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

# TENTAMEN / *EXAM*

# TDDD56

## Multicore and GPU Programming

12 January 2023, 14:00–18:00, TER4

**Jour:**

– Christoph Kessler (070-3666687, 013-282406), visiting ca. 16:00

– Ingemar Ragnemalm (070-6262628), for Questions 5–7, visiting ca. 16:00

**Hjälpmedel /** *Admitted material:*

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

## General instructions

- This exam has 8 assignments and 5 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 may 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 ca. *26 january*. An exam review session at end of january will be announced on the course homepage.

1. (5 p.) **Multicore Architecture Concepts**

   (a) What is the technical difference between *hardware multithreading* and *software multithreading*? (1p)

   (b) Define and explain the following technical terms:

      i. MIMD parallelism
      ii. (Cache) capacity miss
      iii. Sequential (memory) consistency
      iv. Heterogeneous multicore system

      *(Remember that an example is not a definition. Be general and thorough.)* (4p)

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) Define the following two properties:

      - Critical path length
      - Parallel work

      of a parallel algorithm, and explain (commented formula) how they are related to its parallel execution time with $p$ processors through Brent's Theorem. (2p)
      (Hint: Make sure to properly introduce all symbols used and explain their meaning.)

   (c) What is the difference between a *parallel speedup anomaly* and *superlinear speedup*? (1p)

   (d) **Parallel Sorting** (3p)
      (Assume here for simplicity that the available overall number of (PRAM) processors is a power of 2.)
      In the parallel divide-and-conquer algorithm *(fully) parallel mergesort* (and likewise, in the algorithm *bitonic sort*) as presented in the lecture, the set of all $n > 1$ (PRAM) processors executing a call (such as `par_mergesort` $(A[0..n-1])$ ) are, if not at a base-case, split up *evenly* into two disjoint subgroups of equal sizes for the *parallel* processing of the two recursive calls, i.e., each recursive call internally uses a subgroup of $n/2$ processors.
      (i) Why is this 50:50 ratio the optimal split ratio for these algorithms? (1p)
      (ii) Would the same strategy also be advisable for *(simple) parallel quicksort*? If yes, why? If not, what is different with parallel quicksort, and how (commented formula) should you instead split up the processors in each recursive step, in order to minimize overall parallel time? Motivate your answer. (2p)

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

   (a) What does *thread pinning* mean? (1p)

   (b) We have seen in the lecture that mutex-lock acquire operations on spin-locks can be implemented using atomic test&set instructions. Describe shortly the idea and purpose of using *back-off strategies* in mutex-lock acquire operations. (1p)

   (c) How does a work-stealing task scheduler work? (1p)

4. (8 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) Consider the non-atomic read-modify-write operation `active ++` in the `release` function. Why don't we need to protect it against data races here? (1p)

(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) What is the *ABA problem* (in general) that can occur with CAS operations? (1p)

(iv) 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. (1p)

(b) Obviously, heap memory allocation for multithreaded programs must be thread-safe. Why should lock-free concurrent dynamic data structures (allocated on the heap) preferably be used with a lock-free heap memory allocator instead of traditional lock-based `malloc`/`free` implementations? (1p)

(c) What is the difference between a *lock-free* and a *wait-free* concurrent data structure? Which of them is the stronger property, and where could that difference matter? (1.5p)

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

*Note from Ingemar Ragnemalm: In all GPU questions, you may use CUDA or OpenCL style code as you please, but CUDA style is recommended. Exact syntax is not important.*

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

   (a) You need to perform sound processing in an efficient manner, applying sound effects on very large sounds. The sounds are applied using a convolution kernel, which can be up to 1000 samples long, 16 bits each. The entire sound consists of millions of samples. Describe, with pseudo code, how to perform this operation efficiently on the GPU. (3p)

   (b) You are given the task is to produce a histogram of an image. The input are millions of pixels (grayscale, 1 byte per pixel), and the output is a 256 32-bit integer array with the count of each grayscale value. Describe an approach to perform this efficiently on the GPU. (2p)

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

   (a) Describe the major architectural differences between a multi-core CPU and a GPU (apart from the GPU being tightly coupled with image output). Focus on the differences that are important for parallel computing. (3p)

   (b) Some image filters are *separable*. This has potential to improve performance, but it also comes with a limitation. Two tasks work against each other. Describe when and why it could be an advantage to split the filter in two and when it is not (granted that you have 2D filters that are separable). (2p)

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

   (a) Give an example of why you may want to make a *device query*. (1p)

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

   (c) Why is *load balancing* often not a (big) problem in GPU computing, e.g. when computing fractals? (1p)

   (d) Texture access provides two unique features that we otherwise do not have. Name one, and describe with a brief sentence. (1p)

   (e) When performing neighborhood operations like filtering on a GPU, you can use either *gather* or *scatter* operations. Which one would you recommend, and why? (1p)

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

   (a) Consider the following loop nest:

```
for i = 1, ..., M
   for j = 1, ..., N-1
      A[i][j] = x*A[i-1][j-1] + y*A[i-1][j] + z*A[i-1][j+1];
```

   (i) Would it be correct to apply *loop interchange* to this loop nest? Justify your answer (dependence-based argument). (1p)

   (ii) If loop interchange would be applied, would it be likely to improve performance on a cache-based architecture if `N` is large? Justify your answer. (1p)

Good luck!