Missing the Memory Wall: The Case for Processor/Memory Integration

Ashley Saulsbury†, Fong Pong, Andreas Nowatzyk

Sun Microsystems Computer Corporation
†Swedish Institute of Computer Science
e-mail: ans@sics.se, agn@acm.org

Abstract

Current high performance computer systems use complex, large superscalar CPUs that interface to the main memory through a hierarchy of caches and interconnect systems. These CPU-centric designs invest a lot of power and chip area to bridge the widening gap between CPU and main memory speeds. Yet, many large applications do not operate well on these systems and are limited by the memory subsystem performance.

This paper argues for an integrated system approach that uses less-powerful CPUs that are tightly integrated with advanced memory technologies to build competitive systems with greatly reduced cost and complexity. Based on a design study using the next generation 0.25µm, 256Mbit dynamic random-access memory (DRAM) process and on the analysis of existing machines, we show that processor memory integration can be used to build competitive, scalable and cost-effective MP systems.

We present results from execution driven uni- and multi-processor simulations showing that the benefits of lower latency and higher bandwidth can compensate for the restrictions on the size and complexity of the integrated processor. In this system, small direct mapped instruction caches with long lines are very effective, as are column buffer data caches augmented with a victim cache.

1 Introduction

Traditionally, the development of processor and memory devices has proceeded independently. Advances in process technology, circuit design, and processor architecture have led to a near-exponential increase in processor speed and memory capacity. However, memory latencies have not improved as dramatically, and access times are increasingly limiting system performance, a phenomenon known as *the Memory Wall* [1] [2]. This problem is commonly addressed by adding several levels of cache to the memory system so that small, high speed, static random-access-memory (SRAM) devices feed a superscalar microprocessor at low latencies. Combined with latency hiding techniques such as prefetching and proper code scheduling it is possible to run a high performance processor at reasonable efficiencies, for applications with enough locality for the caches.

The approach outlined above is used in high-end systems of all the mainstream microprocessor architectures. While achieving impressive performance on applications that fit nicely into their caches, such as the Spec'92 [3] benchmarks, these platforms have become increasingly application sensitive. Large applications such as CAD programs, databases or scientific applications often fail to meet CPU-speed based expectations by a wide margin.

To appear in the proceedings of the
23rd annual International Symposium on Computer Architecture, June 1996.

The CPU-centric design philosophy has led to very complex superscalar processors with deep pipelines. Much of this complexity, for example out-of-order execution and register scoreboarding, is devoted to hiding memory system latency. Moreover, high-end microprocessors demand a large amount of support logic in terms of caches, controllers and data paths. Not including I/O, a state-of-the-art 10M transistor CPU chip may need a dozen large, hot and expensive support chips for cache memory, cache controller, data path, and memory controller to talk to main memory. This adds considerable cost, power dissipation, and design complexity. To fully utilize this heavy-weight processor, a large memory system is required.

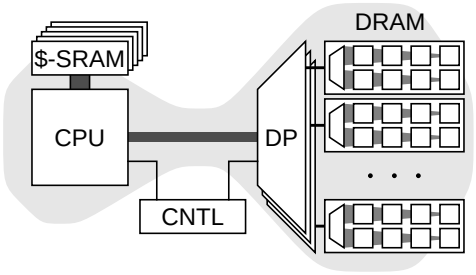


FIGURE 1 : Compute System Components

The effect of this design is to create a bottleneck, increasing the distance between the CPU and memory — depicted in Figure 1. It adds interfaces and chip boundaries, which reduce the available memory bandwidth due to packaging and connection constraints; only a small fraction of the internal bandwidth of a DRAM device is accessible externally.

We shall show that integrating the processor with the memory device avoids most of the problems of the CPU-centric design approach and can offer a number of advantages that effectively compensate for the technological limitations of a single chip design.

2 Background

The relatively good performance of Sun's Sparc-Station 5 workstation (SS-5), with respect to contemporary high-end models, provides evidence for the benefits of tighter memory-processor integration.

Targeted at the "low-end" of the architecture spectrum, the SS-5 contains a single-scalar MicroSparc CPU with single-level, small, on-chip caches (16KByte instruction, 8KByte data). For machine simplicity the memory controller was integrated into the CPU, so the DRAM devices are driven directly by logic on the processor chip. A separate I/O-bus connects the CPU with peripheral devices, which can access memory only through the CPU chip.

A comparable "high-end" machine of the same era is the Sparc-Station 10/61 (SS-10/61), containing a super-scalar SuperSparc CPU with two cache levels; separate 20KB instruction and 16KB data caches at level 1, and a shared 1MByte of cache at level 2.

Compared to the SS-10/61, the SS-5 has an inferior Spec'92-rating, yet, as shown in Table 1, it out-performs the SS-10/61 on a logic synthesis workload (Synopsys[1] [4]) that has a working set of over 50 Mbytes.

Machine	Spec'92 Int	Spec'92 Fp	Synopsys Run Time
SS-5	64	54.6	32 minutes
SS-10/61	89	103	44 minutes

TABLE 1 : SS-5 vs. SS-10 Synopsis Performance

The reason for this discrepancy is the lower main memory latency of the SS-5, which can compensate for the "slower" CPU. Figure 2 exposes the memory access times for the levels of the cache hierarchy by walking various-sized memory arrays with different stride lengths. Codes that frequently miss the SS-10's large level-2 cache will see lower access time on the SS-5.

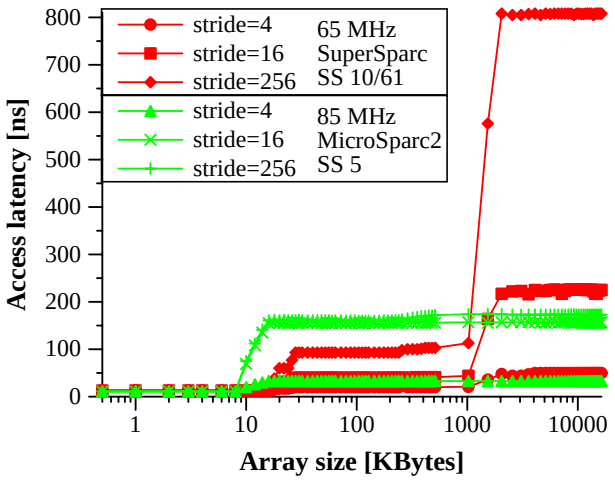


FIGURE 2 : SS-5 vs. SS-10 Latencies[2]

The "Memory Wall" is perhaps the first of a number of impending hurdles that, in the not-too-distant future, will impinge upon the rapid growth in uniprocessor performance. The pressure to seek further performance through multiprocessor and other forms of parallelism will increase, but these solutions must also address memory sub-system performance.

Forthcoming integration technologies can address these problems by allowing the fabrication of a large memory, processor, shared memory controller and interconnection controller together on the same device. This paper presents and evaluates a proposal for such a device.

3 Technology Characteristics and Trends

The main objection to processor-memory integration is the fact that memory costs tend to dominate, and hence economy of scale mandates the use of commodity parts that are optimized to yield the most Mbytes/wafer. Attempts to add more capabilities to DRAMs, such as video-buffers (VDRAM), integrated caches (CDRAM), graphics support (3D-RAM) and smart, higher performance interfaces (RamBus, SDRAM) were hurt by the extra cost for the non-memory areas. However with the advent of 256 Mbit and 1 Gbit devices [5] [6], memory chips have become so large that many computers will have only *one* memory chip. This puts the memory device on an equal footing with CPUs, and allows them to be viewed as one unit.

In the past, the 7% die-size increase for CDRAMs has resulted in an approximately 10% increase in chip cost. Ignoring the many non-technical factors that influence cost, a 256 Mbit DRAM chip could cost $800 given today's DRAM prices of ~$25/Mbyte. Extrapolating from the CDRAM case; if an extra 10% of die area were added for a processor, a processor/memory building block could cost $1000 — i.e. $200 for the extra processor. In order to be competitive, such a device needs to exceed the performance of a CPU and its support chips costing a total of $200. We show that such a device can perform competitively with a much more expensive system, in addition to being much smaller, demanding much less power and being much simpler to design complete systems with.

Older DRAM technologies were not suitable for implementing efficient processors. For example, it was not until the 16Mbit generation that DRAMs used more than one layer of metal. However, the upcoming 0.25 µm DRAM processes, with two or three metal layers, are capable of supporting a simple 200MHz CPU core. Compared to a state-of-the-art logic process, DRAMs may use a larger metal pitch and can have higher gate delays. However, Toshiba [7] demonstrated an embedded 4 bank DRAM cell in an ASIC process that is competitive with conventional 0.5µm ASIC technology. An older version of such a process (0.8µm) was used for the implementation of the MicroSparc-I [8] processor which ran at 85MHz. Shrinking this to 0.25µm should reach the target speed.

