MediaBreeze: A Decoupled Architecture for Accelerating Multimedia Applications

Deependra Talla and Lizy K. John

Laboratory for Computer Architecture
Department of Electrical and Computer Engineering
The University of Texas, Austin, TX 78712
{deepu, ljohn}@ece.utexas.edu

Abstract

Decoupled architectures are fine-grain processors that partition the memory access and execute functions in a computer program and exploit the parallelism between the two functions. Although some concepts from the traditional decoupled access execute paradigm made its way into commercial processors, they encountered resistance in general-purpose applications because these applications are not very structured and regular. However, multimedia applications have recently become a dominant workload on desktops and workstations. Media applications are very structured and regular and lend themselves well to the decoupling concept. In this paper, we present an architecture that decouples the useful/true computations from the overhead/supporting instructions in media applications. The proposed scheme is incorporated into an out-of-order general-purpose processor enhanced with SIMD extensions. Explicit hardware support is provided to exploit instruction level parallelism in the overhead component. Performance evaluation shows that such hardware can significantly improve performance over conventional SIMD enhanced general-purpose processors. Results on nine multimedia benchmarks show that the proposed MediaBreeze architecture provides a 1.05x to 16.7x performance improvement over a 2-way out-of-order SIMD machine. On introducing slip-based data prefetching, a performance improvement up to 28x is observed.

Keywords: decoupling, access and execute mechanisms, microprocessor design, multimedia, SIMD, prefetching, general-purpose processors, hardware accelerators.

L. John is supported in part by the State of Texas Advanced Technology Program grant #403, the National Science Foundation under grants CCR-9796098 and EIA-9807112, and by Dell, Intel, Microsoft, and IBM.

1. Introduction

Decoupled access/execute (DAE) architectures split a program into memory access and execute functions. These architectures make use of an access processor to perform the data fetch ahead of demand by possibly alleviating delays due to memory latency. The access processor also performs indexing and other addressing operations. The execute processor operates on the data to produce results. The concept of access execute decoupling present in the IBM System 360/370, CDC 6600 [8], CDC7600, CRAY-1, CSPI MAP-200, SDP [6], PIPE [7], SMA [5], WM [10], DS [9], etc demonstrated the potential of decoupling memory access and computations [1][2].

A conventional decoupled architecture is illustrated in Fig. 1. Typically there are two instruction streams which may either be fetched from memory separately, or which may be split from a single instruction stream early in instruction processing [1][2]. The access and execute processors (AP and EP) communicate through architectural queues, which allow the two processors to *slip* with respect to each other. Slip is an architectural attribute that allows concurrency between the two processors, by allowing the AP to forward information to the EP, while the EP performs operations on previous information. Slip indicates the distance the access process is running ahead of the execute process.

In addition to the two processors, the system includes a number of first-in-first-out (FIFO) buffers or queues – for instance the load queue, store queue, load address queue, store address queue, branch queues (AEBQ and EABQ), and copy queues (AECQ and EACQ). Data to be used in EP computations are loaded by AP instruction into the load queue. The EP performs computations using the data and deposits any results to be stored into memory into the store queue. These addresses wait in the queue until a corresponding data item appears in the store queue.

