

A Comparison of the VIRAM-1 and VLIW Architectures for use on Singular Value Decomposition

Jeffrey Herman, John Loo, and Xiaoyi Tang
{jefe,jloo,xiaoyi}@eecs.berkeley.edu

Department of Electrical Engineering and Computer Science
University of California, Berkeley CA 94720-1776

Abstract

The singular value decomposition (SVD) has recently taken on a greater significance in embedded applications but has a high computational cost. In this paper we will examine two types of high performance embedded architectures, VLIW and VIRAM, for computing the SVD. VLIW is an attractive solution due to its ability to cheaply exploit instruction level parallelism in statically schedulable algorithms. An alternative is to use a vector processor, which can exploit the high levels of vector data parallelism inherent in SVD. Vector designs can scale without increasing global interconnect or control logic. This property alleviates the Flynn bottleneck thereby reducing delay and energy requirements. To quantify these differences we will benchmark the SVD algorithm on both architectures and examine execution time and energy requirements. The TMS320C67 and TM1300 will represent VLIW while the VIRAM-1 will represent vector architectures. What we find is that VLIW and VIRAM perform identically over a wide range of matrix sizes, but then IRAM eventually overtakes VLIW as the data size grows.

1 Introduction.

Over the last several years microprocessor design has shifted from a general purpose design to a more embedded one. Among the uses for embedded systems are media processing kernels which require high computational bandwidth. Therefore, the current direction in VLSI design is to design high-performance microprocessors with low energy requirements. In particular, these designs must satisfy energy and delay requirements imposed by algorithms prevalent in embedded systems. Singular value decomposition (SVD) has recently taken on a great deal of interest among applications such as image processing, speech recognition [11], and direction-of-arrival computations in multi-antenna arrays [9].

The current trend in high performance embedded computing is to exploit high degrees of instruction-level-parallelism (ILP) by issuing multiple statically scheduled instructions in lockstep. This class of architectures is known as the very long instruction word format (VLIW) that is used in the Trimedia 1300 and TMS320C67 processors. The advantage of the VLIW over superscalar is that the static scheduling removes the hardware and need for dynamic instruction scheduling and hazard detection. There are several problems with VLIW

architectures, however. The static scheduling makes the performance highly sensitive to compiler efficiency, and VLIW compilers are still maturing. Also, although the static scheduling simplifies the decode logic to allow a wider issue width, VLIW still requires extremely multiported register files which results in a delay and power consumption penalty.

An alternative is the VIRAM-1, a vector architecture that exploits embedded DRAM to cheaply provide the high memory bandwidth requirements of a vector processor. The differences between vector and VLIW architectures have implications in both energy and delay [11]. The performance of a vector processor is reliant on the compiler's ability to exploit as much vector data parallelism (VDP) as possible. VIRAM exploits VDP simply by executing a single instruction to perform a single homogeneous operation on a vector of data in lockstep [1]. The advantage of a vector processor its ability to scale to extreme widths while still avoiding the Flynn Bottleneck [7].

Much previous work has been done studying on both VLIW and vector processing, but little work has been done on directly comparing the two competing architectures. Our goal is to determine which architecture is more suited for the SVD in terms of execution time and energy requirements. The remainder of this discussion will be organized as follows: Section 2 gives a brief overview of the SVD algorithm. Section 3 describes the microarchitecture of all three designs we are testing for use on SVD. Section 4 discusses the testing methodologies. Section 5 presents the results of our tests and examines the implications they have in terms of performance and energy, and section 6 presents our conclusions. We also propose a few future directions in section 7.

