Exploiting Parallelism of Deblocking Filter of H.264 on DTA architecture

Roberto Giorgi, Zdravko Popovic, Nikola Puzovic[1]
HiPEAC members
Dept of Information Engineering, University of Siena, Via Roma 56, 53100 Siena, Italy

Arnaldo Azevedo, Ben Juurlink[2]
HiPEAC members
Delft University of Technology Delft, the Netherlands

ABSTRACT
In this paper we present possibilities to parallelize Deblocking Filter (DF) of H.264 video codec and results on Decoupled Threaded Architecture (DTA). We exploited all the available parallelism in the code in order to make it suitable for DTA architecture. Experimental results show that significant speedup can be achieved and that DTA architecture can efficiently exploit available parallelism.

KEYWORDS: H.264, deblocking filter, TLP, many-core, scalability, DTA

1 Introduction

Today's multimedia systems demand more and more computational power since the quality of content that they provide is improving. In particular, users show constant demand for videos with higher resolution even on mobile devices. H.264, also known as MPEG4 part 10 or MPEG-4 AVC (Advanced Video Coding) is a video coding standard aimed at providing high video quality even at lower bitrates.
Many-core architectures have become widely used. Therefore, parallelization of programs that are used for providing multimedia content, such as video codecs, and running them on many-core processors is a promising way to improve the performance. In our work, we have focused on parallelizing Deblocking Filter (DF) of the H.264 codec, and on utilizing the advantages that DTA [1] offers to exploit available Thread Level Parallelism (TLP).
The rest of the paper is organized as follows. Section 2 provides a high-level overview of H.264 deblocking filter, its parallelization possibilities and DTA implementation. Section 3 presents results on the DTA architecture. Conclusions are shown in Section 4. Because of the space limits, we cannot give the description of DTA architecture (there is appropriate reference).

2 H.264 Deblocking Filter

Deblocking filter is one of the steps in encoding and decoding process in H.264 audio/video codec. By profiling of H.264 it can be seen that deblocking filter consumes about 7% of total decoder processing time. Obviously, deblocking filter consumes a significant portion of the decoder and therefore it is important to execute it as efficiently as possible. Steps in H.264 operate on macroblocks (MBs), which are blocks of 16x16 pixels. Because decoding process is block-based, sharp edges may appear between the blocks

[1] http://www.dii.unisi.it/~{giorgi,popovic,puzovic} [2] {azeved, benj} @ ce.et.tudlft.nl

after discrete cosine transformation (DCT) is applied. This is known as "blocking". The purpose of having a deblocking filter is to try to eliminate these artifacts by smoothing the edges of adjacent blocks.

```
void filter_mb( uint8_t *img_y, uint8_t *img_cb, uint8_t *img_cr, unsigned int linesize, unsigned int uvlinesize, int edges[2], int bS[2][4][4], int
qp[2][4], int chroma_qp[2][4], int start[2]) {
  int dir;
/* dir : 0 -> vertical edge, 1 -> horizontal edge */
for( dir = 0; dir < 2; dir++ ) {
      int edge;
      for( edge = start[dir]; edge < edges[dir]; edge++ ) {
        if(bS[dir][edge][0]+bS[dir][edge][1]+bS[dir][edge][2]+bS[dir][edge][3] != 0) {
          if( dir == 0 ) {
            filter_mb_edgev(&img_y[4*edge], linesize, bS[dir][edge], qp[dir][edge] );
            if( (edge&1) == 0 ) {
                    filter_mb_edgecv( &img_cb[2*edge], uvlinesize, bS[dir][edge], chroma_qp[dir][edge] );
                    filter_mb_edgecv( &img_cr[2*edge], uvlinesize, bS[dir][edge], chroma_qp[dir][edge] );
            }
          } else {
                    filter_mb_edgeh(&img_y[4*edge*linesize], linesize, bS[dir][edge], qp[dir][edge]);
                    if( (edge&1) == 0 ) {
                      filter_mb_edgech( &img_cb[2*edge*uvlinesize], uvlinesize, bS[dir][edge], chroma_qp[dir][edge] );
                      filter_mb_edgech( &img_cr[2*edge*uvlinesize], uvlinesize, bS[dir][edge], chroma_qp[dir][edge] );
            }
          }
        }
      }
   }
```

Figure 1 - Main function of deblocking filter.

