Evaluation and Comparison of Existing Cache Designs Implemented as an IRAM

Christoforos E. Kozyrakis and Helen J. Wang
{kozyraki,helenjw}@cs.berkeley.edu

University of California at Berkeley
Computer Science Division

Abstract

The increasing processor-memory speed gap has become a performance limitation for current microprocessors. The integration of processor and memory in a single DRAM chip has been proposed in order to overcome this problem. Such an architecture will provide high memory bandwidth and low memory latency, but may have to compensate for slower logic. In this paper, we use a study of program's execution time and an analytical model in order to evaluate the potential performance of IRAM architectures as a function of process parameters, such as the speed of logic and memory access in a DRAM chip. For memory intensive applications, IRAM is faster than conventional implementations, even when logic is 1.5 times slower compared to microprocessor processes. Maximum speedup achieved varies between 1.3 and 1.9. For CPU intensive applications, almost no logic slowdown is necessary for IRAM to achieve comparable performance. We compare the IRAM implementations of simple and a complex processor/cache architecture and find that the first performs comparably and, for some applications, even better than the first one. Finally, we discover that the IRAM implementation of a simple architecture can be 1.5 to 2.8 times faster than the conventional implementation of a complex one.

1 Introduction

Processor and memory chips have been traditionally implemented in separate processes. Logic processes, developed for microprocessor design, provide fast transistors and many layers of metal in order to accelerate logic and communication within a chip. The speed of microprocessors has been improving by 60% per year for over a decade [19]. On the other hand, DRAM processes, used for production of memory chips, are optimized for high memory density and low cost. While DRAM capacity has been increasing at an exponential rate, its latency has improved by 10% within the same ten years. As result, the cost of memory accesses has been an significant obstacle to improving the overall system performance [30]. The usual way of addressing this problem has been adding a hierarchy of caches between the processor and the main memory. These are implemented as small and fast SRAM blocks. The constantly increasing performance gap between processor and memory has led to the use of multiple layers of cache blocks (2 or 3), which sometimes are multi-ported, non-blocking or have increased associativity. Yet, the results of this approach are disappointing for several categories of applications like databases or CAD programs, where most of the execution time is spent in the memory hierarchy [12]. In addition, it is doubtful if this solution will be able to feed future superscalar and speculative processors with data at low latencies.

An alternative approach is to integrate the processor and the main memory in a single chip. This idea is known as Intelligent RAM (IRAM) [24][25]. Since the largest portion of an IRAM die will be dedicated to memory, it is logical to fabricate such a chip in a DRAM process. The upcoming 256 Mbit and 1 Gbit DRAM generations [20] [22] have enough capacity both for a processor core and enough memory for whole programs and data sets to fit on chip. The expected advantages of IRAM are higher memory bandwidth (up to two orders of magnitude), lower memory latency, higher power efficiency, flexible memory granularity and board space savings [25][17]. The main performance gain of IRAM will come from the ability to serve the vast majority of memory accesses in the fast, on-chip memory blocks.

A potential disadvantage of IRAM is the speed of logic in DRAM processes. As mentioned above, existing DRAM processes are optimized for density, thus logic transistors and gates are slower than those in the corresponding logic processes. This can result to a slower CPU core for IRAM compared to that

	Optimistic Slowdown Factor	Pessimistic Slowdown Factor
Logic	1.0	2.0
SRAM	1.1	1.3
DRAM	0.1	0.2

Figure 1: The regions of performance factors used for estimating the IRAM performance.

in a conventional processor [26]. The logic slowdown can significantly affect the performance of IRAM and even completely cancel the speedup gained by accelerating memory accesses. The first contribution of this work is to estimate effect of the logic slowdown and the memory access speedup on the expected performance of IRAM.

A second open question is which is the optimal processor architecture for an IRAM. Looking at modern microprocessors, we can see great variation in the length of the pipeline, the number and type of instructions that can be issued in parallel, the execution model used (in-order or out-of-order, normal or speculative execution) and the type of caches used (capacity, block size, associativity, number of ports). Aggressive designs usually deliver higher performance at the cost of additional complexity. In the case of IRAM, the speedup gained by using complex features may be small compared to from reducing memory latency. In addition, certain features used for hiding or reducing memory latency, like second level caches, may become unnecessary. This paper addresses this issue by comparing the expected performance of an IRAM implementation of a simple architecture to that of a complex one.

We use two existing processor architectures, the 21064 Alpha [4], a simple dual-issue in-order processor, and the Pentium Pro [9], a complex triple-issue out-of-order speculative processor, to estimate the performance of an IRAM implementation of similar organizations. We develop analytical model in order to calculate the speedup of the IRAM implementations over the conventional one as a function of the logic slowdown and the memory access speedup in a DRAM process. The input to these models is the result of an analysis of the portion of the execution time spent in each architecture component for a selection of programs. We find that IRAM can significantly speedup memory intensive applications when the logic slowdown is less than 1.5. For equally fast logic, the speedup for such applications varies between 1.35 and 1.95. For CPU intensive applications, IRAM achieves equal performance to the conventional system only for small logic slowdown, while it can lead to a performance degradation in other cases. For most applications, the speedup of the IRAM Alpha exceeds that of the IRAM Pentium Pro. In addition, IRAM Alpha performs comparably and, for certain applications, even better than the IRAM Pentium Pro, indicating that a simple architecture is more appropriate for IRAM. We also find that, for memory intensive applications, the IRAM Alpha can be 1.53 to 2.85 faster than the original Pentium Pro. Finally, we discover that the cost of off-chip accesses in an IRAM implementation can reduce the speedup obtained and, therefore should be taken into account in future IRAM designs.

The remainder of the paper is organized as following. Section 2 discusses the performance factors examined and the structure of IRAM implementations. Section 3 presents the evaluation methodology. In section 4, we present the result of the evaluation, while section 5 comments on their accuracy and importance. Section 6 summarizes related work and section 7 suggests future work. Finally, in section 8 we present our conclusions.

