COLLABORATIVE INTER-PREDICTION ON CPU+GPU SYSTEMS

Svetislav Momcilovic, Aleksandar Ilic, Nuno Roma, Leonel Sousa

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

ABSTRACT

In this paper we propose an efficient method for collaborative H.264/AVC inter-prediction in heterogeneous CPU+GPU systems. In order to minimize the overall encoding time, the proposed method provides stable and balanced load distribution of the most computationally demanding video encoding modules, by relying on accurate and dynamically built functional performance models. In an extensive RD analysis, an efficient temporary dependent prediction of the search area center is proposed, which allows dependency-aware workload partitioning and efficient GPU parallelization, while preserving high compression efficiency. 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.

Index Terms— video coding, divisible load theory, load-balancing, CPU+GPU computing

1. INTRODUCTION

The latest generation of video coding standards, such as H.264/AVC [1] and HEVC/H.265 [2], achieve high compression efficiencies, by relying on advantageous encoding techniques (e.g. multiple partitioning modes, large search ranges, quarter-pixel precision). However, all these techniques dramatically increase the computational requirements, and make real-time encoding of high video resolutions 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).

In order to simultaneously employ the several heterogeneous devices that are available on modern desktops for real-time video encoding of High Definition (HD) sequences, an efficient method for collaborative inter prediction on CPU+GPU systems is proposed herein. This method does not only employ highly efficient parallel algorithms for both the CPUs and GPUs and different inter-prediction modules, i.e. Motion Estimation (ME), Sub-Pixel ME (SME) and Interpolation (INT), but also provides scheduling and load balancing routines to efficiently distribute the workload over all the processing devices. In order to ensure efficient cross-device execution, an unified execution environment was designed, which integrates scheduling, load balancing and data access management routines, as well as the parallel algorithms developed in device specific programming environments (e.g. CUDA, OpenMP, etc.). The proposed load balancing, based on Divisible Load Theory (DLT) [3], relies on realistic and dynamically built Functional Performance Models (FPMs) [4, 5], which provide a realistic modeling of communication and computation performance of system resources. This method also introduces specific, communication-aware techniques to maximize data reuse, and to decrease the data transfers overhead in collaborative video encoding. Because of a similar algorithmic structure of the inter-prediction, many solutions provided in this method can be also applied to HEVC/H.265 encoders.

To allow an efficient workload partitioning, while preserving the compression efficiency of the H.264/AVC reference JM encoder [6], a new temporal search area (SA) center prediction is proposed. In an extensive Rate-Distortion (RD) analysis, it was shown that the proposed predictor does not only allow an efficient GPU parallelization and collaborative CPU+GPU encoding, but it also keeps a high RD efficiency.

The obtained experimental results show that the proposed method achieves real-time inter-prediction of full-HD (1080p) video sequences, when applying exhaustive ME and 64×64 pixels SA on a platform equipped with a multi-core CPU and two GPUs. To the best of the authors' knowledge, this is one of the first methods that applies the FPMs to complex multi-module problems, such as video encoding, and that achieves the real time inter-prediction on desktops, when applying such demanding coding parameters.

2. RELATED WORK

Parallel video encoding approaches usually consider CPU and/or GPU parallelization of the most computationally demanding modules, mainly the ME. Due to the absence of branches and the regularity of memory accesses, the Full-Search Block-Matching (FSBM) algorithm is usually chosen for efficient GPU implementation [7–14], since the adaptive

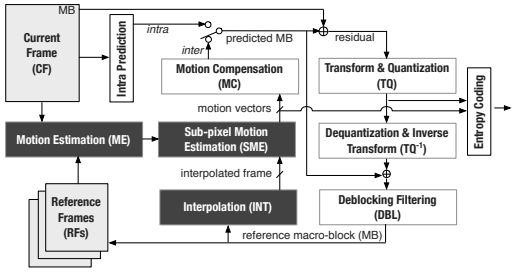

Fig. 1. H.264/AVC encoder.

algorithms do not significantly outperform respective CPU implementations [15, 16]. At this respect, to simultaneously employ hundreds of GPU cores, these parallel algorithms still have to relax the spatial data-dependencies, imposed when relying on adjacent motion vectors (MVs) to define the SA center. This relaxation is usually achieved by applying either the zero [7, 9] or temporary dependent predictors [10–12].

