Program Assignment #2
Andrew Jordan
CS415
Sources:
https://en.wikipedia.org/w/index.php?title=Strassen_algorithm&oldid=498910018#Source_code_of_the_Strassen_algorithm_in_C_language%20%20*/
https://www.youtube.com/watch?v=4Sf6ROfsGO4
Purpose for sources:
Wikipedia: Splitting matrix into smaller matrix as defined by the Strassen algorithm.
Youtube link: Understanding mutexes and avoiding data races.

Original content: The Strassen algorithm was modified solely by me to use Pthreads with matrices of powers of 2. The original Strassen algorithm was improved upon using the strategy to break a large matrix into smaller ones and then run 3 calculations simultaneously for a total of 4 threads running at once to maximize efficiency on a 4-core virtual box.
Program notes: No other contributing authors in CS415 on this project.

Times for each data set:
100
real 0m0.012s
user 0m0.008s
sys 0m0.000s

500
real 0m0.345s
user 0m0.556s
sys 0m0.008s

1000
real 0m2.257s
user 0m3.928s
sys 0m0.076s

5000
real 4m30.266s
user 8m39.068s
sys 0m8.804s

10000
real 30m39.118s
user 60m19.408s
sys 0m48.680s

```
//============================================================================
// Name        : pthreadex.cpp
// Author      : Andrew Jordan
// Version     : 20
// Copyright   : CS415 Freeware
// Description : Matrix Multiplication using Pthreads
// command line build with proper flags g++ pthreadx.cpp -o ex1.out -lpthread
// start program from command line: ./ex1.out
//============================================================================
#include <iostream>
#include <algorithm>
```

```
#include <stdio.h>
#include <pthread.h>
#include <string>
#include <unistd.h>
#include <stdlib.h>
using namespace std;

struct data{
   pthread_mutex_t lock;
   int **sub1;
   int **sub2;
   int **sub3;
   int **sub4;
   int **sub5;
   int max;
};

//[row][col]
void test(int);
void printM(int **, int);
void fill(int **arry, int input, int resized);
int pow2(int data);
void addM(int **A,int **B, int **C, int max);
void subM(int **A,int **B, int **C,int max);
void mulM(int **A, int **B, int **C,int max);
void loadargs(data *ptr,int **A, int **B, int **C,int **D,int **E, int max);
int** createM(int); // creates a nXn matrix and returns a pointer
void split(int **A, int **B, int **C,int max); // Recursive function to break matrix into small matrix
void deleteM(int**,int); // delete used 2D matrix
void strassen(int **A,int **B,int **C, int max);
void *M1f(void *args);
void *M2f(void *args);
void *M3f(void *args);
void *M4f(void *args);
void *M5f(void *args);
void *M6f(void *args);
void *M7f(void *args);

data *ptr1;
data *ptr2;
data *ptr3;



int main()
{
   ptr1 = new data;//Points to data for thread 1
   ptr2 = new data;//Points to data for thread 2
   ptr3 = new data;//Points to data for thread 3
   pthread_mutex_init(&ptr1->lock,NULL); //Initialize lock1
   pthread_mutex_init(&ptr2->lock,NULL);// Initialzie lock2
   pthread_mutex_init(&ptr3->lock,NULL);// Initialzie lock3

   test(10000);
   cout << "Finished execution." << endl;
```

```
   pthread_mutex_destroy(&ptr1->lock);
