GPU ACCELERATED HIGH-QUALITY VIDEO/IMAGE SUPER-RESOLUTION

Zhangzong Zhao[1], *Li Song*[1,2], *Rong Xie*[1,2], *Xiaokang Yang*[1,2]

[1] Institute of Image Communication and Network Engineering, Shanghai Jiao Tong University
[2] Cooperative Medianet Innovation Center, Shanghai China
dragon_nsc2@hotmail.com,{song_li,xierong,xkyang}@sjtu.edu.cn

***Abstract*—This paper presents several novel GPU optimization technologies to accelerate the SRCNN(Super-Resolution Convolutional Neural Network) – one of the best super-resolution algorithm. We first directly parallelize and implement the SRCNN, then accelerate the convolution by making use of the hierarchical feature of GPU memory. We explore different optimization methods on each convolution and select the fastest combination. Further acceleration can be achieved by fusing the convolution and ReLu(Rectified Linear units) operation to eliminate the memory access time of ReLu. Our experiments show that the overall execution time for 1080p to 4K upscaling is reduced from 300s/frame to 0.15s/frame, while the image quality is exactly the same as original SRCNN.**

***Index Terms*—Content adaptation and scaling, Video Processing, GPU, Super resolution**

I. INTRODUCTION

Super-resolution (SR) is one of key technologies for both video non-linear editing and video post-processing application. There are two basic requirements for the video super-resolution technique, speed and quality. Early upscaling methods (e.g. bicubic, Lanczos) typically have low complexity and computational cost, so that they can be implemented on chip and execute in real time. However, the quality of their upscaled images is relatively poor, with visual artifacts such as ringing, aliasing, blurring and blocking. The poor quality of these methods can hardly meet the requirement for high quality video super-resolution. Most recent state-of-the-art SR methods[1-8] are example-based solution that either use the internal similarities of input image or learn mapping functions from external low and high resolution data base. Generally these methods are very time-consuming which limit their usage. On the other hand, existing GPU based acceleration methods achieve significant speed-up at the cost of degraded quality [9-12]. They either depend on relatively poor quality SR methods, or suffer from quality reduction caused by the acceleration.

In this paper, we propose several novel GPU optimization technologies to accelerate the SRCNN[1] (Super-Resolution Convolutional Neural Network) – one of best SR methods. We first parallelize and implement the convolution. The execution time is reduced from 300s/frame to 1s/frame. Then we make use of the hierarchical feature of GPU memory, preload the filter parameters and the input patches to shared memory or registers, to speed up the convolution 2 to 20 times. Further acceleration can be achieved by fusing the convolution and ReLu(Rectified Linear units) operation, to eliminate the memory access time of ReLu. Our experiments show that the overall execution time for 1080p to 4K upscaling is reduced from 300s/frame to 0.15s/frame, while the image quality is exactly the same as original SRCNN.

The rest of this paper is organized as follows. Section II introduces the GPU architecture. Section III introduces the accelerated algorithm SRCNN, the direct parallelism and GPU implementation, the optimization of convolution, and the fusion of convolution and ReLu. In section IV, the experiments for execution time are presented, and the results are discussed.

II. GPU ARCHITECTURE

In this section, we briefly introduce the architecture of GPU device and the hierarchical feature of GPU memory to provide information about our implementation and optimization[13]. The latest GPU has thousands of stream processors and tremendous global memory bandwidth (eg. Nvidia GTX 980TI has 2816 CUDA cores and 336GB/s global memory bandwidth). When handling a CUDA program with millions of threads, GPU will schedule the stream processors onto these threads. Each thread will be distributed to and executed by one stream processor. The CUDA program is concurrently executed by these stream processors so that it is much faster than the CPU program.

The GPU has a hierarchical memory architecture. We introduce and utilize the global memory, the shared memory and the register. These memories have very different access bandwidth and latency, as shown in table I. First, global memory is the largest memory on GPU, and can be accessed by every thread. However, the global memory is the slowest memory and often become the bottleneck of entire program. Second, the shared memory is like cache. In CUDA program, the application-threads are subdivided into blocks, and each block is subdivided into threads. All threads within a block can

978-1-4673-9044-6/16/$31.00 ©2016 IEEE

access the block's shared memory. The bandwidth and latency of shared memory is much larger and faster than global memory. Third, the register is the fastest but smallest memory, and can only be access by the corresponding thread. If the reused data can be stored in the register, the memory access would be much faster.

