

APAC HPC-AI 2024 Final Presentation

SUSTech-0x01

5 Noverber 2024



About us



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

- HOOMD-blue
- Lightgpt-llama2
- Summary

Part 1

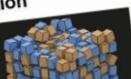
High-Performance Computing

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



Contents

Hoomd Description



```

1 #!/usr/bin/python
2
3 import hoomd
4
5 # Create a simulation
6 simulation = hoomd.Simulation('lattice')
7
8 # Set up a lattice
9 simulation.create_lattice(cubic=True, n=10)
10
11 # Add particles
12 simulation.add_particles(n=100, type='A', pos=[0,0,0])
13 simulation.add_particles(n=100, type='B', pos=[0,0,0])
14
15 # Set up a bond force
16 bond_type = hoomd.bonds.BondLJ()
17 bond_type.set_params(r_min=0.8, r_max=1.2, epsilon=0.01)
18
19 # Add bonds
20 simulation.add_bonds(bond_type=bond_type, pairs=[[0,1], [1,2], [2,3], [3,4], [4,5], [5,6], [6,7], [7,8], [8,9], [9,0], [0,5], [5,10], [10,11], [11,12], [12,13], [13,14], [14,15], [15,16], [16,17], [17,18], [18,19], [19,20], [20,0], [0,15], [15,21], [21,22], [22,23], [23,24], [24,25], [25,26], [26,27], [27,28], [28,29], [29,30], [30,31], [31,32], [32,33], [33,34], [34,35], [35,36], [36,37], [37,38], [38,39], [39,40], [40,41], [41,42], [42,43], [43,44], [44,45], [45,46], [46,47], [47,48], [48,49], [49,50], [50,51], [51,52], [52,53], [53,54], [54,55], [55,56], [56,57], [57,58], [58,59], [59,60], [60,61], [61,62], [62,63], [63,64], [64,65], [65,66], [66,67], [67,68], [68,69], [69,70], [70,71], [71,72], [72,73], [73,74], [74,75], [75,76], [76,77], [77,78], [78,79], [79,80], [80,81], [81,82], [82,83], [83,84], [84,85], [85,86], [86,87], [87,88], [88,89], [89,90], [90,91], [91,92], [92,93], [93,94], [94,95], [95,96], [96,97], [97,98], [98,99], [99,0], [0,100], [100,101], [101,102], [102,103], [103,104], [104,105], [105,106], [106,107], [107,108], [108,109], [109,110], [110,111], [111,112], [112,113], [113,114], [114,115], [115,116], [116,117], [117,118], [118,119], [119,120], [120,121], [121,122], [122,123], [123,124], [124,125], [125,126], [126,127], [127,128], [128,129], [129,130], [130,131], [131,132], [132,133], [133,134], [134,135], [135,136], [136,137], [137,138], [138,139], [139,140], [140,141], [141,142], [142,143], [143,144], [144,145], [145,146], [146,147], [147,148], [148,149], [149,150], [150,151], [151,152], [152,153], [153,154], [154,155], [155,156], [156,157], [157,158], [158,159], [159,160], [160,161], [161,162], [162,163], [163,164], [164,165], [165,166], [166,167], [167,168], [168,169], [169,170], [170,171], [171,172], [172,173], [173,174], [174,175], [175,176], [176,177], [177,178], [178,179], [179,180], [180,181], [181,182], [182,183], [183,184], [184,185], [185,186], [186,187], [187,188], [188,189], [189,190], [190,191], [191,192], [192,193], [193,194], [194,195], [195,196], [196,197], [197,198], [198,199], [199,200], [200,201], [201,202], [202,203], [203,204], [204,205], [205,206], [206,207], [207,208], [208,209], [209,210], [210,211], [211,212], [212,213], [213,214], [214,215], [215,216], [216,217], [217,218], [218,219], [219,220], [220,221], [221,222], [222,223], [223,224], [224,225], [225,226], [226,227], [227,228], [228,229], [229,230], [230,231], [231,232], [232,233], [233,234], [234,235], [235,236], [236,237], [237,238], [238,239], [239,240], [240,241], [241,242], [242,243], [243,244], [244,245], [245,246], [246,247], [247,248], [248,249], [249,250], [250,251], [251,252], [252,253], [253,254], [254,255], [255,256], [256,257], [257,258], [258,259], [259,260], [260,261], [261,262], [262,263], [263,264], [264,265], [265,266], [266,267], [267,268], [268,269], [269,270], [270,271], [271,272], [272,273], [273,274], [274,275], [275,276], [276,277], [277,278], [278,279], [279,280], [280,281], [281,282], [282,283], [283,284], [284,285], [285,286], [286,287], [287,288], [288,289], [289,290], [290,291], [291,292], [292,293], [293,294], [294,295], [295,296], [296,297], [297,298], [298,299], [299,200], [200,201], [201,202], [202,203], [203,204], [204,205], [205,206], [206,207], [207,208], [208,209], [209,2010], [2010,2011], [2011,2012], [2012,2013], [2013,2014], [2014,2015], [2015,2016], [2016,2017], [2017,2018], [2018,2019], [2019,2010], [2010,2020], [2020,2021], [2021,2022], [2022,2023], [2023,2024], [2024,2025], [2025,2026], [2026,2027], [2027,2028], [2028,2029], [2029,2020], [2020,2030], [2030,2031], [2031,2032], [2032,2033], [2033,2034], [2034,2035], [2035,2036], [2036,2037], [2037,2038], [2038,2039], [2039,2030], [2030,2040], [2040,2041], [2041,2042], [2042,2043], [2043,2044], [2044,2045], [2045,2046], [2046,2047], [2047,2048], [2048,2049], [2049,2040], [2040,2050], [2050,2051], [2051,2052], [2052,2053], [2053,2054], [2054,2055], [2055,2056], [2056,2057], [2057,2058], [2058,2059], [2059,2050], [2050,2060], [2060,2061], [2061,2062], [2062,2063], [2063,2064], [2064,2065], [2065,2066], [2066,2067], [2067,2068], [2068,2069], [2069,2060], [2060,2070], [2070,2071], [2071,2072], [2072,2073], [2073,2074], [2074,2075], [2075,2076], [2076,2077], [2077,2078], [2078,2079], [2079,2070], [2070,2080], [2080,2081], [2081,2082], [2082,2083], [2083,2084], [2084,2085], [2085,2086], [2086,2087], [2087,2088], [2088,2089], [2089,2080], [2080,2090], [2090,2091], [2091,2092], [2092,2093], [2093,2094], [2094,2095], [2095,2096], [2096,2097], [2097,2098], [2098,2099], [2099,2080], [2080,20100], [20100,20101], [20101,20102], [20102,20103], [20103,20104], [20104,20105], [20105,20106], [20106,20107], [20107,20108], [20108,20109], [20109,20100], [20100,20110], [20110,20111], [20111,20112], [20112,20113], [20113,20114], [20114,20115], [20115,20116], [20116,20117], [20117,20118], [20118,20119], [20119,20100], [20100,20120], [20120,20121], [20121,20122], [20122,20123], [20123,20124], [20124,20125], [20125,20126], [20126,20127], [20127,20128], [20128,20100], [20100,20130], [20130,20131], [20131,20132], [20132,20133], [20133,20134], [20134,20135], [20135,20136], [20136,20137], [20137,20138], [20138,20100], [20100,20140], [20140,20141], [20141,20142], [20142,20143], [20143,20144], [20144,20145], [20145,20146], [20146,20147], [20147,20148], [20148,20100], [20100,20150], [20150,20151], [20151,20152], [20152,20153], [20153,20154], [20154,20155], [20155,20156], [20156,20157], [20157,20158], [20158,20100], [20100,20160], [20160,20161], [20161,20162], [20162,20163], [20163,20164], [20164,20165], [20165,20166], [20166,20167], [20167,20168], [20168,20100], [20100,20170], [20170,20171], [20171,20172], [20172,20173], [20173,20174], [20174,20175], [20175,20176], [20176,20177], [20177,20178], [20178,20100], [20100,20180], [20180,20181], [20181,20182], [20182,20183], [20183,20184], [20184,20185], [20185,20186], [20186,20187], [20187,20188], [20188,20100], [20100,20190], [20190,20191], [20191,20192], [20192,20193], [20193,20194], [20194,20195], [20195,20196], [20196,20197], [20197,20198], [20198,20100], [20100,20200], [20200,20201], [20201,20202], [20202,20203], [20203,20204], [20204,20205], [20205,20206], [20206,20207], [20207,20208], [20208,20100], [20100,20210], [20210,20211], [20211,20212], [20212,20213], [20213,20214], [20214,20215], [20215,20216], [20216,20217], [20217,20218], [20218,20100], [20100,20220], [20220,20221], [20221,20222], [20222,20223], [20223,20224], [20224,20225], [20225,20226], [20226,20227], [20227,20228], [20228,20100], [20100,20230], [20230,20231], [20231,20232], [20232,20233], [20233,20234], [20234,20235], [20235,20236], [20236,20237], [20237,20238], [20238,20100], [20100,20240], [20240,20241], [20241,20242], [20242,20243], [20243,20244], [20244,20245], [20245,20246], [20246,20247], [20247,20248], [20248,20100], [20100,20250], [20250,20251], [20251,20252], [20252,20253], [20253,20254], [20254,20255], [20255,20256], [20256,20257], [20257,20258], [20258,20100], [20100,20260], [20260,20261], [20261,20262], [20262,20263], [20263,20264], [20264,20265], [20265,20266], [20266,20267], [20267,20268], [20268,20100], [20100,20270], [20270,20271], [20271,20272], [20272,20273], [20273,20274], [20274,20275], [20275,20276], [20276,20277], [20277,20278], [20278,20100], [20100,20280], [20280,20281], [20281,20282], [20282,20283], [20283,20284], [20284,20285], [20285,20286], [20286,20287], [20287,20288], [20288,20100], [20100,20290], [20290,20291], [20291,20292], [20292,20293], [20293,20294], [20294,20295], [20295,20296], [20296,20297], [20297,20298], [20298,20100], [20100,20300], [20300,20301], [20301,20302], [20302,20303], [20303,20304], [20304,20305], [20305,20306], [20306,20307], [20307,20308], [20308,20100], [20100,20310], [20310,20311], [20311,20312], [20312,20313], [20313,20314], [20314,20315], [20315,20316], [20316,20317], [20317,20318], [20318,20100], [20100,20320], [20320,20321], [20321,20322], [20322,20323], [20323,20324], [20324,20325], [20325,20326], [20326,20327], [20327,20328], [20328,20100], [20100,20330], [20330,20331], [20331,20332], [20332,20333], [20333,20334], [20334,20335], [20335,20336], [20336,20337], [20337,20338], [20338,20100], [20100,20340], [20340,20341], [20341,20342], [20342,20343], [20343,20344], [20344,20345], [20345,20346], [20346,20347], [20347,20348], [20348,20100], [20100,20350], [20350,20351], [20351,20352], [20352,20353], [20353,20354], [20354,20355], [20355,20356], [20356,20357], [20357,20358], [20358,20100], [20100,20360], [20360,20361], [20361,20362], [20362,20363], [20363,20364], [20364,20365], [20365,20366], [20366,20367], [20367,20368], [20368,20100], [20100,20370], [20370,20371], [20371,20372], [20372,20373], [20373,20374], [20374,20375], [20375,20376], [20376,20377], [20377,20378], [20378,20100], [20100,20380], [20380,20381], [20381,20382], [20382,20383], [20383,20384], [20384,20385], [20385,20386], [20386,20387], [20387,20388], [20388,20100], [20100,20390], [20390,20391], [20391,20392], [20392,20393], [20393,20394], [20394,20395], [20395,20396], [20396,20397], [20397,20398], [20398,20100], [20100,20400], [20400,20401], [20401,20402], [20402,20403], [20403,20404], [20404,20405], [20405,20406], [20406,20407], [20407,20408], [20408,20100], [20100,20410], [20410,20411], [20411,20412], [20412,20413], [20413,20414], [20414,20415], [20415,20416], [20416,20417], [20417,20418], [20418,20100], [20100,20420], [20420,20421], [20421,20422], [20422,20423], [20423,20424], [20424,20425], [20425,20426], [20426,20427], [20427,20428], [20428,20100], [20100,20430], [20430,20431], [20431,20432], [20432,20433], [20433,20434], [20434,20435], [20435,20436], [20436,20437], [20437,20438], [20438,20100], [20100,20440], [20440,20441], [20441,20442], [20442,20443], [20443,20444], [20444,20445], [20445,20446], [20446,20447], [20447,20448], [20448,20100], [20100,20450], [20450,20451], [20451,20452], [20452,20453], [20453,20454], [20454,20455], [20455,20456], [20456,20457], [20457,20458], [20458,20100], [20100,20460], [20460,20461], [20461,20462], [20462,20463], [20463,20464], [20464,20465], [20465,20466], [20466,20467], [20467,20468], [20468,20100], [20100,20470], [20470,20471], [20471,20472], [20472,20473], [20473,20474], [20474,20475], [20475,20476], [20476,20477], [20477,20478], [20478,20100], [20100,20480], [20480,20481], [20481,20482], [20482,20483], [20483,20484], [20484,20485], [20485,20486], [20486,20487], [20487,20488], [20488,20100], [20100,20490], [20490,20491], [20491,20492], [20492,20493], [20493,20494], [20494,20495], [20495,20496], [20496,20497], [20497,20498], [20498,20100], [20100,20500], [20500,20501], [20501,20502], [20502,20503], [20503,20504], [20504,20505], [20505,20506], [20506,20507], [20507,20508], [20508,20100], [20100,20510], [20510,20511], [20511,20512], [20512,20513], [20513,20514], [20514,20515], [20515,20516], [20516,20517], [20517,20518], [20518,20100], [20100,20520], [20520,20521], [20521,20522], [20522,20523], [20523,20524], [20524,20525], [20525,20526], [20526,20527], [20527,20528], [20528,20100], [20100,20530], [20530,20531], [20531,20532], [20532,20533], [20533,20534], [20534,20535], [20535,20536], [20536,20537], [20537,20538], [20538,20100], [20100,20540], [20540,20541], [20541,20542], [20542,20543], [20543,20544], [20544,20545], [20545,20546], [20546,20547], [20547,20548], [20548,20100], [20100,20550], [20550,20551], [20551,20552], [20552,20553], [20553,20554], [20554,20555], [20555,20556], [20556,20557], [20557,20558], [20558,20100], [20100,20560], [20560,20561], [20561,20562], [20562,20563], [20563,20564], [20564,20565], [20565,20566], [20566,20567], [20567,20568], [20568,20100], [20100,20570], [20570,20571], [20571,20572], [20572,20573], [20573,20574], [20574,20575], [20575,20576], [20576,20577], [20577,20578], [20578,20100], [20100,20580], [20580,20581], [20581,20582], [20582,20583], [20583,20584], [20584,20585], [20585,20586], [20586,20587], [20587,20588], [20588,20100], [20100,20590], [20590,20591], [20591,20592], [20592,20593], [20593,20594], [20594,20595], [20595,20596], [20596,20597], [20597,20598], [20598,20100], [20100,20600], [20600,20601], [20601,20602], [20602,20603], [20603,20604], [20604,20605], [20605,20606], [20606,20607], [20607,20608], [20608,20100], [20100,20610], [20610,20611], [20611,20612], [20612,20613], [20613,20614], [20614,20615], [20615,20616], [20616,20617], [20617,20618], [20618,20100], [20100,20620], [20620,20621], [20621,20622], [20622,20623], [20623,20624], [20624,20625], [20625,20626], [20626,20627], [20627,20628], [20628,20100], [20100,20630], [20630,20631], [20631,20632], [20632,20633], [20633,20634], [20634,20635], [20635,20636], [20636,20637], [20637,20638], [20638,20100], [20100,20640], [20640,20641], [20641,20642], [20642,20643], [20643,20644], [20644,20645], [20645,20646], [20646,20647], [20647,20648], [20648,20100], [20100,20650], [20650,20651], [20651,20652], [20652,20653], [20653,20654], [20654,20655], [20655,20656], [20656,20657], [20657,20658], [20658,20100], [20100,20660], [20660,20661], [20661,20662], [20662,20663], [20663,20664], [20664,20665], [20665,20666], [20666,20667], [20667,20668], [20668,20100], [20100,20670], [20670,20671], [20671,20672], [20672,20673], [20673,20674], [20674,20675], [20675,20676], [20676,20677], [20677,20678], [20678,20100], [20100,20680], [20680,20681], [20681,20682], [20682,20683], [20683,20684], [20684,20685], [20685,20686], [20686,20687], [20687,20688], [20688,20100], [20100,20690], [20690,20691], [20691,20692], [20692,20693], [20693,20694], [20694,20695], [20695,20696], [20696,20697], [20697,20698], [20698,20100], [20100,20700], [20700,20701], [20701,20702], [20702,20703], [20703,20704], [20704,20705], [20705,20706], [20706,20707], [20707,20708], [20708,20100], [20100,20710], [20710,20711], [20711,20712], [20712,20713], [20713,20714], [20714,20715], [20715,20716], [20716,20717], [20717,20718], [20718,20100], [20100,20720], [20720,20721], [20721,20722], [20722,20723], [20723,20724], [20724,20725], [20725,20726], [20726,20727], [20727,20728], [20728,20100], [20100,20730], [20730,20731], [20731,20732], [20732,20733], [20733,20734], [20734,20735], [20735,20736], [20736,20737], [20737,20738], [20738,20100], [20100,20740], [20740,20741], [20741,20742], [20742,20743], [20743,20744], [20744,20745], [20
```

The flowchart illustrates the decision-making process for the `All_Reduce` algorithm in OpenMPI, specifically regarding the `omp/mca/coll/tuned/coll_tuned_decision_fixed.c` file.

```

graph LR
    Start((Start)) --> Q1{chunk file larger than 10000 Byte?}
    Q1 -- No --> Q2{chunk file more than the number of MPI Process?}
    Q2 -- No --> Q3{chunk file larger than 1MB * MPI Process?}
    Q3 -- Yes --> SR[Segmented Ring]
    Q3 -- No --> R[Ring]
    Q2 -- Yes --> BB[Butterfly]
    BB --- LogN[log N]
    R --- LogN[log N]
    SR --- LogN[log N]
    
```

Legend:

- Blue circles:** MPI Process
- Red numbers:** Rank
- Yellow numbers:** Local rank

Butterfly: A network diagram showing a butterfly topology with 8 nodes labeled a-h. The connections are: a to b, c to d, e to f, g to h, and a to d, b to e, c to f, and g to h.

reduce+broadcast: A network diagram showing a reduce+broadcast operation where multiple nodes (a-f) converge to a single central node.

