Reducing 3D Fast Wavelet Transform Execution Time using Blocking and the Streaming SIMD Extensions*

Gregorio Bernabé and José M. García
Universidad de Murcia
Dpto. Ingeniería y Tecnología de Computadores
30071 MURCIA (SPAIN)
gbernabe, jmgarcia@ditec.um.es
Phone: +34 968 367 570
Fax: +34 968 364 151

José González
Intel Barcelona Research Center
Intel Labs, Barcelona
08034 BARCELONA (SPAIN)
pepe.gonzalez@intel.com
Phone: +34 934 137 989
Fax:+34 934 137 755

Abstract

The video compression algorithms based on the 3D wavelet transform obtain excellent compression rates at the expense of huge memory requirements, that drastically affects the execution time of such applications. Its objective is to allow the real-time video compression based on the 3D fast wavelet transform. We show the hardware and software interaction for this multimedia application on a general-purpose processor. First, we mitigate the memory problem by exploiting the memory hierarchy of the processor using several techniques. As for instance, we implement and evaluate the blocking technique. We present two blocking approaches in particular: cube and rectangular, both of which differ in the way the original working set is divided. We also put forward the reuse of previous computations in order to decrease the number of memory accesses and floating point operations. Afterwards, we present several optimizations that cannot be applied by the compiler due to the characteristics of the algorithm. On the one hand, the

*Special Issue on Media and Communication Applications on General Purpose Processors: Hardware and Software Issues / Journal of VLSI SIGNAL PROCESSING SYSTEMS / Dr. Eric Debes, (Lead) Guest Editor. Contact Author: Gregorio Bernabé

Streaming SIMD Extensions (SSE) are used for some of the dimensions of the sequence (y and $time$), to reduce the number of floating point instructions, exploiting Data Level Parallelism. Then, we apply loop unrolling and data prefetching to specific parts of the code. On the other hand, the algorithm is vectorized by columns, allowing the use of SIMD instructions for the y dimension. Results show speedups of $5x$ in the execution time over a version compiled with the maximum optimizations of the Intel C/C++ compiler, maintaining the compression ratio and the video quality (PSNR) of the original encoder based on the 3D wavelet transform. Our experiments also show that, allowing the compiler to perform some of these optimizations (i.e. automatic code vectorization), causes performance slowdown, demonstrating the effectiveness of our optimizations.

Keywords: 3D wavelet transform, video compression, blocking, reuse, Streaming SIMD extensions, vectorization

1 Introduction

In the last few years there has been a considerable increase in the volume of medical images and video generated in hospitals. Medical multimedia information is different from other multimedia data because of its particular properties. There are legal and strict regulations applied to medical multimedia information, since the health of a patient depends on the correctness and the accuracy of this information. Moreover, the integrity, confidentiality and security of medical data is crucial to protect it from accidental or malicious alteration during interchange and storage. Another critical property is that any information on a patient must be available immediately, whenever or wherever, it is required and especially in cases of emergency.

Most of a patient's medical history must be kept and stored as legislation requires all healthcare information to be preserved for a certain period of time (typically 5-10 years) before it can be deleted. Thus, hospitals have to deal with very high storage requirements. On the other hand, tele-diagnosis is becoming a popular technique among hospitals. A doctor may ask for the advice of a colleague who works in another hospital, and even another country, by means of real-time transmission of medical images and video. Due to the huge amount of transmitted data, high-bandwidth networks are needed to maintain the quality of the video and allow for accurate diagnosis. In both cases (storage and transmission), compression techniques are used to drastically reduce the amount of information needed to be handled. Finally, the quality of the compressed data must be good enough to allow for correct

diagnostic when it is reconstructed.

Lately, wavelet transform [16] has been used to acquire the previous features, summarized in the three main areas below:

a) High-quality compression of the medical video.

b) Real-time compression and decompression of the medical video.

c) Real-time transmission of the medical video.

The last problem is outside of the scope of this paper. To find a solution to the first topic, the application of the wavelet transform has been developed drastically on the last few years. The wavelet transform has been applied mainly to image compression. Several coders have been developed using 2D wavelet transform [3][25][35]. Moreover, the last image compression standard, JPEG-2000 [28][34], is also based on the 2D discrete wavelet transform with a dyadic mother wavelet transform.

The 2D wavelet transform has also been used for compressing video [19]. However, three dimensional (3D) compression techniques seem to offer better results than two dimensional (2D) compression techniques that operate in each frame independently. Muraki introduced the idea of using 3D wavelet transform to approximate efficiently 3D volumetric data [29][30]. Since one of the three spatial dimensions can be considered similar to time, a 3D subband coding using the zerotree method (EZW) was presented to code video sequences [11] and posteriorly improved with an embedded wavelet video coder using 3D set partitioning in hierarchical trees (SPIHT) [21]. Today, the standard MPEG-4 [4][5] supports an ad-hoc tool for encoding textures and still images based on a wavelet algorithm.

In previous works [7][8], we have presented the implementation of a lossy encoder for medical video based on the 3D Fast Wavelet Transform (FWT). This encoder achieves both high compression ratios and excellent quality, so that medical doctors can not find longer differences between the original and the reconstructed video.

With regard to the second problem, one of the main drawbacks of using the 3D wavelet transform to code and decode medical video is its excessive execution time. Since three dimensions are exploited to obtain high compression rates, the working set becomes huge and the algorithm becomes limited by memory (memory bound).

In this article, we show the hardware and software interaction for a multimedia application on a general-purpose processor. The manuscript presents a number of approaches to speed up the 3D wavelet transform by reducing

memory bandwidth and exploiting Data Level Parallelism and Instruction Level Parallelism without producing a degradation of the video quality and preserving the compression ratio of the original encoder. Therefore, we present a memory conscious 3D FWT that exploits the memory hierarchy by means of blocking algorithms, reducing the final execution time. We propose and evaluate several blocking approaches that differ in the way that the original working set is divided. We also propose the reuse of some computations to save floating point (FP) operations as well as memory accesses.

Moreover, we attempt to take efficient advantage of the Streaming SIMD Extensions [10] by using the new Intel C/C++ Compiler [15]. We also employ others classic methods like data prefetching and loop unrolling. Finally, we examine the source code to exploit the temporal and spatial locality in the memory cache. A method to enhance the locality of the memory hierarchy, based on the compute of the wavelet transform in the x and y dimensions is presented, taking into account that the mother wavelet function is the Daubechie's of four coefficients (Daub-4).

Results show that the rectangular overlapped approach with the different optimizations provide the best execution times among all tested algorithms, achieving for optimal block size ($512x64x16$) a speedup of 5 over the non-blocking non-overlapped wavelet transform. Therefore, the final proposed approach maintains both the high compression ratio and the excellent video quality of the original encoder [8].

This article is a major revision of two papers published in [9] and [6]. The rest of this article is organized as follows. The background is presented in Section 2. Section 3 describes several approaches to reduce the execution times in the 3D-FWT algorithm in which we will present the main details of each method. In Section 4, we show several techniques using the new Intel C/C++ Compiler and the Streaming SIMD Extensions to reduce the execution times of the rectangular overlapped approach, presented in the previous section. Experimental Results on some test medical video are analyzed in Section 5. Finally, Section 6 summarizes the work and concludes the paper.

2 Background

In this Section, we review the framework on top of which our enhancements have been built. We will first present the general techniques needed to compress medical video and the ways for measuring it. We will then review the theory behind wavelets and finally, introduce the blocking techniques

along with the advanced multimedia extensions.

