UiT The Arctic University of Norway

Home exam in:	INF-3201 Parallel Programming
Date:	2020-11-24
Time (duration):	09:00-13:00
Course coordinator:	John Markus Bjørndalen
Contact information:	john.markus.bjorndalen@uit.no +47 90148037 Also: the course discord.
Number of pages:	4
Weighting of questions, or other information:	See weighting indicated on the questions.
Important information about this exam	1. This is an individual examination and must be completed without cooperation of any kind. 2. You are allowed to use any kind of sources (lecture notes, lectures in pdf. format, books, internet-sources) in answering the topics of the exam. 3. All exam answers delivered in Wiseflow are automatically checked for plagiarism. Be aware that copying sources, web pages, other students or literature without reference is not allowed, and will be considered as cheating.

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.

NB 2: we do not expect executing programs! Do not waste your time with a compiler.

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!

Pre-code

The following two source codes show a simple matrix multiplication algorithm and a part of the Mandelbrot algorithm from mandatory exercise 1. These will be used in the following questions.

```
// 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];
            }
        }
    }
}
```

Listing 1: Matrix multiplication

```
int roadMap[HEIGHT][WIDTH];

int solve(double x, double y)
{
    complex z = {0.0, 0.0};
    complex c = {x, y};
    int itt = 0;
    for (itt = 0; (itt < MAX_ITERATIONS) && (complex_magn2(z) <= 4.0); itt++) {
        z = complex_add(complex_squared(z), c);
    }
    return itt;
}

int CreateMap()
{
    int crc = 0;
    //Loops over pixels
    for (int y=0; y<HEIGHT; y++) {
```

```
        for (int x=0; x<WIDTH; x++) {
            // Store the number of iterations for this pixel
            int c = solve(translate_x(x), translate_y(y));
            roadMap[y][x] = c;
            crc += c;
        }
    }
    return crc;
}
```

Listing 2: Mandelbrot

1) Parallelization (20%)

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

b) *What is a memory bound algorithm? Which one of the above algorithms would be memory bond? Why?*

c) *What is a compute bound algorithm? Which one of the above algorithms would be compute bound? Why?*

2) MPI (25%)

We are using the code from Listing 2 for this part.

a) *On which type of parallel computing resources or computers do you typically run MPI programs?*

b) *Write a rough outline of an MPI program with a master/worker architecture.* You can use pseudocode and the master and worker functions can be empty (they don't need to communicate with each other) as we're just looking for the structure here.

c) *Write pseudocode and a short explanation of how CreateMap can be parallelized with MPI.* You don't need to write an optimal program: just write a first approach to, for instance, a program with static task assignment. The main points are how you break up the `CreateMap` function (you don't need the entire program), what communication you need and how you use the MPI functions.

3) OpenMP (25%)

We are using the code from Listing 2 for this part.

a) *On which type of parallel computing resources or computers do you typically run OpenMP programs?*

b) *What is a race condition? Can you see a potential for a race condition when parallelizing* `CreateMap`*?*

c) *Parallelize* `CreateMap` *using OpenMP.* You don't need to remember exact syntax as long as you explain what you are doing and why it is safe.

4) CUDA (30%)

We are using the code from Listing 2 for this part.

a) *On which type of parallel computing resources or computers do you typically run CUDA programs?*

b) *Write an outline of a typical CUDA program.* You can use pseudocode and the kernel does not have to do any actual work, but write the outline as if you need to run some computations on an array and write back results to a different array. An example could be adding two matrices or vectors ($A = B + C$).

c) *Write a CUDA version of* `CreateMap`. You can optionally ignore the computation of `crc` here (just remove it from the code). You only need to focus on `CreateMap` and how to invoke and use it.

d) *Why is computing* `crc` *a challenge with CUDA?*