Sept. 2005, Vol.20, No.5, pp.– *J. Comput. Sci. & Technol.*

The Value of a Small Microkernel for Dreamy Memory and the RAMpage Memory Hierarchy

Philip Machanick

School of ITEE, University of Queensland, Brisbane, Qld 4072, Australia

E-mail: philip@itee.uq.edu.au

Revised July 10, 2005.

Abstract This paper explores potential for the RAMpage memory hierarchy to use a microkernel with a small memory footprint, in a specialized cache-speed static RAM (tightly-coupled memory, TCM). *Dreamy memory* is DRAM kept in low-power mode, unless referenced. Simulations show that a small microkernel suits RAMpage well, in that it achieves significantly better speed and energy gains than a standard hierarchy from adding TCM. RAMpage, in its best 128kB L2 case, gained 11% speed using TCM, and reduced energy 14%. Equivalent conventional hierarchy gains were under 1%. While 1MB L2 was significantly faster against lower-energy cases for the smaller L2, the larger SRAM's energy does not justify the speed gain. Using a 128kB L2 cache in a conventional architecture resulted in a best-case overall run time of 2.58s, compared with the best dreamy mode run time (RAMpage without context switches on misses) of 3.34s, a speed penalty of 29%. Energy in the fastest 128kB L2 case was 2.18J *vs.* 1.50J, a reduction of 31%. The same RAMpage configuration without dreamy mode took 2.83s as simulated, and used 2.39J, an acceptable trade-off (penalty under 10%) for being able to switch easily to a lower-energy mode.

Keywords low-power design, main memory, virtual memory, cache memories, microkernels

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 paging device. Previous work has shown RAMpage to be a potentially viable design in terms of hardware-software trade-offs[1] and that it scales better as the CPU-DRAM speed gap grows, particularly when taking context switches on misses to DRAM[2].

Preliminary work on RAMpage for low-energy design has shown promise[3]. In this paper, the value of RAMpage in low-energy design is further explored by a more complete study of the idea of *dreamy memory*.

Dreamy memory is kept in a low-power mode unless it is referenced. While waking the memory up incurs significant overhead, RAMpage could hide this overhead as has previously been demonstrated, as illustrated in earlier RAMpage work, aimed at bridging the CPU-DRAM speed gap.

In desktop and server designs, with processor power consumption on the order of tens of watts or even over 100W, reducing memory power usage is not a major issue. However, with a low-energy design, DRAM energy usage becomes significant. A 128Mbyte DRAM as simulated in this study uses about 0.5W, as compared with a 500MHz processor of the ARM11 family[4], which uses about 0.2W in the processor core (excluding caches). A small mobile device, even with a relatively modest memory, therefore has to allocate a significant fraction of its energy budget to DRAM.

In this paper, the approach investigated is to use the self-refresh mode commonly available in double-data rate synchronous DRAM (DDR-SDRAM), which allows DRAM contents to be maintained with 1% of normal power[5], to implement dreamy memory. Simulations are based on parameters suited to a mobile device. The aim is to reduce DRAM energy usage as close as possible to that of self-refresh mode, with performance as close as possible to that of full-power mode.

To extend previous work, a smaller second-level cache or static RAM main memory (L2) is investigated, as is the use of a small cache-speed SRAM memory, a feature of some ARM designs (where it is called "tightly coupled memory", or "TCM"). The notion is that RAMpage, with its ability to control placement in memory by software, should be able to make better use of a specialised layer of the memory hierarchy than can a conventional cache hierarchy. Further, it is hypothesised that a microkernel with a small memory footprint is a good fit to RAMpage, in that the main operating system code and data structures, along with SRAM main memory page tables, could be fitted into a memory like ARM's TCM. A conventional architecture could also place the kernel into TCM, but page tables for the DRAM layer would be too large to fit this small SRAM.

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

RAMpage was proposed[6] in response to the mem-

Financial support for this work has been received from the University of Queensland.

ory wall[7,8], which arises mainly with high-end systems, where processor improvements have not been matched by DRAM speed improvements. At the low end, energy use is a much more significant problem. RAMpage's ability to hide latency of (relatively) slow DRAM can potentially be used to hide the latency of waking a DRAM up from a low-power mode.

In this paper, low energy, rather than low power is the measure of interest, as we are concerned with total energy use over time, rather than an instantaneous measure.

The remainder of this section briefly surveys recent developments in microkernels as they apply to low-energy memory system design, other approaches to low-energy memory design, and an outline of the RAMpage approach to the problem.

2.1 Progress with Microkernels

The notion of a microkernel is conceptually appealing. The component of the operating system which needs to be memory resident and must run without protection is minimised. As much of the operating system as possible runs in user space, improving modularity and simplifying debugging. Unfortunately, early attempts at implementing microkernels in the 1990s, most notably, Mach, led to a perception that a microkernel is inherently inefficient, a perception which more recent designs have attempted to dispel[9,10].

In the meantime earlier, microkernel-based designs have undergone opportunistic transformations like merging protection domains[11], which make them look more and more like strangely designed monolithic kernels.

Possibly the most radical of the second-generation designs is the MIT Exokernel, which completely moves away from defining abstractions, but instead defines primitives which provide secure access to hardware and other low-level machine-specific operations like interrupts[12].

The L4 kernel is a little less radical: it defines a minimal set of abstractions: address spaces, inter-process communication (IPC) and threads. The kernel only supplies 7 system calls, and has a memory footprint of 12kbytes[9]. In this work, 32kB is allowed for kernel memory, to allow for expansions since the original L4 design. This is in contrast to Mach, which has a significantly larger memory footprint, and a large number of system calls: probably a legacy of Mach's history as a rewrite of the UNIX kernel. The overall effect is a factor of improvement of up to 20 over the speed of Mach's IPC[9].

Second-generation microkernels like L4 are of interest in the small device space because a small memory footprint and modular design suit the requirements of these devices. A kernel with a minimal memory requirement and with functionality provided by modules which are outside the kernel provides a basis for designing a flexible operating system, in which functionality can be added or left out as needed.

Operating systems currently used in this space include Symbian and μCLinux (often written as "uCLinux").

