Performance Comparison of SIMD Implementations of the Discrete Wavelet Transform

Asadollah Shahbahrami Ben Juurlink Stamatis Vassiliadis

Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology, The Netherlands
E-mail: {shahbahrami,benj,stamatis}@ce.et.tudelft.nl

Abstract

This paper focuses on SIMD implementations of the 2D discrete wavelet transform (DWT). The transforms considered are Daubechies' real-to-real method of four coefficients (Daub-4) and the integer-to-integer $(5,3)$ lifting scheme. Daub-4 is implemented using SSE and the lifting scheme using MMX, and their performance is compared to C implementations on a Pentium 4 processor. The MMX implementation of the lifting scheme is up to 4.0x faster than the corresponding C program for a 1-level 2D DWT, while the SSE implementation of Daub-4 is up to 2.6x faster than the C version. It is shown that for some image sizes, the performance is significantly hampered by the so-called 64K aliasing problem, which occurs in the Pentium 4 when two data blocks are accessed that are a multiple of 64K apart. It is also shown that for the $(5,3)$ lifting scheme, a 12-bit word size is sufficient for a 5-level decomposition of the 2D DWT for images of up to 10 bits per pixel.

Keywords: *Discrete Wavelet Transform, lifting scheme, SIMD extensions.*

1 Introduction

The wavelet transform is mainly used for image and video compression. Standards such as MPEG-4 and JPEG2000 [13] are based on the 2D discrete wavelet transform (DWT). The DWT has traditionally been implemented by convolution methods such as finite impulse response (FIR) filters. These implementations require both a large number of operations and a large amount of memory, making them unsuitable for either high-speed or low-power implementations. One way to reduce the execution time of the DWT is by using special-purpose hardware. Programmable processors, however, are preferable because they are more flexible and allow different transforms, various filter bank lengths, and various transform levels. Furthermore, multimedia SIMD extensions such as MMX [12] and SSE [14] can be used to reduce the execution time of the DWT.

In this paper the performance of two wavelet transforms, conventional real-to-real filtering and the integer-to-integer lifting scheme, is evaluated. Both methods are implemented using programmable SIMD architectures. Hence, we present an MMX implementation of the lifting scheme and compare its performance to an SSE implementation of the convolution method. The lifting scheme is considered with the goal to provide a fast and efficient implementation of the DWT to reduce the execution time of JPEG2000.

The $(5,3)$ lifting scheme is considered for various reasons. First, the $(5,3)$ transform has low computational complexity and performs reasonably well for lossy as well as lossless compression compared to other filters [1]. Second, the $(5,3)$ transform is included in Part 1 of the JPEG2000 standard [13]. Third, it is possible to implement the $(5,3)$ filter without using multiplication operations (i.e., using only addition, subtraction, and shift operations). Finally, the $(5,3)$ filter has only one lifting step. Transforms with fewer lifting steps tend to perform better than transforms with more lifting steps in terms of speed as well as accuracy [1]. The convolution method considered in this paper is Daubechies' transform with four coefficients (Daub-4). This transform has been considered in many papers [2, 9].

This paper is organized as follows. Section 2 briefly describes the wavelet transform and explains the SSE implementation of the 2D DWT using Daub-4. In Section 3, the MMX implementation of the $(5,3)$ lifting scheme is discussed. In Section 4 the performance of both SIMD implementations and their C counterparts is evaluated and analyzed. In Section 5 we discuss the limitations of MMX and SSE that restrict the performance improvements that can be obtained for the 2D DWT. Related work is described in Section 6 and conclusions are drawn in Section 7.

2 Wavelet Transform

The wavelet representation of a discrete signal X consisting of N samples can be computed by convolving X with the low-pass and high-pass filters and down-sampling the output signal by 2, so that the two frequency bands each contain $N/2$ samples. This process decomposes the original image into two sub-bands: the lower and the higher band. This transform can be extended to multiple dimensions by using separable filters. A 2D DWT can be performed by first performing a 1D DWT on each row of the image followed by a 1D DWT on each column.

