In the Proceedings of the 35th International Symposium on Microarchitecture, Instabul, Turkey, November 2002

Vector Vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks

Christoforos Kozyrakis
Electrical Engineering Department
Stanford University
christos@ee.stanford.edu

David Patterson
Computer Science Division
University of California at Berkeley
pattrsn@cs.berkeley.edu

Abstract

Multimedia processing on embedded devices requires an architecture that leads to high performance, low power consumption, reduced design complexity, and small code size. In this paper, we use EEMBC, an industrial benchmark suite, to compare the VIRAM vector architecture to superscalar and VLIW processors for embedded multimedia applications. The comparison covers the VIRAM instruction set, vectorizing compiler, and the prototype chip that integrates a vector processor with DRAM main memory.

We demonstrate that executable code for VIRAM is up to 10 times smaller than VLIW code and comparable to x86 CISC code. The simple, cache-less VIRAM chip is 2 times faster than a 4-way superscalar RISC processor that uses a 5 times faster clock frequency and consumes 10 times more power. VIRAM is also 10 times faster than cache-based VLIW processors. Even after manual optimization of the VLIW code and insertion of SIMD and DSP instructions, the single-issue VIRAM processor is 60% faster than 5-way to 8-way VLIW designs.

1 Introduction

The exponentially increasing performance and generality of superscalar processors has lead many to believe that vector architectures are doomed to extinction. Even in the supercomputing domain, the traditional application of vector processors, it is widely considered that interconnecting superscalar processors into large-scale MPP systems is the most promising approach [4]. Nevertheless, vector architectures provide us with frequent reminders of their capabilities. The recently announced Japanese Earth Simulator, a supercomputer based on NEC SX-6 vector processors, provides 5 times the performance with half the number of nodes of ASCI White, the most powerful supercomputer based on superscalar technology. Vector processors remain the most effective way to exploit data-parallel applications [20].

This paper studies the efficiency of vector architectures for the emerging computing domain of multimedia programs running on embedded systems. Multimedia programs such as video, speech recognition, and 3D graphics, constitute the fastest growing class of applications [5]. They require real-time performance guarantees for data-parallel tasks that operate on narrow numbers with limited temporal locality [6]. Embedded systems include entertainment devices, such as set-top-boxes and game consoles, and portable electronics, such as PDAs and cellular phones. They call for low power consumption, small code size, and reduced design and programming complexity in order to meet the cost and time-to-market requirements of consumer electronics. The complexity, power consumption, and lack of explicit support for data-level parallelism suggest that superscalar processors are not necessarily a suitable approach for embedded multimedia processing.

To prove that vector architectures meet the requirements of embedded media-processing, we evaluate the VIRAM vector architecture with the EEMBC benchmarks, an industrial suite for embedded systems. Our evaluation covers all three components of VIRAM: the instruction set, the vectorizing compiler, and the processor microarchitecture. We show that the compiler can extract a high degree of data-level parallelism from media tasks described in C and can express it with vector instructions. The VIRAM code is significantly smaller than code for RISC and VLIW architectures and is comparable to that for x86 CISC processors. We describe a simple, low power, prototype chip that integrates the VIRAM architecture with embedded DRAM. The cache-less vector processor is 2 times faster than a 4-way superscalar processors running at a 5 times higher clock frequency. Despite issuing a single instruction per cycle, it

is also 10 times faster than 5-way to 8-way VLIW designs. We demonstrate that the vector processor provides performance advantages for both highly vectorizable benchmarks and partially vectorizable tasks with short vectors.

The rest of this paper is structured as follows. Section 2 summarizes the basic features of the VIRAM architecture. Section 3 describes the EEMBC embedded benchmarks. Section 4 evaluates the vectorizing compiler and the use of the vector instruction set. It also presents a code size comparison between RISC, CISC, VLIW, and vector architectures. Section 5 proceeds with a microarchitecture evaluation in terms of performance, power consumption, design complexity, and scalability. Section 6 presents related work and Section 7 concludes the paper.

2 Vector Architecture for Multimedia

In this section, we provide an overview of the three components of the VIRAM architecture: the instructions set, the prototype processor chip, and the vectorizing compiler.

2.1 Instruction Set Overview

VIRAM is a complete, load-store, vector instruction set defined as a coprocessor extension to the MIPS architecture. The vector architecture state includes a vector register file with 32 entries that can store integer or floating-point elements, a 16-entry flag register file that contains vectors with single-bit elements, and a few scalar registers for control values and memory addresses. The instruction set contains integer and floating-point arithmetic instructions that operate on vectors stored in the register file, as well as logical functions and operations such as population count that use the flag registers. Vector load and store instructions support the three common access patterns: unit stride, strided, and indexed. Overall, VIRAM introduces 90 unique instructions, which, due to variations, consume 660 opcodes in the coprocessor 2 space of the MIPS architecture.

To enable the vectorization of multimedia applications, VIRAM includes a number of media-specific enhancements. The elements in the vector registers can be 64, 32, or 16 bits wide. Multiple narrow elements are placed in the storage location for one wide element. Similarly, each 64-bit datapath is partitioned in order to execute multiple narrower element operations in parallel. Instead of specifying the element and operation width in the instruction opcode, we use a control register which is typically set once per group of nested loops. Integer instructions support saturated and fixed-point arithmetic. Specifically, VIRAM includes a flexible multiply-add model that supports arbitrary fixed-point formats without using accumulators or extended-precision registers. Three vector instructions implement element permutations within vector registers. Their scope is limited to the vectorization of dot-products (reductions) and FFTs, which makes them regular and simple to implement. Finally, VIRAM supports conditional execution of element operations for virtually all vector instructions using the flag registers as sources of element masks [21].

Scalar Core	Single-issue 64-bit MIPS pipeline 8K/8K direct-mapped L1 I/D caches
Vector Coprocessor	8K vector register file (32 registers) 2 pipelined arithmetic units 4 64-bit datapaths per arithmetic unit 1 load-store unit (4 address generators) 256-bit memory interface
Memory System	13 MBytes in 8 DRAM banks 25ns random access latency 256-bit crossbar interconnect
Technology	0.18μm CMOS process (IBM) 6 layers copper interconnect
Transistors	120M (7.5M logic, 112.5M DRAM)
Clock Frequency	200 MHz
Power Dissipation	2 Watts
Peak Performance	Int: 1.6/3.2/6.4 Gop/s (64b/32b/16b) FP: 1.6 Gflop/s (32b)

Table 1. The characteristics of the VIRAM vector processor chip.

