SPATIAL LOCALITY TRADE-OFFS OF WAVELET-BASED APPLICATIONS IN DYNAMIC EXECUTION ENVIRONMENTS

Bert Geelen[*,†,◇], *Aris Ferentinos*[*,‡], *Francky Catthoor*[*,†], *Gauthier Lafruit*[*], *Diederik Verkest*[*,†,¶]

[*] IMEC vzw, Kapeldreef 75, B-3001 Leuven, Belgium, Email: firstname.lastname@imec.be
[†] Department of Electrical Engineering, Katholieke Universiteit Leuven, Belgium
[‡] Department of Electrical and Computer Engineering, University of Patras, Greece
[¶] Department of Electrical Engineering, Vrije Universiteit Brussel, Belgium

ABSTRACT

Future dynamic applications will require new mapping strategies to deliver power-efficient performance. Fully static design-time mappings will not address the unpredictably varying application characteristics and resource requirements. Instead, the platforms will not only need to be programmable in terms of instruction set processors, but also at least partial reconfigurability will be required, while the applications themselves will need to exploit this freedom at run-time to adapt to the dynamism. In this context, it is important for applications to exploit the memory hierarchy under varying memory availability. This paper presents a mapping strategy for wavelet-based applications: depending on the run-time conditions, it switches to different memory optimized instantiations, optimally exploiting temporal and spatial locality under these conditions. A comparison is performed between the gains of fully in-placed lifting-based wavelet transforms, and non in-placed versions with higher spatial locality.

Index Terms— Multimedia systems, Wavelet transforms, Memory Management

1. INTRODUCTION

Portable multimedia applications place stringent requirements on the platforms they run on, which should be energy-efficient for extended battery life, and still provide high performance. Many multimedia applications have both pressing computational as well as huge data storage requirements. Previous studies show that high off-chip memory latencies and energy consumptions are likely to be the limiting factor for future embedded systems [1]. Memory hierarchies have been introduced long ago to improve the data access bandwidth to cope with the growing performance mismatch between processing units and the memory subsystem [2]. Moreover, an SRAM-based domain specific memory hierarchy can be used to reduce the power consumption, as data memory power consumption depends primarily on the access frequency and the size of the data memory [3]. Power savings can be obtained by accessing heavily used data from smaller Level 1 memories instead of large background memories. Due to stringent embedded cost and power limitations, pure hardware-controlled caches are not a desirable option. However, existing software-controlled methods have their limitations, especially in dealing with non-affine and dynamic data dependencies.

The Wavelet Transform (WT) produces a multi-resolution representation of a signal, forming an important but complex component for a new class of scalable applications. In these applications it is possible to successively refine the quality of the reconstructed signal using increasing subsets of the transformed signal. This allows connecting heterogeneous systems to the same network with dynamically varying execution conditions, where each user can download a varying subset of the transformed signal and still achieve a reconstructed signal of optimal quality, according to the system's technical capabilities and the encountered conditions. On the system itself, the application will also have to deal with dynamism at task level by competing for resources with other dynamically generated tasks with real-time constraints, meaning not only the amount of available signal content varies, but also these resources. This offers the freedom to dynamically scale the mapping requirements to the available resources so the battery life is adapted to these varying resources. Obviously, this necessitates real-time mechanisms and mapping guidelines, derived by the compiler flow and added to the middleware.

It is well known that loop transformations change the order in which data items are accessed [4]. Reordering the accesses via loop transformations to bring multiple accesses, or *reuse* of the same data, closer in time leads to improved temporal locality. [5] demonstrated that trade-offs between the amount of temporal locality exploited at different Level 1 memory sizes for different loop transformations or localizations can be exploited under dynamically varying Level 1 availability by switching to the most suitable localization at run-time. In addition to improving the temporal locality, one can reorder accesses and modify placement of data in memory such that data items that are placed close together are also accessed close in time. For cache based systems, good spatial locality means each cache line fetch contains useful data, while for scratchpad memories it results in fewer block transfers by the DMA [6]. The state of the art in general contains very little papers considering the complex and systematic trade-offs which arise for algorithms with complicated non-affine and dynamically changing wavelet-like data-flow graphs, encountered due to input dependencies, prediction modes, scalability layers,... [7, 8]. This paper significantly extends our previous work in [5] to incorporate spatial locality in these trade-offs for more accurate guidelines. It shows the potential for power savings still holds. The analysis is applied to fully in-placed lifting scheme implementations [9], which on one hand require less Level 1 space due to the in-placing, but on the other hand also suffer reduced spatial locality.

