Introduction to Supercomputing

TMA4280 · Problem set 6

Note that

- this problem set is mandatory;
- you can work on it in groups with *up to* 3 members;
- you should write a report describing your solution (max 12 pages);
- you should write your *names* on the report (no student numbers);
- the source code should be handed in together with the report (and not as part of it);
- please make sure that you have answered all the questions;
- the due date is Friday, April 15, 2015;
- the report will count 30% towards the final grade.

Problem description

Consider the solution of the two-dimensional Poisson problem,

$$\begin{aligned} -\nabla^2 u &= f && \text{in } \Omega = (0,1) \times (0,1), \\ u &= 0 && \text{on } \partial\Omega. \end{aligned}$$

Here, f is a given right hand side and u is the solution.

Assume that we discretize this problem on a regular finite difference grid with $(n+1)$ points in each spatial direction. Hence, the grid size is $h = 1/n$. We use the standard 5-point stencil to discretize the Laplace opkerator.

In order to solve the system of algebraic equations, we apply the diagonalization methods discussed in class. In particular, we apply the Discrete Sine Transform (DST) in order to obtain a solution of this problem in $O(n^2 \log n)$ floating point operations.

Exercise 1. Write a program to solve the Poisson problem with P processes using the algorithms described above. You can choose $f \equiv 1$ for the initial development.

Use the provided routines to compute the DST and its inverse based on the FFT. See Appendix A for details on compiling, linking and running the serial version of the Poisson solver. Use the MPI communication library to develop your program for a distributed memory parallel computer, and OpenMP to use t threads on each MPI process. See Appendix C for comments on the transpose operation.

Exercise 2. Run your parallel program on *Vilje*. Obtain detailed timing results for different combinations of $n = 2^k$ and P, t. In particular, demonstrate that your program functions correctly for selected values of P in the range $1 \leq P \leq 36$. Follow the procedure described in Appendix B.

Note: Having a program that runs quickly is useless unless the answer is correct.

Note: It is sufficient and strongly recommended that you test the correctness of your program on a small problem size before solving larger problems.

Exercise 3. Run your program with $n = 16384$ and $pt = 36$, i.e. with about 270 million grid points on 36 threads/processes. Does the hybrid model work better, worse or equivalent to the pure distributed memory model? Explain your observations.

Exercise 4. Report the speedup S_p as well as the parallel efficiency η_p for different values of n and p. Th parallel efficiency is defined as $\eta_p = S_p/p$.

How do your timing result scale with the problem size n^2 for a fixed number of processors? Is it as expected? Do you see an improved speedup if you increase the problem size?

Note: To obtain consistent and reliable timing results, submit individual runs as different jobs.

Exercise 5. Modify the given f to be a function of your own choice. As an example, you could choose f to be a smooth function like

$$f(x, y) = \mathrm{e}^x \sin(2\pi x) \sin(2\pi y).$$

Another example is to let f represent point sources, e.g. $f \equiv 0$ in the whole domain except at two chosen gridpoints where $f = -1$ and $f = 1$ respectively.

Run your program with the new right hand side f for a particular n and p. Do you have to modify anything related to the parallel implementation when you change f, i.e. when solving a different Poisson problem?

Exercise 6. Discuss how you would modify the numerical algorithm to deal with the case where $u \neq 0$ on $\partial\Omega$, i.e. for non-homogeneous Dirichlet boundary conditions. You do not have to implement this.

Exercise 7. In the exercise and lectures we have assumed that the domain is the unit square, i.e. $\Omega = (0, 1) \times (0, 1)$. Discuss how you would modify the Poisson solver based on diagonalization techniques if the domain instead is a rectangle with sides L_x and L_y. You can still assume a regular finite difference

grid with $n + 1$ points in each spatial direction. Does this extension of the original method change anything in terms of your parallel implementation? You do not have to implement this.

General comments

It will be emphasized that the program is well parallelized and load balanced. It will also be important that the program is well organized and easy to understand. It is allowed to use more memory than the minimum required if it speeds up the program, but do so with care.

A report describing the results of parallelization of a scientific problem typically contains

- a description of the problem;
- a discussion of possible solution strategies;
- a brief explanation of the finished program;
- a description of the computer on which the numerical results were obtained, which compiler (with version) and compiler options you used and other relevant information, such as libraries used and their versions;
- numerical results (preferably plots)
- analysis (theoretical and experimental) of the performance of the algorithm and its implementation (time usage, speedup and efficiency as functions of problem size and number of processes or threads);
- a discussion of bottlenecks and possible improvements; and
- possibly a listing of relevant parts of the source code.

A Compiling and linking

The serial code ships with a CMake build system. You can generate a build system using

```
module load intelcomp
module load cmake
CC=mpicc FC=mpif90 cmake . -DCMAKE_BUILD_TYPE=Release
```

assuming you are located in the folder with the `CMakeLists.txt`. On success this will generate a makefile, and you can then build and run the program using

```
make
./poisson 128
```

The first statement builds the program, while the second runs it on a single processor with $n = 128$. Note that n must be a power of two.

