# CPACM: A New Embedded Memory Architecture Proposal

Paul Keltcher, Stephen Richardson
Computer Systems and Technology Laboratory
HP Laboratories Palo Alto
HPL-2000-153
November 21st, 2000*

eDRAM, embedded DRAM, demand paging, cache, CMP, CPACM, multiprocessor,

CPACM, or Combined Paged And Cached Memory, presents a new way to arrange the memory contained in very large caches. CPACM combines the favorable elements of two previous and more well-known schemes, merging the more conventional cached-memory method with demand paging. This new hybrid scheme consistently performs as well or better than its two predecessors on a wide variety of workloads. This study evaluates CPACM against benchmarks ranging from high end commercial and technical workloads down to very small embedded applications, such as imaging algorithms for printers.

* Internal Accession Date Only
Approved for External Publication

# CPACM: A New Embedded Memory Architecture Proposal

Paul Keltcher and Stephen Richardson

## Abstract

CPACM, or Combined Paged And Cached Memory, presents a new way to arrange the memory contained in very large caches. CPACM combines the favorable elements of two previous and more well-known schemes, merging the more conventional cached-memory method with demand paging. This new hybrid scheme consistently performs as well or better than its two predecessors on a wide variety of workloads. This study evaluates CPACM against benchmarks ranging from high end commercial and technical workloads down to very small embedded applications, such as imaging algorithms for printers.

## Introduction

In this paper we present CPACM, or Combined Paged And Cached Memory. In our exploration of applications for chip multiprocessors, we have proposed demand paging for the on-chip level two memory, and compared demand paging with more traditional cache replacement methods [Kelt00]. In that study we found some cases where demand paging performed better than cache, but also some cases where demand paging utterly failed. CPACM combines favorable elements of both methods in an attempt to provide consistently better performance across a variety of workloads. In this study we compare the three memory architectures: demand paging, CPACM and cache.

To better understand how results scale for different chip sizes, this study targets two different cost points: one for larger systems and one for smaller, embedded systems. The larger system targets a higher cost point, and its consequently larger die area lets it support more level-two cache memory than the smaller system. The smaller system targets a lower cost point and would be used in smaller workstations and embedded systems. Therefore, the smaller system will be evaluated against a different set of workloads than the larger system.

Consider a computer architecture consisting of one or more processors, each of which may have one or more small local memory caches. The processors connect to a large, fast store of local memory backed by a larger, slower remote memory, as shown in Figure 1.

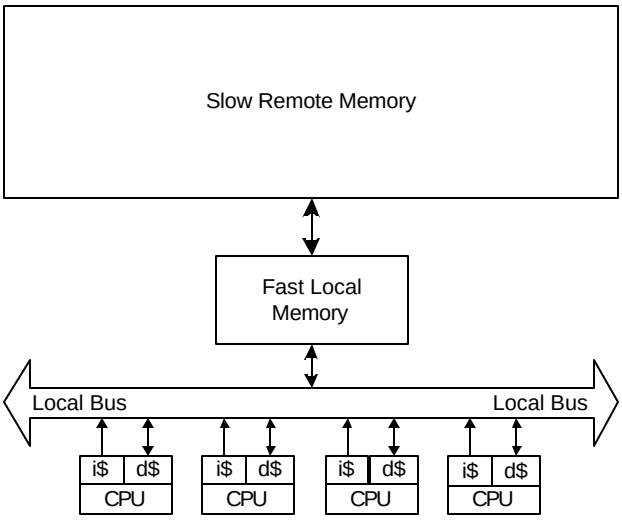


**Figure 1: Several processors, each with local cache, connected to local memory backed by larger remote memory.**

One way to manage the local memory would be as a traditional cache, organized as one or more sets of data lines, as shown in Figure 2. Each time a processor needs a data object, it checks the cache for the existence of a line containing the object. If the line is not there, the cache chooses a victim line. If the victim is dirty, the cache writes it out to remote memory, replacing it with the line from remote memory that holds the desired data object. This approach requires the cache to associate an address tag with each data line it holds, so it can tell an inquiring processor whether the line in cache matches the address of a requested data object. This tag can represent a daunting overhead. For instance, in a 64-bit processor, a 4 megabyte direct mapped cache of 32 byte lines might have 42 bits of tag overhead for every 32 byte line, or 16.4 percent extra memory.

