Name: Rohit Pujar Nagraj
USC ID: 4664 4939 18

Programming Homework 1 - Report

1. Matrix Multiplication

1.1. Naive Matrix Multiplication

- Execution Time: 1,383,238,290,000 ns (1,383.238290 sec)
- Performance: 99,360,287 FLOPs (99.360287 MFLOPs)

1.2 Block Matrix Multiplication

1.2.1 Block Size = 4

- Execution Time: 483,694,805,000 ns (483.694805 sec)
- Performance: 284,143,952 FLOPs (284.143952 MFLOPs)

1.2.2 Block Size = 8

- Execution Time: 342,598,416,000 ns (342.598416 sec)
- Performance: 401,166,342 FLOPs (401.166342 MFLOPs)

1.2.3 Block Size = 16

- Execution Time: 310,128,092,000 ns (310.128092 sec)
- Performance: 443,168,345 FLOPs (443.168345 MFLOPs)

Method	Block Size	Execution Time	Performance
1.1 Naive Matrix Multiplication	1	1,383,238,290,000 ns (1,383.238290 sec)	99,360,287 FLOPs (99.360287 MFLOPs)
1.2 Block Matrix Multiplication	4	483,694,805,000 ns (483.694805 sec)	284,143,952 FLOPs (284.143952 MFLOPs)
	8	342,598,416,000 ns (342.598416 sec)	401,166,342 FLOPs (401.166342 MFLOPs)
	16	310,128,092,000 ns (310.128092 sec)	443,168,345 FLOPs (443.168345 MFLOPs)

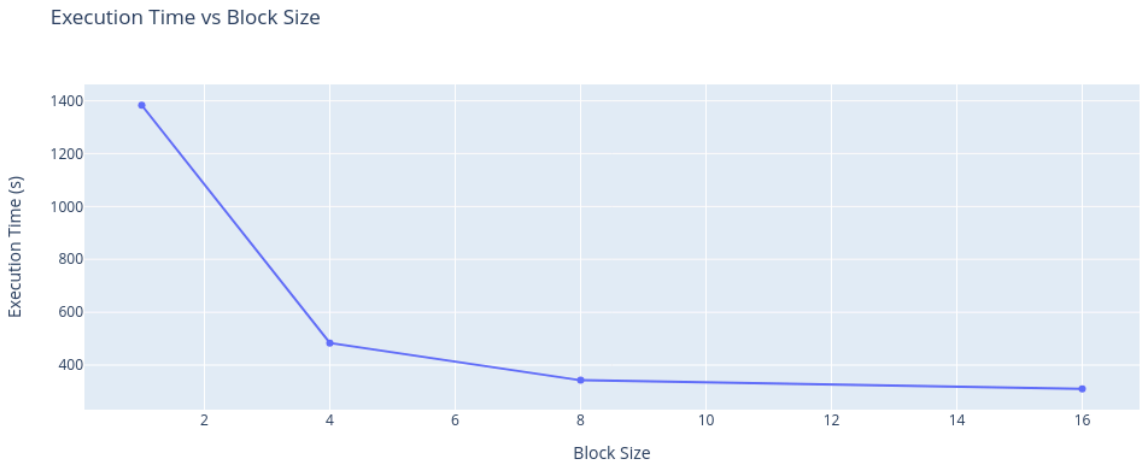

Fig 1: Execution Time vs Block Size

Observations: From Figure 1, we observe that there's almost an exponential drop in execution time as the block size increases. This can be explained by the following:

1. When the block size is 1 (naive matrix multiplication), and the loop order is i-j-k, with the equation C[i][j] += A[i][k] * B[k][j], there is a cache miss at every call of B[k][j] since C store matrices in a row-major form and the matrix is being iterated column-wise. However, when we perform block matrix multiplication, the entire block can be loaded into the cache, thus speeding up the operations.
2. As the block size increases, there is a diminishing return since the cache misses reduce exponentially.
3. If we increase the block size beyond a point that does not fit into the cache, we will observe a plateau or performance degradation.

Test Platform:

- Architecture: x86 64 bit
- OS: Linux (Fedora)
- Number of Runs: 2

2. K-Means Algorithm

Execution Time for 30 iterations: 277,964,000 ns (0.277964 sec)

Output Image: