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

# TENTAMEN / *EXAM*

## TDDD56
### Multicore and GPU Programming

31 march 2016, 14:00–18:00, TER1

**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 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 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.
- An exam review session will be announced on the course homepage.

1. (5.5 p.) **Multicore Architecture Concepts**

   (a) Define and explain the following technical terms:

      i. SIMD parallelism
      ii. Moore's Law
      iii. (Cache) capacity miss
      iv. Heterogeneous multicore system

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

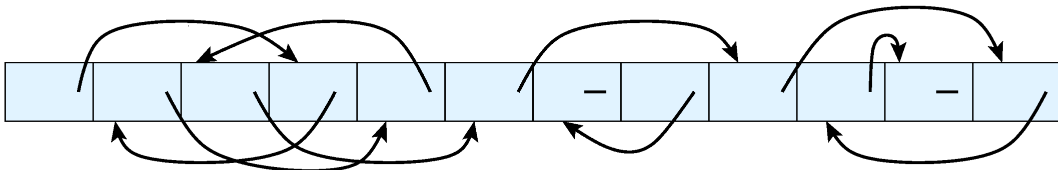
   (b) Why does the MSI coherence protocol, as described in the lecture, guarantee sequential (memory) consistency? (1.5p)

2. (5.5 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) Give an example of a *parallel speedup anomaly* that may occur on a multicore computer, and explain its technical cause. (1p)

   (c) (3.5p) In the lecture, we introduced *list-ranking* as a fundamental computational problem on a shared array containing $N$ list items that belong to one or several linked lists. The simplest scenario considered was calculating, for each list item, a pointer to the end of its list.



   (i) For this problem scenario, describe (pseudocode) and explain the *recursive pointer doubling* based algorithm using $N$ parallel threads, as introduced in the lecture, and derive its asymptotic parallel time, work and cost complexity (in terms of $N$). (3p)

   (ii) What kind of parallelism is exploited here? (0.5p)

   (d) **Bonus question (+1.5p):** Is this pointer-doubling algorithm *work-optimal*? Explain why or why not (NB a simple yes/no answer gives no points).

   (Hint: This requires that you have solved the previous question correctly. For the answer you may remember your course on data structures and algorithms...)

3. (4 p.) **Parallel Programming with Threads and Tasks**

   (a) How does a work-stealing task scheduler work? (2p)

   (b) How does programming with `futures` work? (2p)

4. (6 p.) **Non-blocking Synchronization**

   (a) Name 2 problems of lock-based synchronization that are removed by non-blocking synchronization. (1p)

   (b) An *ordered linked list* (here, for integers) uses list items of the following type:

```
struct elem {
  int value;
  struct elem *next;
}
```

   Pointer variable `head` points to the first list element. Insertion of a new element with value $v$ is done by searching for $v$ and inserting a new list element `e` for $v$ after the element where the search ended. As a simplification we assume that the list always contains at least one element, namely an artificial dummy element with value $-\infty$ as the first element in the list. We also assume for simplicity that *no remove operations* occur.

```
struct elem *e = (struct elem *)malloc(sizeof(struct elem));
struct elem *p = head;
// assume for simplicity that the list contains at least 1 element
// (the first element is a dummy element with value -infinity)
while (p->next!=NULL && p->next->value < v)
  p = p->next;
e->value = v;
// insert e into the list after p:
e->next = p->next;
p->next = e;
```

   Assume that the list elements and the `head` pointer are stored in shared memory.

   i. Show that, without proper synchronization, two concurrent insertions may lead to an incorrect result. (Give a simple scenario, start with a one-element list).
      In general, what is the condition (on concurrently inserted values and list contents) for such a conflict to occur?
      Suggest a simple mutex-lock-based solution to make insertion thread-safe. (1.5p)

   ii. Use appropriate CAS operations (i.e., no mutex locks) to provide a non-blocking *insert* operation (pseudocode). Explain your code.
      Explain how possible conflicts between concurrent *insert* operations are recognized and handled properly by your implementation.
      In particular, argue why even in the case of conflicts at least one of the conflicting operations will succeed. (3p)

   (c) 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) A Mandelbrot algorithm is given as sequential code as follows:

   ```
   for (int x = 0; x < SIZE; x++)
       for (int y = 0; y < SIZE; y++)
           data[x, y] = computeFractal(x, y);
   ```

   that is, the fractal computing code is already available. How can you port this to an efficient GPU implementation? Outline vital parts of the code. (2p)

   (b) Describe, in pseudo code and figures, how an optimized matrix multiplication can be performed on the GPU. Your answer should focus on structure and vital features rather than detailed code. (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). GPU specific aspects should be addressed. (3p)

   (b) 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. Constant memory should not be included since that is a separate question below. (2p)

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

   (a) In CUDA, you can use the modifier `__global__`. What does this signify? (1p)

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

   (c) If you want to process a large array in fragment shader based computing, how will that data typically be represented? (1p)

   (d) Suggest one important feature in GPUs that was added for performing some specific graphics effect. Name the effect too. (1p)

   (e) When do you typically need to synchronize threads? (1p)

8. (3 p.) **Optimization and Parallelization**

   (a) Name and shortly describe two different loop transformations that can improve the cache hit rate of a loop, and explain why. (3p)

9. (1 p.) **Parallel algorithmic design patterns and High-level parallel programming**

   Give two main advantages and two main drawbacks of *skeleton-based parallel programming*. (1p)

Good luck!