The VIRAM architecture includes several features that help with the development of general-purpose systems, which are not typical in traditional vector supercomputers. It provides full support for paged virtual addressing using a separate TLB for vector memory accesses. It also provides a mechanism that allows the operating system to defer the saving and restoring of vector state during context switches, until it is known that the new process uses vector instructions. In addition, the architecture defines valid and dirty bits for all vector registers that are used to minimize the amount of vector state involved in a context switch.

Detailed descriptions of the features and instructions in the VIRAM architecture are available in [13].

2.2 Microarchitecture Overview

The VIRAM prototype processor is a simple implementation of the VIRAM architecture. It includes an on-chip main memory system[1] based on embedded DRAM technology that provides the high bandwidth necessary for a vector processor at moderate latency. Table 1 summarizes the basic features of the chip.

[1]The processor can address additional, off-chip, main memory. Data transfers between on-chip and off-chip are under software control.

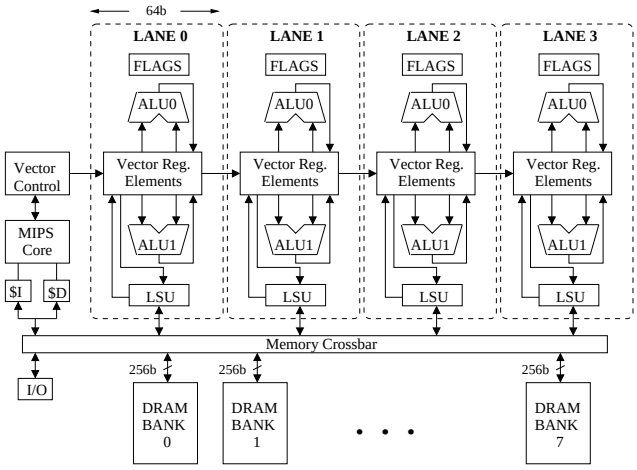

Figure 1. The microarchitecture of the VIRAM vector processor chip.

Figure 1 presents the chip microarchitecture, focusing on the vector hardware and the memory system. The register and datapath resources in the vector coprocessor are partitioned vertically into four identical vector lanes. Each lane contains a number of elements from each vector and flag register and a 64-bit datapath from each functional unit. The four lanes receive identical control signals on each clock cycle. The use of parallel lanes is a fundamental concept in the microarchitecture that leads to advantages in performance, design complexity, and scalability. Assuming sufficiently long vectors, VIRAM achieves high performance by executing on the parallel lanes multiple element operations for each pending vector instruction. Since the four lanes are identical, design and verification time is reduced significantly. Lanes also eliminate most long communication wires that complicate scaling in CMOS technology [12]. The execution of an element operation for all vector instructions, excluding memory references and permutations, involves register and datapath resources within a single lane. Finally, the modular implementation provides a simple way for scaling up or down the performance, power consumption, and area (cost) of the vector processor by allocating the proper number of vector lanes.

The VIRAM pipeline is single-issue and in-order. Vector load and store instructions access DRAM main memory directly without using any SRAM caches. In order to hide the latency of random accesses to DRAM, both load-store and arithmetic vector instructions are deeply pipelined (15 stages). The processor operates at just 200 MHz. VIRAM achieves high performance by executing multiple element operations per cycle on the parallel vector lanes. The intentionally low clock frequency, along with the in-order, cache-less organization allow for low power consumption. They also contribute to reduced design complexity. The 120-million transistor chip was designed by 3 full-time and 3 part-time graduate students over a period of 3 years.

The microarchitecture and design of the VIRAM chip is described in details in [14].

2.3 Vectorizing Compiler

The VIRAM compiler is based on the PDGCS compilation system for Cray supercomputers such as C90-YMP, T3E, and SV2. The front-end allows the compilation of programs written in C, C++, and Fortran90. The PDGCS optimizer has extensive capabilities for automatic vectorization, including outer-loop vectorization and handling of partially vectorizable language constructs.

The two main challenges in adopting a supercomputing compiler to a multimedia architecture were the support for narrow data types and the vectorization of dot-products. We modified the compiler to select the vector element and operation width for each group of nested loops using two passes. During the first pass, it records the minimum data width that satisfies the accuracy requirements of the arithmetic operations in the loop nest. In the second pass, it performs the actual vectorization at the selected width. The compiler also recognizes linear recurrences on common arithmetic, logical, and comparison operations for reductions, and uses the permutation instructions in order to vectorize them.

The code generator in the VIRAM compiler produces correct scalar code for MIPS and vector code for VIRAM. Due to time constraints, it does not include some basic back-end optimizations. It does not move the code for generating constants outside of the loop body (code motion). Due to some legacy code from early Cray machines, it occasionally attempts to use only 8 of the 32 vector registers, which leads to unnecessary spill code. Finally, it does not perform basic block scheduling for the static pipeline of the prototype chip. In other words, it does not take into account the functional unit mix and latencies in the VIRAM processor, while scheduling the instructions within each basic block. The missing back-end optimizations can have a significant effect on code size and performance (see Sections 4 and 5 respectively). However, these optimizations are straightforward to add in the future and have been available for years in all commercial compilers. None of the missing optimizations affects automatic vectorization, which is by far the most crucial compiler task for a vector architecture.

Further details on the algorithms used in the VIRAM compiler are available in [15].

3 Embedded Multimedia Benchmarks

To evaluate the efficiency of the VIRAM instruction set, microarchitecture, and compiler, we used the EEMBC

Consumer Category	
Rgb2cmyk	Converts an RGB image the CMYK format
Rgb2yiq	Converts an RGB image to the YIQ format
Filter	High-pass gray-scale image filter
Cjpeg	JPEG image compression
Djpeg	JPEG image decompression
Telecommunications Category	
Autocor	Voice compression using autocorrelation
Convenc	Convolutional encoder for modems
Bital	Bit allocation to frequency bins for ADSL
Fft	256-point fixed-point fast Fourier transform
Viterbi	Viterbi decoding for wireless applications

Table 2. The benchmarks in the consumer and telecommunications categories of the EEMBC suite.

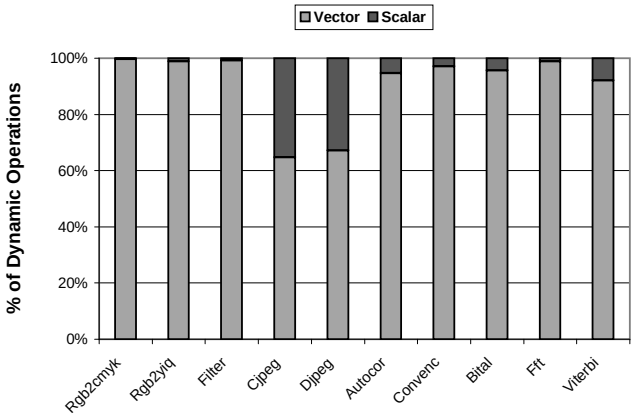

Figure 2. The distribution of vector and scalar operations.