2 IRAM Performance Factors and Implementation Considerations

The performance of an IRAM will differ from that of a conventional implementation of a similar architecture because of three factors: the speed of logic, the speed of SRAM and the speed of accesses to DRAM main memory. The speed of logic affects the speed of the processor core, i.e. its clock rate. According to process and circuit designers, logic in a DRAM processes is currently 1.3 to 2.0 times slower than logic in a microprocessor process. Yet, logic in a DRAM is an active area of research. Several DRAM manufactures already plan to add extra layers of metals and faster transistors in their processes, so it is quite possible that within a few years, logic in a DRAM process will be equally fast to microprocessor logic. The speed of SRAM affects the speed of cache blocks. SRAM on a DRAM chip is expected to be 1.1 to 1.3 times slower compared compared to a logic process. Finally, DRAM, or main memory, access time will dramatically improve due to the elimination of off-chip buses and memory controllers [25]. The expected benefit is a speedup between 5 and 10. Research prototypes

	Alpha 21064	Pentium Pro
Year	1992	1995 (Q4)
Architecture	Alpha	IA-32
Clock Speed	133 MHz (7.5ns)	200 MHz (5ns)
Pipeline Stages (int/fp)	7/10	14/16
Issue Rate	2	3
Out of order issue	no	yes
L1 Icache	8KB (DM)	8KB (4 way SA)
L1 Dcache	8KB (DM)	8KB (4 way SA)
L2 cache	256KB (off-chip)	256KB (off-chip)
Memory Latency (L1/L2/DRAM)	22.5ns/52.5ns/277.5ns	15ns/35ns/285ns
Transistors	1.7M	5.5M
Die Size	234 mm^2	306 mm^2
Technology	0.75μm CMOS	0.35μm BiCMOS
SPECint92	74.7	320
SPECfp92	112.5	283

Figure 2: The main features of the Alpha 21064 and Pentium Pro processors.

have already demonstrated that such low memory access times are achievable [22]. In order to quantify the effect and importance of each factor, we estimate the expected performance of IRAM architectures for the regions of values[1] presented in figure 1.

The structure of an IRAM architecture is also under research for the moment. Yet, we can expect that generally an IRAM will consist of a processor core, instruction and data first-level caches and several DRAM blocks. First level SRAM caches will probably be used for performance reasons, since the processor's clock rate is usually determined by their (fast) access time. Second-level caches will be unnecessary, because the on-chip DRAM memory will have comparable access time (20-30 nsec) and significantly higher density[2]. The on-chip DRAM will have a capacity of 4 to 8 Mbytes in the initial implementations and will increase in the future as the capacity of DRAM chips grows in the future. It may seem that it is hard to fit all these components in a single chip. Nevertheless, this is feasible even with currently available commercial DRAM technology. A 64Mbit DRAM chip, designed in a 0.4μm CMOS process, can be 186mm^2 big [22]. The original Alpha 21064, designed in a 0.75m CMOS process occupies 234mm^2 [11]. Scaling Alpha to the 0.4μm CMOS process, reduces its size roughly to 66mm^2. Therefore, we can implement the processor in the DRAM chip and still have one third of the die size available for DRAM memory. These are rough estimates that do not take into account many issues like the logic density in a DRAM processes, but do point the feasibility of the design, especially considering the fact that the capacity of DRAM increases by 60% year and that the 256Mbit and 1Gbit DRAM generations have already been demonstrated.

[1] Despite the fact the DRAM access time is actually accelerated, we express all three factors in terms of slowdown for uniformity. Consequently, the DRAM slowdown factor is less than one.

[2] Comparing bits per area for DRAM and SRAM reveals that DRAM is 16 to 32 time denser [17].

3 Methodology

To evaluate the expected performance of IRAM, we use two existing processor/cache architectures : the DEC 21064 Alpha [11] and the Intel Pentium Pro [9]. The main features of the two processors are summarized in figure 2. Although it is most likely that a considerably different architecture will be used for an IRAM, evaluating the IRAM implementations of these organizations can give us an estimate of the potential performance of an IRAM and allow us to predict that application groups that would benefit the most out of it.

The main drive for choosing these two specific architectures is to enable the comparison between the IRAM implementation of a simple and a complex processor/cache architecture. The 21064 Alpha is conservative (older) architecture. The execution model is in-order and caches are direct-mapped and blocking. On the other hand, Pentium Pro is one of the most aggressive architectures currently available. The execution model is out-of-order and speculative, a deep pipeline is used, caches are no-blocking and have increased associativity. Because of the difference in size and especially in complexity of the two approaches, it is very interesting to compare the performance of the corresponding IRAM implementations, and find out whether IRAM would benefit from the use of a com-

Benchmark	Description
gcc	Gcc-2.5.3. Compiles pre-processed source code into optimized SPARC assembly.
vortex	A single user OO database transaction program. Builds and manipulates 3 interrelated databases.
ijpeg	Performs JPEG image compression using fixed point integer arithmetic.
m88ksim	Simulates the Motorola 88100 processor running Dhrystone and a memory test.
li	Xlisp 1.6 lisp interpreter.
perl	Perl 4.0 shell interpreter. Performs text and numeric manipulations.
go	Artificial Intelligence: plays the Go game.
compress	Adaptive Lempel-Zif compression.
linpack	1000x1000 linear equation solver (double precision).
mpeg_encode	Encodes 48 720x480 color frames into a mpeg video file.
sort	Disk-to-disk sort of 100-bytes records of a 21Mbyte database.

Figure 3: The benchmarks and applications used for the evaluation.

plex architecture. Some obvious disadvantages from using the aggressive organization are the additional complexity in design and debugging, that increases the design time and cost, and the fact that a significantly bigger portion of the die area would have to be used for the processor core instead of on-chip memory.

The IRAM implementations of the two machines examined in this study are considered to have exactly the same features with the original implementations, integrated in a single chip with DRAM memory. As explained in section 2, second-level caches are neither useful nor area efficient for an IRAM system. Therefore, we assume that both implementations do not include a second-level cache anymore.

3.1 Performance Estimation Method

The estimation of the performance, i.e. execution time, for the two IRAM implementations is based on analysis of the execution time of a program on the original implementation, and the use of of an analytical model for calculating execution time based on this analysis in respect with the IRAM performance factors.

The goal of the program analysis is to measure where execution time is spent for a variety of programs. We are mainly interested in what portion of the execution time is actually spent on computation and in the various levels of memory hierarchy. Such measurements are traditionally performed by annotating programs with profiling code or by developing software simulators for the target architectures. Instead of that, we use the built-in performance counters available in both processors to collect the necessary information. This method of monitoring a system is non-intrusive and requires no special setup or program processing, as in other profiling approaches [8][14]. The effect on the total run time of the programs is negligible. In addition, it is much faster and easier to use compared to simulator development.

Alpha 21064 has two programmable 16-bit hardware counters [4] that allow us to monitor various events in the processor and the memory hierarchy like: cycles, issues, non-issues, pipeline stall cycles, memory misses, mispredictions, instruction types etc. The data collection is based on interrupt handling upon counter overflow. Pentium Pro has two programmable 40-bit hardware counters that can measure a vast number of events related to the processor, memory hierarchy or bus system [5]. Data collection is done by reading the actual counter values[3]. Using the hardware counter involved developing or porting the corresponding device drivers, access tools and result processing tools under DEC OSF1 V3.0 for Alpha, and BSD/OS V2.1 for Pentium Pro. The counters in both processors allow us to measure separately events in user and kernel mode for a program. More information on the about the hardware counters of the two processors and their use is included as an Appendix.

