Characterization of Native Signal Processing Extensions

Jason Law

Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712
jlaw@mail.utexas.edu

Abstract

Soon if not already, multimedia applications such as those that deal with the creation, processing, and communication of digital images, digital audio, and digital video are expected to become one of the dominant computing workloads [6]. Most modern general-purpose processors employ a set of instruction extensions to their architecture to enhance the performance of digital signal processing (DSP) and multimedia applications. It is now possible to perform many tasks, which have in the past required the use of a DSP, using only a general-purpose processor. This is known as Native Signal Processing (NSP). Traditionally, creating code that takes full advantage of NSP extensions is a difficult task that involves writing hand-tuned assembly code [1]. Evaluating the effectiveness of NSP extensions is therefore complicated since it is not easy to use popular benchmarks such as MediaBench, MiBench, or the test suites developed by BDTI and EEMBC. This paper examines the performance benefits seen when using SSE2 extensions on programs not originally tuned for NSP extensions.

1 Introduction

Most modern general-purpose processor architectures now include some sort of native signal processing (NSP) extensions. Intel first released MMX (MultiMedia eXtensions) for its x86 ISA [4]. The MMX extensions on the Pentium and Pentium II included only integer single-instruction-multiple-data (SIMD) instructions so Intel quickly released another set of extensions called SSE (Streaming SIMD Extensions) for the Pentium III that included many floating-point instructions [5]. In November of 2000, Intel released the latest extension to its x86 ISA, Streaming SIMD Extensions 2 (SSE2).

Introduced with the Pentium 4, SSE2 represents a complete overhaul of all previous NSP extensions to the x86 architecture as well as the addition of several new instructions. SSE2 is a substantial evolution of SSE and MMX [5]. SSE2 enables the execution of two double-precision 64-bit floating-point operations per clock; while the original SSE only allowed four 32-bit single-precision operations. SSE2 also extends the original MMX integer instructions from 64 bits to 128 bits. This allows two 64-bit integer operations to be performed at once. Also included in SSE2 are new cache control instructions that allow extensive control of the data cache to minimize cache pollution, and new data prefetch instructions to exploit concurrency between the memory and execution pipelines and to hide memory latency. Using these tools, it should be easier to characterize SSE2 than it was to characterize MMX and SSE.

The main drawback to using NSP extensions is that it is often difficult to write code to take advantage of the extensions. Much time and effort is required to develop code using assembly language, compiler intrinsics, and specialized libraries. This paper

will examine the performance improvements seen when compiling code written without the intention of using SSE2 extensions on a compiler that can intelligently convert vectorizable code into SSE2 code.

In this paper, section 2 describes the benchmark programs. Section 3 describes the methodology used to conduct the experiment. Section 4 analyzes the results, and section 5 concludes the paper.

2 Benchmarks

In this study, four benchmarks consisting of two signal processing kernels and two applications were examined in detail. Table 1 provides gives a brief outline of each benchmark. The rest of this section provides more detail for each of the benchmarks.

Table 1: Benchmark Overview

Benchmark	Description
FIR filter	30-tap FIR filter, 64-bit coefficients, processes 1000000 samples
Dotprod	Calculates dot product of two 2048 arrays 100000 times
Lame3.70	Converts 30 second .wav to 128kb/s .mp3
JPEG	Compress and decompress 780k .ppm to .jpg

2.1 Kernels

Finite Impulse Response (FIR) Filters allow only certain signal components to pass through to the output unchanged. This benchmark implements a 30-tap low pass filter and processes 1000000 samples. Both versions of the FIR filter operate on 64-bit floating-point data. FIR filters are often used in speech and audio processing, modem channel equalization, and general signal filtering.

Dotprod performs a matrix multiplication on two randomly initialized 2048 element arrays. This calculation is performed 10000 times in the benchmark. The benchmark operates on 64-bit floating-point data. Most types of matrix math operations should benefit greatly from vectorization. SSE2 specifically should handle 64-bit data well.