Symbian was originally designed as a general-purpose multitasking operating system, if for small personal computers[13]. Symbian's kernel has a memory footprint of the order of 200kbytes[14].

μCLinux is a cut-down version of Linux for microcontrollers. The main omission from the standard Linux kernel is of support for hardware memory management[15]. Since most of the existing kernel functionality is present, the μCLinux kernel is not as small as Symbian: estimates depending on the version vary from under 512kB to around 1MB.

In the context of energy-sensitive design, the potential for keeping the operating system in a small, fast memory, while juggling the rest of memory between fast (high-power) and slow (low-power) memory components is appealing. Symbian's relatively small memory footprint has made it popular for cell phones, though Linux also has a following.

Given that memories on small mobile devices are generally smaller than on desktop and server computers, and energy-efficiency further encourages the use of small memories, there is a case for revitalising the microkernel idea.

2.2 Low-Energy Memory Design

There have been several approaches to reducing the energy needs of memory.

IRAM (Intelligent RAM) was originally proposed to address the memory wall problem, by implementing a large DRAM on-chip with the processor, instead of the traditional trend of increasing on-chip cache size. While the on-chip DRAM is slower than an SRAM cache, it is faster than an off-chip DRAM[16]. More recently, IRAM has been shown to offer the potential for reduced energy usage, because of DRAM's lower energy requirement as compared to SRAM, and elimination of off-chip buses[17].

At the low end, work has been done on variations on memory organization like multiple banks (less commonly used banks can be put in low-power modes), finding optimum combinations of number of banks and bus width, and exploring compromises between performance-optimal and energy-optimal organization of caches and DRAM[18]. One specific proposal for a low-energy design for system-on-chip (SoC) applications is to organize static RAM into statically allocated banks, based on predicted data referencing behaviour[19]. The main problem with this approach is that it requires static allocation, and does not allow for changes in the relative sizes of the banks for different workloads.

The closest ideas to that reported here are Power-Aware DRAM (PADRAM)[20] and Power-Aware Virtual Memory (PAVM)[21]: page placement is used in a memory in which different chips may be in different power modes. Frequently accessed pages are in a DRAM which

is not in a low-power mode (or often less than other chips).

In a PADRAM study, it was shown that putting all DRAM into the lowest-power mode resulted in execution time of 2 to 60 times that of full-power mode, whereas a dynamic policy resulted in a relatively small speed loss, with significant energy saving. While various details of the PADRAM study differed from those reported in this paper (faster processor, 256kB L2 cache, Rambus memory with higher wakeup latency), the most significant difference is that no operating system effects were modelled: single process execution times were reported, not a mix of workloads[20]. In addition, only a hardware-managed 256kbyte L2 cache was modelled, not a software-managed cache like the RAMpage SRAM main memory. RAMpage, especially with context switches on misses, relies on a multiprogramming workload to hide DRAM latency and is therfore able to get away with a simpler approach to managing DRAM.

PAVM has been investigated in more detail, but using an actual implementation on Linux and an otherwise-conventional memory hierarchy. Exploiting a combination of the different modes available in Rambus and dynamic page placement strategies, with DRAM energy savings of up to 59% with a heavy workload[21].

An initial study of dreamy memory[3] has shown promising results. However, this work used a 1MB L2 layer, which is a bit large for low-energy and low-cost designs, and did not take into account power use of the processor and caches (a factor if run time is increased).

Since other low-energy techniques can apply to dreamy DRAM, approaches in areas such as reducing energy to drive a bus to DRAM[22] and reducing cache energy[23] have not been considered in detail as potential competing work.

2.3 RAMpage Approach

RAMpage makes as few changes from a traditional hierarchy as possible (see Fig.1). The lowest-level cache becomes the main memory (i.e., 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[1].

RAMpage 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, if other work is available for the CPU, waiting for DRAM can effectively be eliminated[2]. Performance characteristics of RAMpage have previously been reported[1,2,24]. For purposes of this paper, the key advantage of RAMpage is the ability to mask latency of DRAM references, with the aim of keeping DRAM in a low-power mode unless it is being referenced, without significant loss of speed.

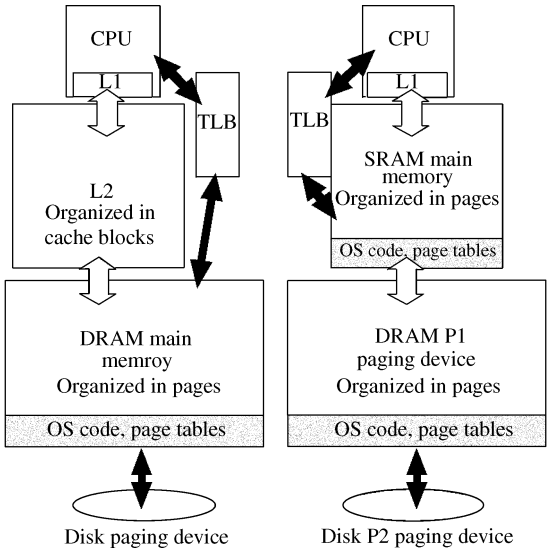

Fig.1. Conventional architecture (left) compared with RAMpage. (Major components are the same but organization differs. RAMpage uses the TLB to map the SRAM main memory instead of DRAM, and some OS code and data structures (shaded) are locked into the RAMpage SRAM main memory.)

Compared with most other approaches to low-energy memory systems, the RAMpage approach is very simple. No special hardware is needed, other than the RAMpage design itself. DRAM is put into a low-power mode, and woken up when it is referenced. As compared with the PADRAM approach, the architecture requires no complex dynamic placement strategy. Provided a process is ready to run on a miss to DRAM, the extra wake-up latency can be masked. PAVM is closer in philosophy, but RAMpage carries the idea further in managing the lowest-level cache in software, which has potential for other wins, as described in previous RAMpage work[1,2].

The dynamic placement strategies of PADRAM and PAVM could be added to RAMpage, combining their benefits with a software-controlled SRAM main memory.

3 Experimental Approach

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.

