Parallel Programming for High-Performance Applications
16 January 2014
Department of Computer Science, Faculty of Sciences

The exam has 8 questions. Your answers should be to the point: address the questions and omit information that is not asked for. The grading system is shown after the last question.

1. (a) What is Moore's law (from 1975)?
   (b) Someone claims that Moore's law currently no longer holds because of the "power wall". Comment on whether this claim is true or false.

2. (a) Explain what a hypercube topology is. What are the diameter and bisection width of a hypercube with dimension N? (Give formula's.)
   (b) What is the main disadvantage of the hypercube topology?

3. (a) How is the *efficiency* of a parallel program defined?
   (b) Compute the efficiency of a parallel bubble sort algorithm (with $O(N^2)$ execution time) relative to a sequential quicksort algorithm (which has O(N log N) execution time), assuming that the bubble sort algorithm scales perfectly well with the number of processors.

4. There are many different load balancing methods, such as:

   - block-wise partitioning of arrays (used in All-pairs Shortest Paths and Successive OverRelaxation);
   - cyclic partitioning of arrays (used in Gaussian elimination);
   - the replicated workers model (used in the Traveling Salesman Problem);
   - application-specific methods (e.g., Costzones in Barnes-Hut).

   Describe for each method for which types of parallel algorithms it is effective. In particular, describe the assumptions that the different methods make about the algorithms for which they can be used efficiently.

5. A problem with asynchronous message passing is that the buffer space for storing outgoing messages is finite, so the sender may have to wait if the buffer space fills up. This is confusing to programmers, who assume that asynchronous sends continue immediately.

   (a) Give a simple example (in pseudo code) of a program that would work correctly with unlimited buffer space but that deadlocks with limited buffer space.

   (b) How does MPI deal with this problem?

6. Consider the following HPF program fragment

```
!HPF$ PROCESSORS pr(3)
integer A(8), C(8)
!HPF$ DISTRIBUTE A(BLOCK) ONTO pr
!HPF$ DISTRIBUTE C(CYCLIC) ONTO pr

FORALL (i=1,7) A(i) = C(i+1)
```

   (a) Explain what the DISTRIBUTE directive does.

   (b) Explain how many (and which) messages will be generated for the FORALL statement.

   For clarity: the FORALL statement is equivalent to the C statement

```
for (i = 1; i < 8; i++) A[i] = C[i+1]
```

7. The parallel Barnes-Hut algorithm for hierarchical N-body problems tries to improve the *data locality* of the parallel program.

   (a) Explain why this is important and how the costzone algorithm manages to improve data locality.

   (b) Why is it a bad idea to improve data locality by simply assigning each processor an equal part of the physical space?

8. A complex number 'a' consists of a real part (a.re) and an imaginary part (a.im). The product of two complex numbers 'a' and 'b' is:

```
(a.re + a.im*i) * (b.re + b.im*i) =
   (a.re*b.re - a.im*b.im) + (a.re*b.im + a.im*b.re)*i
```

(where i is the square root of -1).

A and B are two arrays storing complex numbers: A[2*j],B[2*j] store the real parts and A[2*j+1],B[2*j+1] store the imaginary parts.

(a) Give a pseudocode example for element-wise multiplication of arrays A and B into array C using a *vectorized loop*, where the complex numbers at identical positions are multiplied. (Focus on the vectorization, the rest of the code is less important.)

(b) Give a pseudocode example for element-wise multiplication of arrays A and B using a CUDA kernel and its invocation. You may assume that the data is already loaded in the global memory of the GPU.

Some useful notes on vector instructions:

Assume the length of the vectors is 4 (the machine is 4-way SIMD), and the length of the arrays is N, a multiple of 4. The elements of vectors can be accessed with vec.w, vec.x, vec.y, vec.z

'vec1=load(a)' and 'store(a,vec1)' are instructions for loading data from array a to vector vec1 and storing data from vec1 into a.

'vec3=add(vec1,vec2)', 'vec3=sub(vec1,vec2)', and 'vec3=mul(vec1,vec2)' perform element-wise addition, subtraction and multiplication of vec1 and vec2 and store the result in vec3.

'vec3=shuffle(vec1, vec2, pattern)' stores a mixed and shuffled version of vec1 and vec2. The pattern for shuffling is also a vector, which specifies the new positions of the elements:

```
vec3.w = vec1[pattern.w] and vec3.x = vec1[pattern.x],
vec3.y = vec2[pattern.y] and vec2.z = vec2[pattern.z]
```

Points

1a	1b	2a	2b	3a	3b	4	5a	5b	6a	6b	7a	7b	8a	8b
5	5	5	5	5	5	10	5	5	5	5	5	5	10	10

Total: 90 (+ 10 = 100)