Implementation of Parallel Softmax Regression using CUDA

CSE 633 Parallel Algorithms (Spring 2018)

Shivraj Wabale
Instructor : Dr. Russ Miller

University at Buffalo, State University of New York

shivrajd@buffalo.edu

May 1, 2018

Overview

Image classification

- Classification of the handwritten digits from 0 to 9
- MNIST database : A large database of handwritten digits (images and labels)
- Softmax Regression on 10 classes
- Backpropagation : an algorithm for supervised learning to modify weights according to the errors.

Data Set : MNIST

- MNIST database of handwritten digits
- Training set : 60,000 (images and labels)
- Test set of : 10,000 (images and labels)
- Each image of size 28 * 28 (784) bytes

Softmax regression

- Multi-class Logistic Regression

$$P(y = j|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}} \tag{1}$$

Here K = 10.

Achieved up-to 86% accuracy.

GPU

- Though GPUs inherently support multi-threading, there are several hardware constraints
- GPU in CCR
  - Name: Tesla M2050
  - CUDA Version: 2.0
  - Shared memory per block: 49152
  - Total constant memory: 65536
  - Regs per block: 32768
  - Max threads per block: 1024
  - Max threads per dim: 1024,1024,64
  - Max grid size: 65535,65535,65535
  - Multi processor count: 14

Work Flow

CPU

1. Load images and labels
2. Allocate the space and transfer images to GPU
3. Launching a kernel with a grid of thread blocks
4.
5. Launching a kernel to find the sum
6.
7. Copy the summation from device to host
8. Compute the Softmax
9. Find the gradient from error
10.

