Examination 2020-10-16

Introduction to Parallel Programming (1DL530) – 5 hours

This is an examination **done at home**, so you can have your books and notes open when you take it. You also need to have access to a PC or server where you can compiler and run C/C++ programs with Pthreads, OpenMP and MPI. Please refrain from searching the web for answers to these questions but, if you do, you **need to explicitly mention these sources**. Please avoid submitting hand-written text or low-quality pictures from mobile phones; instead **use an editor or word processor** for all your answers in text and, if possible, a drawing program for your answer to question 6. When you finish, you need to **upload your program for question 2 and a single PDF with your answers to the other questions** to your group in the Studium. Also, **remember to write your anonymous code in both your program and in the PDF with your answers**. Your submission **must be uploaded/received by 13:30**.
The exam contains **40** points in total and their distribution between sub-questions is clearly identifiable. To get a final grade of **5** you must score at least **34**. To get a grade of **4** you must score at least **28** and to pass the exam you must score at least **19**. Whenever in *real doubt* for what a particular question might mean, **state your assumptions clearly**.

DON'T PANIC!

---

1. **Performance Metrics (2 + 1 + 1 + 4 = 8 pts total).**

   - Assume an application where the execution of floating-point instructions on a certain processor P consumes 60% of the total runtime. Moreover, let us assume that 25% of the floating-point time is spent in square root calculations. The design team of the next-generation processor P' believes that they can either improve the performance of all floating point instructions by a factor of 1.5 or alternatively speed up the square root operation by a factor of 15. From which design alternative would the aforementioned application benefit the most? Justify your answer by showing your calculations.
   - **(Amdahl's Law)** Instead of waiting for the next processor generation, the developers of the application decide to parallelize its code.
     (a) What speedup can be achieved on a 16-CPU system, if 90% of the entire program can be perfectly parallelized?
     (b) What fraction of the code has to be parallelized to get a speedup of 10?
   - **(Weak Scalability)** Algorithm A takes an integer k as input, uses $O(k^2)$ workspace, and has cubic (i.e., $O(k^3)$) time complexity. Let $T_p(k)$ be the execution time of A to solve a problem of size k with p processes. It is known that $T_1(4000) = 12$ minutes. Assuming perfect weak scalability, what is the expected execution time for $k = 64000$?

---

2. **Parallel Programming Using Pthreads ($4 + 2 + 2 = 8$ pts total).** Conway's Game of Life should be known to you from Assignment 3 of this (and last) year's course. Please refer to the text of that assignment if you do not recall the details of the game. That assignment also came with a sequential implemetation of the game given in file `Game_of_Life.c`, which took as arguments the size of the array and the number of generations (steps) in the game.

   (a) Parallelize the part of the code between the two `gettimeofday()` calls using Pthreads. You will need to submit your code (the resulting `Game_of_Life_par.c` file), together with a brief description of what its parallelization involved and the main changes you did. Please refrain from doing unnecessary changes in the remaining functions of the file.

   (b) Did you need to use locks or any other synchronization mechanisms and why?

   (c) Using a departmental server (or your laptop/PC if it is powerful enough to do this task), measure and report the speedup your program achieves when running with 2, 4 and 8 threads on a 1024×1024 array using 4000 steps.

---

3. **Parallel Programming ($1 + 1 + 2 + 2 = 6$ pts total).** Consider the code below:

```
int   locks[100];
float data[1000000];

#pragma omp parallel for
for (i = 0; i < 1000000; i++) {
  lock(locks[g(i)%100]);
  f(data[g(i)]);
  unlock(locks[g(i)%100]);
}
```

   Assume that the function `f` only accesses its argument, may read and write it, and does not do any synchronization internally. Give brief answers to the following questions:

   (a) How many threads can be active at any time?

   (b) Assuming locking is free, what is the maximum speedup that this program can achieve?

   (c) Is the program data race free or not? Explain your answer.

   (d) Assuming locking is free, is it possible that the loop executes sequentially because of locking?

---

4. **Parallelization Using OpenMP (6 pts total).** The code below takes an array and "rotates" it by shifting all its elements to the left and appending the first element.

```
void rotate(int n, double *a) {
  for (int i = 0; i < n - 1; i++) {
    double tmp = a[i];
    a[i] = a[i+1];
    a[i+1] = tmp;
  }
}
```

   Parallelize the code of this function using OpenMP and submit your solution. Make sure you properly handle the dependencies that exist in this code.

5. **OpenMP Programming (6 pts total).** A programmer has parallelized a `foo` function using OpenMP pragmas as follows:

```
int foo(int n, int *a, int *b, int *c, int *d) {
  double tmp, total = 0;
  #pragma omp parallel
  {
    #pragma omp sections
    {
      #pragma omp section
      {
        #pragma omp for
        for (int i = 0; i < n; i++) {
          a[b[i]] += b[i];
          total += b[i];
        }
      }
      #pragma omp section
      {
        #pragma omp for
        for (int i = 0; i < n; i++) {
          tmp = c[i];
          c[i] = d[i];
          d[i] = tmp;
          total += c[i];
       }
       #pragma omp for
       for (int i = 0; i < n; i++) {
         total += d[i];
       }
      }
    }
  }
  return total;
}
```

If you think there are any issues (i.e., errors, omissions in the pragmas, or pragmas which are missing) with this code, explain why they are issues and suggest a (good) way to fix them.

---

6. **MPI Programming (3 + 3 = 6 pts total).** In lecture 11, we examined *MPI Collective Communication* and saw different ways that communication can be depicted using trees. We also discussed the functions `MPI_Scatter` and `MPI_Gather`. Suppose `comm_sz` = 8 and `n` = 16.

- Draw a diagram that shows how `MPI_Scatter` can be implemented using tree-structured communication with `comm_sz` processes when process 0 needs to distribute an array containing `n` elements.
- Draw a diagram that shows how `MPI_Gather` can be implemented using tree-structured communication when an `n`-element array that has been distributed among `comm_sz` processes needs to be gathered onto process 0.

Good luck !