Proceedings of the First International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2014)
Porto, Portugal

Jesus Carretero, Javier Garcia Blas
Jorge Barbosa, Ricardo Morla
(Editors)

August 27-28, 2014

Efficient Parallel Video Encoding on Heterogeneous Systems

Svetislav Momcilovic, Aleksandar Ilic, Nuno Roma, Leonel Sousa

INESC-ID / IST-TU Lisbon, Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
{Svetislav.Momcilovic, Aleksandar.Ilic, Nuno.Roma, Leonel.Sousa}@inesc-id.pt

Abstract

In this study we propose an efficient method for collaborative H.264/AVC inter-loop encoding in heterogeneous CPU+GPU systems. This method relies on specifically developed extensive library of highly optimized parallel algorithms for both CPU and GPU architectures, and all inter-loop modules. In order to minimize the overall encoding time, this method integrates adaptive load balancing for the most computationally intensive, inter-prediction modules, which is based on dynamically built functional performance models of heterogenous devices and inter-loop modules. The proposed method also introduces efficient communication-aware techniques, which maximize data reusing, and decrease the overhead of expensive data transfers in collaborative video encoding. The experimental results show that the proposed method is able of achieving real-time video encoding for very demanding video coding parameters, i.e., full HD video format, 64×64 pixels search area and the exhaustive motion estimation.

Keywords video coding, divisible load theory, load-balancing, CPU+GPU computing

I. Introduction

The newest video coding standards, such as H.264/AVC [17] and HEVC/H.265 [16], achieve high compression efficiencies, by relying on advanced encoding techniques (e.g. multiple partitioning modes, large search ranges, quarter-pixel precision). On the other hand, all these techniques dramatically increase the computational requirements, and make real-time encoding of High Definition (HD) video sequences hard to be achieved on any individual device available on modern desktops, such as multi-core Central Processing Units (CPUs) and Graphics Processing Units (GPUs).

To simultaneously employ several heterogeneous devices available on modern desktops for real-time video encoding, an efficient method for collaborative H.264/AVC inter loop on CPU+GPU systems is proposed herein. A unified execution environment was designed to ensure efficient cross-device execution and to guarantee the correctness of the video encoding process. It includes an extensive library of highly optimized parallel algorithms for all inter-loop video encoding modules, which are developed using the device-specific programming models and tools (e.g. CUDA, OpenMP, etc.). In order to minimize the over inter-loop encoding time, the proposed collaborative environment also integrates different scheduling, load balancing and data access management routines.

Highly efficient parallel algorithms are developed for both CPUs and GPUs and for all inter-loop modules, namely Motion Estimation (ME), Sub-Pixel ME (SME), Interpolation (INT), Motion Compensation (MC), Transform and Quantization (TQ), Inverse TQ (TQ^{-1}) and Deblocking Filtering (DBL). The integrated scheduling and load balancing routines allow efficient distribution of the workloads for these modules over all processing devices. For the most computationally intensive modules (i.e., ME, SME and INT), the proposed load balancing, based on Divisible Load Theory (DLT) [20], relies on realistic and dynamically built Functional Performance Models (FPMs) [7,10] of both communication and computation system resources. The workloads of the remaining modules are distributed at the module level by applying the optimal Dijkstra algorithm [4]. Furthermore, the proposed method also includes specific, communication-aware techniques to maximize data reuse, and to decrease the data transfers overhead. Because of a similar algorithmic structure of the video encoding inter-loop, many solutions provided herein can also be applied to HEVC/H.265 encoders.

The obtained experimental results show that the proposed method achieves a real-time inter-loop video encoding for full-HD (1080p) video sequences, when applying exhaustive ME and 64×64 pixels search area (SA) on a commodity desktop platform equipped with a multi-core CPU and two GPUs. To the best of the authors' knowledge, this is one of the first approaches that applies adaptive load balancing with dynamically built partial estimations of the FPMs to tackle efficient collaborative execution of complex multi-module problems, such as video encoding, in heterogenous environments.

