Accelerating Color Space Conversion using Graphic Processing Units

Meng Shen[1, a], Shengdong Ruan[2, b]

[1] Zhejiang Longyou Power Supply Company, State Grid Corporation of China

[2]380 Longzhou Road, Longyou County, Zhejiang Province, 324400, China

[a]shenmengly@126.com, [b]ruanshengdong@126.com

Abstract

Color space conversion(CSC) is an important kernel in the area of image and video processing applications including video compression. CSC is a compute-intensive time-consuming operation that consumes up to 40% of processing time of a highly optimised decoder. Several hardware and software implementations for CSC have been found. Hardware implementations can achieve a higher performance compared to software-only solutions. However, the flexibility of software solutions is desirable for various functional requirements and faster time to market.Multicore processors, especially programmable GPUs, provide an opportunity to increase the performance of CSC by exploitingdata parallelism. In this paper, we present a novel approach for efficient implentation of color space conversion suitable for heterogeneous multicore GPUs. The proposed approach has been implemented and verified using computed unified device architecture(CUDA) on graphics hardware. The experiments results show that the speedup of up to 17× can been obtained. Speedup performance is relevant to GPU compute capability such as the number of scalar-processor, the block and grid sizes, and can be influenced by data transmission between CPU and GPU.

Keywords

Color space conversion, GPU, CUDA, optimization, multicore.

1. Introduction

Many video applications require converting video and image content from one color space to another. Image and video have utilized a wide variety of color spaces including: RGB, YCrCb, HSI, and other formats to represent the colors within the image. Each of these color space representations has its own set of advantages and disadvantages. Color space conversion is required when transferring data between devices that use different color space models. For example, RGB to YCbCr color space conversion is required when compressing video using JPEG and MPEG algorithms. Similarly YCbCr to RGB color space conversion is required when displaying movies on the computer monitor. In fact, color space conversion is a matrix operation which is uniquely defined for conversion from one color space to another, and this work can be seen as a case study of image processing on the GPUs.

Recently, multicore processors have gained much interest for accelerating scientific applications. There are two kinds of multicore called homogeneous and heterogeneous multicore [1]. The former includes only identical cores, e.g., multicore CPUs, and the latter, e.g., graphics processing units (GPUs) and Cell, uses a host processor for program control and coprocessors for computation. In particular, GPUs developed rapidly with the programmability and highly parallel processing feature. CPU-GPU heterogeneous system has become popular in scientific research. Parallel programming techniques can help programmers focus on algorithms instead of architectures. Some parallel programming models such as CUDA [2] and Brook+[3] can be used for NVIDIA and ATI GPUs,

respectively. Additionally, OpenCL[4] is a new industry standard for data-parallel and task-parallel heterogeneous computing on a variety of modern CPUs, GPUs, DSPs and other microprocessor designs.

In this paper, our goal is to explore the performance of heterogeneous multicores GPUs in the context of color space conversion application. Programmable graphics processing units can be adopted to assist the CPU in color space conversions between RGB and YCbCr. We evaluate the performance of the implementation of CSC on existing consumer graphics hardware using CUDA parallel programming model.

The rest of the paper is organized as follows. Section 2 reviews some conventional optimized methods for CSC. Section 3 presents parallel color space conversion on heterogeneous multicore GPUs using CUDA. Experimental results are given in Section 4. Finally, Section 5 gives the conclusions.

2. Background

There are several conventional methods including basic methods, look-up table methods, SIMD-based methods and hardware methods. We describe these methods as follows.

Basic methods, in which floating-point computations [5] and multiplication[6] can be removed by using fixed-point instead of floating-point and fixed-point shift and addition operations respectively.

Look-up table methods, in which two kinds of lookup table methods, i.e., once-check LUT and twice-check LUT, are designed. The difference is that the latter needs additional blocking, shift, and addition operations but with smaller memory occupancy.

SIMD-based methods, in which single instruction multiple data (SIMD) instructions exploit data level parallelism with multiple processing elements that perform the same operation on multiple data simultaneously. In fact, all contemporary processors support SIMD extensions, including desktop CPUs, digital signal processor s(DSPs) as well as some embedded processors such as ARM. For example, Asadollah [7] implemented CSC by using SSE instruction. Zheng [8] presented a method of CSC based on DSPs.