3.1 Simulation Strategy

The approach followed here is similar to that used in previously reported work. However, the processor speed characteristics are based on the ARM11 series running at 500MHz. This processor consumes 0.2W at this speed (without caches)[4]; this power consumption makes the power needs of DRAM significant.

Simulations are trace-driven, and do not model the pipeline. It is assumed that pipeline timing is less significant than variations in DRAM referencing. Given that the ARM11 family only issues one instruction per clock and has accurate branch prediction, this approach to simulation is unlikely to introduce significant inaccuracies. For simplicity, the simulations do not use all features of the ARM11 series. The ARM11's two-level TLB is not simulated. Instead, a relatively small 1-level TLB is simulated. The RAMpage hierarchy is more disadvantaged by this approximation than a conventional hierarchy, since it relies on the TLB for mapping pages in the SRAM main memory, rather than in DRAM[25].

A standard 2-level hierarchy is compared to a similar version of a RAMpage hierarchy, with and without context switches on misses. RAMpage without context switches on misses is intended to convey the effects of adding associativity (with an operating system-style replacement strategy). Adding context switches on misses shows the value of having alternative work on a miss to DRAM. In all cases, the effect of running with DRAM permanently on is compared with the effect of running with DRAM in self-refresh mode, except when it is referenced.

In earlier studies, a 4MB L2 was simulated. L1 size in early experiments was 16kB each of L1i and L1d. In later work, larger L1 sizes were explored, to determine the effect of cache size variations[25]. In the most recent work prior to this[3], given that energy and cost are more significant than for previous studies, L2 was reduced from 4 Mbytes to 1Mbytes (the simulated 1MB SRAM consumes 0.8W[26]; 4MB would use 3.2W, significant compared with a 0.2W processor core). In this study, the 1MB case is compared with a 128kB L2 layer, which uses 0.1W.

These reductions disadvantage RAMpage more than the standard hierarchy: part of the SRAM main memory is reserved for operating system data and code. In the standard hierarchy, on the other hand, operating system references occupy L2 the same way as any other reference (and so may be removed by conflict or capacity misses).

The ARM11 series includes the option of 64kbytes of SRAM (tightly coupled memory, TCM) which could be used for the operating system in RAMpage; the page table would also fit for SRAM page sizes of 256bytes or more. TCM operates at cache speed, and this option was explored in this study, with the expectation that RAMpage simulations would see significant speed gains from the use of TCM.

ARM's TCM is split between instruction and data[27], so a real implementation would need to take this into account. The effect of a split TCM is approximated in this study by reserving 32kB for the operating system (more than a microkernel like L4 needs), and the remaining 32kB for the RAMpage SRAM page tables.

3.2 Simulation Parameters

The processor modelled in energy-oriented RAMpage studies is slower than in recent speed-oriented RAMpage work, if comparable to one of the speeds in older work[1], to take into account the slower speeds of low-energy designs.

The DDR-SDRAM modelled[5] has an average current draw of 200mA, and self-refresh mode which uses 2mA, both at 2.6V. In self-refresh mode, the external clock is turned off, and contents of DRAM is maintained without external intervention. Actual DRAM power usage varies according to the reference pattern, but for this work, an average value is used, and the same value is used for entry to and exit from self-refresh mode. In performance-oriented work, detail of the DRAM was not considered important, as fixing DRAM speed while speeding up the CPU represented the increasing CPU-DRAM speed gap. In energy-related RAMpage studies, DRAM detail is more important because power usage is timing-dependent.

Unlike the previous study which only measured DRAM energy, an approximate model of CPU and cache power is used. For simplicity, SRAM power is used as an approximation to on-chip cache and TCM power (scaled to the size of each component). A detailed model of cache power would be necessary for accuracy, taking into account power-saving features of a specific processor.

In summary, power parameters are:

- CPU core – 0.2W;
- TCM – 0.05W;
- L1 cache – 0.025W;
- L2 (or SRAM main memory) – 128kB 0.1W; 1MB 0.8W;
- DRAM – 0.52W full power and waking up; 0.0052W in self-refresh mode.

The following parameters are similar to previous simulations except as noted, and are common across RAMpage and the conventional hierarchy:

- L1 cache – 16kbytes each of data and instruction cache, physically tagged and indexed, direct-mapped, 32-byte blocks, 1-cycle read hit time, 12-cycle penalty for misses to L2 (or RAMpage SRAM main memory);
- TLB – 64 entries, fully associative, random replacement, 1-cycle hit time, misses handled in software;
- DRAM – DDR400 SDRAM: 40ns before first reference starts, 64-bit 5ns bus (data moves every 2.5ns: transfer rate approximately 0.3ns per byte; DRAM time to exit self-refresh is 1μs, and time to enter self-refresh mode is 20ns);
- paging of DRAM – inverted page table: same organization as RAMpage main memory for simplicity, the workload is preloaded, so there are no page faults to disk; for energy calculations, a 128MB DRAM is assumed;
- TLB and L1 data (L1d) hits are fully pipelined: they do not add to execution time; only instruction fetch hits add to simulated run time; time for replacements or maintaining inclusion are costed as L1d or TLB "hits".

A context switch (modelled by interleaving a trace of text-book code) is generally taken every 500,000 references, though RAMpage with context switches on misses also switches processes on a miss to DRAM. TLB misses are handled by inserting 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[1].

3.2.1 Specific to Conventional Hierarchy

The L2 cache is 2-way associative with two variants: 128kbytes and 1Mbytes. The bus connecting L2 to the CPU is 128bits wide and runs at one third of the CPU clock rate (6ns versus the CPU's 2ns). The miss penalty from L1 to L2 overall is 12 CPU cycles. Inclusion between L1 and L2 is maintained[28], 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 pages to DRAM physical frames.

3.2.2 Specific to RAMpage Hierarchy

The TLB maps the SRAM main memory. Full associativity is implemented by a software miss handler. The operating system is allocated 32kB of SRAM main memory plus whatever is needed for page tables (e.g., in a 128kB memory with a 4kB page size, the available user memory is 23 pages, or 92kB, which drops to 90kB for 128B pages).

