GPU Multiplication Algorithms

J. C. Thomas

The University of Iowa, Dept. of Biostatistics

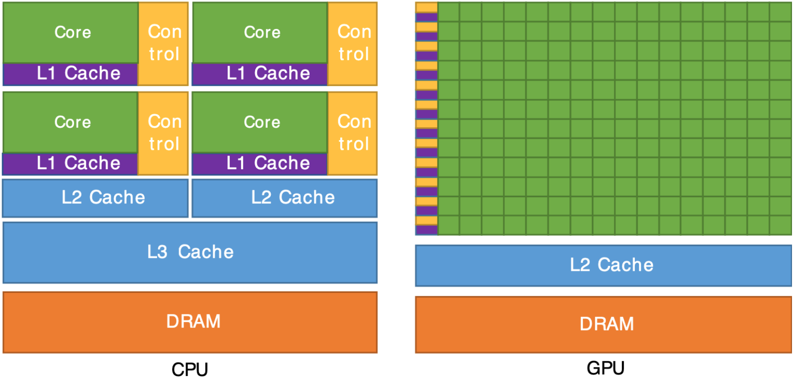
Dec 14 2025

GPUs

Why GPUs?

- Why are GPUs so expensive?
    - It turns out everyone loves to do matrix multiplication
- Why is matrix multiplication so popular?
    - Deep learning and AI are all pretty much just a bunch of matrix multiplications. Analogous: think about the closed form solution of linear regression
- How do GPUs help with matrix multiplication?
    - Matrix multiplication is a bunch of independent arithmetic
    - This can be parallelized via a GPU

How GPUs?


Core
Con trol
L1 Cache
Core
Con trol
L1 Cache
Core
Con trol
L1 Cache
Core
Con trol
L1 Cache
L2 Cache
L2 Cache
L3 Cache
DRAM
CPU
L2 Cache
DRAM
GPU

Attacking a Village

- If we are a fantasy warlord and we need to destroy all the houses in a village, should we send in our 4 trolls or our army of 100 goblins?
    - Our trolls are really strong, but there are only a few of them. They will need to destroy a house, move to the next house, destroy it, ect.
    - Our goblins are not as strong but there are a lot of them and they are quick. Each goblin can concentrate on destroying an individual house.

GPU Computation

- Units are threads, blocks, and grids
- Little green boxes/goblins are threads
  - This is the primary computational unit
- Blocks are groups of threads, can share memory
- Grids are the entire workspace
- For matrix multiplication, this maps nicely. Each thread is an element of the output matrix, the entire output matrix is the grid. Blocks are less intuitive.

Matrix Multiplication

By Hand Approach

- Why is this a good candidate for parallel computing?
- How does this work within the thread, block, grid structure?

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \text{x} \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$

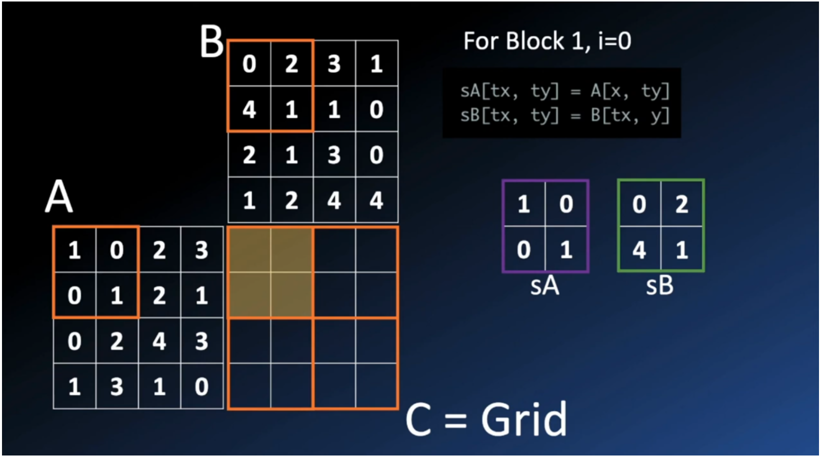
$$= \begin{bmatrix} 1\text{x}10 + 2\text{x}20 + 3\text{x}30 & 1\text{x}11 + 2\text{x}21 + 3\text{x}31 \\ 4\text{x}10 + 5\text{x}20 + 6\text{x}30 & 4\text{x}11 + 5\text{x}21 + 6\text{x}31 \end{bmatrix}$$

$$= \begin{bmatrix} 10+40+90 & 11+42+93 \\ 40+100+180 & 44+105+186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

