Estimating Multimedia Instruction Performance Based on Workload Characterization and Measurement

Adil Gheewala*, Jih-Kwon Peir*, Yen-Kuang Chen**, Konrad Lai**

*Department of CISE, University of Florida, Gainesville, FL 32611
**Microprocessor Research Labs, Intel Corporation, Hillsboro, OR 97124

Abstract: *The increasing popularity in multimedia applications provokes microprocessors to include media-enhancement instructions. In this paper, we describe a methodology to estimate performance improvement of a new set of media instructions on emerging applications based on workload characterization and measurement. Application programs are characterized into a sequential segment, a vectorizable segment, and extra data moves for utilizing the SIMD capability of new media instructions. Techniques based on benchmarking and measurements on existing systems are used to estimate the execution time of each segment. Based on the measurement results, the speedup and the additional data moves of using the new media instructions can be estimated to help processor architects and designers evaluate different design tradeoffs.*

1. Introduction

Digital signal and multimedia processing becomes increasingly popular in many microprocessor applications. Today, almost all the commercial microprocessors, from ARM to Pentium processors, have some types of media-oriented enhancement. For example, the MMX technology has been introduced to the Intel x86 architecture and implemented in Pentium processors [12]. A set of 57 instructions is added to treat data in a Single-Instruction-Multiple-Data (SIMD) fashion. These instructions exploit the data parallelism at sub-word granularity that is often available in digital signal processing or multimedia applications [10].

Performance improvement of using the media-enhanced instructions, such as the MMX technology, is based on three main factors: the amount of data parallelism in applications that can be exploited by the MMX instructions; the granularity of the sub-word parallelism that each MMX instruction can exploit; and the matching of data structure stored in memory and used in MMX registers in order to utilize the MMX instruction capabilities. The third factor involves moving data between MMX registers, and to/from memory or regular registers, which incur additional delays in using the MMX instructions and can offset the overall performance improvement.

One essential issue in defining the media-extension instructions is their ability to exploit parallelism in emerging multimedia applications to achieve certain performance target. Performance evaluation of a new set of media instructions on applications is critical to assess architectural tradeoffs for the new media instructions. The traditional cycle-accurate simulation is a time-consuming process that requires detailed processor models to handle both regular and new SIMD media instructions. In addition, proper methods are needed to generate executable binary codes for the new media-extension instructions to drive the simulator. In this paper, we describe a methodology of estimating performance improvement of new media instructions based on workload characterization and measurement. The proposed method allows processor architects and media experts to quickly estimate the speedup of some emerging applications with a few additional media instructions to the existing instruction set architecture. A uniqueness of this approach is that we only need existing hardware; no cycle-accurate simulator is required.

The basic approach involves several steps. First, a set of multimedia application programs and their equivalent codes with new SIMD instructions are developed. Second, each application program is characterized into three execution segments that include the sequential segment, the segment of the code that can be *vectorized* by a set of new SIMD instructions, and the explicit data-move segment for new media instructions. Third, based on timing measurement on existing systems, the execution time of each segment can be calculated. Finally, the total execution time of an application with additional SIMD instructions can be estimated from the execution times of the three segments. The sequential time will remain unchanged with new multimedia instructions. The execution time of the vectorizable segment can be extrapolated according to the architecture speedup of the new SIMD instructions. The execution time of the data-move segments will depend on the memory hierarchy performance for the data-move instructions.

The proposed method is experimented on Intel Pentium III systems using the equipped MMX technology. To simplify our experiment, we treat several existing MMX arithmetic and logic instructions as new MMX instructions. We further assume that there is no new data move instructions to accommodate for the new set of MMX instructions. In a case study using an inverse discrete cosine transform (IDCT) program, the estimated execution time based on our methodology is within 7\% of the measured execution time. Besides a single estimated execution time, the proposed method can also provide sensitivity studies for a range of speedups based on the performance improvement of new MMX instructions to help architects make design decisions.

The paper is organized as follows. The proposed performance estimation method is described in the next section. This is followed by a case study using the IDCT program on an existing system to verify the method. A summary will be given at the end.

