Evaluation of Existing Architectures in IRAM Systems

Ngeci Bowman, Neal Cardwell, Christoforos E. Kozyrakis, Cynthia Romer and Helen Wang [*]

Computer Science Division
University of California–Berkeley
{bowman,neal,kozyraki,cromer,helenjw}@cs.berkeley.edu

Abstract

Computer memory systems are increasingly a bottleneck limiting application performance. IRAM architectures, which integrate a CPU with DRAM main memory on a single chip, promise to remove this limitation by providing tremendous main memory bandwidth and significant reductions in memory latency. To determine whether existing microarchitectures can tap the potential performance advantages of IRAM systems, we examined both execution time analyses of existing microprocessors and system simulation of hypothetical processors. Our results indicate that, for current benchmarks, existing architectures, whether simple, superscalar or out-of-order, are unable to exploit IRAM's increased memory bandwidth and decreased memory latency to achieve significant performance benefits.

1 Introduction

One proposed solution to the growing gap between microprocessor performance and main memory latency is to integrate a processor and DRAM on the same die, an organization we refer to as Intelligent RAM (IRAM) [10]. Because all memory accesses remain on-chip and the memory bus width is no longer influenced by pin constraints, IRAM should improve main memory bandwidth by two orders of magnitude and main memory latency by one order of magnitude.

It is not clear what processor microarchitecture will best be able to turn these advantages into significant application performance benefits. However, there are several reasons to prefer existing general-purpose microarchitectures, such as wide superscalar, dynamic (out-of-order) execution, or even simple in-order RISC CPUs, for IRAM systems. Current organizations already achieve impressive performance across many classes of applications, including personal productivity applications, graphics, databases, scientific computation, and software development. Furthermore, the performance trade-offs for such architectures are well-understood, and we already have the tools and know-how to design, debug, and tune both the architectures and the software that runs on them. Perhaps most important is the consideration of binary compatibility for existing software: there already exists a large body of system software and applications that could be used "out of the box" if an existing architecture is adopted. Higher application performance would only require tuning programs and compilers to the specific characteristics of the new memory hierarchy.

For this work, we evaluated the performance implications of this evolutionary approach of combining an existing microarchitecture with an IRAM memory hierarchy. Our investigation had two complementary aspects. First we measured and analyzed the performance of applications on two existing microprocessors – one simple superscalar processor and one complex out-of-order processor – and used these results to predict the performance of a hypothetical, otherwise identical system with an IRAM memory hierarchy. Subsequently we used complete system simulations to obtain a detailed performance evaluation of simple IRAM and conventional processors.

The remainder of this paper is organized as follows: Section 2 presents the main implementation and architectural considerations for IRAM systems and section 3 describes the benchmarks used in this study. Section 4 discusses our analytic evaluation of IRAM implementations of two existing architectures. Section 5 describes the results of simulations of simple conventional and IRAM systems. Finally, section 6 presents our conclusions from these studies and suggests directions for future IRAM research.

2 Implementation and Architectural Considerations

There are several significant ways in which IRAM systems will differ from today's microprocessors: the integration of DRAM on-chip, the resulting high-bandwidth main memory bus, the elimination of L2 caches, and a potential slow-down incurred by logic in today's DRAM processes.

The primary difference between IRAM systems and conventional systems will, of course, be the integration of large amounts of DRAM on-chip. For example the DEC Alpha 21164 processor, with 16KBytes of first level caches and 96KBytes of L2 cache, occupies $299mm^2$ in a $0.5\mu m$ CMOS process. In a 256Mbit DRAM $0.25\mu m$ CMOS process [12],

[*] This work was supported by DARPA (DABT63-0056), the California State MICRO Program and research grants and fellowships from Intel, Sun Microsystems and the National Science Foundation.

this would take up approximately $75mm^2$, or one fourth of the die area. This allows up to 24MBytes of on-chip DRAM memory in the remaining area. While this may not be sufficient by itself for high-end workstations, it is enough for low-end PCs and portable computers. The memory access time for such a system could can be as low as 21ns, since off-chip communication over high-capacity busses has been eliminated [10]. This is up to ten times faster than the main memory access times of current conventional systems.

