INTRODUCTION TO SUPERCOMPUTING

TMA4280 · Problem set 4

Exercise 1. Answer the following questions.

1. On which multi-processor systems is it of interest to use message passing?

2. What are the advantages of using a standardized communication library (or message passing library) such as MPI?

3. A communication library consists of many specific message passing operations. How would you classify these operations into a few main groups, or types of communication patterns?

4. Explain what is wrong with the following code segment. It is written in C but the language is not important.

```
MPI_Comm_rank(comm, &rank);
if (rank == 0) {
  MPI_Recv(recvbuf, count, MPI_DOUBLE, 1,
           tag, comm, &status);
  MPI_Send(sendbuf, count, MPI_DOUBLE, 1, tag, comm);
}
else if (rank == 1) {
  MPI_Recv(recvbuf, count, MPI_DOUBLE, 0,
           tag, comm, &status);
  MPI_Send(sendbuf, count, MPI_DOUBLE, 0, tag, comm);
}
```

Exercise 2. Assume a distributed memory multiprocessor computer with the following interconnect charateristic: the time $\tau(k)$ it takes to send a message with k bytes can be approximated as

$$\tau(k) = \tau_s + \beta k,$$

where τ_s is a fixed startup time and β is the inverse bandwidth (units of seconds per byte).

For our setup, $\tau_s = 1\,\mu\text{s}$ and $\beta = 1.25\,\text{ns}\,\text{B}^{-1}$.

1. How many bytes can we send in a single message before the time to send the message is twice the startup time? To how many (double precision) floating-point numbers does this correspond?
2. How long does it take to send a message with a single floating point number? Is it preferable to send many short messages instead of a single, long one?

Exercise 3. Let $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ be matrices of size $n \times n$, and let the matrix $\boldsymbol{D}$ be constructed as

$$\boldsymbol{D} = 3\boldsymbol{A}\boldsymbol{B} + \boldsymbol{C}.$$

How many floating point operations does it take to construct $\boldsymbol{D}$?

Exercise 4. Implement an MPI-based matrix multiplication program. It should accept one command-line argument, which is the size of the matrix and vector to multiply. Use arbitrary data (e.g. random).

You are free to choose the structure of the matrix and vector, but a typical approach is the following:

- Each process "owns" a certain set of indices.
- Each process only stores the part of the vector that it owns, and the part of the matrix corresponding to the *columns* that it owns.
- The result of the local matrix-vector product is then a full-sized vector, with contributions to the vector chunks of all other processes.

Things to consider:

1. What changes will you have to make if you want to store the matrix on each process by rows instead of columns?
2. For some types of matrices it is possible to minimize communication by cleverly choosing the index sets. What characterizes these matrices?
3. What would a matrix transpose operation look like?