2. Workload Characterizations and Measurement

In the proposed method, three program segments: sequential, vectorizable, and data moves, must be able to characterized separately. The execution time of each segment is derived from the timing measurement on an existing system, where the new set of media SIMD instructions are not available. In this section, we describe detailed steps to accomplish the proposed task. The Intel MMX technology is used as the basis of our experiment. We refer the arithmetic, logical, comparison, and shift media instructions as the *computation* instructions, while the data-manipulation instructions, such as MOVE, PACK and UNPACK, as the *data-move* instructions.

2.1. Estimating Speedup for MMX

The fundamental performance advantage of using SIMD media instructions is similar to parallel programming on vector machines, where the Amdahl's Law can estimate the speedup of an application:

$$Speedup = \frac{1}{(1-f)+(f/n)}$$

f is fraction of the program that can be vectorized, and n is the ideal speedup of f.

In using the MMX technology, programmers must explicitly insert the needed data-move instructions to/from MMX registers for the computation instructions. These data-move instructions can be characterized separately to improve the execution time estimation. Furthermore, in the vectorized segment, several program constructs such as loop controls and procedure calls may not be able to take the advantage of the MMX instructions. Therefore, we can modify the Amdahl's formula to accommodate the MMX technology. This formula will be used as the basis in our estimation method.

$$Speedup = \frac{1}{(1-f)+O+(f-O)/n+D/m}$$

O is portion of the code in the vectorizable segment that cannot be replaced by MMX instructions; D represents the fraction of the data-move instructions; and m is the speedup of the data moves.

2.2. Data Move with Rearrangement

There are a limited number of media registers, i.e. 8, each with a limited width, i.e. 64 bits in the MMX technology. These limitations along with the restriction on how SIMD computation instructions can operate on these registers often creates a *gap* between the normal data lay-out in memory and the format that the data needs to be arranged in the media registers.

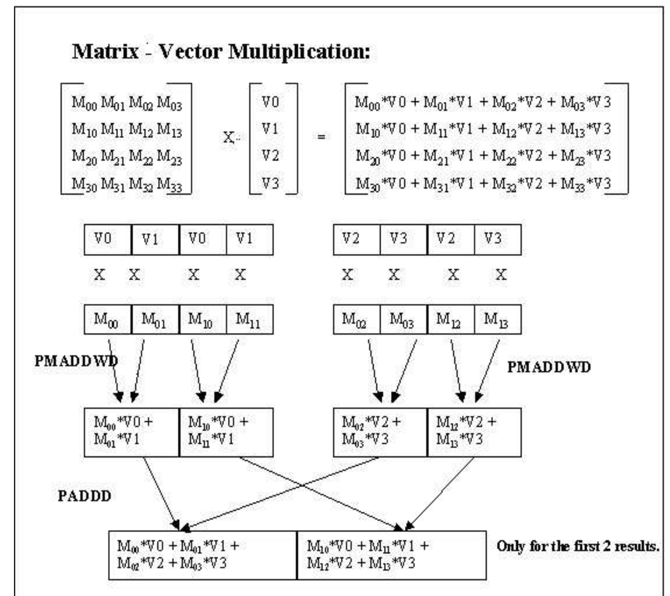


Figure 1. Data Arrangement in Registers for Matrix/Vector Multiplication

Figure 1 illustrates an example of simple matrix/vector multiplication using two MMX instructions. The Packed-Multiply-and-Add (PMADDWD) performs four 16-bits multiplications and two additions to produce two 32-bit partial sums. In conjunction with another PMADDWD and a Packed-Add (PADDD), two elements in the resulting vector can be obtained as shown in the figure. In this

example, each row of the matrix and the multiplicand vector registers to take advantages of the PADDD instructions. This odd data arrangement makes MMX programming difficult. In addition, the extra data arrangement incurs performance penalties that need to be considered in estimating the speedup for adding any new media SIMD instructions.

In contrast, a more natural data arrangement can be accomplished as shown in Figure 2. The entire source vector and each row of the matrix are moved into separate MMX registers for the PMADDWD. To accomplish this, a new PADDD must be invented that adds the corresponding high-order and low-order 32-bit of each of the two source registers to produce the two vector elements in the destination MMX register. However, depending on the subsequent computations, further data rearrangement may still be needed even with the new PADDD instruction.