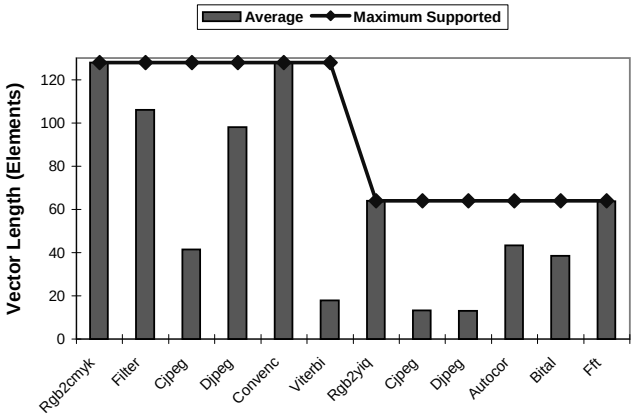

Figure 3. The average vector length in elements.

benchmarks [16]. The EEMBC suite has become the defacto industrial standard for comparing embedded processors. It includes five benchmark categories that represent a wide range of embedded tasks. In this work, we focused on the consumer and telecommunications categories that represent the typical workload of consumer devices that combine multimedia applications with high bandwidth, wired or wireless connectivity. Table 2 presents the ten benchmarks in the two categories. The benchmarks are written in C and use integer and fixed-point arithmetic. In all cases, we used the reference datasets provided by EEMBC.

The comparison metrics for EEMBC are performance and static code size. Performance for individual benchmarks is reported in iterations (repeats) per second. A composite score that is proportional to the geometric mean of the individual benchmark scores summarizes each category. With both individual and composite scores, a higher score indicates higher performance. To measure performance reliably, EEMBC recommends running each benchmark tens of times. However, this provides an unfair advantage to cache-based architectures, because it creates artificial temporal locality. After a couple of iterations, even an 8-KByte cache can effectively capture the small datasets for these benchmarks, which allows a cache-based processor to operate as if it had a single-cycle memory latency. The input data for real multimedia applications are streaming, which means that kernels are repeated numerous times, but always with new inputs coming from main memory or other IO devices.

EEMBC allows for two modes of measurement. The out of the box mode uses the binaries produced with direct compilation of the original benchmark code. The optimized mode allows for extensive optimizations of the C or assembly code, but prohibits any algorithmic changes. We used the optimized mode to perform the basic back-end optimizations missing from the code generator in the VIRAM compiler. However, we did not apply any other optimizations, such as loop unrolling or software pipelining. We did not restructure the original benchmark code either, even though it would be very beneficial for Cjpeg and Djpeg.

4 Instruction Set Evaluation

This section presents an evaluation of the ability of the VIRAM instruction set and the compiler to express the data-level parallelism in the EEMBC benchmarks. Unless otherwise stated, the results represent dynamic measurements using the code generated by the compiler.

4.1 Vectorization

Figure 2 presents the percentage of operations specified by vector instructions, also known as the degree of vectorization. A high degree of vectorization suggests that the

compiler is effective with discovering the data-level parallelism in each benchmark and expressing it with vector instructions. It also indicates that a significant portion of the execution time can be accelerated using vector hardware.

The degree of vectorization is higher than 90% for 8 out of 10 benchmarks. This is the case even for Viterbi, which is considered difficult to parallelize and is only partially vectorized by the compiler. The remaining 10% includes the operations in scalar instructions and the overhead of vector-scalar communication over the coprocessor interface. Cjpeg and Djpeg have the lowest degree of vectorization, approximately 65%. They include functions for Huffman encoding which are difficult to vectorize. Partial vectorization of these functions would only be possible after significant restructuring of the C code.

Figure 3 presents the average vector length, in other words the average number of data-parallel element operations specified by a vector instruction. Most benchmarks use either 16-bit or 32-bit arithmetic operations, for which the maximum supported vector length in the VIRAM processor is 128 and 64 elements respectively. Since Cjpeg and Djpeg use both data types at different points, we report two separate averages for them. Long vectors are not necessary but they are highly desirable. Operations on long vectors can exploit multiple datapaths in parallel vector lanes for several clock cycles. They also lead to power efficiency as a single instruction fetch and decode defines a large number of independent operations.

For five benchmarks, the vector length is close to the maximum supported for the data type they use. The average vector length is slightly lower for Autocor and Bital due to the use of reduction operations, which progressively reduce a long vector down to a single element sum. For Cjpeg, Djpeg, and Viterbi, a vector instruction defines 10 to 20 element operations on the average. In the first two, the inner loop for the DCT transformation on 8x8 pixel blocks is in a separate function from the outer-loop. If we were to inline the inner loop function by modifying the benchmark code, the compiler would be able to perform outer-loop vectorization, which would lead to long vectors. With Viterbi, the short vectors are fundamental to the nature of its algorithm.

4.2 Instruction Set Use

Figure 4 shows the distribution of vector operations for the ten benchmarks. Simple integer operations, such as add, shift, and compare account for 42% of all vector operations on the average. The second most frequent category is unit stride loads (17%). The average ratio of arithmetic operations to memory accesses is approximately 2:1.

In general, every vector instruction type is used in at least a couple of benchmarks, which indicates a balanced instruction set design. The permutation instructions account for only 2% of the dynamic operations count. However, they are crucial with vectorizing Autocor, Bital, and Fft, which include reduction or butterfly patterns. Similarly, conditional execution of element operations is essential with vectorizing portions of Cjpeg, Bital, and Viterbi.

4.3 Code Size Comparison

Static code size is a secondary issue for desktop and server systems, where high capacity hard disks store the application executables. Most embedded systems, however, store executables in non-volatile memory such as ROM or Flash. Small code size is important in order to reduce the capacity and, consequently, the cost of the code storage. It also helps with reducing the power consumed for fetching and decoding instructions.

We compared the density of the VIRAM executables to that for a set of CISC, RISC, and VLIW architectures for high performance embedded processors. Table 3 presents the five alternative architectures, as well as the characteristics of the specific processors we studied. We retrieved their code size and all performance scores for the benchmarks from official EEMBC reports submitted by the corresponding vendors [7]. Since no VLIW organization has been evaluated for both categories, we refer to the Trimedia VLIW architecture [19] for the consumer benchmarks and the VelociTI VLIW architecture [24] for the telecommunications benchmarks.

