ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

M. Troyer, P. Koumoutsakos
ETH Zürich, HIT G 31.8
CH-8093 Zürich

Set 4 - Diffusion equation with MPI

Issued: March 16, 2015
Hand in: March 23, 2015

If we want to spread the heat beyond a single node we need message passing. In this exercise we will parallelize the 2D diffusion equation with MPI.

a) Parallelize the serial 2D diffusion code found in `skeleton/diffusion2d_serial.cpp` with MPI. Divide the 2D domain with an MPI cartesian grid and use appropriate MPI datatypes for the data transfer.

A straightforward way is to split the grid into square tiles with one tile per process. For simplicity we adjust the local grid size in each direction to a multiple of the number of corresponding processes. To minimize communication overhead we compute the interior of the local domain while waiting for the boundaries by using non-blocking asynchronous communication.

Parallel code: `solution_code/diffusion2d_mpi.cpp`

b) Estimate (on paper) the communication overhead of the 2D grid decomposition compared to the horizontal strips that we used in the previous semester. Which one achieve better scaling?

The total amount of communication per process if the domain is divided into tiles is $2[N_x/P_x + N_y/P_y]$ and for strips along the y-axis $2N_x$, where $N_{x,y}$ and $P_{x,y}$ is the number of grid points and processes in each direction. As we are doing sparse matrix-vector multiplication the computation scales as N_xN_y/P_xP_y. Hence finite difference discretization should scale much better if we use tiles rather than strips.

c) Make a strong scaling plot up to 48 cores.

Looking at the strong scaling fig. 1, for a total grid size $N_x = N_y = 2^{10}$ the sub-matrix fits entirely into the cache minimizing memory access and resulting in perfect scaling. As we increase the system size we exceed the cache size and additional memory access reduces performance. Scaling gets better again for larger systems as the ratio between communication and computation decreases. The drop in scaling as the number of processes exceeds 24 is due to inter-node communication which is much more expensive than message passing within a node when using less than 24 processes.

In Figure 2 and 3 we show the same scaling when a different MPI schedule is used, i.e. changing the mapping "rank to node". In particular we tried out `-bynode` in Figure 2 and `-loadbalance` in Figure 3. The Open MPI documentation `man mpirun` has a very explanation of how ranks are distributed to the nodes in each case.

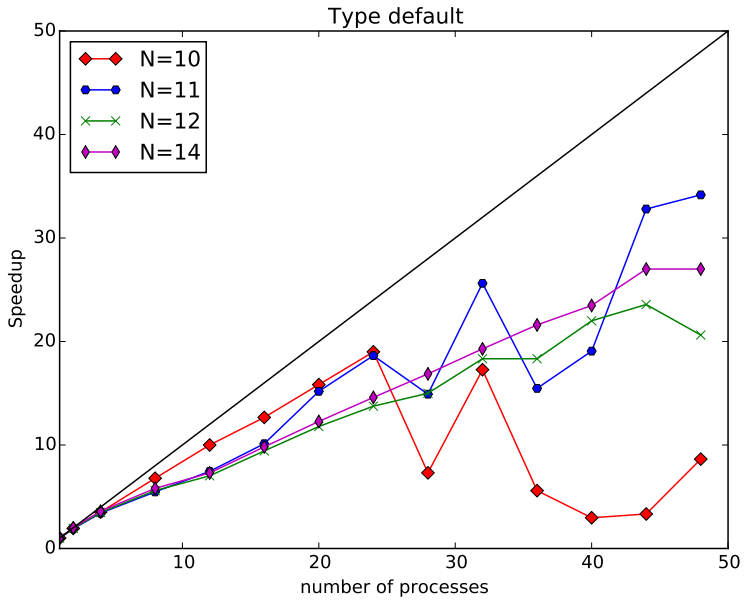

Figure 1: Strong scaling for 20 time-steps. $N_{x,y}$ is the global grid dimension. Ranks are distributed according to the Open MPI 'default' schedule.

