An Equal Area Comparison of Embedded DRAM and SRAM Memory Architectures for a Chip Multiprocessor

Paul Keltcher, Stephen Richardson, Stuart Siu
Computer Systems Technology
HP Laboratories Palo Alto
HPL-2000-53
April, 2000

embedded DRAM, Chip Multiprocessor, CMP, cache hierarchy, pageable memory

Recent architectures in academia and industry have explored placing multiple processors on a single chip, but a consensus has not emerged on the memory architecture. The recent availability of embedded DRAM (EDRAM) has further complicated the formula. In this investigation, we present a new and comprehensive comparison of four very different memory technologies in the same framework: SRAM cache, SRAM configured as pageable memory, EDRAM configured as cache, and EDRAM configured as pageable memory. In addition, these experiments investigate tradeoffs between two levels of on-chip memory, given constant silicon area: as the level one capacity increases, the level two capacity decreases. Having four processors on a single die, each with its own set of level one caches, helps exaggerate the effective memory tradeoffs.

An Equal Area Comparison of Embedded DRAM and SRAM Memory Architectures for a Chip Multiprocessor

Abstract

Recent architectures in academia and industry have explored placing multiple processors on a single chip, but a consensus has not emerged on the memory architecture. The recent availability of embedded DRAM (EDRAM) has further complicated the formula. In this investigation, we present a new and comprehensive comparison of four very different memory technologies in the same framework: SRAM cache, SRAM configured as pageable memory, EDRAM configured as cache, and EDRAM configured as pageable memory. In addition, these experiments investigate tradeoffs between two levels of on-chip memory, given constant silicon area: as the level one capacity increases, the level two capacity decreases. Having four processors on a single die, each with its own set of level one caches, helps exaggerate the effective memory tradeoffs.

1. Introduction

Various research projects have compared SRAM versus DRAM caches [Wils99], SRAM cache versus SRAM and DRAM as paged memory [Mac99, Yam97] and DRAM caches for Level 2 (L2) on-chip memory [Hun94]. Our research is the first to compare all these technologies in the same framework. We have restricted our work to look at the single chip SMP architecture, sometimes called a Chip Multiprocessor (CMP). We have chosen to make each processor simple, with parallelism gained by multiple processors, rather than wide issue. We chose the HP PA-RISC-2.0 Instruction Set Architecture (ISA) for our studies, although at much less aggressive instruction-level parallelism than current implementations. The scope of our research is an equal area comparison of on-chip memory architectures of a CMP, not a comparison or justification of the CMP architecture. Details of our CMP architecture can be found in Section 2. The experiments presented here also investigate tradeoffs between L1 and L2 cache sizes for constant silicon area. This is of special interest in the single chip SMP architecture because with four processors on a chip the available area for the L2 memory reduces quickly as the L1 caches increase in area.

The single chip SMP consists of four 1GHz processors, each with its own L1 instruction and data caches. The processors are connected to a shared L2 memory, which is implemented in either SRAM or on-chip embedded DRAM (EDRAM). The common bus runs at half the processor frequency, or 500MHz. The first level cache is always accessible in a single cycle. The L2 on-chip memory is one of the following:

1. SRAM cache,
2. SRAM as pageable memory (via the OS),
3. EDRAM cache, with the tags, valid and dirty bits held in EDRAM, or
4. EDRAM as pageable memory (via the OS).

In comparing the various architectures, the silicon area is held relatively constant. This means, for example, that increasing the L1 cache size reduces the available silicon for the L2 memory. A smaller L1 cache means more L2 memory, which might benefit parallel data sharing applications but not SpecRate type workloads [SPEC]. In the course of these comparisons, the L1 caches are varied in capacity from 4 kilobytes to 48 kilobytes. The target silicon area for the memory (L1 and L2) is 200mm^2 in 0.18µ technology. For a four processor system with approximately constant silicon area, we target the design points shown below in Table 1.

L1 Cache	Size x 8	L2 cache	L2 Size	Total Size
4KB direct SRAM	6 mm^2	3MB 6-way SRAM	194 mm^2	200 mm^2
4KB direct SRAM	6 mm^2	20MB 5-way EDRAM	195 mm^2	201 mm^2
48KB 12-way SRAM	83 mm^2	2MB 4-way SRAM	125 mm^2	208 mm^2
48KB 12-way SRAM	83 mm^2	12MB 6-way EDRAM	117 mm^2	200 mm^2

Table 1: Silicon Configurations (four processors)

For the full experiment, each L2 configuration shown in Table 1 expands to behave as cache memory and as paged memory, for a total of eight different memory architecture design points for each application. The first level caches are all configured with 4 kilobyte sets. Thus, the 4 kilobyte L1 caches are direct mapped and the 48 kilobyte L1 caches are 12-way set associative. Similarly, the large SRAM L2 caches have 512 kilobyte sets and the large embedded DRAM L2 caches have 2 megabyte sets.

The memory design points of Table 1 were chosen using area models relating memory capacity and architecture to silicon size. The base model for SRAM cache memories was originally presented by Hans Mulder et al. in 1991 [Mul91]. Their model accounts for various cache factors including line size, associativity and write policy. The model measures area in terms of register-bit-equivalents, or RBE's. We compared this model to actual die photos from more than a dozen processors and got fairly consistent results, thus further validating the model. The comparison also helped us translate RBE's to industry-equivalent areas in square millimeters. The comparison adjusts for technology scaling and SRAM cell type (that is, whether the SRAM cell consists of four or six transistors).

Using the Mulder model adjusted to actual die photos gives a rough measure of approximately 1000 register-bit-equivalents per square millimeter when scaled to a 0.18-micron process. The model works well for "typical" cache sizes in the range of zero up to about 64 kilobytes or 128 kilobytes. Very large caches, however, such as the 1.5 megabyte cache on the HP 8500, seem to accomplish much higher densities. For caches in the 1 megabyte-and-up range, we use a figure of 3000 RBE's/mm^2.

