# An optimization of particle information exchange using one-sided communication for the MPS method

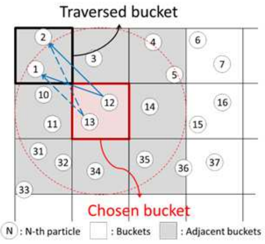
**Aoto Abe**, Kazumi Kayajima, Dai Wada and Takaaki Miyajima
Department of Computer Science, School of Science and Technology, Meiji University, Japan
e-mail:takaaki.miyajima@cs.meiji.ac.jp

## Background

### Moving Particle Simulation method

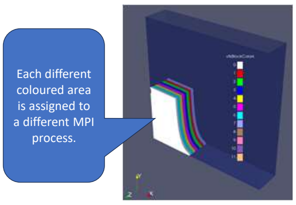
- ✓ Moving Particle Simulation (MPS) method is one of the computational methods for simulating fluid behaviour, classified as a particle-based method.
- ✓ The motion of each particle is calculated through interactions with the neighbour-particles.

### Dynamic domain decomposition



- ✓ Dynamic domain decomposition and load balancing are mandatory for large-scale simulation of the MPS method.
- ✓ The computational domain is divided into small lattices called "buckets".
- ✓ The number of particles in each bucket are different.

### Inter domain communication with MPI



Each different coloured area is assigned to a different MPI process.

- ✓ In the distributed-memory system, the computational domain is divided into subdomains consisting of multiple buckets
- ✓ Each subdomain is assigned to each MPI process.
- ✓ Particle information are needed to communicate for…

1. The particles may be moved to another MPI sub-region.
2. The particles required for neighbour-particle search may exist in another node.

### Communication steps of particle information exchange

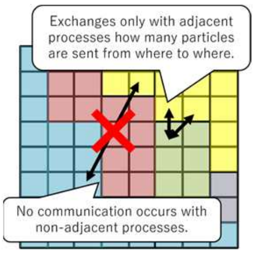
Step 1 : Exchange information on the number of particles to be transferred from where to where.

Step 2 : Transfer actual particle data.

We focused here.

## Proposed technique

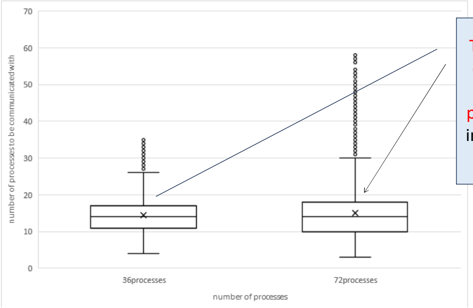
### A physical background of particle movement



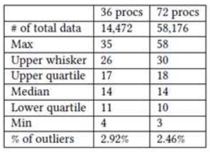
Exchanges only with adjacent processes how many particles are sent from where to where.

No communication occurs with non-adjacent processes.

- ✓ Particle information exchange happens only between neighbour buckets.
- ✓ The particles only move from one process to adjacent processes.
- ✓ As illustrated in figure blow, the number of processes adjacent to a process is smaller than the number of all processes when the computational domain is decomposed.



The increase in the number of adjacent processes with the total number of processes is slower than the increase in the total number of processes ?



| | 36 procs | 72 procs |
|---|---|---|
| # of total data | 14,472 | 58,176 |
| Max | 35 | 58 |
| Upper whisker | 26 | 30 |
| Upper quartile | 17 | 18 |
| Median | 14 | 14 |
| Lower quartile | 11 | 10 |
| Min | 4 | 3 |
| % of outliers | 2.92% | 2.46% |

### One-sided communication instead of all-to-all communication

- ✓ The naïve implementation employs collective communication even if there is no particle to move. (all-to-all is used)
- ✓ One-sided communication can reduce synchronous waiting and intermediate buffer data copying compared to one-to-one or collective communication.

### Active target in One-sided communication

**Active target** ★

- ✓ Both source and destination processes synchronise.
- ✓ The section is enclosed by the MPI_Win_fence where communication is possible.
- ✓ It is effective when the destination changes frequently or when there are many destinations.

