BILATERAL FILTER IN CUDA

CSE 708: Programming Massively Parallel Systems

Guide: Dr. Russ Miller

Presenter: Gaurav Nathani

University at Buffalo The State University of New York

Filters

- All digital images have noise and filters are used to reduce the noise.
- Different type of noise requires different filter.
- Types of Noise: Gaussian, Salt & Pepper, Poisson, Speckle.
- Types of Filters: Mean, Median, Box, Bilateral, Gaussian, Fourier Transform, Wavelet Transform.
- Focus on Bilateral Filters which is used to reduce Gaussian noise.

Bilateral Filter

- Gaussian filter considering neighboring pixel intensity is Bilateral Filter.
- `BF(x,y) = g(x,y) * g(Ix-Iy)`

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)}$$

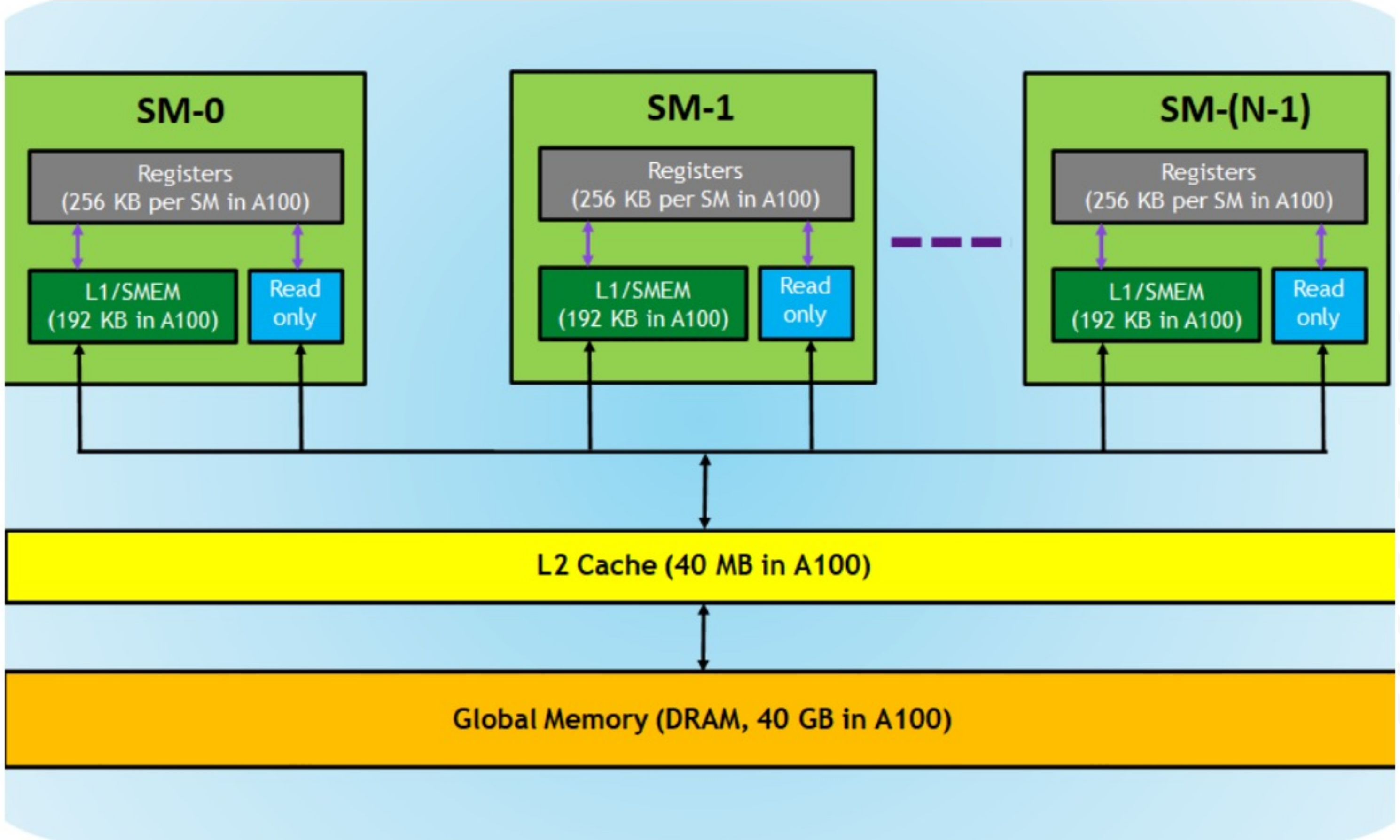
$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/(2\sigma^2)}$$

- Ix & Iy are intensities of pixels at x & y position on the image grid.
- Both sigma values in above equation are different and are parameters of the bilateral filter function which determine the amount of noise reduction in the image.