Table 4 presents the code size comparison. We report the size of executables in bytes, as well as the ratio to the x86 code size in parenthesis. The x86 CISC architecture includes variable length instructions and typically leads to high code density. For VIRAM, we report two numbers for each benchmark. The first one, specified as *cc*, is the size of the executable produced by the VIRAM compiler. The second one, specified as *opt*, is the size of the executable after post-processing the compiler output in order to apply the optimizations missing from the code generator. Even though all optimizations target performance improvements, the elimination of unnecessary spill code and redundant calculations of constants also lead to smaller code size. Similarly, we report two sets of numbers for the two VLIW architectures. The first set (*cc*) represents the size of executables generated by compiling the original benchmark code. The second set (*opt*) is the result of aggressive reorganization of the C code in order to maximize the amount of instruction level parallelism that the compiler can extract. The optimizations include function inlining, loop restructuring, and manual insertion of SIMD or DSP instructions. Unlike the back-end optimizations for VIRAM, the optimizations for the Trimedia and VelociTI architectures are benchmark-specific and difficult to incorporate into VLIW compilers.

For the consumer benchmarks, the original and opti-

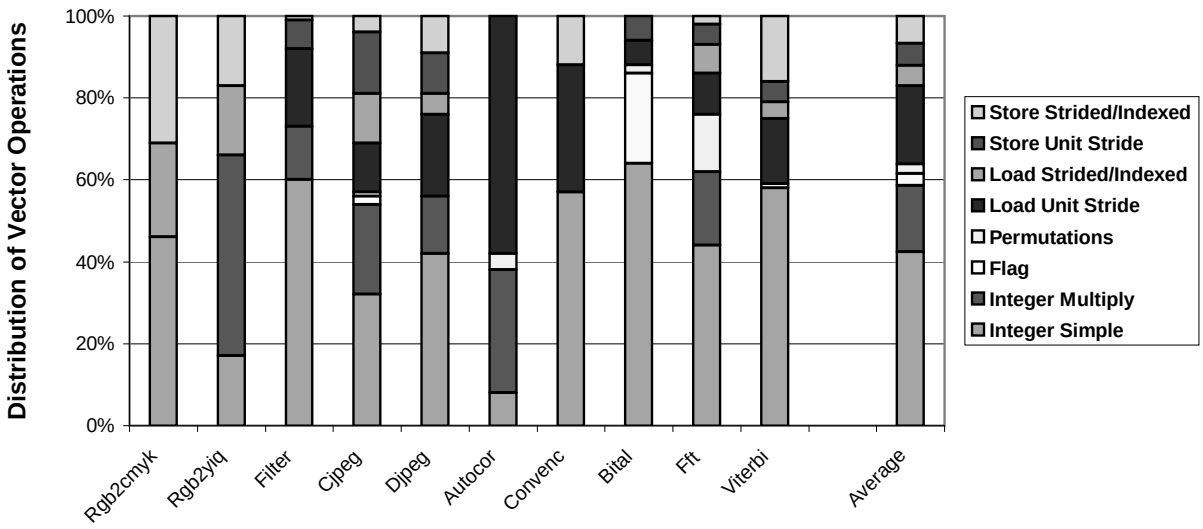

Figure 4. The distribution of vector operations into instruction categories.

	Architecture	Processor	Issue Width	Execution Style	Cache Size L1I	L1D	L2	Clock Freq.	Power
VIRAM	Vector	VIRAM	1	in order	8K	–	–	200 MHz	2.0 W
x86	CISC	K6-IIIE+	3	out of order	32K	32K	256K	550 MHz	21.6 W
PowerPC	RISC	MPC7455	4	out of order	32K	32K	256K	1000 MHz	21.3 W
MIPS	RISC	VR5000	2	in order	32K	32K	–	250 MHz	5.0 W
Trimedia	VLIW+SIMD	TM1300	5	in order	32K	16K	–	166 MHz	2.7 W
VelociTI	VLIW+DSP	TMS320C6203	8	in order	96K	512K	–	300 MHz	1.7 W

Table 3. The characteristics of the embedded architectures and processors we compare in this paper.

mized executables for VIRAM are 40% and 10% larger than the x86 code on the average. However, the VIRAM code is actually denser for Rgb2cmyk, Rgb2yiq, and Filter. Cjpeg and Djpeg contain large portions of rarely used code for error handling for which x86 leads to denser scalar code than MIPS. Optimized VIRAM code has the same code density as PowerPC. However, it is 2 times smaller than the MIPS code, 4 times smaller than the original Trimedia code, and more than 10 times smaller than the optimized Trimedia code. The fundamental reason for the good code density of VIRAM is that the compiler does not use loop unrolling or software pipelining. As we present in Section 5, the VIRAM processor achieves high performance without using the two techniques that lead to bloated code size. On the other hand, VLIW architectures depend on them heavily, especially when targeting maximum performance. VLIW architectures incur the additional penalty of empty slots in their wide instruction format. RISC architectures rely less on loop unrolling, especially if the processor implements out-of-order execution, as with the MPC7455 PowerPC chip. Despite the overhead of coprocessor move instructions, VIRAM compares favorably to the RISC architectures. Vector instructions eliminate the loop indexing overhead for small, fixed-sized loops. In addition, a single vector load-store instruction captures the functionality of multiple RISC instructions for address generation, striding, and indexing.

Table 4 shows that the comparison is similar for the telecommunication benchmarks. The optimized code for VIRAM is smaller than the x86 code for all five benchmarks, as the vector instructions for permutations eliminate the need for complicated loops to express reductions or butterflies. The VelociTI VLIW architecture produces executable as large as the two RISC architectures. VelociTI is built on top of a DSP instruction set, with features such as zero-overhead loops and special addressing modes for FFT.