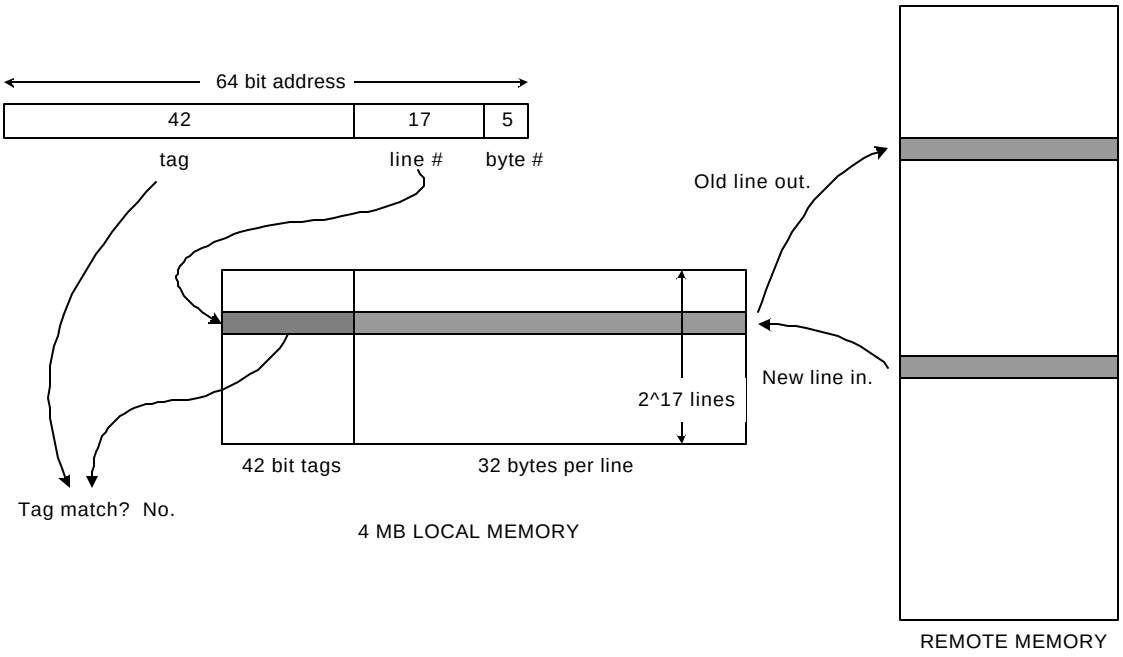


**Figure 2: Traditional cache scheme: on tag mismatch, old line must be replaced by new line.**

*Demand Paging* is another way to manage the local memory. This scheme treats the local memory as part of a unified physical address space that includes both local and remote memory. For example, the local memory might be assigned physical addresses 0 through `0x1fffff` while remote memory might be assigned physical addresses `0x200000` through `0xffffff`. Given such a layout, several schemes are possible to help minimize average memory latency. For instance, in a system without virtual memory translation, a program might be compiled and linked such that its most commonly used addresses reside in local memory, while less frequently used addresses live in remote memory.

Where virtual memory translation exists, the operating system can dynamically decide whether to map a given virtual page to local memory or to remote memory. The map can change on the fly and pages can migrate into and out of local memory from remote memory whenever the operating system chooses. Most probably the page migration would be triggered by some specific event such as a page fault from the translation lookaside buffer. In general, when a new page comes in from remote memory, some victim page must first be replaced in the local memory. If the victim is dirty, it must be written out to remote memory before the new page can come in. This scheme is similar to what one finds in a traditional non-uniform memory architecture (NUMA) or an I/O cache. Demand paging is different from a traditional cache scheme in that each cache line is the size of a memory page, and the tags reside in the TLB and in the data structures of the OS.

Figure 3 illustrates a scheme whereby the operating system dynamically re-maps local pages to keep access times low. In this step by step example, 1) the CPU wants a data object from virtual page VP=`fffc`. 2) Not finding VP in the TLB, the CPU receives a page fault. 3) The Operating System now chooses a victim entry in the TLB and a victim local memory page from the Page Table. In our example, the OS has chosen local memory page `10040` as its victim. 4) This page gets evicted to its current remote home page `110c7`. 5) Consulting the Page Table, the OS now brings in remote page RP=`11008` corresponding to `fffc`. 6) Finally, the OS updates the TLB and the Page Table to reflect the new mappings.

An advantage to the unified physical address scheme is the elimination of the tag overhead in the local memory. A down side is the fact that it writes out an entire page on dirty-page cast out, regardless of how many lines in the page were actually dirty. Another down side is the fact that the local memory then brings in an entire page of data, regardless of how many lines actually get used. To help reduce the magnitude of these problems, we introduce CPACM, or Combined Paged And Cached Memory.

