PERFORMANCE COMPARISON OF DWT SCHEDULING ALTERNATIVES ON PROGRAMMABLE PLATFORMS

N. D. Zervas, I. Tagopoulos, V. Spiliotopoulos, G. P. Anagnostopoulos, D. Soudris and C. E. Goutis

Dep. of Electrical and Computer Engineering
University of Patras
Patras, Greece.

ABSTRACT

The Discrete Wavelet Transformations (DWT) are data intensive algorithms. Energy dissipation and execution time of such algorithms heavily depends on data memory hierarchy performance, when programmable platforms are considered. Existing filtering operations scheduling alternatives for the 1D-DWT, employ different levels of data accesses locality. However locality of data references, usually comes at the expense of complex control and addressing. In this paper, the two main scheduling alternatives for the 1D-DWT are compared in terms of energy and speed. Additionally, we describe and evaluate the effect of an in-place mapping scheme, which minimizes memory requirements and improves locality of data reference, for the 1D-DWT. As execution platform, two commercially available general purpose processors are used.

1. INTRODUCTION

The inherent time-scale locality characteristics of the Discrete Wavelet Transformations has established them as powerful tools for numerous applications such as signal analysis, signal compression and numerical analysis. This has lead numerous research groups to develop algorithms and hardware architectures to implement the DWT. In [1], [2], [3] and [4] VLSI architectures for the 1D and 2D DWT have been proposed. Additionally, exhaustive comparisons among the various scheduling algorithms for the DWT, regarding their efficiency when the DWT is mapped in custom VLSI architectures, has been performed [5], [6].

Nowadays, the rapid advances in the area of programmable processors have made them an attractive solution even for real-time complex DSP applications, since they offer enough processing power combined with flexibility and small time-to-market. However very little has been done, to determine which scheduling algorithms for the DWT perform better when mapped on a DSP or general-purpose or embedded programmable processor. This paper is the first step of an attempt to fill this gap. Specifically, the two main scheduling algorithms for the 1D-DWT, namely the pyramid (PA) and the recursive pyramid (RPA) algorithms, are compared in terms of throughput. Additionally, two alternative methods to apply the *in-place* optimization [7] in the case of 1D-DWT are described and their effect is evaluated.

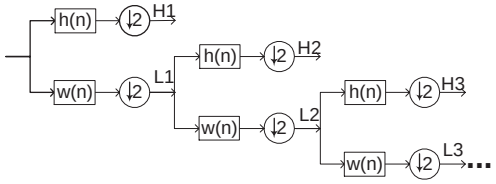

Figure 1: The DWT decomposition

The rest of this paper is organized as follows: Section 2 a briefly describes the multiresolution decomposition with the DWT, and the PA and RPA algorithms. In section 3 the two alternatives ways to apply the in-place optimization on 1D-DWT are described. In section 4 experimental results are presented and analyzed, while in section 5 conclusions and future research direction are drawn.

2. BASIC BACKGROUND

The DWT can be viewed as the multiresolution decomposition of a sequence [8]. It takes a length N sequence $IN[n]$, and generates and output sequence of length N. The output is a multiresolution representation of $IN[n]$. The highest resolution level is of length $N/2$, the next resolution level is of length $N/4$, and so on. We denote the number of frequencies or resolutions levels or levels with the symbol J. The DWT filter bank structure, realizing the DWT decomposition, is illustrated in Fig. 1. Typically, filters $h[n]$ and $w[n]$ shown in Fig. 1 are FIR filters. We denote N_H and N_W the number of taps of the high-pass ($h[n]$) and low-pass ($w[n]$) filters respectively, and define $N_F = \max\{N_H, N_W\}$.

To implement the decomposition of Fig. 1 on a single processor architecture, two main algorithms have been proposed so far. The first is the classical pyramid algorithm (PA) [8], while the second is a reformulation of the PA called recursive pyramid algorithm (RPA) [3]. The two algorithms differ on the filtering operations scheduling. This

