Evaluating VLIW and SIMD Architectures for DSP and Multimedia Applications

Deepu Talla
Department of Electrical and Computer Engineering
The University of Texas at Austin
deepu@ece.utexas.edu

Abstract *– Digital signal processing (DSP) and multimedia applications are expected to be the dominant workloads on future computer systems. In this paper, we evaluate the performance of a very long instruction word (VLIW) processor using Texas Instruments Inc.'s TMS320C6x and a single-instruction multiple-data (SIMD) processor using Intel's Pentium II processor (with MMX) on a set of benchmarks. Our benchmark suite includes kernels (filtering, autocorrelation, and dot product) and applications (audio effects, G.711 speech coding, and speech compression). Each benchmark has three versions – scalar non-MMX C code, MMX optimized code, and C6x optimized VLIW code. Optimized assembly libraries and compiler intrinsics were used to create the MMX and VLIW code. Speedup obtained using SIMD and VLIW techniques was quantified as the ratio of the execution cycles with the scalar non-MMX C code as the baseline. We used the hardware performance counters on the Pentium II and the stand-alone simulator for the C6x to obtain the execution cycle counts. The observed speedup for the SIMD version of the suite ranges from 1.0 to 5.5 while the speedup of the VLIW version ranges from 0.63 to 9.0. In addition to quantifying the speedup, we also quantify the effects of MMX on several architectural features – clock cycles per instruction, branch frequency, micro-operations per instruction, and dynamic instructions.*

1. Introduction

Digital signal processing (DSP) and multimedia applications, where text becomes the exception rather than the rule, are now starting to become exceedingly important for computer systems as a dominant computing workload [1][2]. Dynamic multimedia component technologies such as video conferencing, 3D graphics, animation, speech processing, and speech recognition hold great promise. In contrast to traditional applications, multimedia and DSP-rich applications involve significant computational demands on the processor. With an ever-increasing proportion of CPU cycles being used to run such applications, it is pertinent to design machines that speed up programs that consume a large portion of the computation time.

DSP and multimedia applications have been identified to have significant fine and coarse grained parallelism [1]. Very long instruction word (VLIW) architectures incorporate multiple functional units in the data path to exploit instruction-level parallelism (ILP). A single instruction specifies more than one concurrent operation – the instruction width is quite large (sometimes up to eight times that of conventional architectures) and takes many bits to encode multiple operations. VLIW processors rely on software to pack the collection of operations (compaction). However, in order for the VLIW processor to sustain an average number of cycles per instruction comparable to that of a scalar processor, the operations specified by the VLIW instruction must be independent of one another. Since there is a good deal of instruction-level parallelism in DSP and multimedia applications, VLIW techniques are suitable for such implementations. Single-instruction multiple-data (SIMD) techniques are instruction set architecture extensions to general-purpose superscalar processors. These techniques exploit data parallelism as opposed to ILP – each instruction operates on multiple data in a single instruction (same type of operation on all data elements). Many of the DSP and multimedia applications can use vectors of packed 8-, 16- and 32-bit integers and floating-point numbers that allow potential benefits of SIMD architectures like MMX for the X86 family of processors and the Visual Instruction Set (VIS) extensions for the UltraSparc processors.

In this paper we evaluate the effectiveness of VLIW and SIMD processors for DSP and multimedia applications choosing one modern representative commodity processor from each category – TI's C62x DSP processor as the VLIW representative and Intel's Pentium II with MMX as the SIMD representative. The C62x is a fixed-point processor in the C6x family running at 200 MHz executing up to eight 32-bit instructions every cycle. The eight functional units of the C62 core, which include six ALU's and two load-store units, are highly orthogonal providing the compiler and assembly optimizer with many execution resources. The Pentium II processor has a three-way superscalar architecture with 57 MMX instructions added to the traditional X86 instruction set. By packing many pieces of data into one 64-bit MMX register, several operations can take place simultaneously.