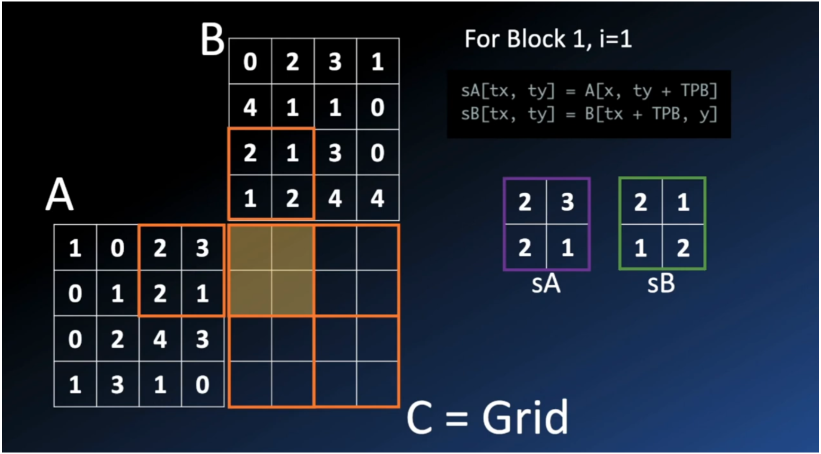
By Hand Approach Considerations

- Perfect use of parallel computing!
- Not very memory efficient. For multiplying two 4x4 matrices, each value is loaded in 4 times
- Tile method improves on this, each value loaded just 2 times

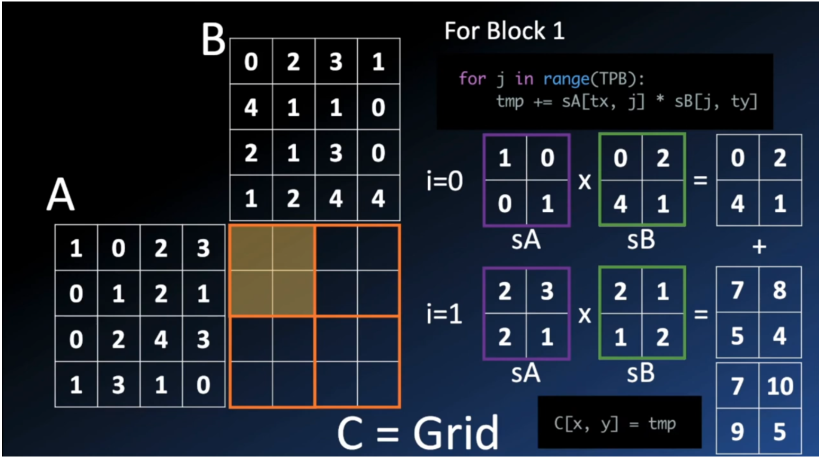
Tiled Approach



Tiled Approach



Tiled Approach



Other Approaches

- Matrix multiplication is essential and this problem has been studied a ton
- There are many, MANY complex algorithms that are much faster than these. However, these give a flavor of how complex ones work with the computational system.

