CSE 6220 PA3 Report

Boqin Zhang(bzhang376) Siying Cen(scen9) Yi Jiao(yjiao47)

April 2021

1 Introduction

In this programming assignment, we solved linear equations by using Jacobi's iterative method. Both sequential algorithm and parallel algorithm were implemented, tested and analyzed on large matrix.

2 Parallel Algorithm

Jacobi's Method
According to the provided framework, the parallel algorithm of Jacobi's Method is listed as follows.
1 Distribute matrix A and vector b to the 2-dimensional mesh communication network.
2 Calculate D, D^{-1} and R.
3 Initialize answer x.
4 Calculate Ax, $Ax - b$ and L2-norm of $Ax - b$.
5 If the L2-norm is greater the given threshold, update x with $D^{-1}(b - Rx)$ and repeat Step 4.
6 gather the answer vector x to processor 0.

Data distribution
Data is evenly distributed in the 2-dimensional mesh network. For matrix, it is evenly distributed onto the grid and each processor holds a portion of the matrix. For vector, it is evenly distributed onto the first column.

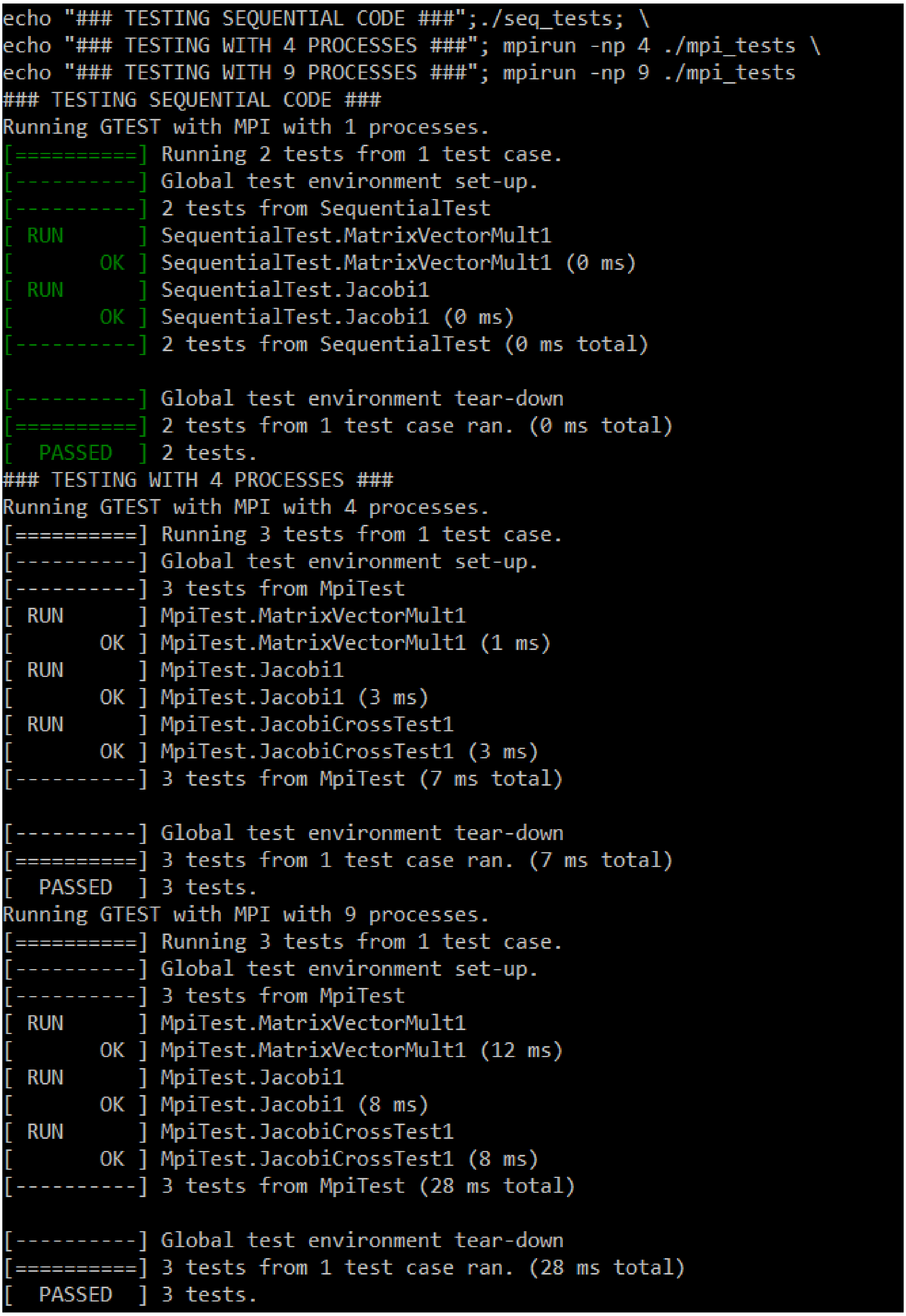
Matrix vector multiplication
Firstly send the elements distributed in the first column to their corresponding diagonal processors. Secondly broadcast the vector elements along each column. Thirdly conduct local matrix-vector multiplication in each processor. Finally reduce the result along the row to the first column.

