Journal of VLSI Signal Processing 34, 209–226, 2003

High-Level Cache Modeling for 2-D Discrete Wavelet Transform Implementations

Y. ANDREOPOULOS AND P. SCHELKENS
Vrije Universiteit Brussel/IMEC, Dept. ETRO, Pleinlaan 2, B-1050, Brussels, Belgium

G. LAFRUIT
Inter-University Micro-Electronics Center—IMEC, Kapeldreef 75, B-3001, Leuven, Belgium

K. MASSELOS
Intracom S.A., Development Programs Department, Athens, Greece

J. CORNELIS
Vrije Universiteit Brussel/IMEC, Dept. ETRO, Pleinlaan 2, B-1050, Brussels, Belgium

Received September 6, 2001; Revised May 14, 2002; Accepted July 22, 2002

Abstract. The main implementations of the 2-D binary-tree discrete wavelet decomposition are theoretically analyzed and compared with respect to data-cache performance on instruction-set processor-based realizations. These implementations include various image-scanning techniques, from the classical row-column approach to the block-based and line-based methods, which are proposed in the framework of multimedia-coding standards. Analytical parameterized equations for the prediction of data-cache misses under general realistic assumptions are proposed. The accuracy and the consistency of the theory are verified through simulations on test platforms and a comparison is made with the results from a real platform.

Keywords: cache memories, discrete wavelet transform implementations, theoretical modeling

1. Introduction

JPEG-2000 [1] and MPEG-4 [2] are the two new standards that base part of their compression efficiency on discrete wavelet transforms (DWT) for the coding of images and textures. The standardization effort of JPEG-2000 showed that the largest part of the complexity of a wavelet-based coding system is associated to the memory organization for the production of the transform coefficients [3]. In several cases, a solution that involves tiling of the input signal is chosen in order to reduce the complexity when dealing with large amounts of input data. Thus, the term "image" in this paper refers to the currently-processed tile, with the tile-size ranging typically from 128×128 to 1024×1024 pixels.

The features that are mainly investigated by researchers are the different image traversal algorithms for the band-bursty production of the wavelet coefficients [4–10] and the efficient coupling of such transform-production methods (producers of information) with coding algorithms (consumers of information) [3, 5, 9]. The transform-production methods that are the most appropriate for coding applications can be classified in two major categories, based on the coefficient-production schedule.

The first category is based on an image traversal that leads to a *strictly breadth-first* production of wavelet

coefficients (SBF methods) [11]. This is usually referred as the *row-column wavelet transform* (RCWT) since, for every wavelet-decomposition level, the input data are filtered-and-subsampled horizontally and vertically in a sequential manner. In this way, the complete set of coefficients of the current level is produced before the initiation of the calculations for the next level; hence, this category has a strictly breadth-first production schedule. The second category consists of image traversals that lead to *roughly depth-first* production of wavelet coefficients (RDF methods) [11]. The two main techniques of this category are the *local* (or block-based) *wavelet transform* (LWT) [6, 8] and the *line-based wavelet transform* (LBWT) [4, 5]. They use a block or a line-based traversal respectively, to input the image and produce block or line parts of the transform subbands of all levels. With minor adaptations, both methods can produce the transform subbands of all levels in a streaming manner, thus a dyadically-decreased number of subband lines is produced for all decomposition levels.

A third category that is based on the traversal that leads to a *strictly* depth-first production of wavelet coefficients [7] is not examined here because, as [10] demonstrates, it demands a vastly parallel architecture that is usually not achievable in software implementations in programmable processors.

In the implementation arena, rapid advances in the area of (embedded) instruction-set processors have turned these systems into attractive solutions for the realization of real-time multimedia applications. This is explainable by the flexibility offered by such architectures, which allows the implementation of several coding algorithms on the same hardware, and also by time-to-market reasons. Typically, the best performance for data-dominated applications such as the DWT, is expected from the method that most efficiently reuses the data in the processor caches and maximally reduces the off-chip memory accesses to avoid additional delays and energy dissipation [12]. As a result, the data-related cache misses determine the efficiency of each method with respect to throughput and power.

Following the above indications, this paper analyzes the data-cache performance of various DWT-production approaches on instruction-set platforms and not the arithmetic or instruction-related complexity of these applications. The typical target architecture of the paper is based on (embedded) instruction-set video or multimedia processors (Philips TriMedia, TMS320C6x but also Pentium MMX), as these are state-of-the-art in real-time multi-dimensional signal-processing applications [13]. The data and instruction-memory multilevel hierarchies consist of a number of caches; in this paper we denote I-Cache, D-Cache the instruction, data cache of level 1 and $L2$-Cache the cache of level 2, which can be a separable configuration of instruction and data caches or a joint configuration for both. The data transfer and storage organization is left to hardware. Thus, the instructions or data enter the processor core after passing through the cache-hierarchy [14]. The typical path followed is: Main Memory $\rightarrow$ $L2$-Cache $\rightarrow$ I-Cache or D-Cache $\rightarrow$ Processor. When the instruction or data do not exist at the cache of a certain level, a miss occurs, in which case the processor waits until a block of instructions or data is fetched from the upper-level cache (or the main memory), causing a delay to the execution not related to the computational complexity of the executed program. Following the classical 3-C model of the cache misses, the latter can be separated into *capacity*, *compulsory* and *conflict* misses [14].

This paper first proposes single-processor software designs for all the transform-production methods in Section 2. Based on them, analytical equations are proposed in Section 3 that allow for the prediction of the expected number of data-cache misses in a generic memory hierarchy. The validity of the proposed equations is bounded by a set of constraints for the cache characteristics. However, the linking between the transform production and the coding consumption is not studied here, since this would specialize the presented results for a specific application, i.e. an MPEG-4 or JPEG-2000 compliant coder. In order to verify the theoretical framework and compare the proposed software designs, results are presented in Section 4 from simulations and from a real platform. It must be noted however that the theoretical analysis does not aim to precisely predict the number of misses in a specified cache architecture, because various system-dependent operations such as data-input from storage or image-retrieval media or memory allocation operations are expected to produce fluctuations in the final results. Nevertheless, using the theoretical results of this paper, one can approximately calculate the relative performance of the presented methods with respect to the cache utilization and, more importantly, analytically determine the advantages and disadvantages of every approach. Similar work for block-based DWT implementations in programmable architectures has been reported in [15], where the authors identify the input block-size

that leads to optimum utilization of the system cache.

2. The Different Wavelet Transform-Production Approaches

This section analyzes the different production schedules that will be used for the memory-related comparisons. The reader is assumed familiar with the lifting scheme (as described in [16]) and the convolution-based implementations of the wavelet transform. The analysis is structured in two sections, depending on the coefficient-production schedule. A pictorial description for each proposed design is presented, which facilitates the analysis of the cache-miss penalties. Since these penalties are caused by the input of data in the processor caches, the description concentrates on the copy activities between the different components and not on the detailed description of the filtering processes, because the latter actions are localized between the internal (on-chip) data-cache and the processor core and it is expected that they do not cause important fluctuations in the data-related miss penalties.

2.1. *The Design of Strictly Breadth-First Methods—The RCWT Approach*

These methods are based on the batch processing of all rows and columns of the input of every decomposition level. Since the filtering is not interleaved for each row or column, a lifting-scheme implementation is assumed [16], so that the minimum number of arithmetic operations is required for the transform production. Hence, a number of passes (S) are performed through every row or column of the input array and for every pass one predict-and-update step is performed in the input data [16]. For example, the 5/3 and 9/7 filter pairs are used in [1], which require one and two predict-and-update steps respectively through the input row or column [16]. Hence, $S = 1$ for the 5/3 and $S = 2$ for the 9/7 filter pair. All steps that are simple scaling operations can be applied during the final storage of the results and are not counted as additional passes. It must be noted that the final result is always stored in binary-tree (Mallat) form into the input array.

Without targeting a specific architecture, the designs of the SBF category can be studied in three different cases [11], depending on the pursued optimizations. Apart from the pictorial presentation of this section, the reader is referred to [11] for detailed pseudocode for the various cases. The basic memory component used is array *IMG* that contains the input image and the final result after every decomposition level. This array requires $N_1 \times (N_2 \cdot c_p)$ bytes of memory, where c_p denotes the number of bytes that represent one wavelet coefficient in memory, and N_1, N_2 denote the size of input data rows and columns, expressed in pixels. Additionally, one extra array is utilized, named *TMP_IMG*. The size of this array varies, depending on the specific case and the value of S. Below, a short description is given for the three distinct cases of the application of a discrete wavelet decomposition at level l, with $0 \leq l < LEVELS$, where *LEVELS* denotes the total decomposition levels. Due to the binary-tree organization of *IMG*, only the upper-right $(N_1 \cdot 2^{-l}) \times (N_2 \cdot 2^{-l})$ elements of the array are used for level l:

