Programming of Supercomputer Assignment 4

Apoorva Gupta

Razieh Rezaei

ShengHsuan Lin

MPI Collectives

Q2.2.1 Patterns for collective communication

During Setup, We see,

____
>> rank0: Send
>> all other: Receive

____
This can be replaced by Bcast   (Sending from same memory)
This can be replaced by Scatter (Sending from shifted memory)

```
if(rank == 0) {
        for(i = 1; i < size; i++){
                MPI_Send(&rows, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
                MPI_Send(&columns, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
} else {
        MPI_Recv(&rows, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        MPI_Recv(&columns, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
}
```

Q2.2.1 Patterns for collective communication

We also see, during computation,

_____
rank: receive from processes 0 to rank-1
Some Computation
rank: send to processes rank+1 to size

_____
This can be replaced by Bcast.

```
for(process = 0; process < rank; process++) {
        mpi_start = MPI_Wtime();
        MPI_Recv(pivots, (local_block_size * rows + local_block_size + 1), MPI_DOUBLE, process, process, MPI_COMM_WORLD, &status);
        mpi_time += MPI_Wtime() - mpi_start;
```

```
   for (process = (rank + 1); process < size; process++) {
           pivots[0] = (double) rank;
           mpi_start = MPI_Wtime();
           MPI_Send( pivots, (local_block_size * rows + local_block_size + 1), MPI_DOUBLE, process, rank, MPI_COMM_WORLD);
           mpi_time += MPI_Wtime() - mpi_start;
   }
```

Q2.2.1 Patterns for collective communication

We also see, after the computation,

_____
>> rank0: receive
>> all other: send

_____
This can be replaced by gather.

```
//      send/receive solutions
if(rank == 0) {
        for(i = 0; i < local_block_size; i++){
                solution[i] = solution_local_block[i];
        }
        mpi_start = MPI_Wtime();
        for(i = 1; i < size; i++){
                MPI_Recv(solution + (i * local_block_size), local_block_size, MPI_DOUBLE, i, 0, MPI_COMM_WORLD, &status);
        }
        mpi_time += MPI_Wtime() - mpi_start;
} else {
        mpi_start = MPI_Wtime();
        MPI_Send(solution_local_block, local_block_size, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
        mpi_time += MPI_Wtime() - mpi_start;
}
```

Q2.2.2 Non-blocking collectives

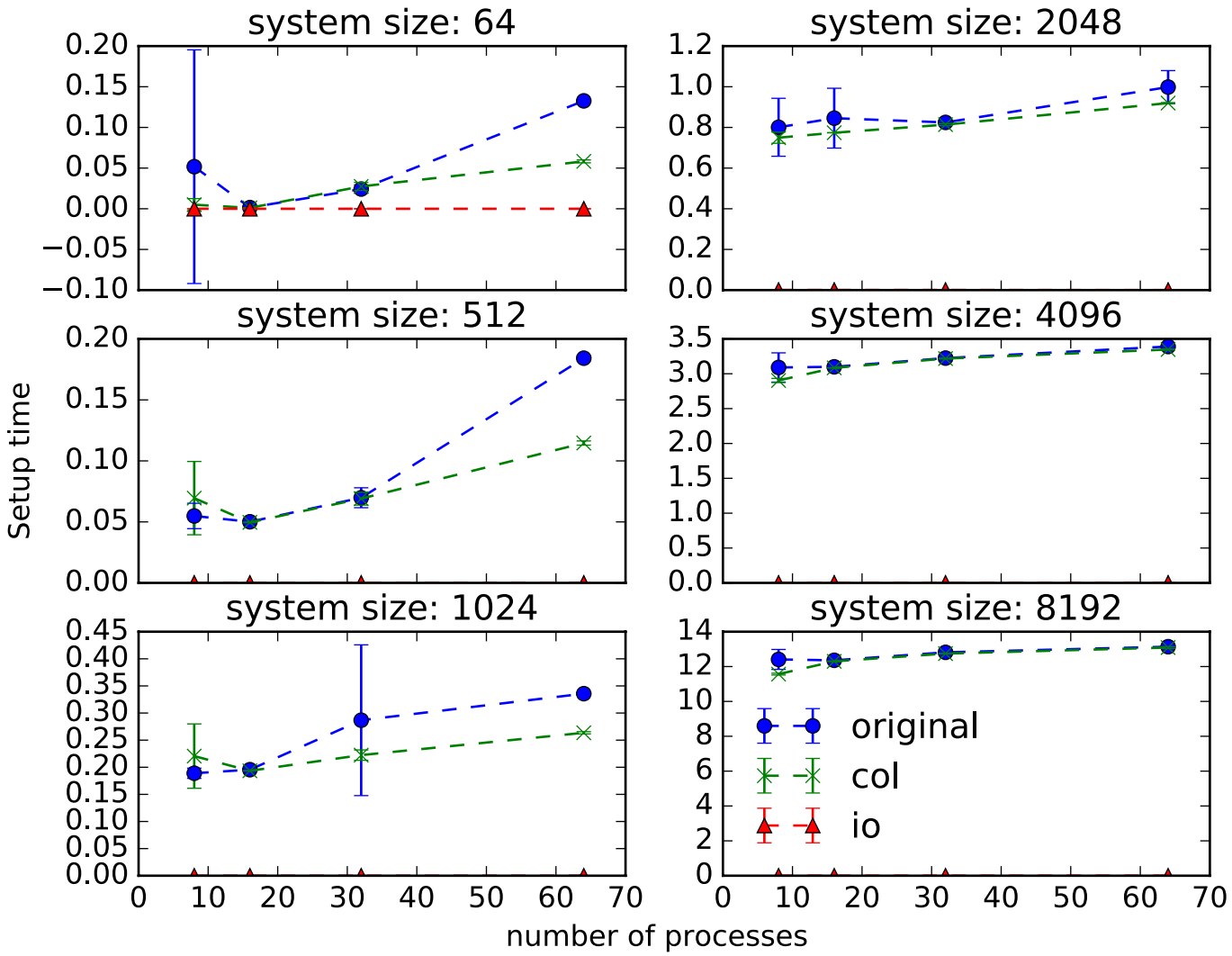
- No potential for overlap of communication and computation was identified.
- Same reason as last assignment: data dependency between each steps

