Cost-effective Hardware Acceleration of 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

General-purpose microprocessors augmented with SIMD execution units enhance multimedia applications by exploiting data level parallelism. However, supporting/overhead related instructions (instructions necessary to feed the SIMD execution units such as address generation, packing/unpacking, permute, loads/stores, and loop branches) dominate media instruction streams accounting for 75-85% of the dynamic instructions. This leads to an under-utilization of SIMD execution units resulting in a throughput that ranges between 1-12% of the peak throughput. We accelerate multimedia applications by providing explicit hardware support to eliminate or reduce the impact of the supporting/overhead related instructions. Performance evaluation shows that such hardware can significantly improve performance over conventional SIMD enhanced general-purpose processors (1.05x to 28x). In this paper, we investigate the cost of incorporating hardware, for efficient execution of supporting/overhead related instructions, into a high-speed SIMD enhanced general-purpose processor and perform area, power, and timing tradeoffs. Our results indicate that – the added hardware requires less than 10% SIMD execution units' chip area and 0.3% overall chip area, and power consumption is less than 1% of the total processor power. This is achieved without elongating the critical path of the processor.

Keywords: microprocessor design, multimedia, SIMD, general-purpose processors, hardware accelerators, area, power, and timing tradeoffs.

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

Multimedia workloads are characterized by structured and regular computations on large data sets with small element-widths. 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 [1]. Four, eight or sixteen data elements of 32-, 16-, or 8-bits width (subwords) can be operated simultaneously in a single register (128-bit wide). Using assembly libraries, compiler intrinsics and limited compiler capabilities, media-rich applications have benefited from SIMD extensions [2][3][4].

However, recent work [5] on characterizing media workloads exposes several inefficiencies in the execution of SIMD enhanced applications, in spite of a performance improvement over non-SIMD code. Experiments on general-purpose processors with SIMD extensions on media applications show that only 1% to 12% of the peak available SIMD units' throughput is achieved. 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 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. Providing explicit hardware support, for processing the supporting/overhead related instructions, significantly accelerates multimedia applications. Analysis of media applications unveils invocation of several address patterns, often multiple simultaneous sequences. Simple hardware can be devised to generate such patterns, but using general-purpose RISC instruction sets is not efficient, as the available addressing modes are limited. Likewise keeping track of multiple array bounds involves a combination of several addressing modes and instructions leading to considerable address generation overhead in conventional general-purpose RISC processors. Similarly hardware to implement multiple nested loops is not overly complicated, however supporting multiple loops with general-purpose instruction sets may not be very efficient.

Based on the above philosophy, the MediaBreeze architecture was designed in [5] to augment a SIMD enhanced GPP with simple hardware for providing an effective and seamlessly integrated solution for media workloads without sacrificing the performance on general-purpose workloads. Results on nine multimedia benchmarks show that adding simple hardware to a general-purpose core with SIMD extensions exhibits a 1.05x to 28x-performance improvement. In addition, the total number of dynamic instructions is reduced by 0.2% to over 99% across the benchmark suite.

The primary contribution of this paper is to investigate the associated cost of adding the hardware to a high-speed GPP with SIMD extensions. Using a cell-based methodology targeting different ASIC libraries (optimized for one or more of performance, density and low-power) and technologies (0.18-, 0.25-, and 0.35-micron), we attempt to answer the following three questions:

- What is the chip area increase due to adding the hardware to assist in processing the supporting/overhead related instructions?
- How much power is consumed by the additional logic?
- Does the added hardware elongate the critical path of a high-speed processor or in other words, will it be a bottleneck in determining the maximum clock rate? How many pipeline stages does the hardware add to the processor core?

We find that the above mentioned 1.05x to 28x performance improvement over MMX- & SSE-like extensions is obtained at an area cost of less than 10% of SIMD execution unit's area (such as MMX and SSE) and a 0.3% increase in overall chip area. The power consumption is less than 1% of the total processor power. The timing results show that the added hardware does not elongate the critical path of the GPP core.

The rest of the paper is organized as follows. Section 2 provides a brief overview of the MediaBreeze architecture. Section 3 discusses about the tools and methodology. Section 4 presents the implementation details of the added hardware. Section 5 evaluates the costs associated with implementing each mechanism in the MediaBreeze architecture and their applicability to mainstream commodity general-purpose microprocessor pipelines. Finally, Section 6 summarizes the paper.

2. Description of the MediaBreeze Architecture

Analyzing media instruction execution streams results in two distinct sets of operations: the useful 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. Due to the dominance of overhead instructions (75-85%), they need to eliminated, alleviated, or overlapped with the useful computations for better performance.

