Introduction to Supercomputing

TMA4280 · Problem set 3

Exercise 1. Go to `http://top500.org`, a website cataloguing the top 500 supercomputers in the world. Study the top 10, in particular their technical specifications. What is meant by the LINPACK benchmark performance?

Exercise 2. Note that some of this was not covered in detail in the lecture. Please have a look at the lecture notes.

1. What limits the scalability of a bus-based interconnect?
2. How are the individual processors connected using a crossbar?
3. How are the individual processors connected using a mesh?
4. What is the difference between a shared-memory and a distributed memory architecture?
5. What characterizes the memory access in an SMP?
6. What is the difference between a NUMA and a ccNUMA architecture?

Exercise 3. *MPI Questions*

Describe shortly in your own words why Message Passing is required for Distributed Memory Multiprocessors and how it works?

What are the different types of MPI

How many bytes are sent in each of the three messages listed below? Here given in C as example.

```
MPI_Send(buf1, 80, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
MPI_Send(buf2, 1024, MPI_INT, dest, tag, MPI_COMM_WORLD);
MPI_Send(buf3, 1024, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
```

What is a tag and how is it encoded? Is it true that a unique tag must be specified each time `MPI_Recv` is called?

Exercise 4. *A simple MPI program step by step* Add the following functionalities:

- *Communicator size*. Print the number of processes "`Running on %u MPI processes`" only on rank 0.

- *Output*. Print "`%2u/%2u: Hello World!`" on each process, with `%2u/%2u` printing *rank / number of processes*.
- *Barrier*. Print "`%2u/%2u: How are you?`" on each process, with `%2u/%2u` printing *rank / number of processes* only after all the "`%2u/%2u: Hello World!`" lines are printed.
- *Linear pattern*. Given N processes, for each process $i = 0, \ldots, N-2$ send an integer to its right neighbour. In short, the program takes data from process zero and sends it to all of the other processes such that process i should receive the data and send it to process i+1, until the last process is reached.
- *Ring pattern* Modify the program to send the data in a ring such that all processes are involved in transferring to their right neighbour modulo N.
- *Highest bidder*. Each process generates a random integer: the winner is the rank with the highest value, and in case of a draw the lowest rank wins. Only this process is allowed to print `I am the winner!` to the standard output.
- *Best player with five cards*. Each process generates five random integers. The winner is the rank with the better hand: for each pair of processes integers are compared one by one for the highest, one point is won per integers and in case of a draw the lowest rank wins. Only this process is allowed to print `I am the winner!` to the standard output.
- Implement

Exercise 5. *Parallelization of a program* Implement the following program from Set 2 Exercise 5 in C/C++ using MPI which performs the following three different operations:

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{a} + \gamma \boldsymbol{b} \\ \boldsymbol{y} &= \boldsymbol{a} + \boldsymbol{A}\boldsymbol{b} \\ \alpha &= \boldsymbol{x}^{\mathsf{T}} \boldsymbol{y} \end{aligned}$$

You can use any compatible vectors and matrices you see fit (e.g. random ones). The constant γ should be read from the command line. It should run on any number of processes. *Hint:* Partition the matrix in column strips.