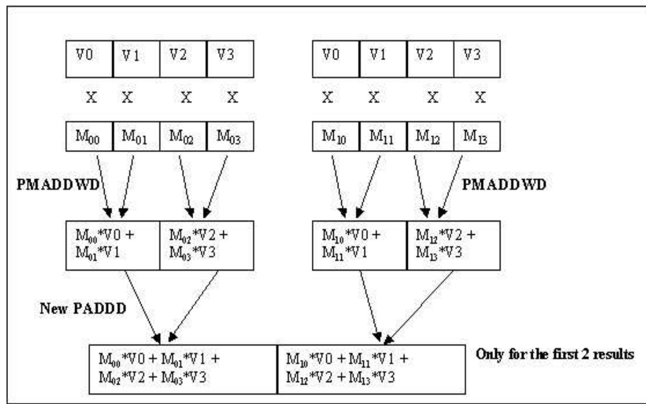


Figure 2. Another Way of Data Arrangement in Registers for Matrix/Vector Multiplication

2.3. Workload Vectorization and Benchmarking

An *8x8* Inverse-Discrete-Cosine-Transform (IDCT) is selected as a case study to demonstrate the proposed performance estimation method. The IDCT is widely used in image and video compression/decompression and has been implemented with the MMX technology [3, 4, 11, 13]. The IDCT program we selected is based on Chen's algorithm that is more suitable for exploiting the MMX technology [1]. The 2-dimension IDCT is decomposed into row-column method of 1-dimension 8-point IDCT. The *idct_row,* which represents a majority of the total execution time, is selected for vectorization in our experiment.

In the first step, the IDCT program (written in *C*, also referred as the *C-code*), and its equivalent vectorized MMX code (referred as the *MMX-code*), are developed. The *MMX-code* includes new SIMD computation instructions along with necessary data-move instructions. In order to test the proposed method on an existing system, four computation must be split and duplicated in the corresponding MMX instructions, PMADDWD, PADDD, PSUBD, and PSRAD are assumed to be *new* to the current MMX instruction set. The *MMX-code* should be optimized to exploit the SIMD capability. We refer this step as a *vectorization* step because of its similarity with respect to developing vector code for vector machines.

The *MMX-code*, with new MMX instructions, is not able to run on the current system. Therefore, we also develop another equivalent MMX code (referred as the *Pseudo MMX-code*). The *Pseudo MMX-code* includes all the data move instructions as that in the original *MMX-code*, along with equivalent MMX-like C instructions for the SIMD computation instructions. Figure 3 shows a portion of the MMX-code from *idct_row* that has been vectorized and its equivalent *Pseudo MMX-code*. In the *Pseudo MMX-code*, the replacement C-code must be perfectly mapped to the corresponding MMX instructions. This *Pseudo MMX-code*, without any new MMX instructions, can now run on an existing system and produces correct execution results.

The equivalent *C-code* is first executed on a host system with the equipped MMX technology. Several time components are measured including the total execution time, the time required for the sequential segment (*1-f*), and the time required for the vectorizable segment (*f*). Note that we use the notations from the modified Amdahl's Law to refer to the respective timing components. The IDCT code is executed one million times to collect all the timing information. The reported timing is an average of 10 separate runs.

The next step is to estimate the time (*D*) needed for the data move instructions. The data move is required for the new computation instructions. The execution time of the *Pseudo MMX-code* is measured on the host Pentium III system. The difference of the execution times between the equivalent *C-code* and the *Pseudo MMX-code* can provide the estimated time (*D*) of the data moves.[1] To verify that all the data moves in the *Pseudo MMX-code* are indeed executed, we use the performance tool *gprof* [14] to collect the execution count of each instruction. In general, multimedia applications have very deterministic program control flows, adding a few data moves does not alter the execution flow of the program.

Similarly, the computation instructions in the *MMX-code* can be removed without replacing them by the equivalent C instructions. This *Cripple-code* will not generate correct results because of the removed computation instructions. Again, using *gprof*, we can verify that all the remaining instructions are executed when

[1] The lowest compiler optimization level is used to make sure the dummy data moves are not removed.