A significant cost of producing either DRAM or processor chips is the need to test each device, which requires expensive testers. Either device requires complementary support from the tester; a cpu test requires the tester to provide a memory sub-system, and a memory is tested with cpu-like accesses. Since an integrated processing element is a complete system, it greatly reduces these tester requirements. All that is required is to download a self-test program [9]. For the system described below, this requires just two signal connections in addition to the power supply.

4 The Integrated Design

Given the cost-sensitivity of DRAM devices, the design described below tries to optimize the balance between silicon devoted to memory, processor and I/O. The goal is to add about 10% to the size of the DRAM die, leading to a processing element with competitive performance and a superior cost-effectiveness.

Currently 10% of a 256 Mbit DRAM is about 30 mm^2. This is slightly more than the size of the MIPS R4300i processor [10] shrunk to a 0.25 µm CMOS process. Thus, the CPU will fit our design constraints. In addition, by using a high-speed serial-link based communication fabric [11] for off-chip I/O, the number of pads and interface circuitry is reduced. The die area saved can accommodate about 60K gates for two coherence and communications engines [12], creating a device with a simple and scalable interconnect.

It is possible to devote more area to the processing element in order to improve performance, for example, by using a superscalar pipeline, larger caches or additional processors. However, such additional complexity will further impact the device yield and its cost-effectiveness — this reduces practicality; designing competitive DRAMs is as capital-intensive as building high-end CPUs. Simpler solutions should enjoy economies of scale from targeting mainstream applications, and should leverage this momentum to provide commodity parts for high-end, massively parallel systems.

1. The most ubiqitous commercial application for chip logic synthesis.

2. The SS-10 has a prefetch unit that hides the memory access time in the case of small, linear strides.

4.1 The Combined CPU and DRAM

Figure 3 shows a block diagram of the proposed integrated processor and memory device.

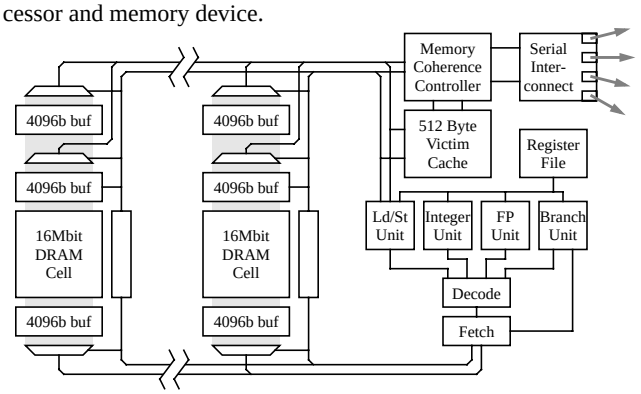


FIGURE 3 : The Design

The chip is dominated by the DRAM section, which is organized into multiple banks to improve speed (shorter wires have less parasitic capacitance to attenuate and delay the signals from the actual DRAM cell).

Sixteen independent bank controllers are assumed in a 256Mbit device. Fujitsu [13] and Rambus [14] currently sell 64Mbit devices with 4 banks, Yoo [15] describes a 32 bank device, and Mosys [16] are selling devices with up to 40 banks. Memory access time is assumed to be 30ns or 6 cycles of the 200 MHz clock. This figure is based on data presented in [17]. Each bank is capable of transferring 4K bits from the sense amplifier array to and from 3 column buffers. These three 512-Byte buffers form the processor instruction and data caches. Two columns per bank are used for a 2-way set-associative data cache making a total of 32 512-Byte lines spread across the 16 banks. The fact that an entire cache line can be transferred in a single DRAM access, combined with much shorter DRAM access latency, can dramatically improve the cache performance, and enable speculative writebacks, removing contention between cache misses and dirty lines. The remaining 16 column buffers make up a direct-mapped instruction cache with 512-Byte lines.

The performance of the 16KByte data cache is enhanced with a fully-associative victim cache [18] of sixteen 32-Byte lines with an LRU replacement policy. The victim cache receives a copy of the most recently accessed 32-Byte block of a column buffer whenever a column buffer is reloaded. This data transfer takes place within the time it takes to access the DRAM array on a miss, and thus is completely hidden in the stall time due to the memory access. Given this transfer time window, it is the bandwidth constraint from the main cache which dictates the shorter 32-Byte line size of the victim cache. The victim cache also doubles as a staging area for data that is imported from other nodes.

The nature of large DRAMs requires ECC protection to guard against transient bit failures, this incurs a 12% memory-size increase if ECC is computed on 64 bit words — the current industry standard. As all reasonable systems require this level of protection, this 12% overhead should not be counted against our design. Given the cost of the ECC circuitry, this function is performed at the instruction fetch unit and the load/store unit in our design, and not in each bank. Integration has the advantage that ECC checking can proceed in parallel with the processor pipeline (faulting an instruction before the writeback stage) while conventional CPU architectures require that the check be completed before the data is presented to the processor.

Two independent 64 (+8 for ECC) bit datapaths connect the column buffers with the processor core, one each for data and instruction access. These busses operate synchronously with the 200 Mhz processor clock, and each provides 1.6 GBytes/sec of memory access bandwidth.

The processor core uses a standard 5-stage pipeline similar to the R4300i [10] or the MicroSparc-II [8]. The evaluation presented in this paper was based on the Sparc instruction set architecture. Although the ISA is orthogonal to the concept of processor integration, it is, however, important to point out that an ordinary, general-purpose, commodity ISA is assumed. While customization could increase performance, economic considerations strongly argue against developing a new ISA. The R4300i currently consumes 1.5W, which will scale down with the smaller feature size and reduced supply voltage. Therefore it is reasonable to assume that the higher clock frequency will not cause a hotter chip.

4.2 System Interconnection and I/O

All I/O transfer and communication with other processing elements are controlled by two specialized protocol engines. These engines execute a downloadable microcode and can provide a message-passing or cache-coherent shared memory functionality. Both access memory via the data path. The protocol engines have been implemented and are described in [19]. The details of their operation is beyond the scope of this paper, but their actual operation is modeled and forms the basis of the multiprocessor evaluation section below. Both CC-NUMA [20] and Simple-COMA [21] shared-memory operations are currently supported.

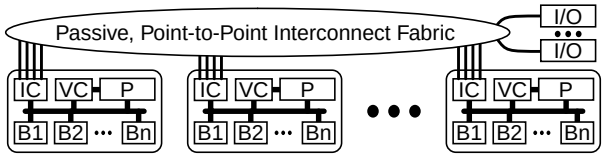


FIGURE 4 : System Overview

All off-chip communication is handled via a scalable serial link inter-connect system [11], which can operate at 2.5 Gbit/sec in a 0.25µm process. Four links provide a peak I/O bandwidth of 1.6 Gbytes/sec, which matches the internal memory bandwidth. Notably, all other I/O traffic is handled via the same interconnect medium. This links the memory of all processing elements into a common pool of cache-coherent shared memory, as depicted in Figure 4. This means I/O devices can behave like memory and access all memory just like the processor. Due to the tight integration between the processor, protocol engines and interconnect system, and because of the smaller, faster process, remote memory latencies can be reduced below 200ns (we have used more conservative numbers in our performance evaluation).

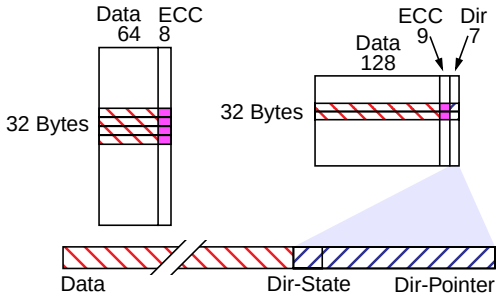


FIGURE 5 : Directory Structure

As described in [12] and shown in Figure 5, cache coherence is maintained by means of a directory structure that is kept in main memory, co-located with the data — avoiding the need for a separate directory cache. To eliminate the apparent storage overhead for the directory, the directory is encoded in extra ECC bits at the expense of reducing the error correction capability from 1 in 64 to 1 in 128 bits. Since cache coherency is maintained on 32 Byte blocks, 14 bits become available for the directory state and pointer.

