EE 352 Lab 3
Spring 2010 Nazarian

Name: ________________________ **Score:**______

Assigned: Thursday, February 11

Due: Sunday, February 21 (Digital submission at 11:59pm)

1 Introduction

In this lab you will write two versions of code to perform an NxN matrix multiply. The first approach is the straightforward triple-nested loop version while the second implements a "blocked" matrix multiply which is a modification of the first method intended to aid performance.

2 What you will learn

This lab is intended to give you greater skill in assembly language programming and translating loops.

3 Background Information and Notes

Matrix Multiplication: Traditional matrix multiplication of an NxN square matrix is achieved by taking the inner/dot product of an entire row and column as shown with the code and figure below:

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

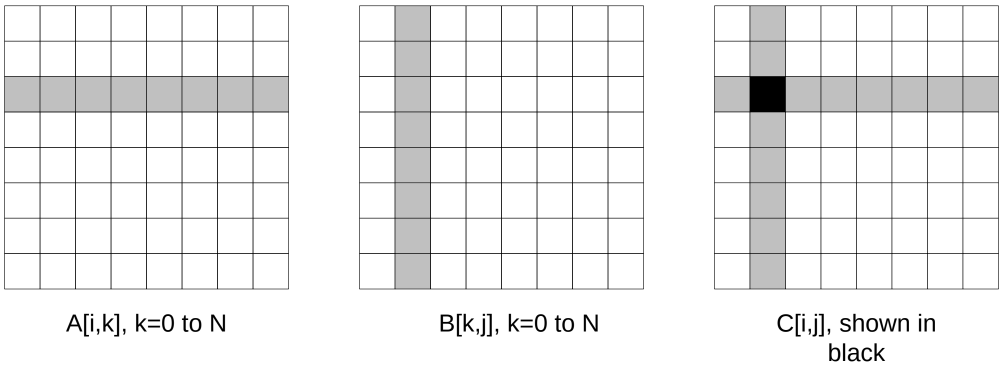
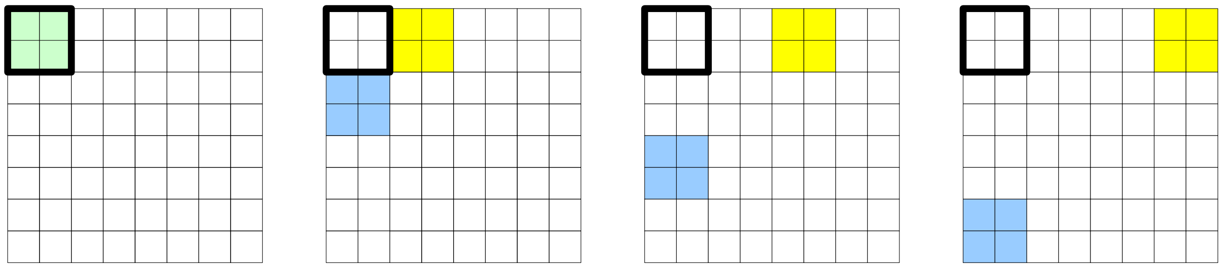


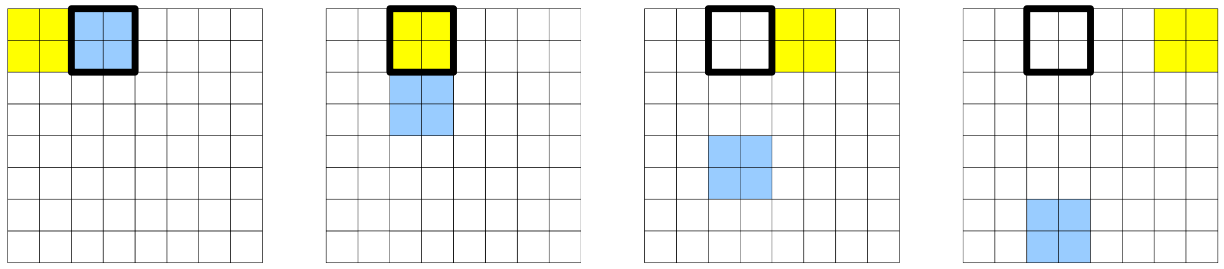
Figure 1 - Matrix accesses made for i,j = 1,1 (K iterating 0 to N-1). Total number of access to matrix A & B is 2N.

When N is large, the number of data points that we need to access is proportional to N. This can lead to poor cache performance and, in turn, poor overall performance. The idea 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. 8x8) into smaller matrices (e.g. 2x2) 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 code for doing this is straightforward though it may require some examination to see exactly what values are being accessed. We have attempted to provide visualization below in addition to the code.

```
// 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];
```



Accesses performed when outer i,j = 0,0
Black border shows C[ii][jj] values being calculated, Yellow = A matrix access, Blue = B Matrix Accesses



Accesses performed when outer i,j = 0,block_size
Black border shows C[ii][jj] values being calculated, Yellow = A matrix access, Blue = B Matrix Accesses

Figure 2 –Matrix accesses made in the blocked version. Each timestep shown is a new value of k.
Total number of access made to each matrix A & B = $2b^2$ where b = block size.

Notice now that in the case of a large matrix (e.g. N = 1024) we would only be accessing small blocks of data in any one iteration of the inner loops. Presumably this data could fit in cache and we would see better performance.