Q2.2.3 Performance in Collective communication

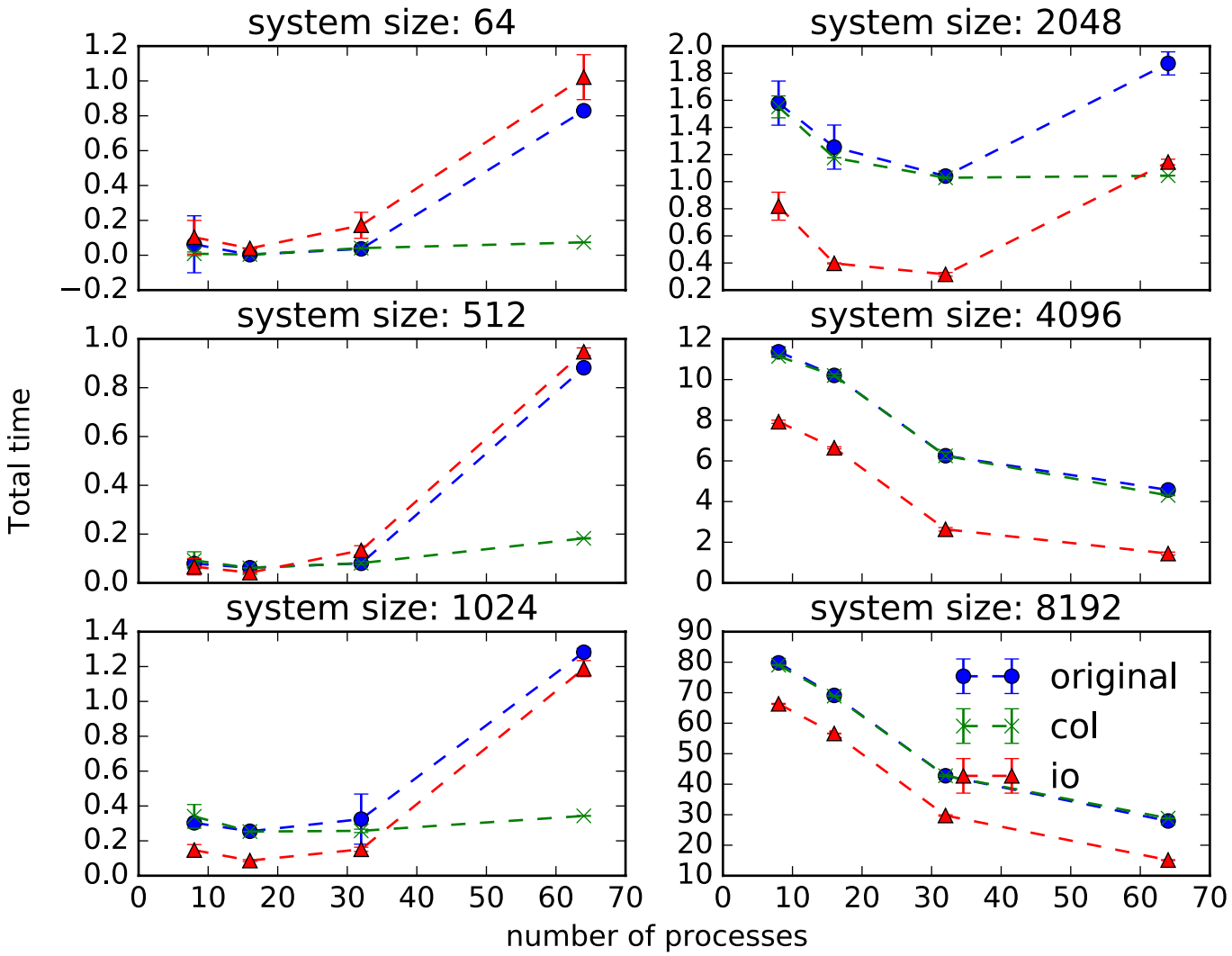
Yes, in the case of small problem size and large number of processes, there is improvement in performance.