These requirements are conservative, assuming 8bytes per page table entry (the actual size depends on factors like how hash collisions are handled). A small microkernel such as L4 could use less than 32kB. For this reason, a few cases of the 1MB L2 have been modelled in which the page tables would not fit a 64kB TCM (128B pages). This variant could be accommodated by a tighter implementation of page tables but since the results for this case turn out not to be competitive, this issue is not significant to the results.

If the page table were more efficiently implemented than as simulated in this paper, RAMpage without TCM would gain most, since the page tables take up SRAM main memory in that case. Results reported here therefore may slightly overstate the value of TCM.

The SRAM main memory uses an inverted page table. TLB misses do not reference DRAM, if the original reference can be found in an SRAM level.

3.2.3 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 energy use, the size of the SRAM main memory page (or L2 block size in the conventional model) was varied from 128bytes to 4kbytes, and the simulation was instrumented to track energy use. The size of the L2 layer was varied as well: 128kB and 1MB, and measurements were taken with and without a 64kB TCM.

In dreamy mode, it was assumed that if a DRAM access started before the previous one had completed, DRAM would still be awake. Otherwise, once a DRAM reference completed, it was put into self-refresh mode. For comparison, simulations were run with DRAM permanently in full power mode. The simulator allows for a lag after references before entering self-refresh mode, but this option is still to be explored.

Total energy was calculated by multiplying time in each mode by the power of that mode.

3.3 Expected Findings

With a significantly smaller SRAM main memory than in earlier experiments, it was expected that RAMpage, which pins parts of the operating system in the SRAM main memory, would be less competitive on speed than in earlier experiments even in dreamy mode, where the increased effective DRAM latency would make this experiment closer to earlier ones with faster processors.

With the previously measured 1MB L2, RAMpage was able to show a better overall combination of speed and energy, and dreamy mode worked best with context switches on misses[3]. With a smaller L2, different trade-offs are likely.

For example, context switches on misses are less attractive an option as L2 becomes smaller, because pollution of working sets is a more serious problem. However, the smaller L2 is likely to make TCM more useful, especially with RAMpage, which is able to use it more effectively, to remove references from the main memory hierarchy. With a 1MB L2, and additional 64kB adds about 6% to SRAM under the RAMpage operating system's control, whereas with the smaller L2, the increase is 50%. With the bigger L2, therefore, performance gains are likely to be slight from adding TCM, and have to be offset against the higher energy cost of adding this additional SRAM.

4 Results

This section presents results of simulations, with some discussion of their significance. The main focus here is on comparing the effects of varying the memory hierarchy on energy and power use. In all graphs, numbers are normalised to best = 1; actual simulated values are presented in tables.

Fig.2 shows a comparison of the 128kB L2 variations measured. There are very big differences, to the extent

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

that it is useful to cut off the scale of the graphs at 10. Contrast this with the 1MB cases in Fig.3, where it is reasonable to view all variations on one scale.

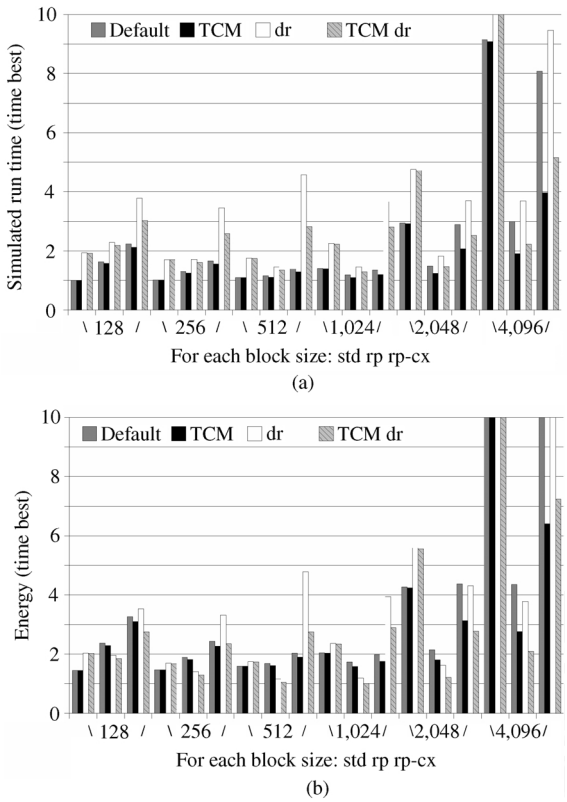

Fig.2. Simulated run time and energy comparison (128kB L2; normalised to best = 1). (a) Normalised speed comparison. (b) Normalised energy comparison. (*In all figures, "rp" means RAMpage, "dr" means dreamy mode and "cx" means with context switches on misses. The case without TCM or dreamy mode is labeled "default". Values are cut off at* 10× *best to make differences in more competitive cases clearer.*)

For the 1MB case, by contrast with the smaller L2, across all variations, TCM has little benefit for speed and in most cases, little (but usually negative) effect on energy use. The performance gain is mostly under 1% for cases of interest – the faster runs – and the extra power needed is not compensated for by a sufficiently shorter run time.

The remainder of this section presents more detail of results. Speed variations are followed by energy variations. Finally, design trade-offs are considered.

4.1 Speed Variations

Speed variations are shown in Table 1. The best dreamy and non-dreamy times are highlighted, for both the 128kB and 1MB L2 cases.

TCM is not a win in general for the 1MB case, as is expected. The speed gains are mostly relatively small (of the order of 1–3%), which is not sufficient to compensate for the extra power needed for the extra cache-speed SRAM (see Subsection 4.3 below). On the other hand, the 128kB L2 does gain from TCM. As the RAMpage results show, the ability to use this extra SRAM effectively can make a significant difference with a smaller L2 by reducing references to DRAM.

