PARALLEL MATRIX MULTIPLICATION

Student: Kyle Pellechia

Instructor: Professor Miller

University at Buffalo The State University of New York

University at Buffalo The State University of New York

SEQUENTIAL MATRIX MULTIPLICATION

Sequential Approach

- Solve C = AxB given 2 matrices
  - A(nxm) and B(mxp) with values from [1,N]
- $$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}.$$
- Runtime: θ(nxmxp) so θ(n^3)

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

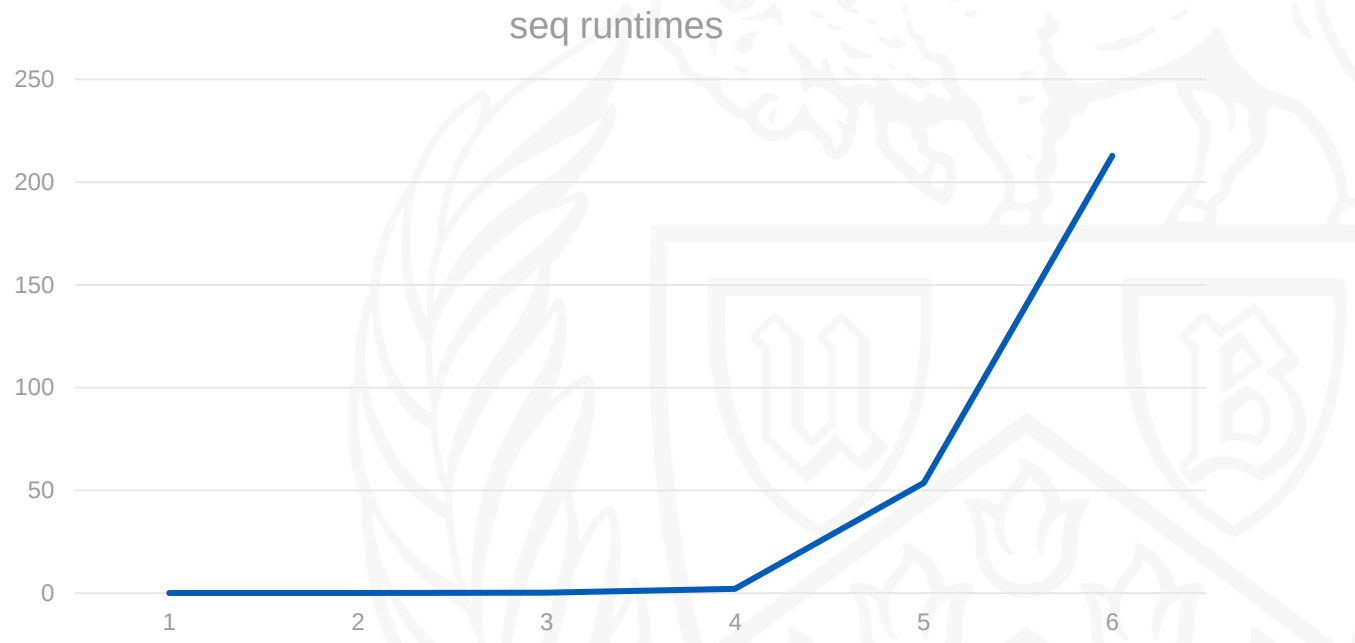
Sequential Implementation

- Input: matrices A and B
- Let C be a new matrix of the appropriate size
- For i from 1 to n:
  - For j from 1 to p:
    - Let sum = 0
    - For k from 1 to m:
      - Set sum $\leftarrow$ sum $+ A_{ik} \times B_{kj}$
    - Set $C_{ij} \leftarrow$ sum
- Return C

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

Sequential Runtimes

Size	Speed(seconds)
1K	0.0045
10K	0.043
100k	0.275
1mil	2.15
25mil	53.58
100mil	212.71



University at Buffalo The State University of New York

CANNON'S ALGORITHM

Parallel Approach

- Cannon's Algorithm
  - https://en.wikipedia.org/wiki/Cannon%27s_algorithm

- Solve C = A x B where A and B are (nxn) matrices with A and B split into p blocks
  - Partition A and B into A(i, j) and B(i, j)
    - $(0 \leq i \leq \sqrt{p})$
    - $(0 \leq j \leq \sqrt{p})$

- Processor P(i, j) stores A(i, j) and B(i, j) and computes C(i, j)

- Shifting submatrices and perform multiplication
  - A shifts left
  - B shifts up

Parallel Approach cont.

