UiT
NORGES
ARKTISKE
UNIVERSITET

EXAMINATION PAPER

Exam in: INF-3201

Date: Wednesday 04. December 2013

Time: Kl 09:00 – 13:00

Place: Adm.bygget, B154

Approved aids:

- None

The exam contains 3 pages including this cover page

Contact person: John Markus Bjørndalen

Phone: 92671202

UNIVERSITETET I TROMSØ

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.

1) Speedup and scaling (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) if we execute it on 32 CPU's?
b) What is Amdahl's law?
c) What is superlinear speedup?
d) Why can we observe superlinear speedup in applications such as search applications?

2) Data partitioning and load balancing (10%)

Explain and contrast the following terms:

a) Data partitioning vs. functional parallelism
b) Static vs. dynamic load balancing

3) 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?

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 optimizations, we're after a rough outline/template and how the GPU and the CPU cooperate.

5) Transactional memory (10%)

a) What is a transaction?
b) From the perspective of the programmer, what are the *advantages* of using transactional memory instead of hand-crafted fine-grained locking for concurrent data structures?

6) Pipelines, CSP and MPI (30%)

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 numbers of length N 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 (PyCSP, JCSP, Go or similar pseudocode) instead of syntax from CSP theory.

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.