Multi-level Parallelization of Advanced Video Coding on Hybrid CPU+GPU Platforms

Svetislav Momcilovic, Nuno Roma, and Leonel Sousa

INESC-ID / IST-TU Lisbon,
Rua Alves Redol, 9, 1000-029 Lisboa, Portugal

Abstract. A dynamic model for parallel H.264/AVC video encoding on hybrid GPU+CPU systems is proposed. The entire inter-prediction loop of the encoder is parallelized on both the CPU and the GPU, and a computationally efficient model is proposed to dynamically distribute the computational load among these processing devices on hybrid platforms. The presented model includes both dependency aware task scheduling and load balancing algorithms. According to the obtained experimental results, the proposed dynamic load balancing model is able to push forward the computational capabilities of these hybrid parallel platforms, achieving a speedup of up to 2 when compared with other equivalent state-of-the-art solutions. With the presented implementation, it was possible to encode 25 frames per second for HD 1920×1080 resolution, even when exhaustive motion estimation is considered.

1 Introduction

The increasing demand for high quality video communication, as well as the tremendous growth of video contents on Internet and local storages, stimulated the development of highly efficient compression methods. When compared to previous standards, H.264/AVC achieves compression gains of about 50%, keeping the same quality of the reconstructed video [1]. However, such efficiency comes at the cost of a dramatic increase of the computational demand, making real-time encoding hard to be achieved on single-core Central Processor Units (CPUs).

On the other hand, the latest generations of commodity computers, often equipped with both multi-core CPUs and many-core Graphic Processor Units (GPUs), offer high computing performances to execute a broad set of signal processing algorithms. However, even though these devices are able to run asynchronously, efficient parallelization models are needed in order to maximally exploit the computational power offered by these concurrently running devices. Such models must assure the inherent data dependencies in the parallelized algorithms, as well as a load balanced execution in the processing devices.

Recently, several proposals have been presented to implement parallel video encoders on GPUs [2–5]. However, most of them were only focused on a single encoding module. On the other hand, the task partitioning between the CPU and the GPU of these hybrid systems [6] has been referred to as the main challenge for efficient video coding on current commodity computers. The adopted

I. Caragiannis et al. (Eds.): Euro-Par 2012 Workshops, LNCS 7640, pp. 165–174, 2013.

approaches [7, 8] usually offload most computationally intensive parts to the GPU, keeping the rest of the modules to be executed in the CPU. However, to the best of the authors' knowledge, there is not yet any proposal that effectively considers a hybrid co-design parallelization approach, where the CPU and the GPU are simultaneously used to implement the whole encoder structure.

By taking this idea in mind, an entirely new parallelization method is proposed in this paper. Such method is based on a dynamic performance prediction for parallel implementation of the entire inter-loop of the encoder, which simultaneously and dynamically exploits the CPU and the GPU computational power. In order to optimize such a hybrid platform, a dependency aware strategy for dynamic task and data distribution is proposed. The presented method relies on a realistic performance model that is built at run-time and improved at each iteration of the algorithm, in order to capture the real system behavior. For such purpose, it exploits several parallelization levels currently available in such a system, ranging from the fine-grained thread-level parallelism on the GPU, to both thread and vector-level parallelization on the CPU side.

2 Dependencies and Profiling Analysis of the H.264/AVC

According to the H.264/AVC standard [9], each frame is divided in multiple Macroblocks (MBs), which are encoded using either an intra- or an inter-prediction mode (see Fig. 1). In the most computationally demanding and frequently applied inter-prediction modes, the best-matching predictor of each MB is searched within already encoded Reference Frames (RFs). This process, denoted as Motion Estimation (ME), considers a further division of each 16×16 pixels MB into sub-blocks, as small as 4×4 pixels. The search procedure is then further refined by interpolating the RFs with half-pixel and quarter-pixel precision. Then, an integer transform is applied to the residual signal, which is subsequently quantized and entropy encoded, before it is sent alongside with the motion vectors (MVs) to the decoder. The decoding process, composed of the dequantization, inverse integer transform and motion compensation, is also implemented in the feedback