Ring: A network diagram showing a ring topology with 8 nodes labeled a-h connected in a circular loop.

Segmented Ring: A network diagram showing a segmented ring topology with 8 nodes labeled a-h, divided into two segments of 4 nodes each.

The figure consists of three side-by-side bar charts. Each chart plots 'Baseline Performance' (Y-axis) against the number of nodes (X-axis). The X-axis for all three charts ranges from 1 to 12 nodes.

- Testing on QiMing 2.0:** The Y-axis ranges from 0 to 30. The data shows increasing performance as more nodes are added, with values approximately: Node 1: 9.913, Node 2: 10.414, Node 4: 12.021, Node 8: 24.421, Node 16: 36.821.
- Testing on Gadi:** The Y-axis ranges from 0 to 5000. The data shows increasing performance as more nodes are added, with values approximately: Node 1: 423.81, Node 2: 978.48, Node 4: 1331.54, Node 8: 2331.13, Node 16: 4331.79.
- Testing on NSCC:** The Y-axis ranges from 0 to 5000. The data shows increasing performance as more nodes are added, with values approximately: Node 1: 423.81, Node 2: 978.48, Node 4: 1331.54, Node 8: 2331.13, Node 16: 4331.79.

Each chart includes a legend indicating 'Baseline Performance'.

IPM analysis

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

Baseline on Gadi (legend)

| node num. | wallclock | %comm | comm | calc |
|-----------|-----------|--------|--------|---------|
| 2 nodes | 16.8237s | 15.01% | 10.26s | 109.03s |
| 16 nodes | 50.723s | 55.83% | 28.32s | 22.40s |

SUSTech SUSTech
Southern University of Science and Technology

Baseline on NSCC

| node num. | wallclock | %comm | comm | calc |
|-----------|-----------|--------|--------|--------|
| 2 nodes | 16.2758s | 28.85% | 2.84s | 73.11s |
| 16 nodes | 81.15s | 56.67% | 45.09s | 35.17s |

The diagram illustrates the communication process between two MPI ranks. On the left, a cloud of colored circles labeled "Particles" is shown. A dashed arrow points from these particles to a "Sub Domain" box within a "Domain Buffer" on the left MPI Rank. A blue square representing MPI communication is labeled ①. An arrow labeled ② points from the MPI box to the "Network". On the right MPI Rank, another "Sub Domain" box is shown with a blue square labeled ③. An arrow labeled ④ points from the MPI box to the "Domain Buffer". A blue square labeled ⑤ represents the "Compute Force" step. The "MPI Rank" and "Network" labels are at the top and bottom respectively.

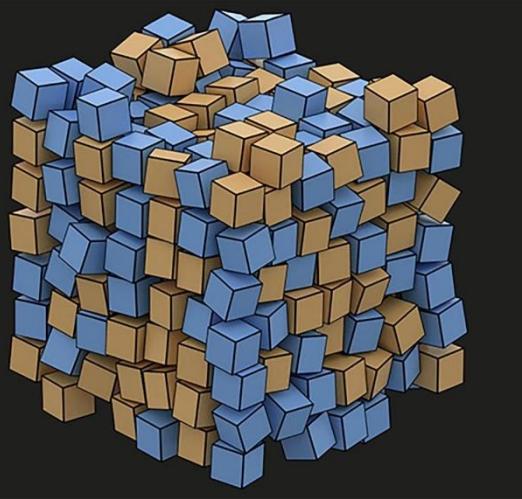
The diagram illustrates the process of domain decomposition and MPI scheduling. It starts with a 3D 'Container' representing the simulation space, which is divided into smaller 'Subdomains' (MPI tasks) using 'Domain Decomposition'. Each subdomain is assigned to a specific MPI rank. The subdomains are then scheduled for execution. The 'Compute Particle Force and Movement' step is shown as a sequence of operations across different MPI ranks, resulting in the final 'Result'.

Hoomd Description

```

1 from hoomd import *
2 from hoomd import hpmc
3 # place particles
4 context.initialize('--mode=cpu')
5 unitcell=lattice.sc(a=1.2, type_name='A')
6 system = init.create_lattice(unitcell, n=7)
7 # hard particle Monte Carlo
8 mc = hpmc.integrate.convex_polyhedron(
9     d=0.1, a=0.1, seed=2)
10 cube_verts = \
11     [[[-0.5,-0.5,-0.5], [0.5,-0.5,-0.5],
12      [-0.5,-0.5, 0.5], [0.5,-0.5, 0.5],
13      [-0.5, 0.5,-0.5], [0.5, 0.5,-0.5],
14      [-0.5, 0.5, 0.5], [0.5, 0.5, 0.5]]]
15 mc.shape_param.set('A',
16                     vertices=cube_verts)
17 # run the simulation
18 run(50e3)

```



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 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

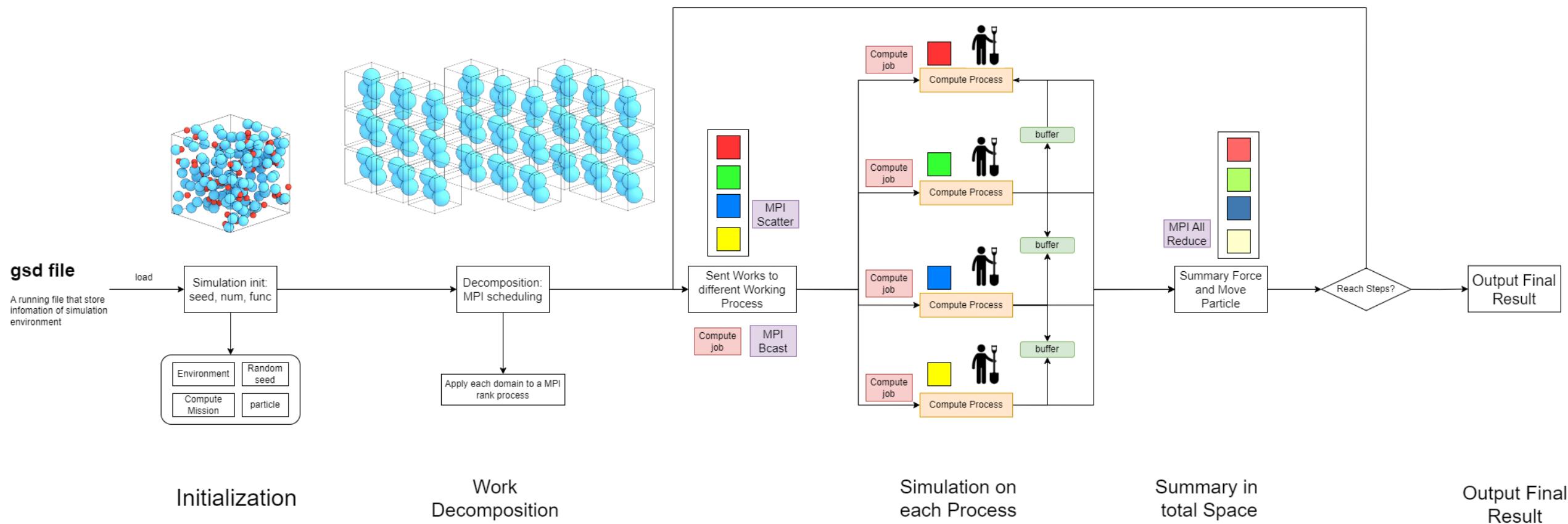
Hoomd-Benchmark calculates WCA potential for N particles as workload.

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



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)

| 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)

| 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 |

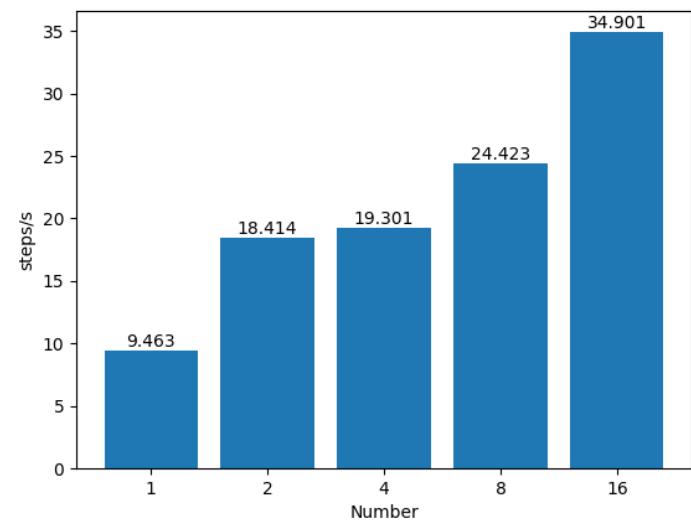


HOOMD-blue Baseline

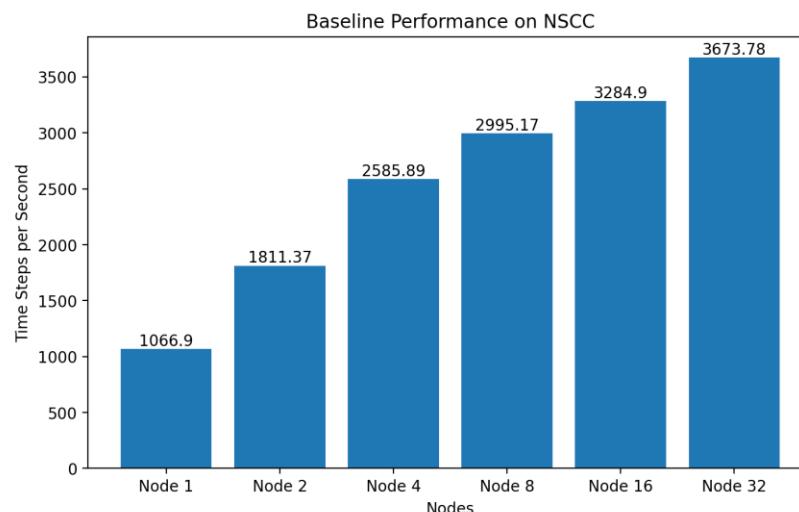
Before four nodes, the expansion effect is good

| Software | Version |
|----------|---------|
| OpenMPI | 4.1.5 |
| Cereal | 1.3.1 |
| Eigen | 3.4.0 |

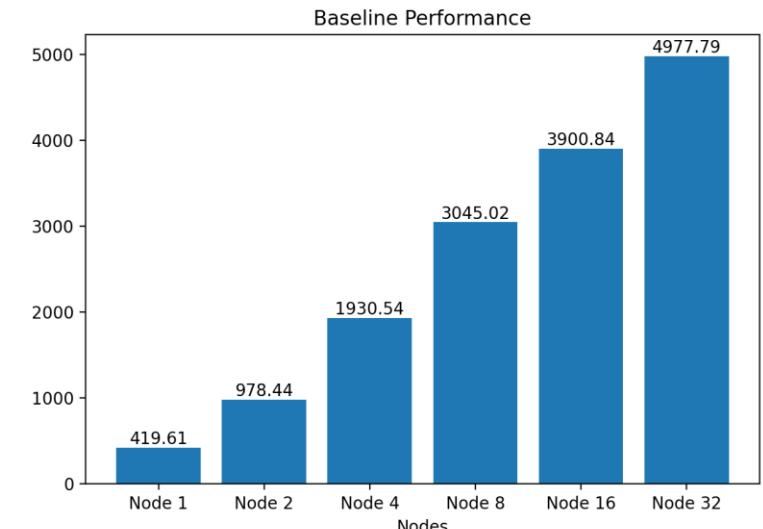
Testing on QiMing 2.0



Testing on Gadi



Testing on NSCC



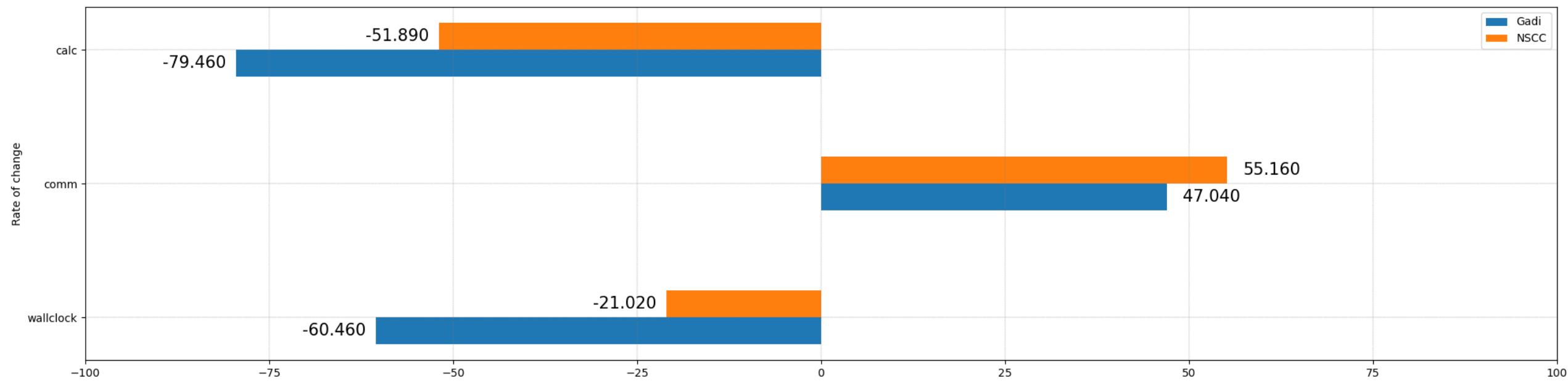
Number denotes to the number of nodes

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

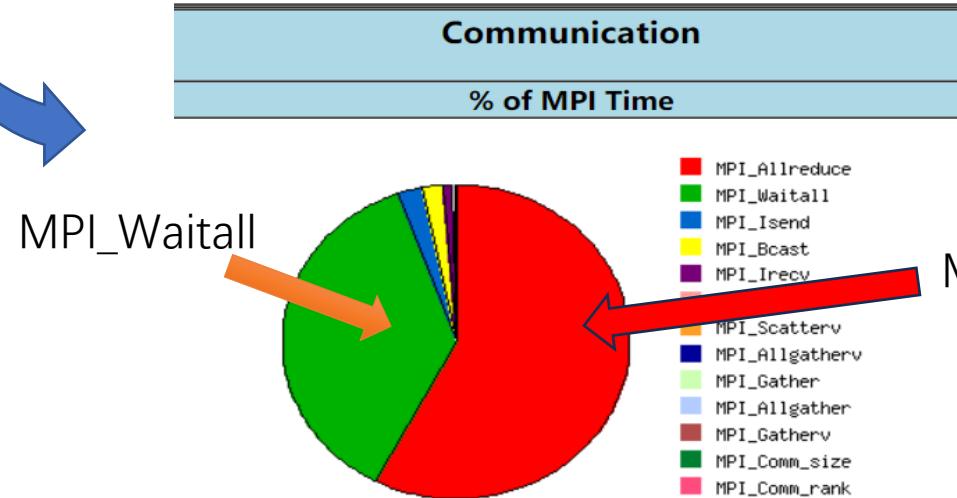
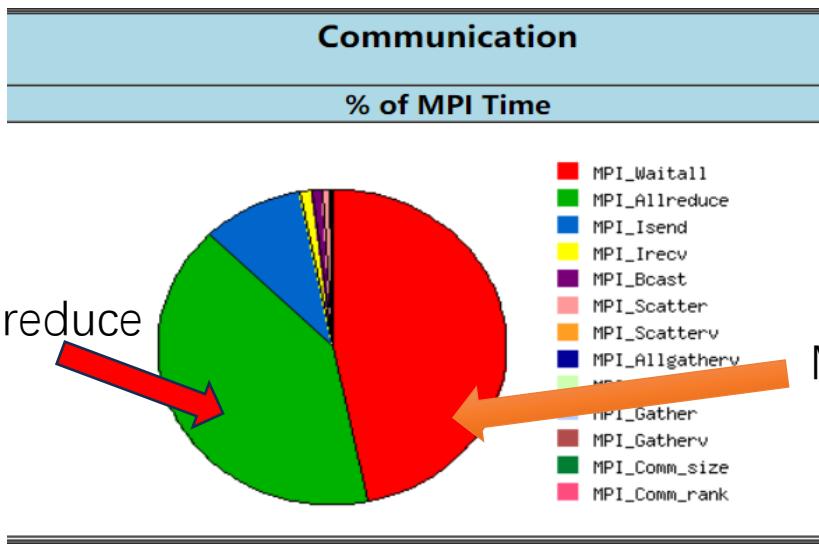
| node num. | wallclock | %comm | comm | calc |
|-----------|-----------|--------|--------|--------|
| 2 nodes | 102.752s | 28.85% | 29.64s | 73.11s |
| 16 nodes | 81.155s | 56.67% | 45.99s | 35.17s |



