp://tracebase.nmsu.edu/pub/traces/uni/r2000/SPEC92/`.

3.4 Expected Findings

As the L1 cache becomes larger, RAMpage without context switches on misses should see less of a gain, since some of its gains result from more efficient management of DRAM at the expense of slower software intervention. While improving L1 should not effect 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; very large TLB sizes should have little effect on larger SRAM page sizes in the RAMpage model because larger pages ensure that a relatively large fraction of the SRAM main memory is mapped, even without a very large TLB. To quantify this, if the TLB has 64 entries and the page size is 4 KB with a 4 MB SRAM 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, not SRAM pages. In our simulations, DRAM pages are fixed at 4 KB. We would accordingly expect to see some improvement in a conventional architecture with a larger TLB, but not as much as with RAMpage (especially with small SRAM main memory page sizes).

4. RESULTS

4.1 Introduction

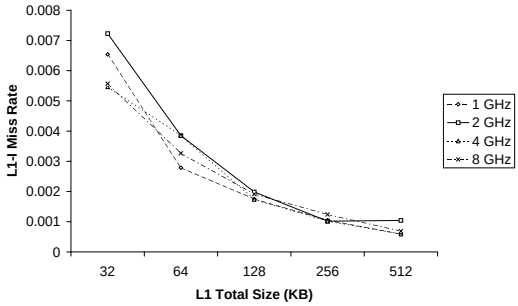
This section presents results of simulations, with some discussion of their significance. The main focus here is on differences introduced by the changes over previous simulations, but some advantages of RAMpage, as previously described, should be evident again from these new results.

Presentation of results is broken down into the effects of increasing L1 cache size, and the effects of increasing the TLB size, since these two improvements have very different effects on the hierarchies modelled.

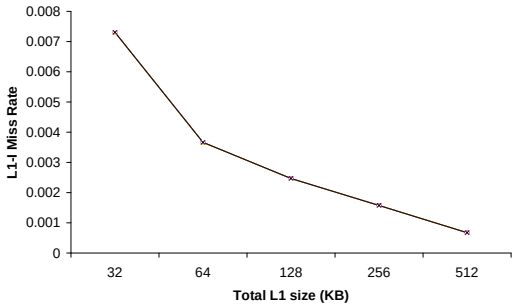
In all cases, results are presented for all three cases: the conventional 2-level cache with a DRAM main memory, and RAMpage with and without context switches on misses.

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

4.2 Increasing L1 Size

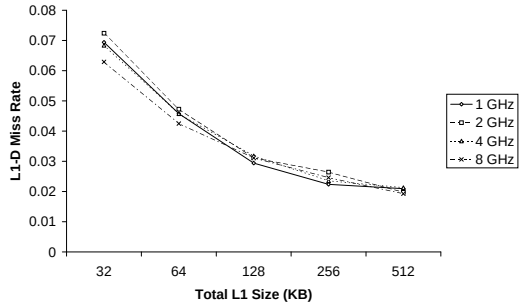


(a) RAMpage with context switches on misses

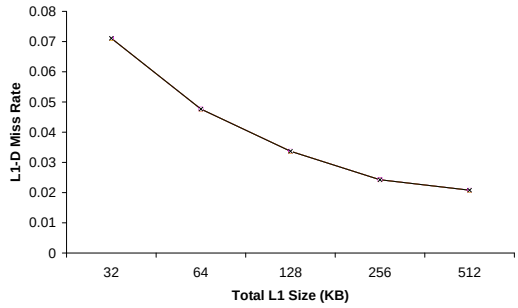


(b) Standard hierarchy

Figure 1: L1i miss rate vs L1 size for differing issue rates. ***L1i size is half the total L1 size.***



(a) RAMpage with context switches on missees



(b) Standard hierarchy

Figure 2: L1d miss rate vs L1 size for differing issue rates. ***L1d size is half the total L1 size.***

Figures 1 and 2 show how the miss rates of the first level instruction and data caches vary as their size is progressively increased for both RAMpage with context switches on misses and the standard hierarchy. (Data for RAMpage without context switches on misses is not shown since it follows the same trend as the standard hierarchy.) These graphs show that as cache sizes increase, the miss rate decreases, initially fairly rapidly. The trend is similar for all models. There is however some variation for different CPU speeds with context switches on misses, because increasing the CPU-DRAM speed gap increases the probability of more than one miss to DRAM being outstanding at once, and hence increases the rate of context switching.

