```
LAB 1 Name: Liam Lefferts

ljcates@127x28 ~/419/Labs/HW1 pThreads MM $MM
Starting Computations...
Matrices Generated
Secs 2 Threaded =       0.398
Secs 4 Threaded =       0.209
Secs 8 Threaded =       0.121
Secs Serial =           0.754
Matrices match...
SUCCESSFUL!
Matrices Free...

Matrices Generated
Secs 2 Threaded =       2.681
Secs 4 Threaded =       1.423
Secs 8 Threaded =       0.829
Secs Serial =           5.092
Matrices match...
SUCCESSFUL!
Matrices Free...

Matrices Generated
Secs 2 Threaded =       2055.315
Secs 4 Threaded =       970.784
Secs 8 Threaded =       513.328
Secs Serial =           >= 1hour    ssh timeout
```

| Square Matrix Size | Execution Time |
|---|---|
| 512 | 0.754 |
| 1024 | 5.092 |
| 8192 | >=1hour ssh timeout |

| Square Matrix Size | 2 Thread Exec Time | 4 Thread Exec Time | 8 Thread Exec Time |
|---|---|---|---|
| 512 | 0.398 | 0.209 | 0.121 |
| 1024 | 2.681 | 1.423 | 0.829 |
| 8192 | 2055.315 | 970.784 | 513.328 |

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <sys/time.h>
#include <time.h>
#include <pthread.h>

/* global variable accessible to all threads */
int SIZE, NTHREADS;
int **A, **B, **C, **D;

/* dtime -
 * utility routine to return the current wall clock time
 */
double
dtime()
{
   double tseconds = 0.0;
   struct timeval mytime;
   gettimeofday(&mytime,(struct timezone*)0);
   tseconds = (double)(mytime.tv_sec + mytime.tv_usec*1.0e-6);
   return( tseconds );
}


void
matrixInitialization()
{
   int i, j;
   time_t t;

   A = (int**)malloc(SIZE * sizeof(int *));
   for(i = 0; i < SIZE; i++)
      A[i] = malloc(SIZE * sizeof(int));

   B = (int**)malloc(SIZE * sizeof(int *));
   for(i = 0; i < SIZE; i++)
      B[i] = malloc(SIZE * sizeof(int));

   C = (int**)malloc(SIZE * sizeof(int *));
   for(i = 0; i < SIZE; i++)
      C[i] = malloc(SIZE * sizeof(int));

   D = (int**)malloc(SIZE * sizeof(int *));
   for(i = 0; i < SIZE; i++)
      D[i] = malloc(SIZE * sizeof(int));
```

```
   srand(time(&t));

   for(i = 0; i < SIZE; i++) {
      for(j = 0; j < SIZE; j++) {
         A[i][j] = rand()%100;
         B[i][j] = rand()%100;
      }
   }

   return;
}

/* printMatrixInitialization -
 * Print global variable matrix results to stdout
 */
void
printMatrixInitialization()
{
   matrixInitialization();

   printf("Matrices Generated\r\n");

   return;
}


/* matrixmultNT -
 * Matrix multiplication non-threaded
 */
void
matrixmultNT(){
   int c, d, k, sum;
   for (c = 0; c < SIZE; c++) {
      for (d = 0; d < SIZE; d++) {
         sum = 0;
         for (k = 0; k < SIZE; k++)
            sum += A[c][k]*B[k][d];

         C[c][d] = sum;
      }
   }
}

/* printNonThreadedMM -
 * Print non threaded matrix multiplication results to stdout
 */
void
printNonThreadedMM()
```

```
{
   double tstart, tstop, ttime;

   /* measure current system time */
   tstart = dtime();

   /* call non threaded matrix multiplication */
   matrixmultNT();

   /* measure new system time */
   tstop = dtime();

