Taylor & Francis
Taylor & Francis Group

Automatika

Journal for Control, Measurement, Electronics, Computing and Communications

ISSN: 0005-1144 (Print) 1848-3380 (Online) Journal homepage: https://www.tandfonline.com/loi/taut20

Performance-efficient integration and programming approach of DCT accelerator for HEVC in MANGO platform

Igor Piljić, Leon Dragić & Mario Kovač

To cite this article: Igor Piljić, Leon Dragić & Mario Kovač (2019) Performance-efficient integration and programming approach of DCT accelerator for HEVC in MANGO platform, Automatika, 60:2, 245-252, DOI: 10.1080/00051144.2019.1618526

To link to this article: https://doi.org/10.1080/00051144.2019.1618526

Published online: 20 May 2019.

AUTOMATIKA
2019, VOL. 60, NO. 2, 245–252
https://doi.org/10.1080/00051144.2019.1618526

REGULAR PAPER

OPEN ACCESS Check for updates

Performance-efficient integration and programming approach of DCT accelerator for HEVC in MANGO platform

Igor Piljić, Leon Dragić and Mario Kovač

Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia

ABSTRACT

Video encoding based on novel HEVC standard is an extremely computationally expensive process and achieving efficient encoding requires intelligent utilization of all available resources, from both software and hardware perspective. Profiling and analysis of the encoding process identified Discrete cosine transform (DCT) as one of the key kernels that consume most of the time in the application's runtime. Therefore, high-throughput, fully-pipelined hardware accelerator was designed in FPGA and integrated into MANGO platform. MANGO platform is heterogeneous HPC system that consists of different types of nodes, from general purpose nodes (GN) to heterogeneous nodes (HN). While executing specific kernels on GN nodes is a straightforward process, executing kernels on accelerator-based HNs is a more complex procedure and requires specific integration to successfully exploit heterogeneous architecture. This paper presents performance-efficient integration of DCT hardware accelerator in MANGO platform, focusing on the performance of the encoder while maintaining coding efficiency and video quality of the encoded bitstream. Several approaches were considered, tested and compared; from the standalone integration where series of single tasks were offloaded to the DCT accelerator, to more complex solutions based on smart buffer utilization.

ARTICLE HISTORY
Received 20 February 2019
Accepted 4 April 2019

KEYWORDS
Video encoding; HEVC; DCT; hardware accelerators; heterogeneous computing; MANGO

Introduction

Latest analysis and statistics show that 82% of global IP traffic will be video traffic by 2022, which is an increase from 75% in 2017 [1]. Handling and transferring this huge amount of data requires efficient systems, in terms of performance, power and predictability, that are able to deliver video content with desired Quality of Experience (QoE). High-efficiency video coding (HEVC) is one of the latest video coding standards that can achieve up to 50% bitrate reduction when compared to the previous standard Advanced Video Coding (AVC) [2]. However, the computational complexity and resource requirements of HEVC are increased by up to 10 times [3]. To deal with the increased computational complexity it is necessary to intelligently utilize all software and hardware components of the system. Although software algorithms can lead to significant improvements, heterogeneous accelerator-based architectures on high performance computers can drastically improve power-performance ratio of the system, so their analysis and exploitation, especially for large-scale video content providers are necessary.

Custom hardware accelerators can improve performance and lower the power consumption, however efficient utilization of such architectures is a challenging task. Overhead of data transfer often suppresses performance benefits gained by the accelerator.

In this paper, we present and compare several approaches for performance-efficient integration of hardware accelerator for one of the key kernels in the HEVC encoder/decoder: discrete cosine transform (DCT). In this paper, we describe HEVC DCT implementation in heterogeneous, accelerator-based architecture developed as a part of MANGO project.

The rest of the paper is organized as follows: Second section describes the motivation for this research, Section three describes system on which all integrations and tests were concluded, while Section four presents integration and programming approaches. Finally, performance evaluation is presented in Section five.

Motivation and related work

Heterogeneous, accelerator-based architectures provide great potential for increasing efficiency of compute and data-intensive applications, such as video encoding from both performance and power perspective. However, exploiting such architectures requires a deep understanding of both application requirements and system design, as well as intelligent utilization of all available resources.