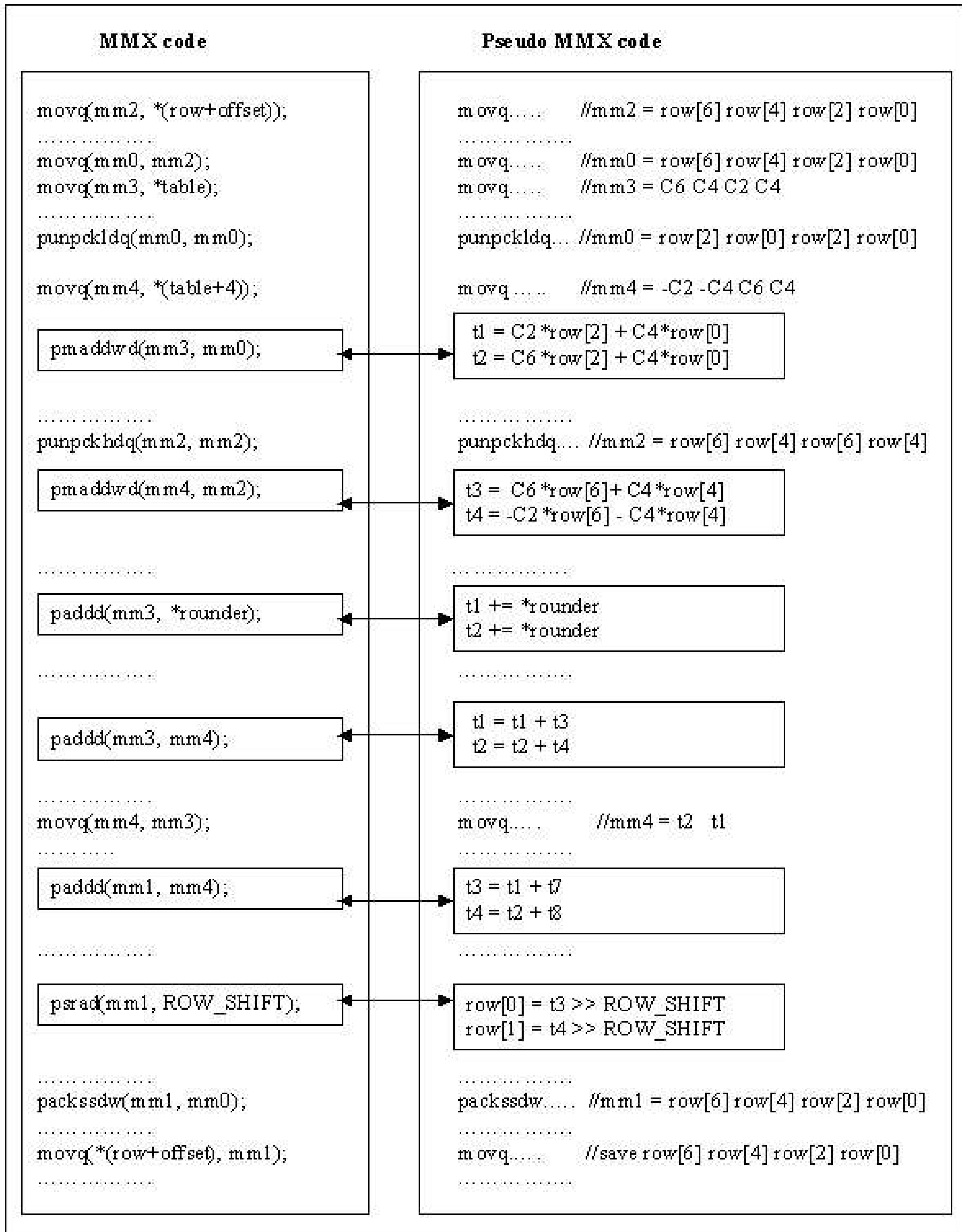


Figure 3. IDCT and Its Vectorized Code using MMX Technology

we measure the execution time for this *Cripple-code*. The measured execution time represents the sequential code (*1-f*) plus the data moves (*D*) that are associated with the MMX-code. Therefore, the difference of the execution times between the *Cripple-code* and the *Pseudo MMX-code* can provide the estimated execution time (*f-O*) for the vectorizable portion of the *C-code*. This vectorizable portion of the total execution time is the main target for improvement with new SIMD computation instructions.

The vectorizable portion (*f-O*) of the *C-code* can be different from the original vectorizable segment (*f*). A few program constructs, such as loop controls and procedure calls, may exist in the vectorizable segment, but cannot be vectorized by the MMX instructions. The execution time (*O*) of this unvectorizable portion can be estimated by subtracting the time (*f-O*) of the vectorizable portion of the *C-code* from the time (*f*) of the original vectorizable segment.