Second, since these access times for the on-chip main memory [8] can be comparable to that of SRAM L2 caches today, using die area for an L2 cache provides little performance improvement. This area can instead be used for on-chip DRAM, which is more than 10 times as dense [5]. Consequently, the IRAM systems we consider in this study have no L2 caches.

Third, because main memory will be integrated on-chip, IRAM systems can be designed with memory busses as wide as desired. Given that the memory bus will connect the main memory to the L1 cache, the bus width should probably be equal to the L1 cache block size.

Finally, initial IRAM implementations may suffer from logic speed degradation. Existing DRAM technology has been optimized for density and yield, so logic transistors and gates in DRAM processes are slower than those in corresponding logic processes. This can translate to a processor clock frequency up to 1.5 times slower than that of similar architectures implemented with conventional logic processes [5] [10]. Fortunately, high-speed logic in DRAM processes has already been demonstrated in prototype systems, so it is expected that within a few years logic in DRAM chips will be as fast as microprocessor logic.

3 Benchmarks and Applications

Table 1 describes the benchmarks and applications used for this evaluation.

SPEC 95 [1] is the current industry-accepted standard for uniprocessor performance evaluation. We used three of the floating point programs of the suite (*tomcatv, su2cor, wave5*) and all eight integer benchmarks. All SPEC 95 programs were compiled with base settings and run on the complete reference inputs.

Unfortunately, SPEC 95 benchmarks do not exercise all levels of memory hierarchy and are not considered representative of possible workloads for current and future systems. To study the behavior of a broader range of applications, we employed three additional benchmarks for this study. *Mpeg_encode*, which encodes static frames into a MPEG video file, is a representative multimedia application. *Linpack1000*, a double precision linear equation solver, is characteristic of many scientific codes. Finally, disk-to-disk *sorting* represents the databases field, where operations are performed on large collections of data.

Benchmark	**Description**
`tomcatv`	Mesh Generation: generates a 2-D mesh
`su2cor`	Quantum Physics: computes masses of elementary particles using a Monte Carlo method and the Quark-Gluon theory
`wave5`	Electromagnetics: solves Maxwell's equations on a Cartesian mesh using a variety of boundary conditions
`gcc`	Compiler: uses the GNU C compiler to convert a set of preprocessed source files into Sparc assembly language
`compress`	Compression: compresses large files using adaptive Lempel-Ziv coding
`li`	Lisp Interpreter: uses a Lisp interpreter to interpret a set of programs
`ijpeg`	Imaging: performs JPEG image compression
`perl`	Perl Interpreter: Uses Perl to perform text and data manipulation
`go`	Artificial Intelligence: Plays the game GO against itself
`vortex`	A single user O-O database transaction benchmark. Builds and manipulated 3 interrelated databases
`m88ksim`	Simulator: Simulates the Motorola 88100 processor
`mpeg_encode`	Video Encoding: encodes 48 720x480 color frames into a MPEG video file
`linpack1000`	Equation Solver: 1000x1000 sparse linear equation solver (double precision)
`sort`	Sorting Program: disk-to-disk sort of 100-byte records of a 21MByte database

Table 1: *The benchmarks and applications used in this study.*

4 Evaluating IRAM through Measurement and Extrapolation

Our initial approach to evaluating current microarchitectures implemented as IRAM systems was analytical. The goal was to analyze the execution behavior of applications on two existing organizations and estimate the effect of an IRAM implementation on the total execution time and its individual components. This allowed us to compare the potential performance of the IRAM implementations to that of the originals.

The two architectures examined in this study were the DEC Alpha 21064 [4] and the Intel Pentium Pro [2]. The Alpha 21064 uses a simple dual-issue, in-order execution organization with direct-mapped, blocking caches. By contrast, the Pentium Pro employs an aggressive triple-issue architecture, with out-of-order and speculative execution, a deeper pipeline, and 4-way set-associative, non-blocking caches. Table 2 summarizes the main characteristics of the two processors. These two organizations represent contrasting approaches to microprocessor architecture. Alpha's simplicity leads to implementations with extremely high clock frequencies. With the second approach, performance comes from advanced architecture techniques like out-of-order execution. Choosing these two architectures enabled us to estimate the potential IRAM performance for both approaches and evaluate the benefit from using similar advanced techniques within IRAM systems.

