Exploiting SIMD Parallelism in DSP and Multimedia Algorithms Using the AltiVec[tm] Technology

Huy Nguyen and Lizy Kurian John*
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712
{hvnguyen, ljohn}@ece.utexas.edu

Abstract

AltiVec technology is Motorola's high-performance vector parallel processing extension to the PowerPC[tm] RISC microprocessor. It is designed to improve the performance of algorithms and applications that can exploit data parallelism such as those in digital signal processing (DSP) and multimedia. In this paper, we investigate the behavior of the AltiVec technology on a set of common DSP and multimedia algorithms. These algorithms include digital filters, fast Fourier transforms, inverse discrete cosine transforms, and vector arithmetic. Each algorithm has one non-AltiVec version, and one version implemented with the AltiVec instruction set by using the AltiVec programming model. The AltiVec version of the algorithms is evaluated using an AltiVec emulator. Traces for both versions are obtained by using an AltiVec-enabled trace generator, and trace-driven simulation is performed by using a cycle-accurate performance simulator. The observed speedup for the AltiVec version of DSP and multimedia algorithms ranges from a factor of 1.60 up to 11.66, and the number of dynamic instructions is reduced by a factor of 1.82 up to 10.25. In addition to quantifying the speedup, we also perform detailed instruction level analysis, which helps to understand issues that become significant while utilizing AltiVec to exploit SIMD parallelism. Efficient utilization of a vector extension such as AltiVec currently requires significant programming effort.

** L. John is supported in part by National Science Foundation under Grants CCR-9796098 (CAREER Award), EIA-9807112, State of Texas Advanced Technology Program grant 403, IBM, Intel, and Microsoft.*

1. Introduction

Traditionally, complex computations in digital signal processing (DSP) and multimedia are performed by specialized DSP processors. In general, general purpose processors (GPPs) can achieve reasonable to excellent floating-point or fixed-point DSP performance via very high clock rates in excess of 200MHz, high-performance branch prediction engine, good memory bandwidth, and superscalar architectures. However, with the emergence of the digital communication technologies along with the networking and telecommunications infrastructure industry, the demand for DSP and multimedia capabilities on GPPs has been increasing tremendously to accommodate 3D graphics, video conferencing and many other applications. The GPP industry has responded to these demands by providing SIMD instruction set extensions to their current instruction set architecture (ISA). For example, Intel's X86 ISA was expanded with the addition of MultiMedia eXtension (MMX) [2, 3, 7]; Sun's UltraSPARC ISA was expanded with the addition of Visual Instruction Set (VIS) [19]; MIPS instruction set was expanded with the addition of MIPS Extension for Digital Media with 3D (MDMX) [20].

Recently, Motorola introduced its own high-performance vector parallel processing expansion to the PowerPC RISC microprocessor [6, 8, 16]. This extension, called AltiVec technology, provides the Single Instruction Multiple Data instruction extension to the current PowerPC instruction set. Like multimedia extensions in other architectures, AltiVec is designed to improve the performance of applications that can exploit data parallelism. The improved performance is achieved through AltiVec by the addition of 162 new unique instructions, which operate on newly defined packed data types and support DSP features such as saturation arithmetic and multiply accumulate. However, unlike other extensions, which established new multimedia data types by devoting the minimum possible amount of silicon to the new features, the AltiVec technology has taken a different approach by devoting a significant portion of the chip area to the new features and emphasizes the growing role of multimedia [8]. With this approach, AltiVec technology promises to offer in a single chip the highest level of processing performance while expanding the processor's capabilities to address high-bandwidth data processing and intensive computations required by DSP and multimedia applications [6].

The AltiVec technology expands the capabilities of the current PowerPC microprocessors by providing a 128-bit vector execution unit, which operates concurrently with the existing integer and floating point units as shown in Figure 1. This new vector execution engine is capable of performing highly parallel operations on multiple data elements by a singe instruction. The AltiVec technology provides support for:

- 16-way parallelism for 8-bit signed and unsigned integers and characters
- 8-way parallelism for 16-bit signed and unsigned integers, and
- 4-way parallelism for 32-bit signed and unsigned integers and IEEE floating-point numbers [6, 15, 16].

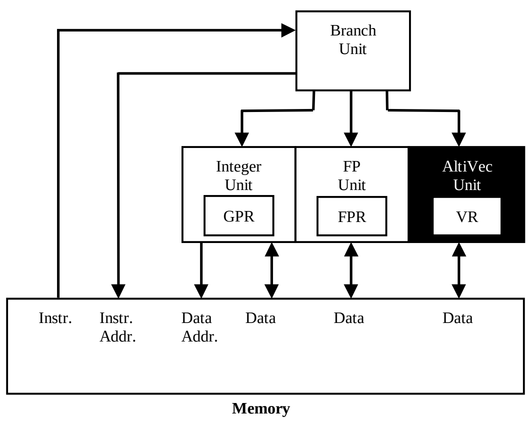

Figure 1: High-level structural overview for PowerPC with AltiVec technology [6]

The AltiVec technology thus has twice the width of competing designs such as MMX [2, 3, 7] and can potentially exploit twice the parallelism that MMX and similar designs can.

