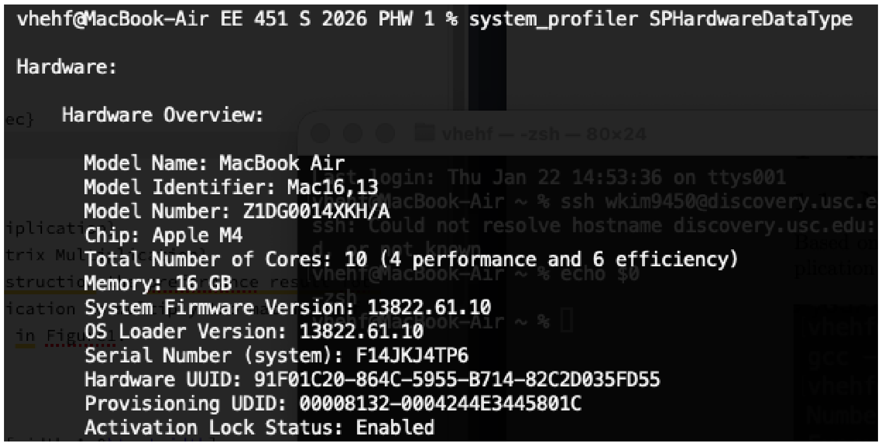
EE-451: Parallel and Distributed Computation

Wooyoung Kim(2717106812) phw1

January 22nd 2026

1 Hardware Specification



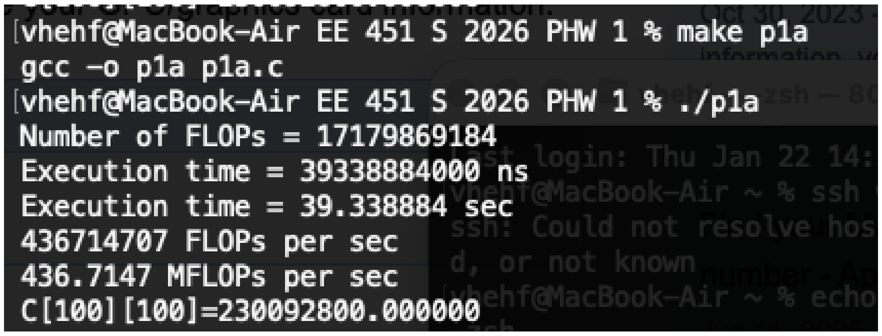
```
vhehf@MacBook-Air EE 451 S 2026 PHW 1 % system_profiler SPHardwareDataType

Hardware:

    Hardware Overview:

      Model Name: MacBook Air
      Model Identifier: Mac16,13
      Model Number: Z1DG0014XKH/A
      Chip: Apple M4
      Total Number of Cores: 10 (4 performance and 6 efficiency)
      Memory: 16 GB
      System Firmware Version: 13822.61.10
      OS Loader Version: 13822.61.10
      Serial Number (system): F14JKJ4TP6
      Hardware UUID: 91F01C20-864C-5955-B714-82C2D035FD55
      Provisioning UDID: 00008132-0004244E3445801C
      Activation Lock Status: Enabled
```

Figure 1: Hardware specification for homework

2 Matrix Multiplication

2.1 Naive Matrix Multiplication

Based on the homework instructions, the performance results of the naive matrix multiplication for two matrices of size 2K × 2K are shown in Figure 1.



```
vhehf@MacBook-Air EE 451 S 2026 PHW 1 % make p1a
gcc -o p1a p1a.c
vhehf@MacBook-Air EE 451 S 2026 PHW 1 % ./p1a
Number of FLOPs = 17179869184
Execution time = 39338884000 ns
Execution time = 39.338884 sec
436714707 FLOPs per sec
436.7147 MFLOPs per sec
C[100][100]=230092800.000000
```

Figure 2: Execution time and performance

2.2 Block Matrix Multiplication

As shown in Figure3, the results clearly demonstrate that memory access patterns matter just as much as raw processing power. The naive implementation was by far the slowest, taking over 39 seconds to complete. This huge delay happens because the code accesses the second matrix in a column-wise fashion, which fights against the row-major layout of C arrays. The CPU is constantly forced to fetch new data from slow main memory because it can't find what it needs in the cache, leading to severe bandwidth bottlenecks.