The potential IRAM implementations of the two architectures

	Aplha 21064	Pentium Pro
Pipeline	in-order	out-of-order
CPU Frequency	133MHz	200MHz
Issue Rate	2-way	3-way
L1 Configuration	8KB I + 8KB D	8KB I + 8KB D
L1 Associativity	Direct Map	4-way
L1 Access Time	22.5ns	15ns
L2 Configuration	512KB	256KB
L2 Associativity	Direct Map	4-way
L2 Type	Off-chip SRAM	Off-chip SRAM
L2 Access Time	37.5ns	20ns
Memory	64MB EDO DRAM	64MB EDO DRAM
Total Latency	180ns	220ns

Table 2: *Architecture characteristics of the Alpha 21064 and the Pentium Pro.*

that we considered for evaluation differ from the originals in only two respects. IRAM models include 24MBytes of on-chip DRAM used as main memory but no second level caches. The remaining system components are exactly the same, including the memory bus width. Therefore, the main parameters that lead to performance differences are the types and delays of memory accesses and the speed of logic in the IRAM systems, which determines the clock frequency.

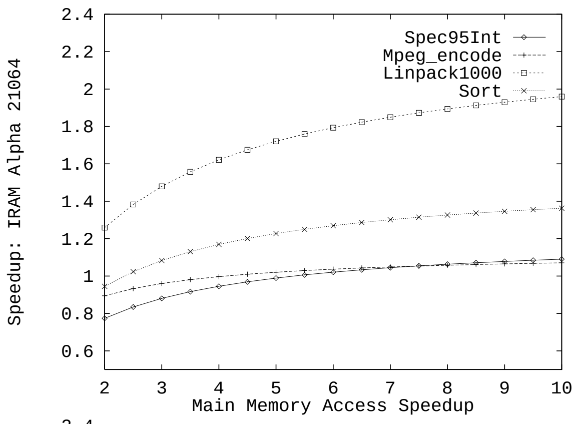
4.1 Methodology

We estimated the performance of the IRAM Alpha and Pentium Pro implementations by analyzing the components of program execution times on the original implementations. The on-chip programmable hardware counters available in both processors were used to perform detailed profiling and measurements without affecting the program behavior [4]. The measurements included processor and memory hierarchy events like issue and stall cycles, branch mispredictions, TLB misses, and memory references and miss rates at each level of the memory hierarchy. For this study, we concentrated mainly on memory-related events whose type and delay may change due to the addition of on-chip DRAM.

The execution time of a program on an IRAM implementation can be predicted by examining how the parameters of IRAM systems affect each of the execution time components on the original architecture. Time spent on events unrelated to the memory hierarchy, like issue cycles or stalls due to mispredictions, is only affected by a potential clock frequency difference. Keeping in mind that L1 cache misses in the IRAM implementations are served by the on-chip DRAM, the time spent in the memory hierarchy is equal to the product of the L1 miss count and the memory access time. A simplified version of the model we used for the execution time of IRAM implementations for a given application is:

$$ET = \frac{T_{computation}}{clock\_speedup} + \frac{L1\_miss\_count * T_{memory\_access}}{memory\_access\_speedup}$$

$T_{computation}$, *L1_miss_count* and T_{memory_access} are, respectively, the time spent on computation and non memory-related events, the L1 cache miss count, and the off-chip memory access time for the original implementation. *Clock_speedup* and *memory_access_speedup* are the ratios of clock frequency



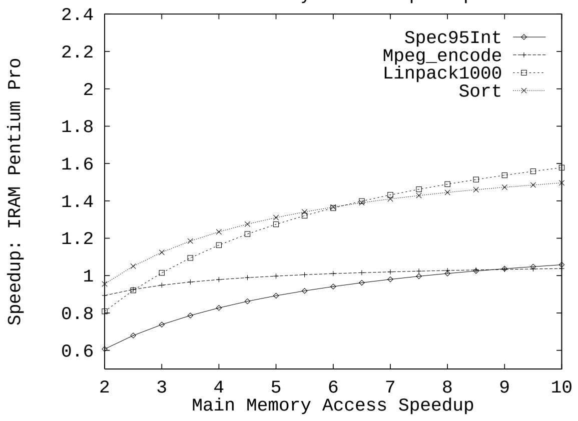


Figure 1: *Applications speedup for IRAM Alpha 21064 (top) and IRAM Pentium Pro (bottom) over the original implementations.*

and main memory access time of the original to that of the IRAM implementation. The complete equation also accounts for memory-related events like TLB and compulsory misses.