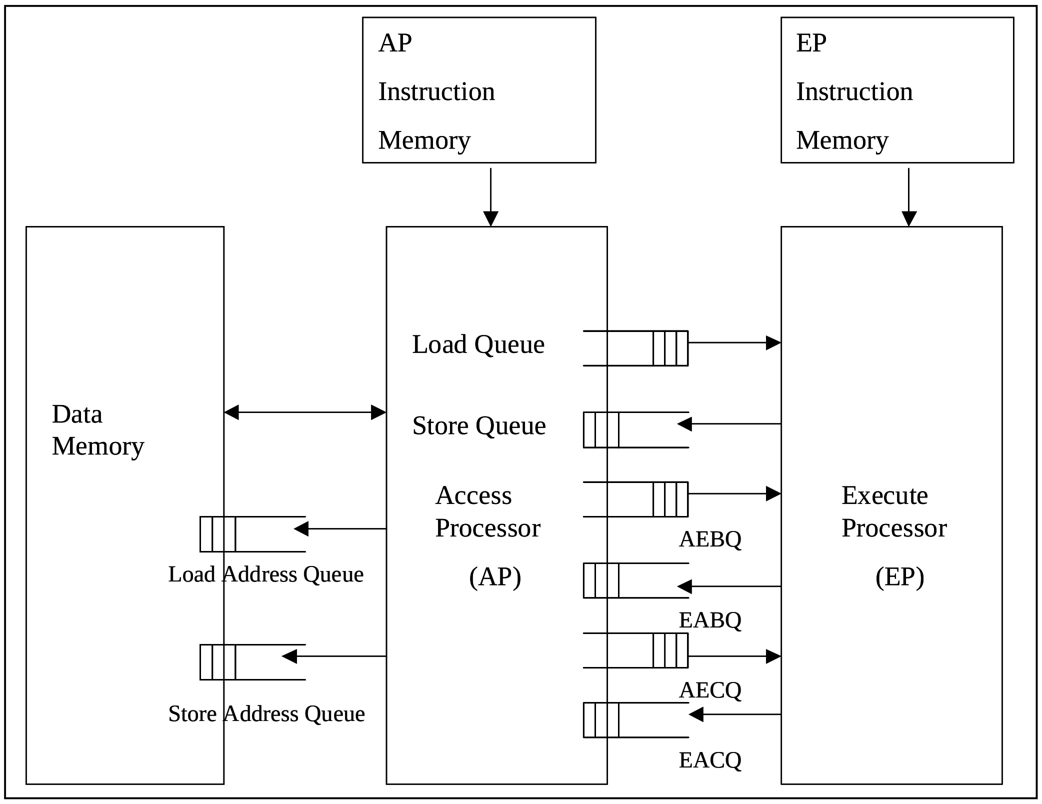

Figure 1. A Typical Decoupled Architecture

The branch queues are used to coordinate the two instruction streams. The AP and EP have their own set of conditional branch instructions. The processor that evaluates the branch condition places a token in the branch queue. The other processor iterates its operation until the token appears in the queue. The copy queue is useful if data is to be transferred between the two processors.

Some of the concepts from the traditional DAE paradigm made its way into commercial processors. However, the decoupling concept encountered resistance in general-purpose applications because these applications are not very structured and regular. Multimedia applications have recently become a dominant workload on desktops and workstations. Contemporary computer applications are multimedia-rich, involving significant amounts of audio and video compression, 2D image processing, 3D graphics, speech and character recognition, communications, and signal processing. Media applications are very structured and regular and lend themselves well to the decoupling concept.

With an increasing number of desktop/workstation execution cycles being spent on media applications, general-purpose processors (GPPs) have been enhanced with single instruction multiple data (SIMD) execution units [18]. Using assembly libraries, compiler intrinsics and limited compiler capabilities, media-rich applications have benefited from SIMD extensions.

However, there are several inefficiencies in the execution of SIMD enhanced applications, in spite of a performance improvement over non-SIMD code. Our experiments on GPPs with SIMD extensions on media applications show that only 1% to 12% of the peak available SIMD units' throughput is achieved [14]. Instructions required for preprocessing of data to feed the SIMD units contribute to the under-utilization of the SIMD execution units. The instructions include address generation, address transformation (permute, data packing and unpacking), loads/stores, and loop branches. On conceptually classifying the instruction stream into either useful computations (core/true computations required by the media algorithm) or supporting/overhead (all other instructions used for assisting the useful computation instructions), 75-85% of the dynamic instructions are found to be supporting/overhead related instructions and only 15-25% are useful computation instructions.

