Jack Melcher

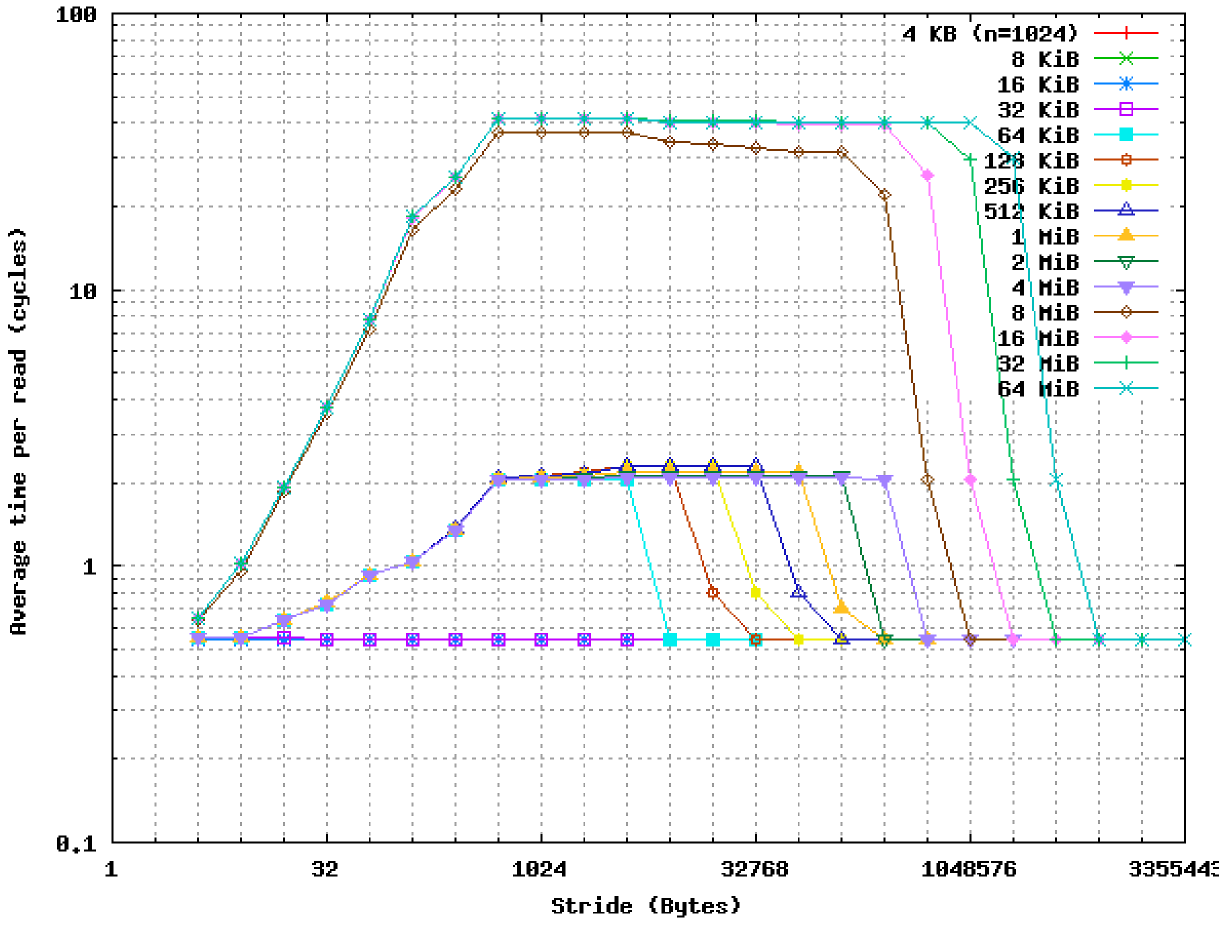
67574625

EECS 117

HW 4

Part 1

Q1. Run the benchmark to collect data for sizes up to 64 MB, i.e., array sizes up to 16777216 (since each element of the array is 4 bytes). Submit both the raw data and a plot of the data. You can use any program (e.g., gnuplot, Excel, MATLAB) to draw the plot.