GPUs

- GPGPUs pack many “simple” cores (shared processors/SPs) in several multiprocessors (shared multiprocessors/SMs) with no-cost of context switching.
- Each SP has its local registers (pixel intensity at position), several SPs within SM have shared memory (submatrix), SMs share device global memory (full image matrix loaded in global memory). Similar hierarchy for cache as well.
- Constant memory used for parameters that don’t change throughout execution (sigma values & pi).
- Texture memory used to prevent bank conflicts. It has associated texture cache used as well.
- User defines CUDA kernel that copies memory from host to device, launches threads on all SMs concurrently.
- Threads grouped (logically) into blocks and blocks grouped into grid by the user's kernel.
- GPUs group threads in warps for execution.

GPU Architecture



NVIDIA GeForce GTX 1660Ti – TU116-400-1A

SMs	24
CUDA Cores	1536
GPU Boost Clock	1770 MHz
FLOPS	11 TFLOPS
Total Amount of Global Memory	14259 MB
Shared Memory	8115 MB
L1 Cache	1536 KB
CUDA	7.5

Parallelizing Bilateral Filter in CUDA

- Create a kernel function to execute bilateral filter with parameters
  - d : Diameter of Pixels Neighborhood
  - Sigma Color : Determines mixing of colors
  - Sigma Space : Determines mixing of far apart pixels
- 2D decomposition of matrix – large submatrix data copied to shared memory of SMs, part of submatrix handled by SPs by moving data from shared memory to its local registers.

Methodology

1. Read a large image in greyscale using OpenCV.
2. Add Gaussian noise to the image.
3. Apply OpenCV's bilateral filter on CPU and time the run.
4. Run the parallel bilateral filter on GPU and time the run.
5. Resize the image (downscale from original) and repeat.

* For parallel algorithm – kept the block size same as 64*64.

Image Processing – Adding Gaussian Noise

Original Greyscale Image

Image with Gaussian Noise Added

Image Processing – Over/Under Filtering

Under-filtered Image

Over-filtered Image

Image Processing – Adequate Filtering

Image with Gaussian Noise

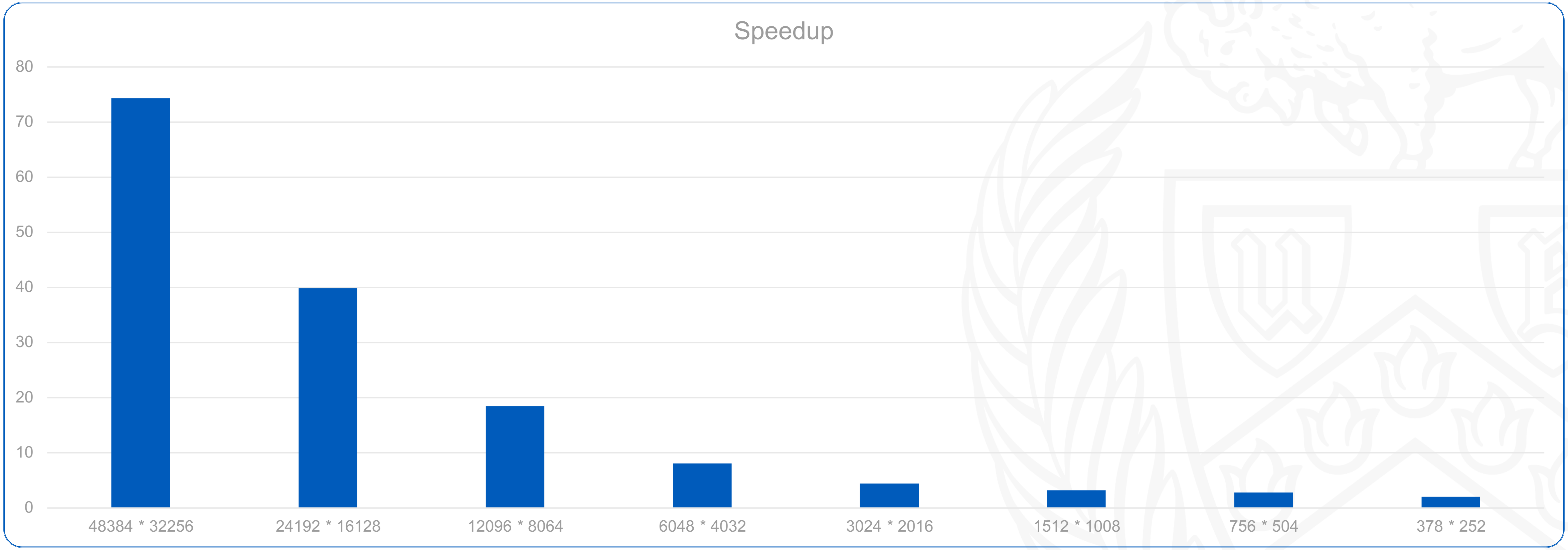
Bilateral Filtered Image with apt Sigma Values