The analytical model developed calculates execution time by properly scaling or predicting the time spent in each component of an architecture. Time spend in the processor is multiplied by the logic slowdown factor. This includes time spent issuing instructions and stalling for reasons not related to memory hierarchy, like mispredictions, busy units, register conflicts etc. We do not try to separate the execution time spent spent executing loads/stores that resulted in cache misses, so that we can multiply that with the SRAM slowdown factor. The reason is the delicate interdependence of the processor clock cycle and the cache access time. Since the SRAM slowdown factor varies within a subset of the logic slowdown factor, we choose to use second one with the cache hits time

[3]Due to their size, the counters can be used to monitor programs for several minutes before an overflow becomes possible.

component as well. Hence, it is like using the worst of these to factors in any case, and therefore calculate a worst-case execution time for IRAM for some specific values. All first-level cache misses will be served in an IRAM by the on-chip DRAM. Thus, the amount of time spent in memory hierarchy is calculated by multiplying the number of misses with the corresponding main memory access latency of the original system and the DRAM slowdown factor for IRAM. By adding these two components, time in processor and time in on-chip DRAM, we can estimate the IRAM execution time as a function of two variables : the logic and DRAM slowdown in a DRAM process.

3.2 Benchmarks and Applications

Figure 3 describes the benchmarks and applications used for the evaluation. SPEC95 [3] is the current accepted standard metric of uniprocessor performance. We use the eight integer programs (SPECint95) from this suite. These benchmarks are considered to be CPU intensive and to put very little pressure on the memory hierarchy. In addition, it is doubtful if they efficiently represent all possible workloads on a current processor. In order to compensate for that and increase the validity of our work, we use three additional applications: sort [7], which is representative of the databases field, where operations are performed on large collections of data; mpeg_encode [28], which is a characteristic multimedia application, and finally linpack, that represents programs for scientific computations.

All these benchmarks have working sets smaller than 4Mbytes, which is the expected capacity of on-chip DRAM in the first IRAM implementation, as explained in section 2. Hence, apart from the cold misses, no other accesses will have to go off-chip in an IRAM implementation. A variation of sort that has a larger working set and a study of the importance of the cost of off-chip accesses in IRAM is presented later on.

We avoid using microbenchmarks like lmbench, STREAM etc [23]. Such benchmarks do not represent any real application or workload. They are especially developed to stress a particular part of the system and measure its behavior and performance. Relating such measurements to the overall system performance, which is execution time of a program, is very difficult (if possible at all). Especially when evaluating the expected performance of new architectures and design methods, as in the case of IRAM, the results of microbenchmarks may be deceptive or pessimistic due to their dependence on the existing operating system used for the measurements/simulations and other assumptions that may not hold for the new system.

4 Evaluation and Results

Figure 4 presents the result of the programs' execution analysis. For each benchmark, the percentage of execution time spent in the processor and serving cache misses of various levels is shown [4]. All benchmarks behave similarly for both processors. Sort and Linpack spend almost half of their execution time waiting for the memory hierarchy (40% to 55%). Mpeg_encode spends less than 10% of execution time in memory, despite the fact that it reads several large frames. This is because mpeg_encode performs such a great amount of computation on each frame, that the time spent in memory is relatively small. The same holds for most of SPECint95 programs (around 10% of execution time in memory), excluding compress and gcc which appear to be more memory intensive.

By comparing the results of the measurements for user and kernel mode, we find out that with the exception of sort, all other programs spent less than 10% of their execution time kernel mode. For sort, 65% of execution time for Alpha and 45% for Pentium Pro are spent in kernel mode.

4.1 Performance of IRAM Implementations

Figures 5 to 8 present the speedup achieved by the two IRAM implementations for each application compared to the original implementations. For SPECint95, we present the average speedup. Speedup is calculated according to analytical model presented in section 3, by dividing the execution time of the original implementation with the estimated execution time for the IRAM one. Naturally, speedup is also expressed as a function of the logic and memory slowdown factors. Speedup greater than one means that the IRAM implementation is faster than the conventional one, while speedup less than one means the opposite.

For sort (figure 5), both IRAM organizations achieve speedup greater than one when the logic slowdown factor is less than 1.5. Maximum speedups are 1.32 and 1.5 for IRAM Alpha and Pentium Pro respectively. Linpack, figure 6, is the application that benefits the most from the IRAM implementations.

[4] Although all the necessary information for detailed decomposition of the time spent in the processor was collected, the results are not presented,since such an analysis is out of the scope of this work.

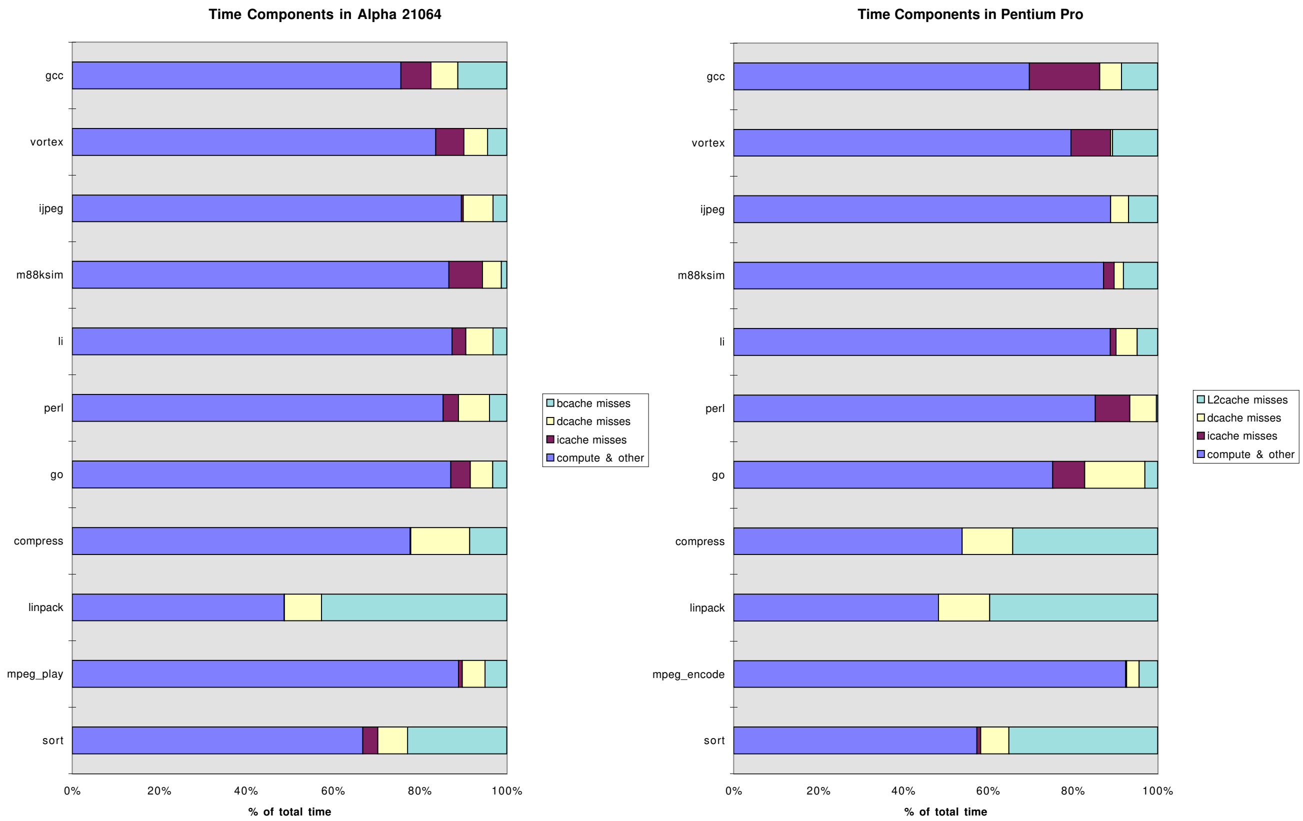

Figure 4: Time components for 21064 Alpha (left) and Pentium Pro (right).