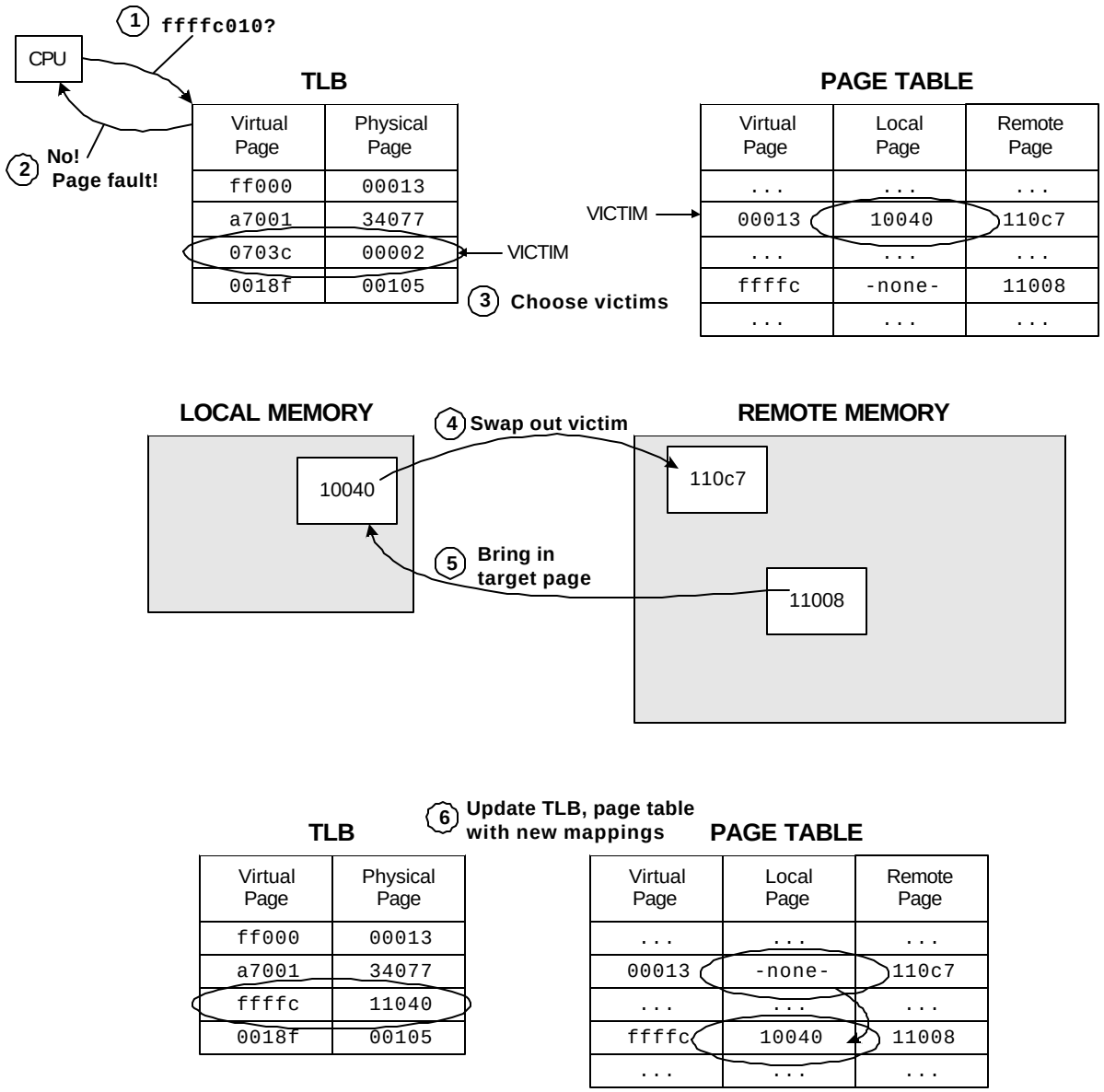


**Figure 3: Operating system manages local and remote page mappings using TLB and Page Table.**

CPACM reduces the Write Wait associated with writing out a dirty page from local to remote memory and the Read Wait associated with reading a new page in to local memory from remote memory. CPACM associates a valid bit and a dirty bit with each line of each page in local memory. Each time a processor writes to a line in local memory, it also sets the dirty bit associated with that line. When the operating system wants to bring a new page into local memory, it first chooses a victim page as usual. Instead of writing out the entire victim page, however, it writes out only those lines whose dirty bit has been set. If adjacent lines are dirty, then the memory controller can burst out the longer data stream. We assume that the memory controller can intelligently burst out arbitrary lengths of adjacent dirty lines.

The operating system then clears the valid bits for all lines in the cache, bringing in and marking as valid *only the missing data line*. Finally, the operating system updates its page table (and the TLB) to reflect the new virtual to physical mapping. When a processor needs a data object from the new page, it first checks the valid bit for that line, bringing it in from remote memory if needed.

The CPACM scheme is similar to the technique of subblocking in ordinary caches, where each block has a tag and each subblock has its own valid and/or dirty bit. By making the block the same size as a page and using the existing virtual memory system, however, CPACM avoids the overhead of per-block tags in the memory. Furthermore, by using its ability to map any given virtual page to any given physical page in local memory, CPACM behaves like a fully-associative cache while retaining the speed benefits of having a direct-mapped cache. CPACM has an additional advantage over subblocked caches in that specific pages can be pinned by the OS to be always resident in memory. Embedded systems would make use of the guaranteed latency of resident pages.

## Related Work

Other research has considered multiple processors on a chip [Barr00, Dief99, IBM, Hamm98, Mura97], and using demand paging for some level of the memory hierarchy [Yama97, Mach99]. In [Saul99] comparisons are made between eDRAM as cache and static page allocation, but not dynamic paging. This study compares CPACM to standard cache and demand paging. For further related work references see [Kelt00].

## Performance Analysis

The applications used in this performance study are targeted to either a small memory system, based on a processor with small chip area, or a large system, based on a processor chip with large area. Both the small and large memory systems start with 4 kilobytes of level one cache (L1), but have different sizes for the on-chip level two cache (L2). For simplicity, both L1 and L2 caches have 32-byte lines. The small system has 1 megabyte of L2 memory and the large system has 4 megabytes. Both large- and small-system L2's are 4-way set associative, accessible in 4ns. Because of the tag overhead associated with caches, the demand paged and CPACM memory solutions have somewhat larger memory capacity. For the small-memory system, this means that demand paged and CPACM have 1152 kilobytes of memory, versus 1024 kilobytes for the memory arranged as traditional cache. The corresponding sizes for the large-area system are 4096 and 4608 kilobytes, respectively. Note that, because demand paged and CPACM both use software interrupts to manage the cache, it becomes feasible to use sizes that are not powers of two.

