CHALMERS TEKNISKA HÖGSKOLA
Institutionen för data- och informationsteknik
Avdelningen för datorteknik

Exam in DAT400 (Chalmers) and DIT430 (GU) High Performance Parallel Programming, Wednesday, October 28th, 2020, 14:00h - 18:00h

Teacher: Mustafa Abduljabbar, tel 7721083, email: `musabdu@chalmers.se`

Language & Submission: Answers shall be given in English. Answers should be digitally prepared. Submission should be done via the Canvas assignment.

Solutions: Solutions will be posted on Tuesday, Nov 4th, on the Canvas page.

Exam review: The review date will be posted on the course Canvas page by the time you receive the email from LADOK.

Aids: All aids are allowed during examination. Plagiarism checks will be conducted!

Grades:

Chalmers				
Points	0-23	24-35	36-47	48-60
Grade	Failed	3	4	5

GU				
Points	0-23	24-41	42-60	
Grade	Failed	G	VG	

Good Luck!

Problem 1 (8p)

OpenMP and MPI are two different libraries used to parallelize an application on a given system.

(a) Motivate a scenario where one must consider using MPI (or similar libraries), either with or without OpenMP, to parallelize their application.

(b) Programming in MPI is known to be prone to deadlocks. While in OpenMP, there is a high risk of race conditions if not carefully considered. In no more than 2 lines, what is the reason of the risk associated with each programming model?

(c) A code snippet is provided below. Suggest an OpenMP and an MPI implementation of this code. You can consider mix of the two as well. Which version do you think is better and why?

```
for(int i=0; i<N; i++)
  for(int j=0; j<M; j++)
    for(int k=0; k<M; k++)
      B[i][j][k] = A[i-1][j][k] * B[i][j][k-1];
```

Problem 2 (10p)

A research computing company is assigned a task, that is to run a large application on their hardware infrastructure. The application includes four programs. Each program depends on the result returned from the one before it. Hence, Program i cannot start before Program $i-1$ has returned the result ($1 < i \leq 4$). The team analyzed and tested the **serial** execution time of each program on an x86 CPU. The results came as follows:

- Program 1 - 10 minutes (can be parallelized)
- Program 2 - 10 minutes (can be parallelized)
- Program 3 - 10 minutes (cannot be parallelized)
- Program 4 - 10 minutes (cannot be parallelized)

(a) Assume the company works 8 hours per day and gets \$10 each time they complete running the whole application. How much do they earn per day?

(b) They get some advice from a GPU sales person that running entirely on a GPU will help a lot, and they could obtain the following speedups for each program:

- Program 1 - Speedup = 100
- Program 2 - Speedup = 10
- Program 3 - Speedup = 1
- Program 4 - Speedup = 0.2

By averaging the speedups = (100+10+1+0.2)/4 = 27.8, they think it is profitable to do so, and decide to buy the GPU product for this application. What do you think?

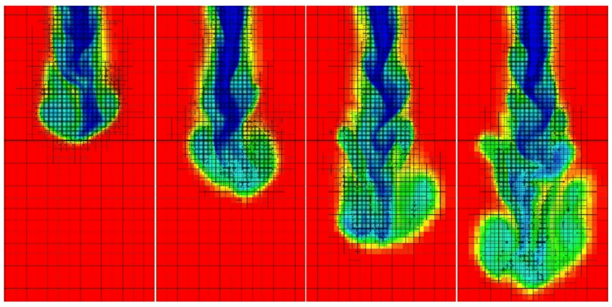
(c) They also think they could run program 4 on the CPU and run the other three on the GPU. For simplicity, ignore the time for communication time sending data from CPU to GPU and vice versa. How about this combined method, comparing with running on CPU totally?

(d) Think about Amdahl's law. What is the upper limit speedup for this application? Give explanations about the speedups you get from parts (b) and (c).

Problem 3 (8p)

Consider that there are 4 regions of computations as visualized by the figure below. You would like to use MPI to parallelize these computations on a small cluster that contains 4 nodes, where each node has 8 processors.

(a) Each region contains computations that require broadcasting the data among processes within each node. How will you divide the work between communicators? Write a pseudo code to demonstrate that.



Domain decomposition for AMR

(b) Assume you would like to block until all processes in the communicator have reached a certain point in the code. How would you achieve such a behavior within a given node and also across all nodes?

```c
float a = rank;
float b;
if(rank != 0)
{
  MPI_Recv(&b, 1, MPI_FLOAT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD, &status)
  printf("Value of b = %f from rank %d", b, rank);
}
else
  MPI_Send(&a, 1, MPI_FLOAT, 2, 1, MPI_COMM_WORLD)
```

(c) You encounter the code above. What could be an issue with this code? Correct the issue using only MPI_Send and MPI_Recv calls.

