Josh Helzerman

Program N

Specifications

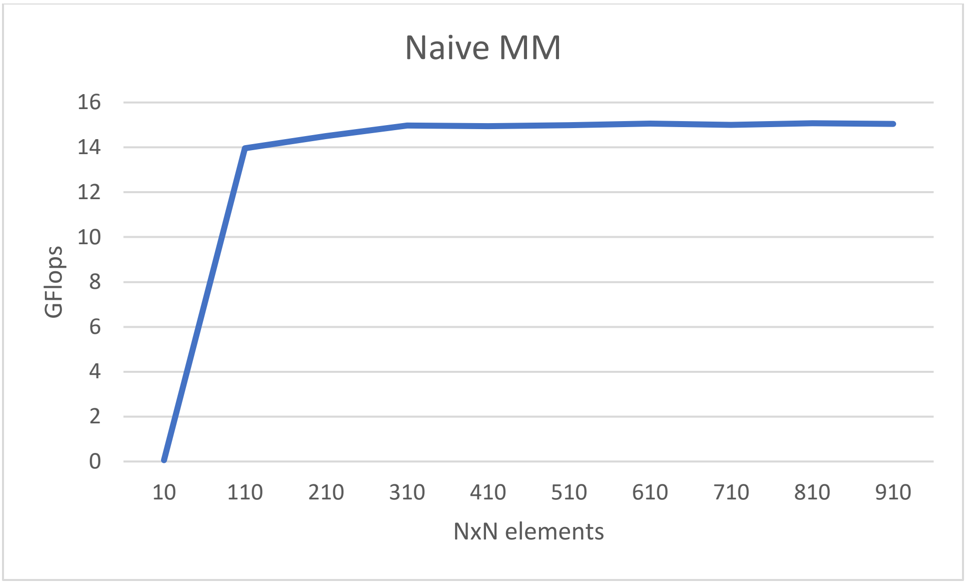
Graphics Card name	GeForce GTX 745
GPU Chip	GM107
Bus	PCIe 3.0 x 16
Memory	4 GB DDR3 128 bit
GPU clock	1033 MHz
Memory clock	900 MHz
Shaders	384
TMUs	24
ROPs	16
Compute Capability	3.0
Microarchitecture	Kepler
Maximum x dimension of grid of thread blocks	2^31 - 1
Maximum y/z dimension of grid of thread blocks	65535
Maximum x or y dimension of block	1024
Maximum z dimension of a block	64
Maximum number of threads per block	1024
Warp size	32
Maximum resident blocks per MP	16
Maximum resident warps per MP	64
Maximum resident threads per MP	2048
Number of 32-bit registers per MP	64 K
Maximum number of 32-bit registers per block	64 K
Maximum number of 32-bit registers per thread	63
Maximum shared memory per MP	48 KB
Maximum shared memory per block	48 KB
Number of shared memory banks	32
Amount of Local memory per thread	512 KB
Constant memory size	64 KB
Cache working set per MP for constant memory	8 KB
Maximum instructions per kernel	512 million
Theoretical peak performance (FP64)	24.79 GFlops

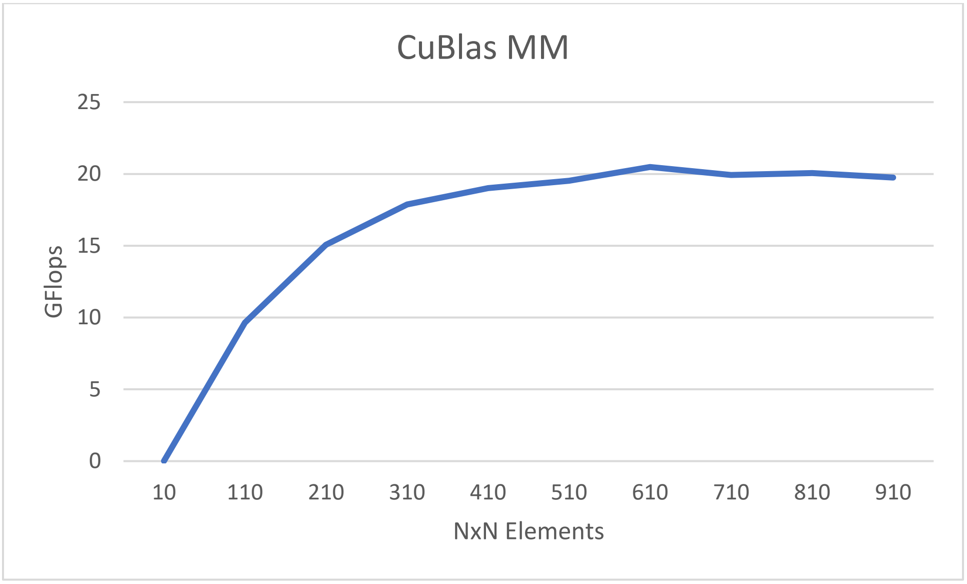
https://www.techpowerup.com/gpu-specs/?mfgr=NVIDIA&sort=name

https://en.wikipedia.org/wiki/CUDA

https://www.techpowerup.com/gpu-specs/geforce-gtx-745-oem.c2561

Results





Implementation

My naïve implementation uses an if statement to ensure the thread is within bounds of the matrix. It then calculates its value of its C element by iterating through a row of A and a column of B. B is not transposed. I compute the same A * B operation using cublas and compare the two matrices to ensure equality. Since cublas returns the results in column-major, I compute the naïve implementation backwards (B * A instead of A * B), this allows me to easily compare the results with cublas results.