An Efficient Software Cache for H.264 Motion Compensation

Arnaldo Azevedo, Ben Juurlink
Computer Engineering Group, Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology, Delft, The Netherlands
Email: {A.P.PereiradeAzevedoFilho, B.H.H.Juurlink}@tudelft.nl

***Abstract*—This paper presents an efficient software cache implementation for H.264 Motion Compensation on scratchpad memory based systems. For a wide range of applications – especially multimedia applications, the data set is predictable, making it possible to transfer the necessary data before the computation. Some kernels, however, depend on data that are known just before they are needed, such as the H.264 Motion Compensation (MC). MC has to stall while the data is transfered from the main memory. To overcome this problem and increase the performance, we analyze the data locality for the MC. Based on this analysis, we propose a 2D Software Cache (2DSC) implementation. The 2DSC exploits the application characteristics to reduce overheads, providing in average 65% improvement over the hand programmed DMAs.**

I. Introduction

Scratchpad memory based processors are very efficient in terms of power and performance [1]. The power efficiency comes from the simple structure of the memory when compared with caches. In a wide range of applications, especially multimedia applications, the data set is predictable, making possible to transfer the necessary data before the computation. These data transfers usually need to be explicitly exposed by the programmer. This structure also makes possible overlapping computation with data transfer by means of double buffering techniques. Scratchpad memories also have predictable latencies. These characteristics make scratchpad memories a common choice for embedded multimedia processors.

In the case of most multimedia applications, the needed data can be determined in advance. However, some kernels depend on data that are known just before they are needed. This is the case, for instance, in the Motion Compensation (MC) kernel of the H.264 video decoding application. As a general idea, motion compensation is the process of copying an area of the reference frame to reconstruct the current frame. For advanced video codecs such as H.264, both the reference frame and the Motion Vectors (MV) need to be calculated. In H.264, this process is known as Motion Vector Prediction (MVP) and is part of the MC. Just after MVP it is possible to request the data necessary to reconstruct the frame. In H.264, MVs can span half of the vertical frame size and it is possible to have up to 16 frames as candidates for reference frame. This makes it impossible to speculatively load all possible areas in advance.

[1]This work was partially supported by the European Commission in the context of the SARC integrated project #27648 (FP6).

The unpredictability of data for MC on scratchpad memory based processors causes two significant problems. The first problem is that the data transfer cannot be overlapped with the computation. The process has to wait for the data to be transferred to the scratchpad memory. Because the H.264 allows very small areas to be copied, up to 4×4 pixels, the waiting time for the data can be significant. The second problem is that data locality cannot easily be exploited. It is difficult to keep track of the memory area present in the scratchpad memory and new data must be requested for each macroblock (MB) partition. Because the MVs are usually small and not randomly distributed, the same area can be copied several times.

In our Cell Processor Synergistic Processing Element (SPE) [2] implementation of the MB decoding, the MC kernel is the most time consuming, representing 62% of the execution time. It requests the reference area through DMA transfers and waits until the data is present in the Local Store (LS). The rest of the execution time is spent in: DMA data in and out (without reference area) 14%, deblocking filter 17%, and idct 7%. These numbers show the importance of improving the MC. The Cell SPE was chosen as the platform of the experiments due to its features for multimedia processing and also due to its power efficiency.

A previous study [3] shows that caching the reference area can save up to 60% of the bandwidth and more than 75% of memory cycles compared to a new request for each reference area. The presented solution is a hardware specific implementation and therefore it is not sufficiently general/flexible to be implemented in a programmable embedded multimedia system.

An alternative is the implementation of a software cache. Implementing a software cache for the Cell processor is a current topic of research in the community [4][5][6]. However, a software cache still has high overhead, representing up to approximately 50% [7] of the application execution time. Such overhead can harm the application performance when compared with hand programmed DMA transfers. Such evaluations are not presented in the cited works.

The current work brings the following contributions:

- An evaluation of the MC data locality;
- An evaluation of the overhead a generic software cache for MC incurs;
- An implementation of an efficient software cache for MC.

