Linköpings universitet
IDA Department of Computer and Information Sciences
Prof. Dr. Christoph Kessler

# TENTAMEN / *EXAM*

# TDDD56
## Multicore and GPU Programming
20 apr 2017, 14:00–18:00, TER3

**Jour:** Christoph Kessler (070-3666687, 013-282406), visiting ca. 16:00.
Ingemar Ragnemalm (070-6262628), visiting ca. 16:00.

**Hjä lpmedel /** *Admitted material:*

– Engelsk ordbok / *Dictionary from English to your native language*

## General instructions

- This exam has 9 assignments and 4 pages, including this one.
  Read all assignments carefully and completely before you begin.
- It is recommended that you use a new sheet of paper for each assignment, because they will be corrected by different persons.
  Sort the pages by assignment, number them consecutively and mark each one on top with your exam ID and the course code.
- You may answer in either English or Swedish. **English is preferred** because not all correcting assistants understand Swedish.
- Write clearly. Unreadable text will be ignored.
- Be precise in your statements. Unprecise formulations may lead to a reduction of points.
- Motivate clearly all statements and reasoning.
- Explain calculations and solution procedures.
- The assignments are *not* ordered according to difficulty.
- The exam is designed for 40 points. You may thus plan about 5 minutes per point.
- Grading: U, 3, 4, 5. The preliminary threshold for passing is 20 points.
- We expect to have the exam corrected by *end of january*. An exam review session will be announced on the course homepage.

1. (4 p.) **Multicore Architecture Concepts**

   (a) Compared to traditional single-threaded processors and cores, for what kind of computations can we expect further speedup due to *hardware multithreading*? Motivate your answer. (1p)

   (b) Define and explain the following technical terms:

      i. SIMD (vector) instructions
      ii. Cache coherence protocol
      iii. Weak memory consistency (in a shared memory system)

      *(Remember that an example is not a definition. Be general and thorough.)* (3p)

2. (7 p.) **Design and Analysis of Parallel Algorithms**

   (a) Why does it make sense to start the design of a new parallel program for a modern multicore processor or GPU with looking for suitable algorithms based on the PRAM model? (1p)

   (b) How could knowing the *working set size* of a (parallel) algorithm possibly help to explain the reason of a speedup anomaly observed with a multicore program? (1p)

   (c) Define the following two properties:

      - Critical path length
      - Parallel work

      of a parallel algorithm, and explain (calculation) how they are related to its parallel execution time through Brent's Theorem. (2.5p)

      (Hint: Make sure to properly introduce all symbols used and explain their meaning.)

   (d) Recall that the *prefix sums* vector $y = (y_1, ..., y_n)$ of an input vector $(x_1, ..., x_n)$ is defined by

   $$y_i = \sum_{j=1}^{i} x_j$$

   for $i = 1, 2, ..., n$ (here, for inclusive prefix sums).

   We learned about several *parallel* algorithms for calculating the *prefix sums* of an array of $n$ elements using (up to) $n$ processors.

   Name and describe (using pseudocode or a well explained annotated picture) one such parallel algorithm of your choice that only performs $O(\log n)$ time steps using $n$ processors (for an arbitrary PRAM model), and explain why its time complexity is $O(\log n)$. (2.5p)

3. (3 p.) **Parallel Programming with Threads and Tasks**

   (a) What is the purpose of back-off strategies in mutex-lock implementations? (1p)

   (b) How does a work-stealing task scheduler work? And for what purpose should it be used? (2p)

4. (7 p.) **Non-blocking Synchronization**

   (a) In the lecture, we considered a *fair lock* implementation using the atomic `FetchAndIncr` operation:

```
// two shared counters, statically initialized to 0:
shared int ticket = 0;  // next waiting ticket to grab
shared int active = 0;  // ticket now entitled to enter CS

void acquire()   // acquire fair lock:
{
  int myticket = FetchAndIncr( &ticket, 1 );
  while (myticket != active)
     ;  // busy waiting
}

void release()   // release fair lock:
{
  active ++;
}
```

   (This implementation assumes a sequentially consistent memory.)

   (i) The busy-waiting `while` loop above contains a potential performance issue on standard (cache-based, sequentially consistent) multicore CPUs. Explain why, and describe one possible workaround. (2p)

   (ii) You are now given a multicore processor that has no atomic *fetch_and_increment* instruction but that has a *compare_and_swap* (CAS) instruction instead. Rewrite the above fair lock implementation using CAS such that the behavior is the same. Explain your solution. (2.5p)

   (iii) Can the ABA problem occur in your CAS-based implementation of the fair lock? Explain why or why not. If yes, how likely is it to occur? Motivate your answer. (2p)

   (b) Do you know another kind of hardware atomic operation that can be used as an (at least equally powerful) alternative to CAS and that does not suffer from the ABA problem? (technical term only, no details) (0.5p)

*[In case of questions about the following 3 assignments, ask I. Ragnemalm in the first hand.]*

5. (5 p.) **GPU Algorithms and Coding**

   (a) Describe, using code or pseudo code, how to transpose large symmetrical matrices efficiently on the GPU. (2p)

   (b) Describe, with code or pseudo code, how reduction can be used to calculate the maximum value of a large array of scalar values on a GPU. (3p)

6. (5 p.) **GPU Conceptual Questions**

   (a) Describe how Bitonic Merge Sort can be implemented on a GPU. A figure to clarify the algorithm is expected. Your solution must be able to handle large data sets (i.e. 100000 items or more). (3p)

   (b) Why can coalescing improve performance? How can you rewrite an algorithm with non-coalesced memory access patterns to take advantage of coalescing? (2p)

7. (5 p.) **GPU Quickies**

   (a) In CUDA, you can use the modifiers `__global__` and `__device__`. What is the difference between them? (1p)

   (b) What kind of algorithms benefit from using constant memory? (1p)

   (c) What concept does a streaming multiprocessor correspond to in CUDA and OpenCL, respectively? (1p)

   (d) Some operations, like image filters, can be implemented using scatter or gather algorithms. If you use scatter, what specific operation must be used to make it work correctly? (1p)

   (e) State one advantage with CUDA/OpenCL over fragment shader based GPU computing. (1p)

8. (2 p.) **Optimization and Parallelization**

   (a) What is a *loop-carried data dependence*? (0.5p)

   (b) Name and shortly describe a loop transformation that can improve the cache hit rate of a loop, and explain why. (1.5p)

9. (2 p.) **Parallel algorithmic design patterns and High-level parallel programming**

   Explain the following parallel algorithmic design patterns. In particular, explain for each of them the dependence structure and where the parallelism comes from.

   (a) Pipelining

   (b) Data parallelism

Good luck!