APAC - SUSTech 0x01

APAC 2024 Optimization Report Summary

HPC Groups:
ZuDong Li (leader)
Haibin Lai
Benxiang Xiao
Zixu Wang
Wenhan Tan
Wenbo An

AI Groups:
Yukun Yang
Honglie Li
Junyu Su

Abstract

In this report, we detail the optimization efforts conducted on two key applications for the APAC2024 competition: HPC HOOMD-blue and AI Llama2. Initially, we performed benchmark tests on both applications, transitioning from a single-node setup to a multi-node, multi-GPU environment.

For **HOOMD-blue**, a high-performance molecular dynamics simulation program, we carried out an in-depth analysis of the source code and implemented several optimizations. These included switching to the HPC-X communication library, applying O3 compiler optimization, fine-tuning UCX parameters, replacing Intel MPI with OpenMPI, adjusting program buffer settings, and employing NUMA-aware scheduling for domain decomposition to enhance performance in a multi-node setup.

In the case of **Llama2**, a large-scale language model used for AI tasks, we built upon benchmark results to profile different batch sizes and optimized performance on A100 GPUs using Tensor Cores. We also optimized the FSDP (Fully Sharded Data Parallel) strategy and parameters, replaced the NCCL communication library with MSCCL for improved multi-GPU communication, and introduced techniques to overlap computation with communication. Additionally, we incorporated faster computational operators to further boost training efficiency.

Through these efforts, we achieved significant performance improvements in both the HPC and AI tasks, deepening our knowledge of high-performance computing optimizations and enhancing our profiling capabilities.

About us

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 HOOMD-blue

Introduction

Listing 1 Molecular dynamics simulation

```
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

