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

TENTAMEN / *EXAM*

TDDD56
Multicore and GPU Programming

20 dec 2012, 14:00–18:00 TER4

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

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 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 *mid of january* 2013. An exam review session will be announced on the course homepage.

1. (7 p.) **Multicore Architecture Concepts**

   (a) There are three main technical limits that prevent a further significant increase of single-thread performance, which finally led to the development of multicore architectures. Name and explain two of these limits. (2p)

   (b) Define and explain the following technical terms:

      i. SIMD (vector) instructions
      ii. Symmetric multiprocessor (SMP)
      iii. Hardware multithreading
      iv. Bus snooping
      v. Sequential (memory) consistency

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

2. (7 p.) **Non-blocking Synchronization**

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

   (b) In a *doubly linked list*, each list element has two pointers, `prev` and `next`, that point to the preceding and subsequent list element respectively. For a multithreaded execution environment (e.g., pthreads), give a thread-safe solution using ordinary mutex locks (pseudocode) to protect concurrent insertion of elements into a doubly linked list in shared memory. (0.5p)

   Give a simple argument (sketch of a scenario) that concurrent insertion without the mutex protection can lead to an incorrect result. (0.5p)

   (c) Assume now that you are given a multicore processor that provides a *double-word compare and swap (DCAS)* instruction:

   int *DCAS*(`struct ddescr` * *pd*);

   where the DCAS descriptor stored in thread-local memory, referenced by pd, has the following structure:

```
struct ddescr {
  word *ptr1, *ptr2;  // pointers to two shared memory locations
  word old1, old2, new1, new2;  // old and new values for these
  word res;  // error code
}
```

   If the DCAS instruction applied to pd succeeds, its effect is the same as two (successful) simultaneous CAS (compare-and-swap) operations `CAS(ptr1,old1,new1)` and `CAS(ptr2,old2,new2)` executed atomically, and it returns 0 (no error).

   If either of the two CAS operations cannot be committed, both will not take effect, and the DCAS instruction returns an error code.

   Write a non-blocking implementation (pseudocode) of concurrent insertion in a doubly linked list, using DCAS instead of locks. Explain your code.

   Explain how your solution will handle cases of conflicting concurrent insertions (see your counterexample in the previous question) correctly. (4p)

   (d) What is the *ABA problem* (in the context of CAS operations)? (1p)

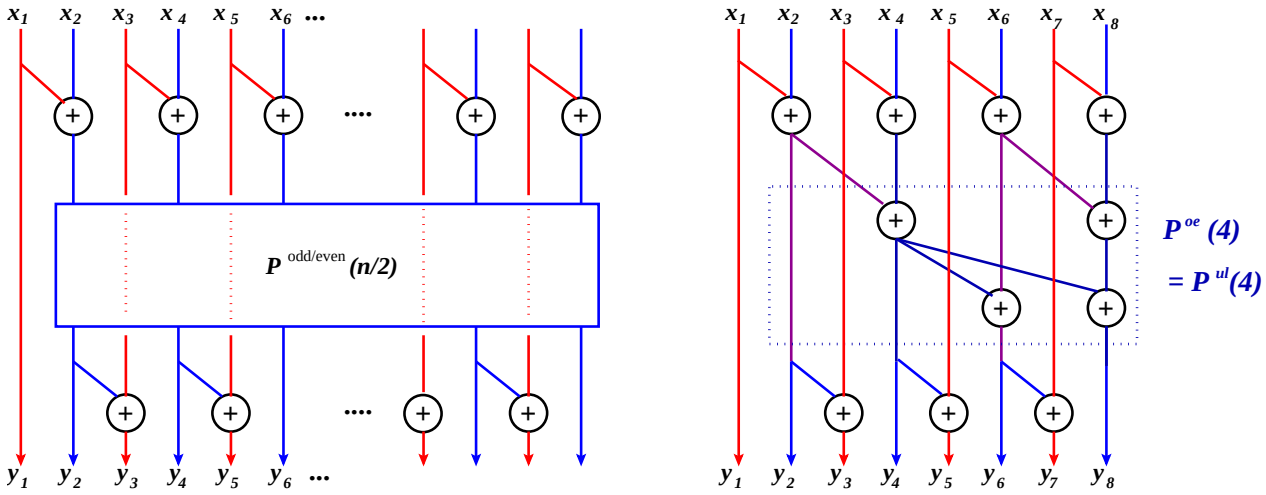

Figure 1: The odd-even parallel prefix sums algorithm. Left: the general recursive case. Right: Recursion unrolled for $n = 8$; the base case for $n = 4$ is taken from lower-upper parallel prefix sums.