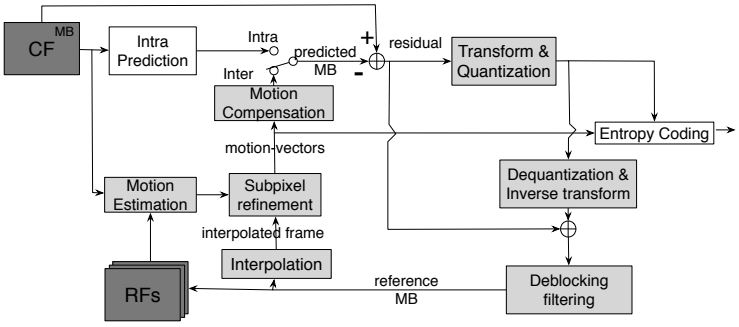
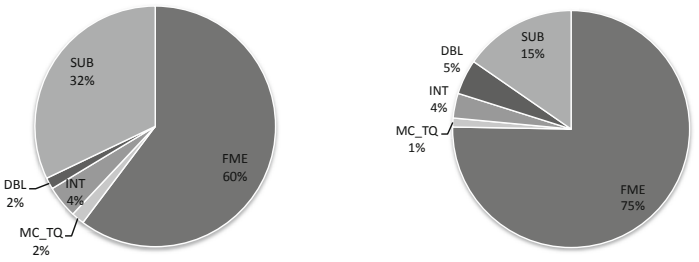


Fig. 1. Block diagram of the H.264/AVC encoder: inter-loop

loop of the encoder in order to locally reconstruct the RFs. A deblocking filter is finally applied to remove blocking artifacts from the reconstructed frame.

Several classes of data dependencies can be identified in this encoding process. *Inter-frame dependencies* exist between different frames in the video sequence. Such dependencies mainly arise from the ME procedure between the current and previously encoded RFs and limit the processing of successive video frames, by forcing them to be sequentially processed. *Intra-frame dependencies* exist between the processing of different regions of the same frame. They mainly exist in the intra-prediction encoding of the MBs, MVs prediction or in the deblocking filtering, when MB edges are filtered using the pixels of neighboring MBs. Functional *dependencies between the H.264/AVC modules* exist when the output data of one module represent the input data of another. For example, the MVs resulting from the ME define the initial search point for the Sub-pixel Motion Estimation Refinement (SME). Similarly, the sub-pixel values, obtained after the interpolation procedure, are the inputs of the SME. On the other hand, the interpolation and the motion estimation do not need to wait for each other, since both of them use the current frame and/or RFs.

From this dependencies analysis, it can be observed that parallel processing can only be considered within the scope of a single frame, since inter-prediction can not start before the list of RFs is updated. Moreover, due to the intra-frame dependencies in the deblocking filter, this module can not be concurrently applied on two different regions of each slice. Hence, the conjunction of all these observations exclude the possibility of dividing each slice in several independent parts to be simultaneously processed in a pipeline fashion. Finally, considering all the functional dependencies between the H.264/AVC modules, it can be concluded that only the interpolation and the ME can be processed in parallel, while the rest of the modules have to be sequentially processed. To simplify the presentation, and without any loss of generality, it will be assumed a single-slice frame configuration for the rest of this paper.



(a) CPU. (b) GPU.

Fig. 2. Breakdown of the H.264/AVC inter-prediction-loop processing time (FME: full-pixel ME; SUB: sub-pixel ME; INT: interpolation; DBL: deblocking filter; MC_TQ: direct transform, quantization, dequantization and inverse transform)

Fig. 2 represents a breakdown of the H.264/AVC processing time for both CPU and GPU implementations, regarding the several encoding modules. These profiling results were obtained for highly optimized implementations on an Intel Core i7 CPU and on an NVIDIA GeForce GTX 580 GPU. As it can be seen, ME is the dominant module, with more than 60% and 75% of the total processing time on CPU and GPU, respectively. Similarly, the SME also occupy a significant amount of the processing time. Finally, it is also observed that the conjunction of motion compensation, integer transform, quantization, dequantization and inverse transform, represented as MC_TQ, only represents about 1%-2% of the total processing time.

3 Performance Model and Task Distribution for Parallel Video Coding on a Hybrid CPU+GPU System

The heterogeneous structure of the H.264/AVC encoder includes modules with very different characteristics regarding the data dependencies and parallelization potential. In this section, it is analyzed the possibility of minimizing the encoding time by efficiently distributing the several tasks on the CPU and on the GPU.