Table 1 shows the simulated execution times obtained for each of the cache sizes examined within each hierarchy. As expected, larger caches decrease execution times due to the reduction in capacity misses, as evident from the reduced miss rates. However, incremental improvements in performance decrease as cache sizes increase, a trend which was also observed for miss rates. For instance, doubling the total L1 size from 32 to 64 KB at the 1 GHz issue rate results in a speedup of about 1.12 for all three hierarchies. In contrast, between the 256 and 512 KB sizes the speedup obtained is about 1.02.

The best overall effect is from the combination of introducing RAMpage with context switches on misses and increasing the size of L1.

Total L1 Size (KB)	**1 GHz**	**2 GHz**	**4 GHz**	**8 GHz**
32	1.322	0.709	0.401	0.248
	1.276	0.670	0.370	0.220
	1.229	0.623	0.302	0.152
64	1.193	0.644	0.370	0.232
	1.141	0.619	0.346	0.207
	1.094	0.556	0.276	0.139
128	1.127	0.611	0.353	0.224
	1.082	0.592	0.330	0.200
	1.022	0.516	0.258	0.131
256	1.082	0.588	0.342	0.218
	1.061	0.580	0.325	0.197
	0.988	0.508	0.248	0.128
512	1.058	0.577	0.336	0.216
	1.042	0.571	0.320	0.195
	0.978	0.489	0.245	0.124

Table 1: L1 cache size (total of instruction and data caches) ***vs.*** **issue rate.** ***Each row shows simulated execution times (s) for standard hierarchy (top), RAMpage no context switches on misses (middle) and RAMpage with context switches on misses (bottom).***

The execution time of 0.124s of the fastest variation represents a speedup of 10.7 over the slowest configuration, as compared with the clock speedup of 8. Only increasing the L1 cache while multiplying clock speed by 8 results in a speedup of 6.12, for the conventional hierarchy, as modelled here. Comparing like with like, RAMpage without context switches on misses has a speedup of 6.5 when the clock speed is sped up by 8 and L1 is increased from 32 KB to 256 KB in total. With context switches on misses, the speedup over its own slowest case is 9.9. So, whether by comparison with a conventional architecture or by comparison with a slower version of itself, RAMpage scales up well with more aggressive hardware, especially when context switches are taken on misses.

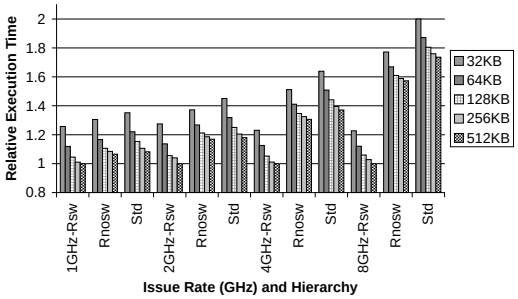


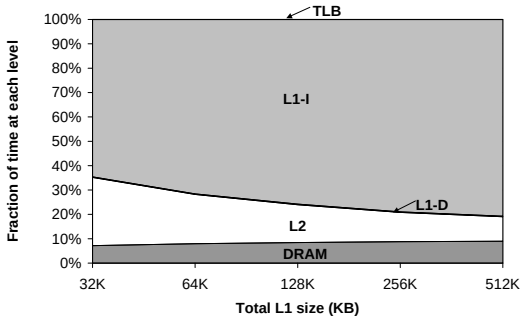
Figure 3: Relative execution times as cache sizes vary with instruction issue rates. ***Execution times normalised: best at each issue rate = 1. ("Std": standard hierarchy; RAMpage with and without context switches on misses: "Rsw", "Rnosw").***

Execution times from Table 1 are plotted in Figure 3, normalised to the best execution time obtained at each CPU speed. Although larger caches may make it harder to scale down CPU cycle time, the need to avoid off-chip latencies has led to designs with relatively large L1 caches, for example, the total of 128 KB of L1 cache on the AMD Athlon CPU [2].

