Advanced Parallel Programming

Labs 3

MPI: Topologies and Neighborhood Collectives

Michel Herrera Sanchez

Parallelization of stencil.c using Topologies and Neighborhood Collectives

The parallelization of code stencil.c using MPI mainly consists on split the whole grid into smaller tiles and assign each of them to an MPI process for energy calculation. The energy is calculated on each point taking into account the value of their neighbours and updated on each iteration. Initially process with rank 0 get the parameters grid\_size, energy and number of iterations. The input parameters are packed in an array of 3 elements and broadcasted to all process using MPI\_Bcast MPI function. Each process call the topology functions to create a cartesian topology (MPI\_Cart\_create) and extract their coordinates in the global grid and find their neighbours rank (MPI\_Cart\_shift) equivalents to north, south, west an east. For each process is then calculated the size of the tile (bx, by) that belong to and check if inside the assigned tile where are the potential energy sources that can be mapped from the global\_grid. Two different MPI\_Datatype are created : north\_south\_type that represent the horizontal halo located on top of the tile and east\_west\_type the vertical halo bounding the left and right extremes of the tile. This is required due to calculate the energy on each process tile it’s important to know information about the neighbors. Then two buffers are created aold, anew. Those buffers have the size of (bx + 2)\*(by + 2) due to the required space to be filled by halo information. For each iteration the cells with energy values are updated and then there is a hallo exchange with all neighbours process. Initially top and bottom most cells are sent to the north and south neighbors and left and write most cells are sent to the west and east neighbors using the cartesian topology communicator created previously. The halo sent from each process to each neighbors is performed using non-blocking MPI calls MPI\_Isend. After the cells are sent each process update their neighbour halo information receiving the data from their neighbors using MPI\_Irecv. It’s important to highlight that the use of cartesian topology allows to easily work out the corner cases, hence a process with a missing neighbour is treated as an MPI\_PROC\_NULL. Any information from or sent to a MPI\_PROC\_NULL rank is discarded. The halo exchange happens with a different set of buffers than aold and anew. After the halo exchange have finished the halo information is updated in the aold buffer. The process proceeds to calculate the heat on each of the point internal to his assigned tile. The energy is updated and the ouput belong the input for the next iteration till all the iteration complete. When the iteration finished the process with rank 0 perform a global reduce to sum the partial heat distributed across all the process tile using MPI\_Reduce and store the value in process with rank 0.

The parallelization of stencil.c using neighbourhood collectives differs for the fact that there is not MPI\_Datatype definition for horizontal and vertical halos and the communication for halo exchange is performed using the function MPI\_Ineighbor\_alltoallv. New buffers are required to be created before the halo exchange with neighbourhood collectives and copy back the results to the process tile halos. It actually reduces a lot of coding complexity due to the four halo exchange only happens in call to the mpi function MPI\_Ineighbor\_alltoallv.

For the current experiment 16 mpi process were involved in the benchmark. The running parameters parameters for the test:

Grid Size : 16

Energy to be Injected Per Iteration: 1

Number of Iterations: 200

As shown in the following table mpi neighbourhood collectives version of stencil parallelization is slower than the pure MPI implementation.

|  |  |
| --- | --- |
| Implementation | Execution Time (s) |
| MPI | 0.006467 |
| MPI Neighborhood Collectives | 0.009019 |

See code stencil\_mpi.c and setencil\_mpi\_ncollect.c for implementation details.