PARALLELIZING AND COMPARING BLOCK MATCHING ALGORITHMS IN H.264 ON NVIDIA CUDA

[1]KOCHMURIYIL JANCY JAMES, [2]RENUKA JOSHI, [3]DARSHAN PAWAR, [3]SAGAR KARWANDE, [4]CHANDRAMA THORAT

[1,2,3,4]Department of Computer Engineering, JSPM's Rajarshi Shahu College of Engineering, Pune, India
E-mail: janamy.16793@yahoo.com, renu.joshi4@gmail.com, darshanp40@gmail.com, sagar_karwande123@yahoo.com, chandrama1684@gmail.com

Abstract- Judging by the amount of research work that is being carried out in the field of parallel computing, it seems to be the next big thing and it is safe to say that this phenomenon is here to stay. H.264 is a powerful yet flexible video encoding standard which provides better video quality at lower bitrates. The motion estimation part of video compression is very time consuming. This computational time can be reduced by parallelizing it on NVIDIA's CUDA a GPGPU which ensures a better performance than its non-parallel counterpart using only CPU. This paper strives to provide a comprehensive look into the three block matching search algorithms, namely Full Search, Three Step Search and Uneven Multi Hexagon Search used in motion estimation. We parallelize and implement the algorithms in H.264 on CUDA and compare their results to decipher which provides better efficacy.

Keywords- Compute Unified Device Architecture (CUDA), General Purpose Graphics Processing Unit (GPGPU), H.264 Video Coding Standard, Codec, Motion Estimation, Block Matching Algorithms

I. INTRODUCTION

Since time immemorial, humans have considered time as one of the most precious things, trying their best never to waste it. The fast paced life of the 21st century calls for faster computations and processing. To fulfil these demands, humans have come up with various techniques like multitasking and parallelization. Humans and their incessant need to emulate the real world and to save precious amount of time, have incorporated parallelism into computing and processing as well. Traditionally, programs and algorithms are written in such a way that they are executed on the CPU in a sequential manner i.e. each instruction is executed one at a time and one after the other. However, in parallel computation a large program is subdivided into smaller parts and then the parts that are data independent of each other are executed in parallel on different processors. GPU has a many-core architecture which compared to a CPU provides a much better performance when it comes to executing parallel data computations.

An uncompressed video data in its native format tends to contain a lot of redundant data and requires a lot of storage space. The redundancy can be in spatial redundancy or temporal redundancy. Software that enables compression and decompression of video data are known as a Video Codec. H.264 or MPEG-4 Part 10, Advanced Video Coding (MPEG-4 AVC) is a video coding standard developed by the collaborative efforts of ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC JTC1 Moving Picture Experts Group (MPEG). The partnership effort is known as the Joint Video Team (JVT). During compression, the temporal redundancy is generally exploited using predictive coding techniques, usually a lossy technique, but the loss is negligible in the eyes of a human, whereas the spatial redundancy is generally exploited using transformation, usually a lossless technique. H.264 video encoding is computationally extensive and with the increase in resolution of the input video, the computational complexity also increases. This computational complexity induces a need to accelerate the working of H.264 video encoder. Motion estimation is known to be the most time consuming part in an H.264 video encoder. The immense parallelism provided by NVIDIA CUDA GPGPU can be used to reduce the computation time consumed by motion estimation. In this paper, we explore the various block matching algorithms and evaluate their performance when parallelized and implemented on CUDA.

II. MOTIVATION AND BACKGROUND

H.264 is a video coding standard which is one of the most popular and one of the most widely used video codecs along with its contemporaries like Ogg Theora and VP8. H.264 encoders provide better quality at lower bitrates making it popular among video uploaders on the internet despite it being surrounded with controversies regarding it being encumbered with patented technologies. H.264 is approximately twice as better as MPEG -2 which has been made possible by trade off among Quality (distortion), Computation (complexity) and Compression (bitrate). Touted as one of the best video codec technology by many, H.264 compression is scalable from small screen devices like cell phones and smartphones to HDTV Broadcasting. It provides good quality even when scaled. Youtube revolution in the 21st century brought about a craze for video streaming and

uploading. A number of software like Adobe Flash which is popular for streaming video on the internet, Microsoft's Quicktime 7, iTunes store etc. use H.264. The need for transcoding has also increased owing to video streaming. All these call for increased speed and better performance which can be achieved by making use of GPU. A GPU consists of thousands of cores in which each core has multiple threads to enable parallelism, whereas CPU only consists of multiple cores.