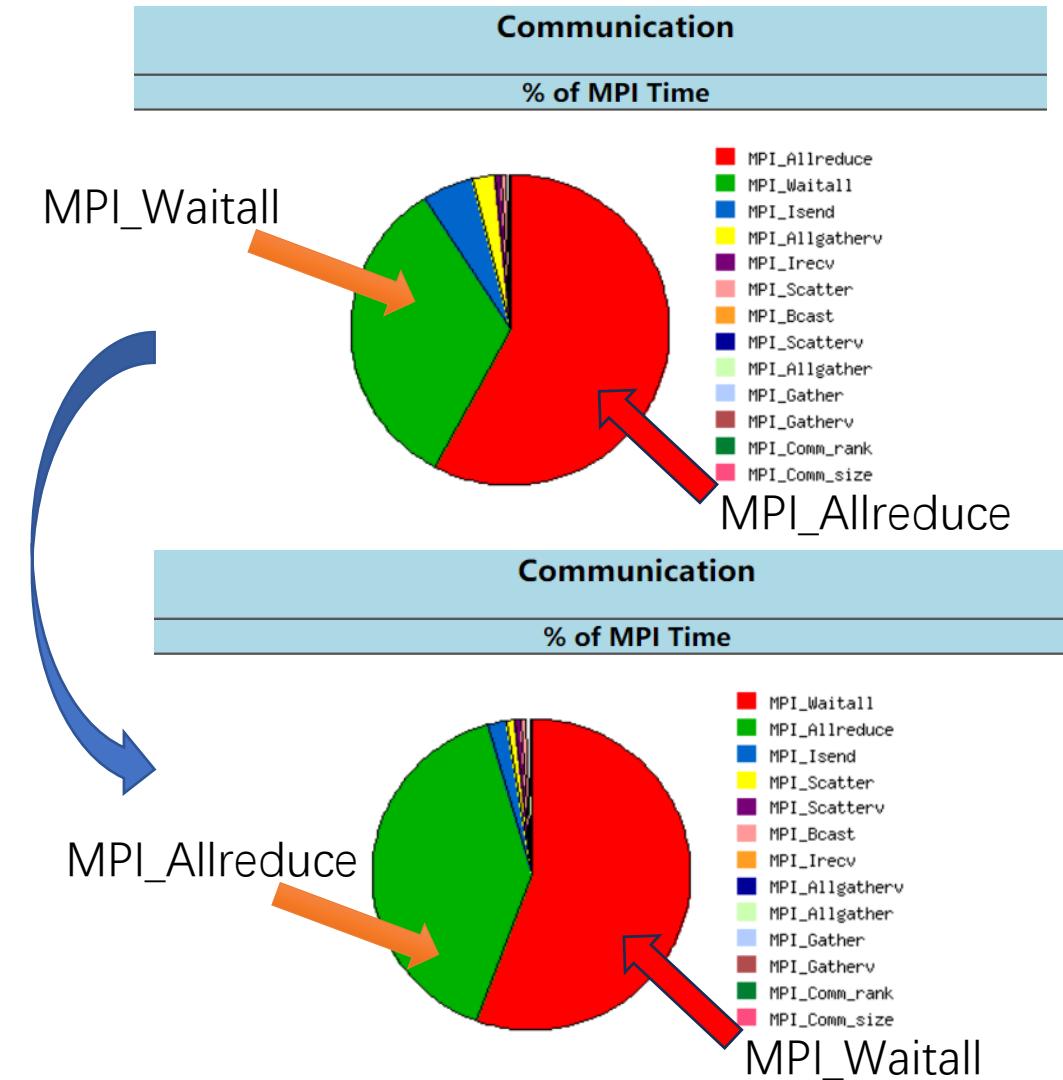
IPM analysis

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

Baseline on Gadi (legend)

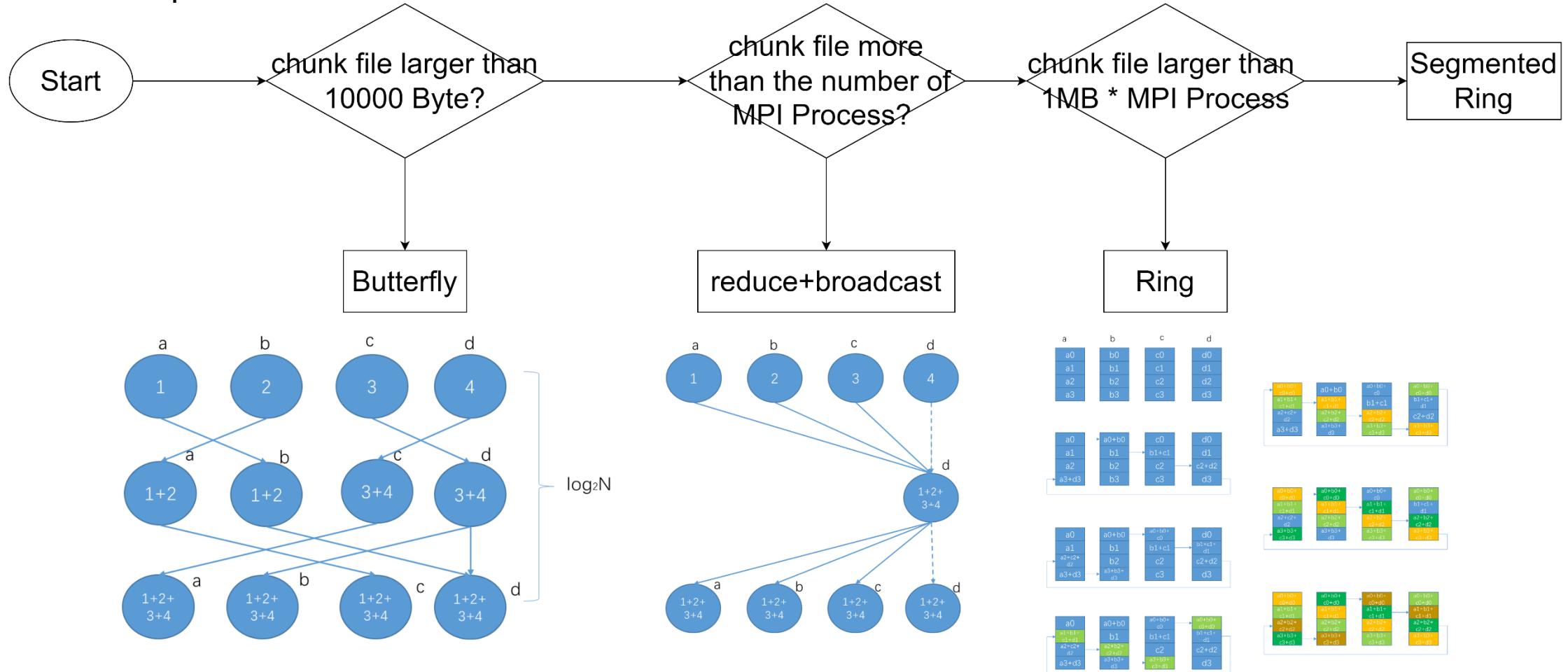


Baseline on NSCC

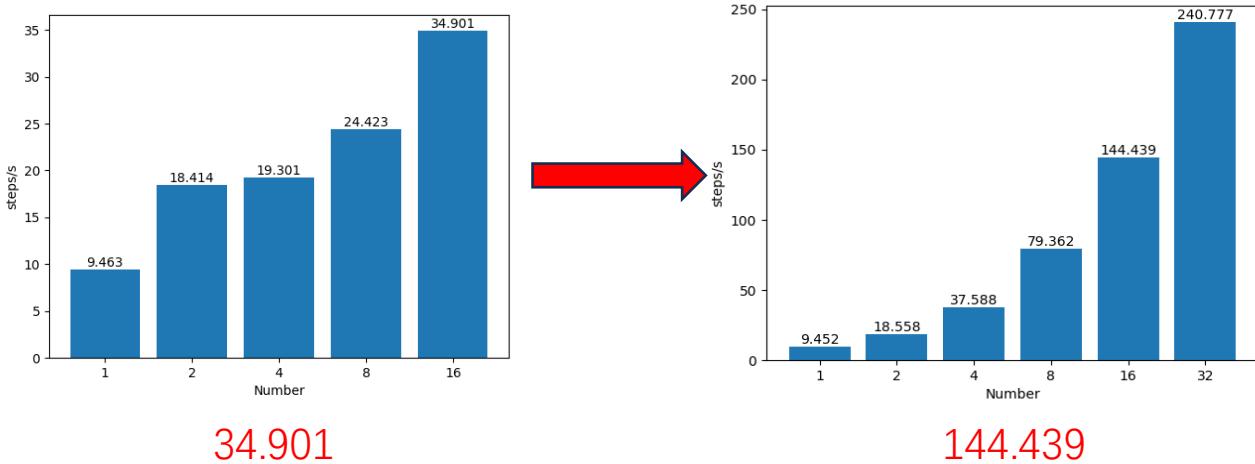


OpenMPI MPI_Allreduce strategy

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



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_i nfo.

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

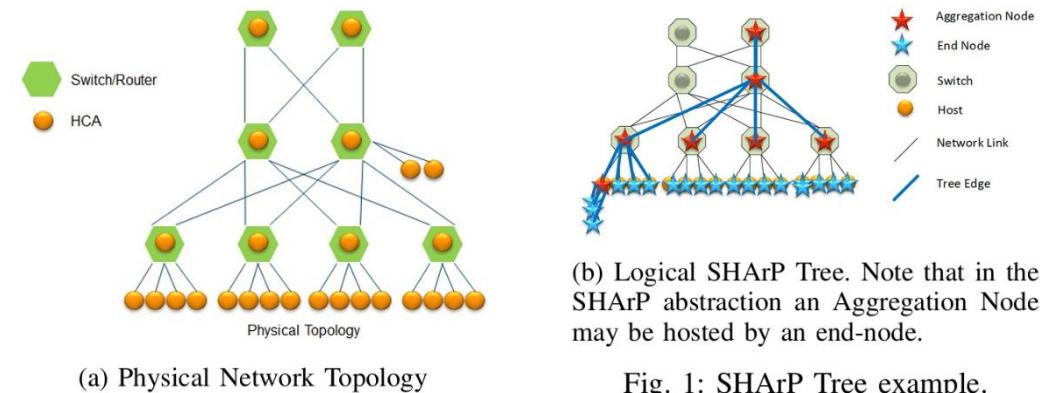
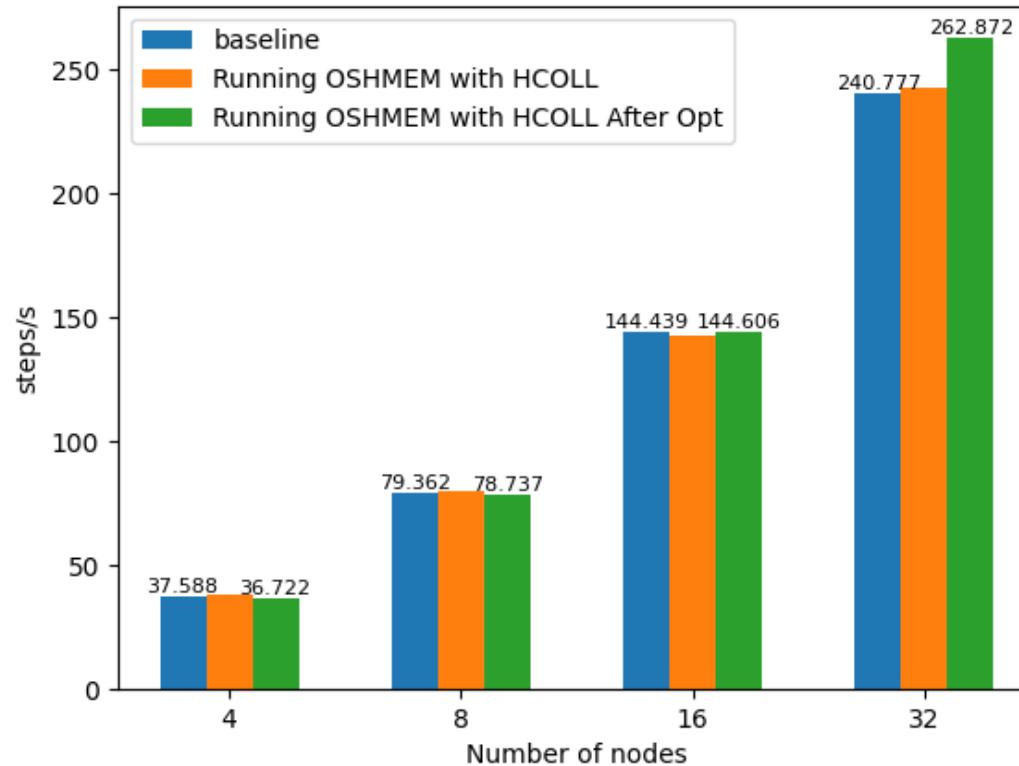


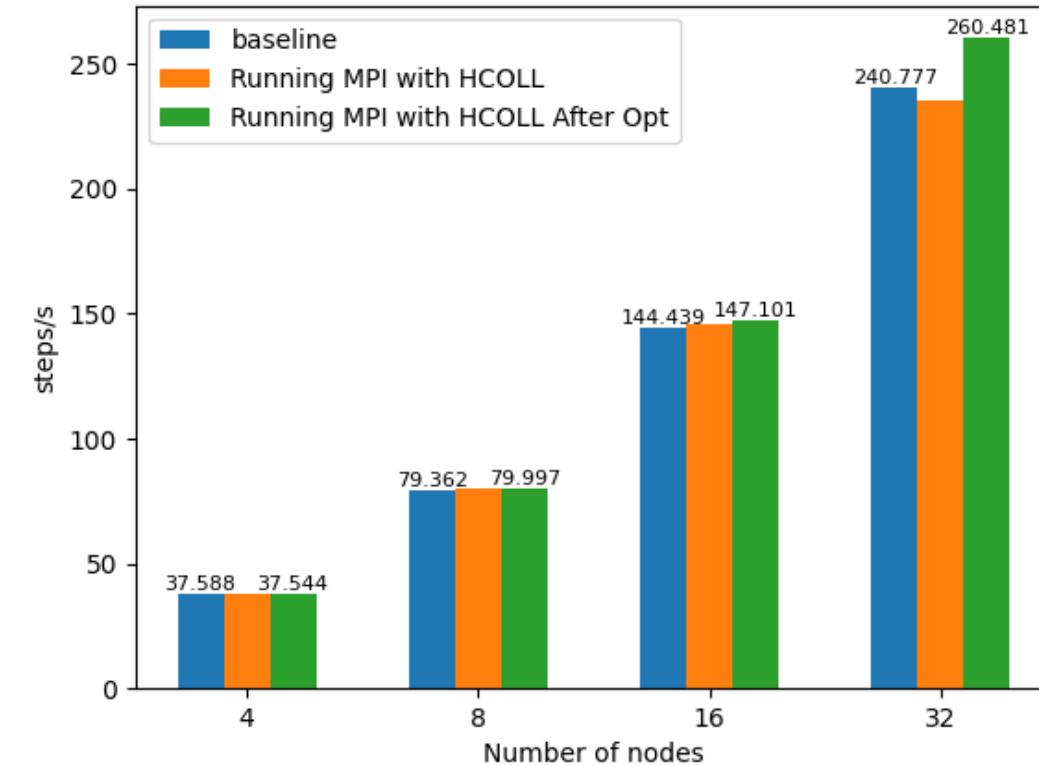
Fig. 1: SHArP Tree example.