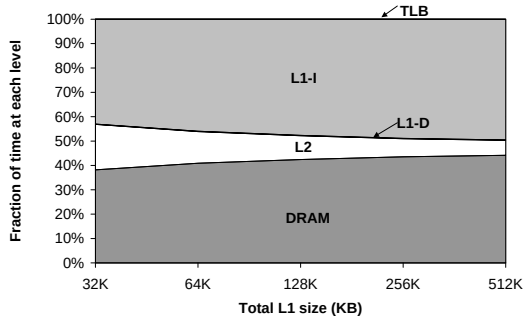
Figure 4 shows the relative times each variation for the slowest and fastest CPU modelled spend waiting for each level of the standard hierarchy, as L1 cache size increases. Since TLB and L1 data references are pipelined, they do not show up as a significant amount of time. The 8 GHz issue rate for the conventional hierarchy spends over 40% of total execution time waiting for DRAM for the largest L1 cache modelled, which is in line with measurements of the Pentium 4, which spends 35% of its time waiting for DRAM running SPECint2k on average at 2 GHz [31]. As simulated, this configuration of the Pentium 4 corresponds roughly to a 6 GHz issue rate in this paper. The similarity of the measure of time waiting for DRAM lends some credibility to our view that our results are reasonably in line with real systems.

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 Figure 5, it can be seen that RAMpage only improves the situation marginally without context switches on misses.



(a) Issue rate 1 GHz



(b) Issue rate 8 GHz

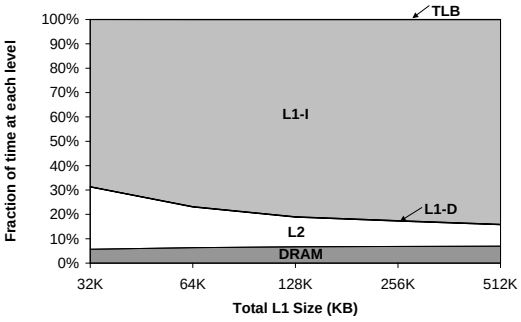
Figure 4: Fraction of time spent in each level of the standard hierarchy. ***Figures in GHz are peak instruction throughput rate. TLB and L1 data (L1-D) hits are fully pipelined, so they only account for a small fraction of total execution time; TLB misses are accounted for in other memory traffic. In all cases L2 (or RAMpage main memory) is 4 Mbytes.***

With context switches on misses, however, the amount of time spent waiting for DRAM in the RAMpage model remains negligible even as the CPU-DRAM speed gap increases by a factor of 8, as illustrated in Figure 6. Figure 6 makes it clear why improving L1 makes it possible for the RAMpage hierarchy to achieve superlinear speedup if L1 is increased significantly as clock speed increases. The largest L1 cache (combined size 512KB, or 256KB each of instruction and data cache) results in only about 10% of execution time being spent waiting for the SRAM main memory, while DRAM wait time remains negligible. By contrast, the other hierarchies simulated, while seeing a significant reduction in time spent waiting for L2 cache or the SRAM main memory, do not see a similar reduction in time spent waiting for DRAM as L1 becomes larger.

