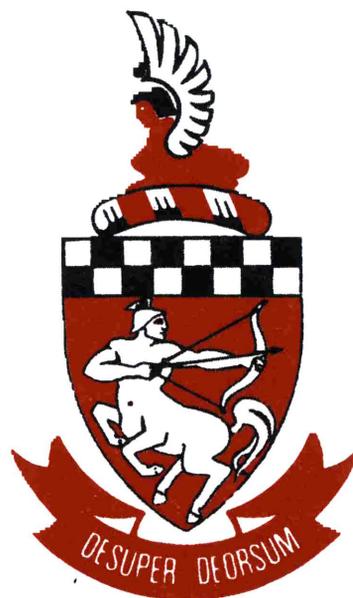


**South African  
Computer  
Journal**

Number 25  
August 2000

**Suid-Afrikaanse  
Rekenaar-  
Tydskrif**

Nommer 25  
Augustus 2000



**The South African  
Computer Journal**

*An official publication of the Computer Society  
of South Africa and the South African Institute of  
Computer Scientists*

**Die Suid-Afrikaanse  
Rekenaartydskrif**

*'n Amptelike publikasie van die Rekenaarvereniging  
van Suid-Afrika en die Suid-Afrikaanse Instituut  
vir Rekenaarwetenskaplikes*

World-Wide Web: <http://www.cs.up.ac.za/sacj/>

---

**Editor**

Prof. Derrick G. Kourie  
Department of Computer Science  
University of Pretoria, Hatfield 0083  
dkourie@cs.up.ac.za

**Production Editors**

Dr. Andries Engelbrecht  
Department of Computer Science  
University of Pretoria, Hatfield 0083

**Sub-editor: Information Systems**

Prof. Nick du Plooy  
Department of Informatics  
University of Pretoria, Hatfield 0083  
nduplooy@econ.up.ac.za

Prof. Herna Viktor  
Department of Informatics  
University of Pretoria, Hatfield 0083  
sacj\_production@cs.up.ac.za

---

**Editorial Board**

Prof. Judith M. Bishop  
University of Pretoria, South Africa  
jbishop@cs.up.ac.za

Prof. R. Nigel Horspool  
University of Victoria, Canada  
nigelh@csr.csc.uvic.ca

Prof. Richard J. Boland  
Case Western University, U.S.A.  
boland@spider.cwrw.edu

Prof. Fred H. Lochovsky  
University of Science and Technology, Hong Kong  
fred@cs.ust.hk

Prof. Trevor D. Crossman  
University of Natal, South Africa  
crossman@bis.und.ac.za

Prof. Kalle Lyytinen  
University of Jyväskylä, Finland  
kalle@cs.jyu.fi

Prof. Donald D. Cowan  
University of Waterloo, Canada  
dcowan@csg.uwaterloo.ca

Dr. Jonathan Miller  
University of Cape Town, South Africa  
jmiller@gsb2.uct.ac.za

Prof. Jürg Gutknecht  
ETH, Zürich, Switzerland  
gutknecht@inf.eth.ch

Prof. Mary L. Soffa  
University of Pittsburgh, U.S.A.  
soffa@cs.pitt.edu

Prof. Basie H. von Solms  
Rand Afrikaanse Universiteit, South Africa  
basie@rkw.rau.ac.za

---

**Subscriptions**

	Annual	Single copy
Southern Africa	R80.00	R40.00
Elsewhere	US\$40.00	US\$20.00

An additional US\$15 per year is charged for airmail outside Southern Africa

to be sent to:

*Computer Society of South Africa  
Box 1714, Halfway House, 1685  
Phone: +27 (11) 315-1319 Fax: +27 (11) 315-2276*

## In Memoriam: Stef Postma

*South Africa has lost one of her most colourful and eminent computer scientists. Professor Stef Postma passed away peacefully in his sleep on May 5, 2000 after a short illness. He will be remembered for his forthright views and total integrity. Never a man to shy from controversy, he always debated his position with vigour, displaying his extensive vocabulary at every opportunity.*

*Those who knew him mourn the loss of a very good friend.*

*Stef was born on August 10, 1938 in Graaff-Reinet and matriculated from H $\ddot{o}$ erskool Linden in Johannesburg. He majored in geology and mathematics at the University of the Witwatersrand and graduated with honours in mathematics from that university. Stef devoted much of his life to promoting computer science as a science and to this end spent a lot of energy and time defining syllabi for undergraduate and honours courses at our universities. He was the prime mover in creating the South African Institute of Computer Scientists and Information Technologists (SAICSIT) in 1982, providing a professional body to represent the interests of local computer scientists. He was also instrumental in establishing Quaestiones Informaticae (now the South African Computer Journal) which afforded South African computer scientists the opportunity to publish papers locally in a refereed journal.*

