System-Level Implications of Processor-Memory Integration

Douglas Burger

Technical Report #1349

June 1997

This paper appeared in the Mixing Logic and DRAM workshop at the 24th International Symposium on Computer Architecture, June, 1997.

System-Level Implications of Processor-Memory Integration

Doug Burger

Computer Sciences Department
University of Wisconsin-Madison
1210 West Dayton Street
Madison, Wisconsin 53706 USA
dburger@cs.wisc.edu - http://www.cs.wisc.edu/galileo

Abstract

In this paper, we address the system-level implications of processor/memory integration. Specifically, we explore the effects that very large on-processor memories will have upon both the memory hierarchy as a whole and the processor organization. Our focus is on the migration of memory to the processor, not the migration of inexpensive processors onto commodity DRAM parts (the feasibility of the latter model in the market is still an unanswered question).

Using cost/performance models coupled with simulation results, we compare three simple on-chip memory organizations (cache, fraction of main memory, and a hybrid of the two). We then examine the constraints under which all of the main memory may migrate onto the processor, thus enabling IRAM-based systems. Finally, we discuss the implications that large on-processor memories have for chip multiprocessors (CMPs), and we discuss appropriate uses for the multiple on-chip processors.

1 Implications of large on-processor memories

The continuing exponential growth in microprocessor performance and real-estate, coupled with the growing gap between processor and stock DRAM performance, is making the performance of the memory hierarchy the key determinant of overall system performance. The growing interest in IRAM chips—which combine processor and physical memory on a single die—reflects the growing importance of the memory hierarchy in system design. IRAM chips have been proposed [2, 10, 13] as a cost-effective way to improve memory bandwidth and reduce memory latency, as opposed to the current conventional approach of multiple levels of expensive caches and high-performance inter-chip buses.

The complete integration of processors and main memory, if it happens, could take one of two paths, and perhaps both. Successively larger memories will be placed onto the processor (or into the processor package), until the entire physical memory may be fit within the processor package. Alternatively, commodity DRAM manufacturers may begin placing small, inexpensive processors on the DRAM die, which, over time, become powerful enough to obviate the need for a large, central processor in the system. Both directions may occur simultaneously, of course, with the central processor aggregating memory as limited "intelligence" (either a small general-purpose processor or PIM-like logic [5]). While there is excitement in the community about both directions, the "memory to the processor" alternative is much less revolutionary, and it is on this alternative that we focus in this work.

Large on-processor memories are a near-certainty in the future. In Figure 1a we plot the recent growth of main memory sizes, the historical and projected [15] increases in the number of microprocessor transistors (Intel x86), and the historical and projected increase in bits per DRAM die. The solid line in Figure 1a represents a least-mean-squares regression for the existing and projected microprocessor transistor count growth. The projected growth of microprocessor transistors remains stable, doubling approximately every 18 months with no slowdown of exponential growth. In Figure 1b, we plot the percentage of processor transistor counts devoted to in-package cache memory for a range of microprocessors. The lines represent LMS regressions for a few of the processor families. While it is impossible to extrapolate quantitatively from these numbers, the trend is clear: a growing percentage—reaching 85% in some cases—of microprocessor transistor budgets are allocated to cache memory. A qualitative extrapolation implies that future microprocessors, with their vast numbers of on-chip transistors, will be mostly memory.

The rest of this paper is organized as follows. In Section 2, we explore the effects that large on-processor memories will have on the memory hierarchy. Specifically, we will quantify the constraints under which the on-chip memory will be treated as a cache, or as a fast fraction of the physical memory. We present a performance model

This work is supported in part by NSF Grant CCR-9509589, an unrestricted grant from the Intel Research Council, and equipment donations from Sun Microsystems.