II. Related Work

There are only few state-of-the-art approaches that deal with the efficient parallel implementation of the entire video encoder (or its main functional parts), namely, for multi-core CPU [21], GPU [22], or CPU+GPU [14,15] environments. In CPU+GPU systems, these approaches either *i)* simply offload a single inter-loop module in its entirety (mainly the ME) to the GPU, while performing the rest of the encoder on the CPU [9,19], or *ii)* exploit simultaneous CPU+GPU processing at the level of a *single* inter-loop module [15,23].

These approaches have a limited scalability (only one GPU can be employed) and cannot fully exploit the capabilities of CPU+GPU systems (since the CPU is idle, while the GPU processes the entire offloaded module) [19]. In [9] the pipelining granularity is decided through a large set of experiments, while in [23] the cross-device load distribution is found by intersecting the experimentally obtained fitted full performance curves. However, both approaches impose limited scalability over the number of processing devices and coding

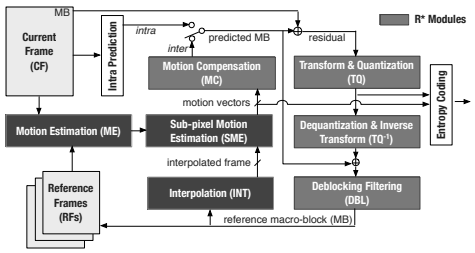


Figure 1: H.264/AVC encoder.

parameters, and, also, introduce huge preprocessing scheduling overheads. The method proposed in [15] use a single GPU and constant compute-only performance parametrization, while in [18] a simple equidistant data partitioning is applied for video encoding in multi-GPU systems, since the CPU is only used to orchestrate the execution across a homogenous set of GPUs.

The methods proposed herein span over three load balancing/scheduling classes for heterogenous environments, namely: simultaneous multi-module load balancing, static DAG-based scheduling and dynamic iterative load balancing. The proposed load balancing method relies on DLT [20] and there are only a few studies targeting the DLT scheduling in CPU+GPU systems either for general [7] or application-specific [1] problems. In [11], the authors apply DLT scheduling for a single-module load distribution in CPU-only cluster environments for a custom video encoder. In [5,13], we proposed load balancing methods that rely on constant performance models and linear programming for determining the cross-device load distributions, while in [12], only the inter-prediction modules were considered.

In this work, for the first time, the real-time video encoding on commodity CPU+GPU platforms is investigated for a complete H.264/AVC inter-loop, where the adaptive iterative load balancing with on-the-fly update of partial FPMs [6,7] is applied at the level of inter-prediction modules. The dynamically built partial estimations of the full FPMs allow realistic and simultaneous modeling of capabilities for communication links and heterogeneous devices during the execution of several inter-loop modules. Moreover, the proposed approach also considers the replication of certain computationally inexpensive encoding modules (i.e., INT) over several processing devices in order to reduce the communication overheads and achieve significant performance gains.

III. Parallel Inter-loop video encoding on CPU and GPU devices

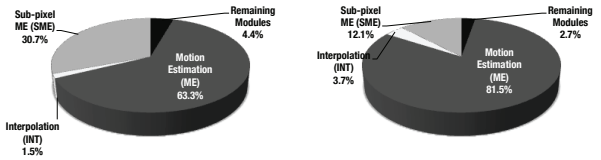
In order to allow an efficient parallelization and collaborative CPU+GPU execution, the inherent data dependences of H.264/AVC inter-loop encoder and the computational requirements of different encoding modules, must be considered. According to the H.264/AVC standard [17], current frame (CF) is divided in multiple square-shaped Macroblock (MB), which are encoded using either an intra- or an inter-prediction mode (see Fig. 1). This standard allows a further subdivision of the MB by considering 7 different partitioning modes, namely 16×16, 16×8, 8×16, 8×8, 8×4, 4×8 and 4×4 pixels. In the most computationally demanding and most frequently applied inter-prediction mode, the prediction of each MB is obtained by searching within already encoded Reference Frames (RFs). This procedure, denoted as ME, is then further refined with previously interpolated Sub-pixel Frames (SFs) from INT module by applying the SME procedure. In the MC module, the residual signal is computed according to the selected MB subdivision mode, which is found as the best trade-off between the size of data required to encode the residual signal and the motion vectors (MVs). The residual is subsequently transformed and quantized in TQ modules, and entropy coded (alongside with the MVs and the mode decision data), before it is sent to the decoder. The decoding process, composed of the TQ^{-1} and DBL, is also implemented in the feedback loop of the encoder, in order to locally reconstruct the RFs.