```
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)
```

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 with a range of pair, bond, angle, and other potentials. Many features are targeted at the soft matter research community, though the code is general and capable of many types of particle simulations. It has been actively developed since March 2007 and available open source since August 2008. [1]

CPU Task Workload And Benchmark

The Benchmark needs us to compute WCA potential for the following parameter:

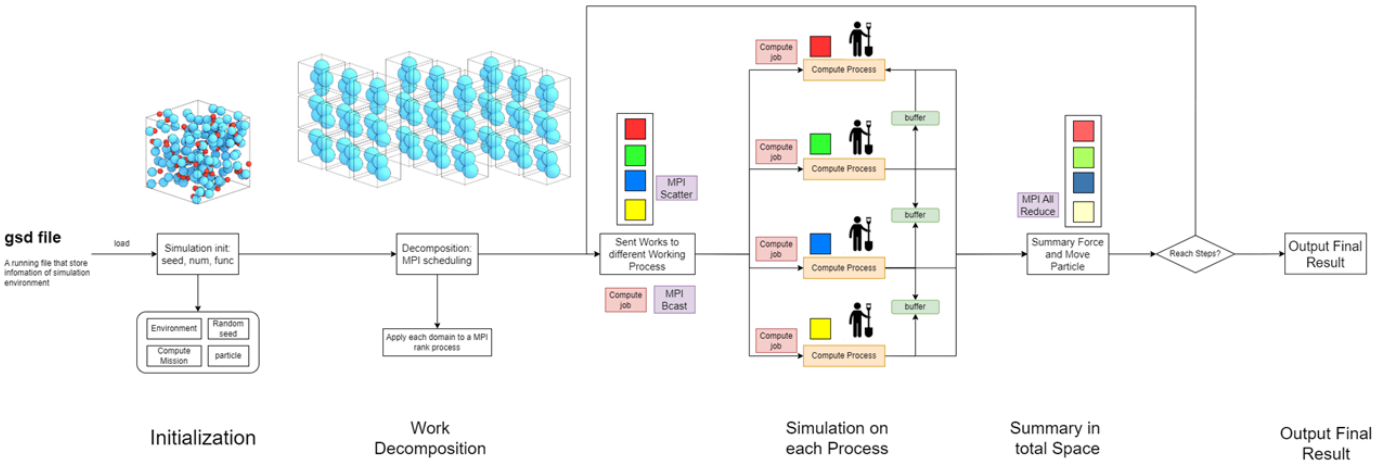
- **Workload**: md_pair_wca
- **Number of particles**: `200,000`
- **Input data**:hard_sphere_200000_1.0_3.gsd

HOOMD-Blue Benchmark calculates **WCA potential** for N particles as workload. WCA potential (Weeks-Chandler-Andersen potential) is a potential energy function used to simulate intermolecular interactions. It is simple, computationally efficient and suitable.

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

- The WCA potential is commonly used in molecular dynamics simulations, especially when studying physical phenomena such as liquids, gases, and their phase transitions.
- It provides higher computational efficiency by removing the attractive part of the potential.

System Architecture



The system Architecture of HOOMD-blue is shown in the figure. In initialization part, the position and number of particle will be set, then the whole space will be divided into subdomain using a technique called `domain decomposition`. Then each subdomain will be sent to a MPI worker to a slot, computing forces and communicate with each other with buffer. After computation, they will gather and sent the result using `MPI_AllReduce` primitive, and do the computation again until reaching the steps and output result.

Testing Platform

To test the programme, we use QiMing cluster on SUSTech CCSE. It has 24 CPU cores for each nodes, 64GB Memory and connected with infiniband.

Test Cluster Configuration - QiMing 2.0 (CPU)

SUSTech Southern University of Science and Technology

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

And we also use Gadi, it has 24 CPU cores per node with 192GB Memory and connected with infiniband.

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

The last cluster we use is NSCC. It has 128 cores per server and 512GB Memory and 100 Gbi infiniband.

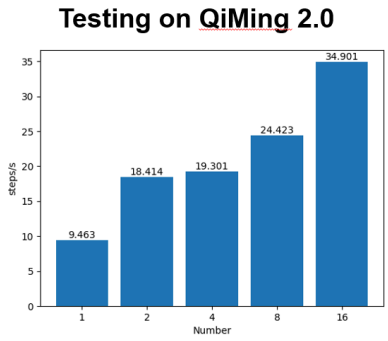
Test Cluster Configuration - NSCC (CPU)

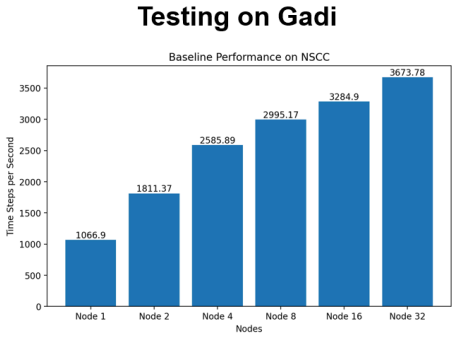
SUSTech Southern University of Science and Technology

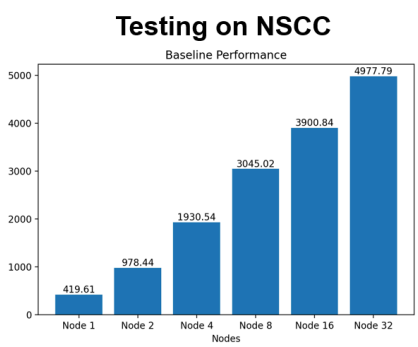
HW Info.	NSCC	
	CPU	Dual-CPU AMD EPYC 7713, 128 cores per server.
A GPU node R4 queue	GPU	4x Nvidia A100(40G)
	Memory	512 GB

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



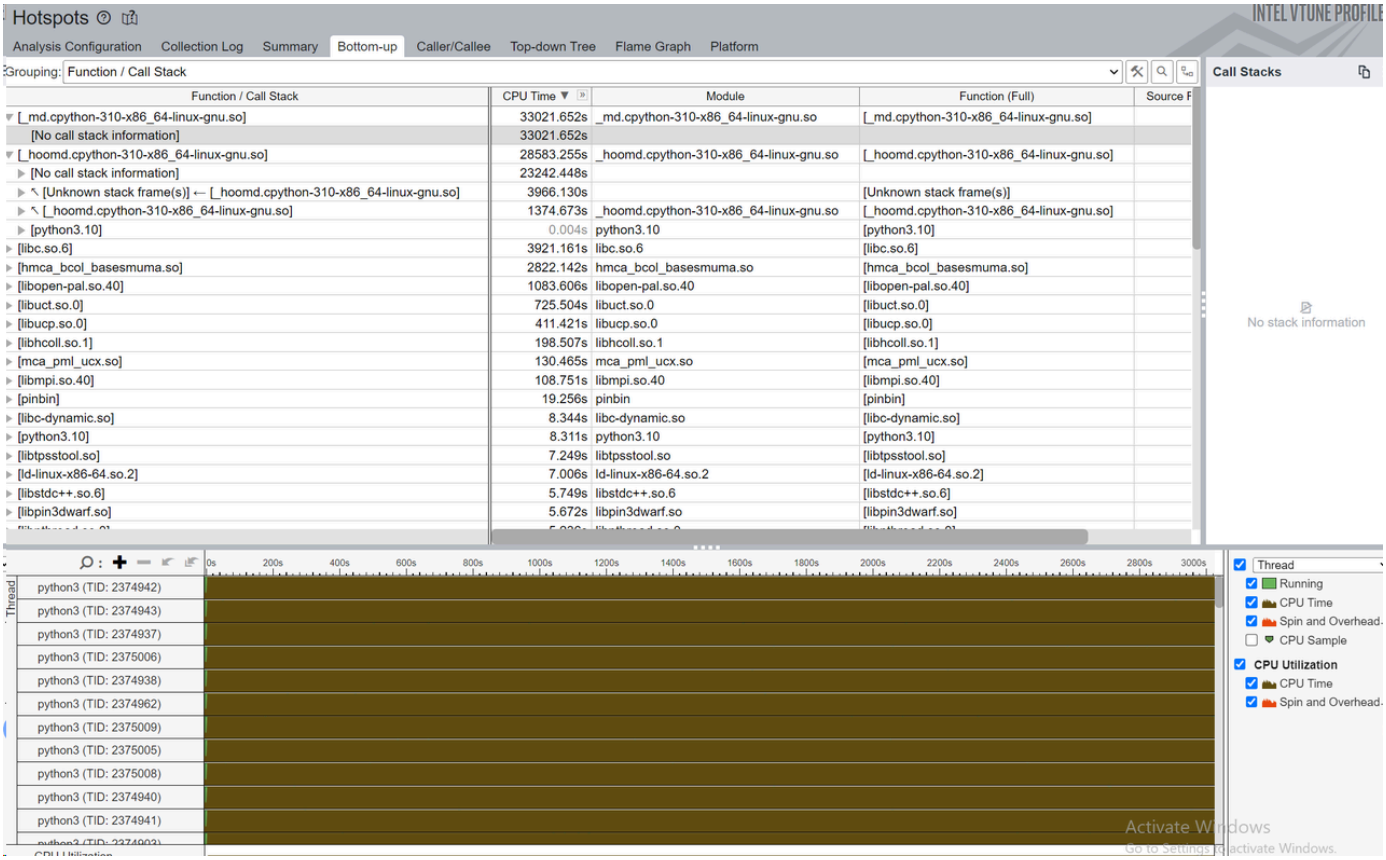