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. Fig. 1 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. 1) 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. 1) 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, permute, etc. 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.

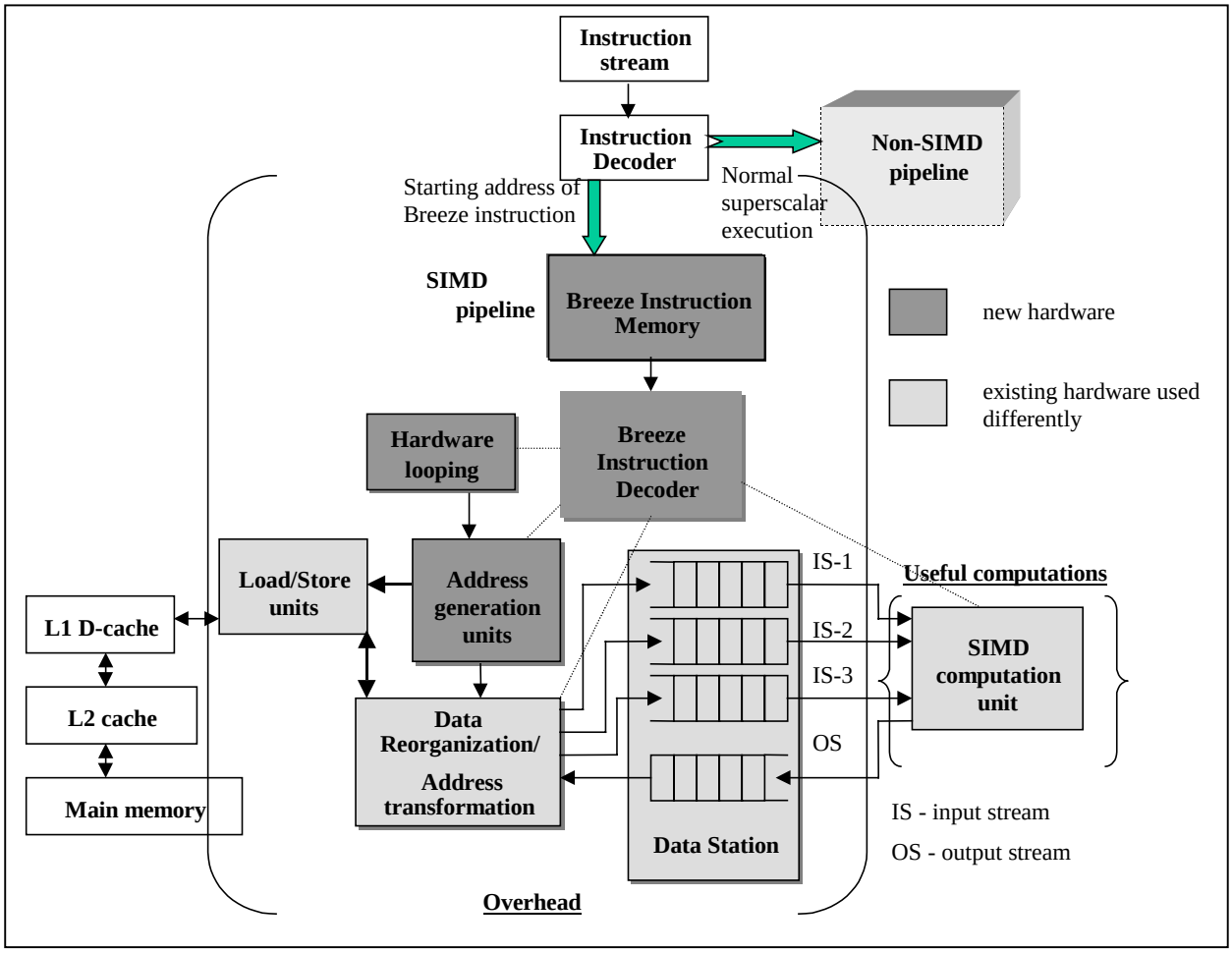

Figure. 1. The MediaBreeze Architecture

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. The Breeze instruction memory stores these instructions once they enter the processor. Fig. 2 illustrates the structure of the Breeze instruction.