-Doug Laing

## Links2Go "Computer Science Journals" Award

The SACJ production editing team received the following unsolicited e-mail:

The page at <http://www.cs.up.ac.za/sacj/>, was selected as a Links2Go "Key Resource" in the Computer Science Journals topic, at [http://www.links2go.com/topic/Computer\\_Science\\_Journals](http://www.links2go.com/topic/Computer_Science_Journals).

How your page was selected:

Each quarter, Links2Go samples millions of web pages to determine which pages are most heavily cited by web pages authors, such as yourself. The most popular pages are downloaded and automatically categorized by topic. At most 50 of the pages related to a topic are selected as "Key Resources." Out of 50 pages selected as Key Resources for the Computer Science Journals topic, your page ranked 19th. For topics like Music, where there are a large number of interested authors and related pages, it is harder to achieve selection as a Key Resource than for a special-interest topic, such as Quantum Physics.

The Links2Go Key Resource award differs from other awards in two important ways. First, it is objective. Most awards rely on hand selection by one or more "experts," many of whom have only looked at tens or hundreds of thousands of pages in bestowing their awards. Selection for these awards means no more than that one person, somewhere, noticed your page and liked it enough to select it. The Key Resource award, on the other hand, is based on an analysis of millions of web pages. Any group or organization who conducts a similar analysis will arrive at similar conclusions. When Links2Go says your page is a Key Resource, we mean that your page is one of the most relevant pages related to a particular topic on the web today, using an objective statistical measure applied to an extremely large data set.

Second, the Key Resource award is exclusive. We get literally hundreds of people requesting that their page be added to one or more topics per week. All of these requests are denied. The only way to get listed as a Key Resource is to achieve enough popularity for our analysis to select your pages automatically. We do not accept fees, offers of link exchanges, free advertising, or bartered livestock as inducements to add new sites to our lists. Fewer than one page in one thousand will ever be selected as a Key Resource.

Once again, congratulations on your award! Links2Go Awards [awards@links2go.com](mailto:awards@links2go.com)

# Scalability of the RAMPAGE Memory Hierarchy

Philip Machanick

*Department of Computer Science, University of the Witwatersrand,  
philip@cs.wits.ac.za*

## Abstract

*The RAMPAGE memory hierarchy is an alternative to the traditional division between cache and main memory: main memory is moved up a level and DRAM is used as a paging device. As the CPU-DRAM speed gap grows, it is expected that the RAMPAGE approach should become more viable. Results in this paper show that RAMPAGE scales better than a standard second-level cache, because the number of DRAM references is lower. Further, RAMPAGE allows the possibility of taking a context switch on a miss, which is shown to further improve scalability. The paper also suggests that memory wall work ought to include the TLB, which can represent a significant fraction of execution time. With context switches on misses, the speed improvement at an 8 GHz instruction issue rate is 62% over a standard 2-level cache hierarchy.*

**Keywords:** *memory hierarchy, memory wall, caches, computer system performance simulation*

**Computing Review Categories:** *B.3.2, B.3.3*

## 1 Introduction

The RAMPAGE memory hierarchy is an alternative to a conventional cache-based hierarchy, in which the lowest-level cache is managed as a paged memory, and DRAM becomes a paging device. The lowest-level cache is in effect an SRAM main memory. Disk remains as a secondary paging device. Major hardware components remain the same as in the conventional hierarchy, except that cache tags are eliminated in the SRAM main memory, as it is physically addressed after page translation. Another difference from the conventional hardware is that the TLB caches SRAM main memory address translations, instead of DRAM address translations. In keeping with a paged memory, misses and replacement policy in the SRAM main memory are handled in software.

The RAMPAGE memory hierarchy is motivated by the need to solve the memory wall problem [12, 5, 11]. A key insight leading to the development of the RAMPAGE model is the fact that cache miss costs with the current CPU-DRAM speed gap are in the same ballpark as page fault costs in early virtual memory systems [6] — as ratios not absolute numbers.