2.1 Medical Video

There are two ways for compressing medical video: lossy and lossless compression techniques. Higher compression ratios can currently be obtained by means of lossy compression techniques, but radiologist are very reluctant to use them, as they might potentially introduce compression artifacts to ensure complicating diagnosis. Doctors normally prefer to use lossless compression techniques (JPEG-LS [20]) so that the quality is preserved.

However, lossless compression achieves compression ratios significantly lower than those achieved by lossy techniques. Therefore, following the legal rules and keeping medical video for ten years may become prohibitive for most hospitals due to the storage requirements. In addition, the constant increase of network traffic may the use of tele-diagnosis difficult if images are not sufficiently compressed. All of this makes the research on lossy compression techniques particularly interesting, especially, if oriented so as to exploit the behavior of the medical video, usually encoded in gray scale, using just 1 byte per pixel, offering very small interframe variations.

In addition, we need to measure the quality of reconstructed video. A numerical evaluation of the quality is achieved by computing the peak signal-to-noise ratio (PSNR) in the reconstructed video.

The PSNR, of an image, is defined as follows

$$PSNR = 10log_{10}\frac{f_{max}^2}{\alpha^2} \tag{1}$$

where f_{max} stands for the highest possible value of pixel, that is 255 for images that use 8-bits to represent a pixel (e.g. gray-scale images), whereas α^2 stands for the Mean Square Error (MSE).

The $PSNR$ of a reconstructed video has been calculated by computing the arithmetic mean of the $PSNR$ for all of frames of the video. We use $PSNR$ as it is the most simple way of comparing the performance among different schemes.

In addition to the PSNR value, the reconstructed video must be evaluated by doctors. Therefore, it is common practice to visually test the quality of the reconstructed video.

2.2 The Wavelet Transform Foundations

The basic idea behind the wavelet transform is to represent any arbitrary function f as a weighted sum of functions; referred to as wavelets. Each

wavelet is obtained from a mother wavelet function by conveniently scaling and translating it. The result is equivalent to decomposing f into different scale levels (or layers), where each level is then further decomposed with a resolution adapted to that level.

In multiresolution analysis, two functions exist: the mother wavelet and its associated scaling function. Therefore, the wavelet transform can be implemented by quadrature mirror filters (QMF), $G = g(n)$ and $H = h(n)\ n\epsilon Z$. H corresponds to a low-pass filter and G is a high-pass filter. The reconstruction filters have impulse response $h^*(n) = h(1-n)$, and $g^*(n) = g(1-n)$. For a more detailed analysis of the relationship between wavelets and QMF see [27].

The filters H and G correspond to one step in the wavelet decomposition. Given a discrete signal, s, with a length of 2^n, at each stage of the wavelet transformation, the G and H filters are applied to the signal and the filter output downsampled by two, generating two bands: G and H. The process is then repeated on the H band to generate the next level of decomposition and so on. It is important to note that the wavelet decomposition of a set of discrete samples has exactly the same number of samples as in the original, due to the orthogonality of wavelets. This procedure is referred to as the 1D Fast Wavelet Transform (1D-FWT).

The inverse wavelet transform can be obtained in a way similar to that of the forward transform, by simply reversing the above procedure following. However, the order of the g's and h's has to be reversed.

It is not difficult to generalize the one-dimensional wavelet transform to the multi-dimensional case [27]. The wavelet representation of an image, $f(x, y)$, can be obtained with a pyramid algorithm. It can be achieved by first applying the 1D-FWT to each row of the image and then to each column. That is, the G and H filters are applied to the image in both the horizontal and vertical directions. The process is repeated several times as in the one-dimensional case. This procedure is referred to as the 2D Fast Wavelet Transform (2D-FWT).

As in 2D, we can generalize the one-dimensional wavelet transform for the three-dimensional case. Instead of one image, there is now a sequence of images. Thus a new dimension has emerged, time (t). The 3D-FWT can be computed by successively applying the 1D wavelet transform to the value of the pixels in each dimension.

It is common in wavelet compression to recursively transform the average signal. The number of transformations performed in each dimension depends on several factors, for example the amount of compression desired, the size of the original video and the mother wavelet function. In general, the higher the

desired compression ratio, the more times the transform is performed. Note that applying the wavelet transform too many times, may have a significant impact on quality. Hence, this parameter must be chosen carefully.

In this paper, we have considered Daubechies W_4 (Daub-4) [16] as the mother wavelet function. We have chosen this function because some previous works have proved its effectiveness [7][8].

2.3 Blocking and Streaming SIMD Extensions (SSE)

Blocking is a well-known optimization technique for improving the effectiveness of memory hierarchies [1][23][26]. Instead of operating on entire rows, columns or frames of the working set, blocking algorithms operate on working subsets or blocks, so that data loaded into the faster levels of the memory hierarchy is reused. Blocking has been shown to be useful for many algorithms in linear algebra like BLAS [17], LAPACK [2] and most recently, ATLAS [38]. Blocking has also been used for the computation of the 2D and 3D wavelet transform, splitting the image or video in several blocks and then perform the transform on tiles such as the standard image compression JPEG-2000 [28] which uses a line based wavelet transform [12] or the reduced memory versions of the embedded wavelet video coder using 3D set partitioning in hierarchical trees (SPIHT) [22] and the 3D scan-based wavelet transform [32]. SPIHT avoids the blocking artifacts in the block bounds at the expense of some extra processing of pixels overlapped of the following blocks whereas the 3D scan method allows the computation of the temporal wavelet decomposition duplicating the input frames by a symmetrical extension. Our technique uses the overlapped approach as we will present in the next section.

The introduction of Multimedia Extensions (MMX™ Technology) [24] and the Streaming SIMD Extensions (SSE) [14] available on modern processors, provide a technology designed to accelerate multimedia and communications software, able to reduce the execution time of the applications. Ranganathan et al. [33] show the Sun VIS media ISA extensions provide an additional 1.1 to 4.2 performance improvement over several image and video processing applications. Nachtergaele et al. [31], proposed a software implementation of the MPEG-4 based on the integer wavelet transform using Multimedia Extensions. Conte et al. [13] evaluated several applications obtaining substantial speed-ups with MMX/SSE code.

™MMX is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

The Pentium III processor introduced the 128-bit streaming SIMD extensions [37], which support floating-point operations on 4 single-precision floating-point numbers, implemented through of eight 128-bit data registers, called xmm0, xmm1, ..., xmm7.

Two options are available when carrying out these extensions: by means of an adequate compiler (automatic vectorization) or by hand. The Intel C/C++ Compiler for Linux (v5.0.1) [15], follows the standard approach to the vectorization of inner loops [39]. First of all, statements in a loop are reordered according to a topological sort of the acyclic condensation of the data dependence graph for this loop. Statements involved in a data dependence cycle are then either recognized as certain idioms that can be vectorized or distributed out into a loop that will remain serial. Finally, vectorizable loops are translated into SIMD instructions.

However, automatic vectorization is still difficult to achieve due to the high restrictions imposed by compilers and the nature of the algorithm of the wavelet transform. Instead, and as will be shown in Section 4, we have manually vectorized the code, as it was simple and more effective than giving hints to help the compiler.

3 Blocking the Wavelet Transform

Our previous Wavelet-based encoder obtained excellent results both in compression rate and quality (PSNR), as observed in [7][8]. Results were obtained with the 3D-FWT working on video sequences of 64 frames of $512x512$ pixels (16 MBytes of working set). This huge working set limits the performance of such algorithm making it unfeasible for real-time video compression and transmission. Initial results showed that this algorithm is completely memory bound, therefore, blocking techniques could be an interesting approach to reduce its memory requirements and consequently the execution time.