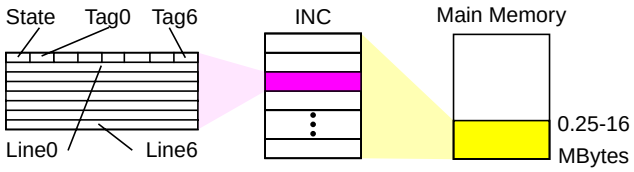


FIGURE 6 : Inter-Node Cache Organization

For the CC-NUMA modeled in this paper, a variable fraction of memory is reserved for an Inter Node Cache (INC) that holds imported data. This cache is 7-way set-associative (Figure 6) by storing seven 32-Byte lines in one 512-Byte column and storing all the tags in the eighth 32-Byte block. Each INC access requires 1 to 2 extra cycles over a normal (local) memory access due to the need to check the tags first.

Benchmark	Description
099.go	Artificial Intelligence: Plays the game Go against itself.
101.tomcatv	Fluid Dynamics/Mesh Generation: Generation of a 2D boundary-fitted coordinate system around general geometric domains.
102.swim	Weather Prediction: Solves system of Shallow Water equations using finite difference approximations.
103.su2cor	Quantum Physics: Computes masses of elementary particles in Quark-Gluon theory.
104.hydro2d	Astrophysics: Solves hydrodynamical Navier Stokes equations to compute galactic jets.
107.mgrid	Electromagnetism: Computes a 3D potential field.
110.applu	Math/Fluid-Dynamics: Solves matrix system with pivoting.
124.m88ksim	Simulator: Simulates the Motorola 88100 processor running Dhrystone and a memory test program.
125.turb3d	Simulation/turbulence: Simulates turbulence in a cubic area.
126.gcc	Compiler: cc1 from gcc-2.5.3. Compiles pre-processed source into optimized SPARC assembly code.
129.compress	Compression: Compress large text files (about 16MB) using adaptive Lempel-Ziv coding.
130.li	Interpreter: Based on xlisp 1.6 running a number of lisp programs.
132.ijpeg	Imaging: Performs JPEG image compression using fixed point integer arithmetic.
134.perl	Shell interpreter: Larry Wall's perl 4.0. Performs text and numeric manipulations (anagrams and prime-number factoring).
141.apsi	Weather: Calculates statistics on temperature and pollutants in a grid.
145.fpppp	Chemistry: Performs multi-electron derivatives.
146.wave5	Electromagnetics: Solve's Maxwell's equations on a cartesian mesh.
147.vortex	A single user O-O database transaction benchmark.Builds and manipulates three interrelated databases. Size is restricted to 40MB for SPEC95
Synopsys	Chip verification operation: compares two logic circuits and tests them for logical identity.

TABLE 2 : Benchmark Components

5 Uniprocessor Performance

Good multiprocessor scalability on its own is not enough to make a system generally commercially viable. Many of the applications a user may wish to execute are not parallelized or even parallelizable. It is important therefore that the integrated processor in the proposed system be capable of executing uniprocessor applications comparably with conventional architectures. Therefore, in this section we concentrate on the performance of the integrated system with uniprocessor applications.

5.1 Methodology

The current industry-accepted-standard metric for uniprocessor performance is the SPEC'95 benchmark suite [3]. This suite of programs (described in Table 2) is supposed to represent a balanced range of uniprocessor applications, the total execution time of which is used to measure the integer and floating point performance of a processor (and its memory subsystem). As well as using this suite of applications to benchmark the proposed design, the Synopsys [4] application was added as an example benchmark application with the large work-load of a real chip design.

Discussions of issues such as processor instruction set architecture, branch prediction or out-of-order execution are essentially orthogonal, and are beyond the scope of this paper. Futhermore, these issues involve degrees of complexity not envisioned for the processor under discussion. Instead, we concentrate on the novel aspect of the proposal, namely the memory system performance. The simplest first-order effect of the proposed design is the cache hit rate afforded. Each of the benchmark programs was compiled for the SPARC V8 architecture using the SunPro V4.0 compiler suite, according to the SPEC base-line rules, and then executed using a simulator derived from SHADE [22]. Cache hit and miss rates were measured for instruction and data caches, both for the proposed architecture and for comparable conventional cache architectures.

5.2 Instruction Cache Performance

Figure 7 compares the instruction cache (I-cache) miss rates for the proposed architecture to the miss rates enjoyed by conventionally dimensioned caches.

The left-most column for each application depicts the miss probability for an 8KByte column buffer cache with 512-Byte lines as proposed, while the remaining bars to the right depict the miss probability for various sizes of conventional direct-mapped caches with 32-Byte lines.

It is clear from these results that a number of the SPEC'95 benchmarks (110.applu, 129.compress, 102.swim, 107.mgrid, and 132.ijpeg) run very tight code loops that almost entirely fit an 8KByte cache. Of the remaining 14 applications with non-negligible miss rates, three of these (104.hydro2d, 141.apsi, 146.wave5) typically have miss rates between a 0.1% and 0.5% even for an 8KByte instruction cache.

The results in Figure 7 show that the proposed I-cache with its 512-Byte lines has a significant performance advantage over conventional first-level caches with 32-Byte lines. For almost all of the applications, the proposed cache has a lower miss rate than conventional I-caches of over twice the size. In some cases the performance benefits of the longer I-cache line size can be very dramatic; for example, in 145.fpppp the miss rate is a factor of 11.2 lower than the conventional cache of the same size, and a factor of 8.2 lower than the conventional cache of twice the size (16KBytes). Note that the benchmark entirely fits a 64KByte I-cache.

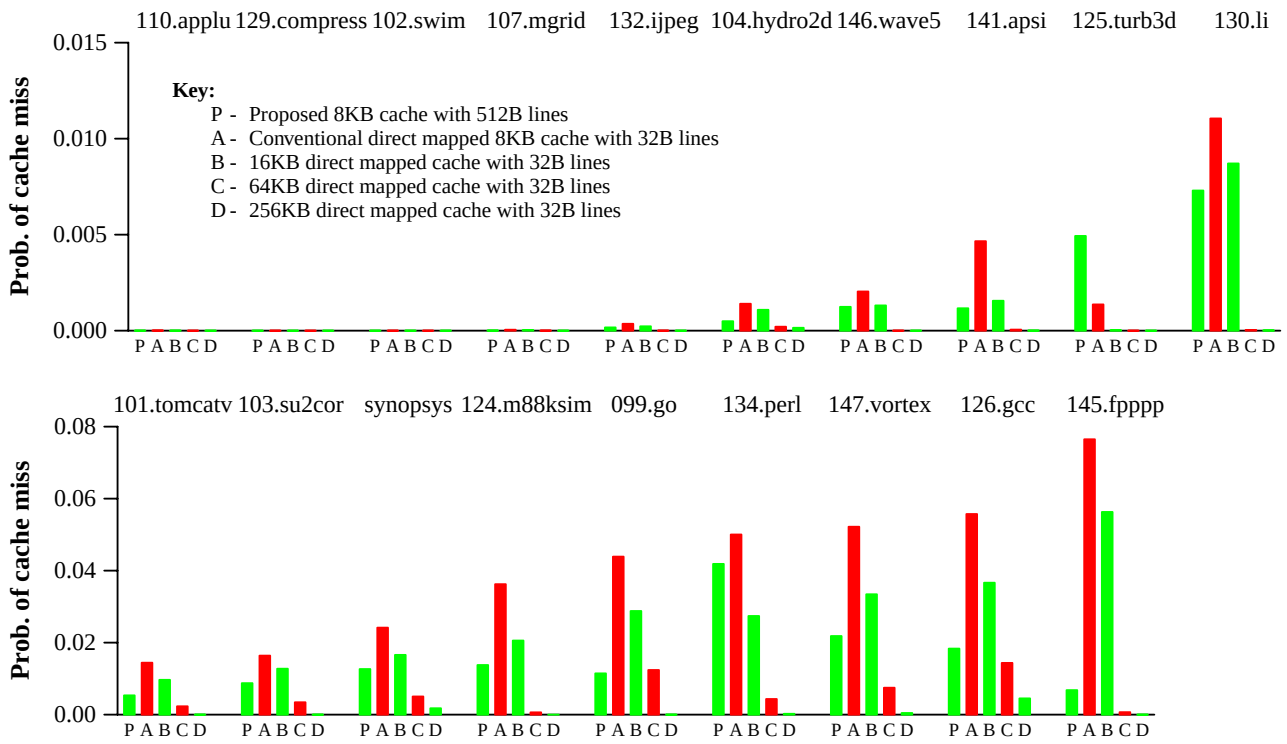


FIGURE 7 : Instruction Cache Miss Rates

The reduced miss rate of the proposed I-cache results directly from the prefetching effect of the long cache line size, combined with the usually high degree of locality found in instruction streams.

