# Information page for written examinations at Linköping University

| | |
|---|---|
| **Examination date** | 2019-08-31 |
| **Room (1)** | TER2(10) |
| **Time** | 14-18 |
| **Edu. code** | TDDD56 |
| **Module** | TEN1 |
| **Edu. code name**<br>**Module name** | Multicore and GPU Programming (Multicore- och GPU-Programmering)<br>Written examination (Skriftlig examination) |
| **Department** | IDA |
| **Number of questions in the examination** | 8 |
| **Teacher responsible/contact person during the exam time** | Christoph Kessler (070-3666687, 013-282406), visiting ca. 16:00, if possible.<br>Ingemar Ragnemalm (070-6262628), via phone. |
| **Contact number during the exam time** | 013-282406 |
| **Visit to the examination room approximately** | 16:00 |
| **Name and contact details to the course administrator** (name + phone nr + mail) | Veronica Kindeland Gunnarsson, IDA, 013-285634, veronica.kindeland.gunnarsson@liu.se |
| **Equipment permitted** | Engelsk ordbok / Dictionary from English to your native language. |
| **Other important information** | See the general instructions on the first page. |
| **Number of exams in the bag** | |

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

# TENTAMEN / *EXAM*

## TDDD56

## Multicore and GPU Programming

31 aug 2019, 14:00–18:00, TER2

**Jour:** Christoph Kessler (070-3666687, 013-282406), visiting ca. 16:00 if possible.
Ingemar Ragnemalm (070-6262628), for Questions 5–7, by phone only.

**Hjälpmedel** / *Admitted material:*

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

## General instructions

- This exam has 8 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 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.
- There is no exam review session for re-exams. After grading, the exams will be archived and available for inspection in the IDA student expedition (E-house, upper floor).

1. (6 p.) **Multicore Architecture Concepts**

   (a) Define and explain the following technical terms:

      i. ILP (instruction-level parallelism) wall (for single-threaded CPUs)
      ii. SIMD parallelism
      iii. (Cache) cold miss (also known as mandatory miss)
      iv. Write-back cache
      v. Bus snooping (for coherence)

      *(Remember that an example is not a definition. Be general and thorough.)* (5p)

   (b) Why does the MSI coherence protocol, as described in the lecture, guarantee sequential (memory) consistency? (1p)

2. (9.5 p.) **Design and Analysis of Parallel Algorithms**

   (a) Define the term *relative parallel efficiency* $E_{rel}(p)$ (of a parallel program) as a commented formula depending on the number $p$ of processors used, and give its interpretation (i.e., what does a high or low value of $E_{rel}$ mean in practice). (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 (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.)

   (d) **Parallel sorting algorithms**

      In the lecture on parallel sorting we took a closer look at, among others, the three algorithms *fully parallel quicksort*, *fully parallel mergesort* and *bitonic sort*. Choose your favorite one among the three algorithms (state which one), describe it with its main subroutines by well-explained pseudocode and a flowchart diagram, and analyze (i.e., derive by calculation, results given in big-O notation) its *parallel time*, *parallel work*, *maximum number of processors used at any time* and *parallel cost* for a problem size of $n$ elements on an ideal parallel computer (CRCW PRAM) with an unlimited number of processors. If you need to make further assumptions, state them carefully. (5.5p)

3. (2.5 p.) **Parallel Programming with Threads and Tasks**

   (a) Spin-locks can, as we know, be implemented using the atomic *test-and-set* instruction in the *mutex_lock* operation. What is the purpose of instead using a *test-test-and-set* (TTAS) strategy for acquiring a spin-lock? Explain how it works. (1p)

   (b) How does a work-stealing task scheduler work, and for what purpose can it be used? (1.5p)

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

   (a) The operation $y = \mathit{fetch\_and\_add}(p, a)$ atomically adds to the memory location pointed to by $p$ the integer value $a$ and returns the (integer) value $y$ that the location had immediately before the update.

   You are given a multicore processor that has no atomic *fetch_and_add* instruction but that provides an atomic *compare_and_swap* (CAS) instruction instead. Write a software implementation of atomic *fetch_and_add* using CAS. Explain your solution. (2p)

   (b) Explain the operations Load-Linked (LL) and Store-Conditional (SC), and explain how the behavior of LL+SC differs from that of CAS if used for atomic updating of a shared memory location in the context of lock-free shared data structures. If you had the choice between LL+SC and CAS, which one would you prefer, and why? (2p)

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

   (a) Describe, using figures and pseudo code in sufficient detail (full CUDA/OpenCL code is not needed), how matrix multiplication of large matrices can be implemented on the GPU. (GPU kernel code only.) Emphasize the most vital considerations for good performance. (3p)

   (b) Describe how an image filter can be efficiently implemented on a GPU. You can not assume that the filter is separable. Use figures and an overview of the algorithm in text. (2p)

6. (5p) **GPU Architecture concepts**

   (a) List three different kinds of GPU memory and describe for each their characteristics in terms of performance, usage and accessibility. CUDA terminology is assumed, please note if you use OpenCL terminology. (3p)

   (b) Some algorithms run faster if you pad a few extra bytes at regular intervals in shared memory. Explain why. (2p)

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

   (a) Suggest a feature that texture memory can do automatically which other kinds of memory access can not. (1p)

   (b) Explain (briefly) why the G80 architecture had significantly higher performance than earlier GPUs. (1p)

   (c) In graphics, data is always input as geometrical shapes. What geometry is usually used for fragment shader based GPU computing? (1p)

(d) OpenGL Compute Shaders has a limitation compared to CUDA or OpenCL. Which one? (1p)

(e) When using separable filters, one filter is likely to run faster than the other one. Why? (1p)

8. (3 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 *tiling* (if applicable) of this loop nest be beneficial for performance if `N` is large? Justify your answer. (1p)

(ii) Is *tiling* of this loop nest (e.g. with $2 \times 2$ tiles) applicable here, or would it change the semantics of the code? Justify your answer. (1p)

(b) Why is it, in general, so hard for C/C++ compilers to statically analyze a given sequential legacy program and parallelize it automatically? (1p)

Good luck!