The AltiVec technology provides 162 new unique instructions, which can be classified into the following major classes:

- *intra-element arithmetic operations*: performing independent parallel arithmetic computations on the independent elements; arithmetic operations include addition, subtraction, multiply, multiply-add, minimum, maximum and average
- *intra-element non-arithmetic operations*: performing independent parallel non-arithmetic computations on the independent elements; non-arithmetic operations include compare, logical, shift and rotate
- *inter-element arithmetic operations*: performing arithmetic computations, such as sum of products or sum across, on elements within a single vector register
- *inter-element non-arithmetic operations*: performing non-arithmetic computations on elements within a single vector register; these non-arithmetic operations include wide field shift, splat, pack, unpack, merge, and permute [6, 8, 9, 15, 16].

Each instruction can specify up to three source operands and a single destination operand. All operands are vector registers, with the exception of the load and store instructions and a few instructions that provide operands from immediate fields within the instruction opcodes. Vector registers, which come from a 32-entry 128-bit wide register file separate from the existing integer and floating-point register files, can be loaded from and stored to memory through vector load and store instructions [6, 8, 9, 13, 15, 16]. Since the vector registers are separate from the integer and floating-point registers, there is no switching overhead. In contrast, in MMX, programmers can use either the floating-point or MMX registers but not both at the same time.

The AltiVec technology also provides some cache and memory hierarchy management instructions to streamline the prefetch engine. These instructions load data into the caches, using otherwise empty bus cycles. For processing a large array of data, these prefetch instructions can eliminate memory latency without consuming any processor resources [6, 8, 9].

The objective of this project is to study and evaluate the behavior of AltiVec technology on some DSP and multimedia algorithms. DSP and multimedia algorithms chosen for this project are commonly used in many applications [1]. These algorithms include finite impulse response (FIR) digital filter, infinite impulse response (IIR) digital filter, fast Fourier transform (FFT), discrete inverse cosine transform (IDCT), and matrix-vector arithmetic. Previously, there have been similar studies on other SIMD extensions [2, 3, 4]. The purpose of this study, however, is not to compare AltiVec with other SIMD extensions, but these previous studies can certainly provide a good methodology framework. Both Apple and Motorola provide performance comparison tables [15, 16] to compare media functions with and without

AltiVec; however, these tables contain solely speedup numbers and do not provide any insight information regarding the behavior of AltiVec that can affect performance, or any information to quantify the detailed instruction level profiling. To our knowledge, there has been no published evaluation of the multimedia algorithms and applications with AltiVec technology.

To study the effects AltiVec on DSP and multimedia programs, both codes with and without AltiVec for these algorithms are needed. Because AltiVec was introduced recently, AltiVec codes and libraries for DSP and multimedia are not widely available yet. There are a few AltiVec library functions available on Apple's website for downloading [15], such as IDCT and transpose matrix; however, not all common DSP and multimedia algorithms are available. Some of the benchmarks used in our study use the libraries from Apple's website [15] and the rest are hand-coded. Even though that our AltiVec code may not be optimized to utilize AltiVec efficiently, this study still gives some indications about the potential performance gain of AltiVec on DSP and multimedia algorithms and applications. It may also give some hints to AltiVec code developers to utilize AltiVec more efficiently in order to obtain maximum possible performance improvement. The observed speedup of AltiVec ranges from about 1.6 up to 11.66, while the dynamic instruction count is reduced by the order of about 1.82 up to 10.25. The speedup is significant, considering the fact that the AltiVec codes are hand-coded and not yet optimized. Similar to other extensions, this speedup, however, comes with a price, and that is the significant amount of time needed to vectorize and implement the algorithms in AltiVec.

This paper is organized as follows. Section 2 discusses in some detail about the DSP and multimedia algorithms used in this study. Section 3 describes the performance evaluation methodology that is used in this study. The experimental results of this study are presented and analyzed in section 4. Section 5 concludes the paper with a summary of the results.

2. DSP and multimedia algorithms

For this study, we use a set of common DSP and multimedia algorithms, including digital filters, fast Fourier transform, inverse discrete cosine transform, and matrix-vector arithmetic. Both non-AltiVec and AltiVec versions perform their computations on single-precision floating-point numbers in all algorithms except IDCT. For IDCT, 16-bit signed integers are used for calculation. Therefore, the theoretical speedup is approximately 16 for IDCT, and 4 for the rest of the algorithms. All algorithms are called many times in a loop (the same number of times in both versions) in order to minimize the effect of cache misses on initial loads. Table 1 gives a summary of the DSP and multimedia algorithms used in this study. These algorithms are described in more detail in the rest of this section.

Matrix-Vector Arithmetic

Matrix-vector arithmetic, especially matrix-vector multiplication, is extremely important in 3D-Graphics because they are used for several point transformations in the 3D world. Matrix-vector arithmetic used in both versions include:

- matrix-vector multiplication (4x4 * 4x1 single precision floating-point)
- matrix-matrix multiplication (4x4 * 4x4 single precision floating-point)
- 4x4 matrix transpose (single precision floating-point)

Besides 3D-Graphics, matrix-vector arithmetic is also used in many numeric computations with some degree of data parallelism, such as image processing applications. Each of the three above types of matrix-vector arithmetic is run separately, and is called many times in a loop to lessen the impact of initial cache misses.

