Examination 2023-08-23

Introduction to Parallel Programming (5 hours)

Please make sure to write your exam code on each page you hand in. When you are finished, please staple the pages containing your solutions together in an order that corresponds to the order of the questions. You can take with you this exam if you want.
This is a **closed book** examination but you are allowed to have with you an **A4 sheet of prepared hand-written notes**. 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**. Note that you will get **credit only for answers that are correct (or a very serious attempt) and clear**. All answers should preferably be **in English**; if you are uncomfortable with English you might of course use Swedish. Whenever in *real doubt* for what a particular question might mean, **state your assumptions clearly**.

DON'T PANIC!

1. **Amdahl's Law (3 + 3 = 6 pts total).**

   (a) You have a choice between buying one uniprocessor that executes five zillion instructions per second, or a ten-processor multiprocessor where each processor executes one zillion instructions per second. Using Amdahl's Law, explain how you would decide which to buy for a particular application. (A *zillion* is a fictitious, indefinitely large number.)

   (b) A friend of yours who works for a company developing parallel software claims that, despite Amdahl's law, in practice many parallel programs obtain excellent speedups. Can you think of reasons for this to hold or is your friend exaggerating?

---

2. **Locks and Synchronization (2 + 2 + 3 + 3 = 10 pts total).**

   (a) What types of mutexes are supported by Pthreads and what are the differences between them?

   (b) What are read-write locks and what are their main advantages compared to mutex locks?

   (c) What are TATAS locks and how do they differ from TAS locks? Why do TATAS locks achieve better scalability than TAS locks as the number of threads increases?

   (d) Explain what the terms blocking and non-blocking synchronization mean and how these two synchronization mechanisms compare against each other.

---

3. **Task and Data Parallelism (3 pts, 1 pt each).** What is task-parallelism? What is data-parallelism? Give an example from *real life* (i.e., not from programming!) that illustrates the difference between these two kinds of parallelism.

---

4. **Parallel Speedup (3 pts total).** A parallel program that obtains a speedup greater than p (the number of processes or threads) is sometimes said to have *superlinear speedup*. However, many authors do not count programs that overcome "resource limitations" as having superlinear speedup. For example, a program that must use secondary storage for its data when it is run on a single processor system might be able to fit all its data into main memory when run on a large multicore or distributed memory system. Give another example on how a program might overcome a resource limitation and obtain speedups greater than p.

---

5. **Safe Locking (4 pts).** The linked list operations `insert` and `delete` consist of two distinct "phases." In the first phase, both operations search the list for either the position of the new node or the position of the node to be deleted. If the outcome of the first phase so indicates, in the second phase a new node is inserted or an existing node is deleted. In fact, it is quite common for linked list programs to split each of these operations into two function calls. For both operations, the first phase involves only read-access to the list; only the second phase modifies the list. Would it be safe to lock the list using a read-lock for the first phase? And then to lock the list using a write-lock for the second phase? Explain your answer.

---

6. **Condition Variables (2 + 4 = 6 pts).**
   (a) What are condition variables?
   (b) How can we implement a *barrier* with a condition variable?

---

7. **OpenMP (4 pts; 2 pts each).** Consider the following recursive function to compute the depth of a binary tree.

```
int getDepth() {
  int hL = 0; int hR = 0;
  if (this.left  != null) { hL = this.left.getDepth();
  if (this.right != null) { hR = this.right.getDepth(); }

  return 1 + max(hL, hR);
}
```

   (a) Rewrite this function by adding pseudo-OpenMP annotations.
   (b) Assume that `this.left` and `this.right` have an operation `.getSize()` that returns the number of values these subtrees contain in constant time (A tree without children has size 1, a tree with two children but no "grand children" has size 3, etc.). Use this operation to speed up your function from question 7a. Explain what problem this solves.

---

8. **Parallel Performance (4 pts; 2 pts each).** Your friends are devastated: their program does not scale! Luckily, you are around. Your friends' program runs four threads, each thread executes the function `doWork`. Here is how it looks like:

```
float partialResults[4];

void doWork(int threadID) {
  for (int i = 0; i < 10000000; ++i) {
    partialResults[threadID] += computeSegment(threadID, i);
  }
}

int main(...) {
  ... // initialize partialResults, spawn worker threads, then join them
  float sum = 0.0;
  for (int i = 0; i < 4; ++i) { sum += partialResults[i]; }
  printf("sum = %f\n", sum);
}
```

The program has very short sequential parts, so your friends expected their program to scale linearly. However, the runtime barely improves for two threads, and it gets even worse for more threads than two. The function `computeSegment` is not our concern, it does only boring, but tedious mathematical computations.

(a) What is the problem, in your opinion?

(b) How would you fix the problem? Rewrite the relevant parts of the code.

Good luck !