The aim behind blocking algorithms, is to exploit the locality exhibited by memory references by means of partitioning the initial working set in limited chunks that fit into the different levels of the memory hierarchy. In this way, two positive effects appear: on the one hand, memory accesses are accelerated, since data is actually at the higher levels of the memory hierarchy (closer to the processor core). On the other hand, traffic between the main memory and the processor chip is drastically reduced, obtaining better use of the bandwidth provided by the baseline computer system.

However, applying blocking algorithms to video coders is a challenge;

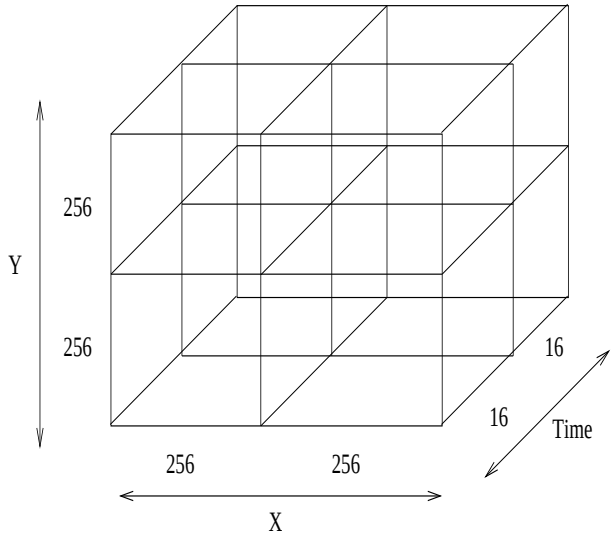

Figure 1: Cube approach

not only the memory hierarchy must be exploited by means of an optimum data partitioning but quality must also be preserved. Note that partitioning the working set into independent blocks may lead to unexpected reductions to the quality of the resulting video due to artifacts in the block bounds.

In this section, we present two different approaches to the blocking version of the 3D-FWT transform: cube and rectangular, both of which differ in the way the original working set is divided.

3.1 Cube approach

In this first approach, we propose to divide the original sequence. For example, a video sequence of 64 frames of $512x512$ pixels is split into several subcubes as we can see in figure 1 and the wavelet transform is independently applied to each of these subcubes. With regards to the size of these subcubes, X and Y axis are the same size (different block sizes have been evaluated), whereas the number of frames in the time dimension is fixed to 16, the minimum number of frames needed to apply the transform twice.

However, this approach has two disadvantages. In the first instance, as the compression ratio increases, the Peak Signal to Noise Ratio (PSNR) drops significantly; in the second, an increasing degree of visibility in the discontinuity of the reconstruction at adjacent subcubes boundaries is detected because artifacts effects appear. This is due to the way the computation is

```
/* c0, c1, c2, c3: Daub-4 coefficients */
/* pixels 1..8 = p[0..7] */
/* temporal vector: low-pass */
float low[8], high[8];
n = 8;
for(i = 0, j = 0; j < (n/2) − 1; i+ = 2, j + +) {


        low[j]=c0*p[i]+c1*p[i+1]+c2*p[i+2]+c3*p[i+3];
          high[j+n/2]=c3*p[i]-c2*p[i+1]+c1*p[i+2]-c0*p[i+3];

}
low[j]=c0*p[n-2]+c1*p[n-1]+c2*p[0]+c3*p[1];
high[j+n/2]=c3*p[n-2]-c2*p[n-1]+c1*p[0]-c0*p[1];
```

Figure 2: Algorithm of 1D-FWT with Daub-4

performed in the FWT, since for a particular pixel, the value of its coefficient after the transform is correlated with the original values of its neighboring pixels.

To illustrate this problem, Figure 2 shows how the wavelet transform is applied for an unidimensional signal of 8 pixels using the Daubechie's of four coefficients as a mother function (Daub-4). This signal is divided into two blocks of 4 pixels where the FWT is computed independently. The resulting coefficient for the first pixel depends on the second, third, fourth and itself, all belonging to the same block. However, the second pixel depends on the third, fourth, fifth and the sixth pixel (the last two pixels belong to a different block unavailable in this original partitioning). The same happens to the rest of the pixels. Since additional pixels are needed to compute the transform in any dimension, two different alternatives can be considered to provide this information. $Non - Overlapped$ approaches utilize pixels from the same block (for instance replicating last pixels, or using first pixels). *Overlapped* approaches use pixels from the following block. Although the latter does not seem to exploit memory locality, it provides better compression and better quality results as we will demonstrate later on.

Furthermore, the 3D-FWT implies the computation of the 1D-FWT in the time dimension. When following the aforementioned approach, information from additional frames is needed, and can be obtained from the block itself or from the following blocks. The amount of frames depends on the

number of steps of wavelet transform. Taking as an example the W_4 mother wavelet, applying the wavelet transform just once needs two more frames, six frames are necessary for two wavelet transforms, and fourteen frames are needed for three wavelet transforms.

Thus, choosing between the *overlapped* and $non-overlapped$ approaches for the 3D-wavelet transform is one of the main decisions that must be taken to achieve a good trade-off between execution time and quality. Whereas the $non-overlapped$ approach seems more memory efficient, since computations are carried out using the working set of the block, the quality of the reconstructed video is clearly affected by the artifacts that appear in the block bounds. This is because the coefficients of the block bounds are computed without taking into account their neighbors.

Subsequently, to avoid the artifacts caused by discontinuities in any reconstruction between adjacent coding subcubes, the X, Y and time axis are overlapped. We refer to this cube modified approach as *cube overlapped*. Since the FWT is applied twice, six rows, six columns and six frames must be overlapped (e.g. for subcubes of 256 rows-columns of 16 frames, subcubes of 262 rows-columns of 22 frames are now needed).

3.2 Rectangular approach

The 3D-FWT algorithm is programmed in C and frames are thus stored in the memory according to row order. For the space locality of memory references to be better exploited, it might be interesting to analyze a different data distribution. In this section, we present the rectangular partitioning; where the original cube is divided into several rectangles, as we can observe in figure 3.

The $overlapped\ wavelet\ transform$ as in the cube approach can be applied to avoid the artifacts and the decrease of PSNR, however, only Y and time dimensions are overlapped. For example, a video sequence of 64 frames of $512x512$ pixels can be divided into 8 rectangles of 16 frames of $512x256$ or 32 rectangles of 16 frames of $512x128$ pixels. After overlapping, rectangles of 22 frames of $512x262$ pixels or 22 frames of $512x134$ pixels are obtained.

4 Optimizing the Rectangular Overlapped Approach

In this Section, several enhancements to the original blocking algorithms are illustrated, with the aim of reducing both the number of FP instructions and the pressure on the memory subsystem.