Finite Impulse Response (FIR) Digital Filters

FIR digital filters allow certain frequency components of the input signal to pass through unchanged while blocking other frequency components of the input signal [2, 3, 10, 12]. Both non-AltiVec and AltiVec versions implement a FIR low-pass filter with length of 35 (N=35) using single precision floating-point numbers. For both versions, FIR digital filter is called 4,000 times to generate 4,000 samples so that the cache misses from the initial loads will have less impact on the cycle count. FIR digital filters are used as general filtering in speech and audio processing, and linear predictive coding.

Infinite Impulse Response (IIR) Digital Filters

Like FIR digital filters, IIR digital filters also allow certain frequency components of the input signal to pass through unchanged while blocking other

Algorithm	Description	Data Type	Approximate Theoretical Speedup	Major Applications
matvec	4x4 * 4x1 matrix-vector multiplication to produce a 4x1 vector (called 12,000 times)	Single-precision floating-point	4	3D Graphics, image processing
matmat	4x4 * 4x4 matrix-matrix multiplication to produce a 4x4 matrix (called 2,500 times)	Single-precision floating-point	4	
mattran	Transpose of 4x4 matrix (called 11,000 times)	Single-precision floating-point	4	
FIR	Low-pass finite impulse response digital filter with order length of 35 (4,000 samples processed)	Single-precision floating-point	4	Speech and audio processing, linear predictive coding, general filtering
IIR	Eighth-order bandpass infinite impulse response digital filter (10,000 samples processed in a block of 4)	Single-precision floating-point	4	Audio equalization, speech compression, linear predictive coding, general filtering
FFT	4,096-point decimation-in-time fast Fourier transform (called twice)	Single-precision floating-point	4	MPEG audio compression, radar processing, sonar processing, ADSL modems, spectral analysis
IDCT	Inverse discrete cosine transform on 8x8 block of halfwords (called 400 times)	16-bit signed integers	8	JPEG image compression, MPEG video compression, H.263

Table 1. Summary of DSP and Multimedia Algorithms

frequency components [2, 3, 10, 12]. However, unlike FIR digital filters, IIR digital filters involve previously computed samples of the output signal as well as samples of the input signal in the computation of the present output sample.

Both non-AltiVec and AltiVec versions implement an eighth-order bandpass IIR digital filter. Like FIR digital filters, all computations of both versions are performed on single precision floating-point numbers. However, unlike the FIR digital filter, both versions of IIR digital filter perform filtering on a block of four samples; each invocation returns four new output samples for each four input samples, and updates the history of intermediate samples. Therefore, to generate 10,000 samples, IIR digital filter is called 2,500 times in a loop so that cache misses from the initial loads will have less impact on the cycle count. Applications of IIR digital filters include audio equalization, speech compression, linear predictive coding and general filtering.

Fast Fourier Transform (FFT)

Fast Fourier Transform is a set of efficient algorithms for computing the discrete Fourier transform (DFT), which is used to calculate the spectrum of the signal [10, 12]. The FFT can simply be regarded as a computational method for taking N numbers in the time domain and creating N numbers in the frequency domain.

Both non-AltiVec and AltiVec versions implement 4,096-point FFT (N=4,096) and perform computations on single precision floating-point numbers. Input data is supplied all at once to the FFT function for computation. FFT is used in many applications, including MPEG audio compression, radar processing, sonar processing, ADSL modems, and spectral analysis.

Inverse Discrete Cosine Transform (IDCT)

The Discrete Cosine Transform (DCT) transforms an image from the spatial domain to the DCT domain to enable a much more compact representation of the data. The Inverse Discrete Cosine Transform (IDCT) transforms an image from the DCT domain back to the spatial domain for image reconstruction. DCT-based image coding is the basis for all the image and video compression standards, including JPEG image compression, MPEG video compression, and H.263 [11].

The IDCT takes an 8x8 block of halfwords of a decoded image in the DCT domain, and produces an 8x8 block of halfwords of a reconstructed image in the spatial domain. All calculations are performed in 16-bit fixed-point arithmetic, where the intermediate results are stored as 16-bit values [15]. To minimize the impact of cache misses from the initial loads, both versions are called 400 times in a loop.

3. Evaluation methodology

This section describes the methodology used in this study. Our intention is to use any existing library functions provided by either Apple or Motorola. This is because vendor codes may be optimized to utilize AltiVec very efficiently. Writing AltiVec code for DSP and multimedia algorithms at assembly language level is very time-consuming in general, and we wanted to perform the coding at higher levels if possible. However, at the time of this study, the only relevant library functions that we can obtain from Apple [15] are the IDCT and matrix transpose. Therefore, except these two, we have to hand-code every algorithm (that we considered) in AltiVec. Non-AltiVec codes, on the other hand, are obtained from [2] and [3]. These non-AltiVec codes were used by Bhargava et al in their study of MMX [2, 3], and have been tested for correctness.