A relatively neglected aspect of the memory wall problem is the fact that TLB management can be a significant fraction of run time. In programs with a largely regular memory access pattern, this issue may not seem so important, but in programs with less benign locality (e.g. databases [10], or programs with many small objects randomly scattered over the address space [7]), TLB behaviour may make a significant difference to performance. Since TLB misses can ultimately result in DRAM references, they should be considered as part of the memory wall problem.

This paper does not present significant TLB data, but

raises the necessity of dealing with the problem, and argues how RAMPAGE can make a contribution.

Previously published work [9] focused on hardware-software trade-offs. Correction of inaccuracies in previously reported results has shown that RAMPAGE is significantly faster than a comparable conventional hierarchy, under the conditions in which it was measured [8].

This paper focuses on how RAMPAGE addresses the memory wall problem, by hiding latency through context switches on misses. In particular, the focus is on showing how time spent waiting for DRAM can be hidden, provided there are available processes.

More detail on the hierarchy simulated here and on related research can be found in previously published work [9].

Measurements here are for a multiprogramming mix, but multithreaded applications could also work well on a RAMPAGE machine.

The remainder of this paper is structured as follows. Simulation parameters are summarized in Section 2, and results presented in Section 3. Finally, conclusions are presented in Section 4.

## 2 Simulated Systems

### 2.1 Introduction

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

The approach used is to measure RAMPAGE against a conventional 2-level cache system, with a 2-way associative L2 cache. The only major hardware difference (in terms of added components) is that the conventional system has tags and associated logic.

RAMPAGE measurements have two variations: without

and with context switches on misses. The version without context switches on misses models the effect of better management of replacement (the cost is software management, which has to be traded against fewer misses). Measurement with context switches on misses is intended to show how RAMpage scales up better than a conventional cache-based hierarchy as the CPU-DRAM speed gap grows.

Upper levels of the memory hierarchy – L1 caches and TLB – are chosen conservatively, to reduce any benefit seen from reducing misses to DRAM. With a more aggressive L1 and a bigger TLB, RAMpage should do better, as time spent in DRAM will be a larger fraction of overall time. A bigger TLB would improve performance of small pages in RAMpage: the simulated configuration has a very high TLB overhead for small pages [9].

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

## 2.2 Benchmarks

Measurements were done with traces containing a total of 1.1-billion references, from the Tracebase trace archive at New Mexico State University<sup>1</sup>.

The traces were interleaved, switching to a different trace every 500,000 references, to simulate a multiprogramming workload. These SPEC92 traces are used for comparability with earlier results. Although individual traces would be too small to exercise a memory hierarchy of the size measured here, the combined effect of all the traces, simulating a multiprogramming workload, is sufficient to warm up the memory hierarchy simulated for this paper [9].

## 2.3 Common Features

Both systems are configured as follows:

- CPU – single-cycle execution, pipeline not modeled
- L1 cache – 16Kbytes each of data (D) and instruction (I) 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 D cache: perfect write buffering, zero (effective) hit time, write-back (12-cycle penalty; 9 cycles for RAMpage – 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 [2] without pipelining: 50ns before first reference started, thereafter 2 bytes every 1.25ns

<sup>1</sup>The traces used in this paper can be found at <ftp://tracebase.nmsu.edu/pub/.tbl/r2000/utilities/> and <ftp://tracebase.nmsu.edu/pub/.tbl/r2000/SPEC92/>.

- paging of DRAM – inverted page table: same organization as RAMpage main memory for simplicity (infinite DRAM with no misses to disk)
- TLB and L1d hits fully pipelined; hit times are used to simulate replacements and maintaining inclusion

Detail of the L1 cache is similar across all variations. A superscalar CPU is not explicitly modeled: the cycle time represents instruction issue rate rather than actual CPU cycle time. Issue rates of 1 GHz to 8 GHz are simulated to model the growing CPU-DRAM speed gap (cache and SRAM main memory speed are scaled up but DRAM speed is not).

## 2.4 Conventional Cache System Features

The cache-based system has a 4 Mbyte 2-way set associative L2 cache using a random replacement policy. Block (line) size is varied in experiments from 128 bytes to 4 Kbytes. The bus connecting the L2 cache to the CPU is 128 bits wide and runs at one third of the CPU clock rate (i.e., 3 times the cycle time). 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 [3], 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.

## 2.5 RAMpage System Features

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

The overhead is lower for a smaller SRAM main memory in terms of page table size, but the size needed for operating system code and data is fixed (in the current model). Preliminary work on a range of different sizes shows that a 1 Mbyte SRAM main memory is the minimum size which is practical, with the overhead of the current design.

