SUSTech Southern University of Science and Technology

APAC HPC-AI 2024 Final Presentation

SUSTech-0x01

5 Noverber 2024

About us

SUSTech Southern University of Science and Technology

We are **SUSTech 0x01 Team**:

- Under supervision and support of **Centre for Computational Science and Engineering in SUSTech**, we formed a diverse and experienced award-winning team
- We aim to **foster next generation of HPC talents** through actual scientific research and attending international HPC challenges
- At 0x01, we have **abundant computational resources** for student to maximize their practical HPC skills, as well as **systematic and cutting-edge training**

HPC Groups:

- ZuDong Li (leader)
- Haibin Lai
- Benxiang Xiao
- Zixu Wang
- Wenhan Tan

AI Groups:

- Yukun Yang
- Honglie Li
- Junyu Su

- **Computer Science**
- **Mechanical Engineering**
- **Data Science**
- **Electrical Engineering**
- **Mathematics**

Outline

SUSTech Southern University of Science and Technology

- HOOMD-blue
- Lightgpt-llama2
- Summary

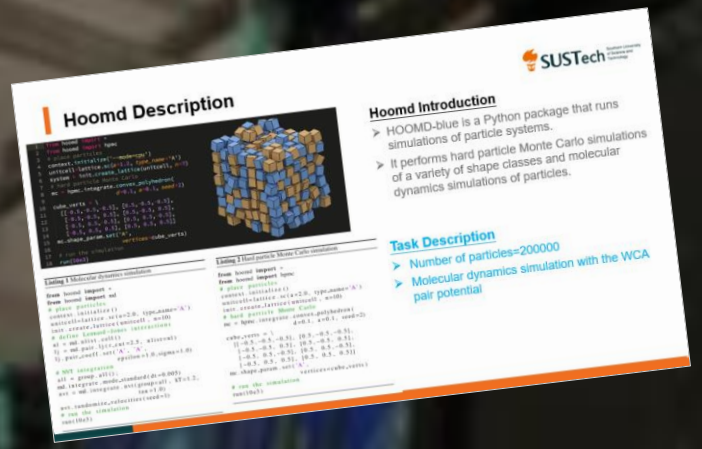
SUSTech Southern University of Science and Technology

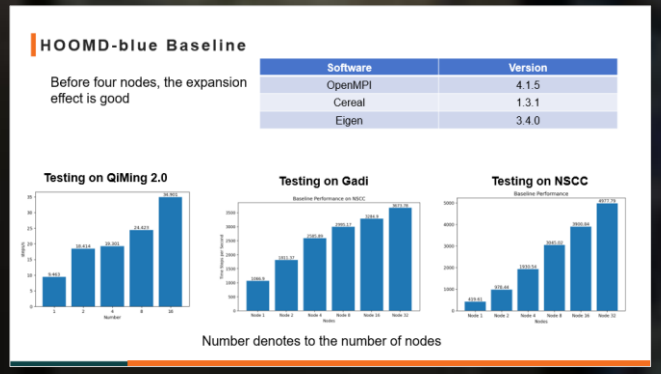
Part 1

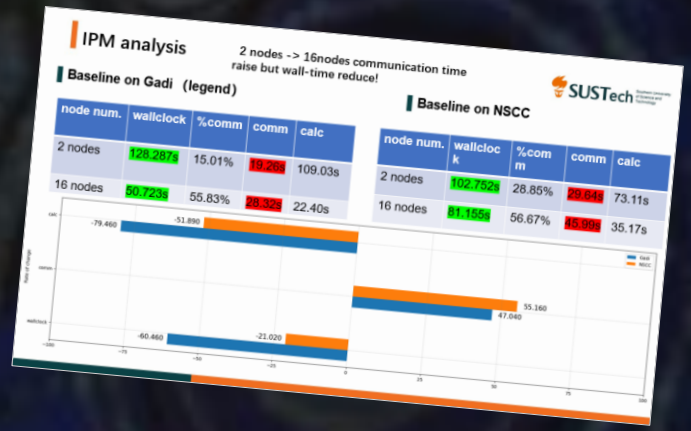
High-Performance Computing

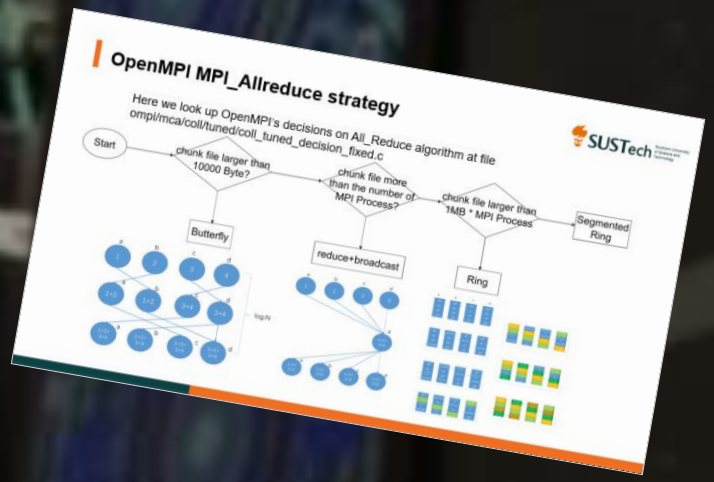
HOOMD-blue (Molecular Dynamics and hard particle Monte Carlo simulations)

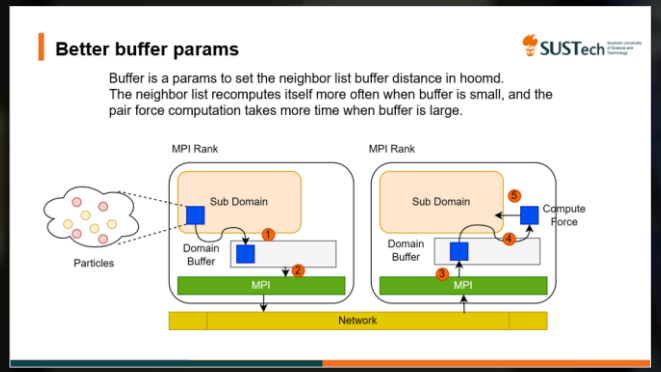
Contents

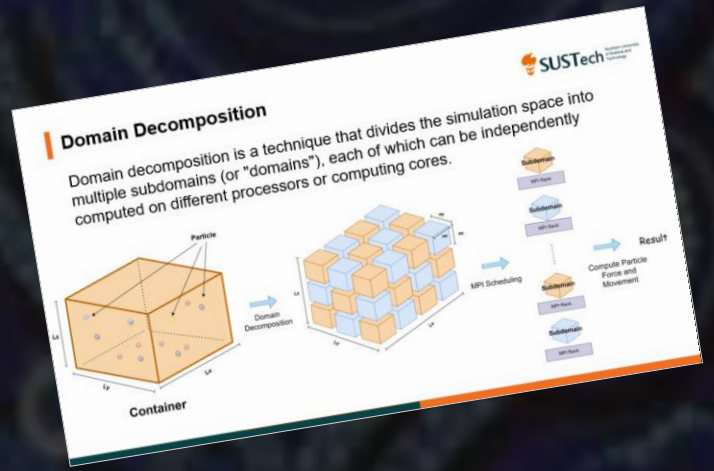

Hoomd Description
Hoomd Introduction
HOOMD-blue is a Python package that runs simulations of particle systems.
It performs hard particle Monte Carlo simulations of a variety of shape classes and molecular dynamics simulations of particles.
Task Description
Number of particles=200000
Molecular dynamics simulation with the WCA pair potential


HOOMD-blue Baseline
Before four nodes, the expansion effect is good
Software
Version
OpenMPI
4.1.5
Cereal
1.3.1
Eigen
3.4.0
Testing on QiMing 2.0
Testing on Gadi
Testing on NSCC
Number denotes to the number of nodes


IPM analysis
2 nodes -> 16nodes communication time raise but wall-time reduce!
Baseline on Gadi (legend)
node num.
wallclock
%comm
comm
calc
2 nodes
128.287s
15.01%
19.26s
109.03s
16 nodes
50.723s
55.83%
28.32s
22.40s
Baseline on NSCC
node num.
wallcloc k
%com m
comm
calc
2 nodes
102.752s
28.85%
29.64s
73.11s
16 nodes
81.155s
56.67%
45.99s
35.17s


OpenMPI MPI_Allreduce strategy
Here we look up OpenMPI's decisions on All_Reduce algorithm at file ompi/mca/coll/tuned/coll_tuned_decision_fixed.c
Start
chunk file larger than 10000 Byte?
chunk file more than the number of MPI Process?
chunk file larger than 1MB * MPI Process
Butterfly
reduce+broadcast
Ring
Segmented Ring

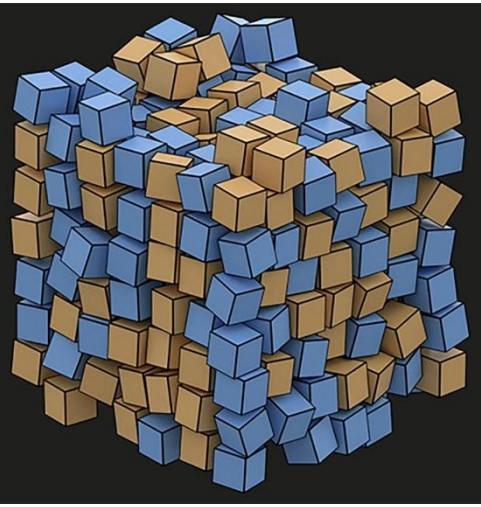

Better buffer params
Buffer is a params to set the neighbor list buffer distance in hoomd.
The neighbor list recomputes itself more often when buffer is small, and the pair force computation takes more time when buffer is large.
MPI Rank
MPI Rank
Sub Domain
Sub Domain
Compute Force
Particles
Domain Buffer
Domain Buffer
MPI
MPI
Network


Domain Decomposition
Domain decomposition is a technique that divides the simulation space into multiple subdomains (or "domains"), each of which can be independently computed on different processors or computing cores.
Particle
Container
Domain Decomposition
MPI Scheduling
Result
Compute Particle Force and Movement

Hoomd Description

SUSTech Southern University of Science and Technology

```python
from hoomd import *
from hoomd import hpmc
# place particles
context.initialize('--mode=cpu')
unitcell=lattice.sc(a=1.2, type_name='A')
system = init.create_lattice(unitcell, n=7)
# hard particle Monte Carlo
mc = hpmc.integrate.convex_polyhedron(
                    d=0.1, a=0.1, seed=2)
cube_verts = \
  [[-0.5,-0.5,-0.5], [0.5,-0.5,-0.5],
   [-0.5,-0.5, 0.5], [0.5,-0.5, 0.5],
   [-0.5, 0.5,-0.5], [0.5, 0.5,-0.5],
   [-0.5, 0.5, 0.5], [0.5, 0.5, 0.5]]
mc.shape_param.set('A',
                    vertices=cube_verts)
# run the simulation
run(50e3)
```



Hoomd Introduction

- HOOMD-blue is a Python package that runs simulations of particle systems.
- It performs hard particle Monte Carlo simulations of a variety of shape classes and molecular dynamics simulations of particles.

Listing 1 Molecular dynamics simulation

```python
from hoomd import *
from hoomd import md
# place particles
context.initialize()
unitcell=lattice.sc(a=2.0, type_name='A')
init.create_lattice(unitcell, n=10)
# define Lennard-Jones interactions
nl = md.nlist.cell()
lj = md.pair.lj(r_cut=2.5, nlist=nl)
lj.pair_coeff.set('A', 'A',
                  epsilon=1.0,sigma=1.0)
# NVT integration
all = group.all();
md.integrate.mode_standard(dt=0.005)
nvt = md.integrate.nvt(group=all, kT=1.2,
                       tau=1.0)
nvt.randomize_velocities(seed=1)
# run the simulation
run(10e3)
```

Listing 2 Hard particle Monte Carlo simulation

```python
from hoomd import *
from hoomd import hpmc
# place particles
context.initialize()
unitcell=lattice.sc(a=2.0, type_name='A')
init.create_lattice(unitcell, n=10)
# hard particle Monte Carlo
mc = hpmc.integrate.convex_polyhedron(
                    d=0.1, a=0.1, seed=2)
cube_verts = \
  [[-0.5,-0.5,-0.5], [0.5,-0.5,-0.5],
   [-0.5,-0.5, 0.5], [0.5,-0.5, 0.5],
   [-0.5, 0.5,-0.5], [0.5, 0.5,-0.5],
   [-0.5, 0.5, 0.5], [0.5, 0.5, 0.5]]
mc.shape_param.set('A',
                    vertices=cube_verts)
# run the simulation
run(10e3)
```

Task Description

- Number of particles=200000
- Molecular dynamics simulation with the WCA pair potential

CPU Task Workload And Benchmark

SUSTech Southern University of Science and Technology

Hoomd-Benchmark calculates WCA potential for N particles as workload.

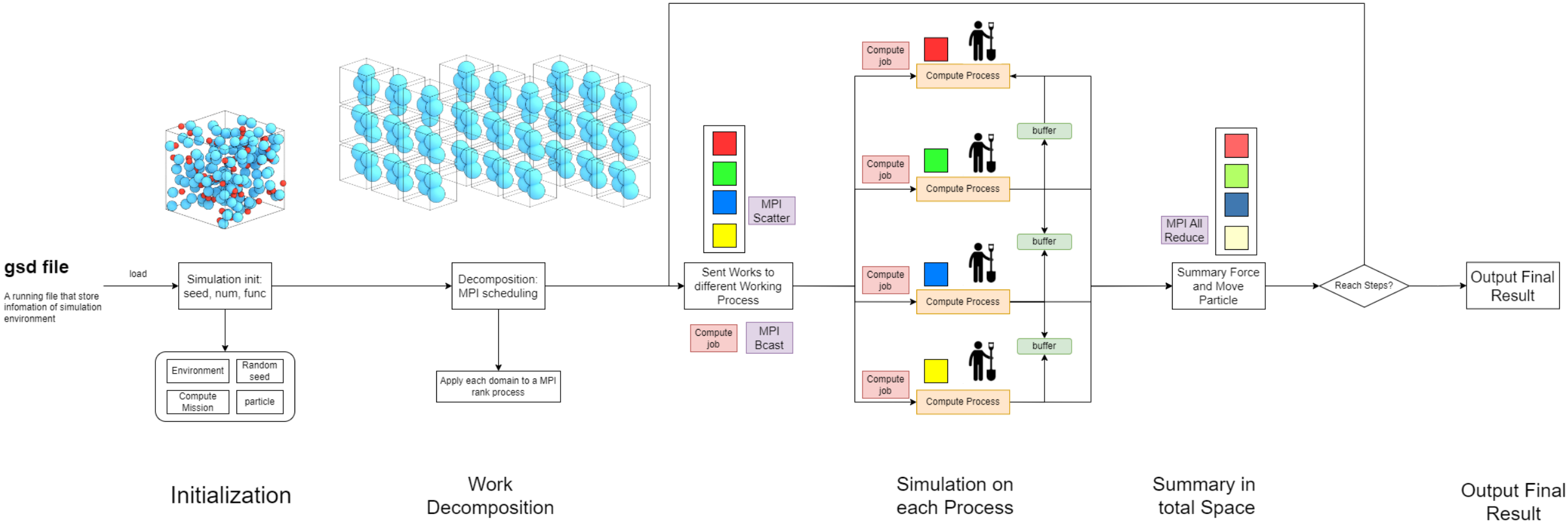
WCA potential（Weeks-Chandler-Andersen potential）

- A potential energy function used to simulate intermolecular interactions
- Simple and computationally efficient, suitable

$$U(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right]$$

System Architecture

SUSTech Southern University of Science and Technology



Test Cluster Configuration - QiMing 2.0 (CPU)

HW Info.	QiMing 2.0	
A CPU node	CPU	Intel Xeon E5-2690v3 (2.6GHz, 24 CPUs)
	Memory	64 GB

SW Info.	QiMing 2.0
Operating System	Rocky Linux 8.6
IB Drive	MLNX_OFED_LINUX-5.9-0.5.6.0
Compilers	Intel Compiler 2022
MPI Libraries	Intel MPI 2021

Application Info.	
Application	HOOMD-blue
Required compilers	Compatible C and Fortran compilers
Required MPI installation	Intel MPI, OpenMPI, HPC-X
Required Libraries	Cereal, Eigen, Numpy， Pybind11，GSD， Pandas， Rowan

Test Cluster Configuration - Gadi (CPU)

SUSTech Southern University of Science and Technology

HW Info.	Gadi	
A CPU node normal queue	CPU	Intel(R) Xeon(R) Platinum 8274 (3.2 GHz 24 CPUs per node)
	Memory	192 GB

SW Info.	Gadi
Operating System	Rocky Linux 8.8
IB Drive	OFED-internal-5.7-1.0.2
Compilers	Intel Compiler 2021
MPI Libraries	Intel MPI 2021, Open MPI 4.1.5

Application Info.	
Application	HOOMD-blue
Required compilers	Compatible C and Fortran compilers
Required MPI installation	Intel MPI, OpenMPI, HPC-X
Required Libraries	Cereal, Eigen, Numpy, Pybind11, GSD, Pandas, Rowan

Test Cluster Configuration - NSCC (GPU)

SUSTech Southern University of Science and Technology

HW Info.	NSCC	
	CPU	Dual-CPU AMD EPYC 7713, 128 cores per server.
A GPU node R4 queue	GPU	4x Nvidia A100(40G)
	Memory	512 GB

SW Info.	NSCC
Operating System	Red Hat Enterprise Linux-8
IB Bandwidth	100Gbi
IB Drive	OFED-internal-5.7-1.0.2
MPI Libraries	openmpi/4.1.2-hpe

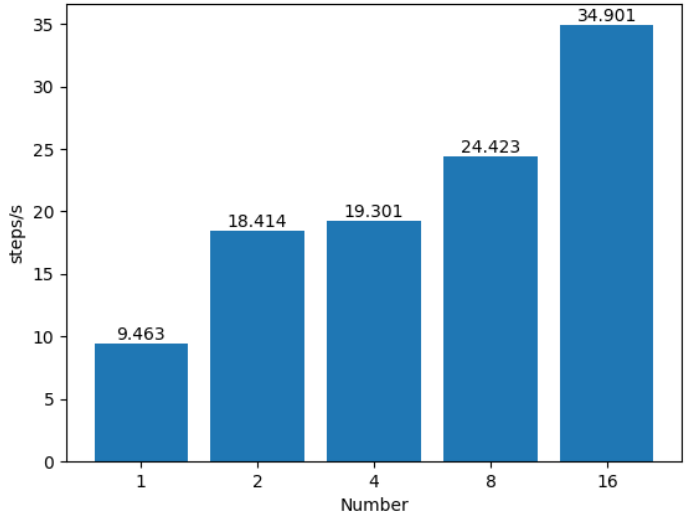
Application Info.	
Application	Litgpt-Llama2-sft
Required MPI installation	openmpi/4.1.2-hpe
Required Libraries	litgpt=0.5.2 lightning=2.4.0 torch=2.4.1 transformers=4.44.2

HOOMD-blue Baseline

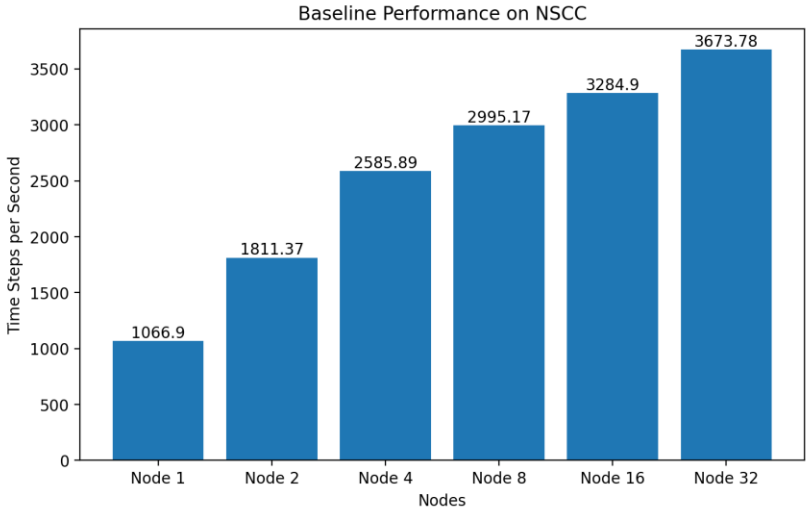
Before four nodes, the expansion effect is good

Software	Version
OpenMPI	4.1.5
Cereal	1.3.1
Eigen	3.4.0

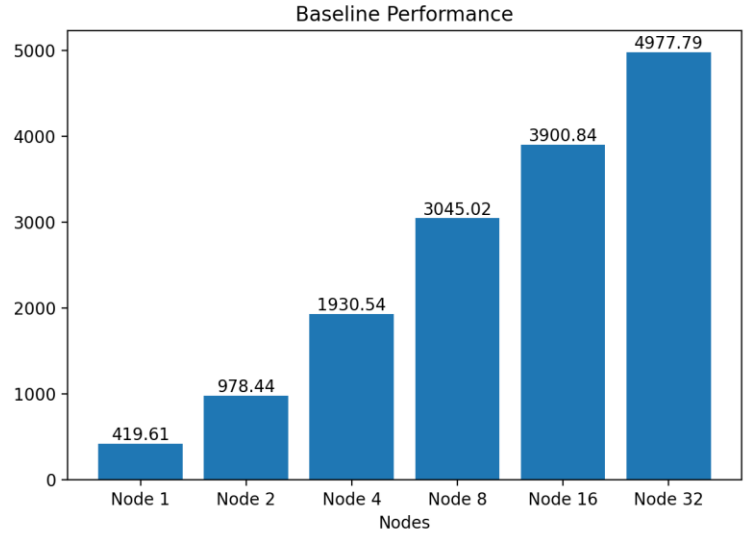
Testing on QiMing 2.0



Testing on Gadi



Testing on NSCC



Number denotes to the number of nodes

IPM analysis

2 nodes -> 16nodes communication time raise but wall-time reduce!

