Programming of parallel computers
Computer Lab no. 3: Derived data types. Linear algebra operations

DCAMM, DTU and IT, Uppsala University

January, 2008

Some of the suggested exercises utilize a number of available test codes, which can be downloaded from
`http://www2.imm.dtu.dk/courses/FortranMPI/MPI/Labs/Lab3`.

Exercise 1 [Derived data types]
Suppose that we want to communicate one column of a 2D array and that we store data row-wise in a 1D data structure, see Figure 1. In order to use `MPI_Send` and `MPI_Recv`, we have to specify their arguments, namely, type, length and starting address of the data to be communicated. This latter requires data in one message to be stored contiguously in memory which causes a problem in the above described data allocation.
We can then either send the elements one by one, which is very inefficient, or copy the column to a temporary array, send the temporary array, receive data in another temporary array and finally unpack the temporary array to the corresponding column (as done in the test implementation in Exercise 4).
MPI provides a more convenient way by the use of derived datatypes.
Remark: Due to the fact that the way how arrays are stored in memory is language-dependent, sending a column in C and sending a row in Fortran is the operation which requires special care.

Tasks

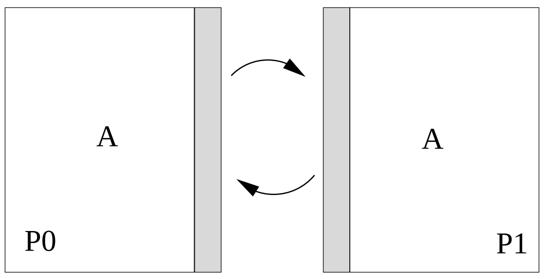

Figure 1: Communicate a column from P0 to P1, assume data stored row-wise in a 1D array

1. Compile, run and study the program `datatypes`. Make sure that you understand the usage of `MPI_Type_vector`.

2. Modify the program to communicate the same portion of information if we have a 2D array instead or to communicate two (or more) rows.

   You may check the code in `array_sect` and `type.f`. (The latter should be tested on 2, 3 and 4 PE.)

Exercise 2 Create your own routine to perform matrix$\times$vector operation. Choose the matrix and the corresponding vector partitioning yourself and allocate the matrix blocks and the sub-vectors locally. Initialize all matrix and vector entries to be equal to 1 so that checking the correctness of your algorithm will be easier.

Exercise 3 [Solving a hyperbolic equation] This example considers the numerical solution of a hyperbolic partial differential equation in two-dimensions, namely the following advection equation:

$$\begin{aligned} u_t + u_x + u_y &= f(x,y) & 0 \le x,y \le 1 \\ u(0,y,t) &= h(y-2t) + u_0(0,y), & 0 \le y \le 1 \\ u(x,0,t) &= h(x-2t) + u_0(x,0), & 0 \le x \le 1 \\ u(x,y,0) &= h(x+y) + u_0(x,y), & 0 \le x,y \le 1 \end{aligned}$$

The functions $f(x,y)$, $u_0(x,y)$ and $h(\tau)$ are chosen in such a way that the problem possesses an analytical solution $u(x,y,t)$ as follows:

$$\begin{aligned} u(x,y,t) &= h(x+y-2t) + u_0(x,y) \\ f(x,y) &= 2e^{x+y} + 3x^2 + 6y^2 + sin((x+y)/2) + cos((x+y)/2) \\ u_0(x,y) &= e^{x+y} + x^3 + 2y^3 + sin((x+y)/2) - cos((x+y)/2) \\ h(\tau) &= sin(2\pi\,\tau) \end{aligned}$$

Assume now that the spatial domain is covered by a rectangular grid with step-sized Δx and Δy in the x- and y-direction, correspondingly. We pose the problem to compute numerically the values of the solution in the discrete points (x_i, y_j, t_k) where $x_i = i\Delta x$, $y_j = j\Delta y$ and $t_k = k\Delta t$. We denote $u(x_i, y_j, t_k)$ as u_{ij}^k.
The usual way to solve such a time-dependent problem is as follows: we compute the solution at time t_k and then, using the so-obtained approximate solution, proceed to compute it on the next time level t_{k+1}.
In this particular implementation the problem is discretized using the so-called *Leap-frog* scheme which reads as follows

$$u_{ij}^{k+1} = u_{ij}^{k-1} + 2\Delta t\left(f_{ij} - D_x^0 u_{ij}^k - D_y^0 u_{ij}^k\right) \tag{1}$$

where $f_{ij} = f(x_i, y_j)$, $D_x^0 u_{ij}^k = (u_{i+1,j}^k - u_{i-1,j}^k)/(2\Delta x)$ and $D_y^0 u_{ij}^k = (u_{i,j+1}^k - u_{i,j-1}^k)/(2\Delta y)$. We see, that $D_x^0 u_{ij}^k$ and $D_y^0 u_{ij}^k$ are the standard central difference approximations to the corresponding first derivatives in x and y.