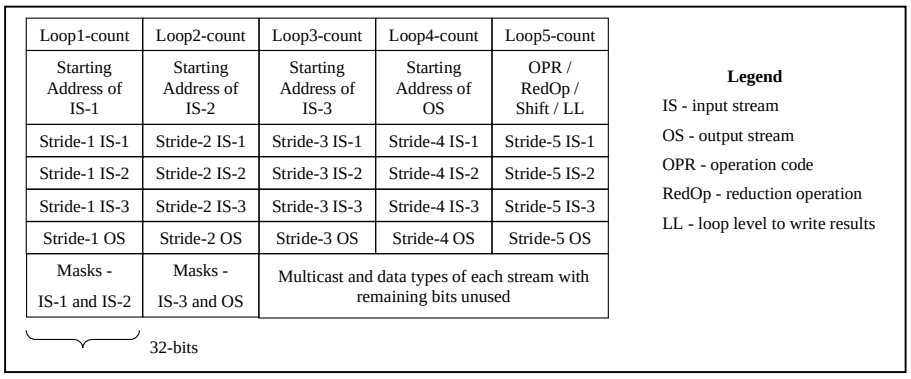

Figure. 2. Structure of the Breeze Instruction

Five loop index counts (bounds) are indicated in the Breeze instruction to support five level nested loops (in hardware). A five-level deep nest can accommodate most algorithms. The MediaBreeze architecture allows for three input data structures/streams and produces one output structure. This was chosen because some media algorithms can benefit from this capability (current SIMD execution units sometimes operate on 3 input registers to produce one output value). Each data structure/stream has its own dedicated address generation unit to compute the address every clock cycle with the base address specified in the Breeze instruction. Due to the sub-block access pattern in media programs, data is accessed with different strides at various points in the algorithm. The Breeze instruction facilitates multiple strides (one at each level of loop nesting, i.e., a total of five strides) for each of the three input streams and one output stream. The strides indicate address increment/decrement values based on the loop-nest level. Depending on the mask values for each stream (indicated in the Breeze instruction) and the loop-nest level, one of the five possible strides is used to update the address pointer.

Data types of each stream/structure are also indicated in the Breeze Instruction. Depending on the size of each element in the data structures, a different amount of SIMD parallelism is achieved. If one data stream is 8-bit data (16-way parallelism for a 128-bit wide execution unit) and the other is 16-bit data (8-way parallelism), the SIMD processing achieves only 8-way parallelism. The maximum achievable SIMD parallelism is the minimum of all the data structures (all commercial SIMD extensions have this limitation). Current SIMD extensions provide data reorganization instructions for solving the problem of having different element sizes across the data structures (packing, unpacking, permute, etc) and introduce additional instruction overhead. By providing this information in the Breeze Instruction, special hardware in the MediaBreeze will perform this function. The MediaBreeze performs reduction operations and this is also indicated in the Breeze Instruction (for example, multiple independent results in a single SIMD register are combined together in dot product which require additional instructions in current DLP techniques). Support for signed/unsigned arithmetic, saturation, shifting/scaling of final results is also indicated in the Breeze Instruction. This eliminates additional instructions that are otherwise needed for conventional RISC processors.

With the support for multiple levels of looping and multiple strides, the Breeze Instruction is a complex instruction and decoding such an instruction is a complex process in current RISC processors. MediaBreeze instead handles the task of interpreting/decoding of the Breeze Instruction. MediaBreeze has its own instruction memory to hold a Breeze instruction. Two additional 32-bit instructions are also added to the ISA of the general-purpose processor for starting and interrupting the MediaBreeze. These 32-bit instructions (fetched and decoded by the traditional instruction issue logic) contain the length of the Breeze Instruction. Whenever a Breeze instruction is encountered in the dynamic instruction stream, the dynamic instructions prior to the Breeze instruction are allowed to finish after which the MediaBreeze instruction interpreter decodes the Breeze instruction. In our current implementation, we halt the superscalar pipeline until the execution of the Breeze instruction is completed because MediaBreeze reuses existing hardware units. Otherwise, arbitration of resources is necessary to allow for overlap of the Breeze instruction and other superscalar instructions.

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. This results in giving the MediaBreeze architecture advantages similar to ASIC-based acceleration.

It is possible that an exception or interrupt occurs while a Breeze instruction is in progress. The state of all the five loops, their current counts, loop bounds, etc are saved and restored when the instruction returns. This is similar to handling of exceptions during move instructions with REP (Repeat Prefix) in x86. MediaBreeze has registers to hold the loop parameters for all the loops. Code development for the MediaBreeze architecture is currently done by hand. Similar to developing code for SIMD extensions, compiler intrinsics may have to be employed to utilize the MediaBreeze architecture. We do not want to underestimate the challenge of compiling for the MediaBreeze architecture; however, the effort is comparable to that of compiling for SIMD extensions.