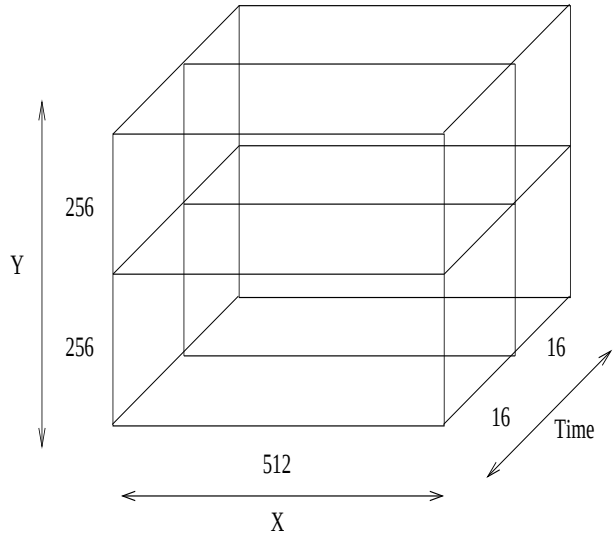

Figure 3: Rectangular approach

4.1 Reuse Operations

In the latter approach, we found another contribution of this work: the reuse of some computations to reduce the number of floating point operations and memory accesses. When the overlapped wavelet transform is used, operations are repeated across different blocks. For example, for the previous video sequence, if divided into 8 rectangles of 16 frames of $512x256$ pixels, 6 rows and 6 frames must be overlapped in the first rectangle. When the first wavelet transform is applied to the Y dimension, 130 low and 130 high rows are obtained. The last two low and high rows are the first ones in the next rectangle, so they should not be computed again in the following block. As seen in figure 4, some computations carried out for the first block are reused for the second block. For instance, if we divide into several rectangles of 16 frames of $512x32$ or $512x16$ pixels, 12% and 25% of the operations will be reused respectively in the Y dimension.

4.2 SSE Extensions for the Wavelet Transform

The SSE extensions are used to exploit fine-grained parallelism by vectorizing loops that perform a single operation on multiple elements in a data set. Therefore, we can apply the SSE in our wavelet overlapped transform algorithm for an unidimensional signal (1D-FWT) of n pixels with the Daub-4

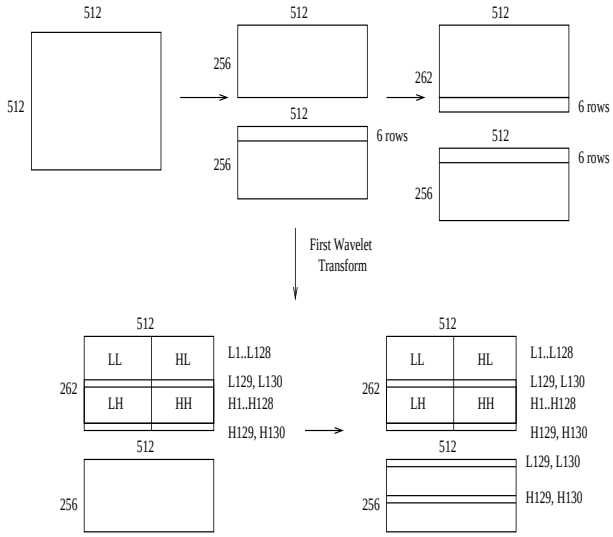

Figure 4: Reuse in Rectangular approach

as the mother wavelet function.

As we can be observed in Figure 2, the value of each resulting wavelet coefficient depends on four pixels, and 8 floating point multiplications and 6 floating points additions are needed to obtain the low and high pass for each pixel. For 4 coefficients, 32 floating point multiplications and 24 floating points additions are then necessary.

Figure 5 shows the computation of the first four low-pass resulting wavelet coefficients. We refer to this optimization as SSE vectorization by hand. First, four SSE registers (xmm0, xmm1, xmm2 and xmm3) are initialized with the Daub-4 coefficients. Second, the pixels are loaded in groups of four into the SSE registers (xmm4, xmm5, xmm6 and xmm7). Finally 4 floating point multiplications and 3 floating point additions, are performed among the SSE registers to obtain the same wavelet coefficients as in the algorithm of 1D-FWT overlapped with Daub-4. We can obtain the high-pass wavelet coefficients in the same way; with 4 floating point multiplications and 3 more floating point additions. Therefore, the total number of floating point instructions has been reduced from 56 to 15 instructions[2].

[2]Each instruction contains operations that are executed in parallel

XMM0	C0	C0	C0	C0	_mm_set_ps(C0, C0, C0, C0)
XMM1	C1	C1	C1	C1	_mm_set_ps(C1, C1, C1, C1)
XMM2	C2	C2	C2	C2	_mm_set_ps(C2, C2, C2, C2)
XMM3	C3	C3	C3	C3	_mm_set_ps(C3, C3, C3, C3)

a) Initialize SSE registers with Daub−4 coefficients.

XMM4	p[0]	p[2]	p[4]	p[6]	_mm_set_ps(p[6], p[4], p[2], p[0])
XMM5	p[1]	p[3]	p[5]	p[7]	_mm_set_ps(p[7], p[5], p[3], p[1])
XMM6	p[2]	p[4]	p[6]	p[8]	_mm_set_ps(p[8], p[6], p[4], p[2])
XMM7	p[3]	p[5]	p[7]	p[9]	_mm_set_ps(p[9], p[7], p[5], p[3])

b) Load pixels in group of 4 into the SSE registers.

XMM0	C0*p[0]	C0*p[2]	C0*p[4]	C0*p[6]	mulps xmm0, xmm4
XMM1	C1*p[1]	C1*p[3]	C1*p[5]	C1*p[7]	mulps xmm1, xmm5
XMM2	C2*p[2]	C2*p[4]	C2*p[6]	C2*p[8]	mulps xmm2, xmm6
XMM3	C3*p[3]	C3*p[5]	C3*p[7]	C3*p[9]	mulps xmm3, xmm7

c) Floating point multiplications among SSE registers.

XMM0	C0*p[0] + C1*p[1]	C0*p[2] + C1*p[3]	C0*p[4] + C1*p[5]	C0*p[6] + C1*p[7]	addps xmm0, xmm1
XMM0	C0*p[0] + C1*p[1] + C2*p[2]	C0*p[2] + C1*p[3] + C2*p[4]	C0*p[4] + C1*p[5] + C2*p[6]	C0*p[6] + C1*p[7] + C2*p[8]	addps xmm0, xmm2
XMM0	C0*p[0] + C1*p[1] + C2*p[2] + C3*p[3]	C0*p[2] + C1*p[3] + C2*p[4] + C3*p[5]	C0*p[4] + C1*p[5] + C2*p[6] + C3*p[7]	C0*p[6] + C1*p[7] + C2*p[8] + C3*p[9]	addps xmm0, xmm3

d) Floating point additions among SSE registers.

Figure 5: Phases for the computation of the first four low-pass wavelet coefficients with the SSE registers

4.3 Loop Unrolling and Prefetching Data

Loop unrolling is usually applied by the compiler if there is a clear room for improvement. However, due to the nature of the Wavelet algorithm (3 nested loops for the time dimension) and the compiler constraints, we have had to unroll the time dimension manually. In this dimension, if the wavelet transform is applied twice, the first iteration will be applied over 22 frames and the second iteration over 10 frames (for blocks of 16 frames).

Therefore, the loop is unrolled for the time dimension, because when using the SSE, the loop is only executed three times in the first iteration (4 low-pass and 4 high-pass coefficients are calculated in each time) and once

more in the second.

Another feasible optimization is data prefetching which improves the performance due to accelerated data delivery. In this way, data prefetching can, in part, hide the memory latency. If we predict which memory page our program will request next, we can fetch that page into cache (if it is not already in cache) before the program asks for it. In our wavelet transform algorithms, it is necessary to reference a lot of data and we can predict what are the next data in order to drop down the latency.