We can find that before 4 nodes, the program's performance scales up as the number of nodes increase. However after 4 nodes, the performance begins to slow down.

So what led to the slow down? We mainly consider on the communication and load imbalance of HOOMD-blue.

We first use Intel Vtune to see what is the bottleneck of the software on multi nodes. However, the profiler can only detect the cpython library and failed to reveal furthur call stack.



So we try to use IPM analysis to find out what happen to the communication.

IPM analysis

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

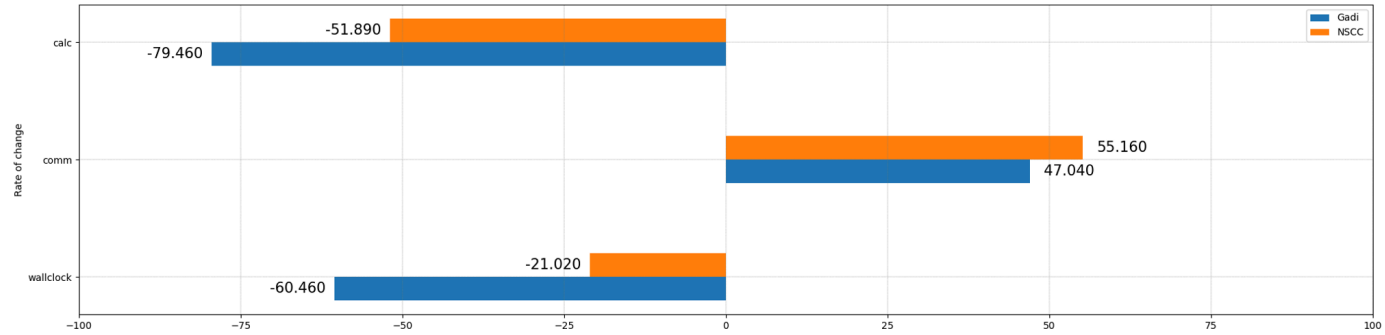
SUSTech Southern University of Science and Technology

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



The IPM analysis shows that the communication time increase from the ratio of 15% to nearly 55%. So the communication time counts for slowing down the performance.

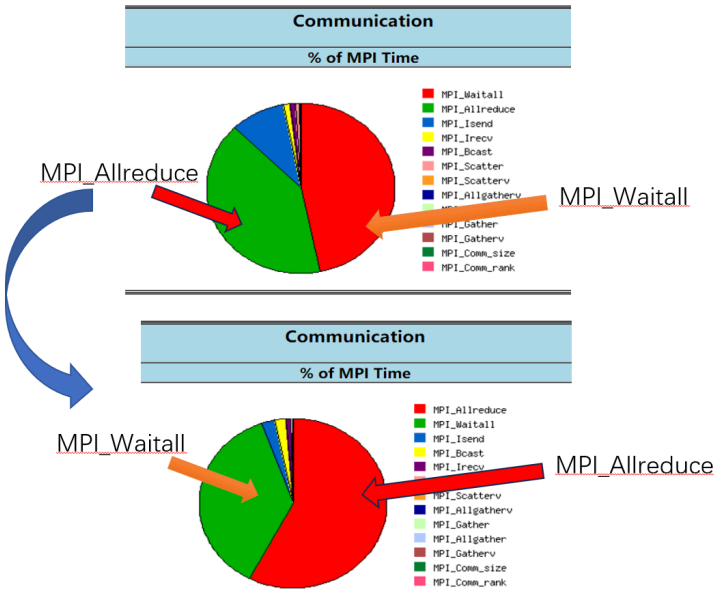
And here the figure shows that different clusters get different communication properties.

IPM analysis

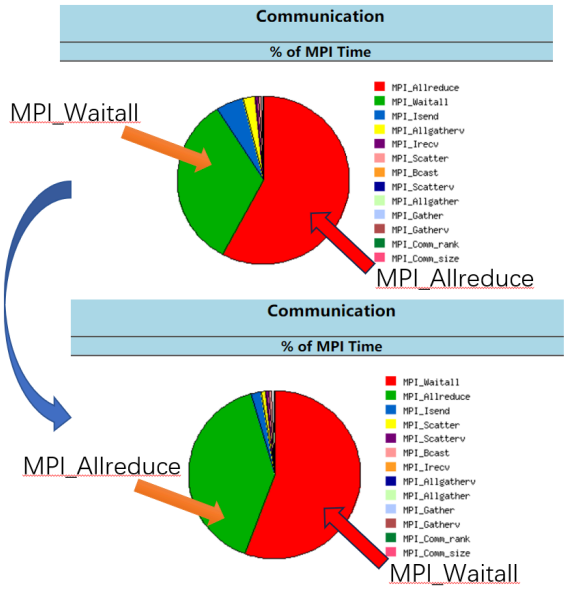
2 nodes -> 16nodes different clusters get different communication properties

SUSTech Southern University of Science and Technology

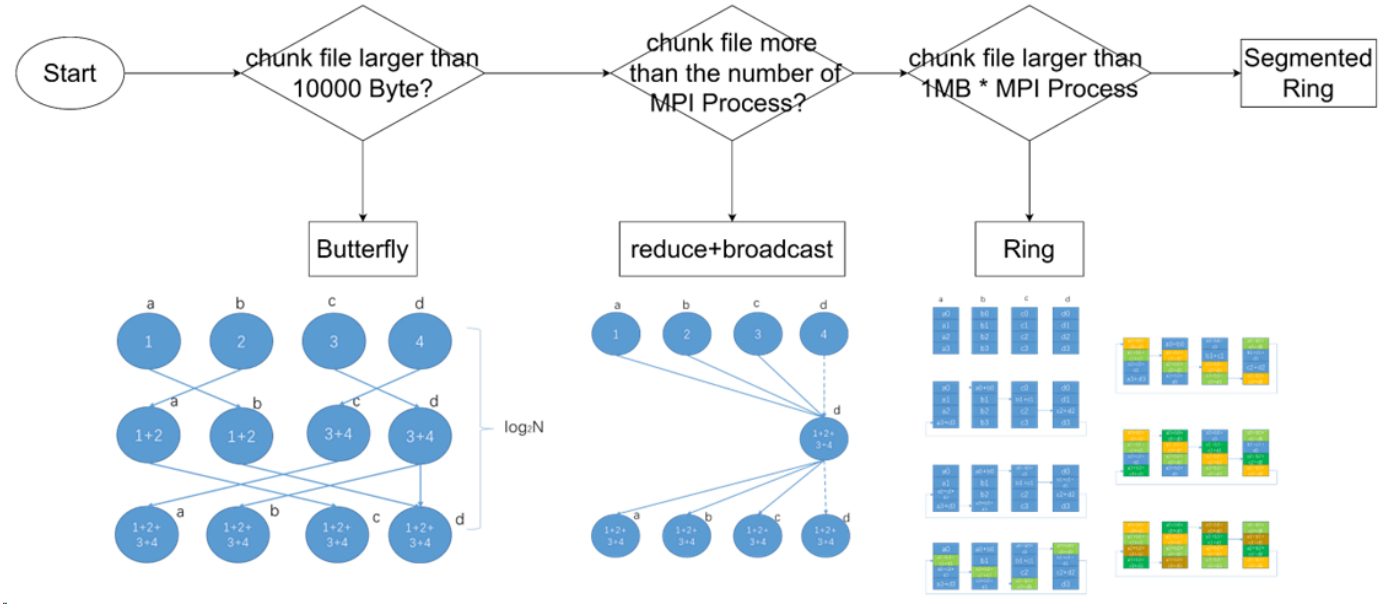
Baseline on Gadi (legend)



Baseline on NSCC



And these properties difference may be due to the strategy of MPI_Allreduce that MPI choosing. The figure shows different strategy that OpenMPI choose for different chunk file on `MPI_AllReduce` Primitive.

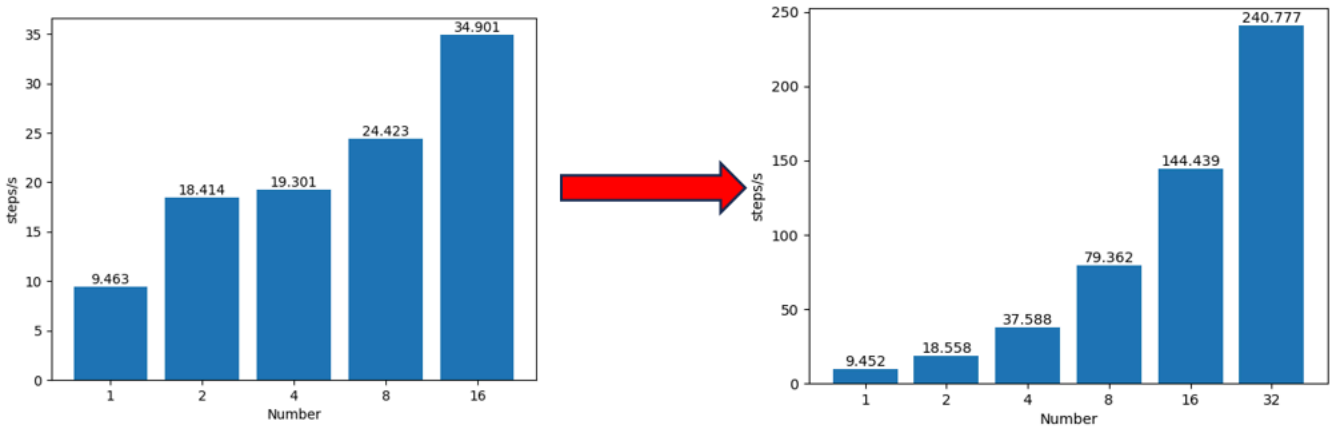


As we can see, the OpenMPI will choose different algorithm based on the size and the number of chunk file. When the file is small, a butterfly algorithm for different nodes will be called. When the file is larger and less than MPI Process, a Reduce&Broadcast algorithm will be called. If the number is larger, a Ring algorithm and even a segmented ring algorithm will be called.

So after this profiling, we think it may be the strategy that OpenMPI used may not suits best for the HOOMD communication.

HPC-X profiling

To profile this, we use HPC-X to select a better strategy for communication. HPC-X™ is a comprehensive software package that includes MPI, SHMEM and UPC communications libraries. It integrates **MPI** (specifically **OpenMPI** and **Intel MPI**) to optimize message passing in distributed computing.By only changing that, the performance on QiMing get 413.8% improvement on 16 nodes.



34.901 144.439

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

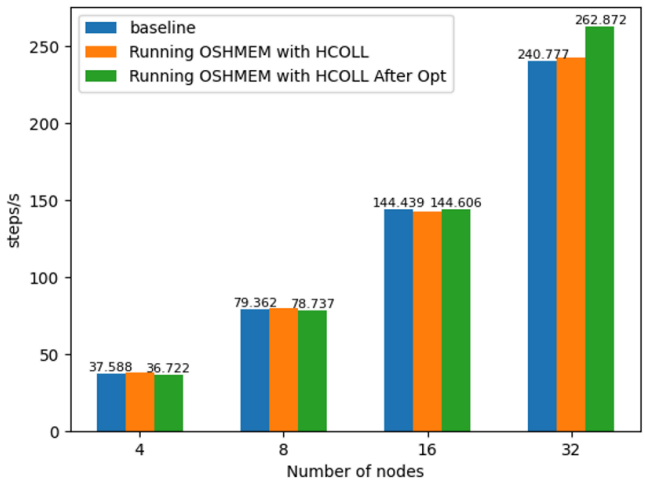
Here we try to find the best params from 32 nodes with each node 24 nodes:

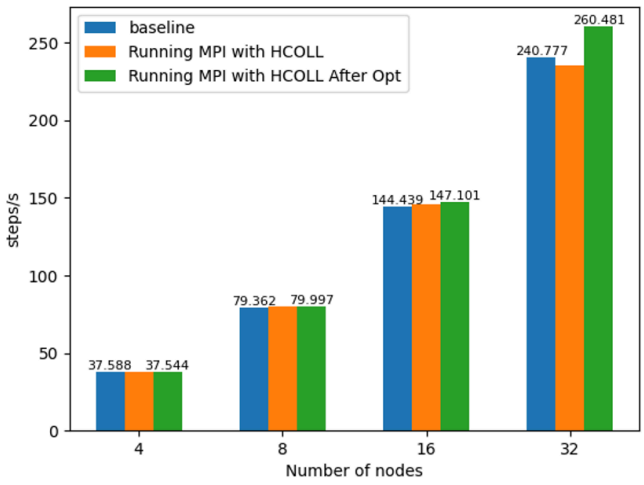
```
--mca coll_hcoll_enable 1
--x HCOLL_ENABLE_SHARP=1 (Correlated to Hardware Switch)
```