2 Singular Value Decomposition

The SVD code that we evaluated is from LAPACK, which is based on the QR algorithm. The QR algorithm begins by upper bidiagonalizing a matrix \mathbf{A} (make all of its elements except for those on the main diagonal and the superdiagonal zero) to form matrix \mathbf{B} using a sequence of Householder transformations ($\mathbf{B}=\mathbf{Q}^T*\mathbf{A}*\mathbf{P}$). The Householder transformation is defined by $\mathbf{Q}=\mathbf{I}-2\mathbf{u}*\mathbf{u}^T/|\mathbf{u}|^2$, where \mathbf{u} is a vector and \mathbf{I} is the identity matrix. It can transform any nonzero vector \mathbf{x} to a vector \mathbf{Qx} whose first element is the only nonzero element by defining $\mathbf{u} = \mathbf{x} - |\mathbf{x}|\mathbf{1}$, where $\mathbf{1}=[1,0,\dots,0]^T$. This property of the Householder

transformation makes it a good choice to bidiagonalize a matrix. Next, the QR algorithm iteratively transforms \mathbf{B} using Givens rotations until the superdiagonal elements become negligible. Therefore, the kernels of the SVD code are various forms of matrix multiplication, highly vectorizable operations.

3 Microarchitecture Overview

3.1 TM1300 Microarchitecture

The TriMedia TM1300 is a high-performance media processor. The core of TM1300 is a 32-bit five-issue VLIW processor running at a clock speed up to 166MHz. It has two FP adders and multipliers and can issue two load/store operations simultaneously. Branch, FP add/multiply, and load instructions all have 3-cycle latency. It implements 128 general-purpose 32-bit registers, a 16KB data cache and a 32KB instruction cache. The registers are not separated into banks so that any operation can use any register. Both data cache and instruction cache are eight-way set-associative and have a 64 byte block size. The data cache is implemented as eight independent single ported banks. Bank conflict occurs when two memory accesses select the same cache bank [16].

3.2 TMS320C67 Microarchitecture

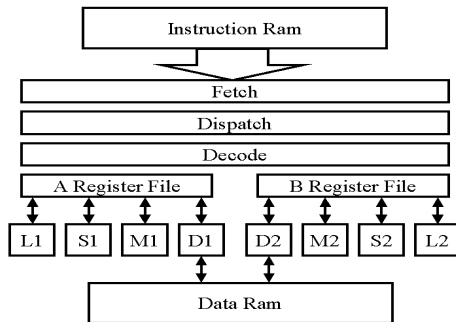


Figure 1: The TMS320C67 Microarchitecture

The TMS320C67 is an 8-way issue floating point VLIW DSP. The C67 contains two register files of 16 registers each rather than a monolithic 32-entry register file. Attached to each of the two register files is a set of four functional units that may only read and write to its register file. There are two cross paths that let each set of functional units read one value out of the other register file per clock cycle to allow limited interaction between the two.

Each set of four functional units contains a load/store unit, a floating-point adder, and a floating pointer multiplier. The load/store unit has a 5-cycle load latency and supports circular buffers and auto-increment addressing modes. The two load/store units may operate in parallel provided that they don't attempt to access the same bank of memory in which case the processor will stall for a cycle. The floating pointer adders and multipliers are fully pipelined with a 4-cycle

latency when operating in single-precision mode, and they may be operated in parallel to allow a total of 4 floating point operations per cycle.

The branch instructions require 6 cycles to complete which translates into a need to schedule 47 instructions additional for every branch. To alleviate the branch penalty, the C67 allows all instructions to be conditionally executed. Five of the 32 general-purpose registers serve a dual purpose as conditional registers, and when they are used to specify conditional execution, it does not matter which register file the conditional register actually resides in [14].

The memory system typically consists of on chip SRAM with separate instruction and data memories. In the C6701 configuration, the data memory consists of two 32k blocks each with 8 16-bit banks while the C6711 features a two level cache hierarchy with 4k instruction and 4k data L1 caches and a unified 64k L2 cache. All versions support off chip SRAM and DRAM, but the usage of such memories can result in a substantial performance penalty [15].

3.3 VIRAM-1 Microarchitecture

In contrast to the VLIW architectures, the IRAM microarchitecture (shown in figure 2) is relatively simple in design. The basic design consists of a 2-way-issue 64-bit MIPS IV scalar core with a 16 Kbyte two-way associative instruction and data cache. This includes two integer datapaths and one decoupled floating point unit [Kozaraki]. However, for purposes of evaluation the `vsim-p` simulator simulates the scalar core pessimistically by limiting the issue rate at 1 CPI. This inaccuracy will be discussed in section 4.