Using the Alpha and Pentium Pro execution time models, we evaluated the speedup of the two IRAM implementations over the corresponding originals for the SPEC 95 integer programs, *mpeg_encode, linpack1000* and *sort*. For both IRAM implementations we assume that the processor logic and L1 caches are as fast as in the conventional systems. Since the working sets of these applications are small enough (less than 24 MBytes) to fit completely in the on-chip DRAM, even for initial IRAM implementations, we assume that, apart from compulsory misses, no time-consuming off-chip access are necessary. Given these two facts, the calculated speedup is a best-case indication of the potential performance of the IRAM implementations of these architectures.

4.2 Results

Figure 1 shows the speedup of the two IRAM implementations over the corresponding originals as a function of main memory access speedup. For the case of the SPEC95int programs, we present the average speedup. Given that the off-chip memory access latency of current systems varies between 150 ns to 250 ns depending on the type of DRAM used, and since memory access time in IRAM systems can be as low as 21ns, this ratio is expected to vary between 2 and 10, with 5-10 being the most likely range.

The execution analysis of these applications revealed that they can be separated into two categories. The first category includes computation-intensive applications, like *mpeg_encode* and *SPEC 95*, where the dominant component of the execution time is not related to the memory hierarchy. For these programs, main memory accesses account for less than 20% of the total execution time. *Sort* and *linpack1000*, on the other hand, spend from 40% to 55% of execution time waiting for memory references, of which a significant part comes from main memory accesses.

For computation-intensive applications, IRAM implementations range from two times slower to slightly faster, depending on the speed of accesses to main memory. Since the dominant component of execution time for these applications is actual computation, in no case are they significantly helped by the faster main memory system offered by IRAM.

For memory-intensive applications, the speedups are more significant, in the range of 1.5 to 2. This is due to the large amount of time spent waiting for off-chip memory accesses in the original implementations, which is almost eliminated in the IRAM.

Main memory access speedups higher than 6 do not lead to corresponding increases in application speedup. For very low memory access speedups (close to 2), performance is surprisingly low because the on-chip DRAM in the IRAM systems is actually slower than the L2 caches of the original implementations. Though memory latency for IRAM systems is likely to be 5-10 times faster than conventional systems, figure 1 shows that IRAM designers must be careful to eliminate L2 caches only if the on-chip DRAM is sufficiently fast.

For example, for the original Pentium Pro with off-chip EDO DRAM, the L2 and main memory access times are 35 ns and 220 ns respectively. An IRAM memory speedup factor of 2 makes L1 cache misses cost 110 ns, significantly worse than the original 35 ns for L2 accesses on the Pentium Pro. Therefore, since most of the L1 cache misses for these programs are served by the Pentium Pro's L2 cache, the performance of IRAM implementations is lower than that of the conventional.

A detailed description of this analysis and its results is presented at [9].

5 Evaluating IRAM through Simulation

In addition to evaluating IRAM implementations of two existing microarchitectures using performance measurements and analytic models, we simulated several simple IRAM systems to verify these results. Detailed simulation can reflect subtle and unanticipated interactions between aspects of a complex system that analytical approaches cannot capture. Furthermore, it enables evaluation of the effect of simple enhancements to the system architecture, like increases to the main memory bus width or the cache size.

There were two problems to address before we could determine our architectural simulation space. First we had to select an appropriate existing microprocessor architecture for an IRAM. We chose to simulate a simple, single-issue, in-order RISC CPU, since today's dynamically scheduled processors are complex, energy-inefficient systems designed largely to minimize slow memory latencies, a problem which should not exist in an IRAM. Given this microprocessor architecture, our next task was to determine the means to assess the relative performance of our simulated IRAM architectures. We decided to do this by comparing them with a hypothetical conventional microprocessor chip of equal die area.