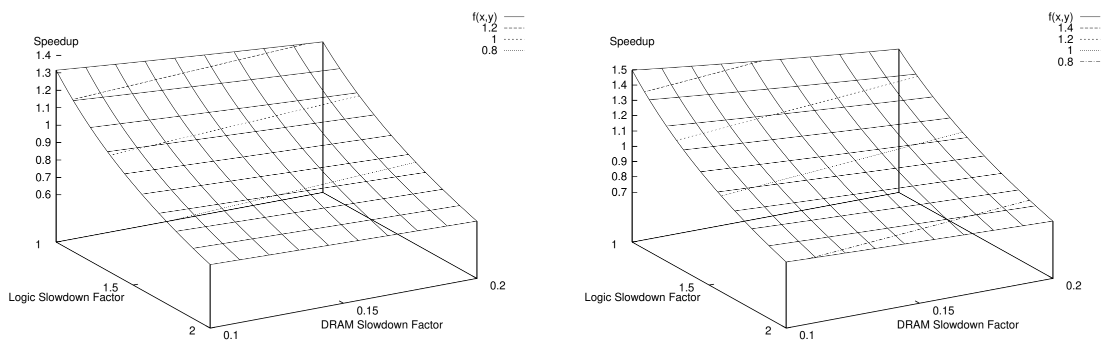

Figure 5: The speedup of the Alpha (left) and Pentium Pro (right) IRAM implementations for sort.

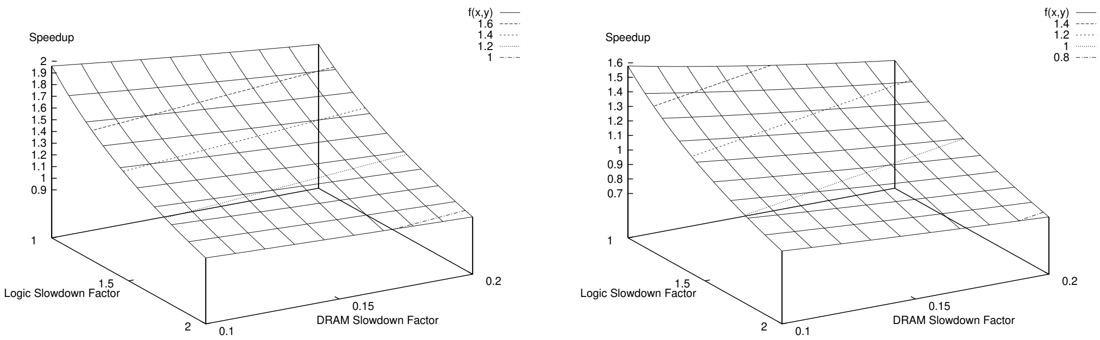

Figure 6: The speedup of the Alpha (left) and Pentium Pro (right) IRAM implementations for linpack.

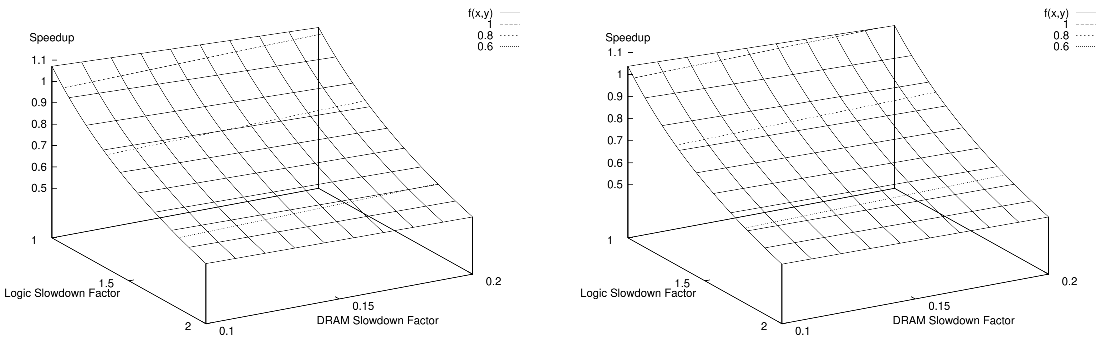

Figure 7: The speedup of the Alpha (left) and Pentium Pro (right) IRAM implementations for mpeg_encode.

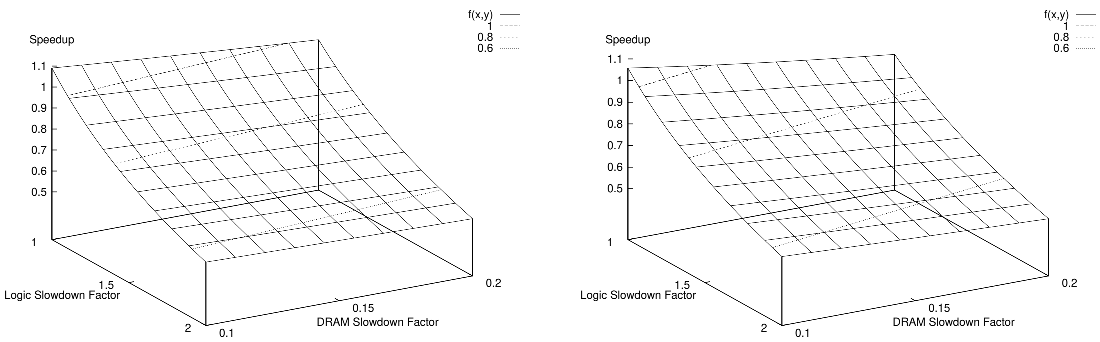

Figure 8: The average speedup of the Alpha (left) and Pentium Pro (right) IRAM implementations for SPECint95 benchmarks.

In the case of Alpha, there is a performance gain essentially for all values of the two factor and the maximum speedup is 1.95. For Pentium Pro, performance advantages over the conventional organization begin for logic slowdown less than 1.4 and the maximum speedup is 1.57.