2.4. Performance Projection and Verification

Figure 4 illustrates different components of the four types of program code used in the proposed methodology. The sequential (*1-f*) and vectorizable (*f*) segments of the equivalent *C-code* are identified and their execution times are measured. The *MMX-code* is developed with additional data moves (*D*) for the new MMX instructions. The *Pseudo-MMX code* replaces the new MMX instructions with the equivalent C instructions (*f-O*). The *Cripple-code* removes the new MMX instructions from the *MMX-code*. Given the execution time of different type of codes through measurement, we can compute the execution time for individual components except for the new MMX instructions because the *MMX-code* cannot run on an existing system.

We can now estimate the total execution time and the speedup using the new set of computation instructions. According to the modified Amdahl's formula, the estimated execution time is equal to the summation of the sequential execution time (*1-f*), the extra unvectorizable time (*O*), the time spent on all the new computation instructions (*f / n*), and the time takes for moving the data to/from the MMX registers (*D / m*). The time spent on new computation instructions can be extrapolated by dividing the measured vectorizable timing of the C-code from an architecture speedup factor (*n*) for the new computation instructions. Due to the complexity in estimating memory hierarchy performance, it is more difficult to estimate the time change factor (*m*) for the data moves. One way to remedy this difficulty is to present a sensitivity study. A range of estimated execution times for different memory delays can help gaining insights on the overall impact of new MMX instructions.

