Information page for written examinations at Linköping University

Examination date	2015-01-15
Room (2)	TER3 TERE
Time	14-18
Course code	TDDD56
Exam code	TEN1
Course name **Exam name**	Multicore and GPU Computing (Multicore- och GPU-Programmering) Written examination (Skriftlig examination)
Department	IDA
Number of questions in the examination	9
Teacher responsible/contact person during the exam time	Christoph Kessler (070-3666687, 013-282406), visiting ca. 16:00. Ingemar Ragnemalm (070-6262628), visiting ca. 16:00.
Contact number during the exam time	013-282406
Visit to the examination room approximately	16:00
Name and contact details to the course administrator (name + phone nr + mail)	Åsa Kärrman, asa.karrman@liu.se, 013-285760. Carita Lilja, carita.lilja@liu.se, 013-281463.
Equipment permitted	Engelsk ordbok / Dictionary from English to your native language.
Other important information	See the general instructions on the first page.
Number of exams in the bag	

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

TENTAMEN / *EXAM*

TDDD56

Multicore and GPU Programming

15 jan 2015, 14:00–18:00, TERE, TER3

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

Hjälpmedel / *Admitted material:*

– Engelsk ordbok / *Dictionary from English to your native language*

General instructions

- This exam has 9 assignments and 7 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) How do SIMD (vector) instructions work (in hardware)? What kind of parallelism do they exploit, for what kind of computations can they boost processor performance, and what are the conditions for their proper usage on the programmer's (or compiler's) side? (2p)

   (b) Define and explain the following technical terms:

      i. Weak memory consistency (in a shared memory system)

      ii. Heterogeneous multicore system

      (Remember that an example is not a definition. Be general and thorough.) (2p)

2. (4 p.) **Parallel Programming with Threads and Tasks**

   (a) What does *thread pinning* mean, and what is its purpose? (2p)

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

3. (5.5 p.) **Non-blocking Synchronization**

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

(b) 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.)

You are 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)

(c) Explain the operations Load-Linked (LL) and Store-Conditional (SC), and explain how the behavior of LL+SC differs from that of CAS if used for atomic updating of a shared memory location in the context of lock-free shared data structures. If you had the choice between LL+SC and CAS, which one would you prefer, and why? (2p)

4. (7.5 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

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

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 $n \leq C$ (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). (1.5p)

(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" $n \leq C$ (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 (c)–(e) for the school method, with half the points.)

[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/pseudo code for computing a 2-dimensional color image filter of some size (e.g. 5×5) in a reasonably optimized way. Clearly describe what optimizations you do and why. (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) Why can *coalescing* improve performance?
   How can you take advantage of coalescing for an algorithm with a non-coalesced memory access pattern? (2p)

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

   (a) In CUDA, you can use the modifier `__device__`. 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) What does a Streaming Multiprocessor correspond to in CUDA and OpenCL, respectively? (1p)

   (e) In graphics, data is always input as geometrical shapes. What geometry is usually used for fragment shader based GPU computing? (1p)

8. (2.5 p.) **Optimization and Parallelization**

   (a) Consider the following sequential loop:

```
void foo( float *x, float *a, int N )
{
  int i;
  x[0] = a[0];
  for (i=1; i<N; i++)
     x[i] = a[i] + x[i-1];
}
```

   Assume that the arrays `x` and `a` do not overlap. Could the iterations of *this* loop be run in parallel? Justify your answer *(formal argument). (If you need to make additional assumptions, state them carefully.)* (1p)

   (b) How does an auto-tuning library generator work? (1.5p)

9. (1.5 p.) **Parallel algorithmic design patterns and High-level parallel programming**

   What is an *algorithmic skeleton*, and what is the motivation for programming parallel computers using algorithmic skeletons? (1.5p)

Good luck!