- *Case 1: Minimum memory implementation*. In this case, *TMP_IMG* is a 1-D array containing $(0.5 \cdot \max\{N_1 \cdot 2^{-l}, N_2 \cdot 2^{-l}\} \cdot c_p)$ bytes. The application of the transform in one row can be seen in Fig. 1 for level l. As the left part of Fig. 1 shows, this case applies the predict-and-update steps of the lifting scheme directly in the elements of *IMG* and replaces the results in-place (gray dots). After the end of each row or column processing, a reordering procedure creates the binary-tree for the newly produced coefficients in the current row or column of *IMG* with the aid of *TMP_IMG* array by extracting all the high-frequency coefficients (white dots), shifting all low-frequency coefficients (black dots) into neighboring positions and then storing back the contents of *TMP_IMG* (Reordering of Fig. 1).
- *Case 2: Minimum number of memory accesses for the transform production*. In this case, as shown in Fig. 2, if $S = 1$ (lower part of the figure) *TMP_IMG* is a 2-D array with $(N_1 \cdot 2^{-l}) \times (N_2 \cdot 2^{-l})$ elements of c_p bytes each. If $S > 1$ (upper part of Fig. 2) *TMP_IMG* is a 1-D array with $(\max\{N_1 \cdot 2^{-l}, N_2 \cdot 2^{-l}\})$ elements. The application of the transform in this case for the l-th decomposition level is pictorially explained in Fig. 2 and leads to the minimum amount of array access/copy operations.
- *Case 3: Maximum locality in the processing*. As shown in Fig. 3, initially each new row or column of *IMG* is copied in *TMP_IMG*, which is a 1-D array occupying $(\max\{N_1 \cdot 2^{-l}, N_2 \cdot 2^{-l}\})$ elements. All the subsequent predict-and-update passes occur locally in *TMP_IMG*. The results of the last pass are directly

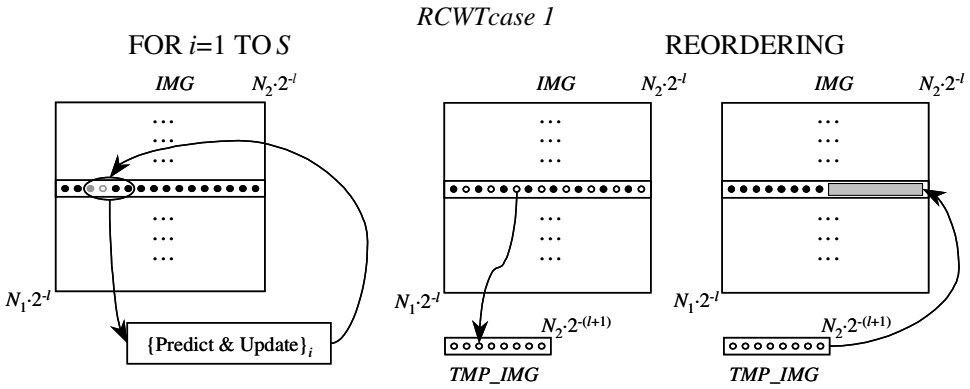

Figure 1. Application of RCWT Case 1. Only the Row-by-Row processing is shown, a similar scheme applies for the columns.

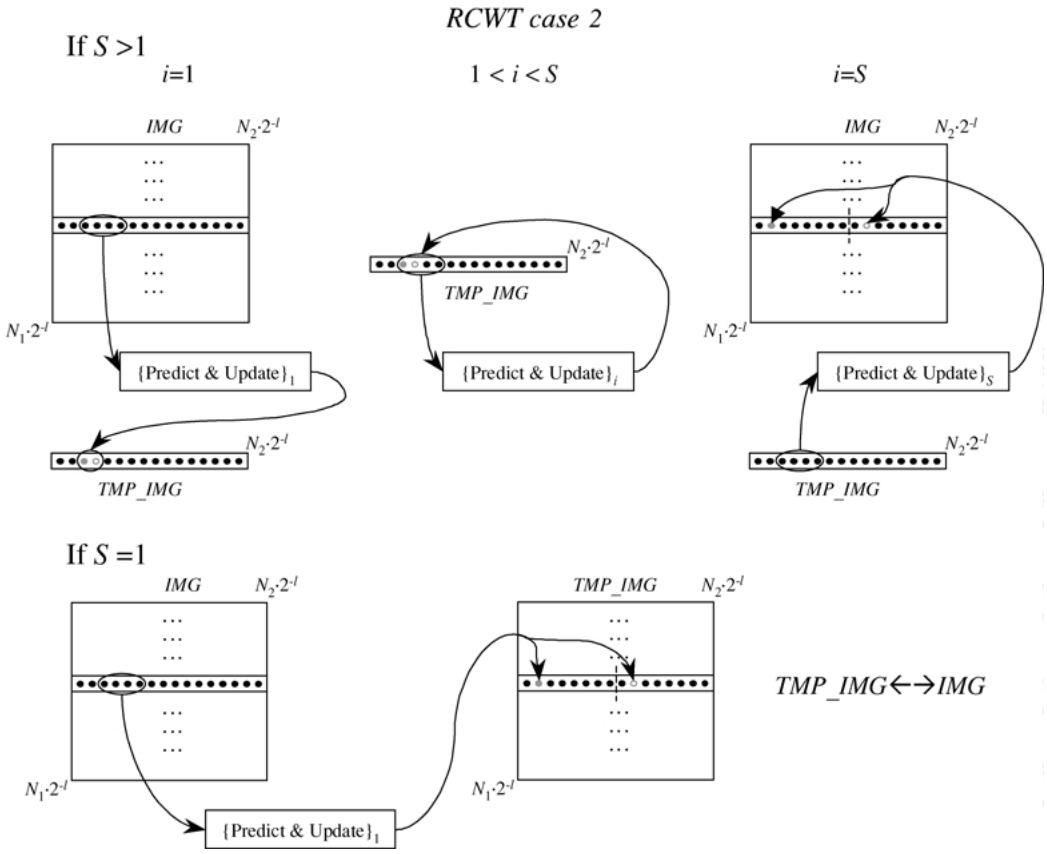

Figure 2. Application of RCWT Case 2. Only the Row-by-Row processing is shown, a similar scheme applies for the columns. Operation *TMP_IMG* ←→ *IMG* denotes the exchange of memory pointers of *IMG* and *TMP_IMG* [11].

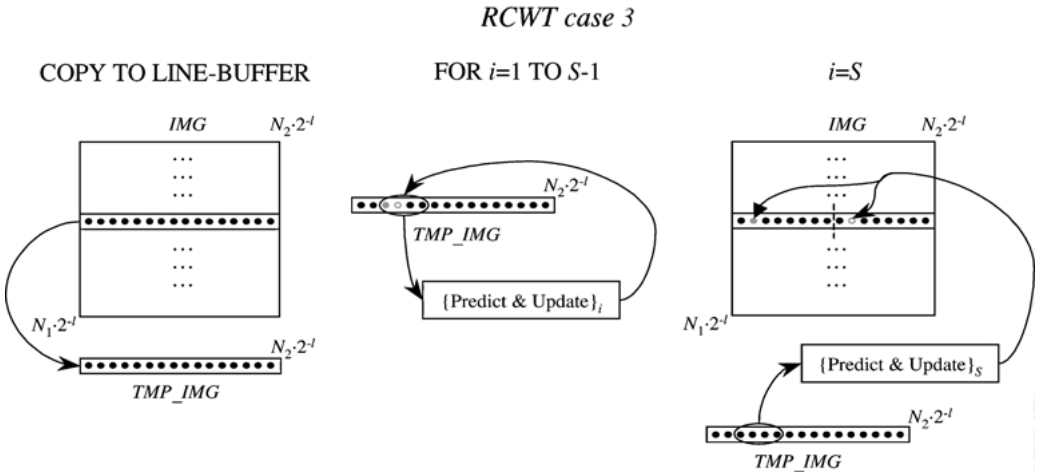

Figure 3. Application of RCWT Case 3. Only the Row-by-Row processing is shown, a similar scheme applies for the columns.

stored back into *IMG* in binary-tree form as seen from the right part of Fig. 3.

2.2. *The Design of Roughly Depth-First Methods—The LWT and LBWT Approaches*

These methods are based on the separation of the input of every decomposition level into non-overlapping components, the processing of these components using some of the previously produced low-frequency coefficients (or image samples for level $l = 0$) and the update of these coefficients with some of the newly-produced low-frequency coefficients.

The outputs of every level (high-frequency coefficients) are usually driven to a compression engine according to the selected compression algorithm. Typically, the outputs are grouped in parent-children trees [6] or intra-subband blocks [1]. However, as mentioned in the previous section, the link with a specific coding system is not studied in this paper.

The input components are blocks for the LWT method (block-based) and lines for the LBWT method (line-based). Since usually a software implementation targets a single-processor system, the single-processor architecture of [8] is selected. However, unlike [8], the presented design allows the implementation of both methods with the same architecture, due to the flexibility of a software implementation, which supports different memory configurations. Thus, the line-based architecture of this paper can be seen as the asymptotic case of the block-based scheme, when the block width becomes equal to the image width. A detailed pseudocode for the software implementation of the methods of this category is presented in [11]. The LWT design that is presented in this section has been successfully used in a coding system in [9], where the reader can also find details about the initialization and finalization phenomena at the image borders.

In Fig. 4, the processing of the RDF methods is shown pictorially for decomposition level l. The used arrays follow the naming conventions of [8] and [9] to emphasize the relationships with that architecture. Thus, *IPM* denotes the *inter-pass memory*, where each new input component is written. In general, *IPM* is a 2-D array occupying $(X \cdot 2^{LEVELS}) \times (Y \cdot 2^{LEVELS} c_p)$ bytes; for the LBWT, *IPM* has a static width of $(N_2 \cdot c_p)$ bytes. Since the filtering is interleaved in the columns for every decomposition level (and also in the rows for the block-based architecture), the intermediate results are stored in the overlap memory, denoted as *OM_ROWS* and *OM_COLS* in Fig. 4. For biorthogonal filter-pairs with $(2M + 1)/(2M - 1)$ taps, *OM_ROWS* is a 3-D array (decomposition level/row of *IPM*/overlap coefficient) that occupies $2^{LEVELS+1} \cdot (2M - 1) \cdot c_p$ bytes (maximally) and *OM_COLS* is a 3-D array (decomposition level/column of current level/overlap coefficient) that occupies maximally $2N_2 \cdot (2M-1) \cdot c_p$ bytes. The actual filtering is performed using a small 1-D array of $(2M + 1) \cdot c_p$ bytes, called *filtering-FIFO* (*FF*) to emphasize the inherent functionality, and is convolution based.