Previous efforts have analyzed the benefits of SIMD extensions on general-purpose processors [3][4][5]. An evaluation of MMX on a Pentium processor on kernels and applications was presented in [3]. However, such an analysis on a modern out-of-order speculative machine like the Pentium II is not reported in literature. Moreover, the impact of MMX on branch frequencies, clock cycles per instruction, and micro-operations per instruction were not reported. Performance of image and video processing with VIS extensions was analyzed in [4] and a performance increase was reported. A number of commercial general-purpose and DSP processors have been benchmarked by [5] on a suite of 11 kernels. However, only the execution time is disclosed in the public domain. Moreover, the Pentium II has not been evaluated in their work. In addition their benchmarks suite did not include any applications. Analysis of the memory system was presented in [6] on a suite of media benchmarks (MediaBench), but again there is no SIMD version of any of their benchmarks. Available parallelism in video workloads was measured in [7] with a VLIW architecture. But they assume infinite functional units and a powerful compiler with 100% accurate prediction capabilities. In this paper, we evaluate the both VLIW and SIMD processing paradigms on common suite of benchmarks. Section 2 discusses the benchmarks and methodology for this work. Section 4 analyzes the results and section 5 concludes the paper.

2. Benchmarks and Methodology

We chose three common DSP kernels and three applications to comprise our suite of benchmarks (as shown in Table 1). Each of our benchmarks operate on 16-bit data and has three versions – a scalar non-MMX code, an optimized MMX version, and an optimized VLIW version. The scalar non-MMX code was mostly hand-written in C and the remainder was obtained from speech coding resources [8]. In the creation of the MMX version of the benchmarks, Intel's C/C++ compiler's MMX intrinsics were used in addition to optimized native signal processing (NSP) libraries from Intel [9]. MMX intrinsics allow the user to use normal variables instead of register variables explicitly as done in assembly coding, and the compiler replaces the intrinsic code with appropriate MMX assembly code [10]. In coding the MMX version of our benchmarks, we had to manually perform loop unrolling to maximize performance. In the case of FIR filtering, in addition to loop unrolling, four copies of filter coefficients were necessary to avoid data mis-alignment. Both the non-MMX and MMX versions were compiled for maximum execution speed performance. In the case of the VLIW code, we used optimized assembly libraries [11] wherever possible in addition to compiler intrinsics provided by the C6x compiler. The C6x compiler provides four levels of optimization and we used the highest optimization flag (-o3) that supports loop unrolling, software pipelining, dead-code elimination, and function inlining.

Kernels	
Dot product	Dot product of a randomly-initialized 1024-element array
Autocorrelation	Autocorrelation of a 4096 element vector with a lag of 256
Finite Impulse Response Filter	Low-pass filter of length 32 operating on a buffer of 256 elements
Applications	
Audio Effects	Adding successive echo signals, signal mixing, and filtering
G.711 speech coding	a-law to u-law conversion and vice versa as specified by ITU-T standard
ADPCM (Adaptive Differential Pulse Code Modulation) speech compression	16-bit to 4-bit compression of a speech signal (obtained from Intel)

Table 1: Summary of benchmark kernels and applications

All of the benchmarks fall under the data flow model of computation. Most of the control flows and memory access patterns are statically determinate. All the kernels and the audio effects application can be modeled as a synchronous data flow model. G.711 and ADPCM are based on table-lookup and require Boolean data flow modeling. In the case of SIMD processing, data parallelism has to be explicitly exploited by the programmer (using intrinsics in this case) as compilers do not adequately utilize this parallelism. However, the VLIW compiler is essential for finding the instruction parallelism in each of the benchmarks (although assembly routines exist for several kernels, compiler is necessary for applications).