3.1 Load Distribution and Scheduling Strategy

As a consequence of the profiling analysis presented in section 2, the most computationally demanding modules (ME and SME) are distributed among the CPU and GPU. These devices simultaneously execute these operations on different parts of the frame, where the frame division is considered to be at the level of rows/columns of MBs. This distribution is performed in a rather dynamic way, according to the performance level for each device that was evaluated in the previous encoded frame. Since the interpolation can be simultaneously executed with ME, this module is also considered when distributing the ME operation. A module-level scheduling is applied to the rest of the modules, in such a way that the overall processing time is minimized. Since all the other modules that can be concurrently processed (MC_TQ) only take 1% of the total time, their distribution among the CPU and GPU would not offer any significant advantage. Nevertheless, their execution is still evaluated in both processing devices (by using a predefined set of test frames), in order to predict further performance gains.

Algorithm 1 presents the implementation of the proposed method. The most complex steps will be explained in the following subsections. Before the encoding starts, the ME and SME modules are set to be performed on both devices on different halves of the frame, in order to perform a preliminary performance evaluation (lines 1 and 2 initializes the number of MBs assigned to each device). For the same reason, the rest of the modules are also assigned on both devices (line 3) and subsequently implemented in parallel (lines 5-13). Then, according to the measured times (line 14), it is decided which device will perform the interpolation operation (line 15). After that, the number of MB-rows to be sent

Algorithm 1. Scheduling/load balancing algorithm

```
1:  n_me_cpu = n_me_gpu = #MB_rows/2
2:  n_sub_cpu = n_sub_gpu = #MB_rows/2
3:  Assign the rest of the modules to both devices
4:  for frame_nr=0 to nr_of_frames do
5:      in parallel on CPU and GPU do
6:          Load n_me_cpu/gpu rows
7:          Perform ME on n_me_cpu/gpu rows
8:          Perform interpolation on assigned device(s)
9:          Exchange results
10:         Perform SUB on n_sub_cpu/gpu rows
11:         Exchange results
12:         Perform the rest of the modules on assigned device(s)
13:     end in parallel
14:     Update times
15:     Decide device for interpolation
16:     Update n_me_cpu and n_me_gpu
17:     Update n_sub_cpu and n_sub_gpu
18:     if next frame is test-frame then
19:         Assign the remaining modules to both devices
20:     else if frame is the last test-frame then
21:         Perform the scheduling algorithm for remaining modules
22:     end if
23: end for
```

to each device is updated (see section 3.2) for both ME and SME (lines 16 and 17). Finally, whenever the following frame is marked as a test frame, the modules that will not be divided between the processing devices (all except the ME and SME) are assigned to both of them, in order to update their evaluation. Otherwise, they are distributed among the devices, in order to minimize the overall processing time. For this distribution, both the processing time and any eventual data transfer time are considered (see section 3.3).

3.2 Dynamic Load Balancing for Motion Estimation and Sub-pixel Motion Estimation Refinement

The distribution of the large computational load that is involved in the ME and SME modules among the CPU and GPU devices is performed at the level of MB-rows, by sending n_{gpu} rows to the GPU, and $n - n_{gpu}$ rows to the CPU, where n is a total number of MB-rows in each frame. By assuming that the attained performance (s), expressed as the number of processed MB-rows per second, does not depend on the considered distribution (i.e., $s_{gpu}, s_{cpu} = c^{te}$), the ME processing times on the two platforms can be expressed as:

$$t_{gpu\_me} = \frac{n_{gpu}}{s_{gpu}}\ , \quad t_{cpu\_me} = \frac{n - n_{gpu}}{s_{cpu}} \tag{1}$$

Hence, the main aim of the dynamic load balancing is to find the optimal distribution of MBs that will provide the most balanced execution time on the two processing devices, as stated in eq. 2. In this equation, t_{cpu_me} and t_{gpu_me} represent the processing time of the ME module on the CPU and the GPU, respectively. Conversely, t_{gpu0} and t_{cpu0} represent the execution time of the remaining processing modules that are supposed to be implemented on the CPU and GPU, together with the ME. As an example, such set of modules can include the interpolation operation, which does not have any data dependency with the ME and is required to be executed on one of these devices. Hence, the assignment of the interpolation module is done according to the ratio of performances on the two devices (i.e. speedup). In particular, if the interpolation is predicted to have a larger speedup than the ME module when sending it from the CPU to the GPU, it is assigned a greater offloading priority. Otherwise, it remains in the CPU, while the ME is simultaneously performed on the GPU. As soon as the interpolation is finished, the ME starts to be simultaneously executed on both devices.