For the small area system the applications are Boise IP and JPEG decode. For the large area systems the applications are Boise IP, Ocean and TPCC. Boise IP is a large C++ application with a 24 megabyte data set. This image-processing application divides an image into 4 megabyte “strips” which can be processed in parallel. Similarly, JPEG decode uses an initial sequential code segment to break the image into blocks that can be then processed in parallel. Ocean is from the SPLASH benchmark suite [Sing92, Woo95]. TPCC is a snapshot of the TPCC benchmark [TPCC98]. Table 1 presents basic characteristics of the applications. Table 2 summarizes the small and large memory systems, and tells which applications pertain to each.

| Application | Millions of Instructions | Millions of Data References | 4 kilobyte L1 miss rates | | Number of 4 kilobyte pages |
|---|---|---|---|---|---|
| | | | Instruction | Data | |
| Boise IP | 390 | 110 | 0.6 % | 7.9 % | 1,000 |
| JPEG Decode | 57 | 17 | 7.3 % | 7.0 % | 1,400 |
| Ocean | 550 | 180 | 0.1 % | 7.4 % | 3,800 |
| TPCC | 540 | 74 | 8.9 % | 5.2 % | 30,400 |

**Table 1: Application Summary**

| **Chip size** | **Small** | **Large** |
|---|---|---|
| **L1 access time**<br>**L1 size** | 1 ns<br>4 KB | 1 ns<br>4 KB |
| **L2 access time**<br>**L2 size, normal cache**<br>**L2 size, DP and CPACM** | 4 ns<br>1024 KB<br>1152 KB | 4 ns<br>4096 KB<br>4608 KB |
| **External memory speed**<br>**Bus width** | 100 MHz SDRAM<br>8 bytes | 100 MHz SDRAM<br>8 bytes |
| **Applications** | Boise IP<br>JPEG decode | Boise IP<br>Ocean<br>TPCC |

**Table 2: Memory Summary**

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, however we do account for the additional OS overhead of demand paging and CPACM. This is done by simply adding 50 instructions per data page transfer between on and off chip DRAM. The trace-driven simulator runs on multiple host types (HPUX and Linux), and the job queue is managed by Condor [Rama98]. All graphs are generated using auto-generated Matlab [MAT] scripts. In this section each application will be analyzed with conclusions to follow.

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 to 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 bus 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 four-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 a previous write from the issuing processor.
- External memory stall. The external memory is accessed on an L2 read miss.
- 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 4 shows the results for the small area (one megabyte L2) Boise IP simulation. The *y* axis measures execution time in seconds. Along the *x* axis are four groups of three bars, where each group has simulation results for the given number of processors. Each group shows the three L2 memory architectures: demand paging, CPACM and cache. The individual bars contain each of the performance criteria mentioned earlier: Instruction execution time, bus stall, bus contention, L2 stall, L2 contention, external memory stall and external memory contention.

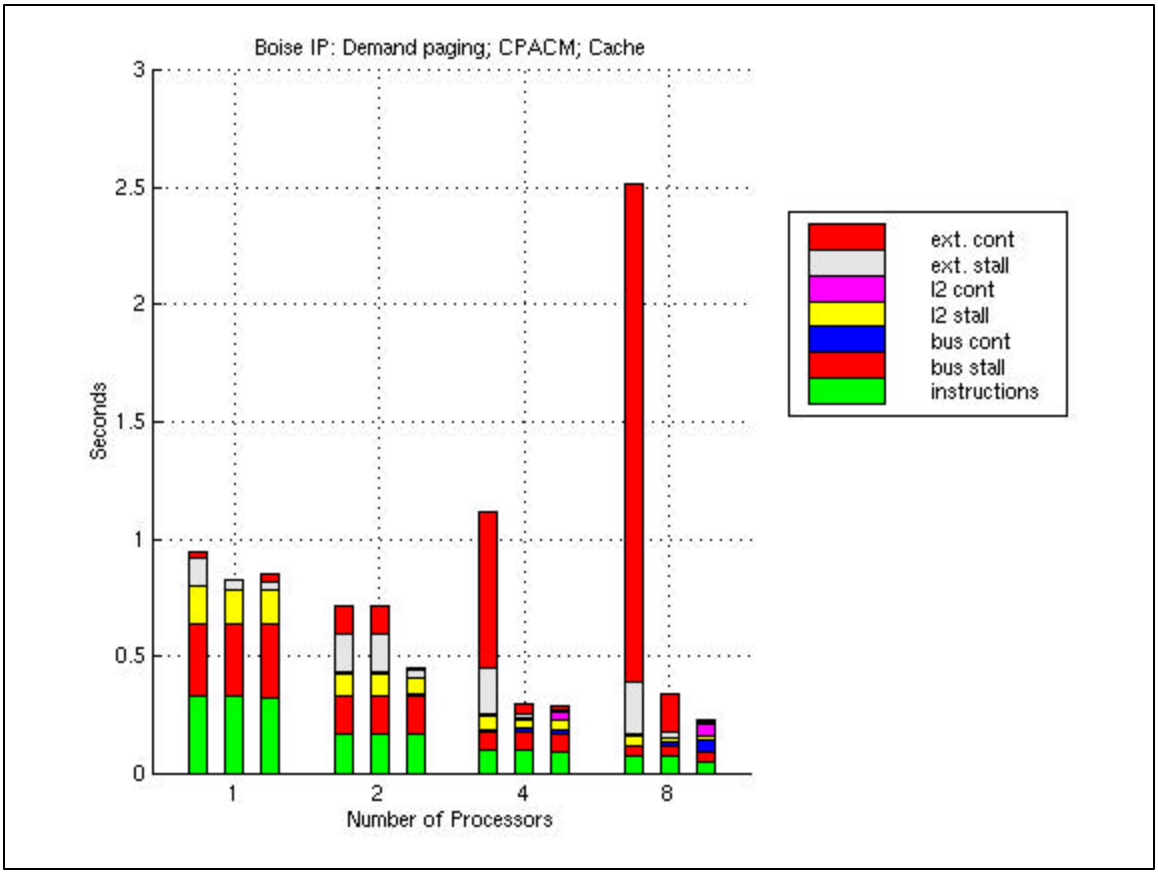


**Figure 4: Boise IP with small L2**

Figure 4 clearly shows that demand paging does not behave well with L2 memory pressure. As the number of processors increases to four and eight, the demand paging solution breaks down entirely. CPACM also has its memory pressure problems, but to a much lesser degree, and it only becomes an issue when moving from four processors to eight processors. The cache model behaves well even under the stress of eight processors, with eight data sets competing for the L2 memory at the same time. The OS overhead is minimal, and only noticeable with eight processors. This data would suggest that (1) CPACM can have the best performance if the L2 has sufficient capacity, and (2) even though CPACM is a paged memory system, it does not fail as demand paging can.