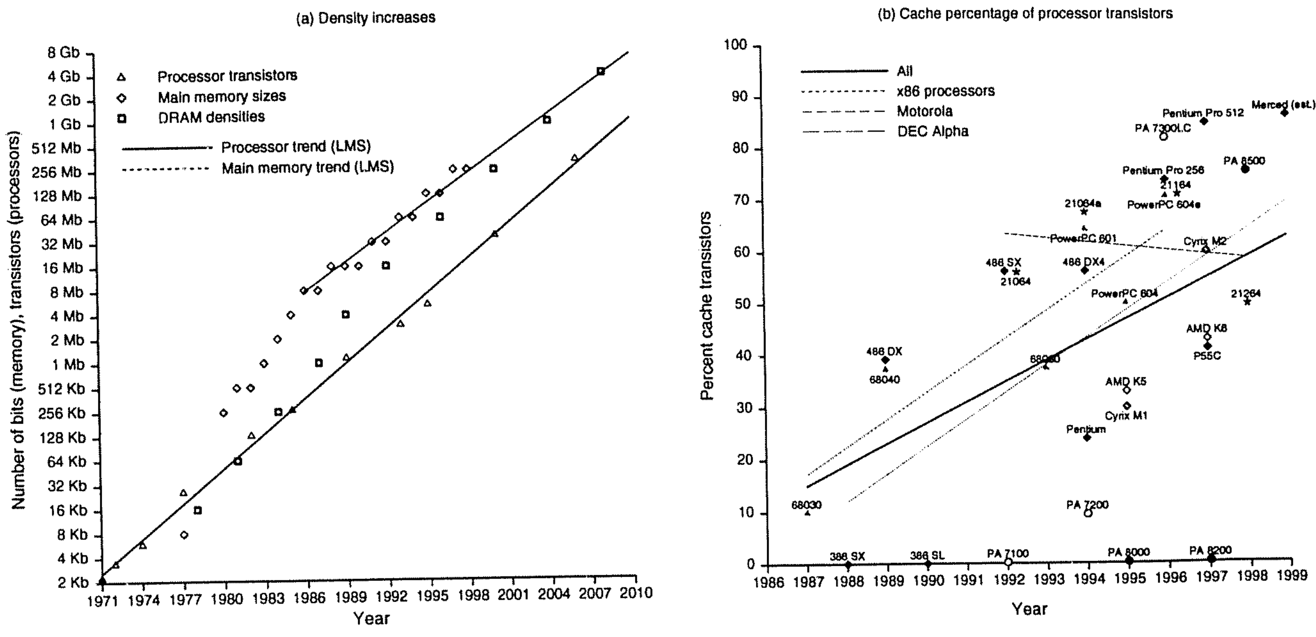

Figure 1. Future trends for processors and memories

In part (a) of this figure, we graph capacities of Intel x86 processors (transistors), single DRAM chips (bits), and main memory sizes in medium-range PCs (bits). The lines represent least-mean-squares regressions. In part (b), we plot the percentage of transistors within a processor package devoted to caches. We include LMS regressions for three of the processor families, plus one for all points.

that quantifies the cost/performance benefits of the two possibilities for a range of future technical parameters. We show that putting a small cache along with main memory on the processor chip can be cost-effective, compared to having all the on-chip memory be treated as cache. We also discuss the necessary constraints for single-chip computing, in which all of the physical memory migrates onto the processor die, and show that, given enough support in the manufacturing process for a denser memory cell, systems with no processor-less memory chips are quite conceivable within the next decade. In Section 3, we discuss issues that arise for the processor when very large memories exist on-chip. We will argue that chip multiprocessors (CMPs) used as throughput engines (as currently envisioned) are the wrong multiprocessor model for the future. We argue that multiple on-chip processors should either be used solely to enhance the performance of a single task, or that the multiple on-chip processors should each be coupled with a fraction of main memory, if they are to run distinct tasks.

2 Changing the memory hierarchy

The average number of DRAM chips in systems is decreasing over time, as the minimum granularity of memory size increases [11]. This effect results from both the increasing depth of DRAM chips (the capacity increases outstrip the growth in I/O widths of the DRAM chips) and the slowed growth in main memory sizes [9] (these two factors are not independent, of course). The dotted line in Figure 1a shows a LMS regression for main memory growths (for medium-cost PCs) from 1986 to 1998, extrapolated to 2010. We see that these trends point to PCs with single-chip memories sometime around the middle of the next decade. We also see that processor resources are converging with main memory sizes, although they are still a moderate factor apart by 2010.

If the processor die grows faster than main memory dice, the number of DRAM chips in systems goes to one. If a majority of the processor die is devoted to memory, the silicon area devoted to memory on the processor and in the one-chip main memory itself will be comparable. However, there is a substantial difference in bits per unit area (density) between current SRAM caches implemented in modern logic processes and DRAM chips fabricated in processes optimized for bit density. This density factor has been cited at various values: 15 in one case [4] and 25 to 50 in others [9, 11]. There is much ongoing work aimed at merging logic and DRAM processes—particularly for the embedded and ASIC markets—but little consensus as to whether the best direction is adding DRAM support to a logic process or vice versa [6]. If support for dense DRAM cells is added to logic fabrication processes, the amount of memory on the processor may grow to be a substantial fraction of (or even comparable to) the system's main

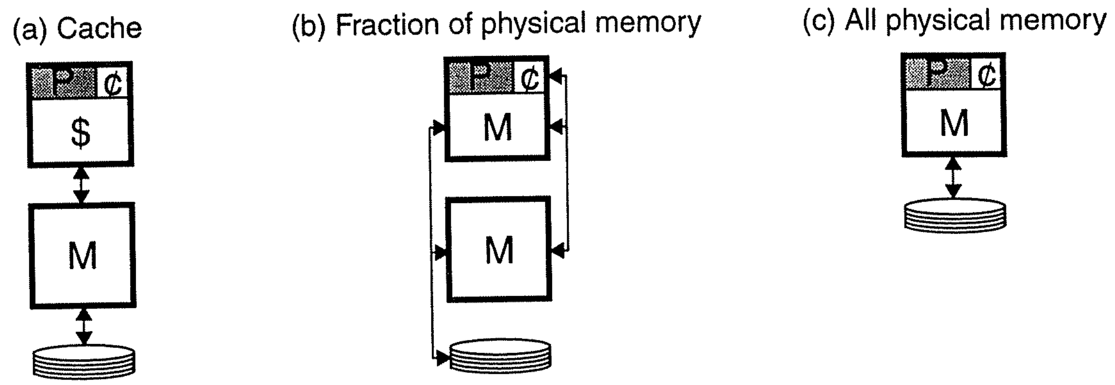

Figure 2. Possible organizations for on-processor memory

memory.