```
--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
```

And now we are trying to select better parameters for HPC-X. We choose SHArP to enable smart switch, P2P_Large AllReduce Algorithm etc.

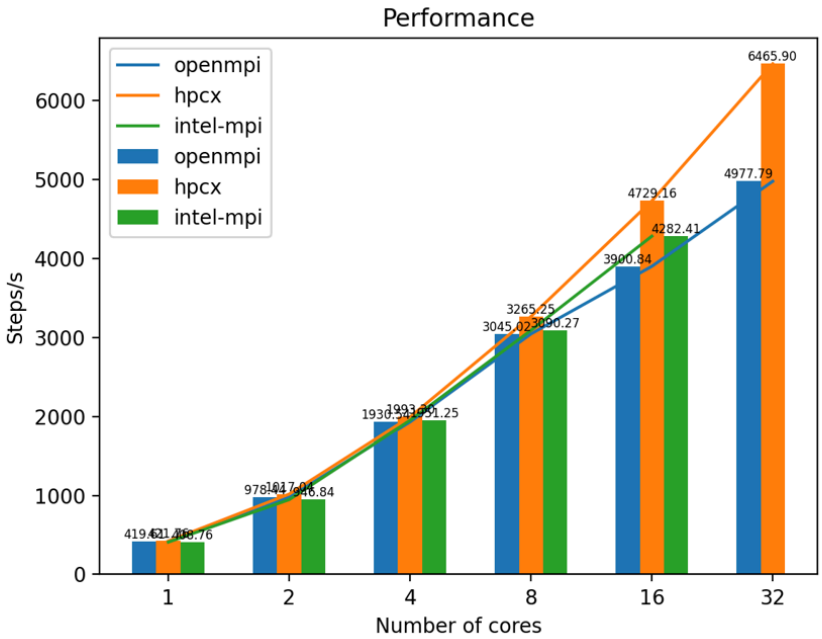
And here we get another 9% improving with OSHMEM and 8% for MPI.





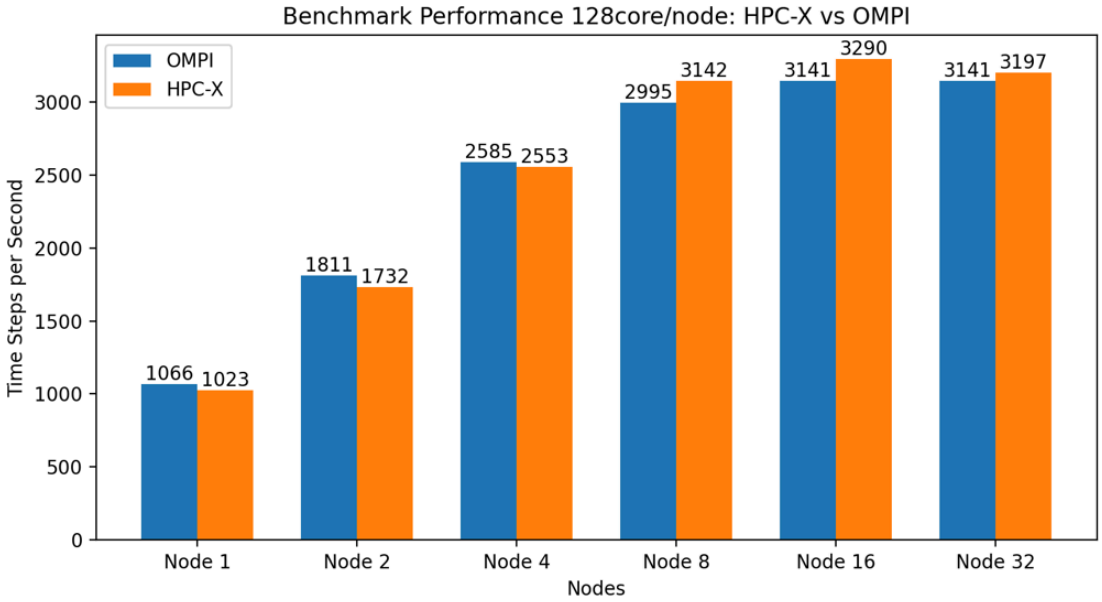
The performance comparison between HPC-X , IntelMPI and OpenMPI shows that HPC-X delivers superior performance and greater scalability with the same number of cores.
And we hope rebuilding OpenMPI with HPC-X can help us make a better decision on improving the Collective Communication Algorithm.

Here HPC-X gives 29.89% improvement for 32 nodes.



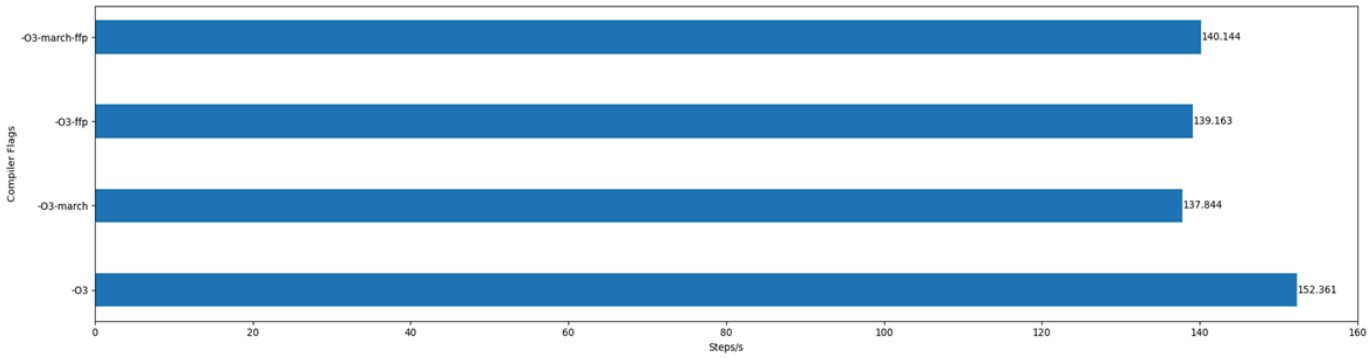
However for NSCC, HPC-X does not get significant improvement in NSCC. It only cause 1.78% improvement for 32

nodes. We gauss that it may be the cause of Hardware like the bandwith limit from infiniband.



O3 Compilation

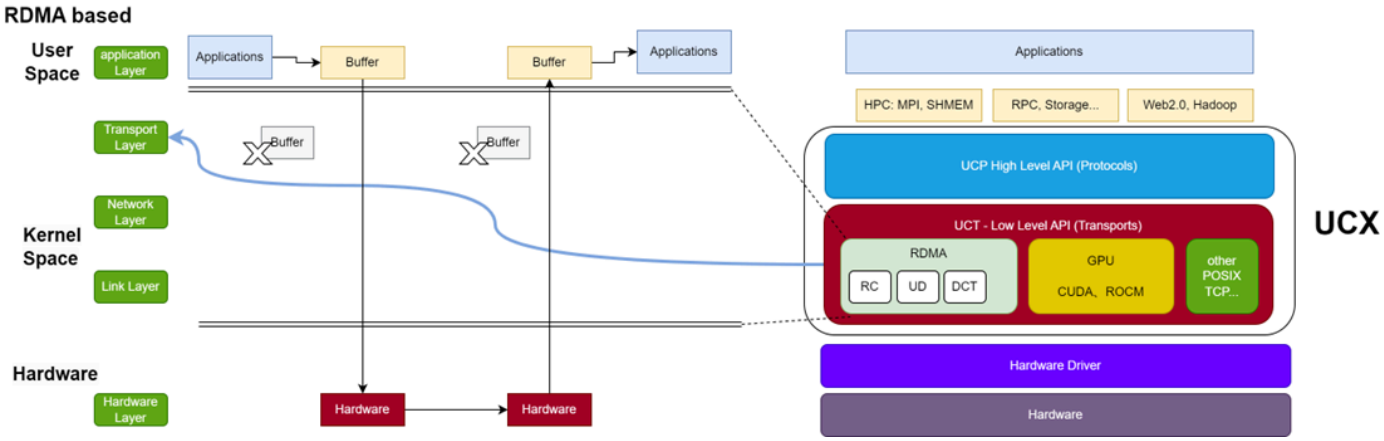
Next we use O3 compiler flags to get 5 more percent optimization. O3 flags will help do optimizations like **Loop unrolling**, **Vectorization** and **Instruction reordering**. Also, we try to use `-march=native` which would generate code optimized for CPU microarchitecture, enabling features like **AVX2**, **FMA** (Fused Multiply-Add), and other instruction sets on the nodes.



Optimal UCX params

