1 Part 1, Theory

Problem 1, General Theory

a)

	Single instruction	Multiple instruction
Single data	SISD	MISD
Multiple data	SIMD	MIMD

Table 1: Flynn's taxonomy

Flynn's taxonomy is used to distinguishe between parallel hardware architectures. It distinguishes between the number of instruction and data streams.

SISD Single instruction, Single data is a uniprocessor computer. This may be compared to the traditional Von Neumann computer. This will not allow any parallelism in neither the instruction stream or data stream.

SIMD Single instruction, Multiple data is for instance an vector processor or GPU. It executes a single instruction on multiple data elements. This allows parallelism. It access' a distrubated memory.

MISD Multiple instruction, Single data is said to have "no commercialy known implementation". This is hetreogenous computers that operate on the same data stream, and together agree upon a result.

MIMD Multiple instruction, Multiple data is computers where different instructions operate on differenct data. It is often split into two seperate classes; tightly- and loosely coupled MIMD. Tightly means multi-cores and loosely means clusters. They also kan be either shared- or distributed memory systems.

MPI fits into SIMD, because MPI is a standard for communication among processes (in a parallel program) running on a distributed memory system. In this problem set, we write a singe program, multiple data, or SPMD. We do not write a different program for each process.

b)

It is possible to use MPI on shared-memory systems, even though it is not designed for it. For shared-memory systems it would be preferable to use POSIX-threads or OpenMP. In a shared-memory system, all the multiple processes work on the same data. To avoid concurrency issues it protects the data using for instance semaphores. Since MPI by design let processes work using a message exchange system, processes cannot modify each other's data. This may well be implemented on a shared-memory system

c)

MPI is as stated before, designed to run on systems with distributed memory. All processes run simultaneously, passing data as messages. Every process has its own private memory, and by using message exchange, it elimates concurrency issues.

Problem 2, Code Theory

a)

b)

Part 2, Code

Problem 1, MPI Intro

c)

1. By program: $O(n) = n$ and per process: $O(n) = \frac{n}{P_i} + n \bmod P_i$
2. $O(P) = 2(P-1)$
3. $O(P) = \frac{2(P-1)}{P}$
4. $O(P) = P - 1$

d)

I were on a trip, and were not available to complete all the graphs. Below, I have added the graph for two runs;

1. P = 1, n = 10^9
2. P = 8, n = 10^9

I were able to run all the computations, and the results are attached in the `result.txt` file

Figure 1: P = 1, n = 10^9

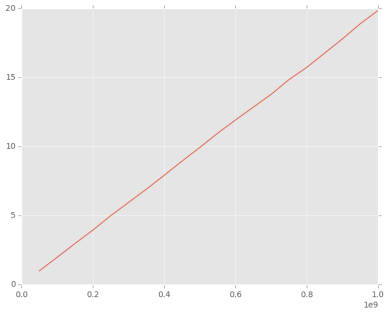

Figure 2: P = 8, n = 10^9