In heterogenous CPU+GPU systems, the state-of-the-art approaches usually *i)* simply offload a single inter-prediction module in its entirety (mainly the ME) to the GPU, while performing the rest of the encoder on the CPU [7, 10, 16], or *ii)* exploit simultaneous CPU+GPU processing at the level of a *single* inter-prediction module [8, 17]. However, 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) [7, 10]. Furthermore, in [8] and [16] the load distribution and pipelining granularity are decided through a large set of experiments, which limits the scalability, and introduces huge preprocessing overheads.

There is only a few studies targeting the DLT scheduling in CPU+GPU systems either for general [4] or application-specific [18] problems. In [19], the authors apply DLT scheduling for a single-module load distribution in CPU-only cluster environments for a custom video encoder. In [12], we proposed an adaptive load-balancing method that relies on constant performance models and linear programming load-distribution techniques. In contrast, the method that is proposed herein relies on more realistic FPM, thus providing more accurate and stable load balancing decisions.

3. COLLABORATIVE INTER PREDICTION

In order to allow an efficient parallelization and collaborative CPU+GPU execution, the inherent data dependences and the computational requirements of H264/AVC encoder, must be considered, with a special emphasis on inter-prediction (see Fig. 1). At this respect, the inter-prediction modules (ME+INT+SME) participate with more than 80% in overall encoding time, which makes their efficient implementation crucial to achieve real-time performance. The inter-prediction starts with the ME module, where the obtained MVs for the left, top and top-right neighbors of the currently processed Macroblock (MB) are used to define the SA center. This impose a spatial data dependency between these MBs, thus preventing their parallel processing. Additionally, in the case of adaptive algorithms, the MVs of neighboring MBs are also used to select the search patterns, as well as to define distortion thresholds for early termination of the ME procedure. At the frame level, the processing of the current frame (CF) can not start until all the previous frames are encoded and the full list of Reference Frames (RFs) is generated, since it represents the input for the ME and INT modules. Moreover, the SME can not be processed until both the ME and the INT are completed, since their outputs are the inputs for the SME.

In what concerns the ME, efficient CPU and GPU **parallelizations** often adopt the FSBM, 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, the state-of-the-art parallelization of the adaptive algorithm [15] (namely UMHexagonS [20]) on the GPUs are unable of achieving better performance than CPU implementations.

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, in order to relax spatial data dependences imposed by the definition of SA center, a set of temporary dependent predictors was analyzed herein. The MVs are than recalculated at the end of ME to provide the correct offsets to the median predictors. Besides ensuring a full compliance with H.264/AVC, these predictors were evaluated in an extensive RD analysis (see Section 4) considering their impact in the resulting compression efficiency.

This set comprises the following temporal predictors:

- **ZERO** motion vector predictor, i.e., $MV_{cf}{=}(0,0)$;
- **16×16**: The best MV for collocated MB in the previous frame $cf{-}1$, considering the 16×16 partitioning mode, i.e., $MV_{cf}^{u\times v}{=}MV_{cf-1}^{16\times 16}$, where $u{\times}v{=}\{4{\times}4, 4{\times}8, 8{\times}4, 8{\times}8, 8{\times}16, 16{\times}8, 16{\times}16\}$;
- **REL_RF**: The best MV for collocated MB in $cf{-}1$, considering only the RF with the same index rf, and 16×16 partitioning mode, i.e., $MV_{cf,rf}^{u\times v}{=}MV_{cf-1,rf}^{16\times 16}$;
- **DEEP SEARCH**: Equal to **16×16** predictor for the first RF. For the other RFs it is displaced by the best MV found in $rf{-}1$, for 16×16 partitioning mode, i.e., $MV_{cf,rf}^{u\times v}{=}\begin{cases} MV_{cf-1}^{16\times 16}, & rf{=}0 \\ MV_{cf-1}^{16\times 16}+MV_{cf-1,rf-1}^{16\times 16}, & rf{>}0 \end{cases}$.