The area consumed by IRAMs will be dominated by main memory. Given current trends in software memory requirements, we estimate that an IRAM will require at least 24MBytes to run interesting desktop and portable applications. This implies a 256Mbit DRAM technology, which should be commercially available in 1998. For this reason we chose to compare hypothetical 1998 IRAM models to a conventional computer system that we think will be adequate for 1998 desktop markets. A likely configuration for a conventional machine is a split L1 cache with 64KByte instruction cache and 64KByte data cache and a large unified L2 cache, with fast SDRAM main memory. As explained in section 2, second level caches will not benefit IRAM systems, so our simulated IRAM models have a single level of cache.

One caveat when comparing such disparate architectures is selecting configurations with equal die sizes. The three main components of die area in our models are the CPU, the caches, and any other on-chip memory. Both the IRAM models and the conventional model have a simple RISC CPU core. Both IRAM and conventional models include a 128KByte L1 SRAM cache on-chip. Our conventional machine uses the remaining on-chip area for a 2 MByte L2 SRAM cache, whereas an IRAM includes 24 MBytes of DRAM, which is treated as main memory. Since DRAM is approximately 16 to 32 times more dense than SRAM embedded on a microprocessor [5], these areas should be roughly equal, again assuming a 256Mbit DRAM 0.25 μm CMOS process.

5.1 Simulation Models

Table 3 lists the parameters for the architectural models used in the simulations. To maintain a manageable simulation space we restricted ourselves to modeling three different IRAM configurations and one conventional machine. All parameters of the conventional machine remained constant for all simulations; the only parameters of the IRAM machines that varied were the CPU clock rate and the DRAM access time. A wider main memory bus width (128 Bytes) was assumed for IRAM models, to take advantage of the additional memory bandwidth. The memory bus is set to be as wide as the L1 cache block size. In order to isolate the effects of changing architectural parameters on IRAM performance, we allowed only one parameter of the IRAM model to vary at any time. The default IRAM parameters we used were a 333 MHz clock rate and a 33 ns memory latency.

We identified CPU clock rate and memory latency as the parameters of primary interest to our investigation. Since it is

	IRAM	**Conventional**
Pipeline	simple in-order	simple in-order
CPU frequency	**vary:** 333, 500 MHz	500 MHz
technology	0.25 μm DRAM	0.25 μm logic
L1 configuration	64 KB I + 64 KB D	64 KB I + 64 KB D
L1 associativity	2-way	2-way
L1 block size	128 Bytes	64 Bytes I / 32 Bytes D
L1 type	SRAM on-chip	SRAM on-chip
L1 access time	1 CPU cycle	1 CPU cycle
L2 configuration	–	2 MB unified
L2 associativity	–	2-way
L2 block size	–	128 Bytes
L2 type	–	SRAM on-chip
L2 access time	–	12 CPU cycles
memory configuration	24 MBytes (192 Mbit) DRAM on-chip	24 MBytes 166 Mhz SDRAM off-chip
memory bus width	1024 bits (128 Bytes)	128 bits (16 Bytes)
total latency	**vary:** 21, 33 ns	116 ns

Table 3: *Architectural models for IRAM and conventional machines of 1998.*

estimated that logic in initial IRAM implementations may be up to 1.5 times slower than logic in a microprocessor process, we picked two possible CPU clock rates that span this entire space. To investigate the memory latency advantages of IRAM we chose two values for IRAM main memory access latency, an optimistic one (21 ns) and a pessimistic one (33 ns).

5.2 Methodology

To simulate these conventional and IRAM architectures we extended the 1.0 release of SimOS, a complete computer system simulation tool developed at Stanford [7] [11]. The standard version of SimOS 1.0 simulates a Silicon Graphics uniprocessor or multiprocessor workstation, including MIPS processors, two levels of cache, multiprocessor memory busses, memory, disk drives, consoles, and Ethernet devices, in enough detail to boot and run a slightly modified version of IRIX 5.3. SimOS allows the processor clock rate and memory bus width to be specified; we modified it to allow for a single-level cache hierarchy and a parameterizable main memory access time.