TABLE I. BANDWIDTH AND ACCESS TIME BY MEMORY TYPE

Storage	Bandwidth	Latency
Global Memory	~130MB/s/thread	400~600 cycles
Shared Memory	~1.5TB/s/thread	1~32 cycles
Registers	~8TB/s/thread	1 cycle

III. PROPOSED METHOD

A. The SRCNN

The prototype of our super-resolution algorithm is the SRCNN[1], which is a technique that not only produces good image quality, but also requires a comparable low computation complexity. The SRCNN is inspired by the sparse coding based super-resolution methods. It achieves almost the same quality as the current state-of-the-art SR methods, and it separates the phase of learning and running, making the running speed much faster than other SR methods. Figure 1 shows the process of SRCNN. It runs without solving any optimization problem or searching the data base, instead it is fully feed-forward convolutional neural network.

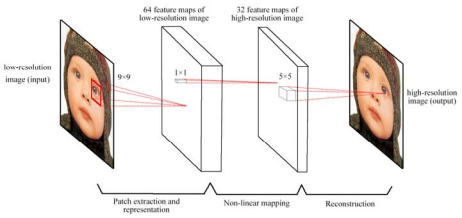

Figure 1. The process of SRCNN

The biggest challenge is the running speed requirement for 4K video processing. The original SRCNN costs 300 second per frame for 1080 to 4K SR using CPU/matlab. The computing complexity is extremely large. To process the entire SRCNN on one 4K image, the floating point multiply-add operations required is 66.6G and the memory I/O is 800GB. It's obviously impossible to be done in a few seconds by CPU.

B. Direct GPU Implementation of convolution

To begin with, we focus on the GPU implementation and optimization of convolution, because the SRCNN consists of 3 layer-wise convolutions and 2 layer-wise RELUs operation, and over 95% of execution time is spent on convolution operation. We split the convolution task into millions of mini-task (along width and height of the convolution output), as shown in figure 2. Each convolution output pixel is computed independent, parallel, and occupies a unique GPU thread. The rest part of SRCNN is also parallelized and GPU implemented so that the entire SRCNN executes on GPU and do not require several rounds of CPU/GPU data transfer. The overall execution time decreased from 300s/frame to 1s/frame by means of GPU.

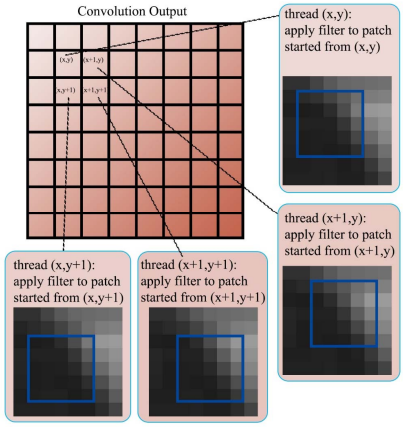

Figure 2. Direct GPU Implementation of convolution

C. GPU Optimization of Convolution

An execution time of 1 seconds per frame is not satisfactory for our video super-resolution. We manage to further accelerate it by making use of the hierarchical feature of GPU memory. The GPU has three types of hierarchical memory: the largest but slowest global memory, the smaller and faster shared memory, the smallest and fastest local register memory. In the direct GPU implementation of convolution, each thread reads the filter parameters and input image data from the global memory. However, the filter parameters and image data are redundantly read and will waste huge global memory bandwidth.

By making use of the hierarchical feature of GPU memory, we preload the filter parameters and the image patches to shared memory or registers, speeding up the convolution 2-20 times.

In *Shared Kernel* method, the filter parameters are preloaded to shared memory to save global memory I/O, as shown in figure 3. In the direct GPU implementation of convolution, exactly same filter parameters are read from global memory by every thread. The global memory I/O is wasted on these redundant read operations. In our *shared Kernel* method, first the filter parameters are preloaded to shared memory of every block, then each thread read the filter parameters from shared memory to execute the convolution. The global memory I/O requirement for filter parameter access is significantly reduced.