Mpeg_encode (figure 7) and SPECint95 benchmarks (figure 8), not only do not benefit from IRAM, but get significant slowdown for most values of the two factors. Performance comparable to that of the original implementations is only achieved for almost no logic slowdown at all, while the performance degradation can be as much as 50% in the worst case. The reason is that these benchmarks are CPU intensive, hence speeding up the memory hierarchy cannot compensate for the slowdown of the time spent in the processor.

Generally, we draw the conclusion that exact value of the memory slowdown factor does not significantly affect performance. The differences in speedup as the memory slowdown varies between 0.2 and 0.1 (5 to 10 times faster memory access), for a given logic slowdown, are very small. On the other hand, small variation in the logic slowdown makes a difference in the speedup. Therefore, the main effort in circuit aspect of IRAM should concentrate on the speed of logic in a DRAM chip.

4.2 Comparison of Alternative Architectures

Looking again at figures 5 to 8, we can see that the speedup achieved by the IRAM Alpha is almost always higher than the that achieved by the IRAM Pentium Pro. This is a first indication that a simple processor architecture is more appropriate for an IRAM implementation.

Figure 9 presents the speedup of IRAM Alpha over IRAM Pentium Pro for linpack and sort. We assume that an equal clock rate can be reached for both implementations [5]. For linpack, IRAM Alpha is 1.15 to 1.35 times better than IRAM Pentium Pro. For sort, IRAM Pentium Pro is 1.38 time faster than the Alpha implementation. Taking into account in addition the complexity and transistor count differences of the two architectures (1.7M transistors for Alpha and 5.5M transistors for Pentium Pro), we can conclude that a simple architecture appropriate for IRAM, since the complex one is either slower or cannot provide a performance increase proportional to its complexity and size.

[5] In reality, one would expect a higher clock rate for the IRAM Alpha because of the simplicity of its pipeline and structure compared to that of Pentium Pro.

To realize better the performance benefit from the use of the IRAM implementation, figure 10 presents the speedup of IRAM Alpha over the original Pentium Pro for linpack and sort. Once more, we assume that, for no logic slowdown in DRAM processes, the clock rate of these two can be the same. This assumption may be pessimistic for the IRAM Alpha, because of the simplicity of its structure. We find that the performance of the IRAM with the simple architecture significantly exceeds that of the conventional implementation of the complex architecture. For linpack, IRAM Alpha can be as much as 2.85 time faster then Pentium Pro, and 1.53 times faster in the case of sort. This suggests that using an IRAM instead of investing time and effort in developing extremely complex processor architectures can lead to significant performance gains for memory intensive applications.

4.3 The effect of Off-Chip Accesses in IRAM

All the measurements and results presented so far, assume that whole programs and working sets fit in the on-chip DRAM of an IRAM and, therefore, almost no off-chip accesses occur. This is true for the benchmarks used, even for implementations of IRAM with currently commercial available DRAM technologies, as their working set is never more than 4Mbytes. Still, there are many programs that will not fit completely on-chip both in initial and future implementations of IRAM. An off-chip access in IRAM will result in the transfer of a full memory page from external storage (off-chip DRAM probably), independently of the exact use of on-chip DRAM (page cache or normal paging memory). The cost of an off-chip access can vary significantly, depending on the exact implementation. The worse thing to do would be to stall until the whole page comes on-chip before continuing the computation. Smarter techniques would be to continue as soon as the critical word comes on-chip, or fetch the critical word first and continue normal operation while a DMA-like controller fetches the rest of the page.

In order to estimate the effect of off-chip accesses on IRAM performance, we developed a second version of the sort benchmarks, in which the total size of keys in the database is 16Mbytes, instead of the 2Mbytes the original version presented in section 3. By restricting the memory available for use by the program to 4 or 8 Mbytes and measuring the number of page-faults, we get an estimate of the number of off-chip accesses that would occur in an IRAM with 4 or 8 Mbytes on-chip DRAM respectively, if it is

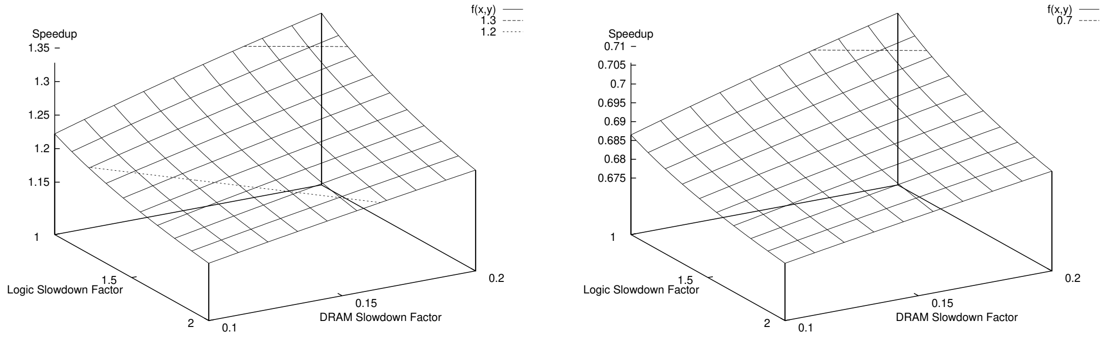

Figure 9: The speedup of IRAM 21064 Alpha over IRAM Pentium Pro for linpack (left) and sort (right).

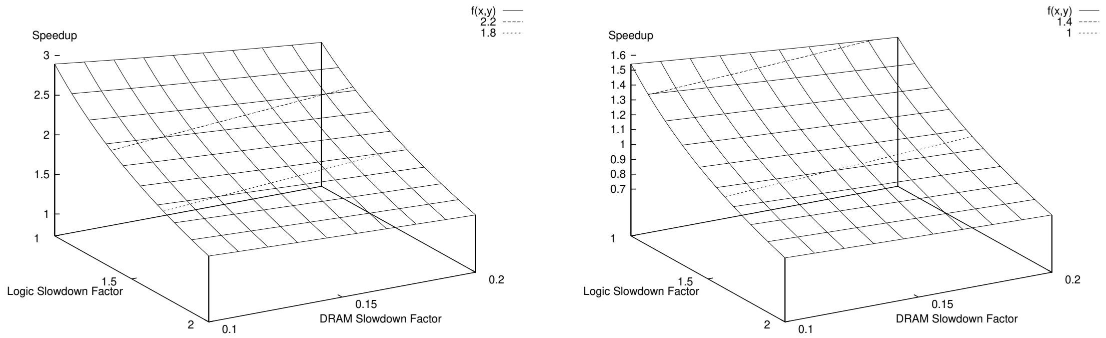

Figure 10: The speedup of IRAM 21064 Alpha over the original Pentium Pro for linpack (left) and sort (right).