GPU

4. Multiply images with weights
6. Run the array summation
10. Updates the weights

Parallel Reduction

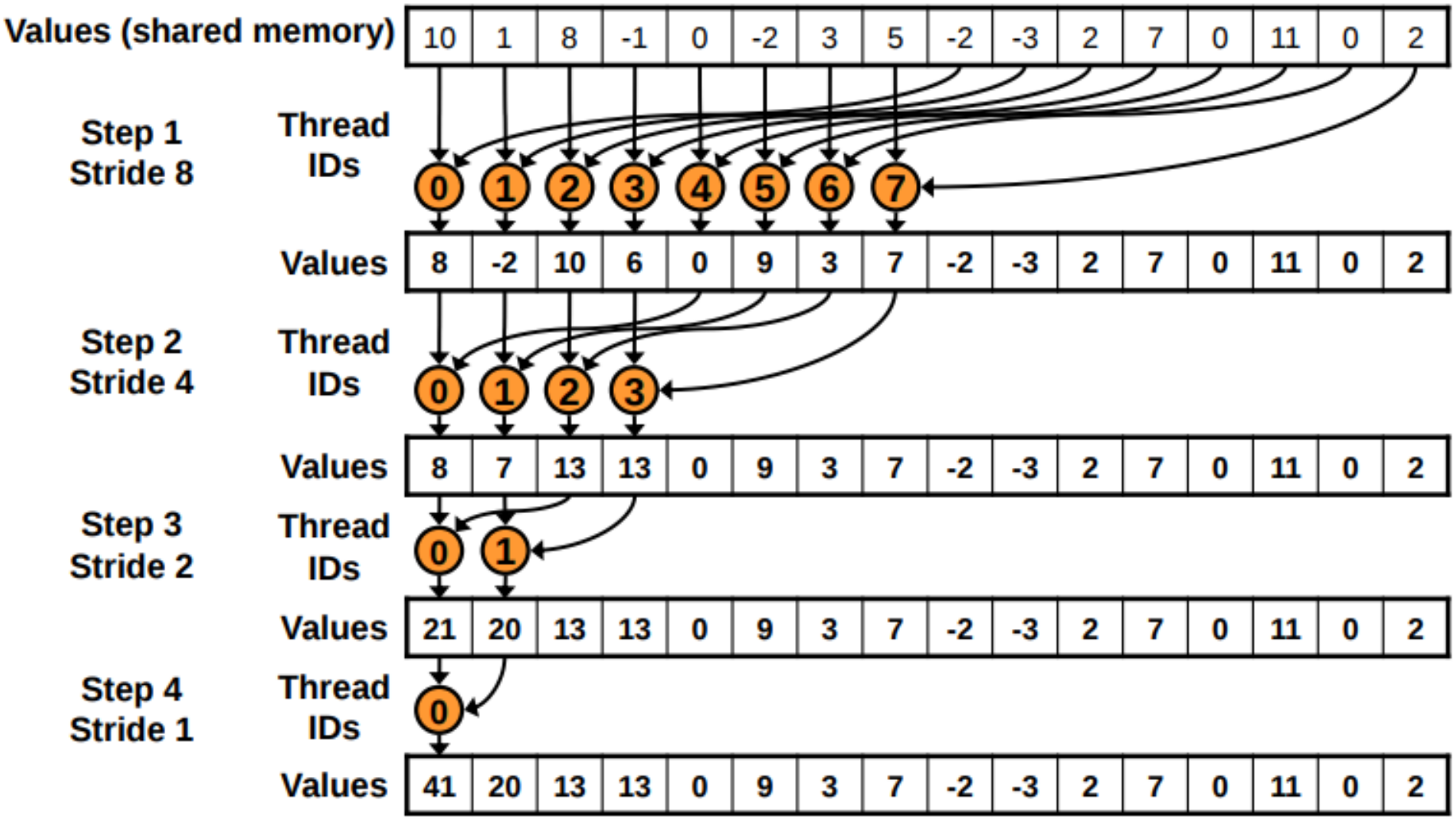

Figure: Parallel Reduction

Parallel Reduction

CCR

GPU	CPU
4.60E-06	3.80E-05
3.80E-06	4.20E-05
1.00E-05	3.60E-05
6.40E-06	5.00E-05
7.40E-06	2.80E-05
5.20E-06	4.00E-05
7.60E-06	4.00E-05

Local

GPU	CPU
4.32E-06	2.75E-05
4.32E-06	2.74E-05
4.34E-06	2.80E-05
4.36E-06	2.89E-05
4.50E-06	2.84E-05
4.30E-06	2.74E-05
4.58E-06	2.79E-05