As soon as we introduced blocking, the performance improved dramatically, but the results also revealed an interesting optimal spot. The fastest execution time was achieved with a block size of 8, which finished in about 15.7 seconds. This indicates that 8×8 tiles fit perfectly into the CPU's fastest L1 cache, allowing the processor to crunch numbers without waiting for data. However, making the blocks larger didn't help. Performance actually started to dip at sizes 16 and 32, likely due to cache conflicts or running out of registers.

By the time we increased the block size to 128, the execution time worsened significantly to over 24 seconds. A block that large requires far more memory than the L1 cache can hold, so the CPU is forced to fall back on the slower L2 cache or RAM. This proves that blocking is a balancing act. The blocks need to be large enough to minimize loop overhead, but small enough to stay resident in the fastest tier of cache memory.

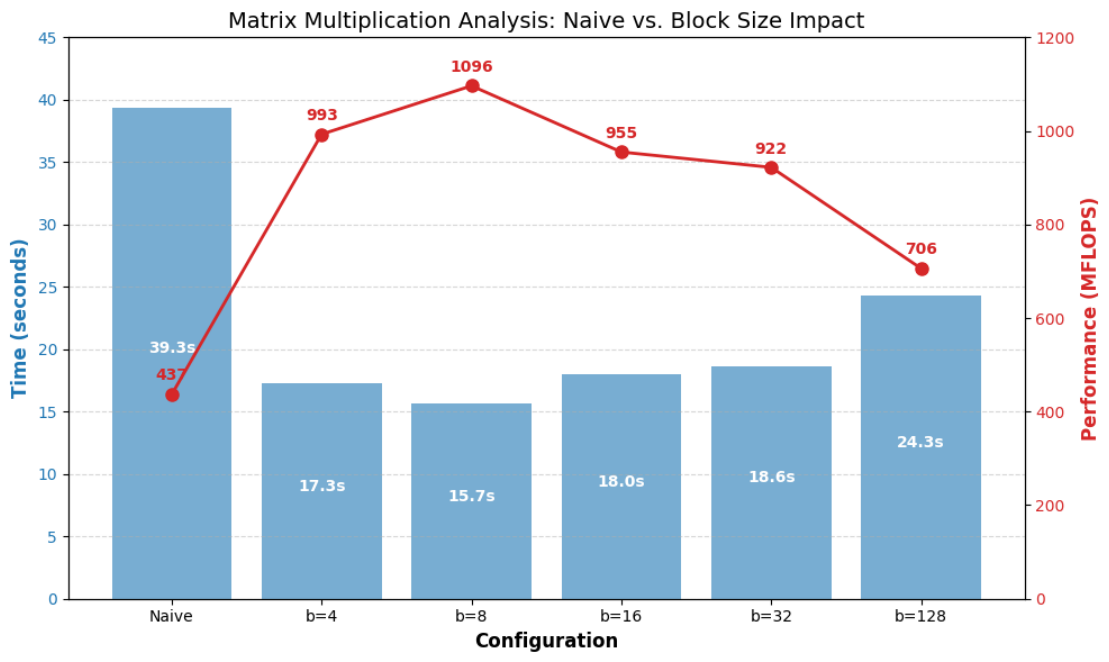
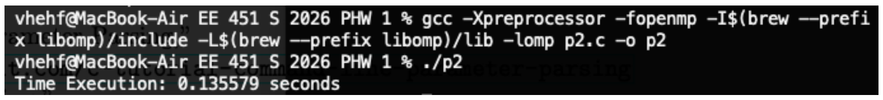