Parallelization at the level of entire inter-loop imposes several hard-to-solve challenges, which must be explicitly taken into account to ensure the correctness of the overall video encoding procedure. In detail, an efficient parallelization requires the observance of **data dependencies** at several levels: *i)* between consecutive frames, *ii)* within a single video frame, and *iii)* between the inter-loop modules.

In the H.264/AVC inter-loop, the encoding of the CF can not start before the previous frames are encoded and the required RFs are reconstructed. Such a dependency prevents the encoding of several frames in parallel. Moreover, the inherent data dependencies between the neighboring MBs in certain inter-loop modules (such as DBL) also limit the possibility to concurrently perform the entire encoding procedure on different parts of a frame. Hence, efficient module-level pipelined schemes can hardly be adopted, either for parts of the frame or for the entire frame. Furthermore, the output data of one module is often the input data for another (e.g., the MVs from ME define the initial search point for the SME), which imposes additional data dependencies between the inter-loop modules. Hence, the data-dependent inter-loop modules have to be sequentially processed (within a single frame). The only exceptions are ME and INT modules, which can be simultaneously processed, since both of them use the CF and/or the RFs.

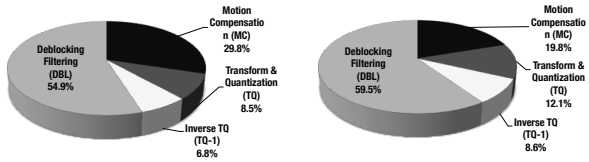
Architecturally different devices in modern CPU+GPU systems impose additional challenges to the **parallelization of individual inter-loop modules**. For example, GPUs require to exploit the fine-gained data-level parallelism suitable for simultaneous processing on hundreds of cores, while for CPU architectures with several general purpose cores the parallelism can be exploited at coarser-grained level. Therefore, it is required to parallelize the modules for each device in the system according to both per-module parallelization potentials and architectural characteristics of devices. For each inter-loop module, the detailed description of parallel algorithms and parallelization techniques applied herein can be found in [13,15].

In what concerns the ME, efficient CPU and GPU parallelizations and collaborative processing the Full-Search Block-Matching (FSBM) is adopted, since the execution pattern of adaptive algorithms highly depends on the video content, which makes the achievable performance hard to be predicted and prevents efficient load balancing. Moreover, the dependency on the video content causes branch divergence for matching candidates examined on different GPU cores, resulting in attaining very poor GPU performance. In fact, even the state-of-the-art approaches dealing with the GPU parallelization of the adaptive algorithms [3] (namely UMHexagonS [2]) were unable of achieving better performance than CPU implementations.



(a) Quad-core Intel i7 950 (CPU_N). (b) NVIDIA GeForce GTX580 (GPU_F).

Figure 2: Execution share of H.264/AVC inter-loop modules in the total encoding time for 1080p HD sequences, 4 RFs and SA of 32×32.



(a) Quad-core Intel i7 950 (Nehalem). (b) NVIDIA GeForce GTX580 (Fermi).

Figure 3: Execution share of remaining H.264/AVC inter-loop modules on different device architectures.

Furthermore, to exploit the fine-grained data-level parallelism required for efficient GPU parallelization and collaborative video encoding in CPU+GPU platforms, it is also required to provide a sufficient amount of data-independent computations that can be simultaneously processed on hundreds of GPU cores. Accordingly, to relax spatial data dependences imposed by the definition of SA center, a set of temporary dependent predictors was analyzed in [12]. It was observed that the best MV found for the 16×16 partitioning mode in the previous frame for the collocated MB represents a good compromise for the SA center predictor. In fact, this predictor is only used herein to compute the SA center, while the selected MVs are then post-computed according to real median vectors of the neighboring MBs.