Evaluation Design Flow and Tools

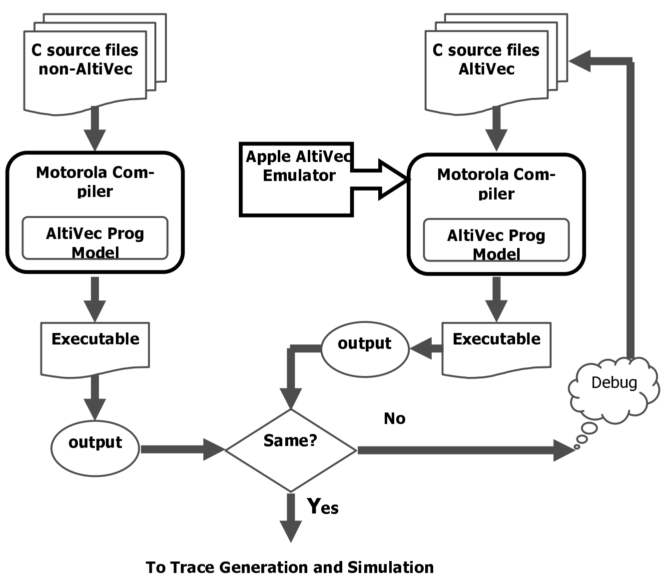

Figure 2: AltiVec Code Generation Design Flow. All development tools indicated are available to the public

Figure 2 shows our AltiVec code generation design flow. Both versions are compiled using the *Motorola C Compiler* (MCC CodeWarrior plug-in) version 4.8.5 with the same compiling and linking options. This compiler supports AltiVec programming model [14], which provides C functions for each AltiVec instruction so that users can write AltiVec code in high-level language instead of in assembly language.

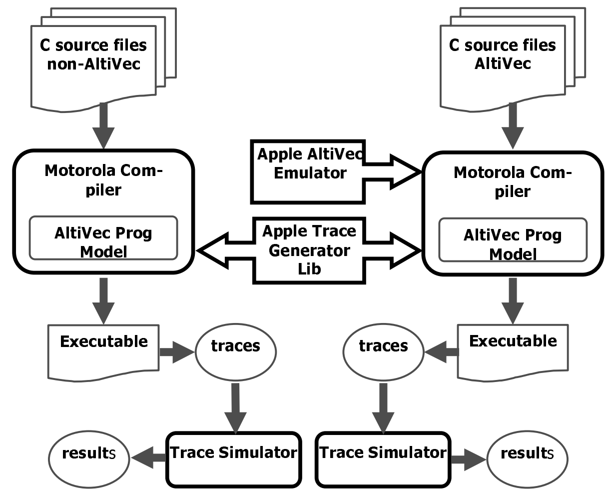

Figure 3: Trace Generation and Simulation. Except the trace simulator, all tools are publicly available

At the time of this study, AltiVec-enabled hardware is not available; hence, an *AltiVec Emulator* version 1.0d4 [17] provided by Apple is used to execute the AltiVec version. The AltiVec version is debugged until its output is the same as the output of the non-AltiVec version.

Once the AltiVec version functions correctly, the performance traces for both versions are generated with the assistance of a trace generator *pitsTT6* version 1.4 [18] provided by Apple. These traces are driven into a cycle-accurate trace-driven simulator* using the same simulation environment and processor configuration for both versions.

Performance Metrics

Our simulation tool provides many performance metrics such as number of clock cycles, number of instructions executed, dynamic instruction mix, cache hit and miss statistics, and branch prediction information. For the purpose of this study, however, we use only the number of clock cycles, number of instructions executed, and dynamic instruction mix as performance metrics in the result analysis. We do not care the cache and branch prediction statistics because they are only meaningful for large applications. While the number of instructions executed and dynamic instruction mix in the traces are processor-independent, the number of clock cycles is not. Therefore, instead of looking at the absolute numbers

* *This is a proprietary PowerPC simulator developed by Motorola*

of clock cycles, we only compute the speedup of AltiVec as compared to non-AltiVec. Because both versions are simulated with the same simulation environment and processor configuration, the computed clock cycle speedup should be valid. Of course, the computed speedup is only valid for that particular simulation environment and processor configuration, but in general, it should be approximately the same with different simulation environments and processor configurations. The instruction counts certainly depend on the compiler. However, because both versions use the same compiler, they are both affected (although equally) by the compiler. Hence, instruction counts for both versions may be different if a different compiler is used, but the ratio of instruction count should be very close to the results obtained in this study.