Hardware methods, in which higher performance can be archived compared to software-only solutions. Several application specific integrated circuits (ASICs) customized for CSC application can been found[9] and field programmable gate arrays(FPGAs) are programmable digital hardware that have been used in computational accelerators due to their flexibility and high performance[10].

3. Text Proposed Approach on Heterogeneous multicore GPUs

3.1 Fundamentals of GPUs

CUDA is developed by NVIDIA in order to offer writing GPU-accelerated software[2]. It provides a general-purpose data-parallel API for developing applications and performs SIMD processing so that it is suitable for applications that involve a large set of data without many complicated control branches. The function runs on GPU is called Kernel, in which the same instructions are executed on different data. Figure 1 represents the call flow between CPU and GPU. It also shows the parallelized distribution in GPU. Block is a unit executed in kernel function on GPU. Grid is a collection of blocks that are executed simultaneously. Each block is consisted of a series of threads that also run in parallel.

3.2 Implementation of CSC on CUDA-enabled GPUs

The detail process of color space conversion is shown in figure 1. In the host CPU, we should configure the environment for CUDA, allocate global memory at device GPU, set the size of block and grid, and copy data to device GPUs. And then, the kernel function is executed, which converts

color space concurrently by multiple threads on scalar processors of GPU. Finally, the result data are transferred to host CPU.