The RAMpage SRAM main memory uses an inverted page table [4], and replacements use a standard clock algorithm [1].

The TLB in the RAMpage hierarchy caches translations of SRAM main memory addresses and not DRAM

physical addresses. Another major difference from the conventional hierarchy is that a TLB miss never results in a reference below the SRAM main memory, unless the reference itself results in a page fault in the SRAM main memory.

### 2.6 Context Switches

Measurement is done by adding a trace of simulated context switch code, based on a standard textbook algorithm, to the conventional system (approximately 400 references per context switch). In the RAMpage system, context switches are also taken in one set of measurements on misses to DRAM. In the RAMpage model, the context switching code and data structures are pinned in the RAMpage SRAM main memory, so that switches on misses do not result in further misses to DRAM by the operating system code or data. In the conventional hierarchy, context switches can result in operating system references to DRAM.

## 3 Results

### 3.1 Introduction

Results in this section are focused on illustrating how RAMpage can address the memory wall problem. In particular, context switches on misses are highlighted as a mechanism for providing scalability as CPU speeds increase relative to memory speeds.

Results are summarized in two forms: fraction of run time spent at each level of the hierarchy, and speedups versus both a cache-based hierarchy and the slowest CPU.

### 3.2 Memory Occupancy

Table 1 presents fractions of times spent in various levels of the memory hierarchy. TLB hits and L1d hits are not counted separately, as they are fully pipelined. The only TLB and L1d references counted are those resulting from replacements.

The biggest difference is the much lower fraction of time spent in the DRAM level when context switches are taken on misses (the time reported as spent in DRAM is only the time when the processor has to stall when no processes are ready). Although the fraction of time spent in DRAM does also increase in the case of context switches on misses, as the CPU-DRAM speed gap increases, the fraction of time waiting for DRAM remains very small.

Waiting time for DRAM is unlikely to be completely eliminated: changes in the overall workload (e.g., when a new user logs in and starts programs) results in a large number of cold start misses. However, scalability of RAMpage with context switches on misses is clear from this data. Without context switches on misses, RAMpage scales better than the conventional cache but not enough to do more than delay the memory wall problem.

	cache	RAMpage	
		no switches	switches
<b>1 GHz</b>			
L1i	0.646	0.690	0.720
L2	0.280	0.251	0.277
DRAM	0.072	0.057	$4.96 \times 10^{-5}$
best L2 size	512	2048	4096
<b>2 GHz</b>			
L1i	0.603	0.681	0.712
L2	0.261	0.211	0.286
DRAM	0.134	0.105	$1.23 \times 10^{-4}$
best L2 size	512	1024	4096
<b>4 GHz</b>			
L1i	0.532	0.685	0.716
L2	0.231	0.177	0.282
DRAM	0.236	0.136	$2.66 \times 10^{-4}$
best L2 size	512	1024	4096
<b>8 GHz</b>			
L1i	0.430	0.515	0.731
L2	0.186	0.165	0.267
DRAM	0.382	0.318	$4.20 \times 10^{-4}$
best L2 size	512	1024	2048

Table 1: Fraction of time spent in each level, for the best L2 block or SRAM main memory page size for each speed (“best L2 size” refers to this best block or page size). L1 results only include L1i references; “L2” means either the L2 cache or the RAMpage SRAM main memory.

It is useful to illustrate the fraction of time spent in each level of the hierarchy graphically, for the fastest and slowest CPU modeled.

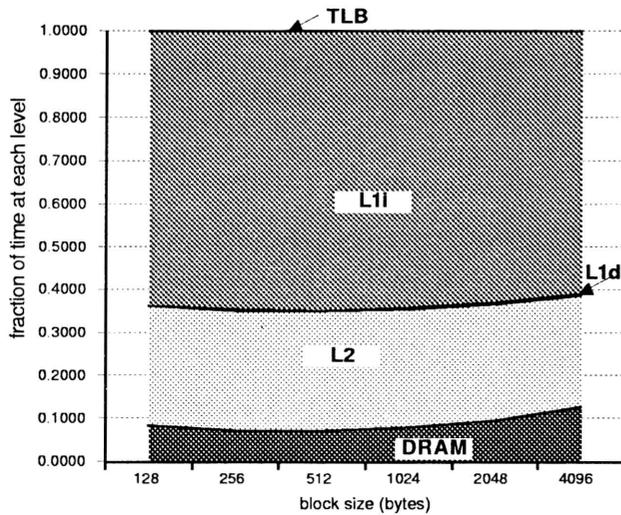
Figure 1 shows the fraction of time spent in each level of memory in the three hierarchies with a 1 GHz instruction issue rate. The fractions are cumulative, totalling 1<sup>2</sup>. The TLB fraction only includes time for references to the TLB itself, not code executed to handle TLB misses (reflected in references in the rest of the hierarchy).