Using HPC-X on QiMing 2.0 with Optimization value



Running OSHMEM with HCOLL

9.17% improvement for 32 nodes

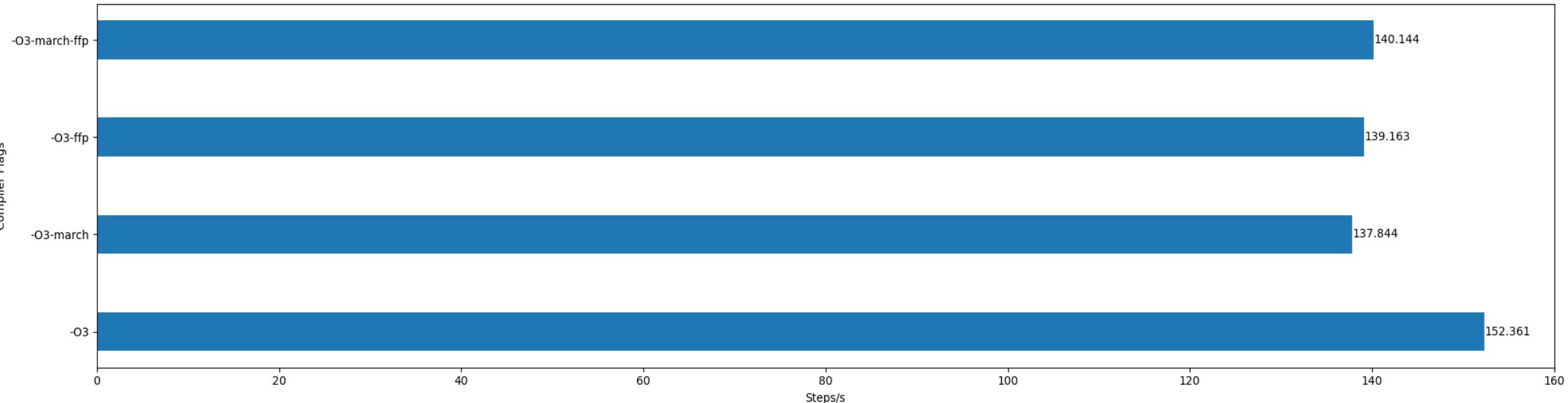


Running MPI with HCOLL

8.18% improvement for 32 nodes

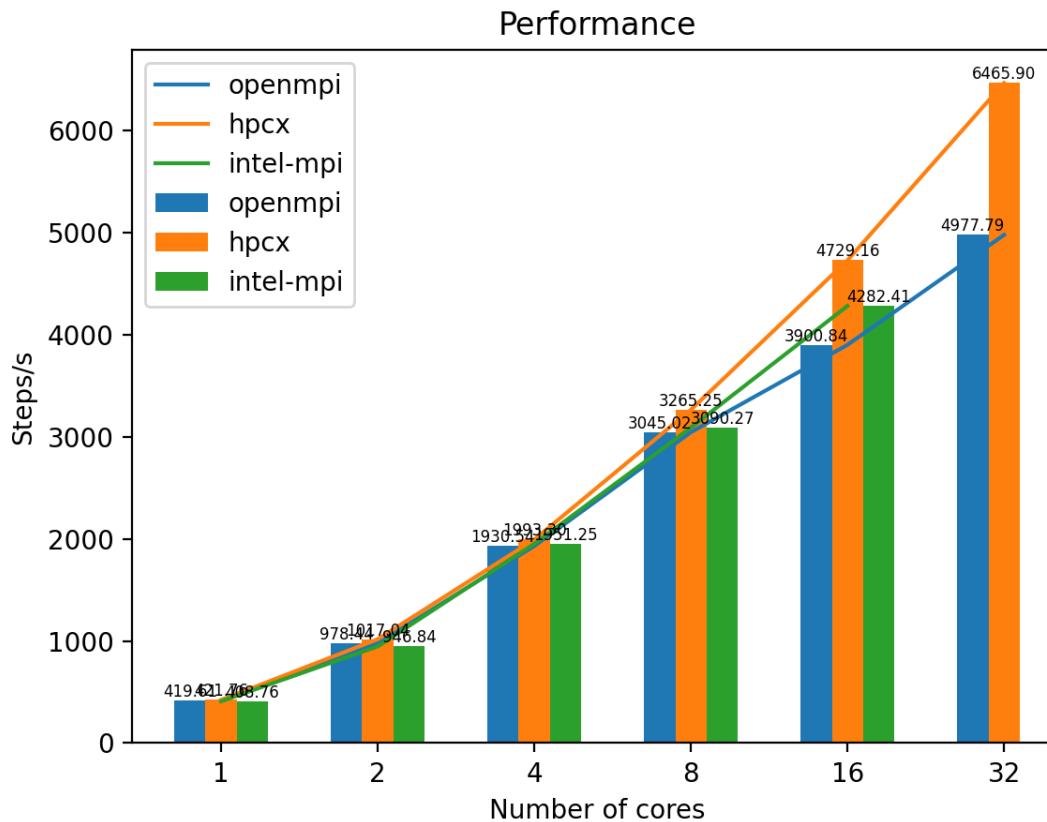
Compiler Flags Optimization(QIMING)

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

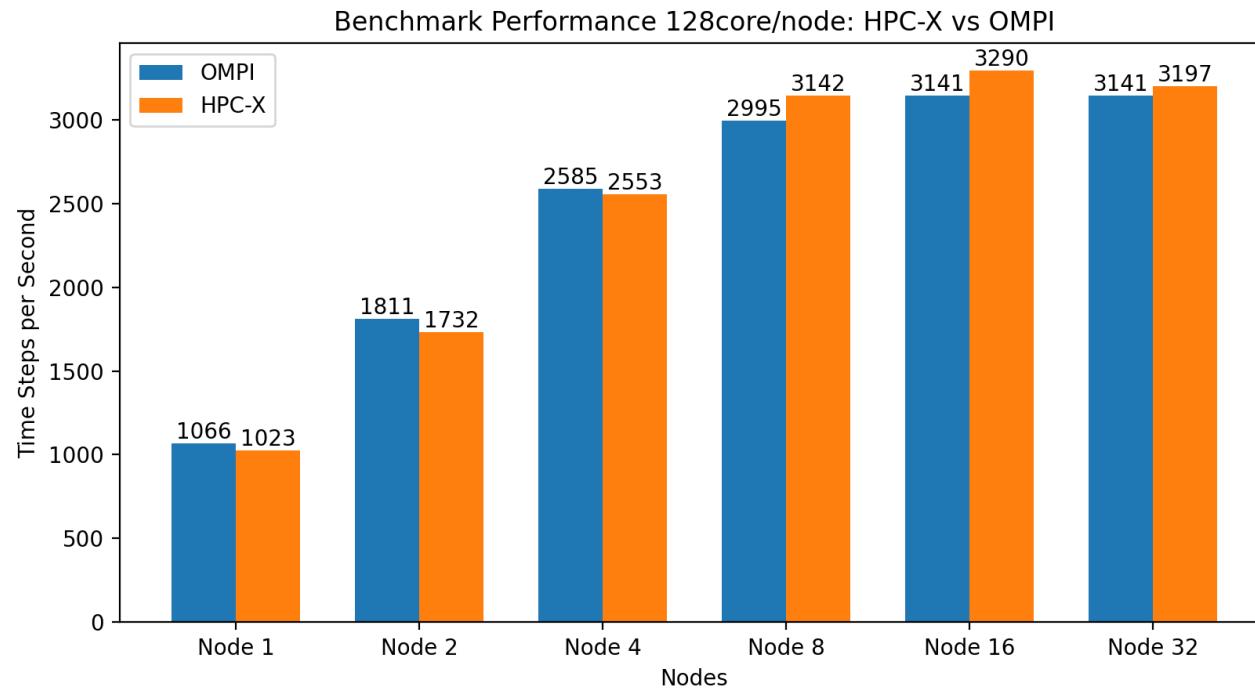


29.89% improvement for 32 nodes

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

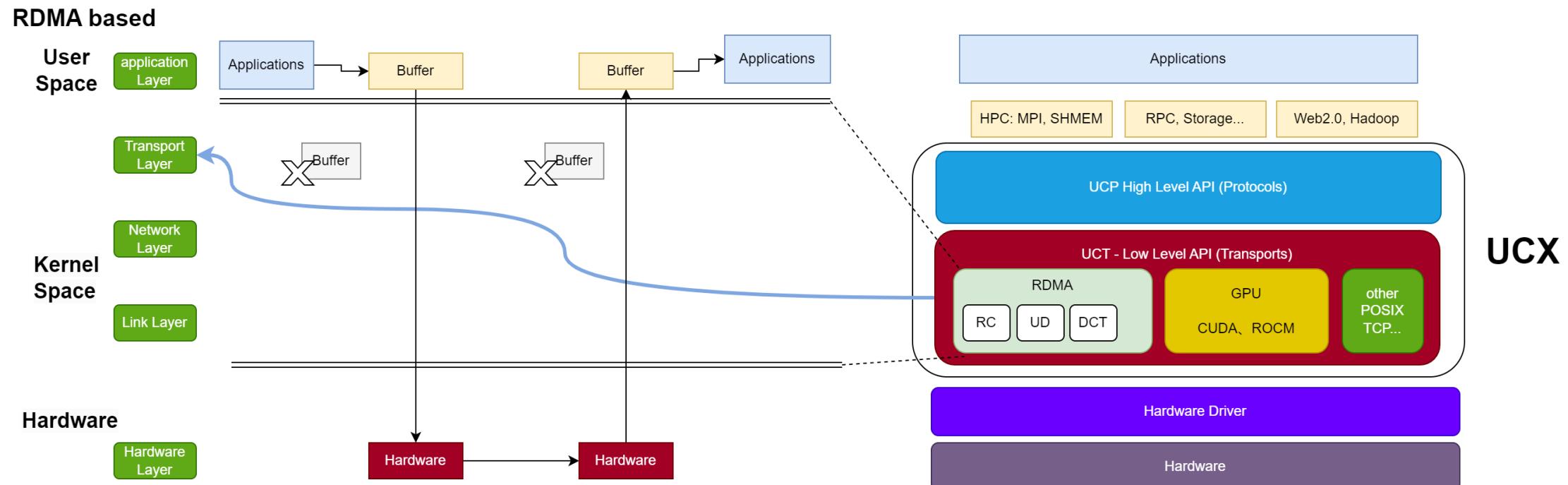
HPC-X does not get significant improvement in NSCC



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.

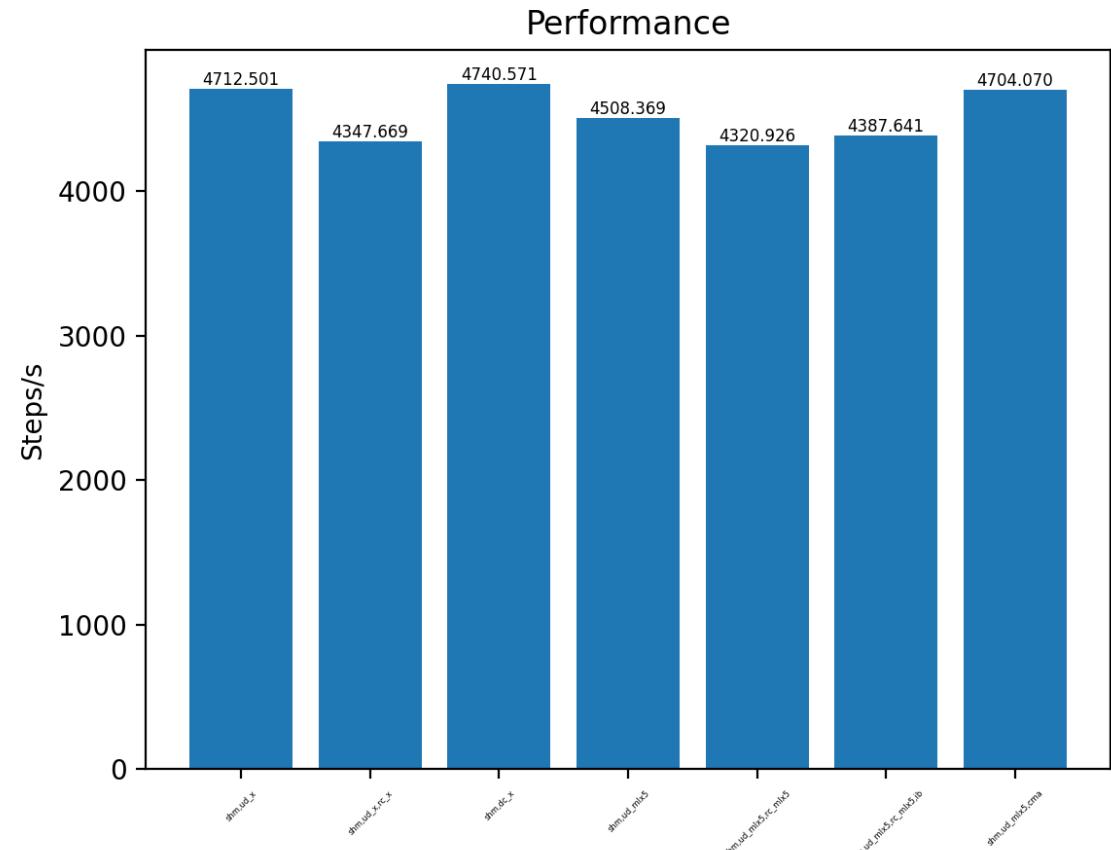


Gadi optimal ucx params

We test multi combination of UCX_TLS to get better performance.

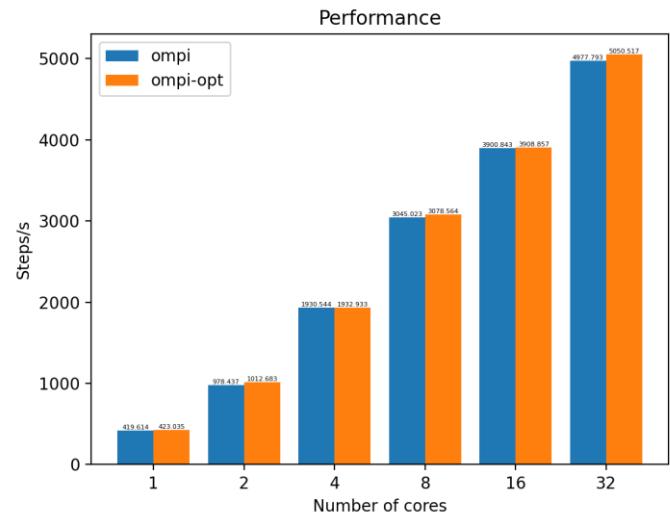
| Params | description |
|--------|--|
| all | use all the available transports |
| sm | all shared memory transports |
| rc | reliable connection |
| ud | unreliable datagram transport |
| dc | Mellanox scalable offloaded dynamic connection transport |
| ... | ... |

Testing UCX_TLS in bottleneck scale 16nodes



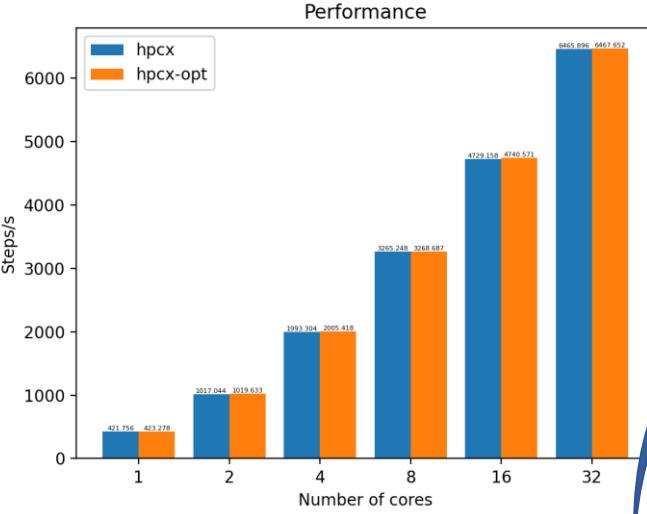
Use UCX_TLS params with hpcx on gadi

Best ucx params in ompi and hpcx(gadi)



1.46% improvement
for 32 nodes

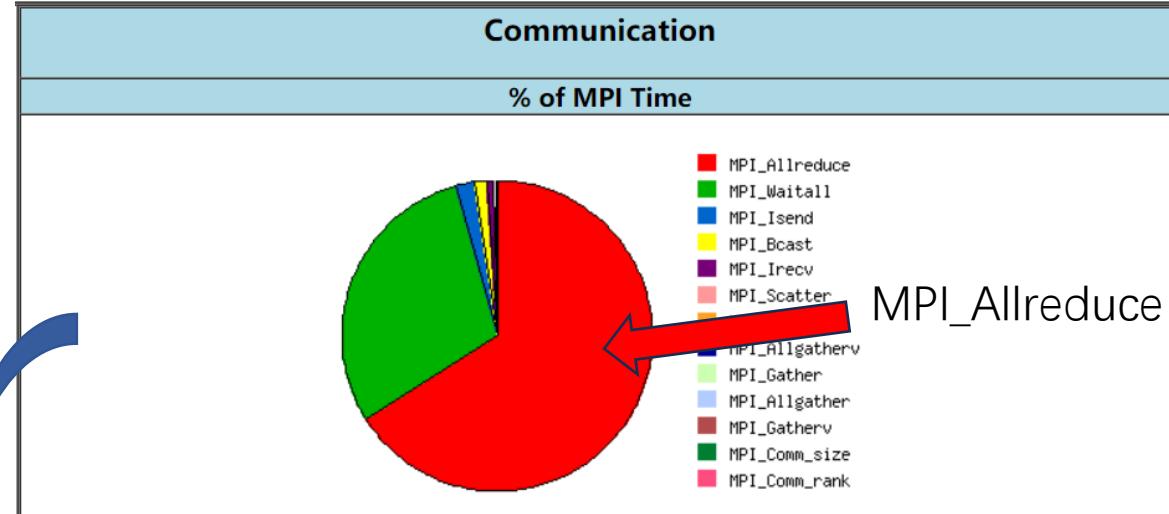
shm, dc_x does not have much improvements
in both ompi and hpcx