The Mulder model, designed specifically to account for on-chip SRAM, will not serve to provide accurate numbers for on-chip DRAM. For this measure, we acquired density numbers from various suppliers of embedded DRAM, including Silicon Access [www.siliconaccess.com], IBM [www.ibm.com], DMEL [www.dmel.com] and TSMC [www.tsmc.com]. Starting from these numbers, we synthesized a formula relating embedded DRAM capacity and layout to its eventual size in 0.18-micron silicon, culminating in the silicon areas shown in Table 1.

The rest of this paper is organized as follows: Section 2 outlines the specifics of the CMP architecture in these experiments; Section 3 gives an overview of related research, Section 4 presents the applications and some characterization of the applications; Section 5 presents the performance analysis of the applications on the various memory architectures and finally in Section 6 some conclusions are drawn from the data presented in Section 5.

2. Architecture

A block diagram of the multiprocessor chip is shown in Figure 1. The processor cores, L1 caches and the central bus are described in Section 2.1. The on chip memory alternatives are described in Section 2.2. Section 2.3 describes the external memory.

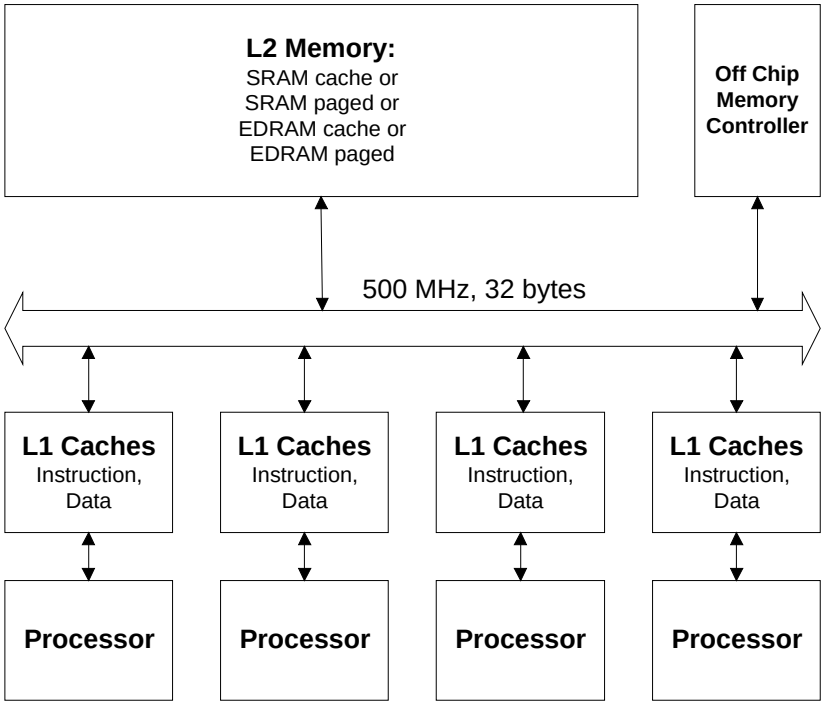

Figure 1: Block Diagram

2.1 Processors Core and the Central Bus

Many architectures today exploit instruction level parallelism to improve single threaded performance. To improve workload throughput and multi-threaded workloads, some designers are placing multiple cores on a chip [Dief99, Ham97, Mur97]. [Olu99] has proposed speculative threading on a CMP to improve single threaded performance. In this paper we are examining architectures to improve multi-threaded and throughput workloads. For this reason, and as described in [Mur97], we chose simple single issue processors, such that more processors can be placed on the die compared to larger multi-issue processors. Hardware techniques to capture instruction level parallelism are expensive in terms of silicon area and cycle time, as well as in time to design and test the chip.

In these experiments all configurations of the L1 cache use the same pipeline and have the same clock frequency. We recognize that a machine with an L1 cache of 48 kilobytes is unlikely to have the same pipeline and frequency of a 4 kilobyte L1 cache machine. By holding this variable constant, however, the memory effects stand out more vividly. The L1 cache misses must contend with each other for the shared central bus, running at 500MHz. The central bus width is equal to a cache line, which is 32 bytes.

The central bus is configured to be split transaction or not split transaction on a case by case basis. Typically, the central bus is configured as a split transaction bus when the L2 is EDRAM, and is configured as a non-split transaction bus when the L2 is SRAM. Because the SRAM latency is low, we found that the overhead of a split transaction bus contributed to lower performance. However, the EDRAM latency is higher, and the split transaction bus allows multiple parallel transactions to a banked EDRAM which contributed to higher performance.

Our cache coherency algorithm allows for shared read only copies, exclusive dirty lines, and no cache to cache transfers. To keep data coherent, the L2 memory must be used to share data. Since the L1 caches are write-back, a separate coherency address bus is used to snoop all writes. The Hydra CMP, in contrast, uses write-through L1 caches to simplify coherency on a write-through address and data bus [Ham98].

2.2 Level Two Memory Alternatives

The four alternative architectures for the L2 memory are SRAM as cache, SRAM as paged memory, EDRAM as cache, and EDRAM as paged memory. As the L1 caches vary from 4 kilobytes to 48 kilobytes, the available area for the L2 SRAM cache allows for capacities from 3 megabytes to 2 megabytes. The SRAM access time is 4ns for all capacities and is not banked. Reading data from the L2 SRAM cache thus requires 10ns: 2ns (one bus cycle) to acquire the bus, plus 2ns to address the L2 cache across the 500MHz central bus, plus 4ns to read the data out of the L2 cache, plus 2ns to transfer data across the central bus to the L1 cache.