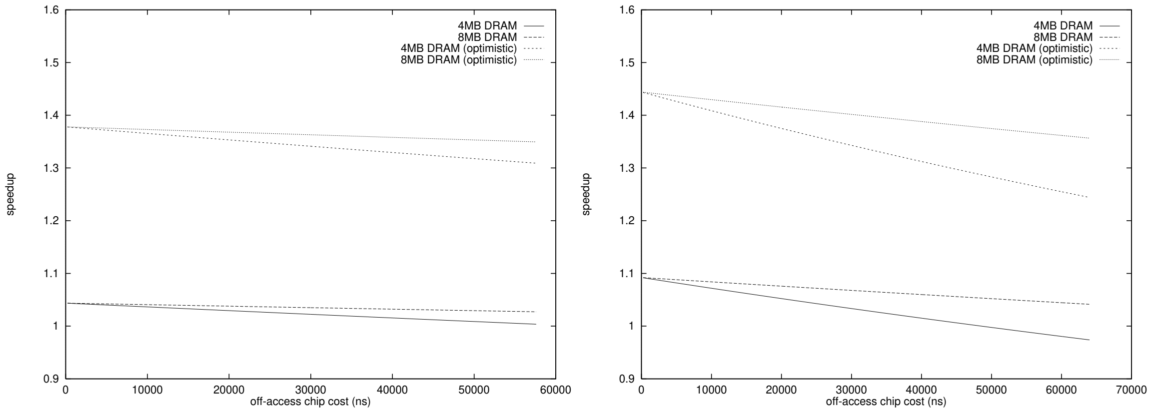

Figure 11: The speedup of IRAM 21064 Alpha and Pentium Pro as a function of the off-chip access cost.

used as a page cache. The estimation of the execution time is now performed by adding the number of page-faults multiplied by the off-chip access cost to the time calculated using the method presented in section 3.

Figure 11 presents the speedup of the two IRAM implementations as a function of the off-chip (page-fault) cost. Off-chip access cost varies between the time necessary to fetch a single cache-line from off-chip memory, to the time necessary to fetch an whole 8-Kbyte page. We assume no bus contention and use the original bus speed of the two systems. For both architectures, we present the speedup of four different IRAM implementations, that have either 4 or 8 Mbytes on-chip cache and use the optimistic performance factors (logic slowdown 1.0, memory slowdown 0.1) or two more realistic, for the moment, factors (logic slowdown 1.3, memory slowdown 0.15). Unfortunately, sort is not really using a 16-Mbyte working set all the time. Because the quicksort algorithm is used, after one or two problem divisions, the working set is reduced enough to fit entirely in the on-chip DRAM. Thus, the number of page-faults is very low, and no striking results can be observed. Yet, for IRAM implementations with 4Mbytes of DRAM, high off-chip access cost can reduce speedup by 0.1. Thus, we can expect that for programs with large working sets, the cost of off-chip accesses will be an important issue in IRAM, and support for low access cost will be necessary.

4.4 Other Results

The nature of our measurements, as well as the structure of the sort program allow us to make a few more observations about the impact of IRAM on the execution of programs. Despite the fact that we cannot claim that the same will hold for all programs running on an IRAM, we state these observations because they are interesting and can motivate further examination of the corresponding issues.

As mentioned earlier, sort spends a significant fraction of its running time in kernel mode. Figure 12 presents the speedup of the two individual components: time in user mode and time in kernel mode. It is very interesting to notice that the behavior for both components and their speedup are almost identical.

Sort can be divided into three distinct phases : read data phase, in-memory sort phase and write result phase. This is typical for many application, which can also be separated in read, computation and write phases. Figure 13 shows the individual speedups of each phase for the Pentium Pro. It is obvious that the read and write phases benefit from the IRAM implementation much more than the sort phase. This difference will be even more obvious in other programs, for which in the computation phase more operations are performed on each data element read. In quicksort it is just a compare and swap. Hence, we can expect that IRAM implementations will significantly speedup the I/O phases of programs and that applications, whose execution time is dominated by these phases, will benefit the most.

5 Accuracy and Importance of the Evaluation

The results of this study are based on an analytical model and the breakdown of programs' execution time in the conventional implementations of the architectures used. This method is inherently unable to model potential interactions of time spent in memory hierarchy and components of time spent in the processor, like busy unit stalls. Yet, this work does not aim to predict the exact performance of IRAM architectures. Its goal is to present performance trends, identify the effect of specific issues on performance and estimate the value of speedup that can be expect from IRAM for specific application groups. Detailed simulations of IRAM architectures using the SimOs [27] environment, that are currently performed by Bowman et.al. [10], present results that accurately match those stated in this paper. In addition, our model only uses the low memory advantage of IRAM in order to evaluate the potential performance gains. A real IRAM implementation will also utilize the high memory bandwidth offered, by using prefetching, multithreading or some other appropriate technique. Efficient utilization of high bandwidth may reduce the effective memory latency even by a factor of 100 or more. Figure 14 presents the expected speedup from the IRAM Alpha implementation over the conventional one as a function of the logic speed factor, for memory performance factor of 0.01 (i.e. speedup of 100). It is clear that the performance gain of an IRAM will be even bigger if the higher memory bandwidth is taken into account. So, one can view our results as a lower bound or worst case prediction of the performance of IRAM.

Analytical models are difficult to develop for the case of out-of-order and speculative architectures, like the one of Pentium Pro, due to the extended parallelism [9]. In order to avoid not including parallelism in our study, we use optimistic delays for all operations in Pentium Pro (e.g. the cost of a first-level cache miss). In addition, we used our analytical model to predict the execution time of the original

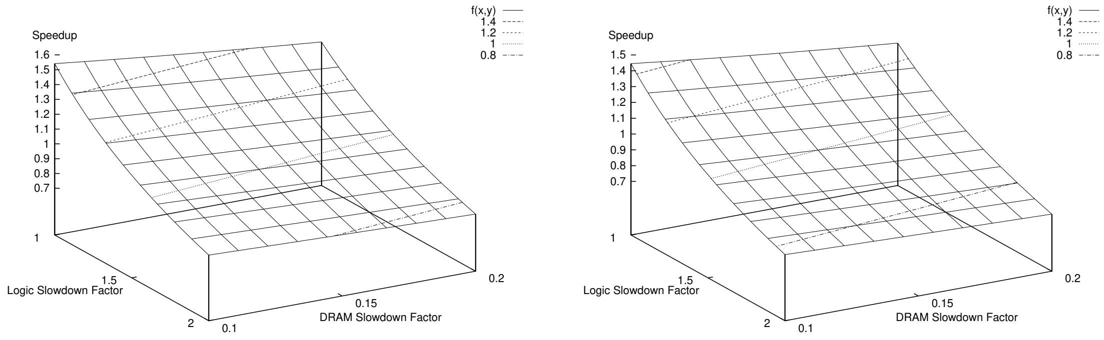

Figure 12: Speedup of user (left) and (kernel) mode time components for sort in the Pentium Pro.