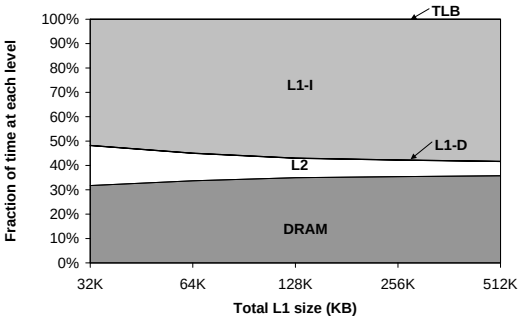
4.3 TLB Variations

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

Figure 7 illustrates TLB miss rate as TLB size increases. The miss

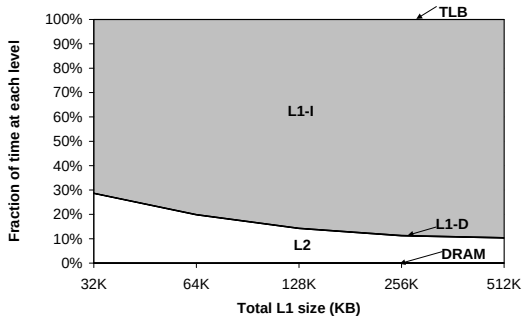


(a) Issue rate 1 GHz

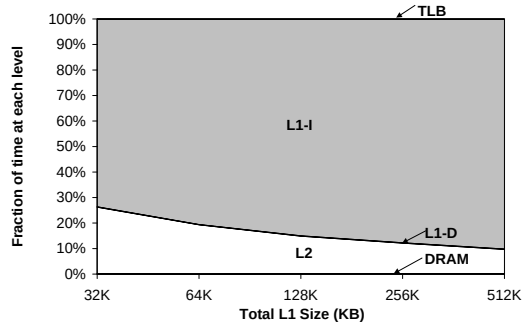


(b) Issue rate 8 GHz

Figure 5: Fraction of time spent in each level of the RAMpage hierarchy (no context switches on misses). ***Interpretation of the graphs is as in Figure 4.***



(a) Issue rate 1 GHz



(b) Issue rate 8 GHz

Figure 6: Fraction of time spent in each level of the RAMpage hierarchy (context switches on misses). ***Interpretation of the graphs is as in Figure 4.***

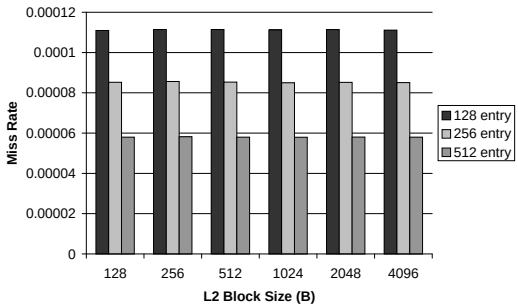
rate is significantly higher in all RAMpage cases than for the standard hierarchy, except for a 4 KB RAMpage page size. As SRAM main memory page size increases, TLB miss rates drop, as expected. Further, as TLB size increases, smaller pages' miss rates decrease. In the case of context switches on misses, the number of context 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 (see Figure 7(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 (Figure 7(a)).

Figure 8 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 purely as extra references generated, which is conservative, as the actual cost can be up to double, once memory hierarchy effects are taken into account [17]. Corresponding with the miss rate measures in the previous section, reductions in overhead diminish as the page and TLB sizes of each hierarchy are increased. In fact, from 1024 B pages upwards percentage differences in overhead between 256 and 512 entry TLBs are minor.

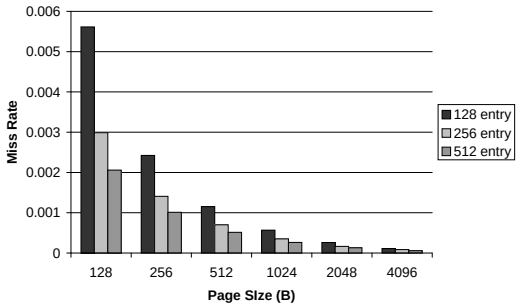
Although the overhead due to TLB and page fault handling are still relatively high for small pages, with a 4 KB page RAMpage without context switches on misses is within 50% of the overhead incurred by the standard hierarchy. RAMpage TLB misses do not result in references to DRAM, unless there is a page fault, so, although the additional number of references seems high, these should not result in a substantial performance hit.

Figure 9 illustrates execution times for the hierarchies at 1 and 8 GHz, the speed gap which shows off the differences most clearly. There are two competing effects: as block (or SRAM page size) increases, the miss penalty to DRAM increases. In the RAMpage hierarchy, reduced TLB misses compensate for the higher DRAM miss penalty, which can be seen as a speed improvement as page size increases in the case of no context switches on misses. The performance of the standard hierarchy becomes worse as the block size increases due to the higher miss penalty. The TLB size variation makes little difference to the 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.