Parallel Reduction

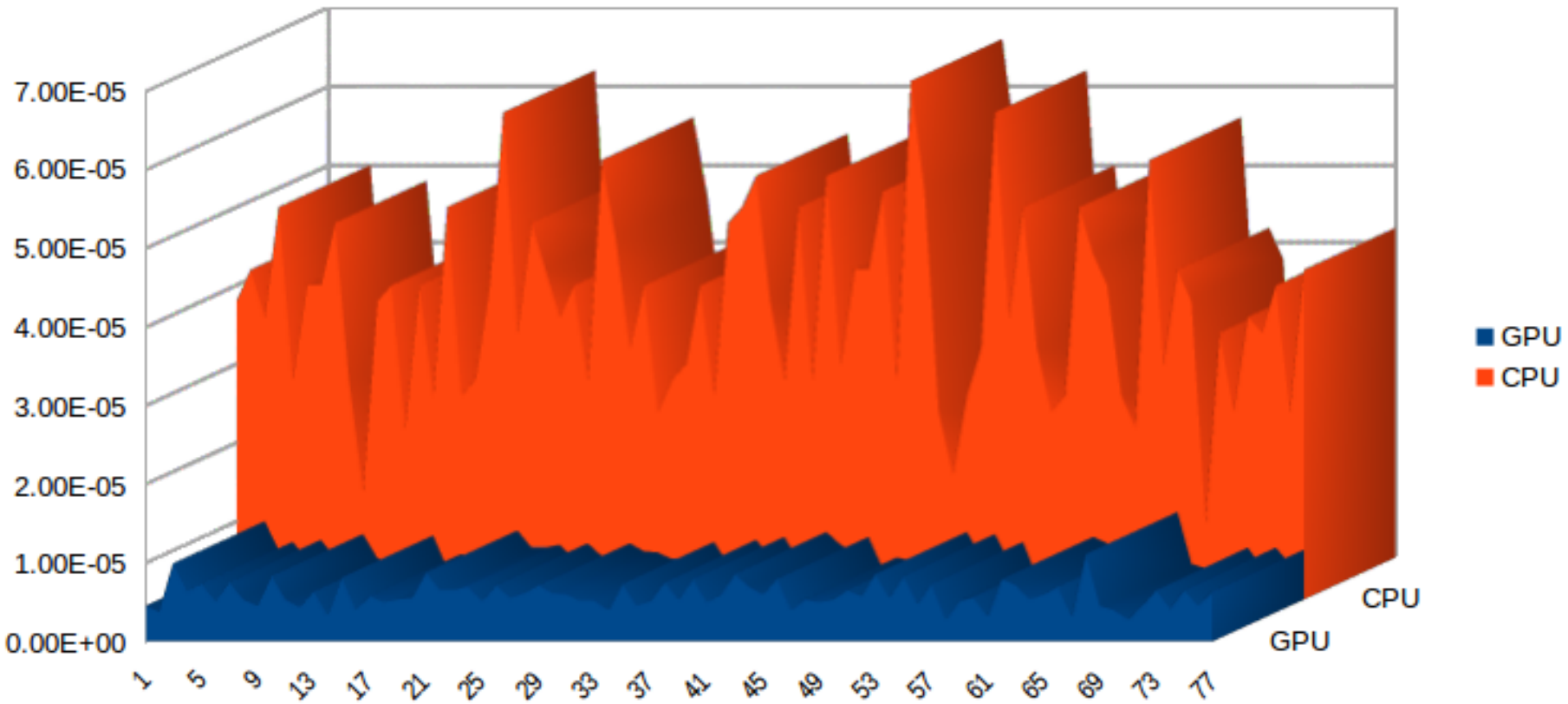

Figure: Serial execution vs parallel reduction CCR

Parallel Reduction

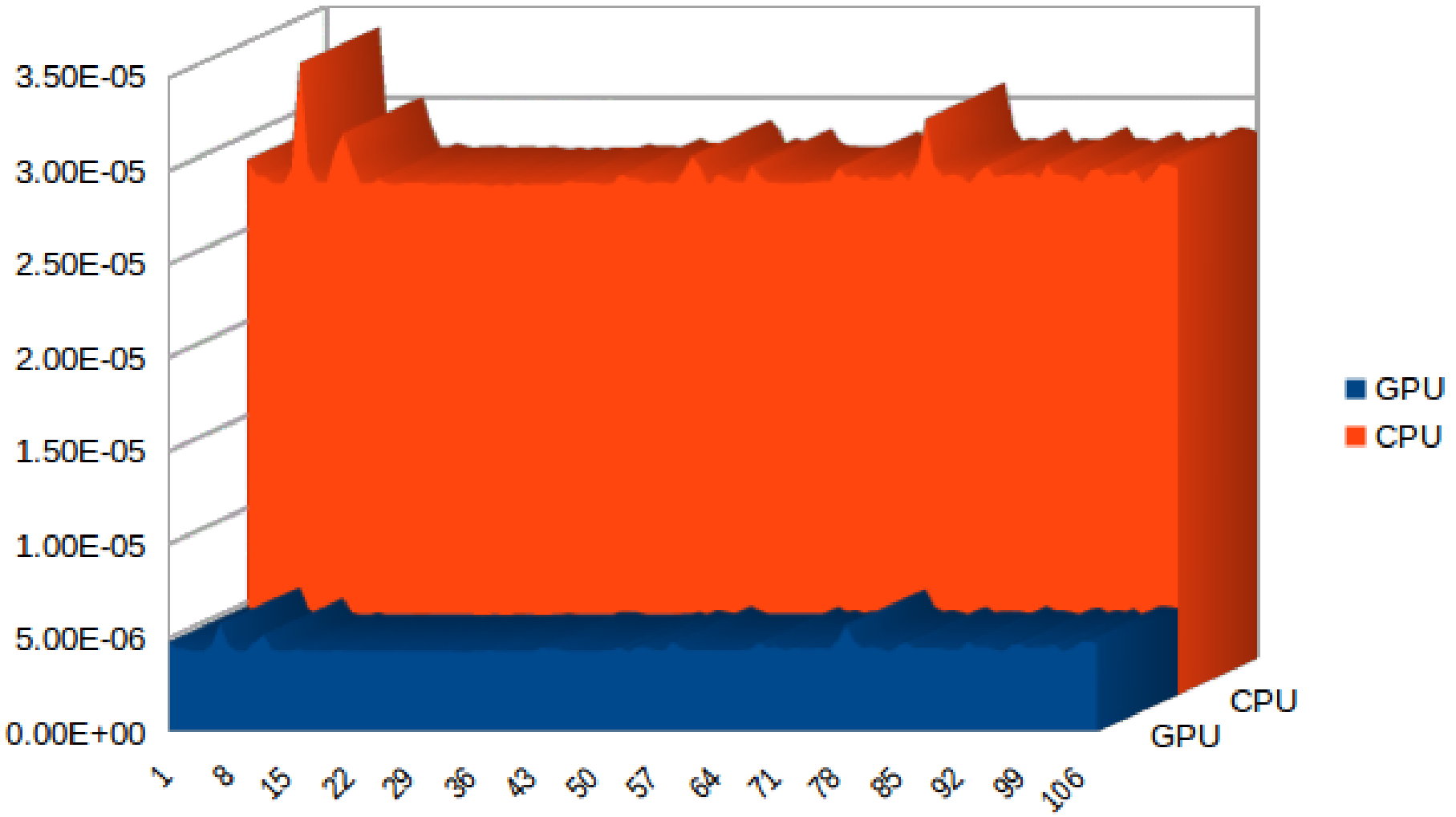

Figure: Serial execution vs parallel reduction on local

Varying Block Size

Threads per block

	numBlocks	
block Size	1	2
4	0.00588897	0.0034006
8	0.00338941	0.00214689
12	0.00256751	0.00178733
32	0.00151549	0.00121056
64	0.0012111	0.00105951
80	0.00115725	0.00103003
160	0.00103155	0.00096255
200	0.000997303	0.000959612
256	0.000971948	0.000941406
508	0.000942387	0.00100652
620	0.000941578	0.00092019
800	0.00093875	0.000916692