Hardware performance counters present in the Pentium line of processors are used for gathering the execution characteristics of both the non-MMX and MMX versions of the benchmarks. Gathering information from the counters is simple and non-obtrusive (the benchmark is allowed to execute at normal speed). In addition to the execution clock cycles, the performance counters can be used to obtain the number of dynamic instructions, number of branches, number of micro-operations, number of MMX operations (including a break-up of each category of MMX instructions such as logical, arithmetic, shift, and pack & unpack) and memory statistics. Intel's VTune has also been used to profile static instructions. VTune is Intel's performance analysis tool that can be used to get complete instruction mix (assembly instructions) of the code, and is designed for analyzing "hot spots" in the code and optimizing them. For the case of the C6x, the execution cycle counts were obtained from the stand-alone simulator that is especially useful for quick simulation of pieces of code. The "clock" function provided in the simulator returns the execution times of our benchmarks. While measuring the execution cycle counts of each version of our benchmarks, we only monitor the processing of data already pre-loaded into memory and avoid measuring initialization and cleanup routines. Input data for DSP and multimedia applications comes from sources like a sound card, a video card, or a network card, or an Analog-to-Digital converter. Each of the benchmark versions was verified to produce similar functional results. Speedup was quantified as the ratio of the execution cycles of the MMX and the VLIW versions with respect to the scalar non-MMX C code. The results were statistically averaged over several runs (over 1000 in all cases) to remove effects of cold-start from caches in the case of Pentium II processor.

3. Analysis of Results

Fig. 1 shows the speedup of the MMX and VLIW version over the non-MMX version of the benchmarks.

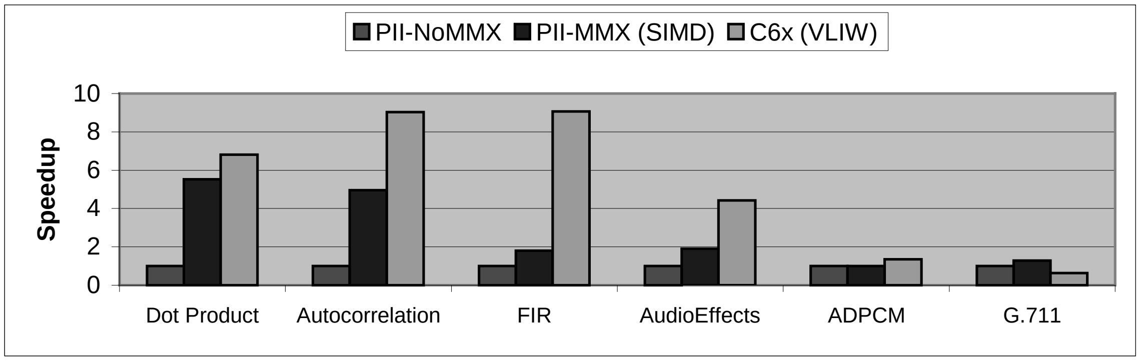

Fig. 1. Ratios of execution times of benchmarks

The dot product kernel shows the most improvement for the SIMD version over the non-MMX version (about 5.5 times). Each data vector operates on four 16-bit data elements and the multiply-accumulate instruction takes 3 cycles whereas the integer multiply in the case of the non-MMX version takes 10 cycles. The C6x code is able to operate on all eight instructions per cycle in the case of the dot product and hence achieve maximum possible performance from the 8-way VLIW processor. It is capable of executing two data elements per clock cycle (in the case of a 1-way scalar processor it would take 4 clock cycles for each data element – two for loads, one multiply and one add). Moreover, it takes advantage of software pipelining to prefetch data three iterations before it is used.

Similarly autocorrelation shows speedups of about 5.0 and 9.0 for the MMX and VLIW versions respectively. For the case of the FIR filter’s MMX version, four copies of filter coefficients were necessary to avoid data misalignment. It shows a moderate speedup of 1.8 for the MMX version (however, the library version of the FIR only shows a speedup of 1.6), but the VLIW version shows a high speedup of over 9.0. For the case of the kernels, the C6x takes advantage of its eight functional units for over 95% of the execution time to achieve maximum parallelism. For the case of the SIMD, speedup of over 5.0 is achieved in the cases of dot product and autocorrelation but not the FIR. This can the attributed to the amount of MMX instructions in each benchmark. Fig. 2 illustrates this in addition to effect of MMX on clocks per instruction.