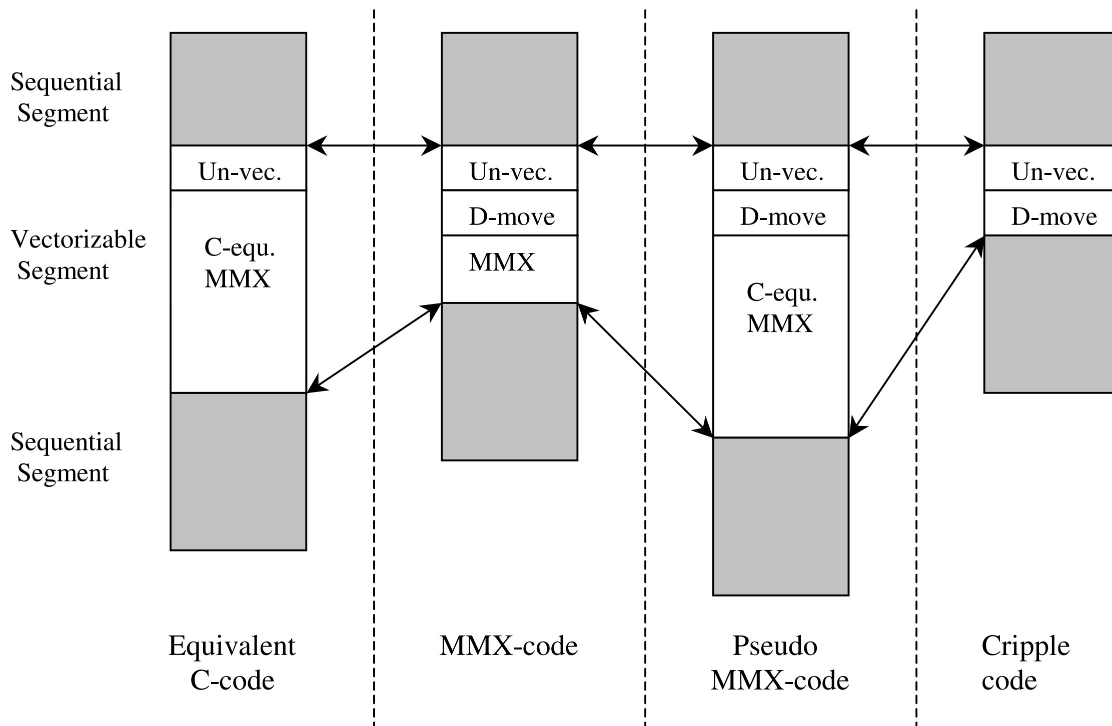


Figure 4. Timing Components of Four Types of Code

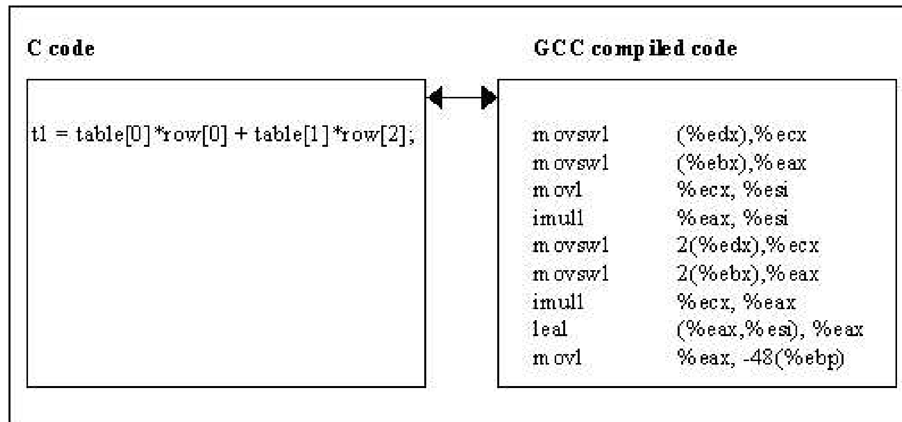


Figure 5. An Example of C Code Segment and Its Assembly Code

To verify the proposed methodology, we pretend that there are four "*new*" SIMD computation instructions, PMADDWD, PADDD, PSUBD and PSRAD. The estimated total execution time can be verified against the measurement time of the *MMX-code* on the host system. (Note that this step is not possible for true new instructions.) Due to the complexity of processor pipeline designs, it is difficult to accurately estimate the speedup factor (n) of the new computation instructions. One alternative approach is to draw a speedup curve based on a range of architecture speedups of the new SIMD instructions. Such a speedup curve, along with the estimated data move overhead can help architects to make proper design tradeoffs.

Several steps have taken for estimating the speedups of the new SIMD instructions. First, the assembly-level code is examined for each new SIMD instruction. Figure 5 shows a C code segment with its corresponding assembly code. This example performs two multiplications and one addition. Using a PMADDMD can benefit a pair of these C instructions. In addition to the two multiplications and one addition, there are several data moves generated by the compiler to move data in and out of registers. In contrast, the programmer explicitly controls the necessary data moves when the PMADDWD is used. Note that by using PMADDMD, we not only put multiple multiplications and additions together in one instruction, but also group a number of memory accesses in one data move instruction. Since the explicit data moves for the PMADDWD are counted separately in our method, the implicit data moves in the assembly code must take into consideration in calculating the speedup for the PMADDMD.

Second, instructions can be executed together in the target Pentium III system with multiple dispatch ports. Each assembly instruction incurs certain delays as specified in the Pentium architecture book. We can estimate the execution latency of the assembly to obtain the estimated speedup factor (n). The details are omitted and can be found in [2].

Third, the above steps are carried out for each new SIMD instruction to obtain the respective speedup. Afterwards, we can calculate the weighted average speedup by counting the number of occurrences of each SIMD instruction in the *idct_row*. An example of estimating the average speedup will be given in the next section.

3. Case Study Results

The IDCT is used as a case study for experimenting the proposed method. We focus on vectorizing *idct_row* using the *new* PMADDWD, PADDD, PSUBD, and PSRAD instructions. We assume that the *idct_column* has been vectorized using the existing MMX technology and is considered as the sequential code segment. The performance timing results from the measurement and projections are summarized in Table 1.

The total execution time the *C-code* takes 1.56 seconds. The sequential segment takes 0.13 seconds, while the vectorizable segment takes 1.43 seconds. Therefore, we are dealing with over 90% of the code that is vectorizable. The *Pseudo MMX code* takes 1.69 seconds. By subtracting the time of the *C-code* from the time of the *Pseudo MMX code,* we can get the estimated delay of 0.13 seconds for the data moves. Although the data moves represent a small portion of the total execution time for the *Pseudo MMX code*, it will be more significant for the *MMX-code* if the delay remains a constant.

With the measured 0.38 seconds of the Cripple-code, we can calculate the vectorizable portion of the C-code by

taking the 0.38 seconds out of the 1.69 seconds of the *Pseudo MMX code.* The result of 1.31 seconds is the portion of the code that can be speeded up by the new SIMD instructions. We can also calculated the time for the unvectorizable portion of the code by taking the time for the sequential segment and the data moves away from the time for the Cripple-code. Finally, the total estimated execution time can be calculated; the result is equal to 0.5419 seconds. Comparing with the measured execution time for the MMX-code, the estimated time is about 7% off the measurement target.