Previous research in this field rarely cover integration of the accelerators in heterogeneous systems containing different types of processing cores, from general

CONTACT Igor Piljić igor.piljic@fer.hr Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia

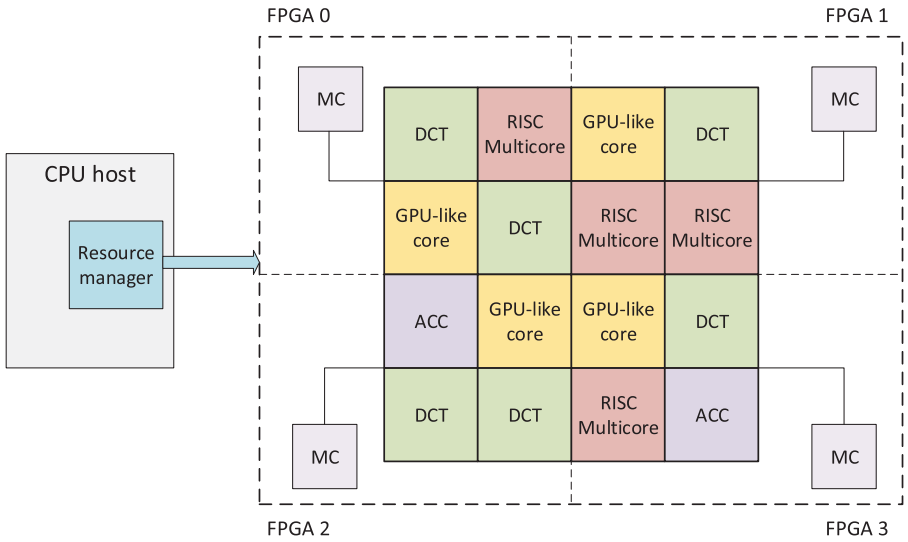

Figure 1. MANGO platform scheme.

processing nodes to heterogeneous nodes. In [4–6] CPU + GPU heterogeneous platform is used to accelerate HEVC decoder, either by deploying a subset of functions (IDCT and de-quantization) or by implementing entire decoder on GPU. In comparison with the HEVC encoder, the decoder is computationally much less demanding than HEVC encoder, which is why our focus is set on improving the encoding process. GPU-based accelerators can offer large performance improvements compared to the CPU, especially for highly parallelizable algorithms, however, FPGA-based accelerators could provide even more performance and energy-efficient solutions.

There is a lot of research on specialized hardware accelerators for several different kernels that are used in HEVC. FPGA-based solutions for Intra coding are presented in [7,8]. Architectures for HEVC standard DCT acceleration introduced in [9–12] show several different approaches for optimizing integer based DCT. Interpolation [13] and deblocking filter [14] as another compute demanding algorithms are also subject of the research in this area. However, all these papers focus their research on a specific accelerator as a stand-alone module, without measuring impacts of its integration within the existing heterogeneous system.

System

The MANGO heterogeneous high-performance computing system was used as an architecture exploration platform for this research such that high-level encoding algorithm was instantiated on general-purpose processor cores while the custom designed HEVC DCT accelerator cores were used for DCT computations. Below we give a more detailed overview of the complete system and the approach.

Platform – MANGO platform

MANGO platform [15,16] is a heterogeneous high-performance computing system consisting of general-purpose compute nodes (GN) that are intertwined with heterogeneous acceleration nodes (HN), linked by an across-boundary homogeneous interconnect. GNs are based on CPUs (i.e. Intel Xeon), while HNs are FPGA-based next-generation manycore chips coupled with deeply customized heterogeneous computing resources. HNs are open and can be tailored for a specific purpose, which was used to incorporate DCT accelerator in MANGO architecture. Figure 1 shows the scheme of one cluster in MANGO platform with integrated DCT accelerator.

CPU host is a general node (GN) connected with the heterogeneous node (HN) deployed on four FPGAs. HN consists of different processing nodes, such as RISC multicore, GPU-like core, different accelerator cores and a hardware accelerator for DCT. Topology and number of the accelerators can be adapted to the requirements of the system. For each FPGA, there is one memory controller (MC) that connects the processing cores with local DDR memory. Resource manager located on a CPU host side manages the allocation of computing resources (GNs and HNs) to multiple concurrent application that can run on MANGO platform.