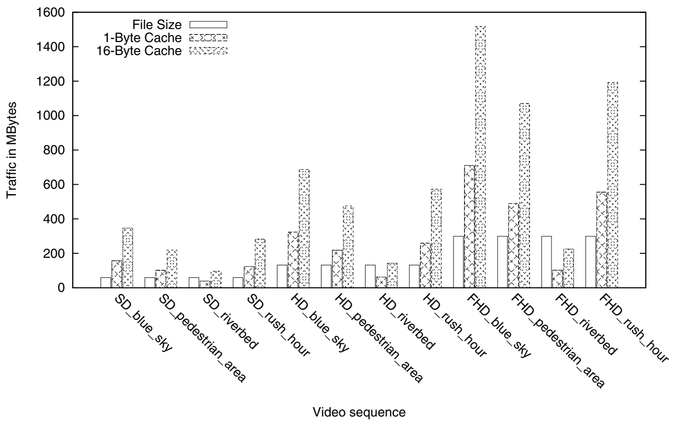

Fig. 1. Data locality

This paper is organized as follows. Section II presents the data locality evaluation of the MC. The software cache implementation and its enhancements are presented in Section III. The results are presented and discussed in Section IV. Conclusions are drawn in Section V.

II. Data Locality in Motion Compensation

In this section, we investigate the data locality for H.264 Motion Compensation. H.264 sequences from HD-VideoBench [8] are used as input for the experiments. Each video sequence is composed by 100 frames in standard (SD), high-definition (HD), and full high-definition (FHD) resolutions at 25 frames per second.

To evaluate the data locality, the number of bytes requested from memory is measured. For the measurement, the motion vectors and reference indexes are extracted from the encoded sequences for each MB partition. Because of the MC quarter-pixel precision, adjacent additional areas need to be fetched from the memory, as follows. For vertical filtering, five extra pixels are required for each line, while for horizontal filtering, five extra lines are required. Details of the MC implementation can be found in [9].

A tool was developed to translate the extracted MVs to memory requests in the DineroIV [10] cache simulator input format. DineroIV was used to report the requested number of bytes for each sequence. Two simulations were performed and the results are reported in Fig. 1. The first simulates a 1-byte cache to depict the temporal locality of MC. The second simulation reports the data traffic for a 16-byte cache with a 16-byte line size.

The results show that the sequences present data locality. In the first case of the 1-byte cache, the data locality is temporal, while in the second case, of the 16-byte cache, the reported data locality is both spatial and temporal. The exceptions regarding the data locality are the Riverbed sequences. These sequences use mostly intra prediction MBs that uses neighboring pixels to predict the area to be reconstructed, thus not making use of the MC. The references for MC use around twice the volume of data of the original sequence (1-byte cache). But, because of memory alignment constraints, the actual volume of transferred data is 3.5 times the volume of the original sequence (as shown by the 16-byte cache result). The presented results suggest that the MC could benefit from a cache to exploit its data locality.

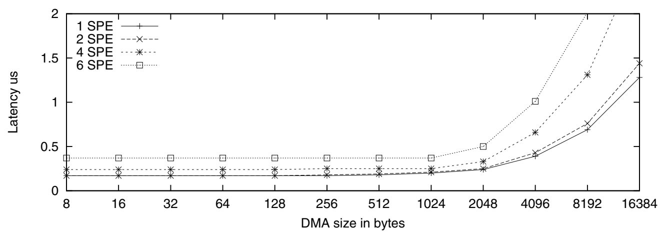

Fig. 2. DMA Latency by transfer size

III. 2D Software Cache

Scratchpad memories are more area and power efficient than hardware caches. However, they require extra effort to be used as they need explicit commands to fetch data from the main memory. These commands can be automatically handled by compilers, but are usually handled by the programmer for better performance or because of lack of tools.

One option for increasing the efficacy of scratchpad memory based systems are software caches. They provide the larger memory abstraction and increase the programmability. Software caches, however, induce additional overhead, which can be prohibitive for embedded applications. This overhead is further increased if the cache does not match the application's data access pattern. This is the case when using a generic cache for MC and for other image processing applications, such as texture mapping. For regular caches, accessing a new image area would result in a new memory request for each line of the new area being accessed. The memory request latencies are approximately the same for 1 to 1024 bytes for the Cell processor, as depicted in Fig. 2. Thus, issuing n requests implies a n fold latency.