We evaluated the performance of the MediaBreeze architecture using nine multimedia benchmarks and found that the performance improvement of the MediaBreeze architecture ranges from 1.05x to 28x over a 4-way out-of-order processor with SIMD extensions. Fig. 3 presents the results of incorporating the MediaBreeze hardware into a 2-way and 4-way out-of-order processor with SIMD extensions.

In summary, the MediaBreeze architecture performs media program execution efficiently by exploiting significant amounts of ILP in the overhead component. In a scenario where all the loop nests and data streams are processed, MediaBreeze executes (in hardware) the following equivalent number of software instructions (in conventional ILP processors) during each cycle - five branches, three loads, one store, four address value generation (one on each stream), one SIMD operation (2-way to 16-way parallelism depending on each data element size), one accumulation of SIMD result, one SIMD reduction operation, four SIMD data reorganization (pack/unpack, permute, etc) operations, and shifting & saturation of SIMD results.

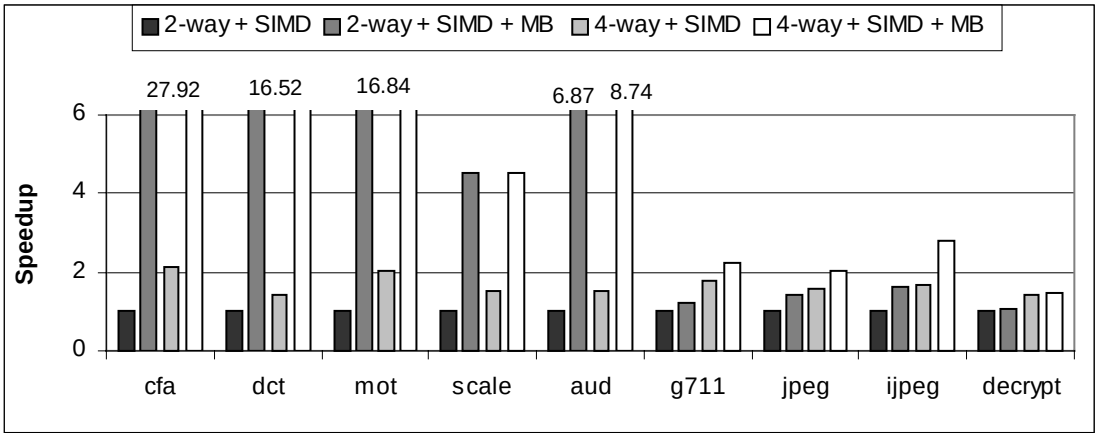

Figure. 3. Performance/Speedup of MediaBreeze (MB) versus SIMD

3. Implementation Methodology

To estimate the area, power, and timing requirements of the MediaBreeze architecture, we developed VHDL models for the various components. Using Synopsys synthesis tools [9], we used a cell-based methodology to target the VHDL models to several ASIC cell-libraries from LSI Logic [10][12]. Table 1 lists the libraries and technologies used for evaluating the implementation costs.

The Synopsys synthesis tools estimate area, power, and timing of circuits based on the ASIC technology library. The ASIC technology library provides four kinds of information.

- Structural information. This describes each cell's connectivity to the outside world, including cell, bus, and pin descriptions.
- Functional information. This describes the logical function of every output pin of every cell so that the

synthesis tool can map the logic of a design to the actual ASIC technology.

- Timing information. This describes the parameters for pin-to-pin timing relationships and delay calculation for each cell in the library.
- Environmental information. This describes the manufacturing process, operating temperature, supply voltage variations, and design layout. The design layout includes wire load models that estimate the effect of wire length on design performance. Wire load modeling estimates the effect of wire length and fanout on resistance, capacitance, and area of nets.

We use the default wire load models provided by LSI Logic's cell-based libraries. The Synopsys synthesis tools compute timing information based on the cells in the design and their corresponding parameters defined in the ASIC technology library. The area information provided by the synthesis tools is prior to layout and is computed based on the wire load models of the associated cells in the design. Average power consumption is measured based on the switching activity of the nets in the design. In our experiments, the switching activity factor originates from the RTL models (the tool gathers this information from simulation).

The area, power, and timing estimates are obtained after performing maximum optimizations for performance in the synthesis tools. High-speed commodity GPPs are implemented using custom circuit design techniques. The results obtained in this paper reflect a worst-case estimate because customized placement and custom circuit design techniques give better results than automated placement by CAD tools [12]. The interested reader is referred to [9] for further information regarding the capabilities and limitations of the synthesis tools.