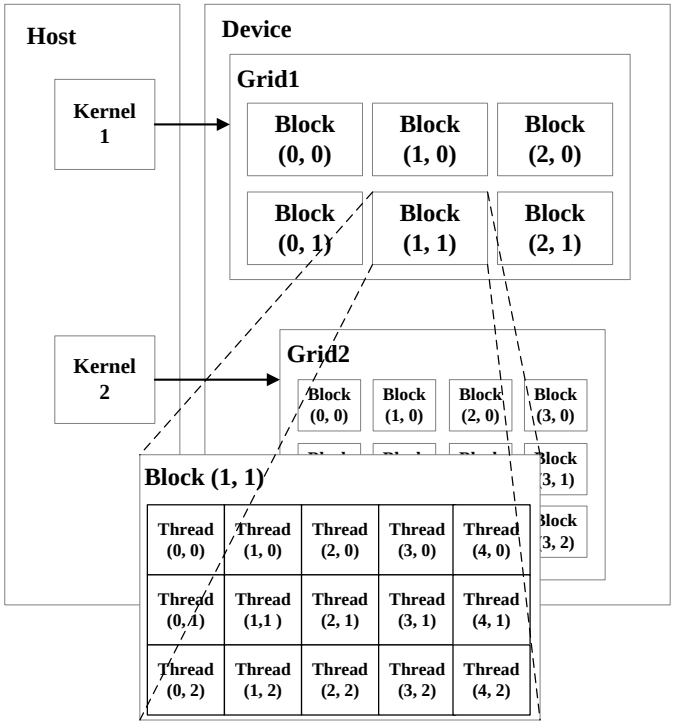

Fig. 1 Thread, block, and grid arrangement inside kernel functions on CUDA

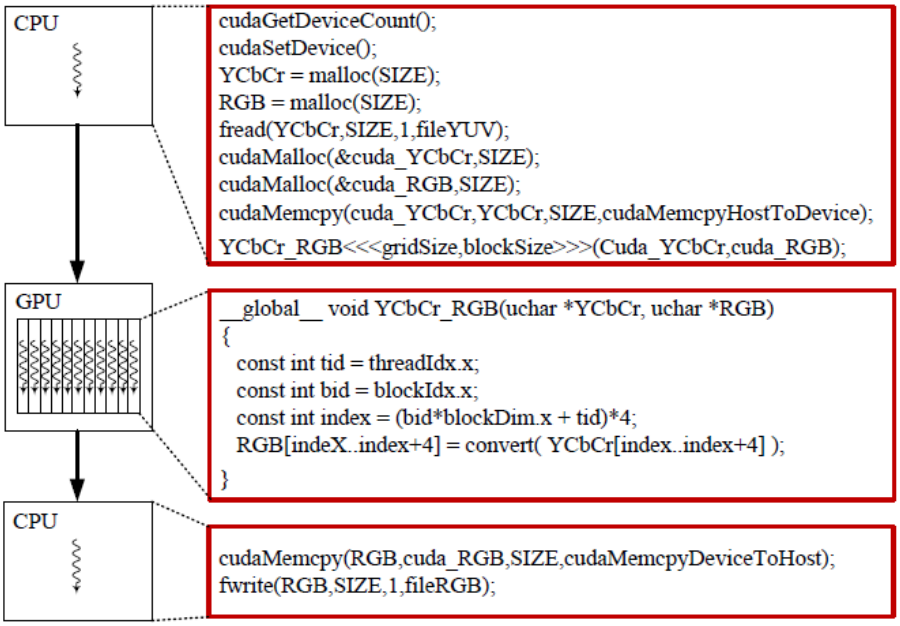

Fig. 2 Implementation of CSC on GPUs using CUDA

3.3 Optimization of CSC on CUDA-enabled GPUs

Note that we do some further optimizations in the implementation process. Vector data type such as uchar4 is adopted and on-chip register is used to accelerate the kernel reading input data. In the kernel function, divergent control flow in a warp is avoided and output data are buffered in shared memory. After permutation, if the sequences2 and itsNper-muted copies were stored contiguously one after another in the global memory, the intuitive memory lay-out would be as shown in Fig.3. Note that, we need one byte (uchar) to store each amino acid residue. Moreover, GPU can read four-byte (packed as

a CUDA built-in vector data typeuchar4) of data from the global memory to registers in one instruction. To achieve high parallelism of global memory access,uchar4 is used to store the permuted sequences. Dummy amino acid sym-bols are padded in the end to make the length of sequences a multiple of 4.

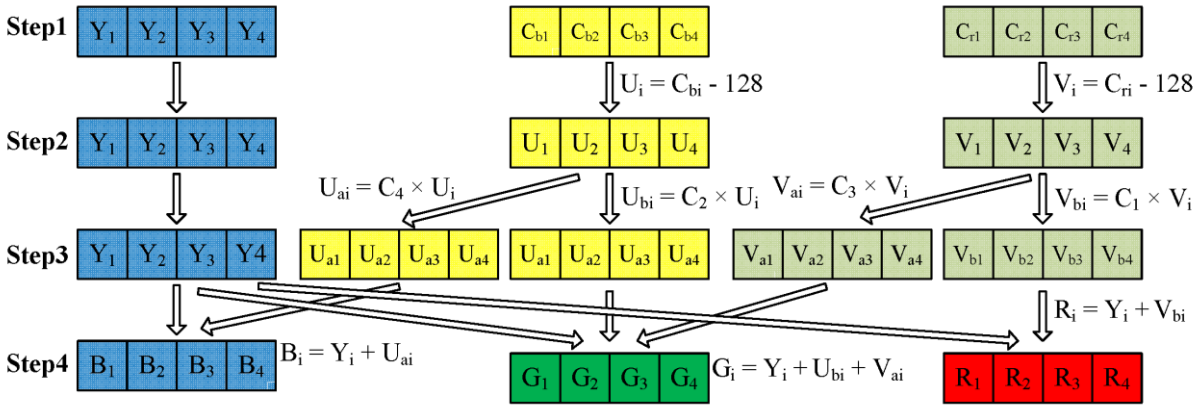

Fig. 3 optimization of CSC on GPUs using CUDA

4. Experimental results

4.1 Experimental environment

To evaluate the performance of the implemented color space conversion, the following experimental environment is used: (1) Intel Core i5-2310 2.9GHz processor with 4GB memory, (2) three kinds of GPUs, i.e. NVIDIA Geforce GTX460, 9800GT, 9500GT, (3) CUDA Toolkit and SDK 4.0, (4) Microsoft Windows 7, (5) Microsoft Visual Studio 2008. In addition, four kinds of video sequences with different resolutions are analyzed: D1, 720p, 1080p and 4096p.