In this section, we present the design of a 2D Software Cache (2DSC) for MC. It exploits the access patterns and data locality of the MC and avoids the overheads of regular software cache. The objectives are to increase system perforamnce and the programmability for unpredictable image accesses.

We start with the description of the cache implementation, then we present the design space exploration, and finally we introduce the enhancements for 2DSC.

A. Cache Implementation

The 2DSC stores frame areas instead of cache lines. A cached frame area is called `slice` in this paper. Instead of using memory addresses to find the data in the cache, the 2DSC uses reference frame numbers and the vertical and horizontal coordinates of the MV. This access method makes it possible to exploit the access pattern as it exposes pattern specific information. Each slice of the 2DSC is a $x \times y$ rectangular area of a frame. The x and y values are configurable at runtime.

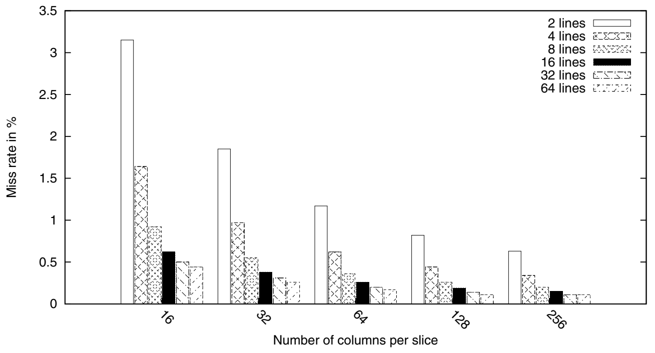

Fig. 3. Miss rate for 2DSC per number of lines

The 2DSC is fully associative as it reduces conflicts. A fully associative cache is possible because of the small number of slices present in the implementation. The 2DSC uses FIFO policy to replace slices when the cache is fully utilized.

The Video frames are stored in YCbCr format instead of the RGB format, and each component is stored in a specific data structure. To increase the compaction, the color components (Cb and Cr) are subsampled to 1:4 as they are less perceptive to the human eye. The MVs are the same for all components, but, because of the subsampling, they need to be adjusted for the Cb and Cr components. The 2DSC implementation exploits this feature and checks and requests at once all the components. This reduces the number of accesses to the 2DSC by 33% and overlaps the memory requests, thus reducing the memory latency.

B. Design Space Exploration

In order to find the best size for the 2DSC slice, a design space exploration was performed. The extracted MVs from the HDVideoBench sequences were used as input.

The 2DSC was tested with the number of slice columns x in the range of powers of 2 from 32 to 256 and slice lines y ranging from 2 to 64. The 2DSC size was fixed at 96KB, 64KB for Y components and 16KB for Cb and Cr, each. The number of slots depends on the size of the slice, from 1024 for 32×2 to 4 for 256×64. The miss rate was calculated for each design point. Fig. 3 depicts the miss rate for each tested point. The results show that the 256×32 slice has a 0.11% miss rate, with the same value for the 128×64 and 256×64 slices. The 256×32 slice was chosen because it requires less DMA requests to be blocked by the DMA request buffer. It result in 8-slot 2DSC. This miss rate reduces the total number of DMA transfers to 32% of the baseline implementation.

For comparison, a similar evaluation was performed for regular hardware caches using the DineroIV simulator. The regular cache was tested with different values for the associativity (1 or 2) and for the size of the cache line (ranging from 16 to 256). The cache size was 128KB and the used replacement policy was LRU. The best performing cache had a 1.3% miss rate. This is only less than half of the worst performing 2D cache and 12× the miss rate of the best performing 2DSC.

C. Enhancements

In this section, we present 4 enhancement strategies to reduce the number of accesses to the 2DSC. The strategies are incremental in the presented order.

1) Extended_X: To reduce the number of accesses to the software cache, an extended line technique was implemented based on the technique described in [9]. The maximum size of an area to be transfered is 21 pixels. This consists of the 16 pixels of the maximum MB partition plus 5 extra pixels for quarter-pixel filtering. Adding these as extra columns for each cache line as non-indexable guarantees that all the data that need to be filtered are present in the cache. This technique incurs an additional 12.5% in the amount of data to be cached for the 256×32 slice. This increases the 2DSC size to 108KB.