Table 1. List of Cell-based Libraries used in synthesis (LSI Logic)

Library name	Description
lcbg12-p (G12-p)	A 0.18-micron L-drawn (0.13-micron L-effective) CMOS process. Highest performance solution at 1.8 V with high drive cells optimized for long interconnects associated with large designs.
lcbg12-d (G12-d)	A 0.18-micron L-drawn (0.13-micron L-effective) CMOS process. Optimized for performance, density, and power for most applications at 1.8 V.
lcbg12-l (G12-l)	A 0.18-micron L-drawn (0.13-micron L-effective) CMOS process. Ultra-low power and high-density solution with a low dynamic and standby leakage current at 1.8 V.
lcbg11-p (G11-p)	A 0.25-micron L-drawn (0.18-micron L-effective) CMOS process. Highest performance solution at 2.5 V.
lcbg11-v (G11-v)	A 0.25-micron L-drawn (0.18-micron L-effective) CMOS process. Ultra-low power and cost sensitive solution at 1.8 V.
lcbg10-p (G10-p)	A 0.35-micron L-drawn (0.25-micron L-effective) CMOS process. Optimized for high performance at 3.3 V.

4. Hardware Implementation of the MediaBreeze Units

4.1 Address generation

The MediaBreeze architecture supports three input and one output data structures/streams. Each of the four data streams has a dedicated address generation hardware unit. Address arithmetic on each stream is performed based on the strides and mask values indicated in the Breeze instruction. For each clock cycle, depending on the mask bits and loop index counts, one of the five possible strides is selected. The new address value is then computed based on the selected stride and the previous address value. Fig. 4 depicts the block diagram of the address generation circuitry for a single data stream/ structure.

The *last_val comparators* determine which of the four inner level loop counters have reached their upper bound. The outermost loop comparison is not necessary because the Breeze instruction finishes execution at the instant when the outermost loop counter reaches its upper bound. The *inc-cond* and *inc-combine* blocks generate

flag signals based on the output from the *last_val comparators* and mask values from the Breeze instruction. If none of the *flag* signals are true, then *stride-5* is used to update the *prev-address*, otherwise the appropriate *stride-(1–4)* is selected depending on *flag- (1–4).* The *address-generate* block uses a 32-bit adder to add the selected stride to the previous address. On either an exception or a stall, only the *prev-address* value needs to be stored as the loop counters are stored by the hardware looping circuitry. For each of the four data structures/streams, the *last_val comparators* portion of the logic is shared, but the remaining hardware needs to be replicated.

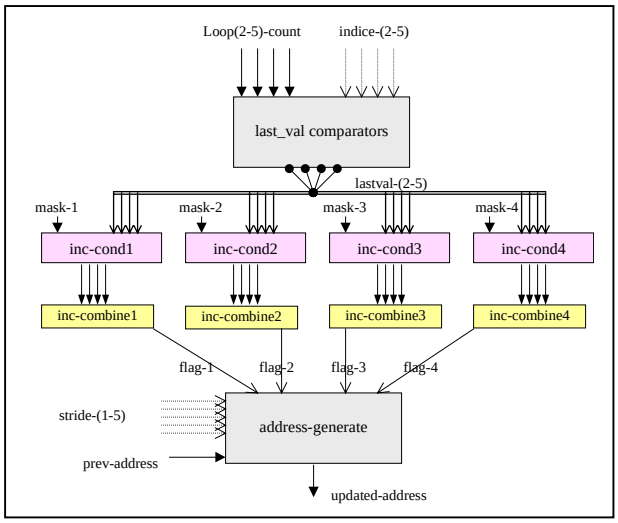

Figure. 4. Block diagram of address generation (per stream)

4.2 Looping

The MediaBreeze architecture incorporates five levels of loop nesting in hardware to eliminate branch instruction overhead for loop increments. A similar mechanism was commercially implemented in the TI ASC [7] (two levels of do-loop nesting in addition to a self-increment loop). Conventional DSP processors such as the TMS320C5x from TI also use such a technique for one or more levels of loop nesting [8]. Fig. 5 shows the block diagram of the looping hardware.