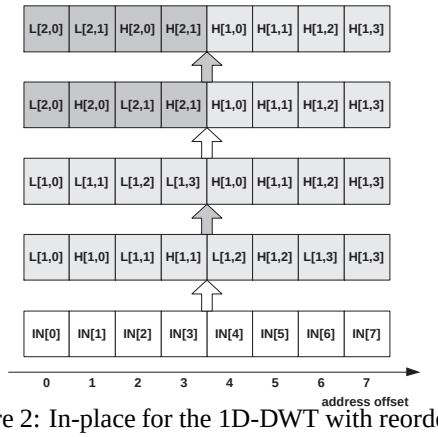

Figure 2: In-place for the 1D-DWT with reordering

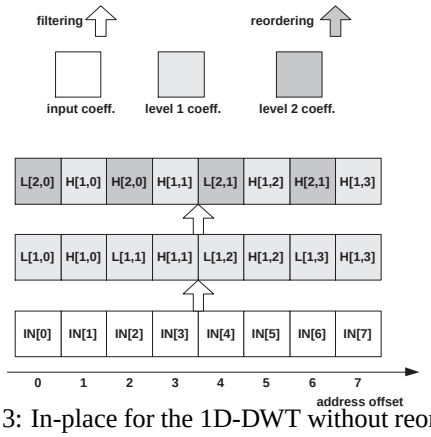

Figure 3: In-place for the 1D-DWT without reordering

imposes different memory requirements and different input access schedules. In the next subsections the two algorithms are briefly described.

2.1. Pyramid Algorithm

The pyramid algorithm is the direct implementation of the decomposition of Fig. 1. Specifically, filtering along level $j+1$ is initiated after the completion of filtering along level j. The pseudo-code for the PA follows:

begin{Direct Pyramid}
  for(j=1 to J)
    for(n=1 to 2j-1)
      $L[j,n] = \sum_{m=0}^{N_F-1} L[j-1, 2n-m]w[m]$
      $H[j,n] = \sum_{m=0}^{N_F-1} L[j-1, 2n-m]w[m]$
end{Direct Pyramid}

2.2. Recursive Pyramid Algorithm

The basic idea behind the recursive pyramid algorithm is the following:*Proceed to next level filtering ASAP*. This means that filtering along level j is interleaved, when enough coefficients to perform next filtering along level $j+1$ are produced. The pseudo-code for the RPA follows:

begin{Recursive Pyramid}
  for(i=1 to N-1)
    rdwt(i, 1);
end{Recursive Pyramid}
rdwt(i, j)
begin{rdwt}
  if(i is odd)
    k=(i+1)/2
      $L[j,k] = \sum_{m=0}^{N_F-1} L[j-1, 2n-m]w[m]$
      $H[j,k] = \sum_{m=0}^{N_F-1} L[j-1, 2n-m]w[m]$
  else
    rdwt(i/2, j+1)
end{rdwt}

Obviously, the RPA has smaller latency than the PA, if filtering operations are executed at the same rate for both algorithms.

3. IN-PLACE MAPPING FOR THE DWT

We assume that input sequence is stored in a memory block of size $N \times d$, where d is the number of bytes per coefficient. Remember now that coefficients $H[j,n]$ and $L[J,n]$ form the output of the DWT, while coefficients $L[j,n]$ ($j \neq J$) are intermediate results. Thus the straightforward selection is to allocate a separate memory block to store the N output coefficients, and $J-1$ memory blocks to store the $N/2^j d$ intermediate results of each level. Hence, the total storage requirements for the 1D-DWT is:

$$\left(N + N + \sum_{j=1}^{L-1} N/2^j\right) \times d = N \times \left(3 + 1/2^{L-1}\right) \times d \quad (1)$$

The amount of memory required for the DWT can be significantly reduced by applying an in-place optimization [7]. The key concept of such an optimization is to *store output and intermediate data in-place, of input data that are no longer needed*.

For example consider the 1D-sequence of Fig. 2 and assume a 5/3 DWT. Additionally assume that the N_F coefficients to be filtered are first fetched to local FIFO buffer, called filtering FIFO and denoted as FF. The pair of coefficients $L[1,0]$, $H[1,0]$ is produced by filtering the 3 first input coefficients, after performing a symmetrical mirroring. Since, input coefficients $IN[0]$ and $IN[1]$ are currently in the FF and will not fetched again from the input memory, we can store $L[1,0]$, $H[1,0]$ in their place (address offsets 0 and 1). In the sane way, coefficients $L[1,i]$, $H[1,i]$, can be stored at address offsets $2 \cdot i$ and $2 \cdot i+1$ respectively. A direct effect of this in-place mapping is that after the completion along the input, the low-frequency coefficients of level 1, that will be consumed to produce the coefficients of level 2, are not stored in consecutive addresses in the memory.

One way to overturn this problem is to to reorder the coefficients so that the first $N/2$ addresses store the coefficients $L[1,i]$, and the addresses from $N/2$ up to N store the coefficients $H[1,i]$. In this way to perform filtering along level 1, coefficients from consecutive memory addresses will be fetched. Generally, after reordering coef-

Algo./Proc.	Pentium I	Pentium MMX
L1 D-Cache	8 KByte	8 Kbyte
L1 I-Cache	8 KByte	8 KByte
L2 Unified Cache	256 KByte	256 KByte

Table 1: Cache memories sizes

ficients $L[j,i]$ and $H[j,i]$, are stored in place of coefficients $IN[i]$ and $IN[N/2^j+i]$, respectively. However, reordering the coefficients of level j after the completion of filtering along level $j-1$, requires $N/2^j$ extra memory accesses. This result to a total of $2N(1-1/2^L)$ overhead in terms of memory accesses, to compute the L-level DWT of an input of size N.