The communication time does not dominate in larger size problem. No obvious improvement in performance for large size problem (in the plot)

SETUP TIME


system size: 64
system size: 2048
system size: 512
system size: 4096
system size: 1024
system size: 8192
Setup time
number of processes
original
col
io

Total TIME


system size: 64
system size: 2048
system size: 512
system size: 4096
system size: 1024
system size: 8192
Total time
number of processes
original
col
io

Q2.2.4 Readability

- Advantages
  - Code is easier to read and understand
  - Because receive ← send, code is harder to read.
  - Bcast is one command, while send/receive are two command.
  - Communication is taken care of.
- Disadvantages:
  - Maintaining different groups and communicators

MPI Parallel IO

Q3.2.1

Before:

fopen → fscanf → fclose

Parallel IO by:

MPI_File_open

MPI_File_read / MPI_File_read_at

MPI_File_close

Q3.2.2 2Phase I/O

- Improves I/O performance for non-contiguous data patterns.
- Without this each process has many file accesses for non-contiguous data.
- Leads to poor performance due to I/O overhead.
- TO-IO divides the data (including data gaps) into chunks equal and make each process reads its aligned chunk into memory.
- Then each process does data exchange operations to get the correct data into its memory.

Q3.2.2 Data Sieving

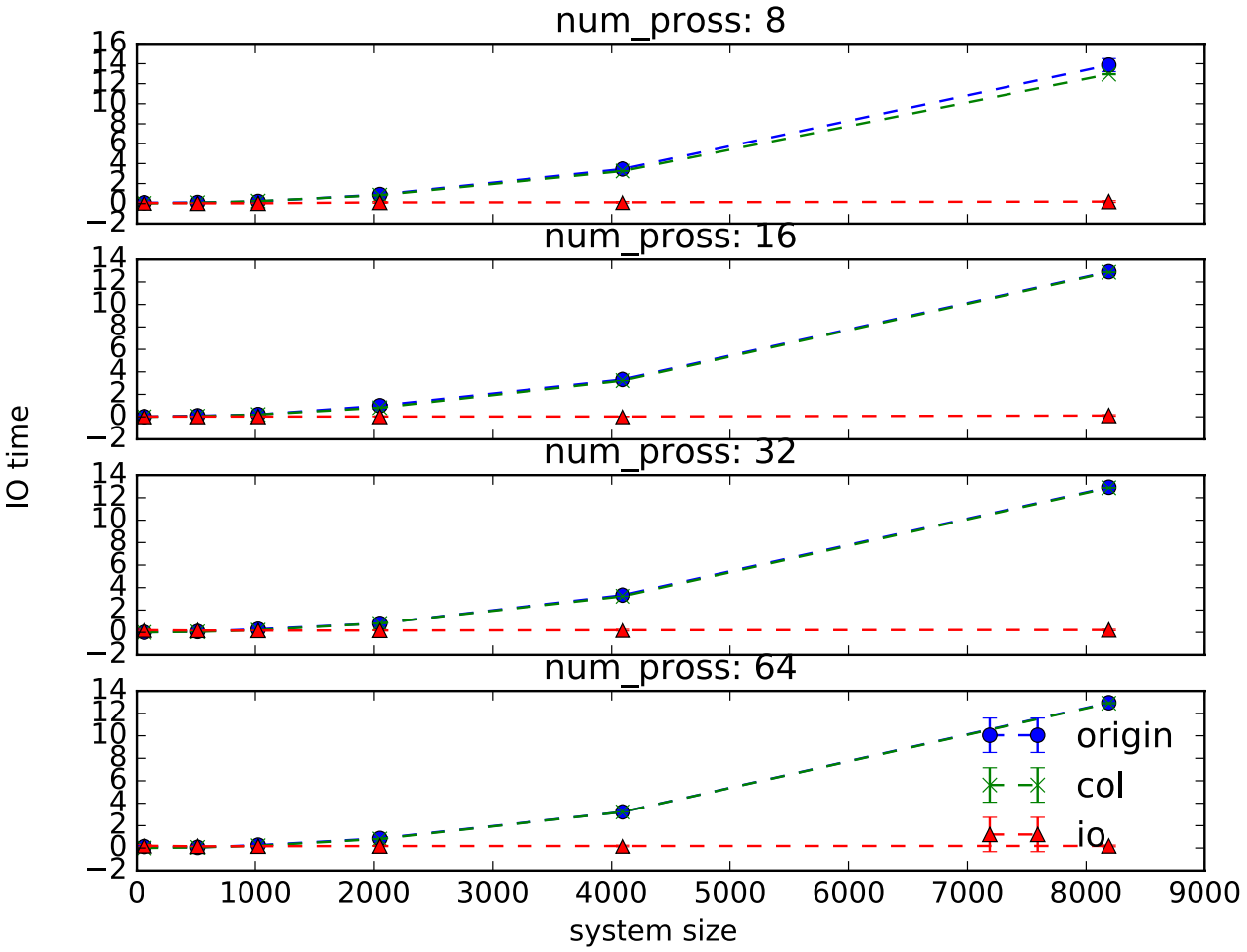
- Requires very few I/O requests compared to the direct method.
- In the read case it reads the entire contiguous chunk starting from the lowest offset of all requests to the highest offset of all requests.
- This contiguous chunk includes non-requested data, also called holes.
- Then the data sieving approach sieves out non-requested data.
- The demanded data are kept and copied into user buffer.