At eight processors, the demand paged scheme imports and exports pages faster than they can be used, thrashing the memory system and destroying performance. The CPACM scheme ameliorates this effect by bringing in only needed lines and casting out only dirty lines, bringing the performance in closer to that of a more traditional cache scheme.

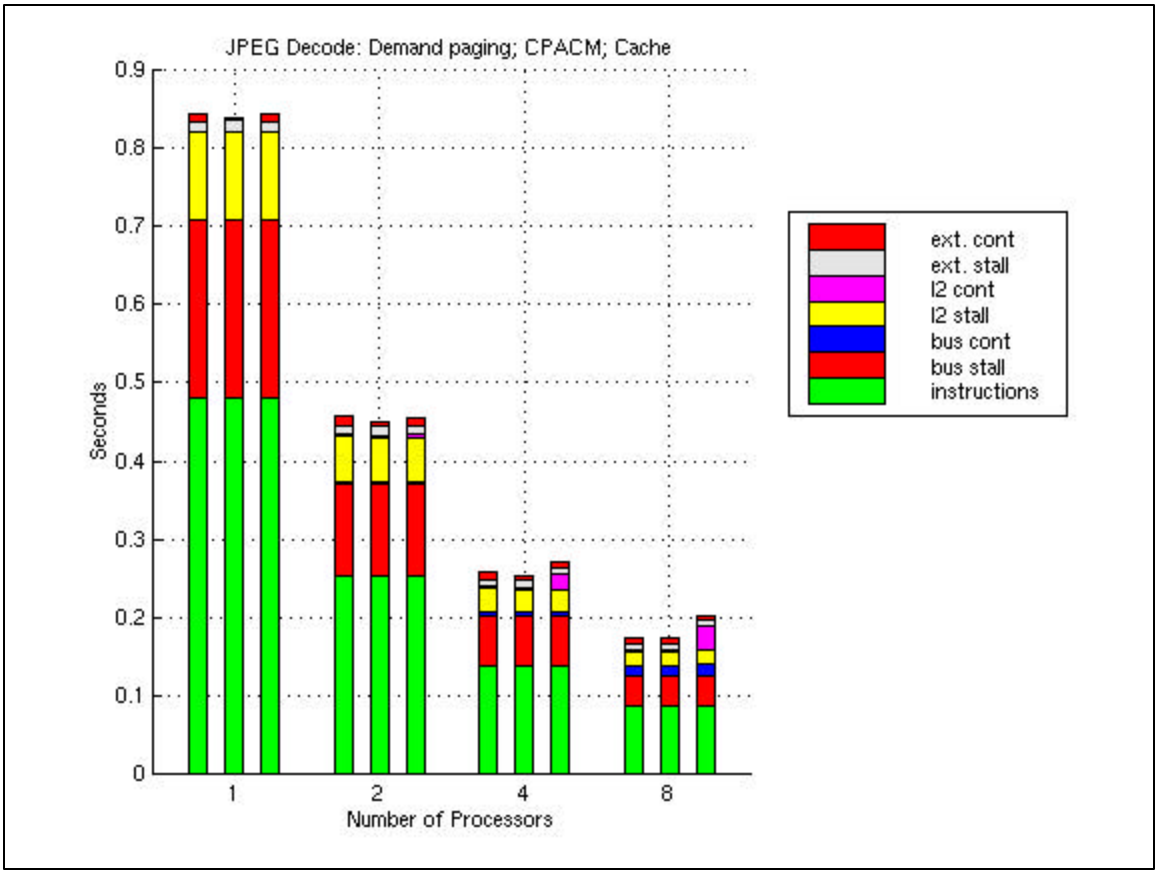


**Figure 5: JPEG Decode with small L2**

Figure 5 shows the other application for the small memory system, JPEG Decode. This is more of a compute bound application. In this application, demand paging and CPACM offer the same performance, which is slightly better than what the cache model offers. The cache model gets increased L2 contention for the four and eight processor case. On an L2 castout, both the L2 and external memories are immediately consumed for the transfer (there are no write buffers and no memory reordering). L2 contention is recorded on every castout. Both CPACM and demand paging have the same rules, but have much lower L2 and external memory contention.

Here, CPACM performs better than either of its rivals, successfully making use of the best elements of both the other schemes: 1) the beneficial write burst nature of demand paging and 2) the demand read efficiency of the cache scheme.

The next set of graphs concern the large L2 memory (four megabytes). The first application is Boise IP again, shown in Figure 6. In this larger system, the memory pressure is less significant. As was hypothesized in the discussion of the small-memory Boise IP, CPACM takes advantage of the large system's reduced memory pressure. There is an interesting transition going from two to four processors where CPACM starts to outperform the cache model. The cache model suffers from increased bus and L2 contention as the number of processors increases. Again, CPACM has managed to strike an effective balance between the benefits and costs of caching and demand paging.

