Department of Computer Engineering, Faculty of Engineering,
University of Peradeniya

CO524: Parallel Computers and Algorithms - 2022

Lab 1 – Introduction to Parallel Programming in OpenMP

1.1 What is Parallel Computing?

Most people are familiar with serial computing, even if they don't realize that is what it's called! Most programs that people write and run day to day are serial programs. A serial program runs on a single computer, typically on a single processor. The instructions in the program are executed one after the other, in series, and only one instruction is executed at a time. Parallel computing is a form of computation that allows many instructions in a program to run simultaneously, and in parallel. In order to achieve this, a program must be split up into independent parts so that each processor can execute its part of the program simultaneously with the other processors. Parallel computing can be achieved on a single computer with multiple processors, a number of individual computers connected by a network, or a combination of the two.

1.2 OpenMP

OpenMP (Open Multi-Processing) was first released in 1997 and is a standard Application Programming Interface (API) for writing shared memory parallel applications in C, C++, and Fortran. OpenMP has the advantages of being very easy to implement on currently existing serial codes and allowing incremental parallelization. It also has the advantages of being widely used, highly portable, and ideally suited to multi-core architectures (which are becoming increasingly popular in everyday desktop computers).

OpenMP is already installed in the aiken server: aiken.ce.pdn.ac.lk

Or else, you can follow these installation guidelines and setup it in your computer:https://www.geeksforgeeks.org/openmp-introduction-with-installation-guide/

1.2.1 How do I compile my code to run OpenMP?

One of the useful things about OpenMP is that it allows users to use the same source code both with OpenMP-compliant compilers and normal compilers. This is achieved by making the OpenMP directives and commands hidden from regular compilers.

In this lab, you can use C / C++ and Fortran 90. Other versions of Fortran should use the same OpenMP directives and commands and the examples should be directly translatable. The same holds for C and C++.

Lab Task

1. Display a very simple multi-threaded parallel program "Hello, world." written in C / C++ / Fortran (you can choose the programming language according to your preference).

2. Creating Threads: calculate π in parallel using pragma omp parallel.

   A simple serial C code to calculate π is the following:

   ```
   unsigned long nsteps = 1<<27; /* around 10^8 steps */
   double dx = 1.0 / nsteps;

   double pi = 0.0;
   double start_time = omp_get_wtime();

   unsigned long i;
   for (i = 0; i < nsteps; i++)
   {
      double x = (i + 0.5) * dx;
      pi += 1.0 / (1.0 + x * x);
   }
   pi *= 4.0 * dx;
   ```

   Questions:

   2.1 How does the execution time change varying the number of threads? Is it what you expected? If not, why do you think it is so?

   2.2 Is there any technique you heard of in class to improve the scalability of the technique? How would you implement it?

3. Calculate π using critical and atomic directives.

   Instructions: Create two new parallel versions of the pi.c / pi.f90 program using the parallel construct #pragma omp parallel and 1) #pragma omp critical 2) #pragma omp atomic. Run the two new parallel codes and take the execution time with 1, 2, 4, 8, 16, and 32 threads. Record the timing in a table.
   Hints:

   - We can use a shared variable π to be updated concurrently by different threads. However, this variable needs to be protected with a critical section or atomic access.
   - Use critical and atomic before the update pi += step

   Questions:

3.1 What would happen if you hadn’t used critical or atomic a shared variable?

3.2 How does the execution time change varying the number of threads? Is it what you expected?

3.3 Do the two versions of the code differ in performance? If so, what do you think is the reason?