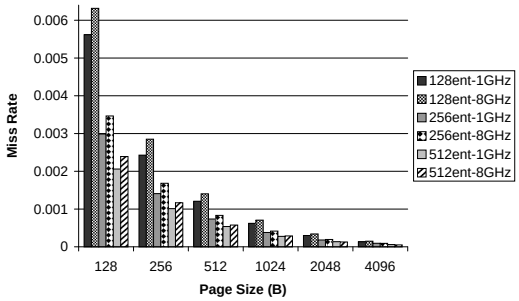
The performance-optimal TLB and page size combination for RAM-



(a) Standard Hierarchy (8 GHz)



(b) RAMpage no switches on misses (8 GHz)



(c) RAMpage with switches on misses

Figure 7: TLB miss rate vs L2 block/SRAM page size. ***Only RAMpage with context switches has significant variation with CPU speed.***

page 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.

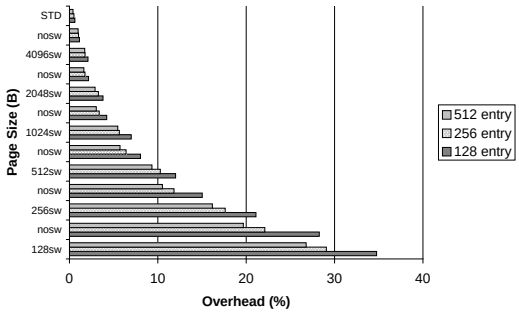
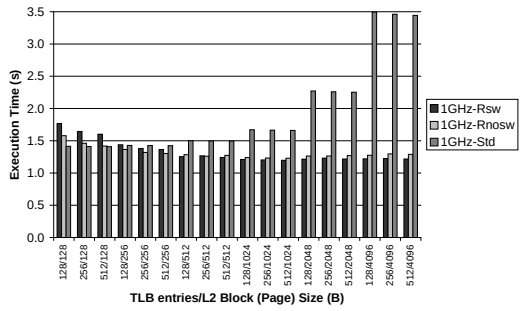
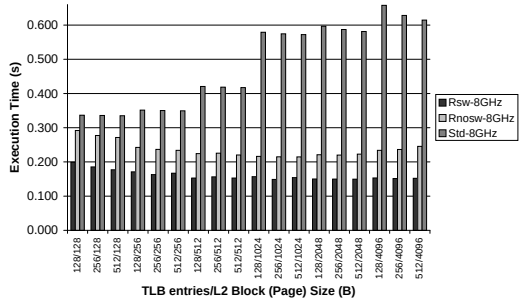


Figure 8: TLB miss and page fault handling overhead (fraction of all references) for each hierarchy with varying TLB and page sizes. ***Processor issue rate is fixed at 8 GHz.***



(a) 1 GHz issue rate



(b) 8 GHz issue rate

Figure 9: Comparison of execution times for each hierarchy with different TLB and page or L2 cache block sizes. ***Best execution time is RAMpage with context switches on misses: 256 entry TLB, 1024 B page.***

For RAMpage with context switches on misses, the performance-optimal page size has shifted to 1024 B with a larger TLB. Previ-

ously 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. Nonetheless, TLB performance is highly dependent on application code, so results presented here need to be considered in that light.

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).

Contrasting the 1 Ghz and 8 GHz cases in Figure 9 makes it clear again how the differences between RAMpage and a conventional hierarchy scale up as the CPU-DRAM speed gap increases. At 1 GHz, all variations are reasonable 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. Increasing the size of the TLB broadens the range of useful RAMpage configurations, without significantly altering the standard hierarchy's competitiveness.

4.4 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

5.1 Introduction

This paper has examined further enhancements on the RAMpage memory hierarchy, which measure its potential for further improvement, as opposed to similar improvements to a conventional hierarchy. As in previous work, the RAMpage model has been shown to scale better as the CPU-DRAM speed gap grows but more specifically, that RAMpage with 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.

5.2 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, as measured here. On the other hand, a RAMpage hierarchy 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 the case of a RAMpage machine with an unrealistically large L1 cache, 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.