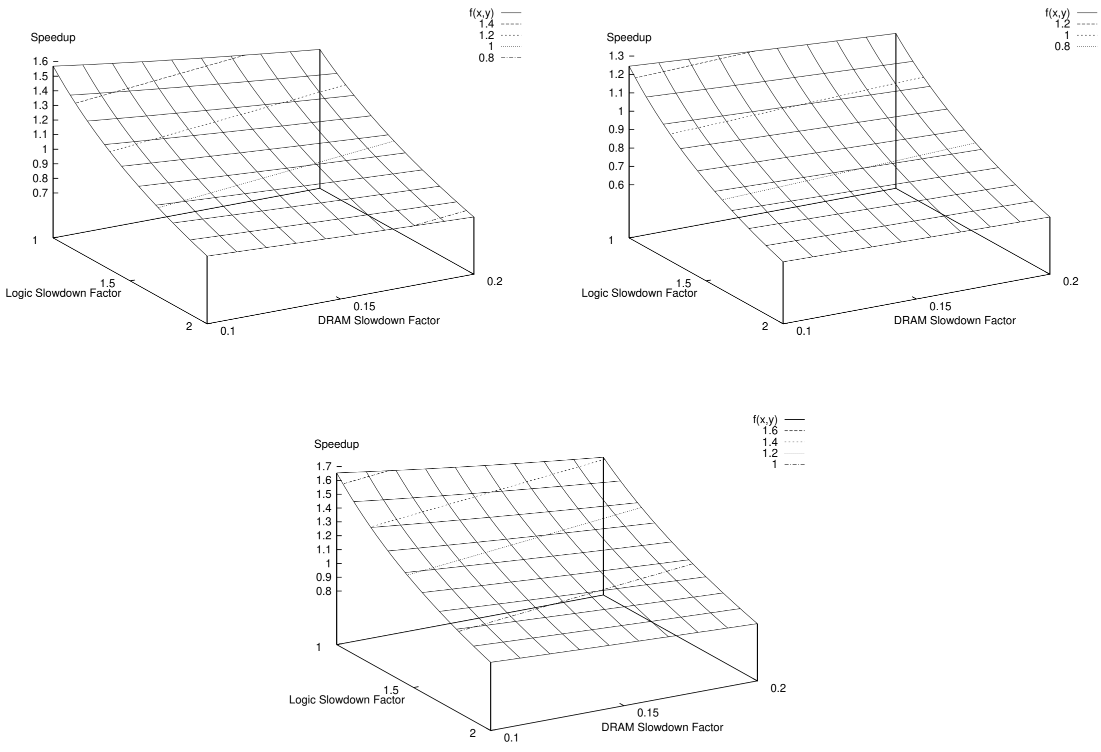

Figure 13: Speedup for the three phases of the sort program on Pentium Pro: write phase (up left), sort phase (up right) and write phase (bottom).

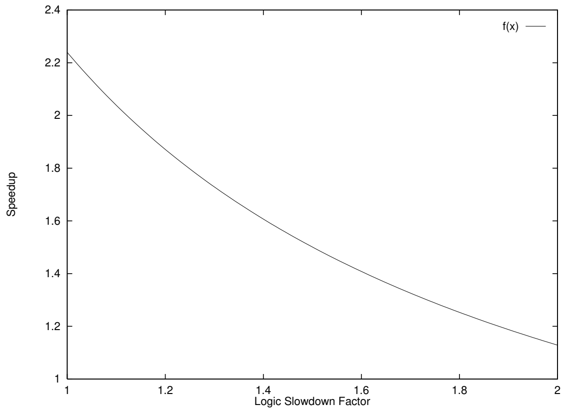

Figure 14: Speedup for sort on IRAM Alpha as a function of the logic slowdown factor, for DRAM slowdown factor of 0.01.

implementation and found that the error of our estimation was always less than 10%.

6 Related Work

Processor and memory integrations has been investigated by other researchers as well. Previous work has concentrated on proposing a specific architecture and evaluating its performance. Instead of that, this work attempts to estimate the performance of IRAM systems as a function of process parameters and compare the expected performance of simple and complex processor/cache designs implemented as an IRAM. These results can be used as points of reference for future research on IRAM architectures and implementations.

Saulsbury et.al. [29] evaluated the performance of a single-scalar SPARC CPU, implemented in a 256Mbit DRAM, using sense-amps as caches. They found that, for optimistic process parameters, they get comparable performance for integer programs and and half performance for floating-point programs. Giacalone et.al. [18] presented an chip that implements second-level cache with DRAM.

Integration of processor and memory has also been used for graphics accelerators and image-processing devices [13][6].

Several attempts to integrate processor and memory have been made in order to design building blocks for parallel computers. Some of the systems in this category are the J-Machine [15], MOSAIC-C, Execube [21] and the M-Machine [16].

Finally, the Galileo project [1] at the University of Wisconsin and PPRAM project [2] at Kyushu University are also exploring various issues in processor-memory integration.

7 Future Work

In terms of this specific study, it would be interesting to repeat the same analysis for some additional applications, like a database server or more floating point programs for example. This would give us a more complete view of the expected performance of IRAM on a wider range of applications. Still, this study aims to indicate the potential advantages of IRAM and attempts to address some implementation and architectural issues. There is still a lot to be done both in the circuit and architecture level.

In the process and circuit level, there is always the issue of the speed of logic in a DRAM process. As indicated by this work, this is a crucial factor affecting the potential IRAM performance. Adding extra layers of metal and faster transistors to DRAM processes will certainly be a hot issue in the near future. Another question that needs investigation is the effect of placing a processor in a DRAM chip on the memory retention (refresh) time. Research on such issues will lead to processes and circuit design techniques that will be properly balanced both for DRAM memory and logic design.

In the architecture level, the main area interest will be the investigation of specific organizations that will efficiently utilize the main advantages of IRAM : low memory latency and high bandwidth. In addition, the high transistor capacity of DRAM processes may allow the integration of other devices within an IRAM system, like a vector processing unit or a reconfigurable device (e.g. a gate-array).

Finally from an application point of view, it will be interesting to study the implications of IRAM in the structure of programs, compilers and operating systems. Some of their characteristics or tuning nodes may need to be altered in order to utilize the performance that IRAM promises to offer. In addition, the high memory bandwidth available in IRAM may make multi-threading very attractive and efficient.

8 Conclusions

The Intelligent RAM (IRAM) has been proposed as a solution to handle the increasing performance gap between processor and memory. In this paper we addressed the issue of the expected IRAM performance as a function of process performance parameters. We find that, memory intensive applications benefit from IRAM, as long as the logic slowdown in a DRAM process is less than 1.5. Maximum speedup can be from 1.35 to 1.95. On the other hand, CPU intensive applications can achieve performance to that of conventional architectures only for almost no logic slowdown. We also discover that a simple processor/cache architecture is more appropriate for IRAM, since it can perform comparably or even better than an IRAM implementation of a complex architecture. In addition, an IRAM implementation of a simple architecture can be 1.53 to 2.85 times faster than a conventional implementations of an aggressive organization. Finally, we observe that the cost of off-chip accesses can affect the performance of IRAM and should be carefully examined in future implementations.