With on-processor memories that grow quite large, there are three main categories of possibilities for designing the memory hierarchy, depicted in Figure 2. The on-chip memory can be managed very much like a traditional cache (Figure 2a), the on-chip memory may be treated as a fast fraction of physical memory (Figure 2b), or the on-chip memory could be sufficiently large to contain all of the physical memory in the system (Figure 2c). We discuss the issues associated with each of these three models below. We do not consider large off-chip caches, since we believe that the combined effects of large on-chip caches and closer coupling of the processor and main memory will eliminate the cost/performance benefits of large off-chip SRAMs

For clarity of the ensuing discussion, we define the following terms: α is the ratio of the area devoted to memory (or lowest level of cache) on the processor to the total chip area in the main memory. For example, in Figure 2a, if the two chips were the same size, and the cache marked "$" took up 80% of the die, $\alpha = 0.8$. If another memory chip of the same size was added, then $\alpha = 0.4$. ρ is the density ratio between the off-chip DRAM and the on-processor memory (for example, if the DRAM array had 15 times as many bits per unit area as the on-processor cache, $\rho = 15$). The desirable main memory size for a given system is M (we assume that this a typical main memory size for a system of the generation in question).

2.1 On-chip memory as a cache

If M is too large to fit on the processor die, then the on-processor memory may be implemented as a cache or as a fraction of physical memory. The cache will pay a capacity penalty because of the larger tag overhead, comparator logic, etc. However, a cache is much more likely to reduce the number of off-chip accesses substantially, since it captures the dynamic, fine-grained nature of the dynamic reference stream (something that memory pages mapped on-chip are unlikely to do).

However, since the cache enforces the principle of inclusion, this performance comes at a cost: a fraction of the physical address space is duplicated. The magnitude of this cost overhead depends on the size of the on-chip cache, relative to M. As $\alpha \rightarrow 1$ and $\rho \rightarrow 1$ (meaning that the on-processor memory and off-chip main memory come to have similar areas and densities), the costs will grow, and the relative performance gains will diminish (discussed in Section 2.2).

2.2 On-chip memory as a fraction of physical memory

As the on-chip memory capacity grows closer to M, the performance differential between caching lines and statically mapping pages (enforcing exclusion, not inclusion as with the cache) into the memory will shrink. Mapping pages into the on-processor memory may become competitive with caching, particularly if the pages to be mapped on the processor are judiciously selected. The total system cost of this scheme is less than for caching. Since some of the physical memory is on-processor, less needs to be present outside of the processor to provide a total of M in the system (for simplicity, we ignore minimum granularity issues here). Thus, as $\alpha \rightarrow 1$ and $\rho \rightarrow 1$, treating the on-chip memory as a fraction of main memory provides superior cost/performance to caching.

There are many techniques by which pages could be chosen for placement into the processor. These include round-robin placement (simple but low performance), simple heuristics (such as mapping in the top of the stack and the text segment, if possible), profiling to determine a set of "hot" pages, having the OS "promote" and "demote" pages upon a page fault (based on usage counts), or perhaps even some sort of dynamic remapping.

A possible optimization to having main memory on-chip is to provide a relatively small on-chip cache for off-chip data along with the on-chip main memory. This cache captures some of the dynamic locality in the off-chip accesses without substantially increasing the cost (this scheme bears some similarity to DASH's remote access cache [8]). In the this paper, we present profiled page assignments, as well as the hybrid caching scheme.

2.3 A cost/performance model

To evaluate the relative merits of these schemes, we developed a simple cost/performance model. The chosen, fixed parameters are as follows: D is the processor die size, P is the fraction of the die dedicated to the processor and L1 cache, and f is the bits per unit area on the processor die. $A = (D-P)f$ is the area dedicated to on-chip memory. c_{off} is the cost per bit of off-processor memory. M is the main memory size.

We will vary one parameter: l, which is the ratio of the latency of an off-chip memory access to an on-chip memory access. Finally, we will measure m (the on-chip miss rate) experimentally. m is a function of the on-processor memory capacity (e.g., A) and organization (cache, fraction of main memory, and hybrid).

We can determine the performance of a given memory organization as the inverse of the average memory access latency, normalized to the on-chip memory access time:

$$P = 1/(ml + (1-m)) \qquad (1)$$

The cost of the memory organization is:

$$C = (M-A)c_{off} \qquad (2)$$

where A in this case is the number of bits on the processor die functioning as a fraction of main memory, for each given experiment. Thus, by maximizing the performance/cost:

$$P/C = 1/((ml + 1 - m)((M-A)c_{off})) \qquad (3)$$

for each of A and l, we can determine the most cost-effective organization (according to our metric).