In the case of the applications, MMX and VLIW versions of audio effects show a speedup of 2.0 and 4.5 respectively. In this application, the original audio signal is mixed with another signal, followed by successive addition of echo signals and some filtering. Again there is only a modest 29% of MMX instructions that contribute to the speedup. The C6x version was mainly developed in C code and some assembly interfacing was done (for the case of kernels, all three VLIW benchmarks were obtained from assembly libraries and called from C code) which is why the speedup is not as high as in the kernels.

The ADPCM benchmark does not have any MMX instructions because this algorithm is inherently sequential and each computation on a data sample depends on the result of the immediate earlier sample (there is no data parallelism). The VLIW code shows only a modest speedup of 1.35 in spite of having multiple functional units because there is a significant amount of control code in this benchmark and the C6x does not employ any branch prediction. The MMX version of the G.711 uses very few MMX instructions (around 4%) and exhibits a speedup of 1.3. The VLIW version surprisingly shows a slowdown (0.63) over the scalar code. A detailed analysis of the assembly code generated by the compiler (no assembly libraries available for either ADPCM or G.711) shows that only one instruction out of the eight possible is being issued each cycle with a number of no-operations placed in between. The G.711 benchmark uses a table-lookup of 8-bit values to compute the conversion from one format to another (u-law to a-law or vice versa). The C6x data width is 32-bits internally and the rest of the 24-bits are being wasted. For the case of MMX version, up to eight calculations can be performed simultaneously, however there is a lot of sequential code preventing from using significant number of instructions.

Fig. 3 shows the effect of branches with MMX instructions. The number of branches is significantly decreased, as the number of MMX instructions in the benchmark is higher. G.711 is the only benchmark that shows an increase in the number of branches with an increase in MMX instructions (however, there are only 4% MMX instructions).