An alternative method to implement the in-place optimization of the DWT, is to slightly modify addressing equations, instead of employing reordering (Fig. 3).Specifically, if reordering is not performed, then the low-frequency coefficients $L[j,i]$, are stored in place of the input coefficients: $IN[2^j \cdot i]$, while the high-frequency coefficients $H[j,i]$, are stored in in place of the input coefficients: $IN[2^j \cdot i+1]$. In this way the overhead of $2N(1-2/2^L)$ accesses is replaced by the overhead of $N(1-2/2^L)$ multiplications of index i times a power of two (2^j), which are reduced to binary shifts.

It is evident that the later approach to apply the in-place optimization is the most efficient, since the $2N(1-2/2^L)$ memory accesses dissipate more energy and require more clock cycles than $N(1-2/2^L)$ binary shifts in any DSP or general purpose programmable processor [9]. Since the result of a comparison among the two alternative ways to apply the in-place mapping is easily predictable, in this paper we save some space by not providing any experimental results regarding the first approach.

4. EXPERIMENTAL RESULTS

The PA and RPA algorithms with and without the application of the in-place optimization have been implemented in C language. All four source codes have been mapped on two Intel's processors, namely a Pentium, and Pentium MMX. Table 1 illustrates the cache memory configuration for both processors. Compilation of the C sources has been performed using the Intel's C/C++ compiler v4.0. All four codes has been fed with input sequence of length $N = 1024$, 2048 and 10240, and executed considering $J = 3$, 4, 5 and 6 decomposition levels. Intel's Vtune environment [10] has been used to acquire profiling data, in terms of execution cycles and number of cache misses.

Tables 2 and 3 give the average, in terms of decomposition layers, number of level 1 (L1) data-cache misses for Pentium I and Pentium MMX respectively. It must be stressed here that the number of data-cache misses increases with J. Especially for $J = 10240$ this phenomenon is more intense, due to the large size of the higher decomposition

Algo./N	1024	2048	10240
PA	332	669	3617
PA_IN	164	414	5263
RPA	3143	47602	241105
RPA_IN	2122	28685	143504

Table 2: Average number of L1 D-Cache misses (Pentium)

Algo./N	1024	2048	10240
PA	282	539	3702
PA_IN	152	284	5220
RPA	160	317	240343
RPA_IN	158	320	143445

Table 3: Average number of L1 D-Cache misses (Pentium MMX)

layers. It is noted that we have chosen to present experimental results in the compact format of Tables 2, 3 for space economy purposes. Table 4 illustrates the number of L1 instruction cache misses, which is constant with respect to J and N. As far as execution speed is concerned the relative results for the four codes are almost identical for both processors. For this reason in Fig. 4 the average number of cycles for the two processors is given.

Comparing Table 4 to Tables 2 and 3 we note that the number of L1 instruction cache misses is very small compared to the number of data cache misses. This is due to the fact that for all cases the code fits in the L1 instruction cache, and thus only compulsory misses occur. In the following experimental results analysis the instruction cache misses will be ignored, due to their minor role.

4.1. PA versus RPA

As shown in Fig. 4, the PA algorithm has been proved to be faster than the RPA algorithm in all cases. This is due to its better data reference locality characteristics. Specifically, the PA algorithm linearly traverses along each level while on the other hand the RPA algorithm continuously interchanges filtering along layers. Hence, since PA scans sequential addresses, it is less valuable to conflict misses than the RPA, which has a less canonical data memory access trace. This fact causes the D-cache to perform better for the PA than the RPA especially for large values of N.

Furthermore, the RPA employs a more complex control than the PA algorithm. As a result the RPA employs a greater number of executed instruction than the PA.

4.2. In-place

As described in section 3, applying the in-place optimization on the one hand reduces memory requirements, and thus can improve data cache performance, but on the other hand increases the complexity of the addressing equations. Thus, it is very difficult to predict its effect on performance in a programmable platform.

In the case of the PA algorithm, the application of the in-place optimization worsens execution speed for all cases.

