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

TENTAMEN / *EXAM*

TDDD56
Multicore and GPU Programming

12 jan 2017, 14:00–18:00, KÅRA, U14, U15

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

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.
- We expect to have the exam corrected by *end of january*. An exam review session will be announced on the course homepage.

1. (5 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. MIMD parallelism
      ii. (Cache) capacity miss
      iii. Cache coherence protocol
      iv. Heterogeneous multicore system

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

2. (6 p.) **Design and Analysis of Parallel Algorithms**

   (a) How does the *working set size* of a (sequential) algorithm relate to the expected amount of cache capacity misses? (1p)

   (b) 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.)

   (c) 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. (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)
   (Hint: If you use a different CAS function than the one used in labs, you should carefully explain the function that you use as compare and swap: what are its parameters, what does it return and to what pseudo-code it is equivalent, using an `atomic{}` keyword/construct or another equivalent construct that needs to be defined.)

   (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. (6p.) **GPU algorithms and Coding**

   (a) A large matrix is given, stored in global GPU memory. Describe, using code or pseudo code, an efficient way to transpose it on the GPU. The transposing does not have to be done in-place. Vital features of the algorithm should be clearly stated. (3p)

   (b) The following algorithm (given as OpenCL code) performs rank sort on the GPU, a simple but not very efficient sorting algorithm for data with unique keys. However, it has a bug, plus, it can be significantly accelerated.

```
__kernel void sort( __global unsigned int *data,
                    const unsigned int length )
{
  unsigned int pos = 0;
  unsigned int i;
  unsigned int val;

  //find out how many values are smaller
  for (i = 0; i < get_global_size(0); i++)
    if (data[get_global_id(0)] > data[i])
      pos++;

  val = data[get_global_id(0)];
  data[pos]=val;
}
```

   (c) What is the bug? (1p)

   (d) Describe a way to accelerate the code. (2p)

6. (4p.) **GPU Conceptual Questions**

   (a) Outline how reduction is implemented in an efficient way, using text and figures. You may assume that the reduction problem in question deals with finding the maximum of a large dataset. Assume that the dataset can be of highly varying size, including very large. (2p)

   (b) Three important kinds of GPU memory include *shared* (*local*), *global* and *texture* memory. Describe these in terms of performance, usage and accessibility. CUDA terminology is assumed, please note if you use OpenCL terminology. (2p)

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

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

   (b) What particular algorithm feature makes bitonic merge sort particularly suitable for parallel implementation? (1p)

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

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

(e) Can you rely on any threads/work groups in a GPU computation to be literally executed in parallel? If so, which ones? (1p)

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

   (a) What is an *anti-dependence*? (0.5p)

   (b) Give a sufficient condition (dependence based argument) that a loop's iterations can be executed in parallel. (1p)

   (c) Name and shortly describe a loop transformation that can help a compiler to effectively utilize SIMD instructions. (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) Stencil computation

   (b) Streaming

Good luck!