In order to experimentally assess the **contributions of individual modules** to the overall video encoding time, the initial inter-loop encoding was performed for an 1080p HD video sequence on two different device architectures, namely on quad-core Intel i7 950 processor (Nehalem) and NVIDIA Fermi GeForce GTX580 (Fermi). During the initial evaluation on each device architecture, the parallelized modules were used and the inter-loop encoding was performed with 4 RFs, the SA size of 32×32 pixels, and FSBM. As it can be observed in Fig. 2, the *inter-prediction modules* (i.e., ME+INT+SME) participate with more than 95% in overall encoding time, for both CPU and GPU parallel implementations. Consequently, their efficient execution is crucial to achieve real-time video encoding on target desktop systems. It is worth noting that the participation of the ME in the overall inter-loop share highly depends on the selected encoding parameters, such as the number of RFs, SA size, and the search algorithm. However, the conclusions provided herein can be equally applied to any selected parameters, considering the clear dominance of the inter-prediction modules.

For simplicity, the *Remaining Modules*, i.e., MC, TQ, TQ^{-1} and DBL, are referred herein as R* modules and their share in the overall encoding time is typically less than 5% (see Fig. 2). In addition, Fig. 3 depicts the breakdown of the computational requirements for each R* module. As it can be observed, in both CPU and GPU parallel implementations the DBL represents the dominant module, with more than 50% in overall computational share.

Algorithm 1 Collaborative Inter-loop video encoding for heterogeneous CPU+GPU systems

1: define the initial distributions
2: **for** all inter frames **do**
3: perform inter-loop for the defined distributions
4: initialize/update the performance models
5: define module-level distribution for R* modules
6: perform the load balancing for inter-prediction modules
7: **end for**

IV. Collaborative Inter-loop Video Encoding

The collaborative inter-loop video encoding for heterogeneous CPU+GPU systems, proposed herein, provides unified execution environment that dynamically instantiates the parallel CPU and GPU algorithms for individual inter-loop modules. It is implemented in OpenMP and CUDA programming models in order to attain high execution control and to maximally exploit the parallelization potential of CPU+GPU system.

According to the analysis provided in Section III, the collaborative inter-loop video encoding is performed at a frame level in several steps. As presented in Algorithm 1, in the first step (Algorithm 1 line 1) the initial cross-device load distributions are defined for all inter-loop modules. In detail, for the ME, SME and INT modules, the equidistant data partitioning is performed, while the R* modules are assigned for execution on all processing devices in their entirety. This evaluation is conducted in order to build the initial FPMs for each device/inter-prediction module pair, as well as to assess the performance disparity among heterogeneous devices for each R* module. Based on this characterization, for each subsequent inter-frame, the inter-loop video encoding is performed on the target CPU+GPU system according to newly determined load distributions (line 3). Afterwards, the execution and data transfer times are recorded for each device/module pair and the corresponding performance models are updated (line 4).

In the proposed method, different scheduling approaches are applied for the inter-prediction (ME, SME and INT) modules and for the R* modules. For the R* modules, the cross-device distribution of entire modules is determined with Dijkstra algorithm [4], such that the overall encoding time is minimized (line 5). On the other hand, the computational load of the inter-prediction (ME+SME+INT) sequence is distributed among all the processing devices according to the dynamically built partial estimations of the full FPMs (line 6).

IV.1 Distribution for the R* modules

In this procedure, each of the least computationally intensive R* modules (MC, TQ, TQ^{-1} and DBL) is mapped to a processing device, such that the overall encoding time is minimized. This procedure also reflects the device-module execution affinities and the required

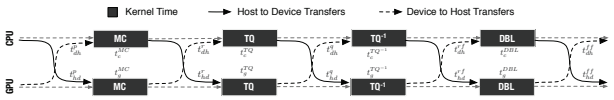


Figure 4: Data-flow diagram and weighted DAG structure of H.264/AVC encoding procedure for mapping the remaining R modules.*

data transfers for possible migration of the encoding procedure among the processing devices.