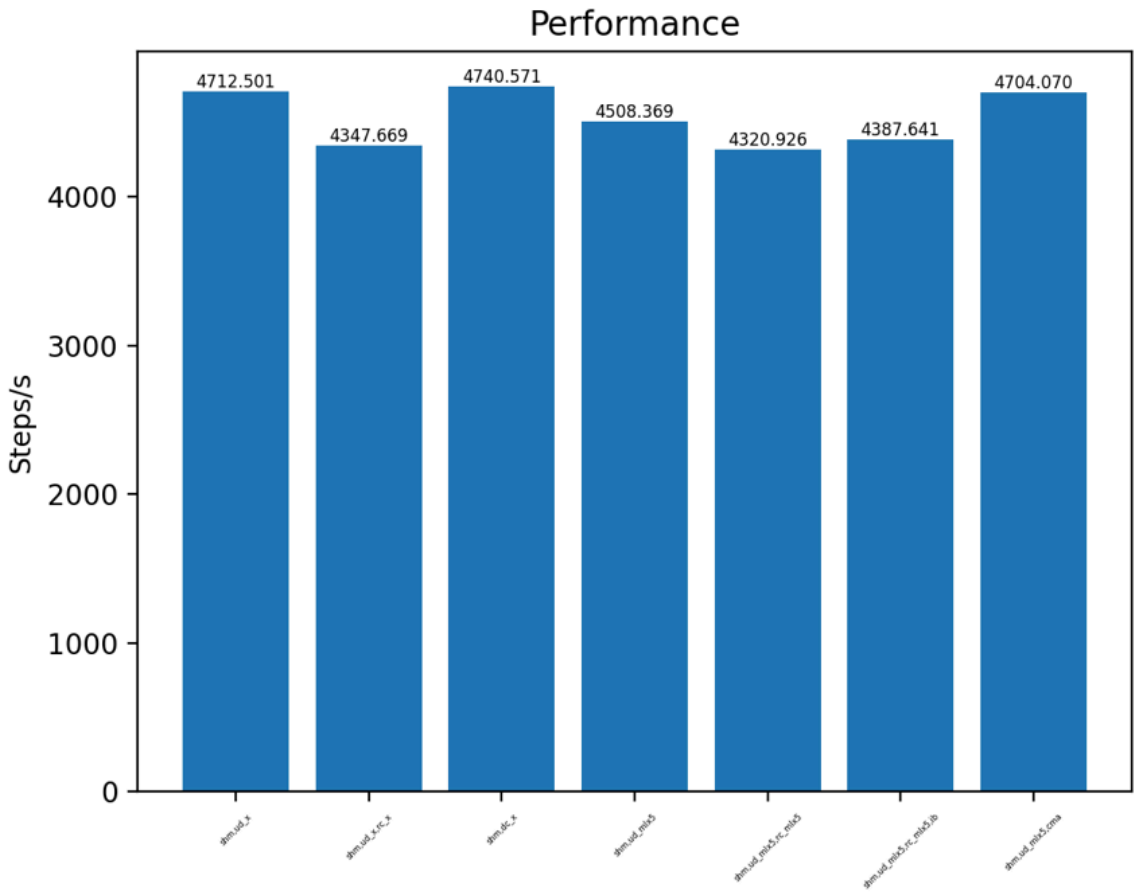
A core component of HPC-X is **UCX (Unified Communication X)**, which is a high-performance communication framework that supports multiple network interfaces, including InfiniBand, Ethernet, and other high-speed networks. UCX offers low-latency and high-bandwidth communication, making it suitable for large-scale parallel applications.

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.



We test multi combination of UCX_TLS to get better performance. Here we test UCX_TLS in bottleneck scale 16nodes.

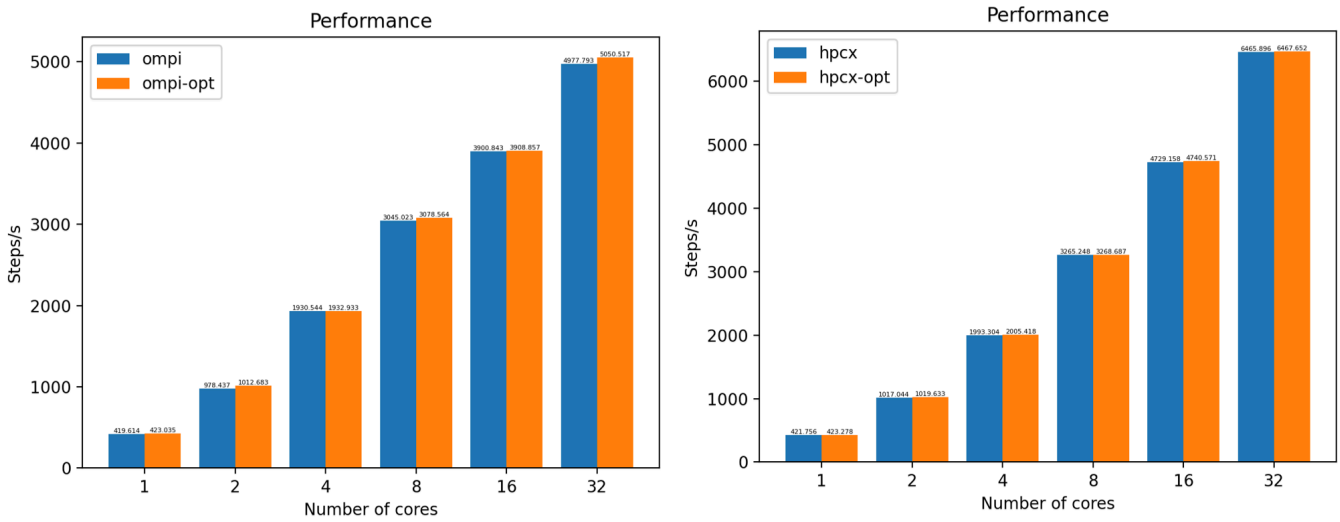
Params	Description
all	use all the available transports
sm	all shared memory transports
rc	reliable connection
ud	unreliable datagram transport
dc	Mellanox scalable offloaded dynamic con_x0002_nection transport



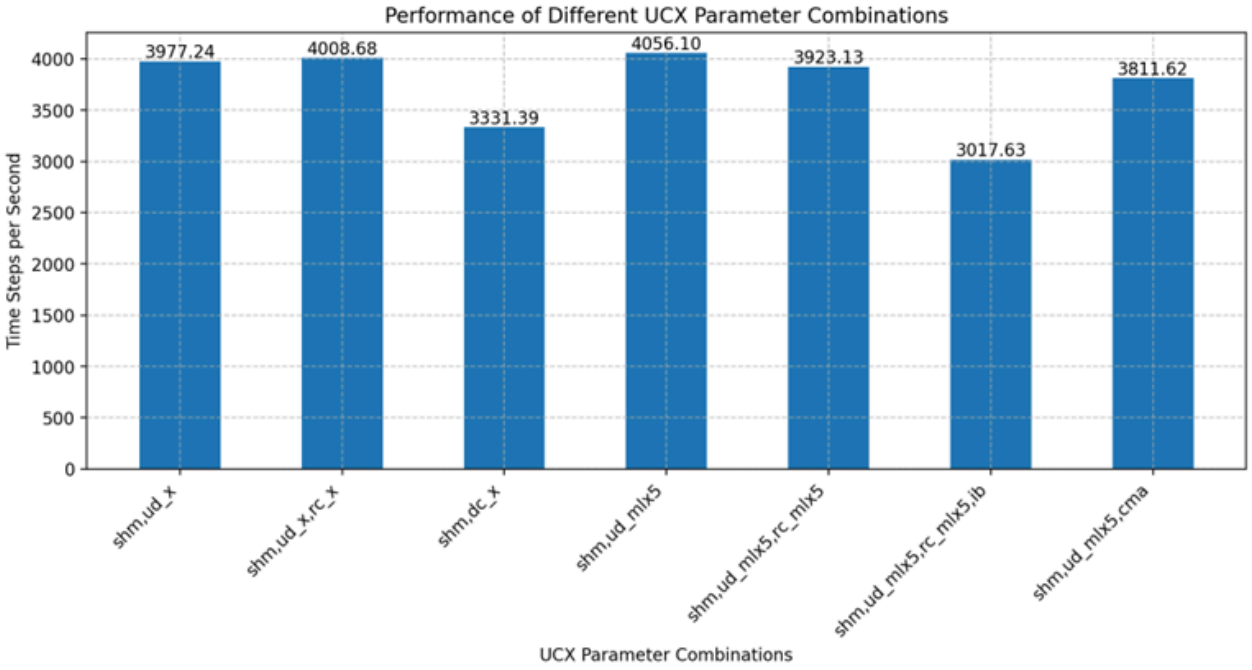
Here from the figure we find that `shm` combine with `dc_x` may best out perform the benchmark with 4740.571(the third one on performance. `dc_x` in UCX refers to the dynamic connected transport mode, used for reliable,

connection-oriented communication between nodes in a distributed system. It utilizes RDMA like InfiniBand for high-throughput, low-latency communication, with direct memory access and minimal CPU involvement.

However, `shm` combine with `dc_x` does not have much improvements in both ompi and hpcx.

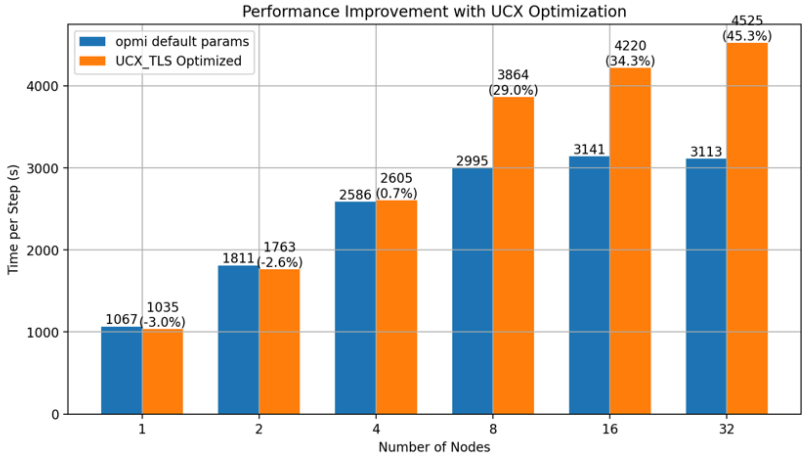