$$t_{cpu\_me} + t_{cpu0} \approx t_{gpu\_me} + t_{gpu0}, \tag{2}$$

By combining eq. 1 with eq. 2, it is obtained:

$$\frac{n - n_{gpu}}{s_{cpu}} + t_{cpu0} \approx \frac{n_{gpu}}{s_{gpu}} + t_{gpu0}, \tag{3}$$

where:

$$n_{gpu} \approx \frac{n + s_{cpu}(t_{cpu0} - t_{gpu0})}{1 + \frac{s_{cpu}}{s_{gpu}}}. \tag{4}$$

However, the measured performance in any real system varies along the time, not only because of the changes in the conditions it operates on, but also because the inherent processing can be data-dependent. Therefore, if the number of MB-rows that is assigned to each device is computed by assuming the encoding time of a single frame, the obtained distribution will hardly be accurate along the time. As a consequence, the number of MB-rows that is submitted to the GPU should be updated in every iteration:

$$n^i_{gpu} \approx \frac{n - s^{i-1}_{cpu}\Delta t_0}{1 + \frac{s^{i-1}_{cpu}}{s^{i-1}_{gpu}}}, \quad n^i_{cpu} = n - n^i_{gpu} \tag{5}$$

where $\Delta t_0 = t_{cpu0} - t_{gpu0}$ is the signed sum of distribution-independent task portions on the CPU (positive sign) and on the GPU (negative sign). The measured performance values (s^{i-1}_{cpu} and s^{i-1}_{gpu}) are updated upon the encoding of each frame, according to the measured ME processing time on both devices. Hence, this iterative procedure starts with a predicted value (e.g., $n^0_{gpu} = \#\mathrm{MB}_{\mathrm{rows}}/2$) and is updated until it converges to the ideal distribution, based on the measured performance on both processing devices.

The same strategy is applied in the case of the SME module. Since there is no other dominant H.264/AVC module that can be processed in parallel with

SME, the values of t_{cpu0} and t_{gpu0} will be set to zero, while the values of t_{cpu_sme}, n_{cpu_sme}, t_{gpu_sme} and n_{gpu_sme} will be updated along the time.

3.3 Scheduling of the Remaining Modules

As it was described in section 3.1, the least computationally intensive modules of the encoder are scheduled to be completely executed on one of the processing devices. The same happens with the deblocking filter, whose implementation can not be efficiently split by multiple devices. The proposed scheduling scheme is then applied, in order to minimize the overall processing time. For such purpose, all these modules are implemented and evaluated on both the CPU and GPU, and the measured processing times are then used as input parameters for the distribution algorithm, altogether with the data transfer times, required for any module transition between the devices.

The proposed distribution procedure is illustrated in Fig. 3. A data-flow diagram for all the encoding modules, considering both the CPU and GPU, is initially constructed. The transform and quantization tasks, as well as the dequantization and inverse transform, are presented together, due to the low computational requirements and simpler parallelization model. When the measured execution times are considered as a parameterization of each task, a weighted Directed Acyclic Graph (DAG) is obtained. The several nodes of such a graph (A, B ... H) are the decision points, where each task can be submitted to any of the two processing devices. The edges represent the individual task transitions, weighted by the respective computing and data transfer times. The shortest path between the starting and ending nodes of this graph represents the minimum encoding time. Dijkstra's algorithm [10] is typically used to find such a path, by

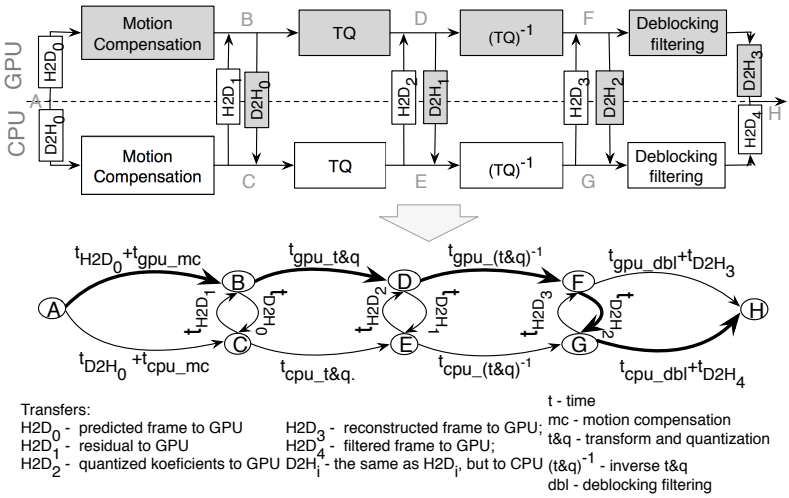


Fig. 3. Construction of weighted DAG from the data-flow diagram of H.264/AVC and a possible minimal path (represented in bold)