One transform is convolution filtering such as Daubechies' filter of four coefficients (Daub-4). The value of each wavelet coefficient using Daub-4 depends on four pixels and 4 floating-point multiplications and 3 floating-point additions/subtractions (ALU operations) are needed to obtain each transform coefficient. For an image of size $N \times M$, $4NM$ multiplications and $3NM$ ALU operations are therefore needed for each dimension. Because SSE instructions perform 4 operations in parallel, an SSE implementation requires at least NM multiplication instructions and $3NM/4$ ALU instructions, not counting overhead instructions.

Vectorization of the column DWT is easier than vectorizing the row DWT (in the row-major storage format). This is because corresponding elements of adjacent columns (for example, $[0][0] \ldots [0][3]$) are stored consecutively in memory while corresponding elements of adjacent rows are not. To vectorize the row DWT, the elements need to be rearranged. For this, overhead instructions such as *unpcklps* and *unpckhps* are needed. We have implemented Daub-4 using SSE instructions and determined that, for an image of size $N \times M$, the row DWT requires approximately $(5M + 4)N$ instructions to be executed and the column DWT about $(7M+4)N/2$. Furthermore, the dynamic number of overhead instructions is $12MN/8$. So, in order to perform a 3-level 2D DWT of an image of size 512×512, the overhead instructions constitute 17.6% of the total dynamic number of instructions.

3 MMX Implementation of the Lifting Scheme

Recently, a new implementation of the DWT has been proposed, known as the *lifting scheme*. The basic idea of this scheme is to use the correlation in the data to remove the redundancy. The lifting operation consists of several stages, as depicted for the $(5, 3)$ filter bank in Figure 1. First, a trivial wavelet transform is computed, by splitting the original 1D signal into odd and even subsequences and then modifying these values using alternating prediction and updating steps. The sequences $\{s_i^0\}$ and $\{d_i^0\}$ denote the even and odd input sequence, respectively.

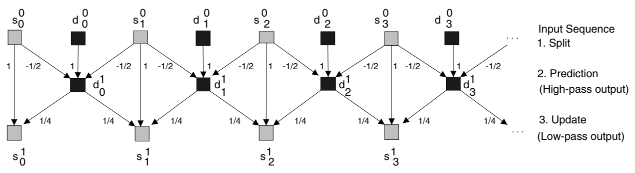

Figure 1. Lifting scheme using the $(5, 3)$ filter bank.

Due to the simple structure of the $(5, 3)$ filter bank, the outputs $\{s_i^1\}$ and $\{d_i^1\}$ are actually the low- and high-pass output coefficients of the DWT filter, respectively. The forward transform of the $(5, 3)$ filter bank used in this paper is given by:

$$d_i^1 = d_i^0 - \lfloor \frac{s_i^0 + s_{i+1}^0}{2} \rfloor, s_i^1 = s_i^0 + \lfloor \frac{d_{i-1}^1 + d_i^1 + 2}{4} \rfloor \quad (1)$$

Figure 2 shows the data flow and dependencies in the $(5, 3)$ lifting scheme based on Equation (1). For an image of size $N \times M$, the MMX implementation based on Figure 2 requires $3NM$ 16-bit additions (corresponding to $3NM$ instructions), NM shift operations ($NM/4$ instructions), and $3NM/8$ load instructions for each dimension. Other authors have omitted the constant 2 in Equation (1) resulting in fewer addition operations [7].