Figure 3: Execution time and performance

3 K-Means algorithm

The experiment successfully applied the K-Means clustering algorithm to segment an 800x800 grayscale raw image into 4 distinct intensity zones. The program iterated 30 times to calculate optimal center values (means) for the pixel intensities. As a result, the original continuous grayscale gradients in input.raw were quantized. The final output.raw image consists of pixels mapped to only one of four specific gray levels, effectively simplifying the image into four flat regions based on brightness. The execution time and the output image are shown in Figure 5 and Figure 6, respectively.



```
vhehf@MacBook-Air EE 451 S 2026 PHW 1 % gcc -Xpreprocessor -fopenmp -I$(brew --prefi
x libomp)/include -L$(brew --prefix libomp)/lib -lomp p2.c -o p2
vhehf@MacBook-Air EE 451 S 2026 PHW 1 % ./p2
Time Execution: 0.135579 seconds
```

Figure 4: Execution time

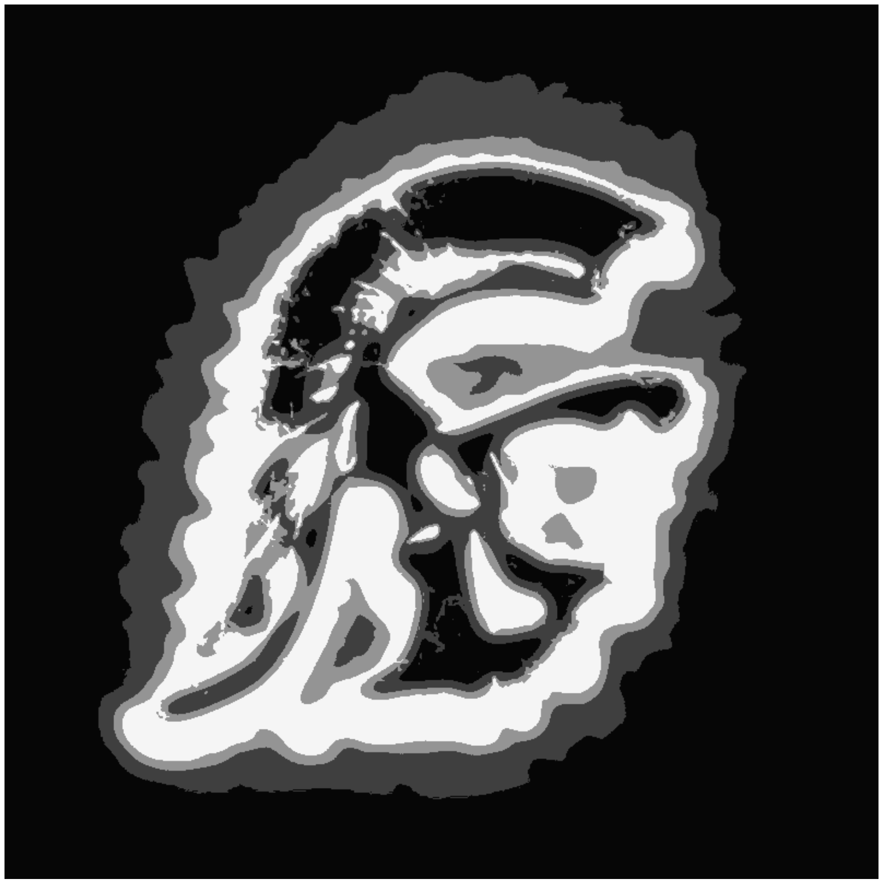


Figure 5: Output Image

4 Appendix

4.1 Raw data for block matrix multiplication

Table 1: Raw Data for Performance Comparison of Naive vs. Block Matrix Multiplication

Configuration	**Block Size (b)**	**Time (sec)**	**Performance (MFLOPS)**
Naive Implementation	N/A	39.34	436.71
Block Implementation	4	17.30	992.85
Block Implementation	8	**15.67**	**1096.38**
Block Implementation	16	17.99	954.89
Block Implementation	32	18.63	922.07
Block Implementation	128	24.33	706.07