By carefully breaking a problem into parallelizable parts, we can achieve high level of efficiency provided that we follow certain guidelines for optimization.

This will, to a great extent, enable the CUDA to work efficiently even if data dependencies are present. If carefully examined, certain parts of H.264, particularly the computational part of SAD (Sum of Absolute Differences), can be parallelized to work and produce results simultaneously.

In this paper, we propose a system where the computationally extensive part of an H.264 encoder can be parallelized using the massive parallelism provided by NVIDIA CUDA GPGPU. The implementation of the above mentioned system would enhance the performance of video encoding.

III. LITERATURE SURVEY

A. GPU (Graphics Processing Unit)

GPU (Graphics Processing Unit) is a graphic card processor which consists of a many-core architecture. Unlike the multi cores of a CPU, the GPU may have thousands of cores and in turn each of these cores has multiple threads. When compared to the execution time required by a CPU working alone, the GPU requires lesser time. GPU was initially developed for 2-D or 3-D Graphics and gaming in various applications namely mobile phones, personal computers etc.

B. GPGPU (General Purpose Graphics Processing Unit)

Now-a-days, GPU has found its use in parallel computing, where the large program is divided into smaller parts and these smaller parts, if data is independent of each other, can be run simultaneously using threads on different processors.

The GPU that performs general computations apart from the graphics or gaming related ones that are usually performed on CPUs are known as GPGPU (General Purpose Graphics Processing Unit).

C. CUDA (Compute Unified Device Architecture)

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA. GPUs can be used as a GPGPU using CUDA. CUDA provides an environment wherein we can write parallel programs in C/C++. The only necessary requirements for its efficient working is that these programs should have some parts that are data independent of each other, which can then be parallelized and executed on GPU and they should also have enough computations such that there is enough to be run parallel using threads.

These programs usually consists two major parts, one part that is not capable of data parallelism or is capable of minimal data parallelism and another part that has massive data parallelism potential.

Generally, the part that has less or no parallelism is run on the CPU, which acts as the host, whereas the part that displays a lot of data parallelism is run on the GPU, the device with parallel architecture. These two phases of the program are separated using NVIDIA C compiler (nvcc).

The program execution starts its execution on the CPU (the host) and when there is a kernel function invocation, the control of the execution of the program is handed over to the GPU which consists of multiple grids where many threads are generated.

A grid may have 65535 blocks and each block may have 512 threads. These threads execute small data independent parts of the program simultaneously on the GPU.

D. Motion Estimation

Finding the relative motion between two frames to remove temporal redundancy is known as Motion Estimation.

Block Matching Algorithm is the most commonly used technique for motion estimation when it comes to video compression.

One of the metrics that is most commonly used in Block Matching Algorithms is SAD (Sum of Absolute Differences) using which the best matching block is selected; this is the metric that we make use of in our proposed system.

H.264 allows the encoder to do motion estimation on 16x16, 16x8, 8x16 and 8x8 blocks and these blocks can be further partitioned into blocks as small as 4x4 for better performance as using smaller blocks allows for more compression and better quality.

Since there are many frames present in a video sequence per unit of time, the differences between the adjacent frames minute, this minute difference leads to temporal redundancy. In a single frame, the adjacent pixels tend to be very similar which leads to spatial redundancy. Now, the inter-prediction

removes temporal redundancy and the intra-prediction removes spatial redundancy.

