Exam

DVA336 Parallel Systems

8:10–12:30

January 8, 2018

This exam includes 8 problems, for a total of 80 points.

General information

1. This exam is closed book.
2. A simple non-programmable calculator is permitted.
3. Write your answers in English.
4. Make sure your handwriting is readable.
5. Answer each problem on a separate sheet.
6. Write your identification code on all sheets as required for anonymous exams.
7. All answers must be motivated. If necessary, make suitable assumptions.

Grading

The passing marks are 90% for grade 5, 70% for grade 4, and 50% for grade 3. This corresponds to the passing grades A, C, and E in the ECTS grading system.

Contact information

For questions on problems 1, 3-7, please contact Gabriele Capannini, 021-101458.
For questions on problems 2 and 8, please contact Björn Lisper, 021-151709.

Good luck!

1. (a) [3 p] Describe Flynn's taxonomy of computer architectures.
   (b) [4 p] In what ways does a NUMA architecture differ from a SMP one?

2. (a) [4 p] What is the Divide and Conquer algorithm paradigm? Explain briefly how it works, and why such algorithms can be good parallel algorithms!
   (b) [4 p] Data parallel primitives are often classified into six groups. Name four of them, and explain briefly how the primitives in each group work!

3. Consider the following program:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
  int n = 1+rand()%1000000;
  float * A = (float *) malloc(sizeof(float)*n);
  for(int i=0; i<n; ++i) A[i] = rand()/(RAND_MAX+1.0f);
  float sum = 0;
  for(int i=0; i<n; ++i)
    sum += A[i];
  sum /= n;
  printf("n=%d avg(A[]) = %f\n", n, sum);
  free(A);
  return 0;
}
```

   (a) [4 p] Rewrite the above program by means of the AVX(or SSE) instruction set (syntax is not important but make sure to point out all the details of the new implementation of the functions).
   (b) [2 p] Discuss and motivate the performance that can be expected for SIMD version with respect to the corresponding sequential implementation.

4. Consider the following program:

```
#define N 1000000
void main() {
  float A[N];
  for(int i=0; i<N; i++)
    A[i] = 2.0f;
  for(int i=0; i<N; i++)
    A[i] = A[i] * A[i];
}
```

   Assume that each iteration of the first loop takes $50\,\mu$s, while each iteration of the second loop requires $150\,\mu$s (the time spent in the remaining part of the code is negligible). Then the second loop is parallelized in the following way:

```
#pragma omp parallel for
for(int i=0; i<N; i++)
  A[i] = A[i] * A[i];
```

(a) [4 p] Give the definition of the Amdahl's Law and explain each part of it.

(b) [5 p] Assuming that the upgrade leads to the ideal speedup. According to the Amdahl's Law, what is the theoretical maximum speedup achievable by the parallel version of the program?

(c) [3 p] Assume that the second loop is successively vectorized by means of the AVX instructions. How does it change the evaluation provided at the previous point (b)?

5. Consider the following C program:

```
#include <stdio.h>
#include <omp.h>
#define P 4
#define N 16
int main() {
  int counter[P] {}; //init all 0s
  int A[N];
  for(int i=0; i<N; ++i) A[i] = i%P;
  #pragma omp parallel num_threads(P)
  {
    int t = omp_get_thread_num();
    #pragma omp for
    for(int i=0; i<N; ++i)
      if(A[i]==t) ++counter[t];
  }
  int sum = 0;
  for(int i=0; i<P; ++i) {
    sum += counter[i];
    printf("counter[%d] = %d\n", i, counter[i]);
  }
  printf("sum = %d\n", sum);
  return 0;
}
```

(a) [3 p] Write the output of the program above.

(b) [3 p] Now remove the line `#pragma omp for` and write the output of the new version of the program.

(c) [5 p] What is the *false sharing* phenomenon? How does it affect this computation?

(d) [3 p] Explain how to modify the program in order to eliminate the false sharing.

6. (a) [6 p] Implement a MPI collective function which calculates the sum of a set floating point values for the processes of a given communicator and communicates the result to all the processes[1] (see the example below). The function to implement has the following signature: `void MPI_Sum(float * x, MPI_Comm comm)`. It isn't allowed to use any other MPI collective functions.

Example:

[1] Better solutions (in terms of communication complexity) yield more credits.

```
int main(int argc, char *argv[]) {
  MPI_Init(&argc, &argv);
  float x = 10.0f;
  MPI_Sum(&x, MPI_COMM_WORLD);
  printf("sum=%.0f\n", x);
  MPI_Finalize();
  return 0;
}
```

The output for the program above ran on 3 processes is:

```
sum=30
sum=30
sum=30
```

(b) [3 p] Calculate the communication complexity of your `MPI_Sum` implementation.

(c) [2 p] What is a point-to-point function?

7. (a) [6 p] Provide an example of CUDA kernel which makes use of the Shared Memory (SMEM) and accesses the data by producing some bank conflicts (assume that the warp size and number of shared memory banks are equal to 32). Describe the behavior of your code, how the shared memory works, and state the entity of the bank conflicts generated.

   (b) [4 p] Describe the memory hierarchy of the device memory: provide information about location and the performance the main memory levels.

8. (a) [12 p] In statistics, the *standard deviation* σ of a finite sequence of random numbers $x_1, \ldots, x_n$ is defined as

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}, \quad \text{where } \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Define a data parallel algorithm for computing the standard deviation of a sequence of random numbers stored in a data parallel array! Use the notation for data parallel algorithms that we have used in the lectures on data parallelism and parallel algorithms (not OpenMP or such).

What is the parallel time complexity of your algorithm? To get full credits, your solution must have a parallel time complexity that is significantly lower than the sequential complexity for the straightforward sequential solution.