This article is organized as follows: Section 2 presents related work. Section 3 gives an overview of the wavelet transform and its implementation methods, while Section 4 extends the analysis to spatial locality. Section 5 applies the analysis on two WT schemes with different spatial locality characteristics, and gives the resulting trade-offs. Section 6 extends the mapping guidelines for WT-based applications to include this analysis, and shows the similar energy gains remain possible. Finally, conclusions are drawn in Section 7.

[◇] Research funded by a Ph.D. grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

2. RELATED WORK

[1] presents an extended Data Transfer and Storage Exploration methodology (DTSE) developed at IMEC and consisting of multiple steps. Of special interest for these experiments is the Data Reuse Exploration step [10] applied in the Memory Hierarchy (MH) tool to find Data Reuse in the code at compile-time and to explore how it can be optimally exploited. [11] presents an automated scheme for improving whole-program locality by applying both loop and data-layout transformations. These optimization schemes are, however, not compatible with the complex index and loop code of the WT.

[12–14] present various memory-optimized execution orders or localizations of the WT, offering various methods to avoid off-chip misses: [12] reduces the cache misses during vertical filtering by computing tiles of merged horizontal and vertical filtering, [13] further avoids misses during the higher WT levels by merging lines of computation over all the WT levels, while [14] offers the same advantages, but by merging in a block-based manner, which corresponds well to further processing blocks. None of these implementations consider dynamically varying memory access or storage requirements, or their impact on the mapping.

In contrast, this paper focuses on energy gains related to the miss-rates of these different localizations for wavelet-based applications. It demonstrates that the execution order leading to the lowest memory-transfers depends on the encountered Level 1 memory space, which varies at run-time in environments with dynamically introduced tasks competing for shared resources. Mapping guidelines are given for the filter sizes employed in JPEG2000, to optimally exploit spatial and temporal locality. This is also a significant extension of our own previous work [5].

3. THE WAVELET TRANSFORM

We here present background required for understanding our main contributions in Sec. 5 and 6. Wavelet-based coding is a powerful enabling technology for scalable applications, such as the JPEG2000 [15] compression standard. It is based on the principles of the Wavelet Transform (WT), which is a special case of a subband transform producing a type of localized time-frequency analysis [16]. Fig. 1 shows an example of the transformed Lena image, as a hierarchy of subimages, grouped in levels. Multiresolution coding can be achieved by selective decoding of transform coefficients related to a certain frequency range or subband image.

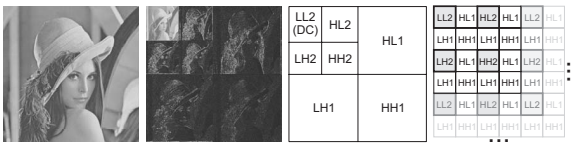

Fig. 1. Lena Input Image, 2 Level Transformed Lena, Regular Output Subband Organization and Extract of In-placed Layout.

In digital compression applications, the WT is traditionally computed as an iterated filter bank [17]. Each level of the FWT consists of a lowpass and a highpass filtering followed by a subsampling operation. The output of the lowpass filtering, after subsampling, represents the input to the next level of the transform, while the highpass samples are directly sent to the output. The iterative filtering or lifting typically leads to code with complex data dependencies and non-linear loop and index expressions. This makes automated optimization strategies difficult to apply to wavelet-based applications.