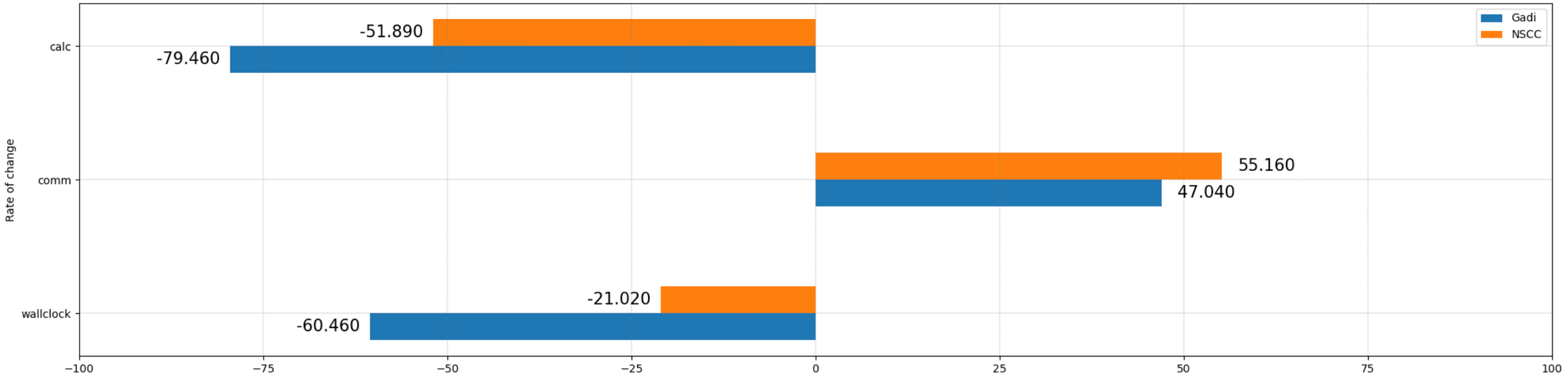
SUSTech Southern University of Science and Technology

Baseline on Gadi (legend)

node num.	wallclock	%comm	comm	calc
2 nodes	128.287s	15.01%	19.26s	109.03s
16 nodes	50.723s	55.83%	28.32s	22.40s

Baseline on NSCC

node num.	wallclock	%comm	comm	calc
2 nodes	102.752s	28.85%	29.64s	73.11s
16 nodes	81.155s	56.67%	45.99s	35.17s

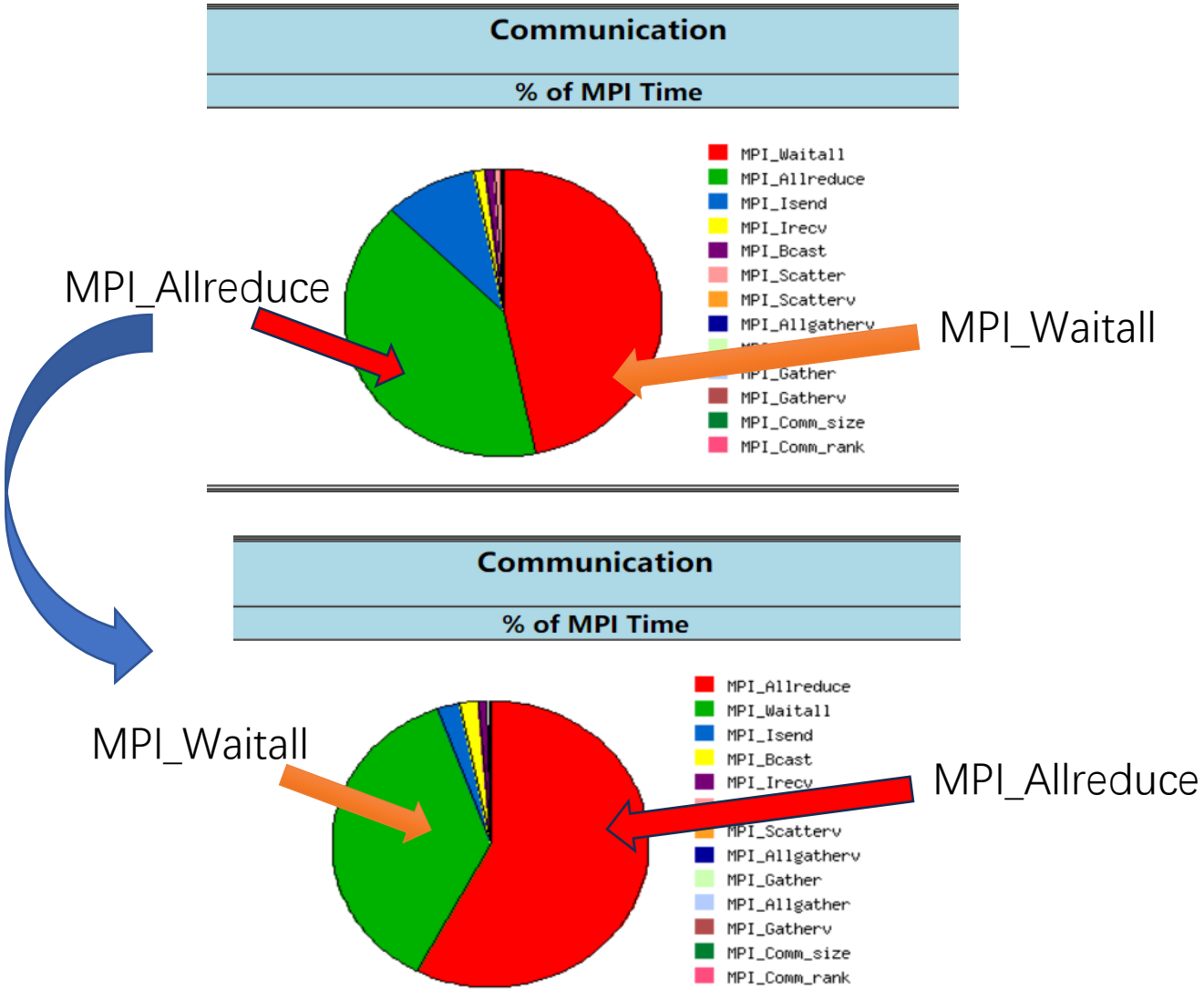


IPM analysis

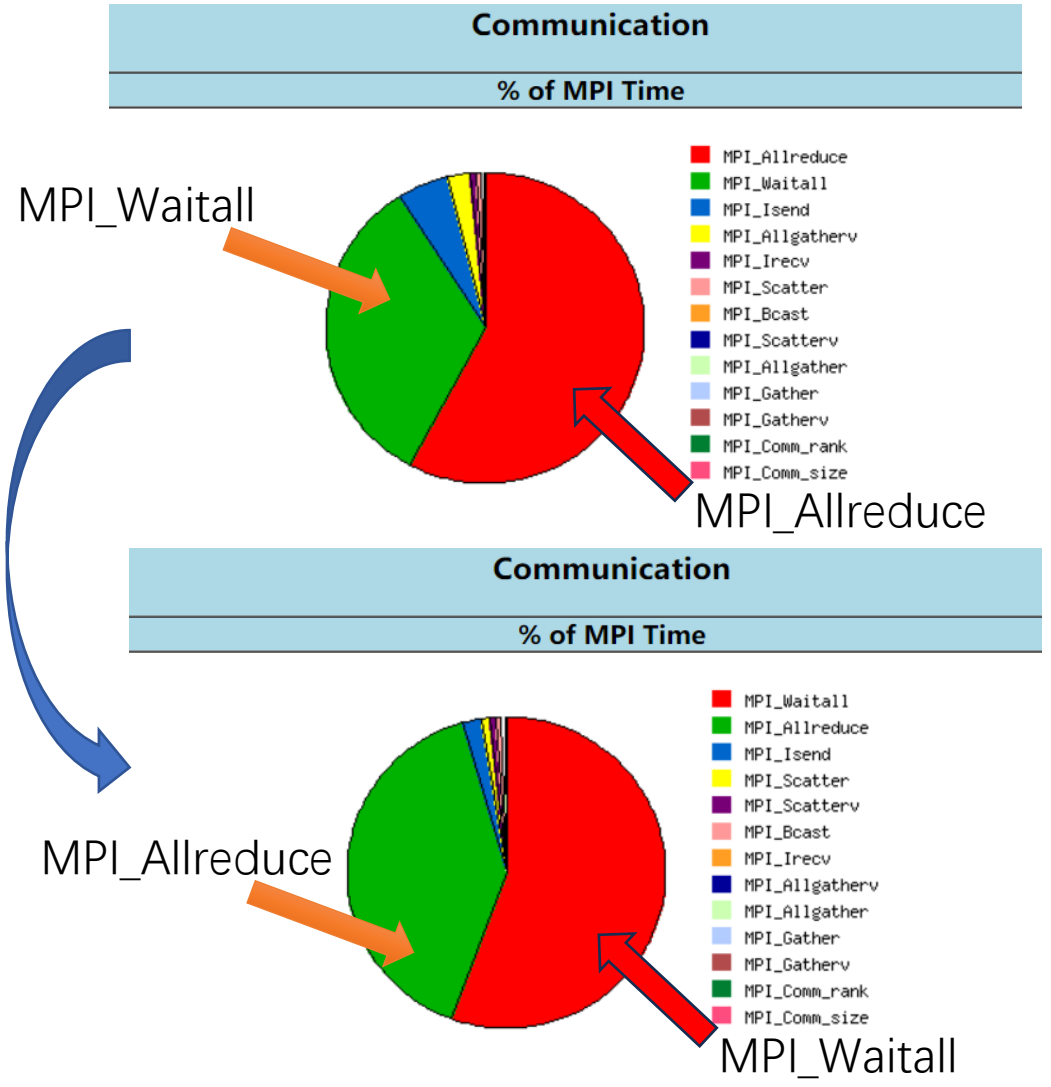
2 nodes -> 16nodes different clusters get different communication properties

SUSTech Southern University of Science and Technology

Baseline on Gadi (legend)

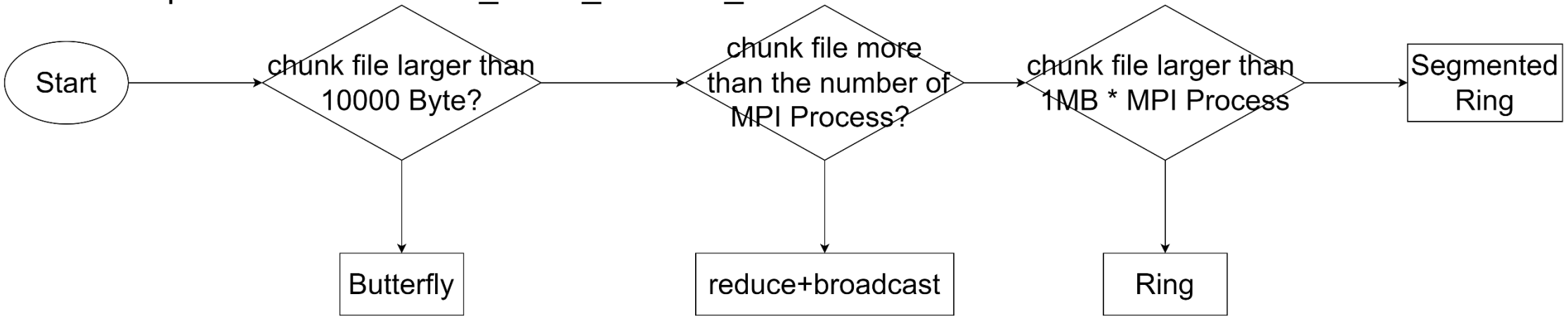


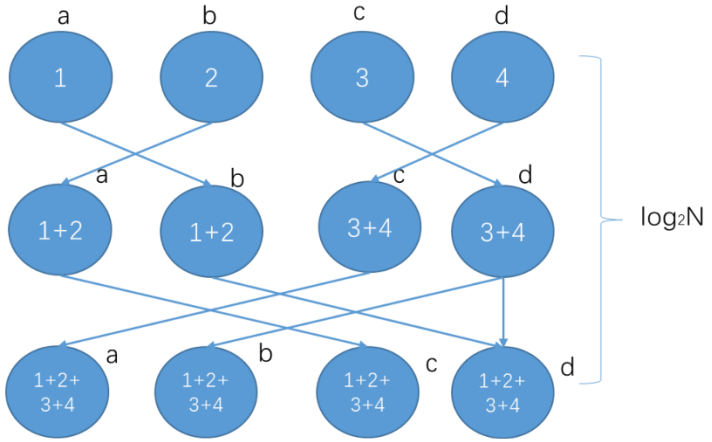
Baseline on NSCC

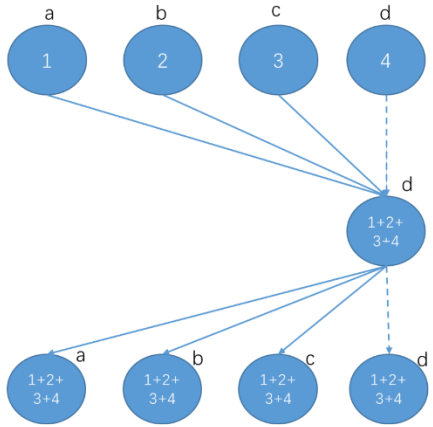


OpenMPI MPI_Allreduce strategy

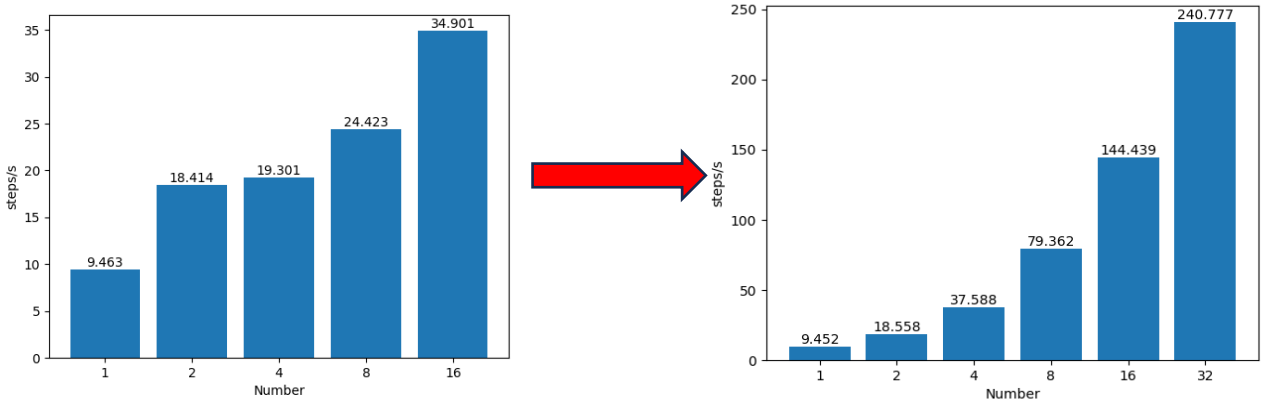
Here we look up OpenMPI's decisions on All_Reduce algorithm at file ompi/mca/coll/tuned/coll_tuned_decision_fixed.c







Using HPC-X on QiMing 2.0



34.901

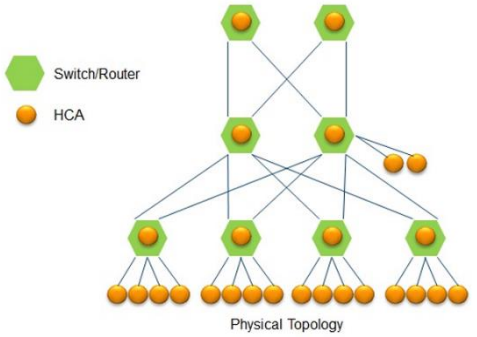
144.439

By changing the communications library to OpenSHMEM programming library, we get **413.8% improvement in 16 nodes!**

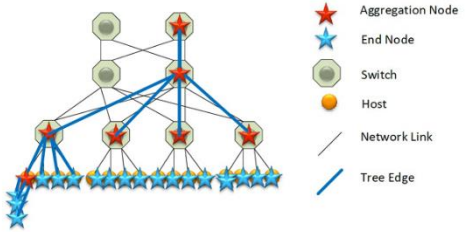
After initiating hcoll, we find all parameters of All_Reduce from hcoll_info.

Here we try to find the best params from 32 nodes with each node 24 nodes:

- --mca coll_hcoll_enable 1
- --x HCOLL_ENABLE_SHARP=1 (Correlated to Hardware Switch)
- --x HCOLL_ALLREDUCE_LB_SUPPORT=0
- --x HCOLL_BCOL_P2P_LARGE_ALLREDUCE_ALG=1
- --x HCOLL_BCOL_UCX_P2P_HYBRID_SRA_NODE_RADIX=3
- - -x HCOLL_BCOL_UCX_P2P_HYBRID_SRA_NET_RADIX=2



(a) Physical Network Topology

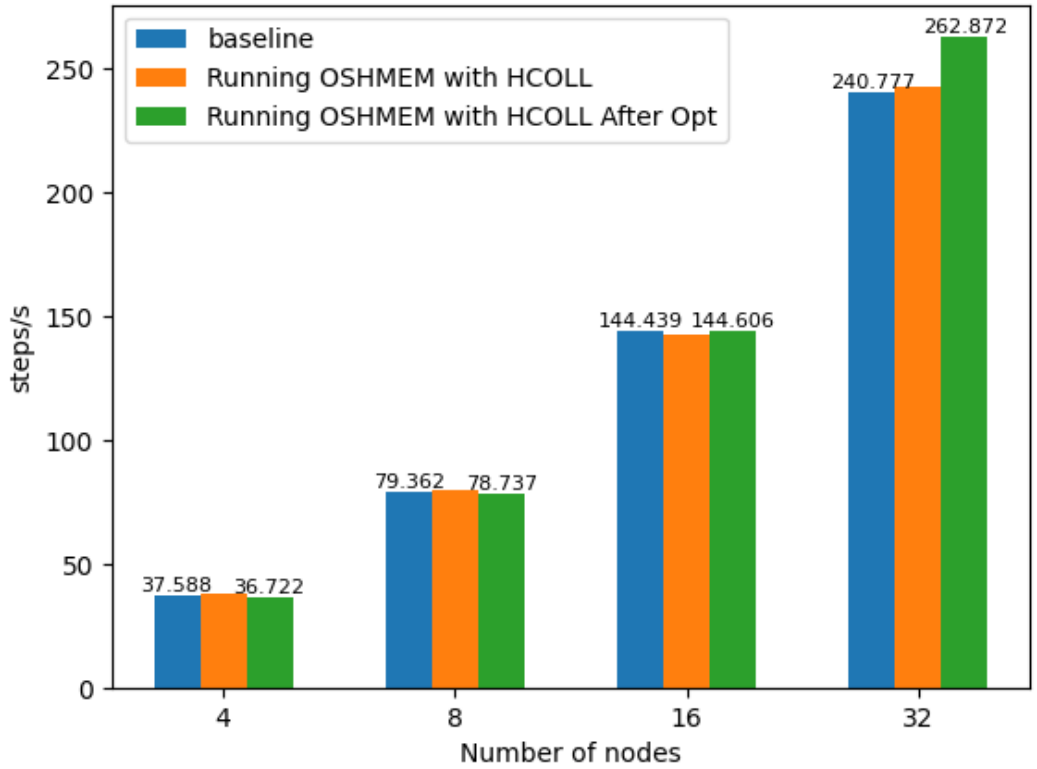


(b) Logical SHArP Tree. Note that in the SHArP abstraction an Aggregation Node may be hosted by an end-node.

Fig. 1: SHArP Tree example.

