MPI Explorations

Lin Feng
Computer Science, University of Bristol
Candidate No: 97302

Abstract **- Message passing interface (MPI) is widely used in large scale parallel applications in science and engineering. Apart from implementing the concept of data streams, it makes usage of the popular halo exchange algorithm. In this report, both row and tile decomposition models are addressed and explained for Halo exchange. Furthermore, several MPI functions are introduced for implementing each model, while also pointing out how computing performance is improved by using MPI.**

I. Introduction

MPI - Message Passing Interface is a specification of message passing libraries. It provides several functions which developers can use to specify multiple computers or multiple processor cores within the same computer in order to identify and implement a single parallel program across distributed memory.

A communicator defines a group of MPI processes. When an MPI application starts, the group is initially given a predefined name called MPI_COMM_WORLD (simple programs have no need to worry about names). Additionally, every process of this group is assigned a unique rank.

Each process starts its own part of the program and explicitly communicates with other processes by exchanging data. For instance, one process sends a copy of the data to another process, and the target process receives it. Communications such as this which involve one sender and receiver are known as point-to-point communications.

Basic send/receive functions are blocking communication, there is also non-blocking communication in MPI. Whereas MPI_Send() and MPI_Recv() are blocking functions. The process of sending data will be stopped until data has been delivered and the buffer is emptied. The process of receiving data will also be blocked until a matching message is received from the system and the buffer is filled. Blocking communication is fairly simple to use but can be prone to deadlocks. On the other hand, MPI_ISEND() and MPI_IRECV() are the non-blocking alternatives. These functions return immediately even if the communication is not finished yet. Non-blocking communication leads to improved performance but needs to use MPI_Wait() or MPI_Test() to see whether the communication has finished.

One commonly used communication pattern for domain decomposition problems is halo exchange. Each process computes its own data in core region. At every timestep, the data is exchanged with neighbour processes so that every process has access to the correct information. The processes then use their newly acquired information to update the halo regions for the next computing. For a big calculation of a grid of cells, decomposing original grid into multiple smaller grids should be considered. Each part is owned by a different process so that each process only stores and calculates fewer data. The gird can be decomposed by columns or rows, or as tiles.

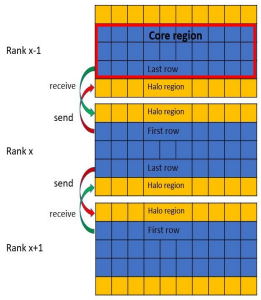

Figure_1> Border exchange decomposed by rows

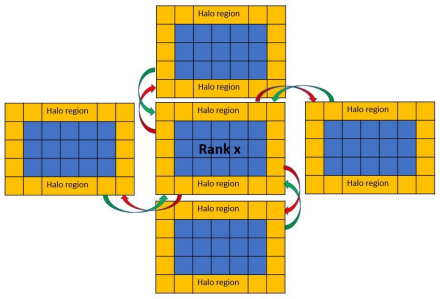

Figure_2> 2-dimensional border exchange decomposed as tiles

II. Halo Exchange by rows

As a strategy of halo exchange, I tried to divide rows of the stencil to achieve better performance. I did this instead of starting with dividing columns since I know that row-major order leads to improved performance in computing from my previous assignment.

Decomposing cells of stencil by rows into multiple chunks as shown in Figure_1, each chunk belongs to a different rank. Allocate additional rows at above and bottom of each chunk for computational work, these cells are not updated locally, but provide values when updating the borders of this chunk. To update those cells, every process sends the information of cells in the first row and the last row to the pair of neighbors rank x-1 and x+1, and places the received borders in the halo region from its neighbors.

In each rank, the cells in the first and last row can be calculated as a 5-point with extended information in the halo region, but the calculation of cells in the first and last column is still 4-point.

A. MPI_Sendrecv

There are a lot of sending and receiving functions in MPI for exchanging message. I choose to use MPI_Sendrecv, so that the send-receive operations combine in one call the sending of a message to one destination and the receiving of another message from another process. In other words, Instead of using MPI_Irecv, MPI_Send and MPI_Wait, MPI_Sendrecv is more convenient.

III. Halo Exchange as tiles

I tried to decompose cells of the stencil as tiles so that each cell can be calculated as a 5-point stencil with extended information in the first and last columns.

Decompose cells into multiple chunks as tiles as shown in Figure_2, each chunk belongs to a different rank. Allocate additional space for a series of cells around the edges of each chunk. As two-dimensional border exchange, processes perform horizontal border exchanges with their up and down neighbours and perform vertical border exchange with their left and right neighbours. Processes receive the messages from their neighbours and update the edge cells in the halo region.

A. MPI_DIMS_CREATE

Unlike row decomposition, the neighbours of processes are more complex to located in tiles decomposition. I choose to use MPI_DIMS_CREATE to partition all the processes into a 2-dimensional topology, so that it is easier to locate all neighbour ranks in a mesh.

In MPI, the function MPI_DIMS_CREATE helps to select a balanced distribution of processes per coordinate direction, depending

on the number of processes in the group to be balanced. And the dimensions will be set to be as close to each other as possible by using an appropriate divisibility algorithm which is friendly to the network communication. Setting dims[i] = 0 allows routine to modify the number of nodes in dimension i.