2.2 Applications

Lame3.70 is an MP3 encoder released under the GNU public license. In this benchmark a thirty second .wav format audio file is converted to .mp3 format at 128kb/s, 44.1kHz. MP3 encoders and decoders represent a commonly used desktop signal processing application.

JPEG is a standard, lossy image compression format. The algorithm it uses for compression and decompression is commonly used to view images embedded in documents such as webpages. In the benchmark a color image is compressed and then decompressed.

3 Methodology

All programs were compiled using the Intel C/C++ compiler for Linux. Command line parameters to the compiler allow the inclusion of SSE2 instructions in the compilers output. This allowed the use of the Intel compiler for both versions of the benchmarks. A tool called Abyss allowed access to the performance monitoring counters, which were used to collect data during the actual execution of the benchmarks.

3.1 Intel Compilers

The Intel C/C++ Compiler Version 6.0 for Linux was used to compile all of the benchmarks. Command line options passed to the compiler allow one to indicate the level of optimization desired. This feature was exploited to produce the benchmarks containing SSE2 code as well as the non-SSE2 versions. Compilation can be targeted at a specific Intel microprocessor architecture. In this study, the code that does not contain SSE2 was compiled by not passing any targeting parameters to the compiler other than the standard optimization parameter –O3. The code containing the SSE2 optimizations was compiled by passing a target parameter specifying a Pentium 4 processor and the standard optimization parameter –O3.

There are four methods to produce code optimized for SSE2. The first and the most time consuming is to write in assembly. This has the possibility of giving the largest performance gains. A less painful, but not as effective method is to use compiler intrinsics. These special macro-like functions enable the user to code at higher level, but still incorporate highly optimized assembly routines. The third traditional level is to use vector classes provided by the processor vendor, that the vendor's compiler can assemble into highly optimized code. The last method is to use a compiler that can "autovectorize". This last method is the easiest for the user, but has the least potential for showing speedup.

3.2 Brink and Abyss

The data gathered in this experiment was taken using the performance monitoring counters on the Pentium 4. Access to the performance monitoring counters was provided

by a tool called Abyss. The tool consists of a modular device driver that can be loaded into the Linux kernel while it is running. Brink is a perl script front-end to the kernel module (Abyss) that allows the user to specify what events to monitor and which program to run.

3.3 Metrics

The primary metric used in this study is execution time measured in clock cycles. Speedup is quantified as the execution time of the SSE2 code divided by the execution time of the non-SSE2 code. The number of dynamic instructions executed and the percent of instructions that are classified as SSE2 instructions is also useful to examine.

4 Analysis of Results

The results found in Table 2 show the number of dynamic instructions and micro-ops executed. Also shown in Table 2 is the percent of all instructions executed that are SSE3 instructions. These include: packed and scalar vector operations on SSE2 datatypes, moves between SSE2 registers, data movement into and out of SSE2 registers, and prefetch instructions.

Table 3 shows the execution time in clock cycles of both versions of each benchmark. It also shows the speedup of the SSE2 enabled version over the non-SSE2 version.

Table 2: Benchmark Instruction Characteristics

Benchmark	Dynamic Instructions	Dynamic Micro-ops	Percent SSE2 Instructions
FIR	7302335228	9204025397	
FIR.SSE2	4402070463	5903577025	77.237
Dotprod	1436642154	2052902022	
Dotprod.SSE2	491722432	902977687	72.681
Lame	1056537431	1551707605	
Lame.SSE2	968832088	1330446825	62.279
JPEG	135017200	206559406	
JPEG.SSE2	113837606	150085966	1.5

4.1 Kernel Performance

Kernel performance was disappointing. Dotprod showed significant speedup. The speedup of FIR was disappointing however. Even though, both kernels showed a speedup a larger improvement was expected. Especially noting that around two-thirds of the instructions executed in both cases were SSE2 instructions it is disappointing to see such a low speedup. The non-SSE2 version of the kernels were compiled with the highest level of optimization other than SSE2 extensions and some loop unrolling did occur. This might account for the lack of a significant speedup.

