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

TENTAMEN / *EXAM*

TDDD56

Multicore and GPU Programming

26 mar 2013, 14:00–18:00 T1

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 7 assignments and 6 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.

1. (8 p.) **Multicore Architecture Concepts**

   (a) What is the *power wall* in (single-core) processor architecture, and why does it lead to the development of multicore architectures? Give a quantitative argument. (2p)

   (b) Define and explain the following technical terms:

      i. SIMD (vector) instructions
      ii. Moore's Law
      iii. Cache coherence
      iv. Bus snooping
      v. Weak memory consistency (in a shared memory system)

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

   (c) What is the purpose of multi-banked memory? (1p)

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) Give an example scenario of the *ABA problem* (in the context of CAS operations). (1p)

Hint: You might use the above linked list scenario or create your own one.

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

Karatsuba Polynomial Multiplication (here, just for binary numbers)

Consider the problem of multiplying two very large binary numbers $x = \langle x_0, ..., x_{n-1}\rangle$ and $y = \langle y_0, ..., y_{n-1}\rangle$ given as bitvector arrays.

The (sequential) school method for this is to separately multiply x with each bit y_i, shifted by i positions, and adding up these partial products, resulting in an algorithm with $O(n^2)$ work. But we can do better.

We can write $x = x^{(1)} \cdot 2^{n/2} + x^{(0)}$ where $x^{(0)} = \langle x_0, ..., x_{n/2-1}\rangle$ and $x^{(1)} = \langle x_{n/2}, ..., x_{n-1}\rangle$, and $y = y^{(1)} \cdot 2^{n/2} + y^{(0)}$ accordingly. Then,

$$\begin{aligned} x \cdot y &= (x^{(1)} \cdot 2^{n/2} + x^{(0)})(y^{(1)} \cdot 2^{n/2} + y^{(0)}) \\ &= x^{(1)}y^{(1)} \cdot 2^n + \left(x^{(1)}y^{(0)} + x^{(0)}y^{(1)}\right) \cdot 2^{n/2} + x^{(0)}y^{(0)}. \end{aligned}$$

This reduces the problem of one length-n multiplication to, for now, four length-$n/2$ multiplications.

Now we know that $(x^{(1)} + x^{(0)})(y^{(1)} + y^{(0)}) = x^{(1)}y^{(1)} + x^{(1)}y^{(0)} + x^{(0)}y^{(1)} + x^{(0)}y^0$. Hence,

$$x^{(1)}y^{(0)} + x^{(0)}y^{(1)} = (x^{(1)} + x^{(0)})(y^{(1)} + y^{(0)}) - x^{(1)}y^{(1)} - x^{(0)}y^{(0)})$$

which we insert above and obtain

$$x \cdot y = x^{(1)}y^{(1)} \cdot 2^n + ((x^{(1)} + x^{(0)})(y^{(1)} + y^{(0)}) - x^{(1)}y^{(1)} - x^{(0)}y^{(0)}) \cdot 2^{n/2} + x^{(0)}y^{(0)}.$$

Of course we compute $M_1 = x^{(1)}y^{(1)}$ and $M_2 = x^{(0)}y^{(0)}$ only once, so that we can get it done with three length-$n/2$ multiplications and some length-n additions (the latter of which can be done in linear time). The pseudocode is thus:

```
Algorithm Multiply ( array $x[0..n-1]$, $y[0..n-1]$ )
        returns array $z[0..2n-1]$
{
    if $n$ is small ($\leq$ some constant $C$) then multiply $x$, $y$ directly and return $xy$
    else
        Set $x^{(0)}, x^{(1)}$ such that $x = x^{(1)} \cdot 2^{n/2} + x^{(0)}$;
        Set $y^{(0)}, y^{(1)}$ such that $y = y^{(1)} \cdot 2^{n/2} + y^{(0)}$;
        $M_1 \leftarrow$ Multiply($x^{(1)}$, $y^{(1)}$);
        $M_2 \leftarrow$ Multiply($x^{(0)}$, $y^{(0)}$);
        $M_3 \leftarrow$ Multiply($x^{(0)} + x^{(1)}$, $y^{(0)} + y^{(1)}$);
        return $M_1 \cdot 2^n + (M_3 - M_1 - M_2) \cdot 2^{n/2} + M_2$;
    fi
}
```

(a) Which fundamental algorithmic design pattern is used in the Multiply algorithm? (0.5p)

(b) Analyze the sequential time complexity of the Multiply algorithm for a problem size n. Assume that length-n additions and subtractions can be done in time $\Theta(n)$. Assume that a direct multiplication for "small" constant n (e.g., for $n = 1$) takes constant time. (1p)

(c) Identify which calculations could be executed in parallel, and sketch a parallel Multiply algorithm in pseudocode (shared memory). (1p)

(d) Analyze your parallel Multiply 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 n processors. Assume that length-n additions and subtractions can be done in time $\Theta(n)$ and can be parallelized perfectly across up to n threads. Assume that a direct multiplication for "small" n (e.g., $n = 1$) takes constant time.

(A solid derivation of the formulas is expected; just guessing the right answer gives no points.) (3p)

(e) How would you adapt the algorithm to work for a fixed number p of processors? What will then be its parallel time with p processors? (1.5p)

(Hint: $F(n) \leq kF(n/b) + cn$ and $F(C) \in O(1) \implies F(n) \in O(n^{\log_b k} + n)$, for constants $c, C > 0$.)

(Hint: If not otherwise possible, you may answer the above questions for the school method, with half the points.)

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

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

   A *histogram* is an array h that records for each possible (integer) value the number of its occurrences in a large integer-valued data structure, e.g., an array a. It can be computed like this:

```
for all elements i in a[] do
    h[a[i]] += 1
```

   (a) With what feature can this algorithm be made to run in parallel in CUDA? (1p)

   (b) Suggest a different approach that should give good (better) performance and does not rely on this feature (written in ”plain” CUDA, OpenCL or shaders). (2p)

   (c) Write this algorithm in CUDA or OpenCL code. (Minor syntax errors are ignored.) (2p)

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

   (a) Describe the major architectural differences between a multi-core CPU and a GPU (apart from the GPU being tightly coupled with image output). Focus on the differences that are important for parallel computing. (3p)

   (b) Compare shared memory, global memory and register memory in terms of performance, usage and accessibility. CUDA terminology is assumed, please note if you use OpenCL terminology. (2p)

6. **GPU Quickies** (5p; 1p each)

   (a) Why is the G80 so much faster than the previous GPUs (e.g. 7000 series)?

   (b) Why can using constant memory improve performance?

   (c) What kind of shaders is most interesting for GPU computing? (What part of the pipeline?)

   (d) What geometry is usually used for shader-based GPU computing?

   (e) List three different kinds of hardware that OpenCL runs on. (Similar systems by different vendors count as one.)

7. (3 (+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) What is *auto-tuning*, where could it be applied, and what is the main motivation for it? (2p)

Good luck!