   pthread_mutex_destroy(&ptr2->lock);
   pthread_mutex_destroy(&ptr3->lock);
   delete(ptr1);
   delete(ptr2);
   delete(ptr3);
   return 0;
}
void test(int input)
{
   int resized = pow2(input);
   // 1st matrix
   int **test1 = createM(resized);
   fill(test1, input - 1, resized);
   // 2nd matrix
   int **test2 = createM(resized);
   fill(test2, input - 1, resized);
   // Resulting matrix
   int **result = createM(resized);

   //strassen(test1,test2,result,resized);

   split(test1, test2, result, resized);
   //printM(result,resized);
   deleteM(result,resized);
   deleteM(test2,resized);
   deleteM(test1,resized);
}
void printM(int **matrix, int max)
{
   //[row][col]
   for (int i = 0; i < max; i++)
      for (int j = 0; j < max; j++)
      {
      cout << matrix[i][j] << " ";
      if ((j + 1) == max)
          cout << endl;
      }
}
void fill(int **arry, int input, int resized)
{
   for (int i = 0; i < resized; i++)
      for (int j = 0; j < resized; j++)
      {
      if (i > input || j>input)
          arry[i][j] = 0;
      else
          arry[i][j] = (rand() % 5) + 1;
      }
}
int pow2(int data)
{
   if (data%4 == 0)
      return data;
   else
   {
```

```
      int i = 2;
      while (i < data)
         i = i * 2;
      return i;
   }
}
void addM(int **A, int **B, int **C, int max)
{
   for (int i = 0; i < max; i++)
      for (int j = 0; j < max; j++)
         C[i][j] = A[i][j] + B[i][j];
}
void subM(int **A, int **B, int **C, int max)
{
   for (int i = 0; i < max; i++)
      for (int j = 0; j < max; j++)
         C[i][j] = A[i][j] - B[i][j];
}
void mulM(int **A, int **B, int **C, int max)
{
   for (int i = 0; i < max; i++)
      for (int k = 0; k < max; k++)
         for (int j = 0; j < max; j++)
            C[i][j] += (A[i][k] * B[k][j]);
}
int** createM(int msize)
{

   int **temp = new int*[msize];
   for (int i = 0; i < msize; i++)
   {
      temp[i] = new int[msize];
      for (int j = 0; j < msize; j++)
         temp[i][j] = 0;
   }
   return temp;
}
void split(int **A, int **B, int **C, int max)
{
   if (((max % 2) > 0) || ((max/2) < 64))
      strassen( A, B, C, max);
   else
   {
      int half = max / 2;
      int **sub1 = createM(half);
      int **sub2 = createM(half);

      int **A11 = createM(half);
      int **A12 = createM(half);
      int **A21 = createM(half);
      int **A22 = createM(half);

      int **B11 = createM(half);
      int **B12 = createM(half);
      int **B21 = createM(half);
      int **B22 = createM(half);
```

```
//[row][column]

for (int i = 0; i < half; i++)
    for (int j = 0; j < half; j++)
    {
    A11[i][j] = A[i][j];
    A12[i][j] = A[i][j + half];
    A21[i][j] = A[i + half][j];
    A22[i][j] = A[i + half][j + half];

    B11[i][j] = B[i][j];
    B12[i][j] = B[i][j + half];
    B21[i][j] = B[i + half][j];
    B22[i][j] = B[i + half][j + half];
    }//end for spliting matrix


//M1
int **M1 = createM(half);
addM(A11,A22,sub1,half);
addM(B11, B22, sub2, half);
// (A11 + A22) * (B11 + B22)
split(sub1, sub2, M1, half);

//M2
int **M2 = createM(half);
addM(A21, A22, sub1, half);
split(sub1, B11, M2, half);