Q 3.2.3

The original implementation is not scalable in terms of IO, because

1. Increasing time with increasing problem size.
2. Always using only one process to perform IO.

IO TIME

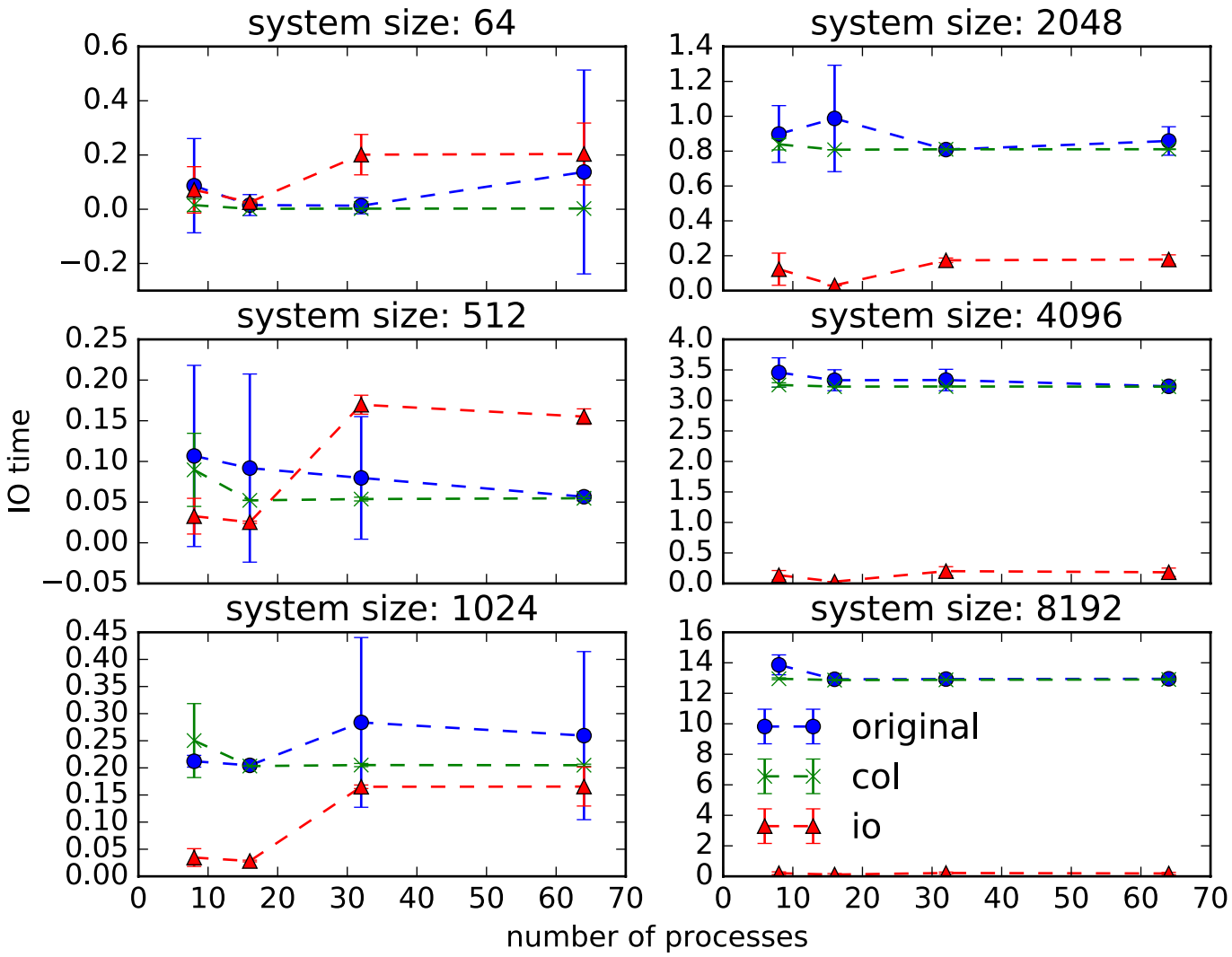

num_pross: 8
num_pross: 16
num_pross: 32
num_pross: 64
IO time
system size
origin
col
io
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