Acknowledgments

We would like to thank Stephen Smoot and Remzi Arpazi-Dusseau for their help with the benchmarks, and David Mazieres for his help with using the Pentium Pro hardware counters.

Appendix

Hardware counters can be a very powerful tool for programmers and system engineers. They can be used for detailed analysis of programs' execution and to provide insights into the system behavior.

The 21064 Alpha has two hardware performance counters each of which can be independently programmed to count certain internal or external events. They can be used for event measuring, ILP histogramming or user/kernel PC profiling both in a uniprocessor and a multiprocessor environment. The specific events that can be measure are : issue cycles, pipeline dry cycles, load instructions executed, pipeline frozen cycles, branch instructions executed, PALmode cycles, total cycles, total non-issue cycles, second-level cache misses with victims (with counter 0), data first-level cache misses, instruction first-level cache misses, dual issue cycles, branch mispredictions, floating-point instructions executed, integer instructions executed, store instructions executed, second-level cache misses without victims (with counter 1). We used the /dev/pfcntr device driver available in DEC OSF1 V3.0. Data selection with this driver is done through interrupt handling on counter overflow. DEC also provides the *uprofile* and *kprofile* tools for using the device driver in order to measure events during the execution of a specific program. Because these two tools do not register separately the number of events measured by each counter, we also developed our own tool for this purpose. In addition, because *uprofile* and *kprofile* have a awkward and quite difficult to use output format, we developed a simple translator in order to get their measurements result in some readable form.

Pentium Pro also has two programmable hardware performance counters. The total number of events that can be measured is 70 and are related to the processor, the caches, the bus and the memory system. All these events can be independently measured for user/kernel activity or each possible cache line status in the MESI protocol, in case of a multiprocessor system. The counters can even be programmed to do inverse measurements or be incremented only when a certain number of events occurs within a cycle. Pentium Pro counters can be read directly from user mode without need to invoke kernel. We developed a device counter for accessing the counters under BSD/OS V2.1, as well as the necessary tool for program specific measurements.

All tools and device drivers developed within this project are public available. For more information please contact the authors.

References

[1] Galileo project. http://www.cs.wisc.edu/ galileo/. Computer Science Department, University of Wisconin - Madison.

[2] PPRAM project. http://kasuga.csce.kyushu-u.ac.jp/ ppram/. Computer Science Division, Kyushu Univeristy, Japan.

[3] SPEC95 CPU Benchmarks. http://www.specbench.org/osg/cpu95/.

[4] DECchip 21064-AA Microprocessor Hardware Reference Manual, 1992.

[5] Pentium Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual, 1996.

[6] Y. Aimoto et al. A 7.68 GIPS, 3.84 GB/s 1W Parallel Image-Processing RAM Integrating a 16 Mb DRAM and 128 Processors. In *Digest of Technical Papers, 1996 IEEE International Solid-State Circuits Conference*, pages 372–373, 476, San Francisco, CA, February 1996.

[7] A. Arpaci-Dusseau et al. High-performance sorting on networks of workstations. 1996.

[8] D. Bhandarkar and D. Clark. Measuring VAX 8800 Performance with Historogram Hardware Monitor. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 301–310, 1984.

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

[10] W. Bowman, N. Cardwell, and C. Cromer. IRAM System Simulation, 1996.

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

[12] Z. Cvetanovic and D. Bhandarkar. Performance characterization of the Alpha 21164 microprocessor using TP and SPEC workloads. In *Proceedings of the Second International Symposium on High-Performance Computer Architecture*, pages 270–280, San Jose, CA, February 1996.

[13] M. Deering and S. Schlapp. FBRAM: a new form of memory optimized for 3D graphics. In *Proceedings of the SIGGRAPH 94 Conference*, pages 167–74, July 1994.

[14] C. Noakes et.al. Instruction Level Profiling of the RS/6000. In *Proceedings of the 18st Annual International Symposiu m on Computer Architecture*, pages 180–189, May 1991.

[15] M. Noakes et.al. The J-Machine multicomputer: an architectural evaluation. In *Proceedings of the 20st Annual International Symposium on Computer Architecture*, pages 224–235, May 1993.

[16] M. Fillo et al. The M-Machine multicomputer. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 146–156, Ann Arbor, MI, November 1995.

[17] R. Fromm et al. The Energy Efficiency of IRAM Architectures. *paper submitted to the 23rd International Symposium on Computer Architecture*, May 1996.

[18] G. Giacalone et al. A 1 MB, 100 MHz integrated L2 cache memory with 128b interface and ECC protection. In *Digest of Technical Papers, 1996 IEEE International Solid-State Circuits Conference*, pages 370–371, San Francisco, CA, February 1996.

[19] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Inc, San Francisco, CA, 1996.

[20] M. Horiguchi et al. An experimental 220MHz 1 Gb DRAM. In *Proceedings of the IEEE International Solid-State Circuits Conference*, volume 38, pages 252–253, February 1995.

[21] P. Kogge. EXECUBE - A New Architecture for Scaleable MPPS. In *Proceedings of the 1994 International Conference on Parallel Processing*, volume 1, pages 77–84, Raleigh, NC, August 1994.

[22] H. Koike 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*, pages 150–151, 270, San Francisco, CA, February 1996.

[23] L. McVoy and C. Staelin. lmbench: Portable tools for performance analysis. In *Proceedings of the USENIX 1996 Annual Technical Conference*, January 1996.

[24] D. Patterson, T. Anderson, and K. Yelick. A Case for Intelligent DRAM: IRAM. In *HotChips VIII*, pages 75–93, Stanford, CA, 1996.

[25] D. Patterson et al. Intelligent ram (IRAM): Chips that remember and compute. In *To appear in 1997 IEEE International Solid-State Circuits Conference*, San Francisco, CA, February 1997.

[26] Steven A. Przybylski. *New DRAM Technologies: A Comprehensive Analysis of the New Architctures.* MicroDesign Resources, Sebastopol, California, 1994.

[27] M. Rosenblum et al. Complete Computer System Simulation: The SimOS Approach. In *IEEE Parallel and Distributed Technology: Systems and Applications*, volume 3, pages 34–43, Winter 1995.

[28] L. Rowe et al. MPEG Video in Software: Representation, Transmission and Playback. In *Proceedings of the SPIE 1994 International Symposium on Electronic Imaging*, February 1994.

[29] A. Saulsbury, F. Pong, and A. 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, Philadelphia, PA, May 1996.

[30] W. Wulf and S. McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(1):20–24, March 1995.