GPU implementation and optimization of Video Super-Resolution

ZhangZong Zhao[1], Li Song[1], Bo Xiao[2]

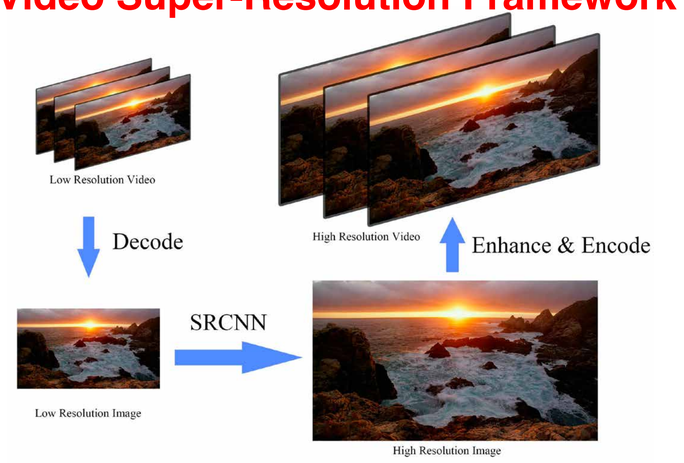
[1] School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China

[2] Baidu Research Silicon Valley AI Lab

Abstract

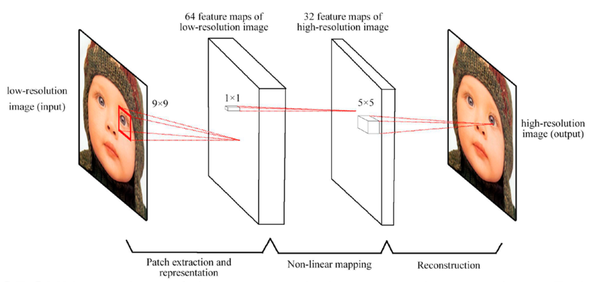
Video Super-Resolution is to increase the resolution of a video. We managed to apply the SRCNN[1] on FHD to 4K super-resolution, which has a very good performance among the current start-of-the-art SR methods. However the SRCNN runs very slow on CPU(300s per frame), and it's not acceptable for practical application. So we parallelize the convolution, implement the SRCNN on GPU and optimize it. The GPU and the optimization significantly accelerates the video super-resolution process and achieve the speed of 0.16s per frame, which is almost 1800 times faster than the CPU version.

Video Super-Resolution Framework



The Super-Resolution Convolutional Neural Networks(SRCNN)

Chao Dong el al.[1] introduced the SRCNN.



The SRCNN consists of three convolution operations and two ReLU operations. We choose SRCNN as our image super-resolution algorithm for three reasons:

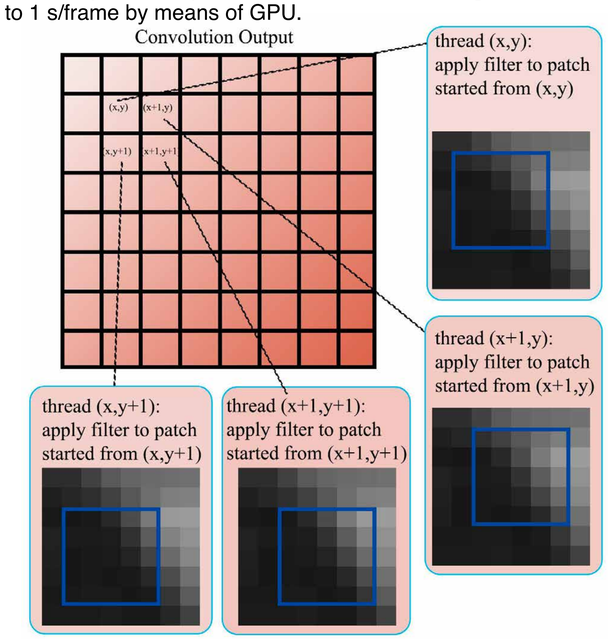
1. SRCNN has a very good performance among the current state-of-the-art SR methods
2. The computation complexity is relatively small compared to other learning based SR techniques.
3. The convolution, which is the core of SRCNN, is parallel processable.

GPU implementation of convolution

Over 95% of the running time is spent on convolution operation. The computing complexity for convolution is extremely large. The required floating point operations of a convolution = height × width ×outputChannel × inputChannel × filterHeight × filterWidth.