We measured m experimentally using the SimpleScalar tool suite [1]. We simulated the on-chip memory performance of eight of the SPEC95 benchmarks, using the "test" inputs. The benchmarks were `compress`, `gcc`, `go`, `hydro2d`, `mgrid`, `su2cor`, `swim`, and `wave5`. We chose a value of 0.01 for c_{off}, and we set $fA = M/(2^i), i = 1, 2, \ldots 7$ where M for each experiment is the data set size of the application being simulated (more intuitively, the on-chip memory will have enough transistors for an on-chip cache of 1/2, 1/4, ..., 1/128 of the program data set size).

For each benchmark, we simulated three memory hierarchy organizations. The first assumed that all main memory resided off-chip, and that the processor contained a cache ranging from 1/2 to 1/128 of the main memory size. The second assumed that the on-chip memory was managed as a fraction of main memory. Since caches pay an extra area overhead (typically between 10% and 20%), we scaled up the capacity of the on-chip main memory an additional 15% over that of the cache. In the third organization, we have a main memory the same size as the cache in the first organization, and instead of using the 15% of area harvested from the cache tags as extra memory, we use a cache that is 1/8 (12.5%) of the on-chip main memory size. These three organizations are thus all roughly equivalent in area.

Since we are attempting to establish an upper bound on the potential performance of on-chip main memories, we statically map the most frequently accessed pages into the on-chip main memory (i.e., we assume a perfect oracle). The caches are 2-way set associative, with LRU replacement and 32-byte lines. The pages in main memory are 4KB each. We show the data set sizes, plus the sizes of the structures we used in the 1/2 experiment, in Table 1.

In Table 2 we list the percentage of memory operations that go off-chip for each experiment (cache misses, or accesses to off-chip banks in lieu of on-chip banks). Even though the fraction of main memory on-chip has a 15% larger capacity in each experiment, and the most heavily accessed pages are mapped on-chip, the miss rates for the cache is much lower in every case.

When the small cache replaces the extra 15% of the on-chip main memory pages, however, the page+cache scheme exhibits fewer off-chip accesses than does the plain cache scheme on numerous occasions. As the on-chip memory grows larger, the performance of the hybrid scheme improves relative to the pure caching scheme. When the on-chip memory size is half that of the program data set, the hybrid scheme generates fewer off-chip accesses than the pure cache for four of the eight benchmarks we evaluated.

In Figure 3 and Figure 4 we show the values of our performance/cost model for a range of off/on-chip memory access latency ratios ($m = 10, 50, 100, 500$). As can be inferred from Table 2, the pure main memory-on-chip scheme is never more cost-effective than the alternatives. For low and moderate m, however, the hybrid scheme is frequently cost-effective for very large (1/4 and 1/2 of the program data set) on-chip memories (five of the eight benchmarks). As l grows, the pure caching scheme does better in a few cases, since in those cases the pure cache generates fewer off-chip accesses, which is of paramount importance when off-chip accesses become very expensive.

These results indicate that if indeed processors begin to have on-chip memories that are within a small constant factor of the main memory size, it may be cost-effective to manage most of the on-chip memory as a fraction of main memory, provided that a smaller cache is still used to cache off-chip data.

Benchmark	go	swim	su2cor	hydro2d	mgrid	gcc	compress	wave5
Data set size	612K	14M	8M	8M	7M	2M	436K	41M
Cache size	256K	8M	4M	4M	4M	1M	256K	16M
Scaled pages	74p	2355p	589p	1178p	1178p	147p	74p	4710p
Pages + cache	64p 8K	2048p 256K	512p 64K	1024p 128K	1024p 128K	128p 16K	64p 8K	4096p 512K

Table 1: Data set and memory sizes for SPEC95 simulations
In the first row of this table, we show the data set sizes for the eight SPEC95 benchmarks we simulated. The following three rows show the memory structure sizes we used in the three experiments ("p" stands for pages). These sizes are shown for the experiment in which the on-chip memory is 1/2 of the main memory size. For the other experiments, the second through fourth rows should be scaled down accordingly (e.g. for the 1/4 experiment, every number in rows two through four should be divided by two).

Benchmark	Experiment	1/128	1/64	1/32	1/16	1/8	1/4	1/2
099.go	Cache	14.3	8.4	4.3	1.741	0.770	0.28	0.04
	Page	66.6	58.3	45.5	34.7	19.0	6.0	0.44
	Page+cache	46.9	36.5	25.2	12.0	3.8	0.63	0.07
102.swim	Cache	3.6	3.6	3.6	3.6	3.5	3.2	1.7
	Page	64.5	62.9	59.8	54.1	46.1	30.1	11.2
	Page+cache	47.5	31.9	14.7	3.3	3.0	2.5	1.3
103.su2cor	Cache	4.1	3.4	2.6	0.9	0.39	0.15	0.04
	Page	71.3	64.2	51.8	34.8	21.0	13.9	6.6
	Page+cache	8.3	5.5	4.0	2.5	1.3	0.78	0.44
104.hydro2d	Cache	5.2	5.2	5.2	5.1	4.7	3.2	1.3
	Page	66.6	65.1	62.0	56.15	48.1	33.7	15.2
	Page+cache	6.1	5.9	5.4	4.8	3.9	3.0	1.5
107.mgrid	Cache	1.8	1.5	1.2	1.1	1.1	1.0	0.73
	Page	76.6	75.0	72.0	66.2	54.3	31.8	5.3
	Page+cache	8.6	3.6	1.9	1.8	1.5	1.1	0.39
126.gcc	Cache	2.2	1.3	0.78	0.45	0.21	0.09	0.03
	Page	82.5	67.2	54.9	37.5	24.9	16.5	7.4
	Page+cache	22.3	14.4	8.1	4.3	2.1	1.1	0.42
129.compress	Cache	6.3	5.4	4.5	3.8	3.1	2.4	1.6
	Page	43.7	35.2	27.3	23.3	18.2	12.8	5.5
	Page+cache	16.3	10.1	7.5	6.0	4.7	2.4	1.1
146.wave5	Cache	0.66	0.50	0.42	0.37	0.33	0.30	0.25
	Page	17.7	16.0	13.0	8.2	4.5	3.0	1.43
	Page+cache	0.95	0.77	0.55	0.40	0.31	0.23	0.13

