High Performance Computing

Homework #6

Due: Wed Nov 3 2021 by 11:59 PM

Email-based help Cutoff: 5:00 PM on Tue, Nov 10 2021

Maximum Points: 20

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`**)

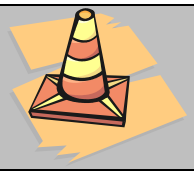
Note that copy-pasting from electronic resources is plagiarism. Consequently, you must suitably paraphrase the material in your own words when answering the following questions.

Name: Maciej Wozniak

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.



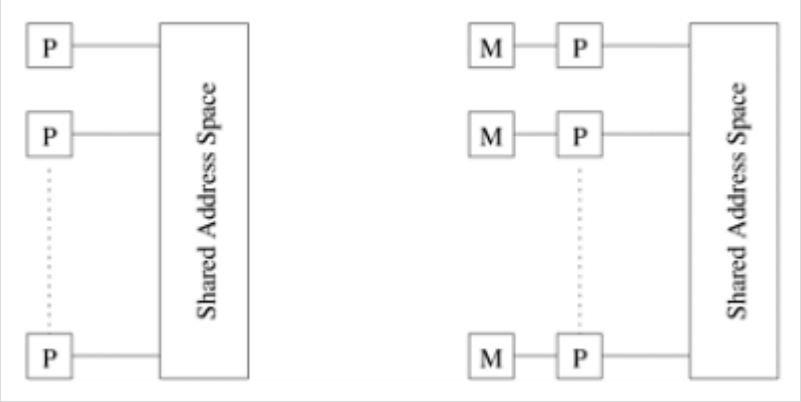
Although the Safari E-books are available to all students there are only a limited number of concurrent licenses to access the books. Consequently, do not procrastinate working on this homework or you may not be able to access the E-books due to other users accessing books.

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

Implicit Parallelism	***Explicit Parallelism***
Parallelizing compiler abilities, libraries or OS	decompose your work into threads and coordinate between them then
Typically operates at multi instruction or multi-thread level	Can be applied to multiple processors, machines, or supercomputing clusters.
Using hardware with parallelism abilities	Manually/programmatically done e.g. Using open MP

2. Using Figure 7.1 from the textbook, briefly describe the logical machine model of a thread-based programming paradigm [**1 points**]

All thread must have the access to the same memory. If memory is shared between the threads and threads are editing it, we may end up with program which is more efficient at all. This would be cased the fact that once the change is done by one thread, other threads must stop and update the file.



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

Task Parallel	**Data Parallel**
Often uses the shared data (different tasks operates on the same copy of data)	Splitting data into subsets and running the process on the separate subset
Works on hared memory architecture (all cores access the same memory)	Each task runs using different memory

4. The following questions deal with explicit multi-threading: [**2 points**]
   a. What is explicit multi-threading?

      Multiple tasks running on the same memory. It requires specification of concurrent tasks and their interactions

   b. State 2 advantages of this approach

      Maximizes efficiency, better control on synchronization

   c. State 2 disadvantages of this approach

      Requires rewriting the programs to efficiently using multithreads, programmer must handle race conditions

5. Describe (2 to 3 sentences) "Programmer Guided automatic Multi-threading" approach used by OpenMP? How is it different from explicit multi-threading? [**2 points**]

   Programmer uses libraries such as OpenMP to identify regions with concurrency, subdivides processes into threads and identifies which methods to use. It is also programmer's task to identify potential issues (such as race conditions). The main differences are (i) explicit approach is not limited to one device and guided multithreading and (ii) explicit programming requires considerable effort, where Programmer Guided automatic Multi-threading requires user to edit/add a couple of lines.

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

      Videos analysis could be video frames would be divided into separate clips and fed to each core/thread individually.

   b. Example of a task parallel application

      Video analysis – video is a shared data type among all the threads but each thread analyses different frames of the video, so there is no conflict.

