Exploiting On-Chip Memory Bandwidth in the VIRAM Compiler

David Judd, Katherine Yelick, Christoforos Kozyrakis,
David Martin and David Patterson
Computer Science Division
University of California, Berkeley*

University of California, Berkeley CA 94720, USA,
{dajudd,yelick,kozyraki,dmartin,pattrsn}@cs.berkeley.edu,
WWW home page: http://iram.cs.berkeley.edu/

Abstract. Many architectural ideas that appear to be useful from a hardware standpoint fail to achieve wide acceptance due to lack of compiler support. In this paper we explore the design of the VIRAM architecture from the perspective of compiler writers, describing some of the code generation problems that arise in VIRAM and their solutions in the VIRAM compiler. VIRAM is a single chip system designed primarily for multi-media. It combines vector processing with mixed logic and DRAM to acheive high performance with relatively low energy, area, and design complexity. The paper focuses on two aspects of the VIRAM compiler and architecture. The first problem is to take advantage of the on-chip bandwidth for memory-intensive applications, include those with non-contiguous or unpredictable memory access patterns. The second problem is to support that kinds of narrow datatypes that arise in media processing, including processing of 8 and 16-bit data.

1 Introduction

Embedded processing in DRAM offers enormous potential for high memory bandwidth without high energy consumption by avoiding the memory bus bottlenecks of conventional multi-chip systems [PAC+97]. To exploit the memory bandwidth without expensive control and issue logic, the IRAM project at U.C. Berkeley is exploring the use of vector processing capabilities in a single-chip system called VIRAM, designed for multimedia applications [FPC+97]. Studies of hand-coded VIRAM benchmarks show that performance on a set of multimedia kernels exceeds that of high-end DSPs and microprocessors with media-extensions [Koz99,Tho00]. In this paper,

* This work was supported in part by the Advanced Research Projects Agency of the Department of Defense under contract DABT63-96-C-0056, by the California State MICRO Program, and by the Department of Energy. The VIRAM compiler was built using software provided by Cray, Inc.

we demonstrate that a vectorizing compiler is also capable of exploiting the vector instructions and memory bandwidth in VIRAM.

The technology for automatic vectorization is well understood within the realm of scientific applications and supercomputers. The VIRAM project leverages that compiler technology with some key differences that stem from differences in the application space of interest and in the hardware design: the VIRAM processor is designed for multimedia benchmarks and with a primary focus on power and energy rather than high performance. There is some overlap between the algorithms used in multimedia and scientific computing, e.g., matrix multiplication and FFTs are important in both domains. However, multimedia applications tend to have shorter vectors, and a more limited dynamic range for the numerical values, which permits the use of single precision floating point as well as integer and fixed point operations on narrow data types, such as 8, 16, or 32 bit values. VIRAM has features to support both narrow data types and short vectors.

2 Overview of VIRAM

The VIRAM instruction set architecture (ISA) extends the MIPS ISA with vector instructions that include integer and floating point arithmetic operations as well as memory operations for sequential, strided, and indexed (scatter/gather) access patterns. The ISA specifies 32 vector registers, each containing multiple vector elements. Logically, a vector operation specifies that the operation may be performed on all elements of a vector register in parallel, and we therefore use the abstract notion of a *virtual processor* in which there is one processor per vector element.

The maximum number of elements per register is determined by two factors: the total number of bits in a register and the width of the data on which operations are being performed. For example, the VIRAM implementation holds 2K bits, which corresponds to 32 64-bit elements, 64 32-bit elements, or 128 16-bit elements. The bit width of the data is known as the *virtual processor* *width* (VPW) and may be set by the application software and changed as different data types are used in the application.

