FALL 2018 CSE 420

Project 4

Due Date: November 29

In this project, you are expected to parallelize matrix multiplication using multi-threading and to analyze the performance achieved as compared to the sequential execution.

Matrix Multiplication Program [2 points]

Step 1: First declare 3 single dimension arrays to represent matrices `A`, `B` and `C`. Size is of each array is of n^2 elements; `n` is a user input. Initialize A and B with random values; a random value is in the range [0, 32767]. Initialize C with zeros. You can also optionally read the values from some file, if you wish to.

Note that this initialization is 1-time for entire program and we do not consider time spent on this step while calculating speedup of a parallel program over a sequential one.

Step 2: Write a program to compute matrix multiplication (sequentially) as follows:

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

Measure the time taken by sequential version of matrix multiplication. You will need it later to compute the speedup achieved by parallelizing the matrix multiplication problem.

Note: You can measure the time taken for a part of the code by using `gettime()` function.

Step 3: Now, write a program with POSIX threads to parallelize the matrix multiplication problem. Your parallel version should operate on the same set of values for A, B and C as your sequential version. Your program should be generalized enough to support different number of P-threads. The number of threads (`t`) to be created is a user argument.

Measure the time taken by parallelizing your program with `t` threads.

Note: You can start step 3 with first parallelizing the matrix multiplication using 2 threads and then can extend it to arbitrary `t` threads.

Step 4: Report the speedup achieved. You can calculate speedup (`s`) as –

$$S = \frac{time\ taken\ by\ sequential\ execution\ of\ matrix\ multiplication\ (step\ 2)}{}$$

$$\textit{time taken by parallel matrix mutliplication problem } (step\ 3)$$

Step 5: To ensure that your parallelization strategy is performing the functionality correctly, compare the output obtained by parallel execution of matrix multiplication (step 3) to the output generated by sequential version (step 2). If outputs do not match, your code should throw an error.

Question 1 [3 points]

Evaluate your parallel matrix multiplication with 2–8 threads and plot the speedup obtained when 2, 4, 6 and 8 threads (`t`) are invoked. You should plot speedup as a function of total number of threads utilized. So, X-axis should represent a total number of threads and Y-axis should represent the speedup obtained (`s`). For the plot of speedup (`s`) vs. threads (`t`), vary the dimension (`n`) as 256, 512, 1024, 2048, and, 4096. Also, tabulate your results.

Note: You can plot the results altogether as a line graph for better visualization.

Analyze the graph that you have plotted. Is speedup linear with the total number of threads? Present your hypothesis about why (or why not) that is the case?

Question 2 [1 points]

To confirm (or to disprove) your hypothesis, measure the amount of time each thread (out of total `t` threads) requires completing its work. You can measure timing for each thread by inserting respective timing code at the beginning and the end of the worker thread function.

Present your analysis by taking 1 scenario (e.g. `n` = 1024, `t` = 8). Measure how much time each thread took to finish the assigned task and then discuss the scenario. How well it fits with your hypothesis (that you presented in problem 1)? Did all threads execute for the same time?

Question 3 [1 point]

What is the best speedup you obtained throughout all invocations? What was the dimension of the matrix (`n`) and total number of threads used (`t`) for that case? Why the speedup achieved for this (`n,t`) combination outperformed speedup achieved elsewhere? Briefly discuss your view on this.

Question 4 [1 point]

Think of other optimization techniques to improve the performance of your parallel version of the matrix multiplication. Briefly discuss them and describe how they would change the parallelization achieved. You can modify the source code and can try out these alternatives, if you wish to.

(For example, one way could be to deal with 2 loops instead a loop nest of 3. Alternatively, you could have operated on 2-D arrays of A, B and C. Or, you could have divided your work in a different optimized fashion than what you opted in step 3)

Resources for Getting Started:

1) Lawrence Livermore National Lab Pthreads programming tutorial
2) IBM developerWorks: POSIX threads explained
3) http://pages.cs.wisc.edu/~travitch/pthreads_primer.html

Note: This project requires to deal with up to 8 threads. So, if your machine does not support up to 8 threads then, you may access the machines located in CIDSE computing lab (BYENG 214) where, you can find machines with necessary compute power.

Project Deliverables

1. Source code of your implementation. You will submit a single program (in C/C++) which runs both the sequential and parallel version of the matrix multiplication problem. User should provide 2 inputs: i) dimension `(n)` and ii) total number of threads `(t)`. Then, your program should report the speedup obtained.
   Please comment well your code.

2. Write your answers to the questions, and briefly explain required details. Plot needed graphs and/or tables. Submit single Word file containing all the answers/plots/tables.

Submission Instructions

1. Submit a word document and a zip file and name it as Project4.zip.
   a The word file will contain all the answers.
   This zip file should contain a directory with needed source code inside it. This directory should have a ReadMe.txt file which explains how to run your program. This will help us to reproduce the results that you reported.

2. If you are doing project in a group, only one group member should make the submission through Canvas.

3. There should be no name or ASU ID number of any of the team members in any part of the zip file or program for the project submission. If a team violates this policy, the score for the project will be ZERO. In short, the submission must be anonymous for a double blinded scoring process.