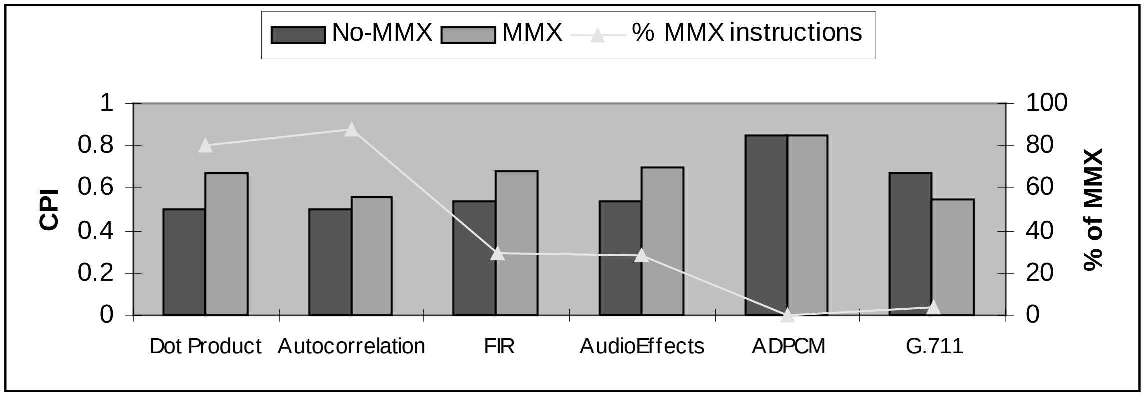

Fig. 2. % of MMX instructions in each benchmark and effect on CPI

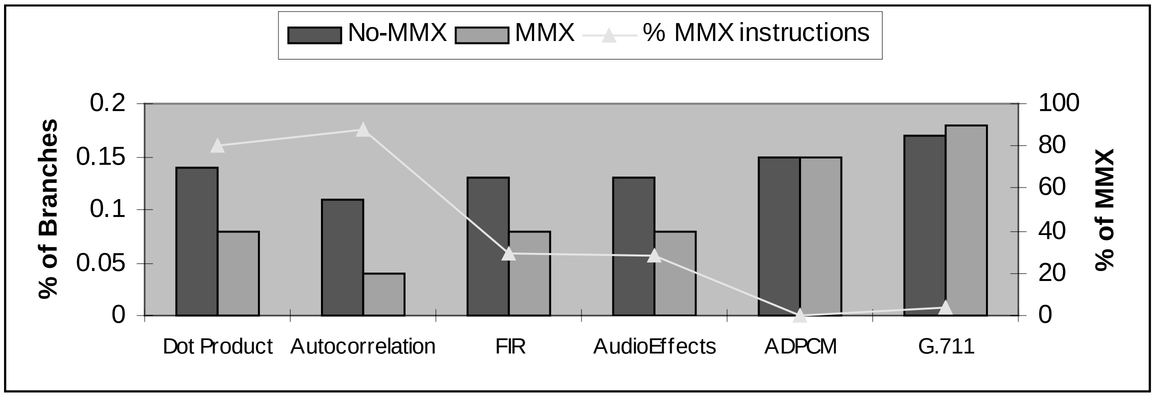

Fig. 3. Effect of MMX on the number of branches

4. Conclusions

In this paper we evaluated the effectiveness of SIMD and VLIW techniques for DSP and multimedia applications. We observed that:

- SIMD techniques provide significant speedup for DSP and multimedia applications. Observed speedups range from 1.0 to 5.5.
- VLIW techniques provide greater benefits (because of more functional units) than SIMD on a number of benchmarks. However, the compiler is critical for achieving maximum benefit. Optimized assembly code is not available for applications and complete benefit of VLIW cannot be realized unless the compiler can schedule the instructions in parallel.

- Compiler intrinsics provide the user with ways to code MMX instructions at a higher level of code development rather than resorting to hand-coded assembly or libraries that can be slow due to error checking routines.
- Using MMX instructions significantly reduces (up to half) the number of branches.
- The number of micro-operations per instruction and the clock cycles per instruction increase by using MMX instructions.

References

[1] K. Diefendorff and P.K. Dubey, "How Multimedia workloads will change processor design", *IEEE Computer Magazine*, pp. 43-45, Sep. 1997.

[2] C.E. Kozyrakis and D.A. Patterson, "A new direction for Computer Architecture Research", *IEEE Computer Magazine*, pp. 24-32, Nov. 1998.

[3] R. Bhargava, L. John, B. Evans and R. Radhakrishnan, "Evaluating MMX Technology using DSP and Multimedia Applications", *Proc. IEEE Sym. on Microarchitecture-31*, pp. 37-46, Dec. 1998.

[4] P. Ranganathan, S. Adve and N. Jouppi, "Performance of Image and Video Processing with General-purpose Processors and Media ISA Extensions", *Proc. Int. Sym. on Computer Architecture*, pp. 124-135, May. 1999.

[5] G. Blalock, "The BDTIMark: A measure of DSP Execution Speed", 1997. White paper from Berkeley Design Technology, Inc. *http://www.bdti.com/articles/wtpaper.htm*.

[6] C. Lee, M. Potkonjak and W.H. Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems", *Proc. IEEE Sym. on Microarchitecture-30*, pp. 330-335, Dec. 1997.

[7] H. Liao and A. Wolfe, "Available parallelism in Video applications", *Proc. IEEE Sym. on Microarchitecture-30*, pp. 321-329, Dec. 1997.

[8] Speech coding Resource. *http://www-mobile.ecs.soton.ac.uk/speech_codecs/*.

[9] Intel, "Performance Library Suite", *http://developer.intel.com/vtune/PERFLIBST/*.

[10] Intel, "C/C++ compiler", *http://developer.intel.com/vtune/icl/*.

[11] Texas Instruments, "TMS320C6000 platform overview", *http://www.ti.com/sc/docs/products/dsp/c6000/62bench.htm*.