And as for NSCC, we test multi combination of UCX_TLS to get better performance.



And the best parameter is `shm` combine with `ud_mlx5`. `ud_mlx5` is a transport mode in UCX that enables connectionless communication (UD) using the Mellanox mlx5 driver, leveraging RDMA capabilities for low-latency, high-throughput data transfer.

Here we get 45.3% performance improvement with ompi and 44.0% with hpcx in 32 nodes. UCX gets high improvement for NSCC cluster with different parameter.



Best ucx params in ompi

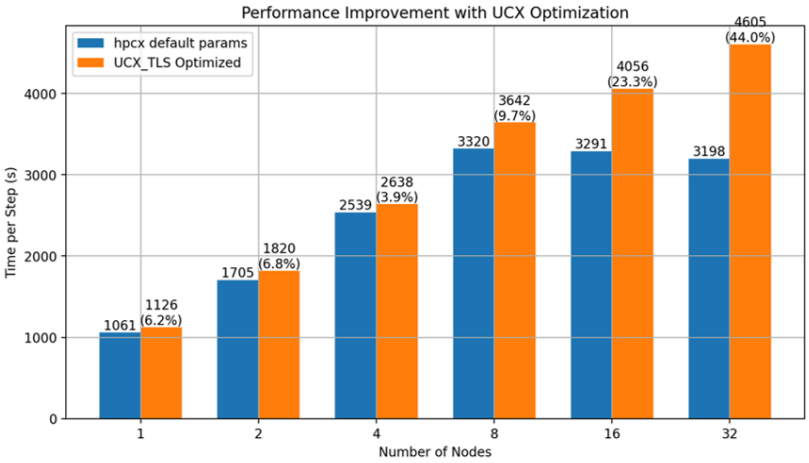
shm, ud_mlx5 get up to 45% improvement in ompi

45.3% improvement for 32 nodes

Best ucx params in hpcx

shm, ud_mlx5 get up to 44% improvement in hpcx

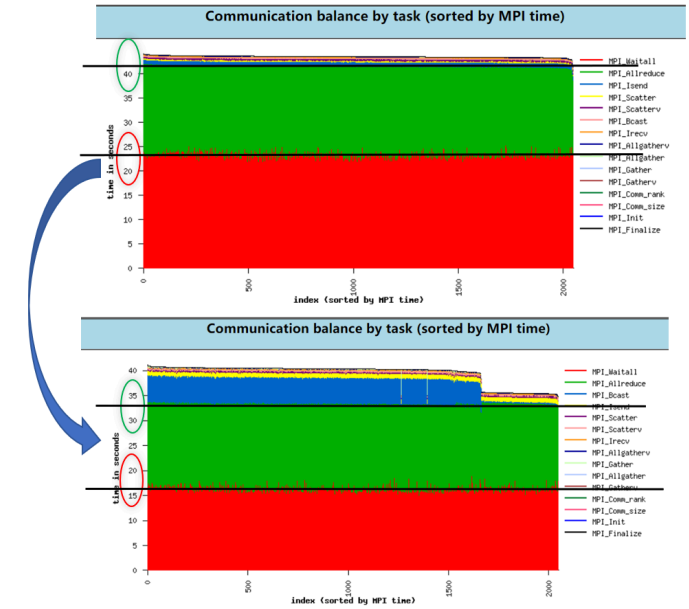
44.0% improvement for 32 nodes



node	1	2	4	8	16	32
openmpi	1066.905	1811.371	2585.888	2995.167	3141.322	3113.135
ompi-ucx	1034.617	1763.496	2605.129	3863.882	4219.513	4524.77
hpcx	1061.069	1704.753	2538.527	3320.069	3290.841	3197.979
hpcx-ucx	1126.378	1819.964	2637.603	3641.610	4056.101	4604.616

The reason may due to a faster communication between nodes. With less buffer on UCX framework and high affinity with Hardware, the time of `MPI_Waitall` dropped, and the time in `MPI_AllReduce` dropped significantly too.

UCX_TLS improvement analysis



IPM analysis

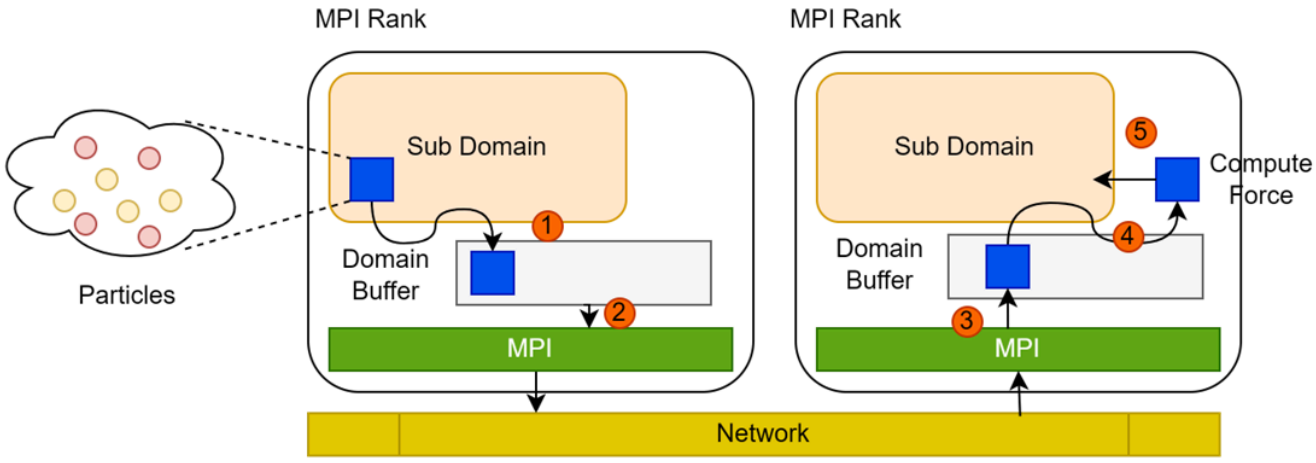
From defualt → optimal ucx params we get improvement in communication bottleneck

Waitall: ≈23 → ≈16