Varying Block Size

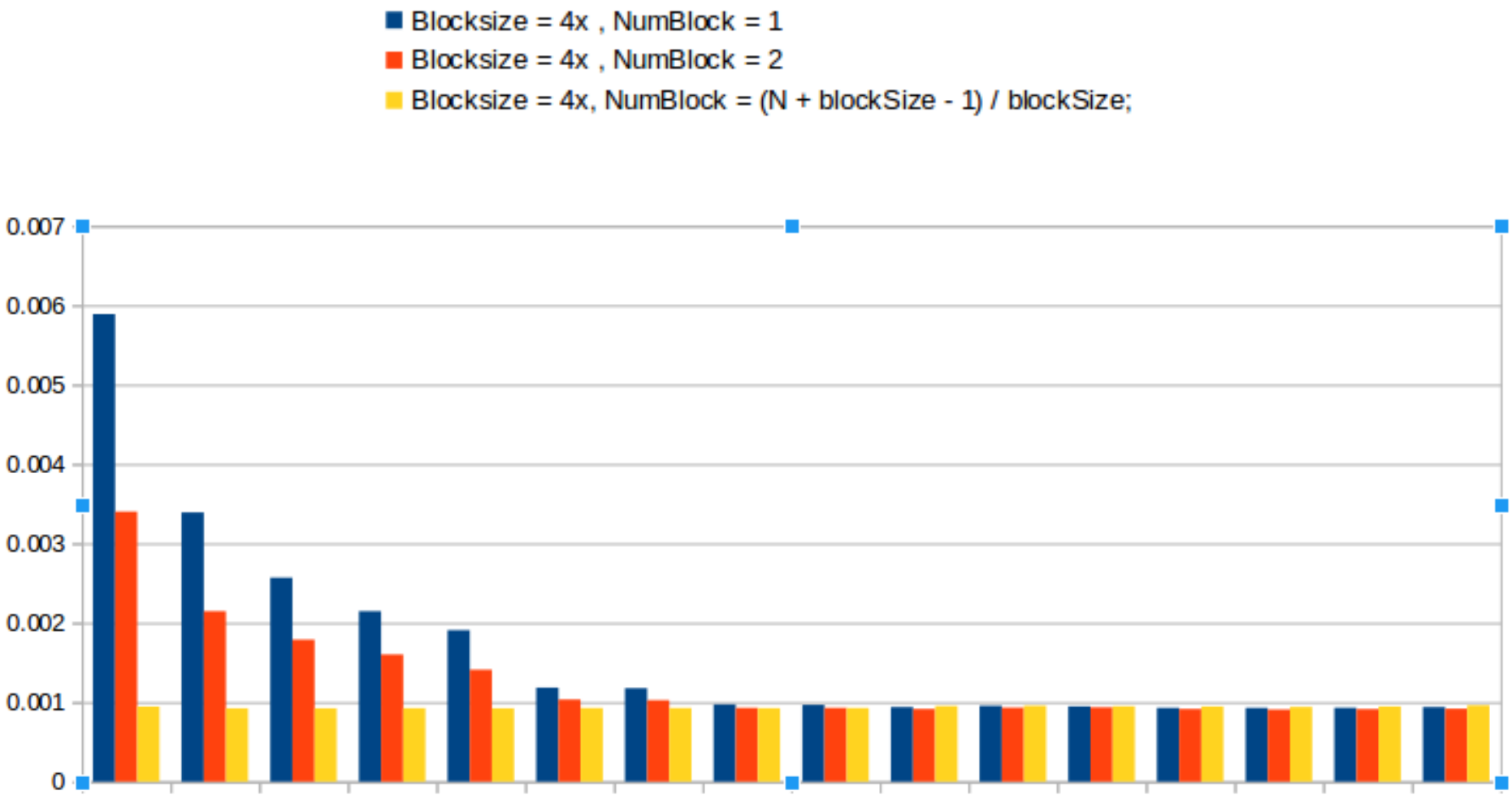

Figure: Performance on different block size

Varying Batch Size

Batch Size	Image Size		
	30030	4096	60000
1	0.00176856	0.00176758	0.00176983
2	0.00167166	0.00167725	0.00167233
3	0.00164436		0.0016355
4		0.00160889	
6	0.00161172		0.00160633
7	0.00159374		
8		0.00159668	0.0015985
10	0.00159474		0.00159517
11	0.00158874		
13	0.00159474		
14	0.00158042		
15	0.00159707		0.00158867
16		0.00156494	0.0015865