Loop index values are produced every clock cycle based on the loop bound for each level of nesting (bounds for each of the five loops are specified in the Breeze instruction). The value of a loop index varies from 1 (lower bound) to the corresponding loop bound (upper bound), and resets to its lower bound once the upper bound is reached in the previous cycle. The execution of the Breeze instruction ends when the outermost loop (loop1 in Fig. 5) reaches its upper bound. On encountering either an exception or a stall, the loop indices are stored and the increment logic is halted; the counting process is started once the exception/stall is serviced. Each of the five *comparators* (32-bit wide) operates in parallel to generate *flag* (1-bit wide) signals that are *priority encoded* to determine which one of the five loop counters to increment. When a loop counter is *incremented-by-1* (circuit for incrementing a 32-bit value by 1), all the loop counters belonging to its inner level are reset (for example, if loop3 is *incremented-by-1*, then loop4 and loop5 are reset to their lower bound).

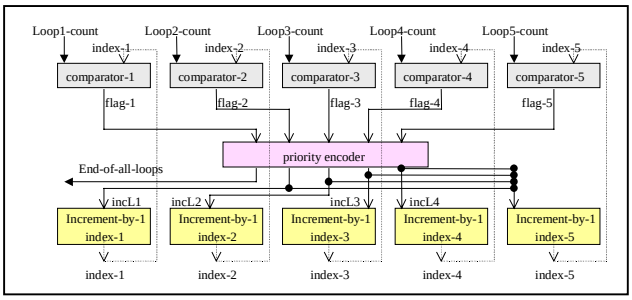

Figure. 5. Block diagram of the five hardware loops

4.3 Breeze instruction decoder

A stand-along instruction decoder for the Breeze instructions eliminates the need to modify the conventional instruction decoder of current GPPs. A Breeze instruction needs to be decoded only once since various control parameters are stored in hardware registers after the decoding process. The implementation of the Breeze instruction decoder was merged into the address generation and looping circuitry.

4.4 Breeze instruction memory

The Breeze instruction memory stores the Breeze instruction once it enters the processor. We do not estimate the cost of this storage because the ASIC libraries we use are not targeted for memory cells. However, the area, power, and timing estimates of the Breeze instruction memory are similar to an SRAM structure. The size of a Breeze instruction is typically 120 bytes.

4.5 Existing hardware units

The remaining hardware units that are required for the operation of the MediaBreeze architecture are the SIMD computation unit, data reorganization, load/store units, and data station. These hardware units are already present in commodity SIMD GPPs. However, the Breeze

instruction decoder controls the operation of these units as opposed to the conventional control path. This mandates an extra multiplexer to differentiate between control from the conventional control path and the Breeze instruction decoder. We do not model any of the existing hardware units.

5. Area, Power, and Timing Results

Table 2 shows the composite estimates of area, power, and timing for the hardware looping and address generation circuitry when implemented by using the cell-based methodology described in section 3. The results for the Breeze instruction decoder are merged into the looping and address generation hardware.

Table 2. Timing, Area, and Power Estimates for hardware looping and address generation

	Hardware Looping (5 loops)			**Address Generation (per stream)**		
	Time (ns)	Area (μm^2)	Power (mW) at 1GHz	Time (ns)	Area (μm^2)	Power (mW) at 1GHz
G12-p (0.18µ)	1.00 ns	72830 μm^2	88.57 mW	1.74 ns	57398 μm^2	85.16 mW
G12-d (0.18µ)	1.16 ns	64666 μm^2	62.40 mW	1.91 ns	41053 μm^2	46.18 mW
G12-l (0.18µ)	1.30 ns	65714 μm^2	55.44 mW	2.22 ns	41144 μm^2	42.34 mW
G11-p (0.25µ)	1.49 ns	273249 μm^2	249.30 mW	2.60 ns	165099 μm^2	193.20 mW
G11-v (0.25µ)	1.90 ns	500864 μm^2	166.00 mW	3.29 ns	204603 μm^2	82.93 mW
G10-p (0.35µ)	2.01 ns	-	846.90 mW	3.76 ns	-	554.30 mw

Area of commercial SIMD and GPP units for comparison [6][17]

VIS – 4 mm^2 in a 0.29-micron process
MMX – 15 mm^2 in a 0.29-micron process
AltiVec – 15 mm^2 in a 0.25-micron process
Pentium III processor – 106 mm^2 in a 0.18-micron process
MMX + SSE in a Pentium III processor – 3.6 mm^2 in a 0.18-micron process