Host – HEVC encoder application

In-house "clean-room" implementation of HEVC encoder was used as a base for integration on the host side. The encoder was profiled and analysed to determine most time-consuming tasks of the application's runtime. The analysis was done on CPU-only single-core implementation configured for fast encoding, aiming at Just-in-Time video encoding. Verification and

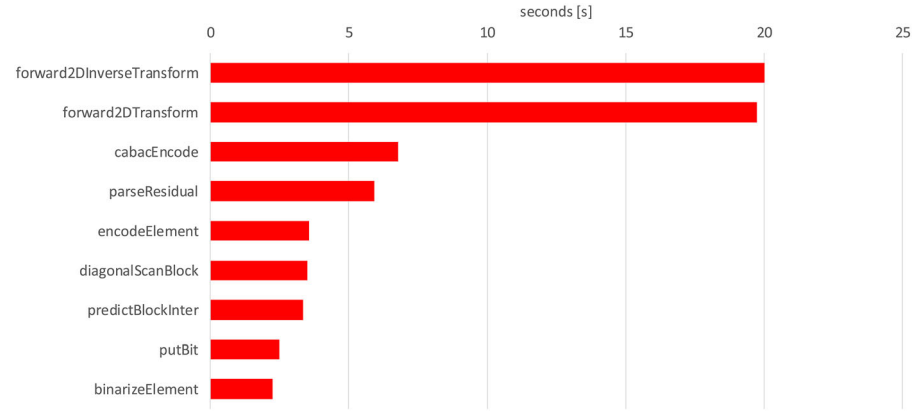

Figure 2. HEVC encoder profiling results.

validation of HEVC encoded bitstreams were conducted using StreamEye software [17].

Results are obtained using Intel Amplifier tool and are shown in Figure 2.

As can be seen from the figure above, most time-consuming tasks are: forward2DTransform and its counterpart forward2DInverseTransform. The main kernel used in both functions is 2-dimensional Discrete Cosine Transform, which is why DCT was chosen as a kernel that can be used to demonstrate benefits of offloading to a custom hardware accelerator. Several modifications to the applications were made to meet the requirements of the DCT accelerator. Residual samples had to be stored in a 16-bit format which halved the amount of data being transferred between host and accelerator. However, this modification leads to additional changes such as packing and unpacking of samples for AVX DCT implementation which had a minor impact on its performance.

Accelerator – DCT

Based on HEVC encoder analysis, a custom hardware accelerator for DCT was designed and implemented in FPGA. Symmetry properties of 2D DCT transform were exploited to design area-optimized, 1D DCT architecture that can be reused to implement full 2D core transform. The accelerator architecture is fully pipelined and applicable for all transform sizes used in HEVC (from 4×4 to 32×32 blocks). After evaluation as a stand-alone module, DCT accelerator core was integrated as a MANGO HN processing core.

Benchmarks

In the test scenario, HW DCT accelerator was used for processing DCT tasks. The baseline for benchmarks were two implementations of DCT kernel on Intel GN:

- Single thread
- Single thread + AVX

In addition to the different accelerator types, the MANGO platform enables the use of a group of units working in parallel, either isolated or in collaboration, on a given task. Therefore, we can also identify and exercise the following working modes in MANGO:

- Standalone mode – units work in standalone mode running parts of the tasks. No communication or collaboration is exercised between the host and the unit and between concurrent units.
- Host iterative mode – the unit works in an iterative mode with the host computing parts of the task. The kernel running on the unit is completely synchronized with the host application. All the units work in the same fashion but there are no direct interactions between units.

Due to the nature of the DCT algorithm and the design of DCT accelerator, the unit collaborative mode, also available in MANGO, was not exercised, and focus was put on standalone and host iterative mode.

Integration and programming approach

Standalone approach

The standalone mode was the first working mode in which HW DCT accelerator was integrated into the MANGO platform. In standalone mode, a single task represents a single matrix with sizes ranging from 32×32 to 4×4 which is then offloaded to the accelerator that processes the data and returns the transformed matrix to the host. For each task, initialization procedures such as registration of kernels, buffers, and events, allocation of resources or transfer of arguments are repeated. In the first phase, initialization of mango context and kernel function takes place. Depending on the matrix size, input, config and output buffers are registered and added to the task graph shown in Figure 3. Input data is then written to the buffers and the kernel is started. After this, host waits (or does some useful work) for the end event which indicates that the accelerator has processed input data. On the accelerator