Here we will give just a rough idea of a deblocking filter process, for more information refer to [2]. In Figure 1 there is a code fragment that shows the main function of sequential filtering process of a single MB (this function is called for every MB). Deblocking filter basically modifies pixels at the edges of macroblocks in cases when they meet certain conditions. Filtering process is done on both vertical and horizontal edges of blocks (first for loop of *filter_mb* function). It starts at the left vertical edge and proceeds at all internal edges. After filtering is done for vertical edges, it is repeated for horizontal edges starting from the top. Filtering is done for all three color components independently (calls to *filter_mb_edgev/h* and *filter_mb_edgecv/h* functions). There are several possibilities to exploit thread level parallelism in the deblocking filter [3]:

- MB level – all MBs that don't have dependencies between them can be processed at the same time. One MB can't be processed before MBs on its left and above it have already been processed (other steps in H.264 introduce additional dependencies, but here we analyze just DF). For example, a frame in CIF resolution of 320x240 pixels, which has 300 MBs, can be processed in total of 34 time slots. Maximal number of MBs that can be processed in parallel is 15 and it lasts for 6 time slots. In higher resolution the number of independent MBs increases (FHD 1920x1080 maximal number is 68, available for 57 out of 187 time slots).
- Color component level – Y, Cb and Cr can be processed in parallel
- 4x4 – at each step in both vertical and horizontal pass, 4 of these blocks are processed. This data level parallelism can be transformed in thread level by processing each of 4 blocks in a separate thread. This is done by unrolling appropriate loops in the code (inside *filter_mb_edgev/h* and *filter_mb_edgecv/h* functions).

We have implemented two versions of deblocking filter for DTA architecture. One is sequential, where MBs are executed one by one and no parallelism is exploited. This code

is for running on a single core only. The other code is parallel and it exploits all three levels of parallelism: independent MBs are processed in parallel, color components are processed in parallel and independent blocks of 4x4 pixels in vertical and horizontal passes are processed in parallel. We have to mention that, depending on input parameters, it is not always possible to exploit all these three levels of parallelism at the same time. Both versions of the code are handwritten. As a reference code, we have used a scalar implementation extracted from [4].

3 Results on DTA architecture

For our tests, we used first eight frames of Lake Wave video sequence. Frame resolution was 320x240 pixels – CIF resolution.

We measured the execution time reduction of each of the first eight frames of Lake Wave example by simply adding more processors in the system (all in a single node). Results are presented in Figure 2 (left side). The speedup was measured using execution time on one processor as a baseline, for both sequential and parallel code. Execution time overhead of parallel code with respect to sequential is very low (Figure 2, right side). For this reason, speedup is very similar in both cases. As it is mentioned in section 2, the number of independent MBs in CIF resolution is at maximum 15 and little less than 9 on average, but it is increasing for higher resolutions. Therefore, from these results, we expect that even better speedups can be achieved for higher resolutions. As stated earlier, not all three levels of parallelism are always available at the same time. That is why scalability is less than it could be expected theoretically.

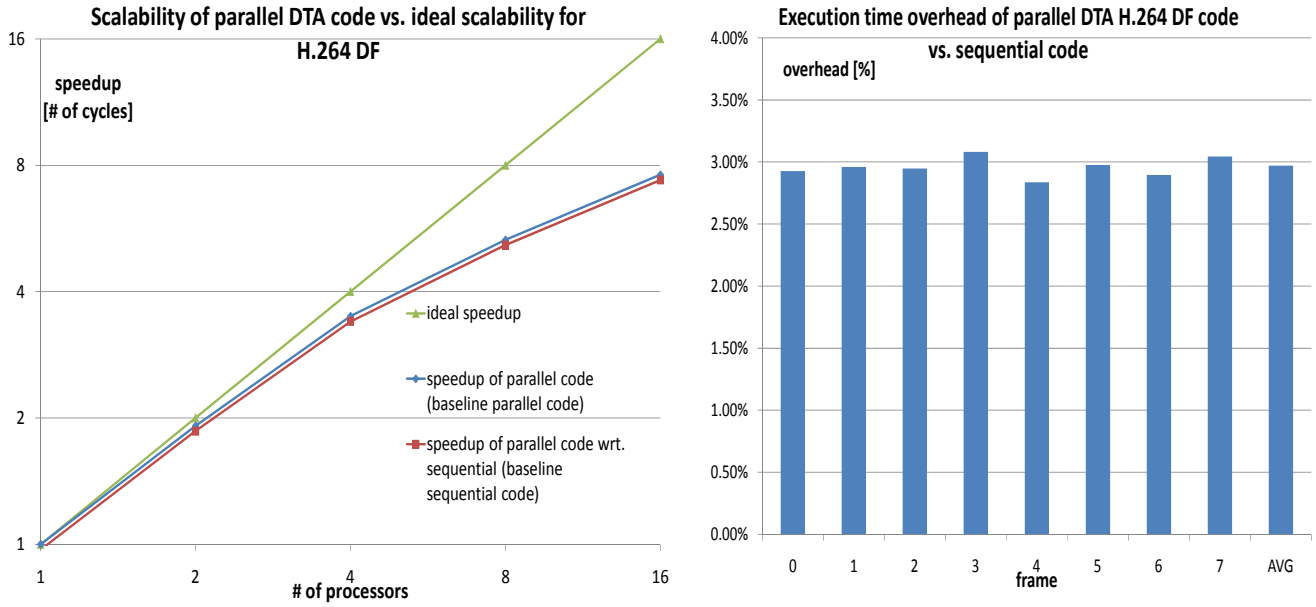

Figure 2 – On the left, speedup of H.264 deblocking filter; DTA parallel code with a different number of processors in a single node vs. ideal case; on the right parallelization overhead.