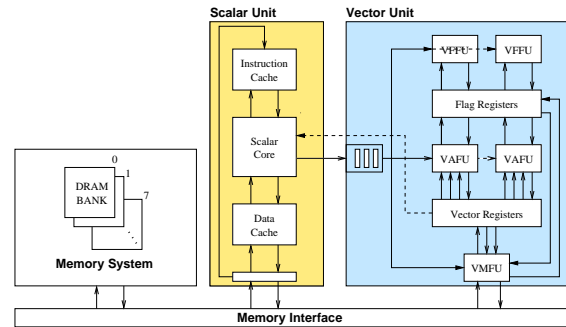


Figure 2: The VIRAM Microarchitecture

The main computation unit for the VIRAM is its loosely-coupled vector core which communicates with the scalar unit through a queue. Similar to the scalar core, the vector unit has two arithmetic function units (VAFU), one of which can perform fused multiply-adds. The vector unit has a general purpose register file which comprise of 32 vector registers, each holding 2 Kbits of data along with a flag, scalar, and control register file used to assist with vector operations and strip-mining.

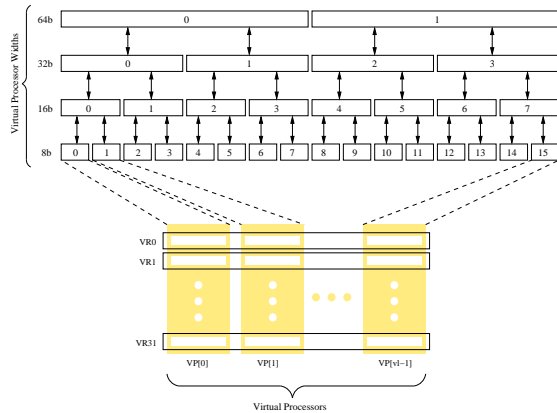


Figure 3: IRAM vector register partitioning

4 Methodology

4.1 Testing Conditions

The underlying SVD implementation was the `sgesvd()` function and dependencies from CLAPACK, a C implementation of LAPACK v2.0 created by a Fortran to C converter. We removed most of the inefficiencies generated by the automated conversion and then ported the code to the three architectures.

The code was optimized on the architectures in the following manner:

- **C67** — The code was developed on CodeComposer v1.0 and compiled with the v3.01 compiler with maximum optimizations. The functions `sgesvd()`, `sgemv()`, `slasr()`, `slartg()`, and the majority of `snrm2()` were coded in 1,700 lines of assembly code. The assembly code made extensive usage of software pipelining with a small amount of additional loop unrolling.
- **TM 1300** — Only C level optimizations, which

consisted mainly of loop unrolling, were made.

- **VIRAM** — Non-vectorizable SVD code was compiled using the IRIX compiler using standard optimizations. Vectorizable code was compiled using a modified Cray vectorizing compiler. The kernels identified as the most time consuming were `sgesr()` and `sgemv()` which were hand optimized at the assembly level..

The SVD input was a set of randomly generated matrices designed to have a rank of 10. The matrix dimensions ranged from 100x10 to 300x30 in steps of 50. Also tested were 50x5 and 1000x100 to give results for the extremes.

All processors were normalized to run at 200MHz. Both the C67 and TM 1300 were simulated with ideal memory systems. Although ideal memory systems don't exist, some testing showed that the C67 would perform only 10% worse if 4-way banked on chip SRAM were employed.

For the IRAM there was the issue of simulator inaccuracies. The `vsim-p` simulator issues scalar code at exactly 1.0 CPI [4]. This is problematic because scalar code issue dominates, particularly in the smaller data sets giving pessimistic results for the IRAM. Theoretically, the IRAM should be able to issue at 2 instructions per cycle but cache misses, branch failures, and scheduling hazards should lower the issue rate making 1.0 CPI a good approximation.