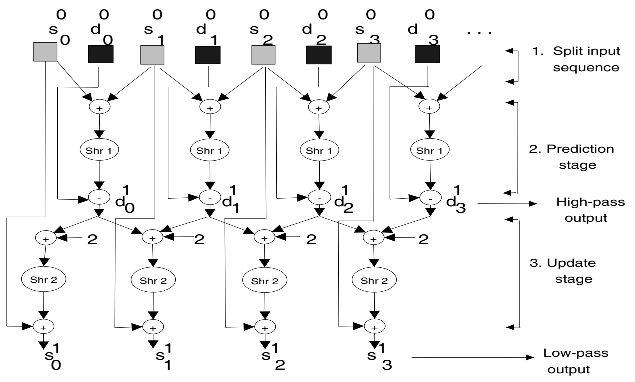

Figure 2. Data flow and dependencies in the forward integer-to-integer lifting using the $(5, 3)$ filter bank (Shr = Shift right).

In order to vectorize the row DWT of the 2D DWT using the $(5, 3)$ filter it is used overhead instructions such as *punpcklbw*, *punpckhbw*, *punpcklwd*, and *punpckhwd*. For an $N \times M$ image, the dynamic number of instructions required by the row DWT is approximately $(5 + 36M/8)N$ and the dynamic number of instructions needed by the column DWT is about $(5+27M/4)N/2$. Of these instructions, $9MN/8$ instructions constitute overhead instructions. For a 3-level 2D DWT of an image of size 512×512, the overhead instructions constitute 21.1% of the total dynamic number of instructions.

4 Performance Evaluation

Four programs have been implemented, each consisting of three parts. One part is for reading the image, the second part is for processing, and the last part is for storing the transformed image. Two programs are completely written in C. One performs the 2D DWT using the integer-to-integer $(5,3)$ lifting scheme, the other performs the 2D DWT using Daub-4. These programs will be referred to as C-Lifting and C-Daub, respectively. They were compiled using the gcc compiler with optimization level *-O2*. The reading and storing parts of the other two programs were also written in C, but the processing part was implemented using MMX (for the $(5,3)$ filter) and SSE (for Daub-4). They are referred to as MMX-Lifting and SSE-Daub, respectively. The processing part of each program has also been divided in two parts, row and column processing.

Our goal is to measure the time required to execute the processing part of each program under the same conditions (same algorithm, data types, and system). In the MMX implementation, image pixels are represented as 16-bit values, using the `short` data type. For the C-Lifting program, we have experimented with `shorts` as well as `ints`. Contrary to our expectations, in most cases the implementation that uses `ints` is faster than the program that employs `shorts`. Accordingly, we compare our results to the program that represents image pixels as `ints`. Our system is a 3.0GHz Pentium 4 with an L1 data cache of 8KB that is 4-way set-associative and has a line size of 64 bytes. The 512KB L2 cache is 8-way set-associative with a 64-byte line size.

The lifting scheme computes the DWT *in-place*. This means that the transform is performed without using an extra array. But this memory saving is at the cost of a post-processing step, where the wavelet coefficients are rearranged. In order to avoid this rearrangement step, an output matrix for storing the results of the horizontal filtering is used, as was done in [5]. The vertical filtering processes this output array and writes the processed coefficients back to the input matrix. After performing the transformation, the wavelet coefficients are stored in the input matrix in the order expected by the quantization step.

Performance was measured using the cycle counters [11]. Cycle counters provide a very precise tool for measuring the time that elapses between two different points in the execution of a program [3]. The IA-32 counter is accessed with the *rdtsc* (read time stamp counter) assembly instruction. In order to eliminate the effects of context switching and compulsory cache misses, the *K-best* measurement scheme and a *warmed up* cache have been used, as explained in [3].

First, we compare two ways for performing the column DWT. The first way, referred to as *vertical column processing*, processes four consecutive columns completely before advancing to the next four columns. The second way, referred to as *horizontal column processing*, first processes all columns of four rows in SSE-Daub and three rows in MMX-Lifting and then advances to the next set of rows. This was called loop-tiling in [4]. Figure 3 illustrates vertical and horizontal column processing.

