UNIVERSITY OF TROMSØ UiT

FACULTY OF SCIENCE AND TECHNOLOGY

EXAMINATION PAPER

Exam in:	**INF-3201**
Date:	**December 5, 2012**
Time:	**09:00 – 13:00**
Place:	**Adm.bygget, B154**

Approved aids:

- English dictionary

- English-Norwegian/Norwegian-English dictionary

The exam contains 4 pages including this cover page

Contact person:	**Lars Tiede**
Phone:	**45500550**

Please give short and concise answers. State explicitly any assumptions you do.

1) Pipelined computations (15%)

a) Write a pseudo-code for adding a sequence of integers using pipeline technique with message-passing, where each process will add a number to the sum and the partial sum is passed from one process to the next. The number of processes ***P*** equals the number of integers ***N*** and the result is in the last process.

b) Compute the time complexity (in term of pipeline cycles) of adding *a single* instance of the sequence.

c) Compute the time complexity (in term of pipeline cycles) of adding ***M*** instances of the sequence, where **M** is much greater than **N** (or **M** >> **N**).

2) Partitioning and Divide-and-conquer (15%)

a) What is the main difference between the partitioning strategy and the divide-and-conquer strategy?

Suppose there is a function Divide(***S***, ***P***) that divides a sequence ***S*** of integers into ***P*** equal subsequences {s_1, …, s_P}.

b) Write a master-slave pseudo-code for adding a sequence ***S*** of ***N*** integers on 8 processes using the *partitioning technique* together with message-passing.

c) Write a pseudo-code for the *master process* (e.g. process 0) to add a sequence ***S*** of ***N*** integers on 8 processes using the *divide-and-conquer technique* together with message-passing.

3) Parallelism and Performance (15%)

a) Describe how the work-stealing scheduler works in Cilk++.

b) Compute Work, Span and Parallelism of the following code. Note that cilk_for is implemented using the divide-and-conquer technique. Explain your computation.

```
void foo(i) { int a = i;}

cilk_for( int i=0; i<n; i++) {
        foo(i);
}
```

c) Compute Work, Span and Parallelism of the following code. Explain your computation. What lessons do you learn from b) and c)?

```
for( int i=0; i<n; i ++) {
        cilk_spawn foo(i);
}
cilk_sync;
```

4) Synchronous computations (10%)

a) What is the difference between fully synchronous and locally synchronous computations? Name one example for each.

b) Outline local synchronization using message passing between a process P_i and its two logical neighbors P_{i-1} and P_{i+1}. Pseudocode is sufficient.

c) How can deadlocks be avoided in local synchronization? Describe at least one technique.

5) GPGPU computing with CUDA (15%)

a) Discuss the differences between CPU and GPU threads.

b) Discuss the GPU thread synchronization mechanisms that CUDA offers to the programmer. (I.e. synchronization between threads running on the GPU, not synchronization between threads running on the CPU and threads running on the GPU).

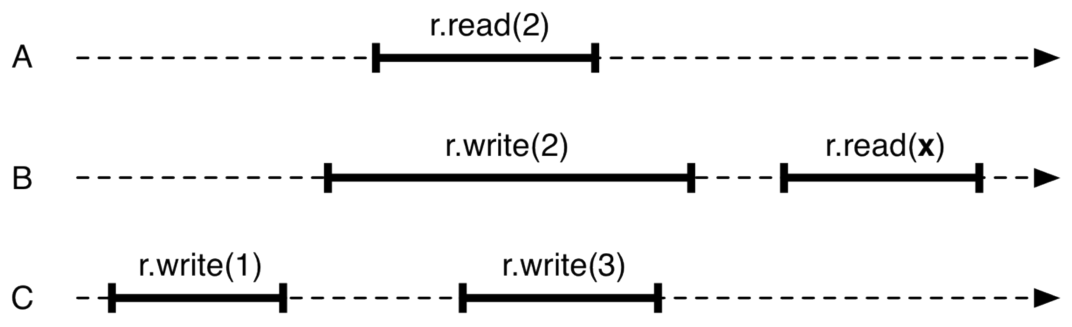
c) What is coalesced memory access on a GPU, and how can it be used? How much of a performance (bandwidth) benefit do we get when using it to the best of its abilities, compared to not being able to use it at all?

6) Concurrent objects (15%)

Three processes A, B, and C access a shared register r as shown in the figure below. In the execution, process C writes the value 1 to the register. Then, processes A, B, and C overlap in reading the value 2, writing the value 2, and writing the value 3. Finally, process B reads value **x**.

a) Which values (i.e., “1”, “2”, or “3”) can **x** assume so that the depicted execution history is linearizable? Explain your answer.

b) Which values can **x** assume so that the depicted execution history is sequentially consistent? Explain your answer.



7) Transactional memory (15%)

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?

c) Discuss the differences between Encounter Order Locking and Commit Time Locking in locking software transactional memory systems.

d) What is a zombie transaction? Why is it difficult to handle zombie transactions? Can we avoid having to deal with zombie transactions altogether?

Good luck!