As seen in the right part of Fig. 4, for every row of *IPM*, first the coefficients of the corresponding row of *OM_ROWS* are inserted in *FF*, followed by each new

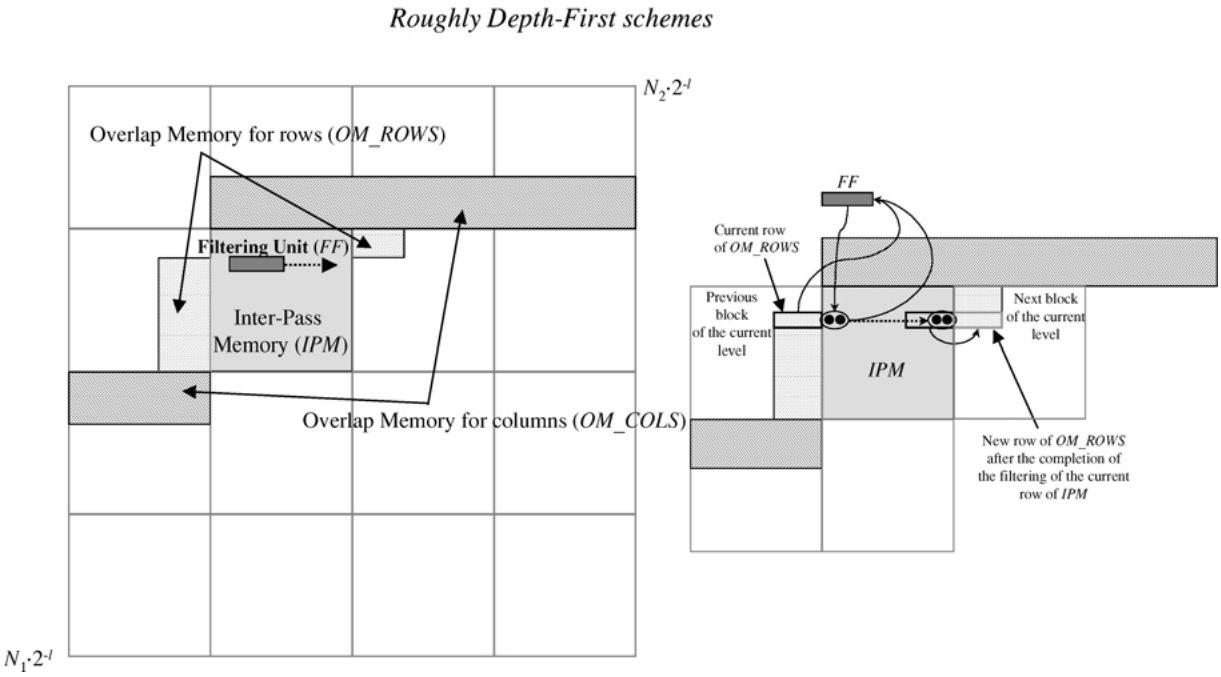

Figure 4. Illustration of the processing of one input component for the RDF methods.

pair of coefficients of the current row of *IPM*. After the filtering, the resulting low and high-frequency coefficients are written in-place in *IPM*. When the filtering of the current row of *IPM* is completed, the results remaining in the *FF* are written back into the current row of *OM_ROWS* for the processing of the neighboring block. A similar scheme applies for the columns, with the aid of *OM_COLS*. It must be noted that in the line-based approach the row filtering is not split in successive parts, thus only *OM_COLS* is utilized. In addition, lifting-based approaches can be considered as well [3], if the memory sizes of the various buffers are adapted to the specific data-dependencies of the lifting-based filtering.

3. Calculation of the Data-Related Cache Penalties

3.1. Constraints of the Presented Analysis

For every different method of Sections 2.1 and 2.2, the expected number of D-Cache misses varies, depending on the cache organization of the target execution-platform. For simplicity in the presented analysis, the case of a fully-associative data-cache is selected; in this way however, the conflict misses of an n-way set-associative cache are excluded from the description [14]. In the case of a miss due to lack of space in the cache (capacity miss), the LRU replacement strategy [14] is assumed, where the cache block that was least-recently used is replaced with the new input block. For each of the different approaches of the DWT, the boundary effects (initialization and finalization phenomena) are ignored so as to facilitate the description. In addition, the I-Cache behavior will not be discussed because, as the experimental results show (see Table 1), for the systems of interest, the I-Cache misses are much less frequent than the data-related cache-misses. Apart from the algorithm parameters, the important cache parameters must be defined. In this paper we denote s_{L2}, s_D the cache size of the $L2$ (data-cache) and D-Cache respectively and b_{L2}, b_D the corresponding blocksizes. The blocksize represents the number of sequential bytes that are inserted in every cache from the higher level of the memory hierarchy whenever an access to that level is performed [14]. In general, due to paging and burst-mode effects of the SDRAM chips, the number of bytes transferred in every RAM access cannot be fixed [14]. Thus, if a more detailed analysis is desired, these effects should be incorporated in the model.

Table 1. Cache misses for the SBF category, as measured with the tools of [17] for various image sizes. The 9/7 filter-pair was used.

Strictly breadth-first methods (Average for Cases 1, 2, 3 with 5 decomposition levels)						
Image size	D-Cache	I-Cache	Ratio	$L2$-Cache (data)	$L2$-Cache (instruction)	Ratio
128×128	37559	1174	32:1	1465	753	1.95:1
256×256	236156	1187	199:1	17475	765	23:1
512×512	1050275	1188	884:1	74905	766	98:1
1024×1024	4483127	1242	3610:1	305505	783	390:1

This paper targets realistic cache implementations where only a part of the input image can be buffered in the $L2$-Cache and also, in some cases, only a fraction of the currently-processed input (row, column or block, line) can fit in the D-Cache. This is because, even in future designs of custom DSPs that will target wavelet-related compression applications, the minimization of the on-chip cache will be a critical issue that determines performance and cost. Thus, a solution where the entire input image can fit in the cache is not expected to be feasible. In order to theoretically define the cases for which the analysis of this section is applicable, we denote the following constraints:

$$2N_2 \cdot c_p < s_D < N_1 \cdot b_D, \tag{1}$$

$$2N_1 \cdot b_{L2} < s_{L2} < N_1 \cdot N_2 \cdot c_p, \tag{2}$$

where $b_D > 2(N_2/N_1) \cdot c_p$ so that (1) is valid. These constraints come from the way the utilized 2-D arrays are stored in memory. Typically, a 2-D array is allocated in memory as a sequence of rows. Thus, when accessing elements in the row direction, sequential memory accesses are performed that in fact are grouped together since the cache organization of every level always inputs b_{L2} or b_D sequential bytes. However, when accessing elements in the column direction, these elements are not sequential in the physical memory and so, for accessing k sequential coefficients of one column, $k \cdot b_{L2}$ or $k \cdot b_D$ bytes will be copied in the $L2$ or D-Cache due to the unavoidable grouping. Having this in mind, the constraint of (1) means that the D-Cache is sufficiently large to hold input coefficients that correspond to two image rows ($2N_2 \cdot c_p$), but not large enough to sustain the coefficients of one entire image column ($N_1 \cdot b_D$). The constraint of (2) means that the $L2$-Cache is sufficiently large to hold input coefficients that correspond either to two portions of b_{L2} sequential columns when each portion is coming from a different 2-D array, or to $2b_{L2}$ sequential columns when they come from the same array. In addition, an upper constraint is set; the $L2$-Cache is always smaller than the original image size, otherwise only compulsory misses for the initial image input are expected in the $L2$-Cache. In general, (1) and (2) are chosen so that the lower bound certifies that no exponential increase happens in the data-related misses, while the upper bound certifies that the data-related misses are not negligible in comparison to the instruction-related misses.

As seen from the description of this section, we focus on a two-level cache architecture. It must be noted however that the presented theoretical analysis is invariant to the number of levels; it only depends on the cache characteristics of the system. This is a natural consequence from the fact that the behavior of the cache memories is invariant to the specific level they belong to, since they are designed to operate independently, ignoring the upper and lower-level memory organization [14]. In this way, our theoretical model can cover both a two-level and a one-level cache hierarchy, if the equations for the prediction of the misses in level 2 are ignored. As a result, the vast majority of processors found in the market is covered by our description.

3.2. The Expected Data-Related Cache Misses

Under the constraints of (1) and (2), the data-related cache misses can be estimated. For all the presented methods, one iteration through the input data consists of the input of an image component (row, column or block), the application of the filtering process to that component and the production and storage of the results. Since the various methods perform similar operations, such as sequential accessing along a row or a column of a 2-D matrix, the calculation of the various

data-related cache misses, was made using some templates for the most typical cases, specifically adapted for each method. The interested reader can find in detail the various templates in Appendix 1. In short, template $T1$ gives the expected misses from a row-by-row processing of a 2-D array when a large cache is concerned, i.e. the $L2$-Cache. Template $T2$ gives the expected misses from a subsequent column-by-column processing of the input 2-D array, and template $T3$ shows the misses from the column-by-column processing when a small cache is concerned, i.e. the D-Cache. The following two sections analyze our results.

