YANAN ZHANG

ASSIGNMENT #4 REPORT

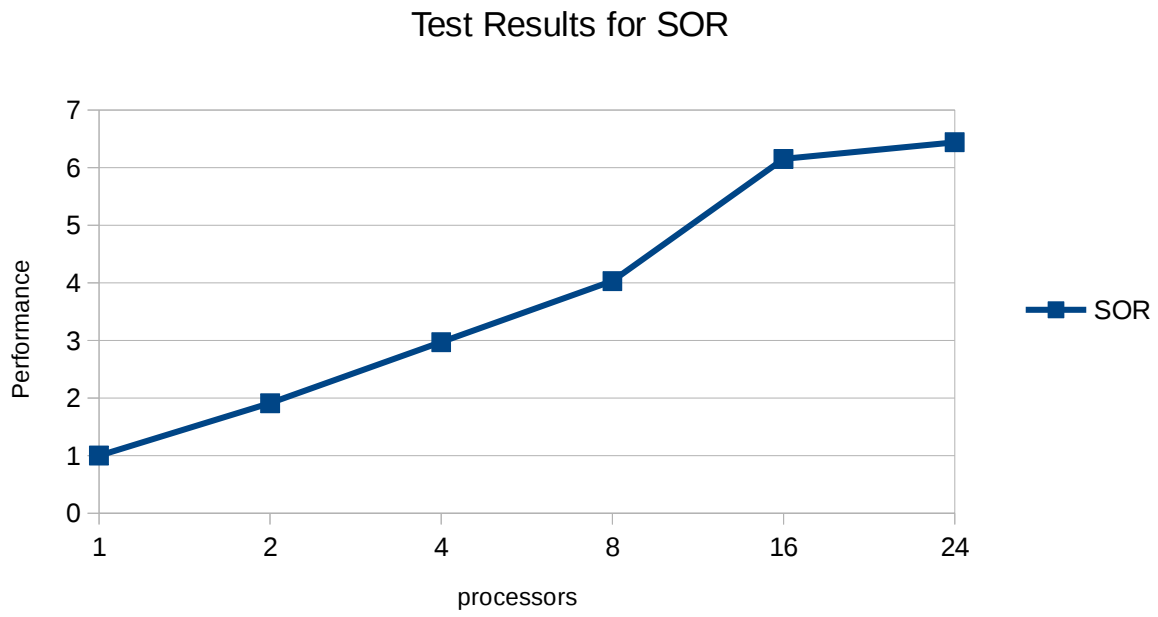
PART I: SOR TEST

1. Parameters:

Iterations = 1000 , M = 4096, N =2048, nprocs (number of threads) = 1, 2, 4, 8, 16, 24. Experiment Machine: cycle3, node2x12x1a.

2. Test Results

nprocs	1	2	4	8	16	24
TIme(sec)	41.84	21.89	14.06	10.38	6.80	6.50
Performance	1	1.91	2.97	4.03	6.15	6.44



3. Analysis

As we can see, the performance grows up as running processors increased, and the degree that the

performance enhanced by double processors is good (nearly 2x enhance/ double processors at processors 1-8).

The reason why SOR could give a great performance enhance is probably that whole part of program could be paralleled. That means, whole tasks could be divided and run in same time for reduced tasks to each processors.

PART II: GAUSS ELIMINATION

1. Introduction

In this assignment, the programs parallelize factorization step, and other steps are done by thread 0 in sequential. The programs use two different methods to assign the tasks to threads, one method is block assignment, the other is stripe assignment, executable files are denoted by gauss1_v2 (block assignment), gauss_v2(stripe assignment). And the programs run in cycle3 and node2*12*1a for distributed tests.

2. Details of the program

The main structure of both programs follows the sequential version of gauss elimination. The programs run on the following steps: initialization, compute gauss, including getting pivot, scaling the main row, factorize the rest of matrix, and solve gauss equations. Only factorization step could be paralleled.

In factorization step, due to the loop is in this formula: j = i+1; j< nsize; j++, the amount of tasks which need to assign to each threads is varied. So the programs define an global variable “num_iter”, to record the amount of total tasks to be assigned, and define local variables “begin”, “end”, to delimit the tasks to do for each threads. All the three variables need to be calculated at the beginning of the loop.