Area - The overall chip area required for implementing the hardware loops, address generation (for all four data streams), and the Breeze instruction interpreter (merged into looping and address generation logic) is approximately 0.31 mm^2 in a 0.18-micron library optimized for high performance (G12-p) and 0.23 mm^2 in a 0.18-micron library optimized for density, performance, and power (G12-d). Table 2 also shows the hardware area cost of commercial SIMD GPP implementations for comparison. In a 0.29-micron process, the increase in chip area for implementing the Visual Instruction Set (VIS) hardware into the Sparc processor family was 4 mm^2, MMX into the Pentium family was 15 mm^2, and AltiVec into the PowerPC family was 30 mm^2 [6]. In a 0.25-micron process, the AltiVec hardware was expected to occupy 15 mm^2 [6]. In a 0.18-micron technology, the die size of a Pentium III processor was 106 mm^2 with the MMX and SSE execution units requiring approximately 3.6 mm^2 [17]. Thus, the increase in area due to the MediaBreeze units for SIMD-related hardware is less than 10% and the overall increase in chip area is less than 0.3%. Furthermore, in order to maintain wirability, placement automation tools typically achieve an active area that is about 50 to 80% of the area more than when the cells are packed closer by custom design [12]. This is to facilitate more number of available wiring tracks for optimal routing by the placement tool. Fig. 6 shows the fraction of area required by interconnect for the hardware looping and address generation logic.

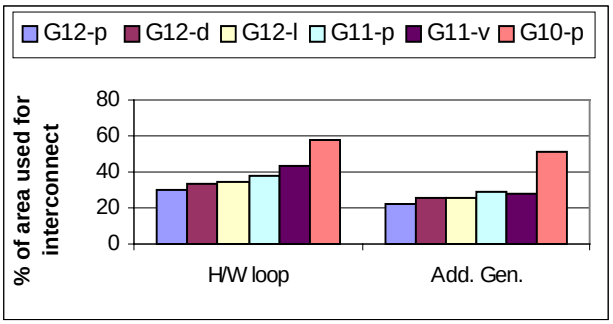

Figure. 6. Percentage of interconnect area in the overall area

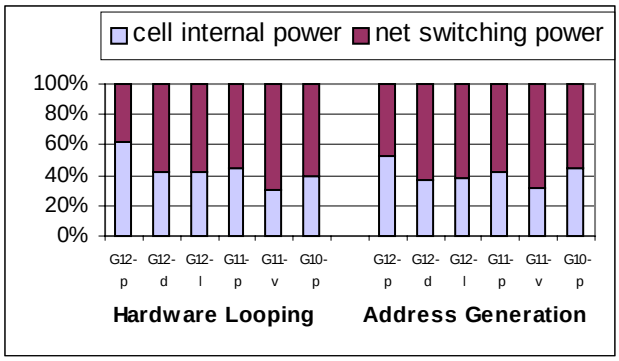

Figure. 7. Breakdown of dynamic power into cell internal power and net switching power

Power - The power consumed by the looping, address generation (all four streams), and the Breeze instruction interpreter is approximately 430 mW in a 0.18-micron library optimized for performance (G12-p) and 225 mW for a library optimized for low-power (G12-l). GPPs with speeds over 1 GHz typically consume a power ranging from 50 W to 150 W. The MediaBreeze hardware consumes less than 1% of the total processor power. We believe that the overall energy consumption of the MediaBreeze architecture would be less than that of a superscalar processor with SIMD extensions because the Breeze instruction reduces the total dynamic instruction count. Since the Breeze instruction is densely encoded very few Breeze instructions are needed for any media-processing algorithm. The number of dynamic instructions that need to be fetched and decoded reduces tremendously, leading to a minimal use of the instruction fetch, decode, and issue logic in a superscalar processor. The instruction fetch and issue logic are expected to consume greater than 50% of the total execution power (not including the clock power) in future speculative processors [16]. Once the Breeze instruction is interpreted, the instruction fetch, decode, and issue logic in the superscalar processor can be shutdown to save power.

Fig. 7 shows the divisions of dynamic power into net switching power and cell internal power for the added MediaBreeze hardware units. Dynamic power consists of two parts: net switching power and cell internal power. Net switching power is the power consumed on a net when the signal it is carrying is toggling and is proportionally dependent on the switching activity, the net load and the square of voltage. Cell internal power is the power consumed when one or more cell input signals toggle. During the transition time when an input or an output signal changes state, both the pull-down and pull-up transistor will be open and a large current will flow through the cell (short-circuit power). Cell internal power is proportionally dependent on the switching activity and the square of voltage [18][19].

Timing - Pipelining the hardware looping logic into two stages would allow for incorporating it into current high-speed superscalar out-of-order processors with over 1 GHz clock frequency. Similarly the address generation stage needs to be divided into three pipe stages to achieve frequencies greater than 1 GHz. The timing results show that incorporating the MediaBreeze hardware into a high-speed processor does not elongate the critical path of the processor (after pipelining). The Breeze instruction decoder multiplexers that control the hardware units introduce an extra gate delay in the pipeline. However, using a cell-based methodology gives a worst-case result while custom design (typically used in commercial general-purpose processors) would allow for greater clock frequencies for the added MediaBreeze hardware. In spite of adding five pipeline stages, the overall pipeline depth of a processor is not affected because the looping and address generation stages bypass the conventional fetch, decode and issue pipe stages.