Q 3.2.3

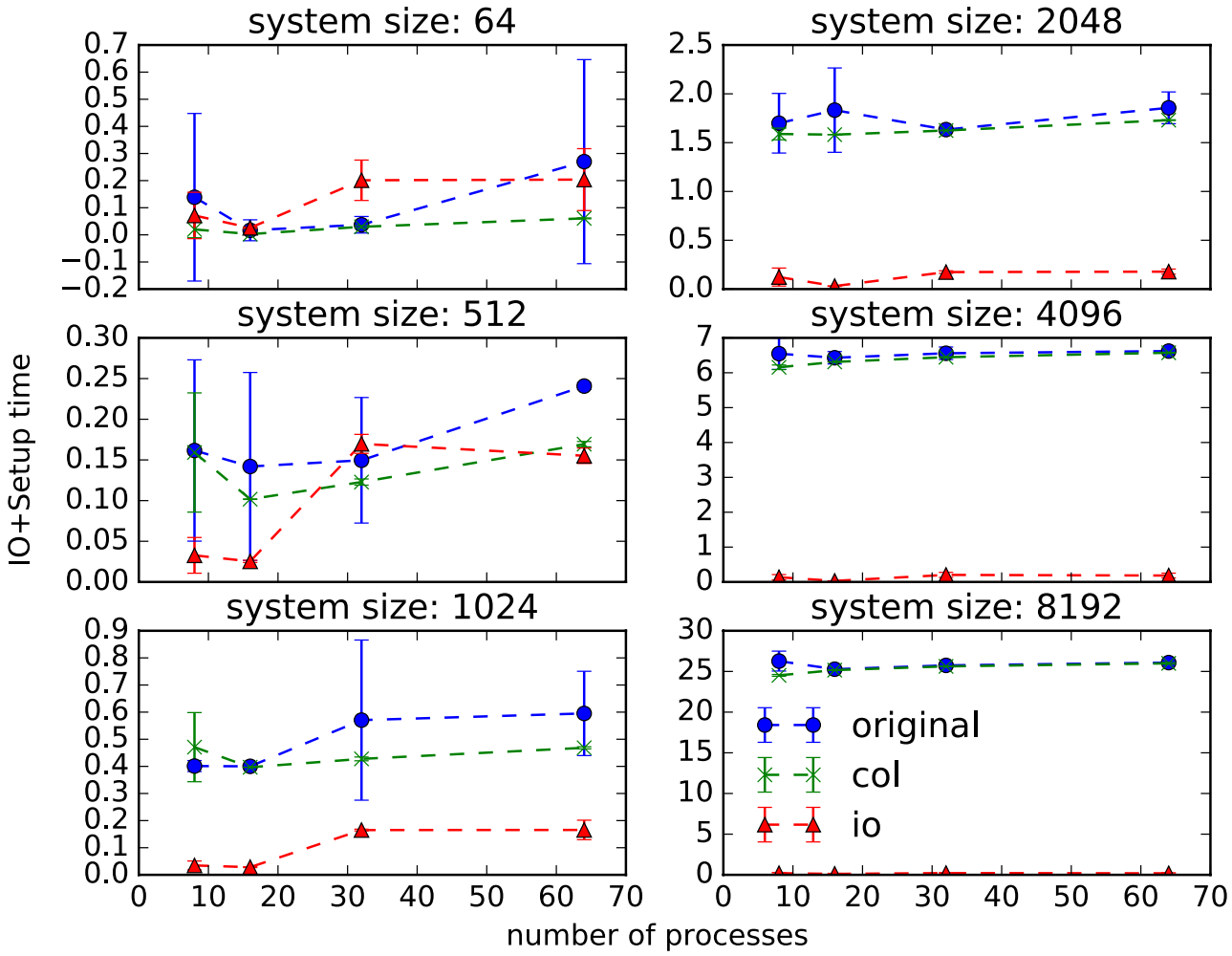
The original implementation is not scalable in terms of IO, because

1. Increasing time with increasing problem size.
2. Always using only one process to perform IO.

IO TIME


system size: 64
system size: 2048
system size: 512
system size: 4096
system size: 1024
system size: 8192
IO time
number of processes
original
col
io

IO+SETUP TIME



3.2.4

The original implementation is not scalable in terms of RAM.

Limit on the RAM on one node.

Q3.2.5

Before:

- **fopen → fscanf → fclose**
- fprintf
- **setup part: MPI_Send, MPI_Receive**
- **Solution collection: MPI_Send, MPI_Receive**