3. (7 p.) **Design and Analysis of Parallel Algorithms**

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

   (b) *Odd-even parallel prefix sums*

   The odd-even parallel prefix sums algorithm (see Fig. 1) transforms a prefix sums problem of size n into one subproblem of size $n/2$, solves that one recursively and re-transforms its solution into a solution of the original problem instance. For problems of size ≤ 4, for instance the upper-lower algorithm from the lecture (as in the figure on the right) or the sequential algorithm can be taken as base case.

   i. Even though it is a degenerated special case, the high-level structure of the algorithm follows a well-known (parallel) algorithmic paradigm. Which one? (0.5p)
   (Hint: it is not "*recursion*")

   ii. Analyze where there is parallelism in the algorithm (and when, and how much) and what kind of parallelism it is, and express it appropriately in parallel pseudocode. (2p)

   iii. Analyze the algorithm for its *parallel execution time*, *parallel work* and *parallel cost* (each as a function in n, using big-O notation) for a problem size n using up to n processors. Explain your calculations. Assume that the base case for $n \leq 4$ can be solved on one or a few processors using constant time and work. (1.5p)

   iv. Is the algorithm (asymptotically) *work-optimal*? Justify your answer (calculation). (1p)

   v. Given n processing elements, is the algorithm (asymptotically) *cost-optimal*? (calculation)
   If yes, explain why.
   If not, sketch a way to reduce its cost asymptotically. (1p)

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

4. (5 p.) **GPU Algorithms**

   Matrix transposing is an operation with no computations, but its efficiency depends heavily on a certain GPU computing feature. Which feature, why is it important? With code or pseudo code, describe an efficient way to implement matrix transposing. (5p)

   (Hint: The operand matrix is given in global device memory of the GPU. The matrix transpose needs not be done in-place.)

5. (5 p.) **GPU Concepts**

   (a) Motivate why GPUs can give significantly better computing performance than ordinary CPUs. Is there any reason to believe that this advantage will be reduced over time? (2p)

   (b) Describe how computing is mapped onto graphics in shader-based computing (expressed as kernel, input data, output data and iterations over the same data). What limitations are imposed on your kernels compared to CUDA or OpenCL? (3p)

6. (5 p.) **GPU Quickies**

   (a) Describe how multiple CUDA streams can be used to accelerate a computation. (1p)

   (b) Translate the following CUDA concepts to corresponding concepts in OpenCL: (1p)

      i. shared memory
      ii. block
      iii. thread

   (c) Texture memory provides interpolation in hardware. Why is this a questionable feature to rely on? (1p)

   (d) Give one argument each in favor of using

      i. CUDA
      ii. OpenCL
      iii. GLSL

      for a general computing task (which can benefit from a parallel implementation).

   (e) When can constant memory give a performance advantage? (1p)

7. (2 p.) **Optimization and Parallelization**

   (a) Consider the following sequential loop:

```
#define N 100000
double x[N];  // array of double precision floats
...
for (i=8; i<N; i++) {
   x[i] = veryexpensivefunction( x[i-8] );
}
```

   i. Can the above `for` loop be simply rewritten into a parallel `forall` loop? Give a formal argument. (1p)

   ii. **Bonus question (optional):** Suggest a method to parallelize the computation as far as possible.
   (Hint: Assume shared memory and a multithreaded execution environment. Draw the iteration dependence graph. Consider only parallelism, ignore possible cache performance problems for now. Assume that the time taken by `veryexpensivefunction` is large but independent of its argument. Do you need special synchronization?)
   Show the resulting pseudocode. How much speedup can you expect in the best case? (+2p)

   (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)

8. (2 p.) **Parallel algorithmic design patterns and High-level parallel programming**

   Give two main advantages and two main drawbacks of *skeleton-based parallel programming*.

Good luck!