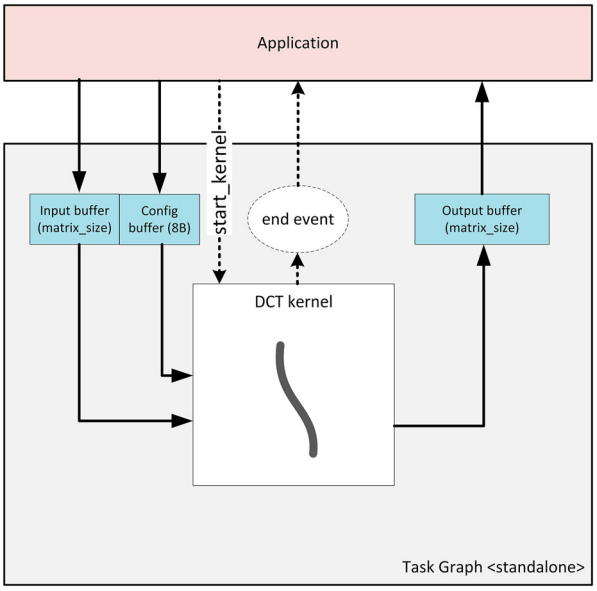

Figure 3. Task graph for standalone integration.

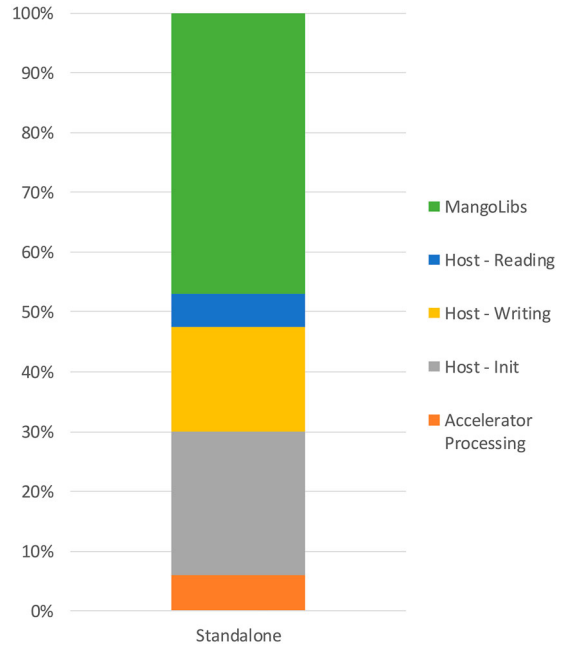

Figure 4. Time distribution for standalone integration.

side, starting kernel initiates the loop in which accelerator fetches the data through 512-bit data bus from MANGO memory. When data is fetched, it is put into the pipeline of the accelerator. With each clock cycle, data propagates through the pipeline until it is written in the local buffer from which it is stored to MANGO memory. When all data is stored, the task is finalized by triggering end event which notifies the host application that the data is ready to be retrieved from MANGO memory. The application then initiates the memory transfer and if needed restarts the above described cycle with a new set of input data.

To benchmark the performance, the application was run with DCT kernel offloaded to HW DCT accelerator operating in standalone working mode. A single 32×32 matrix of 16-bit residuals was transferred to the MANGO memory, processed by the accelerator and then returned to the host application. Figure 4 shows the time distribution for processing a single 32×32 residual block. Accelerator part corresponds to the time required by the accelerator to load, process and store data to MANGO memory. Host part corresponds to the time spent on executing the application on Intel GN which consists of filling the buffers with data intended for processing on the accelerator, transferring the data from host to MANGO memory and vice versa. As it can be seen from the figure, almost 50% of the runtime is spent on the *mangolibs* initialization which makes standalone working mode performance inefficient. Mangolibs is a software stack that consists of:

- API provided by the MANGO application library
- HN library that implements the communication functions between applications and resource manager with the FPGA subsystem
- BOSP – Barbeque Run-Time Resource Manager [18] and several MANGO unit specific modules.