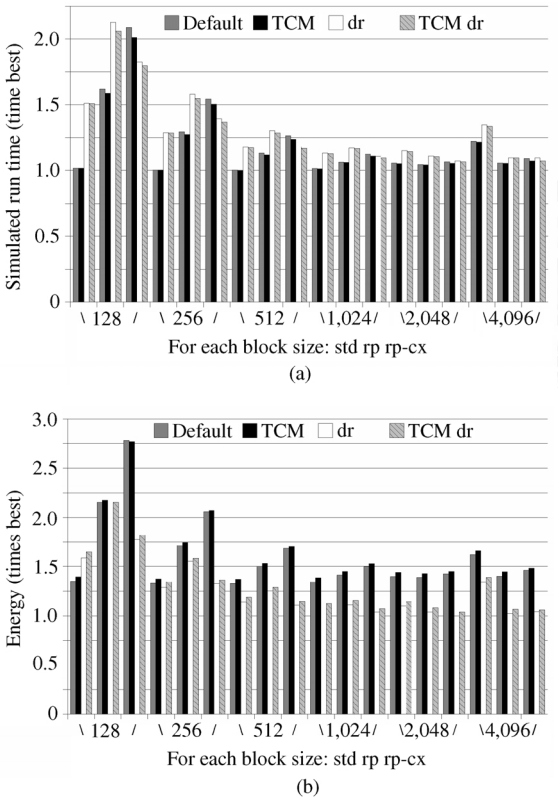

Fig.3. Simulated run time and energy comparison (1MB L2; normalised to best = 1). (a) Normalised speed comparison. (b) Normalised energy comparison. (Variations are less extreme than for the 128kB configuration.)

Fig.4 contrasts speedups resulting from the use of TCM for the 128kB and 1MB L2 (or SRAM main memory) cases. It is noteworthy that the biggest speedups are generally for RAMpage with context switches on misses, reflecting the fact that a higher rate of context switching makes it more useful to remove system code and data structures from the caches.

Given that speed gains from TCM in the 1MB case are so low and energy use is generally higher, from here on, 1MB L2 discussion and analysis is purely for the non-TCM version of the 1MB hierarchy, though full data is presented for completeness.

For the 128kB case, the best dreamy run time is for RAMpage without context switches on misses (1kB SRAM page size). Context switches on misses are not competitive with this configuration, with best-case run times 20% to 138% slower than those of the best alternative for each case.

Although context switches on misses are not generally competitive for the 128kB cases, it is worth noting that these configurations see relatively large gains in simulated run time from adding TCM (11% for non-dreamy; 27% for dreamy). By comparison, the fastest 128kB case (standard hierarchy) gains less than 1% in

Table 1. Speed Variations (s) (Each row shows standard hierarchy (std), RAMpage without (rp) and with context switches on misses (cx). The best time in each column is highlighted.)

Version	L2 block/ page size	128kB L2/SRAM main memory				1MB L2/SRAM main memory			
		Non-dreamy		Dreamy		Non-dreamy		Dreamy	
		Non-TCM	TCM	Non-TCM	TCM	Non-TCM	TCM	Non-TCM	TCM
st	128	2.59	2.58	5.00	4.97	2.40	2.39	3.56	3.56
rp		4.21	4.08	5.91	5.64	3.82	3.74	5.01	4.86
cx		5.77	5.48	9.73	7.82	4.92	4.74	4.30	4.23
st	256	2.61	2.61	4.40	4.38	2.36	2.36	3.03	3.03
rp		3.36	3.24	4.42	4.16	3.04	3.00	3.72	3.65
cx		4.29	4.02	8.93	6.65	3.64	3.54	3.28	3.22
st	512	2.84	2.83	4.55	4.52	2.36	2.36	2.77	2.77
rp		3.00	2.87	3.75	3.50	2.66	2.63	3.07	3.02
cx		3.58	3.33	11.80	7.29	2.98	2.91	2.78	2.75
st	1,024	3.64	c3.62	5.80	5.75	2.39	2.38	2.66	2.66
rp		3.08	2.83	3.77	3.34	2.51	2.50	2.76	2.75
cx		3.48	3.09	9.44	7.27	2.64	2.62	2.61	2.58
st	2,048	7.59	7.55	12.26	12.17	2.48	2.47	2.71	2.70
rp		3.83	3.22	4.72	3.78	2.46	2.45	2.61	2.60
cx		7.46	5.34	9.53	6.52	2.51	2.48	2.52	2.51
st	4,096	23.60	23.44	33.24	33.01	2.88	2.86	3.17	3.15
rp		7.73	4.90	9.53	5.74	2.49	2.48	2.58	2.58
cx		20.85	10.20	24.42	13.3	2.57	2.53	2.58	2.53

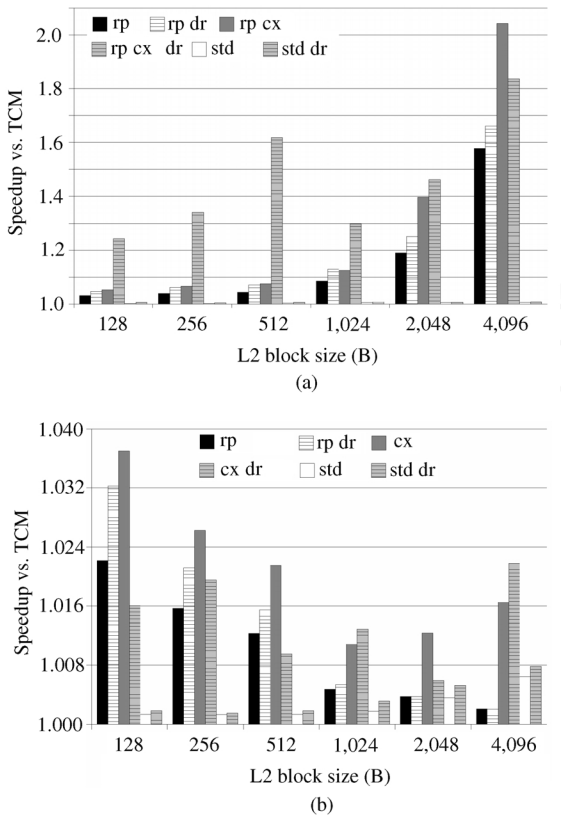

Fig.4. Speedups for TCM over like cases without. (a) 128kB L2. (b) 1MB L2.

speed from adding TCM (dreamy and non-dreamy). RAMpage without context switches on misses sees gains of 6% for the non-dreamy and 11% for the dreamy case. These numbers support the view that a 128kB L2 is too small to support context switches on misses: adding TCM has a disproportionate effect in this case (though not enough to catch up with the other variants).