4.2 Energy Analysis

Energy analysis on these architectures is somewhat trickier. Since neither the VLIW or VIRAM simulators support power or energy analysis we will use the average power consumption from published datasheets for the VLIW designs and VIRAM SPICE estimations. What is important to note is that all three designs either run at different frequencies or are built on top of different process technologies making it inappropriate to equate IC characteristics with architectural decisions. In order to deal with this discrepancy, we will standardize process technologies to a 0.18μ process at a 0.4V threshold voltage using first order approximations. This will change the performance and power consumption used on the Trimedia. To calculate delay and power we first calculate the change in transistor gate capacitance based on the channel length and oxide thickness [12]. Capacitance for a short-channelled device is defined as:

$$(1) \quad C_{\infty} \frac{A}{T_{ox}} = L^2 \frac{\epsilon}{T_{ox}}$$

Since delay is proportional to capacitance, we can recalculate the average power and delay of the TM1300. Each processor will be compared using their default configuration under a normalized process technology and under a scaled configuration. For the VLIW architectures this means raising the voltage from the default configuration in order to meet the 200MHz frequencies as defined by our tests using the

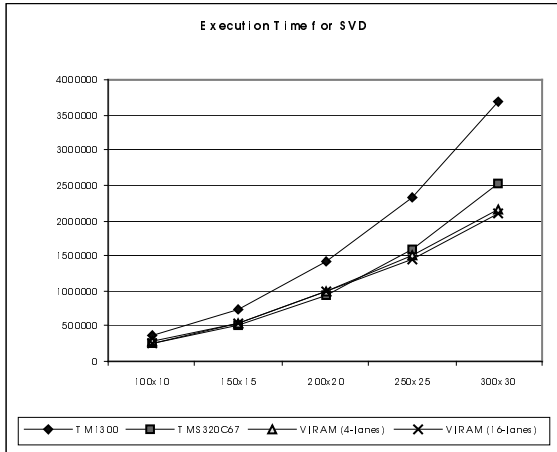


Figure 5: Execution time of all architectures measured in cycles. All tests are normalized to 200Mhz.

following equation:

$$(2) \quad \text{Delay} \propto \frac{L^2 V_{dd}}{(V_{dd} - V_{th})^{1.3}}$$

The new power consumption numbers are simply scaled based on the value CV^2f , where f is the clock speed. The IRAM will be scaled to a 16-lane configuration. Assuming that dynamic power dominates most of the power consumption, the delay, power-delay-product (PDP), and the energy-delay-product (EDP) of each design will be calculated based on the 200x20 data set for SVD [13].

5 Evaluation

5.1 Performance

5.1.1 TM1300 vs. TMS320C67 Performance

Figure 4 shows that the TM 1300 doesn't ever match the performance of the C67 with the C67 performing a steady 45% better for the entire range of matrix sizes. On the 1000x100-matrix size in figure 5, the C67 performs 60% better¹. C67 has two advantages, assembly code and issue width. Texas Instruments encourages programmers to use assembly code for the best performance and code size while Philips discourages the usage of assembly code. As a result, the C67 runs carefully hand optimized assembly code, while the TM 1300 runs optimized C code. On the architectural side, the C67 has the advantage of 3 more issue slots than the TM 1300. The main kernels are memory bound, and both the processors have enough issue slots to saturate their two load/store units. However for some of the less important but still significant kernels, the TM 1300 lacks the necessary issue width to saturate the memory system while the C67 does not.

¹ The 60% figure is calculated from a test run of an older version of the code. The new code runs 16% faster on a wide range of dimensions and should bring the difference to 85%.