If we compare separately VIRAM to MIPS, the RISC architecture it is based on, we conclude that adding vector instructions to a RISC architecture leads to significant code size reductions. Vector instructions behave as ”useful macro-instructions” for the MIPS architecture and allow for code density similar or better than that of the x86 architec-

	VIRAM (Vector)		x86 (CISC)	PowerPC (RISC)	MIPS (RISC)	Trimedia (VLIW+SIMD)	
	(cc)	*(opt)*				*(cc)*	*(opt)*
Rgb2cmyk	672 (0.9)	272 (0.4)	720 (1.0)	484 (0.7)	1,782 (2.5)	2,560 (3.6)	6,144 (8.5)
Rgb2yiq	528 (0.6)	416 (0.5)	896 (1.0)	492 (0.5)	1,577 (1.8)	4,352 (4.8)	34,560 (38.6)
Filter	1,328 (1.4)	708 (0.7)	944 (1.0)	1,188 (1.3)	1,997 (2.1)	4,672 (4.9)	3,584 (3.8)
Cjpeg	60,256 (2.0)	58,880 (1.9)	30,010 (1.0)	48,440 (1.6)	58,655 (1.9)	114,944 (3.8)	180,032 (6.0)
Djpeg	70,304 (2.0)	68,448 (1.9)	35,962 (1.0)	47,672 (1.3)	58,173 (1.6)	117,440 (3.3)	163,008 (4.5)
Average	**(1.4)**	**(1.1)**	**(1.0)**	**(1.1)**	**(2.0)**	**(4.1)**	**(12.3)**

	VIRAM (Vector)		x86 (CISC)	PowerPC (RISC)	MIPS (RISC)	VelociTI (VLIW+DSP)	
	(cc)	*(opt)*				*(cc)*	*(opt)*
Autocor	1,072 (1.9)	328 (0.6)	544 (1.0)	1,276 (2.3)	1,137 (2.1)	992 (1.8)	1,472 (2.7)
Convenc	704 (0.9)	352 (0.4)	784 (1.0)	1,788 (2.3)	1,618 (2.1)	1,120 (1.4)	2,016 (2.6)
Bital	1,024 (1.5)	592 (0.9)	672 (1.0)	1,820 (2.7)	1,495 (2.2)	2,304 (3.4)	1,376 (2.0)
Fft	3,312 (0.2)	720 (0.1)	15,670 (1.0)	5,868 (0.4)	5,468 (0.3)	2,944 (0.2)	3,552 (0.2)
Viterbi	2,592 (1.9)	1,152 (0.9)	1,344 (1.0)	6,648 (4.9)	3,799 (2.8)	1,920 (1.4)	2,560 (1.9)
Average	**(1.3)**	**(0.6)**	**(1.0)**	**(2.5)**	**(1.9)**	**(1.6)**	**(1.9)**

Table 4. The code size comparison between vector, RISC, CISC, and VLIW architectures. The numbers in parenthesis represent the ratio to the x86 code size.

ture.

5 Processor Evaluation

This section compares the VIRAM chip to the embedded processors in Table 3. Their features are representative of the great variety in high performance, embedded designs. The group includes representatives from three basic architectures (CISC, RISC, VLIW), two execution techniques (in-order, out-of-order), and a range of instruction issue rates (2 to 8), clock rates (166MHz to 1GHz), and power dissipations (1.7W to 21.6W). The only common feature among all five processors we compare to VIRAM is that they use SRAM caches for low memory latency.

Before we proceed with the performance comparison, it is interesting to discuss power consumption and design complexity. The typical power dissipation numbers in Table 3 are not directly comparable. K6-III+, VR5000, and TM1300 were designed in 0.25μm CMOS technology, VIRAM and MPC7455 in 0.18μm, and TMS320C6203 in 0.13μm. The VIRAM power figure includes the power for main memory accesses, which is not the case for any other design. In addition, the VIRAM circuits were generated with a standard synthesis flow and were not optimized for low power consumption in any way. The power characteristics of the VIRAM chip are entirely due to its microarchitecture.

Nevertheless, we can draw the following general conclusions about power consumption. Superscalar, out-of-order processors like K6-III+ and MPC7455 have the highest power consumption because of their high clock rates and the complexity of their control logic. The simplicity and lower clock frequency of VLIW processors allows for reduced power consumption despite their high instruction issue rates. The microarchitecture of the VIRAM processor allows for additional power savings. The parallel vector lanes are power efficient because they operate at a low clock frequency and use exclusively static circuits. The control logic dissipates a minimum amount of power because of its simplicity (single-issue, in-order) and because it needs to fetch and decode merely one vector instruction in order to execute tens of element operations. Finally, the on-chip main memory provides high bandwidth without wasting power for driving off-chip interfaces or for accessing caches for applications with limited temporal locality.

Design complexity is also difficult to compare unless the same group of people implements all processors in the same technology with the same tools. However, we believe that the VIRAM microarchitecture has significantly lower complexity than superscalar processors. Both the vector coprocessor and the main memory system are modular, the control logic is simple, and there is no need for caches or circuit design for high clock frequency. These properties allowed 6 graduate students to implement the prototype chip while carrying a full-time course load. On the other hand, superscalar processors include complicated control logic for out-of-order execution, which is difficult to design at high clock rates. The development of a new superscalar microarchitecture typically requires hundreds of man-years [2]. Even though, VLIW processors for embedded applications are

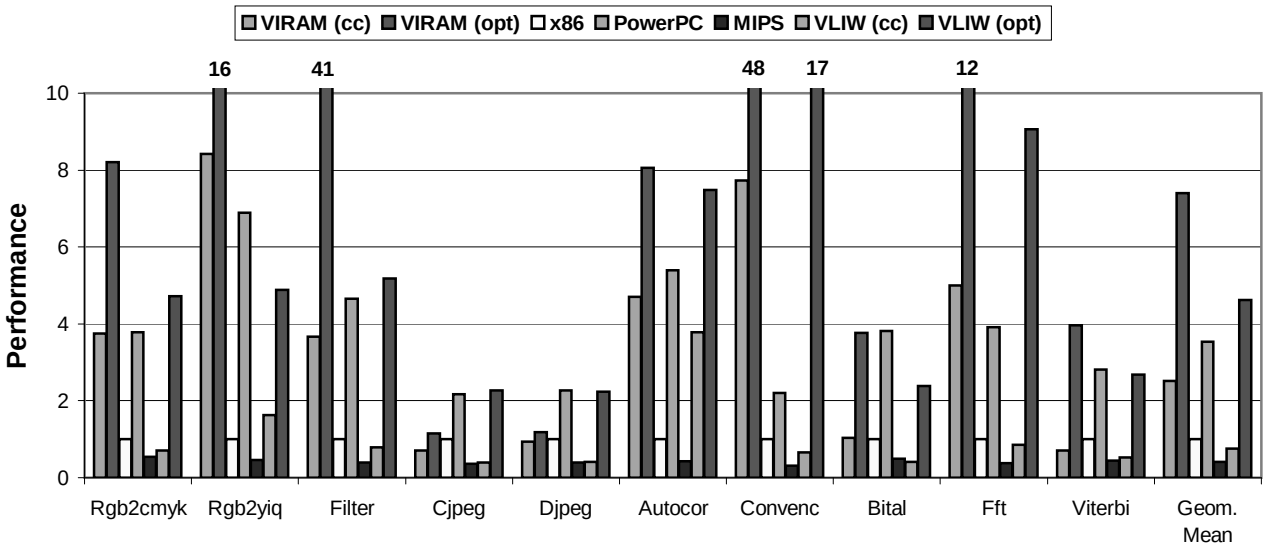

Figure 5. The performance of the embedded processors for the EEMBC benchmarks.

simpler than superscalar designs, the high instruction issue rate makes them more complicated than single-issue vector processors. In addition, VLIW architectures introduce significant complexity to the compiler development.

