UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

EXAMINATION QUESTION PAPER

Exam in:	INF-3201 Parallel Programming
Date:	Wednesday 2019-11-27
Time:	Kl 15:00 - 19:00
Place:	ADM-B154
Approved aids:	None
Type of sheets (squares/lines):	Digital exam
Number of pages incl. cover page:	4
Contact person during the exam: Phone:	John Markus Bjørndalen 90148307
	Will a lecturer/person in charge visit the venue? **YES, approx. time: 16:30** **NO:**

NB! It is not allowed to submit scratch paper along with the answer sheets.
If you do submit scratch paper, it will not be evaluated.

NB: for questions that ask for pseudocode, you can, if you prefer to do so, use languages such as Python, Java, C and Go instead. There is no requirement for remembering exact parameter lists, parameter orders, or exact names of API calls, but please indicate important information that must be passed to function calls.

Please make sure you read the questions fully before answering. There may be hints or suggestions that could make it simpler for you to answer.

You are free to answer in Norwegian or English, or a combination of the two.

Good luck!

1) Terms (15%)

a) Assume that we have a program where 10% of the code must be executed sequentially. What is the maximum potential *speedup* of the program (ignoring communication overhead) running on 32 CPU's? Assume a single core per CPU. An approximation is enough if you provide the right equation.

b) What is *Amdahl's law*?

c) A term related to speedup is *efficiency*. Explain the term in the context of running a parallel program on a cluster.

d) What is *superlinear speedup*? Why can this happen?

2) Load Balancing (10%)

Give a short explanation of the following:

a) Load balancing.

b) Static load balancing.

c) Dynamic load balancing.

d) What separates applications (or algorithms) that benefit from static load balancing vs. ones that benefit from dynamic load balancing?

3) Parallel execution (15%)

The following source code show a simple matrix multiplication algorithm.

```
// Matrix multiplication
void mxmult()
{
    // Given the following arrays (initialised elsewhere)
    int A[N][N];
    int B[N][N];
    int C[N][N];
    ...
    // Simple matrix multiplication C = A*B
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            C[i][j] = 0;
```

```
            for (int k = 0; k < N; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

a) *What are Bernstein's conditions? Give a short description.*

b) *Can you run the iterations of any of the loops in parallel without causing problems? If so, which? Explain why.*

Hint: Bernstein's conditions are useful for analysing the problem.

4) Collective/group operations (10%)

a) *Explain the term group (or collective) operations.* A short description is enough.

Explain the following operations:

b) Broadcast

c) Reduce

d) Gather

e) Scatter

5) Pipelines, CSP and MPI (20%)

We are going to make a simple pipelined program with P processes. Each process adds a constant to a received partial sum and passes it to the next process. The first process is the *source* and the last process is the *sink*. For simplicity, just assume that the source has a list of N numbers that it wants to send through the pipeline (one by one), and that the last process collects the finished numbers in a result list.

When writing the programs, you can use simple pseudocode. You don't need to remember all of the parameters or exact names for API calls. You can also use a CSP-based language or library (PyCSP, JCSP, Go or similar pseudocode) instead of CSP syntax from the Hoare paper.

a) Write pseudocode for a simple CSP program implementing the program.

b) Write pseudocode for a simple MPI program implementing the program.

It often makes sense for the source and the sink to be the same process (the process with the problem is often the one using the solution). This changes the configuration from a line to a circle.

c) This can easily cause deadlocks. Why?

d) CSP has a mechanism we can use to avoid deadlocks in this situation. What is it and how does it work? Modify your program to avoid deadlocks using this mechanism.

e) MPI provides multiple options for avoiding deadlocks with the circle configuration. Describe one and show how you can change your program to use it.

6) GPGPU computing with CUDA (20%)

a) What is a “Kernel”?

b) What are “thread blocks”, “warps” and “grids”? How are they related?

c) Outline how a program that uses a GPU looks and works. The description should be high level and can use pseudocode. Assume that the program has some data on the host that the GPU should use and that the GPU code should produce some resulting data for the host/CPU. Don’t worry about performance optimisation, we’re after a rough outline/template and how the GPU and the CPU cooperate.

A suggestion for a simple GPU program would be to add two vectors.

7) GPU vs CPU (10%)

Given two architectures from one of the papers we looked at this year (for simplicity, we only include some of the numbers):

	Num. PE	BW (GB/sec)	Peak SP Scalar, GFLOPS
CPU: Core i7-960	4	32	25.6
GPU: GTX280	30	141	116.6

As a rule of thumb, which of the architectures would you use for algorithms that are:

a) Compute bound

b) Memory bound

c) Synchronisation bound

Explain why for each. You only need to describe the main reason for each.