Note again that the time spent waiting DRAM is close to zero for the RAMpage hierarchy with context switches on misses: this happens because it is almost always possible to find another ready process before it’s necessary to wait for a DRAM access to finish.

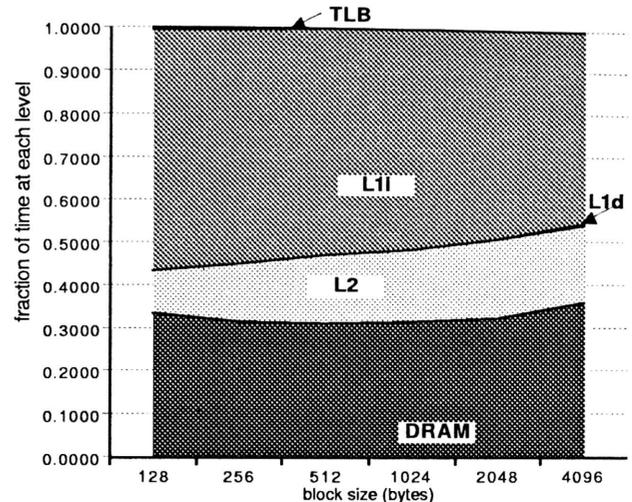
Figure 2 shows the fraction of time spent in each level of memory in the three hierarchies with a 8 GHz instruction issue rate.

A clear difference can be seen between the fraction of time spent in DRAM as the CPU-DRAM speed gap grows, in the cache and RAMpage hierarchies. With context switches on misses, the RAMpage hierarchy becomes much more scalable in its DRAM usage. For this latter case, on the scale of the graphs, it is not possible to see a

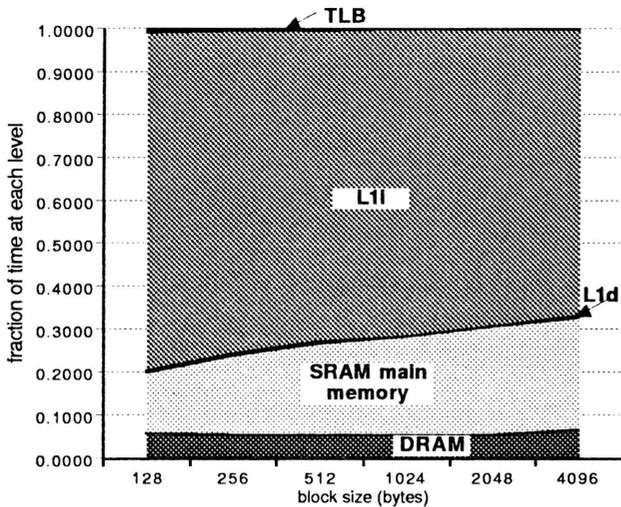
<sup>2</sup>Totals in Table 1 don’t add up to 1 though: L1d and TLB fractions are omitted because they are insignificant.