Allreduce: >40 → ≈33

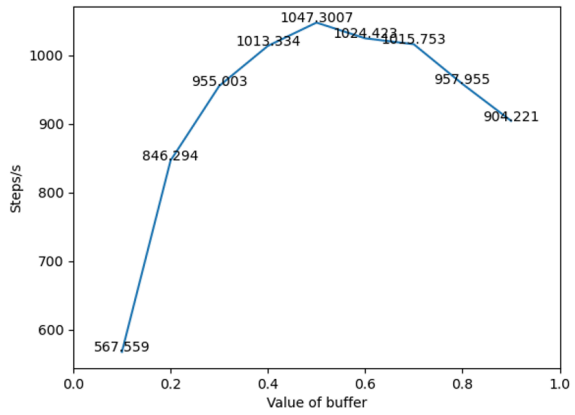
Better buffer parameters

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.

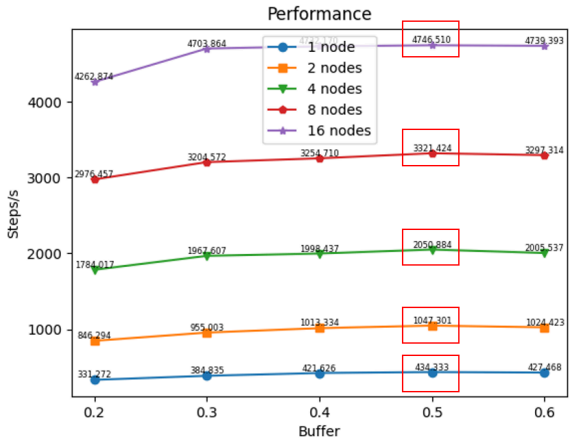


Here we test different buffer across nodes to find out which value of buffer suits best.

Better buffer params(gadi)



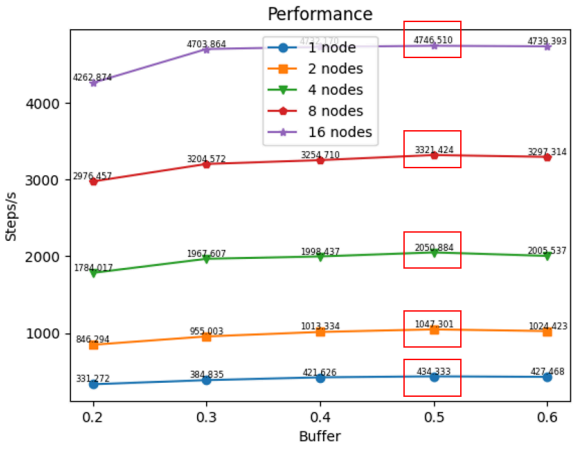
Testing buffer in bottleneck scale 2 nodes



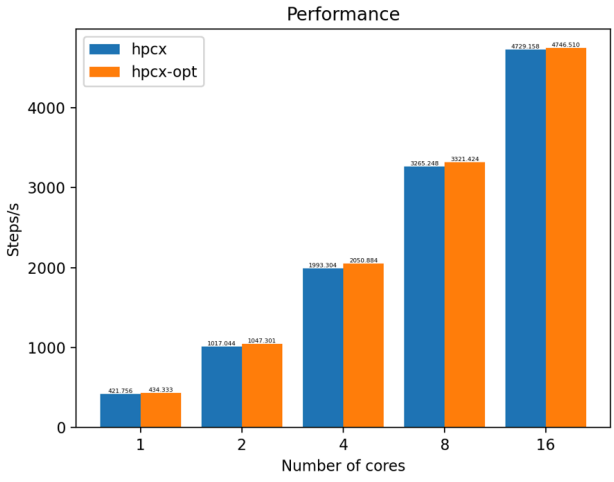
For more tests according to buffer

buffer	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1node	506.095	731.806	880.917	1071.763	1079.723	1042.2876	1180.068	1161.815	1073.476	1012.111
2node	830.841	1123.111	1586.330	1746.411	1742.075	1851.083	1792.209	1702.659	1599.901	1547.372
4node	1349.643	1853.253	2551.631	2760.421	2688.182	2703.875	2683.200	2577.528	2345.347	2164.911
8node	2099.468	2710.481	3481.783	3832.302	3814.768	3820.551	3709.626	3541.606	err	err
16node	2420.985	3006.0644	3839.213	4200.254	3931.873	4061.711	3718.369	3723.109	error	error
32node	3409.979	4186.882	4980.921	5410.716	5221.058	5332.792	4987.663	4927.151	error	error

Best buffer params in hpcx



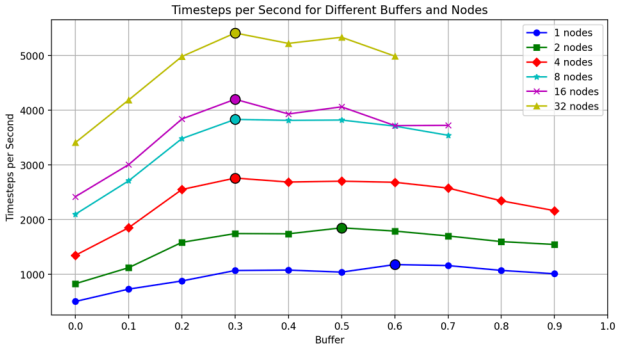
For more tests according to buffer



0.37% improvement for 32 nodes

buffer params(NSCC)

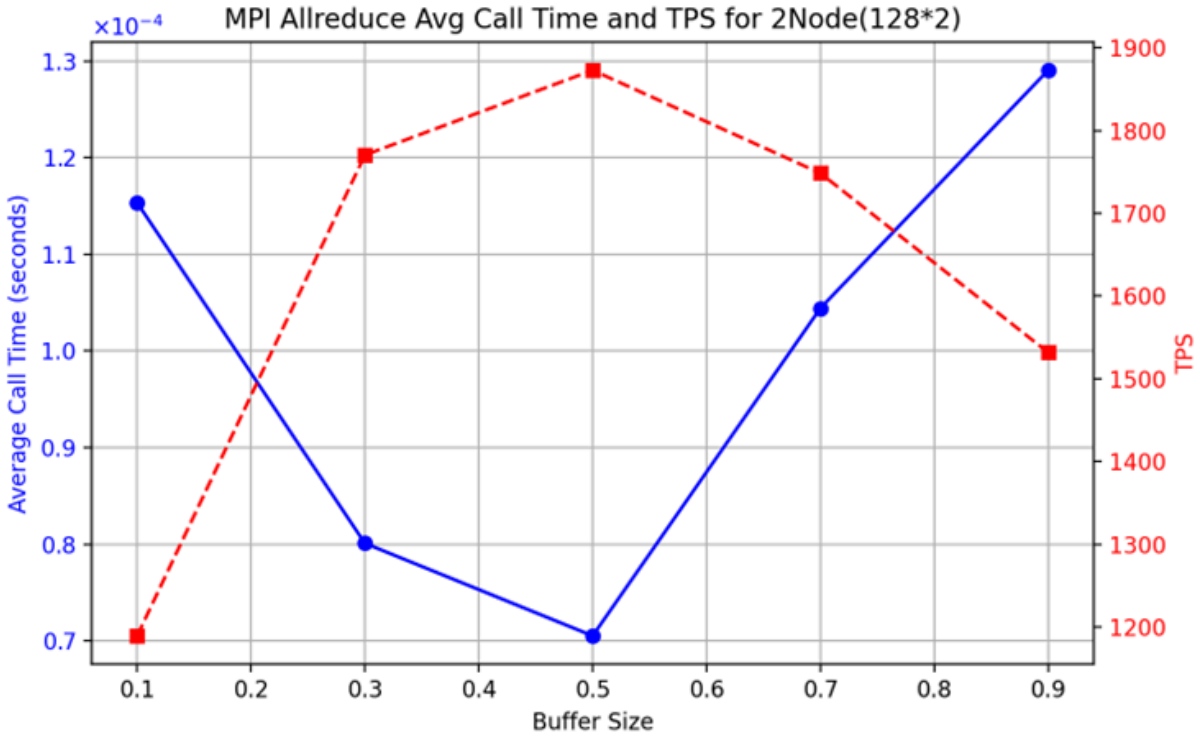
buffer is a key params in domain composition parallel algorithm



4.86% improvement for 32 nodes

N-list updated are infected by buffer params