2) Extended_XY: This technique is an extension of the Extended_X technique and can be applied only when the vertical span of the slice is equal or larger than 32 lines. Because the maximum MB partition plus the additional area are 21 lines long in the vertical direction, just two accesses to the 2DSC are sufficient to guarantee that the data are present in the cache. Only the first and last lines of the partition need to be accessed.

3) SIMD: Since the Cell SPE is a SIMD architecture, a natural step was to vectorize the tag search. The SPE allows vector operations with 4 32-bit words. In this optimization, four positions of the tag array are compared simultaneously with the searched tag. Once the tag is found, each of the four positions of the tag array are compared individually to find the slice index.

4) Fixed: As previously stated, the parameters for the software cache are configurable at runtime. In this version, the cache parameters were fixed, meaning that the loop boundaries are known at compile time. This allows for certain loop optimizations to be performed, including the elimination of branches and loop unrolling.

IV. Experimental Results

In this section, we present the performance evaluation of the software cache for MC. Fig. 4 depicts the results of the experiments. It shows the time in seconds to access the reference area from main memory to SPE scratchpad for the DMA and different 2DSC enhancements. Our baseline to comparison is the DMA version of the MC. In this version, the reference area for each MB partition is requested via DMA. The result for DMA request time includes the frame border detection, additional quarter-pixel area, and alignment for 128-bit for Cb and Cr components. They are included because they are overlapped with memory transfers. However, they account for less than 1% of the total DMA access time. In the baseline 2DSC implementation, an access to 2DSC is realized for each 16-byte quadword. The $RealTime$ line depicts the required performance for the sequence to be completely decoded in real time, and not only the MC as for the other depicted results. The used sequences are composed of 100 frames and are 4 seconds long (25 fps).

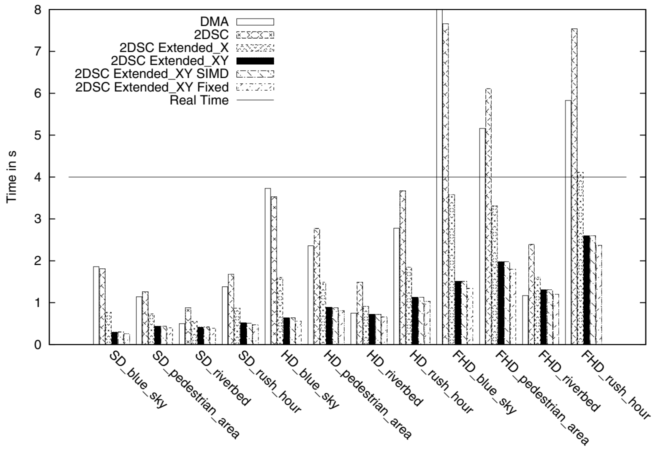

Fig. 4. Performance results

Results show that it is faster to issue a DMA transfer than to use the 2DSC and check for each memory position, because each 2DSC access has to calculate the index and to check for its presence. This overhead is relatively time consuming when compared with the DMA transfers. When the number of accesses is reduced with the Extended_X, an average of 25% improvement over DMA is noticed. Checking only for the beginning and end of the MB partition (Extended_XY) results in only two 2DSC accesses per MB partition. This doubles the efficiency of the 2DSC implementation and reaches an average of 60% reduction in execution time compared with the DMA.

The SIMD version of the 2DSC does not improve further the performance. Its overhead cancels the benefits because of the low number of slices (8) on the implemented 2DSC. Fixing the parameters of the 2DSC adds another 5% execution time reduction, leading to a total of 65% performance improvement.

The results confirm the usefulness of a software cache to exploit the data locality in the MC. Comparing the different access techniques, it is clear that lowering the 2DSC access overhead is crucial to efficiently exploit the data locality.

V. Conclusions

In this work, an efficient implementation of a software cache for H.264 MC on scratchpad memory based systems was presented. The objective of the software cache is to exploit the data locality, reducing the number of memory requests to load reference areas to the SPE scratchpad memory. At first, the data locality of the MC was analyzed. This analysis showed that the MC has a significant amount of data locality to be exploited by a cache. Then the data access pattern of the MC was evaluated and used to design a software cache that exploits it. The proposed software cache stores frame areas instead of memory lines, hence the 2D behavior. It reduces 12 times the number of cache misses, when compared with a regular cache of the same size. Further enhancements were proposed to reduce the number of accesses to the 2DSC and its overhead on the application.