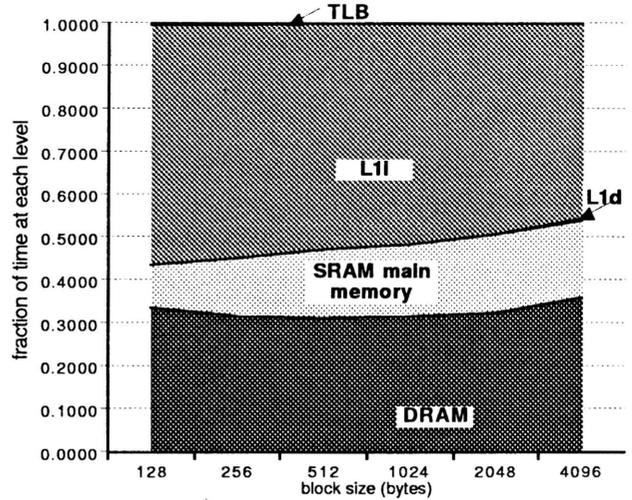
(a) 2-way associative L2



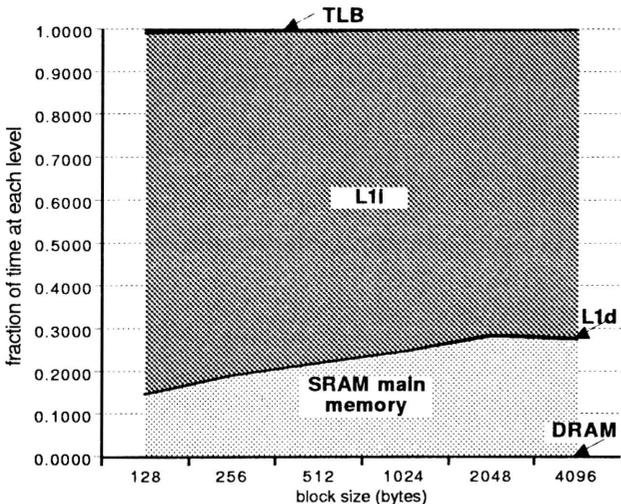
(a) 2-way associative L2



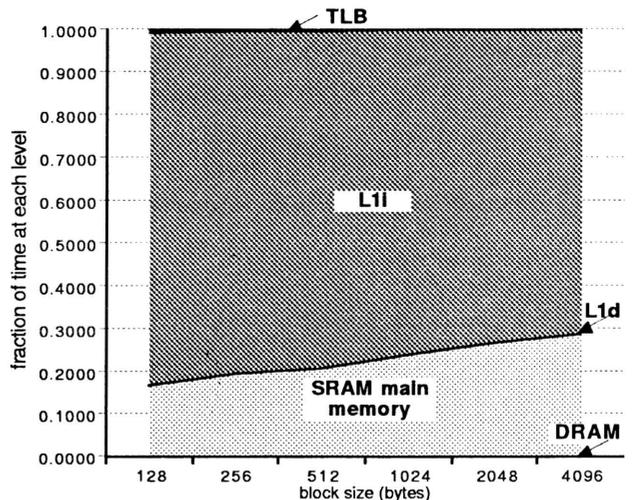
(b) RAMpage no switches on misses



(b) RAMpage no switches on misses



(b) RAMpage with switches on misses



(b) RAMpage with switches on misses

Figure 1: Fraction of time spent at each level of the hierarchy (1 GHz issue rate). *L1d* references are fully pipelined and only *L1d* references caused by replacements appear; TLB time only includes TLB references during replacements.

Figure 2: Fraction of time spent at each level of the hierarchy (8 GHz issue rate). For explanation, see Figure 1.

difference between the 1 GHz and 8 GHz cases (for numbers see Table 1).

### 3.3 Speedups

Results presented here represent speedup over a conventional hierarchy, and over the slowest hierarchy measured. The purpose of presenting the data in this way is to illustrate the significantly better scalability of RAMpage with context switches more directly.

	speedup vs. cache		speedup vs. 1 GHz		
	RAMpage no switches	context switches	cache	RAMpage no switches	context switches
1 GHz	1.042	1.087	–	–	–
2 GHz	1.062	1.151	1.9	1.9	2.0
4 GHz	1.087	1.315	3.3	3.4	4.0
8 GHz	1.143	1.622	5.3	5.8	7.9

Table 2: Speedups. “Context switches” refers to switches on misses; all comparisons are against the best block size for each case.

Table 2 shows that the RAMpage model without context switches on misses shows a modest improvement over the conventional cache hierarchy for lower CPU-DRAM speed gaps; this improvement increases as the speed gap grows, to 14% faster than the cache hierarchy at an 8 GHz instruction issue rate.

While the RAMpage hierarchy does scale better, taking context switches on misses (for the workload measured here) is significantly better. Again, the improvement is modest for lower speed gaps, but for the 8 GHz case, the speed improvement is 62% (or a speedup of 1.62).

The improved scalability of RAMpage and more particularly, RAMpage with context switches on misses, is shown more clearly in looking at the speedup of each hierarchy over the same hierarchy at 1 GHz. While RAMpage without switches on misses does scale better than the cache hierarchy, only RAMpage with switches on misses scales almost linearly with clock speed.

### 3.4 Summary of Results

The RAMpage hierarchy scales better than a 2-way associative cache. However, without context switches on misses, RAMpage still spends an increasingly high fraction of its time in DRAM, as the CPU-DRAM speed gap grows.