(d) Can MPI_Send and MPI_Recv calls be replaced by any collective in the given scenario? What would be such a collective?

Problem 4 (8p)

The code below approximates a mathematical constant.

```
const int steps = 100000;
double step = 1.0/(double) steps;
double ax;
int i;
double x;
double sum = 0.0;
#ifdef PARALLEL
#pragma omp parallel private(...) shared(...)
#endif
{
  for (i=0; i<steps; ++i)
  {
    x=step*(i+0.5);
    x*=x;
    ax=4.0/(1.0+x);
    sum = sum + ax;  // accumulate
  }
}
double result=step*sum;
```

(a) You turn on parallelism (i.e. you `#define PARALLEL` preprocessor flag) and you discover that even though multiple threads are created and the program runs to completion, the `result` does not match that of the serial version. **Assume that the missing** `private,shared` **fields are already set correctly**. In no more than two lines, explain why the parallel code is broken the way it is.

(b) For the 6 variables defined before the parallel region. Write the appropriate scoping levels from the list `private,shared` for a parallel version to work correctly.

(c) Add the necessary OpenMP clause(s) for the code to work correctly. For each added clause, briefly motivate the use of such clause.

(d) If you decide to use any synchronization/coordination in (c), is it possible to avoid such usage? Show the code that avoids it, and mention 1 advantage and 1 disadvantage of each approach (i.e. an approach that uses synchronization/coordinate versus an approach that does not).

Problem 5 (8p)

You would like to run the 3D N-Body code below on System X. For that, you would have to understand the system's performance and the expected behavior and limitations of your code on such system. So you decide to use DRAM Roofline model to simplify the task. System X hosts a CPU that is clocked at 2.5 GHz and is capable of 1 single precision operation per cycle (i.e. 1 FLOP/cycle). It has a DRAM bandwidth of 5GB/s.

```
for (i=0; i<N; i++) {
  float pi = 0;
  float axi = 0;
  float ayi = 0;
  float azi = 0;
  float xi = x[i];
  float yi = y[i];
  float zi = z[i];
  for (j=0; j<N; j++) {
        float dx = x[j] - xi;
        float dy = y[j] - yi;
        float dz = z[j] - zi;
        float R2 = dx * dx + dy * dy + dz * dz + EPS2;
        float invR = rsqrtf(R2);
        float mj   = m[j];
        float invR3 = mj * invR * invR * invR;
        pi += mj * invR;
        axi += dx * invR3;
        ayi += dy * invR3;
        azi += dz * invR3;
  }
  p[i] = pi;
  ax[i] = axi;
  ay[i] = ayi;
  az[i] = azi;
}
```

The CPU has 2 generations: **Gen.1)** single core CPU. **Gen.2)** 4-core CPU.

(a) What is the arithmetic intensity of the code above? Note that `rsqrtf` is equivalent to 6 floating point operations on the underlying instruction set. Show your steps. Remember that Arithmetic Intensity (AI) is the ratio of total floating-point operations (FLOPs) performed by a given code or code section, to the total data movement (Bytes) required to support those FLOPs.

(b) For the 2 generations, specify if the code is bounded by compute or memory. Show your steps.

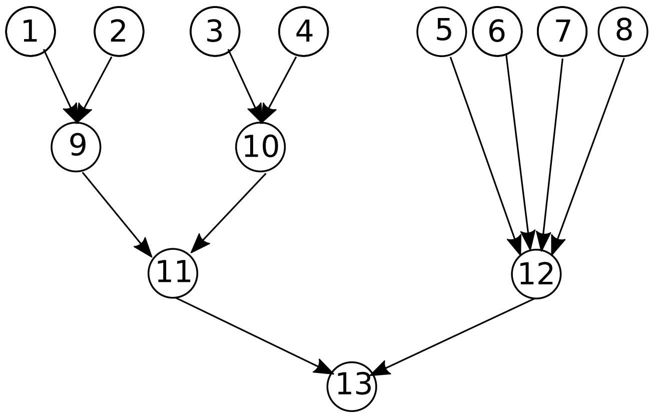
(c) On **Gen.2**, you run the code for some large N, and you discover that for the specific arithmetic intensity that you calculated in (a), the value of the FLOP/s falls below the roof. State 1-2 reasons why this could be the case.