The scaling plots for small system sizes present some strage behavior between 24 and 32 processes. We understand this behavior by checking the domain decompositions: for $P = 24$ we have a grid 6×4, for $P = 28$ we have a grid 7×4, for $P = 32$ we have a grid 8×4, for $P = 36$ we have a grid 6×6. Because of this change in the shapes, the amount of communication which takes place between nodes varies substantially.

Summary

Summarize your answers, results and plots into a short PDF document. Furthermore, elucidate the main structure of the code and report possible code details that are relevant in terms of accuracy or performance. Send the PDF document and source code to your assigned teaching assistant.

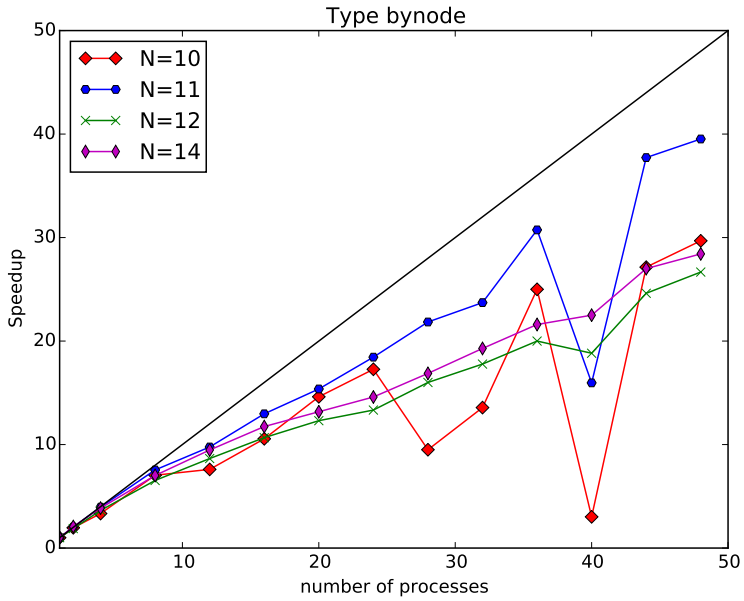

Figure 2: Strong scaling for 20 time-steps. $N_{x,y}$ is the global grid dimension. Ranks are distributed according to the Open MPI 'bynode' schedule.

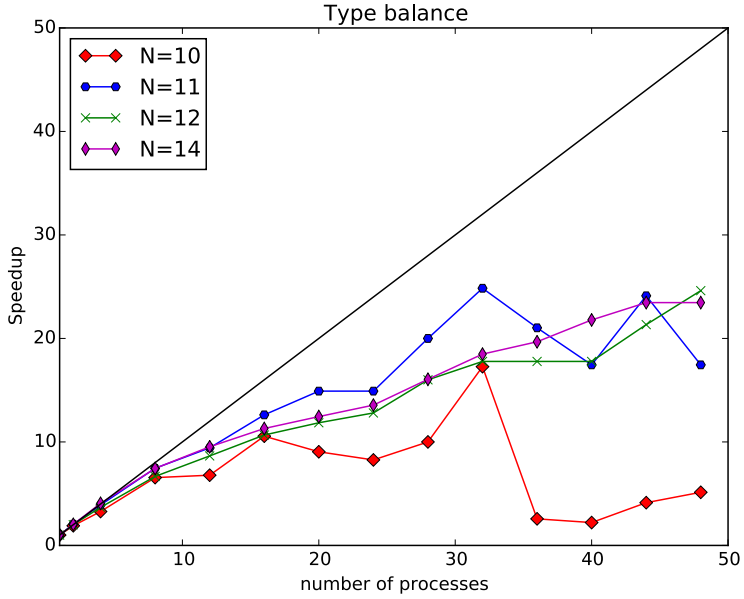

Figure 3: Strong scaling for 20 time-steps. $N_{x,y}$ is the global grid dimension. Ranks are distributed according to the Open MPI 'loadbalance' schedule.