Conventional processor designs are unable to reap the benefits of an increased cache line size because the time it takes to fill such a line introduces second-order contention effects at the memory interface. The proposed integrated architecture fills the 512Byte line in a single cycle (after pre-charge and row access) directly from the DRAM array, so these contention effects do not appear. We return to this issue of contention is again in Section 5.5.

Only two of the SPEC benchmarks stand out for their somewhat disappointing I-cache performance; 134.perl has a surprisingly high miss rate, though still lower than the equivalent conventional cache of the same size, because the code is large and has poor locality. 126.gcc has similar characteristics, but the I-cache miss rates for this application are within 27% of those of a 64KByte conventional I-cache. Perhaps code profiling to reduce cache conflicts may improve the miss rates for perl. The only application to produce a higher miss rate on the proposed architecture was 125.turb3d. This appears to be the result of a direct code conflict between a loop and a function it calls, rather than a general capacity or locality problem. The problem is an artifact of the reduced number of cache lines, but can be removed by a code profiler noting the subroutine being called by the loop — the respective loop and function code can then be re-laid by the compiler or linker to avoid the conflict.

5.3 Data Cache Performance

Instruction caches are important to keep the processor busy, and the generally good locality of instruction streams means that the prefetching effect of the proposed cache works well. However, as the SPEC benchmarks show, even a modest size cache is sufficient to cover much of the executing code. Data caches, on the other hand, need to cope with more complex access patterns in order to be effective — often there is no substitute for cache capacity.

As described in Section 4.1, the proposed architecture has thirty-two column buffers (each 512-Bytes long attached to each of the sixteen DRAM banks) dedicated to serving data accesses from the cpu, effectively making a 16KByte 2-way associative data cache (D-cache) with 512-Byte lines. This configuration was simulated in much the same way as the I-cache in order to compare its effectiveness with direct mapped and 2-way associative first-level caches having a more conventional 32-Byte line size. Figure 8 presents the miss rates resulting from these simulations. Each vertical bar shows both the load and the store cache miss probabilities — the combined height is the total cache-miss fraction. The bar to the left for each application is the miss rate for the proposed D-cache structure. The right-most bar for each application illustrates the miss rates after the addition of a small victim cache — we return to this in Section 5.4. The remaining bars represent the conventional cache miss rates.

Figure 8 shows that the application suite has a significantly more varied D-cache than I-cache behavior. Given the generally reduced temporal and spatial locality of data references compared to instructions, this is to be expected. In turn, there is a more pronounced difference between the performance of the proposed D-cache structure and conventional cache designs for most of the benchmarks.

Those applications that have a high degree of locality benefit from the prefetching effect of the long lines, but the long lines can also increase the number of conflict misses. For example, 107.mgrid and 104.hydro2d exhibit markedly reduced D-cache miss rates — over a factor of ten lower for mgrid on the proposed architecture compared to a conventional direct-mapped D-cache of the same capacity, and still a factor of 5 lower than a 2-way associative 256KByte conventional cache configuration.

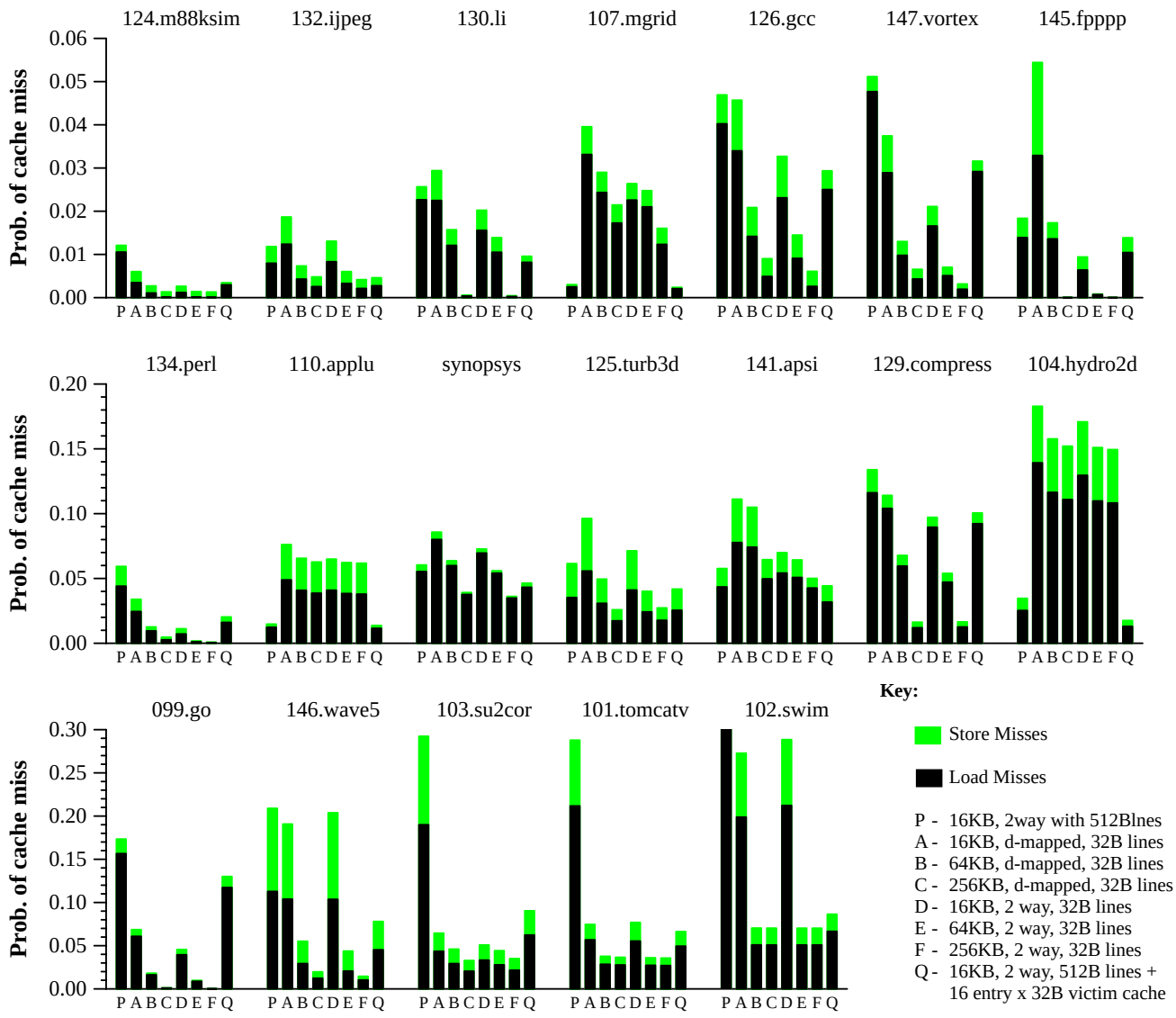


FIGURE 8 : Data Cache Miss Rates

Unfortunately, the reverse is true of other applications; for 103.su2cor, 102.swim and 101.tomcatv the 512-Byte line size of the proposed cache increases the number of conflict misses by almost a factor of five over a conventional cache of the same size.

Early design simulations gave unacceptable miss rates for only an 8KB direct-mapped cache with 512-Byte lines — partly due to the reduced capacity, but mostly due to the conflicts arising from having only 16 cache lines.

Introducing an additional data column buffer to each DRAM cell doubled the capacity of the architecture's D-cache to 16KBytes, and provided two-way associativity, which dramatically improved the performance. While the prefetching benefits of the large D-cache lines are desirable, as can been seen from the miss rates in Figure 8, the conflict misses caused by the long line size can be equally detrimental for other applications. It is desirable, therefore, to reduce the incidences of conflict without reducing the 512B line size or increasing the cache capacity.

5.4 Adding a Victim Cache

Jouppi [18] showed that a small, fully-associative buffer ("victim cache") could be used to hold cache lines most recently evicted from the data cache. This buffer works to increase the effective associativity of the cache in cases where the cache miss rate is dominated by conflicts, reducing the number of main memory accesses.

To reduce the dominating effect of cache conflicts seen in Section 5.3 for certain of the benchmark programs, a small associative victim cache was added, as proposed.