4. Analysis of results

Table 2 presents the total dynamic instruction counts and clock cycles of each algorithm obtained in this study. Numbers for both non-AltiVec and AltiVec are listed along with their ratios (non-AltiVec over AltiVec). As shown in Table 2, the algorithms show reasonable speedup with AltiVec code, ranging from 1.60 for *IIR* up to 11.66 for *IDCT*. As mentioned, the theoretical speedup is approximately 8 for *IDCT*, and 4 for others. However, this theoretical speedup is only based on the degree of parallelism: 8-way parallelism of 16-bit signed integers for *IDCT*, and 4-way parallelism of 32-bit single-precision floating-point numbers for others. Besides the degree of parallelism, there are many other factors that may affect the performances, such as:

- overhead for data reorganization (packing and unpacking) into and out of packed data types; this, in effect, reduces the theoretical performance gain out of data parallelism
- usage of DSP-intensive instructions provided by AltiVec, such as multiply-accumulate, sum of products, or sum across; these instructions are not available in non-AltiVec versions, and can potentially improve the theoretical performance speedup
- shorter latency for some AltiVec instructions as compared to their non-AltiVec counterparts; this, in effect, increases the gain beyond the projected theoretical performance gain. For example, AltiVec multiply instructions have shorter latency than PowerPC scalar multiply instructions [13, 21]; hence, algorithms with lots of multiplication operations tend to have performance gain better than projected
- different architecture pipelines in the AltiVec execution units as compared to those in scalar units. For instance, the execution stage of PowerPC scalar multiply instructions is not pipelined, allowing only one multiply instruction in the execution stage at one time, and dispatch is stalled until that multiply instruction is finished [21]; in contrast, the execution stage of AltiVec multiply instructions is fully pipelined, allowing multiple multiply instructions in the execution stage at the same time and resulting in the execution throughput of one multiply instruction per cycle [13]
- usage of cache and memory hierarchy management instructions to prefetch the data using otherwise empty bus cycles.

From Table 2, the experimental speedup for most algorithms is less than its theoretical speedup. This confirms the negative effect of the overhead for data reorganization in packed data types. It will be noted later that indeed the AltiVec permute instructions, which perform data packing and unpacking, are used very often in all algorithms, reducing the theoretical performance gain. *IDCT* and *matmat*, however, have experimental speedup greater than their theoretical ones. This is because they have high percentage of AltiVec multiply-accumulate instructions (35.6% and 32.4% of total AltiVec instructions, respectively, as shown in Table 4). These multiply-accumulate instructions indeed help improve the performance of AltiVec. Furthermore, the AltiVec code for *IDCT* obtained from Apple is well optimized, while our hand-coded programs are not. Therefore, if considering the architecture of the AltiVec execution

Algorithm	Total Dynamic Instruction Counts			Clock Cycles		
	Non-AltiVec	AltiVec	Ratio	Non-AltiVec	AltiVec	Ratio
matvec	1,032,001	312,001	3.31	768,355	264,301	2.91
matmat	1,025,001	155,001	6.61	765,391	160,419	4.77
mattran	1,056,001	286,001	3.69	550,379	187,300	2.94
FIR	1,048,004	439,004	2.39	728,759	305,851	2.38
IIR	1,130,004	620,004	1.82	740,783	461,722	1.60
FFT	1,204,490	428,248	2.81	1,021,571	445,849	2.29
IDCT	1,000,401	97,601	10.25	924,856	79,349	11.66

Table 2: Total Dynamic Instruction Counts and Clock Cycles

unit such as dispatch limitations, register dependency, pipeline latency and architectures, and prefetching, the performance gain of other algorithms can potentially be improved further.

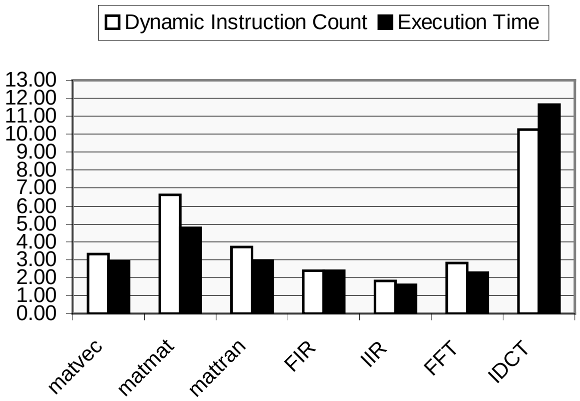

Figure 4: Ratios of Dynamic Instruction Counts and Execution Time (in clock cycles) (non-AltiVec:AltiVec)

Table 2 also shows the total dynamic instruction counts. The dynamic instruction count reduction ranges from 1.82 for *IIR* to 10.25 for *IDCT*. The dynamic instruction count reduction is consistent with the clock cycle speedup, as shown in Figure 4. The more dynamic instruction count reduction, the more data parallelism the algorithm has, resulting in more overall speedup. In theory, the number of instructions should be reduced approximately by the degree of parallelism; however, with the addition of instructions to handle data reorganization into and out of packed data types, the reduction amount is a little bit less than the theoretical value.

Of those AltiVec code we developed (everything except *IDCT* and *mattran*), IIR digital filter is the most difficult to vectorize because it involves the feedback path from the previously computed output samples. Hence, it has the least amount of data parallelism, resulting in the least speedup of 1.60. Matrix-vector arithmetic, on the other hand, is straightforward to implement in AltiVec; therefore, we are able to obtain reasonable data parallelism, resulting in reasonable dynamic instruction count reduction and overall speedup.

Table 3 shows the instruction characteristics of all DSP and multimedia algorithms. Instruction characteristics shown are also consistent with dynamic instruction counts and clock cycles. AltiVec programs that have high percentage of AltiVec instructions have a high dynamic instruction count reduction and high overall speedup. *IDCT*, which has the highest dynamic instruction count reduction and speedup, has the highest percentage of AltiVec instructions (69.1%), while *IIR*, which has the lowest dynamic count reduction and speedup, has the lowest percentage of AltiVec instructions (24.2%).