3 Optimizations

The function to distribute the matrix to all processors is composed of two steps. The first step is the (0, 0) processor scatters the rows of the matrix to corresponding processors in the first column (i, 0); the second step is the processors in the first column (i, 0) scatter the received lines of the matrix to processors in the same row (i, j). The (i, 0) processor scatters the matrix line by line.
By using this method, the distribution of the matrix in a row of the Cartesian mesh gird can be independent to other rows. The distribution in rows can be executed in parallel.

4 Experiment

(1) First, we checked if our sequential and parallel algorithm were implemented correctly by running "*seq_tests*", "*mpi_tests*" and "*check_output.py*". Our scripts pass all the tests.

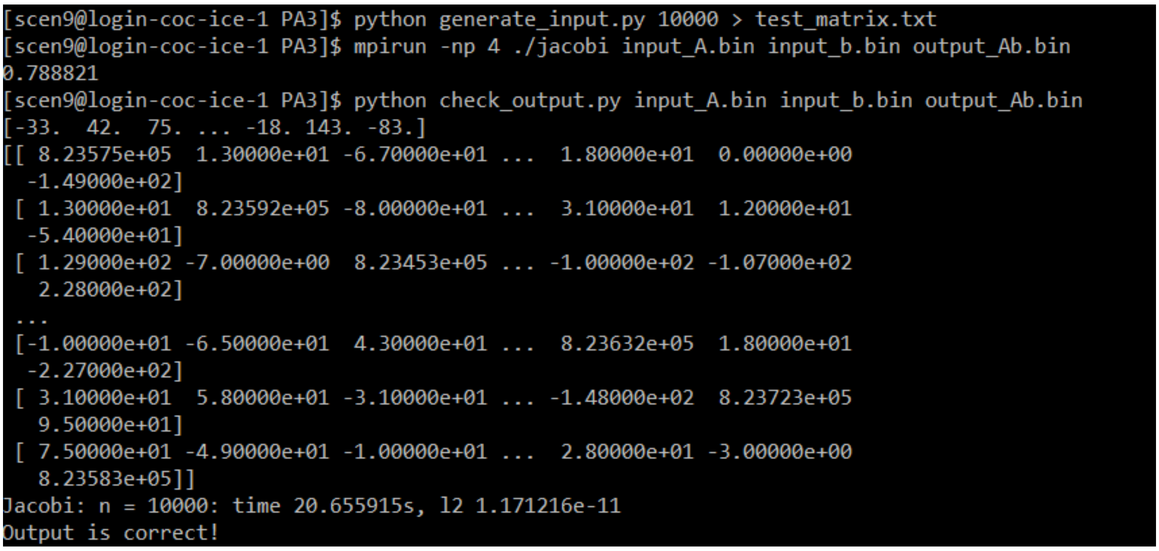


```
echo "### TESTING SEQUENTIAL CODE ###";./seq_tests; \
echo "### TESTING WITH 4 PROCESSES ###"; mpirun -np 4 ./mpi_tests \
echo "### TESTING WITH 9 PROCESSES ###"; mpirun -np 9 ./mpi_tests
### TESTING SEQUENTIAL CODE ###
Running GTEST with MPI with 1 processes.
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from SequentialTest
[ RUN      ] SequentialTest.MatrixVectorMult1
[       OK ] SequentialTest.MatrixVectorMult1 (0 ms)
[ RUN      ] SequentialTest.Jacobi1
[       OK ] SequentialTest.Jacobi1 (0 ms)
[----------] 2 tests from SequentialTest (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 2 tests.
### TESTING WITH 4 PROCESSES ###
Running GTEST with MPI with 4 processes.
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from MpiTest
[ RUN      ] MpiTest.MatrixVectorMult1
[       OK ] MpiTest.MatrixVectorMult1 (1 ms)
[ RUN      ] MpiTest.Jacobi1
[       OK ] MpiTest.Jacobi1 (3 ms)
[ RUN      ] MpiTest.JacobiCrossTest1
[       OK ] MpiTest.JacobiCrossTest1 (3 ms)
[----------] 3 tests from MpiTest (7 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (7 ms total)
[  PASSED  ] 3 tests.
Running GTEST with MPI with 9 processes.
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from MpiTest
[ RUN      ] MpiTest.MatrixVectorMult1
[       OK ] MpiTest.MatrixVectorMult1 (12 ms)
[ RUN      ] MpiTest.Jacobi1
[       OK ] MpiTest.Jacobi1 (8 ms)
[ RUN      ] MpiTest.JacobiCrossTest1
[       OK ] MpiTest.JacobiCrossTest1 (8 ms)
[----------] 3 tests from MpiTest (28 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (28 ms total)
[  PASSED  ] 3 tests.
```