The entire buffer is the same size as a single cache line, though it is organized as a 16-way fully associative bank of 32-Byte lines. When a cache line is evicted from a DRAM column buffer, due to a miss, the last accessed 32-Byte sub-block of that line is copied into a selected entry in the victim cache before the new cache line is read. This has no performance penalty, since the CPU is frozen on the miss, and the new cache line cannot be loaded until the access of the DRAM array completes. This leaves four free cycles during which to copy the 32-Byte sub-block from the old cache line into the victim cache (64-bits at a time).

On each memory access cycle, as the main D-cache is searched, each of the 16 entries in the victim cache are also checked to see if it holds the data to be accessed. Unlike a conventional victim cache, the contents cannot be reloaded into a line of the main D-cache because of the size disparity.

The addition of the victim cache was simulated, and the resultant miss rates collected and presented in the right-most result bars for each application in Figure 8. A *least-recently-used* replacement algorithm used for the entries in the victim cache.

The victim cache had a dramatic effect on the cache miss rates. In all but one application the combined D-cache and victim cache has a lower miss rate than the 16KByte direct-mapped data cache. For the applications where the plain D-cache performed poorly (particularly 103.su2cor and 101.tomcatv), the victim cache absorbed the conflict misses reducing the miss rate to approximately that of a conventional 2-way 16KByte cache. Moreover, for three other applications (102.swim, 146.wave5, 130.li) the miss rate was reduced between two and five-fold to a level equivalent to a 64KByte conventional cache. For 102.swim and 146.wave5 the 2-way associative 16KByte D-cache has a slightly higher miss rate than the direct-mapped one. It appears that for these applications a high degree of associativity is needed when the D-cache is small. The victim cache provides this associativity — absorbing the accesses with poor spatial locality, and leaving the remaining accesses to benefit from the prefetching effect of the longer cache lines. Hence, the miss rates are almost as low as for the 64KByte conventional caches.

There are always exceptions: while the victim cache helps reduce the miss rate by 25%, it does not have the capacity to absorb the conflicts from the main D-cache whose long lines are not suited to the poor locality and small data structures used by the 099.Go search algorithm.

In summary, the long lines of the column buffer based D-cache provide a beneficial prefetching effect for some applications, but also cause significant cache conflicts for others due to the reduced number of lines in the cache. A modest victim cache the size of a single column buffer can absorb nearly all of the poor locality misses and in some cases allow the prefetching effect to be seen.

5.5 Modeling Memory Contention

Although the column buffers comprising the data and instruction caches of the proposed design are filled in parallel and in a single cycle (after the DRAM array access and pre-charge times), simply measuring the hit and miss ratios of the caches does not provide any information about bottlenecks and contention that affect the processor throughput. Moreover, it does not provide an entirely fair comparison with conventional processor designs that suffer more from similar second-order contention effects arising from memory system bottlenecks. To examine this arena the integrated processor/memory system was modeled using generalized, stochastic Petri Nets (GSPNs) [23] that take into account contention for shared resources (such as memory banks) and event dependencies.

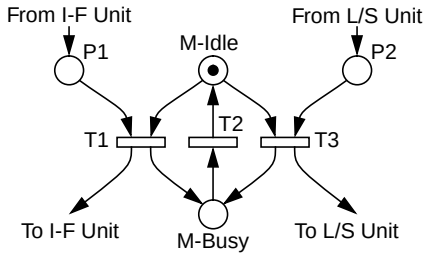


FIGURE 9 : Memory Bank GSPN

In addition, we adapted the Petri nets to model a conventional CPU design with separate (Harvard Architecture) first-level instruction and data caches accessing a larger unified second-level cache, which in turn accesses a dual-banked main memory.

Instruction hit and miss ratios derived directly from the simulations described in the preceeding sections were dialed directly into the models in order to derive processor Cycles-Per-Instruction (CPI) performance numbers.

The GSPNs were evaluated using a Monte-Carlo simulator.

Figure 9 depicts the model for one memory bank that can either serve an instruction cache miss or a data cache miss, but not both simultaneously. The deterministically timed transitions T1 and T3 reflect the access time, while the transition T2 models pre-charge operations that prevent the memory from accepting a new transaction for a certain time after the last access. The places P1 and P2 are entered via immediate transitions from the instruction fetch unit or the load/store unit. It is assumed that these transitions have equal rates, so that memory banks are accessed with a uniform, random distribution. For the integrated processor/memory system, 16 memory banks are used. The conventional reference system which is used to validate the model has 2 independent memory banks.

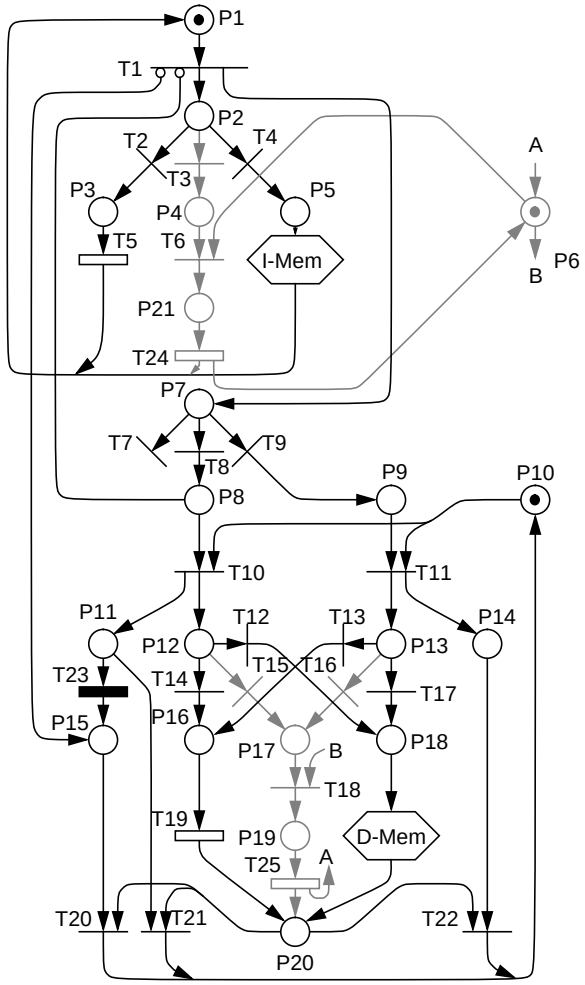


FIGURE 10 : Processor/Cache GSPN

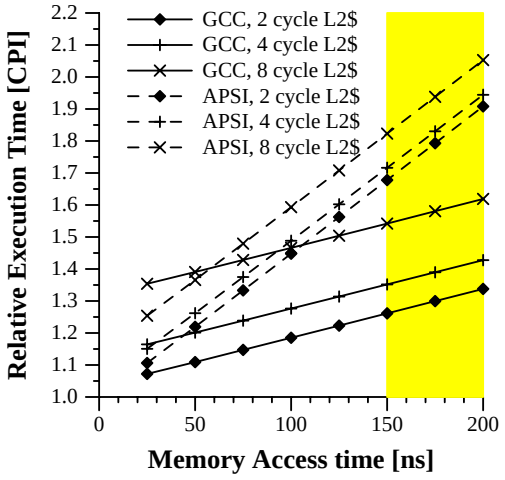


FIGURE 11 : Cache Latency Impact

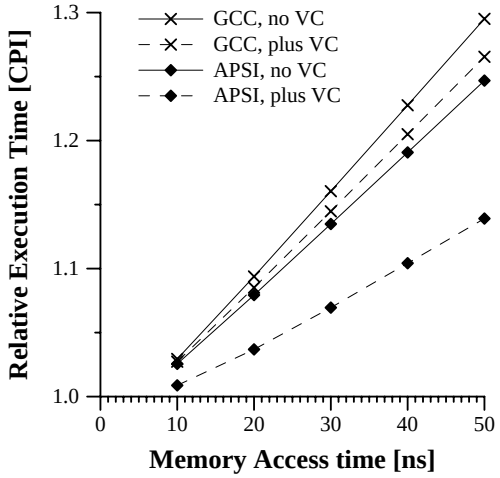


FIGURE 12 : Memory Latency Impact

Figure 10 depicts the Petri-net used to simulate the proposed processor. The components added to this model to simulate a conventional "reference system" with a shared second-level cache are shown in grey. These (grey) components are removed for simulating our proposed "integrated" system.

The top part of Figure 10 is the instruction fetch unit. Each time the transition T1 fires, an instruction is presented to the execution units, and the transitions emanating from place P7 reflect the probability for an non-Load/Store operation (T7), a Load (T8) or a Store (T9). T1 depends on a loaded instruction (P1) that is retrieved from either the instruction cache (P3/T5), the optional second-level cache (P21/T24) or the memory (P5...). The rates on the transitions T2, T3, and T4 reflect the probabilities for an instruction cache hit, a second-level cache hit, or a fill from main memory, respectively. These rates were determined from the Shade simulations of the actual benchmarks.