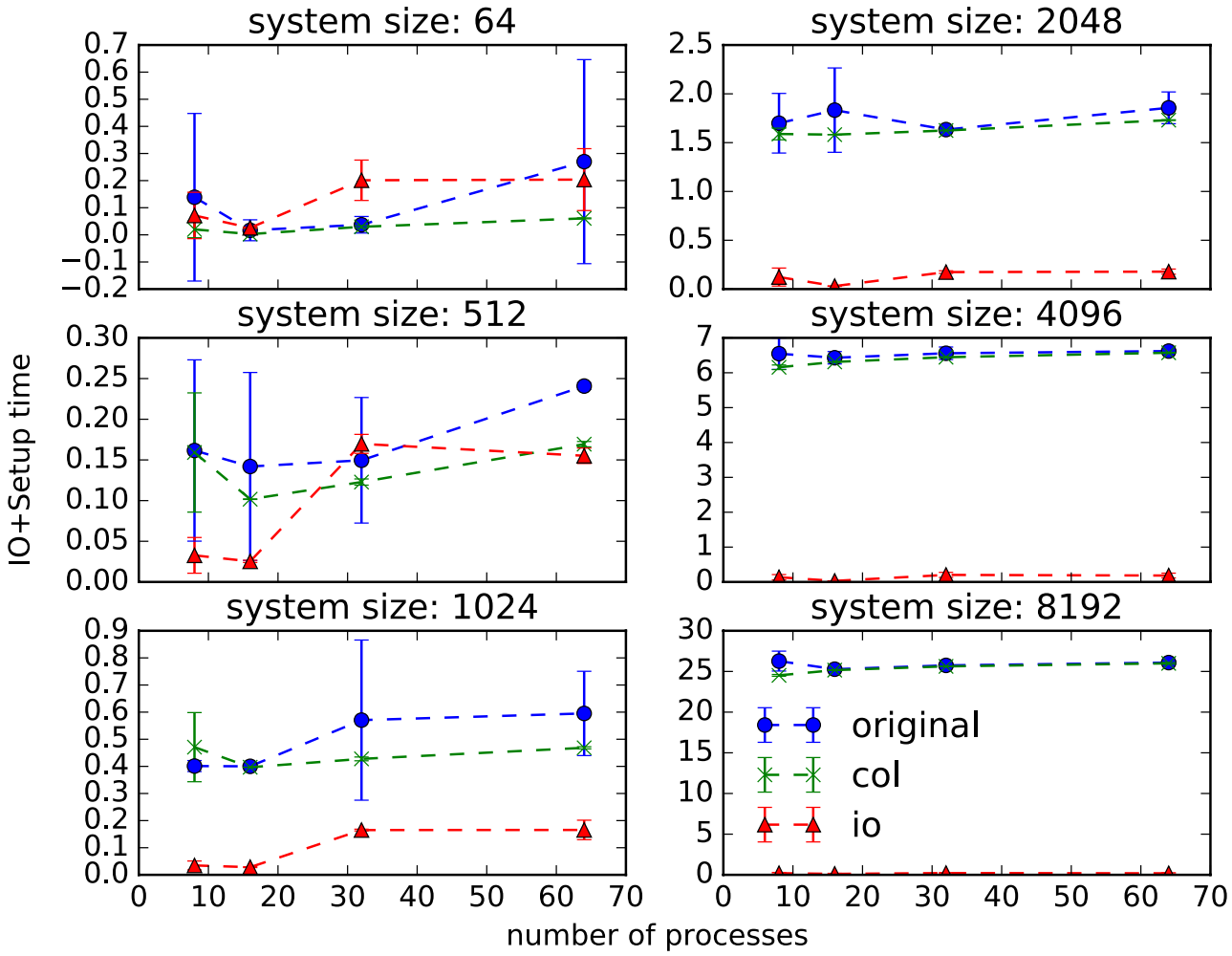
Replaced by:

- **MPI_File_open, MPI_File_read, MPI_File_close**
- MPI_File_write_at
- ~~**Setup part:**~~
- ~~**Solution collection**~~