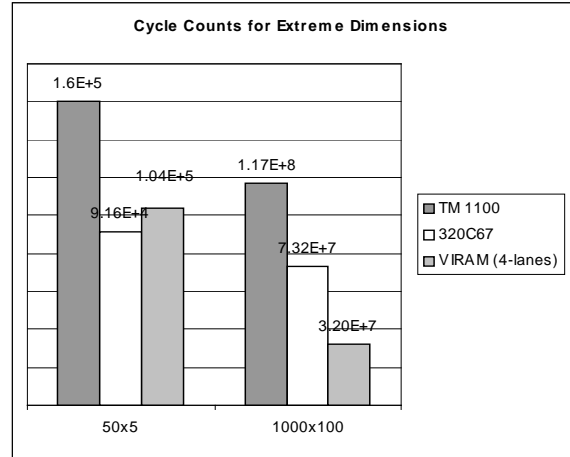


Figure 6: Cycle counts for more extreme sizes. Note that the scale for 50x5 and 1000x100 are different. The presented C67 data for 1000x100 is from an older test run and is estimated to be 16% higher than it should be.

On all the hand coded C67 kernels, it never had a problem saturating the memory system.

The long instruction latencies of the C67 compared to the TM 1300 are not a problem. Through careful coding and software pipelining, it is possible to almost completely hide these long latencies.

It should be noted here that the poor TM 1300 performance for 50x5 in figure 5 is from a slight shifting of the balance between kernels that occurs as the matrix approaches small sizes. The TM 1300 code is not as extensively optimized as the code for the other processors, and the shifting caused some unoptimized routines to consume a larger proportion of the computing time.

5.1.1 VLIW vs. VIRAM Performance

Despite the vectorizability of the SVD, VIRAM performance is only marginally better than VLIW for some of the tested matrix dimensions and performs only about 22% better at 300x30. The performance ratio between VIRAM and VLIW rises slowly with increasing matrix size, and at 1000x100, a size that is probably large enough for the ratio to have maximized, VIRAM performs 2.3 times better.

A surprising result was the fact that the C67 could match the performance of the VIRAM for all matrix sizes from 250x25 and below. In fact, for much of that range, the C67 actually performs slightly better, and at 50x5 the C67 performs 13% better. The similar level of performance over such a wide range of dimensions is actually a product of chance. There is nothing fundamental about the SVD or the tested implementation of it that would dictate such an occurrence. The two processors are limited by different factors — the C67 by bandwidth and the VIRAM by the scalar core.

5.2 Improving Performance

5.2.1 Scaling VIRAM

For the VIRAM, there is the option to scale by adding lanes to each functional unit, lowering the execution latencies of vector instructions. The rate at which vector operations need to be completed is simply the product of the average vector length and the rate of vector issue. This means that performance will increase only when computations are long and scalar execution is fast enough to saturate the vector core.

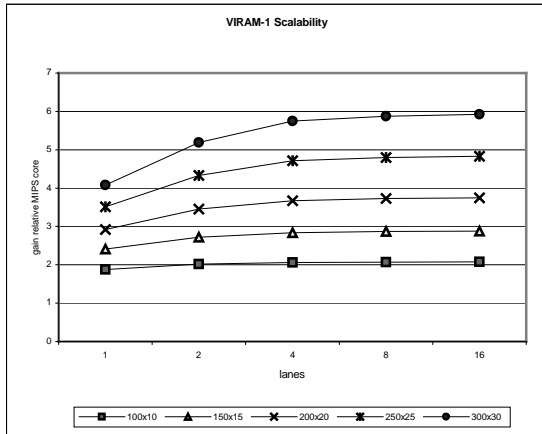


Figure 4: VIRAM-1 scalability under various lane configurations.

What we find from scaling the VIRAM for SVD is that the gains as a result of adding vector lanes diminish per lane increase. A more interesting perspective is to look at the vector core utilization (shown in Figure 6) where it is apparent that the VIRAM vector core is having problems issuing vector instructions at a high enough rate.