It is noticed that because of loop unrolling in parallelizing the codes for AltiVec, the numbers of load store instructions are reduced accordingly. In general, they are reduced by a factor about equal to the degree of parallelism. However, because of the way the codes are parallelized, the load/store reduction may be even higher. For instance, the number of floating-point load and stores in a 4x4 * 4x4 floating-point matrix multiplication of non-AltiVec code is approximately proportional to 2x4x4x4 + 4x4 = 144. With AltiVec, the matrices can be loaded into vector registers at once with 2x4 = 8 AltiVec loads and the result is stored directly with only 4 AltiVec stores. Therefore, the reduction in number of load and stores for matrix-matrix multiplication is approximately equal to 144/12 = 12, which is very close to the number obtained from Table 3. Similarly, for *mattran*, the number of load and stores for its non-AltiVec version is proportional to 2x4x4 = 32, while its number for AltiVec version is proportional to 2x4 = 8; so, the number of loads and stores reduction is approximately equal to 8.

Theoretically, without any data reorganization overhead, the number of load/store instructions should be reduced by the same order as the total dynamic instruction count. However, because of data reorganization, there are some extra permute instructions needed, resulting in a lower percentages of load/store instructions, as shown in Table 3. One exception to this is the *FFT* algorithm, in which the percentage of load/store instructions is actually increased in AltiVec. This can be explained by noticing that the *FFT* does not require as much data reorganization as other algorithms (only 10.5% of AltiVec instructions are permute instructions, which are used for data reorganization); furthermore, because of its complexity, *FFT* is not as easy to parallelize, resulting in the smaller reduction in the number of load and store instructions. The number of branches, on the other hand, is affected by calling the algorithms multiple times (the same number of times for both versions) to minimize the effect of initial cache misses. Hence, the branch reduction for all algorithms is not as high as expected.

Algorithm		Integer Arithmetic		Floating-point Arithmetic		Branch		Load/Store[1]		System Unit Instructions[2]		AltiVec[3]	
		count	% of total	count	% of total	count	% of total	count	% of total	count	% of total	count	% of total
matvec	Non-AltiVec	468,001	45.3	192,000	18.6	84,000	8.1	288,000	27.9	0	0.0	N/A	N/A
	AltiVec	96,001	30.8	0	0.0	36,000	11.5	72,000	23.1	0	0.0	108,000	34.6
matmat	Non-AltiVec	440,001	42.9	160,000	15.6	57,500	5.6	367,500	35.9	0	0.0	N/A	N/A
	AltiVec	35,001	22.6	0	0.0	7,500	4.8	30,000	19.4	0	0.0	82,500	53.2
mattran	Non-AltiVec	594,001	56.3	0	0.0	77,000	7.3	385,000	36.5	0	0.0	N/A	N/A
	AltiVec	77,001	26.9	0	0.0	33,000	11.5	88,000	30.8	0	0.0	88,000	30.8
FIR	Non-AltiVec	292,003	27.9	140,000	13.4	60,000	5.7	548,001	52.3	8,000	0.8	N/A	N/A
	AltiVec	135,004	30.8	0	0.0	23,000	5.2	158,000	36.0	8,000	1.8	115,000	26.2
IIR	Non-AltiVec	400,003	35.4	180,000	15.9	90,000	8.0	450,001	39.8	10,000	0.9	N/A	N/A
	AltiVec	300,004	48.4	0	0.0	37,500	6.0	122,500	19.8	10,000	1.6	150,000	24.2
FFT	Non-AltiVec	245,974	20.4	393,216	32.6	65,560	5.4	491,550	40.8	8,190	0.7	N/A	N/A
	AltiVec	59,512	13.9	0	0.0	16,404	3.8	215,104	50.2	2,050	0.5	135,178	31.6
IDCT	Non-AltiVec	697,200	69.7	0	0.0	71,600	7.2	231,201	23.1	400	0.0	N/A	N/A
	AltiVec	11,200	11.5	0	0.0	2,000	2.1	16,801	17.3	0	0.0	67,200	69.1

1. Load/Store counts include AltiVec Load/Store
2. System Unit instructions include move-to and move-from special purpose registers, and logical operations on condition register
3. AltiVec counts exclude AltiVec Load/Store

Table 3: Instruction Characteristics of DSP and Multimedia Algorithms

Algorithm	AltiVec Integer Arithmetic		AltiVec Floating-point Arithmetic		AltiVec Permute		AltiVec Load/Store		AltiVec Multiply Accumulate[1]	
	count	% of AltiVec	count	% of AltiVec	count	% of AltiVec	count	% of AltiVec	Count	% of AltiVec
matvec	0	0.0	48,000	26.7	60,000	33.3	72,000	40.0	48,000	26.7
matmat	0	0.0	40,000	35.6	42,500	37.8	30,000	26.7	40,000	35.6
mattran	0	0.0	0	0.0	88,000	50.0	88,000	50.0	0	0.0
FIR	0	0.0	37,000	13.7	78,000	28.9	155,000	57.4	37,000	13.7
IIR	2,500	0.9	57,500	21.7	90,000	34.0	115,000	43.4	57,500	21.7
FFT	0	0.0	98,304	28.1	36,874	10.5	215,096	61.4	49,152	14.0
IDCT	47,200	57.8	0	0.0	20,000	24.5	14,400	17.6	26,400	32.4

1. Multiply Accumulate is a subset of either AltiVec Integer Arithmetic and/or AltiVec Floating-point Arithmetic. It is listed separately here for further analysis.