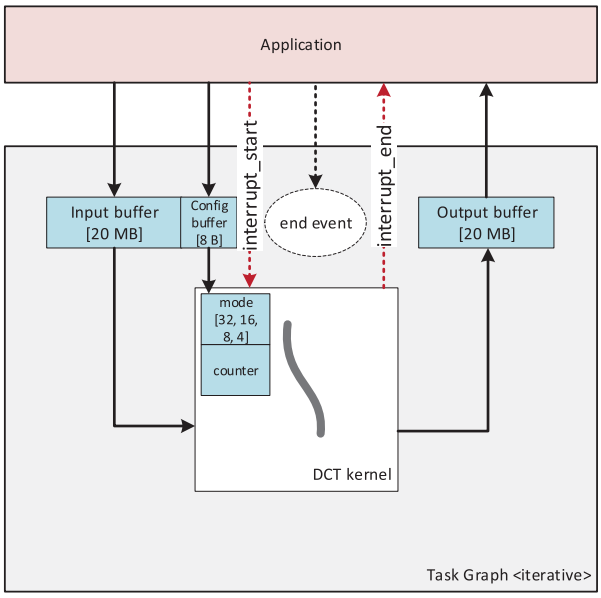

Figure 5. Task graph for iterative integration.

Iterative approach

To tackle the problem of *mangolibs* overhead, iterative working mode of the accelerator was proposed. In iterative working mode, resources are registered and allocated only once for any number of tasks offloaded to the accelerator. Several modifications were necessary to adapt the accelerator to iterative working mode. Since buffers now get registered only once, buffer sizes have been increased to *max_buffer_size* which is defined as 20 MB for the current version of the accelerator. The iterative working mode also required additional synchronization mechanisms which were implemented in the form of interrupts. Introduced changes reflected the task graph structure shown in Figure 5.

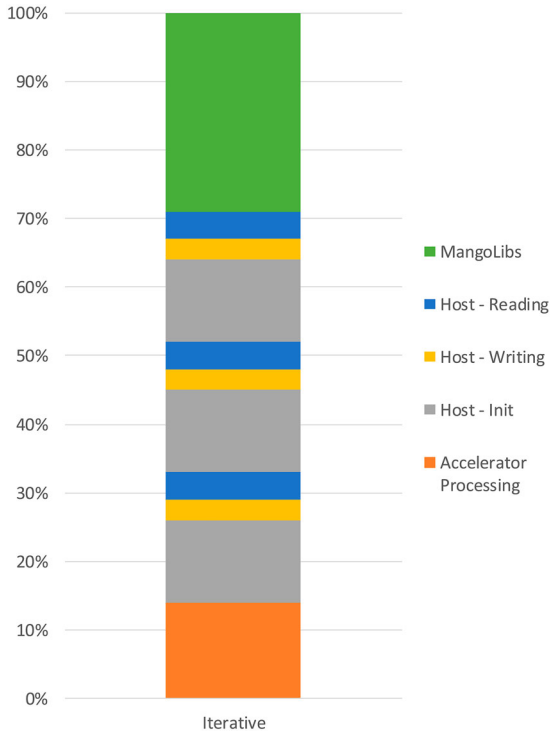

Figure 6. Time distribution for iterative integration.

The kernel is now started before the data is written to the buffer. After host transfers the input data to MANGO memory, it issues the *interrupt_start* which triggers the accelerators to start loading and processing the data. After all the data is processed and stored, accelerator issues the interrupt back to the host, signifying that the output data is available for reading. The host then reads the data and the cycle ends. This cycle is repeated if the host has more data to process and when there is no more data, the host issues the end event which indicates that the accelerator is no longer used by the application. Figure 6 shows time distribution between different stages when processing three 32 × 32 blocks in iterative working mode. As it can be seen, *mangolibs* overhead, in this case, is lowered and will continue to drop as the number of blocks for processing increases.

To benchmark the performance of the iterative working mode, 10,000 tasks of 32 × 32, 16 × 16, 8 × 8, 4 × 4 blocks were offloaded in series to the HW DCT accelerator. The results are shown in the following table:

It is interesting to note that block size does not affect the time needed to process the data even though the pipeline for processing 4 × 4 matrices is much shorter than the pipeline for processing 32 × 32 matrices. The reason for this is that the bottleneck for processing time is the time spent on loading data by the accelerator and then storing it once the processing is finished. This is a limitation introduced by the fact that the MANGO platform is indeed architecture exploration platform for HPC and not processing efficient HPC platform itself. The memory bandwidth impacts the performance of

Table 1. Time distribution – 10,000 × 2 kB.

Buffer	Block Size	Total	Processing time	Read time	Write time
2 kB	32 × 32	17,68	14,45	1,800	0,406
2 kB	16 × 16	17,67	14,45	1,798	0,406
2 kB	8 × 8	17,69	14,46	1,801	0,406
2 kB	4 × 4	17,69	14,46	1,804	0,407

the accelerator, which is an obvious and expected conclusion. However, additional optimizations which can lead to better performance of the accelerator are possible. These optimizations will be explained in the next section. Since the matrix size doesn't impact the performance, the rest of the document will consider only 32 × 32 matrices.

