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

TENTAMEN / *EXAM*

TDDD56
Multicore and GPU Programming

14 jan 2016, 14:00–18:00, U6, U7, U10, U11

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

   (a) Define and explain the following technical terms:

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

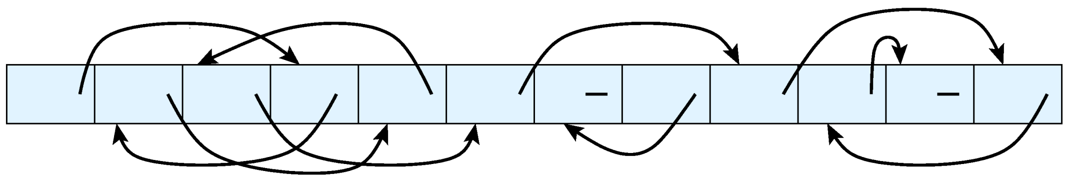
      (Remember that an example is not a definition. Be general and thorough.) (4p)

2. (6.5 p.) **Design and Analysis of Parallel Algorithms**

   (a) Formulate Brent's Theorem (explained formula), give its interpretation, derive it (proof), and describe its implications for parallel algorithm design and analysis. (3p)

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

   (b) (3p) 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)

   (c) **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. (3.5 p.) **Parallel Programming with Threads and Tasks**

   (a) What does *thread pinning* mean, and what is its purpose? (1.5p)

   (b) How does a work-stealing task scheduler work? (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)

(b) 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)

(c) Do you know another kind of hardware atomic operation that can be used as an 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**

   Write code/pseudocode for computing a 2-dimensional color image filter of size 5×5 pixels in a reasonably optimized way. Clearly describe what optimizations you do and why. The filter weights should be specified (i.e. a 5×5 matrix), and should be normalized properly. Full score requires a close-to-real-code solution taking more than one optimization technique into account. You may use CUDA-style syntax or OpenCL-style syntax as you please.

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) Describe how reduction can be used to calculate the maximum value of a large array of scalar values on a GPU.
   Also give at least two examples of other problems that are solved by reduction. (2p)

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

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

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

   (c) Compare OpenCL and Compute Shaders in terms of portability. You should know at least one strong point of each. (1p)

   (d) Imagine a CUDA programmer who uses the practice to always use as big block size as possible. Why will this not always result in the highest possible performance? (1p)

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