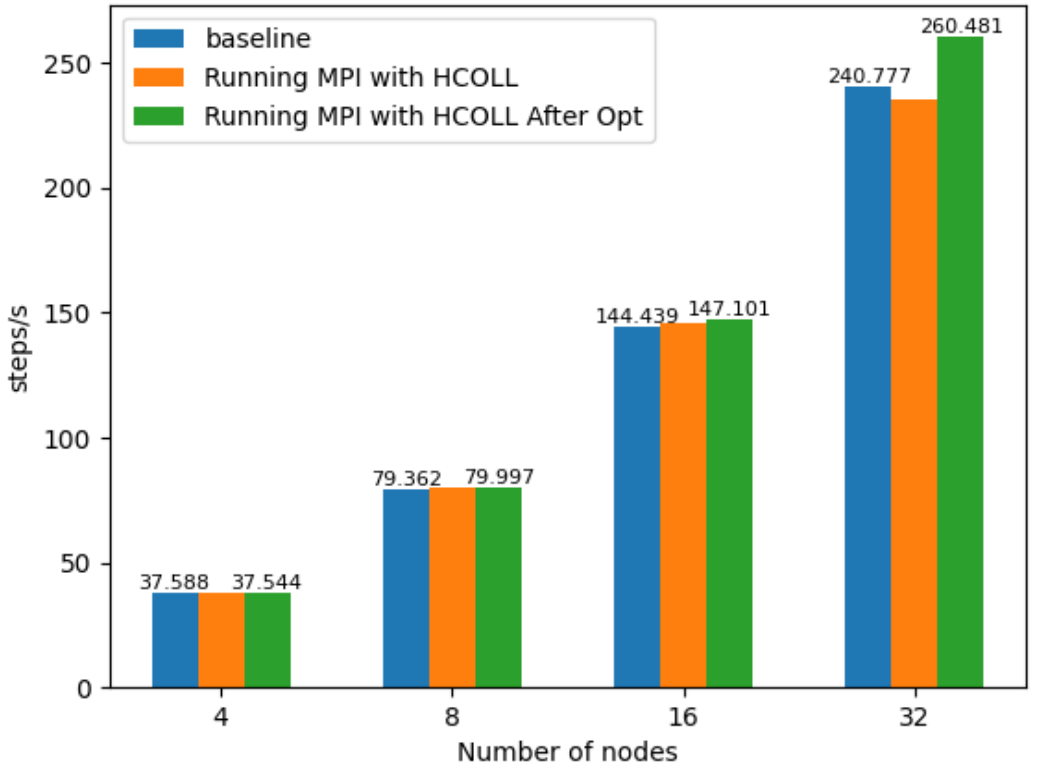
Using HPC-X on QiMing 2.0 with Optimization value

SUSTech Southern University of Science and Technology



Running OSHMEM with HCOLL

9.17% improvement for 32 nodes

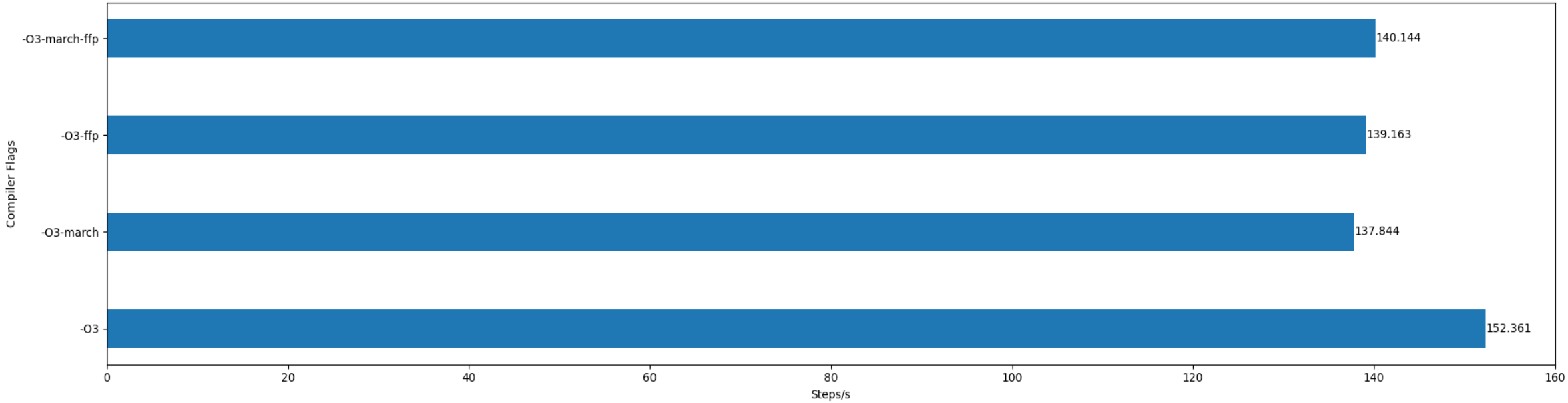


Running MPI with HCOLL

8.18% improvement for 32 nodes

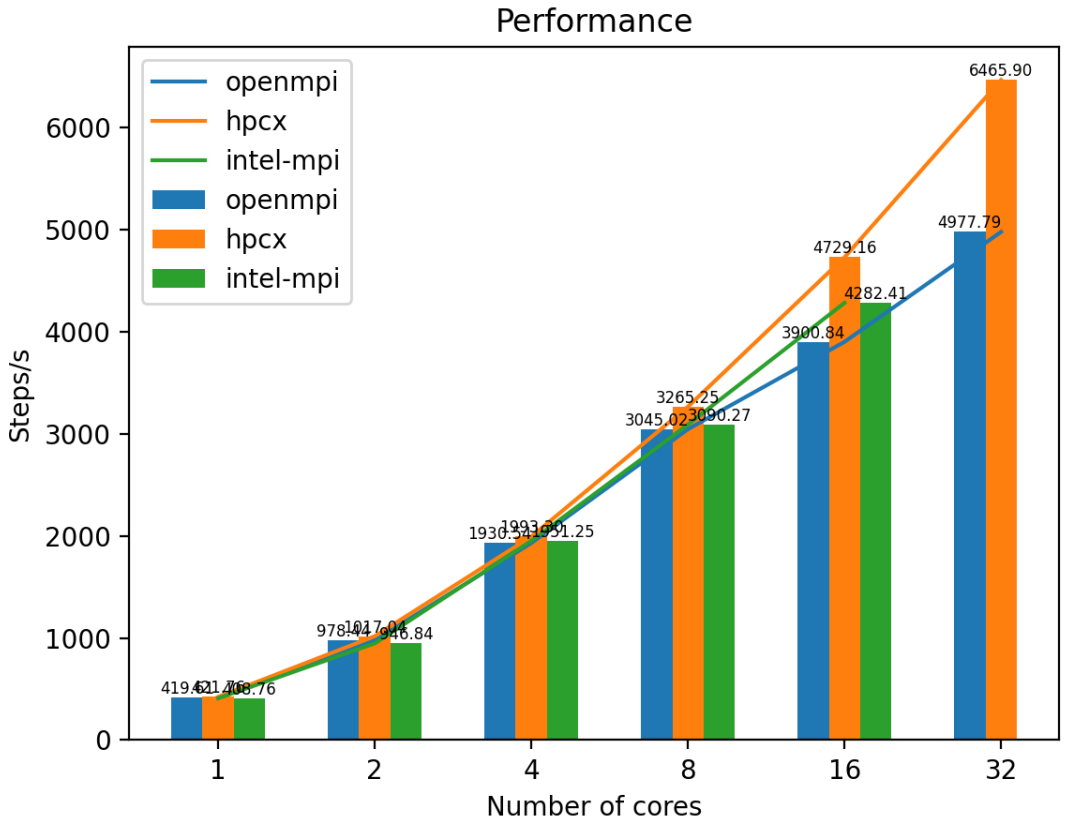
Compiler Flags Optimization(QIMING)

Change different compiler flags can get 5% improvement
we test performance on 16 nodes using HPC-X



Baseline Flags	Optimized Flags	Explain
Default	-O3	Optimization with option of O3, generate highest instructionenable additional interprocedural optimizations for single-filecompilation and generate fuse multiply-add instructions

Using HPC-X on Gadi



29.89% improvement for 32 nodes

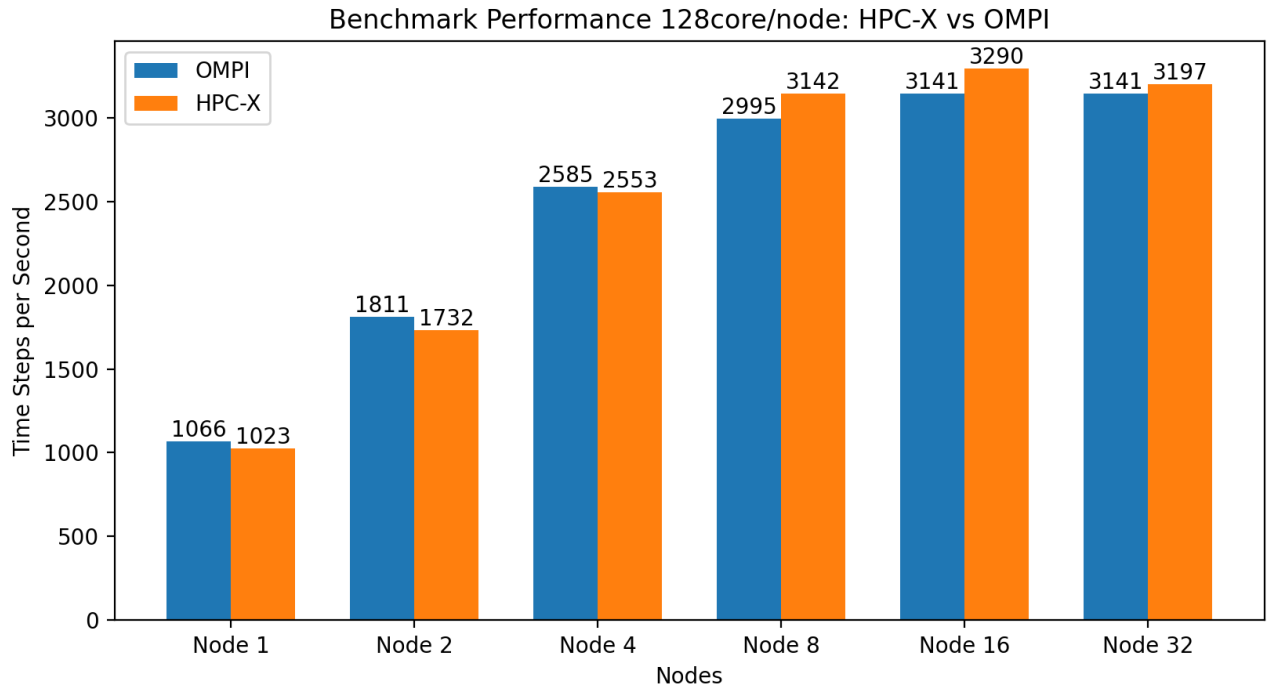
The performance comparison between HPC-X , IntelMPI and OpenMPI shows that HPC-X delivers superior performance and greater scalability with the same number of cores.

And we hope rebuilding OpenMPI with HPC-X can help us make a better decision on improving the Collective Communication Algorithm.

Using HPC-X on NSCC

SUSTech Southern University of Science and Technology

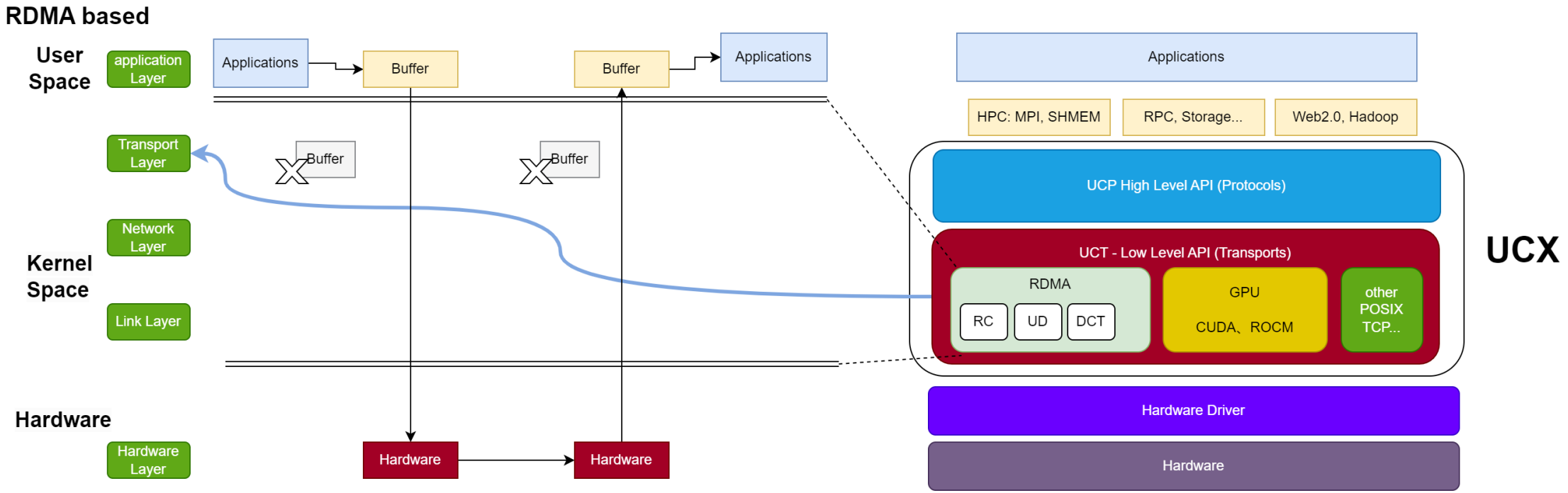
HPC-X does not get significant improvement in NSCC



1.78% improvement for 32 nodes

Gadi optimal ucx params

UCX_TLS refers to the "Transport Layer Selection" feature in the Unified Communication X (UCX) framework. It allows users to specify which transport protocols UCX should use for communication in a parallel or distributed computing environment.

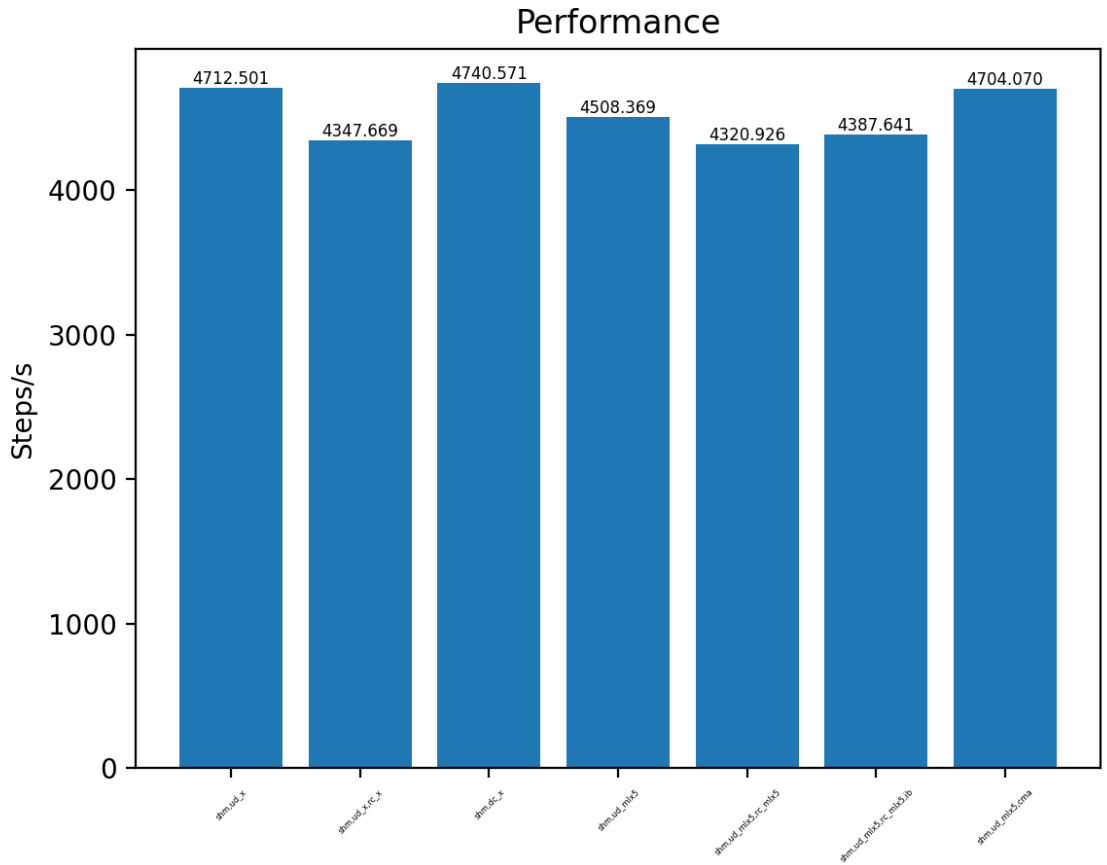


Gadi optimal ucx params

We test multi combination of UCX_TLS to get better performance.

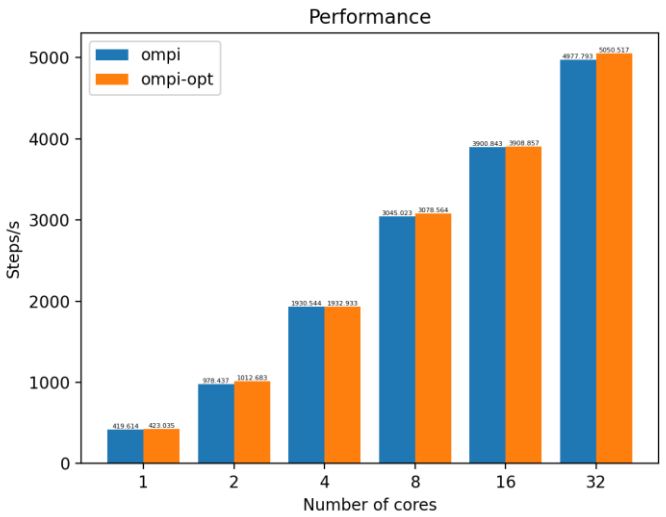
Params	description
all	use all the available transports
sm	all shared memory transports
rc	reliable connection
ud	unreliable datagram transport
dc	Mellanox scalable offloaded dynamic con_x0002_nection transport
...	...

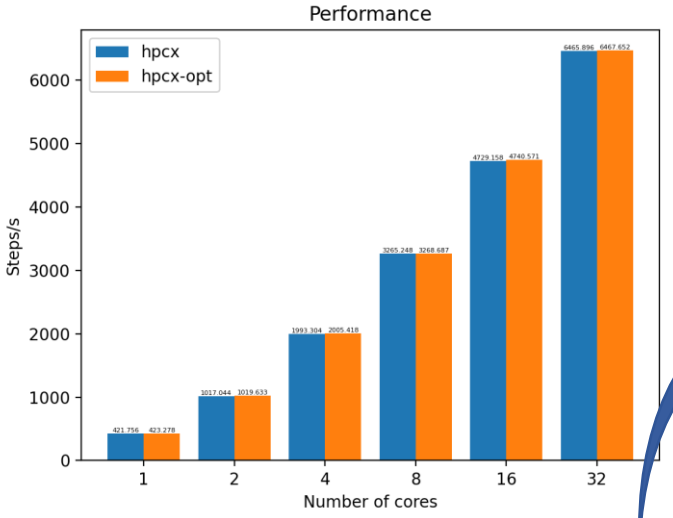
Testing UCX_TLS in bottleneck scale 16nodes



Use UCX_TLX params with hpcx on gadi

Best ucx params in ompi and hpcx(gadi)