Increasing the width/number of SIMD execution unit/s results in diminishing returns in performance improvement since parallelism is exploited mainly in the useful computation instructions. Efficient execution of media workloads mandates exploiting parallelism in the supporting/overhead related instructions rather than concentrating solely on the useful computation instructions. Not only do media applications have data level parallelism (DLP) in the useful computations, but also plenty of available instruction level parallelism (ILP) in the supporting/overhead related instructions.

In this paper, we present the MediaBreeze architecture that decouples the media program execution into two parts, namely, the useful computations and overhead/supporting instructions. We provide explicit hardware support for processing the supporting/overhead related instructions. The proposed scheme works in an integrated manner (incorporated into a GPP with SIMD extensions) in which media workloads can excel without sacrificing performance on general-purpose workloads. The rest of the paper is organized as follows. Section 2 provides a brief overview of the MediaBreeze architecture. Section 3 presents the performance benefits of the MediaBreeze architecture. Section 4 summarizes the paper.

2. The MediaBreeze Architecture

Analyzing media instruction execution streams results in two distinct sets of operations: the useful (true) computations as required by the algorithm and the supporting computations (overhead) such as address generation, address transformation (packing, unpacking, and permute), loads/stores, and loop branches. The sole purpose of overhead instructions is to aid in the execution of the useful computation instructions. The execution of overhead instructions is mandatory due to the programming conventions of general-purpose processors, abstractions and control flow structures used in programming and a mismatch between how data is used in computations versus how data is stored.

The MediaBreeze architecture was designed to accelerate SIMD enhanced media applications by decoupling the useful computation and overhead instructions, and process the overhead related instructions efficiently. In traditional DAE processors, overhead related instructions included memory access (loads/stores) and addressing arithmetic. In our classification, we include loop branches and SIMD specific data reorganization (permute, pack, and unpack) [17] in the overhead related instructions. The useful computations are the SIMD arithmetic and logical computations necessary for the media algorithm. Fig. 2 illustrates the block diagram of the MediaBreeze architecture. In order to perform the SIMD operations, the MediaBreeze architecture uses existing hardware units (that is, such hardware is already present in current state-of-the-art SIMD enhanced GPPs) as well as introduces new hardware units. The existing hardware units are programmed differently than by the conventional control path (explained later in this section). The existing hardware units (lightly shaded blocks in Fig. 2) are the load/store units, SIMD computation unit, data reorganization/address transformation, and the data station. The new hardware units (darkly shaded blocks in Fig. 2) are the address generation units, hardware looping, Breeze instruction memory, and Breeze instruction decoder. The functionality of the existing hardware units is:

- Load/store units: Loading/storing data from/to memory and to/from SIMD registers.
- SIMD computation unit: All arithmetic and logical SIMD computations along with multiplication and special media operations such as sum-of-absolute-differences are executed in this unit. Current GPPs typically have two SIMD ALUs and one SIMD multiplier in their SIMD datapath.
- Data Reorganization: SIMD processing mandates several data reorganization mechanisms such as packing, unpacking, and permute [17]. Reduction operations, scaling, and shifting of the results are also required for SIMD processing. Current commodity SIMD enhanced GPPs have data reorganization hardware in their SIMD datapath.
- Data station: The data station acts as a register file for the SIMD computation. Current SIMD enhanced GPPs either have dedicated SIMD register files or share the floating-point register file for intermediate SIMD results.

The functionality of the newly added hardware units is:

- Address generation: Address arithmetic functions are moved from the execution unit subsystem in current processors to a MediaBreeze hardware unit where dedicated address arithmetic hardware would generate all input and output address data structures/streams concurrent with the SIMD computation unit. Such a mechanism of providing dedicated address arithmetic hardware would reduce overhead as address calculations are performed explicitly by ALUs in current ILP processors. This involves some combination of extra instructions, parts of instructions, registers, memory accesses, and computation time.
- Looping: Using dedicated hardware looping (zero-overhead branch processing), branch instructions related to loop increments are eliminated.
- Breeze instruction memory and decoder: In order to program/control the hardware units (both existing and newly added units) in the MediaBreeze architecture, a special instruction called the Breeze instruction is formulated that contains all the overhead related instructions along with the useful computations. The Breeze instruction memory stores these instructions when they enter the processor.