```
if (needsUpdating(timestep))
    {
    bool overflowed = false;
    do
        {
        buildNlist(timestep);
        overflowed = checkConditions();
        if (overflowed)
            ...
        } while (overflowed);
    ..
}
```



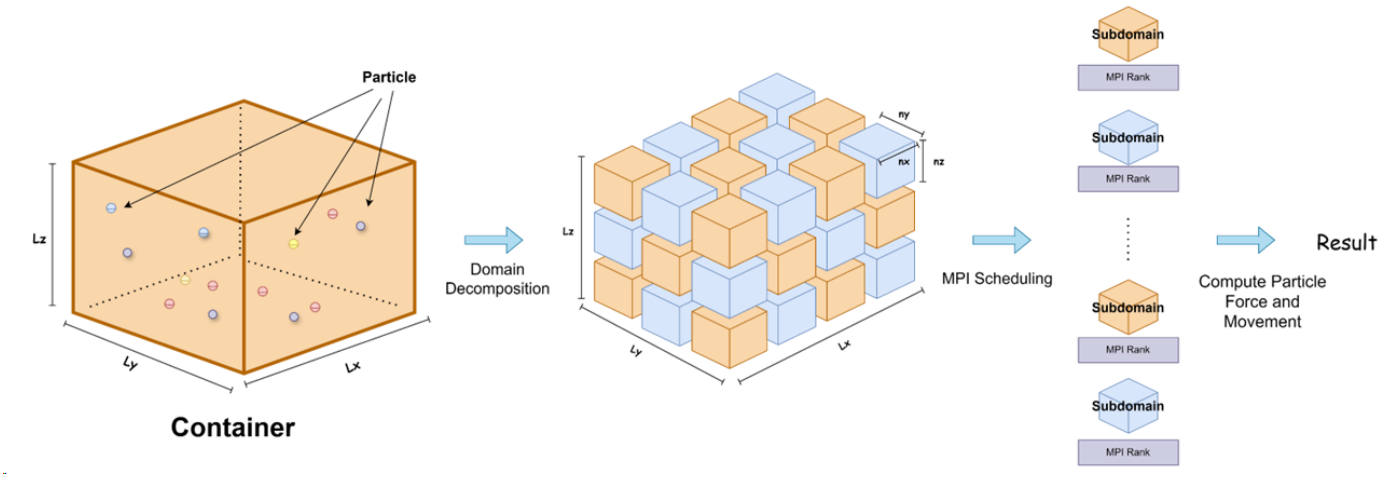
Single Node Optimization Model Formulation

As we mentioned before, HOOMD-blue use a technique called `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.

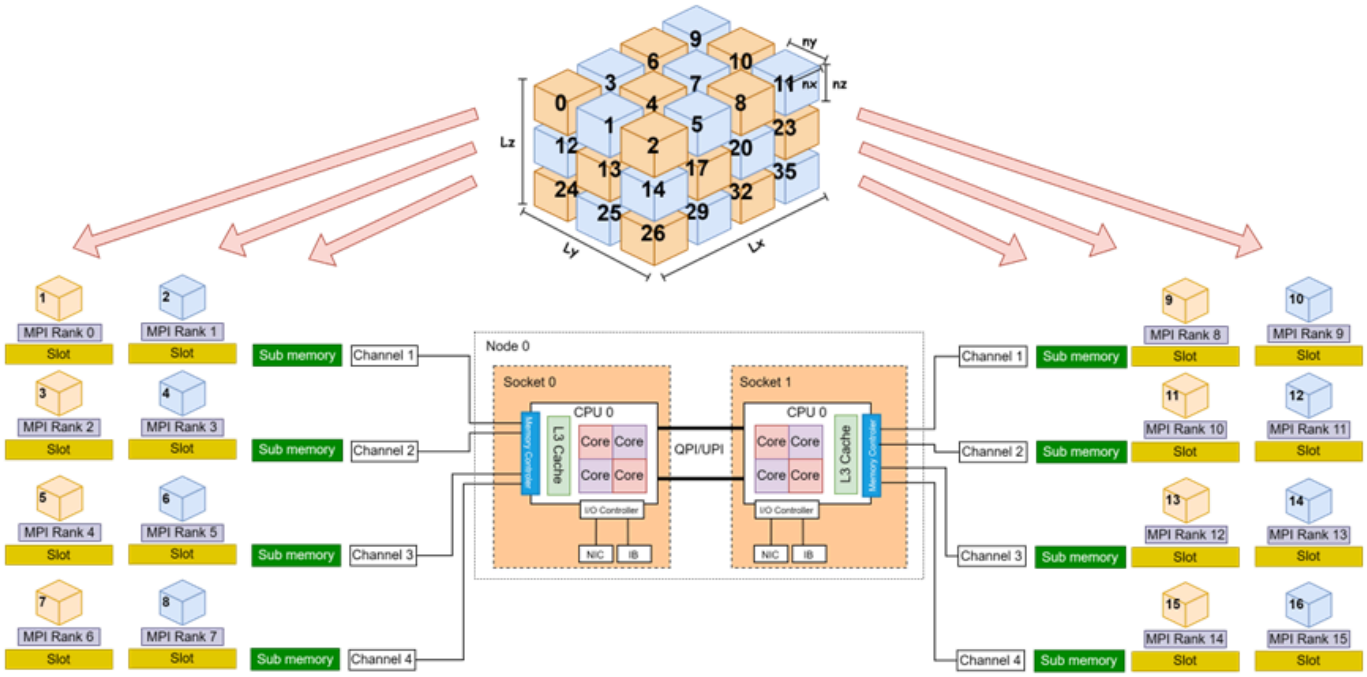
When you create the State object in an MPI simulation on more than 1 rank, HOOMD-blue splits the simulation box into `k x l x m` domains. The product of k, l and m is equal to the number of ranks you execute. The domains are defined by planes that split the box. By default, the planes are evenly spaced and chosen to **minimize the surface area between the domains**.

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



Computations like pair forces in MD or hard particle overlap checks in HPMC, need to compute interactions with particles from a neighboring domain. This establishes a lower limit on the domain size. For MD, this is the sum of the largest pair potential r_cut and the neighbor list `buffer` .

So here we have a problem on NUMA node scheduling. As we know the cluster is a NUMA achitecture and the data and computation will be seperate to different computation slot. While when different MPI ranks are communicating, the distance cause by NUMA will lead to communication effort. So how to assign each MPI rank to each slot, so that the communication between different NUMA socket will be minimize?



Here we try to solve the problem with mathematical modeling as following

Objective

Minimize the deviation from the target number of neighboring domains on the same NUMA node:

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

Where:

- Deviation is the difference between the actual and target number of neighboring domains on the same NUMA node.

Constraints

1. **Deviation Constraint:**

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

2. **Each NUMA node can bind at most 16 processes:**

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

3. **Each process is bound to exactly one NUMA node and two logical cores:**

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

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

4. **Logical cores are paired (hyper-threading):**

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

5. **NUMA and logical core mapping:**

$$\sum_{j \in \text{numa\_node\_mappings}[k]} y[i,j] = 2 \cdot x[i,k] \quad \forall i \in \{0, \ldots, P-1\}, \forall k \in \{0, \ldots, K-1\}$$

6. **Each logical core can bind at most one process:**

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

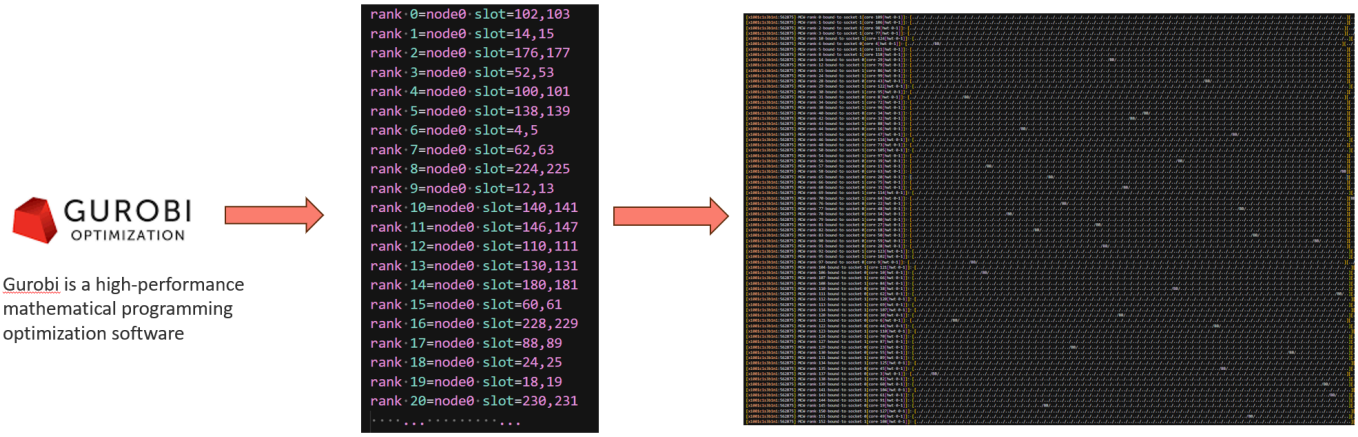
7. **NUMA node usage indicator:**

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

Variables

- ($x[i,k]$): Binary variable indicating if process (i) is bound to NUMA node (k).
- ($y[i,j]$): Binary variable indicating if process (i) is bound to logical core (j).
- ($z[k]$): Binary variable indicating if NUMA node (k) is used.

we use Gurobi, a mathematical programming optimization software for non-linear problems to solve the modeling problem. Then we gengerate new MPI rankfile that will assign the ranks to slot.



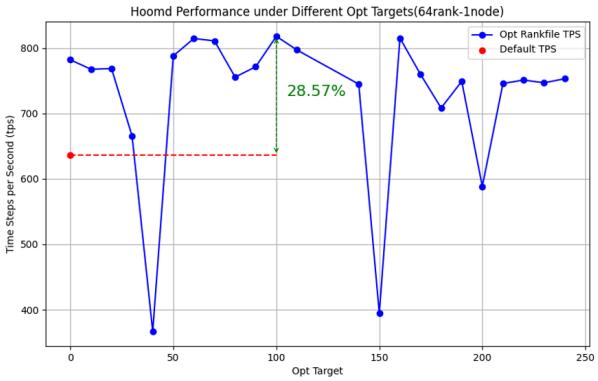
- General solver
- Rankfile format
- Core Binding Strategy in Openmpi

This model aims to optimize the binding of processes to NUMA nodes and logical cores within a single node, minimizing the deviation from the target adjacency.

And by scheduling, we make 28.57% improvement for one node on NSCC with 64 ranks. And for 128 ranks we get 7.97% improvement. However sometimes the benchmark will drop down than the baseline. This may because the non-linear model is complicated to solve and a worse scheduling method comes before Gurobi finds better solution.

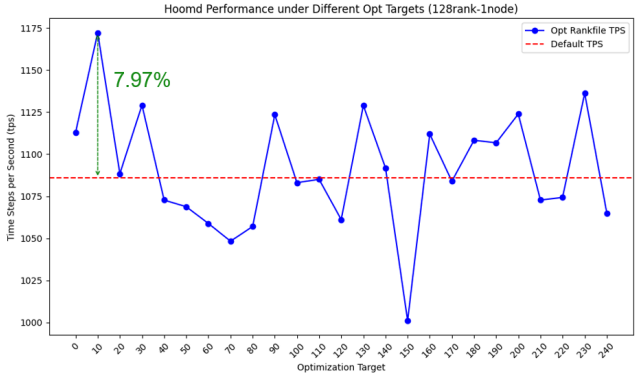
Improvement of optimization (single node)

SUSTech Southern University of Science and Technology



- 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

And since here we get information for different ranks and their neighbors, we believe that our NUMA aware scheduling may extend to multinode with more sufficient learning.

Works for different numbers of ranks

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
```

Also, we explore and get some knowledge on PAC, a ICS' 23 paper. It's a scheduler on a NUMA node with different CPUs, which lead to a harder modeling problem. And we think we can furthur explore that part to enhance our understanding on NUMA.

Summary

So in summary, we take different optimization on Qiming2.0, Gadi and NSCC. The chart shows our optimization and our future work.

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.

And that will be the end of our HPC profiling.

AI Llama 2

LitGPT Llama2 is a specialized integration of Llama 2, a powerful language model developed by Meta. With LitGPT, a framework designed to enable fine-tuning and customization of large language models for specific tasks or domains.

Testing Platform

Test Cluster Configuration - NSCC (GPU)

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