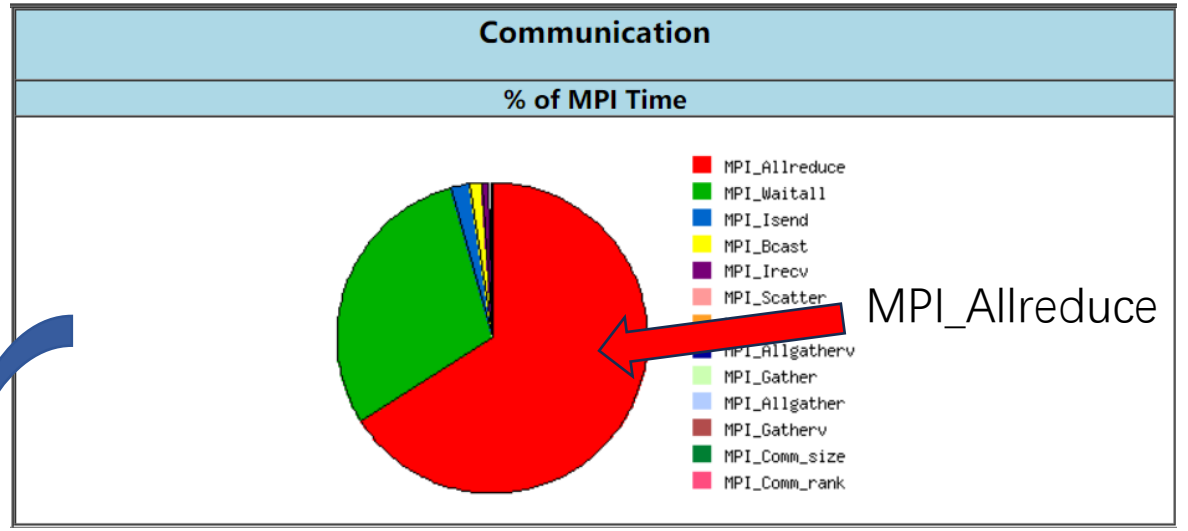
1.46% improvement for 32 nodes

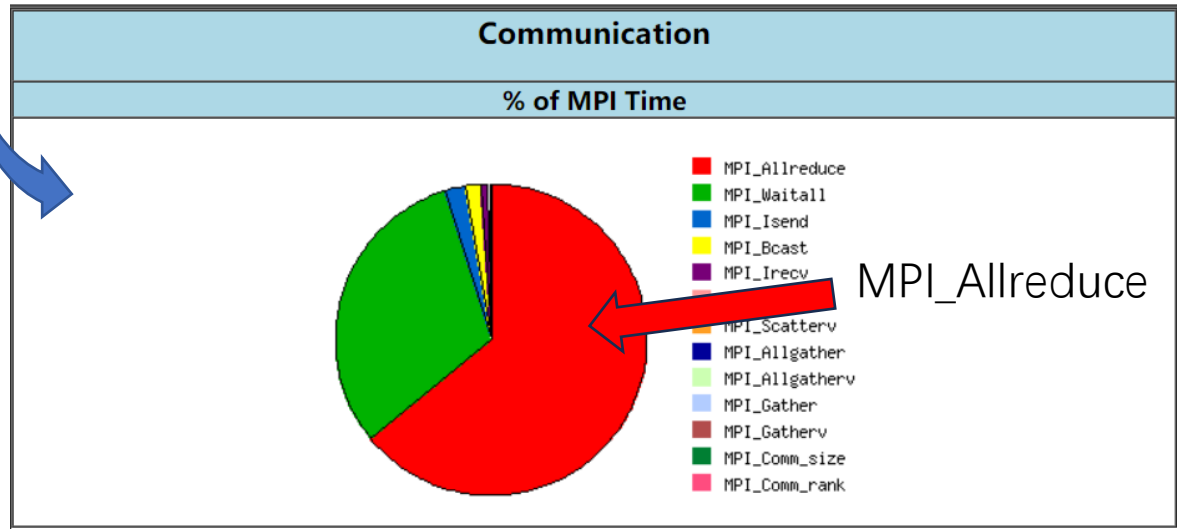
0.27% improvement for 32 nodes

shm, dc_x does not have much improvements in both ompi and hpcx

IPM analysis

Ucx params doesn't change much



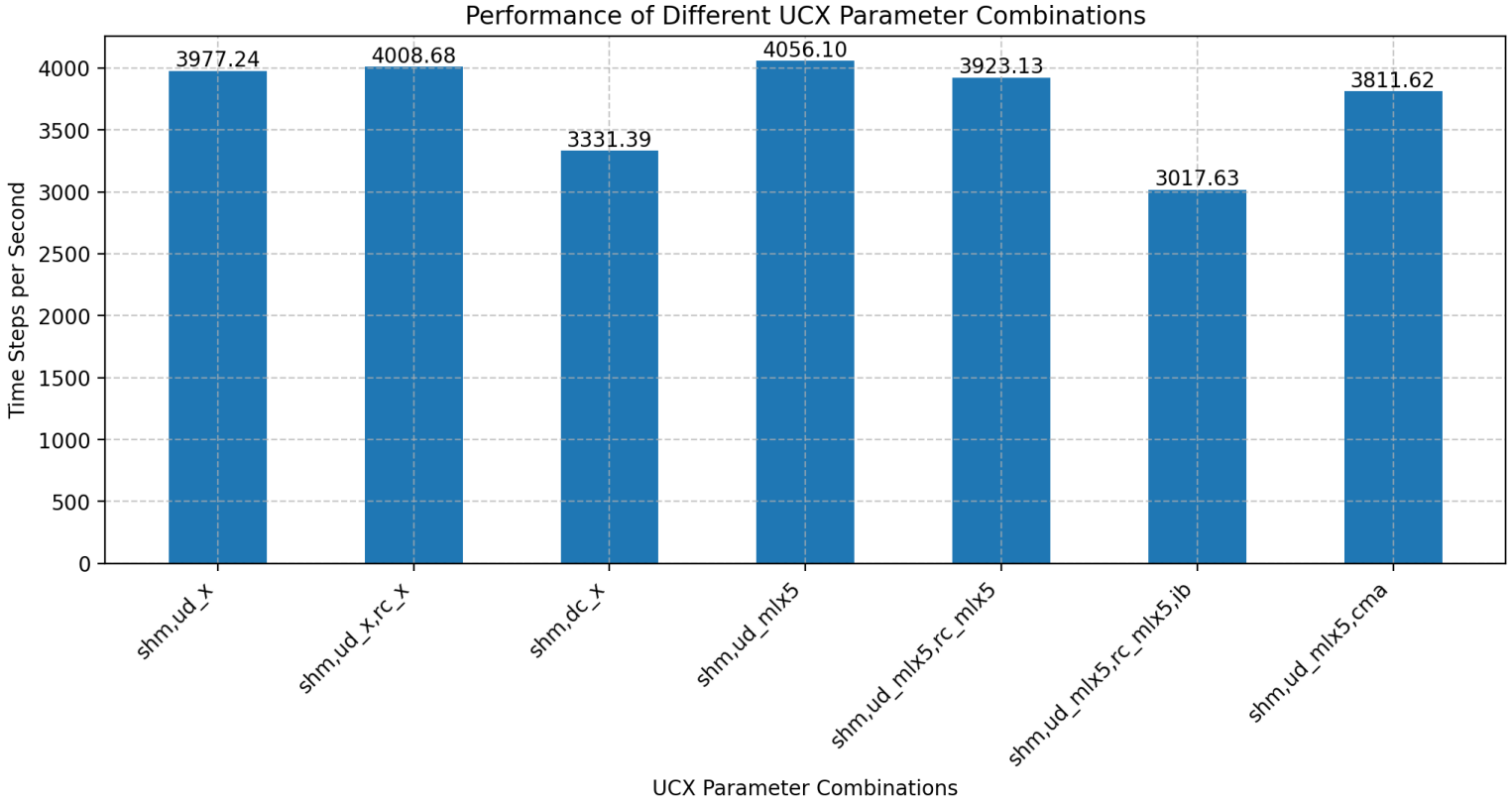


SUSTech Southern University of Science and Technology

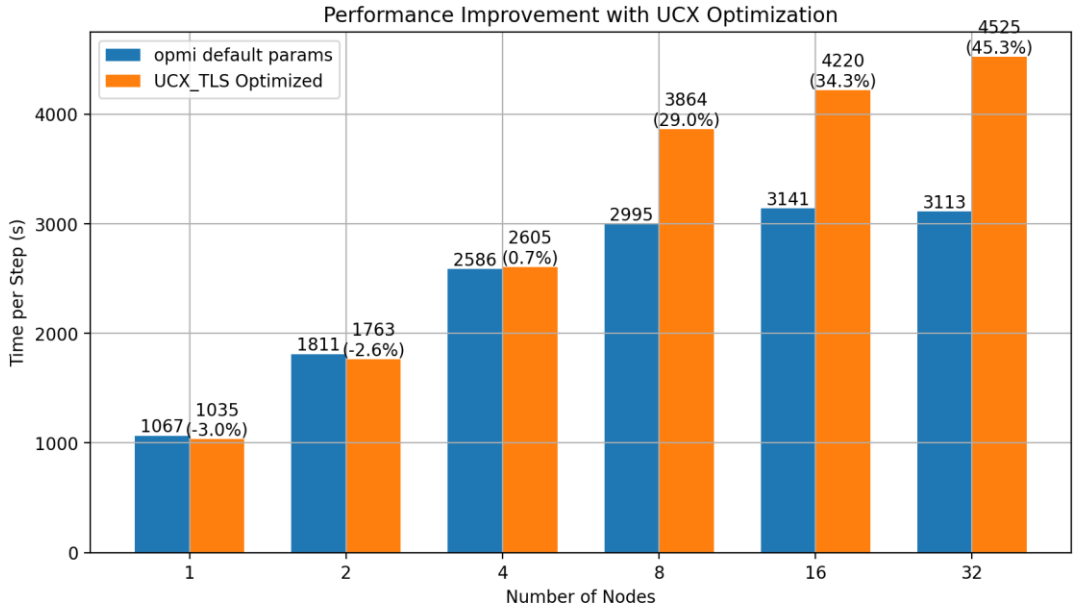
NSCC optimal ucx params

We test multi combination of UCX_TLS to get better performance.

Params	description
all	use all the available transports
sm	all shared memory transports
rc	reliable connection
ud	unreliable datagram transport
dc	Mellanox scalable offloaded dynamic con_x0002_nection transport
...	...



Testing UCX_TLS in bottleneck scale 16nodes



Best ucx params in ompi

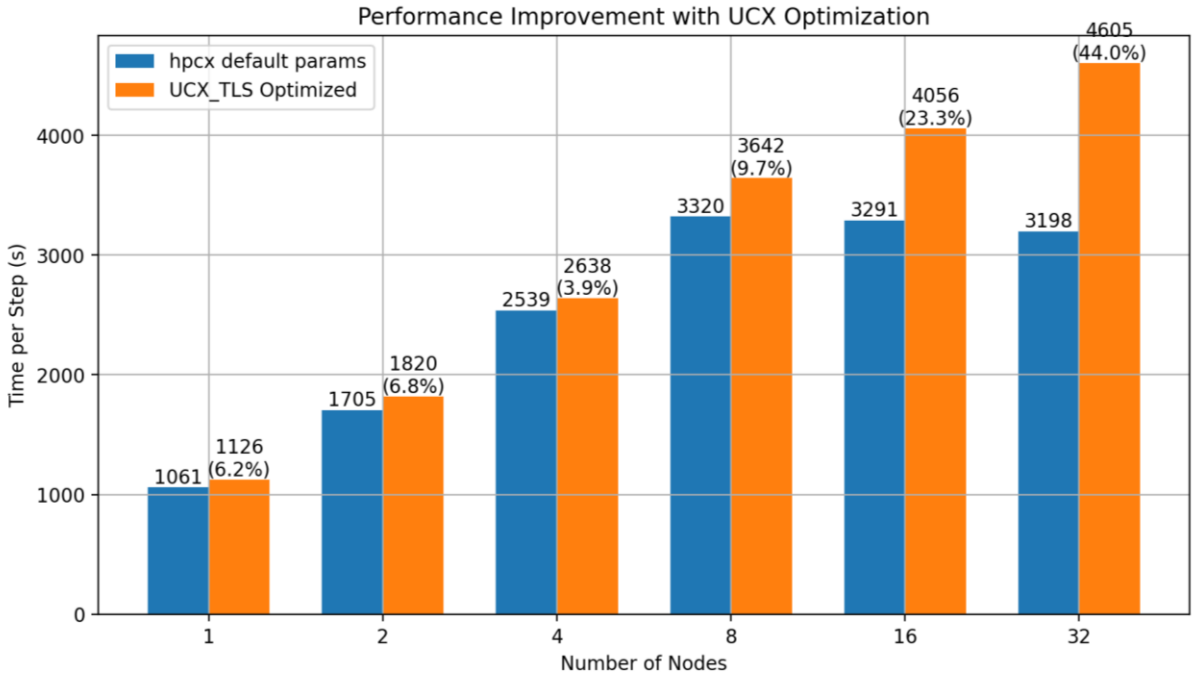
shm, ud_mlx5 get up to 45% improvement in ompi

45.3% improvement for 32 nodes

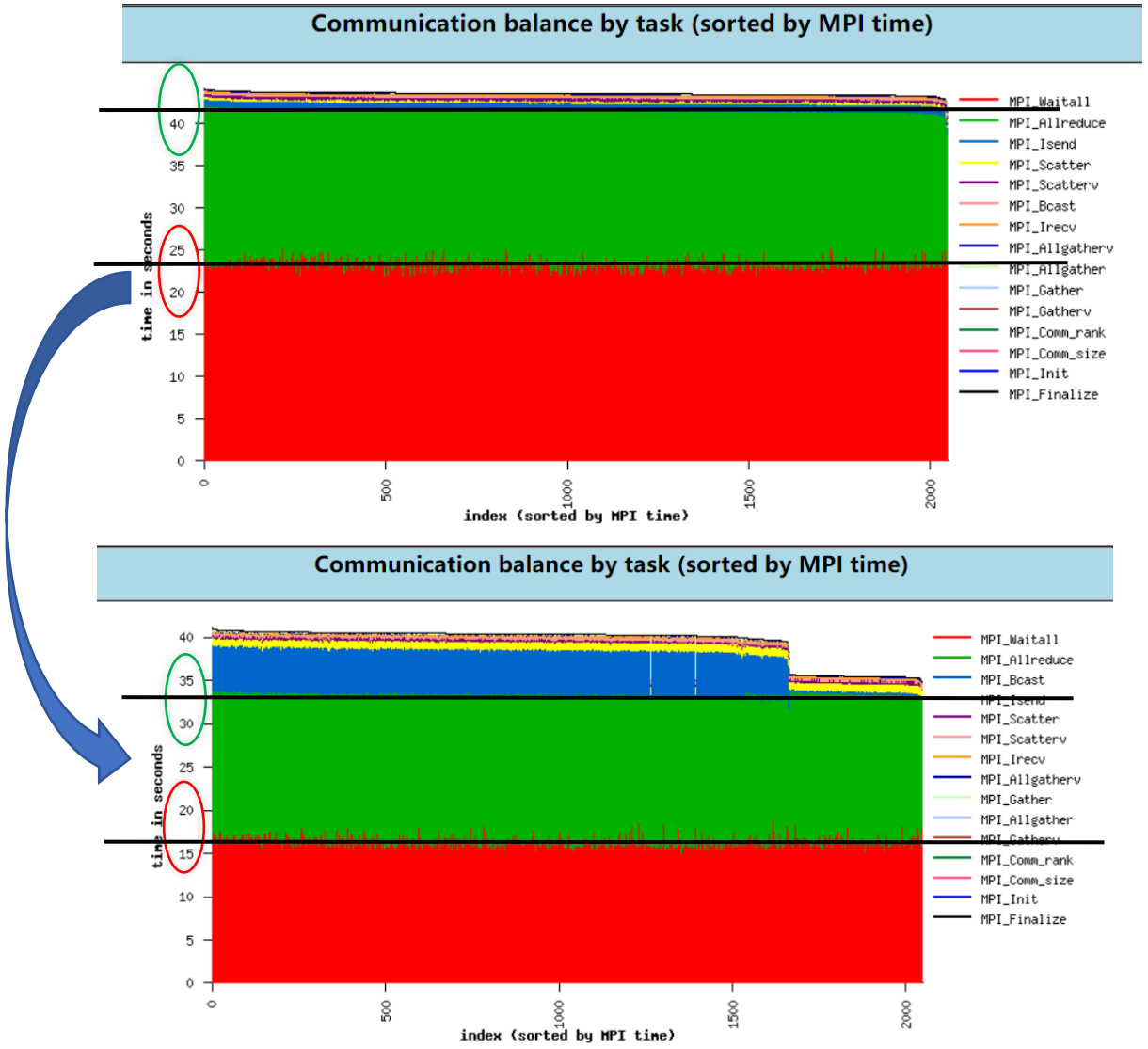
Best ucx params in hpcx

shm, ud_mlx5 get up to 44% improvement in hpcx

44.0% improvement for 32 nodes



UCX_TLS improvement analysis



IPM analysis

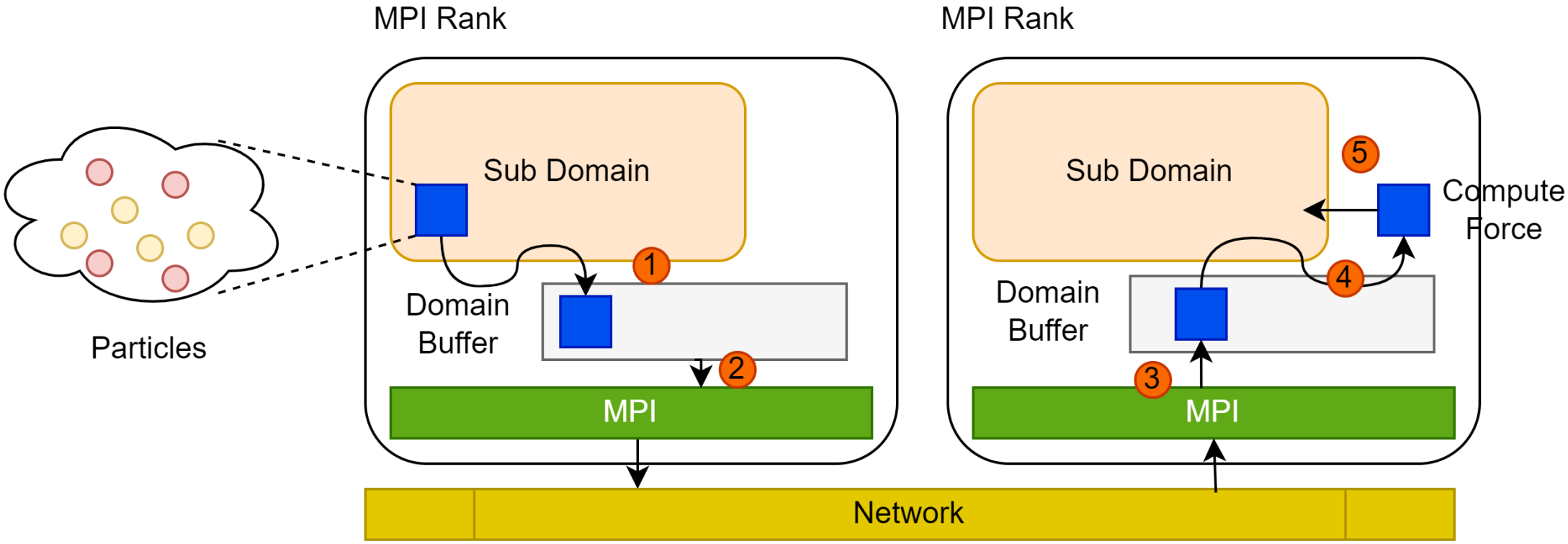
From defualt → optimal ucx params we get improvement in communication bottleneck

Waitall: ≈23 → ≈16

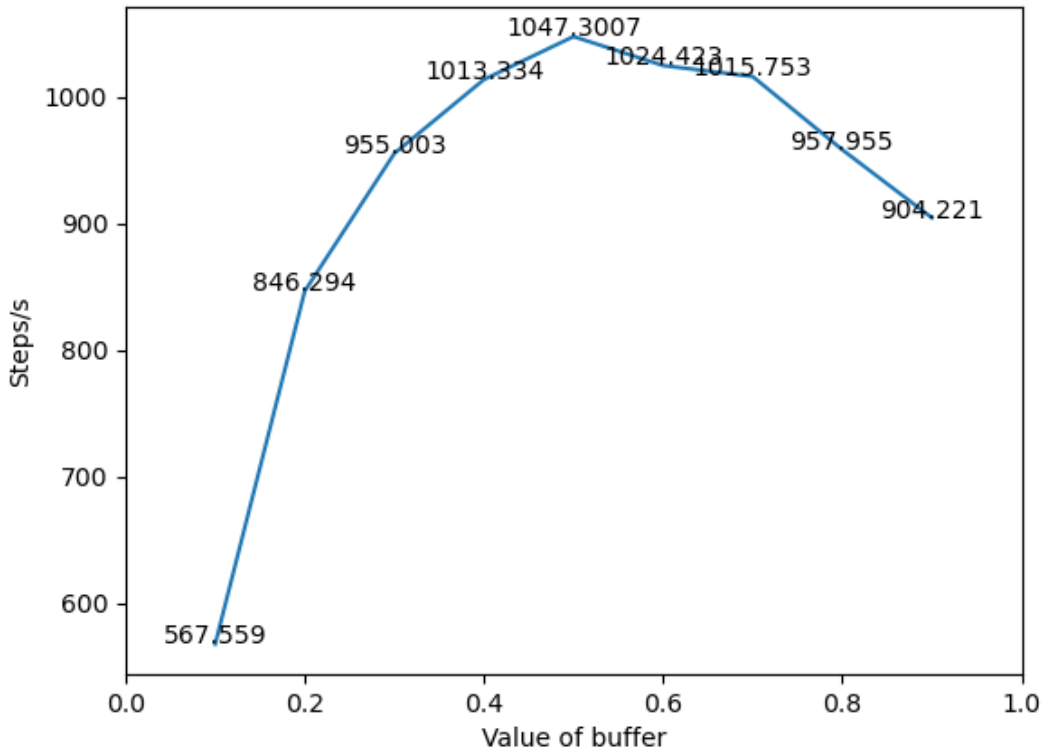
Allreduce: >40 → ≈33

Better buffer params

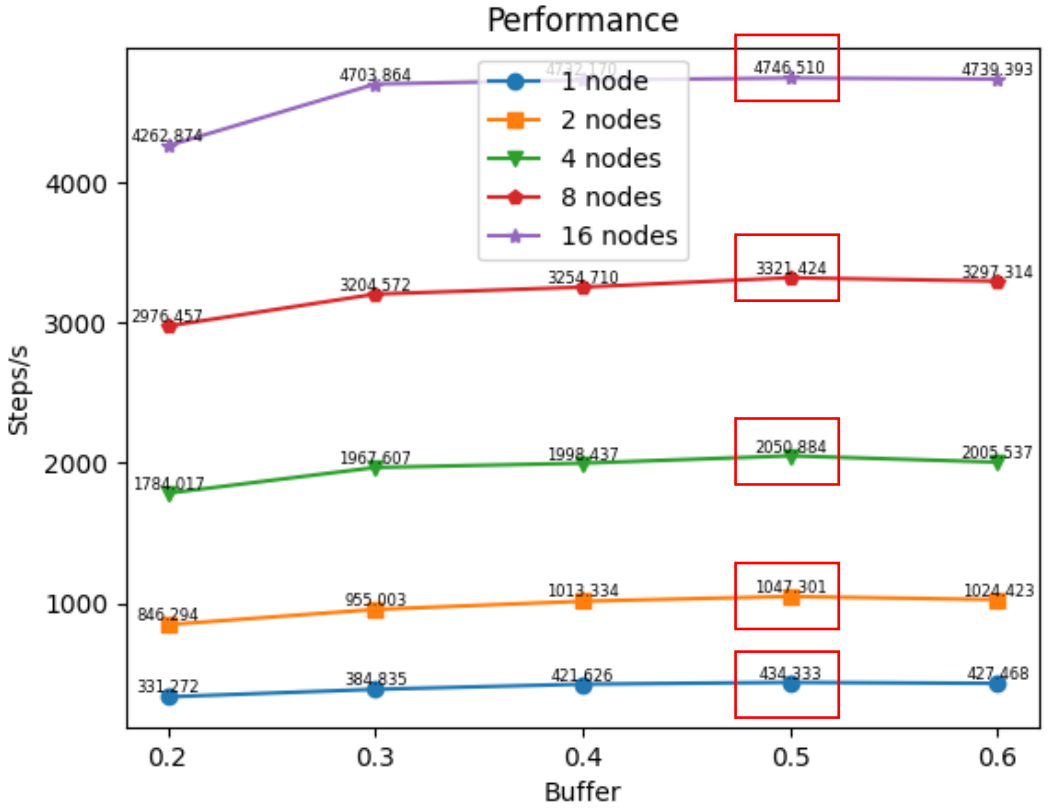
Buffer is a params to set the neighbor list buffer distance in hoomd.
The neighbor list recomputes itself more often when buffer is small, and the pair force computation takes more time when buffer is large.