**Passive target**

- ✓ Source only synchronise.
- ✓ The section is enclosed by the MPI_Win_lock and MPI_Win_unlock where the sender can access.
- ✓ It is inefficient when the communication destination changes frequently.

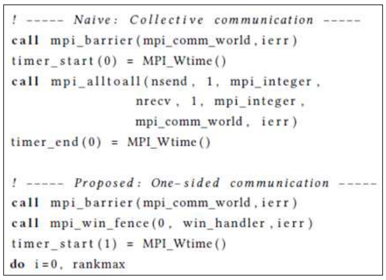
## Evaluation and result

### Test program modelling the particle information exchange

- ✓ This test program compares and evaluates the time required for MPI communication concerning the proposed technique.

1. The naïve implementation uses MPI_Alltoall.
2. The proposed implementation uses MPI_Put.

- ✓ The test data is extracted from the particle information exchange in original program.

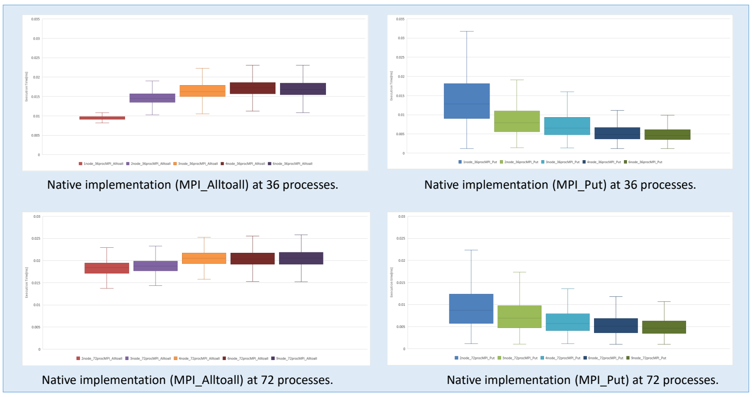


```
! ----- Naive: Collective communication -----
call mpi_barrier(mpi_comm_world,ierr)
timer_start(0) = MPI_Wtime()
call mpi_alltoall(nsend, 1, mpi_integer,
                  nrecv, 1, mpi_integer,
                  mpi_comm_world, ierr)
timer_end(0) = MPI_Wtime()

! ----- Proposed: One-sided communication -----
call mpi_barrier(mpi_comm_world,ierr)
call mpi_win_fence(0, win_handler,ierr)
timer_start(1) = MPI_Wtime()
do i=0, rankmax
```

Test program comparing collective and one-sided communication

### Result: Data communication times



Native implementation (MPI_Alltoall) at 36 processes.

Native implementation (MPI_Put) at 36 processes.

Native implementation (MPI_Alltoall) at 72 processes.

Native implementation (MPI_Put) at 72 processes.

- ✓ The data exchange was completed in shorter times with MPI_Put compared to MPI_Alltoall when comparing median values.
- ✓ The median time was reduced to 1/3.63 for six nodes with 36 processes. 1/4.38 for nine nodes with 72 processes.
- ✓ Latency (rather than throughput) determines performance.
- ✓ The size of data to be sent and received is 4 bytes x (# of processes).

## Conclusion

- ✓ We propose a technique using one-sided communication for exchanging information on the number of particles among processes in the MPS method using the bucket.
- ✓ We made and ran a test program modelling the particle information exchange.
- ✓ Compared to the naïve implementation, the proposed technique's communication time (median) was reduced by a factor of 1/3.63 for six nodes in 36 processes and by a factor of 1/4.38 for nine nodes in 72 processes.
- ✓ In future, we will verify the generality of the proposed method.
- ✓ Using a larger number of processes or system with different network topologies.
- ✓ Other domain decomposition methods.

## Acknowledgements

This work used computational resources TSUBAME3.0 supercomputer provided by Tokyo Institute of Technology through the HPCI System Research Project. The experiments were also conducted using the JAXA supercomputer JSS3.