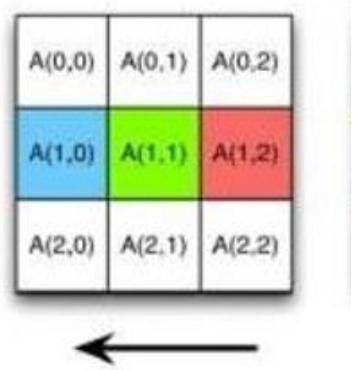
- A(nxn) and B(nxn) are partitioned into submatrices
  - Create p blocks of size (n /$\sqrt{p}$) x (n /$\sqrt{p}$)
- Initial skewing of matrices alignment
  - Shift A blocks left $\sqrt{p}$ -1 times
  - Shift B blocks up $\sqrt{p}$ -1 times
- Processor P(i, j) contains blocks A(i, j) and B(i, j)
  - Multiply A and B blocks
    - Perform sequential algorithm on A(i, j) and B(i, j)

Parallel Approach cont.

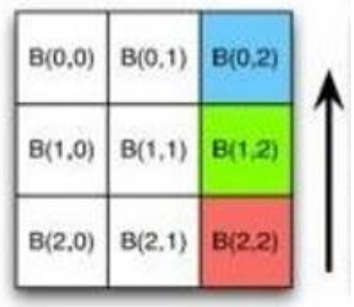
- Shift procedure
  - Move A(i, j) to A(i-1, j)
    - Wrap if i value < 0
  - Move B(i, j) to B(i, j-1)
    - Wrap if j value < 0
- Repeat shifting and multiplying $\sqrt{p}$ - 1 times

Cannon's Algorithm Example

- A



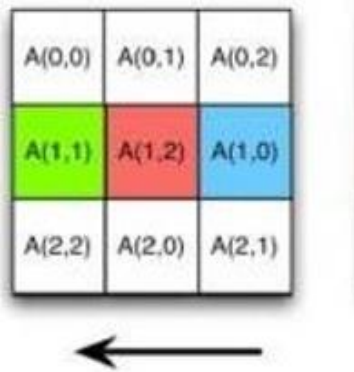
- B



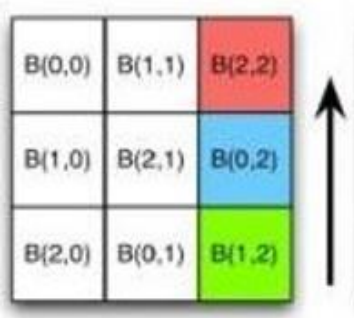
Initial A, B

Initial Skewing

- A

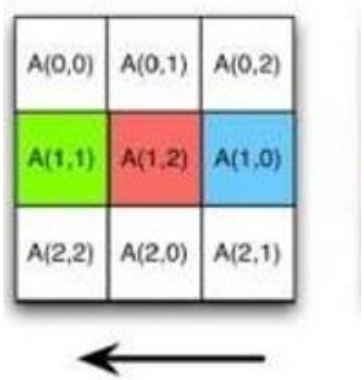


- B

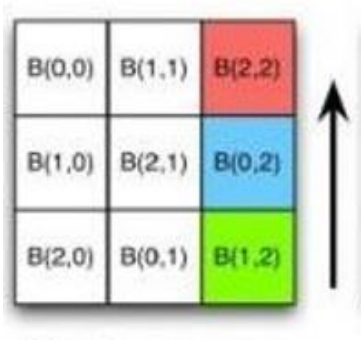


Cannon's Algorithm Example cont.

- A



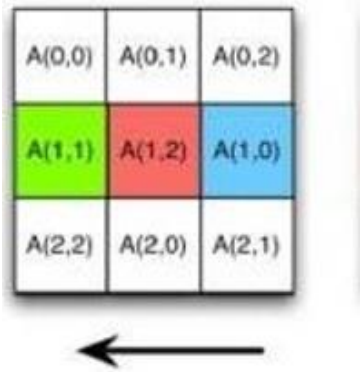
- B



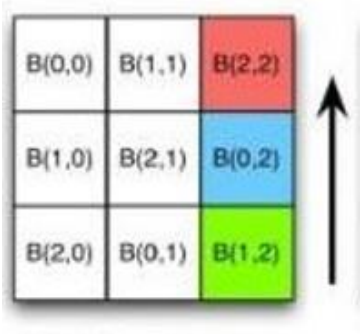
- Local Multiplications
  - EX. C(1,2) = C(1,2) + A(1,0) X B(0,2)

Cannon's Algorithm Example cont.