   /* measure system time difference */
   ttime = tstop - tstart;

   if ((ttime) > 0.0)
      printf("\nSecs Serial = %10.3lf\n",ttime);

   return;
}


/* Matrix multiplication threaded
 *
 * Takes: void*
 *
 * Returns: void
 */
void*
matrixmultT(void* id){

   int i, j, k, sum;
   int tid = (intptr_t)id;
   int start = tid * SIZE/NTHREADS;
   int end = (tid+1) * (SIZE/NTHREADS) - 1;

   for(i = start; i <= end; i++) {
      for(j = 0; j < SIZE; j++) {
         sum = 0;
         for(k = 0; k < SIZE; k++)
            sum += A[i][k] * B[k][j];

         D[i][j] = sum;
      }
   }
}

/* Print non threaded matrix multiplication results to stdout
```

```
 *
 * Takes: void
 *
 * Returns: void
 */
void
printThreadedMM()
{
   int i,j;
   double tstart, tstop, ttime;
   pthread_t* threads;



   for(NTHREADS = 2; NTHREADS <= 8; NTHREADS*=2)
   {
      threads = (pthread_t*)malloc(NTHREADS * sizeof(pthread_t));

   /* measure current system time */
      tstart = dtime();

   /* call non threaded matrix multiplication */
      for(i = 0; i < NTHREADS; i++)
         pthread_create(&threads[i], NULL, matrixmultT, (void *)(intptr_t)i);

      for(i = 0; i < NTHREADS; i++)
         pthread_join(threads[i], NULL);

   /* measure new system time */
      tstop = dtime();

   /* measure system time difference */
      ttime = tstop - tstart;

      ttime=tstop-tstart;

      if ((ttime) > 0.0)
         printf("Secs Threaded = %10.3lf\n", ttime);

      free(threads);
   }

   return;
}


/* printMatrices
 * Prints matrices to stdout
```

```
 *
 *
 * warning: use low matrix dimensions
 *_/
void
printMatrices()
{
     for (i=0; i<SIZE; i++) {
     for(j=0; j<SIZE; j++)
     printf("%d, %d ", C[i][j], D[i][j]);
     printf("\n");

     return;
}

/* printMatricesCheck
 * Prints matrices check status to stdout
 *
 * Takes: void
 *
 * Returns: void
 */
void
printMatricesCheck()
{
   int i, j;
   float dif;

   for (i=0; i<SIZE; i++){
      for(j=0; j<SIZE; j++) {
         dif=abs(C[i][j]-D[i][j]);
         if(dif!=0)
         {
            printf("FAILED\n");
            return;
         }
      }
   }

   printf("Matrices match...\n");
   printf("\nSUCCESSFUL!\n");

   return;
}

/* printMatrixCleanUp
 * Prints matrices clean up status to stdout
 *
```

```
 * Takes: void
 *
 * Returns: void
 */
void
printMatrixCleanUp()
{
   int i, j;

   for(i = 0; i < SIZE; i++)
      free((void *)A[i]);
   free((void *)A);

   for(i = 0; i < SIZE; i++)
      free((void *)B[i]);
   free((void *)B);

   for(i = 0; i < SIZE; i++)
      free((void *)C[i]);
   free((void *)C);

   for(i = 0; i < SIZE; i++)
      free((void *)D[i]);
   free((void *)D);

   printf("Matrices Free...\r\n");
   return;
}

/* runComputation
 * Manages individual matrix operations
 * initialization,
 * nonthreaded and threaded computations
 * and matrix clean up
 *
 * Takes: int Size of Square Matrix
 *
 * Returns: void
 */
void
runComputation(int size)
{
   SIZE = size;
   printMatrixInitialization();

   printThreadedMM();
   printNonThreadedMM();
```

```
   printMatricesCheck();
   printMatrixCleanUp();

   return;
}

int
 main(int argc, char *argv[] )
{
   if(argc != 1)
   {
      printf("Usage: %s", argv[0]);
      exit(1);
   }

   printf("Starting Computations...\r\n");
   runComputation(512);
   runComputation(1024);
   runComputation(2048);
   runComputation(4096);
   runComputation(8192);
   printf("Computations Complete... \r\n");

   return(0);
}
```