The difference between block assignment and stripe assignment is the way of loop formula. In block assignment, the loop ranges (j = begin + I; j<=end + I; j++) , whereas in stripe method, the loop ranges I ~ nize, but the step unit is task_num, which are 2,4 8, 16, etc...

Also, there are some difference between MPI version and pthread version: First, all processes have to initial matrix if they have to. MPI programs could run on distributed machines. It makes sense that the program should allocate spaces for separate machines. Second, different from the last assignment that factorization could be done in different loop iterations (j = begin + I; j <= end + I; j++, “begin” and

“end” are calculated at the beginning in each iterations), MPI's factorization cannot be done in different loop iterations. Since the function MPI_Bcast should be seen in all of the processes, which means if the factorization are done in different iterations, it will possibly cause a deadlock since some processes within the loop while some processes are out of the loop. So, all the processes should be forced to execute same time of iterations, and add condition judge to decide which process(es) should work within the loop.

3. Test Results

Table1: Test matrix jpwh_991.dat , test machine: node2x12x1a

Procs		1	2	4	8	16	24
Block Method	Time(sec)	16.93	17.12	17.35	16.83	16.87	16.60
	Normalize	1	0.90	0.98	1.01	1.00	1.01
Stripe Method	Time(sec)	16.63	17.19	16.80	16.99	17.42	16.63
	Normalize	1.01	0.98	1.01	1.00	0.97	1.02

Table2: Test matrix saylr4.dat, test machine: node2x12x1a

Procs		1	2	4	8	16	24
Block Method	Time(sec)	39.51	200.58	359.55	417.48	589.10	703.23
	Normalize	1	0.20	0.11	0.09	0.07	0.06

Table3: Test Matrix sherman3.dat, test machine: node2x12x1a

Procs		1	2	4	8	16	24
Block Method	Time(sec)	109.77	553.66	1036.89	1121.25	1527.48	1822.34
	Normalize	1	0.20	0.11	0.10	0.07	0.06

Table4: Test matrix matrix_2000.dat, test machine: node2x12x1a

Procs		1	2	4	8	16	24
Block Method	Time(sec)	7.00	37.87	67.10	87.80	113.19	140.45
	Normalize	1	0.18	0.10	0.08	0.06	0.05