A detailed explanation of the parallel algorithms for CPU and GPU architectures used herein is provided in [21].

The **collaborative inter-prediction** strategy proposed herein 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. As it is depicted in Fig. 2, the CF is partitioned among all CPU and GPU devices. These devices collaboratively perform the ME on the assigned portions of the CF and produce the respective parts of MV_m buffer. This buffer is further partitioned among devices, to collaboratively perform the SME module. The MVs simultaneously produced by SME on different devices, are finally collected in the MV_s buffer in the CPU main memory. It is worth emphasizing that while CPU cores can directly access their buffers portions from the main memory, for the GPUs explicit data transfers need to be performed (over PCIExpress), as shown in Fig. 3.

In order to eliminate the cost of the expensive data-transfer corresponding to the interpolated Sub-pixel Frame (SF) (16 times larger than CF and RF), the execution of INT module is replicated on all devices. In fact, considering that the INT procedure is much faster than the corresponding data transfers, this approach allows maximizing the efficiency. In such a way, 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 the proposed method, the load distribution is implemented at the level of individual MB rows. The major rationale to adopt this granularity lies in the fact that it provides low scheduling overheads, while efficiently exploiting bandwidth of communication lines and device performance. In contrast, at finer-grained level of MBs, the latency might dominate the execution, and an inevitable repacking of the original frame format from a matrix/array of pixels (in raster scan order) to an array of structures (MBs) would be required.

The proposed method is a two step multi module load balancing procedure that considers two synchronization points. In particular, the τ_{b1} at the end of the simultaneous processing of the ME and INT modules (including the transfers of CF/RF to the GPUs and the transfer of full pixel MVs to the CPU); and the τ_{b2} at the end of the SME (including the final transfers of quarter-pixel MVs to the CPU) as depicted in Fig. 3. In each iteration (video 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 [4] and by relying on dynamically built FPMs for each device-module pair with minimum set of points, considering the performance from previous iterations, and the asymmetric bandwidth of communication lines.

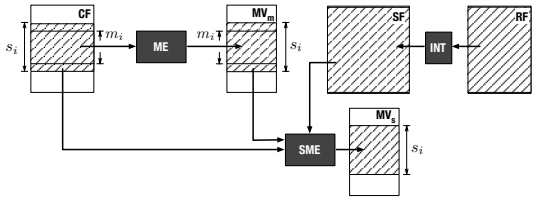

Fig. 2. Data access management for collaborative inter prediction.

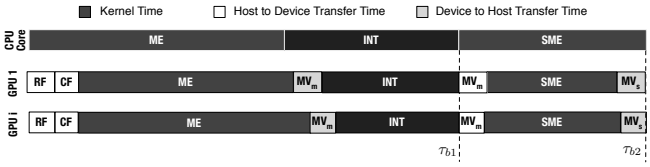

Fig. 3. Scheduling strategy for collaborative inter prediction.

4. EXPERIMENTAL RESULTS

To evaluate the proposed method, a set of experimental results considering both the RD and the processing performance is presented, by relying on JM 18.6 reference coder [6]. For such purpose, a vast set of video sequences with CIF, 720p, and1080p resolutions is considered. The encoder was executed on a desktop platform with a quad-core Intel 4770K (Haswell, 3.5GHz) CPU, and two NVIDIA Tesla K40c (Kepler, 876MHz) GPUs. The RD analysis was performed by strictly following the VCEG recommendations [22] for IPPP sequences, Baseline Profile, and the following quantizer values {ISlice, PSlice}: {22,23}, {27,28}, {32,33} and {37,38}.