5.1 Performance Comparison

Figure 5 presents the performance comparison for the EEMBC benchmarks. Performance is reported in iterations per cycle and is normalized by the K6-III+ x86 processor. Even with unoptimized code, VIRAM outperforms the x86, MIPS, and the two VLIW processors (TM1300 and TMS320C605) running original code by factors of 2.5x to 6.1x for the geometric mean. It is 30% and 45% slower than the 1GHz MPC7455 PowerPC and the two VLIW processors running optimized code respectively. With optimized (scheduled) code, on the other hand, the VIRAM chip is 1.6 to 18.5 times faster than all other processors. To grasp the significance of the performance results, one should also consider that VIRAM is the only single-issue design in the processor set, it is the only one that does not use SRAM caches, and its clock frequency is the second slowest, just one fifth of that for the 4-way issue MPC7455. In addition, the VLIW performance with optimized code is the result of program-specific optimizations, which are currently not available in research or commercial compilers.

It is also interesting to separate the contributions to performance from microarchitecture from the contributions from clock frequency. VIRAM and the two VLIW designs include simple control circuits and could be clocked as fast as the superscalar processors, if not faster. Nevertheless, they use modest clock frequencies (166MHz to 300MHz) in order to reduce power consumption. Figure 6 presents the performance results normalized to the clock frequency of each processor[2]. For highly vectorizable benchmarks with long vectors, such as Filter and Convenc, VIRAM is up to 100 times faster than the MPC7445 PowerPC, the best superscalar design. Each vector instruction defines tens of independent element operations, which can utilize the parallel execution datapaths in the vector lanes for several clock cycles. On the other hand, the superscalar processors can extract a much smaller amount of instruction-level parallelism from their sequential instruction streams. For benchmarks with strided and indexed memory accesses, such as Rgb2cmyk and Rgb2yiq, VIRAM outperforms MPC7455 by a factor of 10. In this case, the limiting factor for VIRAM is the address translation throughput in the vector load-store unit, which is $\frac{1}{2}$ or $\frac{1}{4}$ of the throughput of each arithmetic unit for 32-bit or 16-bit operations respectively. For partially vectorizable benchmarks with short vectors, such as Cjpeg, Djpeg, and Viterbi, VIRAM maintains a 3x to 5x performance advantage over MPC7455. Even with just ten elements per vector, simple vector hardware is more efficient with data-level parallelism than aggressive, wide-issue, superscalar organizations.

With the original benchmark code, the two VLIW processors perform similarly to the superscalar designs. Source code optimizations and the manual insertion of SIMD and DSP instructions lead to significant improvements and allow the VLIW designs to be within 50% of VIRAM with optimized code. For the consumer benchmarks, this is due to Cjpeg and Djpeg, for which the performance of the

[2]Due to the measurement methodology for EEMBC, even cache-based designs perform almost no off-chip memory accesses (see Section 3). Hence, we can ignore the clock frequency of the off-chip memory system during normalization.

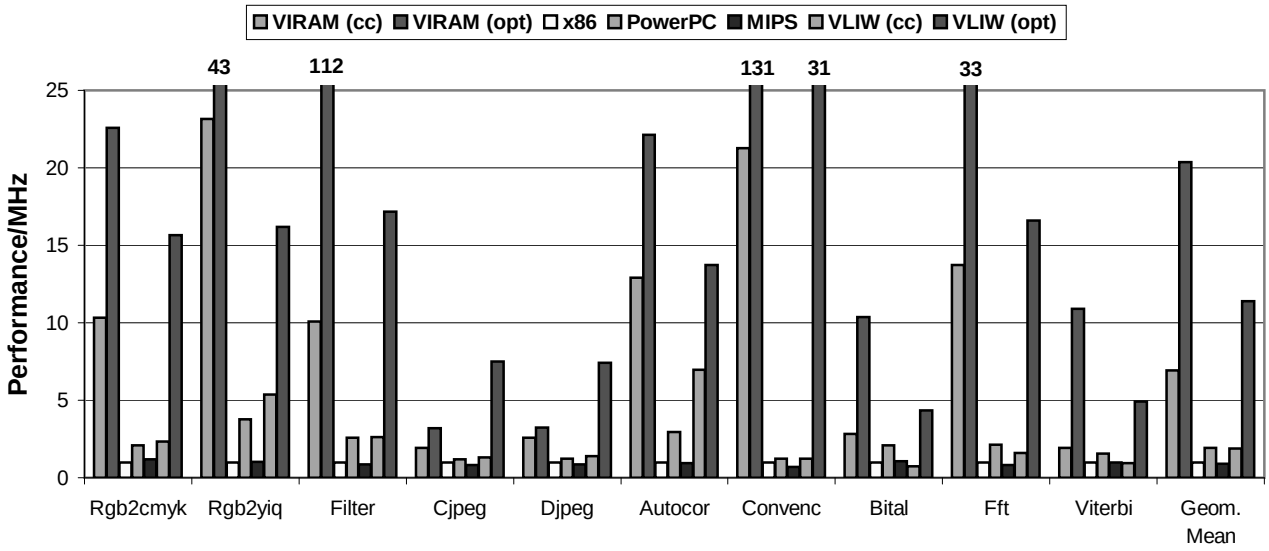

Figure 6. The performance of the embedded processors for the EEMBC benchmarks normalized by their clock frequency.

TM1300 VLIW processor improves considerably once the benchmark code is restructured in order to eliminate the function call within the nested loop for DCT. The same transformation would have been beneficial for VIRAM, as it would allow outer-loop vectorization and would lead to long vectors. For the telecommunication benchmarks, the DSP features of the TMS320C605 VLIW design lead to enhanced performance. However, it is very difficult for a compiler to achieve similar performance levels for an architecture that combines VLIW and DSP characteristics.

The results in Figure 6 would be similar if we normalized performance by the typical power consumption instead of clock frequency. The microarchitecture of VIRAM allows for both higher performance and lower power consumption than superscalar and VLIW processors for the EEMBC benchmarks. We also believe that Figure 6 would be similar if we could normalize performance by some acceptable metric of design complexity.