Better buffer params(gadi)

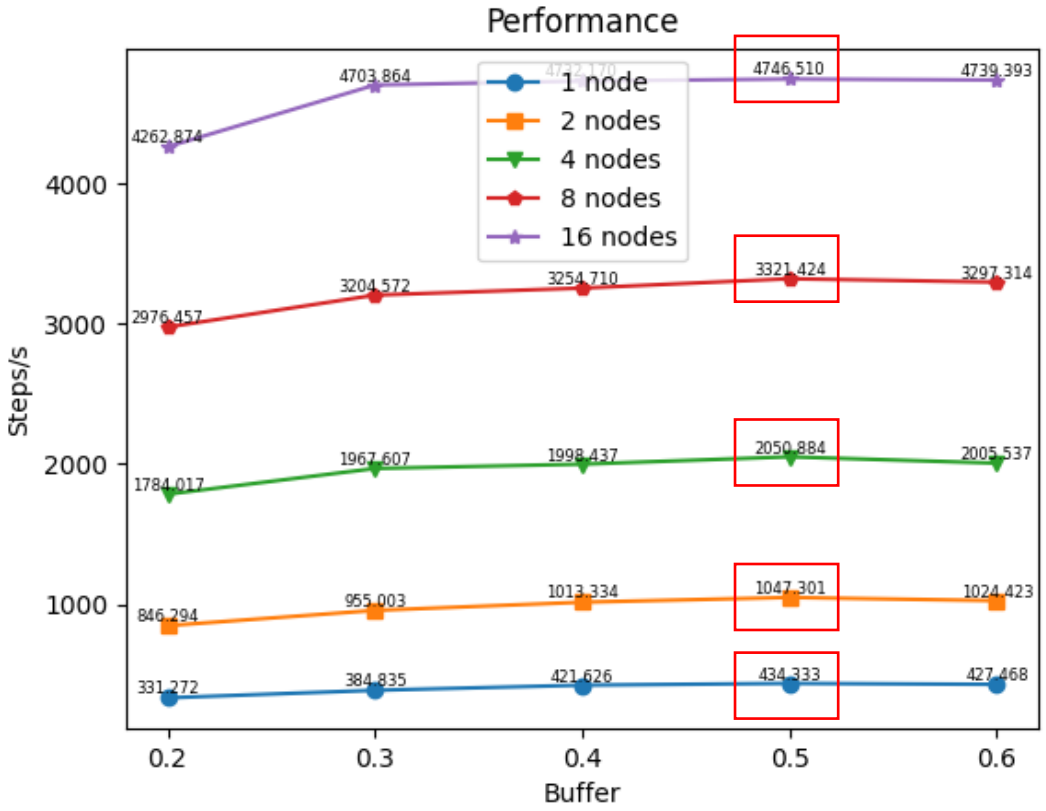


Testing buffer in bottleneck scale 2 nodes

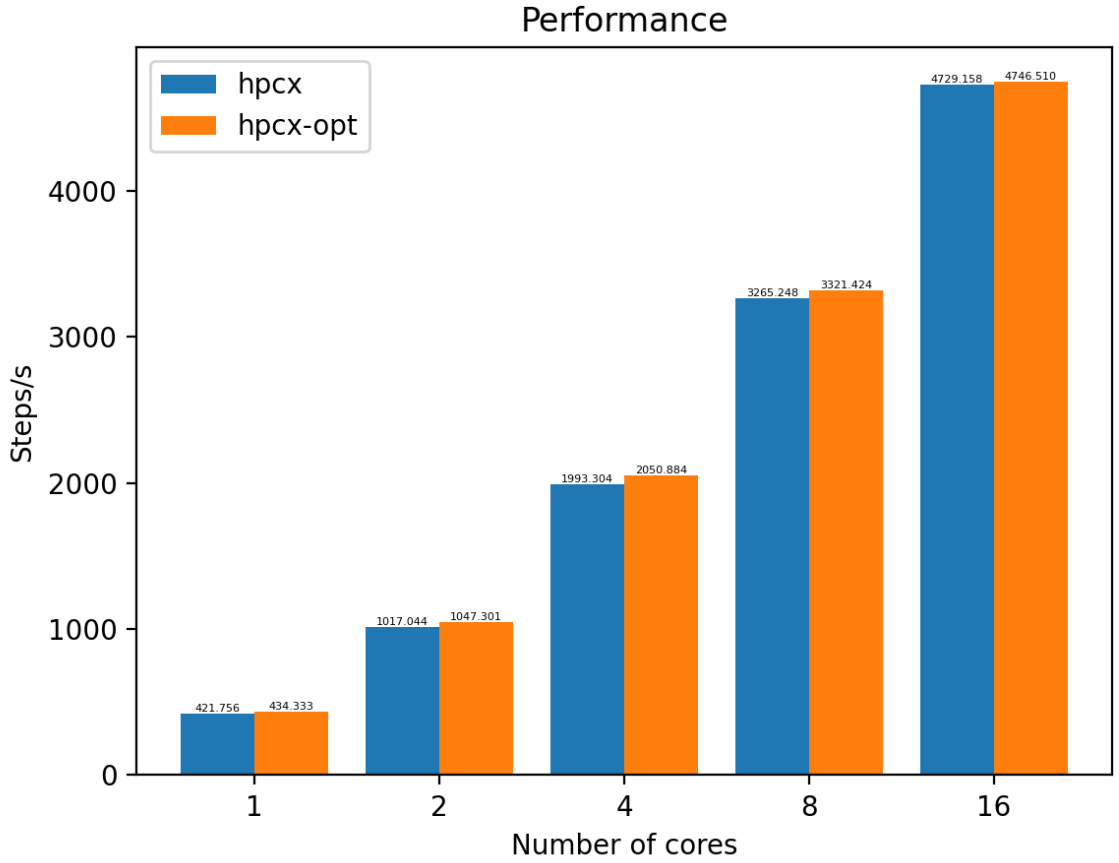


For more tests according to buffer

Best buffer params in hpcx



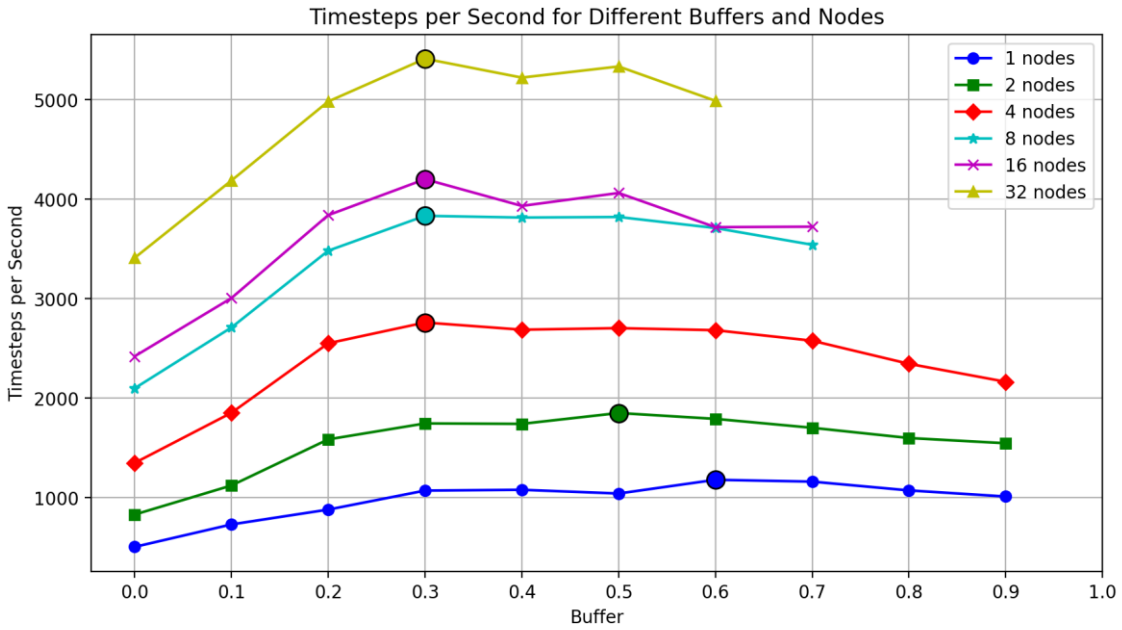
For more tests according to buffer



0.37% improvement for 32 nodes

buffer params(NSCC)

buffer is a key params in domain composition parallel algorithm



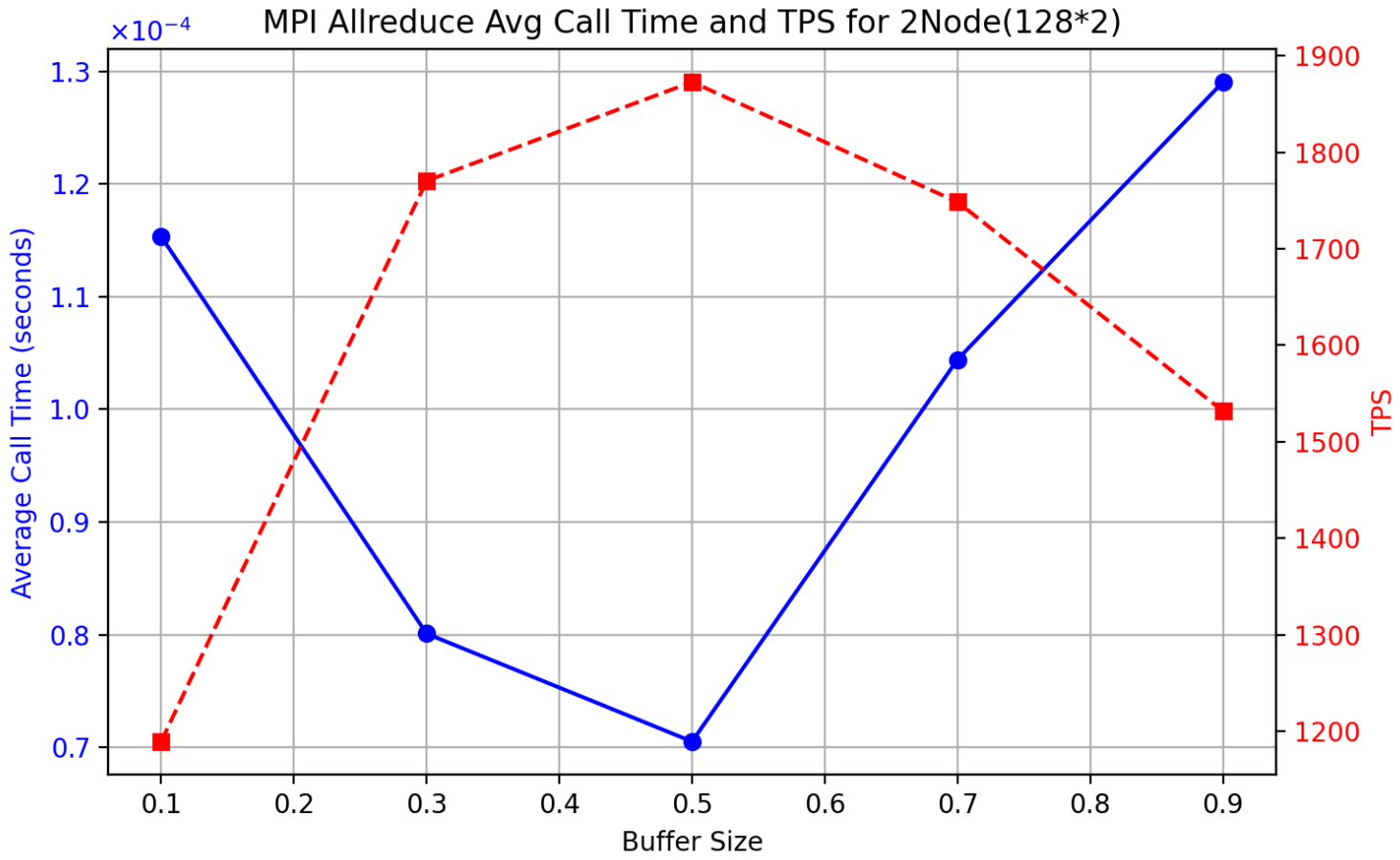
N-list updated are infected by buffer params

```
1  if (needsUpdating(timestep))
2      {
3      bool overflowed = false;
4      do
5          {
6          buildNlist(timestep);
7          overflowed = checkConditions();
8          if (overflowed)
9              ...
10         } while (overflowed);
11     ..
12 }
```

4.86% improvement for 32 nodes

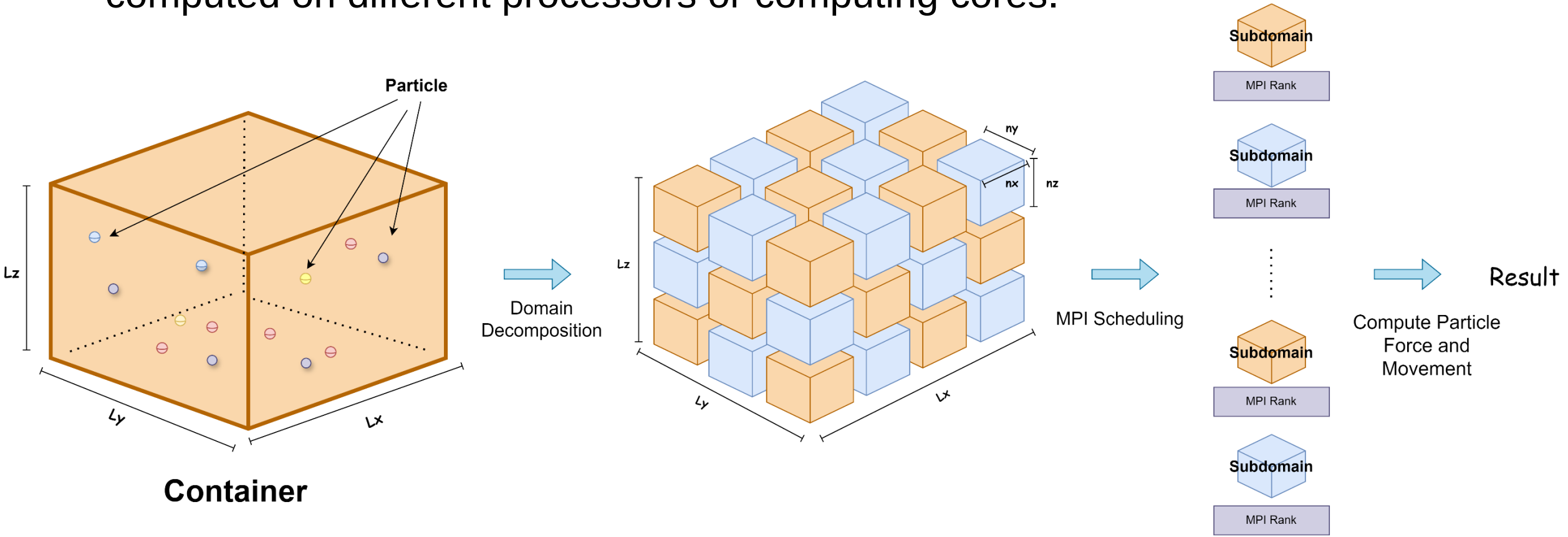
ipm analysis

buffer params strike a balance between rebuild nlist and compute load



Domain Decomposition

Domain decomposition is a technique that divides the simulation space into multiple subdomains (or "domains"), each of which can be independently computed on different processors or computing cores.

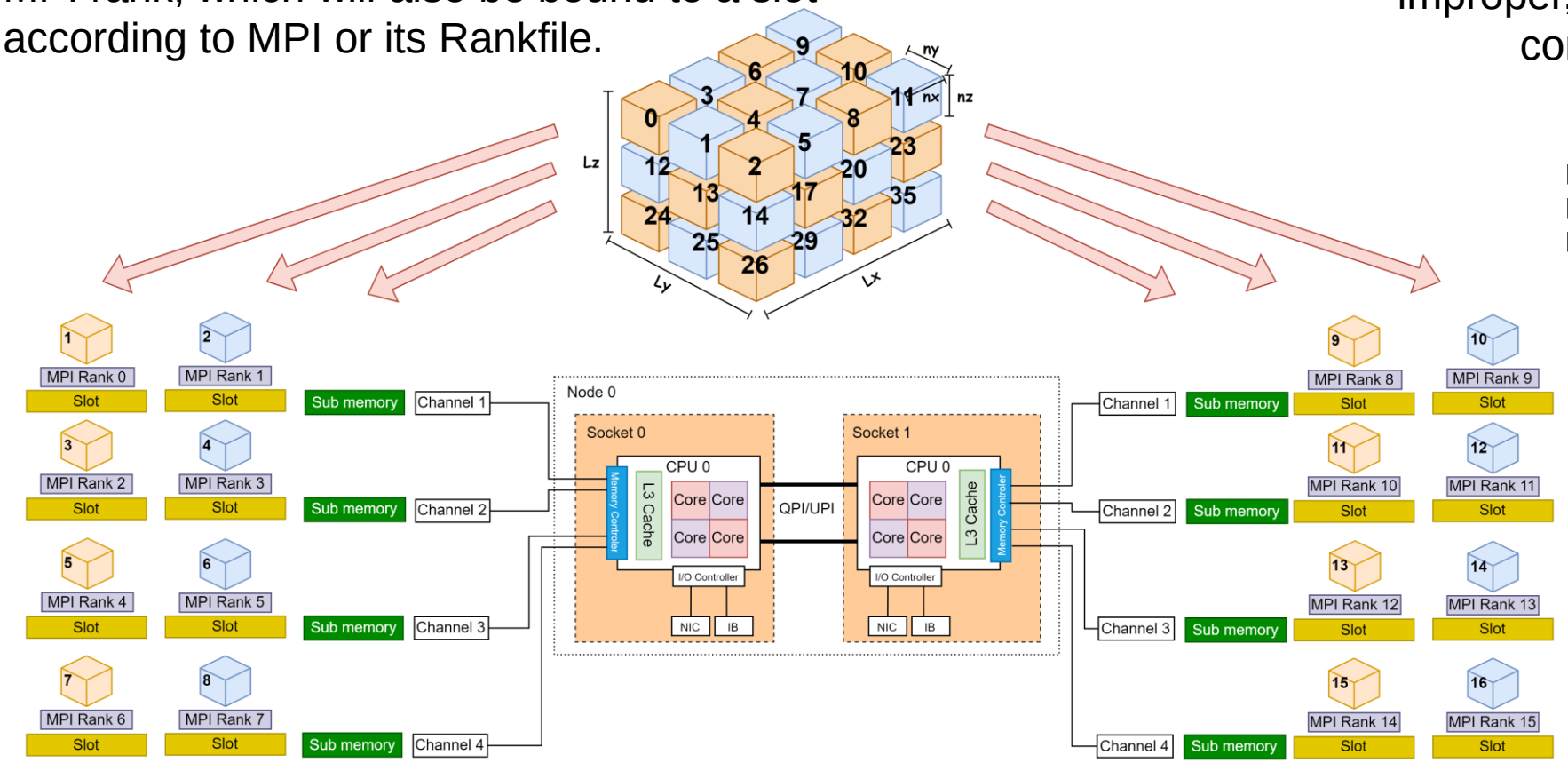


NUMA Distribution in NSCC （single node）

The decomposition will attempt to maximize the contact area between subdomains:

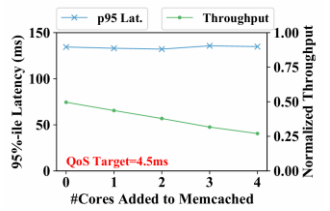
$$S = L.x \times L.y \times (n_{z_{try}} - 1) + L.x \times L.z \times (n_{y_{try}} - 1) + L.y \times L.z \times (n_{x_{try}} - 1)$$

Then, each subdomain will be assigned to a MPI rank, which will also be bound to a slot according to MPI or its Rankfile.



However, if the strategy for allocating slots is improper, it may lead to NUMA memory communication overhead issues.

PAC: Preference-Aware Co-location Scheduling on Heterogeneous NUMAArchitectures To Improve Resource Utilization (ICS 23)



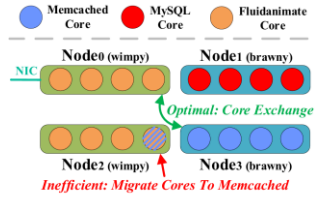


Table 2: The 95%-ile latency under different load when using different cores. The green cell indicates that the QoS target is satisfied while the red one indicates a QoS violation.

	MySQL									
QPS	W_0^3	W_0^4	W_0^3-B_3^1	$W_0^3B_1^1$	W_0^2-B_3^2	$W_0^2B_1^2$	W_0^1-B_3^3	$W_0^1B_1^3$	B_3^4	B_1^4
3K	11.65	9.73	9.45	9.39	9.28	9.06	9.02	8.86	6.43	6.09
6K	20.74	12.08	11.27	10.09	9.97	9.22	9.13	9.06	6.43	6.32
9K	66.84	22.74	19.28	12.98	12.65	10.27	9.63	9.06	7.17	6.91
12K	5K	59.99	36.9	20	19.26	12.98	11.28	10.09	8.43	7.98
15K	15K	1.7K	139.9	50.11	46.63	21.37	20.81	12.52	10.84	9.56
18K	22.8K	11.7K	4K	404.6	112.7	53.85	49.21	18.61	15	12.98
19K	24.1K	13.6K	6.4K	2.8K	150.3	81.48	63.32	22.69	19.29	13.7
20K	25.9K	15.7K	9K	6K	320.2	121.1	75.82	29.72	23.52	17.01

NUMA Distribution Optimization in NSCC（single node）

SUSTech Southern University of Science and Technology

Objective

$$\text{Minimize} \quad (\text{deviation})^2$$

vars	description
$x[i,k]$	Binary variable if process i is bound to NUMA node k
$y[i,j]$	Binary variable if process i is bound to logical core j

$$\text{deviation} = \sum_{i=0}^{P-1} \sum_{j=0}^{P-1} \sum_{k=0}^{K-1} \text{adjacency\_matrix}[i][j] \cdot x[i,k] \cdot x[j,k] - \text{target}$$

- Target number of neighboring domains on the same NUMA node.