Optimizations

Table 1 shows average time distribution per phase when processing 10,000 instances of 32 × 32 blocks. Write and read time corresponds to the time needed to transfer data from host to the MANGO memory or from MANGO memory to the host respectively. Processing time is the duration from the moment when the *interrupt_start* has been sent to the accelerator to the moment when *interrupt_end* has been received by the host. Processing time thus includes the time needed for the accelerator to load data from the memory, process it, and then store it back. Here we do not address how the total processing time of the accelerator is distributed between memory accesses and data passing through the pipeline. Usually, small buffer sizes suffer from the problem of significant memory transfer overhead but in this case, almost 90% of the time is contributed to the processing time of the accelerator, as shown in Table 1. However, a detailed analysis of the different buffer sizes showed that processing time is significantly affected by the time accelerator spends on loading and storing the data. The results of this analysis are shown in Table 2.

Total time spent on transferring and processing data linearly grows with the increase of buffer size from 2 kB up to 64 kB. However, when the buffer becomes 64 kB or larger, the average time needed to transfer and process data significantly shortens and thus we witness significantly higher efficiency. The explanation for this lies in the fact that for buffer sizes smaller than 64 kB, data is transferred from the host to MANGO using item network while for larger buffers, shared buffers are used. Figure 7 shows the time cost per single block for buffer sizes ranging from 2 to 20,000 kB. The total time cost for transferring and processing single block in a 60 kB buffer is 14,67 milliseconds while for 64 kB buffer it is 0,61 milliseconds. The time cost continues to decline to 0,44 for a buffer size of 2000 kB after which it flatlines. This analysis shows that to efficiently exploit accelerator, buffer sizes should be greater than 2 MB. In the accelerator, the max buffer size is limited to 20 MB

Table 2. Benchmarked performance of DCT kernel for different I/O buffer sizes, all time are shown in milliseconds.

Buffer size	# of runs	Processing time	Read data time	Write data time	Total time	32 × 32 blocks processed	Total per block
2 kB	100	15.5345	1.8409	0.4142	18.8748	1	18.8748
4 kB	100	27.7040	2.1224	0.7477	32.1012	2	16.0506
20 kB	100	138.3400	4.1918	3.3358	150.1410	10	15.0141
60 kB	100	414.2460	7.6073	9.5798	440.0900	30	14.6697
64 kB	100	2.6461	8.9927	7.1893	19.4663	32	0.6083
200 kB	100	4.5282	25.1933	19.0508	49.4060	100	0.4941
2 MB	100	29.9313	233.6350	176.7760	440.9960	1000	0.4410
10 MB	100	143.0100	1233.7600	897.8580	2275.2900	5000	0.4551
20 MB	100	284.5090	2461.4800	1790.8300	4537.5000	10000	0.4538

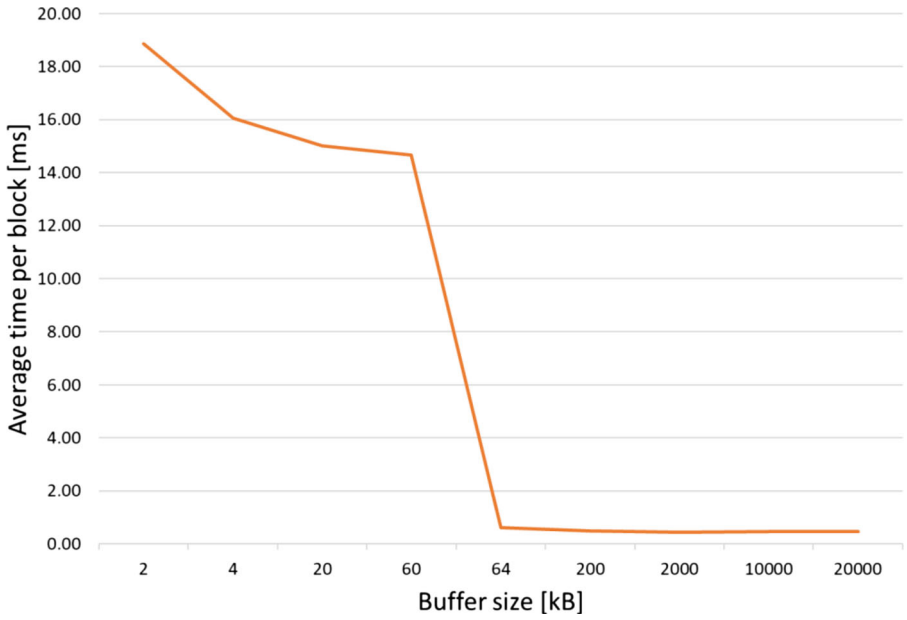

Figure 7. Total time for transferring and processing per block.