Next we consider the two EDRAM alternatives. As the L1 caches vary from 4 kilobytes to 48 kilobytes the EDRAM core (including tags, dirty and valid bits) will vary from 20 megabytes to 12 megabytes. The EDRAM access time is 10ns for all capacities. Reading data from the L2 EDRAM thus requires 18ns: 2ns (one bus cycle) to acquire the bus, plus 2ns to address the L2 cache across the 500MHz central bus, plus 10ns to get the data out of the L2 cache, plus 2ns to re-acquire the bus (split transaction), plus 2ns back across the central bus to the L1 cache. Both the SRAM and EDRAM paged memory configurations have a space advantage over the cache configurations, because they do not need the tags and other bits associated with a cache design. The L2 cache has a 32 byte line, so with 4 bytes of overhead per line for tags and other status bits the overhead is 12.5%, thus the L2 as paged memory is 12.5% less than the area shown in Table 1.

The paged memory architecture requires the operating system to actively page data to and from the external memory. The page size is fixed at 4 kilobytes. Demand paging is used without critical word first, meaning that on a page miss the requesting processor must wait for the entire 4 kilobytes of data to be brought into memory. Additionally, if a dirty page is selected as the victim, it must be cast out of the EDRAM before the external page can be brought into EDRAM memory. We model a FIFO replacement algorithm for page replacement in this paper.

2.3 External Memory

The external memory does not vary in this study, and is configured to be SDRAM at 100MHz on a 128-bit bus. External memory is accessed on an L2 cache miss (either SRAM or EDRAM), or for an entire 4 kilobyte page burst in the case of paged memory. In all architectures the external memory is placed in a closed page mode.

3. Related Work

The scope of our work extends only to comparing various memory hierarchies of a Single Chip Multi-processor (CMP), not comparing or justifying the CMP architecture versus other architectures, such as superscalar, VLIW, SMT, and speculative multithreading [Olu99, Kri99]. For a comparison between superscalar and chip-multiprocessing see [Olu96]. Other CMP projects include Hydra [Olu96, Ham98] and PPRAM [Mur97] from academia, and recent industry announcements such as the Power-4 [Dief99].

Recent Hydra work considers a small SRAM write-through L1 cache and an on-chip L2 cache [Ham97, Ham98]. The L2 cache acts as a write buffer for the off-chip memory hierarchy in this architecture, as the L1 caches are write-through. They note that the L1 cache tends to capture much of the data locality, so that the miss rate of the L2 cache (512 kilobytes in [Ham98]) is high, often 50% to 80%.

Yamauchi [Yam97] proposes using embedded DRAM for the Hydra CMP, and compares an equivalent area EDRAM as paged memory solution with SRAM cache. Data movement between the EDRAM and external memory is controlled by the operating system. However, the applications in their performance study all fit in the 32MB of EDRAM. Three architectures are compared, 1MB and 2MB SRAM caches, and EDRAM as paged memory. Our work expands on [Yam97] by extending the architecture space to EDRAM as cache, and the constant silicon exploration of two L1 and L2 memory capacities. Some of the applications we use do not fit in the on-chip memory and we accurately model paging behavior to and from the off-chip memory.

Nayfeh [Nay94] and Wilson [Wils99] explore several other architectural spaces for on-chip cache organizations. Nayfeh compares three architectures which are shared primary cache, shared secondary

cache, and shared memory. Nayfeh et al. characterize application and application workloads according to which architecture is best suited for that workload. Our work is a more in depth analysis of the second architecture (shared secondary cache) in terms of the memory technologies, architectures, and area tradeoffs. Nayfeh concludes that the higher latency of EDRAM compared to SRAM may be tolerable when used for the L2 memory, as the access is not frequent. The work in [Nay94] and [Wils99] lends weight to our architectural choices. [Wils99, Wils97, Hun94] and many others have compared tradeoffs in L1 cache sizes verses L2 cache sizes with constant silicon area for uni-processor architectures. In the multiprocessor context however, *n* first level caches increase in size as their capacity is increased, thus rapidly decreasing the silicon area for the L2 memory. Our work expands on [Wils99] in two ways. First we explore not just SRAM and EDRAM as cache, but also both memory types as paged by the OS. Second, our research is a multiprocessor study of L1 versus L2 size tradeoffs.

The "RAMpage" memory architecture written about by Machanick [Mac99] is very similar to our pageable memory architecture in that it requires the OS to page data between a faster memory and a slower memory. In our case the faster memory is on-chip SRAM or on-chip EDRAM, and the slower memory is off-chip SDRAM. The RAMpage study compares an off-chip SRAM paged memory (faster memory) with an off-chip SRAM cache, both having equal area, and using RDRAM [Cri97] as a paging device. Additionally, Machanick looks at variable page sizes, while we use fixed 4 kilobyte pages. Both studies compare equal area paged memory with cache, although there are a few differences. Our study is both multi-processor, multi-technology (SRAM and EDRAM) and multi-area (variable L1 caches). Machanick notes that RAMpage simplifies hardware, and requires additional software support. We too find simpler hardware appealing. However, in their comparison with caches, they only employ two way set associative caches. Although this is comparable hardware complexity, it is not typical. We compare paged SRAM and EDRAM with four to six way set associative caches.

Embedded DRAM can enable other architectural choices, as considered in [Ino99] (see also [Joh97]). In our architecture, as in others, the bus width connecting the L2 to the L1 is one L1 line width or 32 bytes. Inoue et al. [Ino97, Ino99] proposed to make the L1 cache line size variable, and make use of the large data bandwidth available with EDRAM. In their architecture the L1 line size is configurable, from 32 bytes to 128 bytes. Some applications exhibit more locality than others, and this is one mechanism to exploit data locality. We model fixed line sizes.