//M3
int **M3 = createM(half);
subM(B12, B22, sub1, half);
// A11*(B12 - B22)
split(A11, sub1, M3, half);

//M4
int **M4 = createM(half);
subM(B21, B11, sub1, half);
split(A22, sub1, M4, half);

//M5
int **M5 = createM(half);
addM(A12, A11, sub1, half);
// (A12 + A11)*B22
split(sub1, B22, M5, half);

//M6
int **M6 = createM(half);
subM(A21, A11, sub1, half);
addM(B12, B11, sub2, half);
// (A21 - A11)*(B12+B11)
split(sub1, sub2, M6, half);

//M7
int **M7 = createM(half);
```

```
subM(A12, A22, sub1, half);
addM(B21, B22, sub2, half);
//(A12 -A22)*(B21+B22)
split(sub1, sub2, M7, half);

//Delete used matrix
deleteM(A11,half);
deleteM(A12,half);
deleteM(A21,half);
deleteM(A22,half);
deleteM(B11,half);
deleteM(B12,half);
deleteM(B21,half);
deleteM(B22,half);

int **C11 = createM(half);
int **C12 = createM(half);
int **C21 = createM(half);
int **C22 = createM(half);


// C11 ->(M1 +M7)+(M4-M5)
addM(M1, M7, sub1, half);
subM(M4, M5, sub2, half);
addM(sub1, sub2, C11, half);

//C12 -> M3 + M5
addM(M3, M5, C12, half);
//C21 -> M2 + M4
addM(M2, M4, C21, half);
//C22 -> (M1 - M2) + (M3 + M6)
subM(M1, M2, sub1, half);
addM(M3, M6, sub2, half);
addM(sub1, sub2, C22, half);

for (int i = 0; i<half; i++)
    for (int j = 0; j<half; j++)
    {
        C[i][j] = C11[i][j];
        C[i][j + half] = C12[i][j];
        C[i + half][j] = C21[i][j];
        C[i + half][j + half] = C22[i][j];
    }

// Delete temp matrix M1 - M7
deleteM(M1,half);
deleteM(M2,half);
deleteM(M3,half);
deleteM(M4,half);
deleteM(M5,half);
deleteM(M6,half);
deleteM(M7,half);
// Delete temporary sub matricies
deleteM(sub1,half);
deleteM(sub2,half);
// Delete Sub Matrix C11,C12,C21,C22
```

```
    deleteM(C11,half);
    deleteM(C12,half);
    deleteM(C21,half);
    deleteM(C22,half);
  }

}//end void split
void deleteM(int **data,int S)
{
  for (int i = 0; i < S; i++)
    delete[] data[i];
  delete[] data;
  data = nullptr;
}
void loadargs(data *ptr,int **A, int **B, int **C,int **D,int **E, int max)
{
  pthread_mutex_lock(&ptr->lock);
  ptr->sub1 = A;
  ptr->sub2 = B;
  ptr->sub3 = C;
  ptr->sub4 = D;
  ptr->sub5 = E;
  ptr->max = max;
}
void strassen(int **A,int **B,int **C, int max)
{
        pthread_t Mpool[7]; //Number of threads used
        int half = max / 2;

        int **null = NULL;
        int **A11 = createM(half);
        int **A12 = createM(half);
        int **A21 = createM(half);
        int **A22 = createM(half);

        int **B11 = createM(half);
        int **B12 = createM(half);
        int **B21 = createM(half);
        int **B22 = createM(half);
        // Copies for threads
        int **A11copy = createM(half);
        int **A22copy = createM(half);
        int **B11copy = createM(half);
        int **B22copy = createM(half);

        int **M1 = createM(half);
        int **M2 = createM(half);
        int **M3 = createM(half);
        int **M4 = createM(half);
        int **M5 = createM(half);
        int **M6 = createM(half);
        int **M7 = createM(half);


        //[row][column]
        for (int i = 0; i < half; i++)
```

```
    for (int j = 0; j < half; j++)
    {