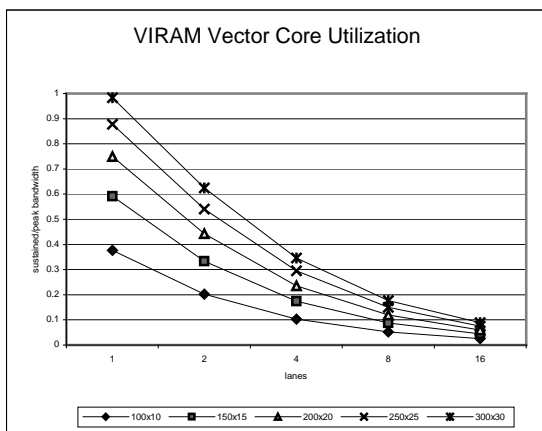


Figure 5: vector core utilization across different lane configurations.

The reason for this is clear when we look at the results. Given the best-case scenario under SVD, the average vector length is 20 (far short of the 64 element maximum vector length) while scalar code comprise 85% of all instructions issued.

Therefore, under a 1.0 CPI, there is 6-cycle window to execute all 20 vector-operations where a 4-lane design could complete in only 5 cycles, which is not enough to saturate the vector unit. Statistically, however, there is enough variance in the vector lengths and instruction issue rate that adding extra lanes would increase the sustained throughput rate but this number greatly diminishes at 4-lanes. These results are pessimistic, however, since the IRAM has a maximum issue rate of 2 instructions per cycle (twice the rate that `vsim-p` issues). It is possible that the IRAM can scale efficiently up to 8-lanes. The lesson from this is that a vector unit's scalability is limited by the performance of the scalar core so that increasing the performance of the IRAM not only involves scaling the vector unit but applying traditional architectural speedup techniques to the scalar unit.

5.2.2 Improving VLIW

Neither of the two VLIW processors was designed with linear algebra applications in mind, so there are a number of modifications which can improve their performance. The following is a short list possible improvements:

- Fused multiply-add — Both VLIW processors lack this capability even though almost all floating point operations in the code take this form. The inclusion of the instruction would reduce latency, code size, issue slot consumption, and register usage due to easier scheduling.
- More load/store units — The addition of two more load/store units would give much needed bandwidth.
- Packed instructions — This can be used to vastly increase both computational power and bandwidth without increasing issue slot usage. Strong unaligned memory address support would make such instructions much easier to use. The C67 included an instruction to load 64-bit words from memory, but the extra code complexity necessary to deal with the unaligned memory access issues and the added register consumption prevented its incorporation into the code.
- More registers (C67 only) — Despite the simplicity of the functions, the main kernels tended to require 30 out of 32 registers. To achieve the next level of performance, at least 64 registers are necessary.

5.3 Energy

Scaling is important because it has implications in energy and performance. In order to meet a certain performance criteria, a processor may need to be scaled [12]. For VLIW, this means increasing the voltage supply which increases energy dissipated per transition. On the other hand, the IRAM could potentially gain performance without expending more energy. The following table shows the results of scaling the frequencies for the VLIW and the lanes for the IRAM.

	TM1300 (default)	TM1300 (scaled)	TMS320C67 (default)	TMS320C67 (scaled)	VIRAM-1 (default)	VIRAM-1 (scaled)
Clock (Mhz)	166Mhz	200Mhz	166Mhz	200Mhz	200Mhz	200Mhz
Voltage (V)	1.5V	2.0V	1.8V	2.7V	1.2V	1.2V
Lanes	N/A	N/A	N/A	N/A	4-lanes	16-lanes
Power (W)	0.8W	1.71W	1.7W	4.6W	3.45W	3.53W
Delay (ms)	8.57ms	7.12ms	5.67 ms	4.71ms	5.02ms	4.91ms
PDP (mJ)	6.86mJ	12.18mJ	9.639mJ	21.67mJ	17.3mJ	17.3mJ
EDP (J x s)	5.88 E-5	8.67 E-5	5.47 E-5	10.2 E-4	8.68 E-5	8.49 E-5

Table 1: Power and energy numbers for all architectures for 200x20 data sets under the SVD algorithm [6].