4.4 Columns Vectorization

In the 3D-FWT, the wavelet is applied in the x, y and *time* dimensions. In previous subsections, we analyzed the time dimension and applied the SSE vectorization by hand, loop unrolling and data prefetching. In the x dimension, the wavelet transform is applied successively for all rows of each frame. As the video sequence is stored in the memory according to a row order, spatial locality is exploited when the transform is applied in this dimension. The main problem with memory appears when the transform is applied in the y dimension. Pixels from successive rows are needed to compute the coefficients of each column of the y dimension, causing many cache misses even for the blocking version of the algorithm (for this version, L1 data cache still presents a high number of misses).

In this Section, we give "columns vectorization", as an effective way to apply the transform in the y dimension, exploiting the locality of references and the fact that the transform was already applied in the x dimension.

As the wavelet transform is applied by rows in the x dimension, to compute the first coefficient in the y dimension, only the resulting wavelet coefficients of the first four rows are needed, since each coefficient of the Daub-4 mother function depends on four pixels, as we can observe in Figure 2. To compute a new row in the y dimension, two more rows of wavelet coefficients in the x dimension are needed.

Figure 6 is an example of a piece of frame of 6 rows by 12 columns. Once the wavelet transform is applied for the four first rows in the x dimension, it can be applied to the first row in the y dimension (i.e. in order to compute a coefficient, values obtained for rows 0,1,2,3 are needed). Furthermore, this computation is carried out using SSE extensions (4 coefficients fit in a XMM register). The second row in the y dimension depends on rows 2, 3, 4 and 5. Therefore, only two new rows in the x dimension are necessary.

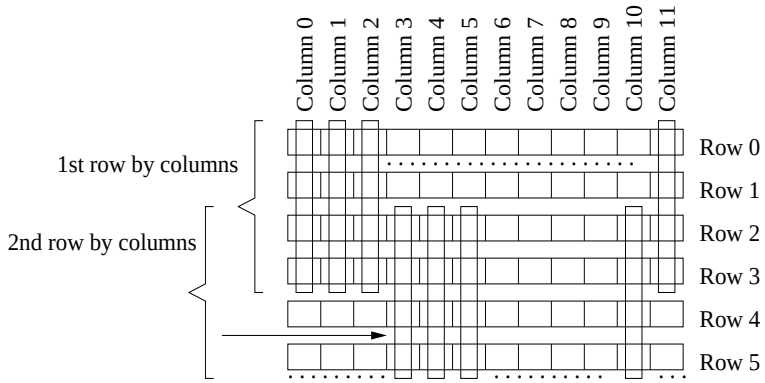

Figure 6: Columns vectorization

TLBs	L1 instr TLB, 4K page, 4-way, 32 entries L1 data TLB, 4K page, 4-way, 64 entries
Level 1	L1 instr cache, 16 KB, 4-way, 32 byte line L1 data cache, 16 KB, 4-way, 32 byte line
Level 2	L2 unified cache, 256 KB, 8-way, 32 byte line
Level 3	512 Mbytes DRAM

Table 1: Description of the memory hierarchy

5 Experimental Results

5.1 Workbench Environment

The evaluation has been carried out on a 1GHz-Intel Pentium-III processor with 512 Mbytes of RAM. The main properties of the memory hierarchy are summarized in table 1. The operating system used was Linux 2.2.14. The programs have been written in the C programming language.

Performance has been measured using the monitoring counters available in the P6 processor family. The Intel Pentium-series processors include a 64-bit cycle counter and two 40-bit event counters with a list of events and additional semantics dependent on the particular processor. We have used a library, Rabbit (v.2.0.1) [18], to read and manipulate Intel processor hardware event counters in C under the Linux operating system.

We compared execution time consumed by the 3D-wavelet transform for the different blocking approaches proposed in section 3, with the original 3D-FWT lossy compression method [7], and the different optimizations

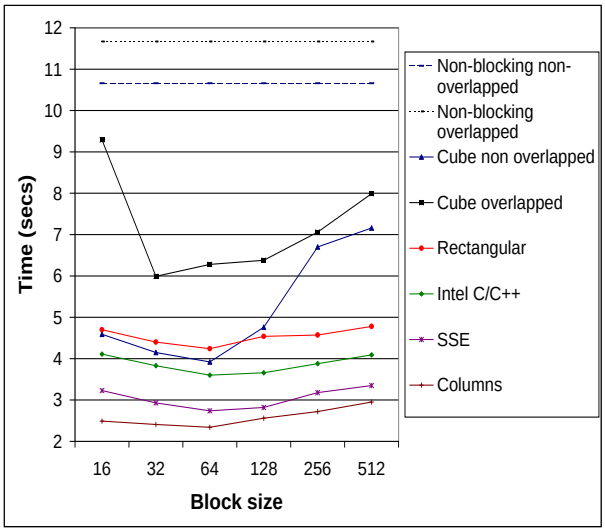

Figure 7: Execution Time for the different approaches for the Heart video sequence

presented in section 4 on a *heart* video medical sequence of 64 frames of $512x512$ pixels coded in gray scale (8 bits per pixel).

5.2 Evaluating the Execution Time

Figure 7 shows the execution time obtained with two levels of the fast wavelet transform to compute 64 frames of $512x512$ pixels and for the different blocking approaches: cube non-overlapped, cube overlapped and rectangular overlapped (*Rectangular*) compiled with the gcc/gnu compiler. *Intel C/C++* represents the same blocking rectangular approach compiled with the Intel Compiler [15]. *SSE* includes SSE vectorization by hand as well as loop unrolling and data prefetching, all of them for the time dimension. Finally, *Columns* includes Columns Vectorization and the SSE vectorization by hand in the computation of wavelet coefficients for the y dimension. Results are presented according to different block sizes, from $16x16x16$ to $512x512x16$ in the cube approaches and from $512x16x16$ to $512x512x16$ in the other approaches and optimizations. We have also included the execution time without blocking for reference, using the non-overlapped and the overlapped wavelet transform, plotted as dotted lines, taking the same execution time for all configurations because there are not divisions into different blocks.

First of all, we can observe that blocking approaches clearly reduce the

execution time of the original algorithm for all configurations. The optimal block size in the cube non-overlapped approach ($64x64x16$) obtains a speedup of 2.71 over the original non-overlapped wavelet transform, whereas overlapped blocking approaches, cube (optimal block size $32x32x16$) and rectangular (optimal block size $512x64x16$), provide a speedup of 1.77 and 2.42 respectively, compared to the non-overlapped wavelet transform.

As we expected, the rectangular approach obtains the best results among the different blocking approaches. This behavior is due to the better exploitation of the locality of its memory accesses and the reuse of floating point operations. This reuse improves the execution time over an average of 6% for all configurations.

Higher execution times on overlapped blocking approaches compared to the non-overlapped ones are caused by the increase of the working set of blocks since data from the following blocks must be incorporated. However, these overlapped approaches obtain much better quality, which makes them more attractive to be used.

For instance, in the rectangular approach, the optimal configuration is $512x64x16$, obtaining a speedup of 1.48 over the $64x64x16$ in the cube overlapped approach. In some configurations ($16x16x16$, $32x32x16$ and $64x64x16$), the cube non-overlapped approach obtains faster times than the rectangular approach. However this approach presents artifacts and a decrease of the PSNR (from 37 to 33) in the reconstructed video, that it then discards to obtain a higher quality compression of the medical video.

In summary, overlapped approaches maintain the compression rate and quality of the video whereas the non-overlapped approach produces an unacceptable degree of visibility in the reconstructed video.