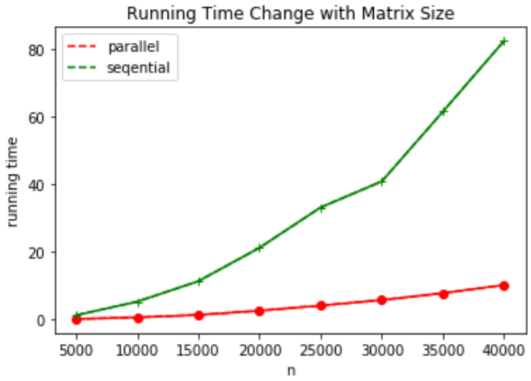


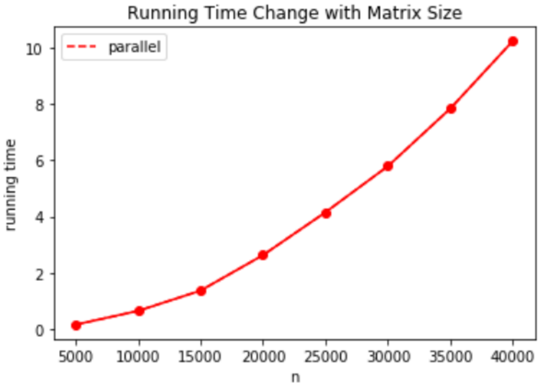
```
[scen9@login-coc-ice-1 PA3]$ python generate_input.py 10000 > test_matrix.txt
[scen9@login-coc-ice-1 PA3]$ mpirun -np 4 ./jacobi input_A.bin input_b.bin output_Ab.bin
0.788821
[scen9@login-coc-ice-1 PA3]$ python check_output.py input_A.bin input_b.bin output_Ab.bin
[-33.  42.  75. ... -18. 143. -83.]
[[ 8.23575e+05  1.30000e+01 -6.70000e+01 ...  1.80000e+01  0.00000e+00
  -1.49000e+02]
 [ 1.30000e+01  8.23592e+05 -8.00000e+01 ...  3.10000e+01  1.20000e+01
  -5.40000e+01]
 [ 1.29000e+02 -7.00000e+00  8.23453e+05 ... -1.00000e+02 -1.07000e+02
   2.28000e+02]
 ...
 [-1.00000e+01 -6.50000e+01  4.30000e+01 ...  8.23632e+05  1.80000e+01
  -2.27000e+02]
 [ 3.10000e+01  5.80000e+01 -3.10000e+01 ... -1.48000e+02  8.23723e+05
   9.50000e+01]
 [ 7.50000e+01 -4.90000e+01 -1.00000e+01 ...  2.80000e+01 -3.00000e+00
   8.23583e+05]]
Jacobi: n = 10000: time 20.655915s, l2 1.171216e-11
Output is correct!
```

(2) After confirming that our algorithm are correct, we tested the running time of sequential and parallel algorithm by changing matrix size n (n = 5000, 10000, 15000, 20000, 25000, 30000, 35000, 40000), number of processors p (p = 1, 4, 9, 16, 25, 36, 49, 64) and difficulty d (d = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0). All the tests were running on PACE cluster and the results will be shown in the following section.
Calculate Speedup according to running results. Plot results.