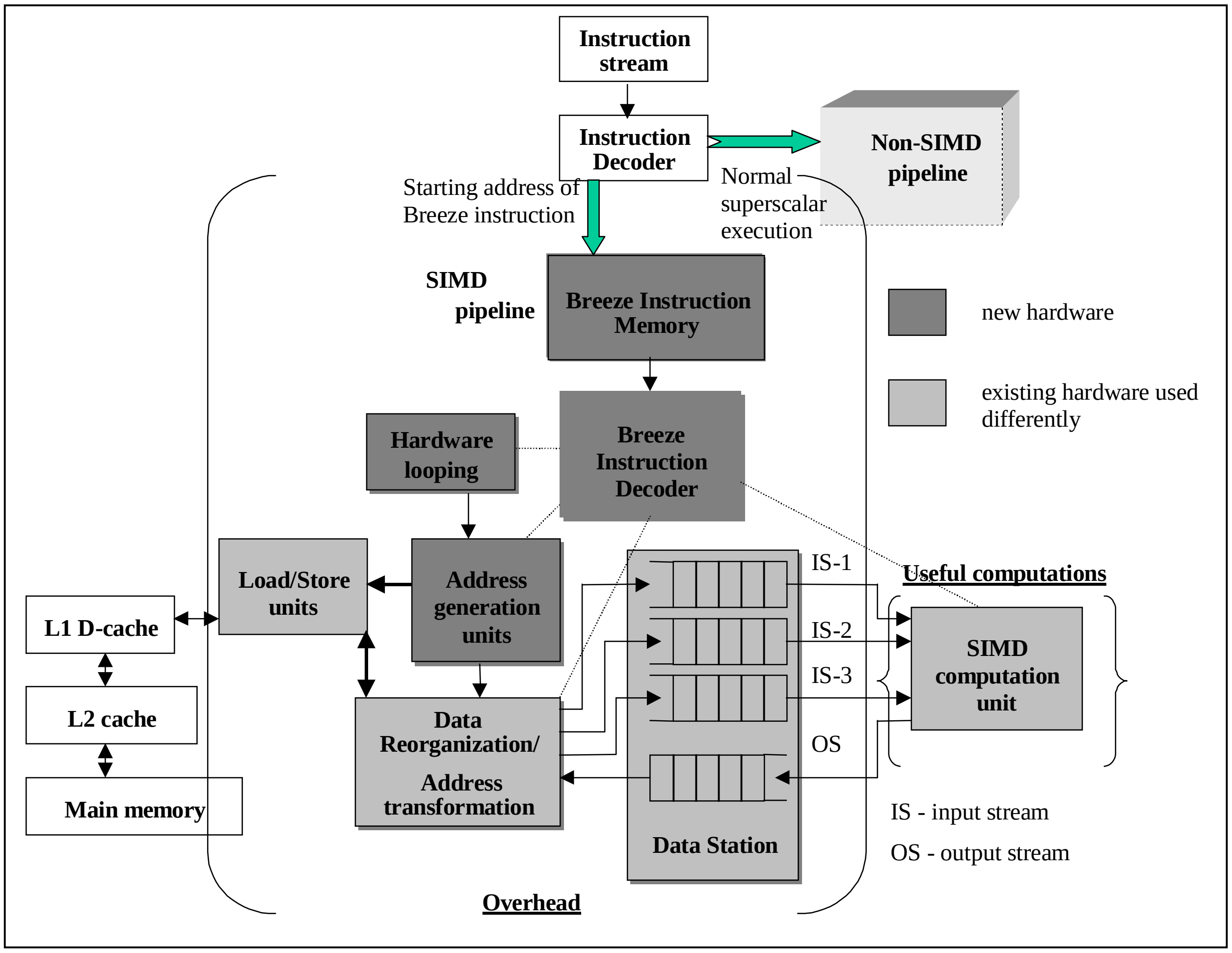

Figure. 2. The MediaBreeze Architecture

SIMD instructions reduce the dynamic instruction count because they operate on multiple data in a single instruction. Due to the repetitive operations required by media applications such a technique reduces the total number of instruction fetches and decodes. However, SIMD instructions capture only the useful computation operations. Encoding all the overhead/supporting operations along with the SIMD useful computation instructions has the advantage that the Breeze instruction can potentially replace millions of dynamic RISC instructions that have to be fetched, decoded, and issued every cycle in a normal superscalar processor.

3. Results

To measure the impact of the MediaBreeze architecture, we modified the PISA version of Simplescalar-3.0 (sim-outorder) [13] to simulate SIMD instructions and Breeze instructions using instruction annotations. We use two 64-bit SIMD ALUs and one 64-bit SIMD multiplier in our processor configuration. The memory system for the MediaBreeze architecture is modified to allow for cache miss stalls and memory conflicts (i.e., the SIMD pipeline stalls in the event of a cache miss). We incorporate the MediaBreeze hardware into a 2-way and 4-way SIMD GPP. We evaluated the performance of the MediaBreeze architecture using nine multimedia benchmarks. Our benchmarks include kernels and applications. The kernels are *cfa* – color filter interpolation, *dct* – discrete cosine transform, *motest* – motion estimation, and *scale* – linear image scaling. The applications are *g711* – G711 speech coding, *aud* – audio effects, *jpeg* – JPEG image compression, *ijpeg* – JPEG image decompression, and *decrypt* – IDEA decryption. Figure 3 shows the speedup obtained for nine multimedia benchmarks using the MediaBreeze architecture with a 2-way SIMD processor as the baseline.