As for the blocking overlapped approaches, the rectangular approach better exploits the memory hierarchy than the cube and consequently the execution time is significantly reduced. Rectangular or cube blocking achieve execution times 12% ($512x64x16$) and 33% ($32x32x16$) faster than blocks of $512x512x16$.

5.3 Analyzing the Effects of further Optimizations

Furthermore, it can be observed that each new optimization clearly reduces the execution time of previous approaches for all configurations. The optimal block size ($512x64x16$) is maintained in all approaches. For this block size, the version just compiled with the Intel C/C++ obtains a speedup of 1.18 to that the one compiled with gnu/gcc. From this point on, we will refer to the Intel C/C++ version as the baseline, since it represents our

previous proposal, re-compiled with a better compiler. Combining SSE extensions with prefetching and loop unrolling, obtain a speedup of 1.31 for the baseline and a speedup of 1.54 for the Columns vectorization. It is important to note that all of these optimizations in the algorithm, maintain the same compression rate and quality as the rectangular overlapped approach confirming the potential of these methods.

The results in Intel C/C++ are obtained with the $-tpp6$ options which generates a code optimized for Pentium III processors and the advantages of the new compiler, improving the original execution time. In addition, we have enabled the automatic vectorization with the $-xK$ and $-axK$ options, generating a code specialized for Streaming SIMD extensions. Although the execution times are better than those of the original rectangular overlapped approach (i.e. that compiled with gnu/gcc), they are worse than without automatic vectorization for the Intel C/C++ compiler. The reason for the decrease in performance experienced with automatic vectorization is that the vectorization of the Wavelet Transform is tricky, i.e. it has to be carefully applied to the computations that could obtain benefit from it. Remember that there are three nested loops and that, for instance, vectorizing the innermost loop does not provide any benefit. Thus, manually vectorizing the code, as proposed in this work is, so far, the best option for achieving benefits when SIMD extensions are applied in the Wavelet Transform.

In SSE optimizations, to achieve performance benefits there are three different ways: first, the utilizations of SSE extensions for the time dimension, second, the effect of loop unrolling to increase Instruction Level Parallelism, and third, the effect of data prefetching. At the same time as the four wavelet coefficients are being calculated, pixels needed for the next coefficients are being prefetched.

Finally, with *Columns* optimization, execution times are significantly reduced for all configurations. This optimization allows the real-time video compression and transmission (24 frames per second) for all but the block sizes $512x512x16$ and $512x256x16$. The configurations from block sizes $512x16x16$ to $512x128x16$ obtain 25.7, 26.6, 27.4 and 25.0 frames per second respectively. This is due to better exploitation of the temporal and spatial locality of the cache memory by the columns vectorization. We also manually vectorized the computations of the y dimension without the *Column* optimization (i.e coefficients in the y dimension are computed after the wavelet has been completed in the x dimension). This optimization does not provide any performance benefit thus, to obtain improvement from SSE extensions in the y dimension, the code reordering that we propose in this work must be carried out.

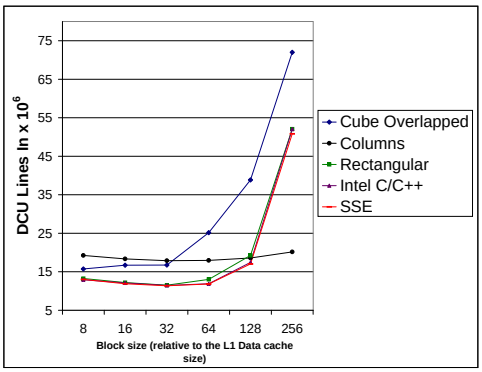

Figure 8: DCU Lines In for Heart video sequence

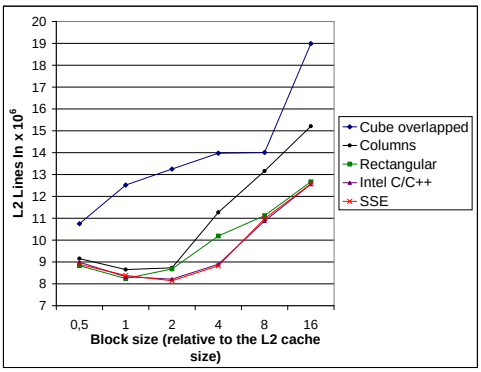

Figure 9: L2 Lines In for Heart video sequence

5.4 Analyzing the Memory Cache Behavior

To gain some insight into the speedups obtained by the previous approaches, Figures 8 and 9 present the memory cache behavior for the heart video sequence. This behavior is measured using Data Cache Unit (DCU) Lines In and L2 Lines In events of the performance counters, which represent the number of lines allocated in the L1 Data Cache and the L2 cache respectively (i.e. the number of accesses that miss in each cache respectively). The block size is presented as a number of times the size of each cache.

The trend of the curve illustrating the rectangular overlapped approach, is quite simple; the smaller the block, the lesser the misses in the L1 Data and L2 Cache, until a certain block size. The other approaches also produce very similar curves; a blocking factor of 32 the size of the L1 Data or two times the L2 Cache size, are the best configurations, that is, to say an optimal block size of $512x64x16$. Smaller block sizes do not improve the misses because when the block size decrease, the overlapped wavelet transform needs a higher number of pixels in the following blocks implying an overhead of data in the L1 Data and L2 Cache. In figures 10 and 11, we can observe that the DCU Lines In and L2 Lines In belonging to the rectangular overlapped approach divided by axis for the first iteration of the wavelet transform. In both cases, the time and y axis show the same behavior of all the misses in the L1 Data and L2 Cache, while in the x axis, the smaller the block, the higher misses in the L1 Data and L2 Cache by the overhead of the overlapped wavelet transform in the smaller blocks. This confirms that smaller block sizes that fit into the caches, reduce the number of L1 and L2 misses until

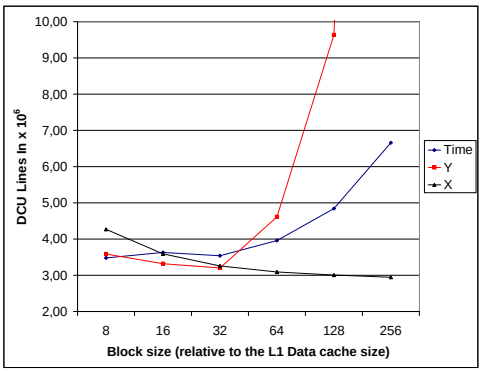

Figure 10: DCU Lines In divided by axis of the Rectangular Overlapped Approach

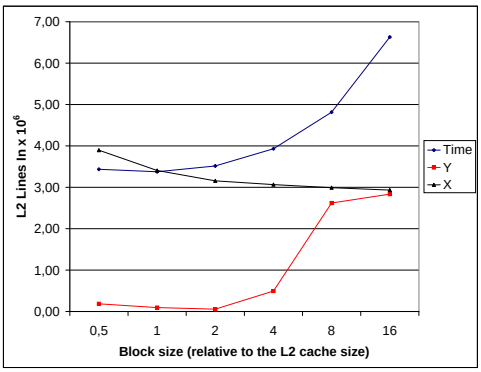

Figure 11: L2 Lines In divided by axis of the Rectangular Overlapped Approach

a configuration where the overhead of pixels of the following blocks become very important, implying a higher number of misses in L1 Data and L2 Cache than higher block size configurations.

It can be observed that the rectangular approach allocates a lesser number of L1 and L2 lines than the cube overlapped approach justifying the decrease in the execution time in the blocking approaches. Remember that data is stored by rows, since the rectangular approach retains more coefficients in a row than the cube approach; spatial locality is better exploited and the number of compulsory misses is drastically reduced.