Though convolution based filtering architectures are possible for performing the WT operations, lifting based implementations offer certain advantages. The lifting scheme is a method for simplifying the WT by decomposing the filters into a set of *prediction* and *update* lifting stages [9], as shown in Fig. 2 for 2 lifting stages. By exploiting the redundancy between the highpass and lowpass filters, the number of computation steps are reduced to almost half those of conventional filtering approaches. Moreover, lifting scheme algorithms do not require temporary arrays in the calculation steps, as the output of the lifting stage may be directly stored "in-place" of the input. Traditional filter bank algorithms require additional memory to store both the original and the transformed signal during the computations. Since after each transform level, one iterates on the lowpass samples, located on the "even" positions of the previous level in a regular row-major layout, this leads to an "interleaved" representation in the wavelet domain (cf. Fig. 1 and 2). This interleaving causes a non-optimal memory access pattern because the WT operates on data that is not contiguous in memory. The combined effect of the reduced memory requirements and the lower spatial locality is evaluated in Section 5.

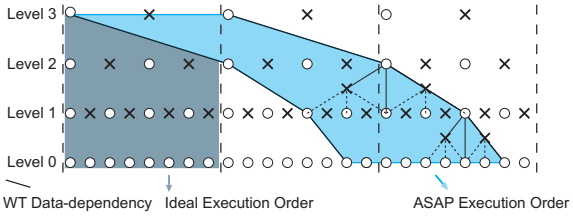

Fig. 2. Block-based 1D-WT, with the skewed As-Soon-As-Possible execution schedule compared to the ideal (but not feasible) schedule.

3.1. Level-by-level vs. Block-based

The execution order of the filtering instructions has a big influence on the memory and energy requirements of the WT. Two important cases are the traditional level-by-level and the block-based localized execution orders. Both can be implemented fully in-placed. The iterative filtering creates strong dependencies between the operations of the various levels of the transformation. The simplest order to respect these dependencies is the traditional level-by-level implementation. Here, the WT is sequentially calculated *level by level*, starting from the original input signal. However, this schedule requires large amounts of memory to store results at intermediate levels. More precisely, the calculations of a particular level start only if all outputs of the previous level have been computed and temporarily stored in memory. Consequently, the required memory size to store the temporary data is equal to the input length. These large amounts of data are unlikely to fit in small, efficient Level 1 (L1) memories and will need to be stored in large L2 memories or even off-chip SDRAM, if on-chip space is restricted for area or power reasons.

The block-based order can avoid these costly background accesses [14]. It is a scheduling which consumes inputs and intermediate results as soon as possible after their creation, so that immediate storage in SDRAM can be avoided. Ideally, for each input block calculation step, the execution schedule prioritizes the vertical order through the wavelet levels. However, because of the wavelet filtering data dependencies mentioned earlier, this is not feasible, and thus a "skewed" schedule is followed. For the 1D WT, the difference between the ideal and the best possible execution schedule is illustrated in Fig. 2. The space required for this ordering is more likely to fit in more efficient L1 memory, leading to a potentially faster and more energy efficient implementation of the WT in comparison to the level-by-level approach, but then the available opportunities for data reuse have to be exploited very effectively.

4. SPATIAL LOCALITY EFFECTS

[5] compared the miss-rates of a level-by-level and block-based implementation, focusing on temporal locality. The block-based implementation showed a superior performance at larger sizes where it can exploit the filtering reuse and the reuse between WT levels. However, if the space required to exploit this reuse is not available, its miss-rate grows larger than that of level-by-level, which can still exploit the filtering reuse at limited L1 sizes. When dynamically introduced, pre-emptable tasks are present, especially in a multi-threading context, this L1 space can be traded off at run-time with other applications, meaning the amount of space varies over time and cannot be statically predicted. If this is handled using static worst-case design strategies, designs will typically be severely overdimensioned making inefficient use of system resources. If the design is instead adapted to the encountered conditions, resources will be exploited more efficiently and energy gains can be obtained. One way of doing this is by switching between execution orders which are more suited for specific L1 memory size ranges. At design-time, the system is profiled to determine the most likely set of operating conditions or scenarios allowing the selection of a minimal set of memory-optimized implementations [18]. At run-time the middleware can then determine what the actual execution scenario is and, using guidelines derived at design-time, switch to the most compatible localization. This requires knowledge of the behavior of the different localizations under different execution conditions and algorithmic settings, as was derived in [5] for temporal locality.