By default CMake does not show you the compiler commands. You can see them by doing

```
make VERBOSE=1
```

It is recommended to perform out-of-tree builds. For example, using a `build` folder:

```
mkdir build
cd build
cmake ..
```

The CMake setup has options for compiling with MPI and OpenMP. They are on by default, but can be disabled at the configuration stage:

```
cmake . -DENABLE_OPENMP=0 -DENABLE_MPI=0
```

B Verification of correctness

One way to verify that the code works correctly is to do a *convergence test*. Following this approach, we first assume an exact solution to our Poisson problem. For example, we can assume that the exact solution is given as

$$u(x, y) = \sin(\pi x) \sin(2\pi y).$$

This solution satisfies the boundary conditions.

Next, evaluate $-\nabla^2 u$, which should be equal to f, i.e.

$$f(x, y) = -\nabla^2 u = 5\pi^2 \sin(\pi x) \sin(2\pi y).$$

Assuming the given data f we now solve the Poisson problem numerically, and compare the computed solution with the exact solution at the grid points. The maximum pointwise error should decrease to zero as $O(h^2)$the given data f we now solve the Poisson problem numerically, and compare the computed solution with the exact solution at the grid points. The maximum pointwise error should decrease to zero as $O(h^2)$. Remember that finding the maximum pointwise error will require communication.

C Comments on the transpose operation

The implementation of the transpose operation is trivial in a serial context. In a parallel context, using a distributed memory programming model, it is quite tricky. In this case the transpose of a matrix will involve all-to-all communication.

Part of the solution of this exercise requires a parallel implementation of the transpose operation. We are given a matrix of a certain dimension. We can distribute the matrix so that each process is responsible for a certain number of columns (or rows).

The most convenient way to implement the transpose operation is to use the MPI function `MPI_Alltoallv`. For a detailed description of this function, please use Google.

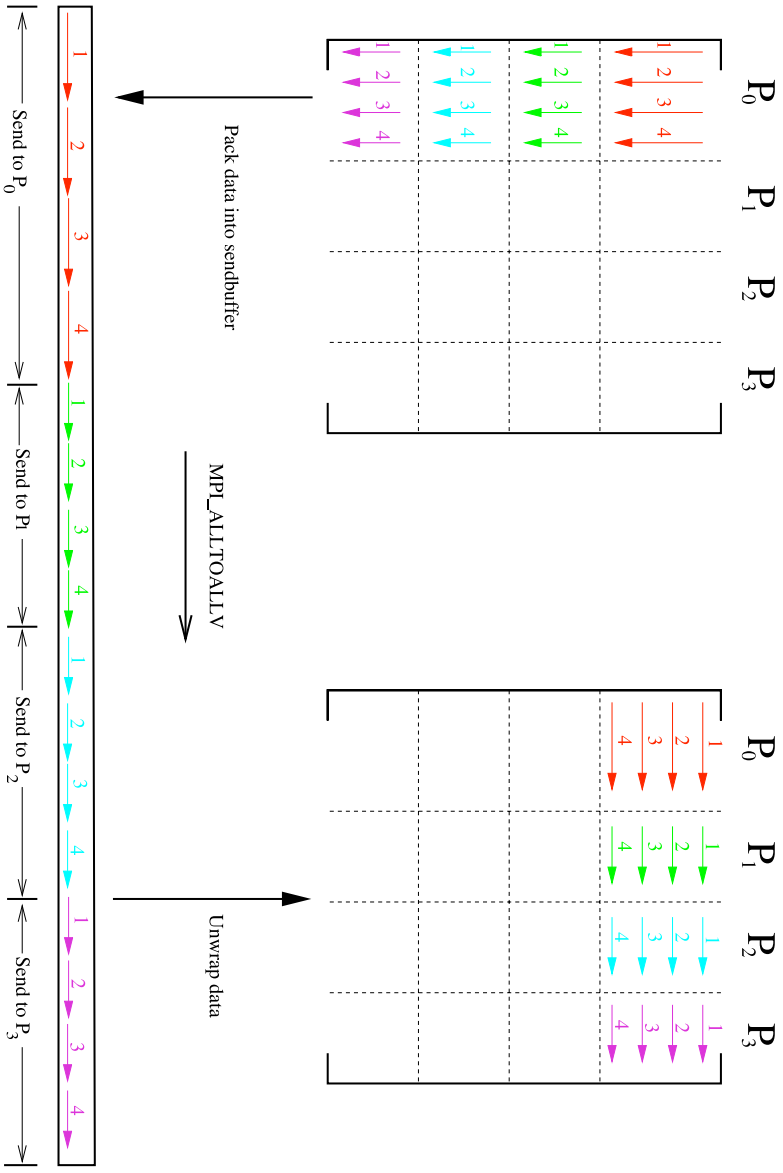

Figure 0.1: The transpose operation using message passing: the packing and unpacking of data. The figure is due to Bjarte Hægland.