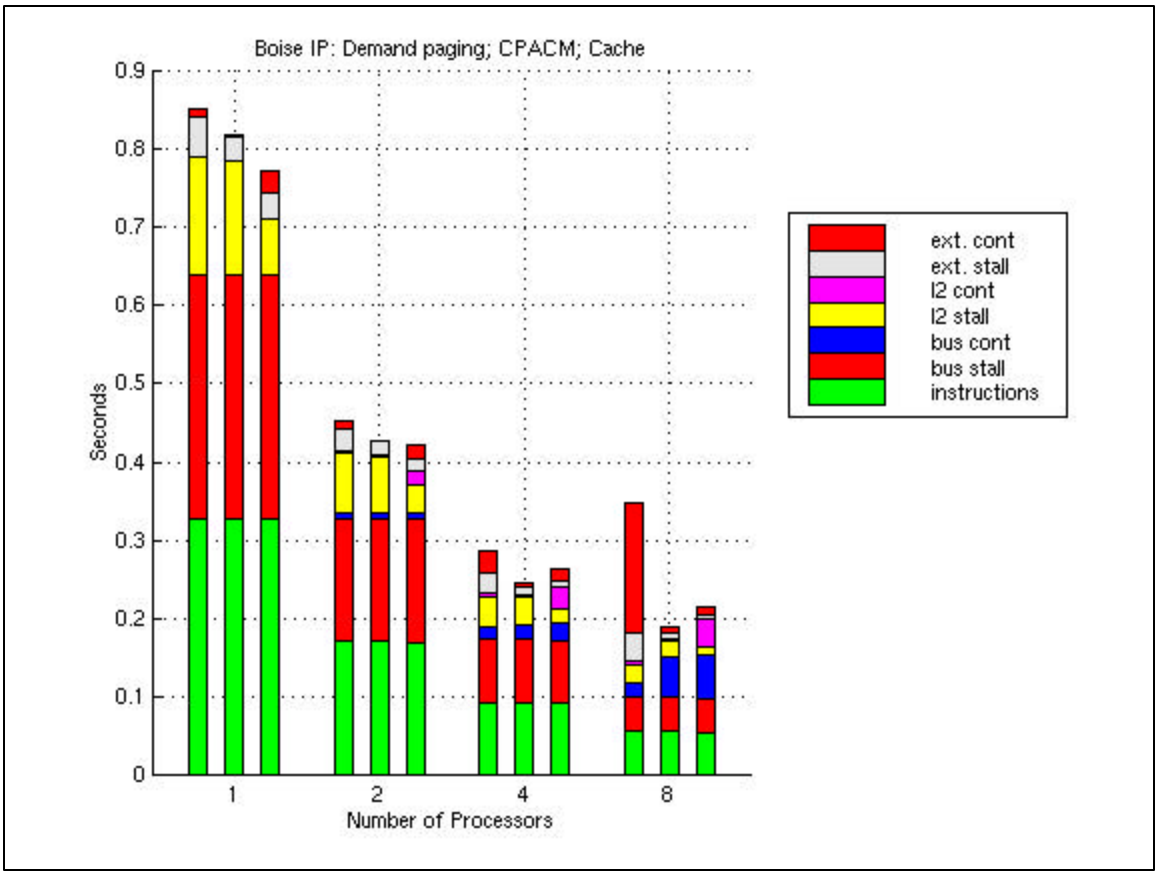


**Figure 6: Boise IP with large L2**

Figure 7 shows results for Ocean. Here, demand paging and CPACM closely contend for the best performance. The caching model does not perform well. Ocean uses a lot of data per page, as can be seen by the large Average Write metric (551 byte bursts out of 4096 byte pages in the single processor case---see Appendix A). Thus the bursting behavior of Demand paging and CPACM are beneficial. Demand paging outperforms CPACM in the single and eight processor cases.