Computation

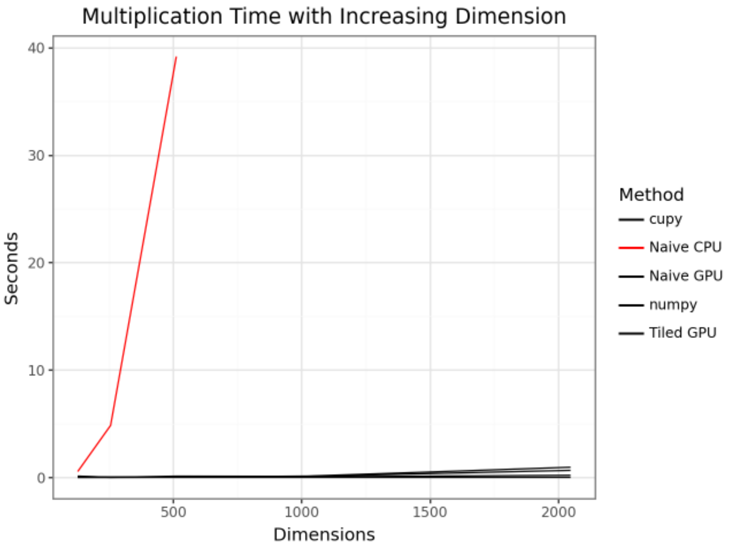
Interfacing

- GPUs (specifically Nvidia GPUs) can be communicated with via a C-like language called CUDA
- Instructions are passed to threads in things called kernels, function-like syntax
- Most softwares have tools and extensions for interfacing with GPUs (R, python, Julia, ect)
- Good python packages: cupy and numba

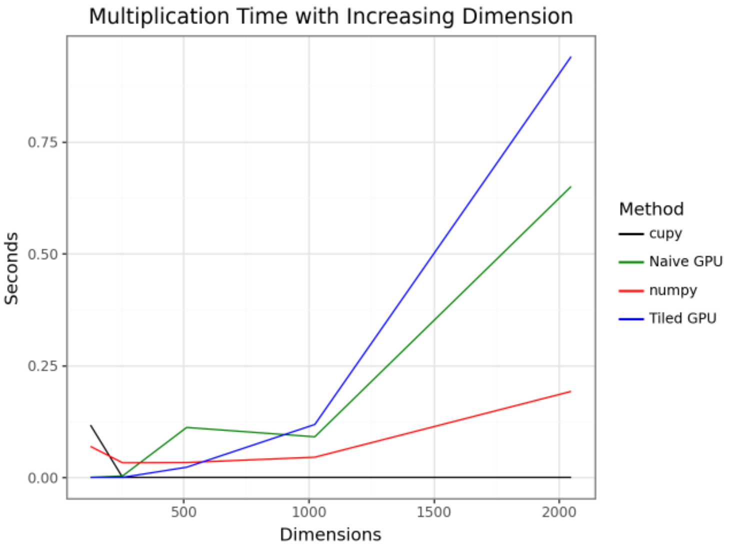
Simulation 1

- Goal: compare speed with increasing matrix dimension for the following algorithms
  - naive method on CPU
  - pre-built in method on CPU (numpy)
  - naive method on GPU
  - tiled method on GPU
  - pre-built in method on GPU (cupy)
- Square matrices of dimension 128, 256, 512, 1024, and 2048
- Threads: 16