        A11[i][j] = A[i][j];
        A11copy[i][j]= A[i][j];
        A12[i][j] = A[i][j + half];
        A21[i][j] = A[i + half][j];
        A22[i][j] = A[i + half][j + half];
        A22copy[i][j] = A[i + half][j + half];

        B11[i][j] = B[i][j];
        B11copy[i][j]= B[i][j];
        B12[i][j] = B[i][j + half];
        B21[i][j] = B[i + half][j];
        B22[i][j] = B[i + half][j + half];
        B22copy[i][j] = B[i + half][j + half];
    }//end for splitting matrix


//M1
loadargs(ptr1,A11,A22,B11,B22,M1,half);
pthread_create(&Mpool[0],NULL,&M1f,ptr1);
//M2
loadargs(ptr2,A21,A22copy,B11copy,M2,null,half);
pthread_create(&Mpool[1],NULL,&M2f,ptr2);
//M3
loadargs(ptr3,A11copy,B12,B22copy,M3,null,half);
pthread_create(&Mpool[2],NULL,&M3f,ptr3);

// Wait for M1,M2,M3 to finish
pthread_join(Mpool[0],NULL);
pthread_join(Mpool[1],NULL);
pthread_join(Mpool[2],NULL);

//M4
loadargs(ptr1,A22,B21,B11,M4,null,half);
pthread_create(&Mpool[3],NULL,&M4f,ptr1);
//M5
loadargs(ptr2,A11,A12,B22,M5,null,half);
pthread_create(&Mpool[4],NULL,&M5f,ptr2);
//M6
loadargs(ptr3,A21,A11copy,B11copy,B12,M6,half);
pthread_create(&Mpool[5],NULL,&M6f,ptr3);

// Wait for M4,M5,M6 to finish
pthread_join(Mpool[3],NULL);
pthread_join(Mpool[4],NULL);
pthread_join(Mpool[5],NULL);
//M7
loadargs(ptr1,A12,A22,B21,B22,M7,half);
pthread_create(&Mpool[6],NULL,&M7f,ptr1);

// Can't delete A12,A22,B21,B22 yet
// Delete copies
deleteM(A11copy,half);
deleteM(A22copy,half);
deleteM(B11copy,half);
```

```
deleteM(B22copy,half);
// Delete unused matrix by M7
deleteM(A11,half);
deleteM(A21,half);
deleteM(B11,half);
deleteM(B12,half);
// Wait for M7 to finish
pthread_join(Mpool[6],NULL);
// Delete matrix used to get M7
deleteM(A12,half);
deleteM(A22,half);
deleteM(B21,half);
deleteM(B22,half);

int **C11 = createM(half);
int **C12 = createM(half);
int **C21 = createM(half);
int **C22 = createM(half);
int **sub1 = createM(half);
int **sub2 = createM(half);

// C11 ->(M1 +M7)+(M4-M5)
addM(M1, M7, sub1, half);
subM(M4, M5, sub2, half);
addM(sub1, sub2, C11, half);
//C12 -> M3 + M5
addM(M3, M5, C12, half);
//C21 -> M2 + M4
addM(M2, M4, C21, half);
//C22 -> (M1 - M2) + (M3 + M6)
subM(M1, M2, sub1, half);
addM(M3, M6, sub2, half);
addM(sub1, sub2, C22, half);
for (int i = 0; i<half; i++)
    for (int j = 0; j<half; j++)
        {
            C[i][j] = C11[i][j];
            C[i][j + half] = C12[i][j];
            C[i + half][j] = C21[i][j];
            C[i + half][j + half] = C22[i][j];
        }
// Delete temp matrix M1 - M7
deleteM(M1,half);
deleteM(M2,half);
deleteM(M3,half);
deleteM(M4,half);
deleteM(M5,half);
deleteM(M6,half);
deleteM(M7,half);
// Delete temporary C partitions
deleteM(C11,half);
deleteM(C12,half);
deleteM(C21,half);
deleteM(C22,half);
deleteM(sub2,half);
deleteM(sub1,half);
```

```
}//end strassen
void *M1f(void *args)
{
   /*
    * Data in struct:
    * sub1 = A11
    * sub2 = A22
    * sub3 = B11
    * sub4 = B22
    * sub5 = M1
    */
   data *ptr = (data*)args;
   int max = ptr->max;
   int **temp1 = createM(max);
   int **temp2 = createM(max);
   addM(ptr->sub1,ptr->sub2,temp1,max);
