Problem set 1 - MPI Intro

Part 1 - Theory

Problem 1 - General Theory

a)

Flynn's Taxonomy:

SISD: Single Instruction Single Datastream describes a system what can only do one instruction at the time, and all data is received from the same stream. A machine with one CPU and one main memory will fitt here, like the classic von Neumann.

SIMD: Single Instruction Multiple Datastream describes a system which can do the same single instruction on multiple ALUs, with each ALU working on different data. A GPU is a good example of a SIMD implementation.

MISD: Multiple Instruction Single Datastream describes a system where multiple ALUs/functional units perform different operations on the same data. This is not a widely used architecture.

MIMD: Multiple Instruction Multuple Datastream describes a system where all the different processors/ALUs can function independently. Each processor can work on different instructions and different datastreams.

MPI fits into Flynn's Taxonomy within SIMD and MIMD. MPI is used for sending messages between the different instances of the instruction/program. It is not however needed in MISD and SISD.

b)
MPI selects the fastest available communication channel between two processes and and if two instances of the same program runs on the same node, shared memory is typically used to achieve the highest performance.

c)
MPI can also use distributed memory and will try to find the fastest connection available to achieve this. For example, when using Vilje, each node is connected with Infiniband to provide higher performance and coupling between the nodes. If technology like Infiniband is not available it will fall back to for example TCP/IP network.

Problem 2 - Code Theory

a)

i)
When running on the ITS machine, the only communication needed is between the actual processor and therefor the bottleneck is minimal. If the exercise is runned on Vilje, or another distributed system,

the bottleneck between the different nodes will be higher than between processors.
ii)
Higher connectivity between the servers would reduce the bottleneck. If there is a implementation way to do this, i am not aware of it.

b)

i)
First i thought that there might be a computational bottleneck if the computed local ranges would be for example be splitt equally from bottom to top. E. g. If a range from 2 to 1000 would be splitted on 4 processes. Then a split with the 2-252, 253-500, 501-750, 751-1000 would be bad. This is because the first range would be finished way before the rest.

ii)
As i discovered the bottleneck i tried to describe in the proviuse task, i implemented the split so each process takes the next integer in a queue like manner. E. g.

Process one: 2, 6, 10 ....
Process two: 3, 7, 11 ....
Process three: 4, 8, 12 ....
Process four: 5, 9, 13 ....

I think this aproach would be a little more efficient.

Part 2 - Code

Problem 1 - MPI Intro

a) See the computeMPI.c file.
It is not optimally implemented as one process is per now used only for management. But i ran out of time.

b) See the Makefile.

c)

i) O(n/p)

ii) O(2(p-1))

iii) O(2(p-1)/p)

iv) O(2p-1)

v)
I did not understand what i was suppose to graph, but i provided the run-times.

```
1 100 Time: 0.000090 The sum is: 29.774290
```

```
1 1000000 Time: 0.020934 The sum is: 78627.269730
1 1000000000 Time: 22.607368 The sum is: 50849234.689638
2 100 Time: 0.000073 The sum is: 29.774290
2 1000000 Time: 0.021501 The sum is: 78627.269730
2 1000000000 Time: 10.676788 The sum is: 50849234.689636
4 100 Time: 0.000429 The sum is: 29.774290
4 1000000 Time: 0.009580 The sum is: 78627.269730
4 1000000000 Time: 8.183556 The sum is: 50849234.689635
8 100 Time: 0.000305 The sum is: 29.774290
8 1000000 Time: 0.013695 The sum is: 78627.269730
8 1000000000 Time: 4.941481 The sum is: 50849234.689636
```