Exam

DVA336 PARALLEL SYSTEMS

08:10–12:30

January 13, 2020

This exam includes 8 problems, for a total of 80 points.

General information

1. This exam is closed book.
2. A simple non-programmable calculator is permitted.
3. Write your answers in English.
4. Make sure your handwriting is readable.
5. Answer each problem on a separate sheet.
6. Write your identification code on all sheets as required for anonymous exams.
7. All answers must be motivated. If necessary, make suitable assumptions.

Grading

The passing marks are 90% for grade 5, 70% for grade 4, and 50% for grade 3. This corresponds to the passing grades A, C, and E in the ECTS grading system.

Contact information

For questions on problems 1, 3-7, please contact Gabriele Capannini, 021-101458.
For questions on problems 2 and 8, please contact Björn Lisper, 021-151709.

Good luck!

1. (a) [6 p] Explain what the *cache-coherence* mechanism is and how it works.

   (b) [4 p] Explain what the *false-sharing* phenomenon is and how to avoid it.

2. (a) [4 p] What is Schwartz' algorithm? What does it compute? Explain briefly how it works! (Hint: it may help to draw a tree illustrating the execution of the algorithm.)

   (b) [3 p] What is the time complexity for applying Schwartz' algorithm to an array with n elements on p processors? You may assume that $n \gg p$.

3. Answer the following questions about the vectorized CPU instructions:

   (a) [2 p] What type of parallelism they implement?

   (b) [3 p] For which kind of computations they are useless? Provide a simple C-like program as an example.

   (c) [3 p] How should data be stored in memory to enable/facilitate their usage?

4. (a) [7 p] Give the definition of the *Gustafson's Law* and the *Amdahl's Law* and discuss the differences between them.

   (b) [4 p] Let A be a serial algorithm and assume that two thirds of the work performed by A are perfectly parallelizable. Give an approximation of the scalability achieved by the parallel version of A when running on 150 processors according to the Amdahl's Law and the Gustafson's Law.

5. Let S be a system performing a stream-based computation. S consists of 3 sequential modules $M_{0..2}$ computing, respectively, the functions $f_{0..2}$ having latency $10t$, $45t$, and $5t$ where t is a generic time unit. M_0 produces the input stream.

$$\begin{tikzcd} M_0 \arrow[r] & M_1 \arrow[r] & M_2 \arrow[r, "T_p"] & {} \end{tikzcd}$$

   In particular, f_1 is a linear composition of three sub-functions:

$$f_1(x) = f_{13}(f_{12}(f_{11}(x)))$$

   Assume that f_{11}, f_{12}, and f_{13} are pure mathematical stateless functions and the latency of each of them is $15t$ and the communication overheads don't affect the performance of S.

   (a) [4 p] What is the inter-departure time T_p of S?

   (b) [4 p] What are the ideal performance of S? Explain and motivate your answer.

(c) [8 p] Is there any bottleneck in the computation? If yes, provide a parallelization of the bottleneck such that S can achieve its ideal performance and explain your solution by showing the related calculations.

6. (a) [9 p] Describe what the *symmetric* and *asymmetric synchronization* mechanisms are and what is the main difference between them with respect to *mutual exclusion* problem.

7. (a) [5 p] Mind the functionality of the `MPI_Barrier(MPI_Comm comm)` function and write a special version of it which implements an additional feature: the barrier returns an `int*` which is `NULL` for processes with rank grater than 0 while, for the the process 0, it points to an array which stores the order in which the other processes has reached the barrier. *Note*: you can't force the processes to be registered in a predefined order. To implement such a barrier you could need to use: the "wild-char" `MPI_ANY_SOURCE` in the `MPI_Recv` function to receive messages without restricting the rank of the sender, and the member `st.MPI_SOURCE` which stores the rank of the sender of the message received by means of the `MPI_Recv` using `st` as parameter (see the example below). *Hint*: the ring schema used in the assignment does not help.

   (b) [1 p] What is the complexity of your barrier?

   Example for the `MPI` syntax:

```
int main(int argc, char* argv[]) {
  MPI_Init(&argc, &argv);
  int rank, msg, tag = 123;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if(rank == 0)
    MPI_Send(&msg, 1, MPI_INT, 1, tag, MPI_COMM_WORLD);
  if(rank == 1) {
    MPI_Status st;
    MPI_Recv(&msg, 1, MPI_INT, MPI_ANY_SOURCE, tag, MPI_COMM_WORLD, &st);
    printf("received value from rank %d.", st.MPI_SOURCE);
  }
  MPI_Finalize();
  return 0;
}
```

8. (a) [13 p] In statistics, the *standard deviation* σ of a finite set of random numbers $\{x_1, \ldots, x_n\}$ is defined as

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2}, \quad \text{where } \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

   Define a data parallel algorithm for computing the standard deviation of a set of random numbers stored in a data parallel array! Use the notation for data parallel algorithms that we have used in the lectures on data parallelism and parallel algorithms (not OpenMP or such).

What is the parallel time complexity of your algorithm? To get full credits, your solution must have a parallel time complexity that is significantly lower than the sequential complexity for the straightforward sequential solution.