3.2.1. Data Misses in the L2-Cache. Beginning from the strictly breadth-first (SBF) production methods, as shown in Figs. 1–3, all cases will process sequentially (in the row or column direction) the elements of array *IMG* in the $L2$-Cache. Additionally Case 2 with $S = 1$ will input the elements of array *TMP_IMG* (2-D array), since the complete array does not fit in the $L2$-Cache due to the constraint of (2). In all cases where *TMP_IMG* is a 1-D array, it is assumed that, after the initial input, no additional misses occur from this array, since it is reused in every row or column processing and hence it is expected to remain in the cache. This case is simplified to the row-by-row and column-by-column processing of a 2-D array with the exception of Case 2 with $S = 1$, where another 2-D array is inserted in the cache at the same time. By applying the template $T1$ for the rows, the misses due to the row filtering of level $l + 1$ are:

$$\begin{aligned} &SBF\_row\_miss_{L2}(l+1) \\ &\quad = f_S \cdot f_{L2} \cdot T1\left(N_1 \cdot 2^{-(l+1)}, N_2 \cdot 2^{-(l+1)}, b_{L2}\right) \\ &\quad = f_S \cdot f_{L2} \cdot N_1 \cdot N_2 \cdot 2^{-2(l+1)} \cdot c_p / b_{L2}, \end{aligned} \tag{3}$$

where: $f_S = \{2\text{: Case\_2} \cap S = 1;\ 1\text{: otherwise}\}$, $f_{L2} = \{0\text{: } f_S \cdot N_1 \cdot N_2 \cdot c_p \cdot 2^{-2l} < s_{L2};\ 1\text{: otherwise}\}$.

The factor f_S of (3) distinguishes that sole case where two 2-D arrays are inserted in the cache. The factor $N_1 \cdot 2^{-(l+1)}$ represents the number of rows of the level $l + 1$. Every row of this level has $N_2 \cdot 2^{-(l+1)} \cdot c_p$ bytes. The f_{L2} factor is a bound that checks whether the output of the previously produced decomposition level fits in the $L2$-Cache. If so, then no additional misses are accounted for. Otherwise, all low-frequency coefficients are assumed non-resident in the cache.

For the $L2$-Cache misses of the column processing, after the completion of the row processing of level l, the template $T2$ is applicable; thus, the number of $L2$-Cache misses for the processing of level l is:

$$\begin{aligned} &SBF\_column\_miss_{L2}(l) \\ &\quad = T2(N_1 \cdot 2^{-l}, N_2 \cdot 2^{-l}, b_{L2}, s_{L2}) \\ &\quad = \sum_{i=1}^{\frac{f_S N_2 2^{-l} c_p}{b_{L2}}} \left(M_l + \sum_{k=1}^{i} a_k \right), \end{aligned} \tag{4}$$

where the f_S factor, as before, determines the total amount of coefficients inserted, according to the specific case and the number of passes (S) through the newly-inserted row or column. The factors M_l and a_k are defined in Appendix 1.

For the roughly depth-first (RDF) methods the calculation of the data misses in the $L2$-Cache is simpler. For modest values of X, Y, based on the constraint of (1), after the compulsory cache-misses caused by the Row-by-Row processing of each new block (template $T1$), the Column-by-Column processing is not expected to cause any additional misses since all the data for the current block are assumed resident in the cache. Hence, template $T2$ does not apply and no misses are expected during this procedure. The interested reader can define exactly the set of values of X, Y according the specific case where the constraint of $T2$ is not satisfied (and thus the template is not applicable). Essentially this means that s_c must be larger than $R \cdot C \cdot c_p$, where $s_c = s_{L2}$ and $R = X \cdot 2^{LEVELS}$ and $C = Y \cdot 2^{LEVELS}$ (for the block-based). In this case, for the block-based method:

$$\begin{aligned} RDF\_miss_{L2} &= [N_1/(X \cdot 2^{LEVELS})] \cdot [N_2/(Y \cdot 2^{LEVELS})] \\ &\quad \cdot T1(X \cdot 2^{LEVELS}, Y \cdot 2^{LEVELS}, b_{L2}) \\ &= N_1 \cdot N_2 \cdot c_p / b_{L2.} \end{aligned} \tag{5}$$

The expected number of misses of the line-based method is given also from (5) if one replaces $Y \cdot 2^{LEVELS}$ with N_2 (the line width), since the input-block of the line-based method consists of entire rows. It must be noted that both the LWT and LBWT methods are tuned to a streaming data input; hence, the image data are collected in a sequence of 2^{LEVELS} rows either directly in *IPM* (LBWT method), or in a row-buffering memory of size 2^{LEVELS} image rows (LWT method), overwriting the previously inserted data. Since this memory is expected to be resident in the $L2$-Cache, if the image-input procedures are not taken into account in the total cache-miss budget, the expected misses in the $L2$-Cache for the RDF methods are then obtained by

replacing N_1 by 2^{LEVELS} in Eq. (7):

$$RDF\_miss_{L2,\text{DIRECT\_INPUT}} = 2^{LEVELS} \cdot N_2 \cdot c_p/b_{L2}. \quad (6)$$

Equation (6) is useful because it excludes the image-input procedures, which are system-dependent functions. Thus it models the extreme (fictitious) case where the input data are written directly in the $L2$-Cache from the input and not from the main memory. The actual experimental results are expected to be between the numbers reported by (5) and (6), depending on how much the used profiling tools that produce these results take into account the image-input procedures.

3.2.2. Misses in the D-Cache. For the strictly breadth-first methods, the misses occurring in the D-Cache due to the row processing can be calculated using template $T1$ since the constraint of (1) assures the applicability of this template for all cases. Thus, the D-Cache misses for the row processing of level l are simply:

$$\begin{aligned} SBF\_row\_miss_D(l) &= T1(N_1 \cdot 2^{-l}, N_2 \cdot 2^{-l}, b_D) \\ &= f_S \cdot N_1 \cdot N_2 \cdot 2^{-2l} \cdot c_p/b_D. \end{aligned} \quad (7)$$

The misses during the column processing are more complex, since, as the constraint of (1) shows, an entire input column does not always fit in the D-Cache.

For Case 1 (see Fig. 1), if each column of *IMG* of decomposition level l does not fit in the D-Cache, the application of the lifting scheme in S passes can be calculated using template $T3$. The reordering procedure will cause an additional pass per column of *IMG* and the input of *TMP_IMG* for every column. These misses can be calculated by templates $T3$ and $T1$ respectively. Furthermore, as shown in the right part of Fig. 1, the last step of the reordering requires another pass through the last half of every column, hence these additional misses (if any) are calculated with template $T3$. Summarizing:

$$\begin{aligned} &SBF\_case1\_col\_miss_D(l) \\ &\quad = T3(N_1 \cdot 2^{-l}, N_2 \cdot 2^{-l}, S+1, s_D) \\ &\qquad + N_2 \cdot 2^{-l} \cdot T1(1, 0.5N_1 \cdot 2^{-l}, b_D) \\ &\qquad + f_{Da} \cdot T3(N_1 \cdot 2^{-l}, 0.5N_2 \cdot 2^{-l}, 1, s_D) \\ &\quad = (f_{T3} + 0.5c_p/b_D + 0.5f_{Da}) \cdot N_1 \cdot N_2 \cdot 2^{-2l}, \end{aligned} \quad (8)$$

where f_{T3} is defined in template $T3$ of Appendix 1 and f_{Da} checks if half a column fits in the cache:

$$f_{Da} = \{1\ \forall l\colon N_1 \cdot 2^{-l} \cdot b_D > 2s_D; 0\colon \text{otherwise}\}. \quad (9)$$

For Case 2 (see Fig. 2), if one pass is performed ($S = 1 \Rightarrow f_S = 2$) then one column from both *IMG* and *TMP_IMG* (2-D) arrays of the current decomposition level must enter the D-Cache, causing compulsory misses that can be calculated with template $T3$. Otherwise, one column of the current level and *TMP_IMG* array (1-D) must enter the D-Cache and templates $T3$ and $T1$ are used (in this case $f_S = 2$). Hence, the misses in the D-Cache from the column filtering of level l are:

$$\begin{aligned} &SBF\_case2\_col\_miss_D(l) \\ &\quad = f_S \cdot T3(N_1 \cdot 2^{-l}, N_2 \cdot 2^{-l}, S, s_D) \\ &\qquad + (2 - f_S) \cdot N_2 \cdot 2^{-l} \cdot T1(1, N_1 \cdot 2^{-l}, b_D) \\ &\quad = N_1 \cdot N_2 \cdot 2^{-2l} \cdot [f_S \cdot f_{T3} - (f_S - 2) \cdot c_p/b_D]. \end{aligned} \quad (10)$$

For Case 3 (see Fig. 3), for the filtering of each column of the current decomposition level, the column itself and *TMP_IMG* must enter D-Cache (copy operation of Fig. 3). The compulsory misses of this operation are calculated with templates $T3$ and $T1$ respectively. The application of S-1 predict-and-update steps (middle of Fig. 3) does not cause any additional misses since the constraint of (1) certifies that *TMP_IMG* is expected to reside in the D-Cache. Finally, the last predict-and-update step (last pass) will directly store the final results in binary-tree (reordered) form as shown in Fig. 3 and the capacity misses expected from this operation are calculated with template $T3$. Summarized:

$$\begin{aligned} &SBF\_case3\_col\_miss_D(l) \\ &\quad = 2 \cdot T3(N_1 \cdot 2^{-l}, N_2 \cdot 2^{-l}, 1, s_D) \\ &\qquad + N_2 \cdot 2^{-l} \cdot T1(1, N_1 \cdot 2^{-l}, b_D) \\ &\quad = N_1 \cdot N_2 \cdot 2^{-2l} \cdot (2 + c_p/b_D). \end{aligned} \quad (11)$$