Table 2: Miss rates for on-chip memory structures
In this table we list the percentage of memory operations that go off-chip for each of the benchmarks and the three experiments. The fraction in the header row represents the size of the on-chip memory (approximately that fraction times the program's data set size).

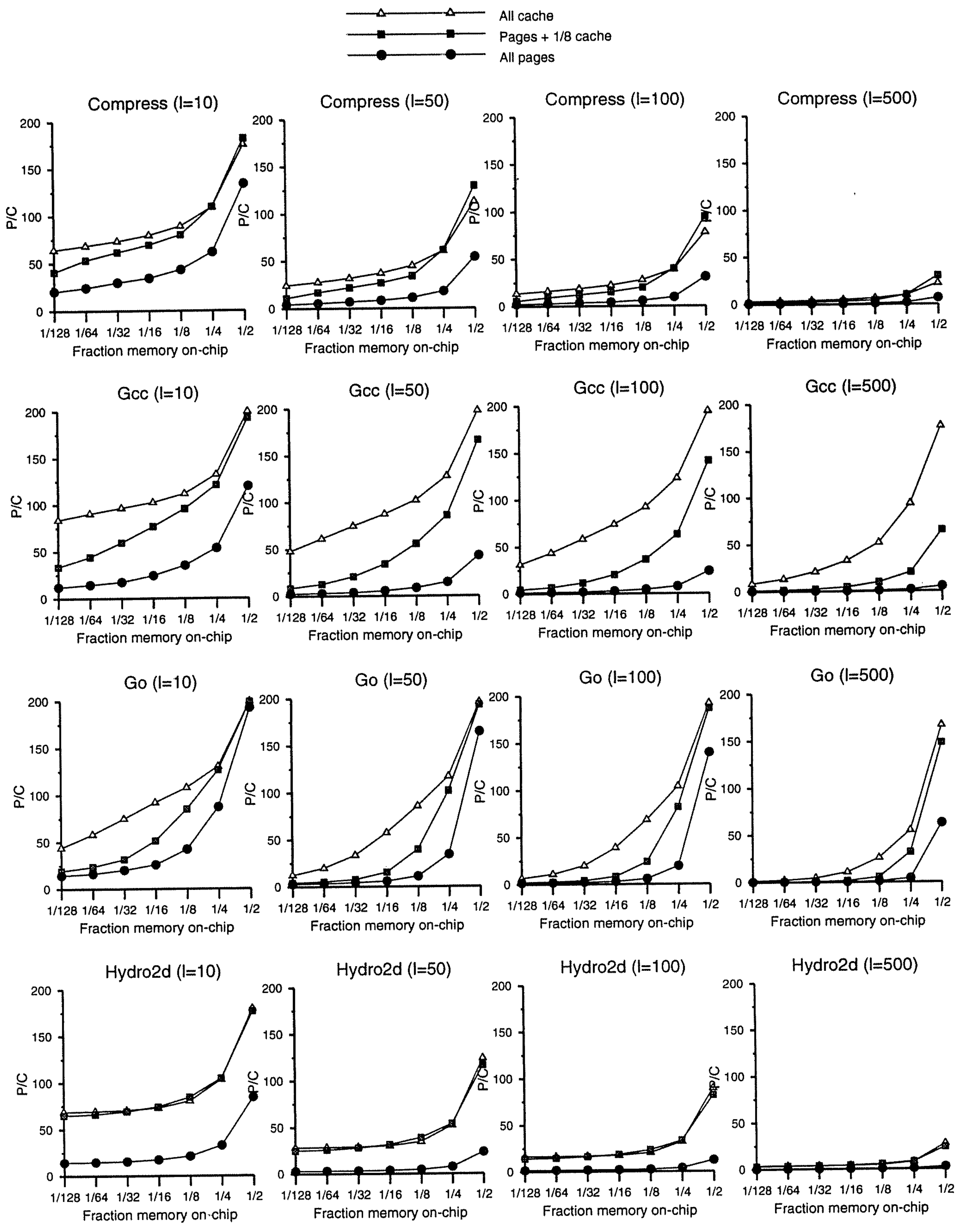

Figure 3. Performance/cost graphs for four SPEC95 benchmarks

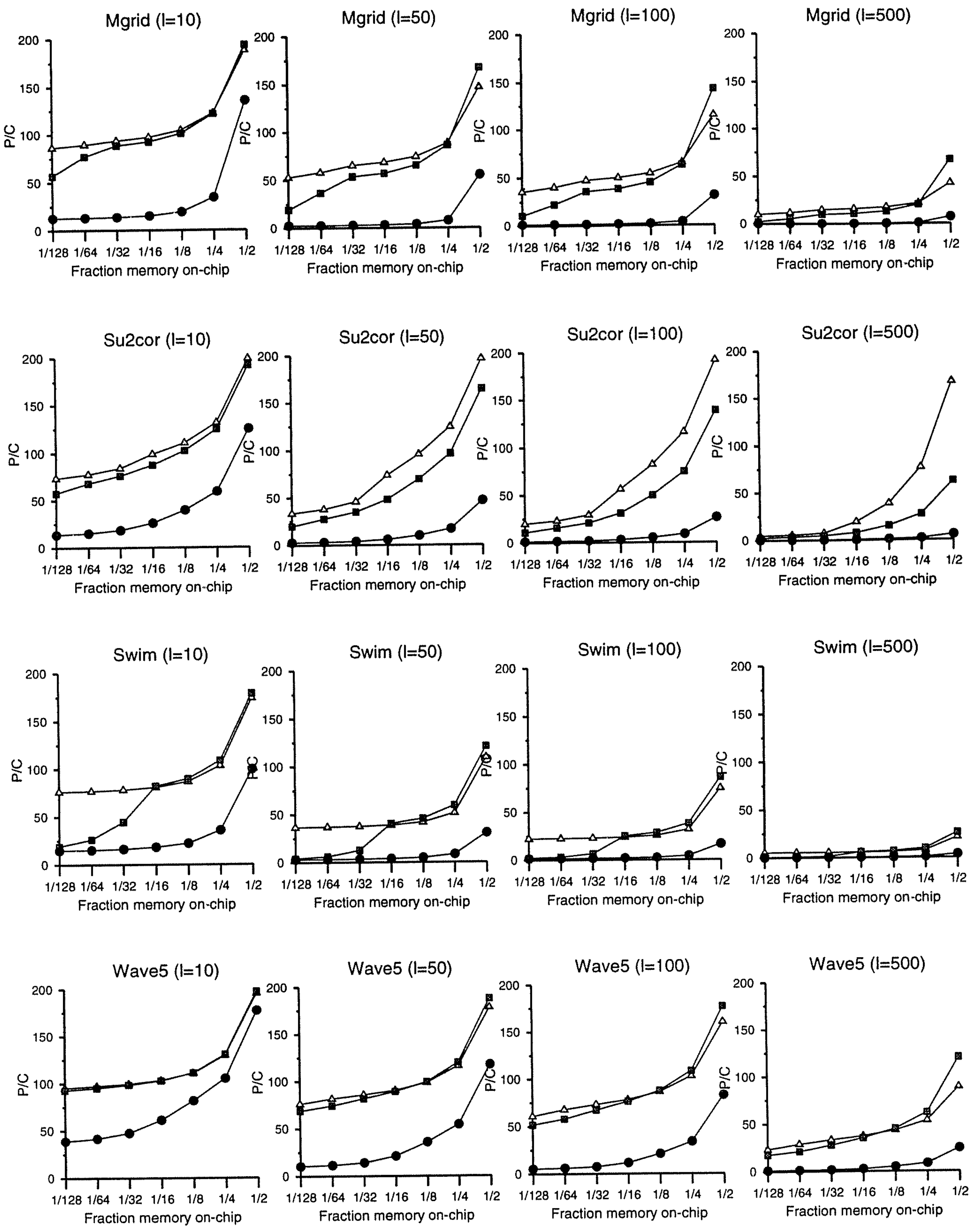

Figure 4. Performance/cost graphs for four more SPEC95 benchmarks

2.4 Single-chip computers

In Figure 1a, we see that projected main memory sizes cross single DRAM densities (thus projecting single-chip main memories) and begin to converge with on-processor real estate. The memory capacity of these processor chips will be quite large, with the exact amount depending on the on-chip memory cell density improvements brought about through hybrid memory/logic processes.