Table 3. Comparison of DCT kernel on different processing cores.

Core	Block size	Process	Read	Write	Total per block
DCT	32 × 32	284.50	2461.48	1790.83	0.4538
DCT	16 × 16	283.10	2465.58	1790.20	0.1135
DCT	8 × 8	279.32	2451.88	1791.44	0.0283
DCT	4 × 4	286.05	2459.17	1789.57	0.0071
CPU	32 × 32	1981.51	–	–	0.1985
CPU	16 × 16	1016.66	–	–	0.0254
CPU	8 × 8	541.27	–	–	0.0033
CPU	4 × 4	331.89	–	–	0.0005
AVX	32 × 32	580.54	–	–	0.0580
AVX	16 × 16	363.17	–	–	0.0090
AVX	8 × 8	181.70	–	–	0.0011
AVX	4 × 4	220.33	–	–	0.0003

which is enough to support use cases of transcoding full HD video sequences.

Performance evaluation

Performance evaluation was performed for three types of accelerator kernels: HW DCT accelerator, CPU and CPU + AVX. In all tests, a buffer size of 20 MB was used and blocks sizes ranging from 32 × 32 to 4 × 4 were transferred. Results are shown in Table 3.

If we take into consideration the time spent on average for processing a single block of data, AVX is the fastest, followed by CPU and HW DCT accelerator. Total time consists of processing time, read time and write time. Read and write time represent time spent for transferring data from the host to MANGO and vice versa and are equal to 0 for CPU and AVX since in this case, the data never leaves the host. However, read and write time participate with a share of over 90% when it comes to HW DCT accelerator. This can be explained with MANGO platform being an exploration platform for HPC. Implemented data transfer buses do not exercise real-world scenarios in which high bandwidth buses are fully exploited. If we take only processing time into consideration, for 32 × 32 and 16 × 16 blocks, HW accelerator, even running at 40 MHz, outperforms Intel (running at 3.3 GHz) AVX optimized implementation. However, for smaller blocks, 8 × 8 and 4 × 4, HW DCT accelerator doesn't provide the fastest results. The main issue that explains this behaviour is slow memory access from the accelerator to MANGO memory which is the bottleneck of accelerator processing time. Because of this bottleneck, processing times of HW DCT accelerator for all block sizes are approximately the same even though the pipeline for processing 4 × 4 blocks is much shorter than the one for processing 8 × 8, 16 × 16 or 32 × 32 blocks. The second fact that needs to be considered is that the accelerator runs on

MANGO architecture with the clock of 40 MHz while in real world scenario it can run with a frequency that is one order of magnitude larger for FPGA implementation and even higher for ASIC implementation.

Conclusion

In this paper, we have investigated several approaches for integration of custom designed hardware accelerator for discrete cosine transform in novel heterogeneous architecture developed as a part of Horizon 2020 project MANGO: exploring Manycore Architectures for Next-generation HPC systems. Fully pipelined, area optimized DCT accelerator is used to improve the performance of compute and data-intensive video encoding process based on a HEVC standard. Different approaches of accelerator utilization were identified as a standalone and iterative approach. Each accelerator and approach were analysed and benchmarked. Iterative mode proved to be more efficient than standalone mode due to high *mangolibs* initialization overhead. Analysis of the data transfer from MANGO to host and vice-versa showed that for efficient data transfer buffer sizes should be 200 kB or larger.

Performance comparison of different processing cores showed that DCT HW accelerator, even running at 40 MHz mode, outperforms Intel (running at 3.3 GHz) AVX optimized implementation for block sizes of 16×16 or larger when it comes to processing time. However, due to the MANGO platform being an exploration platform for HPC, data transfers and memory access provide a bottleneck for maximum utilization of HW DCT accelerator.