The implementation of this procedure is illustrated in Fig. 4 for a typical CPU+GPU system. Initially, a data-flow diagram is constructed, such that both processing and data transfer times (in each direction) are included for each R* module and for each devices in the system. In fact, such data-flow diagram is a weighted DAG that encapsulates all possible communication paths between the accelerators and the CPU. In detail, t^{MC}, t^{TQ}, $t^{TQ^{-1}}$ and t^{DBL} represent the time required to process each R* module, i.e. MC, TQ, TQ^{-1} and DBL, respectively, on different device architectures (where index c designates CPU and index g GPU device). In Fig. 4, the edges represent the input/output data transfers to/from a certain device, namely: *i)* t^p is the transfer time of needed MVs from SME and/or SFs from INT to perform the MC; *ii)* t^r is the transfer time of produced residual data to initiate TQ; *iii)* t^q is the time required to transfer quantization coefficients for TQ^{-1}; and *iv)* t^{rf} and t^{ff} represent the transfer times of reconstructed and filtered frames, respectively. The transfer direction is designated by *hd* or *dh* indices to represent the transfers occurring from the CPU (host) to the GPU (device) or from the GPU to the CPU, respectively.

As soon as the DAG is constructed, the minimal path between the first and last node is found. This path represents the optimal mapping of the modules to the processing devices in the CPU+GPU system. On the other hand, the sum of the weights of the edge within the path represents the prediction for the smallest R* encoding time achievable by applying the proposed method. Considering the fixed and limited number of DAG nodes, the optimal Dijkstra's algorithm [4] is applied to find the minimal path.

As it can be observed in Fig. 2, individual R* modules might have different device affinities, according to their data dependencies and parallelization potential, as well as the characteristics of target devices (number of cores, memory hierarchy etc.). However, the migration of R* sequence among devices rarely compensate the imposed data transfers, thus the entire R* sequence is usually performed on a single (fastest) device. For simplicity, in the remaining text, it will be assumed that the R* modules are processed on a single device. According to the selected device/architecture, the applied scheduling will be considered as *GPU-centric* and *CPU-centric*. However, it is worth emphasizing that this simplification is only introduced for presentation purposes and it does not influence the generality of the proposed load balancing approach.

IV.2 Load balancing for inter-prediction modules

The load balancing for inter-prediction (ME+INT+SME) sequence proposed herein relies on data parallelism at the level of the parts of the frame. In detail, it considers that each of the k CPU cores and w GPU accelerators (i.e. p_i processing devices, where i=$\{1,..,k+w\}$) performs the same algorithm on different parts of the input buffers.

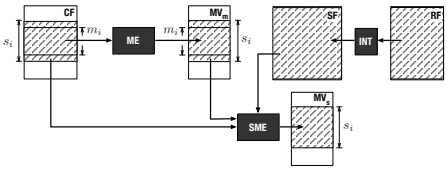


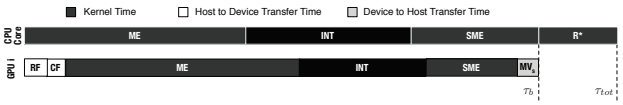
Figure 5: Data access management for collaborative processing of ME+SME+INT sequence.

As it is depicted in Fig. 5, the CF is partitioned among all CPU and GPU devices, such that the ME is collaboratively performed on the assigned CF portions, in order to produce the respective parts of MV_m buffer. The MV_m buffer is further partitioned among devices, to collaboratively perform the SME module. The simultaneously produced MVs by SME on different devices, are then collected in the MV_s buffer in the CPU main memory. It is worth noting that while the CPU can directly access the buffers from the main memory, for the GPUs explicit data transfers need to be performed.

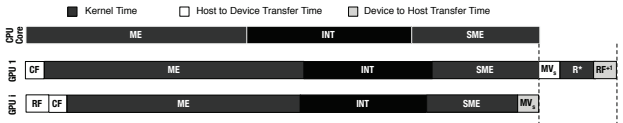
In the proposed method, the per-device load distributions are determined at the level of MB-rows. The major rationale behind adopting such granularity lies in the fact that it provides low scheduling overheads, while efficiently exploiting bandwidth of communication lines and device performance. In contrast, at the finer-grained level, the latency might dominate the execution, and the inevitable repacking is required of the original frame format from a matrix/array of pixels (in raster scan order) to an array of structures (MBs).