Another merged Logic/DRAM chip-multiprocessor project is PPRAM [Mur97]. PPRAM differs from Hydra and our work in that each processor on the die has its own local DRAM memory. The processors that are on the same chip share a communication bus, and for inter-chip communication a custom "PPRAM-Link" is used. The target granularity is coarse, leading to a design of multiple simple processors, not larger high-ILP super scalar processors. More simple processors can be placed in a given silicon area, compared to complex processors. Other advantages listed in [Mur97] not already mentioned for multiple processors on a chip with EDRAM are lower power consumption (less off chip accesses) and enhancing yield and reliability of chips by exploiting redundant processors.

The IBM Power-4 "processor" [Dief99] is a recently announced hierarchical system with multiple processors per chip and multiple chips per multi-chip module. Two 1GHz processors on a chip share an L2 cache, estimated to be 1.5 megabytes. Four chips are placed on a MCM for an eight processor package. The 32 megabyte L3 cache is off-chip, but the tags are on chip. IBM's embedded DRAM process may find a use in this architecture as hypothesized by [Dief99], but this is not yet known.

4. Applications

For this experiment, we have chosen two applications from the SPLASH benchmark suite [Sing92, Woo95], namely Ocean and Raytrace, along with a snapshot of TPCC [TPCC98], and some multi-user workload type applications drawn from the SPECint 2000 suite, referred to as "SPECsubset" throughout the remainder of this paper. Table 2 shows a summary of our characterization of the applications.

Application	Args	Millions of Instructions	Millions of Data Refs.	4KB L1 miss rates		48KB 12w L1 miss rates		Total 4KB Pages
				Inst	Data	Inst	Data	
Ocean	-n258	550	180	0.1%	7.4%	0.01%	3.7%	3,800
RayTrace	teapot	340	130	5.7%	4.3%	0.01%	0.3%	1,800
TPCC	--	540 (snapshot)	74	8.9%	5.2%	3.1%	0.7%	30,400
SPECsubset	Test	16,000	5,700	5.3%[1]	4.5%	0.00%	0.7%	7,100

Table 2: Application Characteristics

A metric which we find useful for identifying how an application will perform when using paged memory systems is the number of times the application touches each page at the L2 memory (for a given L1 cache organization), divided by the number of instructions in millions (column 3). The last column in Table 2 only shows the number of pages touched. If a page is brought in and used heavily, then the penalty of waiting for the entire 4 kilobyte burst is minimized. However, if the page is touched only a few times, then the penalty is high. Table 3 shows this page touch metric as seen by the L2 memory for the applications in this study given a 48 kilobyte L1 cache. It would appear from Table 3 that Ocean may be more suited to a paged memory architecture than TPCC and SPECsubset. Section 5 (Performance Analysis) will attempt to show the usefulness of this metric.

Application	Page touch metric
Ocean	6.9
RayTrace	2.1
TPCC	1.0
SPECsubset	0.8

Table 3: Page touch metric

5. Performance Analysis

An execution-driven simulator generates multi-processor traces for each application. A trace driven simulator consumes these traces and collects statistics. Operating system execution is not included in the simulation trace file. The trace-driven simulator runs on multiple host types (HPUX and Linux), and the job queue is managed by Condor [Ram98]. All graphs are generated using auto-generated Matlab [MAT] scripts. In this section each application will be analyzed with conclusions to follow in Section 6.

The performance criteria is execution time, broken down into the following seven components. Each component is represented by a bar in the performance graphs to follow.

- Instruction execute time. This is the time spent executing instructions and time spent waiting on pipeline stalls.
- Bus stall. Each resource has both stall and contention components. Stall is the latency of the resource. It is how long the processor must wait to acquire and use the resource. In this case, it is the time acquire the bus and to transfer one cache line of data on the bus.
- Bus contention. Contention is time spent by a CPU waiting on another device to free up the resource.
- L2 Stall. The stall (latency) time of the level 2 memory depends on the architecture, either 4ns for SRAM or 10ns for EDRAM. Since the EDRAM is operated in closed page mode stall calculation is trivial.
- L2 Contention. The on-chip EDRAM memories are multi-way banked, however with four processors conflicts can still occur. Contention will show when a processor is waiting on a read from the L2, and the requested bank is in use. The bank could be in use from another processor performing a read or write, or even from a previous write from the issuing processor.
- External memory stall. The external memory is accessed on an L2 read miss.

[1] The 4 kilobyte numbers for SPECsubset are not correct and will be fixed in the final paper.

- External memory contention. External memory is multi-banked according to typical SDRAM memory part specifications. An application with a large memory footprint will have more SDRAM banks.

The performance analysis primarily concerns runtime, as presented by the sum of the components just described. In terms of the application characterization described in the previous section, the impact of L1 miss rate can be seen in bus and L2 stall and contention times, and the impact of the L2 miss rate can be seen in the external memory stall and contention times. The miss rates by themselves do not tell the whole performance picture, however. A cache line L1 castout (dirty victim) will not show up as latency in these graphs, nor is it reflected in the miss rates. The castout must be written back to memory, and makes its way through the memory hierarchy, possibly causing misses, page faults, and external memory accesses. A read (L1 miss) may encounter contention from the castout, which would be reported since the CPU must wait for the read to complete.

Figure 2 shows the results for Ocean. The *y* axis measures execution time in seconds. Along the *x* axis are eight bars. The first four bars are the SRAM configurations, and the last four bars are the EDRAM configurations. Within each group of four, the first two configurations are L2 as cache, and the last two are L2 as pageable memory. Lastly, 4 kilobyte L1 caches are given before 48 kilobyte 12-way L1 caches. The individual bars contain each of the performance criteria mention earlier: Instruction execution time, bus stall, bus contention, L2 stall, L2 contention, external memory stall and external memory contention.