In Figure 3, we present the contribution of each level of parallelization used in overall speedup. We measured these contributions incrementally. First we analyzed speedup when only processing color components in parallel. Then, we added parallelism at 4x4 block level (MBs processed sequentially), and finally MB level parallelization was included. Available parallelism of color component level is limited by the fact that all three components are not processed in each pass (sub-sampling of Cb and Cr components). On

the other hand, contribution of 4x4 block level parallelism is not at its theoretical maximum because this parallelism is dependent on input parameters (not in every case all blocks are processed) and also it introduces some overhead in order to be exploited. Overall conclusion is that MB-level parallelism is most significant, and it can increase even more with higher resolution, while the other two types of parallelism are expected to remain at the same level.

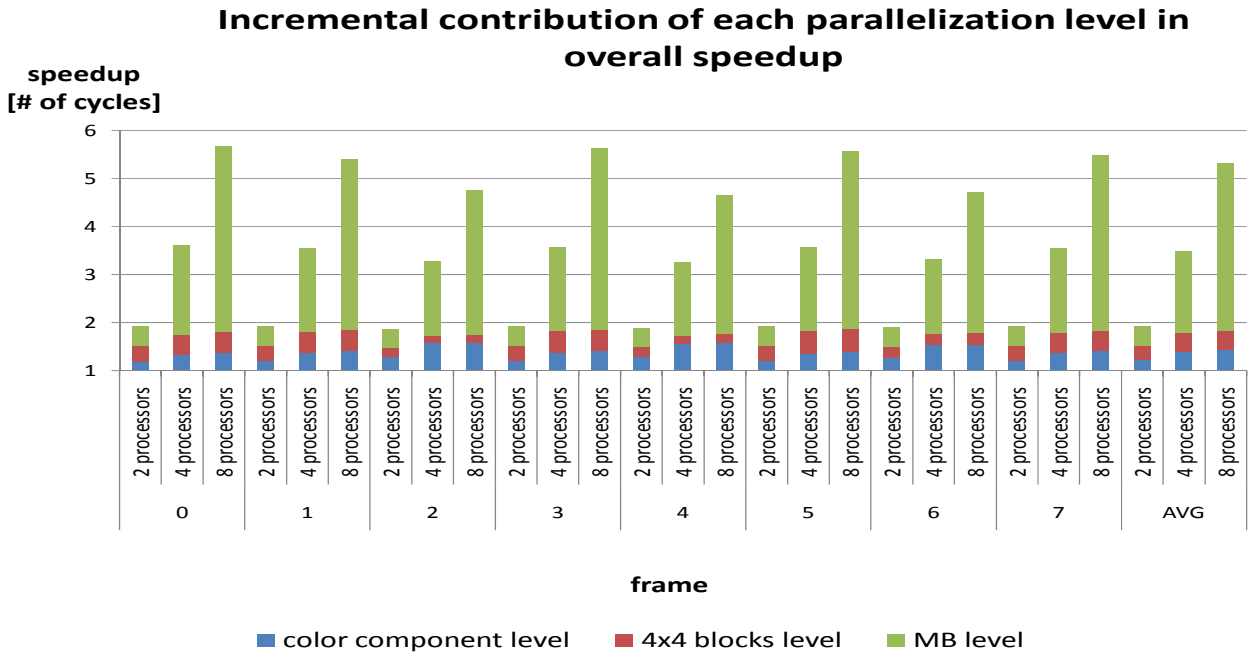

Figure 3 – Contribution of each level of parallelization in overall speedup

Conclusions

In this work we have presented parallelization possibilities of H.264 deblocking filter and its performance on DTA architecture. We have exploited three levels of thread level parallelism: macroblock level, color component level and parallel processing of portions of macroblocks. We wrote parallel code for DTA by hand and executed it on a cycle accurate simulator. The results show that scalability of the architecture is very good.

As our future work, we plan to perform these tests on DTA architecture with more realistic memory system and with higher resolution inputs as well. A paper with more detailed results of these tests is to be published soon.

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

References

[1] R. Giorgi, Z. Popovic, and N. Puzovic, "DTA-C: A Decoupled multi-Threaded Architecture for CMP Systems," in *Proceedings of IEEE SBAC-PAD*, Gramado, Brasil, 2007, pp. 263-270.

[2] P. List, A. Joch, J. Lainema, G. Bjntegaard, and M. Karczewicz, "Adaptive deblocking filter," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, pp. 614-619, July 2003.

[3] C. H. Meenderinck, A. Azevedo, M. Alvarez, B. H. H. Juurlink, and A. Ramirez, "Parallel Scalability of H.264," in *Proceedings of the first Workshop on Programmability Issues for Multi-Core Computers*, 2008.

[4] "The FFmpeg Libavcoded," http://ffmpeg.mplayerhq.hu/.