defining, for each encoding module of the inter-loop, the processing device on which it will be executed when encoding the subsequent frames. Due to the small number of nodes and edges, the application of this algorithm does not add a significant delay to the encoding procedure.

4 Experimental Results and Evaluation

The validation and evaluation of the proposed dynamic load distribution model was conducted with the H.264/AVC encoder implemented by the JM 17.2 reference software [11]. The considered test video sequences were *blue_sky*, *rush_hour* and *river_bed*, with a spatial resolution of 1920 × 1080 pixels. The ME module was parametrized with a search area of 32×32 pixels. The used evaluation platforms (presented in Table 1) adopted Linux operating system, CUDA 4.1 framework, icc 12.0 compiler and OpenMP 3.0 API to parallelize the video encoder. As it can be seen, *Platform 1* has a slightly faster GPU than *Platform 2*, and a significantly less powerful CPU.

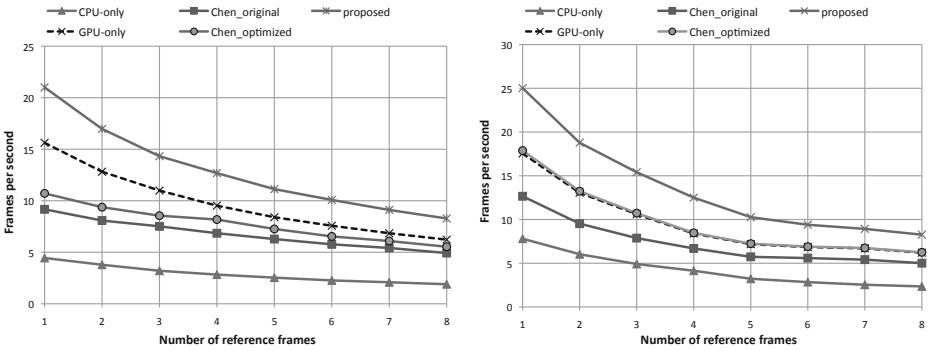
Table 1. Hybrid (CPU+GPU) platforms adopted in the considered evaluation

	Platform 1		Platform 2	
	CPU	GPU	CPU	GPU
Model	Intel Core i7	GeForce 580GTX	Intel Core 2 Quad	GeForce 580GTX
Cores	4	512	4	512
Frequency	3GHz	1.54 GHz	2GHz	1.59GHz
Memory	4GB	1.5GB	4GB	1.5GB

The achieved encoding performance is presented in Fig. 4, for both hybrid platforms. The presented charts compare the resulting performance (in the time per frame (ms)) of five different scheduling strategies:

- *CPU-only* - the whole encoder is implemented in the CPU;
- *GPU-only* - the whole encoder is implemented in the GPU;
- *Chen_original* - method proposed by Chen [7], where the ME, SME and interpolation modules are statically offloaded to the GPU (the rest are kept in the CPU);
- *Chen_optimized* - Chen's encoder [7], optimized with OpenMP and SSE4 vectorization techniques;
- *Proposed* - presented dynamic load distribution strategy.

Contrasting to Chen's approach [7], which considers a static offloading to the GPU of only the ME, SME and interpolation modules, the proposed distribution method combines an adaptive data-level partitioning (see eq. 5) and a dynamic selection of the device that offers the best performance for each of the processing modules of the video encoder (see Fig. 3).



(a) Platform 1. (b) Platform 2.

Fig. 4. Encoded frames per second (fps) for a varying number of RFs, using the 1920×1080 video format. Comparison of the proposed approach with CPU-only, GPU-only and the approach proposed by Chen [7].

The ME module of all these implementations (except the *CPU-only*) adopted the improved search algorithm proposed in [3]. Furthermore, OpenMP parallelization techniques to exploit the available number of CPU cores were extensively considered (except in *GPU-only* and *Chen_original*), as well as a broad set of CUDA optimizations to exploit, as much as possible, the GPU computational resources (except in *CPU-only*).