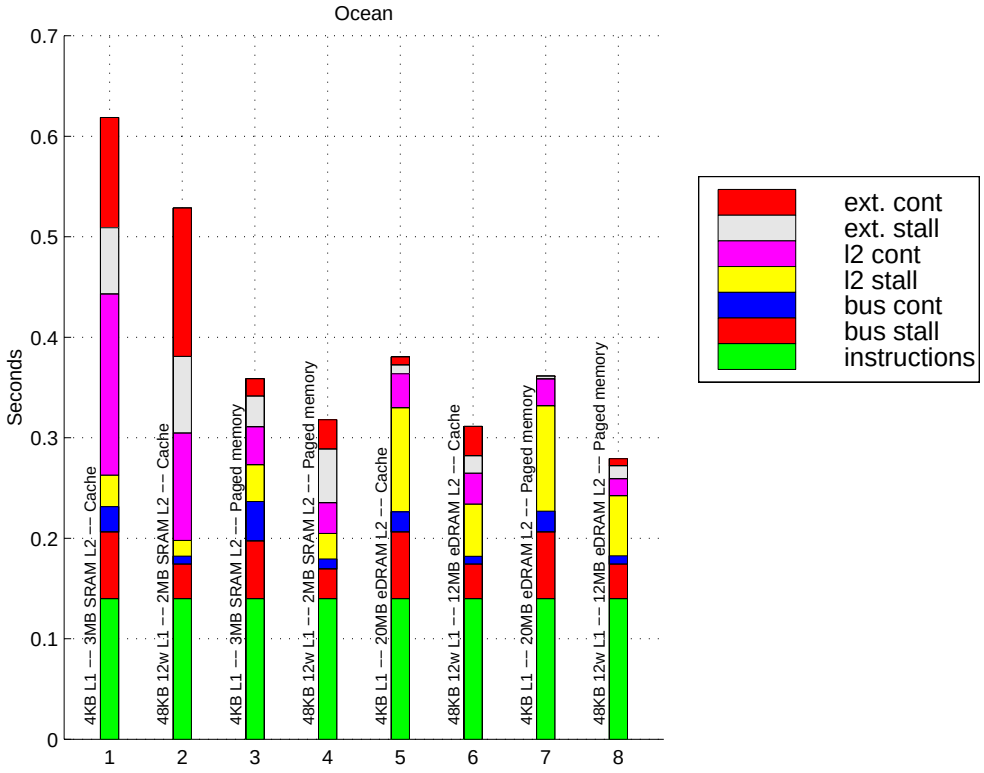

Figure 2: Ocean

The high level points to be taken from Figure 2 are (1) paged memory architectures perform better than cached memory architectures; (2) the best performing solution is bar 8, a 48 kilobyte L1 cache with a 12 megabyte EDRAM L2 cache; (3) large L1 caches are good but not great, even at the expense of L2 capacity. The following text elaborates and explains these high level conclusions.

The Instruction component is constant across all memory architectures. It represents the longest execution path among the four processors. The bus stall and contention are the first component that varies with the memory architecture. The smaller L1 has more misses, hence more accesses to the bus, hence more bus stall and contention. The second group of bars, L2 stall and contention, varies more depending on the specifics of the memory architecture. Most remarkable is the large contention component of bars 1 and 2, where the L2 is SRAM cache. This is due to (1) the single bank SRAM, (2) frequent L2 accesses from L1, and (3) frequent L2 misses causing L2 transfers to off-chip memory. The lower capacity of the SRAM cache results in the larger off-chip components, which are external memory stall and contention. If architectural changes were made to reduce the L2 contention, such as increasing the number of banks, the contention would shift to the bus (parallel accesses would contend for the bus), and to external memory contention.

An interesting comparison is the component breakdown comparing paged memory and cache memory for equivalent capacities. Bars 6 and 8 have the same technology and capacity, yet the paged organization gives better performance. The Instruction, Bus stall and Bus contention all have the same values, as expected. However L2 stall time is larger for bar 8, which is not immediately expected. If the L1 organization is the same, would the two not have the same number of access to the L2, and hence the same stall time? The paged memory system has more accesses because of the paging behavior. As mentioned in section 2.2 the processor must wait for data to be burst on and off chip, which will show up as L2 stall time. The external memory access is hidden (or vice-versa, if the external memory is faster), but the resource is still consumed and another parallel access to external memory may encounter contention.

The L2 contention, external stall and external contention times are all lower for the paged memory architecture for this application. All accesses to off-chip memory are 4 kilobyte transfers which are burst transfers. If the application makes good use of the data that is burst on-chip, and no longer needs the data burst off-chip, then the paged architecture offers superior performance. Similar comparisons can be done for the other architectures.

Figure 2 leads to several conclusions about this application. For Ocean, paged memory systems always outperform the equivalent capacity cached memory system. Ocean has a high “average touches per page” value at 3,800, meaning that for each page brought into the chip is going to have good usage. The best performing configuration is with a 48 kilobyte L1 cache with paged EDRAM (bar 8). Also, it is noted that EDRAM solutions outperform SRAM solutions, except for bar 7. This would indicate that L2 capacity is more important than latency. Finally, a larger L1 cache always outperforms a smaller L1 cache.