Constrains

$$\sum_{i=0}^{P-1} x[i,k] \leq 16 \quad \forall k \in \{0,\ldots,K-1\}$$

- Each NUMA node can bind at most 16 processes

$$\sum_{k=0}^{K-1} x[i,k] = 1 \quad \forall i \in \{0,\ldots,P-1\}$$

$$\sum_{j=0}^{C-1} y[i,j] = 2 \quad \forall i \in \{0,\ldots,P-1\}$$

- Each process is bound to exactly one NUMA node and two logical cores

$$y[i,j] = y[i,j+1] \quad \forall i \in \{0,\ldots,P-1\}, \forall j \in \{0,2,4,\ldots,C-2\}$$

- Logical cores are paired (hyper-threading)

$$\sum_{i=0}^{P-1} y[i,j] \leq 1 \quad \forall j \in \{0,\ldots,C-1\}$$

- Each logical core can bind at most one

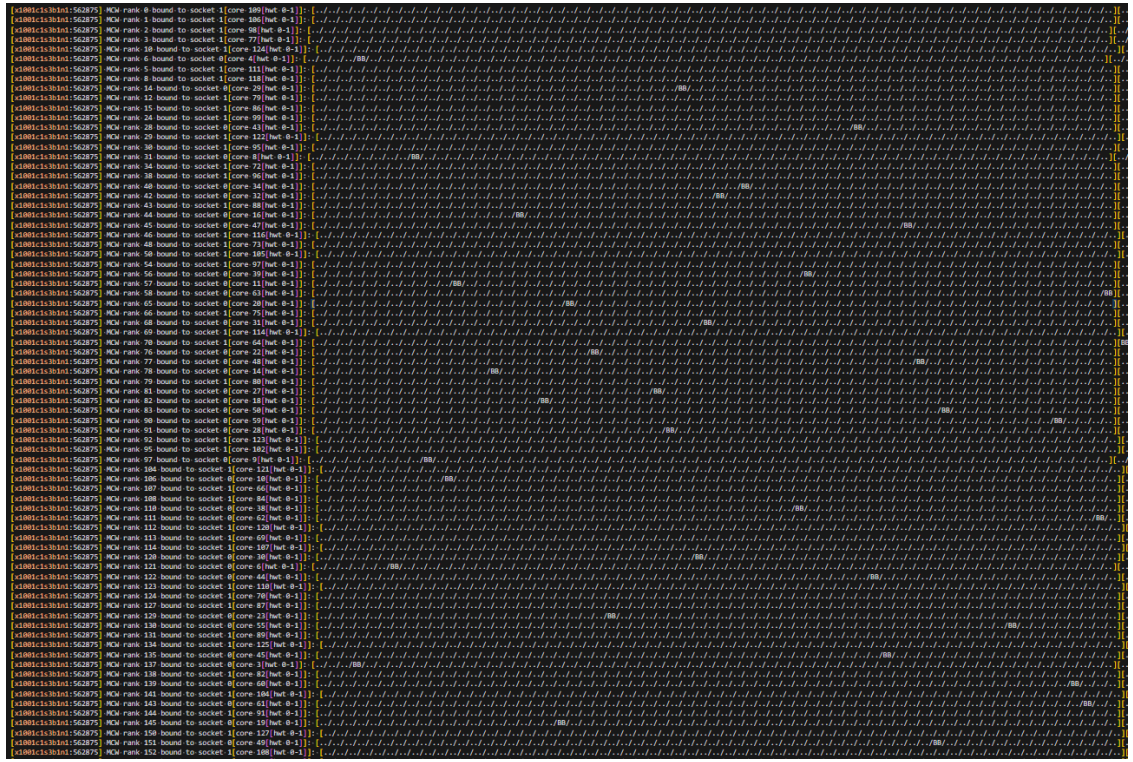
Output of optimization

SUSTech Southern University of Science and Technology

GUROBI OPTIMIZATION

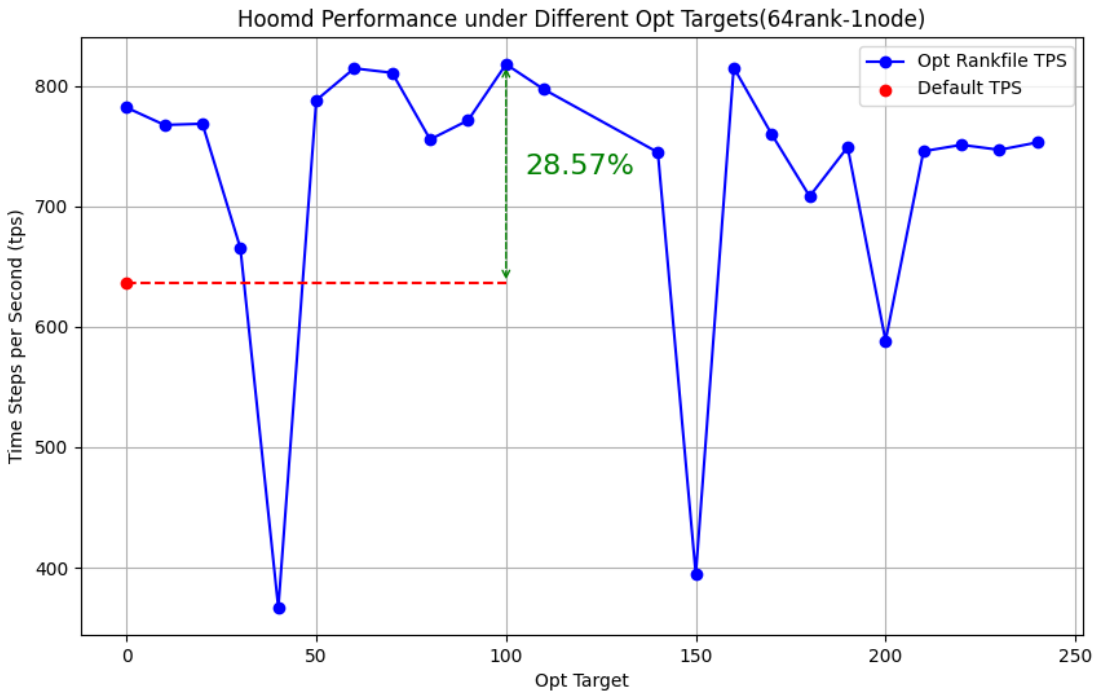
Gurobi is a high-performance mathematical programming optimization software

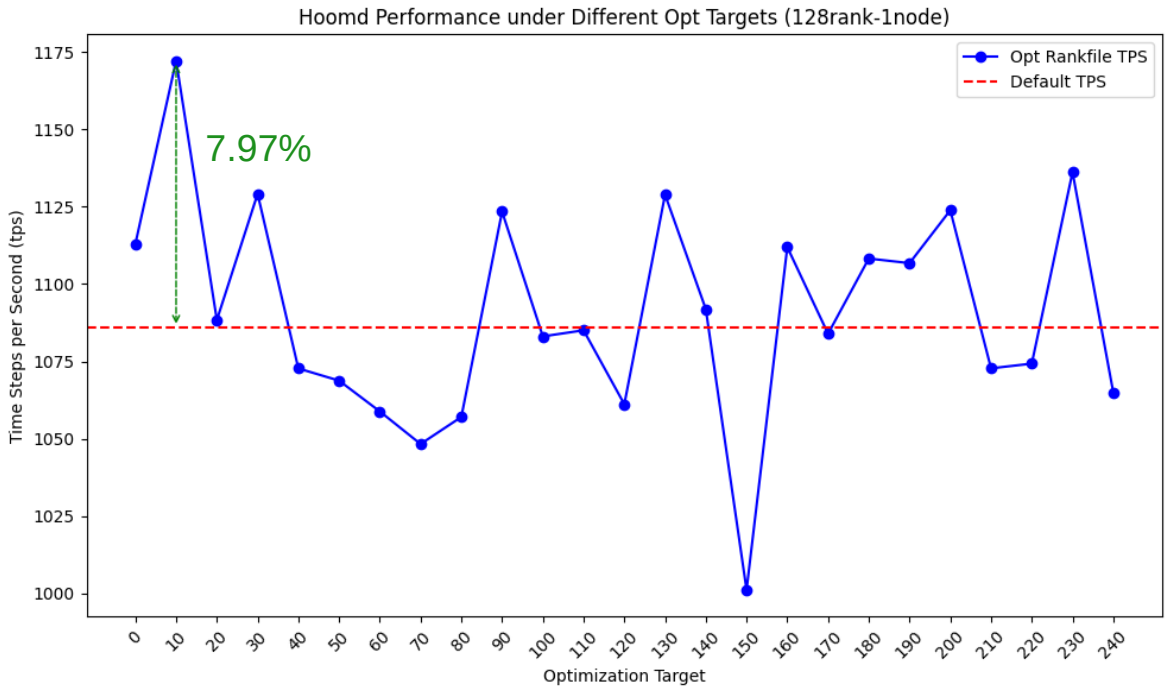
```
rank 0=node0 slot=102,103
rank 1=node0 slot=14,15
rank 2=node0 slot=176,177
rank 3=node0 slot=52,53
rank 4=node0 slot=100,101
rank 5=node0 slot=138,139
rank 6=node0 slot=4,5
rank 7=node0 slot=62,63
rank 8=node0 slot=224,225
rank 9=node0 slot=12,13
rank 10=node0 slot=140,141
rank 11=node0 slot=146,147
rank 12=node0 slot=110,111
rank 13=node0 slot=130,131
rank 14=node0 slot=180,181
rank 15=node0 slot=60,61
rank 16=node0 slot=228,229
rank 17=node0 slot=88,89
rank 18=node0 slot=24,25
rank 19=node0 slot=18,19
rank 20=node0 slot=230,231
.....
```



- General solver
- Rankfile format
- Core Binding Strategy in Openmpi

Improvement of optimization （single node)





- The solver can quickly obtain the optimized rankfile in the case of a single node

28.57% improvement for 1 nodes

- The distribution policy for the rank of adjacent domains in NUMA nodes is not the denser the better

7.97% improvement for 1 nodes

Works for different numbers of ranks

SUSTech Southern University of Science and Technology

If we have the rank-neighbors information

This optimization can fit any rank and NUMA architect in different CPUs

```
    for (unsigned int r = 0; r < nranks; ++r)
        {
        m_grid_pos = m_index.getTriple(h_cart_ranks_inv.data[r]);
        std::ostringstream oss;
        oss << "Rank " << r << " neighbors:";
        for (unsigned int dir = 0; dir < 6; ++dir)
            {
            unsigned int neighbor_rank = getNeighborRank(dir);
            oss << " " << neighbor_rank;
            }
        m_exec_conf->msg->notice(2) << oss.str() << std::endl;
        }

    // compute position of this box in the domain grid by reverse look-up
    m_grid_pos = m_index.getTriple(h_cart_ranks_inv.data[rank]);
    }
```

```
notice(2): Using domain decomposition: n_x = 4 n_y = 8 n_z = 8
notice(2): Rank 0 neighbors: 1 3 4 28 32 224
notice(2): Rank 1 neighbors: 2 0 5 29 33 225
notice(2): Rank 2 neighbors: 3 1 6 30 34 226
notice(2): Rank 3 neighbors: 0 2 7 31 35 227
notice(2): Rank 4 neighbors: 5 7 8 0 36 228
notice(2): Rank 5 neighbors: 6 4 9 1 37 229
notice(2): Rank 6 neighbors: 7 5 10 2 38 230
notice(2): Rank 7 neighbors: 4 6 11 3 39 231
```

Summary

Optimization at most is shown:

Optimization Method	Qiming 2.0	Gadi	NSCC
Using HPC-X communications library	**413.8% improvement for 32 nodes**	**29.89% improvement for 32 nodes**	**1.78% improvement for 32 nodes**
O3 Compiler Flags	**10.53% improvement for 16 nodes**	-	-
Tuning with UCX Framework	-	**1.46% improvement for 32 nodes**	**45.3% improvement for 32 nodes**
Better buffer params	**9.17% improvement for 32 nodes**	**0.37% improvement for 32 nodes**	**4.86% improvement for 32 nodes**
NUMA – aware Distribution	-	-	**28.57% improvement for 1 nodes**

Future Works

1. Generate a more effective strategy from PAC (ICS 23) and expand the optimization to multi-nodes.
2. Change Domain Decomposition from space-based to particle-based.
3. Optimize MPI rank allocation to minimize physical communication overhead across multiple nodes.

SUSTech Southern University of Science and Technology

Part 2

Artificial Intelligence

LitGPT Llama-2-7b finetune-full

Test Cluster Configuration - NSCC (GPU)

SUSTech Southern University of Science and Technology

HW Info.	NSCC	
	CPU	Dual-CPU AMD EPYC 7713, 128 cores per server.
A GPU node R4 queue	GPU	4x Nvidia A100(40G)
	Memory	512 GB

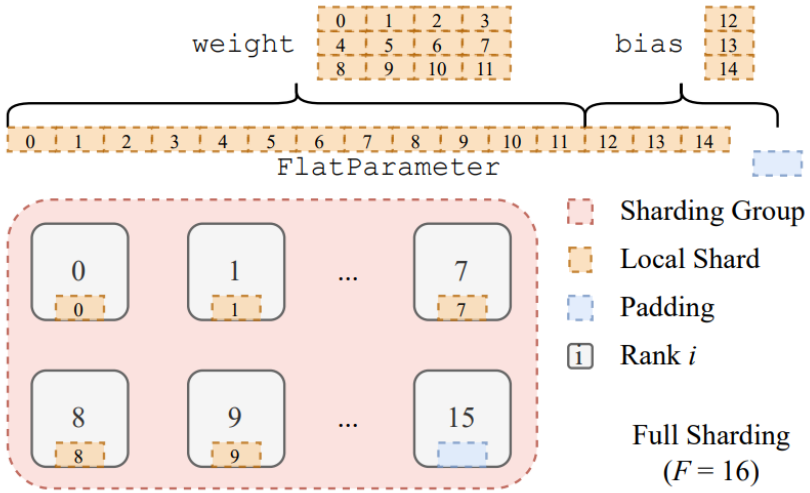
SW Info.	NSCC
Operating System	Red Hat Enterprise Linux-8
IB Bandwidth	100Gbi
IB Drive	OFED-internal-5.7-1.0.2
MPI Libraries	openmpi/4.1.2-hpe

Application Info.	
Application	Litgpt-Llama2-sft
Required MPI installation	openmpi/4.1.2-hpe
Required Libraries	litgpt=0.5.2 lightning=2.4.0 torch=2.4.1 transformers=4.44.2

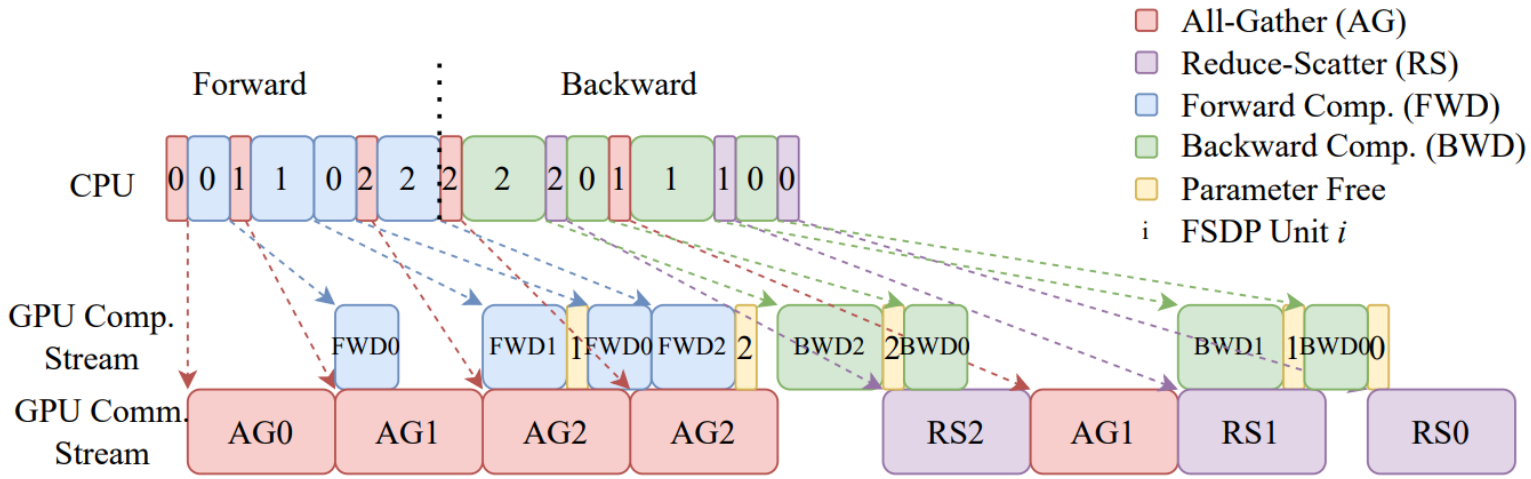
GPU task Introduction

- Litgpt: a llm train and inference framework
- Llama2-7B: a llm model raise by meta
- FSDP: a algorithm of llm distributed training
- Target: Less training time in alpaca1024 SFT

Sharding params opt grade

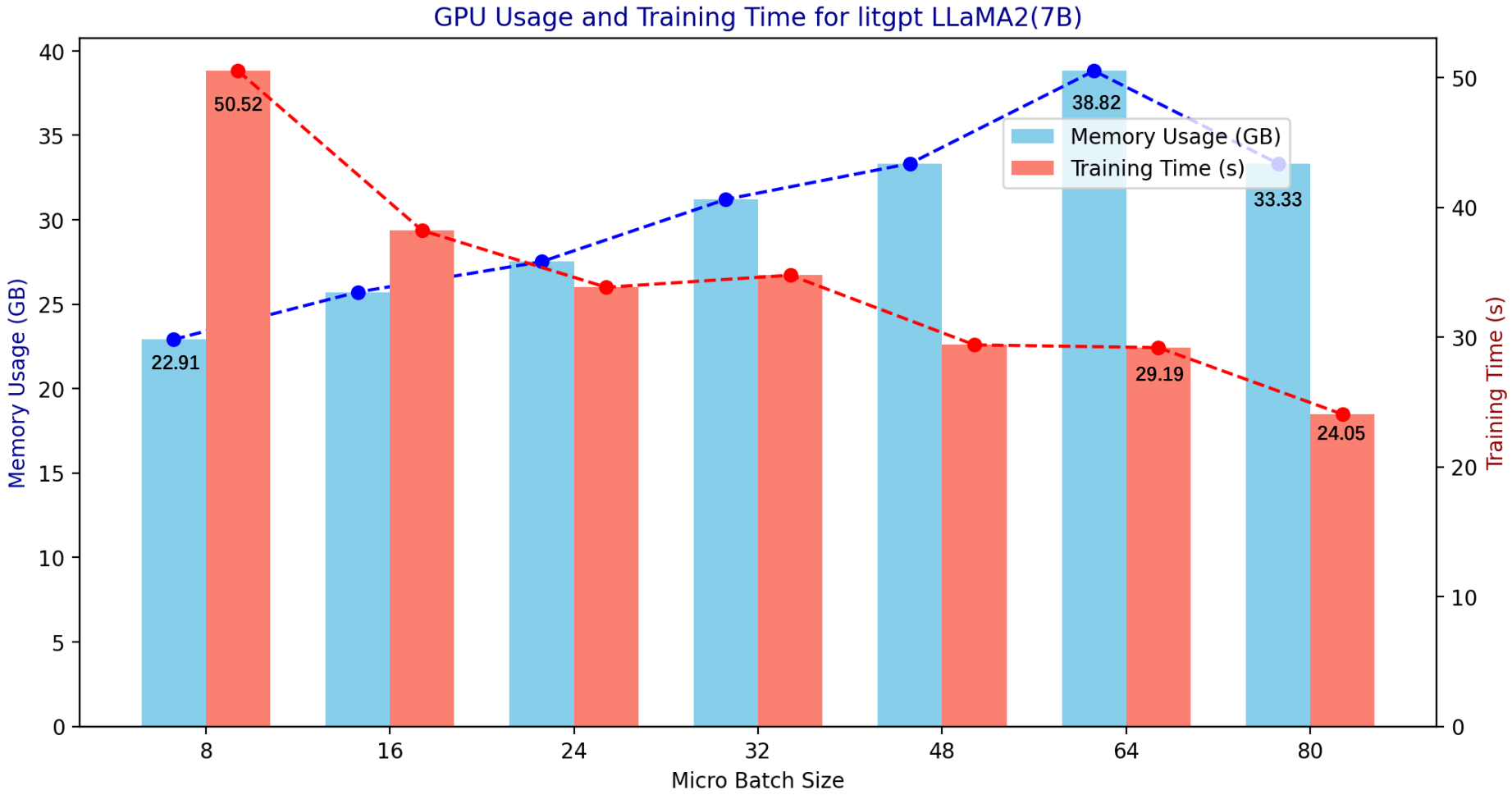


Overlap Communication and Computation



PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel

GPU Benchmark in NSCC(A100 40G)



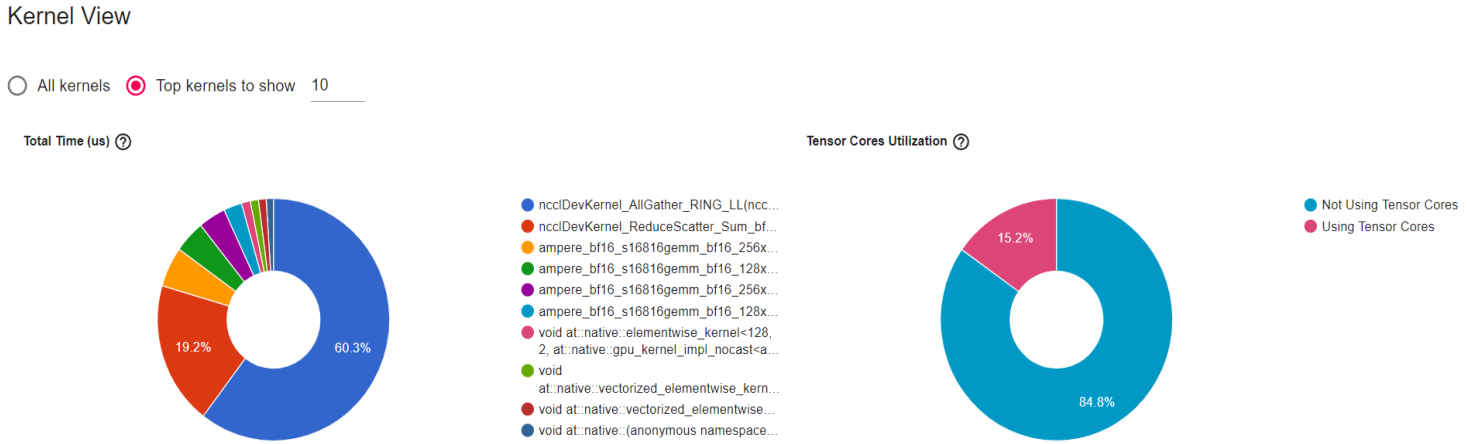
- Bathsize↑
  - Training time↓
  - GPU memory↑