- A

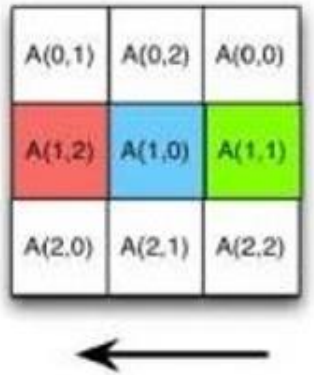


- B

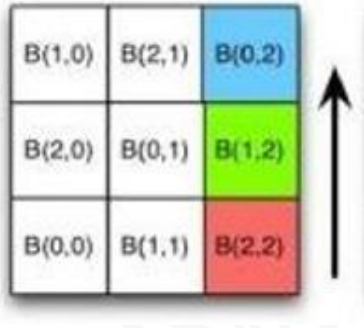


Shift with wrapping

- A

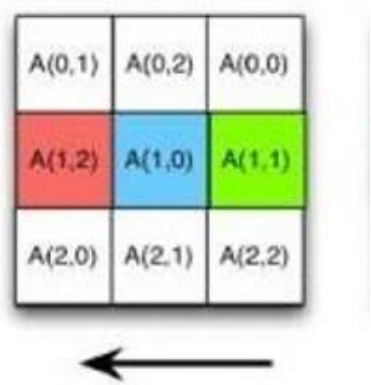


- B

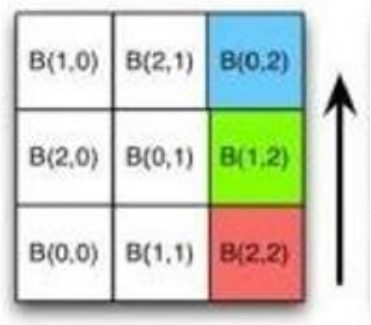


Cannon's Algorithm Example cont.

- A



- B



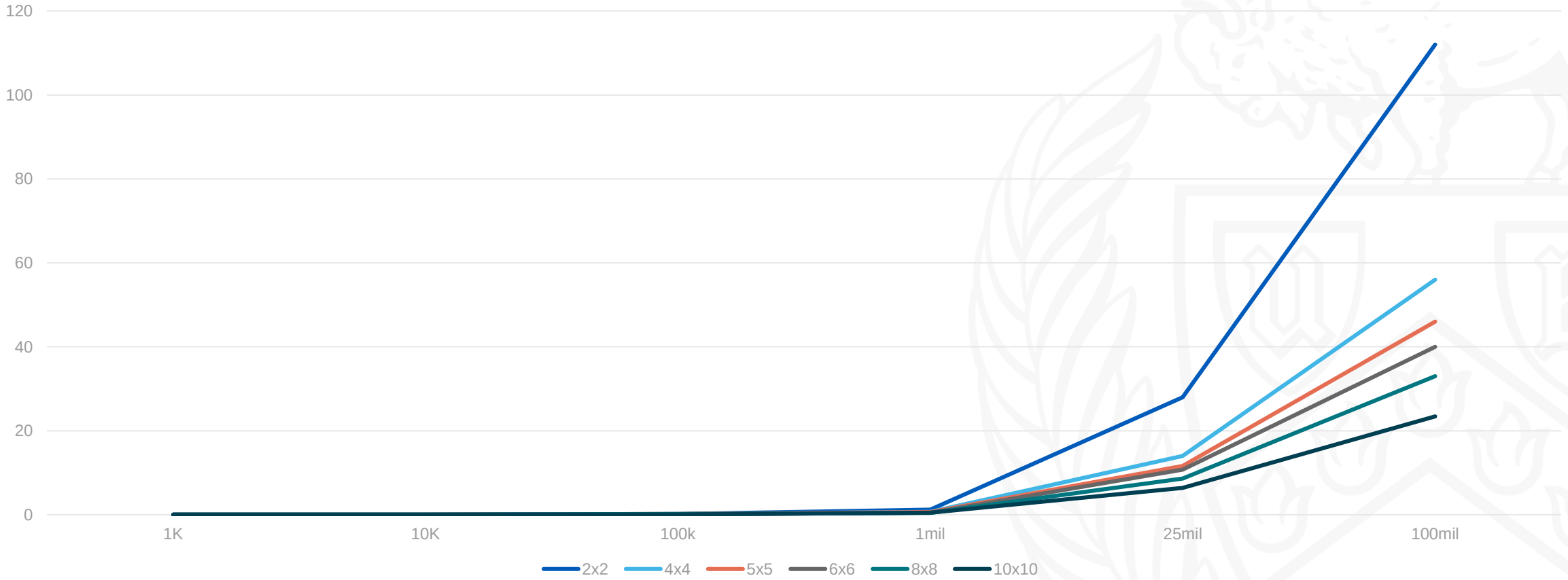
- Local Multiplications
  - EX. C(1,2) = C(1,2) + A(1,1) X B(1,2)