For the roughly depth-first methods, the expected number of D-Cache misses can be calculated based again on the templates presented in Appendix 1. For simplicity we study the case where $X = Y = 1$. For the LWT method, all the memory components used for the filtering procedures of every level are expected to reside in the D-Cache after their initial input, except the *OM_COLS* array, which is the only array that does not completely fit in the cache according to the constraint set in (1). For the initial input of every image block in the *IPM*, the compulsory misses in the D-Cache can be calculated with the application of template $T1$. The additional misses occurring in the D-Cache are due to the copying of the coefficients stored in *OM_COLS* to

FF, as seen in the right part of Fig. 4; for every decomposition level l, this procedure consists of sequential processing along a row of *OM_COLS* of level l, hence again template $T1$ can be utilized. In total, the following D-Cache misses are expected for the block-based method:

$$\begin{aligned} &RDF\_LWT\_miss_D \\ &= N_1 \cdot N_2 \cdot 2^{-2LEVELS} \cdot \Bigg[2^{LEVELS} \cdot T1(1, 2^{LEVELS}, b_D) \\ &\quad + T1(1, 2M-1, b_D) \cdot \sum_{l=1}^{LEVELS} 2^l \Bigg] \\ &= N_1 \cdot N_2 \cdot 2^{-2LEVELS} \cdot [2^{2LEVELS} \cdot c_p/b_D \\ &\quad + (2^{LEVELS+1} - 2) \cdot (2M-1) \cdot c_p/b_D]. \end{aligned} \tag{12}$$

The value of M depends on the used filter-pair taps. Usually a biorthogonal $(2M+1)/(2M-1)$ filter-pair is used with $M = 2$ or $M = 4$ [1].

For the line-based method, *IPM* consists of entire rows of the input image and for the row processing, due to the constraint of (1), only the two most-recently accessed rows of *IPM* are expected to reside in the D-Cache. The coefficients of every level are skewed in the *IPM* in order to perform in-place storage of the filtering results. For every decomposition level l, for the row filtering, template $T1$ is used for the D-Cache miss calculation and for the column filtering, template $T3$ gives the expected misses from the column input of *IPM*, while template $T1$ estimates the misses from the copying of the elements of *OM_COLS* of the current level to *FF*. In total, for *LEVELS* decomposition levels, the expected D-Cache misses are:

$$\begin{aligned} &RDF\_LBWT\_miss_D \\ &= N_1 \cdot 2^{-LEVELS} \cdot \Bigg[\sum_{l=1}^{LEVELS} T1(2^l, N_2, b_D) \\ &\quad + \sum_{l=1}^{LEVELS} T3(2^{LEVELS}, N_2, 1, s_D) \\ &\quad + T1(1, 2M-1, b_D) \cdot \sum_{l=1}^{LEVELS} 2^l \Bigg] \\ &= N_1 \cdot 2^{-LEVELS} \cdot [N_2 \cdot (c_p/b_D) \cdot (2^{LEVELS+1} - 2) \\ &\quad + LEVELS \cdot 2^{LEVELS} \cdot N_2 + (2M-1) \cdot (c_p/b_D) \\ &\quad \cdot (2^{LEVELS+1} - 2)]. \end{aligned} \tag{13}$$

The presented theoretical analysis leads to equations similar to the ones reported in [11], with a few simplifications in the calculations. Using the formulas of Eqs. (3)–(6) for the $L2$-Cache and (7)–(13) for the D-Cache, the data-related cache-misses can be found for the binary-tree wavelet decomposition of a $N_1 \times N_2$ image in *LEVELS* decomposition levels for any typical wavelet filter-pair. For the filter operations, either the lifting-scheme approach can be used for the strictly breadth-first approaches, hence S passes will be applied for every input row or column, or a classical convolution-based implementation can be utilized for the roughly depth-first methods. As mentioned before, the convolution-based filtering was chosen so as to facilitate the generality of the proposed design for the roughly depth-first methods without considering filter-specific lifting dependencies.

4. Experimental Results

To experimentally check which traversal method appears to be the most efficient one with respect to the minimization of the data-related cache penalties, the different approaches illustrated in Section 2 were implemented using ANSI-C and were compiled and simulated under the same conditions. All source codes were platform-independently optimized for speed.

The cache-miss penalties were measured using the tools of [17] in a typical 64-bit simplescalar processor architecture. The specified cache organization simulated a 2-level, separable instruction and data cache with the LRU replacement-strategy. Each of the two cache levels was considered fully associative, with 128 and 2048 blocks for the first and second level respectively. This configuration satisfies the constraints of (1) and (2) for the majority of input image sizes. A block size of 64-bytes was selected for both levels. Figure 5 shows the results for the 9/7 filter-pair with floating-point precision for the wavelet coefficients ($S = 2$, $M = 4$, $c_p = 4$), while Fig. 6 shows the results for the 5/3 filter-pair with fixed-point precision ($S = 1$, $M = 2$, $c_p = 2$). All results are averages of the execution for 4 to 6 decomposition levels, which correspond to typical settings for image and texture compression applications. Instead of absolute values, the theoretical curves have been uniformly scaled to the maximum experimental point of every graph and all measurements are shown as a percentage of the maximum measurement, so as to facilitate the comparisons. As discussed in the previous section, the streaming-input design of the RDF methods is expected to have a linear increase of the data-related cache misses when

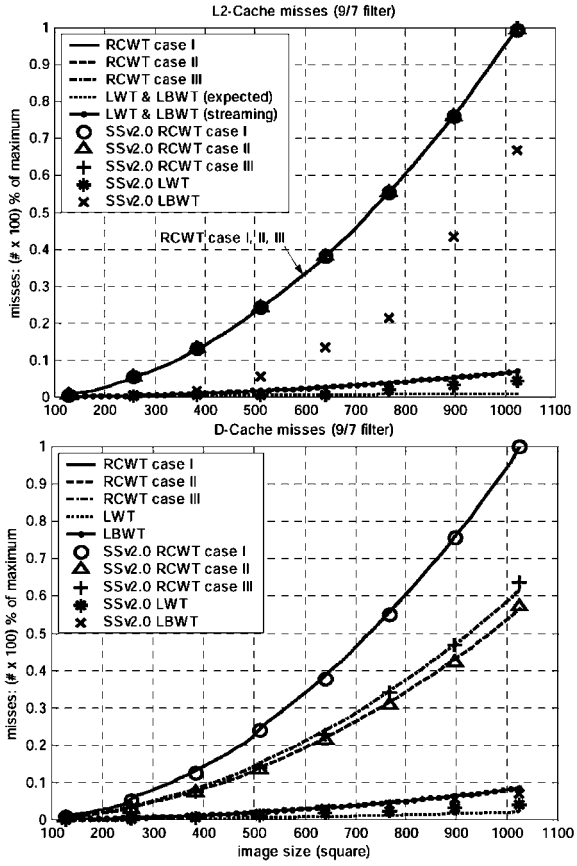

Figure 5. Theoretical (lines) and experimental (points) data-related cache misses, as measured with simulator tools, for the 9/7 filter-pair. SS denotes the SimpleScalar simulator toolset of [17].

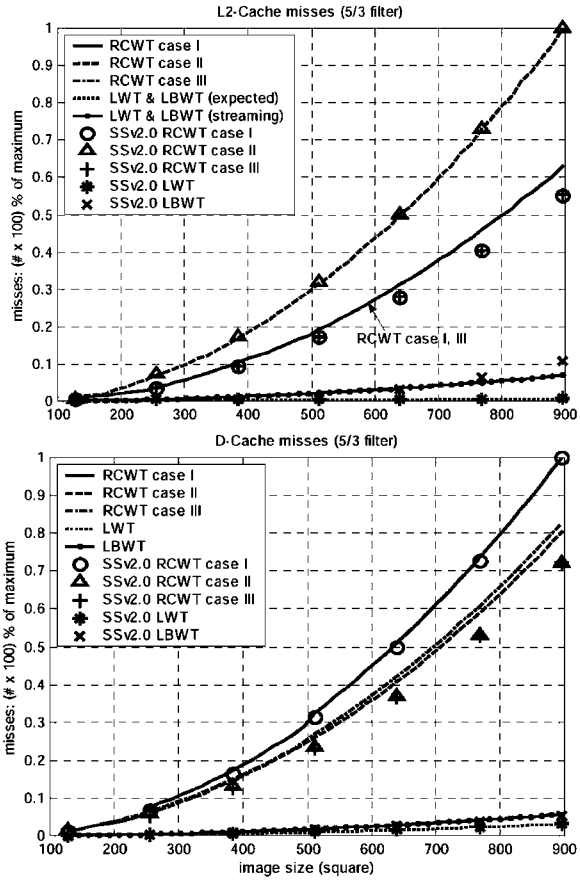

Figure 6. Theoretical (lines) and experimental (points) data-related cache misses, as measured with simulator tools, for the 5/3 filter-pair. SS denotes the SimpleScalar simulator toolset of [17].

the image size increases (Eq. (5)). However, since the image I/O from/to the storage medium is not measured by the simulation tools, the experimental results follow Eq. (6) instead. An abnormality is seen in the $L2$-Cache misses of the LBWT method, where the experimental points do not follow the theoretical curve; instead an exponential increase is observed. This difference is explained from the fact that for the LBWT method, for large image sizes and 5 or 6 decomposition levels, the *IPM* size becomes larger than the $L2$-Cache size; thus the assumption that the complete *IPM* is resident in the $L2$-Cache does not hold any more and the theoretical calculations fail to comply with the actual measurements. This is especially noticeable for the 9/7 filter-pair (Fig. 5) since there the *IPM* is two times larger than for the 5/3 filter-pair (Fig. 6).