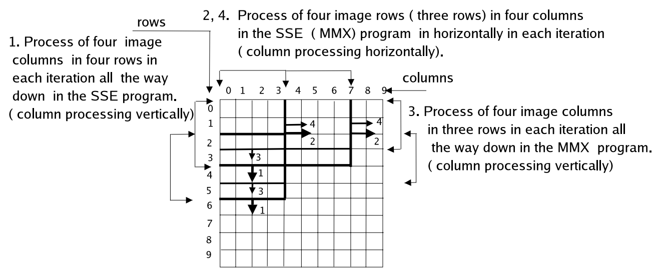
rows
1. Process of four image columns in four rows in each iteration all the way down in the SSE program. (column processing vertically)
2, 4. Process of four image rows (three rows) in four columns in the SSE (MMX) program in horizontally in each iteration (column processing horizontally).
columns
3. Process of four image columns in three rows in each iteration all the way down in the MMX program. (column processing vertically)
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
4
2
4
2
3
1
3
1

Figure 3. Vertical versus horizontal column processing.

Figure 4 depicts the speedup of horizontal column processing over vertical column processing for the C-Lifting program. It can be seen that horizontal column processing is much more efficient than vertical column processing. The reason is that vertical column processing is unable to exploit spatial locality, because the cache blocks have been replaced when the algorithm advances to the next four columns. Horizontal column processing, on the other hand, is able to exploit the spatial locality. Similar behavior has been observed in the C-Daub program. Therefore, from now on all programs use horizontal column processing.

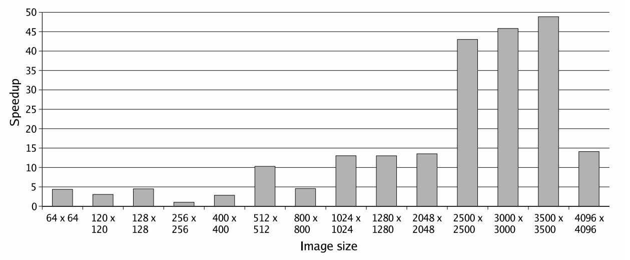

Figure 4. Speedup of horizontal column processing over vertical column processing for the C-Lifting program.

Figures 5 and 6 depict the speedup of the MMX-Lifting and the SSE-Daub implementations of the 1D row DWT over the C-Lifting and C-Daub programs, respectively. MMX-Lifting is up to 3.8x faster than C-Lifting and SSE-Daub is up to 1.9x faster than C-Daub. On average, the speedup of MMX-Lifting over C-Lifting is higher than the speedup of SSE-Daub over C-Daub.

We now consider the 1D column DWT. As explained before, we use horizontal column processing. Figures 7 and 8 show the speedup of the MMX-Lifting and the SSE-Daub implementations of the 1D column DWT over the C-Lifting and C-Daub programs, respectively. For the column DWT,

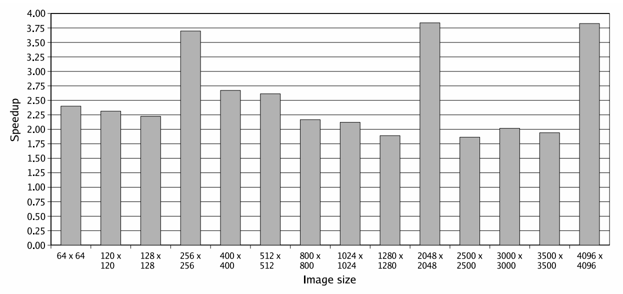

Figure 5. Speedup of MMX-Lifting over C-Lifting for the 1D row DWT.

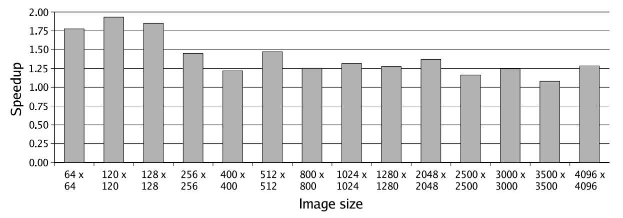

Figure 6. Speedup of SSE-Daub over C-Daub for the 1D row DWT.