In this case, we selected for simulation eight of the SPEC 95 floating point and integer benchmarks in addition to *linpack1000*. These were *tomcatv, su2cor, wave5, gcc, compress, li, ijpeg* and *perl*. We simulated the execution of each benchmark program on all three IRAM models as well as the conventional model.

5.3 Results

Figure 2 presents the application speedup of IRAM models over the 500MHz conventional model for the SPEC 95 programs and *linpack100*. For the case of the SPEC 95 benchmarks, we present the geometric mean of application speedups. As with the analytic results in section 4, the benchmarks fall into two classes: memory hierarchy intensive, and CPU-bound.

As with the analytic results, the *linpack1000* benchmark demonstrates the behavior of a memory-intensive application. Memory stall time accounts for 32% of execution time for the 500MHz conventional machine, but only 7% of execution time for the 500MHz IRAM, due to increased bandwidth and decreased memory latency. However, this results in only a 40% speedup for the 500MHz IRAM compared to our conventional model.

Figure 2 shows that, in accordance with with the results from section 4, simple IRAM architectures do not appear to offer large performance improvement over conventional machines for SPEC 95 applications. The maximum speedup ranges from almost 1 to 1.15. IRAM models with clock frequency of 333 MHz are actually slower than the conventional model. This is in part because the SPEC 95 applications fit well within the 2MByte L2 cache of the conventional machine, so that even the decreased memory latency of the IRAM implementations offers no significant advantage.

A more complete discussion of the simulation results can be found at [3].

6 Concluding Remarks

Both the execution analysis and simulation approaches suggest that the potential performance benefits from using existing microarchitectures in IRAM systems are limited. Such evolutionary IRAM implementations will range from equally fast for CPU intensive applications up to two times faster in the best case for some memory-intensive applications. For comparison it is useful to note that conventional processors double in speed every 18 months [6], so that similar speedups can be reached with conventional architecture and process technology improvements in less than two years, without requiring the modification to current integrated circuits fabrication and processing techniques that IRAM will entail.

The fundamental reason for this limitation is the inability of today's conventional microarchitectures to take advantage of the phenomenal main memory bandwidth that becomes available on-chip in IRAM systems [10]. Current architectures have been developed under the implicit assumption that the connection with main memory has both high latency and low

