Parallel Programming for High-Performance Applications
22 October 2015
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. The parallel algorithms for both SOR (Successive overrelaxation) and ASP (All-pairs shortest paths) perform well only for large input problems; in particular, if the algorithms use NxN matrices on P processors, then N should be much bigger than P to obtain good performance.

   (a) Explain why N needs to be much larger than P by analyzing the communication behavior of the 2 algorithms

   (b) What is the impact of the communication network on the performance of these algorithms? Which network properties are important for SOR resp. ASP?

   (c) Why does the parallel TSP (Traveling Salesman Problem) algorithm not need a large input problem (number of cities) to obtain good performance?

2. (a) Describe two similarities and one key difference between NUMA multiprocessors and Distributed Shared Memory.

   (b) Describe what the relation is between the diameter of a communication network and the latency of messages that are sent over the network. How did this relation change when networks changed from packet-switched message routing to circuit-switched message routing?

3. (a) What is a Remote Procedure Call in the SR language? Give an example where it is easier to use than normal message passing.

   (b) The Ibis programming system uses *connections* (with send-ports and receive-ports) for communication. What are the advantages of this approach?

   (c) What are the most important advantages of HPF (High Performance Fortran) over message-passing systems?

4. An MPI program needs to send a large message (hundreds of megabytes) from one machine to another, where the exact size depends on runtime conditions. What SEND and RECEIVE primitives from MPI would you use to make the transfer efficient and avoid unnecessary copying? Give some code fragment for the sending and receiving processes. The syntax does not matter, but do make clear which exact SEND and RECEIVE primitives (e.g., communication modes) you use.

5. Someone wants to run the IDA* search algorithm with transposition tables on a wide-area system, consisting of six (identical) clusters at different locations connected by high-bandwidth (10 Gb/s) wide-area links.

   (a) How much would the performance of the original Work Stealing algorithm with partitioned transposition tables be affected, compared to running the algorithm on a single cluster? Explain your answer.

   (b) How much would the Transposition-Driven Scheduling (TDS) algorithm be affected? Explain your answer.

6. Please give short answers to the following questions:

   (a) What is thread divergence for a GPU and what is its impact on performance? Give an example (in pseudocode or C) of code that exhibits divergence.

   (b) What is global synchronization (for a GPU)? How can it be achieved?

7. Assume the following code, which transposes a matrix:

```
for (i=0; i<N; i++)
   for (j=0; j<N; j++)
       B[j][i] = A[i][j];
```

   The A and B matrices contain integer numbers, with 1 int = 4 Bytes.

   Answer the following questions:

   (a) What is the arithmetic intensity of the kernel? Can the arithmetic intensity of the kernel be improved?

(b) This kernel runs on an architecture with 1 TFLOP theoretical peak, and 200GB/s memory bandwidth. What is the expected performance (execution time) for N=10000?

8. Take the following application:

```
__kernel Kernel1(int N, int *array1, int *array2) {
   int myID = ...; //myThreadID
   if (myID < N)
      array2[myID] = process(array1[myID]);
}

Init;
Kernel1 (N, array1, array2); //requires 2GB memory, 1GB per array
WriteResults;
```

Assume *process* is an arithmetic expression that is evaluated only using its own input data (i.e., the application is embarrassingly parallel). The execution time of the kernel on the CPU is $Tc(N)$ and on any GPU is $Tg(N)$ for the full array; the data transfer time is $Td(N)$. All execution and transfer times are proportional with the size of the array (i.e., $Tg(2N) = 2 * Tg(N)$). Assume that $Tc(N) >> (Tg(N) + Td(N))$.

(a) Assume a system with a GPU with 1GB of memory, while the arrays are 1GB each. How would you use this system for accelerating this application? What would be the performance gain?

(b) Assume the kernel needs to process arrays that are 100 times bigger. Give examples of three types of architectures that you would use. What would be the performance expectation there?

Points

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

Total: 90 (+ 10 = 100)