To extend this analysis to spatial locality effects and evaluate different data-layouts such as that of a fully in-placed implementation, we derive the total amount of bytes transferred from L2 memory including the data fetched due to spatial locality mechanisms only: data is fetched from the L2 memory using a granularity of multiple words combined into a line. When a word is accessed, all words situated on the same line are fetched. Therefore the layout of data elements in L2 memory, relative to the partitioning into lines must be considered. This is illustrated in Fig. 3, which gives a top down view of the dependency bands of Fig. 2, for a 2D in-placed WT. The higher WT levels are visible here due to their reduced spatial locality, leading to more redundant data transferred for these levels.

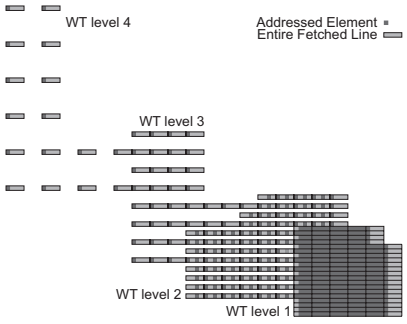

Fig. 3. Top down view of data addressed and fetched for band of blocks of an in-placed 4 level, $9/7$ filter WT.

Increasing the line size will lead to increasing amounts of redundant data transferred and stored. This does not mean large line sizes should be avoided, as the cost of transfers also decreases thanks to the higher communication efficiency. Here we are primarily interested in deriving accurate switching guidelines by comparing the miss-rate performance of different WT implementations to each other, and not so much in determining what the optimal line size is for the WT. This would require evaluating the trade-off between the increased miss-rates and gains obtained in controller complexity, communication efficiency, . . . As indicated in Sec. 1, the locality can be influenced both by data layout and loop transformations. Modifying the data layout will have an impact on the following functional blocks of the application, e.g. the bitplane coder in JPEG2000 consuming data in a stripe like manner [15]. In this paper we analyze the WT in isolation to the rest of the functional components. For a global optimization, we can also evaluate costs of switching data layouts between blocks or the effect of different layouts on the consecutive functional blocks.

5. IMPACT OF IN-PLACING ON MISS-RATES

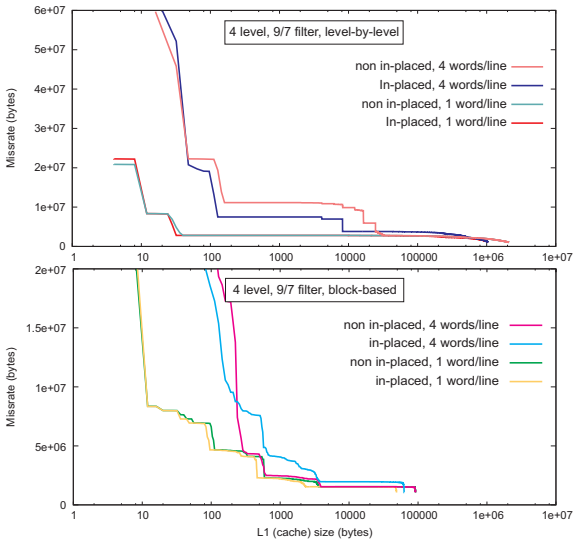

Fig. 4. Read miss-rates for a 512×512 image of level-by-level and block-based WT, at 1 and 4 words/line.