The analysis presented in Table 1 shows the objective differences between the RD-curves of the tested predictors/algorithms and the original JM encoder with FSBM algorithm, computed using the Bjøntegaard method (with the tool recommended by VCEG). The tested predictors/algorithms comprise the proposed set of temporary dependent predictors (**16×16**, **DEEP_SEARCH** and **REL_RF**), **ZERO** predictor, and the JM implementation of the UMHexagonS (UMHS) algorithm. As it can be observed, there is no single predictor that can ensure optimal performance for all tested sequences and coding parameters. However, for the smaller 64×64 SA, the temporary dependent predictors (particularly the **16×16**) succeed to approach to the performance of the original JM encoder for all tested sequences. Contrary, for the $Jets$ sequence, with the **ZERO** predictor the large MVs remain outside of the smaller 64×64 pixels SA, which results in significant degradation of the RD performance. However, when larger SA is used, this predictor provides the RD performance very close to the reference JM implementation for all tested sequences. Finally, it can be also observed that for the most tested sequences and both SA sizes, the proposed predictors significantly outperform the UMHS. According to this analy-

Table 1. Overall RD Analysis for different predictors/algorithms using Bjøntegaard metric (ΔPSNR [dB] and Δbitrate [%])

Format	Sequence	Frame rate	Encoded frames	SA 64×64										SA 128×128									
				ZERO		16x16		DEEP_S		REL_RF		UMHS		ZERO		16x16		DEEP_S		REL_RF		UMHS	
				dB	%	dB	%	dB	%	dB	%	dB	%	dB	%	dB	%	dB	%	dB	%	dB	%
CIF	Paris	30	300	-0.03	0.59	-0.23	4.50	-0.20	3.85	-0.17	3.19	-0.12	2.19	-0.04	0.80	-0.13	2.49	-0.12	2.41	-0.10	1.92	-0.15	2.92
	Paris[1]	30	300	-0.03	0.46	-0.41	7.69	-0.41	7.62	-0.33	6.13	-0.13	2.35	-0.02	0.39	-0.35	6.57	-0.36	6.83	-0.27	4.91	-0.12	2.28
	Foreman	30	300	-0.10	2.39	-0.29	6.86	-0.31	7.38	-0.25	5.78	-0.31	7.26	-0.11	2.55	-0.19	4.40	-0.26	5.95	-0.22	5.09	-0.32	7.47
	Mobile	30	300	-0.02	0.40	-0.05	0.97	-0.11	2.19	-0.05	1.03	-0.06	1.25	-0.02	0.45	-0.03	0.50	-0.08	1.53	-0.04	0.75	-0.07	1.39
	Tempete	30	260	-0.03	0.60	-0.05	0.98	-0.18	3.48	0.03	-0.45	-0.21	4.03	-0.03	0.61	-0.04	0.77	-0.14	2.64	0.07	-1.27	-0.22	4.20
	Stefan	30	260	-0.10	1.84	-0.36	6.71	-0.41	7.58	-0.35	6.43	-0.60	11.68	-0.08	1.44	-0.30	5.69	-0.36	6.78	-0.24	4.41	-0.47	9.08
	Table	30	260	-0.06	1.30	-0.21	4.98	-0.29	6.74	-0.19	4.38	-0.24	5.72	-0.06	1.38	-0.15	3.54	-0.17	3.79	-0.14	3.30	-0.23	5.41
720p	BigShips	60	150	-0.02	0.54	-0.03	0.88	-0.05	1.59	-0.03	1.07	-0.09	3.18	-0.01	0.51	-0.03	0.88	-0.05	1.56	-0.03	1.09	-0.09	3.18
	Crew	60	150	-0.04	1.43	-0.19	6.61	-0.42	15.28	-0.23	8.07	-0.14	4.93	-0.05	1.63	-0.19	6.58	-0.44	16.16	-0.24	8.53	-0.14	4.93
	Jets[2]	60	150	-1.06	24.43	-0.16	3.04	-0.36	7.14	-0.34	6.81	-1.37	33.21	-0.21	4.33	-0.40	8.29	-0.55	11.37	-0.50	10.40	-0.88	20.40
	Night	60	150	-0.05	1.35	-0.27	7.48	-0.21	5.67	-0.21	5.71	-0.24	6.56	-0.04	1.10	-0.21	5.65	-0.19	5.23	-0.16	4.43	-0.21	5.86
	Raven	60	150	-0.10	2.53	-0.20	5.11	-0.23	5.69	-0.21	5.31	-0.47	12.53	-0.11	2.64	-0.21	5.21	-0.25	6.19	-0.22	5.38	-0.43	11.69
	City_corr	60	150	-0.05	1.38	-0.05	1.37	-0.07	2.03	-0.05	1.36	-0.19	5.88	-0.05	1.36	-0.04	1.36	-0.05	1.56	-0.05	1.33	-0.19	6.06
	ParkJoy	50	150	-0.04	0.71	-0.65	12.98	-0.47	9.23	-0.71	14.44	-0.12	2.34	-0.04	0.69	-0.52	10.38	-0.37	7.36	-0.38	7.51	-0.12	2.33
	CrowdRun	50	150	-0.05	1.04	-0.07	1.37	-0.05	1.01	-0.06	1.14	-0.18	3.40	-0.06	1.13	-0.09	1.76	-0.06	1.22	-0.06	1.13	-0.17	3.34
1080p	Toys&cal.	25	125	-0.14	5.39	-0.17	6.19	-0.22	8.06	-0.17	6.25	-0.75	32.72	-0.15	5.69	-0.15	5.54	-0.20	7.57	-0.15	5.75	-0.76	32.92
	Sunflower	25	125	-0.12	3.27	-0.08	2.27	-0.10	2.76	-0.10	2.80	-0.45	13.71	-0.11	3.27	-0.08	2.20	-0.11	3.15	-0.11	3.04	-0.44	13.32
	R. Tomatoes	24	60	-0.03	1.53	-0.11	5.97	-0.12	7.17	-0.11	6.26	-0.16	9.60	-0.03	2.02	-0.11	6.22	-0.13	7.24	-0.12	6.56	-0.17	10.13