B. MPI_CART_CREATE

I used MPI_CART_CREATE with setting reorder = false(0) and periods[i] = false(0), it is paired with MPI_DIMS_CREATE for handling the cartesian topology information in the communicator.

There are some optional constraints that can be specified by the users. I set reorder = false(0), so that the number of each rank in the new group is identical to it in the old group; if reorder is setted to true, the rank of each process will be reordered onto the physical machine, which might cause the incorrect calculation for ordered computing. Additionally, I set periods[i] = false(0) specifying the grid is not periodic in each dimension. In other words, coordinate 0 in dimension n is not a neighbour of coordinate n_max as shown in Figure_3, so that the border cells of boundary rank will not be updated.

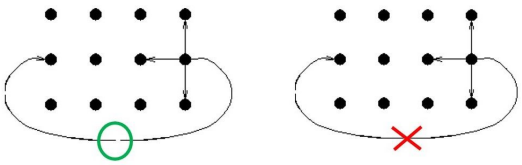

Figure_3> periods[i] = true on ritht, periods[i] = false on left

C. MPI_Neighbor_alltoall

Considering that working with 16 cores, rather than using MPI_Send/MPI_IRecv to optimise each message individually, I used MPI_Neighbor_alltoall for message exchange, so that the code is much simpler and easier to read.

MPI_Neighbor_alltoall is a collective operation in which all processes send and receive the same amount of data to each neighbour. Processes package all edges in one sending buffer, MPI will send the specified size of messages to neighbours which have been determined in cartesian topology, then processes receive messages from neighbours, unpackage the messages and update the cells in halo region.

IV. Performance Analysis

I compared the runtime of row decomposition and tile decomposition by running code on 2~16 cores as shown in Figure_4. The results shows that tile decomposition performs much better than row decomposition. The runtime does not change much with more than 4 cores with row decomposition. However, the runtime is getting shorter on 16 cores with tile decomposition.

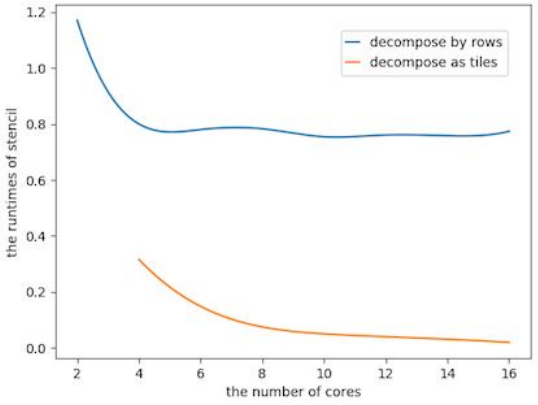

Figure_4> runtime with 1024*1024*100

I tried to analyse the performance of tile decomposition since it does not increase linearly as expected. First of all, I get that the operational intensity of stencil is 0.375 with using single precision, referring to roofline model of Intel Xeon CPU E5-2670 (Sandy Bridge) on 16 cores, I know the rate limiting is memory bandwidth bound when OI is 0.375, so performance is all about moving bytes. The peak DRAM bandwidth is about 66 GB/s on 16

cores, while I couldn't measure the memory bandwidth used by the given program. However, to increase the usage of memory bandwidth, except single instruction multiple Data (SIMD), a little-known feature non-temporal instructions can be considered.

V. Conclusion

There is room for improvement, the best result I could get from tile decomposition of halo exchange as shown in Figure_5.

nx	ny	niters	Runtime (on 16 cores)
1024	1024	100	0.02s
4096	4096	100	0.53s
8000	8000	100	1.97s

* Reference

Wikipedia. 2018. Message Passing Interface. [ONLINE] Available at: https://en.wikipedia.org/wiki/Message_Passing_Interface.

Wikipedia. 2018. SPMD. [ONLINE] Available at: https://en.wikipedia.org/wiki/SPMD.
Definition from WhatIs.com. 2018. message passing interface (MPI). [ONLINE] Available at: https://searchenterprisedesktop.techtarget.com/definition/message-passing-interface-MPI.

Partnership for Advanced Computing in Europe, 2013, “An Interface for Halo Exchange Pattern.” Bianco, Mauro. [ONLINE] Available at http://www.prace-project.eu/IMG/pdf/wp86.pdf.

Argonne MPI Tutorials. 2013. Introduction to MPI. [ONLINE] Available at: http://www.mcs.anl.gov/~balaji/permalinks/2014-06-06-argonne-mpi-basic.pptx

Open MPI Softwarem Documentation. 2018. MPI_Sendrecv(3) man page (version 4.0.0). [ONLINE] Available at: https://www.open-mpi.org/doc/v2.0/man3/MPI_Sendrecv.3.php

Cartesian Convenience Function. 2000. 6.5.2. Cartesian Convenience Function: MPI_DIMS_CREATE. [ONLINE] Available at: https://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-1.1/node134.htm

Open MPI Software. 2018. MPI_Cart_create(3) man page (version 1.6.5). [ONLINE] Available at: https://www.open-mpi.org/doc/v3.1/man3/MPI_Cart_create.3.php