The VIRAM implementation includes a simple in-order MIPS processor with a cache and floating point unit, a DMA engine for off-chip access, a memory cross-bar, and a vector unit which is managed as a co-processor. Both the vector and scalar processors are designed to run at 200 MHz using a 1.2V power supply, with a power target of 2 Watts [Koz99]. There are 16 MB of on-chip DRAM organized into 8 banks. The memory is directly accessible from both scalar and vector instructions. The vector unit contains a 256-bit datapath, which can be used to execute 4 64-bit operations, 8 32-bit operations, or 16 16-bit operations simultaneously. Thus, the virtual processors that one may imagine acting on an entire vector register are implemented by a smaller set of vector *lanes*, which execute a vector operation by computing a subset of the elements in each clock cycle. The implementation has 4 64-bit lanes, which can be viewed as 8 32-bit lanes or (for integer operations) 16 16-bit lanes. There are two integer functional units and one floating point unit, so each lane is capable of executing 2 integer operations or one floating point operation per cycle. The ISA allows for 32 and 64-bit floating point as well as 8, 16, 32, and 64-bit integer operations. The IRAM implementation will support only 32 bit floating point and 16, 32, and 64-bit integer operations.[1] VIRAM's peak performance is 1.6 GFLOPS for 32-bit floating point, 3.2 GOPS for 32-bit integer operations, and 6.4 GOPS for 16-bit integer operations.

3 Compiler Overview

The VIRAM compiler is based on the Cray vectorizing compiler, which has C, C++ and Fortran frontends and is used on Cray's supercomputers. In addition to vectorization, the compiler

[1] The ISA also contains fused multiply-add instructions, which will be implemented in the VIRAM chip for integer operations but not floating point. These instructions are not targeted by the compiler and will therefore not be considered here.

performs standard optimizations including constant propagation and folding, common subexpression elimination, inlining, and a large variety of loop transformations such as loop fusion and interchange. It also performs outer loop vectorization, which is especially useful for extracting large degrees of parallelism across images in certain multimedia applications. Pointer analysis in C and C++ are beyond the scope of this paper, so we use the Cray compiler strategy of requiring "restrict" pointers on array arguments to indicate they are unaliased.

The strategy in this project was to re-use as much compiler technology as possible, innovating only where necessary to support the VIRAM ISA. The Cray compiler has multiple machines targets, including vector machines like the Cray C90 and parallel machines like the T3E, but it did not support the MIPS architecture, so our first step was to build a MIPS backend for scalar code and then a VIRAM code generator for vectorized code. Differences in VIRAM and Cray vector architectures lead to more interesting differences:

1. VIRAM supports narrow datatypes (8, 16, and 32 bits as well as 64 bits). Cray vector machines support 32 and 64 bit types, but, historically, the hardware ran 32-bit operations at the same speed as 64-bit, so there was no motivation to optimize for narrower types.
2. In both Cray and VIRAM architectures, vector instruction may be computed with a flag (or mask) to effectively turn off the virtual processors at those positions. VIRAM treats flags as 1-bit vector registers, while the Cray machines treat them as 64 or 128-bit scalar values.
3. VIRAM has special instructions for fixed point arithmetic, such as saturation and other rounding modes for integer operations, which are not provided on Cray machines.
4. VIRAM allows for speculative execution of vector instructions, which is particularly important for vectorizing loops with conditions for breaking out of the loop, e.g., when a value has been found.

This list highlights the differences in the architectures as they affect compilation. In this paper we focus mainly on the first of these issues, i.e., generation of code for narrow data widths, which is critical to the use of vector processing for media processing. The last two features, fixed point computations and speculative execution, have no special support in our compiler, and while the flag model resulted in somewhat simpler code generation for VIRAM, the difference is not fundamental. In VIRAM the vector length register implicitly controls the set of active flag values, whereas the Cray compiler must correctly handle trailing flag bits that are beyond the current vector length.

4 On-Chip Memory Bandwidth

The conventional wisdom is that vector processing is only appropriate for scientific computing applications on high end machines that are expensive in part because of their SRAM-based memory systems. By placing DRAM and logic on a single chip, VIRAM demonstrates that vector processing can also be used for application domains that demand lower cost hardware. In this section, we use a small set of benchmarks with predictable memory access patterns to explore the question of how well VIRAM supports memory-intensive applications. Our performance results are produced using a simulator that gives a cycle-accurate model of the VIRAM vector unit.