For reference, the EEMBC scores of the VIRAM processor for the consumer and telecommunications benchmarks are 81.2 and 12.4 respectively with unscheduled code. With optimized code, the scores are 201.4 and 61.7 respectively[3]. In addition, VIRAM running optimized vector code is over 20 times faster for all benchmarks than VIRAM running scalar code on the simple MIPS core it includes.

[3]The benchmark scores for the two categories are not directly comparable to each other.

5.2 Scalability

One of the most interesting advantages of organizing the vector hardware in lanes is that we can easily scale the performance, power consumption, and area (cost) by allocating the proper number of vector lanes. This is a balanced scaling approach, as each lane includes both register and datapath resources. Figure 7 presents the performance of the VIRAM microarchitecture as we scale the number of lanes from 1 to 8. We normalize performance to that with a single lane. In all cases, we use the same executables and we assume the same clock frequency (200 MHz) and on-chip memory system.

With each extra lane, we can execute more element operations per cycle for each vector instruction. Hence, benchmarks with long vectors benefit the most from additional lanes. For Rgb2cmyk, Convenc, and Fft, performance scales almost ideally with the number of lanes. For Rgb2yiq and Filter, the fast execution of vector instructions with 8 lanes reveals the overhead of instruction dependencies, scalar operations, and vector-scalar communication over the coprocessor interface. Autocor and Bital include reductions, which operate on progressively shorter vectors, and cannot always exploit a large number of lanes. Finally, for the benchmarks with short vectors, such as Cjpeg, Djpeg, and Viterbi, additional lanes lead to no significant performance improvements.

Overall, the geometric mean in Figure 7 shows that performance scales well with the number of lanes. Compared to the single-lane case, two, four, and eight lanes lead to

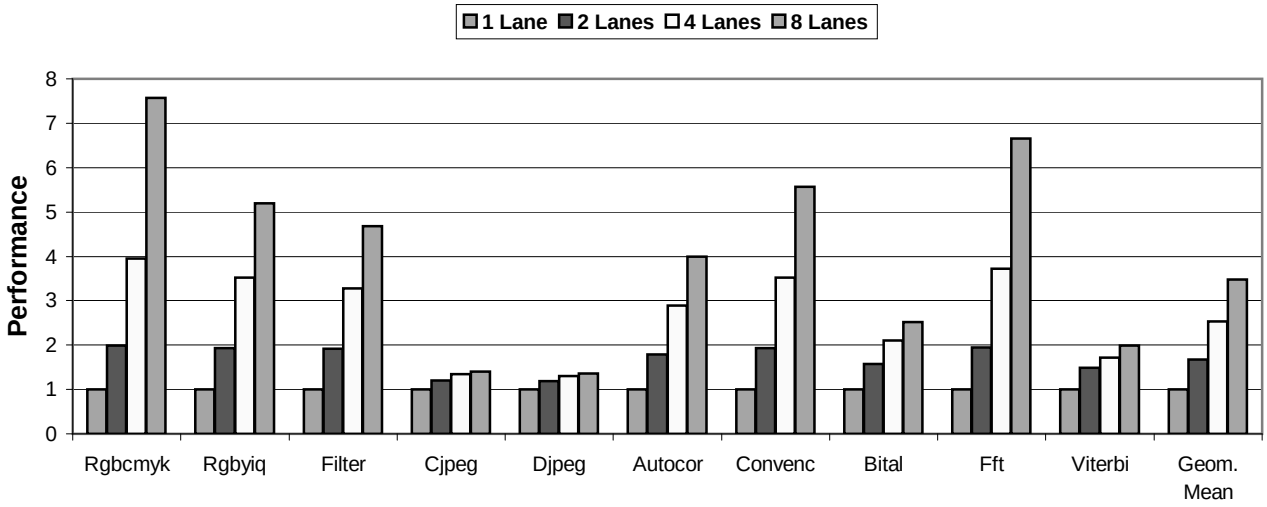

Figure 7. The performance of the VIRAM processor as a function of the number of vector lanes.

approximately 1.7x, 2.5x, and 3.5x performance improvement respectively. Comparable scaling results are difficult to achieve with other architectures. Scaling a 4-way superscalar processor to 8-way, for example, would probably lead to small overall performance improvement. It is difficult to extract such a high amount of instruction-level parallelism [25] and the complexity of the wide out-of-order logic would slowdown the clock frequency [1]. A VLIW processor could achieve similar scaling with highly optimized code, at the cost of even larger code size and increased power consumption for the wide instruction fetch.

6 Related Work

The popular approach to efficient media processing on superscalar and VLIW processors is the use of SIMD extensions, such as SSE [23] and Altivec [17]. SIMD instructions define arithmetic operations on short, fixed-length vectors of 32-bit or 16-bit elements, stored in 64-bit or 128-bit registers. In Section 4 (Figure 3), we showed that all EEMBC benchmarks include vectors with more than 4 to 8 elements. Hence, a superscalar processor must issue multiple SIMD instructions per cycle in order to express the available data-level parallelism and use parallel datapaths. In addition, SIMD extensions do not support vector memory accesses. The overhead of the sequence of load, unpack, rotate, and merge instructions necessary to emulate a strided or indexed vector access often cancels the benefits of SIMD arithmetic and complicates automatic compilation.

The majority of recent work in vector architectures has focused on adopting techniques from superscalar designs (decoupling [9], out-of-order execution [11], simultaneous multithreading [10]) in order to accelerate scientific applications. In [8], Espasa showed that Tarantula, an 8-lane vector extension to the EV8 Alpha processor, achieves a 5x speedup over a dual-processor EV8 chip of similar complexity and power consumption. Nevertheless, a few academic groups have studied vector designs with multimedia. T0, a SRAM-based vector processor, was the precursor to VIRAM processor [3]. For speech recognition software, it was 10 times faster than its contemporary superscalar designs. In [22], Stoodley and Lee showed that simple vector processors outperform superscalar designs with SIMD extensions by up to a factor of 3x for a variety of media tasks. The two main problems with the previous studies of vector processors have been the performance of the vector memory system and automatic vectorization. This paper addresses the former with the embedded DRAM main memory system in the VIRAM chip, and the latter with the Cray-based vectorizing compiler for the VIRAM architecture.

The Imagine stream processor introduces an alternative architecture for multimedia processing [18]. It is based on a set of clusters that include a datapath and a local register file. It implements computation by streaming data through the clusters under microcoded control. Imagine exposes its computation and communication resources to software, which makes it possible to exploit data-level parallelism in the most flexible way. The price for the flexibility is a complicated programming model based on a special dialect of C. In contrast, the VIRAM programming model is based on standard C or C++ with automatic vectorization.

