High Performance Computing

Homework #4

Due: Tuesday March 15 2016 by 11:59 PM

Email-based help Cutoff: 5:00 PM on Mon, March 14 2016

Maximum Points: 13

Submission Instructions

This homework assignment must be turned-in electronically via Canvas. Type in your responses to each question (right after the question in the space provided) in this document. You may use as much space as you need to respond to a given question. Once you have completed the assignment upload:

1. The MS-Word document (duly filled) and saved as a PDF file named with the convention `MUid.pdf` (example: **`raodm.pdf`**)

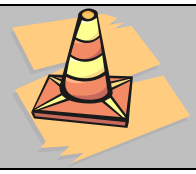
Note that copy-pasting from electronic resources is plagiarism. Consequently, you must suitably paraphrase the material in your own words when answering the following questions.

Name: Henry Ni

Objective

The objective of this homework is to review the necessary background information about shared memory parallelism using OpenMP.

Read subchapter 7.1 from E-book "Introduction to Parallel Computing" (all students have free access to the electronic book). Links available off Syllabus page on Canvas.



Although the Safari E-books are available to all students there are only a limited number of concurrent licenses to access the books. Consequently, do not procrastinate working on this homework or you may not be able to access the E-books due to other users accessing books.

1. In the space below tabulate three significant differences contrasting implicit and explicit parallelism: [**1 points**]

Implicit Parallelism	***Explicit Parallelism***
Automatically enabled without manual interference	Programmer must create new threads directly, or use other libraries such as OpenMP directives
Generally limited to a single machine/processor	Benefits can be extended to multiple computers in a compute node/cluster
Trivially accomplished	Can be a lot of work ensuring all threads/processes communicate without inconsistencies

2. What is shared address space architecture? What are the two types of memory access architectures commonly found on shared address space machines? [**1 points**]

Shared address space architecture means all processors on the CPU have a data access space in common. Can be generally classified into UMA (Uniform Memory Access) or NUMA (Non-UMA).

3. What is a thread? What are the resources that are shared and not shared between threads in a single process? [**1 point**]

Threads are one piece of a single process and often share code, heap space (between threads of the same process) and other resources in the same scope like sockets and open files.

Threads do not share stack space or general purpose registers.

4. In the space below tabulate three significant differences contrasting a thread and a process: [**1 points**]

Process	**Thread**
Resources are not shared between processes	Resources between threads of the same process are shared
Process can be stopped and started independently	All threads must be stopped or none at all
I/O streams are separate from other streams	Threads do not get their own I/O streams, they use their parent resources

5. In the space below tabulate 2 significant differences between task parallel and data parallel applications: [**1 point**]

Task Parallel	**Data Parallel**
Threads and processes perform the same task	Threads and processes perform different tasks
Data is not shared between different operations	Data is shared between different operations

6. Briefly describe one example application (such as: video encoding, image processing, etc.) for data parallel and task parallel applications. Your examples cannot be one of those already mentioned in lecture notes [**1 point**]
   a. Example of a data parallel application

      Sorting a large amount of data can be parallelized if each thread is given a small chunk of the larger set to work with.

   b. Example of a task parallel application

      Rendering a large area in a virtual environment (a video game or VR) can be done more quickly if the application can take advantage of task parallelization since one large area can be broken into many smaller areas.

7. Why does the output from the following OpenMP program appear garbled as shown in the adjacent area? What would an ideal output from the program look? [**1 point**]

```
#include <iostream>
#include <omp.h>

int main() {
  #pragma omp parallel
  { // start parallel
    int numThreads = omp_get_num_threads();
    int threadID   = omp_get_thread_num();
    std::cout << "hello OpenMP from "
              << threadID   << " of "
              << numThreads
              << " threads.\n";
  } // end parallel
  return 0;
}
```

```
hello OpenMP from hello OpenMP
from hello OpenMP from 0 of 4
threads.
3 of 4 threads.
1 of 4 threads.
hello OpenMP from 2 of 4 threads.
```

The output is not very readable because all the threads share the same I/O stream, so each stream prints at a time where it may overlap with another thread doing the same thing.

Ideal output would be:

Hello OpenMP from 0 of 4 threads.
Hello OpenMP from 1 of 4 threads.
Hello OpenMP from 2 of 4 threads.
Hello OpenMP from 3 of 4 threads.

8. What is the output from the adjacent OpenMP program when it is compiled and run as shown below? Briefly describe how you determined the output from the program? [**2 points**]

```
$ g++ -fopenmp q8.cpp -o q8
$ export OMP_NUM_THREADS=4
$ ./q8
```

```
#include <iostream>
#include <omp.h>

int main(int argc, char *argv[]) {
#pragma omp parallel if (argc > 1)
  { // fork
    std::cout << "Output #1\n";
  } // join

#pragma omp parallel num_threads(1)
  { // fork
    std::cout << "Output #2\n";
  } // join
  return 0;
}
```

Output:

Output #1
Output #2

Since only one argument is passed, the first #pragma directive is not used, and that portion of code is not parallelized. The second #pragma forces the CPU to use a single thread, so there is no risk of having overlapping thread output.

9. What is a race condition? How does a race condition impact outputs from a program or results from a method? [**1 point**]

Race conditions occur when multiples threads or processes operate on the same data and cause the value of the data being operated on to become indeterminable. The order in which the data is handled can change the outcome so threads must be ordered in the same way each execution or the data must be divided per thread to resolve this problem.

10. Parallelize the following data parallel method using OpenMP using a parallel block (but not using `omp parallel for construct`) [**3 points**]

```
// Assume this method is correctly implemented
bool isPrime(int num);

std::vector<bool> isPrime(const std::vector<int> numList) {
	std::vector<bool> primes(numList.size());
	for (size_t i = 0; (i < numList.size()); i++) {
		primes[i] = isPrime(numList[i]);
	}
	return primes;
}
```

```
 // Assume this method is correctly implemented
bool isPrime(int num);

std::vector<bool> isPrime(const std::vector<int> numList) {
	std::vector<bool> primes(numList.size());
```

```
        #pragma omp parallel
        {
          const int iterPerThr = numList.size() / omp_get_num_threads();
          const int startIndex = omp_get_thread_num() * iterPerThr;
          const int endIndex   = startIndex + iterPerThr;
          for (int idx = startIndex; (idx < endIndex); idx++) {
              primes[idx] = isPrime(numList[idx]);
           }
        }
       return primes;
}
```