First, we look at a dense matrix-vector multiplication (MVM) benchmark as a demonstration of memory bandwidth utilization. Although MVM is a simple compilation problem, it does not perform well on conventional microprocessors due to the high memory bandwidth requirements– there are only two floating point operations per matrix element and in row-major layout, the MVM operation requires either strided memory accesses or reduction operations. In Figure 1, we show the performance in MFLOPS for 8 possible IRAM configurations, varying the number of lanes (datapath width) from 1 to 8 lanes (64 to 512 bits). In addition, we consider both a 16 MB, 8 bank DRAM and a 32 MB, 16 bank version. The VIRAM implementation is a 4-lane, 16 MB chip, while an 8-lane, 32 MB chip might be a reasonable next generation chip. One might also scale IRAM down to 2 or even a single lane to save power and area.

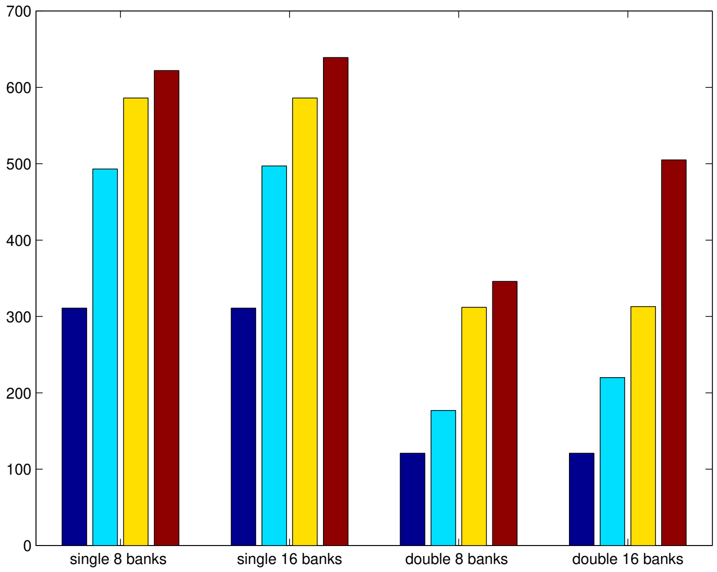

Fig. 1. Performance of Matrix-Vector Multiplication on VIRAM.

As indicated by this data, an 8-lane vector unit does not perform well with only 8 banks and even with 4 lanes there are significant advantages to having more banks. In addition, for historical reasons, our model of double precision floating point uses a multipler that is not fully pipelined, resulting in a peak performance of .4 GFLOPS rather than half of the single precision peak of 1.6 GFLOPS.

Nevertheless, the performance is impressive even for the 4-lane, 8 bank VIRAM design when compared to conventional microprocessors with higher clock rates, higher energy requirements, and more total area. For example, the hand-coded vendor implementation of MVM as reported in the LAPACK manual shows only the Compaq AlphaServer DS-20 with better performance at 372 MFLOPS; the Dec Alpha Miata (66 MFLOPS), IBM Power 3 (304 MFLOPS), Intel Pentium III

(141 MFLOPS), SGI O2K (216 MFLOPS) and Sun Enterprise 450 (267 MFLOPS) all have lower performance [ABB+99].

5 Narrow Data Types

One of the novel aspects of the IRAM ISA is the notion of variable width data which the compiler controls by setting the VPW. The compiler analyzes each vectorizable loop nest to determine the widest data width needed by vector operations in the loop and then sets the VPW to that width. The compiler uses static type information from the variable declaration rather than a more aggressive, but also more fragile, variable width analysis [SBA00].

The process of computing the VPW is done with multiple passes over a loop nest. On the first pass, the VPW is assumed to be 64 bits, and the loop nest is analyzed to determine whether it is vectorizable, and at the same time, what the widest datatype is within the loop. If the data width is determined to be narrower than 64, vectorization is re-run on the loop nest with the given estimate of VPW. The reason for the second pass is that narrowing the data width increases the maximum available vector length, and therefore allows more loops to be executed as a single sequence of vector instructions without the strip-mining loop overhead.