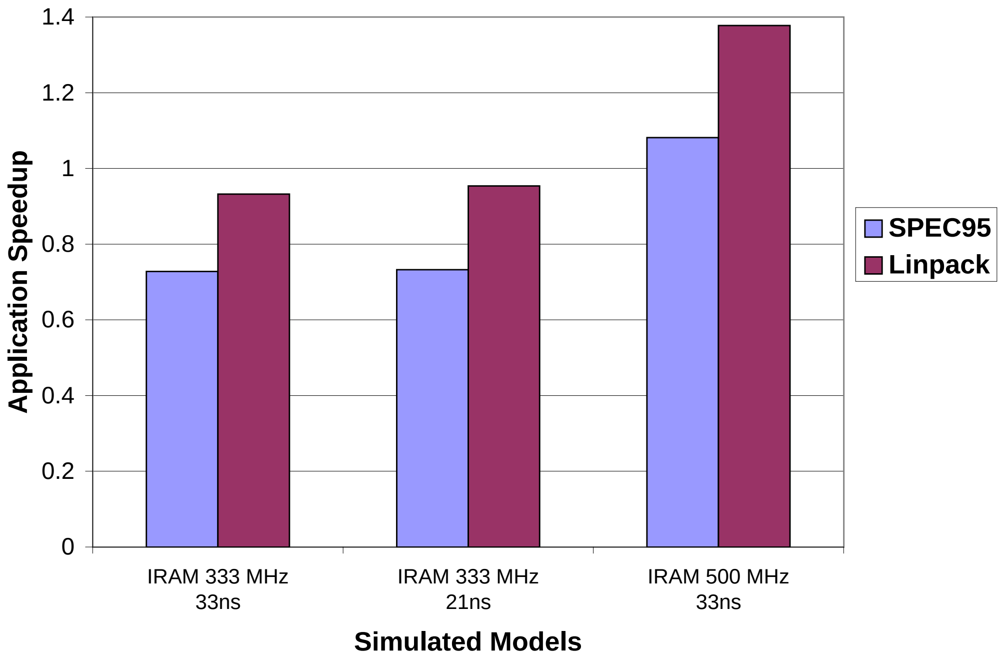


Figure 2: *Simulation results for SPEC 95 (left) and linpack1000 (right) benchmarks, presented as speedups over the execution time for the 500MHz conventional model with 116 ns main memory latency. Larger bars are better. See Table 3 for configuration details.*

bandwidth, and thus these architectures strive to minimize their use of main memory. Consequently, to use IRAM to beef up main memory for an attack on this already-small fraction of execution time on conventional microarchitectures is to ignore Amdahl's law.

Still, using simple conventional architectures in IRAM systems has some advantages that are very important in certain areas. Simple, energy-efficient RISC processor cores combined with IRAM memory systems can be very energy-efficient [5], which translates to longer battery life for portable systems. In addition, significant cost reduction derives from the higher integration level and smaller overall system area (system-on-a-chip). Therefore, IRAM implementations of existing simple microarchitectures can be an attractive alternative for the embedded and portable domains, where cost and power are more important that pure performance.

But ultimately, in order to achieve high performance in IRAM systems, new architectures must be employed. The basic characteristic of such organizations must be the ability to make efficient use of the high memory bandwidth available and to turn it into application performance. A second desirable feature is performance which scales as the capacity and speed of integrated circuits increases. Architectures that are thought to have such qualities include vector processors, multi-threaded processors, and multiprocessors-on-a-chip. Evaluating these and other, perhaps new, architectures within the IRAM context is an open area for research.

References

[1] SPEC CPU95 benchmarks. http://www.specbench.org/osg/cpu95/.

[2] Bhandarkar, D., and Ding, J. Performance characterization of the Pentium Pro processor. In *Proceedings of the Third International Symposium on High-Performance Computer Architecture* (February 1997).

[3] Bowman, N., Cardwell, N., and Romer, C. The performance of Real Applications and Operating Systems on Simple IRAM Architectures. http://www.cs.berkeley.edu/~neal/iram/simIRAM.html.

[4] Cvetanovic, Z., and Bhandarkar, D. Characterization of ALPHA AXP performance using TP and SPEC woarkloads. In *Proceedings of the 21st Annual International Symposium on Computer Architecture* (April 1994), pp. 60–70.

[5] Fromm, R., et al. The Energy Efficiency of IRAM Architectures. In *the Proceedings of the 24th International Symposium on Computer Architecture* (June 1997).

[6] HENNESSY, J., AND PATTERSON, D. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc, San Francisco, CA, 1996.

[7] HERROD, S., ET AL. The SimOS simulation environment. http://www-flash.stanford.edu/SimOS, Sept. 1996.

[8] KOIKE, H., ET AL. A 30ns 64Mb DRAM with built-in self-test and repair function. In *Digest of Technical Papers, 1996 IEEE International Solid-State Circuits Conference* (San Francisco, CA, Feb. 1996), pp. 150–151, 270.

[9] KOZYRAKIS, C., AND WANG, H. Evaluation and Comparison of Existing Cache Designs Implemented as an IRAM. http://www.cs.berkeley.edu/∼kozyraki/project/252/.

[10] PATTERSON, D., ET AL. A Case for Intelligent RAM. *IEEE MICRO 17*, 2 (April 1997), 34–44.

[11] ROSENBLUM, M., ET AL. Complete computer system simulation: The SimOS approach. In *IEEE Parallel and Distributed Technology: Systems and Applications* (Winter 1995), vol. 3, pp. 34–43.

[12] SAEKI, T., ET AL. A 2.5ns clock access 250MHz 256Mb SDRAM with a synchronous mirror delay. In *Digest of Technical Papers, 1996 IEEE International Solid-State Circuits Conference* (San Francisco, CA, Feb. 1996), pp. 374–375.