Cannons Algorithm Results

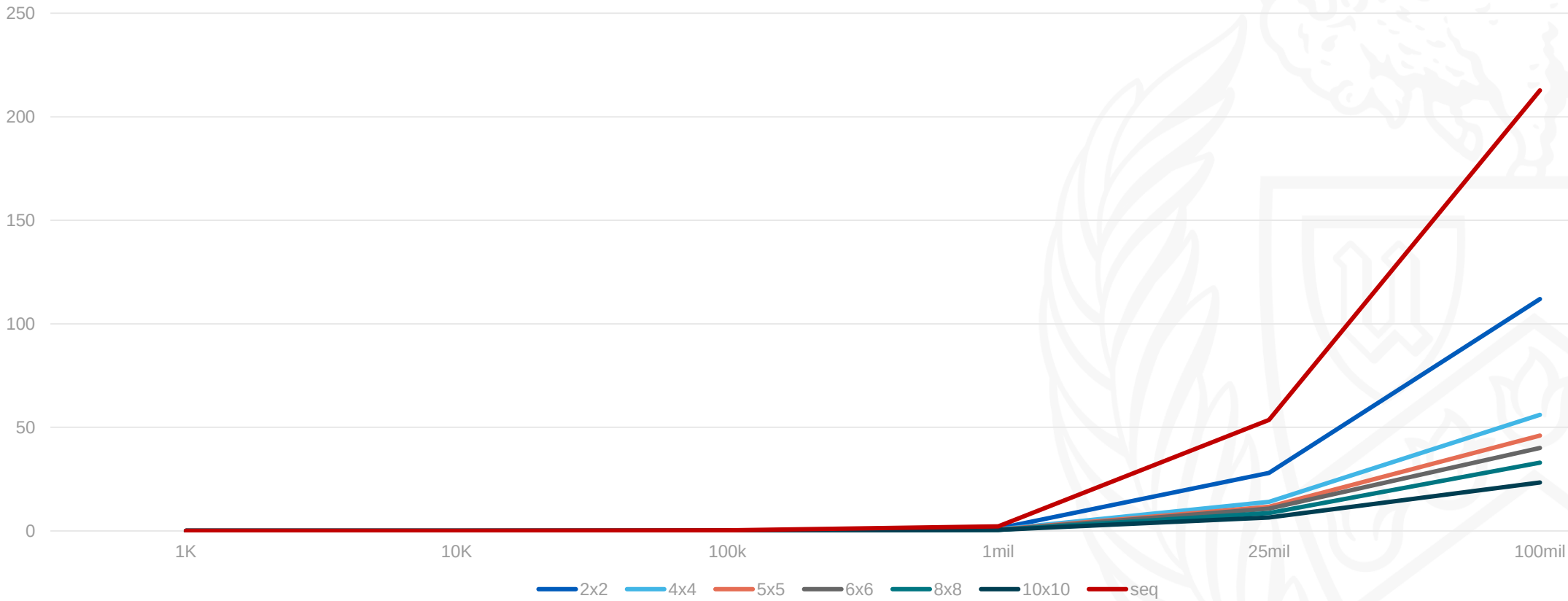
	2x2	4x4	5x5	6x6	8x8	10x10
1k	0.003	0.0035	0.006	0.006	0.008	0.011
10k	0.02	0.015	0.015	0.0131	0.015	0.017
100k	0.19	0.11	0.18	0.079	0.18	0.055
1mil	1.21	0.7	0.64	0.48	0.45	0.472
25mil	28	14	11.6	10.75	8.59	6.403
100mil	112	56	46	40	33	23.4