The above strategy works well for 32-bit datatypes (either integer or floating point) but does not work well for 16 bits in standard C. The reason is that the C language semantics require that, even if variables are declared as 8-bit characters or 16-bit shorts, most operations must be performed as if they were in 32 bits. To enforce these semantics, the compiler front-end introduces type casts rather aggressively, making it difficult to recover information about narrower types in code generation. Our initial experiments found that only the most trivial of loops, such as an array initialize, would run at a width narrower than 32 bits. As a result, we introduced a compiler flag to force the width to 16 bits; the compiler will detect certain obvious errors such as the use of floating point in 16 bit mode, but trusts the programmer for the bit-width of integer operations.

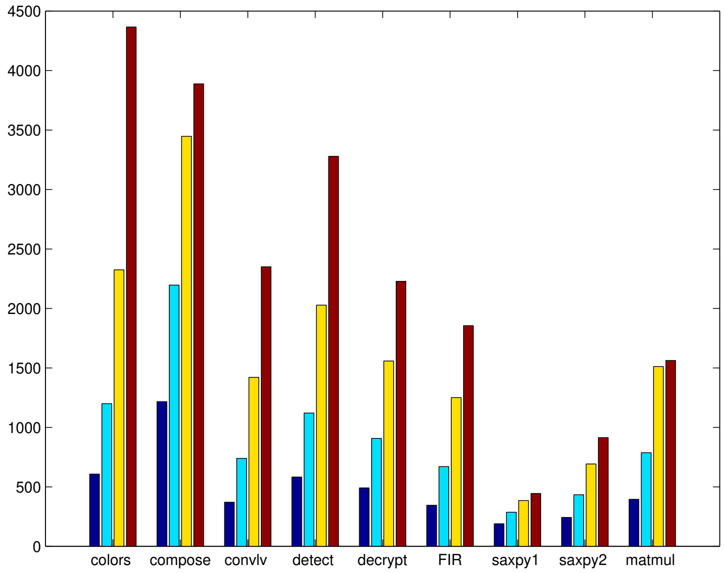

Fig. 2. Performance of Media Kernels on VIRAM. The Y axis is in MOPS (or MFLOPS where appropriate). Each cluster shows the performance on 1, 2, 4, and 8 lanes for a given benchmark.

Figure 2 shows the performance of several media processing kernels on a VIRAM configuration with 16 memory banks and varying numbers of lanes. The first 5 benchmarks are integer benchmarks taken from the UTDSP benchmark suite, and the last 4 are single precision floating point kernels.

- `colors` (colorspace conversion) is a simple image processing kernels. They read and write 8-bit data and operate internally on it as 16-bit data. Strided memory operations are convenient for selecting pixels of the same color that are interleaved in RGB format in the image. Loads of 8 bit data while the VPW is 16 loads each 8-bit value into a single vector element, without any shuffling or expanding of data.

- `compose` (image composition) is similar to `colors` in the data types, but can be performed using unit stride memory accesses.
- `convlv` is an image convolution, which like `colorspace` performs a 16-bit integer computation on images stored in RGB format using strided memor accesses. Each result pixel is computed by multiplying and summing the pixels in a 3x3 region. The source code contains 4 nested loops, and it is critical that the compiler vectorize an outer loop to obtain reasonable vector lengths. By default, the compiler merges the middle two loops and vectorizes them while unrolling the innermost one. The performance shown here improves on that default strategy by unrolling the two inner loops in source code, while the compiler vectorizes the 2nd loop that goes over rows of the image.
- `detect` is an edge detection algorithm for images and `decrypt` performs IDEA decryption. Both of these use a mixture of 16-bit and 32-bit integer operations.
- `FIR` is an FIR filter, `saxpy1` and `saxpy2` are, respectively, 64 element and 1024 element SAXPYs and `matmul` is a 64x64 matrix multiplication. All of these use 32-bit floating point computations.

Although hand-optimized VIRAM code is not available for all of these kernels in this form, two floating point kernels are available and are quite close in performance. The hand-coded performance numbers for a 4-lane configuration with 8 memory banks are: 720 MFLOPS for `saxpy2` and 1,580 MFLOPS for `matmul`. Note that the performance in both cases depends on the size of the input data. The two versions of SAXPY are included to show how performance improves with longer application level arrays, because they better amortize the vector pipeline delays. This shows an advantage IRAM for computations that operate on large images or matrices: does IRAM does not depend on caches, performance often increases rather than decreases with increases data size.