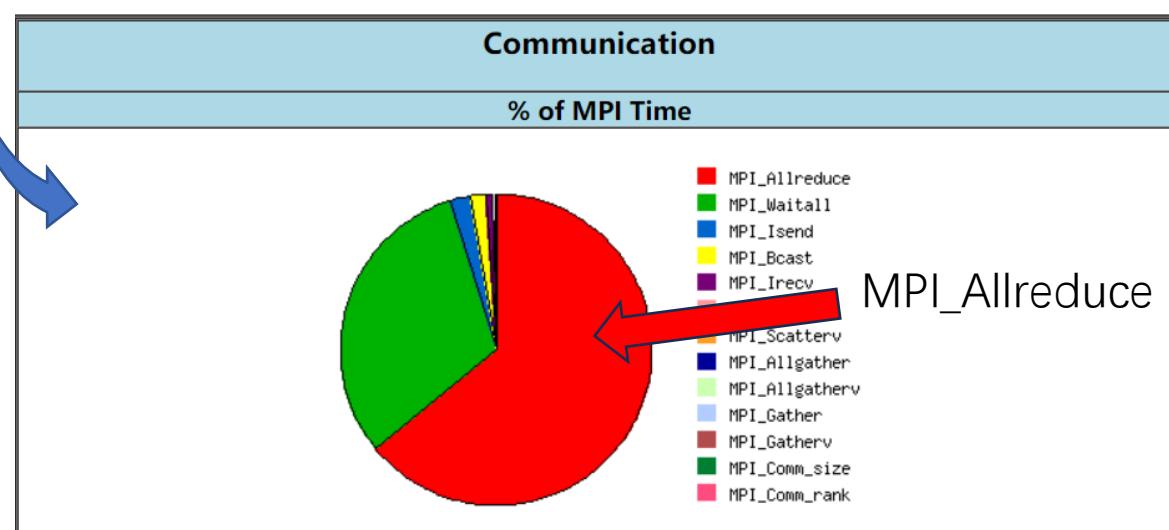
0.27% improvement
for 32 nodes

IPM analysis

Ucx params doesn't change much



MPI_Allreduce

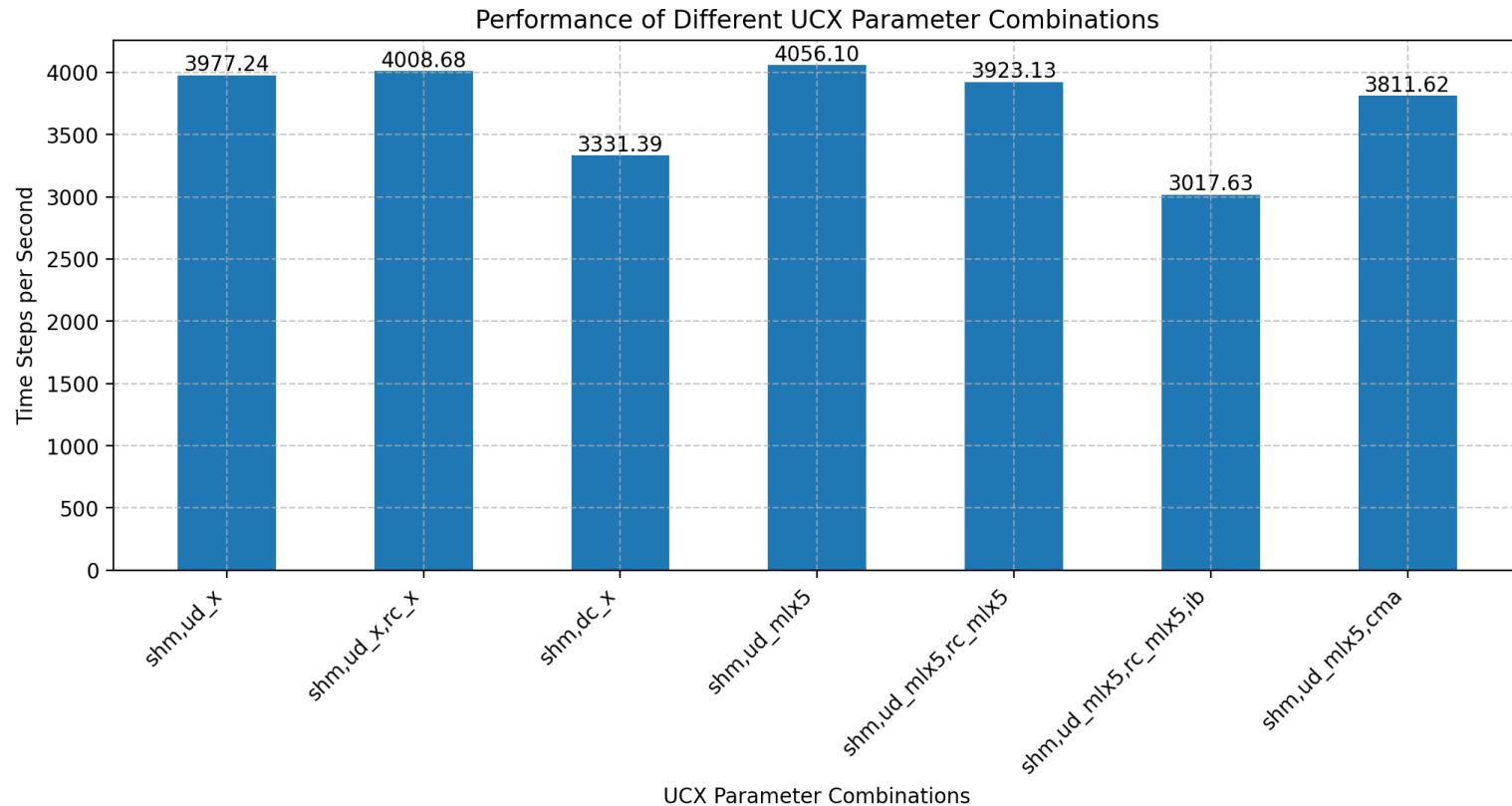


MPI_Allreduce

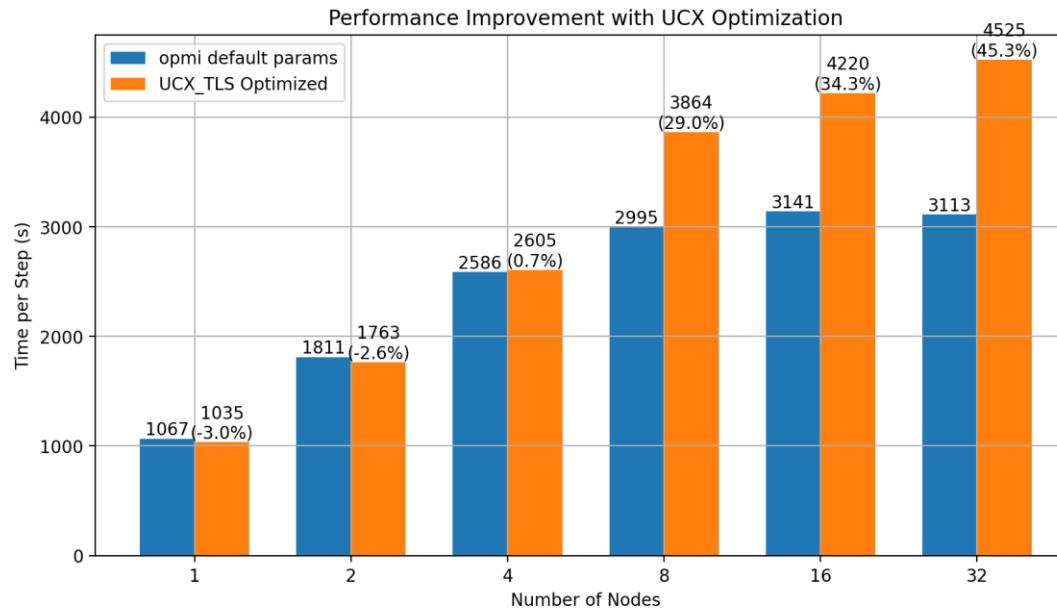
NSCC optimal ucx params

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_nnection transport |
| ... | ... |



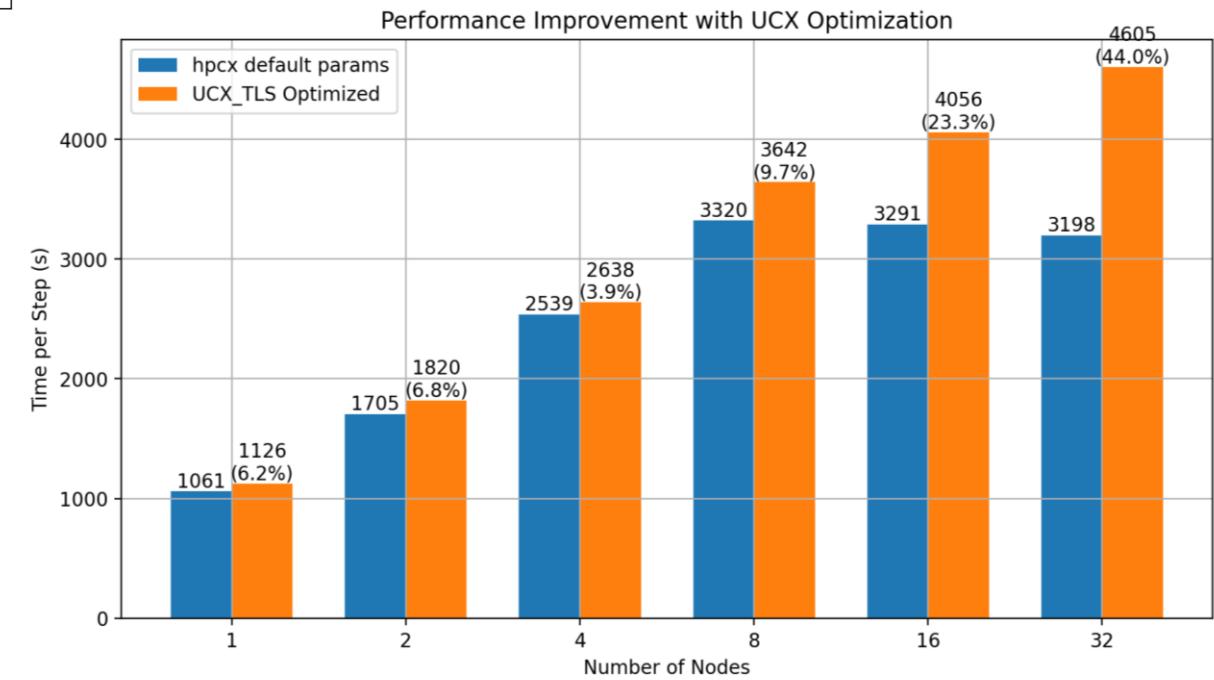
Testing UCX_TLS in bottleneck scale 16nodes



Best ucx params in ompi

shm, ud_mlx5 get up to 45% improvement in ompi

45.3% improvement for 32 nodes

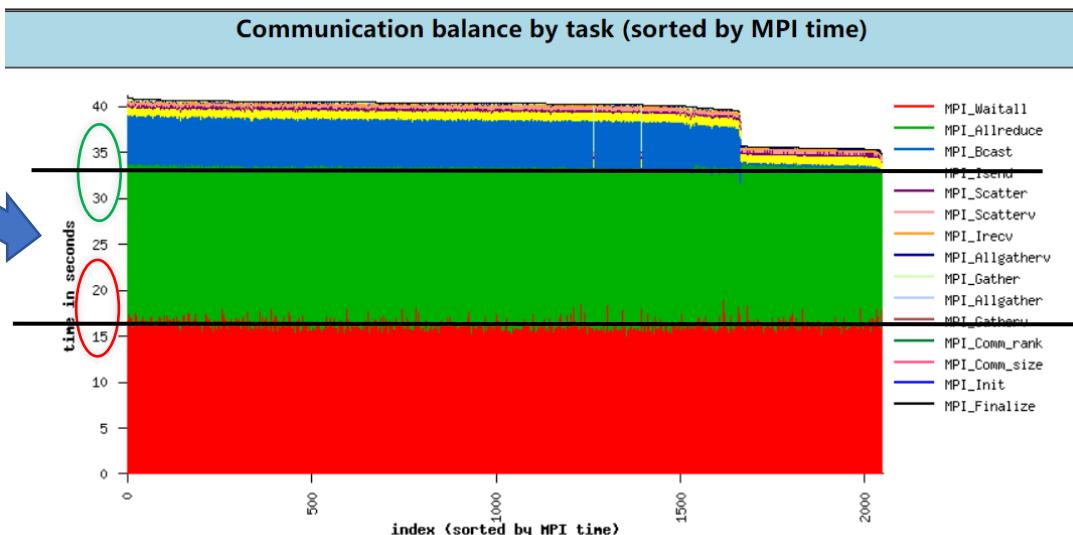
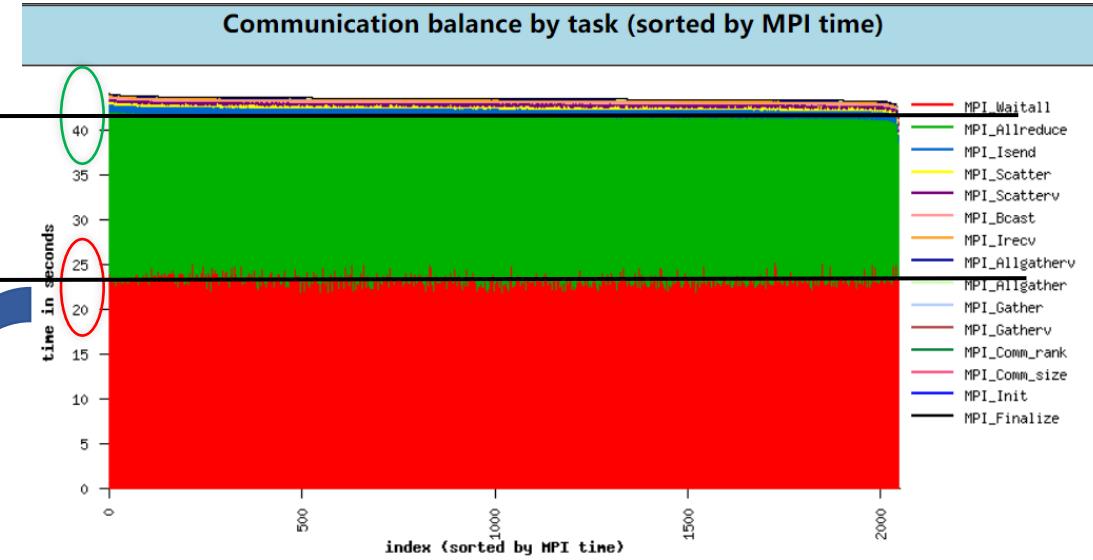


Best ucx params in hpcx

shm, ud_mlx5 get up to 44% improvement in hpcx

44.0% improvement for 32 nodes

UCX_TLS improvement analysis



IPM analysis

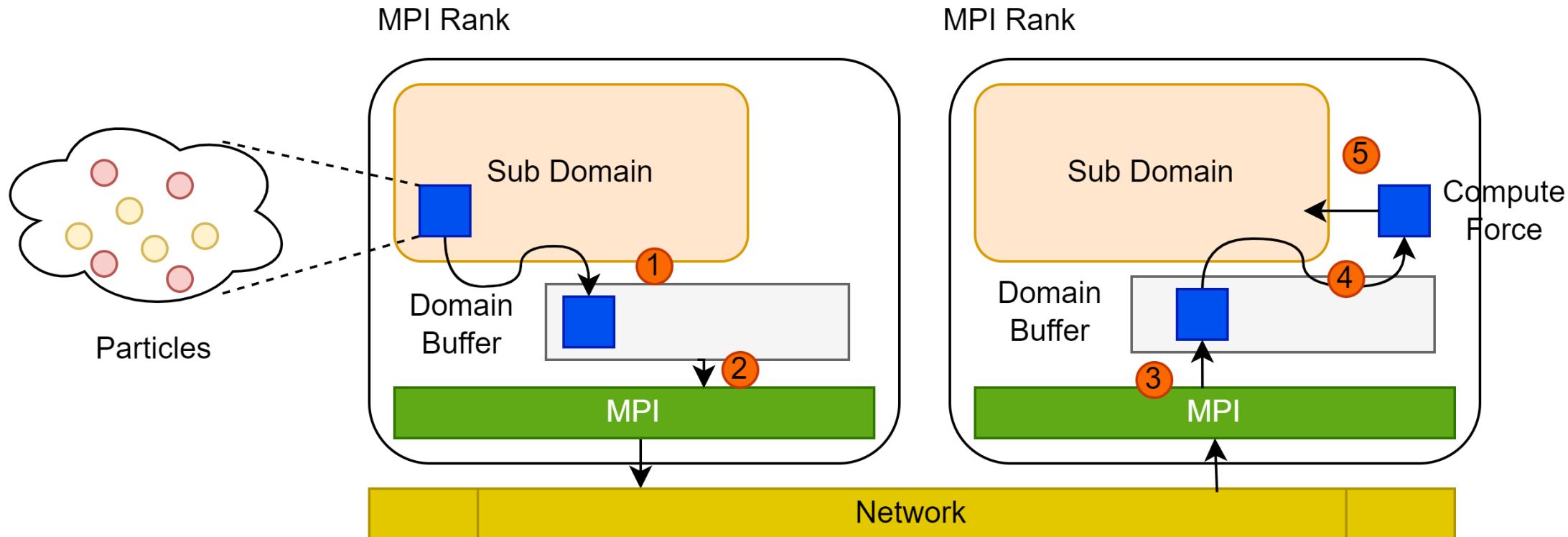
From defualt → optimal ucx params we get improvement in communication bottleneck