Q2. From your plot, try to deduce the following:

1. How many levels of cache do there appear to be?

2. For each level of cache, what is its capacity (size)? Line size?

- L1 cache
    - cache size: 32 KiB
    - line size: 64 B
- The second may be the TLB
- The third may be the memory

Again, please submit your answers, raw data, and plot.

Part 2

Q3. Run the program several times and record the smallest execution time. And then, compute the performance in GFLOP/s for the naive kernel using this time.

```
[jmelcher@compute-13-3 hw4]$ ./mm 1024 1024 1024
N: 1024 K: 1024 M: 1024
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Naive matrix multiply
Done
time for naive implementation: 17.1483 seconds
```

Performance of naïve kernel

$$f = 2n^3 = 2(1024)^3 = 2{,}147{,}483{,}648\ Flops = 2.147\ GFlops$$

$$\frac{f}{T} = \frac{2.147}{17.1483} = 0.125\ \frac{GFlops}{s}$$

Q4. Implement a cache-blocked version of matrix-matrix multiply. This is essentially a blocked/tiled implementation where you compute the matrix in smaller sub-block increments. This is shown in the figure below. You can also refer to lecture slides for more information and the psuedo code. In your implementation, make sure that the size of the sub-block can be varied.

The blocked matrix-matrix multiply has been implemented with variable block size.

Q5. Try varying the block size and measure/report the performance of your code in terms of GFLOP/s. Use this information to determine the approximate size of the cache in the system. Justify your reasoning.

The algorithm was tested with matrix size N = M = K = 1024 with block size from 4 to 512
The execution time of the algorithm doesn't reduce with increasing block size past a block size of 16. Therefore, the line size of the cache is 16*4bytes = 64bytes

```
[jmelcher@compute-13-3 hw4]$ ./mm 1024 1024 1024 16
N: 1024 K: 1024 M: 1024 blocksize: 16
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Naive matrix multiply
Done
time for naive implementation: 18.3849 seconds

Cache-blocked matrix multiply
Done
time for cache-blocked implementation: 1.99173 seconds
SUCCESS
SIMD-vectorized Cache-blocked matrix multiply
Done
```

Performance of block kernel

$$f = 2n^3 = 2(1024)^3 = 2{,}147{,}483{,}648\ Flops = 2.147\ GFlops$$

$$\frac{f}{T} = \frac{2.147}{1.99173} = 1.078\ \frac{GFlops}{s}$$

According to the performance of the program, the timing plateaus when the block size is 16. This implies a cache line size of 64 Bytes.

Q6. Implement a SIMD vectorized version of your cache blocked matrix-matrix multiply.

Report the performance of your code in terms of GFLOP/s.

The SIMD vectorized version of blocked matrix-matrix multiply has been implemented with variable block size. It requires a minimum block size of 4

```
[jmelcher@compute-13-3 hw4]$ ./mm 1024 1024 1024 256
N: 1024 K: 1024 M: 1024 blocksize: 256
Timer: gettimeofday
Timer resolution: ~ 1 us (?)
Naive matrix multiply
Done
time for naive implementation: 18.2993 seconds

Cache-blocked matrix multiply
Done
time for cache-blocked implementation: 1.97818 seconds
SUCCESS
```

```
SIMD-vectorized Cache-blocked matrix multiply
Done
time for SIMD-vectorized cache-blocked implementation: 1.05535 seconds
SUCCESS
```

Performance of SIMD-vectorized block kernel

$$f \;=\; 2n^3 = 2(1024)^3 = 2{,}147{,}483{,}648\;Flops = 2.147\;GFlops$$

$$\frac{f}{T} = \frac{2.147}{0.46402} = 4.627\;\frac{GFlops}{s}$$