To process the entire SRCNN on 4K image, the floating multiply-add operations required is 66.6G, and the memory I/O size is over 800GB.

It's obviously impossible for CPU to handle it within seconds. There comes the GPU implementation of convolution. Each convolution output pixel is computed independent, parallel and occupies a unique thread. The overall running time decreased from 300 s/frame to 1 s/frame by means of GPU.

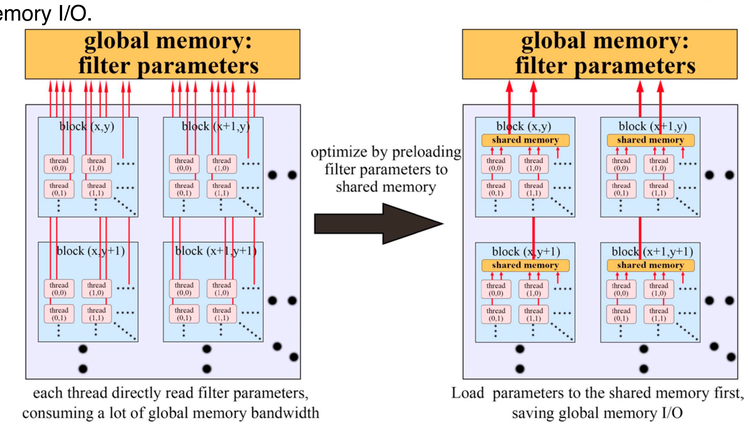


GPU optimization of convolution

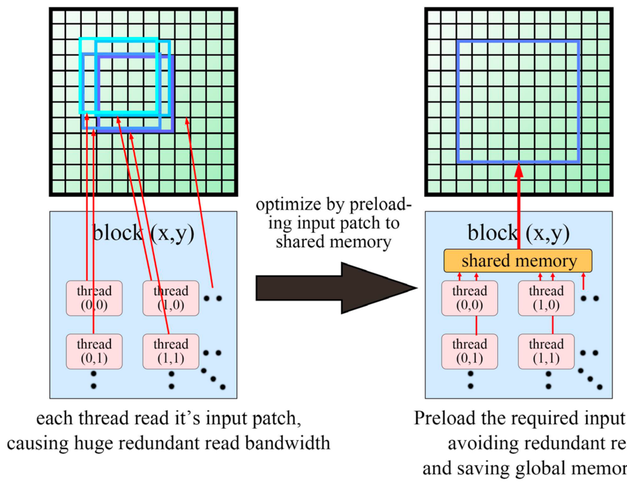
In the direct GPU implement of convolution, every thread reads the filter parameters and input image data from the global memory. The filter parameters and image data are redundantly read and will cost huge global memory bandwidth.

By making use of the hierarchical memory of GPU, we preload the filter parameters and image patches to shared memory or registers, speeding up the convolution 2-10 times.

Shared Kernel: Preload filter parameters to shared memory of blocks, saving a lot of global memory I/O.



Shared Patch: Preload input patch to shared memory of blocks, avoiding redundant image read operations and saving global memory I/O.



Registered Pixel: Preload input pixels to registers, to get maximum rate of speed up. This method only works when convolution size is small.

Other attempts of optimization: We implemented and tested other kinds of optimizations, including Caffe's GEMM, cuDNN. These optimizations have very different running time when processing different size of convolution.

Experimental Result

We test each optimization algorithm on each convolution to select the best optimization for each convolution. The experiment shows the GPU optimization has a speed up factor of 1800 than the CPU implementation.

Operation	Filter size (output channel *width*height* input channel)	Best algorithm	Computing time	Speed up (compared to CPU/matlab)
Convolution 1	64*9*9*1	cuDNN	0.056s	230
Convolution 2	32*1*1*64	Shared Kernel and Registered Pixel	0.024s	12800
Convolution 3	1*5*5*32	Shared Kernel and Patch	0.041s	137
SRCNN			0.160s	1800

Experimental environment
CPU: dual E5 2697V2
GPU: GTX980TI
video size: 1920*1080 -> 3840*2160

References

[1] C. Dong, C. Loy, K. He, X. Tang, "Learning a Deep Convolutional Network for Image Super-Resolution", Proceedings of European Conference on Computer Vision(ECCV), 2014.