In order to eliminate the cost of expensive data-transfers for the SF (16 times larger than CF and RF), the execution of INT module is replicated on all devices. In fact, since the INT procedure is much faster than the corresponding data transfers, this replication also allows minimization of the overall inter-prediction time. Hence, the list of complete SFs is kept updated on each processing device. Accordingly, the distribution of the SME workload, considers only the input and output transfers of the full-pixel and quarter-pixel MVs, respectively. In order to minimize the memory requirements, both lists of SFs and RFs are updated in the form of FIFO circular buffers, where the newest SF/RF replaces the oldest one.

In Fig. 6, the proposed scheduling method is presented in two variants, i.e., *CPU-centric* (Fig. 6(a)) and *GPU-centric* (Fig. 6(b)). In



(a) CPU centric.



(b) GPU centric.

Figure 6: Scheduling strategy for collaborative inter-loop video encoding.

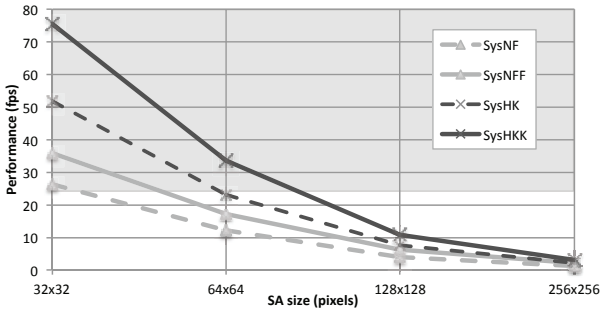


Figure 7: Performance of the proposed method for different SA sizes, single RF and 1080p resolution.

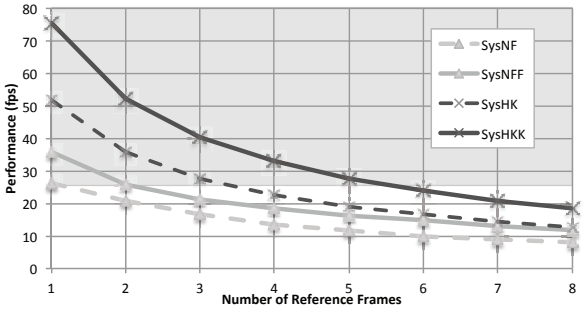


Figure 8: Performance of the proposed method for different number of RFs, 32×32 pixels SA and 1080p resolution.

both cases, there are only two synchronization points for video encoding inter-loop, namely: *i*) τ_b at the end of collaborative processing of inter-prediction sequence; and *ii*) τ_{tot} at the end of inter-loop.

For both variants, the inter-prediction processing on the GPU that does not perform R* modules (i.e., GPU_i) also requires the initial transfer of the CF and RF input buffers, and the output transfer of the produced part of MV_s buffer (see Fig. 6 and 5). However, for the GPU that processes the R* modules (i.e., GPU_1 in Fig. 6(b)), the output transfer of the MV_s buffer is not required, since the MVs are only needed on the device that performs the MC module. However, since the MVs produced on the other devices must be collected, the input transfer for the remaining part of the MV_s buffer is required after the τ_b point. At the end of R* sequence, the produced RF must be returned to the CPU, in order to allow processing of the next inter-frame on other devices. Due to the replication of the INT module, the transfer of SF buffer is not required.

In each iteration (inter-frame), the distribution vectors $m=\{m_i\}$ and $s=\{s_i\}$ are determined, where the m_i and s_i represent the number of MB-rows assigned to the ME and SME, respectively, for each processing device p_i. The m and s distribution vectors are determined by the application of the MSLBA algorithm [7] and by relying on dynamically built FPMs for each device-module pair. The partial estimations of the full FPMs are constructed by applying piece-wise linear approximation on a minimum set of points, i.e., by considering the performance obtained in previous iterations, and the asymmetric bandwidth of communication lines. In this algorithm the distributions are firstly found in the real domain, while the final integer distributions (m and s) are obtained in a refinement procedure [7].