MMX-Lifting is up to 4.4x faster than C-Lifting and SSE-Daub is up to 3.7x faster.

Because MMX as well as SSE perform four operations in one instruction, the expected maximum speedup is 4. It can be seen that for some image sizes MMX-Lifting indeed achieves a speedup of 4 (sometimes even higher), while for other image sizes the speedup is between 2 and 3. Similar behavior can be observed for SSE-Daub. For some images sizes (in particular those that are a power of two larger than or equal to 256×256 and 1280×1280) the speedup is higher than 3 while for others it is significantly smaller.

This behavior is due to a design flaw in the Northwood-core Pentium 4 called *64K aliasing* [10]. This problem occurs when two data blocks are accessed whose addresses differ by a multiple of 64K. If this occurs, the cache's associativity is of no use and the effectiveness of the cache is greatly reduced. Let the image size be $N \times M$. Since in column processing `img[i][j]` and `img[i+N/2][j]` are accessed simultaneously, 64K aliasing occurs when NM (resp. $NM/2$) is a multiple of 64K if each pixel is stored as two bytes (resp. four bytes). Because MMX and SSE pack 4 loads in one instruction (in other words, they reduce the

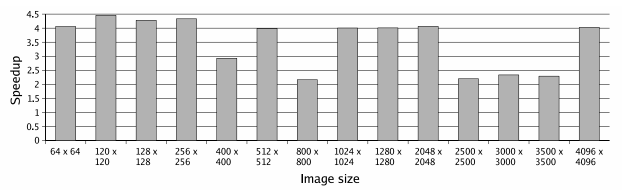

Figure 7. Speedup of MMX-Lifting over C-Lifting for the 1D column DWT.

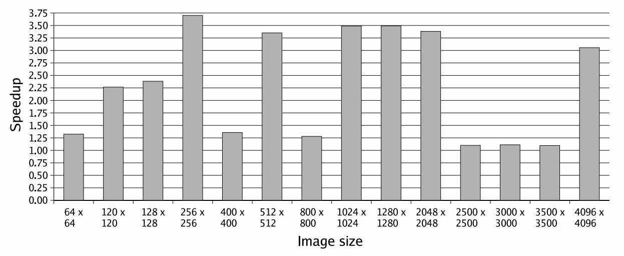

Figure 8. Speedup of SSE-Daub over C-Daub for the 1D column DWT.

number of memory accesses by a factor of 4), the speedup will be approximately 4 when 64K aliasing occurs. If 64K aliasing does not occur, the speedup will be less than 4 due to loop overhead instructions required for managing address and induction variables and branching.

In Figure 7 it can also be seen that for images smaller than 256×256 (when 64K aliasing does not occur), MMX-Lifting achieves a speedup slightly larger than 4. This can be attributed to the fact that the working set of MMX-Lifting is smaller than the working set of C-Lifting, because MMX-Lifting uses two bytes to represent a pixel whereas C-Lifting employs four bytes. MMX-Lifting, therefore, incurs fewer cache misses. But, as remarked before, the C-Lifting variant that uses the `int` data type was found to be faster than the variant that employs `shorts`.

Several related studies (e.g., [5, 6, 2, 8, 4]) mention that vertical filtering requires more time than horizontal filtering because vertical filtering lacks spatial locality. In order to investigate this issue, Figure 9 depicts the speedup of the 1D row DWT (horizontal filtering) over the 1D column DWT (vertical filtering) for MMX-Lifting. For images of size 128×128 and smaller, vertical filtering is faster than horizontal filtering. Furthermore, if 64K aliasing does not occur horizontal filtering is only slightly faster than vertical filtering. This is because vertical filtering can be vectorized more efficiently than horizontal filtering. Because horizontal filtering does not suffer from 64K aliasing, for other image sizes it is much more efficient than vertical filtering.