4.2 Performance evaluation

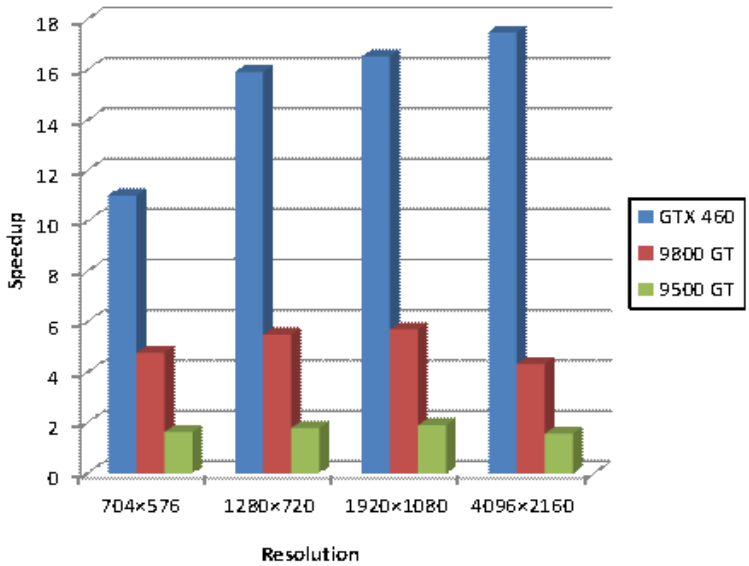

Fig. 4 Performance comparison with different resolutions on different GPUs

Figure 4 depicts the speedup performance of CSC with different resolutions on different GPUs. The results show that the GTX 460 outperforms 9800GT and 9500GT significantly. For instance, the speedups with resolution of 4096p are 17.48×, 4.33× and 1.57×, respectively. This is due to the different compute capability and architecture of GPUs. Specifically, the numbers of scalar-processor

of above motioned GPUs are 336, 112 and 32. Comparing these two sets of data, we can conclude that speedups with different GPUs are relevant to the numbers of built-in scalar-processor.

For the same GPU, speedups with different resolutions fluctuate irregularly. This is due to data transmission overhead between CPU and GPU. Data transmission time accounts for more than 60% of the total time, as the CSC is a relatively simple process. The memory bandwidth between CPU and GPU is limited, so the memory access is often the bottleneck for many applications on graphics cards. If read-backs from GPU memory to main memory are not avoided, it is necessary to offload enough parallel computations to GPU in order to hide such memory access latency.

5. Conclusion

In this paper, we compare the performance of heterogeneous multicore GPUs in the context of color space conversion application. We implement color space conversion on three kinds of multicore GPUs using CUDA respectively. Testing results show that GPUs can be adopted to accelerate image processing application such as color space conversions. The performance of GPUs is mainly affected by the number of compute units and the number of processing elements. Besides, data transmission

References

[1] H. Kim and R. Bond, Multicore software technologies, Signal Processing Magazine, IEEE, vol. 26 (2009) , No. 6, p.80-89

[2] NVIDIA, NVIDIA CUDA C Programming Guide Version 4.0, Nvidia Corporation, May 2010.

[3] AMD, HW Guide, Brook+ SDK 1.4, 2009.

[4] A.Munshi, The OpenCL Specification version 1.1, Khronos OpenCL Working Group, 2011.

[5] E. Dupuis. Optimizing YUV-RGB color space conversion using Intels SIMD technology. [Online]. Available: http://lestourtereaux.free.fr/papers/data/yuvrgb.pdf

[6] Y. Yang, P. Yuhua, and L. Zhaoguang, "A Fast Algorithm for YCbCr to RGB Conversion," IEEE Transactions on Consumer Electronics, vol. 53, no. 4, pp. 1490–1493, 2007.

[7] A. Shahbahrami, B. H. H. Juurlink, and S. Vassiliadis, Versatility of extended subwords and the matrix register file, ACM Transactions on Architecture and Code Optimization, vol. 5 (2008), p. 1-30.

[8] Z. Chun, Z. Yongjun, C. Xin, and G. Xiaoguang, "Research on technology of color space conversion based on dsp48e," in Proc. Third Int Measuring Technology and Mechatronics Automation (ICMTMA) Conf, vol. 3, 2011, pp. 87–90.

[9] R. Wang, G. Du, H. Li, X. Zhu, and W. Zhan, VLSI design of universal color conversion circuit, Experimental Eye Research, 2004.

[10] F. Bensaali and A. Amira, Accelerating colour space conversion on reconfigurable hardware," Image Vision Comput., vol. 23, (2005), pp 935-942. [Online]. Available: http:// dx.doi. org/10.1016/j.imavis.2005.03.006