Waitall: $\approx 23 \rightarrow \approx 16$
Allreduce: $> 40 \rightarrow \approx 33$

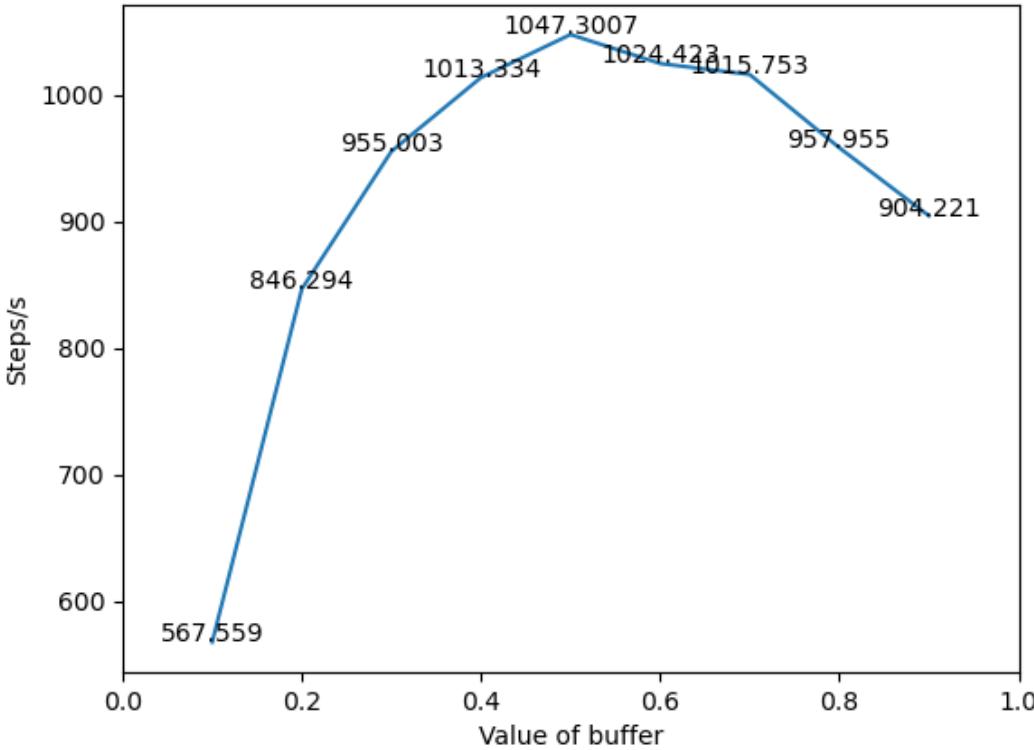
Better buffer params

Buffer is a params to set the neighbor list buffer distance in hoomd.

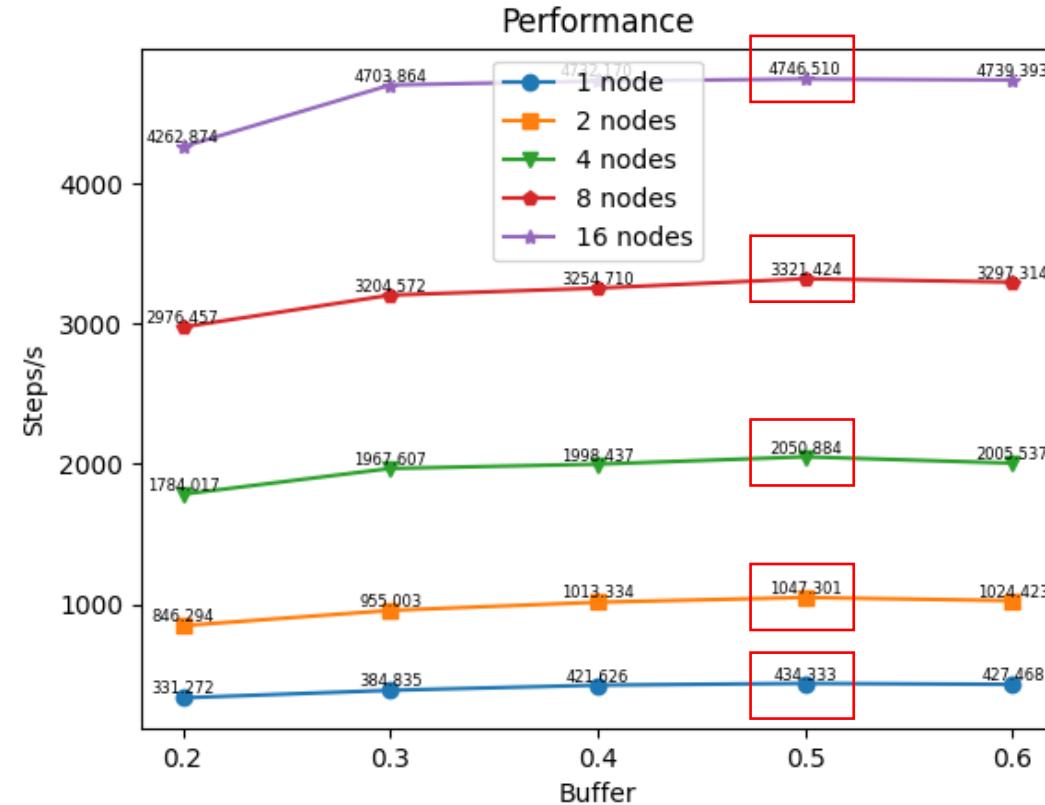
The neighbor list recomputes itself more often when buffer is small, and the pair force computation takes more time when buffer is large.



Better buffer params(gadi)

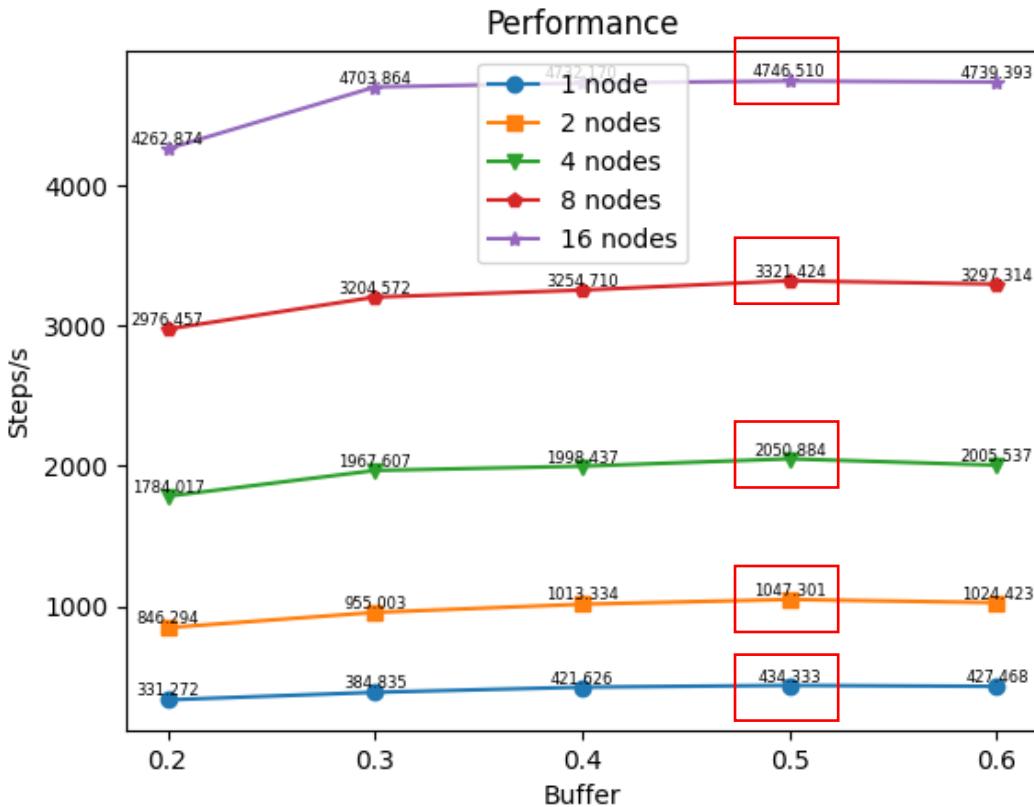


Testing buffer in bottleneck scale 2 nodes

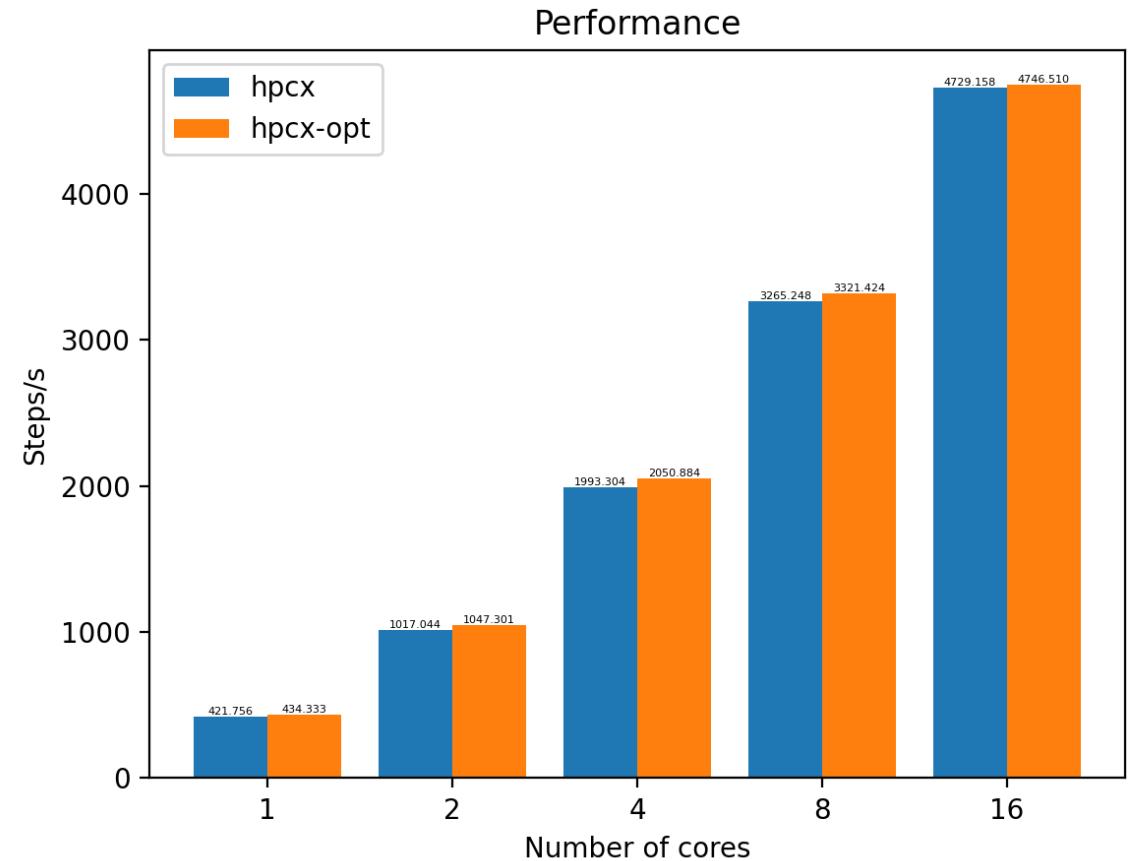


For more tests according to buffer

Best buffer params in hpcx



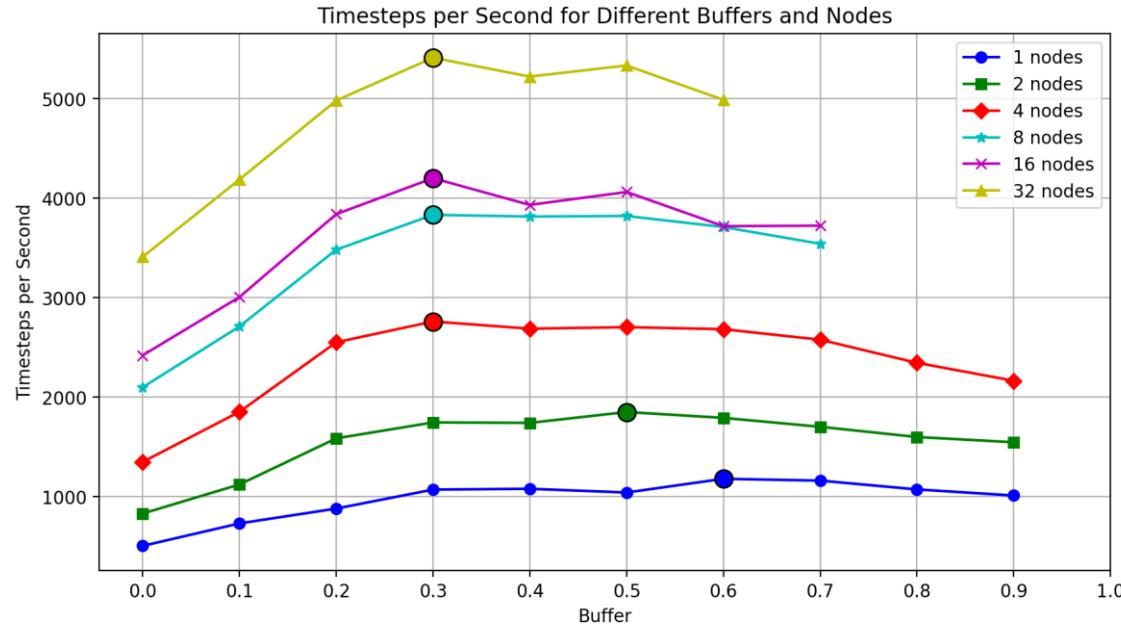
For more tests according to buffer



0.37% improvement for 32 nodes

buffer params(NSCC)