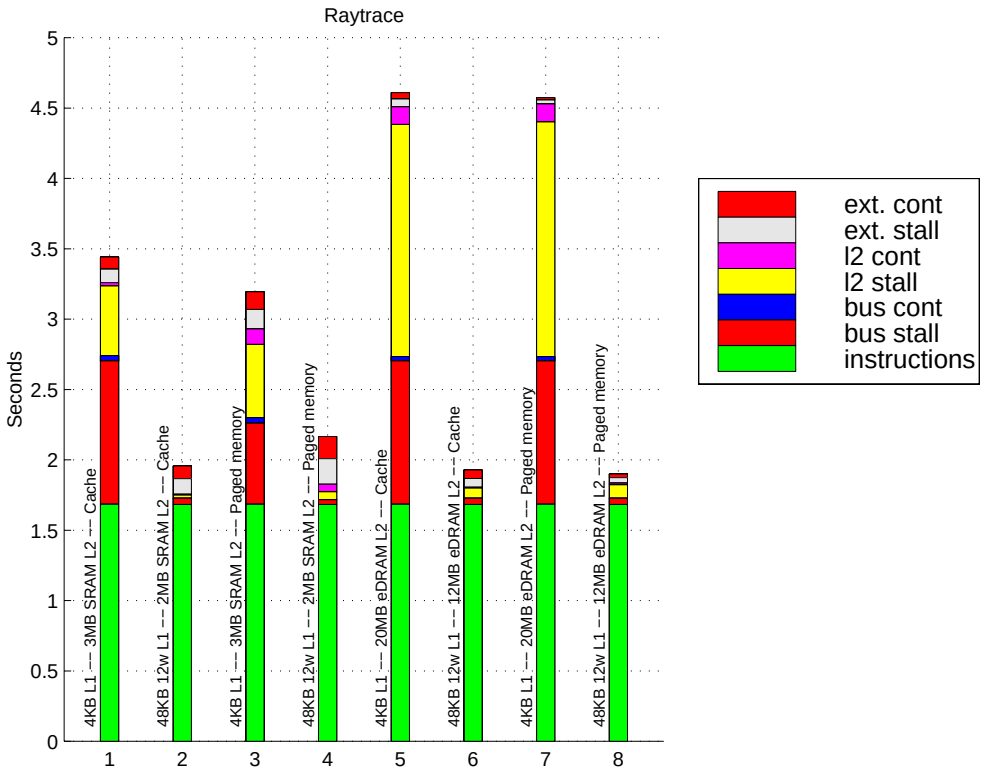

Figure 3: Raytrace

The next SPLASH application is Raytrace, shown in Figure 3. After some experimentation, the best performance was achieved by simulating cases 1 and 3 with a non-split transaction bus, and cases 4 and 6 with a split transaction bus. This application presents a much easier analysis than Ocean. Essentially, Raytrace requires a moderate L1 capacity. With a 48 kilobyte 12-way L1 cache, Raytrace is compute bound.

Looking more at the details, in comparing 4 kilobyte and 48 kilobyte L1 caches with equivalent L2 memory architectures, we see Raytrace showing no preference between paged and cached memory. Bars 2 and 4 have the same L2 capacity, and the cache organization has better performance. Yet, looking at bars 1 and 3, 4 and 6, and 5 and 7, the paged memory system is marginally better.

Among the architectures with 4 kilobyte L1 caches, those with faster SRAM L2 memory outperform those with slower EDRAM L2. The results strongly indicate a working set larger than 4 kilobytes yet smaller than 48 kilobytes for Raytrace. Minimizing the latency for this small working set maximizes the application's performance.

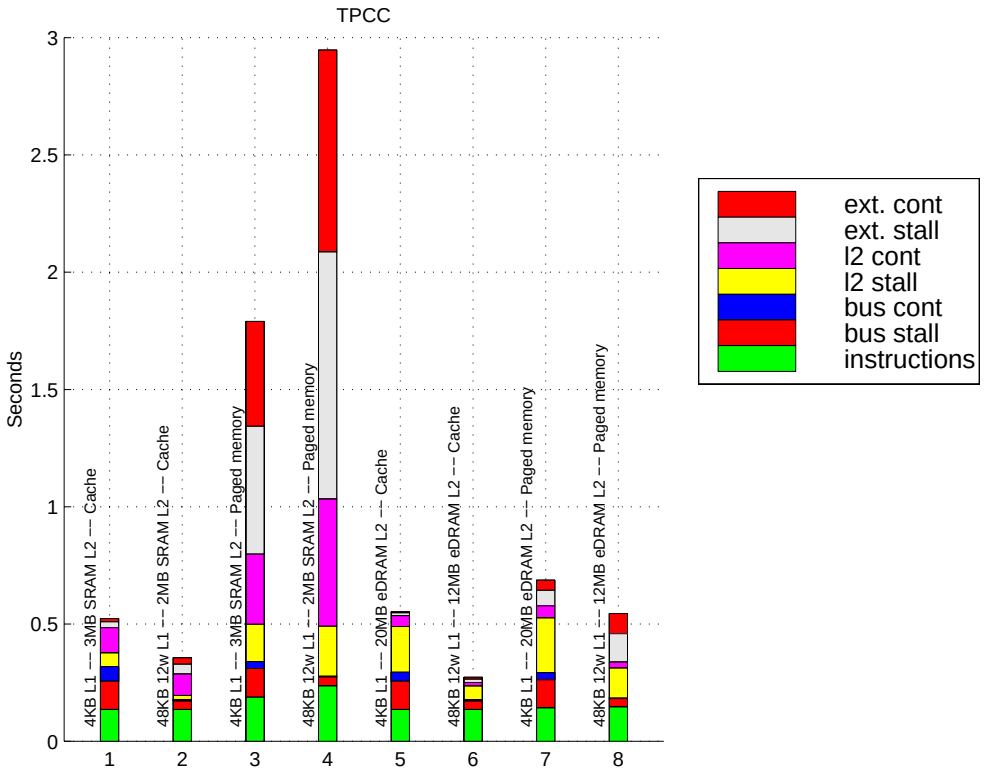

Figure 4: TPCC

TPCC, shown in Figure 4, presents a different picture. The high level observations from Figure 4 is that (1) cached memory is better than paged, and paged can be very bad; (2) Large L1 is better than small. Two architectures are clearly worse than the others, bars 3 and 4, which are the SRAM paged memory solutions. Looking at these two bars, they have more instruction execution time, more L2 contention, and more external stall and contention than the other memory organizations. The reason for the extra instruction execution time is the increased paging activity, which requires more instructions to find the replacement page and start the transfer[2]. The larger L1 cache of bar 4 (48 kilobytes) results in fewer cache misses, as seen in the lower bus stall numbers. However, the larger L1 means less L2, and the architecture represented in bar 3 can hold more pages. Bar 3 shows less L2 contention and less external memory usage; a result of less paging activity.