YANAN ZHANG

Chart1: Comparison between two methods of task assignments

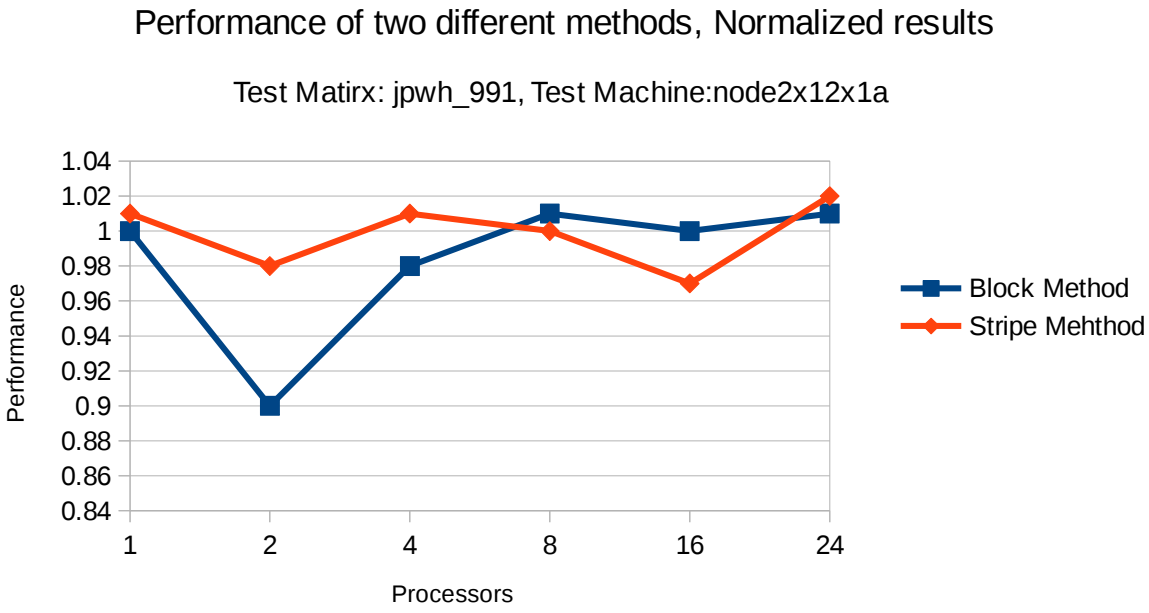
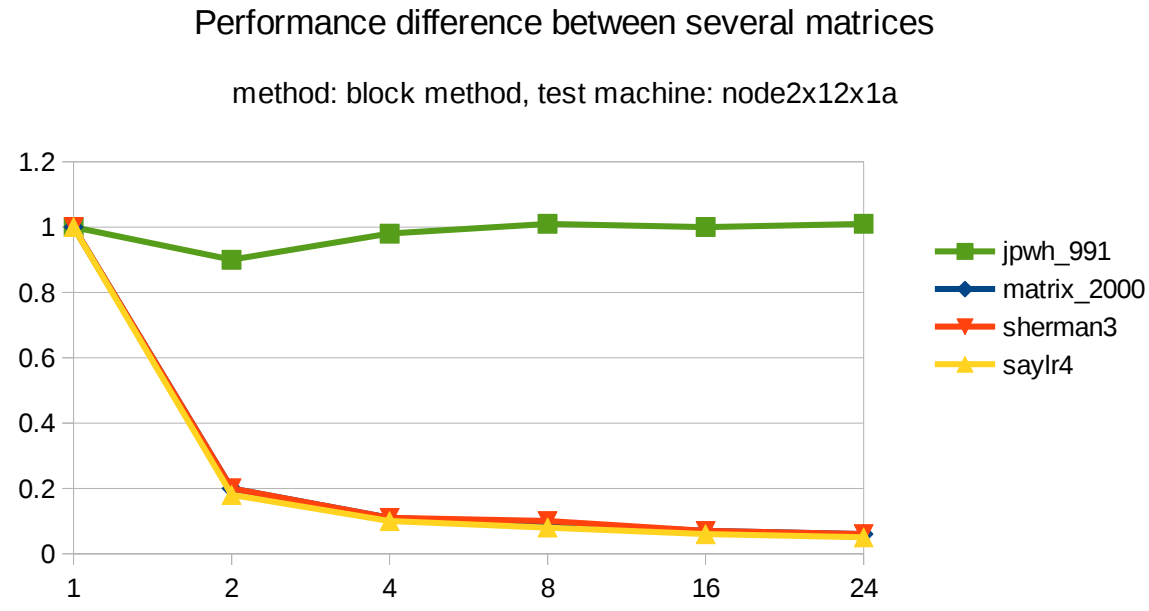


Chart2: Comparison among different input matrices



4. Conclusion from the charts

YANAN ZHANG

Unfortunately, MPI programs do not reach any enhancements and far below our expectations. As wee can see, for jpwh_991, which has a small amount of elements, it could maintain a performance in serial version. But as for other big matrices like matrix_2000, sherman3 and saylr4, the performance are extremely poor as the processes increased. I have tried to find any bugs from the program, but, there is no acutual difference between pthread version and MPI version. A possible reason for the result is that the program is not actual performing parallels although the program “writes in a parallel form”, threads may get stuck in loops, and maintaining hang on within the loop for the function MPI_Bcast. Plus, a lot of communications are needed. Take jpwh_991 as an example, for its outer loop, it needs for 991*2 times communications, plus a sum of (2+4+...990*2) ~= 490,000 times of communications among buses connecting to each processors, which means, it could cause slow down the efficiencies inevitably for bus limitations. The evidence that could support my analysis is that the program runs fine with jpwh_991, though it does not give any enhancements. But look at other more larger matrices (3000^2~5000^2 elements), the performances of paralleling are extremely poor. Buses cannot support such large amount of communications. (nearly million times of communications)

Plus, it seems that my method has some problem. In my parallel part of the program (factorizations), times of iterations should be calculated at the beginning of outer loop, which may harm the performance more intensive since the compiler cannot do any optimizations for the inner loop (factorizations), and cache block may not organized well, like, number of blocks that the cache(s) bring to the processor or memory bring to the cache(s) are not fixed. This is a possible reason, but I am not sure the poor performance are from this since I also did that in my last assignment and it works well.

Anyway, in conclusion, MPI are not suitable for this assignment, since it would cost large amount of communications and meaningless blocking synchronizations during the calculation.