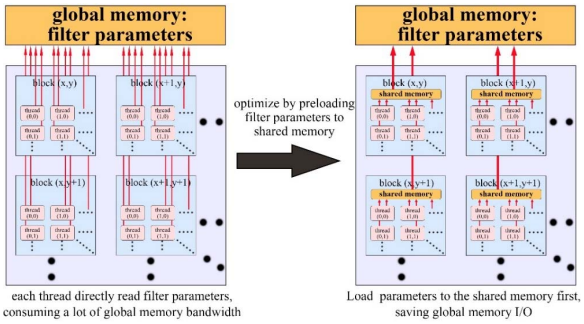

Figure 3. Shared Kernel

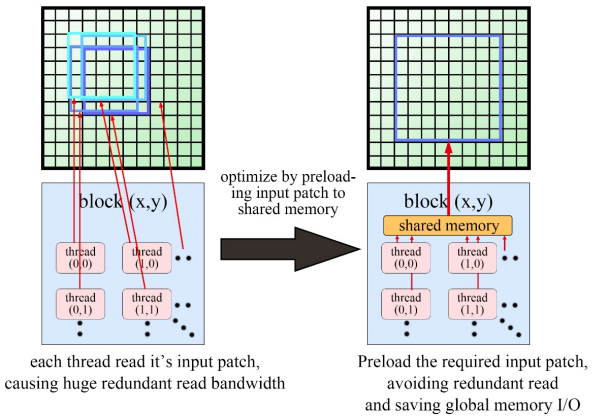

Figure 4. Shared Patch

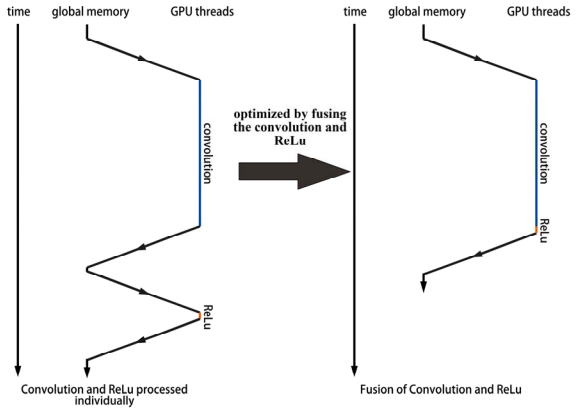

Figure 5. Fusion of Convolution and ReLu

In *Shared Patch* method, the input patches are preloaded to shared memory, avoiding redundant image read operations to save global memory I/O, as shown in figure 4. When dealing with convolution of height or width size more than 1, the adjacent output pixels rely on an overlapping area of input patch. In the direct implementation of convolution, the overlapping is not considered and input patches are redundantly read and waste global memory I/O. The waste become significantly worse if the convolution height and width are large. In our *Shared Patch* method, we firstly figure out the input patch that all threads within one block rely on, and preload that input patch to the shared memory. Then each thread read the required input small patch from shared memory to execute the convolution. The global memory I/O requirement for input image data access is significantly reduced.

In *Shared Kernel and Patch* method, we combine the advantages of *Shared Kernel* and *Shared Patch* methods to get further acceleration.

In *Shared Kernel and Registered Pixel* method, the input patches are loaded into registers to achieve the fastest image data access and reduce the global memory access as much as possible. However, this method requires quite large register size, so it only works when the convolution size is small.

D. Fusion of Convolution and ReLu Operation

Ordinary acceleration for convolution neural network focus on convolutions, because convolutions are the bottleneck of execution time and nonlinear layer can hardly get more acceleration. However, as we have accelerated convolution so far, the ReLu layer's execution time can't be ignored.

We propose fusing the execution of convolution and RELU, as shown in figure 5. In convolution neural network, the nonlinear layer always follows the convolution layer, and the nonlinear operation output pixel only relates to corresponding input pixel. So we combine the convolution and ReLu operation into a single procedure. In each thread, the convolution is processed, followed by a ReLu operation. This eliminates the write to global memory of convolution layer, and the subsequent read from global memory of ReLu layer. The ReLu's memory access time can be totally eliminated.

IV. Experiments

In order to accelerate the super-resolution algorithm as much as possible, we optimize each convolution layer separately by testing each optimization method on that convolution and select the fastest. The SJTU 4K video sequences[14] are used to test the performance and speed. The experiments are done on a PC with dual Intel E5-2697V2 @2.7GHz 12 cores processers and Nivida GTX 980TI.

As our proposed solution are pure GPU platform acceleration scheme, there are no performance loss compared to the original SRCNN implementation. To save space, we do not include these results here. Furthermore, our GPU acceleration schemes are independent on video content, thus lead to a consistent speed-up gain across different video sequences at same resolution. The execution time experiments are summarized in table II.