The *Intel* $C/C++$ approach also allocates less L1 and L2 lines than the rectangular overlapped approach for all configurations, justifying the decrease in the execution time. With respect to SSE vectorization by hand, we observe that in most configurations, this approach produces less L1 and L2 misses than the Intel C/C++ but the difference is not very significant. Note that the main benefit provided by SSE optimizations comes from the reduction in the number of Floating Point Instructions, as we can observe in Figure 12, due to manual vectorization. Applying SSE extensions does not reduce the overall number of FP operations, which only depends on the algorithm, but it does reduce the number of FP instructions, since operations are performed in parallel in single SIMD instructions. Thus, what we are exploiting is *Data Level Parallelism*. Furthermore, the benefit provided by data prefetching cannot be measured in the number of L1 or L2 misses, as prefetching instructions do cause cache misses. However data is prefetched enough in advance so that misses do not cause dependent instructions to

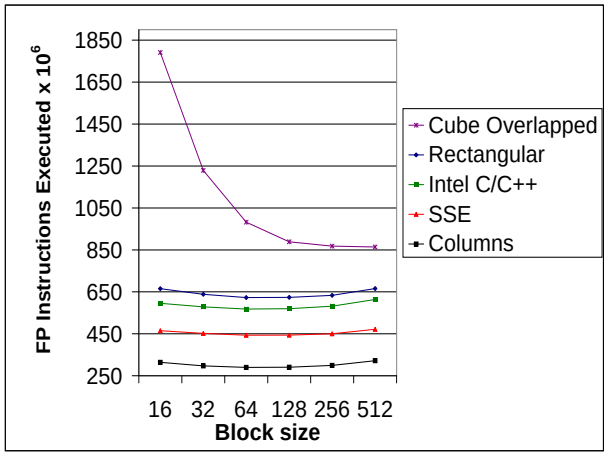

Figure 12: Floating point operations executed for the Heart video sequence

wait on the processor.

Finally, in the Columns approach, there is a significant increase in the number of lines allocated in L1 and L2 caches compared with previous approaches. Although the columns vectorization better exploits the spatial and temporal locality for the calculation of the x and y dimension. This increase in L1 and L2 misses is due to an implementation issue. When columns vectorization is applied, two rows are generated for the y dimension, a low-pass and a high-pass. High pass coefficients must be saved in another space different to the frame itself so as not to delete the original pixels, already required for the rest of the x computations. This increases the number of memory lines used for the transform (the original ones, plus those needed for the temporal location of high-pass coefficients). Also, because of data movements back and forth to temporal locations, locality is not so exploited, affecting the end performance of memory operations.

However, taking into account this problem of the memory instructions, the execution time have been drastically reduced for all configurations. This reduction is due to two reasons. First, since this optimization is built on top of the previous ones (the original blocking, prefetching and so on), the original 3D-FWT is not as memory bound even with the latter's memory "inefficiency". Second, since the algorithm is not memory bound and following the Amdahl Law, any optimization in the computation side has a great impact on performance. Note that with the *Columns* approach, the number of FP instructions executed have dropped spectacularly, 71%, 53%, 48% and 34% regarding to Cube Overlapped, Rectangular compiled with

gnu/gcc, rectangular compiled with Intel C/C++ and SSE vectorization by hand respectively, as we can see in Figure 12. This reduction in the number of instructions occurs again by exploiting Data Level Parallelism, achieved by manually vectorizing computation in the y dimension.

Finally, it can be observed that the behavior of FP instructions executed in the Cube Overlapped approach is very different to that of the other approaches in the configurations. This is due to the division of the original sequence into subcubes implying an increase in FP instructions executed when the block sizes drop down, because the overlapped wavelet transform needs rows, columns and frames of the following blocks implying the compute of a higher number of operations in the smaller blocks. Therefore, it is very important the reuse of floating point operations introduced by the rectangular approach, which decreases the number of FP instructions executed for all configurations from 2% to 20%. This reuse also helps maintain a similar number of FP operations for all configurations.

6 Conclusions

In this work, we have focused on reducing the execution time of the 3D-Fast Wavelet Transform when it is applied to code medical video. We have presented six proposals. First, we have developed and evaluated several blocking algorithms to exploit the memory hierarchy. Second, we have proposed the reuse of computations to decrease the number of floating point operations and memory accesses. Third, we have proposed and evaluated the automatic and SSE vectorization by hand, that exploits Data Level Parallelism by collapsing FP operations on single SIMD instructions. We have showed that the native compiler, Intel C/C++, is not able to obtain performance benefits through automatic optimizations and we have proposed several modifications on the algorithm that provide significant benefits by vectorizing computations in the y and $time$ dimensions. Fifth, we have manually unrolled the time dimension loop and inserted prefetching instructions, both to reduce the impact of cache misses and exploit Instruction Level Parallelism. Sixth, we have proposed and evaluated the columns vectorization in the y dimension, in order to reduce the floating point instructions and the memory accesses exploiting the spatial and temporal locality of the memory hierarchy.

Results have showed that the columns vectorization approach, which includes the different optimizations, has obtained the best results, achieving a speedup of 5 for optimal block size ($512x64x16$) over the non-blocking

non-overlapped wavelet transform, 2.68 over the optimal ($64x64x16$) cube overlapped approach, 1.81 over the rectangular overlapped wavelet transform (compiled with gnu/gcc), 1.54 compared to the rectangular overlapped wavelet transform compiled with the Intel C/C++ compiler and 1.17 with respect to SSE vectorization by hand. Furthermore, all the approaches presented maintain the video quality and the compression ratio of the original encoder. Finally, the execution time achieved via the columns vectorization approach allows for real-time video compression and transmission.

Furthermore, the presented techniques could be generalized to other multimedia applications based on the computation of a transform such as JPEG-2000 [28], MPEG-2 [36] (based on the DCT transform) or MPEG-4 [4][5] on general-purpose processors. All transforms follow a common computation model. Therefore, the memory bandwidth requirement could be reduced through exploiting the memory hierarchy, the Data Level Parallelism could be exploited by the multimedia extensions available in all current general-purpose processors and others classic methods like data prefetching and loop unrolling could exploit the Instruction Level Parallelism.

7 Acknowledgments

We thank the anonymous reviewers for their valuable comments that have helped us to improve the quality of the paper. This work has been funded in part by the Spanish Ministry of Science and Technology and the Feder European Funds under grant TIC $2003 - 08154 - C06 - 03$.

References

[1] N. Ahmed, N. Mateev, and K. Pingali. Tiling imperfectly-nested loop nests. *In Proceedings of Supercomputing*, November 2000.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. M. Kenney, and D. Sorensen. Lapack: A portable linear algebra library for high-performance computers. *Tech. Report CS-90-105, (LAPACK Working Note #20), Univ. Of Tennessee, Knoxville*, 1990.

[3] M. Antonini and M. Barlaud. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1(2):205–220, April 1992.

[4] S. Battista, F. Casalino, and C. Lande. Mpeg-4: A multimedia standard for the third millenium, part 1. *IEEE Multimedia*, 6(4):74–83, October 1999.

[5] S. Battista, F. Casalino, and C. Lande. Mpeg-4: A multimedia standard for the third millenium, part 2. *IEEE Multimedia*, 7(1):76–84, January 2000.