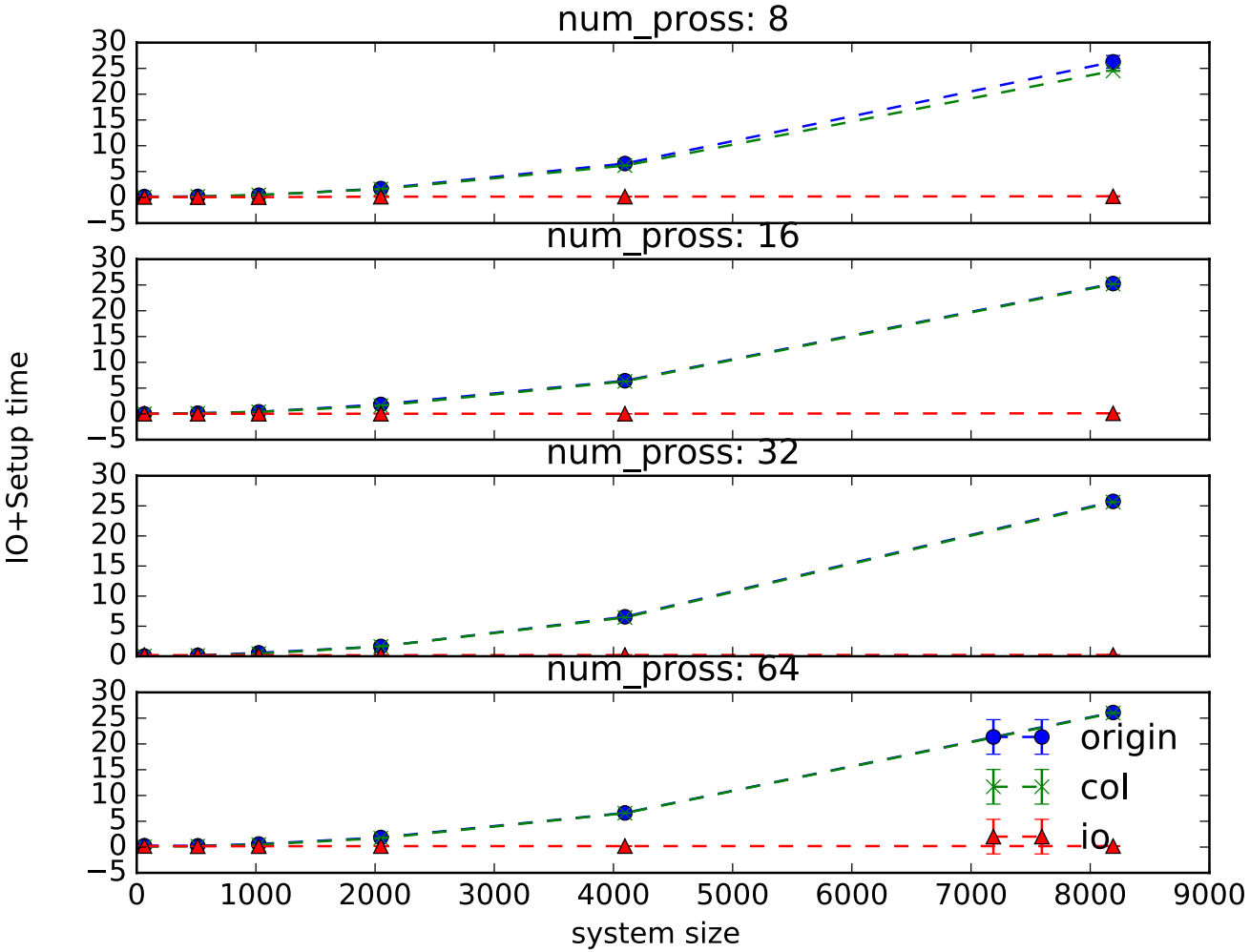
Q 3.2.6

Yes, Performance Improves.