Results show that without tuning the software cache for the application, the performance degrades when compared with hand programmed DMA transfers. This performance degradation is the result of the access overhead to the software cache to check the presence of the desired data. The enhancements proposed in order to reduce the number of accesses to the 2DSC lead to an average of 65% performance improvement over the hand programmed DMA version. In only one test case, the 2DSC did not show any performance improvement over the DMA version. The reason behind this is the lack of data locality in the Riverbed sequence. This experiment shows that while increasing the programmability for scratchpad memory based systems, software caches can degrade performance. The performance degradation can be avoided using application information to reduce the number of memory requests.

As future work, a lightweight hardware accelerator for software caches will be investigated to reduce the 2DSC overhead without significantly increasing the area or power consumption.

References

[1] R. Banakar, S. Steinke, B. sik Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: A design alternative for cache on-chip memory in embedded systems," in *In Tenth inter. Symposium on Hardware/Software Codesign (CODES), Estes Park*. ACM, 2002, pp. 73–78.

[2] M. Gschwind, H. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "Synergistic processing in Cell's multicore architecture," *IEEE Micro*, vol. 26, no. 2, pp. 10–24, 2006.

[3] B. Zatt, A. Azevedo, L. Agostini, A. Susin, and S. Bampi, "Memory hierarchy targeting bi-predictive motion compensation for H.264/AVC decoder," in *Proc. of the IEEE Symposium on VLSI*. IEEE Computer Society Washington, DC, USA, 2007, pp. 445–446.

[4] J. Balart, M. Gonzàlez, X. Martorell, E. Ayguadé, Z. Sura, T. Chen, T. Zhang, K. Obrien, and K. Obrien, "A novel asynchronous software cache implementation for the Cell-BE processor," in *Languages and Compilers for Parallel Computing: 20th inter. Workshop, LCPC 2007, Urbana, Il, USA, October, 2007, Revised Selected Papers*. Springer-Verlag New York Inc, 2007, pp. 125–140.

[5] J. Lee, S. Seo, C. Kim, J. Kim, P. Chun, Z. Sura, J. Kim, and S. Han, "COMIC: a coherent shared memory interface for Cell-BE," in *PACT '08: Proc. of the 17th inter. conf. on Parallel Architectures and Compilation Techniques*. New York, NY, USA: ACM, 2008, pp. 303–314.

[6] T. Chen, T. Zhang, Z. Sura, and M. G. Tallada, "Prefetching irregular references for software cache on Cell," in *CGO '08: Proc. of the sixth annual IEEE/ACM inter. symposium on Code generation and optimization*. New York, NY, USA: ACM, 2008, pp. 155–164.

[7] M. Gonzàlez, N. Vujic, X. Martorell, E. Ayguadé, A. E. Eichenberger, T. Chen, Z. Sura, T. Zhang, K. OBrien, and K. OBrien, "Hybrid access-specific software cache techniques for the Cell BE architecture," in *PACT '08: Proc. of the 17th inter. conf. on Parallel architectures and compilation techniques*. New York, NY, USA: ACM, 2008, pp. 292–302.

[8] M. Alvarez, E. Salami, A. Ramirez, and M. Valero, "HD-VideoBench: a benchmark for evaluating high definition digital video applications," in *Proc. IEEE Int. Symp. on Workload Characterization*, 2007. [Online]. Available: http://personals.ac.upc.edu/alvarez/hdvideobench/index.html

[9] A. Azevedo, B. Zatt, L. Agostini, and S. Bampi, "MoCHA: a bi-predictive motion compensation hardware for H.264/AVC decoder targeting HDTV," in *Circuits and Systems, 2007. ISCAS 2007. IEEE inter. Symposium on*, May 2007, pp. 1617–1620.

[10] J. Edler and M. D. Hill, "Dinero IV trace-driven uniprocessor cache simulator." [Online]. Available: http://pages.cs.wisc.edu/˜ markhill/DineroIV/