We can quantify the circumstances under which all of the main memory will exist on one or more processor chips. We assume that $\beta = A/D$ is the percentage of memory on the processor, that k is the number of transistors needed to implement one bit of storage in logic-process SRAM, and that $g(t)$ is the vertical gap between the processor transistor line and the main memory size line in Figure 1a for any year t. Recall that ρ is the density difference between unoptimized, logic process SRAM and a heavily optimized, DRAM process cell. Let $\rho'(t)$ represent the minimal factor improvement in processor memory density necessary to eliminate "dumb RAM" in a given year t. We can calculate $\rho'(t)$ as follows:

$$\frac{\beta\rho'(t)}{k} \geq \frac{g(t)}{2} \Rightarrow \rho'(t) \geq \frac{kg(t)}{2\beta} \qquad (4)$$

β/k is the normalized amount of memory on the processor, barring process support (when multiplied by $\rho'(t)$, we obtain the normalized amount of memory with process support). That amount must be greater than $g(t)/2$, meaning that the amount on the processor should be at least half of the desired main memory size. If it is half, then the desired main memory size may be reached simply by adding another processor, thus eliminating the need for more than one type of chip in the system and improving economies of scale (to certain vendors' dismay). As an example, if we assume that processor transistors are 95% memory in 2010, and still use 6T SRAM cells, then, since $g(2010) \cong 6.1$, we obtain $\rho'(2010) \geq 19.2$. This factor is within the realm of the density differentials between these two memory technologies (logic SRAM cells and optimized DRAM cells), so we conclude that single-chip computers are feasible within a 15-year time frame (assuming the trends continue as presented in this analysis, which is certainly not necessarily true).

Since we showed that on-chip memories a quarter the size of main memory (with the hybrid cache support) can be cost effective for many benchmarks (at least under our performance/cost model), we now estimate $\rho'(2001)$ as another illustrative example. $(\beta\rho'(t))/k$ in this case must be greater than $g(t)/4$ (since we are now considering a quarter of the on-chip main memory, as opposed to a half as before). $g(2001) \cong 11.3$, so using the same assumptions as in the preceding example, we obtain $\rho'(2001) \geq 18.9$, which implies that it will be possible to fit a quarter of the main memory on-chip by 2001 with a sufficiently good merged logic/DRAM process.

3 Changing the processor

In this section, we discuss the implications that large on-chip memories have for the future of chip multiprocessors (CMPs).

On-chip resources will soon be sufficient to permit multiple, high-performance processors to be placed on the same die. Conventional wisdom is that such CMPs would run parallel programs when able, and act as throughput engines to run multiple programs in parallel when parallel codes were not available. The data in Figure 1b hint as to why the latter model is unlikely to succeed. The increasing percentage of the processor die devoted to memory results from the growing relative cost of off-chip accesses. Processor designers thus put on as much cache as they can in an attempt to contain the working set of the target applications. If the dominant workloads had small, well-contained working sets that fit easily within small on-chip caches, industry would find some other use for the transistors on the die to improve performance (such as massive branch prediction tables).

The ratio of on-chip memory transistors to processor logic transistors ranges from one to nine, and is still growing. If designers wish to place multiple processors on the same chip to run independent jobs, they will need to greatly increase the size (or number) of the on-chip caches to prevent massive conflicts among working sets. This increase (scaling up the die size by the same factor as the number of processors) is likely to be far more expensive than simply using multiple separate processors for high throughput (since the cost increases dramatically as the die size grows). If the cache working sets of future applications grow more slowly than do on-chip resources, CMPs as throughput engines may become more feasible, however.

3.1 When to use a CMP

A CMP would be cost-effective when the multiple processors could share data in the on-chip memory. While processors running a parallelized program are certainly likely to share code (we include parallel instances of a task, such as `pmake`, in this category), the datasets of large programs that lend themselves well to parallelization (computational fluid dynamic codes, for example) tend to have relatively distinct working sets. Such codes would thus run more cost-effectively on multiple separate processors, even though they would benefit from the lower on-chip communication latencies.

One good candidate for a CMP is a coarse-grained speculative processor—such as a Multiscalar processor [14]—in which multiple on-chip execute speculative, temporally ordered tasks from the dynamic instruction stream. These processors tend to exhibit much data sharing in their shared caches, and thus the overlapped working sets would allow the processors to remain utilized without requiring vast increases in the on-chip cache size.

3.2 Looking farther ahead

So long as the on-chip memory is treated as cache, with its contents dynamically changing from cycle to cycle, the cost-effectiveness of CMPs for running independent jobs or even many parallel programs will be limited.

Once the on-chip memory banks are treated as physical memory, however, this situation changes fundamentally. The increasing RC delays on global intrachip wiring—as feature sizes are scaled down—will drive future processor chips to consist of multiple partitioned modules. One possibility is coupling multiple processors with the multiple memory banks on a chip, such that each processor is closely coupled with a nearby part of the on-chip memory, and can run extremely fast when data it needs are in its local bank.

In this model, the question of resource allocation disappears somewhat, as the number of processors allocated to a job becomes proportional to the amount of physical memory that the job requires. There are fewer issues with conflicting working sets, because if a job requires all of the memory on a chip, all of the processors on that chip will be dedicated to that job. If part of main memory does eventually migrate onto the processor die (which is certainly possible as discussed in Section 2), it is likely that economies of scale will drive the system to contain only that one type of chip.

While this distributed processor/memory bank model may be attractive from a design perspective, the perennial problem of how to program this architecture looms. Traditional parallelization techniques will work for many codes, but there are many others that parallelize either poorly or not at all.