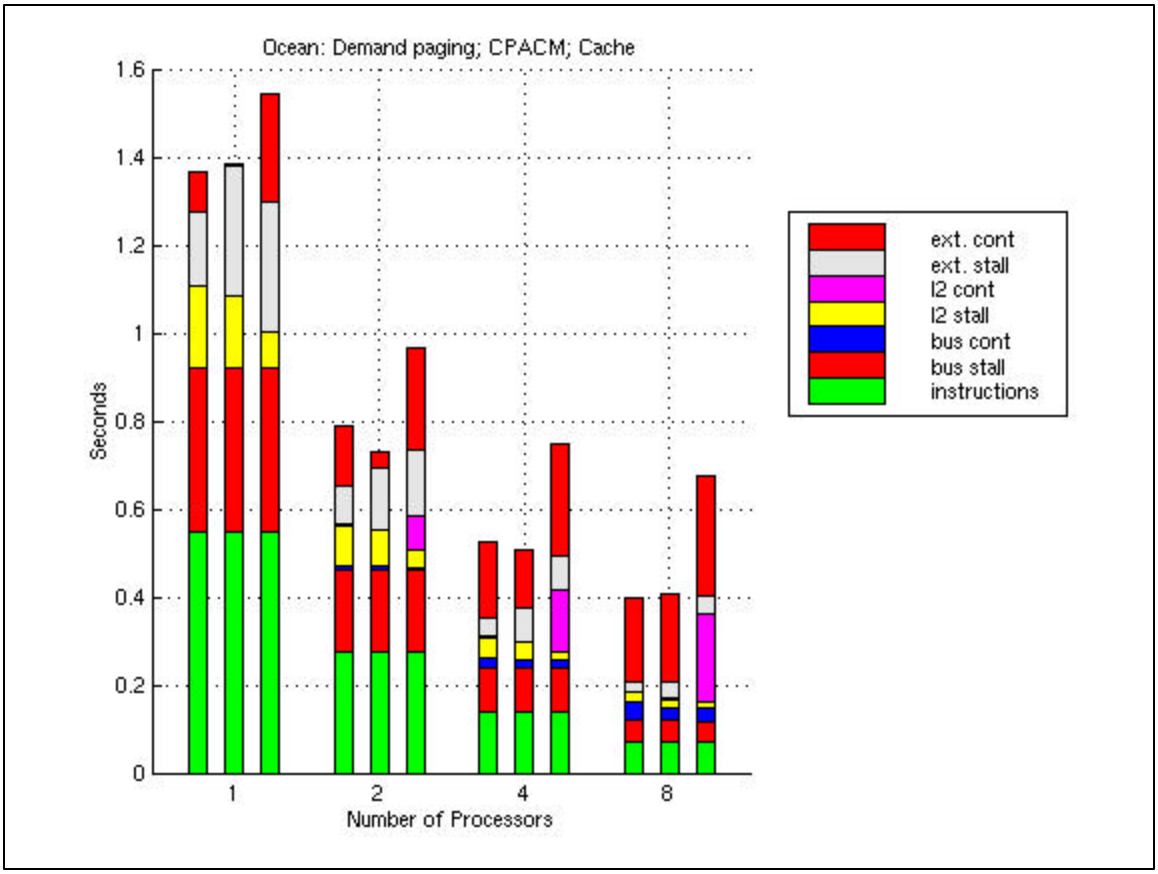


**Figure 7: Ocean with large L2**

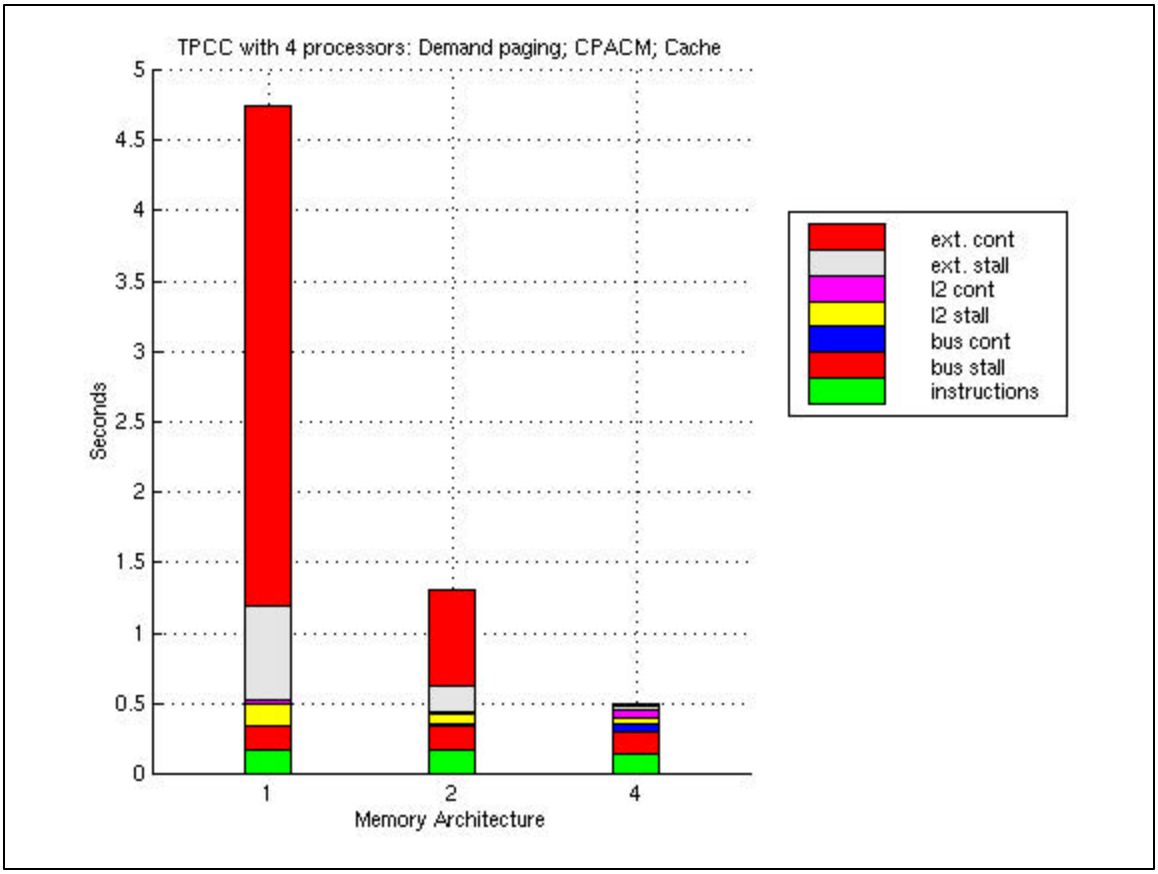


**Figure 8: Four Processor TPCC with large L2**

Lastly, Figure 8 shows the data for a four processor run of TPCC (one, two and eight processor traces were unavailable). The bars from left to right represent demand paging, CPACM and cache. For this application the cache scheme performs the best, while demand paging has serious problems. As with the small- and large-memory Boise IP, CPACM does not follow the failure pattern established by demand paging. Even at four megabytes, the memory pressure is heavy for this benchmark, resulting in a large performance discrepancy between CPACM and the cache model.