We assumed a store buffer that can postpone stores (hence P9 does not stall the processor). However, the load/store unit is limited to **one** outstanding operation at a time (the token from P10 is require to issue either a load or a store). The exponentially timed transition T23 models stalls due to incomplete loads. To model a system without scoreboarding, this rate for T23 is set to infinity. However, we assumed the presence of scoreboarding logic for the integrated system, therefore the rate of T23 to was set 1 so that on average one operation can be performed before an incomplete load will stall the processor.

The transition ratios for T14, T15 and T12 reflect the probabilities that a load is a data cache hit, a second-level cache hit or a reference to main memory, respectively. Similarly, T13, T16, and T17 model the cache behavior of stores.

The deterministically timed transitions T24 and T25 model the access times for the optional second-level cache model. The place P6 ensures mutual exclusion between data and instruction fetches.

The assumption in the model is that operations other than memory accesses will not stall the CPU pipeline. This is a simplification in regard to floating-point operations that usually do not complete in a single cycle. To correct this, a cycle accurate MicroSparc-II simulator (with a zero-latency memory system)[1] was used to calculate a base CPI component due to functional unit dependencies within the CPU for each of the SPEC applications. These results were then combined with the additional CPI component derived from the Petri-Net models.

Two applications with representatively high and low CPI figures are 141.apsi and 126.gcc, respectively. Using the Petri-Net model we can track the CPI performance of a simple 200MHz 5-stage uni-scalar CPU (with 16KByte first-level I+D caches and a 256KByte second level combined cache) as a function of the second-level cache and main memory access latencies. These results are depicted in Figure 11. The grey area indicates the typical operating region of a conventional CPU, and indicates that the memory access latency alone can cost up to a factor of 2 in the CPU's CPI performance above the raw (zero latency memory) performance.

Based on data in [17], the proposed integrated device could have a 30ns access time. The CPI performance of the proposed architecture with this memory access time is depicted in Figure 12, and indicates that at 30ns access time the CPI impact is between 10% and 25% above the raw CPI figure.

To put this performance in more concrete terms, Tables 3 and 4 show estimated Spec'95 numbers for the proposed architecture running at 200MHz with a 30ns DRAM array access latency.

For comparison, Table 4 also includes the Spec results for today's fastest available CPU [3]. To be fair, given current trends in CPU development, we can expect the DEC Alpha family to increase in performance over the time-frame in which the proposed architecture might be realized. However, this performance improvement will be at the cost of further complexity, beyond that of the 21164 — which is already a factor of 10 larger than our proposed architecture, not including the necessary support chips required.

5.6 Impact of Memory Organization

The number of independently controllable memory banks in a DRAM device is strongly dependent on the actual DRAM implementation. This number impacts the proposed architecture in two ways: first, it determines the amount of contention and second, it constrains the cache architecture.

To address the first issue, 4, 8 and 16 banks were simulated for all the Spec benchmarks using the GSPN memory contention model of the integrated processor/memory device. For comparison 2 to 8 banks were simulated for the conventional CPU system. In all cases, the performance differences were below the error limits of the simulation — which is understandable given the relatively low activity at each bank. For example, in gcc each of the 16 banks are busy only 1.2% of the time, and increases to only 9.6% with 2 banks, which is still low for the probability of contention.

1. Derived using Sun's internal MicroSparc-II performance simulator.

Name	CPI [cpu + memory]	Spec-ratio
099.go	1.01 + 0.48	6.0
124.m88ksim	1.01 + 0.12	4.3
126.gcc	1.01 + 0.14	7.6
129.compress	1.03 + 0.17	6.4
130.li	1.02 + 0.06	6.7
132.ijpeg	1.00 + 0.01	5.8
134.perl	1.04 + 0.21	6.0
147.vortex	1.02 + 0.27	6.4
101.tomcatv	1.15 + 0.50	8.2
102.swim	1.56 + 0.97	12.7
103.su2cor	1.41 + 0.44	3.2
104.hydro2d	1.74 + 0.04	4.2
107.mgrid	1.20 + 0.01	3.2
110.applu	1.53 + 0.01	3.9
125.turb3d	1.16 + 0.05	4.3
141.apsi	1.70 + 0.08	5.0
145.fpppp	1.34 + 0.08	7.5
146.wave5	1.31 + 0.25	7.6

TABLE 3 : Spec'95 Estimates, no Victim Cache

Name	Total CPI	Spec-ratio	Alpha 21164 [DEC 8200 5/300]
099.go	1.30	6.9	10.1
124.m88ksim	1.10	4.5	7.1
126.gcc	1.13	7.8	6.7
129.compress	1.16	6.6	6.8
130.li	1.07	6.8	6.8
132.ijpeg	1.01	5.8	6.9
134.perl	1.21	6.2	8.1
147.vortex	1.17	7.1	7.4
101.tomcatv	1.23	11.1	14.0
102.swim	1.65	19.5	18.3
103.su2cor	1.51	3.9	7.2
104.hydro2d	1.75	4.2	7.8
107.mgrid	1.21	3.2	9.1
110.applu	1.54	4.0	6.5
125.turb3d	1.20	4.3	10.8
141.apsi	1.76	5.1	14.5
145.fpppp	1.42	7.5	21.3
146.wave5	1.41	8.4	16.8

TABLE 4 : Spec'95 Estimates, with Victim Cache

The second and much larger effect is the impact upon practical cache organization. If the number of banks were to be reduced, either the line size or the associativity of the cache needs to be increased. Mitsubishi [9] opted to use 4 banks and add a small amount of SRAM to each bank to increase the number of available lines. In this case, the cache organization essentially remains unchanged and the overall performance only decreases slightly due to some increase in contention. However, simulation shows that increasing the line size will degrade performance due to higher resultant cache conflicts.

6 Multiprocessor Performance

Thus far, only the uniprocessor performance of the proposed integrated system has been considered. The unusual design of the processor caches was shown, in most cases, to be beneficial, but it is not clear that this should be the case for shared memory multiprocessing where the addition of coherence misses affects the caches.

Benchmark	Description	Data Set
LU	LU decomposition	200x200 matrix
MP3D	3-D particle-based wind-tunnel simulator	10 K particles, 10 steps
Ocean	Ocean basin simulator	128x128 grids, tolerance 10^{-7}
Water	N-body water molecular dynamics simulation	288 molecules, 4 time steps
PTHOR	Distributed digital time digital circuit simulator	RISC circuit, 1000 time steps

TABLE 5 : Splash Benchmarks

System	Access	Latency
Proposed Combined CPU & DRAM	Hit in column buffer	1
	Hit in victim cache	1
	Access local memory & INC	6
	Invalidation round trip delay	80
	Load remote data	80
Comparative CC-NUMA	Hit in FLC	1
	Hit in SLC	6
	Invalidation round trip delay	80
	Load remote data	80

TABLE 6 : Memory Latency (processor cycles)

6.1 Simulation Models and Methodology

To understand the performance potential of the integrated design, an execution driven simulation, with a timing-accurate architecture model, was used to evaluate a simple system based on the building blocks of Figure 3. These performance figures are then compared with results obtained from a reference CC-NUMA design [20] with 16KByte direct-mapped First-Level Caches (FLCs) and infinitely sized Second-Level Caches (SLCs). The use of infinite SLCs removes the SLC capacity misses, providing an ideal upper-bound of the system performance when only the cold and the coherence misses are considered [24]. This enables the direct study of the integrated cache system compared to a conventional FLC in a shared memory system.

To simulate the proposed integrated system, the Inter-Node Cache (INC) size was set at 1MByte per memory/processor node — larger than the working sets of the applications used, and so comparable to the infinite SLCs of the reference CC-NUMA architecture. However, a finite size was chosen because the INCs are configured as 7-way set-associative caches; making them infinitely large would remove the beneficial effects of set associativity. The granularity of coherence (or *coherence unit*) is always 32-Bytes. For the purposes of these simulations, cache coherence was maintained by a write-invalidate protocol [24].

The simulation benchmarks are taken from the SPLASH [25] suite, for which brief descriptions and the sizes of the data sets used are given in Table 5.