7. Consider the following OpenMP program that is being run using 4 threads

```
#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;
}
```

Actual output from above program (with 4 threads) copy-pasted below:

```
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.
```

a. Why does the output appear garbled as shown above? [**1 point**]

Multithreading does not guarantee which thread will finish first, it can happen that if we run this program 2nd time, order will be different

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;
}
```

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

Race condition is 2 or more threads accessing and modifying the same data at the same time. That can lead to erroneous output and conflicts

10. Parallelize the following data parallel method using OpenMP using a parallel block by manually computing starting and ending index/bounds for each thread (**no**, you cannot use `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;
}
```

11. Exploring "Programmer Guided automatic Multi-threading" via parallel algorithms in C++ [**2 points data + 3 points for inferences = 5 points**]

Background: The C++ Standard Library implementation provided with the GNU C++ compiler called `libstdc++`) also provides a “parallel mode". Using this mode enables existing serial code to take advantage of many parallelized standard algorithms that enables effective use of multi-core processors or multi-CPU machines. The parallel mode operations can be enabled using compiler flags (namely: `-fopenmp -D_GLIBCXX_PARALLEL`) to enable use of parallel mode of operation whenever possible. The

compiler and standard library use heuristics to decide if an algorithm should be run in parallel. Consequently, some of the algorithms may not be run in parallel mode for your specific program/data. Note that this mode of operation does not require any changes to the program – however this is still explicit parallelism – just that some other programmer has written the library for us using OpenMP.

For this part of the exercise, you are supplied with a slightly modified version of the solution from a prior exercise in this course in `/fs/ess/PMIU0184/cse443/homeworks/homework6`.
Do NOT modify the program but study it. You just have to run it (on `pitzer.osc.edu`) using the supplied SLURM script.

Once the job completes, collect runtime statistics from the output file in the following table:

Threads	Run #	User time	System time	Elapsed time	%CPU
1	1	49.83	0	0:49.96	99
1	2	49.92	0	0:49.98	99
1	3	49.84	0	0:49.97	99
1	4	49.79	0	0:49.92	99
1	5	49.64	0	0:49.76	99
1	**Average**		**0**		**99**
2	1	51.51	0.04	0:25.96	198
2	2	51.55	0	0:25.92	198
2	3	52.07	0	0:26.15	199
2	4	51.24	0	0:25.83	198
2	5	51.12	0	0:25.77	198
2	**Average**		**0.01**		
4	1	51.08	0	0:14.35	355
4	2	50.82	0	0:14.26	354
4	3	50.66	0	0:14.17	355
4	4	50.90	0	0:14.27	355
4	5	51.03	0	0:14.37	353
4	**Average**		0		
6	1	53.94	0.01	0:09.36	569
6	2	54.4	0.03	0:09.56	570
6	3	54.22	0.02	0:09.49	569
6	4	54.09	0.03	0:09.48	569
6	5	54.22	0.03	0:09.52	570
6	**Average**		0		
8	1	53.84	0	0:07.26	741
8	2	53.70	0	0:07.25	739
8	3	53.76	0	0:07.25	741
8	4	53.65	0	0:07.25	740
8	5	53.76	0	0:07.25	740
8	**Average**		0		

Speedup S is defined as: S = Ts ÷ Tp, where Ts is serial runtime (recorded in Task 2) and Tp is parallel runtime. Note that in theory, under ideal conditions, given *n* threads it is desirable to attain a speedup of *n*. Consequently, the theoretical runtime with *n* threads is computed using the formula T_{THEORY} = Ts ÷ *n*. Using these relationships (or formulas) compute the expected theoretical and observed speedups using a varying number of threads:

Serial runtime	Threads	Theoretical or expected runtime	Observed Runtime	Observed Speedup
53.75	1	53.75	53.75	1
53.75	2	26.875	27.01	1.99
53.75	4	13.43	15.01	3.58
53.75	6	8.95	9.75	5.51
53.75	8	6.72	7.84	6.86

Briefly (3 to 4 sentences each) describe your inferences from the experiment in the space below:

1. Does the user time vary a lot when using multiple threads? Why not?
It does not change. User time is spent on tasks such as printing and writing to the file which is almost the same at each step.

2. Does the elapsed time vary a lot? Why?
Elapsed time vary a lot since we can see that the instructions are divined on multiple threads. That is significantly speeding up the whole program.

3. Does the system time vary with threads? Why?
No system time is almost 0 at each time. We do not access any files in the memory except of the one which we are editing, thus it is always 0.

4. Does the %CPU increase with number of threads? Why?
Yes it does. More cores are available, thus program will use more if we provide them to it.

5. Does the program achieve theoretical speedup? Why not?
It achieves good speed up but not theoretical. We always *loose* some percentage due to e.g. noise, parts of the program which are not possible to parallelize etc.

6. Although not recorded in tables above, you should notice a slight growth in memory usage of the program when number of threads are increased. Why?
It could be caused by the fact that we program uses first privates – i.e. making a copy of the variables/data for each thread.

Due:	**Wed Nov 3 2021 Before 11:59 PM (Midnight)**