For the fastest CPU measured, time spent in DRAM for the conventional hierarchy is almost 40% of the total (for the best choice of L2 block size). For RAMpage without context switches on misses, time in DRAM is reduced to 32% – still high, if better. This result suggests that increasing associativity alone, even in large L2 caches, has limited potential for addressing the memory wall. By contrast, the fraction of time spent waiting for DRAM with

the 8 GHz RAMpage hierarchy with context switches on misses is only 0.04%.

## 4 Conclusions

### 4.1 Introduction

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

### 4.2 Discussion of Results

RAMpage with context switches on misses has a relatively small increase in time spent waiting for DRAM as the CPU-DRAM speed gap is increased. This increase should be related to the fact that RAMpage performs best with relatively large page sizes (2Kbytes for this case). With a more aggressive TLB, RAMpage may perform better with smaller page sizes. Smaller page sizes will likely result in fewer cases where a miss to DRAM is still not completely handled when there are no more processes ready to run.

Taking context switches on misses is a promising approach to the memory wall problem. RAMpage relies on a CPU-DRAM speed gap high enough to recover the cost of the extra software miss handling and the increased number of context switches. In the results here, at an 8 GHz issue rate, a speed improvement over a conventional cache of 62% is seen. Clearly, improvements on a conventional hierarchy could reduce this benefit. However, a key factor in RAMpage’s favour is that it scales well as the CPU-DRAM speed gap grows, as can be seen from its near-linear speedup with CPU speedup of 8.

### 4.3 Future Work

Preliminary results show that a 1 Mbyte SRAM main memory is practical for the faster CPUs modeled. It would be useful to extend this work further, and determine the break-even point for RAMpage across a number of different SRAM main memory sizes.

A more aggressive L1 and TLB will favour RAMpage (as well as being more realistic).

A more aggressive L1 will result in a higher fraction of the runtime being in DRAM, assuming that the absolute number of misses from L2 (or SRAM main memory) is not reduced. Consequently, RAMpage benefits will be increased. The current hierarchy has deliberately been designed to be conservative, so as not to favour RAMpage.

RAMpage relies on the TLB for page translations to the SRAM main memory. A larger TLB would likely make RAMpage more competitive with smaller page sizes.

Work done in the mid-90s [7] showed that TLB overhead could at that time account for as much as 25% of execution time in a specific class of application, one with a large number of randomly allocated objects. Given that

TLB management can result in DRAM references, we should also give consideration to a *TLB wall*. The traces used here do not exhibit poor TLB behaviour, so it is necessary to investigate applications which will have problematic locality properties. An important aspect of the work in the 1990s was that the program which had poor TLB behaviour had been designed for good cache behaviour. The fact that pages differed significantly in size from cache blocks made it difficult to optimize for both areas of the memory hierarchy simultaneously. The RAMpage model has the advantage that a TLB miss will never result in a DRAM reference if the referenced page is at a higher level of the hierarchy (the SRAM main memory inverted page table contains all page translations for pages at that level).

It would be interesting to implement a more complete CPU simulation. It is likely that out of order execution and non-blocking L1 caches are second-order effects, given the high latency of DRAM. However, it would be useful to have a more accurate CPU model to investigate other aspects of RAMpage in more detail, particularly the cost of context switches in terms of pipeline stalls.

It would also be interesting to implement a more complete operating system simulation, to investigate possible variations in standard paging and context switching strategies, adapted to the relatively low latency of DRAM, as opposed to disk, as a paging device.

Finally, it would be interesting to implement a RAMpage machine.

#### 4.4 Final Summary

RAMpage is a promising approach. With the addition of taking context switches on misses, it has the potential to avoid the memory wall problem, at least in cases where processes are available to run while servicing a miss. For applications where running a single process at maximum speed is the requirement, RAMpage has less to offer – at best, fewer misses to DRAM.

Results presented here show that as the CPU-DRAM speed gap grows, the benefits of RAMpage grow. Even though complete implementation would require both hardware and software changes, these changes are not complex. In fact, RAMpage hardware is simpler than that of a conventional cache.

The major obstacle to building a RAMpage system is the fact that a large fraction of the computing world consists of systems where the hardware and software do not originate from the same company.