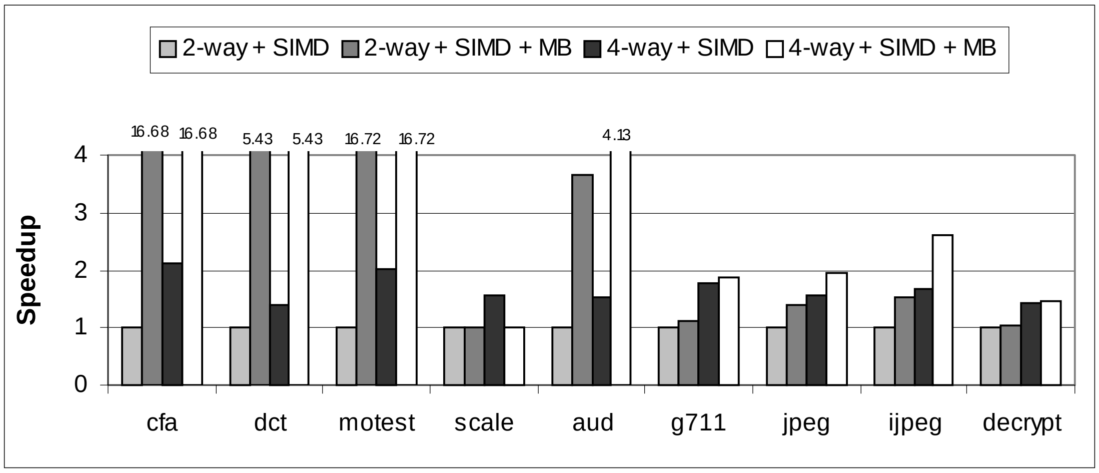

Figure. 3. Performance of the MediaBreeze (MB) architecture versus SIMD

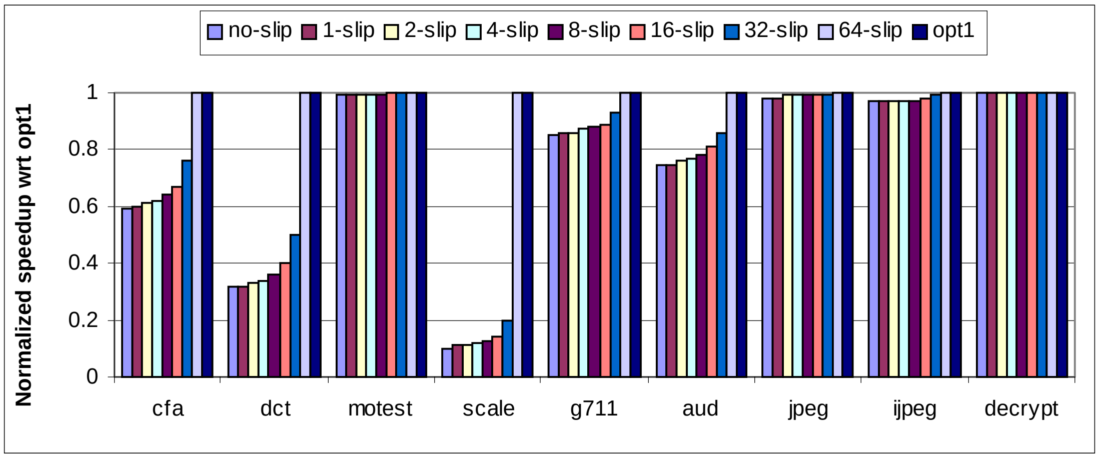

Figure. 4. Impact of prefetching on the MediaBreeze architecture. 'opt1' symbolizes the MediaBreeze with perfect memory

The speedup of the 2-way MediaBreeze architecture over a 2-way SIMD processor ranges from 1.05x to over 16x. The speedup of the 2-way and 4-way MediaBreeze enhanced architecture is the same for kernels because both the 2-way and 4-way configurations contain the same number of SIMD execution units (2 SIMD ALUs and 1 SIMD multiplier) and the Breeze instruction is able to capture the complete kernels with no other superscalar instructions necessary. In spite of the speedup, the MediaBreeze architecture is susceptible to memory latencies because the SIMD pipeline operates in-order. To reduce the impact of memory latencies on the MediaBreeze architecture, we introduced a prefetch engine to load future referenced data into the L1 cache. The prefetch engine loads data that is going to be used in the SIMD computation unit since the access pattern of the media algorithm is known in advance. The regularity of media access patterns prevents the risk of superfluous fetch commonly encountered in many prefetching environments. The prefetch engine slips ahead of the loads for computation and the computation itself to gather data into the L1 cache. The slip can be varied from 0 (no prefetch) to the main memory latency. Fig. 4 shows the performance obtained by varying the slip for each of the benchmarks on a 4-way SIMD GPP enhanced with the MediaBreeze hardware.

Increasing the slip from 0 to 32 gives a steady but not enough improvement for some benchmarks when compared to a slip of 64. Prefetching with a slip of 64 achieves opt1 performance in all cases. The main memory latency was 65 cycles in our simulation. A slip of 64 translates into a 1-cycle latency for each load. If the prefetched data were evicted from the cache before it is used for computation, maximum benefit cannot be realized. However, we did not encounter such a situation. The speedup of a 2-way MediaBreeze enhanced processor with slip-based prefetching is observed to be up to 28x over a 2-way SIMD GPP.

We investigated the cost of incorporating the MediaBreeze hardware into a state-of-the-art general-purpose superscalar processor with SIMD extensions in [19]. Our results indicate that the newly added MediaBreeze hardware requires less than 10% of the SIMD execution units' chip area and 0.3% overall chip area, and consumes less than 1% of the total processor power. This is achieved without elongating the critical path of the processor.