The scheme 1 entails and additional requirement to compute extra numerical boundary conditions on the lines $x = 1$ and $y = 1$. For simplicity, in the program implementation, we choose to use the analytic solution.
The above problem is implemented in C++ and in F90. Choose your preferable language and download the corresponding files from
`http://www2.imm.dtu.dk/courses/FortranMPI/MPI/Labs/Lab3/wave_CC/` or
`http://www2.imm.dtu.dk/courses/FortranMPI/MPI/Labs/Lab3/wave_F90/`.
The implementation in C++ includes the following routines:

wave.cc	main program
wave.h	header file, function declarations
bound.cc	boundary conditions
diffop.cc	differential operators
initcomm.cc	communication setup
residual.cc	error norm
force.cc	forcing functions f, u_p, h
Array2_dbl.cc	2D arrays
Array2_dbl.h	header file 2D arrays
wave.dat	data file, problem specification
Makefile	makefile
batch.q	batch script

The code is parallelized by splitting the spatial domain into squares and each processor is responsible for the calculations on a block $x_1^p \leq x \leq x_2^p$, $y_1^p \leq y \leq y_2^p$.
The processors are organized in a 2D Cartesian topology to match the data decomposition. The node-points are ordered row-wise. Communication is required when computing the D_x^0 operator, D_y^0 operator, and in the residual computation. The communication is local neighbor-to-neighbor and follows the same pattern in each timestep. In the y-direction the data is not contiguous and therefore derived datatypes are used. To lower some of the communication initialization overheads *persistent communication* is used. Moreover, the communication is overlapped with the computations to minimize the synchronization overheads.
The implementation in F90 includes the following routines:

adveq.f90, params.dat, bound.f90, force.f90, Makefile

The input data (wave.dat and params.dat) is set so that the number grid-points in each spatial direction is 255.

Tasks:

1. Observe

   (a) the initialization of the logical topology (`wave.cc, adveq.f90`)

   (b) derived data types (`initcomm.cc, adveq.f90`)

   (c) using persistent communications and overlapping communication and computation (`adveq.f90, subroutine init_comm, MPI_Waitall`, `initcomm.cc`)

2. Compile and run the program using different numbers of processors, note the timings for each run.

   Plot the so-called *speedup*, $S_p = T_1/T_p$, where T_1 is the time to run the program on one processor and T_p - the time to run the program on p processors.

   What do you observe? Is the scaling linear, can you explain the results, is it what you expected?

3. To get exclusive runs without interference from other users, one has to use the batch system. Specify your number of processors in the file `batch.q` and submit the script with the command `qsub batch.q`. To check the status of the system you can use `qstat`. When your job has finished you will get an output file `wave.out` with the results.

Exercise 4 Consider the code in sub-directory `cg_F77`.
The implemented problem there is to solve a discrete Laplace equation using the standard (unpreconditioned) Conjugate Gradient (CG) method.
The model continuous problem is

$$-u_{xx} - u_{yy} = f(x, y), \quad (x, y) \in \Omega = [0, 1]^2,$$

with Dirichlet boundary conditions imposed on the whole boundary ofΩ. The spatial domain (the unit square) is discretized using rectangular grid, not necessarily equidistant.
The discretization is done using standard central difference approximation,

$$\frac{1}{h_x^2}(-u_{i+1,j} + 2u_{i,j} - 2u_{i-1,j}) + \frac{1}{h_y^2}(-u_{i,j+1} + 2u_{i,j} - 2u_{i,j-1}) = f_{ij}$$

Parallelism is achieved using spatial domain decomposition. It is to be noted that the so-arising matrix, which has a five-diagonal structure with diagonal entries equal to 4 and off-diagonal entries equal to -1, is kept in a nonstandard way. In each subdomain, instead of one sparse matrix with 5-diagonal structure and size $N \times N$ with $N = nx * ny$, five dense arrays are allocated, named `m, e, w, n, s` from *middle, east, west, north, south*, each of size $nx \times ny$.

1. Apart from some scalar products, communications are required when performing a matrix-time-vector multiplication (`grid_cmnmat.f`). The implementation is 'stupid' since the data is first gartered in a buffer, then exchanged and finally unpacked.

   One can modify this routine by creating proper data types and then time the two versions to see if and how much the overall time has improved.

2. The program is interactive and permits to have different types of partitionings, see Figure 2 and different sizes of the locally allocated sub-meshes. It asks the user to input px and py, to determine the processor geometry, and to input nx and ny to determine the size of the local sub-grid, allocated per PE. By varying px, py and nx, ny we can do some performance studies.

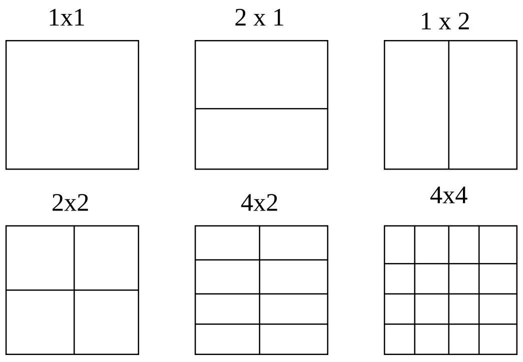

Figure 2: Processor grids $px \times py$

For example,

$$px = py = 1, \quad nx = ny = 200$$
$$px = py = 2, \quad nx = ny = 100$$

will solve a problem of the same size on one and four processors. Our expectation is that the time on four processors will ideally be four times smaller.

The choice

$$px = py = 1, \quad nx = ny = 100$$
$$px = py = 2, \quad nx = ny = 100$$

will solve four times larger system on four processors, keeping the load per PE constant. Ideally, we would expect the time in both cases to be the same. However, this will be true only if we use an algorithm, which scales linearly with the number of degrees of freedom, such as Multigrid (MG) or Algebraic Multigrid (AMG). With the present method, the number of iterations will roughly double when the number of degrees of freedom is increased four times. Thus, the quantity to watch is the time spent per iteration. We expect this to be the same if the implementation is done in a good way.

(a) Check the above two effects by running a few experiments.

(b) Do we have to use `MPI_BARRIER` after each communication in the code `grid_cmnmat`?

Please write your observations and comments regarding the tasks from Exercises 3 and 4 on a piece of paper and hand it in to Maya Neytcheva before you leave the computer lab session. Thank you.