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

TENTAMEN / *EXAM*

TDDD56

Multicore and GPU Programming

17 dec 2011, 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.

  Students in international master programs and exchange students will receive ECTS grades. Due to the anonymization of written exam correction, ECTS grades will be set by one-to-one translation from swedish grades (5=A, 4=B, 3=C, U=FX), according to the regulations by Linköping university.

1. (7 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? (2p)

   (b) Define and explain the following technical terms:

      i. SIMD instructions
      ii. Hardware multithreading
      iii. Cache coherence
      iv. Heterogeneous multicore system
      v. Sequential (memory) consistency

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

2. (5 p.) **Non-blocking Synchronization**

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

   (b) A bounded push buffer is a data structure consisting of an array B dimensioned for a capacity of N elements and an unsigned integer counter *top*, initially 0, which indicates the next free insert position at index *top*. B and *top* reside in shared memory.

      A number of threads push elements into the buffer concurrently. Use the atomic *compare-and-swap* (CAS) instruction to implement the operation *push*, such that a call *push*(e) atomically inserts an element e at the current top position. If a thread tries to push to a full buffer, *push* returns an error code. The application guarantees that not more than *MAXINT*-1 push operations will be performed in total, so that the counter *top* will never overflow.

      Use C or equivalent pseudocode. Make clear which of your variables are shared and which are thread-local.

      Explain your code!

      In particular, explain what CAS does and what its parameters mean.

      Explain why, with your solution, concurrent *push* operations can not lead to ”gaps” or lost entries in the buffer. (4p)

```
Algorithm FFT ( array x[0..n − 1] )
        returns array y[0..n − 1]
{
    if n = 2 then
        y[0] ← x[0] + x[1];  y[1] ← x[0] − x[1];
    else
        allocate temporary arrays u, v, r, s
            of n/2 elements each;
        for l in { 0.. n/2-1} do
            u[l] ← x[l] + x[l + n/2];
            v[l] ← ω^l ∗ (x[l] − x[l + n/2]);
        od
        r ← FFT ( u[0..n/2 − 1] );
        s ← FFT ( v[0..n/2 − 1] );
        for i in { 0.. n-1} do
            if i is even then y[i] ← r[i/2] fi
            if i is odd then y[i] ← s[(i − 1)/2] fi
        od
    fi
    return y[0..n − 1]
}
```

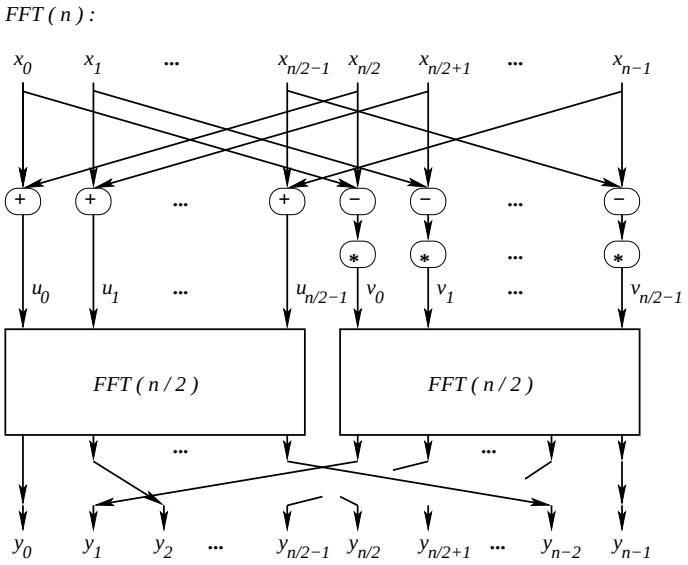

Figure 1: The sequential FFT algorithm.

3. (6.5 p.) **Design and Analysis of Parallel Algorithms**

   (a) Formulate Brent's Theorem (explained formula), give its interpretation, and describe its implications for parallel algorithm design and analysis. (2p)

   (b) The Fast-Fourier-Transform (FFT) is a (sequential) algorithm for computing the Discrete Fourier Transform of an array x of n elements (usually, complex numbers) that might represent sampled input signal values, and a special complex number ω that is a nth root of unit, i.e., $w^n = 1$. The result y is again an array of n elements, now representing amplitude coefficients in the frequency domain for the input signal x. Assume for simplicity that n is a power of 2. A single complex addition, subtraction, multiplication and copy operation each take constant time.

   Figure 1 shows the pseudocode of a recursive formulation of the FFT algorithm and gives a graphical illustration of the data flow in the algorithm.

      i. Which fundamental algorithmic design pattern is used in the FFT algorithm? (0.5p)

      ii. Identify which calculations could be executed in parallel, and sketch a parallel FFT algorithm for n processors in pseudocode (shared memory). (0.5p)

      iii. Analyze your parallel FFT 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. *(A solid derivation of the formulas is expected; just guessing the right answer gives no points.)* (2.5p)

      iv. Is your parallel FFT algorithm *work-optimal*? Justify your answer (formal argument). (1p)

      v. How would you adapt the algorithm for $p < n$ processors cost-effectively? (1p)

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

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

   The following CUDA kernel performs a rank sort (as in lab 6). However, the code is inefficient. Rewrite it for better performance. For full score, the code should be able to accept any length. Make comments to clarify what you do and why.

   You may make any assumptions you like about block size, document as necessary.

   Minor bugs, syntax errors, as well as mistakes in names of built-in symbols are generally ignored, no point deductions for things that you would easily look up in documentation (like, how many underscores you should use for that particular modifier or exact names on function calls) or that is trivial to fix on the first compilation (within reason). Your effort should be on describing your algorithm properly, including important concepts. Relevant pseudo-code qualifies for score (partial or even full depending on detail and relevance).

```
__global__ void gpu_Sort ( unsigned int *data,
                           unsigned int *outdata, int length )
{
  unsigned int pos = 0;
  unsigned int i, ix;

  ix = blockIdx.x * blockDim.x + threadIdx.x;

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

  outdata[pos] = data[ix];
}
```

5. (5 p.) **GPU Conceptual Questions**

   (a) A GPU computation calculates 2048 elements. Each element can be computed in its own thread. The algorithm is not sensitive to any particular block size. It may run on many different GPUs. What number of threads and blocks would you use in such a case? Motivate your answer. (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) In what way(s) is a texturing unit more than just another memory? (1p)

(b) List and briefly explain (short comments of a few words) the importance of two major features of the Fermi architecture that were not available in earlier GPUs. (1p)

(c) How can a non-coalesced memory access be converted to a coalesced memory access? (1p)

(d) What concept in OpenCL corresponds to a thread processor? (1p)

(e) Explain why the G80 architecture had significantly higher performance than earlier GPUs. (1p)

7. (3.5 p.) **Optimization and Parallelization**

(a) Describe the program transformation *Loop interchange* (for a sequential program). How does it affect performance on modern processor architectures? Give a general rule for when its application should be beneficial for performance, and a general rule for when it is safely applicable. (2.5p)

(b) Consider the following sequential loop:

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

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)

8. (3 p.) **High-level parallel programming**

Explain the main idea of parallel and accelerator programming using *algorithmic skeletons*. Make sure to explain what skeletons are and how they are used. Give two main advantages and two main drawbacks of skeleton programming. (3p)

Good luck!