Table 4: Breakdown of AltiVec Instructions

To understand more about the effect and characteristics of AltiVec instructions on these algorithms, Table 4 and Figure 5 show the percentage breakdown of AltiVec instructions for each AltiVec version of the programs. As expected, *IDCT*, which performs computation on 16-bit fixed-point integers, contains a high percentage of AltiVec integer arithmetic, while other algorithms, which perform their computations on single-precision floating-point numbers, contain a high percentage of AltiVec floating-point arithmetic. One exception to this is the *mattran*, which only uses AltiVec permute instructions to transpose the matrix, and does not perform any computation. Also, the *IIR* algorithm uses a few AltiVec integer arithmetic instructions just to initialize some vector registers used to control the data reorganization.

From Table 4 and Figure 5, it is also noticed that the permute instructions are used quite often in all algorithms. These instructions not only are used to pack and unpack the data into packed data types, but also are used for data reorganization and manipulation to simplify the parallelization process. The *mattran* actually uses only permute instructions to accomplish its task. Permute instructions are quite useful, but they do not translate into performance gain from AltiVec. For example, *IDCT*'s percentage of permute instructions is lower than most others, but it has the highest performance speedup. In general, optimum performance gain depends on how permute instructions are used. If the algorithms need to use them for data packing and unpacking, then permute instructions have the negative overhead (like in *IDCT*, less percentage of permute instructions translate to less overhead and high performance gain). On the other hand, if the algorithms can take advantage of the highly flexible byte-wise data manipulation feature of the permute instructions, then higher percentage of permute instructions often translates to higher performance speedup (like in *mattran*).

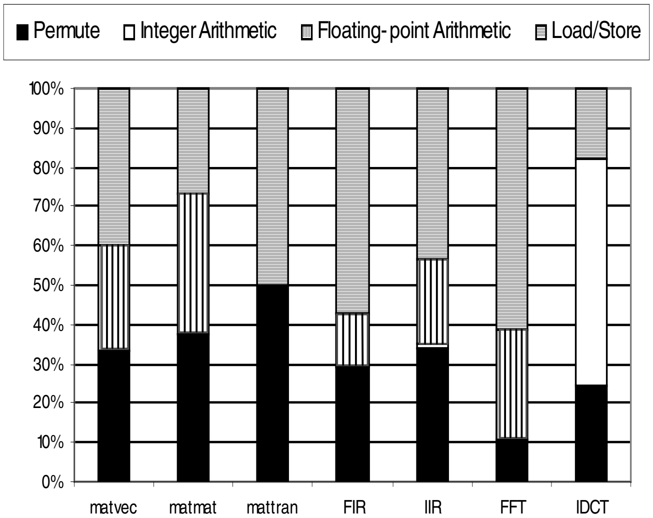

Figure 5: Breakdown of AltiVec Instructions

5. Conclusion

The behavior of AltiVec instruction set on some common DSP and multimedia algorithms is studied and analyzed in this paper. AltiVec can offer significant speedup on DSP and multimedia algorithms even with non-optimized AltiVec codes. Performance speedup from AltiVec depends on many factors, such as:

- data types used in computations
- amount of data parallelism in the algorithms
- amount of data packing and unpacking overhead
- amount of taking the advantage of the very highly flexible bytewise data manipulation instructions

AltiVec is well suited for algorithms that can exploit high bandwidth data parallelism, such as those in DSP and multimedia. These algorithms are commonly used in many applications such as internet protocol telephony, digital communication technologies, speech and audio processing, image and video processing systems, 3D graphics, etc… In this study, the speedup for some common DSP and multimedia algorithms ranges from 1.6 to 11.66 and the instruction count reduction ranges from 1.82 to 10.25.

The performance speedup, however, comes with a price of AltiVec code development. It is time-consuming to parallelize a DSP and multimedia algorithm efficiently. Like other SIMD instruction set extensions, compiler for AltiVec is not available; therefore, AltiVec codes for DSP and multimedia algorithms have to be written manually and optimized by hand, and this is even more time-consuming. In general, it takes several years of optimization to fully utilize innovative hardware introduced [5]. This time includes the time to develop libraries and compilers to utilize the new technology. This paper can be considered as an example of what can be achieved by one person in about 150 hours of programming effort (excluding learning curve of the ISA and the tools) on a new and powerful technology just recently introduced.

AltiVec can potentially offer higher performance speedup than the results obtained in this study when considering:

- the AltiVec execution units' dispatch limitations
- register dependency and renaming
- pipeline latency and architectures of AltiVec execution units
- prefetching
- different implementations of the same algorithm that may utilize AltiVec more effectively.