The architectural simulator is built on top of the CacheMire Test Bench [26], which is an execution-driven simulator of multiple SPARC processors. The processors in the CacheMire simulator issue memory accesses, and the architectural simulator then delays the processors according to the latencies shown in Table 6. The SPLASH application codes are sufficiently small that for both the integrated and the reference architectures, instruction fetches are assumed to always hit in the instruction caches. Therefore, only data references were simulated.

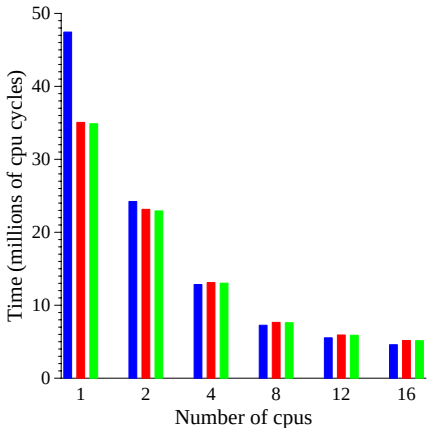


FIGURE 13 : LU

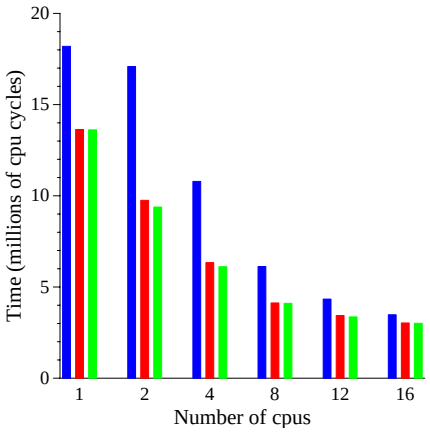


FIGURE 14 : MP3D

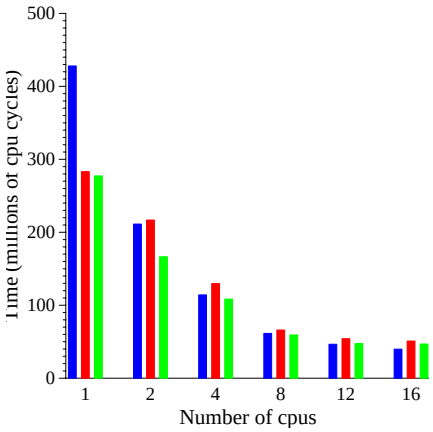


FIGURE 15 : OCEAN

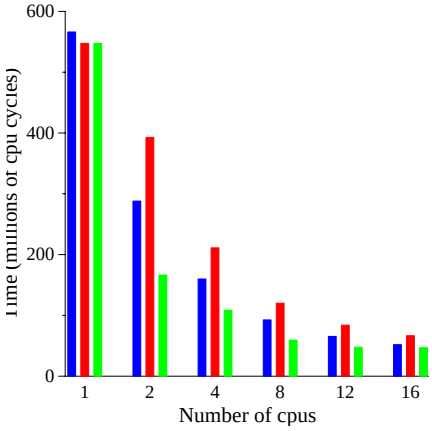


FIGURE 16 : WATER

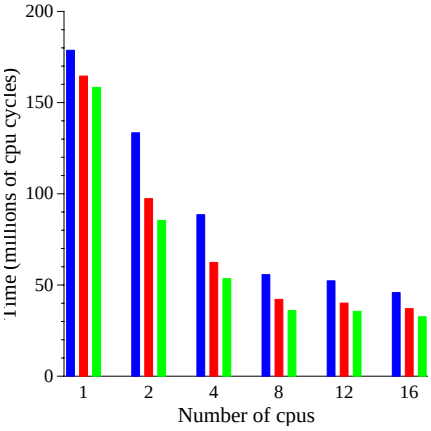
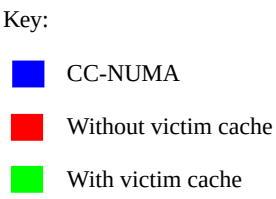


FIGURE 17 : PTHOR



6.2 Performance Results

The total execution times with increasing numbers of processors for the benchmarks LU, MP3D, OCEAN, WATER and PTHOR are depicted in Figures 13 to 17.

The column buffers, which load 512-Bytes at a time, exploit the spatial locality of local memory accesses well due to the long-line prefetching effect mentioned earlier. As a result, the integrated design outperforms the traditional CC-NUMA designs for small numbers of processors in all cases — until the working set per cpu becomes too small for the effect to be observed.

Since the coherence units transferred between nodes are always 32-Byte blocks, the long-line prefetching effect does not help for accesses to data held on another node. Therefore the proposed architecture has the same number of cold and coherence misses as the traditional CC-NUMA. In fact, it is important not to use the long cache lines as coherence units, because the false-sharing costs would outweigh the prefetching benefits for most applications. However, the long cache lines can improve the performance of the integrated multiprocessor's remote data caches. The long column buffer lines enable access to seven 32-Byte INC blocks each - providing 7 way associativity for cached remote memory reducing conflict misses.

When applications exhibit good locality for local memory accesses or have a large number of conflicts on remote data accesses, the proposed column buffer design can produce good results. WATER is the only benchmark for which the reference CC-NUMA design shows better results than the integrated architecture unaided by a victim cache. In WATER, the main data structure is a shared vector that maintains all molecules. Sets of molecules are allocated to processors statically. In this application, true sharing misses dominate. As each molecule is described by a data structure of approximately 600 Bytes, and is only partially accessed, the limited numbers of column buffers of 512-Bytes suffer from the lack of spatial and temporal locality. As shown in Figure 16, when the victim caches are used, the total execution time is significantly reduced.

A further observation is that the column buffer D-cache itself is not enough to support scalable computation. As shown in Figures 15 and 16, the reference CC-NUMA performs better than the integrated design with only column buffers. Again, this is because the addition of coherence traffic from shared memory operation further increases the conflict misses caused by the small number of column buffer long lines. As a result, most memory accesses miss the D-cache and need 6 cycles to reach the data in the INC DRAM in the integrated design; while similar accesses may be served in 1 cycle by the FLC of the reference CC-NUMA design. This is not quite a fair comparison due to the small nature of the standard problems' working sets — with larger, more realistic, problems, the FLC of the reference CC-NUMA architecture would start experiencing more capacity misses, incurring the 6 cycle penalty to it's SLC.

Based on preliminary observations, each node in the integrated design was augmented just as in the uniprocessor case by the addi-

tion of a small 16-way by 32-Byte associative victim cache. Compared to the 16KByte FLC of the reference CC-NUMA, the 512-Byte capacity of the victim cache is very small, however even this can reduce the execution time of the integrated design by up to a factor of 2 (in the case of WATER). With the aid of a victim cache, the integrated design shows the best performance results for the tested applications.

7 Related Work

Transputer

In 1982 Inmos introduced the Transputer family of RISC processor chips [27] that were combined with a small amount of local memory and tightly integrated with a 4 serial link communications interface. The processor was tailored for a message-passing paradigm (specifically, the language Occam), and efficiently supported high speed context switching and message handling. While this chip has some similarities with the idea presented it did not provide a shared memory capability or even virtual memory. Moreover, the local memory was simply 1KB of static RAM, so for all practical applications the chip had to be combined with external DRAM.

M machine

The M-Machine [28] project plans to integrate 4 super-scalar CPUs with memory on a single chip. However, the on-chip memory size is small as 4Mbits. In most cases, external I/O and memory chips are needed. Moreover, the M-Machine is a message passing design.

Exacube

The EXECUBE [29] exploited a similar idea of utilizing the memory bandwidth to the maximum by CPU memory integration. A CPU element, 64KBytes of memory and four communication links form a processing unit. Each chip, based on the dense DRAM technology of the day, holds eight such processing units. Unlike our design, the EXECUBE chip is intended to serve as building block for MPP systems. There is no shared memory capability, or ability to use DRAM column buffers for instruction and data caches. Furthermore, it is primarily intended as a SIMD architecture.

Sharc

The Analog Devices ADSP-21060 chip [30] has a single DSP (digital signal processor) CPU integrated with 4Mbits of memory. Like the EXECUBE, MPPs systems can also be built using this chip, but not shared-memory systems.

Mitsubishi, Toshiba & NEC

Mitsubishi have already fabricated a combined 32-bit RISC core and DRAM device [9]. While this device demonstrates that a commercial DRAM process can be used to fabricate complex logic on the same chip, it contained a conventional cache design separated from the DRAM arrays, thus failing to exploit the full memory bandwidth potential. Futhermore, the device contains no scalable interconnect capability.

Other DRAM vendors, for example Toshiba and NEC, are in the process of making available DRAM devices with dedicated semi-custom logic areas, so that customers can design their own intelligent DRAMs.