On the other hand, for the 1MB cases, the best dreamy run time is for RAMpage with context switches on misses, with a 2kB SRAM page size. Execution time here is 6% slower than that for the best non-dreamy case (conventional hierarchy, 512B L2 block size). More speed variation is accounted for by variations in the SRAM page or L2 block size than by using or not using dreamy mode. The slowest dreamy simulated execution time is 5.01s; the slowest non-dreamy time is 4.92s, a difference of under 2%.

4.2 Energy Variations

Table 2 shows the simulated DRAM energy usage for each variation.

For the 1MB L2 measurements, the lowest-energy non-dreamy case is the standard hierarchy with a 512B L2 block size, which also has the quickest execution time. For dreamy runs, likewise, the lowest-energy case is the same as the fastest case, RAMpage with context switches on misses for a 2kB SRAM page size. Context switches on misses are a marginal benefit: the energy gain over the best RAMpage case without context switches on misses (4kB L2 page size) is only 2% (similar to the speed gain).

For 128kB L2 measurements, the picture is rather different. Context switches on misses are never an advantage, reflecting the significant slowdown as reported above in Subsection 4.1.

A 1MB L2 is therefore a reasonable minimum for RAMpage with context switches on misses to be effective – probably because a smaller L2 loses too much from an increase in context switches. More context switches increase the chances of an L2 page being evicted by a competing reference from another process.

4.3 Overall Trade-Offs

Switching from the non-dreamy 1MB standard hierarchy to the non-dreamy 128kB standard hierarchy with

Table 2. Energy Variations (J) (Each row shows standard hierarchy (std), RAMpage without (rp) and with context switches on misses (cx). The lowest energy in each column is highlighted.)

Version	L2 block/ page size	128kB L2/SRAM main memory				1MB L2/SRAM main memory			
		Non-dreamy		Dreamy		Non-dreamy		Dreamy	
		Non-TCM	TCM	Non-TCM	TCM	Non-TCM	TCM	Non-TCM	TCM
std	128	2.19	2.18	3.06	3.03	3.70	3.81	4.35	4.52
rp		3.55	3.44	2.95	2.78	5.90	5.96	5.86	5.90
cx		4.91	4.66	5.32	4.13	7.63	7.59	4.87	4.99
std	256	2.21	2.20	2.54	2.52	3.65	3.76	3.54	3.68
rp		2.84	2.73	2.11	1.95	4.70	4.78	4.26	4.34
cx		3.65	3.42	5.00	3.54	5.64	5.67	3.64	3.73
std	512	2.40	2.40	2.62	2.60	3.65	3.76	3.13	3.26
rp		2.53	2.43	1.74	1.58	4.12	4.20	3.44	3.53
cx		3.06	2.84	7.19	4.14	4.62	4.67	3.04	3.15
std	1,024	3.07	3.06	3.55	3.52	3.69	3.8	2.96	3.08
rp		2.60	2.39	1.79	1.50	3.87	3.98	3.05	3.17
cx		2.98	2.65	5.93	4.36	4.11	4.20	2.85	2.94
std	2,048	6.42	6.38	8.41	8.34	3.84	3.95	3.02	3.14
rp		3.24	2.72	2.44	1.82	3.80	3.91	2.85	2.97
cx		6.58	4.71	6.47	4.16	3.90	3.98	2.74	2.85
std	4,096	19.94	19.81	23.98	23.81	4.45	4.56	3.68	3.80
rp		6.53	4.14	5.68	3.15	3.84	3.96	2.80	2.92
cx		19.62	9.63	20.52	10.88	4.01	4.07	2.85	2.90

TCM results in a speed drop of 9%. The best dreamy-mode execution time for the 1MB case (RAMpage with context switches on misses, 2kB SRAM page size) results in a similar (7%) speed drop.

The fastest 1MB case uses 3.65J, whereas the fastest non-dreamy standard hierarchy with TCM uses 2.18J, an improvement of 40%. By contrast, the best dreamy-mode 1MB case uses 2.74J, an improvement of 25% on the fastest 1MB case.

In terms of a simple speed-energy trade-off, going to a 128kB L2 with a standard hierarchy is a good choice. Adding TCM results in a small improvement in both speed and energy — in both cases, under 1%.

However, if dreamy mode is introduced to the 128kB configuration, a more significant energy gain is possible. The best case without TCM (RAMpage, no context switches on misses, 512B SRAM page) uses 1.74J, a saving of 52% of the fastest 1MB case, for a speed penalty of 59% (an increase from 2.36s to 3.75s). The ability of RAMpage to make more effective use of the whole memory hierarchy shows when TCM is added to dreamy mode on the smaller L2. Energy use drops to 1.5J, and simulated time drops to 3.34s — still an increase of 41.5% on the best 1MB L2 case, but a more acceptable trade-off for an energy saving of 59%.

Given that the 128kB cases can achieve reasonably competitive times but significantly better energy use, they are a better overall choice than the 1MB variants. The remainder of this discussion therefore focuses on the 128kB variants, with TCM.

In non-dreamy mode, RAMpage achieves its best result without context switches on misses with a 1kB SRAM page size. Simulated run time is 2.83s, 20% slower than the fastest 1MB case, or about 10% slower than the fastest standard 128kB variant. This RAMpage variant uses 2.39J, a saving of 35% over the fastest 1MB case, but less than 10% more than the most energy-efficient non-dreamy 128kB case (standard hierarchy, 512B L2 block size).

If dreamy mode is not an option, the best case is the conventional hierarchy with TCM. However, if dreamy mode is an option, a good overall compromise would be a design using RAMpage in which dreamy mode could be turned off for maximum performance, and turned on for minimum energy use. Ideally, it should be possible to vary the SRAM page size, since the best full-power case is with a 512B page, whereas the best dreamy case is with a 1kB page. However, the speed cost of using a 1kB page in full-power mode is less than 1%, so a good overall design compromise would be:

- *organisation*: RAMpage, TCM, no context switches on misses;
- *L2 size*: 128kB (1kB SRAM pages);
- *power mode*: dreamy, option of full power for highest speed.