[6] G. Bernabé, J. M. García, and J. González. Reducing 3d wavelet transform execution time through the streaming simd extensions. *Proceedings of the 11th Euromicro Conference on Parallel Distributed and Network based Processing*, February 2003.

[7] G. Bernabé, J. González, J. M. García, and J. Duato. A new lossy 3-d wavelet transform for high-quality compression of medical video. *Proc. of IEEE EMBS International Conference on Information Technology Applications in Biomedicine*, pages 226–231, November 2000.

[8] G. Bernabé, J. González, J. M. García, and J. Duato. Enhancing the entropy encoder of a 3d-fwt for high-quality compression of medical video. *Proc. of IEEE International Symposium for Intelligent Signal Processing and Communication Systems*, November 2001.

[9] G. Bernabé, J. González, J. M. García, and J. Duato. Memory conscious 3d wavelet transform. *Proceedings of the 28th Euromicro Conference. Multimedia and Telecommunications*, September 2002.

[10] A. Bik, M. Girkar, P. Grey, and X. Tian. Efficient exploitation of parallelism on pentium iii and pentium iv processor-based systems. *Available at http://developer.intel.com/.*

[11] Y. Chen and W. A. Pearlman. Three-dimensional subband coding of video using the zero-tree method. *Proc. of SPIE-Visual Communications and Image Processing*, pages 1302–1310, March 1996.

[12] C. Chrysafis and A. Ortega. Line based reduced memory wavelet image compression. *IEEE Transactions on Image Processing*, 9:378–389, March 2000.

[13] G. Conte, S. Tommesani, and F. Zanichelli. The long and winding road to high-perfomance image processing with mmx/sse. *Proceedings of the Fifth IEEE International Workshop on Computer Architectures for Machine Perception*, 2000.

[14] I. Corporation. Ia-32 intel architecture software developer's manual. *Available at http://developer.intel.com/.*

[15] I. Corporation. Intel c/c++ compiler for linux. *Available at http://www.intel.com/software/products/compiler/c50/linux.*

[16] I. Daubechies. *Ten Lectures on Wavelets.* Society for Industrial and Applied Mathematics, 1992.

[17] J. Dongarra, J. D. Croz, I. S. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprogram. *ACM Trans. Math. Soft*, 14:1–17, 1988.

[18] D. Heller. Rabbit: A perfomance counters library for intel/amd processors and linux. *Available at http://www.scl.ameslab.gov/Projects/Rabbit/.*

[19] M. L. Hilton, B. D. Jawerth, and A. Sengupta. Compressing still and moving images with wavelets. *Multimedia Systems*, 2(3), 1994.

[20] I. J. W. (JPEG/JBIG). Fcd 14495, lossless and near-lossless coding of continuous tone still images (jpeg-ls).

[21] B.-J. Kim and W. A. Pearlman. An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (spiht). *Proceedings of Data Compression Conference*, 1997.

[22] Y. Kim and W. A. Pearlman. Stripe-based spiht lossy compression of volumetric medical images for low memory usage and uniform reconstruction quality. *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, 2000.

[23] M. S. Lam, E. E. Rothberg, and M. E. Wolf. The cache perfomance and optimizations of blocked algorithms. *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, April 1991.

[24] O. Lempel, A. Peleg, and U. Weiser. Intel's mmx technology - a new instruction set. *Proceedings of 42nd IEEE Computer Society International Conference*, 1997.

[25] A. S. Lewis and G. Knowles. Image compression using the 2-d wavelet transform. *IEEE Transactions on Image Processing*, 1(2):244–256, April 1992.

[26] A. W. Lim, S.-W. Liao, and M. S. Lam. Blocking and array contraction across arbitrarily nested loops using affine patitioning. *In Proceedings of the 8th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, June 2001.

[27] S. Mallat. A theory for multiresolution signal descomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[28] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek. An overview of jpeg-2000. *Proceedings of Data Compression Conference*, March 2000.

[29] S. Muraki. Approximation and rendering of volume data using wavelet transforms. *Proceedings of Visualization*, pages 21–28, October 1992.

[30] S. Muraki. Multiscale volume representation by a dog wavelet. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):109–116, June 1995.

[31] L. Nachtergaele, G. Lafruit, J. Bormans, and I. Bolsens. Fast software implementation of the mpeg-4 reversible integer wavelet transform on pentium mmx, sharc adsp and trimedia tm1000. *Proceedings of Packet Video*, 2000.

[32] C. Parisot, M. Antonini, and M. Barlaud. 3d scan-based wavelet transform and quality control for video coding. *EURASIP Journal on Applied Signal Processing*, 1, January 2003.

[33] P. Ranganathan, S. Adve, and N. P. Jouppi. Performance of image and video processing with general-purpose processors and media isa extensions. *International Symposium on Computer Architecture*, May 1999.

[34] D. Santa-Cruz and T. Ebrahimi. A study of jpeg 2000 still image coding versus others standards. *Proc. of the X European Signal Processing Conference*, September 2000.

[35] J. M. Shapiro. Embedded image coding using zerotrees of wavelets coefficients. *IEEE Transcations on Signal Processing*, 41(12):3445–3462, December 1993.

[36] T. Sikora. *MPEG Digital Video Coding Standars*. McGraw Hill Company, 1997.

[37] S. Thakkar and T. Huff. Internet streaming simd extensions. *IEEE Computer*, 32:26–34, 1999.

[38] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimizations of software and the atlas project. *Parallel Computing*, 27(1-2):3–35, 2001.

[39] M. J. Wolfe. *High Perfomance Compilers for Parallel Computer*. Addison-Wesley Publishing Company, 1996.

Gregorio Bernabé was born in Antibes (Alpes Maritimos, France) on 21 November 1974. He received the M.S. in Computer Science from the University of Murcia (Spain) in 1997. In 1998, he joined the Computer Engineering Department of the University of Murcia, where he is an Assistant Professor as well as a Ph. D. candidate. His current research interests include video compression using the Wavelet Transform, and the development of optimizations to improve the performance of the video compression algorithms based on the 3D wavelet transform.

Pepe Gonzalez received the M.S. and Ph.D. degrees from the Universitat Politecnica de Catalunya (UPC). In January 2000, he joined the Computer Engineering Department of the University of Murcia, Spain, and became an Associate Professor in June 2001. In March 2002, he joined the Intel Barcelona Research Center, where he is a Senior Researcher. Currently, Pepe is working in new paradigms for the IA-32 family, in particular, Thermal- and Power-Aware clustered microarchitectures.

Jose M. Garcia was born in Valencia, Spain on 9 January, 1962. He received the MS and the PhD degrees in electrical engineering from the Technical University of Valencia (Valencia, Spain), in 1987 and 1991, respectively. In 1987 he joined the Computer Science Department at the University of Castilla-La Mancha at the Campus of Albacete (Spain). From 1987 to 1993, he was an Assistant Professor of Computer Architecture. In 1994 he became an Associate Professor at the University of Murcia (Spain). From 1995 to 1997 he served as Vice-Dean of the School of Computer Science. At present, he is the Director of the Computer Engineering Department, and also the Head of the Research Group on Parallel Computing and Architecture. He has developed several courses on Computer Structure, Peripheral Devices, Computer Architecture and Multicomputer Design. His current research interests include Multiprocessors Systems, Interconnection Networks, File Systems, Grid Computing and its Application in Multimedia Systems. He has published over 45 refereed papers in different Journals and Conferences in these fields. Dr. Garcia is a member of several international associations as IEEE Computer Society, ACM, USENIX, and also a member of some European associations (Euromicro and ATI).