Performance Grading of GPU-based Implementation of Space Computing Systems Image Compression

Olympia Kremmyda Vasilis Dimitsas Dimitris Gizopoulos
Department of Informatics and Telecommunications
University of Athens, Greece
{ol-krem, vdimi, dgizop} @di.uoa.gr

***Abstract*— Modern GPUs can accelerate the execution of inherently data-parallel applications in General-Purpose GPU (GPGPU) computing. However, real-world algorithm realizations on GPUs differ from synthetic benchmarks. In this poster we stress the performance limits of GPUs. We measure the performance of our CUDA implementation of a recommended image compression algorithm (CCSDS-122.0-B-1) for space data systems on NVIDIA GPUs for various image sizes and compare it to the execution on x86 CPUs. The significant control-intensive parts and irregular memory access patterns under-utilize the GPU architecture and effectively serialize large parts of the GPU kernel execution.**

We present the execution results on two systems with different CPU/GPU setups. In both systems, our GPU implementation of the algorithm eventually leads to a very moderate 1.1x to 2.1x speedup. Discussion of the performance bottlenecks leads to insights on how GPU execution can be improved in the future by either revising the algorithms or by supporting control-intensive execution on the GPU cores.

Keywords— CCSDS-122.0-B-1, image compression, DWT, BPE, GPGPU, CUDA implementation

I. Introduction

We characterize the GPU execution performance of the first full CUDA implementation of an important image compression algorithm for space data systems recommended in CCSDS standards [2] [3]. The algorithm consists of two major parts that almost equally determine the overall performance of the algorithm: the Discrete Wavelet Transform (DWT) on the image raw data and the subsequent Bit Plane Encoding (BPE).

II. Algorithm and Implementation

The first part uses DWT to decompose and transform the image data into wavelets coefficients. We focus on the floating-point DWT version and lossy compression because it can lead to more effective image compression quality. Pixels are processed first in rows and then in columns, in 3 levels. Low-pass and high-pass frequency zones of wavelets are generated. In the GPU implementation of the algorithm, the image is distributed into blocks of threads using shared memory to store the transformed pixels during the 3 levels and each thread processes a single pixel.

The decomposed data are grouped into blocks and segments in order to be encoded from the BPE. Coefficients with high information density (DC coefficients) are encoded separately using Rice Algorithm. The rest of the coefficients (AC) are processed into bit planes, groups of bits in a specific bit position of a binary number, and their encoding is completed in 4 stages producing the compressed stream using also Rice Algorithm. The encoding of each segment is performed independently and as a result each thread processes a single segment.

III. Experimental Results

We evaluate our GPU implementation of the algorithm on two CPU/GPU systems and compare them against the sequential CPU implementations. The configurations of the two systems are summarized in Table I.

TABLE I. Configuration Of The Two Evaluation Systems

	Server 1	**Server 2**
CPU	AMD Phenom™ II X4 965 4 cores, @ 3.4GHz, 45nm (Released November 2009) 8 GB main memory	Intel® Core™ i7-3970X 6 cores, @ 3.50GHz, 32nm (Released November 2012) 32 GB main memory
GPU	NVIDIA Tesla C2070 (Fermi architecture), 6GB GDDR5 RAM, 448 CUDA cores, 40nm (Released July 2010)	NVIDIA Tesla K20 (Kepler architecture), 5GB GDDR5 RAM, 2496 CUDA cores, 28nm (Released November 2012)

Fig. 1, Fig. 2 and Fig. 3 visualize the relation between the execution times of the algorithm's implementation (Table II) in baseline CPU code and GPU code and the corresponding speedups or slowdowns that GPU execution delivers. The values are the ratios between the CPU time and the GPU time.

We performed runs of the total algorithm execution for:

- 1, 2, 4, 8, 16, 32, 64, 128, 256 and 512 threads per CUDA block; and
- 4, 16 and 32 image coefficient blocks per segment that BPE processes

The best cases after testing all the possible combinations are represented below.

Our performance assessment reveals that despite the very large speedup that GPUs offer to the DWT part (ranges from 4x to 110x depending on the CPU/GPU combination and the image sizes), the inherent control-intensive execution and irregular memory access patterns of the BPE part seriously under-utilize the GPU hardware and perform significantly slower than the baseline CPU code. The aggregate effect of these conflicting behaviors is a very moderate overall speedup for the complete algorithm between 1.1x to 2.1x for the larger images (8K).

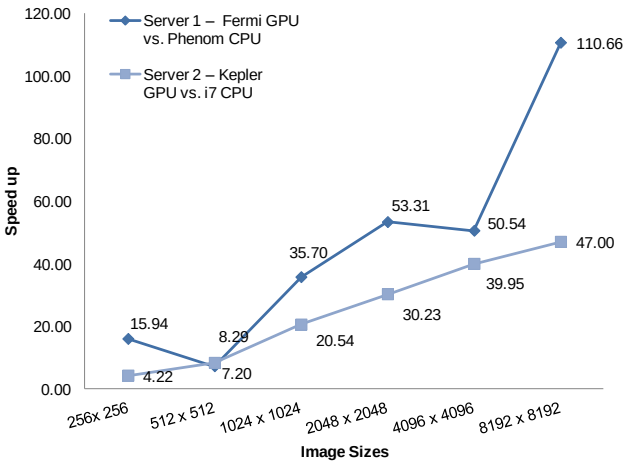

Fig. 1: Speedup of DWT execution on the GPUs of both systems compared to baseline CPU execution.