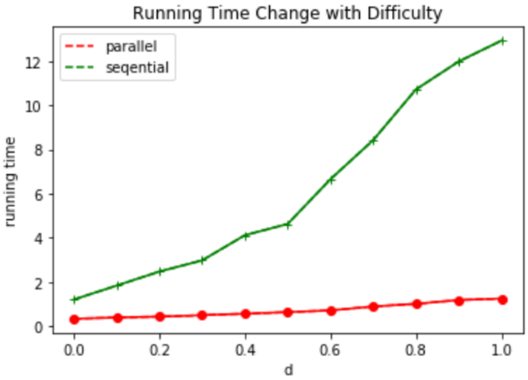
5 Results

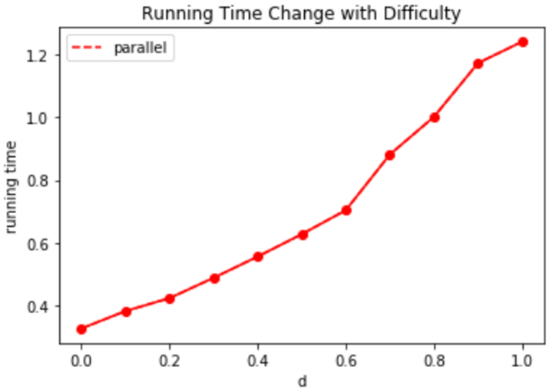
(1) Running time (in second) changes with matrix size.
($p = 16, d = 0.5$)



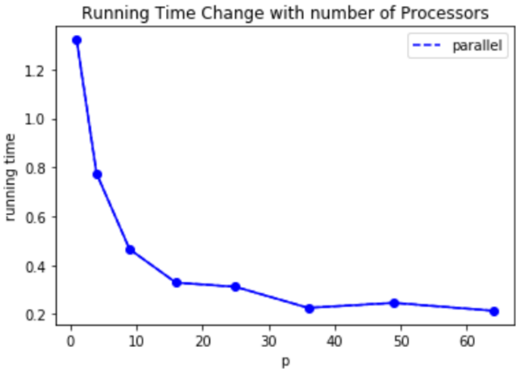


(2) Running time (in second) changes with difficulty.
(n = 10000, $p = 16$)



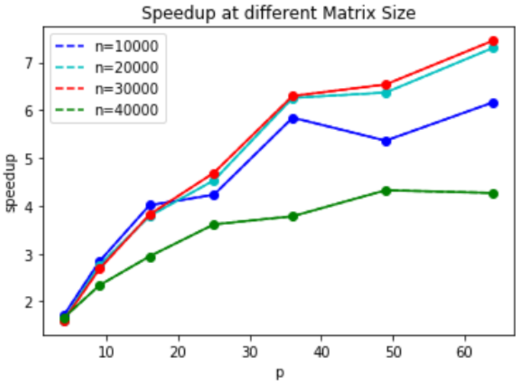


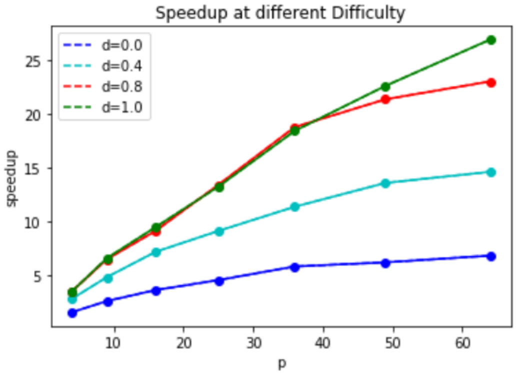
(3) Running time (in second) changes with number of processors.
$(n = 10000, d = 0.5)$



(4) Speedup when changing matrix size $n * n$.
$(d = 0.5)$

Speedup when changing difficulty d.
$(n = 10000)$





6 Conclusion

(1) Overall, our parallel algorithm runs faster than sequential algorithm on testing large matrix $(n * n \leq 10000 * 10000)$, no matter choosing which matrix size or difficulty.

(2) According to result (1), when increasing matrix size n, the runtime of both sequential and parallel algorithm increases, and it seems to increase exponentially.

(3) According to result (2), when increasing difficulty d, the runtime of both sequential and parallel algorithm increases, which is consistent with the assumption that 'Increasing difficulty increases the number of iterations needed for Jacobi's method to converge'.

(4) According to result (3), when increasing the number of processors from 1, 4,9 to 64, the runtime always decreases. This demonstrates that our parallel algorithm is designed elegantly. However, after $p = 16$, the decreasing rate of runtime slows down due to the increasing communications cost between processors. So we choose $p = 16$ to test different matrix size or difficulty.

(5) According to result (4), changing matrix size does not influence the speedup too much, while changing difficulty is more related to speedup. When increasing difficulty, the speedup also increases.