Training time and GPU Memory Usage are greatly affected by batch size

Profile between different batch size

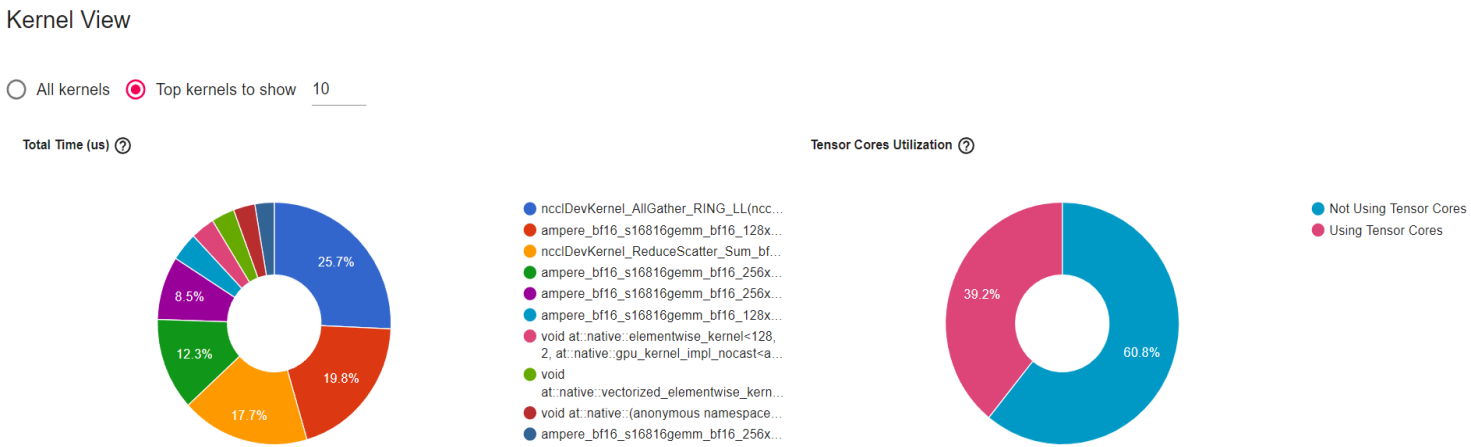
Southern University of Science and Technology

micro batch size = 8



- Bathsize↑
  - Percentage of communication time↓
  - Percentage of TensorCore usage↑

micro batch size = 64



Nvidia A100 Tensor Core Usage

SUSTech Southern University of Science and Technology

NVIDIA A100 TENSOR CORE GPU SPECIFICATIONS (SXM4 AND PCIE FORM FACTORS)

	A100 40GB PCIe	A100 80GB PCIe	A100 40GB SXM	A100 80GB SXM
FP64	9.7 TFLOPS			
FP64 Tensor Core	19.5 TFLOPS			
FP32	19.5 TFLOPS			
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*			
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
INT8 Tensor Core	624 TOPS | 1248 TOPS*			

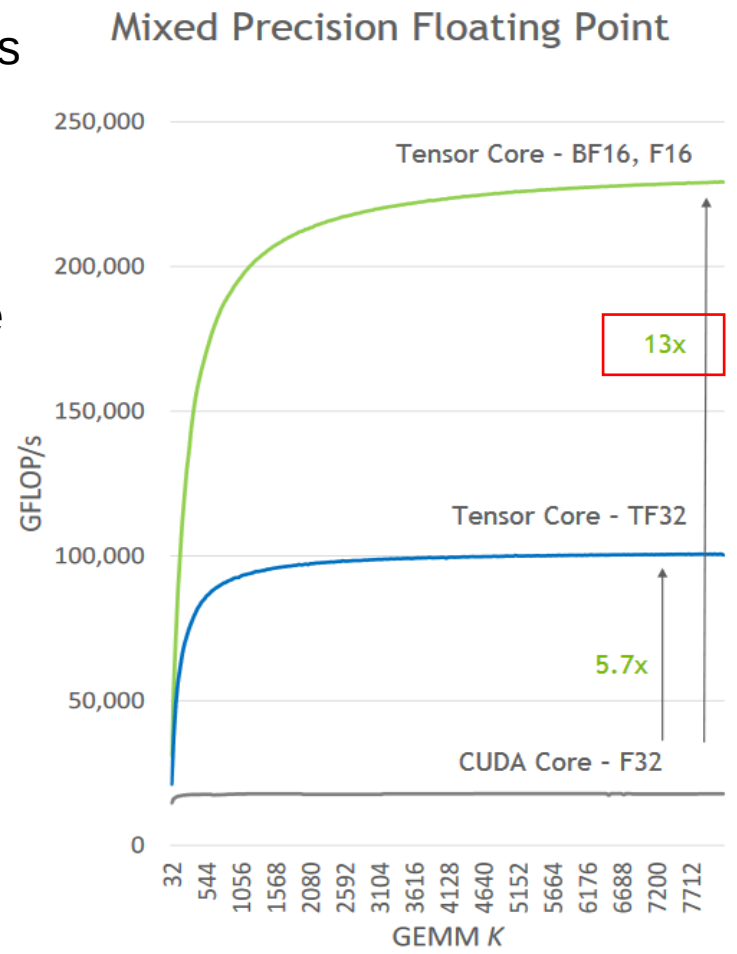
speedup for (large) matrix computations

⇩

Use Tensor Core as much as possible

⇩

- **Memory bandwidth Utilization ↑**
- **Memory access Locality↑**
- **Data throughput↑**

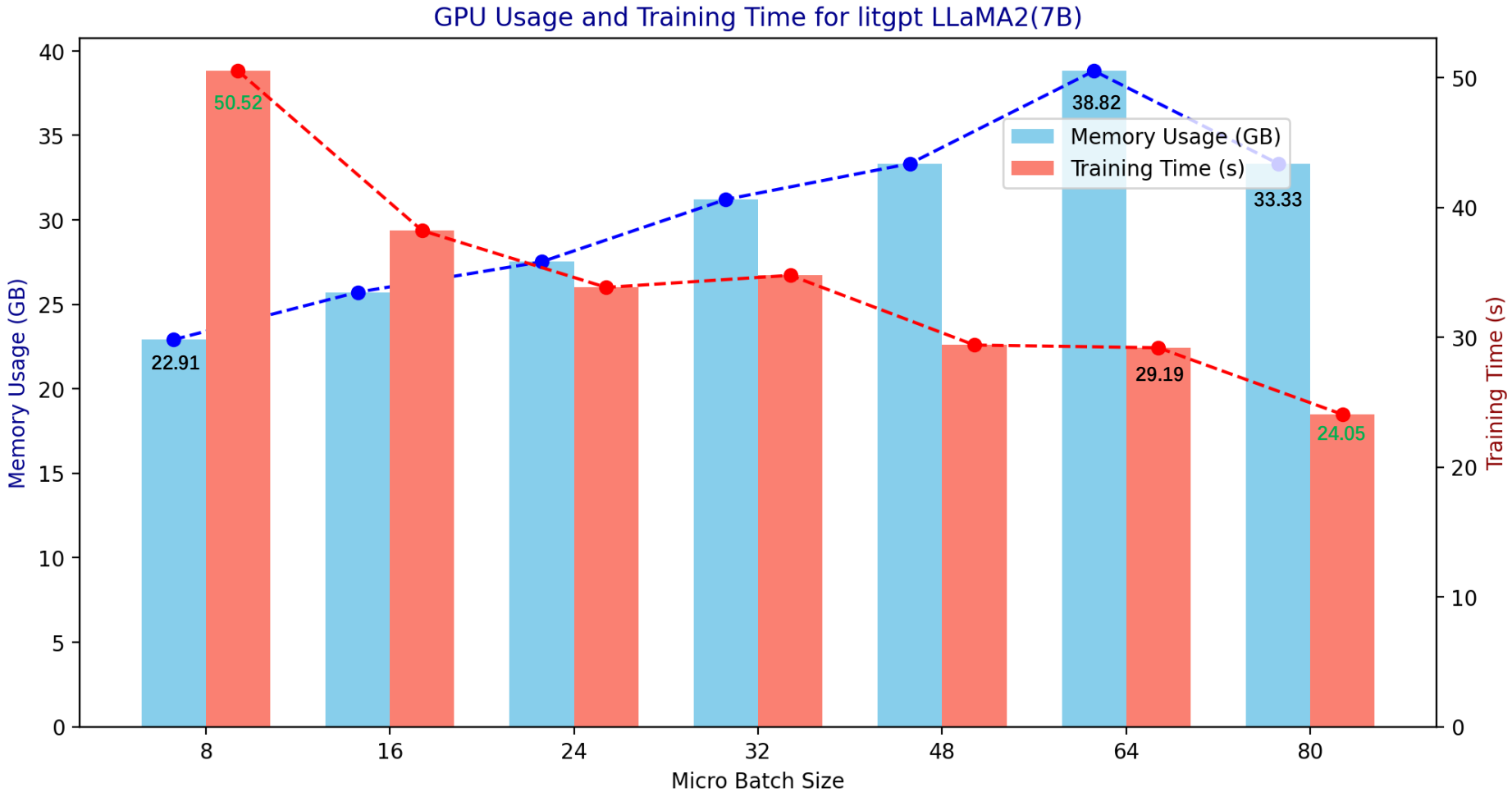


CUTLASS Performance

Just give it enough data to compute

- Vectorized memory access
- Asynchronous memcpy

Batch Size Benchmark



- Batchsize↑
  - Training time↓
  - GPU memory↑

Training Time speedup about 2.10x for 2 nodes 8 GPUs

Different strategy

SUSTech Southern University of Science and Technology

```python
from lightning.fabric.strategies import FSDPStrategy
```

```python
if devices * num_nodes > 1:
    strategy = FSDPStrategy(
        auto_wrap_policy={Block},
        activation_checkpointing_policy={Block},
        state_dict_type="full",
        limit_all_gathers=True,
        cpu_offload=False,
    )
else:
    strategy = "auto"

fabric = L.Fabric(
    devices=devices,
    num_nodes=num_nodes,
    strategy=strategy,
    precision=precision,
    loggers=logger
)
```

litgpt src: finetune/full.py#L100

FSDP Strategy

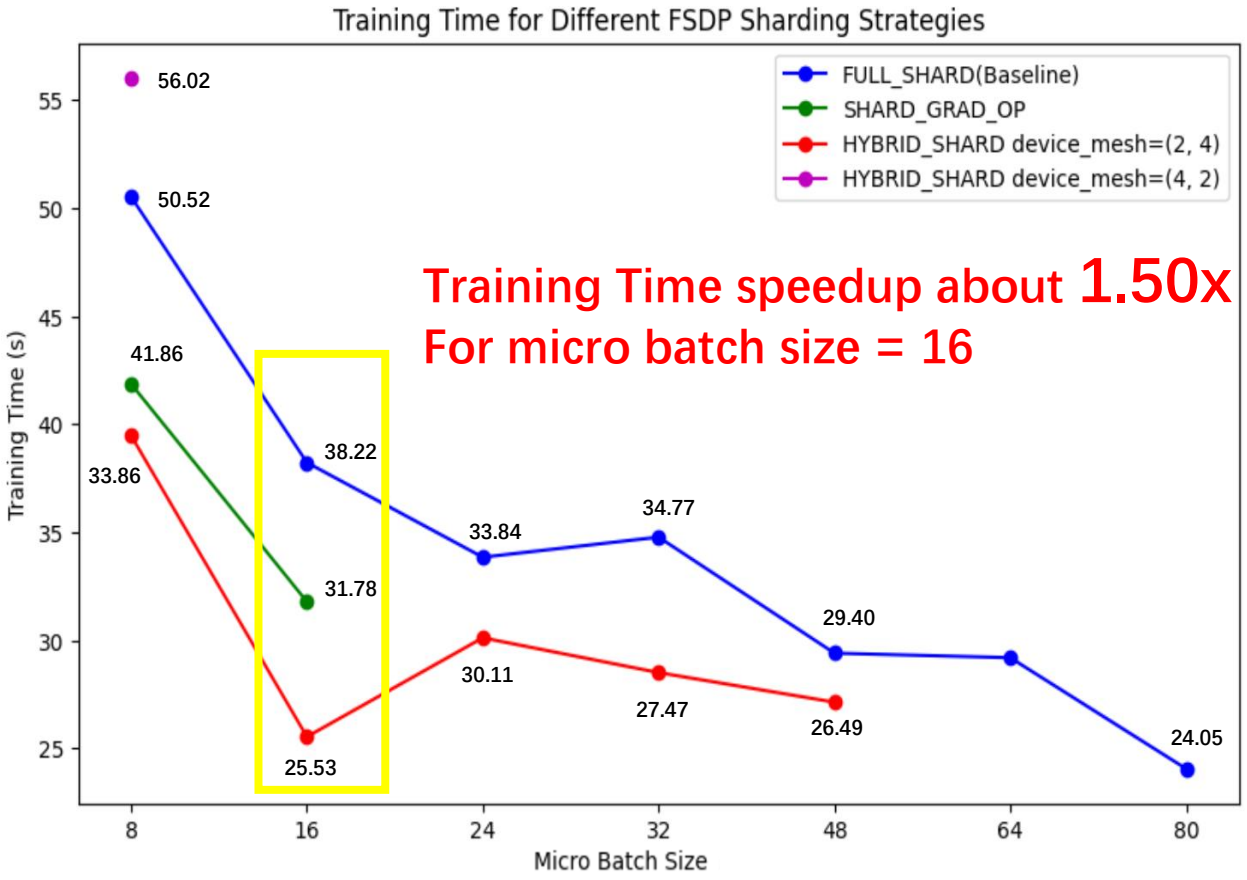
- FULL_SHARD → parameters, gradients, optimizer states
- SHARD_GRAD_OP → gradients, optimizer states
- NO_SHARD → no sharding = regular DDP
- HYBRID_SHARD → shards in node, replicates across nodes

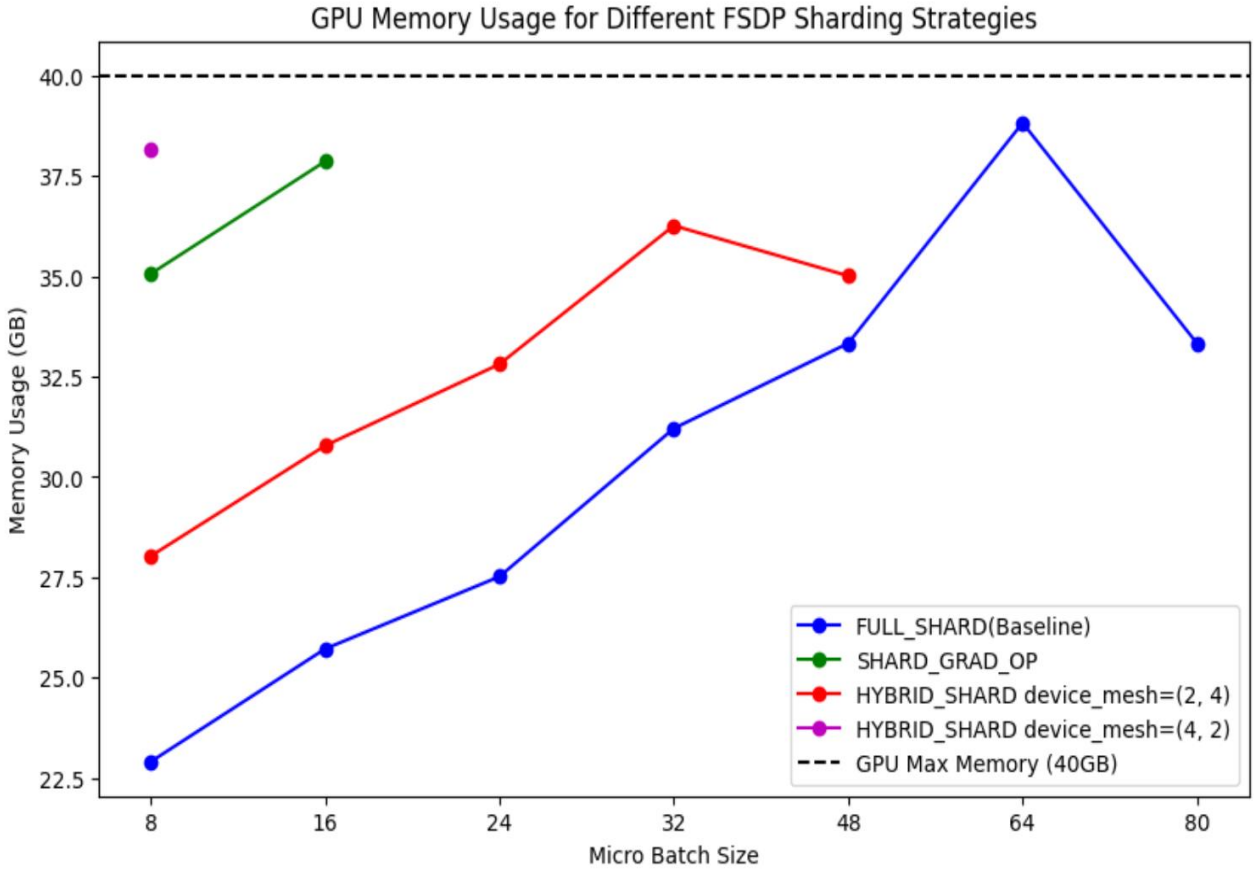
Full Sharding

- memory utilization ↑
- communication cost ↑

What if the **batch size** is increased simultaneously?

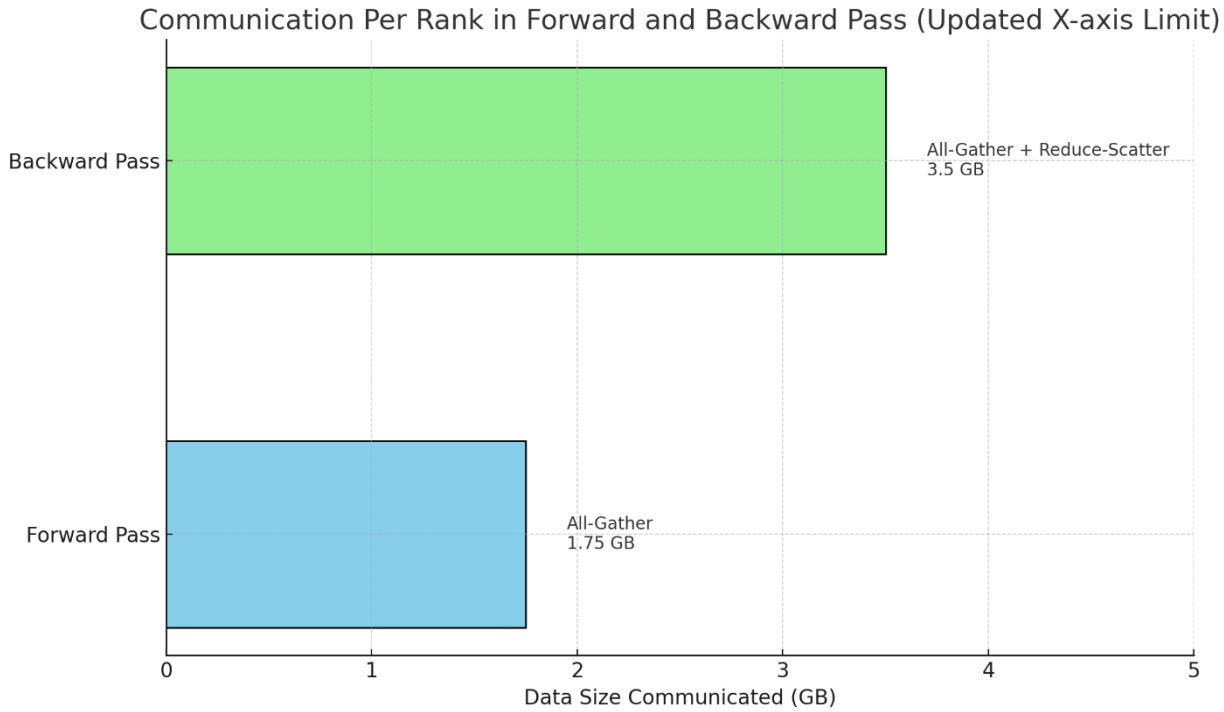
Optimization of FSDP Sharding Strategy and Parameters

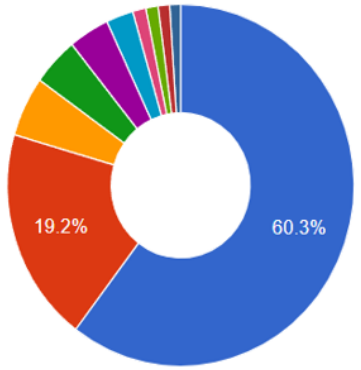


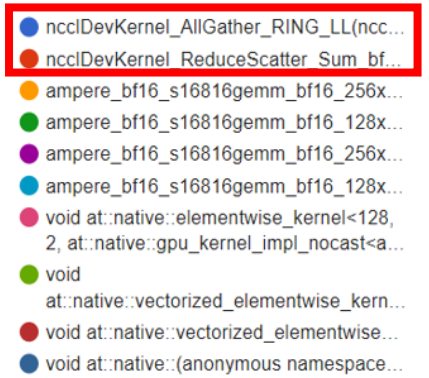


HYBRID_SHARD device mesh= (2, 4) achieves the shortest training time for micro batch sizes under 48