These numbers show the IRAM's overall energy and energy delay product is slightly lower than those of the VLIW architectures under normalized testing conditions. It must be noted, however, that the TM1300's energy numbers are a result of first order approximations of technology scaling which could affect the accuracy of these numbers. In comparison to the TMS320C67, the IRAM is slightly more energy efficient than the C67 and offers higher performance at a lower energy cost.

6 Conclusion

Despite the large differences between VIRAM and VLIW, there isn't a significant performance difference between the two as measured by execution time, energy, or the energy-delay product. In fact, for matrix sizes of 250x25 and below the execution times are essentially identical, but this will most likely change in VIRAM's favor if it were simulated with its superscalar core rather than a scalar one. Energy and the energy-delay product vary a bit more, but energy varies by at most a factor 1.7 and the energy-delay product by at most 1.2 amongst the three processors.

Our analysis shows that the architectural bottleneck for VIRAM is its slow scalar core while memory bandwidth is the limiting factor for the C67. The TM 1300 suffers from a memory bandwidth deficiency as well but has the added bottleneck of insufficient issue width at certain times.

7 Future Directions

New more efficient and parallel algorithms for the SVD have been developed in recent years. It would be interesting to see if that changes the performances of these processor relative to one another.

Another possibility is to study the performance of a VLIW core with a vector coprocessor. Under such a configuration, the two technologies are complementary rather than competing. The VLIW core will provide the badly needed scalar performance while the vector core efficiently executes all the vectorizable code.

References

[1] Asanovic, K., *Vector Microprocessors*, Technical report

UCB/CSD-98-1014, University of California at Berkeley, May 1998.

[2] Castille, K., *TMS320C6000 Power Consumption Summary*, Technical report SPRA486B, Texas Instruments, November 1999.

[3] Fromm, R., *Vector IRAM Memory Performance for Image Access Patterns*, Technical report, University of California at Berkeley, January 2000.

[4] Fromm, R., *Vector IRAM Performance Modeling: vsim-p – The Performance Simulator*, Technical report, University of California at Berkeley, 2000.

[5] Fromm, R., Perissakis, S., Cardwell, N., Kozyrakis, C., McGaughy, B., Patterson, D., Anderson, T., Yelick, K., "The Energy Efficiency of IRAM Architectures." *ISCA 1997 24th Annual International Symposium on Computer Architecture*, Denver, CO, USA, 2-4 June 1997.

[6] Gebbis, J., and Kozyrakis, C. Personal communication, May, 2000.

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

[8] Kozyrakis, C., *A Media-Enhanced Vector Architecture for Embedded Memory Systems*, Technical Report UCB/CSD-99-1059, Computer Science Division, University of California at Berkeley, July 1999.

[9] Liu, K. J. R, O'Leary, Dianne P., Stewart, G. W., and Wu, Yuan-Jye J., "URV ESPRIT for Tracking time-Varying Signals," Department of Electrical Engineering and Institute of Systems Research, University of Maryland, March 14, 1994.

[10] Martin, D., *Vector Extensions to the MIPS-IV Instruction Set Architecture*. Computer Science Division, University of California at Berkeley, January 1999.

[11] Patterson, D., Asanovic, K., Brown, A., Fromm, R., Golbus, J., Gribstad, B., Keeton, K., Kozyrakis, C., Martin, D., Perissakis, S., Thomas, R., Treuhaft, N., and Yelick, K., "Intelligent RAM (IRAM): the Industrial Setting, Applications, and Architectures." *ICCD 1997 International Conference on Computer Design*, Austin, TX, USA, 10-12 October 1997.

[12] Rabaey, J., *Digital Integrated Circuits: A Design Perspective*. Prentice Hall Electronics and VLSI Series, Saddle River, NJ, 1996

[13] Rabaey, J., Personal communication, May 2000.

[14] *TMS320C6000 CPU and Instruction Set Reference Guide*. Technical report SPRU189E, Texas Instruments, January 2000.

[15] *TMS320C6000 Peripherals Reference Guide*. Technical report SPRU190C, Texas Instruments, April 1999.

[16] *Trimedia TM1300 Data Book*. Philips Electronics, July 1999.