Table II execution time of each method on each convolution

Method	Conv1	Conv2	Conv3
CPU	13064ms	306622ms	5650ms
Direct	453ms	475ms	86ms
Shared Kernel	375ms	396ms	56ms
Shared Kernel And Patch	139ms	1602ms	**41ms**
Shared Kernel And Registered Pixel	Unable	**24ms**	Unable
cuDNN	**56ms**	37ms	150ms

For the first convolution layer of size 64*9*9*1 (output channel*width*height*input channel), the best method is *cuDNN* from Nvidia CUDA SDK [13] at the speed of 0.056 seconds, which is 230 times faster than CPU. For the second convolution layer of size 32*1*1*64, the best method is *Shared Kernel And Registered Pixel* at the speed of 0.024 seconds, which is 12800 times faster than CPU. For the third convolution layer of size 1*5*5*32, the best method is *Shared Kernel And Patch* at the speed of 0.041 seconds, which is 137 times faster than CPU. The overall computing time of SRCNN is accelerated to 0.15 seconds, which is 2000 times faster than CPU.

By analyzing the execution speed experiment for each layer, we have observed that best solution to get the maximum rate of acceleration come from an optimized combination by applying different schemes to different convolution layers. This is reasonable because each convolution layer has a specific size of convolution, and would have a specific feature for a series of *filter loading*, *data inputting*, *computing* and *data outputting* operation. When dealing with that specific size of convolution, A suitable combination version would be better than others.

V. Conclusion

By the GPU implementation and optimization of the high-quality super-resolution algorithm, we succeed to form the video super-resolution framework, which increase the video resolution, gets a state-of-the-art visual quality and execute at an acceptable speed.

VI. Acknowledgment

This work was supported by NSFC (61527804,61521062), the 111 Project (B07022 and Sheitc No.150633) and the Shanghai Key Laboratory of Digital Media Processing and Transmissions.

Reference

[1] C. Dong, C. Loy, K. He, X. Tang, "Learning a Deep Convolutional Network for Image Super-Resolution," in *Proceedings of European Conference on Computer Vision*, pp.184-199, Sep. 2014

[2] J. Yang, J. Wright, T. Huang, Y. Ma, "Image Super-Resolution Via Sparse Representation," *IEEE Transactions on Image Processing*, vol.19, no.11, pp.2861-2873, Nov. 2010

[3] J. Yang, Z. Lin, S. Cohen, "Fast Image Super-Resolution Based on In-Place Example Regression," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp.1059-1066, Jun. 2013

[4] D. Glasner, S. Bagon, M. Irani, "Super-resolution from a single image," in *Proceedings of IEEE International Conference on Computer Vision*, pp.349-356, Sept 2009.

[5] C. Yang, M. Yang, "Fast Direct Super-Resolution by Simple Functions," in *Proceedings of IEEE International Conference on Computer Vision*, pp.561-568, Dec. 2013

[6] R. Timofte, V. D. Smet, L. V. Gool, "Anchored Neighborhood Regression for Fast Example-Based Super-Resolution," in *Proceedings of IEEE International Conference on Computer Vision*, pp.1920-1927, Dec. 2013

[7] R. Timofte, V.D. Smet, L. V. Gool, "A+: Adjusted Anchored Neighborhood Regression for Fast Super-Resolution," in *Proceedings of* A*sian Conference on Computer Vision*, pp.1-15, Nov. 2014

[8] H. Chang, D. Yeung, Y. Xiong, "Super-resolution through neighbor embedding," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, vol.1, June 2004

[9] K. Wang, L. Wang, J. Lu, Y. Sun, S. Zhao, "A GPU-based implementation on super-resolution reconstruction," in *Proceedings of IEEE International Conference on Image Processing (ICIP)*, pp.849-852, Sep. 2012

[10] E. Perez-Pellitero, J. Salvador, J. Ruiz-Hidalgo, B. Rosenhahn, "Accelerating Super-Resolution for 4K upscaling," in *Proceedings of IEEE International Conference on Consumer Electronics* , pp.317-320, Jan. 2015

[11] Y. Shen, X. Wu, X. Deng, "GPU-aided real-time image/video super resolution based on error feedback," in *Proceedings of IEEE Visual Communications and Image Processing Conference*, pp.286-290, Dec. 2014

[12] J. Hu, H. Li, Y. Li, "Real time super resolution reconstruction for video stream based on GPU," in *Proceedings of IEEE International Conference on Orange Technologies*, pp.9-12, Sep. 2014

[13] NVIDIA CUDA Toolkit Document Programming Guides Version 7.0, http://developer.nvidia.com/cuda-downloads

[14] L. Song, X. Tang, W. Zhang, X. Yang, P. Xia, "The SJTU 4K video sequence dataset," in *Proceedings of International Workshop on Quality of Multimedia Experience*, pp.1-2, Jul. 2013.