4. Summary and Conclusion

Multimedia applications lend themselves well to the decoupling concept because they are very regular and structured. In this paper, we presented the MediaBreeze architecture that exploits some of the key concepts of DAE processors along with dedicated hardware to accelerate SIMD enhanced multimedia applications on GPPs. The MediaBreeze architecture incorporates mechanisms to improve the balance between the useful/true SIMD computation instructions and the overhead/supporting instructions in order to exploit high concurrency and achieve high performance. Results on nine multimedia benchmarks show that the MediaBreeze architecture provides a 1.05x to 16.7x performance improvement over a 2-way out-of-order SIMD machine. On introducing slip-based data prefetching, a performance improvement up to 28x is observed.

References

[1] J. E. Smith, "Decoupled access/execute computer architectures," *ACM Trans. on Computer Systems*, vol. 2, no. 4, pp.289-308, Nov. 1984.

[2] J. E. Smith, S. Weiss, and N. Y. Pang, "A simulation study of decoupled architecture computers," *IEEE Trans. on Computers*, vol. C-35, No. 8, pp. 692-701, Aug. 1986.

[3] L. Kurian, "Issues in the design of a decoupled architecture for a RISC environment," *Ph.D. thesis*, The Pennsylvania State University, Aug. 1993.

[4] H. G. Cragon, and W. J. Watson, "The TI advanced scientific computer." *IEEE Computer Magazine*, pp. 55-64, Jan. 1989.

[5] A. R. Pleszkun and E. S. Davidson, "Structured memory access architecture," *Proc. IEEE. Int. Conf. on Parallel Processing*, pp. 461-471, 1983.

[6] R. R. Shively, "Architecture of a programmable digital signal processor," *IEEE Trans. on Computers*, vol. C-31, pp. 16-22, Jan. 1978.

[7] J. R. Goodman, T. J, Hsieh, K. Liou, A. R. Pleszkun, P. B. Schechter, and H. C. Young, "PIPE: A VLSI decoupled architecture," *Proc. IEEE Sym. on Computer Architecture*, pp. 20-27, Jun. 1985.

[8] J. E. Thornton, "Parallel operation in the Control Data 6600," *Fall Joint Computers Conference*, vol. 26, pp. 33-40, 1961.

[9] Y. Zhang, and G. B. Adams, "Performance modeling and code partitioning for the DS architecture," *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 293-304, Jun. 1998.

[10] Wm. A. wolf, "Evaluation of the WM architecture," *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 382-390, May 1992.

[11] D. J. Kuck, and R. A. Stokes, "The Burroughs scientific processor (BSP)," *IEEE Trans. on Computers*, C-31 (5), pp. 363-376, 1982.

[12] P. Ranganathan, S. Adve, and N. Jouppi, "Performance of image and video processing with general-purpose processors and media ISA extensions," *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 124-135, May 1999.

[13] D. Burger, and T. M. Austin, "The SimpleScalar tool set," Version 2.0. *Technical Report 1342*, Univ. of Wisconsin-Madison, Comp. Sci. Dept, 1997.

[14] D. Talla, "Architectural techniques to accelerate multimedia applications on general-purpose processors," *Ph.D. Thesis*, The University of Texas at Austin, Aug. 2001. Available: http://www.ece.utexas.edu/~deepu/phd_thesis.pdf

[15] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals: Architectures and Features*, Chapter 8, IEEE Press series on Signal Processing, ISBN 0-7803-3405-1, 1997.

[16] S. A. Mckee, "Maximizing memory bandwidth for streamed computations," *Ph.D. Thesis*, School of Engineering and Applied Science, University of Virginia, May 1995.

[17] J. Corbal, R. Espasa, and M. Valero, "On the efficiency of reductions in micro-SIMD media extensions," *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques*, Sep. 2001.

[18] R. B. Lee, "Multimedia extensions for general-purpose processors," *Proc. IEEE Workshop on Signal Processing Systems*, pp. 9-23, Nov. 1997.

[19] D. Talla and L. K. John, "Cost-effective hardware acceleration of multimedia applications," Proc. Int. Conf. on Computer Design, pp. 415-424, Sep. 2001.