buffer is a key params in domain composition parallel algorithm

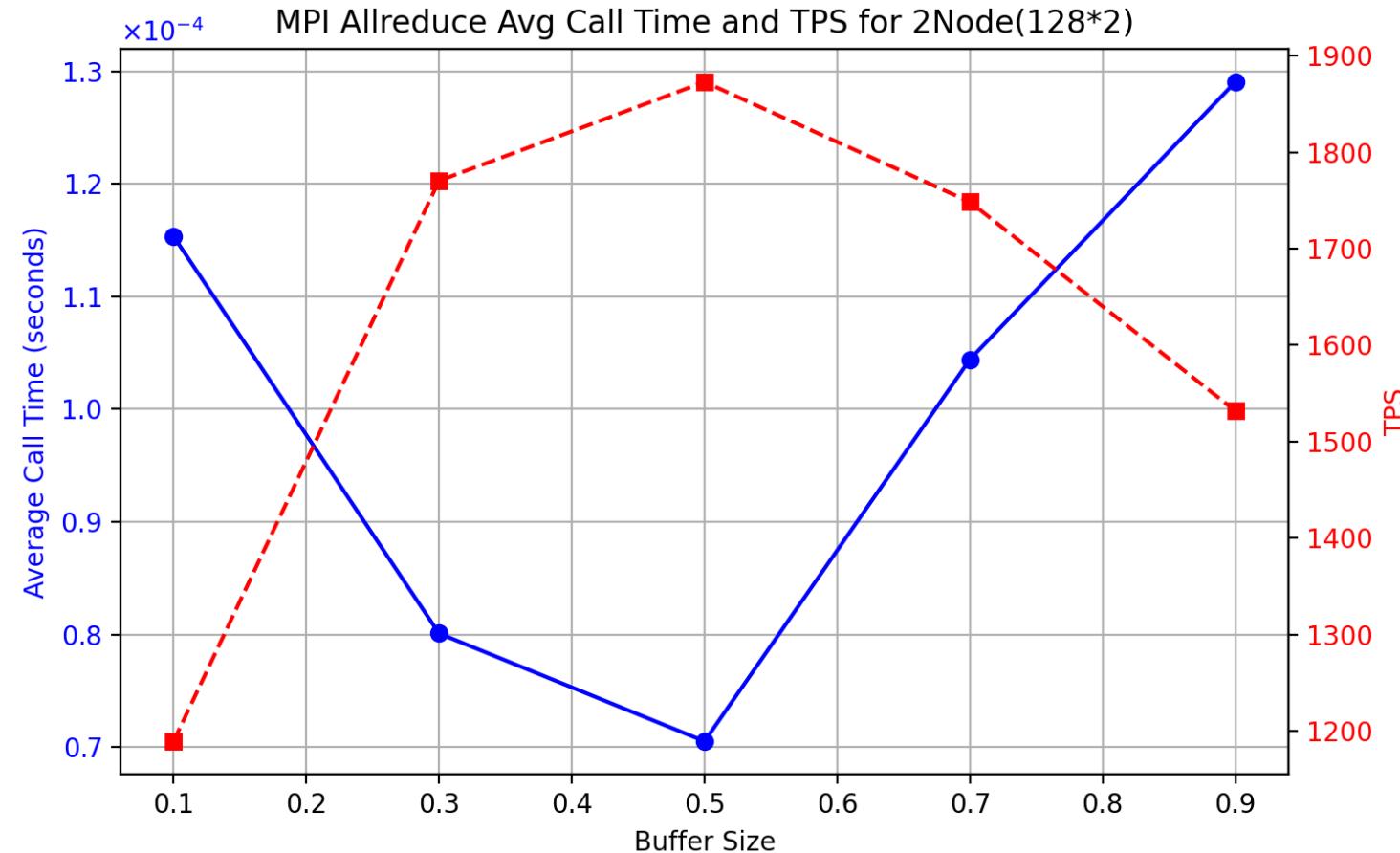


4.86% improvement for 32 nodes

N-list updated are infected by buffer params

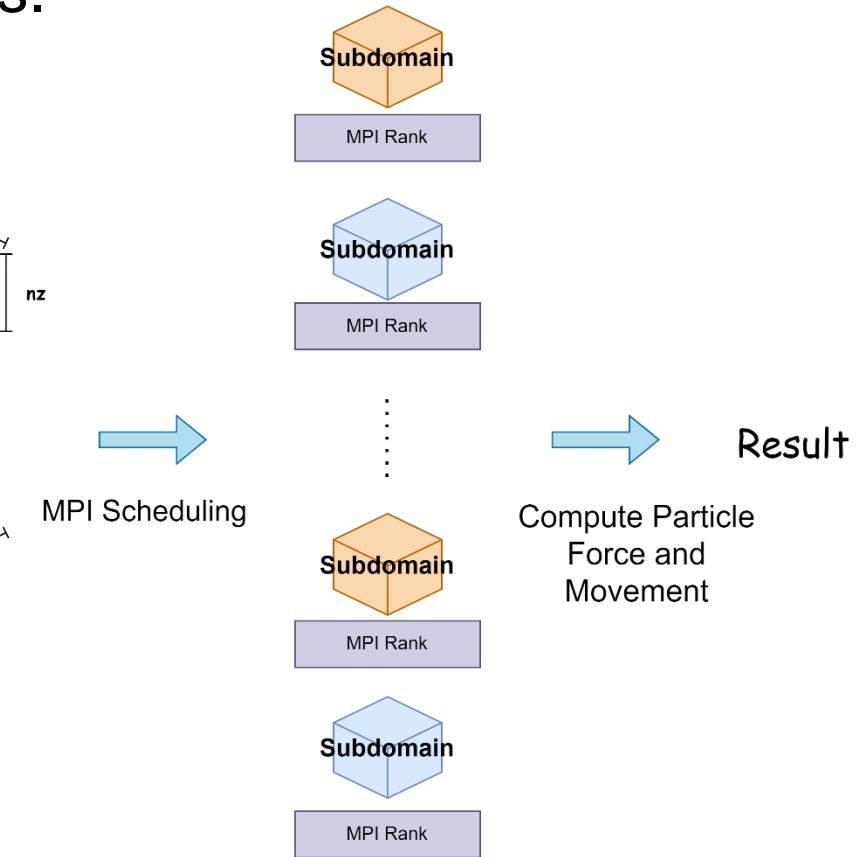
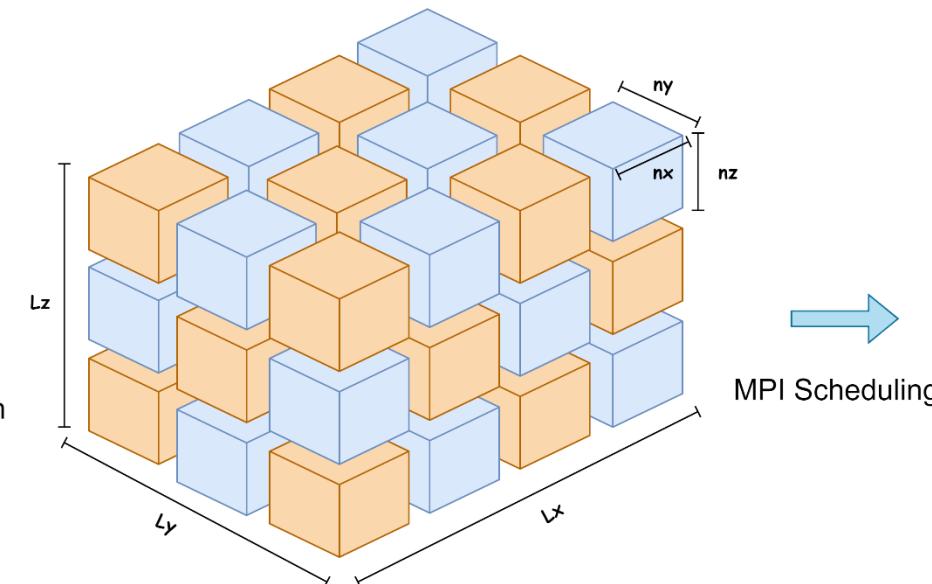
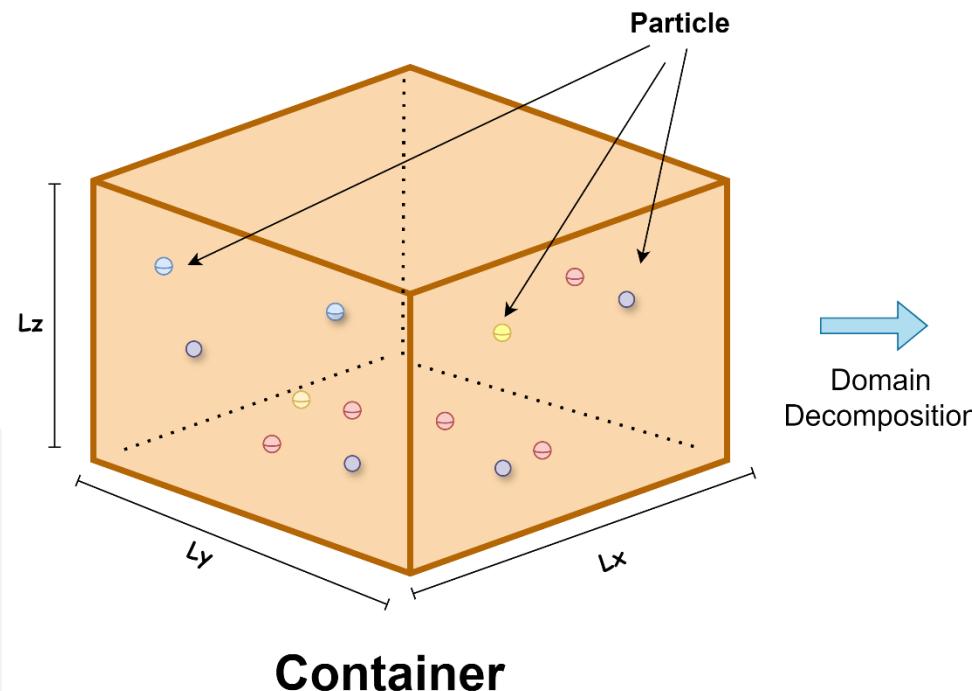
```
1 if (needsUpdating(timestep))
2 {
3     bool overflowed = false;
4     do
5     {
6         buildNlist(timestep);
7         overflowed = checkConditions();
8         if (overflowed)
9             ...
10    } while (overflowed);
11 }
12 }
```

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.

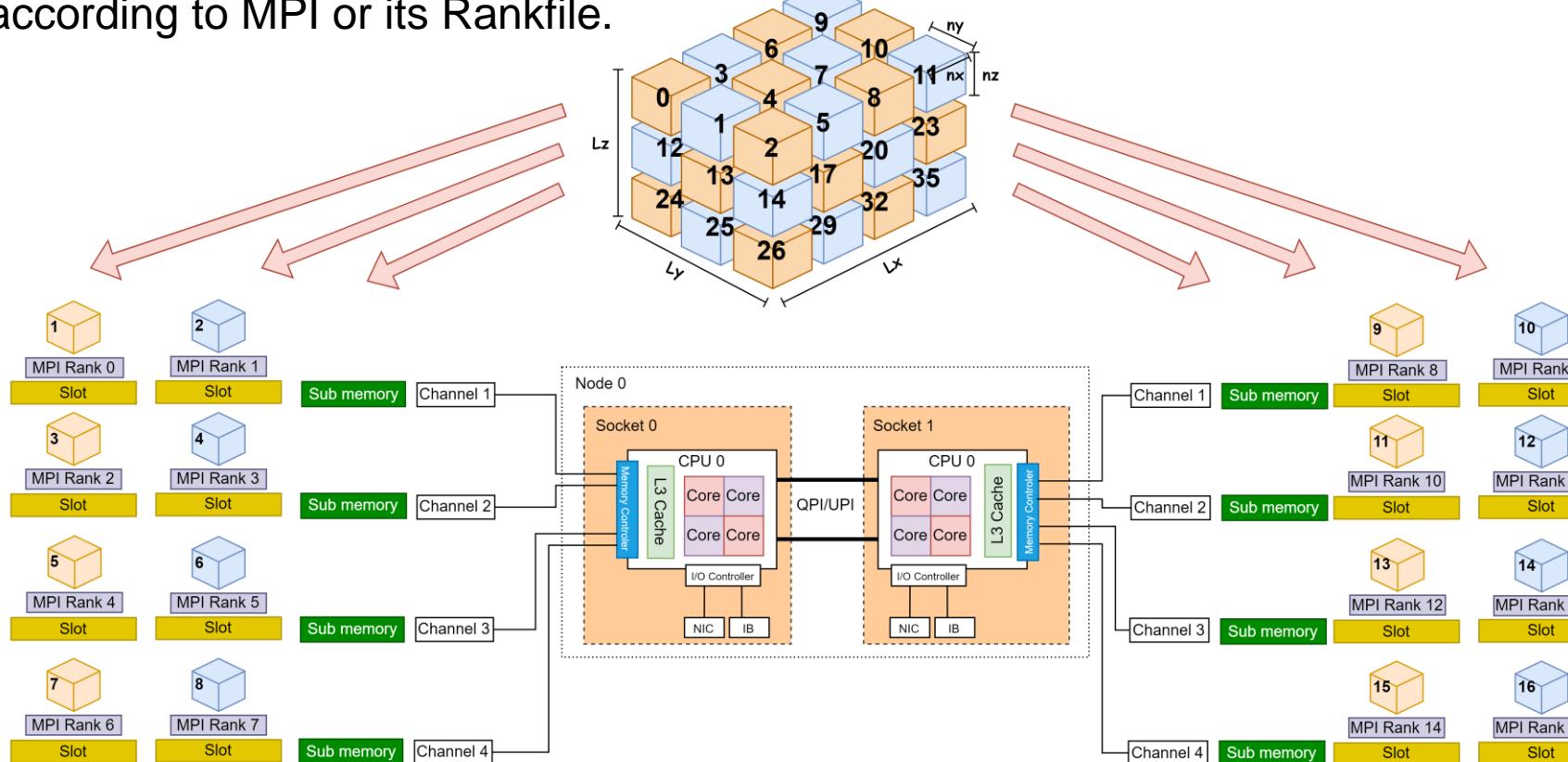


NUMA Distribution in NSCC (single node)

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 NUMA Architectures To Improve Resource Utilization (ICS 23)

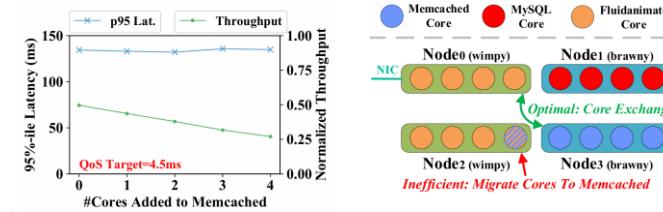


Table 2: The 95%-ile latency under different load when using different cores. The green cell indicates that the QoS target is satisfied while the red one indicates a QoS violation.

| | MySQL | | | | | | | | | |
|-----|-----------------------------|-----------------------------|--|---|---|---|---|-----------------------------|-----------------------------|-------|
| QPS | W ³ ₀ | W ⁴ ₀ | W ³ ₀ -B ³ ₃ | W ³ ₀ B ¹ ₁ | W ² ₀ B ² ₃ | W ² ₀ B ² ₁ | W ¹ ₀ B ³ ₃ | B ⁴ ₁ | B ⁴ ₀ | |
| 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 | 40.46 | 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 |

Objective

Minimize (deviation)²

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

| 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 |

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

Constraints

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

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

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

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

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

- Each NUMA node can bind at most 16 processes
- Each process is bound to exactly one NUMA node and two logical cores
- Logical cores are paired (hyper-threading)
- Each logical core can bind at most one