8 Vision

The integrated processor/memory chip could be regarded as the equivalent to a Lego^{TM} building block. Dwarfed by its modest heat-sink to cool some 1.5W, the processing elements could be added incrementally to a silicon-less mother board (Figure 20) that provides only sockets with power and signal traces to other such sockets. When needed, more integrated chips can be added to a logically seamless shared memory multiprocessor.

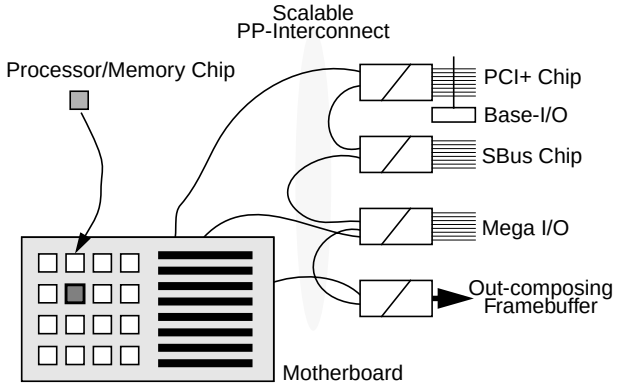


FIGURE 18 : Silicon-Less Motherboard

I/O functions are handled by chips that can translate to standard interfaces, such as PCI, SCSI, ATM, Fiber-Channel, etc. Among the more interesting capabilities of such a system is to build a framebuffer that retrieves its data from the main memory as it refreshes a screen or LCD panel. This is made feasible by the high memory bandwidth that is available internally.

Due to the scalable interconnect fabric, the system's bi-sectional bandwidth increases as components are added. This system is not limited to one mother-board, rather the delay insensitive point-to-point connections of the S-Connect systems allow multiple boards to communicate via cables (<10m) or fiber-optic connections (up to 200m).

9 Conclusions

The trend towards larger DRAM devices exacerbates the processor/memory bottleneck, requiring costly cache hierarchies to effectively support high performance microprocessors. A viable alternative is to move the processor closer to the memory, by integrating it onto the DRAM chip. Processor/memory integration is advantageous, even if it requires the use of a simpler processor.

We have shown that a conventional, single-scalar processor with a small cache, integrated with a 256-Mbit DRAM array can form a self-contained, general purpose processing element with competitive performance that can approach that of high-end super-scalar processors with large, multilevel caches. Small (8KByte) direct mapped instruction caches with long lines (512-Bytes) perform surprisingly well with a zero-fill cost, which is a feature of the integration. Victim caches were shown to be very effective when combined with a multi-banked column buffer data cache. Combined with the lower latency inherent in an integrated design, the memory induced degradation of processor performance was greatly reduced.

Including scalable Distributed Shared Memory (DSM) with hardware assisted cache coherency results in a versatile building block that can outperform an ideal (i.e. infinite second-level cache) conventional DSM system.

Acknowledgments

The authors would like to acknowledge the valuable help, feedback and inspiration they received from Gunes Aybay, Clement Fang, Howard Davidson, Mark Hill, Sally McKee, William Radke, Eugen Schenfeld, Sanjay Vishin, the engineers of the Sparc Technology Business organization and the reviewers.

References

[1] Wulf, Wm.A and McKee, S.A. *Hitting the Memory Wall: Implications of the Obvious.* ACM Computer Architecture News. Vol.23, No.1 March 1995.

[2] Wilkes, M.V., The Memory Wall and the CMOS End-Point, ACM Computer Architecture News. Vol. 23, No. 4 September 1995.

[3] *SPEC Newsletter*; URL: `http://www.specbench.org/results.html`

[4] *Synopsys Inc.*, 700 East Middlefield Rd. Mountain View, California, CA 94043.

[5] Horiguchi, M. et.al., *An Experimental 220MHz 1Gb DRAM*, IEEE International Solid-State Circuits Conference 1995. San Francisco, p.252.

[6] Sugibayashi, T. et.al., *A 1Gb DRAM for file Applications, IEEE International* Solid-State Circuits Conference 1995. San Francisco, p.254.

[7] Miyano, S. et.al., A 1.6GB/s Data-Transfer-Rate 8Mb Embedded DRAM, IEEE International Solid-State Circuits Conference 1995. San Francisco, p.300

[8] *MicroSparc documentation*, internal communication with Sparc Technology Business Inc.

[9] Shimizu, et.al. *A Multimedia 32b RISC Microprocessor with 16Mb DRAM*, International Solid-State-Circuits Conference, February 1996, pp216-217.

[10] *MIPS R4300i Processor Reference Manual*, `URL: http://www.mips.com/r4300i/R4300i_B.html`

[11] Nowatzyk, A., Browne, M., Kelly, E. and Parkin, M. *S-Connect: from Network of Workstations to Supercomputer Performance*. Proceedings of the 22nd International Symposium on Computer Architecture, June 1994.

[12] Nowatzyk, A., Aybay, G., Browne, M., Kelly, E., Parkin, M., Radke, B. and Vishin, S. *The S3.mp Scalable Shared Memory Multiprocessor.* Proceedings of the 24th International Conference on Parallel Processing, 1995.

[13] *MB81164840 - CMOS 4x2Mx8 Synchronous DRAM*, Fujitsu Microelectronics Inc., 3455 N. first St., San Jose CA 95134,

[14] *RDRAM Reference Manual*, Rambus Inc., 2465 Latham Street, Mountain View, CA 94040.

[15] Yoo, J.H. et.al., *A 32-bank 1Gb DRAM with 1GB/s Bandwidth*, IEEE International Solid-State Circuits Conference 1996, San Francisco, p.378.

[16] Przybylski, S., *MoSys Reveals MDRAM Architecture*, Microprocessor Report, Vol 9:17, Dec 25, 1995, MicroDesign Resources, Sebastopol, CA95472. ISSN 0899-9341

[17] Koike, H., et.al., *A 30ns 64Mb DRAM with Built-in Self-Test and Repair Function*,ISSCC 1992, San Francisco, p150

[18] Jouppi, N. *Improving Direct-Mapped Cache Performance by Addition of a Small Fully-Associative Cache and Prefetch Buffer*, Proceedings of the 17th Annual International Symposium on Computer Architecture, 1990 pages 364-373

[19] Nowatzyk, A., Aybay, G., Browne, M., Kelly, E., Parkin, M., Radke, B. and Vishin, S.*Exploiting Parallelism in Cache Coherency Protocol engines*, Europar 1995, Stockholm, Sweden

[20] Lenoski, D. *The Design and Analysis of DASH: A Scalable Directory-Based Multiprocessor.* PhD Dissertation, Stanford University, December 1991.

[21] Saulsbury, A. et.al. *An Argument for Simple COMA*, 1st IEEE Symposium on High Performance Computer Architecture January 22-25th 1995, Rayleigh, North Carolina, USA; pages 276-285.

[22] Cmelik, B. *The SHADE simulator*, Sun-Labs Technical Report, 1993

[23] Marsan, G.,Conti, *A class of generalized stochastic petrinets for the performance evaluation of multiprocessor systems*, ACM Transactions on Computer Systems, 2(2): 93, May 1984

[24] Dubois, M., Skeppstedt, J., Ricciulli, L., Ramamurthy, K. and Stenström, P. *The Detection and Elimination of Useless Misses in Multiprocessors*. Proceedings of the 20th Annual International Symposium on Computer Architecture, pp. 88-97, May 1993.

[25] Singh, J.P., Weber, W.-D., and Gupta, A. *SPLASH: Stanford Parallel Applications for Shared-Memory.* Computer Architecture News, 20(1):5-44, March 1992.

[26] Brorsson, M., Dahlgren, F., Nilsson, H. and Stenström, P. *The CacheMire Test Bench - A Flexible and Effective Approach for Simulation of Multiprocessors*. Proceedings of the 26th Annual Simulation Symposium, pp. 115-124, 1993.

[27] *The Transputer Reference Manual*, 1988, INMOS Ltd., Pub. Prentice Hall, ISBN 0-13-929001-X.

[28] Dally, W.J. et. al. *M-Machine Microarchitecture*, Tech Report, Artificial Intelligence Lab MIT, Cambridge, MA. Jan 1993

[29] Kogge, P.M., *EXECUBE - A New Architecture for Scalable MPPs*, 1994 International Conference on Parallel Processing.

[30] *ADSP-21060 SHARC Super Harvard Architecture Computer*, ANALOG DEVICES, Norwood, MA, Oct. 1993.