*Left column in overall data size

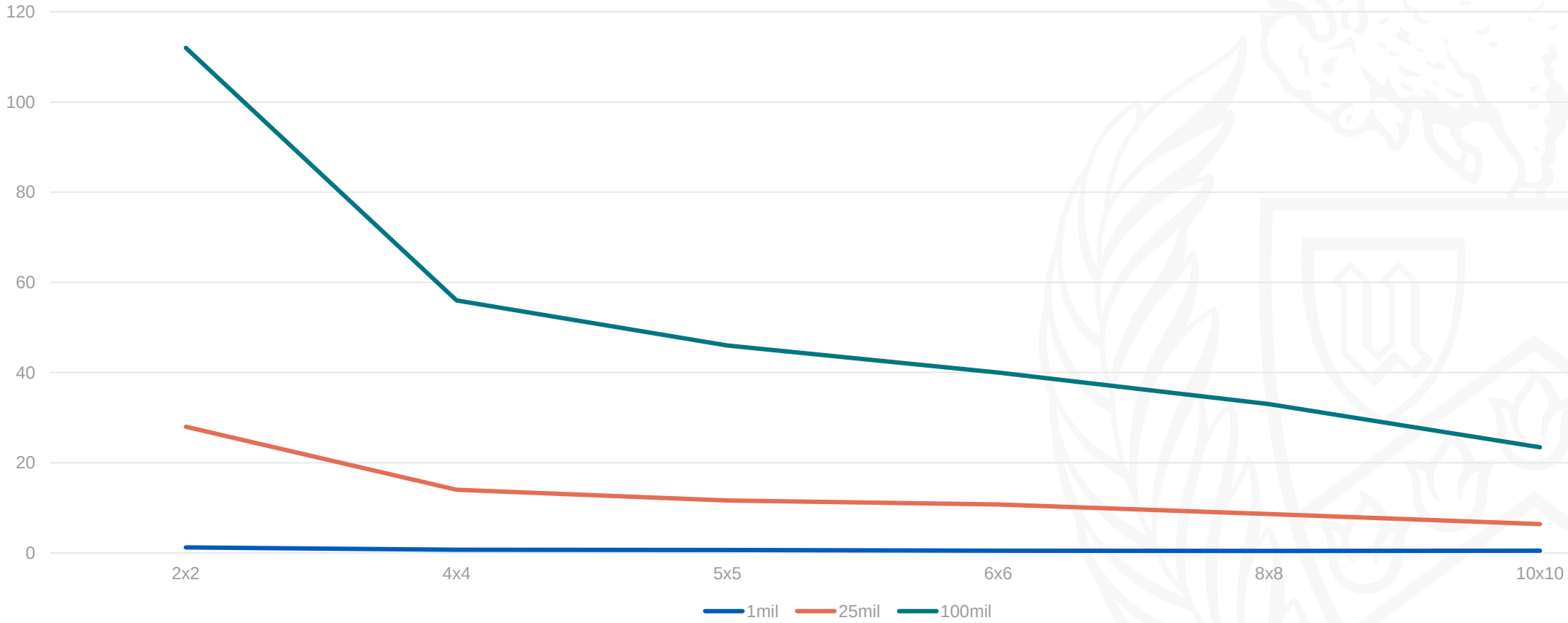
Comparing increase in data



Comparing increase in data w/sequential



Comparing increase in processors



Observations

- Parallelizing matrix multiplication drastically reduced runtime
- Amount of processors must be a perfect square
- Runtime of Cannons algorithm
  - $\theta(n^3/ p)$ where p is the amount of processors
  - Asymptotically this is $\theta(n^3)$
- For this scope of this problem, increasing the amount of processor decreased runtime

What I learned

- How to use CCR systems
- Slurm Scripting
  - Careful with ENV variables
- Utilizing MPI for processor communication
- Understanding Cannon’s Algorithm

Questions?