V. Experimental Results

For the evaluation purposes, the proposed method is integrated in JM 18.6 reference coder [8]. The Baseline Profile was applied, and two different quantizer values (28 and 33). Also, two different 1080p sequences are tested, namely *RollingTomatoes* and $Sunflower$. Presented values are the average performance obtained for these parameters/sequences. However, it is worth noting that the obtained performance does not significantly depend on the video content. The tests were executed on different desktop platforms composed of the following devices: Intel i7 4770K (Haswell) CPU, Intel Core i7 950 (Nehalem) CPU, NVIDIA Tesla K40c (Kepler) GPU and NVIDIA GeForce GTX 580 (Fermi). The system composed of single Haswell CPU and single Kepler GPU is assigned as *Sys_HK*, while the system composed of the same CPU and two Kepler GPUs is *Sys_HKK*. On the other hand, the system composed of a single Nehalem CPU and a single Fermi GPU is assigned as *Sys_NF*, while the system composed of the same CPU and two Fermi GPUs is *Sys_NFF*.

Figure 7 presents the average performance in frames per second (fps) obtained with the proposed method, when considering different SA sizes, single RF and full HD (1080p) video sequences. As it can be observed, a real time encoding (more than 25 fps) was achieved in all tested systems for 32×32 pixels SA. In *SysHKK* platform, a real-time encoding (more than 33 fps) was achieved even for more demanding 64×64 pixels SA, while in *SysHK* a near real-time performance was achieved (more than 22 fps) with the same coding parameters.

Figure 8 shows the experimentally achieved average performance (in fps) with the proposed method in different systems and for different number of RFs and 32×32 pixels SA. As it can be observed, all the systems except the *Sys_NF* were able of achieving a real-time performance of more than 25 fps for multiple RFs. Moreover, in *Sys_HKK* a real-time performance for up to 5 RFs was achieved.

In addition to the ability of the proposed method to achieve a real-time performance for very demanding coding parameters, Fig. 7 and 8 also show that the proposed method is scalable over both the SA size and the number of RFs. This is achieved mainly due to the high efficiency of the proposed load balancing approach and the developed highly optimized parallel CPU and GPU algorithms.

VI. Conclusions

In this study, an efficient method for collaborative H.264/AVC inter-loop in heterogeneous CPU+GPU systems was proposed. In order to cope with the inherent data-dependencies and computational complexity of inter-loop modules, a unified execution environment was designed to ensure efficient cross-device execution and to guarantee the correctness of the video encoding process. It also includes an extensive library of highly optimized parallel algorithms for all inter-loop video encoding modules, which are developed using the device-specific programming models and tools. The integrated scheduling and load balancing routines allow efficient distribution of the workloads for these modules across all processing devices. For the most computationally intensive inter-prediction modules,

the proposed adaptive load balancing relies on realistic and dynamically built FPMs of both communication and computation system resources. The workloads of the remaining modules are distributed according to their module-device affinities by applying the optimal Dijkstra algorithm. In order to minimize the overall inter-loop encoding time, the proposed collaborative encoding method also integrates data access management and specific, communication-aware replication techniques to maximize data reuse, while decreasing the data transfers overheads. The experimental results shown that the proposed method is able of achieving real-time video encoding for full HD resolution, with a 64×64 pixels SA and exhaustive ME on the state-of-the-art commodity CPU+GPU platforms. Moreover, the scalability of the proposed method over the SA size, number of RFs and number of processing devices was experimentally shown.

Acknowledgment

This work was supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under projects PEst-OE/EEI/LA0021/2013, PTDC/EEI-ELC/3152/2012 and PTDC/EEA-ELC/117329/2010.

References

[1] G. Barlas, A. Hassan, and Y. Al Jundi. An analytical approach to the design of parallel block cipher encryption/decryption: A CPU/GPU case study. In *Proceedings of the Int. Conf. on Par., Dist. and Network-Based Proc. (PDP)*, pages 247–251, 2011.

[2] Z. Chen, J. Xu, Y. He, and J. Zheng. Fast integer-pel and fractional-pel motion estimation for H.264/AVC. In *Journal of Visual Communication and Image Representation*, pages 264–290, October 2005.

[3] N.-M. Cheung, X. Fan, O. C. Au, and M.-C. Kung. Video coding on multicore graphics processors. *IEEE Signal Processing Magazine*, 27(2):79–89, 2010.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Num. Math.*, 1(1):269–271, Dec. 1959.