The two other paged memory solutions (bars 7 and 8) are worse than all other cache solutions as well. This is in contrast to Ocean, where the paged memory systems outperformed the cached memory systems. The "average number of touches per page" for TPCC is very low compared to Ocean, at 560. The best performing solution is bar 6, a large L1 cache with an EDRAM L2 cache. Given equal L1 capacities and equal memory technologies (SRAM vs. EDRAM), cache is substantially better.

Bars 5 and 7 have the same L2 capacity (20 megabytes) and technology (EDRAM), and have similar performance. The primary difference in performance between these two architectures is the extra external stall and contention of the paged memory system. The L2 stall numbers are slightly larger for the paged memory system, the same as for Ocean. Whereas Ocean made good use of memory bandwidth in transferring 4 kilobyte blocks at a time, TPCC makes better use of many smaller transactions.

[2] We add 50 instructions to the instruction count on a page transfer.

Of the architectures evaluated in this study, TPCC prefers high capacity with small granularity. With a better paging algorithm (we use FIFO for simplicity), the paged memory solutions may be more competitive with cache solutions for TPCC.

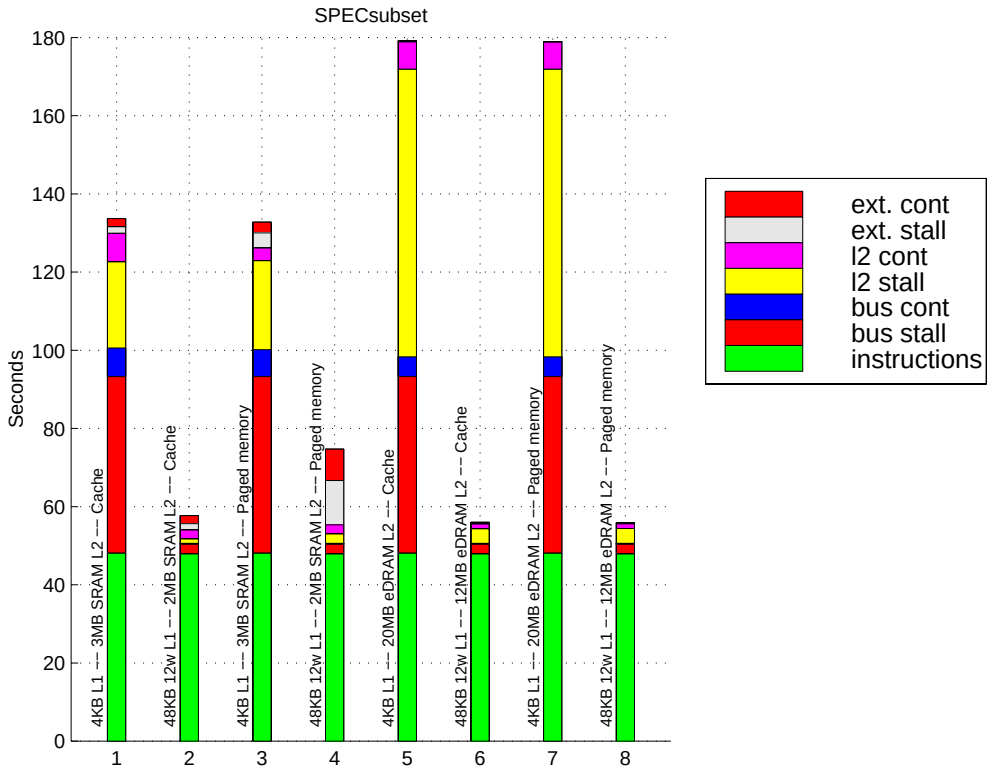

Figure 5: SPECsubset

The last application, or application suite, is made up of four applications from SPECINT2000, which are 164.gzip, 175.vpr, 186.crafty and 197.link with two instances of 164.gzip and 175.vpr. For this application suite, individual single processor traces were scheduled to completion using First Come First Serve to four processors. The performance data is shown in Figure 5. The conclusions here mirror Raytrace, that the SPECsubset suite is very dependent on the L1 cache. The behavior the L2 memory architectures when the L1 cache is 4 kilobytes is very similar to Raytrace. Bars 2, 6 and 8 have the same performance, and bar 4 shows slightly worse performance. For the 48 kilobyte 12-way L1 cache we see the same behavior as Raytrace. The two compute bound applications in this study (Raytrace and SPECsubset) prefer a fast L2 when given an insufficient L1.

Table 4 summarizes the high level points for each application.

	Favored Configuration (strong results in bold)			
Application	Paged vs. Cached	L1 working set	Large L1 or Large L2	SRAM vs. EDRAM
Ocean	Paged	> 48 kilobytes	Large L1	Don't care
Raytrace	Don't care	< 48 kilobytes	**Large L1**	SRAM
TPCC	Cached	>> 48 kilobytes	Large L1	Don't care
SPECsubset	Don't care	< 48 kilobytes	**Large L1**	Don't care

Table 4: Application Summary

6. Conclusions

We have compared eight different memory architectures in the CMP context: SRAM and EDRAM L2 configured as cache and pageable memory, all with two different L1 configurations. All of the evaluated architectures have approximately the same area. These experiments point up the difficulty in producing a single configuration for a general purpose microprocessor. The same machine is asked to perform optimally on a wide variety of different application, ranging from scientific and technical workloads, such as Ocean and Raytrace, to commercially crucial functions like TPCC.

