UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

EXAMINATION PAPER

Exam in:	**INF-3201 Parallel Programming**
Date:	**Wednesday December 2, 2015**
Time:	**09:00 - 13:00**
Place:	**Åsgårdveien 9**

Approved aids: None

The exam contains 5 pages, including this cover page.

Contact person: John Markus Bjørndalen.
Phone: 90148307

NB! It is not allowed to submit scratch paper along with the answer sheets

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 or exact names of API calls.

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

Good luck!

1) Load Balancing (15%)

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?

2) Message Passing (15%)

a) Give a short explanation of what blocking vs. non-blocking communication is in message passing systems and show how the two can be used. You can use MPI as an example (or MPI-like pseudocode).

b) What is the difference between asynchronous and synchronous operations?

c) When does a locally blocking send routine in MPI behave as a synchronous routine?

3) Collective/group operations (15%)

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

4) 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 that the GPU should use and that the GPU code should produce some resulting data for the CPU. Don't worry about performance optimisations, we're after a rough outline/template and how the GPU and the CPU cooperate.

5) Classification of computations (10%)

The following two source codes show a simple matrix multiplication algorithm and a part of the Mandelbrot algorithm from mandatory exercise 1.

Matrix multiplication:

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

Mandelbrot:

```
/**
 * Mandelbrot divergence test (from mandatory exercise 1)
 *
 * @param        x,y     Space coordinates
 * @returns      Number of iterations before convergence
 */
inline int solve(double x, double y)
{
  double r=0.0,s=0.0;

  double next_r,next_s;
  int itt=0;

  while((r*r+s*s)<=4.0) {
    next_r=r*r-s*s+x;
    next_s=2*r*s+y;
    r=next_r; s=next_s;
    if(++itt==ITERATIONS)break;
  }

  return itt;
}
// somewhere in the main code:
  ....
  //Loops over pixels
  for(y=0;y<HEIGHT;y++){
    for(x=0;x<WIDTH;x++){
      ...
      colorMap[y][x]=palette[solve(translate_x(x),translate_y(y))];
    }
  }
  ....
```

Two of the classifications we can use for algorithms are *memory bound* and *CPU/compute bound*.

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

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

6) 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.

7) CSP mechanisms (15%)

The following PyCSP program uses the Alternation construct.

```
# Using the old PyCSP 0.3 library
@process
def lazy(out):
    while True:
        time.sleep(5)
        out("kebab")

@process
def eager(out):
    while True:
        time.sleep(1)
        out("beer")

@process
def reader(ch1, ch2):
    alt = Alternative(ch1, ch2)
    while 1:
        ret = alt.select()
        print "Got a", ret()

ch1 = One2OneChannel()
ch2 = One2OneChannel()
Parallel(lazy(ch1.write),
         eager(ch2.write),
         reader(ch1.read, ch2.read))
```

The following code is equivalent if you prefer to read Go:

```
func lazy(out chan string) {
   for {
      time.Sleep(5 * time.Second)
      out <- "kebab"
   }
}

func eager(out chan string) {
   for {
      time.Sleep(1 * time.Second)
      out <- "beer"
   }
}
func reader(ch1 chan string, ch2 chan string) {
   for {
      select {
         case s := <- ch1:
            fmt.Println("Got a", s)
         case s := <- ch2:
            fmt.Println("Got a", s)
         }
   }
}

func main() {
   ch1 := make(chan string)
   ch2 := make(chan string)
   go lazy(ch1);
   go eager(ch2);
   go reader(ch1, ch2);
   time.Sleep(30*time.Second)  // some arbitrary time
}
```

a) Draw the process network.

b) Why do we need an external choice construct in CSP (called Alternative in PyCSP, select in Go)? What does it solve?

c) What (approximately) does the program print?