To show the relative comparison between instruction and data-related cache misses, in Table 1 the average misses for the various system caches are displayed for the SBF category for a variety of cases. The ratios of Table 1 show that, under the constraints of (1) and (2), the classical implementation of the wavelet transform is a data-dominated application; the misses in the I-Cache are always 1–4 orders of magnitude less than the ones reported for the D-Cache. These facts validate our assumptions that the data-related cache penalties play a dominant role in the transform execution-speed.

To show the effectiveness of the different approaches in real systems, the Intel-Pentium processor [18] was selected, since it has a typical, widely used, superscalar architecture. The processor cache is a two-level 4-way set-associative cache. The first level consists of separate instruction and data cache of size 8 Kb each, while the second level has a joint organization of the instruction and data cache with total size 256 Kb. The blocksize is 32 bytes. As a result, significant differences exist between this cache organization and the cache model of

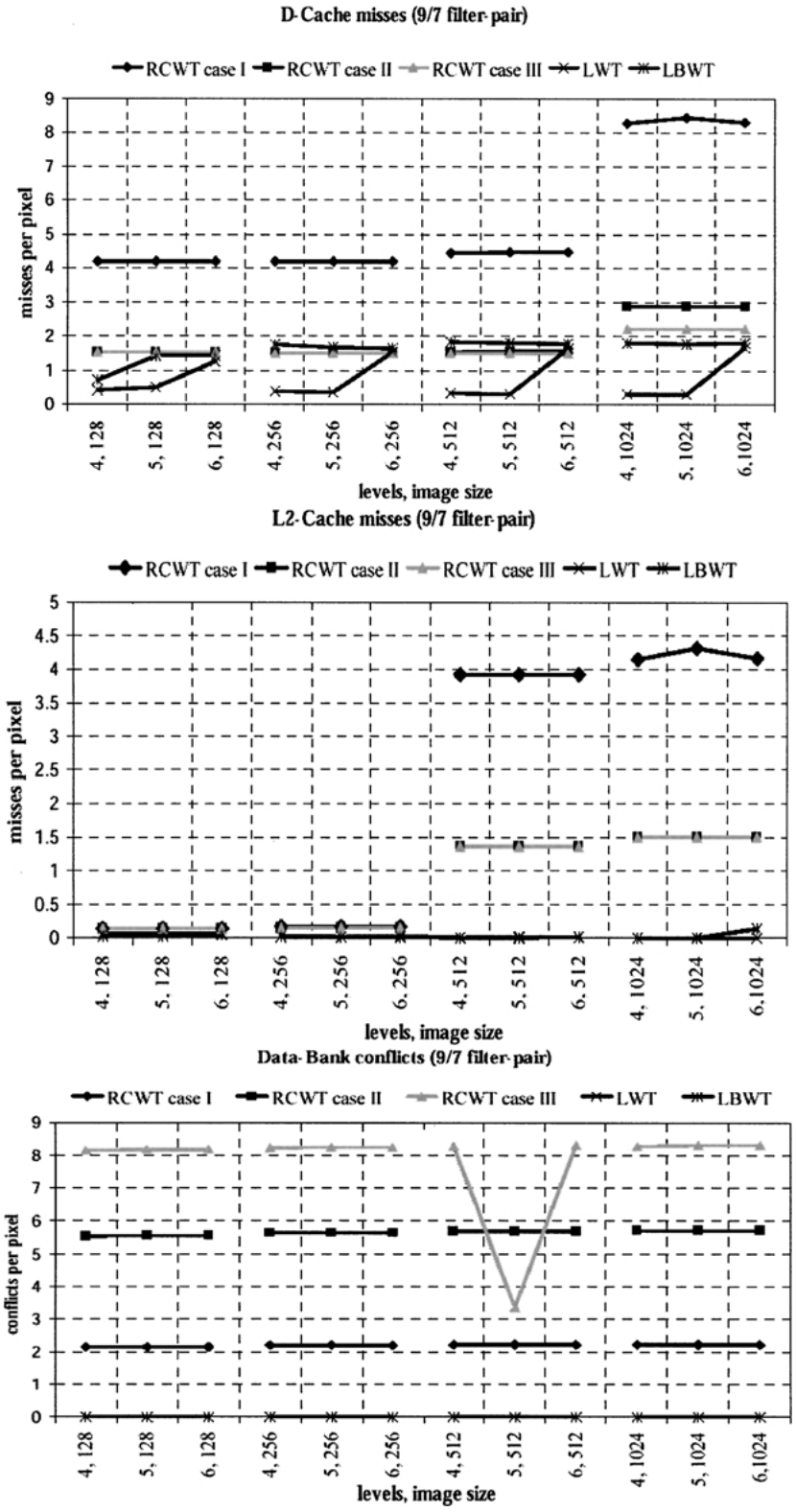

Figure 7. Experimental data-related penalties per pixel in the Intel Pentium platform. The 9/7 filter-pair was used.

the simplescalar processor that was used for the verification of the theoretical analysis. In the Pentium cache, apart from the capacity misses and compulsory misses, the total miss-rate comes also from conflict misses. In addition, the data-bank conflicts reflect the effects of the multiple Functional Units, because they refer to conflicts in the *D*-Cache caused by simultaneous accesses to the same data-bank by the execution-pipelines (U and V pipes [18]).

In Fig. 7, the experimental results in the Pentium architecture are displayed as cache-miss rate or conflict-miss rate versus image-size, decomposition-levels graphs. In addition, the theoretical calculations of this case, based on the equations of Section 3, are shown in Fig. 8. All the experimental results of Fig. 7 were produced by sampling the processor registers during the dedicated, single-process execution using the tool of [19] for increased accuracy. Only the specific portions of the source codes that produce the transform are profiled, in order to avoid any system-dependent I/O procedures.

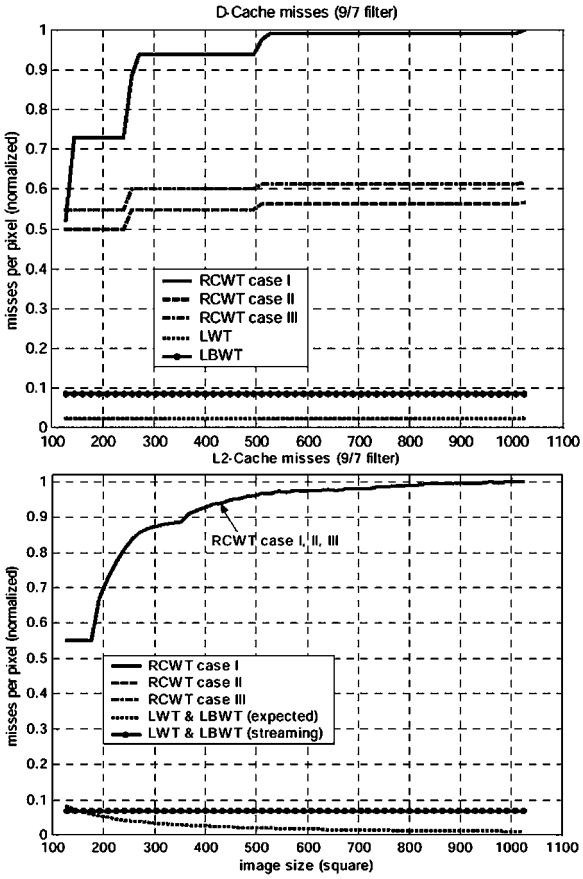

Figure 8. Theoretical calculations based on the fully-associative model for the application of the 9/7 filter-pair with the Intel-Pentium cache-characteristics (cache size and blocksize).

For the *D*-Cache, if one takes into account the data-bank conflicts shown in Fig. 7 (which will also produce stall cycles), then the theoretical (predicted) ordering of the methods agrees with what is seen in the experiments, although not numerically (as for the simplescalar processor). In addition, in the RDF category, both methods appear to minimize the bank conflicts, mainly because the localization of the processing into separate small memory components (*OM*, *FF*, etc.) helps during the parallel execution, since the simultaneously executed commands are expected to access memory arrays that are usually resident in different (non-conflicting) banks.

For the *L*2-Cache, the theoretical calculations of Fig. 8 are not in complete agreement with the experimental results of Fig. 7 with respect to the ordering of the methods for the cache misses, mainly because this cache is a joint organization of instruction and data caches, thus the miss penalties are affected also by the instruction-related behavior of every method. Nevertheless, in general, the RDF category again appears to minimize the measured penalties in comparison to the SBF methods.

5. Conclusions

In this paper, it has been shown that the data-related cache penalties are expected to dominate the execution of the 2-D multilevel DWT in typical programmable platforms since they are by far more frequent than the instruction-related cache penalties. In Table 2 we list the conclusions drawn from the presented experiments with respect to the software design and the cache efficiency of the various DWT-production methods. From the relative comparisons shown in Table 2, one can identify the various trade-offs for the software design of DWT systems. However, apart from quantitative results, this paper presents also a theoretical framework for the analytical estimation of the data-related misses, which can be utilized for high-level parametrical estimation of the DWT implementation-efficiency in a variety of cache architectures. This was verified by simulation results in a conflict-free cache model and partly by experimental results from a real superscalar architecture with a 2-level, set-associative cache. Software relevant to this paper as well as further experimental results can be found in [20].