In the above calculations, we use an estimated architecture speedup of 8.09 based on the execution cycles from Intel's Reference Manuals [5, 6, 7, 8, 9]. As shown in Table 2, each PMADDWD equivalent C code takes 14 cycles, which is analyzed based on the assembly code using the architectural information of the current system. The new MMX instruction will take only 1 cycle. The system architect can provide the cycle count information for the new media instructions. Hence the estimated speedup of PMADDWD is 14. The speedups of other new SIMD instructions can be estimated similarly [2]. We can calculate a weighted speedup by averaging the sum of the products of the individual speedup and the number of occurrences of the SIMD instructions in the program.

$$WeightedSpeedup = \frac{(14*8+4*2+4*4+5.5*4+5*4)}{22}$$

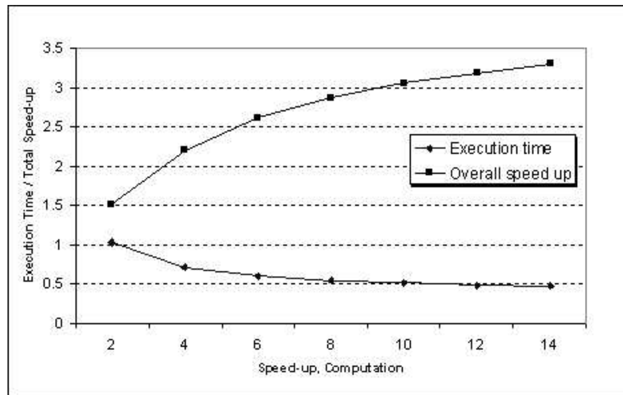
$= 8.09$

Program	Timing Component	Time (sec)
1. C-code	Sequential (idct_col) + Vectorizable portion (idct_row) + Unvectorizable (calls in idct_row)	1.56
	Sequential (idct_col)	0.13
	Vectorizable portion (idct_row) + Unvectorizable	1.43
2. Pseudo MMX code (Computation MMX is replaced by C-equivalent)	Sequential (idct_col) + Vectorizable portion (idct_row) + Moves + Unvectorizable	1.69
	Moves = (2) - (1) = 1.69 - 1.56	0.13
3. Crippled MMX code (Computation MMX is removed)	Sequential (idct_col) + Moves + Unvectorizable	0.38
	Vectorizable portion = (2) – (3) = 1.69 - 0.38	1.31
	Unvectorizable = (3) – Sequential (idct_col) – Moves = 0.38 - 0.13 - 0.13	0.12
4. Estimated Execution time	Total = (0.13 + 0.12 + 0.13) + 1.31/*8.09* (8.09 is the estimated speedup for new MMX)	0.5419
5. Overall Speedup	1.56 / 0.5419	2.878
6. Measured MMX code (for verification purpose)	Sequential (idct_col) + Unvectorizable + New MMX + Moves	0.505

Table 1. Performance Measurement and Projection – A Case Study, IDCT

Code Segment	Operations	C code Cycles	MMX Cycles	Speedup	Number of Occurrence
pmaddwd for a0-b3	4 multiply 2 add	14	1	14	8
paddwd for rounder	2 add	4	1	4	2
paddwd for a0-b3	2 add	4	1	4	4
paddwd and psubwd, row[]	2 add, or 2 subtract	5 6	1 1	5.5	2 2
prad, row[]		5	1	5	4

Table 2. Estimated Speedup for New SIMD Instructions



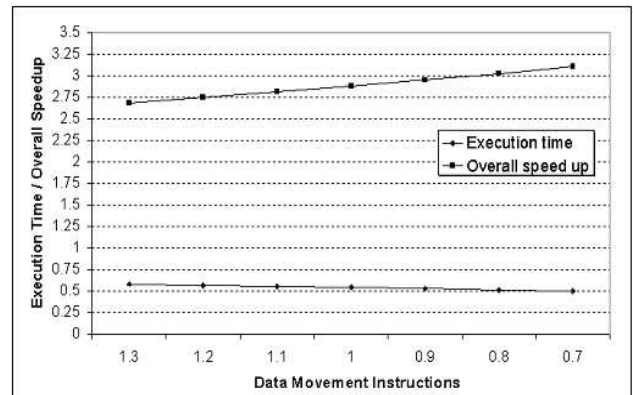


Figure 6. The Total Execution Time and Speedup with Respect to Performance of New SIMD Instructions