Overall, RAMpage is not a clear win in as many cases as in earlier work where the focus was on a large CPU-DRAM speed gap.

It is instructive to compare these results with PADRAM, which used *more* energy in its equivalent of a simply dreamy mode than without[20], probably because of its relatively small (256kB) L2 cache. For the 128kB case of the standard hierarchy, similar results are seen here. For the best dreamy-mode 128kB case without TCM (the most directly comparable case with the PADRAM study), the lowest-energy run used 2.54J, as compared with 2.19J for the best (non-TCM standard hierarchy) case. Adding TCM does not significantly alter this finding. The respective simulated run times are 2.59s and 4.40s, consistent with the PADRAM finding that their equivalent runs took about twice as long with a simple model of DRAM sleep.

In the PADRAM study, best energy savings with reasonable speed costs were achieved with a much more complex model than in this study, using a range of low-power options not available in all commodity DRAMs,

and dynamic placement in banks in different modes. RAMpage, with its lower miss rate to DRAM, is able to make effective use of both dreamy mode and TCM, resulting in significant energy savings without the complexity of multiple low-power modes, or dynamic placement strategies.

5 Conclusion

This paper has presented a study of use of the RAMpage memory hierarchy to reduce DRAM energy usage. The approach used was to simulate a *dreamy* memory, in which DRAM is turned off except when referenced. As compared with previous work, a smaller L2 (or static RAM main memory) has been explored, with the addition of a cache-speed physically-addressed SRAM (TCM in terminology of ARM processors).

The remainder of this section summarizes results, outlines future work and presents overall conclusions.

5.1 Summary of Results

RAMpage, with the option of context switches on misses, presents some useful trade-offs in choosing an energy-speed design trade-off. Assuming a relatively low-energy processor design (as well as low-energy components for the remainder of the system), dreamy energy savings could be significant. The fastest 1MB L2 configuration uses 2.4 times the energy of the most energy-efficient (128kB L2) one, for a performance gain of only 29%.

While a significant fraction of the energy gain arises from switching from a 1MB L2 to a 128kB L2, significant gains are still made at fixed L2 size. The fastest 128kB L2 case uses 45% more energy, for a speed gain of 23%.

For a lesser performance cost over the fastest version, the RAMpage model can be run in full-power mode, switching to dreamy mode when power use is more important than speed. Compared to this case, the fastest version uses 17.4% less energy, and is 16.5% faster. The trade-off in choosing this version is a loss of speed in full-power mode, versus lower energy in dreamy mode (and less of a performance penalty for dreamy mode).

The best overall compromise is achieved by RAMpage without context switches on misses, though by a less significant margin than in earlier studies, which showed RAMpage *with* context switches on misses to be the variant most tolerant of high DRAM latencies[2]. A relatively small SRAM layer makes RAMpage less competitive than in these earlier studies. However, RAMpage is able to make more efficient use of a fast but small physically-addressed memory such as TCM, by virtue of the greater control a software memory manager can exercise over placement in memory.

5.2 Future Work

This study has extended previous work by including a model for power use by the processor subsystem. However, power could be more accurately modelled. The approximations used here are sufficient for identifying major differences but a more detailed model would be useful to verify more subtle effects.

Results should be extended to a more detailed analysis of overall system energy, including low-energy variations on caches, and low-energy versions of faster processors. The simulated SRAM has a relatively low latency for waking up from low-power mode (50% of the latency of an L2 hit). Since 1MB SRAM (0.8W) uses more power than DRAM in full-power mode (0.52W) – as simulated here – this would be a useful variation to explore. The 1MB variants would be a lot more competitive on energy use if the large SRAM could be in low-power mode most of the time. Coupled with TCM, the operating system could function a significant fraction of the time without L2. It is possible that the speed loss from a dreamy L2 would be minor. Further investigation of this variation is worthwhile for performance-sensitive applications.

TCM could be modelled more accurately, including simulating a split memory, as in the ARM11 design.

Results here are consistent with earlier studies like PADRAM, which have shown that an approach similar to dreamy mode is not effective with a standard cache hierarchy. RAMpage achieves energy gains using a simple dreamy mode, but could be extended to explore alternatives such as dynamic page placement in banks in different states, and using alternative power saving modes with different power-speed trade-offs. However, PAVM[21] and PADRAM[20] require significantly more complex hardware and software than is proposed in this paper, and the suitability of these approaches needs to be re-evaluated for small-scale devices.

A RAMpage implementation on the L4 Pistachio kernel[10] is planned. This kernel is small enough to permit implementation of its minimum memory-resident data and code in the 64kB static RAM memory (TCM) in the ARM11 family, as simulated in this paper. L4 has been ported to the M5 architecture simulator[29] by the NICTA group at University of New South Wales, creating the possibility of RAMpage on a full-system simulator, a goal of earlier work.

The existing simulator will be used to experiment with further variations on energy-efficient memories. For example, instead of an SRAM main memory, main memory could be implemented in a small permanently powered up DRAM, with the remaining DRAM operating as a dreamy paging device.

5.3 Overall Conclusion

In this latest study, investigating the dreamy memory model with a smaller L2 and a fast physically-addressed cache-speed SRAM has further demonstrated the potential for RAMpage in low-energy designs. While RAMpage did not run in the shortest time in full-power mode (expected with a relatively slow processor), it did have

both the fastest- and lowest-energy measurements in dreamy mode.

The design trade-offs discussed here represent a further data point in the design space: overall low-energy system design requires design of the whole system to minimise energy use. Just as Amdahl's Law shows that focus on one area of speed improvement has diminishing returns, we need to be careful not to interpret energy savings in isolation. Nonetheless RAMpage shows promise in the area of low-energy design, and this study will be followed up with others.

Low-energy design is especially interesting for small-scale devices, so it is of most interest to investigate relatively simple approaches which can work with a small memory, and relatively slow processor. However, the resulting design principles can potentially be interesting to large-scale systems.