Larger line sizes are essential for modern memories to increase the access bandwidth, and to reduce the access-related latency per word and energy per word. At larger line sizes, spatial locality starts contributing to the miss-rate trade-off for different execution orders. Fig. 4 illustrates this for both execution orders at line sizes of 1 and 4 words per line. The graphs demonstrate that in-placed schemes are preferable at small L1 sizes, whereas non in-placed schemes are superior at larger L1 sizes. A larger line size increases the potential for spatial reuse, but at smaller cache sizes this is more difficult to exploit, leading to more redundant data transfers. What is relevant for our purposes, is how this influences both schemes relatively. In-placed schemes suffer a transfer increase even at large L1 sizes, because the higher WT levels have a low spatial locality which is difficult to exploit. However, these WT levels correspond to a low number of accesses. The lower WT levels, with more accesses, have an increasing spatial locality resulting in a gradual miss-rate increase for decreasing L1 sizes. Non in-placed schemes have a high spatial locality at all levels, which is fully exploited at larger L1 sizes. However, at some cache size, even this locality can no longer be fully exploited, resulting in a sudden steep increase in miss-rates, because the non in-placed schemes then suffer extra read- *and* write-loads which was not the case for in-placed. These phenomena also occur for other WT instantiations, where the increases depend on the instantiation. Therefore this behavior can be summarized in instantiation-dependent mapping guidelines.

6. EXTENDED MAPPING GUIDELINES

To efficiently exploit varying system resources, mapping guidelines should contain information on the sizes where a certain localization

is better, and the gains that can be obtained by switching to another localization. Fig. 5 illustrates this for varying numbers of words per line, for the $9/7$ and $5/3$ filter sizes of JPEG2000 [15].

When evaluating temporal locality alone, level-by-level execution orders were preferable at small L1 sizes, where they are able to exploit more reuse, whereas block-based execution orders manage to exploit more reuse at larger sizes. When also considering spatial locality, this behavior still holds for the execution orders, but at small sizes the miss-rate can be further reduced by applying a fully in-placed scheme. At a line size of 4 words/line, Fig. 5 shows a cross-over point for $5/3$ filters at 165 bytes, where level-by-level incurs 51% less misses before that size, and block-based 81% less after. Likewise for $9/7$ filters we observe a cross-over at 240 bytes, with a 60% lower miss-rate for level-by-level at smaller sizes and a 77% lower miss-rate for block-based at larger sizes.

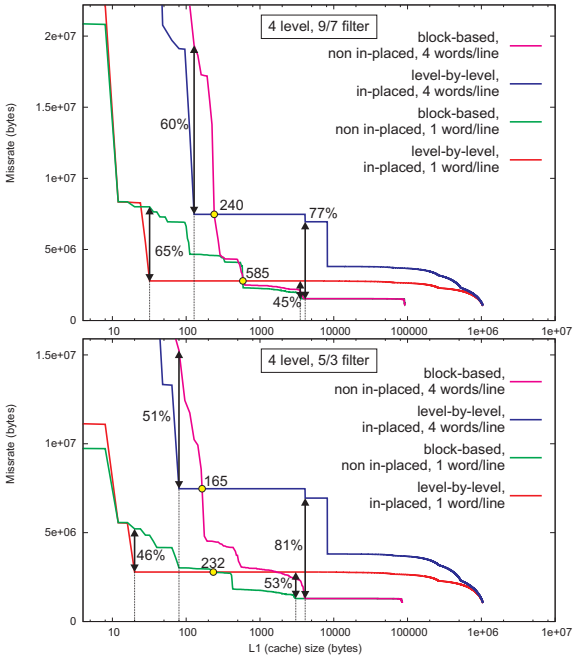

Fig. 5. Read miss-rates for 9/7 and 5/3 WT filter sizes.

Using this design-time generated information, the middleware can make a run-time evaluation, trading off these gains with costs such as the overhead of switching between execution orders. Clearly, the line size has a large impact on the potential gains, and should be taken into account when deriving the guidelines. In practice the size range for which level-by-level is optimal is quite small for these straightforward layouts, so the block-based execution order will be chosen most of the time. At times when there is a large pressure on L1 memory due to applications executing simultaneously or due to dynamism present in the wavelet-based applications themselves and caused by more complicated prediction modes or scalability layers, the middleware can switch to the level-by-level order, thereby avoiding up to 60% of the misses that would be incurred if no switch had been performed. Assuming these peak memory requirements occur in a realistic range of 10-20% of the time, and considering the total miss-rates, this would result in a 28-40% global miss-rate reduction.