The performance of the 16-bit operations is limited primarily by address generation bandwidth when non-unit stride accesses are involved. VIRAM generates at most 4 addresses per cycle, while each of the two integer units can perform 16 integer operations per cycle. The convolution is further limited

6 Related Work

The most closely related compiler effort to ours is the vectorizing compiler project at the University of Toronto, although it does not target a mixed logic and DRAM design [LS98]. Compilers for the SIMD microprocessor extensions are also related. For example, Larsen and Amarsinghe present a compiler that uses *Superword Level Parallelism* (SLP) for these machines; SLP is identical to vectorization for many loops, although it may also discover vectorization across statements, which typically occurs if a loop has been manually unrolled.

Several aspects of the VIRAM ISA design make code generation simpler than with the SIMD extensions [PW96,Phi98]. First, the VIRAM instructions are independent of the datapath width, since the compiler generates instructions for the full vector length, and the hardware is responsible for breaking this into datapath size chunks. This simplifies the compilation model and avoids the need to recompile if the number of lanes changes between generations. Indeed, all of the performance numbers below use a single executable when varying the number of lanes. Second, VIRAM has powerful addressing modes such as strided and indexed loads and stores that eliminate the need for packing and unpacking. For example, if an image is stored using 8-bits per pixel per color, and the colors are interleaved in the image, then a simple array expression like `a[3*i+j]` in the source code will result in a strided vector load instruction from the compiler.

7 Conclusions

This paper demonstrates that the performance advantages of IRAM with vector processing do not require hand-optimized code, but are obtainable by extending the vector compilation model to multiple data widths. Although some programmer-control over narrow data types was required, the programming model is still easy to understand and use. VIRAM performs well and demonstrates scaling across varying numbers of lanes, which is useful for obtaining designs with lower power and area needs or for high performance designs appropriate for a future generations of chip technology. The compiler demonstrates good performance overall, and is often competitive with hand-coded benchmarks for floating pointer kernels. The availability of high memory bandwidth on-chip makes a 2 Watt VIRAM chip competitive with modern microprocessors for bandwidth-intensive computations. Moreover, the VIRAM instruction set offers an elegant compilation target, while the VIRAM implementation allows for scaling of computation and memory bandwidth across generations through the addition of vector lanes and memory banks.

Acknowledgements

The authors would like to thank the other members of the IRAM group and Corinna Lee's group at the University of Toronto who provided the UTDSP benchmarks. We are also very grateful for the support provided by the Cray, Inc. compiler group in helping us use and modify their compiler.

References

[ABB+99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorenson. *LAPACK Users' Guide: Third Edition.* SIAM, 1999.

[FPC+97] R. Fromm, S. Perissakis, N. Cardwell, C. Kozyrakis, B. McGaughy, D. Patterson, T. Anderson, and K. Yelick. The energy efficiency of IRAM architectures. In *the 24th Annual International Symposium on Computer Architecture*, pages 327–337, Denver, CO, June 1997.

[Koz99] Christoforos Kozyrakis. A media-enhanced vector architecture for embedded memory systems. Technical Report UCB//CSD-99-1059, University of California, Berkeley, July 1999.

[LS98] C.G. Lee and M.G. Stoodley. Simple vector microprocessors for multimedia applications. In *31st Annual International Symposium on Microarchitecture*, December 1998.

[PAC+97] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A case for Intelligent DRAM: IRAM. *IEEE Micro*, 17(2):34–44, April 1997.

[Phi98] M. Phillip. A second generation SIMD microprocessor architecture. In *Proceedings of the Hot Chips X Symposium*, August 1998.

[PW96] A. Peleg and U. Weiser. MMX technology extension to the intel architecture. *IEEE Micro*, 16(4):42–50, August 1996.

[SBA00] M. Stephenson, J. Babb, and S. Amarasinghe. Bitwidth analysis with application to silicon compilation. In *Proceedings of the SIGPLAN conference on Programming Language Design and Implementation*, Vancouver, British Columbia, June 2000.

[Tho00] Randi Thomas. An architectural performance study of the fast fourier transform on VIRAM. Technical Report UCB//CSD-99-1106, University of California, Berkeley, June 2000.