Total Parameters and Sharding Across GPUs







micro batch size = 8

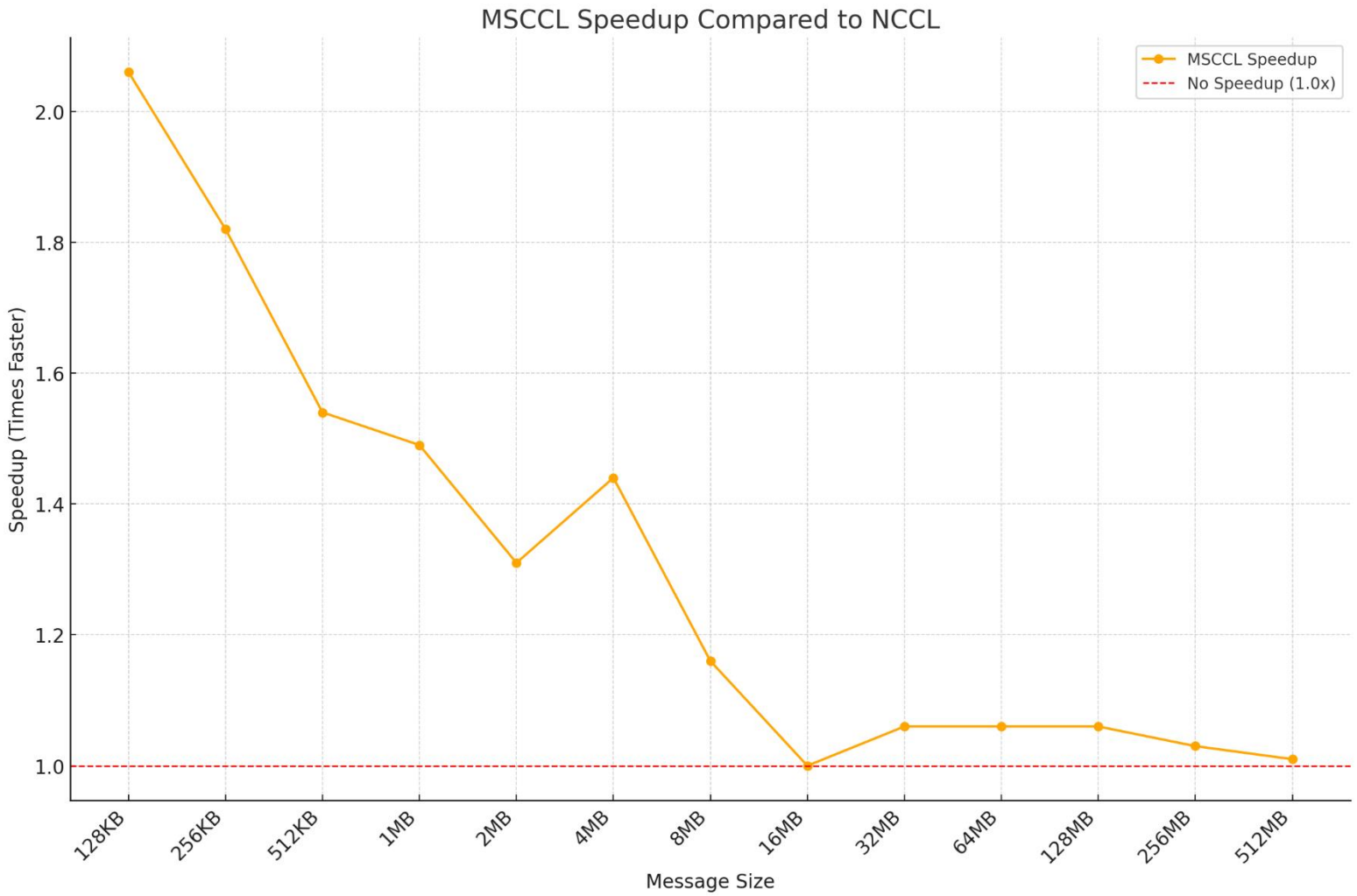
communication data size is large

⇩

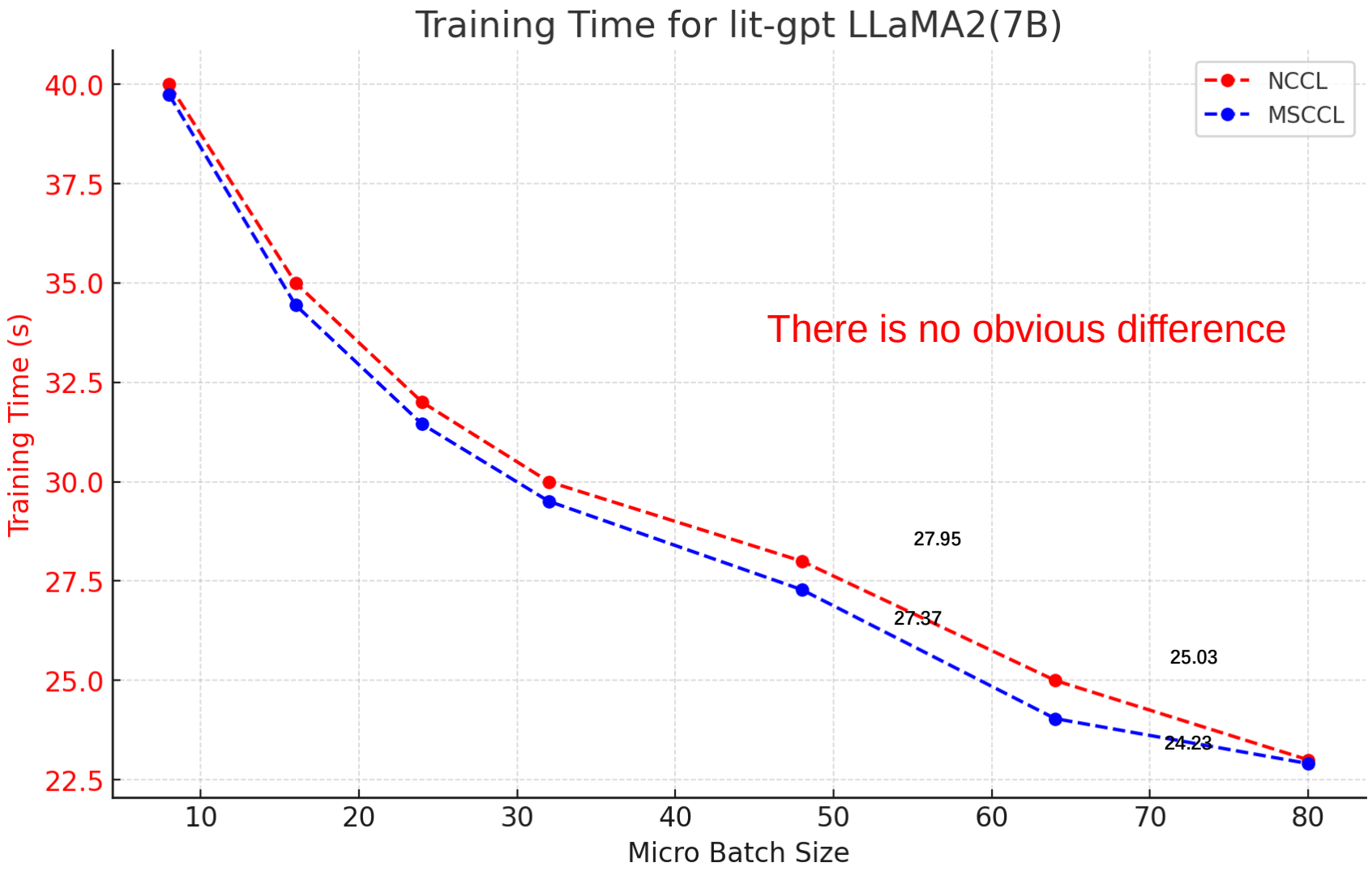
Reduce communication time

MSCCL Speedup Compared to NCCL

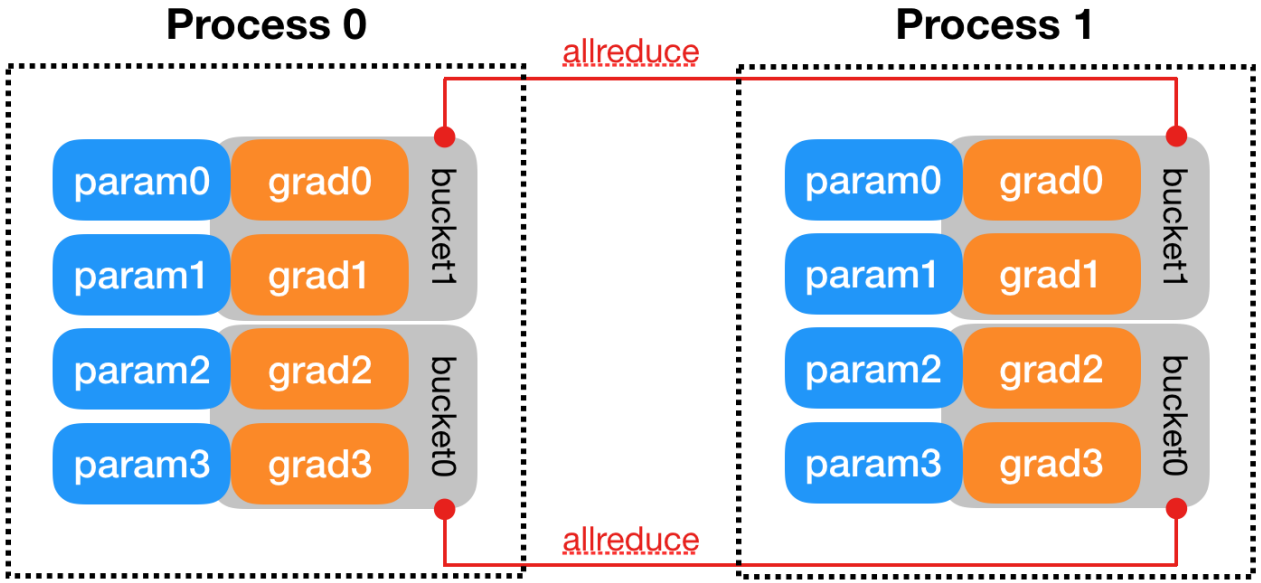
SUSTech Southern University of Science and Technology

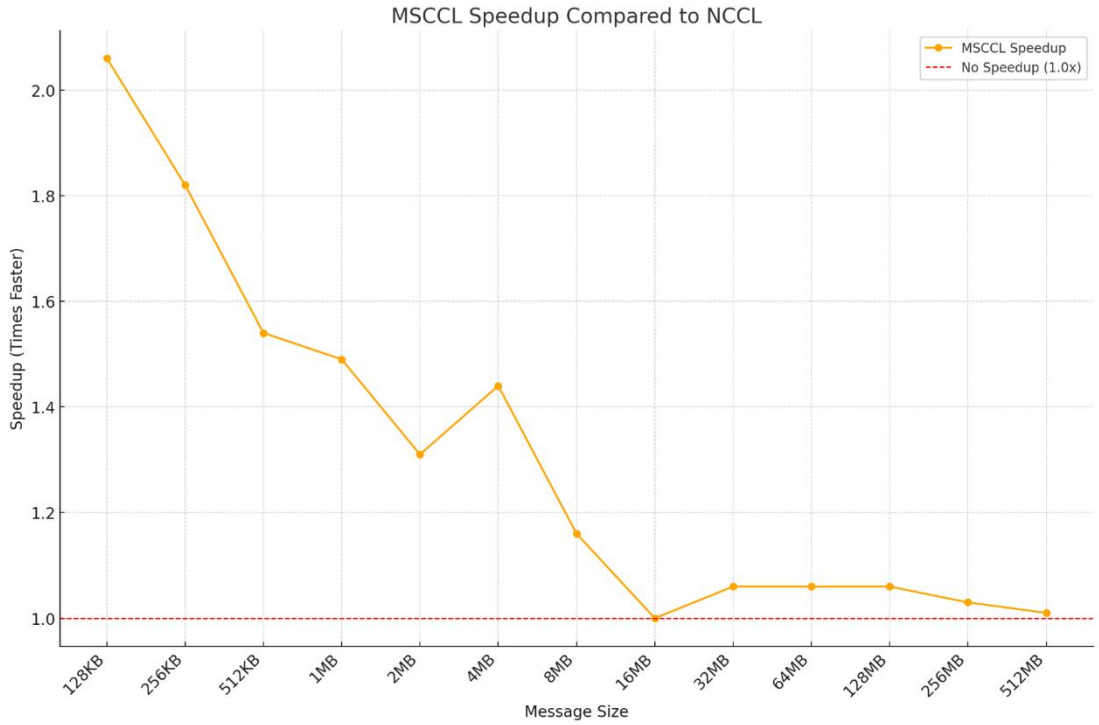


Result in NCCL and MSCCL



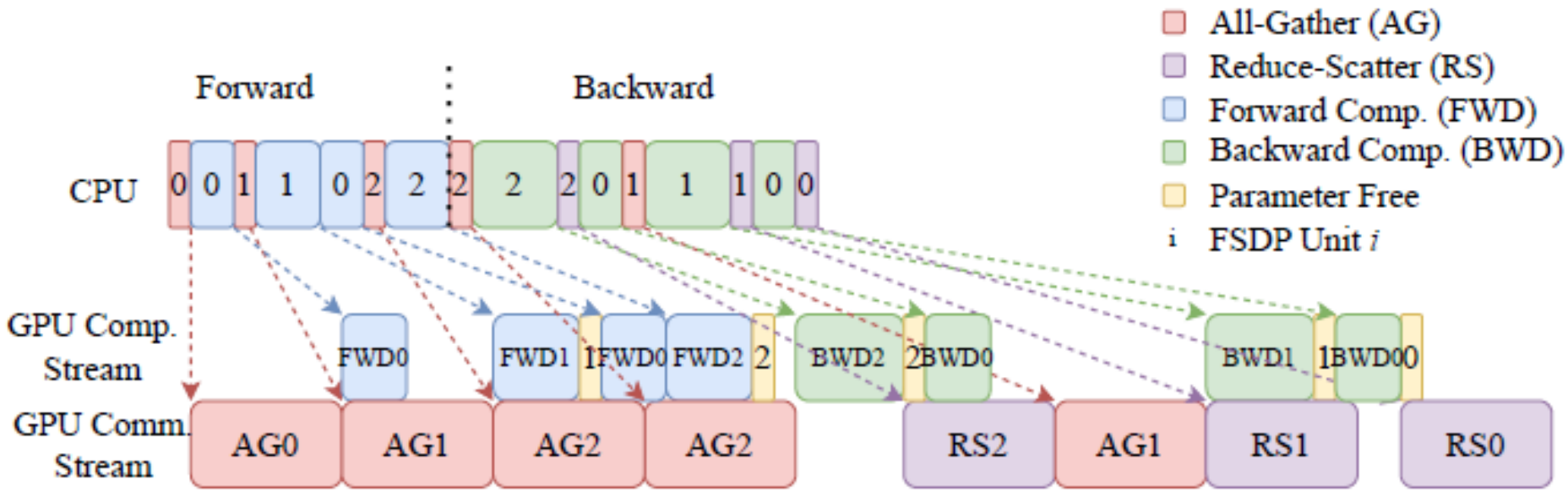
MSCCL Optimization at Default Bucket Size





- PyTorch's default bucket size: **25MB**.
- **MSCCL** optimization for bucket sizes around **16MB-32MB** shows:
  - Speedup close to **1.0x** compared to NCCL.
  - Indicates **minimal performance improvement** in this size range.

Overlaps computation and communication



The scheduling overlaps computation (FWD/BWD) with communication (AG/RS) to minimize idle times.

Faster Computational Operator

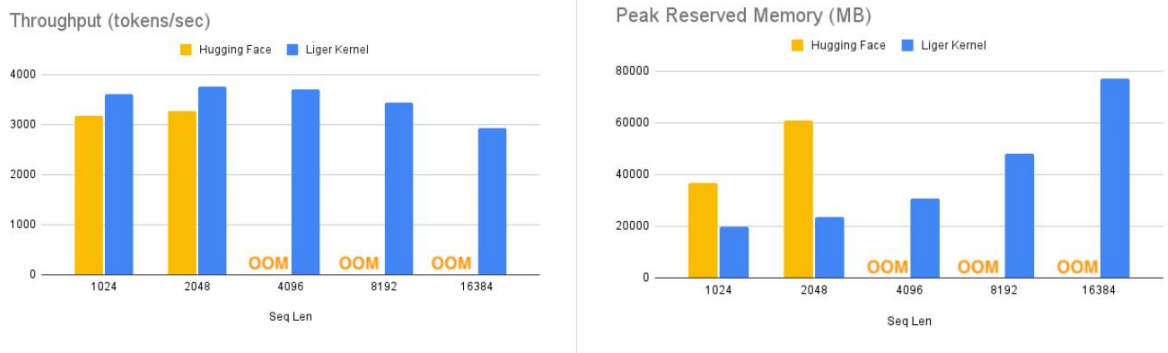
SUSTech Southern University of Science and Technology

Core computational operators in Transformer models

- Attention
- MLP
- Norm
- Rotary Position Embedding
- Embedding

We want to improve training performance with Liger Kernel

Which is A collection of Triton kernels



From Liger Kernel Github

But

litgpt does not directly use transformers and has own blocks for loading weight

Not available out of the box

We need:
Looking forward to the integration
or participated in development of litgpt train

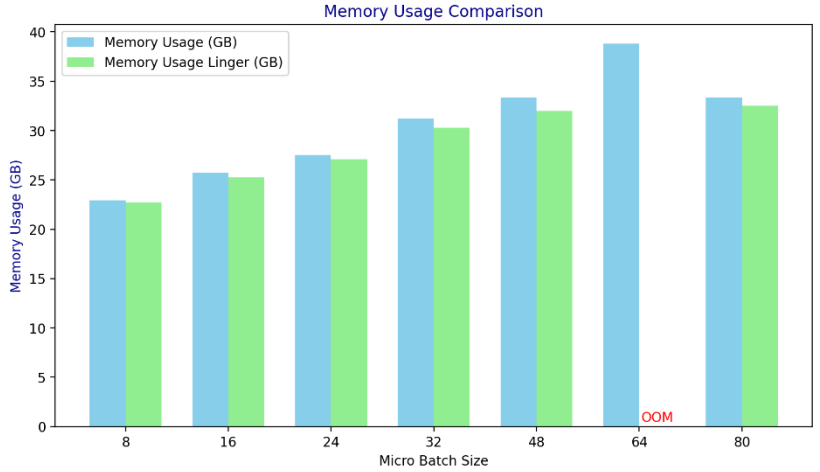
Faster Computational Operator

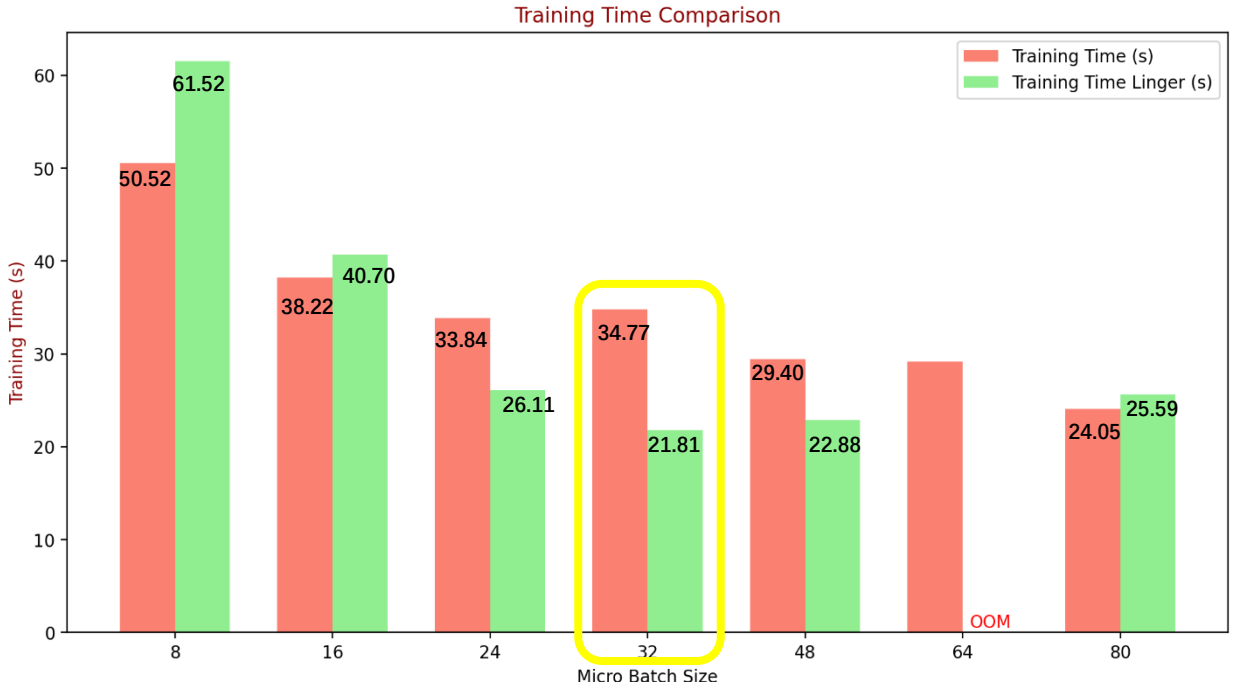
SUSTech Southern University of Science and Technology

Core computational operators in Transformer models

- **Attention → Flash Attention**
- **MLP**
- **Norm → Liger Kernel Implementation**
- **Rotary Position Embedding**
- **Embedding**

We replaced *RMSNorm* with Liger Kernel implementation





Training Time speedup about **1.594x**
For micro batch size = 32