7. CONCLUSION

Future dynamic applications will lead to dynamically and unpredictably varying platform resource requirements. In this context it is important for applications to optimally exploit the memory hierarchy under varying memory availability. This paper demonstrates how to achieve this for wavelet-based applications by trading off L1 memory space required for various execution orders with their potential temporal and spatial locality gains, permitting up to 60% lower energy costs in memory accesses. Including spatial locality in the analysis results in more accurate mapping guidelines and allows switching to localizations with better data layout. Using this compile-time generated information, the middleware can make an informed switching decision at run-time. The results can be formalized to be applicable to more general wavelet-based applications.

References

[1] F. Catthoor, S. Wuytack, E. De Greef, et al., *Custom Memory Management Methodology*, Kluwer, 1998.

[2] D.A. Patterson and J.L. Henessy, *Computer Architecture: A quantitative approach*, Morgan Kaufmann Publ., 1996.

[3] B. Amrutur and M. Horowitz, "Speed and power scaling of SRAM's," in *IEEE Journal of Solid-State Circuits*, February 2000, vol. 35.

[4] Utpal K. Banerjee, *Loop Transformations for Restructuring Compilers: The Foundations*, Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[5] B. Geelen, V. Ferentinos, et al., "Exploiting varying resource requirements in wavelet-based applications in dynamic execution enviroments," *Journal of VLSI Signal Processing (subm)*.

[6] Mahmut Kandemir, Ismail Kadayif, et al., "Compiler-directed scratch-pad mem. opt. for embedded multiprocessors," *IEEE Trans. on VLSI Sys.*, vol. 12, no. 3, pp. 281–287, 2004.

[7] V. Ferentinos, B. Geelen, et al., "Adaptive mapping to resource availability for dynamic wavelet-based applications," in *Proc. of ESTIMEDIA*, 2007.

[8] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264 / AVC standard," *IEEE Trans. on Circ. and Sys. for Video Tech.*, September 2007.

[9] W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," in *Wavelet Applications in Signal and Image Processing III*, 1995, pp. 68–79.

[10] T. Van Achteren, R. Lauwereins, and F. Catthoor, "Data reuse exploration techniques for loop-dominated applications," in *Proc. 5th ACM/IEEE DATE Conf.*, April 2002, pp. 428–435.

[11] Mahmut Kandemir, "A compiler technique for improving whole-program locality," in *Proc. of ACM SIGPLAN-SIGACT POPL*, New York, NY, USA, 2001, pp. 179–192.

[12] Gregorio Bernabe et al., "Reducing 3D fast WT execution time using blocking and the streaming SIMD extensions," *Jour. of VLSI Sig. Proc.*, vol. 41, no. 2, pp. 209–223, September 2005.

[13] C. Chrysafis and A. Ortega, "Line-based, reduced memory, wavelet image compression," *IEEE Trans. Image Proc.*, pp. 378–389, March 2000.

[14] G. Lafruit, L. Nachtergaele, and J. Bormans, "Opt. mem. org. for scalable texture codecs in MPEG-4," *IEEE Trans. on Circ. and Syst. for Video Tech.*, , no. 2, pp. 218–243, March 1999.

[15] D.S. Taubman and M.W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*, Kluwer Academic Publishers, 2002.

[16] M. Vetterli and J. Kovacic, *Wavelets and Subband Cod.*, Prentice-Hall, 1995.

[17] S. G. Mallat, "A theory for multires. signal decomposition: The wavelet representation," *IEEE Trans. on Pattern Analysis and Machine Intell.*, vol. 11, no. 7, pp. 674–693, 1989.

[18] Martin Palkovic, Henk Corporaal, et al., "Global memory opt. for emb. sys. allowed by code duplication," in *Proc. of SCOPES*, New York, 2005, pp. 72–79, ACM Press.