Assignment One

Ye Cai 9503101967
Yeca16@student.bth.se

Part1 MPI-based Blocked Matrix-Matrix Multiplication

Implementation

Basically this two-dimensional Block-wise parallel matrix multiplication divides the C matrix' s (the result matrix 1024*1024) columns into 2 parts and rows into nodes/2 parts like Figure 1, Figure 2 is for four nodes, Figure3 is for eight nodes:

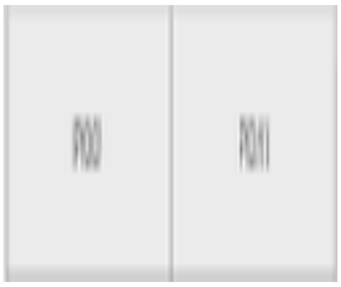
Figure 1

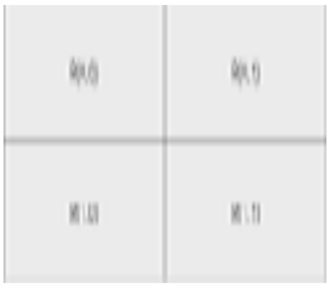
Figure2

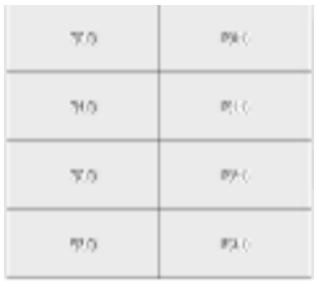
Figure3

And it use the Master/Slaver model refer to the row-wise version provided, which means the master node send the (SIZE/(node/2)*SIZE)rows of A matrix and half of B(SIZE*SIZE/2) matrix to the else slaver nodes instead of sending whole B matrix to slaver nodes., then they will do calculation at same time. After that the slavers will send the rows (SIZE/(node/2)*SIZE/2)of C matrix to the master node. Then the master node could choose to print the result matrix(C matrix).

Measurement

This is a table about compare of running the row-wise parallel program and row-col-wise parallel program with 1024*1024 matrixes.

	Sequential	Row-wise 1D		Block-wise 2D	
Node	74.31s	Execution time	Speedup	Execution time	Speedup
1		71.74s	-	73.79s	-
2		40.11s	1.85	43.36s	1.71
4		27.56s	2.69	25.97s	2.86
8		20.96s	3.55	18.60s	3.99

Part2 MPI-based LaPlace Approximation

Implementation

This parallel version program of SOR LaPlace approximation equally divided the matrix by row-wise, each node is responsible for a part include SIZE/node rows data of matrix. And each node will store two extra boundary rows data for calculation which is the end row of data belongs to last node and the first row of data of next node. Figure 6 is an example on a 16*16 matrix with 4 nodes.

First, the program initials a same matrix ‘A[SIZE+2][SIZE+2]’ in each node then according each node’s rank to locate which part to calculate. Then calculate data in a fixed time of iteration. (Because of some reasons, it’ hard to put calculation into a while loop then run it until match some condition. The sequential one provided also has been changed and put into the make-file for compare.) At each iteration, at first it calculates the even(red)half data, exchange the boundary data with last and next node, after exchange data it calculates the odd(black) half data, then exchange boundary data again. So the nodes can do the job parallel and only relay on the boundary data. After iterations, all node except node 0 will send the result back to node 0. And then node 0 will calculate print the execution time.

Row 0
Row 1
Row 2
Row 3
Row 4
Row 5

Node 0

Row 4
Row 5
Row 6
Row 7
Row 8
Row 9

Node1

Row 8
Row 9
Row 10
Row 11
Row 12
Row 13

Node2

Row 12
Row 13
Row 14
Row 15
Row 16
Row 17

Node3

Figure 6: The row with grey word as belong to neighbor node

Measurement

This is a compare of execution time and speedup between sequential and parallel SOR LapLace approximation program on a 2048*2048 size matrix within 100 times of iteration.

	Sequential one	Parallel one	
Node	Execution time	Execution time	Speedup
	63.08s	–	–
2	–	54.32s	1.16
4	–	35.81s	1.76
8	–	26.12s	2.41

As above, the speedup of 2 ,4,8 nodes is calculated by using the sequential execution time 7.91 to divide the parallel program on 2,4,8 nodes so the result is 63.08s/54.32s=1.16, 63.08s/35.81s=1.76, 63.08s/26.12s=2.41.