Simulation 1 Results



Simulation 1 Results

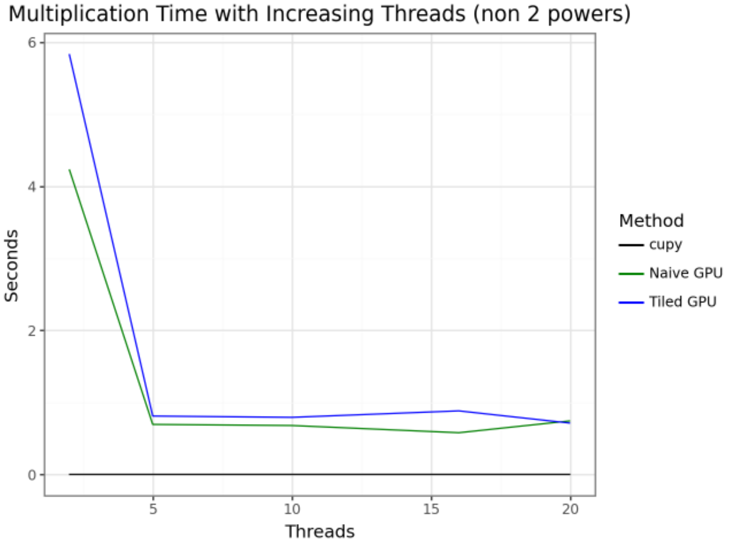


Computation

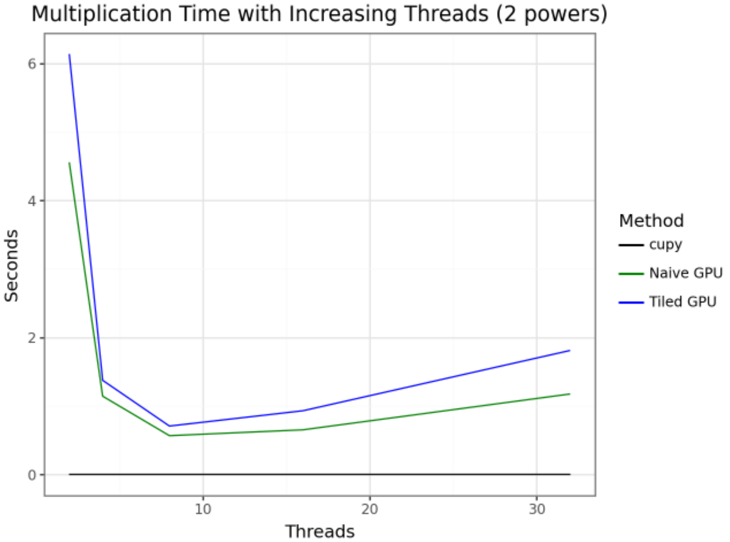
Simulation 2

- Goal 1: compare speed with static dimension and increasing block size for the following algorithms
  - naive method on GPU
  - tiled method on GPU
  - pre-built in method on GPU (cupy)
- Goal 2: compare speed between 2-power sizes and non-2 power sizes
  - Square matrices with approximately equal dimension (2000 and 2048)
  - Threads for non 2-power: 2, 5, 10, 16, 20
  - Threads for 2-power: 2, 4, 8, 16, 32

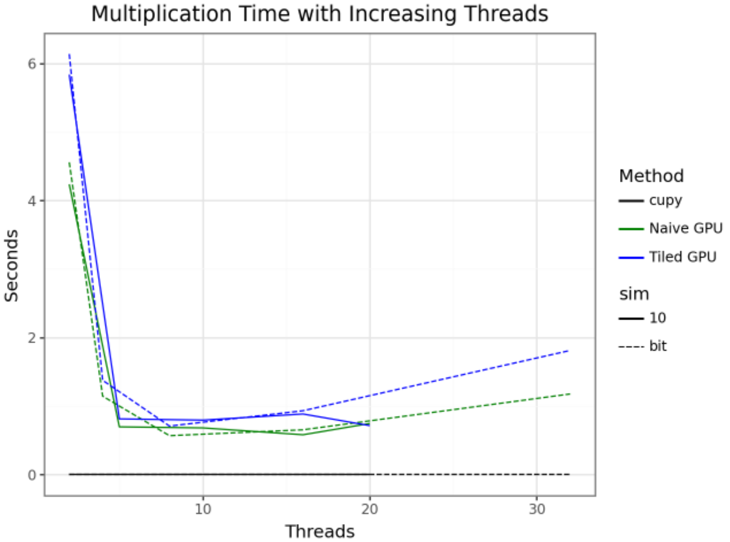
Simulation 2 Results



Simulation 2 Results



Simulation 2 Results



Conclusion

Conclusions

- What should happen vs what does happen
- How does this relate to the Warhammer 40k Universe

Recources

- Python packages: numba, cupy
- Youtube: nickcorn93, "Tutorial: CUDA programming in Python with numba and cupy"
- My code is all posted on my github under jcthomas531 if you want to play around with it