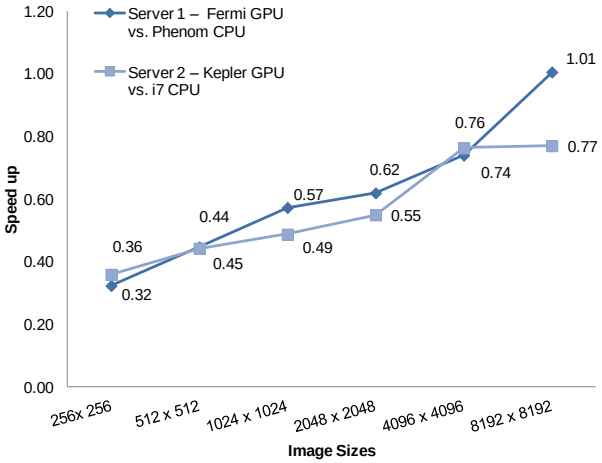

Fig. 2: BPE kernel vs. BPE function (CPU) speedup (in most cases a slowdown is measured; CPU/GPU < 1).

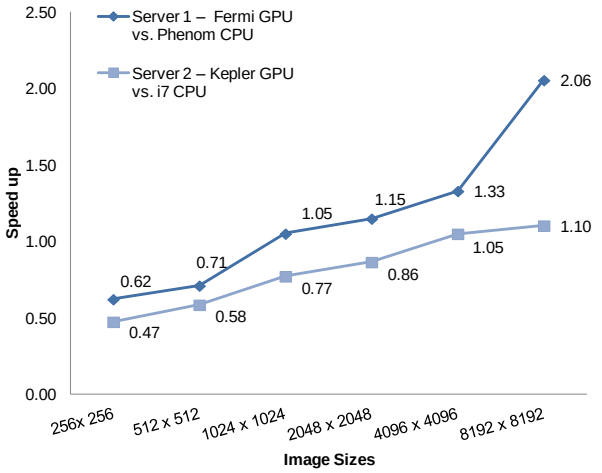

Fig. 3: GPU vs. CPU overall speedup (CPU/GPU time).

TABLE II: EXECUTION TIMES (MSEC) OF THE COMPLETE ALGORITHM ON SERVER 1 AND SERVER 2. BASELINE CPU AND GPU EXECUTIONS ARE SHOWN FOR ALL DIFFERENT IMAGE SIZES.

Image size	Server 1 GPU (Fermi)	Server 1 CPU (Phenom)	Server 2 GPU (Kepler)	Server 2 CPU (i7)
256 x 256	19.63	12.19	18.83	8.90
512 x 512	49.56	35.19	47.52	27.78
1024 x 1024	142.69	149.86	123.33	95.08
2048 x 2048	533.14	612.45	449.46	388.50
4096 x 4096	1983.89	2636.69	1795.63	1884.27
8192 x 8192	6344.08	13038.89	7008.64	7733.71

IV. RELATED WORK

The national space agencies led CCSDS committee recommends the CCSDS-122.0-B-1 algorithm as the most appropriate image data compression mechanism for 2D digital, spatial image data [2] [3]. This work is the first to port the complete CCSDS Recommended Standard 122.0-B-1 in CUDA and to execute the whole algorithm in GPU. In [1] authors implement the 2D wavelet transform in CUDA but they also transpose the image data in the GPU; the shared memory of the GPU is not exploited and all transformation take place in the global memory. Tenllado et al. [4] discuss an OpenGL implementation of DWT. Van der Laan et al. [5] focus on DWT; both works report significant speedup for DWT as in our case. Furthermore, there are no studies for parallelizing BPE algorithm in CUDA.

V. CONCLUSIONS

We have presented and evaluated the first full CUDA implementation of the space data systems image compression algorithm recommended by the national space agencies. Results show that the performance bottlenecks of the BPE part of the algorithm limit the overall speedup that GPU execution delivers to a moderate 1.1x to 2.1x. An exploration of the major parameters of GPU execution that affect performance of CUDA implementation is also done.

ACKNOWLEDGMENTS

This work is supported by EU's European Social Fund (ESF) and Greek national funds under the "DIaSTEMA" project and the "Thales/HOLISTIC" project as well as by the Special Account for Research Grants (SARG) of the University of Athens.

REFERENCES

[1] Franco, J., Bernabé, G., Fernández, J., and Acacio M. E., "A Parallel Implementation of the 2D Wavelet Transform Using CUDA", Euromicro PDP, 2009.

[2] Image Data Compression CCSDS-120.1-G-1. 2007. http://public.ccsds.org/publications/archive/120x1g1e2.pdf

[3] Image Data Compression CCSDS-122.1-B-1. 2005. http://public.ccsds.org/publications/archive/122x0b1c3.pdf

[4] Tenllado, C., Setoain, J., Prieto, M., Piñuel, L., Tirado, F., "Parallel Implementation of the 2D Discrete Wavelet Transform on Graphics Processing Units: Filter Bank versus Lifting", IEEE Transactions on Parallel and Distributed Systems, vol. 19, no. 3, Mar 2008.

[5] Van der Laan, W., Jalba A. C., Roerdink, J., "Accelerating Wavelet Lifting on Graphics Hardware Using CUDA", IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 1, Jan 2011.