The strongest result coming out of the experiment seems to be that larger L1 caches provide better performance. Unfortunately, the L1 cache tends to be a cycle-time limiter in modern processor designs. Thus, a larger L1 leads to more time spent on instruction execution, an effect that was not modeled in the experiments presented here.

Results comparing paged to cached memory show a strong preference for cached memory. The one exception, Ocean, is the only benchmark that emphasizes a dense regular data access pattern. Interestingly, this is the most likely access pattern for next-generation streaming multimedia applications. A natural extension of the work presented here would be to include such applications in the benchmark suite. We presented a new metric to help explain when paged memory systems outperform cached memory systems, the "average number of touches per page" as seen by the pageable level of memory, which in this case is the L2 memory. Applications with a higher "touches per page" metric had better performance on an L2 paged architecture compared to an L2 cache architecture.

Finally, we see little difference in overall performance when comparing smaller, faster SRAM L2 caches to much larger, somewhat slower embedded DRAM caches. The results here are at least consistent. Though the difference may be small, the larger EDRAM caches always perform better than the smaller SRAM caches. Offsetting the performance gain, of course, is the increased die cost for adding DRAM capability to what would otherwise be a straight CMOS process.

The final recommendation to be made, considering the outcome of this study, would be

1. Include the largest L1 cache that can fit in the processor cycle time, even at the expense of L2 capacity.
2. Larger, slower L2 memories outperform smaller, faster L2 memories, but not by much.
3. Tradeoffs between cached and paged memories are application dependent, although cached memory seems to perform better in the general case.

7. References

[Cri97] Richard Crisp. Direct Rambus Technology: The New Main Memory Standard. *IEEE Micro*, 17(6), November/December 1997.

[Dief99] Keith Diefendorff. Power4 Focuses on Memory Bandwidth. *Microprocessor Report*, 13(13), October 1999.

[Ham97] Lance Hammond, Basem A. Nayfeh and Kunle Olukotun. A Single-Chip Multiprocessor. *IEEE Computer Special Issue on "Billion-Transistor Processors"*, September 1997.

[Ham98] Lance Hammond and Kunle Olukotun. Considerations in the Design of Hydra: A Multiprocessor-on-a-Chip Microarchitecture. *Stanford University Technical Report CSL-TR-98-749*, February 1998.

[Hun94] R. Hundal and V. G. Oklobdzija. Determination of Optimal Sizes for a First and Second level SRAM-DRAM On-Chip Cache Combination. Proceedings *1994 IEEE Conference on Computer Design: VLSI in Computers and Processors*, October 1994.

[Ino97] Koji Inoue, Koji Kai and Kazuaki Murakami. Dynamically Variable Line-Size Cache Exploiting High On-Chip Memory Bandwidth of Merged DRAM/Logic LSIs. *Japan-Germany Forum on Information Technology*, November 1997.

[Ino99] K. Inoue, K. Kai and K. Murakami. Dynamically Variable Line-Size Cache Exploiting High On-Chip Memory Bandwidth of Merged DRAM/Logic LSIs. Proceedings of *HPCA-5. 5th International Conference on High Performance Computing*, Jan 1999.

[Joh97] Teresa Johnson, Matthew Merten and Wen-mei Hwu. Run-time Spatial Locality Detection and Optimization. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, December 1997.

[Kri99] Venkata Krishnan and Josep Torrellas. A Chip-Multiprocessor Architecture with Speculative Multithreading. *IEEE Transactions on Computers, Special Issue on Multithreaded Architecture*, September 1999.

[Mac99] Philip Machanick, Pierre Salverda and Lance Pompe. Hardware-Software Trade-offs in a Direct Rambus Implementation of the RAMpage Memory Hierarchy. 1999.

[MAT] Matlab, The MathWorks, Inc. Version 5.2.0.3084.

[Mul91] Johannes M. Mulder, Nhon T. Quach and Michael J. Flynn, "An Area Model for On-Chip Memories and its Application." IEEE Journal of Solid-State Circuits, Vol. 26, No. 2, February 1991.

[Mur97] Kazuaki Murakami, Koji Inoue and Hiroshi Miyajima. Parallel Processing RAM (PPRAM). *Kyushu University Technical Report PPRAM-TR-27*, November 1997.

[Nay94] Basem A. Nayfeh and Kunle Olukotun. Exploring the Design Space for a Shared-Cache Multiprocessor. In *Proceedings of the 21st International Symposium on Computer Architecture*, February 1994.

[Olu96] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson and Kun-Yung Chang. The Case for a Single-Chip Multiprocessor. In *Proceedings of the Seventh International Symposium on Architectural Support for Parallel Languages and Operating Systems*, October 1996.

[Olu99] Kunle Olukotun, Lance Hammond and Mark Willey. Improving the Performance of Speculatively Parallel Applications on the Hydra CMP. In *Proceedings of the 1999 ACM International Conference on Supercomputing*, June 1999.

[Ram98] Rajesh Raman, Miron Livny and Marvin Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Computing*, July 1998.

[Sing92] J.P. Singh, W. Weber, A. Gupta. Splash: Stanford Parallel Applications for Shared Memory. *Computer Architecture News*, 20(1), 1992.

[SPEC] Standard Performance Evaluation Company (SPEC) Web Page, "http://www.spec.org".

[TPCC98] TPC-C Description, Transaction Processing Performance Council Web Page, "http://www.tpc.org/faq_TPCC.html".

[Wils97] Kenneth Wilson and Kunle Olukotun. Designing High Bandwidth On-Chip Caches. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, June 1997.

[Wils99] Kenneth Wilson. The Use of DRAM in Processor Cache Design. *Stanford University Technical Report* ----------, October 1999.

[Woo95] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995.

[Yam97] Tadaaki Yamauchi, Lance Hammond and Kunle Olukotun. A Single Chip Multiprocessor Integrated with DRAM. *Stanford University Technical Report CSL-TR_97-731*, August 1997.