6. Summary and Conclusion

Contemporary SIMD techniques extract data level parallelism in the useful computation instructions of multimedia applications, but the SIMD execution units are under-utilized due to a dominance of overhead instructions (75-85%). Eliminating and reducing the overhead using specialized hardware can dramatically improve the performance of media workloads without deteriorating the performance of general-purpose workloads. The MediaBreeze architecture was designed to decouple the useful and overhead instructions and exploit instruction level parallelism in the overhead instructions realizing a 1.05x to 28x performance improvement over a SIMD-enhanced general-purpose processor.

In this paper, we estimated the cost of incorporating explicit hardware support into a SIMD GPP to execute the supporting instructions. Using an ASIC cell-based meth-

odology targeting several libraries and technologies, we obtained area, power, and timing information for the MediaBreeze architecture components. The major findings of this paper are:

- The area cost is less than 10% of the SIMD execution unit's area (such as MMX and SSE). When compared to the overall processor chip area, the increase is less only 0.3%.
- Power consumption of the added units is less than 1% of the total processor power.
- The MediaBreeze hardware units do not increase the effective pipeline depth of a high-speed GPP. A SIMD GPP with MediaBreeze hardware can be operated well over 1 GHz in a 0.18-micron technology.

We expect the overall energy consumption to be potentially lower when using a MediaBreeze augmented processor because the Breeze instruction is heavily encoded and encompasses multiple operations. Power consumption is decreased due to the reduced use of traditional instruction fetch, decode, and issue logic for the duration of the Breeze instruction execution.

References

[1] K. Diefendorff, P. K. Dubey, R. Hochsprung, and H. Scales, "AltiVec extension to PowerPC accelerates media processing," *IEEE Micro*, vol. 20 No. 2, pp. 85-95, Mar/Apr 2000.

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

[3] P. Ranganathan, S. Adve, and N. Jouppi, "Performance of image and video processing with general-purpose processors and media ISA extensions," *Proc. 26th IEEE/ACM Sym. on Computer Architecture*, pp. 124-135, May 1999.

[4] R. Bhargava, L. K. John, B. L. Evans, and R. Radhakrishnan, "Evaluating MMX technology using DSP and multimedia applications," *Proc. 31st IEEE/ACM Sym. on Microarchitecture*, pp. 37-46, Dec. 1998.

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

[6] L. Gwennap, "AltiVec vectorizes PowerPC," *Microprocessor Report*, vol. 12, no.6, May 11, 1998.

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

[8] 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.

[9] "Synopsys online Sold Documentation system," version 2000-0.5-1.

[10] "LSI Logic ASIC technologies," Available online at: *http://www.lsilogic.com/products/asic/technologies/index.html*

[11] "LSI Logic ASKK Documentation System".

[12] D. H. Allen, S. H. Dhong, H. P. Hofstee, J. Leenstra, K. J. Nowka, D. L. Stasiak, D. F. Wendel, "Custom circuit design as a driver of microprocessor performance," *IBM Journal of Research and Development*, vol. 44, no. 6, Nov. 2000. Available at: *http://www.research.ibm.com/journal/rd/446/allen.html*

[13] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of Solid-State Circuits*, pp. 1277-1284, Sep. 1996.

[14] 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.

[15] A. Chang, W. J. Dally, S. W. Keckler, N. P. Carter, and W. S. Lee, "The effects of explicitly parallel mechanisms on the Multi-ALU processor cluster pipeline," Proc. IEEE Conf. on Computer Design, pp. 474-481, Oct. 1998.

[16] K. Wilcox and S. Manne, "Alpha processors: A history of power issues and a look to the future," *Cool Chips Tutorial in conjunction with IEEE/ACM Sym. on Microarchitecture*, Nov. 1999.

[17] IA-32 implementation (Intel P3). Available at: *http://www.sandpile.org/impl/p3.htm*.

[18] D. Hansson, "Reducing power in a RISC/DSP core," *Electronic Engineering Times*, Aug. 7, 2000. Available at: *http://www.eetimes.com/story/OEG20000807S0034*.

[19] G. K. Yeap, *Practical Low Power Digital VLSI Design*, Kluwer Academic Publishers, 1998.