(d) There are several algorithms to implement the N-Body problem (e.g. Barnes-Hut, Fast Multipole, the code above, etc.). Your task is to experimentally decide on the best algorithm on System X for some N. You choose the FLOP/s as your metric, such that the algorithm that scores the highest FLOP/s is the most optimal. Is this a good or a bad choice? Why or why not? If not, what is the alternative?

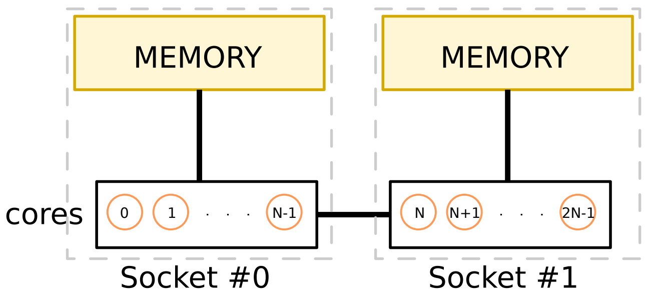
Problem 6 (8p)

You have written a program that includes an asymmetric reduction pattern shown below as a task DAG, where each task is a single-threaded computation executing on one core and taking the same time (1 time-unit) and arrows indicate precedence constraints. Consider the following assumptions:

- The DAG is executed on a machine with a number of cores that are equally distributed among two chips, each with its local memory (i.e. dual-socket), as shown on the right-side figure.
- Assume initially that communication costs between tasks do not add to the execution time.
- The best sequential algorithm has an execution time of 8 time-units.



Task DAG



System Architecture

Tasks:

(a) For systems with 2, 4 and 8 cores, come up with a schedule and mapping that minimizes execution time. For each machine and your proposed optimal schedule, compute the Cost and the Efficiency.

(b) Repeat the problem considering now that each task execution incurs a scheduling overhead that is proportional to the number of incoming edges (num_edges) such that $T_{overhead} = \text{num\_edges}$ time-units?

(c) What is the largest number of cores that can accelerate the computation? Why? Explain in 1-3 sentences.

(d) Assume now that communication costs add one time-unit for each edge that crosses across the two sockets. Repeat task (a) with this new assumption.

Problem 7 (10p)

Assume a vector x is sized of n floats and n = 67,108,864. The programmer wants to compute the sum of all vector elements on a GPU platform. In order to do the experiments, the programmer writes two versions of CUDA kernel code and launches them with different configurations. However, they are either incorrect and/or inefficient. Your task is to identify the issue(s) in each kernel version.

(a) In the following scenarios, you are expected to list the errors arising from the indicated usages. Errors can be related to hitting a limitation when launching the kernel, and/or incorrect implementation of the highlighted vector sum kernel (program correctness).

[i] Launching one block with 67,108,864 threads executing the kernel, i.e., VectorSum1 ⋘1,67108864⋙ (x,67108864):

```
__global__ void VectorSum1(float *x, int n) {
    int tid = blockDim.x * blockIdx.x + threadIdx.x;
    if(tid < n) {
        for(int stride = 2; stride < n; stride *= 2){
            __syncthreads();
            if((tid % stride) == 0)
                x[tid] = x[tid] + x[tid + stride];
        }
    }
    // The result is in x[0]
}
```

[ii] Launching the same code as [i] but with VectorSum1⋘65536,1024⋙ (x, 67108864).

[iii] Launching only one block with 1024 threads executing the kernel where each thread computes the sum of n/1024 elements before the threads perform a reduce, i.e., VectorSum2 ⋘1,1024⋙ (x, 67108864, 65536).

```
__global__ void VectorSum2(float *x, int n, int m) {
    int tid = blockDim.x * blockIdx.x + threadIdx.x;
    if(tid < n) {
        float sum = 0.0;
        // Compute the sum for my part of the array
        for(int i = 0; i < m; i++)
            sum += x[tid + i];
        // Reduce
        for(int stride = 2; stride < n; stride *= 2){
            __syncthreads();
            if((tid % stride) == 0)
                x[tid] = x[tid] + x[tid + stride];
        }
    }
    // The result is in x[0]
}
```

(b) Firstly, correct your listed errors in VectorSum2 kernel. There are many ways to improve the performance of the slow kernel, such as coalescing global memory references, using shared memory, and so on.

[i] State one optimization you would like to exploit in this kernel (i.e. VectorSum2) and explain why you chose it?

[ii] Write the correct version of VectorSum2 that contains the optimization that you chose in (b.i).