The performance regarding the frames per second considering different number of RFs are presented in Fig. 4 for two platforms specified in Table 1. As it can be seen, the *Proposed* method achieves speedup levels of up to 1.5 and 2, when compared with the *GPU-only* implementation and with the *Chen_optimized* strategy, respectively, and a speedup of up to 5 when compared with the *CPU-only* implementation. Due to the fact that *Platform 1* has a significantly slower CPU, the *Chen_optimized* strategy achieves a lower performance when compared to the *GPU-only* implementation, while in the case of *Platform 2* their performances are very similar. However, *GPU-only* requires a CUDA implementation of all the H.264/AVC modules of the inter-loop. The impact of the considered OpenMP and SSE4 vectorization [12] optimizations of the CPU code are emphasized when comparing the results obtained for the *Chen_original* and *Chen_optimized* approaches.

Contrasting with *Chen_optimized* and *GPU-only* approaches, the *Proposed* algorithm achieves a higher performance that is less dependent on the adopted hybrid platform. In particular, by simultaneously using the computational resources of the CPU and the GPU devices, in an adaptive and dynamic load balanced fashion, both processing devices participate in a much better distribution of the computational load, based on a constantly updated prediction of the offered performance by each device. Finally, it can be seen that a real time encoding with more than 20 fps is achieved on both platforms for a single RF.

5 Conclusions

A new dynamic load distribution model for hybrid CPU+GPU advanced video encoders was proposed. The distribution is carried out by exploiting both intra/inter task-level and data-level parallelism. The possibility of asynchronously processing on the CPU and GPU is also fully exploited to efficiently distribute the computational load of the most demanding H.264/AVC modules among these devices. A dynamic load balancing strategy is defined based on a performance model that uses prediction techniques from the previous processing results. Based on the proposed model and method, a speedup up to 2 for the total encoding time comparing state-of-the-art approaches was achieved, as well as the ability to encode more than 25 fps for a HD 1920×1080 resolution, considering all the sub-block prediction modes and an exhaustive ME algorithm.

Acknowledgment. This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT), project PEst-OE/EEI/LA0021/2011.

References

1. Wiegand, T., Schwartz, H., Kossentini, F., Ulivan, G.S.: Rate-constrained coder control and comparison of video coding standards. IEEE Transactions on Circuits and Systems for Video Technology 13(7), 668–703 (2003)
2. Schwalb, M., Ewerth, R., Freisleben, B.: Fast motion estimation on graphics hardware for H.264 video encoding. IEEE Transactions on Multimedia 11(1), 1–10 (2009)
3. Momcilovic, S., Sousa, L.: Development and evaluation of scalable video motion estimators on GPU. In: Workshop on Signal Processing Systems, SiPS (October 2009)
4. Obukhov, A., Kharlamovl, A.: Discrete cosine transform for 8x8 blocks with CUDA. Research report, NVIDIA, Santa Clara, CA (February 2008)
5. Pieters, B., et al.: Parallel deblocking filtering in MPEG-4 AVC/H.264 on massively parallel architectures. IEEE Transactions on Circuits and Systems for Video Technology 21(1), 96–100 (2011)
6. Cheung, N.M., Fan, X., Au, O.C., Kung, M.C.: Video coding on multicore graphics processors. IEEE Signal Processing Magazine 27(2), 79–89 (2010)
7. Chen, W.N., Hang, H.M.: H.264/AVC motion estimation implementation on Compute Unified Device Architecture (CUDA). In: International Conference on Multimedia and Expo, ICME, pp. 697–700 (April 2008)
8. Rodriguez-Sánchez, R.: Optimizing H.264/AVC interprediction on a GPU-based framework. Concurrency and Computation: Practice & Experience (2012)
9. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., Stockhammer, T., Wedi, T.: Video coding with H.264/AVC: tools, performance, and complexity. IEEE Circuits and Systems Magazine 4(1), 7–28 (2004)
10. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1(1), 269–271 (1959)
11. ITU-T: JVT Reference Software unofficial version 17.2 (2010), `http://iphome.hhi.de/suehring/tml/download`
12. Intel: SSE4 Programming Reference (2007), `http://edc.intel.com/Link.aspx?id=1630`