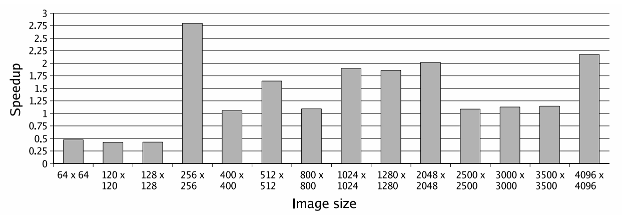

Figure 9. Speedup of the 1D-row DWT over the 1D-column DWT for MMX-Lifting.

Figure 10 compares the performance of all programs. It depicts the speedup attained by MMX-Lifting over the other three programs, for the first level of the 2D DWT (i.e., one row and one column 1D DWT). As can be expected, MMX-

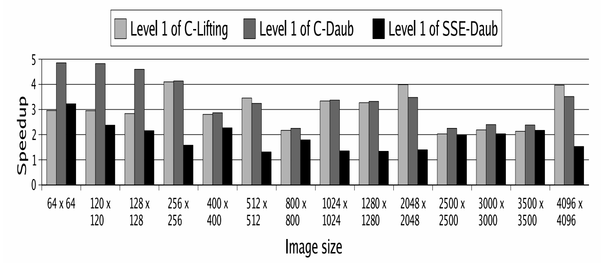

Figure 10. Speedup of MMX-Lifting over the other three programs for the first level of the 2D DWT.

Lifting is faster than all other programs. It is up to 4, 4.8, and 3.2 times faster than C-Lifting, C-Daub, and SSE-Daub, respectively. Furthermore, in most cases SSE-Daub is faster than C-Lifting and C-Lifting is faster than C-Daub.

Figure 11 depicts the speedup of MMX-Lifting over C-Lifting for the first three levels of the 2D DWT. It can be seen that the performance improvement is independent of the decomposition level. In addition, the MMX implementation of the 2D DWT attains a speedup of up to 4 compared to the C implementations, in particular when 64K aliasing occurs. If 64K aliasing does not occur, the speedup is generally smaller.

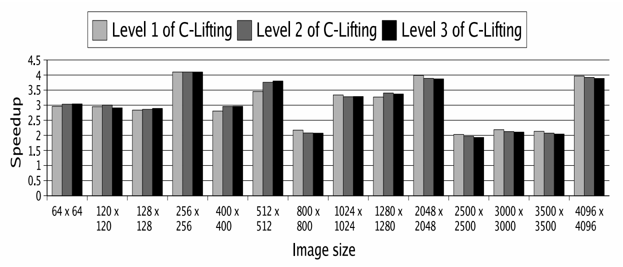

Figure 11. Speedup of MMX-Lifting over C-Lifting for the first three levels of the 2D DWT.

5 Discussion

Although MMX-Lifting is faster than C-Lifting and SSE-Daub is faster than C-Daub, MMX-Lifting achieves the maximum speedup of 4 only if 64K aliasing occurs in which case SSE-Daub attains a speedup higher than 3. If 64K aliasing does not occur, the speedup is significantly smaller (between 2 and 3 for MMX-Lifting and between 1 and 2 for SSE-Daub). In this section we discuss the limitations of MMX and SSE that restrict the performance and discuss possible solutions.

First, as discussed before, overhead instructions that permute subwords in a register are needed in order to vectorize the row DWT. Second, there is a mismatch between the storage and computational format. While image pixels are usually 8 bits, in the MMX implementation they have to be converted to 16-bit values because intermediate results can be larger than 8 bits, and in the SSE implementation they have to be converted to 32-bit floating-point values because the filter coefficients are real numbers. For example, for the first level of MMX-Lifting about 12.7% of the dynamic number of instructions are needed to convert the pixels to 16-bit values. Third, there is misalignment in the 1D row DWT, because the data to be loaded in a 128-bit SSE register is not necessarily stored at a byte address that is a multiple of 16. Although there are SSE instructions that permit unaligned memory accesses, they are much slower than aligned accesses.