Table 2. Relative comparisons of the various methods of the two categories with respect to cache efficiency and design flexibility. Image sizes of 128×128 up to 1024×1024 are considered and the reported numbers are percentages that are normalized to the maximum measurement.

Coefficient-production scheme	Strictly breadth-first (RCWT)			Roughly depth-first (RDF)	
Software implementation	Case 1	Case 2	Case 3	LWT	LBWT
C-code design complexity	low	low	low	high	high
Average assembly-code size in two systems: superscalar (Pentium)/VLIW (TriMedia)	72	72	72	100	86
Expected gain from platform-dependent optimizations	low	low	medium	high	high
Cache behavior: Simulator [17] % average → 5/3, 9/7, 4–6 Levels					
Level-1 Cache					
I-Cache accesses	64	53	61	100	74
Instruction-related misses	12	12	12	100	65
D-Cache accesses	100	78	93	94	82
Data-related misses	100	63	66	4	6
prediction-accuracy of proposed model	very good	very good	very good	very good	very good
Level-2 Cache					
Instruction-related misses	54	54	54	100	65
Data-related misses	28	100	20	1	10
prediction-accuracy of proposed model	very good	very good	very good	good	constraint violation
Cache behavior: Real platform [18] % average → 5/3, 9/7, 4–6 Levels					
Level-1 Cache					
Instruction-related misses	2	2	1	100	6
Data-related misses	100	47	29	9	29
model validation—prediction of ordering of the methods with respect to the data-cache penalties			correct		
Level-2 Cache					
Misses	100	40	38	0.1	1
Note: joint *I* and *D*-Cache					
Model validation—prediction of ordering of the methods with respect the cache misses			partially correct, not fully verified		

Appendix 1: Templates for the Calculation of the Data-Related Cache Misses

For the various cases that follow, the term processing means the input of the specific array elements in the cache, the application of the filtering (for the convolution-based implementation) or a number of predict-and-update steps (for the lifting-scheme implementation), and the storage of the final results. The notation $A \gg B$ denotes that A is *up to one order of magnitude larger* than B, while $A \approx B$ denotes that A is *comparable or smaller* than B. In addition, the notations introduced in Section 3 are used.

***T1*:** *Row-by-row processing of a 2-D array of* $R \times (C \cdot c_p)$ *bytes in a cache of* s_c *bytes with blocksize* b_c.

***Constraint*:** $s_c \gg C \cdot c_p$

***Expected misses*:** $T1(R, C, b_c) = (R \cdot C \cdot c_p)/b_c$. (14)

***Explanation*:** Due to the fact that the processing occurs along a row, all data are taken sequentially from the memory. Thus, the cache blocking strategy will input b_c (useful) bytes simultaneously. Due to the constraint mentioned above, all elements of each row fit in the cache. Thus, if each row is processed many times (many passes per row), no additional misses

are expected per row. Hence, for a total of $(R \cdot C \cdot c_p)$ bytes, the compulsory misses reported in (14) are expected.

T2: *Column-by-column processing of a 2-D array after the completion of the row-by-row processing. The array has $R \times (C \cdot c_p)$ bytes and is inserted in a cache of s_c bytes with blocksize b_c.*

Constraint: $s_c \approx R \cdot C \cdot c_p$ AND $s_c \gg R \cdot b_c$

Expected misses: $T2(R, C, b_c, s_c)$

$$= \sum_{i=1}^{\frac{C \cdot c_p}{b_c}} \left(M_l + \sum_{k=1}^{i} a_k \right). \quad (15)$$

Explanation: The situation after the completion of the row processing can be seen in the left part of Fig. 9. Due to the sequential processing of the rows, $(R - M_l)$ rows of the 2-D array are expected to be resident in the cache. Of course, the least-recently-accessed row of the array may not be fully stored in the cache, but Fig. 9 shows the simple case of a number of *full rows of the array* in the cache in order to facilitate the drawing. The term M_l can be easily calculated since the total cache size (s_c) is known:

$$M_l = (R \cdot C \cdot c_p - s_c)/(C \cdot c_p). \quad (16)$$

The left part of the constraint set for this template assures that almost in all cases $M_l > 0$. In addition, the right part of the constraint assures that any additional passes per column are not expected to cause misses, since the currently-processed column elements are expected to fit in the cache. The on-cache part of the array is stored into the various cache blocks in portions of b_c bytes, as seen in the example of Fig. 9. For the input of the first column, the grouping of the cache blocks will cause the simultaneous input of (b_c/c_p) sequential columns, as the middle of Fig. 9 shows. These blocks are expected to replace a number of rows of the array that were stored at the least-recently accessed blocks of the cache, due to the LRU replacement strategy. The replaced rows are denoted as a_1. As seen in Fig. 9, some of the last elements of the first (b_c/c_p) columns will not be inserted, since they are already on-cache. The number of rows that are replaced can be calculated since the dimensions of the array are known and the number of input blocks of data is $M_l + a_1$:

$$b_c(M_l + a_1) = a_1 \cdot C \cdot c_p$$
$$\Rightarrow a_1 = M_l \cdot b_c/(C \cdot c_p - b_c). \quad (17)$$

Similarly, the right part of Fig. 9 shows the input of the second group of sequential (b_c/c_p) columns. Following the same idea, the term a_2 can be calculated as:

$$b_c(M_l + a_1 + a_2) = a_2 \cdot C \cdot c_p$$
$$\Rightarrow a_2 = b_c(M_l + a_1)/(C \cdot c_p - b_c). \quad (18)$$

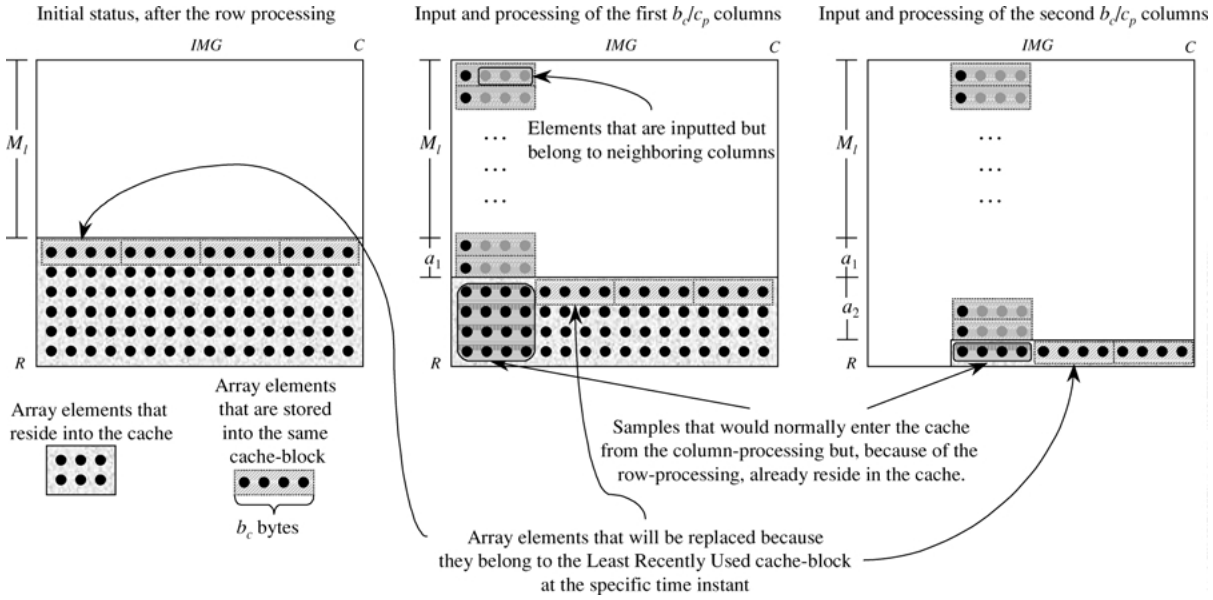

Figure 9. The Column-by-Column input of a 2-D array in the cache, after the completion of the Row-by-Row processing.

Generalizing, for the ith input set of b_{L2}/c_p columns, with $i > 1$:

$$a_i = \left\{ b_c \left(M_l + \sum_{k=1}^{i-1} a_i \right) \Big/ (C \cdot c_p - b_c) : M_l + \sum_{k=1}^{i-1} a_i \leq R; 0: \text{otherwise} \right\}. \quad (19)$$

The reader can notice that a constraint is set in (19): the total size of the ith inserted group of columns must be smaller than the array height (R). Decimal values for the a_i are acceptable in the sense that an incomplete row-replacement was made during the input of the ith set of b_c/c_p columns.

Every replacement of a cache block in this case constitutes a capacity miss. Thus, for the input of the first column, $(M_l + a_1)$ misses are expected. Since s_c is larger than $R \cdot b_c$, the cache is sufficiently large to fit one group of (b_c/c_p) columns and no additional misses are expected to occur for the rest $(b_c/c_p - 1)$ columns. Thus every group of columns stays in the cache after it has been used. This explains the rationale behind the right part of the constraint stated for this template. The total number of groups of columns is $(C \cdot c_p/b_c)$. Hence, the expected amount of misses, under the constraint set, is reported in (15).