Varying Batch Size

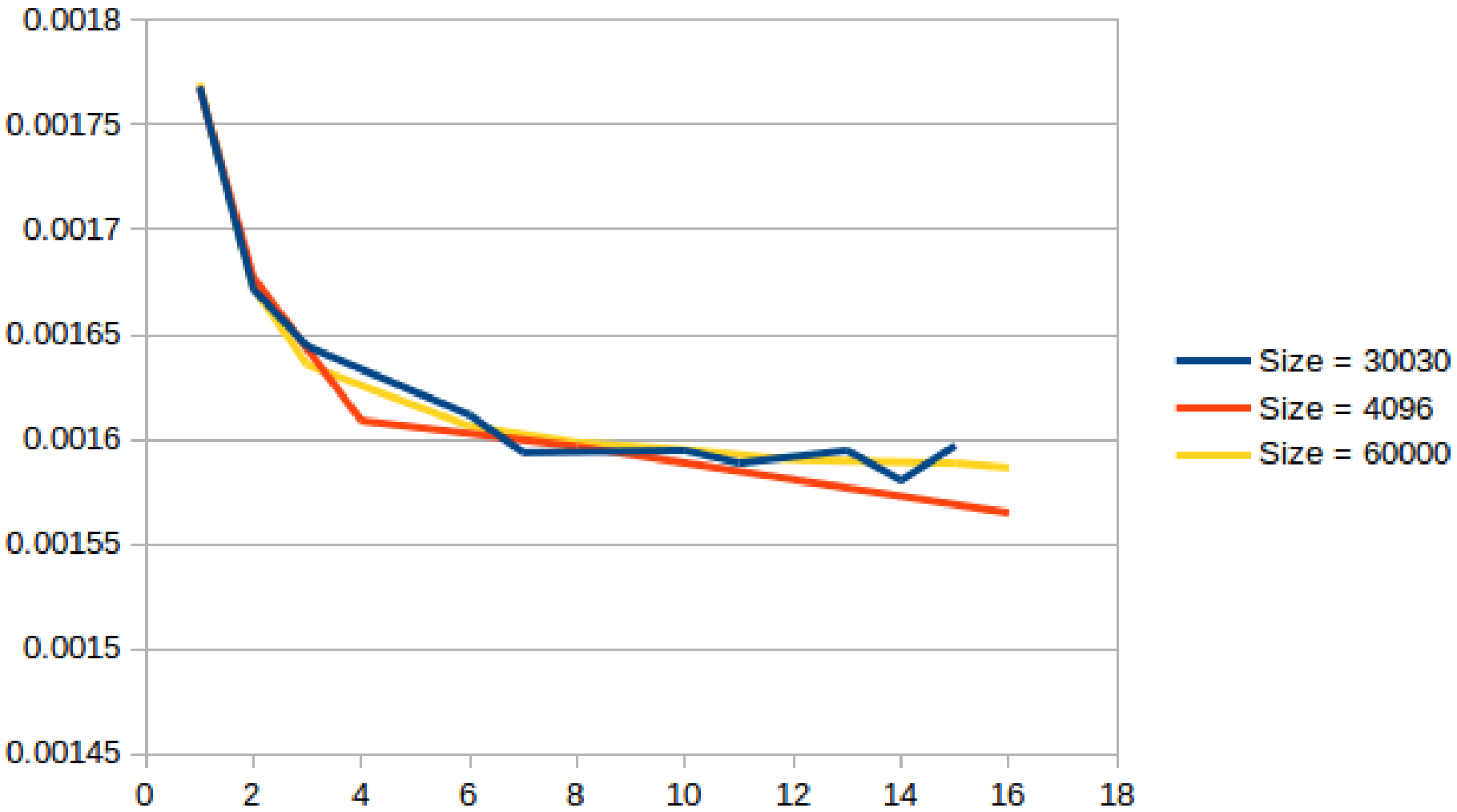

Figure: Time to train

Varying Batch Size Memory

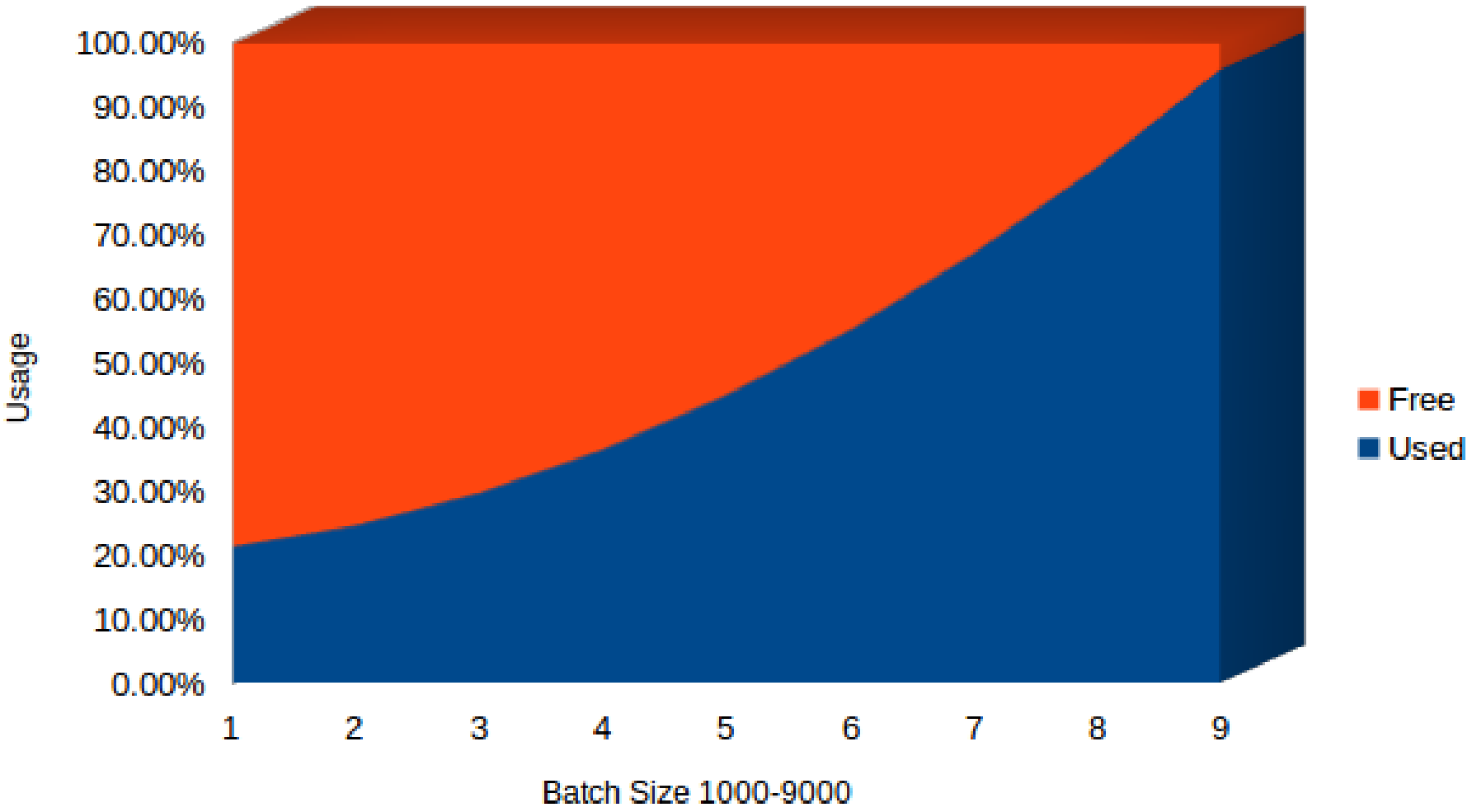

Figure: Memory Usage

Challenges and Learning

- Allocating memory to ensure the coalesced access
- Building the grid and the blocks
- Memory transfer for 2 dimensional arrays
- Lots of memory utilization for small size image - cudaMallocPitch

Challenges and Learning

cudaMallocPitch

cudaError_t cudaMallocPitch (void ** devPtr, size_t * pitch, size_t width, size_t height)

- The pitch returned in *pitch by cudaMallocPitch() is the width in bytes of the allocation
- the smallest upper multiple of 32/64/128 bytes
- For array of size 784 * batchSize
  - It allocates the 3584 for single row
  - 3584 = 128 * 28
- For array of size 7840 * batchSize
  - It allocates the 31744 for single row
  - 31744 = 128 * 248
- This padding of arrays is because bank mechanism in cuda, concerning shared memory access.
- Check if the number of bytes allocated makes it correctly aligned, for full coalesced access

Conclusion

- For pure computation, CUDA offers enormous speedups through parallelism
- Use coalesced access for better performance
- CUDA is not well-suited to problems which require a moderate amount of memory
- Transfer data only once and doing multiple analysis on it helps
- Using 2D Grids helps to improve performance due to fast access to shared memory and nice alignment

Future Work

- Try the Parallel Softmax Regression with multi dimensional images
- Use of 3 dimensionality of grid and blocks
- Add more layers to the classification network
- Run on more power full GPUs

References

A http://on-demand.gputechconf.com/gtc/2012/video/S0624-Monday-Introduction-to-CUDA-C.mp4

B Chong, Wang, David Blei, and Fei-Fei Li. "Simultaneous image classification and annotation." Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009.

C Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

D http://cuda-programming.blogspot.com/

E https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html