While small microkernels are not mainstream in large-scale computers, the value of being able to place the entire kernel in a small fast SRAM is illustrated with the RAMpage results, especially with the smaller L2. Insights resulting from efficient implementation of small microkernels in this context may result in a resurgence of interest in this kind of operating system design in a wider field.

References

[1] Machanick P, Salverda P, Pompe L. Hardware-software trade-offs in a Direct Rambus implementation of the RAMpage memory hierarchy. In *Proc. 8th Int. Conf. Architectural Support for Programming Languages and Operating Systems* (*ASPLOS-VIII*), San Jose, CA, October 1998, pp.105–114.

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

[3] Machanick P. Initial experiences with dreamy memory and the RAMpage memory hierarchy. In *Proc. Ninth Asia-Pacific Computer Systems Architecture Conf.*, Beijing, September 2004, pp.146–159.

[4] ARM. The ARM11 Microprocessor and ARM PrimeXsys Platform. ARM, October 2002. http://www.arm.com/pdfs/ARM11%20Core%20&%20Platform%20Whitepaper.pdf.

[5] Micron Technology. 256Mb: ×4, ×8, ×16 DDR SDRAM, December 2003, Data Sheet. http://download.micron.com/pdf/datasheets/dram/ddr/256M×4×8×16DDR.pdf.

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

[7] Wulf W A, McKee S A. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, March 1995, 23(1): 20–24.

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

[9] Jochen Liedtke. Toward real microkernels. *Commun. ACM*, 1996, 39(9): 70–77.

[10] Uwe Dannowski, Kevin Elphinstone, Jochen Liedtke *et al.* The L4Ka vision. Technical report, University of Karlsruhe, System Architecture Group, April 2001. http://i30www.ira.uka.de/research/documents/l4ka/L4Ka.pdf.

[11] Jay Lepreau, Mike Hibler, Bryan Ford *et al.* In-kernel servers on Mach 3.0: Implementation and performance. In *USENIX MACH III Symp.*, USENIX Association, 1993, pp.39–56.

[12] Engler D R, Kaashoek M F, O'Toole J Jr. Exokernel: An operating system architecture for application-level resource management. In *Proc. 15th ACM Symp. Operating Systems Principles*, ACM Press, 1995, pp.251–266.

[13] Alessandro Maccari. Experiences in assessing product family software architecture for evolution. In *Proc. 24th Int. Conf. Software Engineering, ICSE'02*, ACM Press, 2002, pp.585–592.

[14] Peter Sanders. Creating Symbian OS phones. White Paper, April 2002. http://www.symbian.com/technology/create-symb-OS-phones.html.

[15] David McCullough. uCLinux for Linux programmers. *Linux J.*, July 2004, (123).

[16] Ashley Saulsbury, Fong Pong, Andreas Nowatzyk. Missing the memory wall: The case for processor/memory integration. In *Proc. 23rd Ann. Int. Symp. Computer Architecture*, 1996, pp.90–101.

[17] Richard Fromm, Stylianos Perissakis, Neal Cardwell *et al.* The energy efficiency of IRAM architectures. In *Proc. 24th Int. Symp. Computer Architecture*, Denver, CO, 1997, pp.327–337.

[18] Luca Benini, Alberto Macii, Massimo Poncino. Energy-aware design of embedded memories: A survey of technologies, architectures, and optimization techniques. *ACM Trans. Embedded Computing Sys.*, 2003, 2(1): 5–32.

[19] Yun Cao, Hiroyuki Tomiyama, Takanori Okuma, Hiroto Yasuura. Data memory design considering effective bitwidth for low-energy embedded systems. In *Proc. 15th Int. Symp. System Synthesis*, Kyoto, Japan, 2002, pp.201–206.

[20] Alvin R Lebeck, Xiaobo Fan, Heng Zeng, Carla Ellis. Power aware page allocation. In *Proc. 9th Int. Conf. Arch. Support for Programming Languages and Operating Systems* (*ASPLOS-9*), Cambridge, MA, November 2000, pp.105–116.

[21] Hai Huang, Padmanabhan Pillai, Kang G Shin. Design and implementation of power-aware virtual memory. In *Proc. USENIX 2003 Annual Technical Conference*, San Antonio, Tx, June 2003, pp.57–70.

[22] Hojun Shim, Yongsoo Joo, Yongseok Choi *et al.* Low-energy off-chip SDRAM memory systems for embedded applications. *ACM Trans. Embedded Computing Sys.*, 2003, 2(1): 98–130.

[23] Stefanos Kaxiras, Zhigang Hu, Margaret Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proc. 28th Ann. Int. Symp. Computer Architecture*, Gteborg, Sweden, 2001, pp.240–251.

[24] Machanick P. Correction to RAMpage ASPLOS paper. *Computer Architecture News*, September 1999, 27(4): 2–5.

[25] Machanick P, Patel Z. L1 Cache and TLB Enhancements to the RAMpage Memory Hierarchy. In *Proc. Eighth Asia-Pacific Computer Systems Architecture Conf.*, Aizu-Wakamatsu City, Japan, September 2003, pp.305–319.

[26] NEC. MOS integrated circuit μPD4482162, 4482182, 4482322, 4482362, Data Sheet No. M14522EJ3V0DS00, December 2002. http://www.necel.com/memory/pdfs/M14522EJ3V0DS00.pdf.

[27] ARM. ARM1136JF-S and ARM1136J-S Technical Reference Manual. ARM, revision r0p2 edition, 2003. http://www.arm.com/pdfs/DDI0211D_arm1136_r0p2_trm.pdf.

[28] Hennessy J L, Patterson D A. *Computer Architecture: A Quantitative Approach* 3rd Edition. Morgan Kauffmann, San Francisco, CA, 2003.

[29] Binkert N L, Hallnor E G, Reinhardt S K. Network-oriented full-system simulation using M5. In *Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads* (*CAECW*), February 2003, pp.36–43.

Philip Machanick is a senior member of the IEEE, and a member of ACM. He is director of IT programs at the School of IT and Electrical Engineering at the University of Queensland, Australia. He has published works in memory hierarchy design, computer science education and a number of areas of computer science.