Results

Width	Height	Time (CPU)	Time (GPU)	Speedup
48384	32256	37636.8	506.13492	74.3612
24192	16128	2423.09	60.800693	39.853
12096	8064	586.018	31.784889	18.437
6048	4032	162.748	20.174538	8.067
3024	2016	37.6955	8.4861549	4.442
1512	1008	11.1166	3.4416718	3.23
756	504	3.0405	1.078574	2.819
378	252	2.475	1.1921965	2.076

** Block Size = 64 * 64*

Speedup


Speedup
80
70
60
50
40
30
20
10
0
48384 * 32256
24192 * 16128
12096 * 8064
6048 * 4032
3024 * 2016
1512 * 1008
756 * 504
378 * 252

Profiler Observations

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	99.99%	69.3475s	1	69.3475s	69.3475s	69.3475s	process_bilateral_filter(…)
	0.00%	2.2434ms	4	560.85us	1.1200us	2.2400ms	[CUDA memcpy HtoD]
	0.00%	2.1547ms	1	2.1547ms	2.1547ms	2.1547ms	[CUDA memcpy DtoH]
API calls:	99.74%	69.3526s	2	34.6763s	2.5085ms	69.3501s	cudaMemcpy
	0.26%	179.43ms	2	89.716ms	220.15us	179.21ms	cudaMalloc
	0.00%	423.53us	2	211.77us	99.944us	323.59us	cudaFree
	0.00%	304.88us	1	304.88us	304.88us	304.88us	cuDeviceTotalMem
	0.00%	181.13us	101	1.7930us	117ns	79.053us	cuDeviceGetAttribute
	0.00%	71.666us	1	71.666us	71.666us	71.666us	cudaLaunchKernel
	0.00%	62.397us	3	20.799us	8.6280us	44.373us	cudaMemcpyToSymbol
	0.00%	27.370us	1	27.370us	27.370us	27.370us	cuDeviceGetName
	0.00%	8.6890us	1	8.6890us	8.6890us	8.6890us	cuDeviceGetPCIBusId
	0.00%	3.1250us	2	1.5620us	156ns	2.9690us	cuDeviceGet
	0.00%	1.0600us	3	353ns	164ns	700ns	cuDeviceGetCount
	0.00%	220ns	1	220ns	220ns	220ns	cuDeviceGetUuid

** Profiling for one of the runs of 24192 x 16128 with block size of 64*64*

Future Work

- Can we use texture memory in a better way?
- Compare runtime with varying block sizes; expected larger block size suitable for larger inputs.
- Run the bilateral filter on RGB scale images – handle 3 different matrices.
- Try implementing other filters to compare results.

References

1. https://www.geeksforgeeks.org/python-bilateral-filtering/
2. https://www.tutorialspoint.com/opencv/opencv_bilateral_filter.htm
3. https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html
4. https://en.wikipedia.org/wiki/Bilateral_filter
5. https://github.com/aashikgowda/Bilateral-Filter-CUDA

Questions?

Thank you!

THANK YOU!