Because of project time limitation and its complexity, we do not try to heavily optimize the AltiVec codes to utilize AltiVec more efficiently.

As mentioned, one limitation of this study is to use our own hand-written AltiVec codes, which are not optimized. Once Motorola releases its AltiVec library for DSP and multimedia kernels, this same study should be done for more accurate comparison. In addition, more DSP and multimedia algorithms, such as median filter, L-filter, separable convolution, bilinear convolution, quantization, and motion estimation should be included in the future study for further analysis. Furthermore, besides DSP and multimedia kernels, real applications such as image, audio, video and speech processing, and Doppler radar could not be included in the benchmarks for measurement due to the lack of libraries from Motorola or Apple. Handcoding such entire applications is not very practical. Finally, the simulation for this study is done by using trace-driven simulation, which may not be as accurate as running the codes on the actual AltiVec hardware. Therefore, future study should perform simulation on actual AltiVec hardware when it is available [9] to obtain more accurate results. In spite of all the limitations of the study, the paper points to the tremendous opportunities for exploiting SIMD parallelism in multimedia and DSP applications using instruction set extensions to general purpose processors. This paper also points to several nonproprietary tools [17, 18] that may be used to perform detailed performance evaluations of an innovative technology that is just being commercially introduced.

AltiVec is the registered trademark of Motorola Inc.
PowerPC is the registered trademark of International Business Machines Corp.

Acknowledgement

We express our gratitude to Motorola for providing us with their PowerPC cycle accurate simulator. We also thank Apple for making several of their tools and resources [14, 17, 18] publicly available.

References

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

[2] R. Bhargava, R. Radhakrishnan , L. K. John, and B. L. Evans, "Characterization of MMX-enhanced DSP and Multimedia Applications on a General Purpose Processor," *Digest of Workshop on Performance Analysis and Its Impact on Design* (PAID98), Barcelona, Spain, July 1998

[3] R. Bhargava, L. K. John, B. L. Evans, and R. Radhakrishnan, "Evaluating MMX Technology Using DSP and Multimedia Applications," *Proceedings IEEE International Symposium on Microarchitecture*, Dallas, Texas, November 1998, pp. 37-45

[4] W. Chen, H. J. Reekie, S. Bhave, and E. A. Lee, "Native Signal Processing on the UltraSPARC in the Ptolemy Environment, " *Proceedings IEEE Asilomar Conference on Signals, Systems, and Computers*, November 1996

[5] "20 Years of RISC with Lessons for the Future," Dave Ditzel, keynote speech at Micro-31, http://www.acm.org/sigmicro/micro31/pdf/m31-ditzel.pdf

[6] S. Fuller, "Motorola's AltiVec Technology," Networking & Computing Core Technologies, Motorola Inc., Austin, Texas, *http://www.mot.com/SPS/PowerPC/teksupport/teklibrary/papers/altivec_wp.pdf*

[7] L. Gwennap, "Intel's MMX Speeds Multimedia," *Microprocessor Report*, Volume 10, Issue 3, March 5, 1996

[8] L. Gwennap, "AltiVec Vectorizes PowerPC," *Microprocessor Report*, Volume 12, Number 6, May 11, 1998

[9] L. Gwennap, "G4 Is First PowerPC With AltiVec," *Microprocessor Report*, Volume 12, Number 12, November 16, 1998

[10] J. H. McClellan, Ronald W. Schafer, and Mark A. Yoder, *DSP First, A Multimedia Approach*, Prentice Hall, New Jersey, 1998

[11] R. Steinmetz, "Data Compression in Multimedia Computing – Standards and Systems," *Multimedia Systems*, vol. 1, page 187-204, 1994

[12] S. A. Tretter, *Communication System Design Using DSP Algorithms with Laboratory Experiments for the TMS320C30*, Plenum Press, New York and London, 1995

[13] "AltiVec Technology Programming Environments Manual," rev 0.2, May 1998, *http://www.mot.com/SPS/PowerPC/teksupport/teklibrary/manuals/altivec_pem.pdf*

[14] "AltiVec Support in MrC[pp]," rev 1.2, June 6, 1998, *http://developer.apple.com/dev/tools/mpw-tools/compilers/docs/altivec_support.pdf*

[15] Apple's AltiVec Home Page, *http://developer.apple.com/hardware/altivec/*

[16] Motorola's AltiVec Technology Home Page, *http://www.mot.com/SPS/PowerPC/AltiVec/*

[17] AltiVec Emulator, *http://developer.apple.com/devworld/Tools_Chest/Core_Mac_OS_Tools/MPW_Website_Downloads/Emulator-1.0d4.hqx*

[18] pitsTT6 Trace Generator, *http://ftp.apple.com/devworld/Development_Kits/altivec/pitsTT6.sit.hqx*

[19] Sun's Visual Instruction Set Home Page, http://www.sun.com/microelectronic/vis

[20] MIPS Extension for Digital Media with 3D, MIPS Technologies Inc., http://www.mips.com/Documentation/isa5_tech_brf.pdf

[21] MPC750 RISC Microprocessor User's Manual, Motorola Inc., 1997