In [15] we have evaluated two techniques that can resolve the first two limitations mentioned above. The first technique, called matrix register file, allows row-wise as well as column-wise access to the register file. This is useful for rearranging the subwords as is required for vectorizing the row DWT. The second technique, called extended subwords, uses four bits of extra precision for every byte in a media register. This allows many SIMD operations to be performed without overflow and avoids packing/unpacking conversion overhead because of mismatch between the storage and computational format. In MMX, if the storage format is 8-bit but intermediate results can be larger, data has to be converted to 16-bit, which implies that the 8-way instructions that process 8-bit values cannot be used.

In order to evaluate if extended subwords can also be used to improve the performance of the DWT, we determined the minimum and maximum wavelet coefficient and intermediate result for a 5-level decomposition using the $(5, 3)$ lifting scheme. As input, we employed the well-known "Lena" image as well as randomly generated images with 7 to 10 bits per pixel (bpp). The results are depicted in Table 1. The first column shows the range of the input image pixels, the second and third column the minimum resp. maximum coefficient/intermediate result, and the last column the number of bits required to represent each coefficient and intermediate result. The table shows that a 12-bit data format is sufficient for a 5-level decomposition of images of up to 10 bpp. As future work we, therefore, intend to employ the matrix register file and extended subwords in order to improve the performance of the 2D DWT.

6 Related Work

Chaver et al. [5, 6] implemented Daubechies' $(9, 7)$ real-to-real filter using SSE instructions and measured performance on the Pentium III as well as the Pentium 4. They used the single-precision floating-point format for both image pixels and wavelet coefficients. They focused on the memory hierarchy and considered several techniques such as tiling to improve spatial and temporal locality. Although they considered images whose dimensions are a power of 2, they did not mention the 64K aliasing problem.

Image Data Between	Min. value	Max. value	# bits
0, 127	-238	239	9 bits
-128, 127	-472	477	10 bits
0, 255	-475	478	10 bits
-256, 255	-950	955	11 bits
0, 511	-953	957	11 bits
-512, 511	-1904	1915	12 bits
0, 1023	-1906	1916	12 bits

Table 1. Minimum and maximum wavelet coefficients and intermediate results for a 5-level decomposition using the $(5,3)$ lifting scheme for 7- to 10-bpp images.

Bernabé et al. [2] used SSE to reduce the execution time of the 3D wavelet transform on the Pentium III. They also employed a real-to-real filter (Daub-4) and focused on the memory hierarchy. In these works as well as others [8, 4] it is shown that vertical filtering requires more time than horizontal filtering because vertical filtering lacks spatial locality. However, as shown in this paper, in order to vectorize horizontal filtering the subwords in a media register have to be rearranged, which incurs significant overhead.

Our work differs from others in the following aspects. First, we have not only implemented the Daub-4 real-to-real filter using SSE but also the $(5,3)$ lifting scheme using MMX. Since the fixed-point MMX instructions are generally faster than the floating-point SSE instructions, integer-to-integer transforms are often faster than real-to-real transforms. Second, we have shown that the 64K aliasing problem hampers performance significantly. Third, we have implemented the $(5,3)$ lifting scheme using only addition, subtraction, and shifting operations (i.e., without expensive multiplications). Finally, we have shown that 12 bits are sufficient for performing a 5-level decomposition of images of 7 to 10-bpp.

