Pre-Class Questions:

Consider the following naive row-based N x N matmul (matrix multiplication):

```
for (i = 0; i < N; i++){
    for (j = 0; j < N; j++){
        tmp = 0
        for (k = 0; k < N; k++)
            tmp += A[i,k] * B[k,j]
    }
        C[i,j] = tmp
}
```

Suppose data is in double-precsion floating point. We are interested in estimating the memory-based arithmetic intensity (AI) of this code. The memory-based AI is defined that (# flops) / (# bytes transferred between memory and cache), and depends on the cache size. Suppose the cache uses a least-recently-used (LRU) policy for deciding which data to flush when moving something into an already-full cache.

1. Suppose 16N is significantly larger than the size of our L3 cache. What is the memory-based AI of this code? (Hint: What is the memory-based AI of just the innermost loop?)

For the following calculations(question1~3) assume that N>>1 so we roughly make that $N \approx N+1$ and $\frac{1}{N} \approx \frac{1}{N+1}$. Also, we assume each multiplication over double-precision float is 1 flop.

For each inner loop (loop over k), we have to read 2*N numbers into cache, each number is at size 8 Bytes(size of double). And we did N flops. For the following loops, since 16N is significantly larger than the cache, so each loop, all the numbers used for

the loop have been kicked out of memory. So we have to reload it into cache. So the overall AI=N/8*2*N =1/16

Overall, we have to make $O(N^3)$ flops, and we need to write $O(N^2)$ into C. So the write into C is negligible. So AI_1= 1/16

2. Now suppose that the cache is substantially larger than 16N, but substantially smaller than 8N^2. What is the AI now?

For the first inner loop(loop over k), we have to read 2*N numbers into cache. And we did N flops. But from the second loop, we have all the A[i,:] in the memory, we still need to load all the B[:,j] into memory since they have never been loaded. So for each loop (except for the 1st one), we need to move N doubles into the cache. and we does N flops in the inner round. So AI=N/8*N =1/8

Overall, we have to make $O(N^3)$ flops, So we need to do $O(8*N^3)= O(N^3)$ read of A and B, and we need to write $O(N^2)$ into C. So the write into C is negligible. So AI= AI_2=1/8

3. Now suppose the cache is large enough to hold all of A, B, and C. What is the AI now? (Hint: Writing to a byte of memory not already in the cache incurs two memory transfers: one to move the data to the cache for writing, and one to move the written data back to main memory.)

We need to read all numbers into Memory and write the numbers in C. So overall, we need to read $3*N^2$ read and write N^2 numbers, each at size 8 Bytes. So we have AI_3 $=N^3/((3+1)*8*N^2)=N/4$

4. Cache overflowing. On my CPU (Intel i7-4700 HQ), L1, L2, and L3 caches are 32 KB, 256 KB, and 6 MB respectively. What is the largest problem size N that will fit in each cache? What is the arithmetic intensity associated with each problem size?

We need the cache to be at size $3*8*N^2$ So the largest N at size to fit into the L1 cache

is SQRT(1024*32/24)=36

the largest N at size to fit into the L2 cache is SQRT(1024*256/24)=104

the largest N at size to fit into the L3 cache is SQRT(1024*6*1024/24) =512

5. My CPU has 4 cores, each of which can do 8 fused multiply-adds per cycle, has a clock rate of 2.4 GHz, and a memory bandwidth of 25.6 GB/s. At what arithmetic intensity does my machine become CPU-bound?

In 1s, your CPU can do 4cores* 8 flop/(core*cycle) * 2.4 G cycle/s = 76.8 Gflops

So the CPU bound of AI is 76.8 Gflops/25.6GB/s = 3

6. So, for what size range for N will naive matmul be CPU-bound on my machine?

For case in question 1 and 2, the CPU bound will never be reached, So for case in question 3 when N/4>3 which is when N >12, the naive mat-mul will be CPU-bound on your machine. And when N reaches size of 512, the size of problem will exceed the case 3 limit. So it's 3~512.

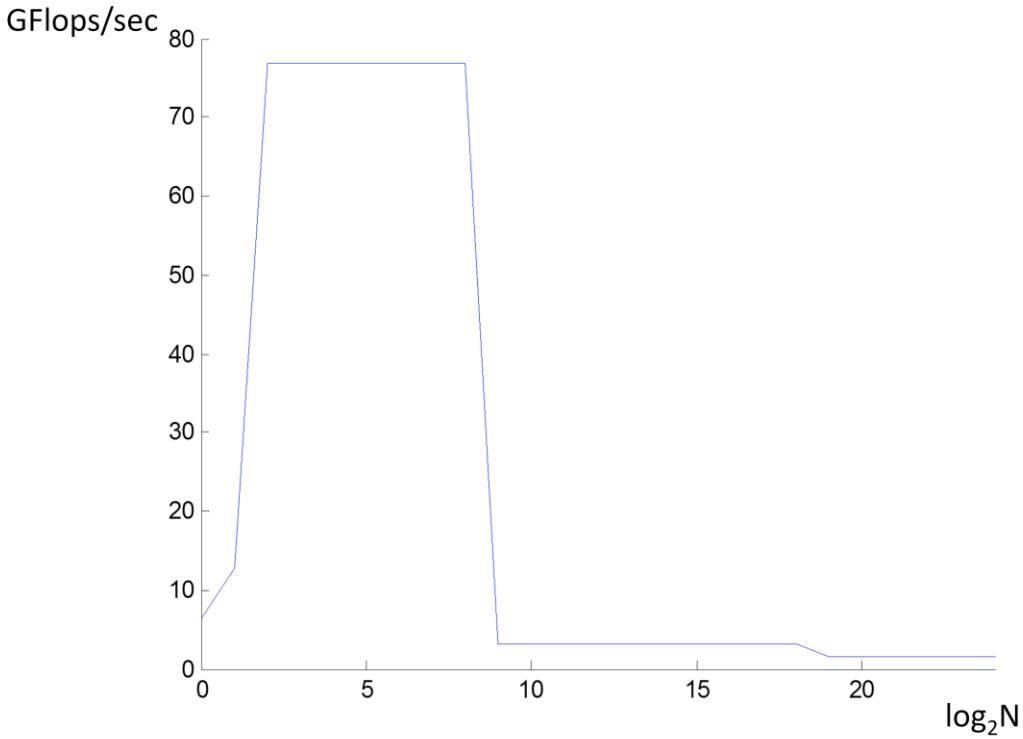
7. So, what will a plot of Flops/sec vs N look like?

(Notice that our assumption $N \approx N+1$ does not hold when N is so small, so there may be a little different from the real situation when N is small)

At the beginning the Flops/sec is bounded by the memory speed, then when N reaches 12, Flops/sec is bounded by 76.8 Gflops/sec.

Then when N reaches 512, it comes to case 2. Then the flops/sec = Memory_Speed* AI_2 = Memory_Speed/8

When N reaches 6M, then flops/sec = Memory_Speed* AI_1 = Memory_Speed/16


GFlops/sec
80
70
60
50
40
30
20
10
0
0
5
10
15
20
log2N