4.2 Application Performance

The applications also showed a speedup that was less than or equal to 2. This is not as disappointing as it was for the kernels. Both applications required no changes be made to the source code. Lame was dominated by file I/O. The .wav file that it encodes to .mp3 is 2.5 megabytes and the output file is 0.5 megabytes. The most surprising result is that with only 1.5 percent of SSE2 instructions JPEE is able to have a speedup of 2.

Table 3: Benchmark Execution Characteristics

Benchmark	Execution Time	Speedup
FIR	4409992088	
FIR.SSE2	3870847952	1.139
Dotprod	1760063740	
Dotprod.SSE2	880663052	2.362
Lame	1972419120	
Lame.SSE2	1410014448	1.398
JPEG	352719592	
JPEG.SSE2	176322068	2.000

5 Conclusions and Future Work

I analyzed the performance gains found when using SSE2 extensions on two DSP kernels and two applications on a Pentium 4 processor. Speedup and dynamic instruction counts were used to evaluate the performance gains due to native signal processing enhancements. I observed that:

- A speedup is seen when using SSE2, even with code that was not written with the intention of using SSE2.
- SSE2 seems well suited for applications that require 64-bit floating-point precision in computations.
- The effort spent tuning an application to use SSE2 is proportional to the performance gain realized.

Future work along these lines should include the use of the vector class libraries and the new signal processing libraries recently released by Intel. It would also be helpful to look at other application and different types of kernels.

References

[1] R. Bhargava, L.K. John, B.L. Evans, R. Radhakrishnan, "Evaluating MMX Technology using DSP and Multimedia Applications," *Proc. 31st Annual ACM/IEEE International Symposium on Microarchitecture*, 1998, pp.37-46.

[2] C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems", *Proc. 30th Annual ACM/IEEE International Symposium on Microarchitecture*, 1997, pp 330-335.

[3] D. Talla, L. John, V. Lapinski, and B. Evans, "Evaluating Signal Processing and Multimedia Applications on SIMD, VLIW and Superscalar Architectures", *Proc. IEEE International Conference on Computer Design*, 2000.

[4] A. Peleg and U. Weiser, "The MMX Technology Extension to the Intel Architecture," *IEEE Micro*, vol.16, no. 4, pp 42-50, Aug. 1996.

[5] S. Raman, V. Pentkovski, and J. Keshava, "Implementing Streaming SIMD Extensions on the Pentium III Processor," *IEEE Micro*, vol. 33, no. 4, July 2000.

[6] P. Ranganathan, S. Adve, N.P. Jouppi, "Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions," *Proc. of the 26th International Symposium on Computer Architecture*, 1999, pp.124-135

[7] R.B. Lee, "Multimedia Extensions for General-Purpose Processors," *IEEE Workshop on Signal Processing Systems*, 1997, pp. 9-23.

[8] T.M. Conte, P.K. Dubey, M.D. Jennings, R.B. Lee, A. Peleg, S. Rathnam, M. Schlansker, P. Song, A. Wolfe, "Challenges to Combining General-Purpose and Multimedia Processors," *IEEE Computer*, Dec. 1997, vol.30, no.12 p.33-7.

[9] M.R. Guthaus, J. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, R.B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," *IEEE Workshop on Workload Characterization*, 2001, pp. 3-14.

[10] "Intel(R) Software College-Streaming SIMD Extensions 2 (SSE2)," *http://developer.intel.com/software/products/college/ia32/sse2*, March 2002.

[11] "EEMBC - Embedded Microprocessor Benchmarking Consortium," *http://www.eembc.org*, March 2002.

[12] "Berkeley Design Technology, Inc. -- Independent DSP Analysis - Optimized DSP Software," *http://www.bdti.com*, March 2002.

[13] BDTi, "Evaluating DSP Processor Performance," *http://www.bdti.com/articles/benchmk_2000.pdf*, March 2002.