## References

- [1] C. Crowley. *Operating Systems: A Design-Oriented Approach*. Irwin Publishing, 1997.
- [2] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. Performance comparison of contemporary dram architectures. In *Proc. 26th Annual Int. Symp. on Computer Architecture*, pages 222–233, Atlanta, Georgia, May 1999.
- [3] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Francisco, CA, 2nd edition, 1996.
- [4] 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.
- [5] E.E. Johnson. Graffiti on the memory wall. *Computer Architecture News*, 23(4):7–8, September 1995.
- [6] T. Kilburn, D.B.J. Edwards, M.J. Lanigan, and F.H. Sumner. One-level storage system. *IRE Transactions on Electronic Computers*, EC-11(2):223–35, April 1962.
- [7] P. Machanick. Efficient shared memory multiprocessing and object-oriented programming. *South African Computer Journal*, (16):23–30, April 1996.
- [8] P. Machanick. Correction to RAMpage ASPLOS paper. *Computer Architecture News*, 27(4):2–5, September 1999.
- [9] 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.
- [10] M. Rosenblum, E. Bugnion, S.A. Herrod, E. Witchel, and A Gupta. The impact of architectural trends on operating system performance. In *Proc. 15th ACM symposium on Operating systems principles*, pages 285–298, Copper Mountain, CO, December 1995.
- [11] M.V. Wilkes. The memory wall and the CMOS end-point. *Computer Architecture News*, 23(4):4–6, September 1995.
- [12] 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. Students who have worked on the project are Pierre Salverda (original simulation coding), Lance Pompe (context switches on misses), Ajay Laloo (the context switching strategy does not introduce artifacts) and Jason Spriggs (measurements of 1 Mbyte SRAM main memories). Zunaid Patel, who is working on new variations on the RAMpage hierarchy, proofread this paper, and I would also like to thank Fiona Semple, Scott Hazelhurst and Brynn Andrew for proofreading.

## Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research notes. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications of Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

### Form of Manuscript

Manuscripts for *review* should be prepared according to the following guidelines:

- Use wide margins and  $1\frac{1}{2}$  or double spacing.
- The first page should include:
  - the title (as brief as possible)
  - the author's initials and surname
  - the author's affiliation and address
- an abstract of less than 200 words
- an appropriate keyword list
- a list of relevant Computing Review Categories
- Tables and figures should be numbered and titled.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text according to the Harvard. References should also be according to the Harvard method.

Manuscripts accepted for publication should comply with guidelines as set out on the SACJ web page,

<http://www.cs.up.ac.za/sacj>

which gives a number of examples.

SACJ is produced using the  $\LaTeX$  document preparation system, in particular  $\LaTeX 2_{\epsilon}$ . Previous versions were produced using a style file for a much older version of  $\LaTeX$ , which is no longer supported.

Please see the web site for further information on how to produce manuscripts which have been accepted for publication.

Authors of accepted publications will be required to sign a copyright transfer form.

### Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect typesetting, reproduction and other costs. Currently, the minimum rate is R30.00 per final page for contributions which require no further attention. The maximum is R120.00, prices inclusive of VAT.

These charges may be waived upon request of the author and the discretion of the editor.

### Proofs

Proofs of accepted papers may be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

### Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words. Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

### Book Reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

### Advertisement

Placement of advertisements at R1000.00 per full page per issue and R500.00 per half page per issue will be considered. These charges exclude specialised production costs, which will be borne by the advertiser. Enquiries should be directed to the editor.

**South African  
Computer  
Journal**

Number 25, August 2000  
ISSN 1015-7999

**Suid-Afrikaanse  
Rekenaar-  
tydskrif**

Nommer 25, August 2000  
ISSN 1015-7999

---

## Contents

### Editorial

<b>Stef Postma Memorial</b> .....	1
<b>Links2Go "Computer Science Journals" Award</b> .....	2

---

### Research Articles

A New Approach for Program Integration <b>Z-E Bouras, T Khammaci, S Ghoul</b> .....	3
Technological Experience and Technophobia in South African University Students <b>MC Clarke</b> .....	12
Image coding with Fractal Vector Quantization <b>E Cloete, LM Venter</b> .....	18
A Declarative Framework for Temporal Discrete Simulation <b>H Abdulrab, M Ngomo, A Drissi-Talbi</b> .....	23
Multilingual Training of Acoustic Models in Automatic Speech Recognition <b>C Nieuwoudt, EC Botha</b> .....	32
Syntactic Description of Neighbourhood in Quadtree <b>JR Tapamo</b> .....	38
Object Oriented Programs and a Stack Based Virtual Machine <b>JT Waldron</b> .....	45

---

### Technical Reports

Orthogonal Axial Line Placement in Chains and Trees of Orthogonal Rectangles <b>ID Sanders, DC Watts, AD Hall</b> .....	56
Scalability of the RAMpage Memory Hierarchy <b>P Machanick</b> .....	68

---