CMPE 297 HW#1

Due: Friday, Sep 23, 1:00pm
Total Score: /100

Instructor: Hyeran Jeon

Computer Engineering Department, San Jose State University

Homework is not a group work. Each student should submit his/her own homework solution.

In this Homework, you will modify matrix multiply code developed in Lab3 to run blocked matrix multiply.

Block-based Matrix Multiplication: In a traditional NxN matrix multiply, all the entries in a row and a column in the two input matrices are accessed to compute one entry value in the output matrix. Then, the accessed row and column in the input matrix needs to be accessed again to compute different entries in the output matrix. If N is small enough, the accessed input data may be found from the cache in the next accesses. But, when N is large, the input data that have accessed might have been already replaced by another data due to insufficient cache size. This can lead to poor cache performance and, in turn, poor overall performance. The advantage of cache memory is predicated on our ability to fit the "working data set" into the cache. Thus, we can break the large matrix (e.g. 12x12) into smaller matrices (e.g. 4x4) and define the matrix multiply operation recursively (i.e. calculate the overall matrix product by calculating the product of smaller "blocks" of the matrix). The corresponding block-base matrix multiply code and illustration is like below:

8,also is block dimension 32,also is matrix size

```
// b = block size, N = matrix dimension
for(i=0; i < N; i=i+b)
    for(j=0; j < N; j=j+b)
        for(k=0; k < N; k=k+b)
            for(ii=i; ii < i+b; ii++)
                for(jj=j; jj < j+b; jj++)
                    for(kk=k; kk < k+b; kk++)
                        C[ii][jj] = C[ii][jj] + A[ii][kk] * B[kk][jj];
```

accumulation of C[ii][jj] by adding different A,B blocks

block calculated result

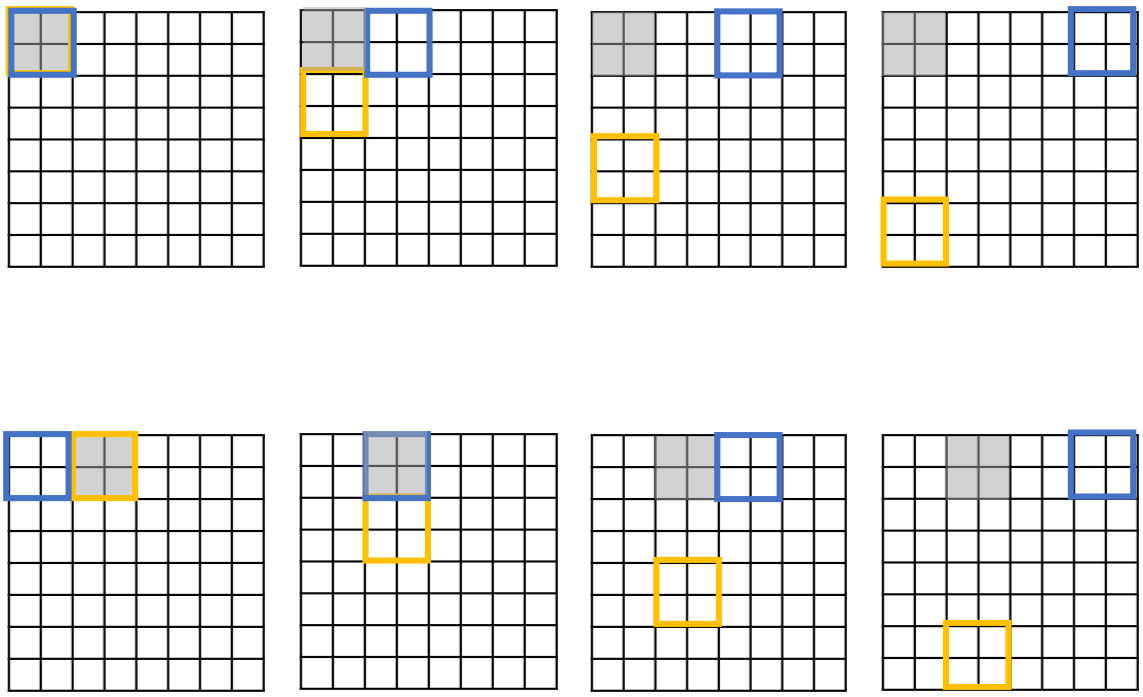

Figure 1 Matrix accesses made in the blocked version when matrix is 8x8 (N = 8) and the block size is 2x2 (b = 2). The gray-colored block is the position of the output block in matrix C that is currently being computed, and the blue-outlined block and the yellow-outlined block show the positions of input blocks in matrix A and B that are used to calculate the gray-colored block value.

Requirements and guidelines:

Blocked matrix multiply in CPU aims to increase cache hit ratio thereby enhancing overall performance. In GPU, we have better memory, *shared memory*, that we don't need to worry about eviction. But, as shared memory size is limited, blocked matrix multiply is an efficient way to do a large matrix multiply in GPU. A large matrix multiplication is partitioned into small block units such that individual block fits well in shared memory. Each block of the given matrix is computed by a thread block. Each thread block may run loop iterations where each iteration fetches a block of data from the input matrices and computes intermediate results until all the blocks are explored. Details of the homework requirement are like below:

1. Use Lab3 code as a baseline.
2. A, B, and C matrices are 32x32 as we used in Lab3.
3. The matrix multiply is computed by 4x4 thread blocks and each thread block is run by 8x8 threads.
4. Each thread block will calculate 8x8 elements in the output matrix C. In other words, each thread will calculate one element in the output matrix.
5. Each thread block will run loop iterations to do block-level matrix multiply, where each iteration
    a. declares two 8x8 matrices in shared memory.
    b. fetches 8x8 elements from matrix A and B in global memory to shared memory.

    c. computes intermediate outputs and accumulates them until all the sub-blocks are explored.
6. Your code should compile and run without an error.
7. The code should generate a correct output; this will be checked by seeing the output message “Test PASSED” as in Lab3.
8. Submit the code to Canvas before the deadline.