Dept. of Computer Science and Engineering,

CS69011: Computing Lab 1

Assignment 7

Maximum Marks: 30

In this assignment, you will write multi-threaded programs using the Unix POSIX threads Library: Pthreads (https://en.wikipedia.org/wiki/POSIX_Threads). You will be using c programming for the same. For a tutorial on pthreads programming using c, see: https://computing.llnl.gov/tutorials/pthreads/

Task 1

For this assignment, you will be writing multi-threaded programs for matrix-multiplication. Let A and B be two input matrices of sizes $r_1 \times c_1$ and $r_2 \times c_2$ respectively, where $c_1 = r_2$. You will be implementing the naïve matrix multiplication algorithm, where each element of the result matrix C of size $r_1 \times c_2$ can be computed independently from one another. The overall scheme is:

- Create a function `void *mult(void *arg)`: which multiplies a row of matrix A with a column of matrix B, and stores the result in the appropriate cell of preallocated matrix C.
- The argument `arg` is a structure of type `ThreadData`, which stores the input and output parameters for the function shown below. Note that the actual matrix data is stored in the memory of the process and is accessible to all threads. Hence, it need not be copied.
- In the main program, create threads to compute all the elements of the matrix C.

The structure for passing parameters can be:

```
// compute A[i,:] * B[:,j]  and store in C[i,j]
typedef struct _thread_data {
     double **A;
     double **B;
     double **C;
     int veclen,i,j;
} ThreadData;
```

Implementation 1:

The first implementation should simply create $r_1 \times c_2$ threads each for computing each element and the thread scheduler takes care of scheduling the threads to actual physical cores.

Note that no mutual exclusion is needed as all the output entries are created independently and all the inputs are already available.

Implementation 2:

The problem with the above implementation is that you have to allocate memory for $r_1 \times c_2$, threads objects and input arguments. However, in most practical systems you will not have enough cores to execute all the threads concurrently. In the second implementation, you should create a thread pool of `nthreads` threads, which is an input (can be set to the number of cores). The main thread should use the pool of threads for computing each value of the result matrix. Below are the pseudo codes for main thread and worker threads:

```
Main thread:

create nthread-many worker threads

for each result value C[i,j]:
     wait for free thread
     calculate target thread
     populate thread data
     signal target thread

signal all threads to finish


Worker Thread:

while (1):
     wait for signal from main thread
     read thread data
     if the signal from main thread is to finish
          exit thread
     perform computation
     signal free to main thread
```

You should use condition variables, with appropriate mutexes for signalling between threads.

Implementation 3:

In the above implementation, the master thread starts creating a thread data after it receives a signal from one of the free threads. It might be more efficient to have master thread and worker threads work concurrently. In the third implementation, create and maintain a buffer of jobs to be completed. The problem is similar to producer consumer problem, where the master is producer and worker threads are consumers. The master keeps producing jobs till the buffer is full, and then waits for a signal from workers till the buffer is empty. It signals all workers when there is a job in the buffer. It also signals the workers to exit when all the jobs are done, and then terminates. The workers keep consuming jobs from the buffer (and calculating the results) till the buffer is empty. If the buffer is empty, it signals the master and then waits for signal from the master. If the signal from the master is to exit, it then terminates.

Hint: here you need to use two condition variables: `empty` and `not_empty`, with appropriate mutexes.

Input:

The final program should take the A and B matrices as input from a file with the following format:

```
<# rows 1>    <# columns 1>
<row 1 of matrix 1 (space separated)>
…
<row m of matrix 1 (space separated)>
<blank line>
<# rows 2>    <# columns 2>
<row 1 of matrix 2 (space separated)>
…
<row n of matrix 2 (space separated)>
```

Here m and n are number of rows of matrix 1 and matrix 2 respectively. Filename is given as command line argument.

Submission:

Submit 3 c source files for: implementation 1, implementation 2, and implementation 3, respectively. Write your name and roll number in a comment at the top of the file along with compilation instructions. Also submit a report comparing the performance of all three implementations. Specifically submit the following plots:

a. Time taken as a function of Matrix size for all the implementations.
b. Time taken as a function of number of worker threads, for multiplying two 10000 x 10000 matrices.