Figure 7. The Total Execution Time and Speedup with Respect to Performance of Data Moves

Figure 6 plots the overall execution time and speedup given improvements of the computation instructions. In this figure, we assume the time for the data moves remain unchanged. With 2 times of performance improvement for the new computation instructions, the overall speedup is close to 1.5, while the speedup is over 2.5 given 10 times improvement of the computation instructions. The data moves along take about 8% of the total execution time without any improvement on the vectorizable code. The percentage increases to 26% when the new computation instructions get 10 times of performance improvement.

We also did a sensitivity study on the data move overhead. As shown in Figure 7, with 30% more penalty for the data moves, the speedup is reduced from 2.9 to 2.7. On the other hand, the speedup increases from 2.9 to 3.1 if the data move overhead can be reduced by 30%. In this figure, we use the estimated architecture speedup for the SIMD computation instructions.

4. Summary

A performance estimation method for using new media instructions is presented. Instead of using cycle-accurate simulation, the proposed method estimates execution times with new media instructions based on characterization media workload with benchmarking and measurement on existing systems. Given a range of performance improvement of the new media instructions, the proposed method can provide a range of speedups of using the new media instructions. A simple case study is provided to verify the proposed method.

5. Acknowledgement

This work is supported in part by NSF grants MIP-9624498, EIA-0073473 and by Intel research and equipment donations. Anonymous referees provide very helpful comments.

6. References

[1] W.-H. Chen, C. H. Smith, and S. C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform," *IEEE Transactions on Communications*, vol. COM-25(9), Sept. 1977, pp. 1004--1009.

[2] A. Gheewala, "Estimating Multimedia Instruction Performance Based on Workload Characterization and Measurement," MS Thesis, University of Florida, 2002.

[3] Intel Corp., "A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMX Instructions," Intel Application Notes AP-922, Apr. 1999. On-Line at: (http://developer.intel.com/vtune/cbts/strmsimd/appnotes/ap922/ap922.pdf)

[4] Intel Corp., "Using MMX Instructions in a Fast iDCT Algorithm for MPEG Decoding," Intel Application Notes AP-528, On-Line at: (http://developer.intel.com/software/idap/resources/technical_collateral/mmx/AP528.HTM).

[5] Intel Corp., Intel Architecture Software Developer's Manual, Intel Corporation, Order Number 243190, 1997.

[6] Intel Corp., Intel Architecture Optimization Reference Manual, Order Number 245127-001, 1999.

[7] Intel Corp., Intel Architecture MMX Technology Programmer's Reference Manual, Intel Corporation, Order Number 243007.

[8] Intel Corp., Intel MMX Technology at a Glance, Order Number 243100-003, June 1997.

[9] Intel Corp., IA-32 Intel Architecture Software Developers Manual with Intel Pentium 4 Processor Information Volume 1: Basic Architecture, 2000, (http://developer.intel.com/design/processor/future/manuals/245470.htm).

[10] R. Lee and M. Smith, “Media Processing: A New Design Target,” *IEEE Micro*, Vol. 16(4), Aug. 1996, pp. 6--9.

[11] E. Murata, M. Ikekawa, and I. Kuroda, “Fast 2D IDCT Implementation with Multimedia Instructions for a Software MPEG2 Decoder,” *Proceedings of ICASSP*, Vol. 5, 1998, pp. 3105--3108.

[12] A. Peleg, and U. Weiser, “The MMX Technology Extension to the Intel Architecture,” *IEEE Micro*, Vol. 16(4), Aug. 1996, pp. 42--50.

[13] Y.-S. Tung, C.-C. Ho, and J-L. We, “MMX-based DCT and MC Algorithms for Real-Time Pure Software MPEG Decoding,” *Proceedings IEEE International Conference on Multimedia Computing and Systems*, Vol. 1, 1999, pp. 357--362.

[14] gprof: (http://www.gnu.org/manual/gprof-2.9.1/html_mono/gprof.html)