5.3 Future Work

Given that the CPU-DRAM speed gap has grown to the point where finding alternative work on a miss to DRAM is looking increasingly useful, it will be interesting to match RAMpage with models for supporting more than one instruction stream without operating system changes, such as simultaneous multithreading. SMT, while adding hardware complexity, is an established approach [22], and there are existing implementations [5].

It would also be interesting to explore alternative interconnect architectures, so that data could be streamed to more than one outstanding request on a miss to DRAM [27]. HyperTransport is a fast point-to-point interconnect infrastructure [3], which could be adapted to such a purpose.

A more detailed simulation capable of modelling operating system effects accurately would be useful. SimOS [29], for example, could be adapted to this purpose.

Finally, it would be interesting to build a RAMpage machine to validate performance claims in more detail. In principle, since the number of hardware changes is relatively small, this should not be difficult.

5.4 Overall Conclusion

The RAMpage architecture has been simulated in a variety of forms. In this latest study, enhancing the L1 cache and TLB have shown that it gains as much and in some cases significantly more from such improvements than a conventional architecture.

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 represents a potentially viable solution.

6. 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 (HPCA)*, pages 254–263, San Jose, CA, February 1996.

[2] AMD. AMD Athlon processor model 4 data sheet [online]. November 2001. Available from World Wide Web: `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/23792.pdf`.

[3] AMD. HyperTransport technology: Simplifying system design [online]. October 2002. Available from World Wide

Web: `http://www.hypertransport.org/docs/26635A_HT_System_Design.pdf`.

[4] Gorden Bell and W. D. Strecker. Retrospective: what have we learned from the PDP-11–what we have learned from VAX and Alpha. In *25 years of the international symposia on Computer Architecture (selected papers)*, pages 6–10, New York, NY, 1998.

[5] J. M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla, and S. R. Kunkel. A multithreaded PowerPC processor for commercial servers. *IBM Journal of Research and Development*, 44(6):885–898, November 2000. `http://www.research.ibm.com/journal/rd/446/borkenhagen.pdf`.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] R. Crisp. Direct Rambus technology: The new main memory standard. *IEEE Micro*, 17(6):18–28, November/December 1997.

[11] 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.

[12] 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.

[13] J. Handy. *The Cache Memory Book*. Academic Press, San Diego, CA, 2nd edition, 1998.

[14] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kauffmann, San Francisco, CA, 2nd edition, 1996.

[15] 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.

[16] 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.

[17] 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.

[18] E.E. Johnson. Graffiti on the memory wall. *Computer Architecture News*, 23(4):7–8, September 1995.

[19] Norman P. Jouppi. Cache write policies and performance. In *Proc. 20th annual Int. Symp. on Computer Architecture*, pages 191–201, San Diego, California, United States, 1993.

[20] D. Kroft. Lockup-free instruction fetch/prefetch cache organisation. In *Proc. 8th Int. Symp. on Computer Architecture (ISCA '81)*, pages 81–84, May 1981.

[21] 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.

[22] 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.

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

[24] P. Machanick. Scalability of the RAMpage memory hierarchy. *South African Computer Journal*, (25):68–73, August 2000.

[25] 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.

[26] Philip Machanick. *An Object-Oriented Library for Shared-Memory Parallel Simulations*. PhD Thesis, Department of Computer Science, University of Cape Town, October 1996.

[27] Philip Machanick. What if DRAM is a slow peripheral? *Computer Architecture News*, page in press, December 2002.

[28] 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.

[29] 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.

[30] 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, Pennsylvania, United States, 1996.

[31] Eric Sprangle and Doug Carmean. Increasing processor performance by implementing deeper pipelines. In *Proceedings of the 29th annual international symposium on Computer architecture*, pages 25–34, Anchorage, Alaska, 2002.

[32] J. M. Tendler, J. S. Dodson, Jr. J. S. Fields, H. Le, and B. Sinharoy. POWER4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–25, 2002. `http://researchweb.watson.ibm.com/journal/rd/461/tendler.html`.

[33] 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 the University of the Witwatersrand and the South African National Research Foundation.