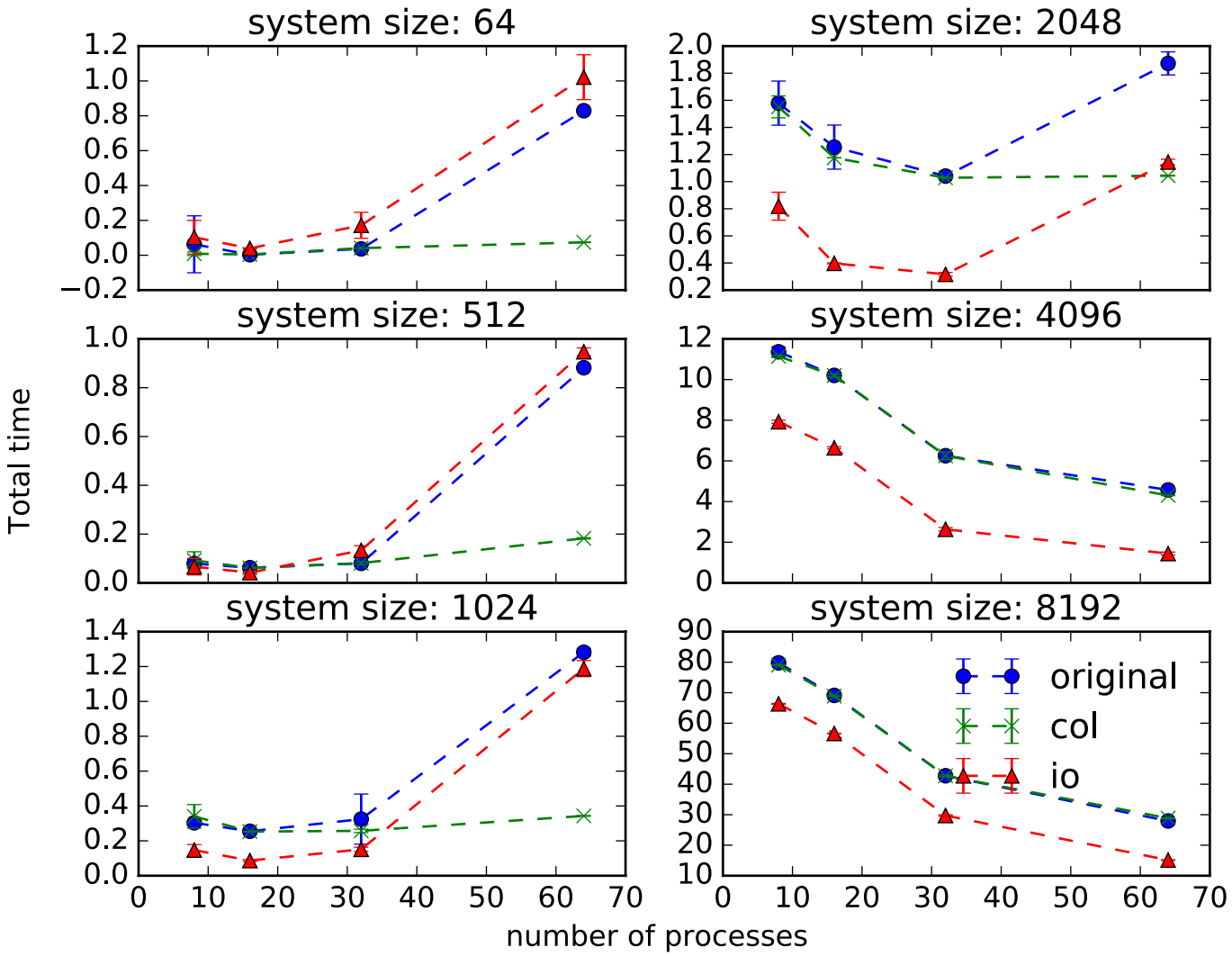
IO+SETUP TIME



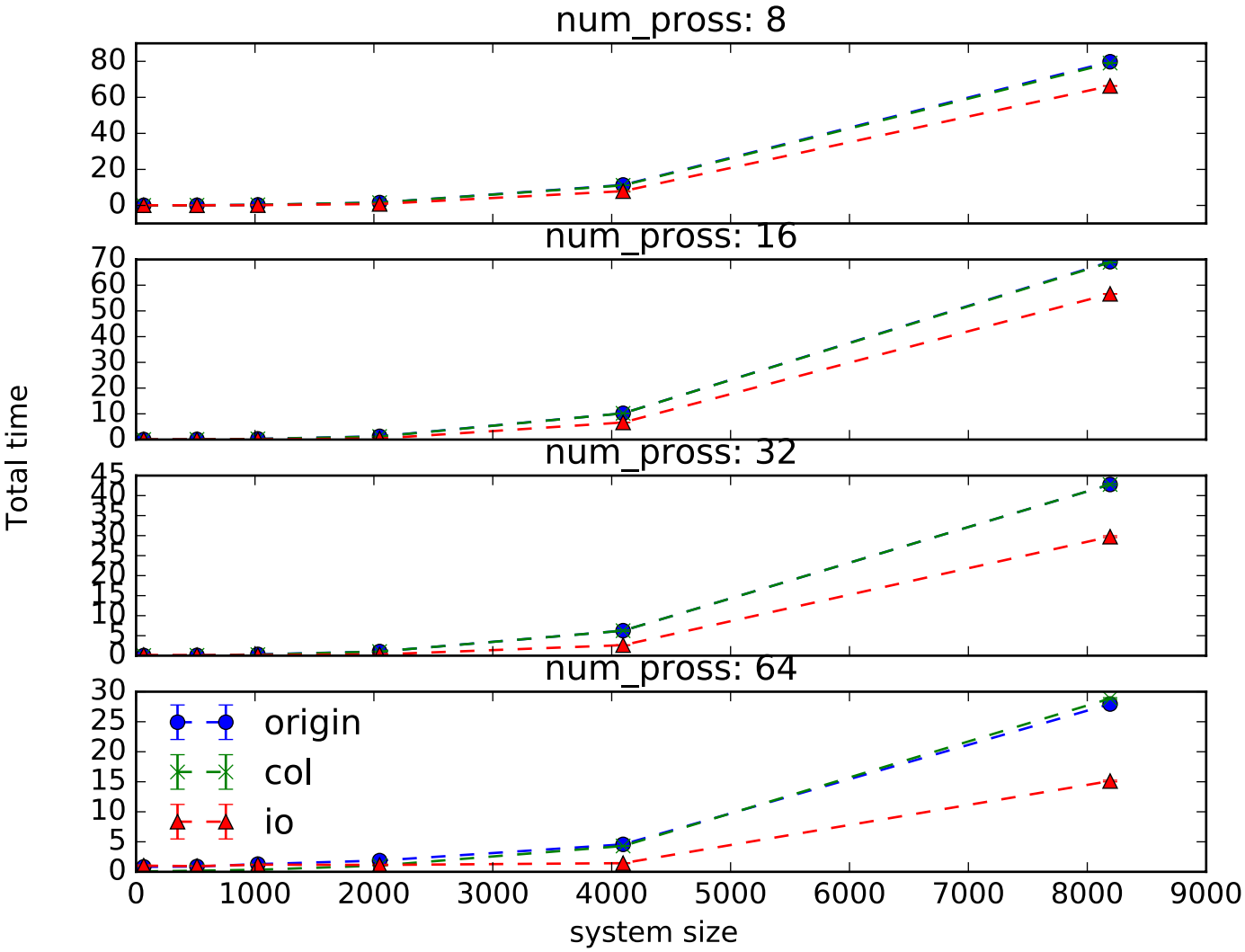
IO+SETUP TIME



Total TIME


system size: 64
system size: 2048
system size: 512
system size: 4096
system size: 1024
system size: 8192
Total time
number of processes
original
col
io

Total TIME


num_pross: 8
num_pross: 16
num_pross: 32
num_pross: 64
Total time
system size
origin
col
io
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

THE END