[1] FrameSkip=1 [2] StartFrame=300

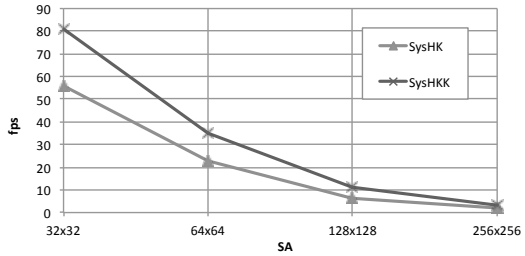

Fig. 4. Performance of the proposed method for different SA sizes, single RF and 1080p resolution.

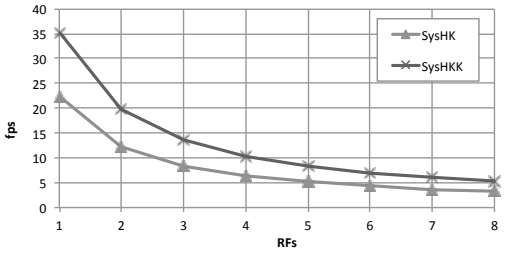

Fig. 5. Performance of the proposed method for different number of RFs, 64×64 pixels SA and 1080p resolution.

sis, it can be concluded that the strategy of choosing the predictors according to the SA size (e.g. **16×16** for smaller and **ZERO** for larger SAs) provides the performance very close to the original JM encoder with FSBM. However, in contrast to the original implementation this strategy allows efficient GPU parallelization and collaborative CPU+GPU execution, which result in a very high processing performance.