***T3*:** *Column-by-column processing of a 2-D array after the completion of the row-by-row processing. The array has $R \times (C \cdot c_p)$ bytes and is inserted in a cache of s_c bytes with blocksize b_c.* Each column is processed k times (k passes per column).

***Constraint*:** $s_c \approx R \cdot b_c$

***Expected misses*:** $T3(R, C, k, s_c) = f_{T3} \cdot R \cdot C$,

with $f_{T3} = \{1 : s_c > R \cdot b_c; k: \text{otherwise}\}$. (20)

***Explanation*:** The template is similar to the previous one, however, the constraint set for this case shows that it applies for a small cache that has comparable size to a group of input columns. After the row-by-row processing, when one column is inserted, a group of columns comes in the cache as shown in the middle of Fig. 9. This column causes R misses for one pass. For k passes, if s_c is smaller than $R \cdot b_c$, a total of $k \cdot R$ misses are expected since this case implies that the entire column does not fit in the cache. If however s_c is larger than $R \cdot b_c$ then, after the first input of each column, no additional misses are expected for the subsequent passes. Thus only R misses are expected for all the k passes. However, since s_c is comparable to $R \cdot b_c$, the subsequent $(b_c/c_p - 1)$ columns of every group will have to be re-inserted (in the vast majority of cases), causing misses as well and as a result a total of $k \cdot R \cdot C$ or $R \cdot C$ misses are expected for the processing of all the groups of columns, depending on the cache size s_c. This is shown in (20) and the distinction between the two cases is made with the factor f_{T3}.

References

1. ISO/IEC JTC 1/SC29/WG1, FCD 15444-1, "JPEG 2000 Image Coding System".
2. ISO/IEC JTC1/SC29/WG11, FCD 14496-1, "Coding of Moving Pictures and Audio".
3. P. Schelkens, "Multidimensional Wavelet Image Coding," Ph.D. Thesis, Vrije Universiteit Brussel, Dept. ETRO, 2001.
4. C. Chrysafis and A. Ortega, "Line-Based, Reduced Memory, Wavelet Image Compression," *IEEE Trans. Image Proc.*, vol. 9, no. 3, 2000, pp. 378–389.
5. E. Ordentlich, D. Taubman, M. Weinberger, and G. Seroussi, "Memory Efficient Scalable Line-Based Image Coding," in *Proc. of the 1999 Data Compression Conference'99*, 1999, pp. 218–227.
6. G. Lafruit, L. Nachtergaele, J. Bormans, M. Engels, and I. Bolsens, "Optimal Memory Organization for Scalable Texture Codecs in MPEG-4," *IEEE Trans. Circuits & Systems for Video Tech.*, vol. 9, no. 2, 1999, pp. 218–243.
7. M. Vishwanath, "The Recursive Pyramid Algorithm for the Discrete Wavelet Transform," *IEEE Trans. Signal Proc.*, vol. 42, 1994, pp. 673–676.
8. G. Lafruit, L. Nachtergaele, B. Vanhoof, and F. Catthoor, "The Local Wavelet Transform: A Memory-Efficient, High-Speed Architecture Optimized to a Region-Oriented Zero-Tree Coder," *Integrated Computer-Aided Engineering*, vol. 7, no. 2, 2000, pp. 89–103.
9. Y. Andreopoulos, N.D. Zervas, P. Schelkens, T. Stouraitis, C.E. Goutis, and J. Cornelis, "A Wavelet-Tree Image-Coding System with Efficient Memory Utilization," in *Proc. of the 2001 IEEE International Conf. on Accoustics Speech and Signal Processing*, vol. 3, 2001, pp. 1709–1712.
10. C. Chakrabarti and C. Mumford, "Efficient Realizations of Encoders and Decoders Based on the 2-D Discrete Wavelet Transforms," *IEEE Trans. VLSI Syst.*, vol. 7, no. 3, 1999, pp. 289–298.
11. Y. Andreopoulos, P. Schelkens, and J. Cornelis, "Analysis of Wavelet-Transform Implementations for Image and Texture Coding Applications in Programmable Platforms," in *Proc. of the 2001 IEEE Signal Processing Systems*, 2001, pp. 273–284.
12. F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, "Custom Memory Management Methodology—Exploration of Memory Organisation for Embedded Multimedia System Design," ISBN 0-7923-8288-9, Kluwer Academic Publishers, Boston, 1998.
13. P. Pirsch, H.-J. Stolberg, Y.-K. Chen, and S.Y. Kung, "Implementation of Media Processors," *IEEE Signal Processing Magazine*, no. 4, 1997, pp. 48–51.

14. S.K. Przybylski, *Cache and Memory Hierarchy Design—A Performance-Directed Approach*, San Fransisco: Morgan-Kaufmann, 1990.
15. H. Komi and A. Ortega, "Analysis of Cache Efficiency in 2-D Wavelet Transform," in *Proc. of the 2001 IEEE International Conference on Multimedia and Expo*, paper no TP11.06, 2001.
16. W. Sweldens, "Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions," in *Proc. SPIE-2569, Wavelet Applications in Signal and Image Processing III*, A.F. Laine and M. Unser (Eds.), pp. 68–79, 1995.
17. D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0," Tech. Report #1342, Un. of Winsconsin-Madison, Computer Sciences Dept., http://www.simplescalar.org.
18. Intel Corp., "Intel's Architecture Software Developer's Manual—Volume 1," 1997.
19. Intel Corp., "VTune Performance Analyzer v4.5," http://developer.intel.com/vtune.
20. *Cache Modeling for DWT Implementations*, http://www.etro.vub.ac.be, in web-pages related to multimedia activities.

Yiannis Andreopoulos received the Electrical Engineering Diploma and a M.Sc. degree in signal and image-processing systems from the University of Patras, Patras, Greece. Since October 2000, he has been working toward the Ph.D. degree at Vrije Universiteit Brussel, Brussels, Belgium. His research interests are in the fields of transforms, fast algorithms and video coding, specializing in combined algorithmic and implementation topics for hardware and software systems. He is a delegate of the ISO/IEC JTC1/SC29/WG11 (MPEG) committee and a student member of the IEEE.
yandreop@etro.vub.ac.be

Peter Schelkens received his Electrical Engineering degree in Applied Physics in 1994, his degree of Medical Physicist in 1995 and his Ph.D. in Applied Sciences in 2001 from the Vrije Universiteit Brussel (VUB). Since October 1994 he has been a member of the IMEC-ETRO laboratory at the VUB, where he had a position as assistant of Prof. Jan Cornelis. Since 2000, he is leading a research team in the field of image and video compression, and related multimedia technologies. The team's research focus is both on algorithmic and implementation aspects. The team is strongly involved in the ISO/IEC JTC1/SC29/WG1 (JPEG2000) and WG11 (MPEG) committees. Peter Schelkens is also the Belgian head of delegation for the ISO/IEC JPEG standardization committee, and co-editor of part 10 of JPEG2000: "Extensions for Three-Dimensional and Floating Point Data".
pschelke@etro.vub.ac.be

Gauthier Lafruit was a research scientist with the Belgian National Foundation for Scientific Research from 1989 to 1994, being mainly active in the area of wavelet image compression. Subsequently, he was a research assistant at the VUB (Free University of Brussels, Belgium). In 1996, he joined IMEC (Interuniversity MicroElectronics Centre, Leuven, Belgium), where he was involved as Senior Scientist with the design of low-power VLSI for combined JPEG/wavelet compression engines. He is currently the Principal Scientist in the Multimedia Image Compression Systems Group at IMEC. His main interests include progressive transmission in still image, video and 3D object coding, as well as scalability and resource monitoring for advanced 3D applications. In this role, he has made decisive contributions to the standardisation of 3D-implementation complexity management in MPEG-4.
lafruit@imec.be

Konstantinos Masselos was born in 1971 in Athens, Greece. He received a first degree in Electrical Engineering from University of Patras, Greece in 1994 and a M.Sc. degree in VLSI Systems Engineering from University of Manchester Institute of Science and Technology (UMIST), United Kingdom in 1996. For his M.Sc. studies he was awarded a UMIST graduate research scholarship. In April 2000 he got a Ph.D. degree in Electrical and Computer Engineering from University of Patras, Greece. His Ph.D. research was related to high level low power design methodologies for multimedia applications realized on different architectural platforms. From 1997 until 1999 he was associated as a visiting researcher with the System Exploration for Memory and Power (SEMP) group of the Design

technology for integrated Information and Communication Systems (DESICS) division of the Inter-university Micro Electronics Centre (IMEC) in Leuven, Belgium, where he was involved in research related to the ACROPOLIS multimedia pre-compiler. Currently he is with INTRACOM S.A, Greece where he is involved in the realization of wireless communication systems on reconfigurable hardware. His main interests include (hardware and software) optimizing compilers, reconfigurable architectures and related design technologies, power optimization, and design of reusable cores. He is a member of the IEEE and of the Technical Chamber of Greece.
kmas@intracom.gr

Jan P.H. Cornelis (MD 1973, Ph.D. 1980) born in Wilrijk, Belgium, in 1950; professor in electronics, medical imaging and digital image processing at the Vrije Universiteit Brussel—VUB; currently also Vice-rector Research of the VUB. He is director of the department of Electronics and Information Processing—ETRO, at the Faculty of Applied Sciences, and coordinates the research group on image processing and machine vision—IRIS. Current research projects of the IRIS group include: various applications in medical vision, remote sensing, mine- and minefield detection, and design of algorithms and computer architectures for image processing. His current research interest is mainly concentrated in the area of image and video compression.
jpcornel@etro.vub.ac.be