7 Conclusions

In this paper, we have applied the SSE and MMX instruction set extensions to implement the 2D DWT using the real-to-real Daub-4 transform and the integer-to-integer (5, 3) lifting scheme, respectively. Their execution times on a Pentium 4 have been compared to corresponding C-implementations. The MMX and the SSE codes are up to 4x and 2.6x faster than the corresponding C-programs for a 1-level 2D DWT, respectively. It has been shown that when 64K aliasing occurs the speedups are significantly higher than when it does not occur. This is because with 64K aliasing the programs are entirely memory-bound and MMX and SSE reduce the number of memory accesses by a factor of 4. If 64K aliasing does not occur the processing time is not insignificant but the maximum speedup of 4 cannot be achieved due to overhead required for rearranging data, loop overhead, and due to lack of spatial locality. We have also shown that for the $(5,3)$ filter, a 12-bit word size is sufficient for a 5-level decomposition of the 2D DWT for images of up to 10 bits per pixel. Our future work will focus on considering the 3D wavelet transform and investigating how we can exploit a 12-bit word size.

References

[1] D. M. Adams and F. Kossentini. Reversible Integer-to-Integer Wavelet Transforms for Image Compression: Performance Evaluation and Analysis. *IEEE Trans. on Image Processing*, 9(6):1010–1024, June 2000.

[2] G. Bernabe, J. M. Garcia, and J. Gonzales. Reducing 3D Wavelet Transform Execution Time Through the Streaming SIMD Extensions. In *Proc. 11th Euromicro Conf. on Parallel Distributed and Network-Based Processing*, 2003.

[3] R. E. Bryant and D. R. O'Hallaron. *Computer Systems: A Programmer's Perspective*. Prentice Hall, 2003.

[4] D. Chaver, M. Prieto, L. Piuel, and F. Tirado. Parallel Wavelet Transform for Large Scale Image Processing. In *Proc. IEEE Int. Symp. on Parallel and Distributed Processing*, pages 4–9, 2002.

[5] D. Chaver, C. Tenllado, L. Pinuel, M. Prieto, and F. Tirado. Vectorization of the 2D Wavelet Lifting Transform Using SIMD Extensions. In *Proc. IEEE Int. Symp. on Parallel and Distributed Image Processing and Multimedia*, 2003.

[6] D. Chaver, C. Tenllado, L. Piuel, M. Prieto, and F. Tirado. 2-D Wavelet Transform Enhancement on General-Purpose Microprocessors: Memory Hierarchy and SIMD Parallelism Exploitation. In *Proc. Int. Conf. on High Performance Computing*, December 2002.

[7] M. Farid, F. Kurugollu, and F. Murtagh. Adaptive Wavelet Eye-Gaze Based Video Compression. In *Proc. Int. Society for Optical Engineering (SPIE)*, pages 255–263, 2003.

[8] M. Feil, R. Kutil, P. Meerwald, and A. Uhl. Wavelet Image and Video Coding on Parallel Architectures (Invited Paper). In *Proc. 2nd IEEE - EURASIP Symp. on Image and Signal Processing and Analysis*, 2001.

[9] D. He and W. Zhang. The Parallel Algorithm of 2-D Discrete Wavelet Transform. In *Proc. 4th IEEE Int. Conf. on Parallel and Distributed Computing Applications and Technologies*, pages 738–741, August 2003.

[10] Intel Corporation. *IA-32 Intel Architecture Optimization*, 2004. Order Number: 248966-011.

[11] Intel Corporation. *The IA-32 Intel Architecture Software Developer's Manual Volume 3 System Programming Guide*, 2004. Order Number: 253668.

[12] A. Peleg, S. Wiljie, and U. Weiser. Intel MMX for Multimedia PCs. *Communications of the ACM*, January 1997.

[13] M. Rabbani and R. Joshi. An Overview of the JPEG2000 Still Image Compression Standard. *Signal Processing: Image Communication*, 17(1):3–48, January 2002.

[14] S. K. Raman, V. Pentkovski, and J. Keshava. Implementing Streaming SIMD Extensions on the Pentium 3 Processor. *IEEE Micro*, 20(4):47–57, July - August 2000.

[15] A. Shahbahrami, B. Juurlink, and S. Vassiliadis. Matrix Register File and Extended Subwords: Two Techniques for Embedded Media Processors. In *Proc. 2nd ACM Int. Conf. on Computing Frontiers*, May 2005. To appear.