IV. PROPOSED SYSTEM

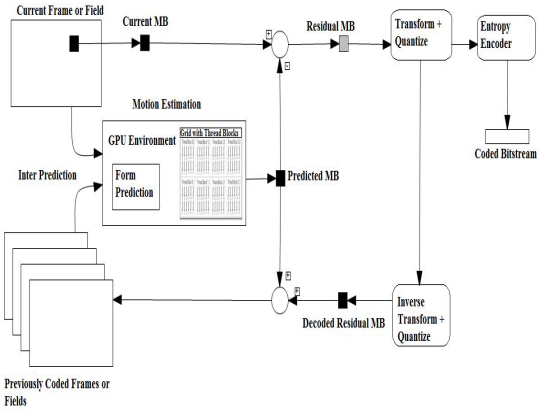

Fig. 1. Proposed System Architecture

Fig. 1 shows the architecture of the proposed model. As is shown in fig. 1, the main blocks of the architecture are: i) Motion Estimation ii) CUDA implementation iii) Transformation and Quantization iv) Entropy Encoding.

The proposed system will work as follows:
1) Input raw file (.yum/.y4m) to the system.
2) Motion Estimation – selects the best match by reducing the temporal redundancy by finding the relative motion between frames.
3) CUDA implementation – SAD calculation in motion estimation is parallelized and executed.
4) Transformation and Quantization – transformation exploits statistical redundancy and quantization exploits psychovisual redundancy.
5) Entropy Encoding – encodes the quantized data.
6) The output is coded bitstream.
The operating system we use is Ubuntu 12.04 with CUDA toolkit installed. The proposed system has minimum requirements as 256MB RAM, 20MB Hard Disk space, Intel Dual Core Processor (64 bit) microprocessor and an NVIDIA GPU with CUDA support.

V. IMPLEMENTATION

The execution of the encoding process starts off on CPU and as soon as the kernel function is invoked, the encoding process is offloaded to GPU where a number of threads are generated to carry out the computation of SAD (sum of absolute differences) in a parallel manner and then the best matching block is selected by finding the minimum SAD. Three different block matching algorithms that are used in motion estimation are mentioned below.

E. Exhaustive Full Search Algorithm
Full search allows for the most parallelism. FSA is the most robust among all block matching algorithms as it uses brute force to find the best matching pixel. It performs a thorough search by considering every pixel in the search range as a candidate for best match. It ensures accurate results but takes a lot of time. Full search tends to computationally expensive.
Initially, we consider the centre pixel and a search range is set and macro blocks are defined. SAD values are calculated for macro blocks. Each macro block is compared to the reference macro block using SAD values and the best match is selected.

F. Three-Step Search Algorithm
TSS was originally proposed by Koga et al in 1981. TSS is popular due to its simplicity.

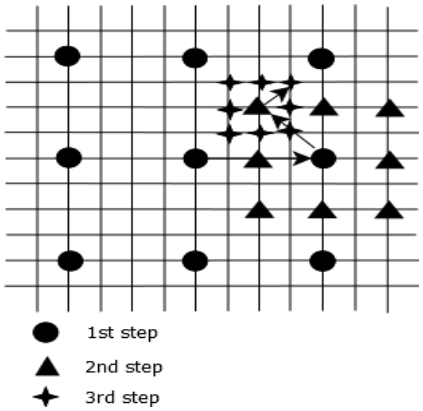

Fig. 2. Three-Step Search Process

The search process can be illustrated as:
1) A step size is chosen and the search starts from the centre pixel. Eight pixels that are at a distance equivalent to the step size from the pixel are chosen as the candidates for comparison.
2) The step size is halved and now we move to a point with minimum distortion.
Until the step size becomes smaller than 1, steps 1) and 2) are repeated. In TSS the step size is chosen such that search process stops in three stages as shown in fig. 2.

G. Uneven-multi Hexagon Search Algorithm
UMHexagon Search Algorithm is combination of various block matching algorithms providing both accuracy and speed. It adaptively adopts different search pattern according to SAD.
The search process of UMHexagon Search Algorithm with step size 16 is as follows:

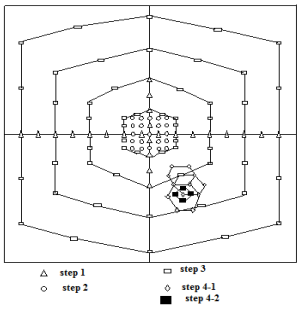

Fig. 3. Uneven-multi Hexagon Search Algorithm

1) Unsymmetrical Cross Search
2) Small Rectangular Full Search
3) Uneven-Multi Hexagon Grid Search
4) Extended Hexagon based Search