Figures 4 and 5 present the average performance in frames per second (fps) that is obtained for the proposed method, when considering different SA sizes and different number of RFs, respectively, for full HD (1080p) video sequences. The proposed method was tested on two hybrid configurations, i.e. $SysHK$, equipped with a multi-core CPU and a single GPU, and $SysHKK$ with multi-core CPU and two GPUs. As it can be observed, the proposed method achieves more than 35 fps, with 64×64 pixels SA, on $SysHKK$ configuration. Even for the configuration with a single GPU ($SysHK$), this method achieves a performance very close to the real-time (more than 22 fps) with the same coding parameters.

To evaluate the efficiency of the proposed method, the performance that is obtained on these hybrid systems is also compared with the CPU and GPU single-device implementations. In detail, the execution on $SysHK$ provides speedups of up to 1.4× and 2.5× in comparison to the GPU and CPU implementations, respectively. Moreover, on the $SysHKK$, the corresponding speedups are 2.3× and 4×. Finally, Figures 4 and 5 also demonstrate the scalability of the proposed method with the SA size and the number of RFs, respectively.

5. CONCLUSIONS

In this paper, an efficient method for collaborative H.264/AVC inter-prediction in heterogeneous CPU+GPU systems was proposed. By relying on realistic, dynamically built functional performance models, the proposed method provides a stable and balanced distribution of the most computationally demanding video encoding modules. An efficient collaborative CPU+GPU encoding with preserved RD efficiency was allowed by temporal SA center prediction, which resulted from an extensive RD analysis, and several tested predictors. 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.

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.

6. REFERENCES

[1] J. Ostermann et al., "Video coding with H.264/AVC: tools, performance, and complexity," *IEEE Circuits and Systems Magazine*, vol. 4, no. 4, pp. 7–28, April 2004.

[2] J. Ohm and G.J. Sullivan, "High efficiency video coding: the next frontier in video compression [standards in a nutshell]," *Signal Processing Magazine, IEEE*, vol. 30, no. 1, pp. 152–158, Jan 2013.

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

[4] 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)*, 2012, pp. 683–690.

[5] 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*, 2010, pp. 91–101.

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

[7] W.-N. Chen and H.-M. Hang, "H.264/AVC motion estimation implementation on compute unified device architecture (CUDA)," in *Proceedings of the IEEE Int. Conf. on Multimedia & Expo (ICME)*, 2008, pp. 697–700.

[8] 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.*, 2012, pp. 41–45.

[9] S. Momcilovic and L. Sousa, "Development and evaluation of scalable video motion estimators on GPU," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS)*, 2009, pp. 291 – 296.

[10] 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*, pp. 551–562. Springer, 2012.

[11] Y. Gao and J. Zhou, "Motion vector extrapolation for parallel motion estimation on GPU," *Multimedia Tools and Applications*, pp. 1–15, 2012.

[12] 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*, vol. 16, no. 1, pp. 108–121, Jan 2014.

[13] D. Chen, H. Su, W. Mei, L. Wang, and C. Zhang, "Scalable parallel motion estimation on muti-GPU system," in *Proceedings of the Int. Symposium on Computer, Communication, Control and Automation (ISCCCA)*, 2013, p. 6.

[14] 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*, 2009, p. 12.

[15] N.-M. Cheung, X. Fan, O. C. Au, and M.-C. Kung, "Video coding on multicore graphics processors," *IEEE Signal Processing Magazine*, vol. 27, no. 2, pp. 79–89, 2010.

[16] 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 Processing*, pp. 1–14, 2013.

[17] 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 Processing*, pp. 1–17, 2013.

[18] 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)*, 2011, pp. 247–251.

[19] 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*, vol. 15, no. 9, pp. 1098 – 1112, September 2005.

[20] 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*, October 2005, pp. 264–290.

[21] S. Momcilovic, A. Ilic, N. Roma, and L. Sousa, "Advanced Video Coding on CPUs and GPUs: Parallelization and RD Analysis," Technical report (available online), INESC-ID, February 2013.

[22] T. K. Tan, G. J. Sullivan, and T. Wedi, "Recommended simulation common conditions for coding efficiency experiments - revision 3," *ITU - Telecommunications Standardization Sector, VCEG, Doc. VCEG-AI10*, July 2008.