A good candidate for such codes is the DataScalar architecture [3]. Each participating processor runs the same program, performing redundant computation. In a CMP-based DataScalar system (or multi-CMP-based), when a processor loads an operand from its local bank, it broadcasts the operand to the rest of the participating processors. All communication is one-way, and thus cuts down on the latency of requests for operands in remote memories (a cross-chip request may take tens of cycles in future processors). Processors that find multiple consecutive, dependent operands in their local banks can run ahead of the others on that dependence chain, broadcasting the entire chain to the other processors much earlier than would normally be possible. Best of all, the base DataScalar model is fully transparent, and requires no recompilation or binary rewriting ... unmodified codes that were not good candidates for parallelization can thus exploit the multiple processors and improve performance.

DataScalar architectures require global broadcasts, however, and are thus unlikely to scale to large numbers of nodes, whether in a one-chip system with many processor/bank modules, or in a system with multiple such chips. By dividing computational slices among subsets of the participating processors, broadcasting only to local subsets, and allowing much deviation from the base DataScalar model (in which processors may dynamically decide to perform private computation, broadcasting only a result or perhaps directions to other processors), we can improve the scalability of the computation, allowing a huge range of codes to run efficiently on a tightly integrated, IRAM/CMP system. More important, heroic software support is not required for codes to run on this system (but may improve performance on codes that are amenable to such analysis).

4 Summary

In this paper, we have examined several issues concerning processor/memory integration from the CPU (logic process) perspective. We speculated about the directions in which the increasing size of on-chip memories will drive future processors and systems. We first reiterated others' point that if the current trends hold (which of course they may not), we may have only one memory chip in future PC-class systems. We showed that the percentage of processor transistors devoted to memory is high (and growing), and that the processor transistor budget is slowly converging with main memory size.

We then analyzed the implications of these trends for the memory hierarchy. We showed that, given on-processor memory cells that are comparable in density to DRAM cells, that the processor may eventually contain the entire main memory. We then used a cost/performance model to examine the space between here and single-chip systems, showing the point at which it may become cost-effective to treat a part of on-chip memory as main memory instead of as a cache.

Finally, we engaged in a qualitative discussion of how CMPs, as traditionally thought of, are unlikely to emerge due to memory restrictions. We believe that the CMP model makes much more sense when the processors are each coupled with a different main memory bank (homogeneously throughout the system), rather than competing for space in a shared cache. We are currently exploring new techniques (derived from our DataScalar work) to allow hard-to-parallelize codes to run efficiently on such a system without mandating excessive or impractical software support.

Acknowledgments

The author would like to thank David Wood for his helpful comments on an earlier draft, and Jim Goodman for both good feedback and for not firing the author while he worked on this paper.

References

[1] Doug Burger and Todd M. Austin. The SimpleScalar Tool Set Version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin, Madison, WI, May 1997.

[2] Doug Burger, James R. Goodman, and Alain Kägi. Memory Bandwidth Limitations of Future Microprocessors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 79–90, May 1996.

[3] Doug Burger, Stefanos Kaxiras, and James R. Goodman. DataScalar Architectures. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, May 1997.

[4] Richard C. Foss. Implementing Application Specific Memory. In *Proceedings of the 1996 International Solid-State Circuits Conference*, pages 260–261, February 1996.

[5] Maya Gokhale, Bill Holmes, and Ken Iobst. Processing in Memory: the Terasys Massively Parallel PIM Array. *IEEE Computer*, 28(3):23–31, April 1995.

[6] Osamu Kimura, Richard Crisp, Michael Nagy, Henry Lie, Roelof Salters, Kenji Numata, Takao Watanabe, and Kazunori Saitoh. Panel Session: DRAM + Logic Integration: Which Architecture and Fabrication Process. In *Proceedings of the 1997 International Solid-State Circuits Conference*, February 1997.

[7] Jeffrey Kuskin, David Ofelt, Mark Heinrich, John Heinlein, Richard Simoni, Kourosh Gharachorloo, John Chapin, David Nakahira, Joel Baxter, Mark Horowitz, Anoop Gupta, Mendel Rosenblum, and John Hennessy. The Stanford FLASH Multiprocessor. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 302–313, April 1994.

[8] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica Lam. The Stanford DASH Multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[9] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A Case for Intelligent RAM. *IEEE Micro*, 17(2):34–44, March/April 1997.

[10] David Patterson, Tom Anderson, and Kathy Yelick. The Case for IRAM. In *Proceedings of HOT Chips 8*, Stanford, CA, August 1996.

[11] Steven A. Przybylski. *New DRAM Technologies: A Comprehensive Analysis of the New Architectures.* MicroDesign Resources, Sebastopol, CA, 1994.

[12] Steven K. Reinhardt, James L. Larus, and David A. Wood. Tempest and Typhoon: User-Level Shared Memory. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 24–33, April 1994.

[13] Ashley Saulsbury, Fong Pong, and Andreas Nowatzyk. Missing the Memory Wall: The Case for Processor/Memory Integration. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 90–101, May 1996.

[14] Guri Sohi, Scott E. Breach, and T.N. Vijaykumar. Multiscalar Processors. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, May 1995.

[15] Albert Yu. The Future of Microprocessors. *IEEE Micro*, pages 46–53, December 1996.