# Conclusions

CPACM combines the low read latency advantages of caching and the write burst advantages of paged memory. CPACM has the disadvantages of caching by not utilizing read bursts, if all the data is to be used, and the disadvantage of paged memory block contention. From our analysis the tradeoff is a good one, as CPACM offers good to best case performance in comparison to caching and paged memory for most applications.

The performance of paged memory might be improved by critical line first hardware, but further improvements like a write back buffer is probably not a good idea: a write back buffer for page writes would be prohibitively large (at least four kilobytes in length). Caching might be improved with write back buffers, an option that was not evaluated here. CPACM would benefit neither from critical line first nor from write back buffers. Future research includes examining CPACM for off-chip memories.

In each benchmark studied, we see that CPACM performs near to or better than the best of the other two competing systems. By contrast, each of the other two systems shows worst-case performance on one or more benchmarks.

# Appendix A

Although the best metric is runtime, we also present a few other statistics including *page probability*, *line probability*, *average transfer* and *average write transfer*. The first metric indicates if the requested 4096 byte page is present in the L2 memory. The second metric, *line probability*, is the "hit rate" for a cache line request from the L1. The next two metrics, *average transfer* and *average write transfer* are the number of bytes transferred between the L2 and the external memory for all transfers, and just write transfers (L2 to external memory). The Tables in this appendix correspond to bar charts presented earlier in the paper.

| Number of Processors | Page Probability | Line Probability | Average Transfer (B) | Average Write Transfer (B) |
|---|---|---|---|---|
| 1 | 99.9% | 98.3% | 36.8 | 161.6 |
| 2 | 99.8% | 97.7% | 40.3 | 122.2 |
| 4 | 99.5% | 96.2% | 44.3 | 94.6 |
| 8 | 99.0% | 93.0% | 46.8 | 82.2 |

**Table A1: Boise IP (small) stats.**

| Number of Processors | Page Probability | Line Probability | Average Transfer (B) | Average Write Transfer (B) |
|---|---|---|---|---|
| 1 | 99.9% | 98.6% | 36.8 | 147.0 |
| 2 | 99.9% | 98.1% | 36.7 | 137.6 |
| 4 | 99.9% | 97.5% | 36.8 | 138.4 |
| 8 | 99.9% | 97.0% | 38.2 | 136.3 |

**Table A2: JPEG Decode stats**

A high average write transfer, like in Ocean and to a less extent JPEG, indicates high page utilization, and the effect is good memory and memory bus utilization. Referring back to Figures 5 (JPEG) and 7 (Ocean), both demand paging and CPACM perform well. Contention for L2 and external memory, as seen in the performance graphs, is kept down by high write bursting (Average Write Transfer) and read on demand. A high Average Transfer may indicate more castouts, and explains why this number increases as the number of processors increases. This is indicative of worse performance, and higher L2 and external memory contention. For example, TPCC (Table A5) has a very high Average Transfer, meaning a larger percentage of transactions are write bursts, which brings down both the write burst number (less time between writes to fill up a page) and increases the number of transactions.

| Number of Processors | Page Probability | Line Probability | Average Transfer (B) | Average Write Transfer (B) |
|---|---|---|---|---|
| 1 | 99.9% | 98.6% | 33.8 | 189.1 |
| 2 | 99.9% | 98.6% | 33.7 | 169.1 |
| 4 | 99.9% | 98.5% | 34.7 | 141.3 |
| 8 | 99.8% | 98.1% | 37.7 | 108.2 |

**Table A3: Boise IP (large) stats**

| Number of Processors | Page Probability | Line Probability | Average Transfer (B) | Average Write Transfer (B) |
|---|---|---|---|---|
| 1 | 99.9% | 90.5% | 32.4 | 551.2 |
| 2 | 99.9% | 91.0% | 32.6 | 513.5 |
| 4 | 99.9% | 90.8% | 32.5 | 393.7 |
| 8 | 99.9% | 90.6% | 32.5 | 292.8 |

**Table A4: Ocean stats**

| Number of Processors | Page Probability | Line Probability | Average Transfer (B) | Average Write Transfer (B) |
|---|---|---|---|---|
| 4 | 99.3% | 86.7% | 53.8 | 179.1 |

**Table A5: TPCC stats**

# References


[Barr00] L.A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets and B. Verghese. "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing", in Proceedings of the 27th International Symposium on Computer Architecture, June 2000.

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

[Hamm98] 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.

[IBM] Blue Gene Architecture: http://www.research.ibm.com/news/detail/architecture_fact.html

[Kelt00] Paul Keltcher, Stephen Richardson and Stuart Siu. "An Equal Area Comparison of Embedded DRAM and SRAM Memory Architecture for a Chip Multiprocessor", HP Labs Technical Report HPL-2000-53.

[Mach99] 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.

[Mura97] Kazuaki Murakami, Koji Inoue and Hiroshi Miyajima. "Parallel Processing RAM (PPRAM)", Kyushu University Technical Report PPRAM-TR-27, November 1997.

[Rama98] 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.

[Saul99] Ashley Saulsbury, Su-Jaen Huang and Fredrik Dahlgren. "Efficient Management of Memory Hierarchies in Embedded DRAM systems", in Proceedings of the 1999 International Conference on Supercomputing, June 1999.

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

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

[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.

[Yama97] Tadaaki Yamauchi, Lance Hammond and Kunle Olukotun. "A Single Chip Multiprocessor Integrated with DRAM", Stanford University Technical Report CSL-TR-97-731, August 1997.