CONCLUSION AND FUTURE SCOPE

In this paper, we parallelize the mentioned block matching search algorithms in H.264 on NVIDIA CUDA GPGPU making use of the immense parallelism that it provides. Practical implementation of the above would have a significant impact on the present method of video encoding. This paper primarily focuses on just one part of the proposed system i.e. motion estimation, leaving the rest of the system remain untouched. Further increase in scope calls for exploration of transformation, quantisation and entropy encoding on a parallel platform. Furthermore, the same implementation can be explored on different GPGPUs.

ACKNOWLEDGMENT

This paper manifests the research work that is being done by us as a part of our final year curriculum, in which we implement H.264 on CUDA platform in the CUDA research centre established at RSCOE, University of Pune. We could achieve this milestone due to the unstinted support, dedication and encouragement of our project guide Prof. C.G.Thorat and our project coordinator Prof. V.M.Barkade. We would like to express our immense gratitude towards them. We would also like to thank our Head of Computer Department Prof. Dr. A.B.Bagwan for his support and resource provision at every step of the way.

REFERENCES

[1] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, “Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC)”, Std. JVTG050, 2003.

[2] W. Chen, H. Hang, “H.264/AVC motion estimation implementation on Compute Unified Device Architecture(CUDA)”, pp – 697 -700, IEEE 2008.

[3] Aleksandar Colic, Hari Kalva, Borko Furht , “Exploring NVIDIA-CUDA for Video Coding”, Dept. of Electrical and Computer, Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, pp- 13-22, ACM, 2010

[4] Youngsub Ko, and Youngmin Yi, Soonhoi Ha, “An Efficient Parallel Motion Estimation Algorithm and X264 Parallelization in CUDA”, Dept. of Electrical and Computer, Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, pp- 1-8, IEEE, 2011.

[5] I. Richardson. “H.264/AVC and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia”. Chichester: John Wiley and Sons. 2003.

[6] The H.264 Advanced Video Compression Standard, second edition, http://as.wiley.com/WileyCDA/WileyTitle/productCd-0470516925.html

[7] Parallel computing, http://en.wikipedia.org/wiki/Parallel_computing

[8] CUDA, http://www.nvidia.com/object/cuda_home_new.html.

[9] http://www.videomaker.com/article/15362-video-formats-explained

[10] http://www.videomaker.com/article/15374-the-current-state-of-codecs

[11] H.264/MPEG-4 AVC, http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC

[12] NVIDIA CUDA Programming Guide version 4.0 (5/6/2011), http://www.shodor.org/media/content//petascale/materials/UPModules/matrixMultiplication/cudaCguide.pdf

[13] Jinglin Zhang, Jean-Francois Nezan, Jean-Gabriel Cousin,”Implementation of Motion Estimation Based on Heterogeneous Parallel Computing System with OpenCL”, Universite Europeene de Bretagne, France, 14th International Conference on High Performance Computing and Communications, pp- 41-45, IEEE, 2012.

[14] Mohsen Jamali, Joseph Peters, Shervin Shirmohammadi, “Complexity Aware Encoding of the Motion Compensation Process of H.264/AVC Video Coding Standard”, NOSSDAV’14, March 19-21 2014, pages-103, ACM, 2014.

[15] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, T. Ishiguro, “Motion compensated inter-frame coding for video conferencing”, Pro. Nat. Telecommun. Conf., New Orleans, pp. G5.3.1-5.3.5, Nov. 1981.

[16] Zhibo Chen, Jianfeng Xu, Yun He, Junli Zheng, “Fastinteger-pel and fractional-pel motion estimation for H.264/AVC”, Journal of Visual Communication & Image Representation, April 2006, pp. 264-290, Elsevier, 2006.

[17] LI Hong-ye, LIU Ming-jun, ZHANG Zhi-qiang “A New Fast Motion Estimation Algorithm Based on H.264”, 2009 International Conference on Multimedia Information Networking and Security, pp- 287-290, IEEE, 2009.

★★★