[5] A. Ilic, S. Momcilovic, N. Roma, and L. Sousa. FEVES: Framework for efficient parallel video encoding on heterogeneous systems. In *Proceedings of the International Conference on Parallel Processing*, pages 165–174, 2014.

[6] A. Ilic and L. Sousa. Scheduling divisible loads on heterogeneous desktop systems with limited memory. In *Proceedings of the Euro-Par Workshops*, pages 491–501, 2012.

[7] A. Ilic and L. Sousa. Simultaneous multi-level divisible load balancing for heterogeneous desktop systems. In *Proceedings of the IEEE Int. Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 683–690, 2012.

[8] ITU-T. *JVT Reference Software unofficial version 18.6*. http://iphome.hhi.de/suehring/tml/, 2014.

[9] Y. Ko, Y. Yi, and S. Ha. An efficient parallelization technique for x264 encoder on heterogeneous platforms consisting of CPUs and GPUs. *Journal of Real-Time Image Proc.*, pages 1–14, 2013.

[10] A. Lastovetsky and R. Reddy. Distributed data partitioning for heterogeneous processors based on partial estimation of their functional performance models. In *Proceedings of the Euro-Par*, pages 91–101, 2010.

[11] Ping Li, B. Veeravalli, and A.A. Kassim. Design and implementation of parallel video encoding strategies using divisible load analysis. *IEEE Trans. on Circuits and Systems for Video Technology*, 15(9):1098 – 1112, September 2005.

[12] S. Momcilovic, A. Ilic, N. Roma, and L. Sousa. Collaborative inter-prediction on cpu+gpu systems. In *Proceedings of the IEEE International Conference on Image Processing*, 2014.

[13] S. Momcilovic, A. Ilic, N. Roma, and L. Sousa. Dynamic load balancing for real-time video encoding on heterogeneous CPU+GPU systems. *IEEE Transactions on Multimedia*, 16(1):108–121, Jan 2014.

[14] S. Momcilovic, N. Roma, and L. Sousa. Multi-level parallelization of advanced video coding on hybrid CPU+GPU platforms. In *Proceedings of the Euro-Par Workshops*, pages 165–174, 2012.

[15] S. Momcilovic, N. Roma, and L. Sousa. Exploiting task and data parallelism for advanced video coding on hybrid cpu+ gpu platforms. *Journal of Real-Time Image Proc.*, pages 1–17, 2013.

[16] J. Ohm and G.J. Sullivan. High efficiency video coding: the next frontier in video compression [standards in a nutshell]. *Signal Processing Magazine, IEEE*, 30(1):152–158, Jan 2013.

[17] J. Ostermann et al. Video coding with H.264/AVC: tools, performance, and complexity. *IEEE Circuits and Systems Magazine*, 4(4):7–28, April 2004.

[18] B. Pieters, C. F. Hollemeersch, P. Lambert, and R. Van De Walle. Motion estimation for H.264/AVC on multiple GPUs using NVIDIA CUDA. In *Proceedings of the Society Photo-Optical Instrumentation Engineers*, page 12, 2009.

[19] R. Rodríguez-Sánchez, J. L. Martínez, G. Fernández-Escribano, J. L. Sánchez, and J. M. Claver. A fast GPU-based motion estimation algorithm for H. 264/AVC. In *Advances in Multimedia Modeling*, pages 551–562. Springer, 2012.

[20] B. Veeravalli, D. Ghose, and T. G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6:7–17, 2003.

[21] H. Wei, J. Yu, and J. Li. The design and evaluation of hierarchical multi-level parallelisms for H.264 encoder on multi-core architecture. *Comput. Sci. Inf. Syst.*, 7(1):189–200, 2010.

[22] N. Wu, M. Wen, J. Ren, H. Su, and D. Huang. High-performance implementation of stream model based H.264 video coding on parallel processors. In *Multimedia and Signal Processing*, volume 346, pages 420–427. Springer Berlin Heidelberg, 2012.

[23] J. Zhang, J. F Nezan, and J.-G. Cousin. Implementation of Motion Estimation Based on Heterogeneous Parallel Computing System with OpenCL. In *Proceedings of the IEEE Int. Conf. on High Perf. Comp. and Comm.*, pages 41–45, 2012.