Algo./Proc.	Pentium I	Pentium MMX
PA	35	27
PA_IN	26	23
RPA	37	33
RPA_IN	36	29

Table 4: Number of L1 I-Cache misses

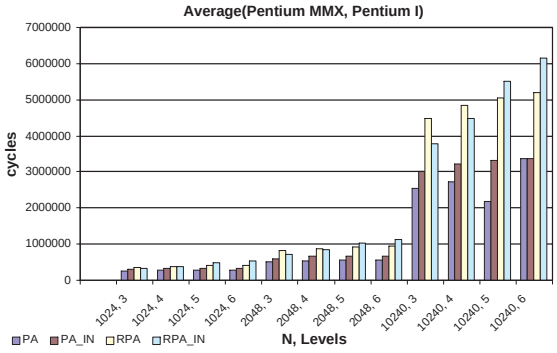

Figure 4: Average number of cycles

This is due to the fact that memory requirements minimization did not achieve to reduce further the already small number of cache misses, while on the other hand increased the number of executed instructions.

In the case of RPA, the application of the in-place optimization has contradictory effect $J = 3,\ 4$ and $J = 5,\ 6$; in the first case it improves execution speed, while in the latter case it worsens it. This can be explained as follows: The majority of data cache misses occurs due to interchange between the lower levels. This is because the smallest the level is, the greater the level's size becomes. On the other hand the overhead of more complex addressing equations remains for all levels of decomposition of the DWT. Thus the speed benefits from localization of accesses by in-place mapping offset the addressing complexity penalty, for small number of decomposition levels. On the other hand addressing complexity penalty dominates for great number of decomposition levels of the DWT.

5. CONCLUSIONS AND FUTURE WORK

The two main scheduling algorithms for the computation of the 1D-DWT, have been mapped on two general-purpose programmable processors, and compared in terms of execution speed. Experimental results indicate that the classic PA algorithm which employs the lower control and addressing complexity, has also better data reference locality characteristics than the RPA algorithm. This makes the PA to perform better on a programmable platform than the RPA, which is however favorable for VLSI hardware implementations. Additionally, two alternative methods to apply the in-place optimization have been described. The application of this optimization results in a significant reduction of memory requirements at the cost of slightly more complex addressing. This fact makes the in-place mapping always favorable for VLSI implementations of the DWT. However, experimental results indicated that in a programmable domain the memory size savings comes at the cost of slower execution in most cases.

Hence, at least as far as DWTs are concerned, it can be said that scheduling algorithms and optimizations developed targeting custom hardware platforms, can not migrate directly to the programmable domain. The focus of our future research is on theoretical exploration as well as on further experimentation regarding the 1D and 2D-DWT. The final aim is to identify which algorithms and optimizations, under which conditions perform better when executed on a programmable platform.

6. REFERENCES

[1] G. Knowles, "*VLSI architecture for the discrete wavelet transform*", Elec. Letters, vol. 26, no. 5, pp. 1184-1185, July 1990.

[2] K. K. Parhi, T. Nishitani, "*VLSI Architectures for Discrete Wavelet Transforms*", IEEE Tran. on VLSI Systems, Vol. 1, No. 2, pp. 191-202, June 1993.

[3] C. Chakrabarti, R. M. Owens, M. J. Irwin, *VLSI Architectures for the Discrete Wavelet Transform*, IEEE Tran. on Circuits and Systems, Vol. 42, pp. 305-316, May 1995.

[4] J.T. Kim et al, "*Scalable VLSI architectures for lattice structure-based discrete wavelet transform*", IEEE Tran. on Circuits and Systems II, Vol. 46, No. 8, pp. 1031-1043, Aug. 1998.

[5] C. Chakrabarti, M. Vishwanath, R. M. Owens, "*Architectures for Wavelet Transform: A Survey*", Journal of VLSI Signal Processing, Kluwer Academic Publishers, Vol. !4, Issue 2, pp. 171-192, 1996.

[6] G. Lafruit, L. Nachtergaele, J. Bormans, M. Engels, I. Bolsens, "*Optimal Memory Organizations for Scalable Texture Codecs in MPEG-4*", IEEE Tran. on Circuits and Systems for Video Technology, Vol. 9, No. 2, pp. 218-242, March 1999.

[7] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, A. Vandecappele, "CUSTOM MEMORY MANAGEMENT METHODOLOGY: Exploration of Memory Organization for Embedded Multimedia System Design", Kluwer Academic Publishers, Boston, 1998.

[8] S. Mallat, *Multifrequency chanel decompositions of images and wavelet models*, IEEE Tran. on Acoustics, Speech, Signal Process, Vol. 37, no 12, pp. 2091-2110, Dec. 1989.

[9] V. Tiwari, S. Malik and A. Wolfe, "*Instruction level power analysis and optimization of software*", Journal of VLSI Signal Processing Systems, 13: 223-228,1996.

[10] http://developer.intel.com/vtune.