   addM(ptr->sub3,ptr->sub4,temp2,max);
   mulM(temp1,temp2,ptr->sub5,max);
   // Clean up
   deleteM(temp1,max);
   deleteM(temp2,max);
   // Leave M1
   pthread_mutex_unlock(&ptr->lock);
   pthread_exit(NULL);
}
void *M2f(void *args)
{
   /*
    * Data in struct:
    * sub1 = A21
    * sub2 = A22
    * sub3 = B11
    * sub4 = M2
    * sub5 = NULL
    */
   data *ptr = (data*)args;
   int max = ptr->max;
   int **temp1 = createM(max);
   addM(ptr->sub1,ptr->sub2,temp1,max);
   mulM(temp1,ptr->sub3,ptr->sub4,max);
   // Clean up
   deleteM(temp1,max);
   // Leave M2
   pthread_mutex_unlock(&ptr->lock);
   pthread_exit(NULL);
}
void *M3f(void *args)
{
   /*
    * Data in struct:
    * sub1 = A11
    * sub2 = B12
    * sub3 = B22
    * sub4 = M3
    * sub5 = NULL
```

```
    * int max = half
    */
   data *ptr = (data*)args;
   int max = ptr->max;
   int **temp1 = createM(max);
   subM(ptr->sub2,ptr->sub3,temp1,max);
   mulM(ptr->sub1,temp1,ptr->sub4,max);
   // Clean up
   deleteM(temp1,max);
   //delete(ptr);
   // Leave M2
   pthread_mutex_unlock(&ptr->lock);
   pthread_exit(NULL);
}
void *M4f(void *args)
{
   /*
    * Data in struct:
    * sub1 = A22
    * sub2 = B21
    * sub3 = B11
    * sub4 = M4
    * sub5 = NULL
    * int max = half
    */
   data *ptr = (data*)args;
   int max = ptr->max;
   int **temp1 = createM(max);
   subM(ptr->sub2,ptr->sub3,temp1,max);
   mulM(ptr->sub1,temp1,ptr->sub4,max);
   // Clean up
   deleteM(temp1,max);
   // Leave M2
   pthread_mutex_unlock(&ptr->lock);
   pthread_exit(NULL);
}
void *M5f(void *args)
{
   /*
    * Data in struct:
    * sub1 = A11
    * sub2 = A12
    * sub3 = B22
    * sub4 = M5
    * sub5 = NULL
    * int max = half
    */
   data *ptr = (data*)args;
   int max = ptr->max;
   int **temp1 = createM(max);
   addM(ptr->sub1,ptr->sub2,temp1,max);
   mulM(temp1,ptr->sub3,ptr->sub4,max);
   // Clean up
   deleteM(temp1,max);
   // Leave M2
   pthread_mutex_unlock(&ptr->lock);
```

```
    pthread_exit(NULL);
}
void *M6f(void *args)
{
    /*
     * Data in struct:
     * sub1 = A21
     * sub2 = A11
     * sub3 = B11
     * sub4 = B12
     * sub5 = M6
     * int max = half
     */
    data *ptr = (data*)args;
    int max = ptr->max;
    int **temp1 = createM(max);
    int **temp2 = createM(max);
    subM(ptr->sub1,ptr->sub2,temp1,max);
    addM(ptr->sub3,ptr->sub4,temp2,max);
    mulM(temp1,temp2,ptr->sub5,max);
    // Clean up
    deleteM(temp1,max);
    deleteM(temp2,max);
    // Leave M1
    pthread_mutex_unlock(&ptr->lock);
    pthread_exit(NULL);
}
void *M7f(void *args)
{
    /*
     * Data in struct:
     * sub1 = A12
     * sub2 = A22
     * sub3 = B21
     * sub4 = B22
     * sub5 = M6
     * int max = half
     */
    data *ptr = (data*)args;
    int max = ptr->max;
    int **temp1 = createM(max);
    int **temp2 = createM(max);
    subM(ptr->sub1,ptr->sub2,temp1,max);
    addM(ptr->sub3,ptr->sub4,temp2,max);
    mulM(temp1,temp2,ptr->sub5,max);
    // Clean up
    deleteM(temp1,max);
    deleteM(temp2,max);
    // Leave M1
    pthread_mutex_unlock(&ptr->lock);
    pthread_exit(NULL);
}
```