Our future work in this area will include the development and integration of other types of accelerators designed for HEVC video encoding and transcoding. Different types of processing units, besides custom accelerator-based cores, will also be investigated, such as RISC-V and GPU-like cores. Resource manager will be adapted and improved to facilitate management of all integrated modules.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This project has received funding from the European Union's H2020 Future and Emerging Technologies under [grant agreement No. 671668].

ORCID

Igor Piljić http://orcid.org/0000-0003-2345-0322
Leon Dragić http://orcid.org/0000-0002-4558-7269
Mario Kovač http://orcid.org/0000-0002-8365-7002

References

[1] Cisco. Cisco visual networking index: forecast and methodology, 2017–2022. [cited 2018 Nov 26]. Available from: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html
[2] Sullivan GJ, Ohm J-R, Han W-J, et al. Overview of the high efficiency video coding (HEVC) standard. IEEE Trans Circuits Syst Video Technol. 2012;22(12):1649–1668.
[3] Bossen F, Bross B, Sühring Ket al. HEVC complexity and implementation analysis. IEEE Trans Circuits Syst Video Technol. 2012;22(12):1685–1696.
[4] d Souza DF, Roma N, Sousa L. Opencl parallelization of the HEVC de-quantization and inverse transform for heterogeneous platforms. 2014 22nd European Signal Processing Conference (EUSIPCO); Lisbon; 2014. p. 755–759.
[5] Wang B, et al. Efficient HEVC decoder for heterogeneous CPU with GPU systems. 2016 IEEE 18th International Workshop on Multimedia Signal Processing (MMSP); Montreal, QC; 2016. p. 1–6.
[6] Ma A, Guo C. Parallel acceleration of HEVC decoder based on CPU + GPU heterogeneous platform. 2017 Seventh International Conference on Information Science and Technology (ICIST); Da Nang; 2017. p. 323–330.
[7] Amish F,Bourennane E. A novel hardware accelerator for the HEVC intra prediction. 2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS); Grenoble; 2015. p. 1–4.
[8] Sjövall P, Viitamäki V, Vanne J, et al. FPGA-Powered 4K120p HEVC Intra encoder. 2018 IEEE International Symposium on Circuits and Systems (ISCAS); Florence; 2018. p. 1–5.
[9] Meher PK, Park SY, Mohanty BK, et al. Efficient integer DCT architectures for HEVC. IEEE Trans Circuits Syst Video Technol. 2014 Jan;24(1):168–178.
[10] Chatterjee S, Sarawadekar KP. A low cost, constant throughput and reusable 8X8 DCT architecture for HEVC. Proceedings of IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS); Abu Dhabi, United Arab Emirates; 2016 Oct 16–19. p. 1–4.
[11] Bolaños-Jojoa JD,Velasco-Medina J. Efficient hardware design of N-point 1D-DCT for HEVC, Proceedings of 20th Symposium on Signal Process, Images and Computer Vision (STSIVA), Bogota, Colombia; 2015 Sept 2–4. p. 1–6.
[12] Abdelrasoul M, Sayed MS,Goulart V. Scalable integer DCT architecture for HEVC encoder. Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI); Pittsburgh, Pennsylvania, 2016 Jul 11–13. p. 314–318.
[13] Diniz CM, Shafique M, Bampi S, et al. A reconfigurable hardware architecture for fractional pixel interpolation in high efficiency video coding. IEEE Trans Comput-Aided Des Integr Circuits Syst. 2015 Feb;34(2):238–251.
[14] Diniz CM, Shafique M, Dalcin FV, et al. A deblocking filter hardware architecture for the high efficiency video coding standard. 2015 design, Automation & Test in Europe Conference & Exhibition (DATE); Grenoble; 2015. p. 1509–1514.
[15] Flich J, Agosta G, Ampletzer P, et al. MANGO: exploring manycore architectures for next-Generation HPC systems. 2017 Euromicro Conference on Digital System Design (DSD); Vienna; 2017. p. 478–485.

[16] Flich J, et al. The MANGO FET-HPC project: an overview. 2015 IEEE 18th International Conference on Computational Science and Engineering; Porto; 2015. p. 351–354.
[17] Elecard: StreamEye software. Available from: https://www.elecard.com/products/video-analysis/streameye.
[18] Massari G, Libutti S, Fornaciari W, et al. Resource-aware application execution exploiting the BarbequeRTRM. Proceedings of 1st Workshop on Resource Awareness and Application Autotuning in Adaptive and Heterogeneous Computing (RES4ANT); CEUR; 2016. p. 3–7.