7 Conclusions

Multimedia processing on embedded systems is an emerging computing area with significantly different needs from the desktop domain. It requires high performance for data-parallel tasks, low power consumption, reduced design

complexity, and compact code size. Despite their theoretical generality, superscalar processors cannot exploit the data-level parallelism in order to improve performance or reduce power dissipation and design complexity.

In this paper, we have demonstrated the efficiency of the VIRAM vector architecture for the EEMBC embedded benchmarks. The VIRAM code is up to 10 times smaller than VLIW code and comparable to the x86 CISC code. A simple, modular implementation of the VIRAM architecture is 2 times faster than a 4-way superscalar processor with a 5 times faster clock frequency and a 10 times higher power consumption. Despite its lack of SRAM caches and the fact that the EEMBC measurement methodology favors cache-based designs, the VIRAM chip outperforms VLIW processors by a factor of 10. Even after manual optimization of the VLIW code and insertion of SIMD and DSP instructions, the single-issue VIRAM processor is 60% faster than 5-way to 8-way VLIW designs.

The overall effectiveness of VIRAM suggests that vector architectures can play a leading role in embedded multimedia systems.

8. Acknowledgments

We would like to acknowledge all the members of the IRAM research group at U.C. Berkeley and in particular Sam Williams, Joe Gebis, Kathy Yelick, David Judd, and David Martin. This work was supported by DARPA (DABT63-96-C-0056), the California State MICRO Program, and an IBM Ph.D. fellowship. IBM, MIPS Technologies, Cray, and Avanti have made significant hardware and software contributions to the IRAM project.

References

[1] V. Agarwal, M. Hrishikesh, S. Keckler, and D. Burger. Clock Rate vs IPC: The End of Road for Conventional Microarchitectures. In *the Proceedings of the 27th Intl. Symposium on Computer Architecture*, pages 248–259, Vancouver,Canada, June 2000.

[2] A. Allan, D. Edenfeld, W. Joyner, A. Kahng, M. Rodgers, and Y. Zorian. 2001 Technology Roadmap for Semiconductors. *IEEE Computer*, 35(1):42–53, Jan. 2002.

[3] K. Asanović, J. Beck, B. Irissou, B. Kingsbury, and J. Wawrzynek. T0: A Single-Chip Vector Microprocessor with Reconfigurable Pipel ines. In *the Proceedings of the 22nd European Solid-State Circuits Conference*, Sept. 1996.

[4] G. Bell and J. Gray. What's Next in High performance Computing? *Communications of the ACM*, 45(2):91–95, Feb. 2002.

[5] W. Dally. Tomorrow's Computing Engines. Keynote Speech, the 4th Intl. Symposium on High-Performance Computer Architecture, Las Vegas, NV, Feb. 1998.

[6] K. Diefendorff and P. Dubey. How Multimedia Workloads Will Change Processor Design. *IEEE Computer*, 30(9):43–45, Sept. 1997.

[7] EEMBC Benchmark Scores. http://www.eembc.org.

[8] R. Espasa, J. Emer, et al. Tarantula: a Vector Extension to the Alpha Architecture. In *the Proceedings of the 29th Intl. Symposium on Computer Architecture*, Anchorage, AL, May 2002.

[9] R. Espasa and M. Valero. Decoupled Vector Architecture. In *the Proceedings of the 2nd Intl. Symposium on High-Performance Computer Architecture*, pages 281–90, San Jose, CA, Feb. 1996.

[10] R. Espasa and M. Valero. Simultaneous Multithreaded Vector Architecture. In *the Proceedings of the 4th Intl. Conference on High-Performance Computing*, pages 350–7, Bangalore, India, Dec. 1997.

[11] R. Espasa, M. Valero, and J. Smith. Out-of-order Vector Architectures. In *the Proceedings of the 30th Intl. Symposium on Microarchitecture*, pages 160–70, Research Triangle Park, NC, Dec. 1997.

[12] R. Ho, K. Mai, and M. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, Apr. 2001.

[13] C. Kozyrakis. *Scalable Vector Media-processors for Embedded Systems*. PhD thesis, Computer Science Division, University of California at Berkeley, 2002.

[14] C. Kozyrakis, J. Gebis, et al. VIRAM: A Media-oriented Vector Processor with Embedded DRAM. In *the Conference Record of the Hot Chips XII Symposium*, Palo Alto, CA, Aug. 2000.

[15] C. Kozyrakis, D. Judd, et al. Hardware/compiler Codevelopment for an Embedded Media Processor. *Proceedings of the IEEE*, 89(11):1694–709, Nov 2001.

[16] M. Levy. EEMBC 1.0 Scores, Part 1: Observations. *Microprocessor Report*, pages 1–7, Aug. 2000.

[17] M. Phillip. A Second Generation SIMD Microprocessor Architecture. In *the Conference Record of the Hot Chips X Symposium*, Palo Alto, CA, Aug. 1998.

[18] S. Rixner, W. Dally, et al. A Bandwidth-Efficient Architecture for Media Processing. In *the Proceedings of the 31st Intl. Symposium on Microarchitecture*, pages 3–13, Dallas, TX, Nov. 1998.

[19] G. Slavenburg, S. Rathnam, and H. Dijkstra. The Trimedia TM-1 PCI VLIW Media Processor. In *the Conference Record of the Hot Chips VIII Symposium*, Palo Alto, CA, Aug. 1996.

[20] J. Smith. The Best Way to Achieve Vector-Like Performance? Keynote Speech, the 21st Intl. Symposium on Computer Architecture, Chicago, IL, April 1994.

[21] J. Smith, G. Faanes, and R. Sugumar. Vector Instruction Set Support for Conditional Operations. In *the Proceedings of 27th Intl. Symposium on Computer Architecture*, pages 260–9, Vancouver, BC, Canada, June 2000.

[22] M. Stoodley and C. Lee. Vector Microprocessors for Desktop Computing. In *the Proceedings of the 32nd Intl. Symposium on Microarchitecture*, Haifa, Israel, Nov. 1999.

[23] A. Strey and M. Bange. Performance Analysis of Intel's MMX, and SSE. In *the Proceedings of the 7th EuroPAR Conference*, pages 142–147, Manchester, UK, Aug. 2001.

[24] C. Truong. The VelociTI Architecture of the TMS230C6x. In *the Conference Record of Hot Chips IX Symposium*, Palo Alto, CA, Aug. 1997.

[25] D. Wall. Limits of Instruction-level Parallelism. In *the Proceedings of the 4th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 176–88, Santa Clara, CA, April 1991.