Output of optimization



Gurobi is a high-performance mathematical programming optimization software

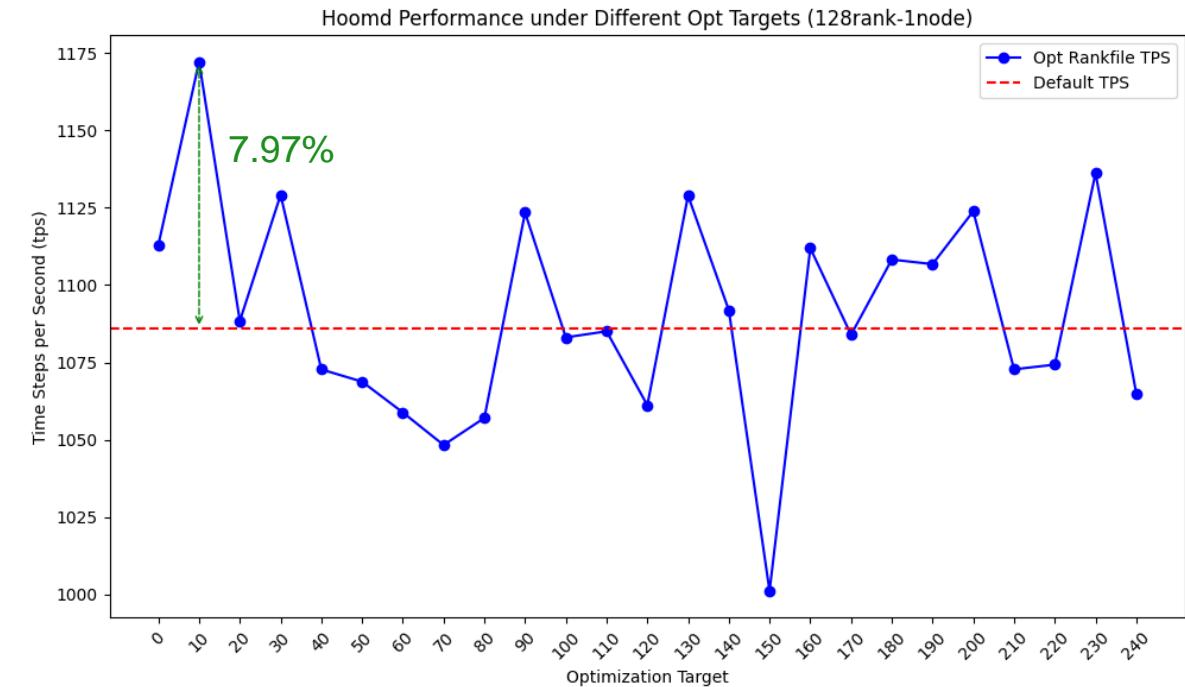
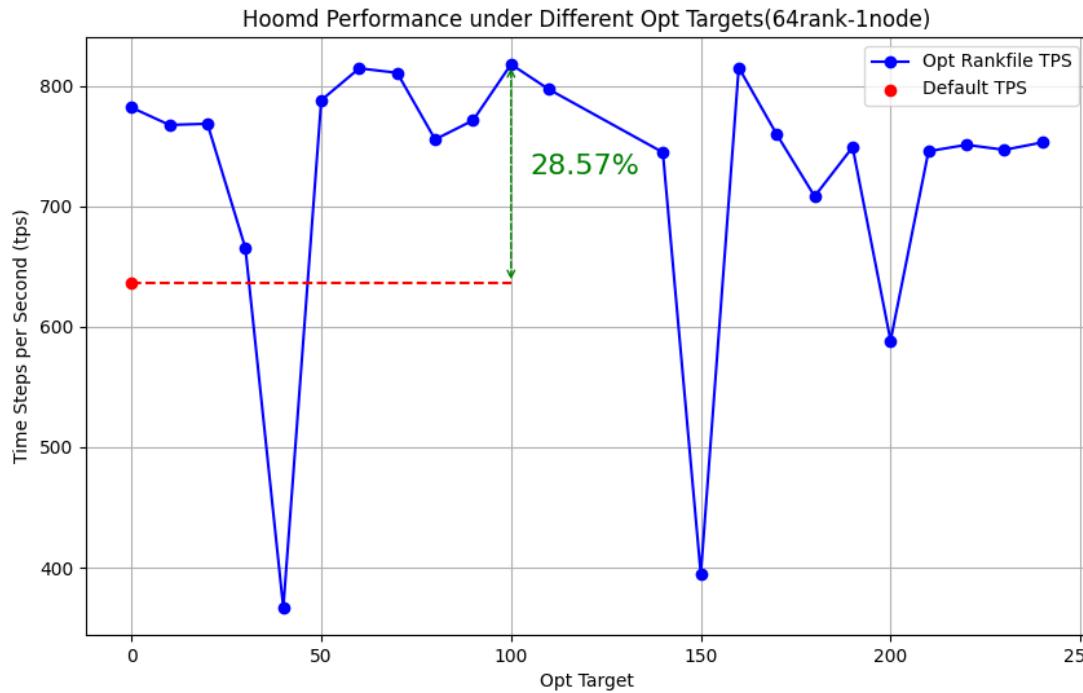
```
rank · 0=node0 · slot=102,103
rank · 1=node0 · slot=14,15
rank · 2=node0 · slot=176,177
rank · 3=node0 · slot=52,53
rank · 4=node0 · slot=100,101
rank · 5=node0 · slot=138,139
rank · 6=node0 · slot=4,5
rank · 7=node0 · slot=62,63
rank · 8=node0 · slot=224,225
rank · 9=node0 · slot=12,13
rank · 10=node0 · 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

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 < n_ranks; ++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.

Part 2

Artificial Intelligence

LitGPT Llama-2-7b finetune-full



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 |

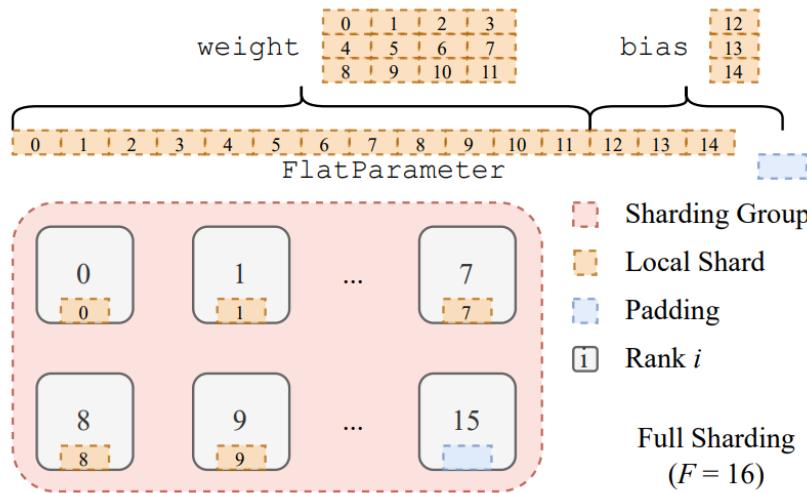
| Application Info. | |
|---------------------------|---|
| Application | Litgpt-Llama2-sft |
| Required MPI installation | openmpi/4.1.2-hpe |
| Required Libraries | litgpt=0.5.2 lightning=2.4.0 torch=2.4.1 transformers=4.44.2 |



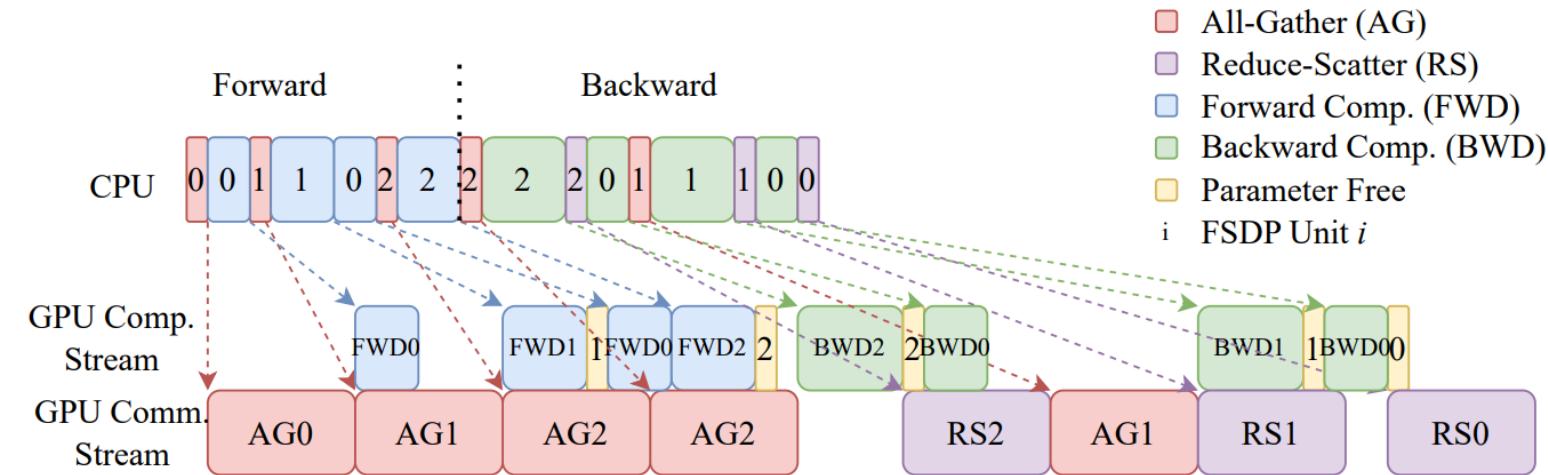
GPU task Introduction

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

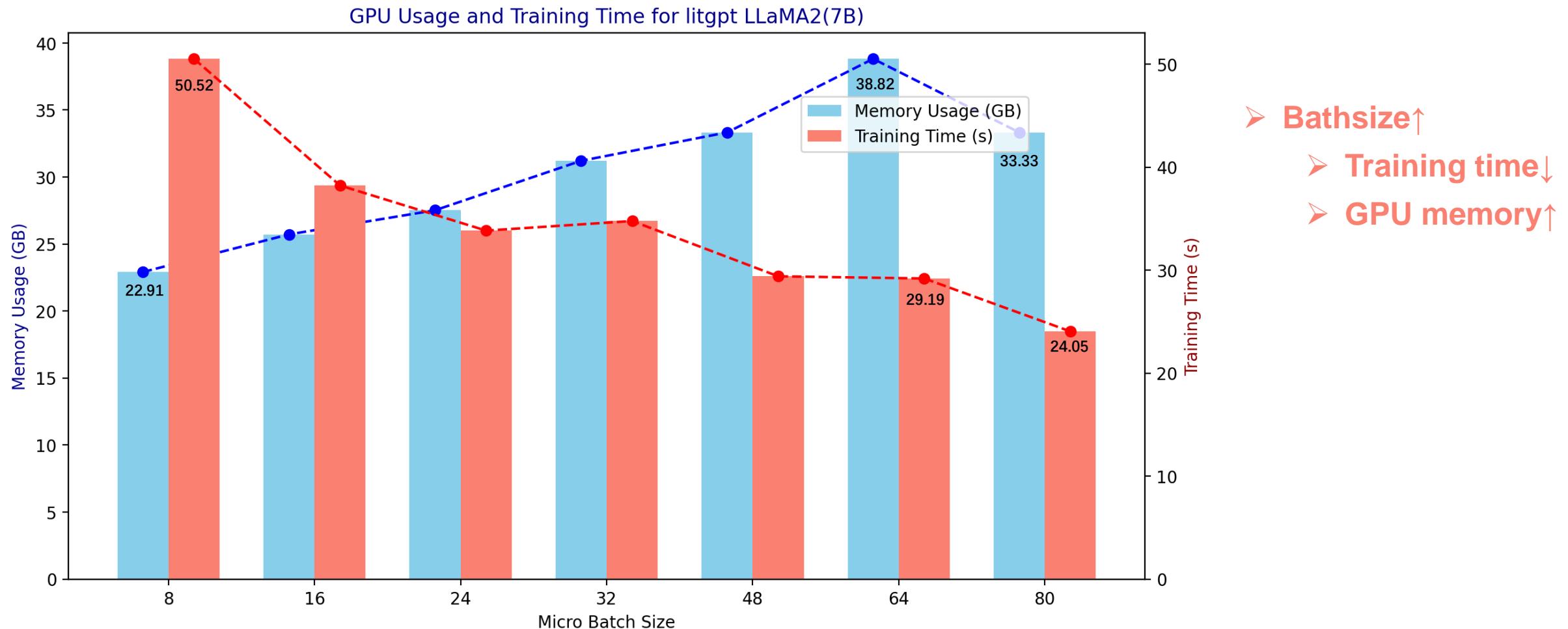
Sharding params opt grade



Overlap Communication and Computation



GPU Benchmark in NSCC(A100 40G)



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

Profile between different batch size

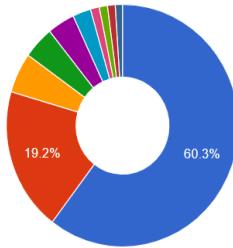
micro batch size = 8



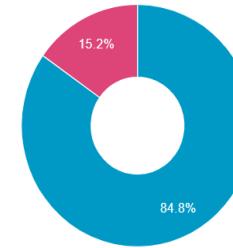
Kernel View

All kernels Top kernels to show 10

Total Time (us) ②



Tensor Cores Utilization ②



Not Using Tensor Cores
Using Tensor Cores

➤ Bathsize↑

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

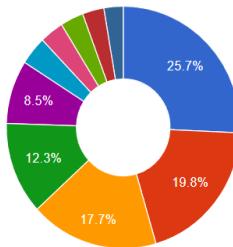
micro batch size = 64



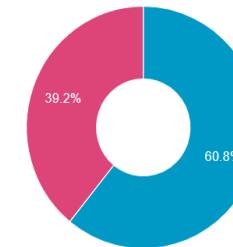
Kernel View

All kernels Top kernels to show 10

Total Time (us) ②



Tensor Cores Utilization ②



Not Using Tensor Cores
Using Tensor Cores

Nvidia A100 Tensor Core Usage

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

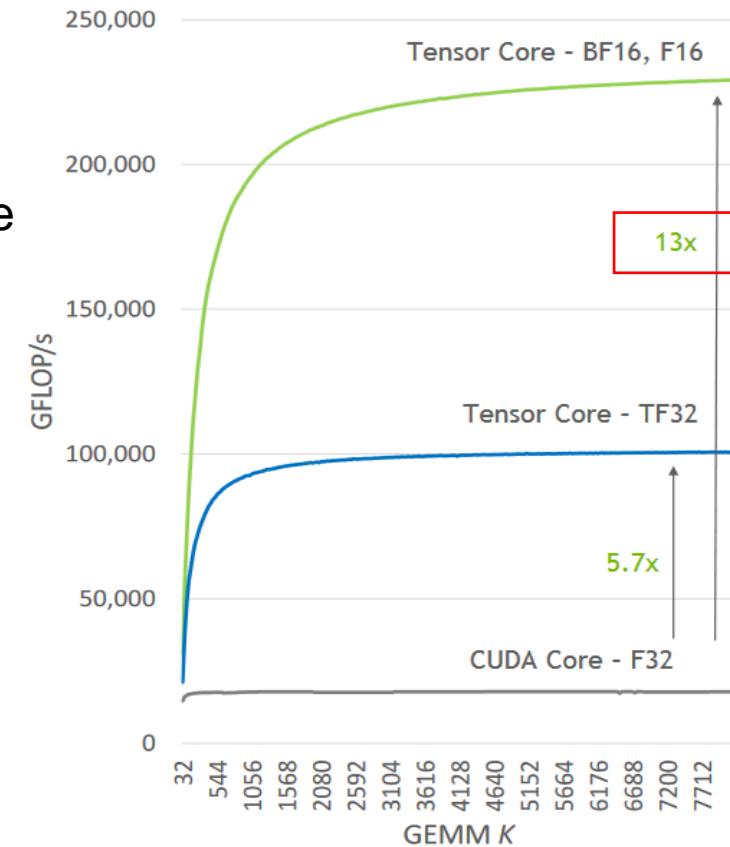


- Memory bandwidth Utilization ↑
- Memory access Locality↑
- Data throughput↑

Just give it enough data to compute

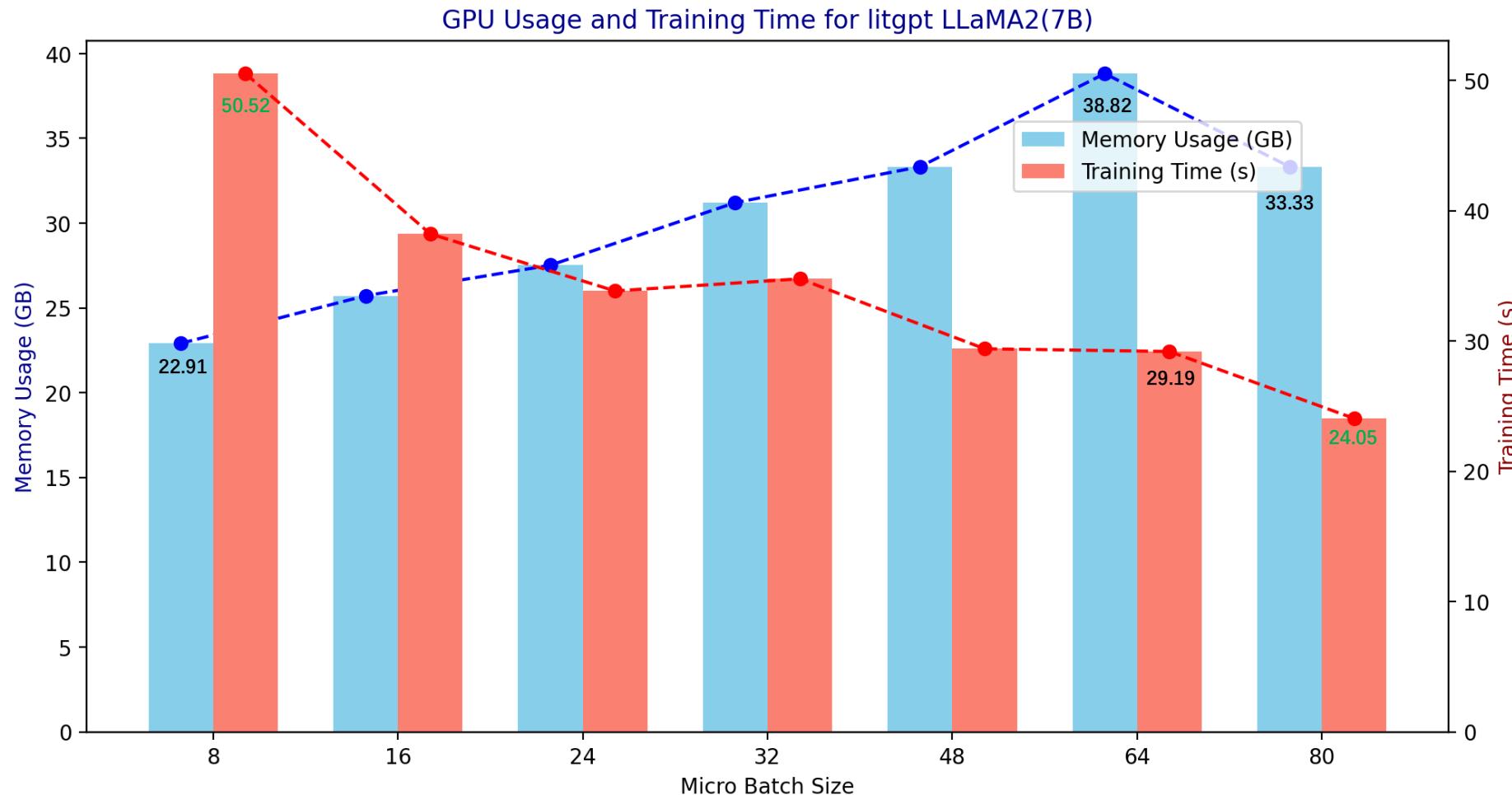
- Vectorized memory access
- Asynchronous memcpy

Mixed Precision Floating Point



CUTLASS Performance

Batch Size Benchmark



- Batchsize↑
- Training time↓
- GPU memory↑

Training Time speedup about 2.10x for 2 nodes 8 GPUs

Different strategy

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

```
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
)
```

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

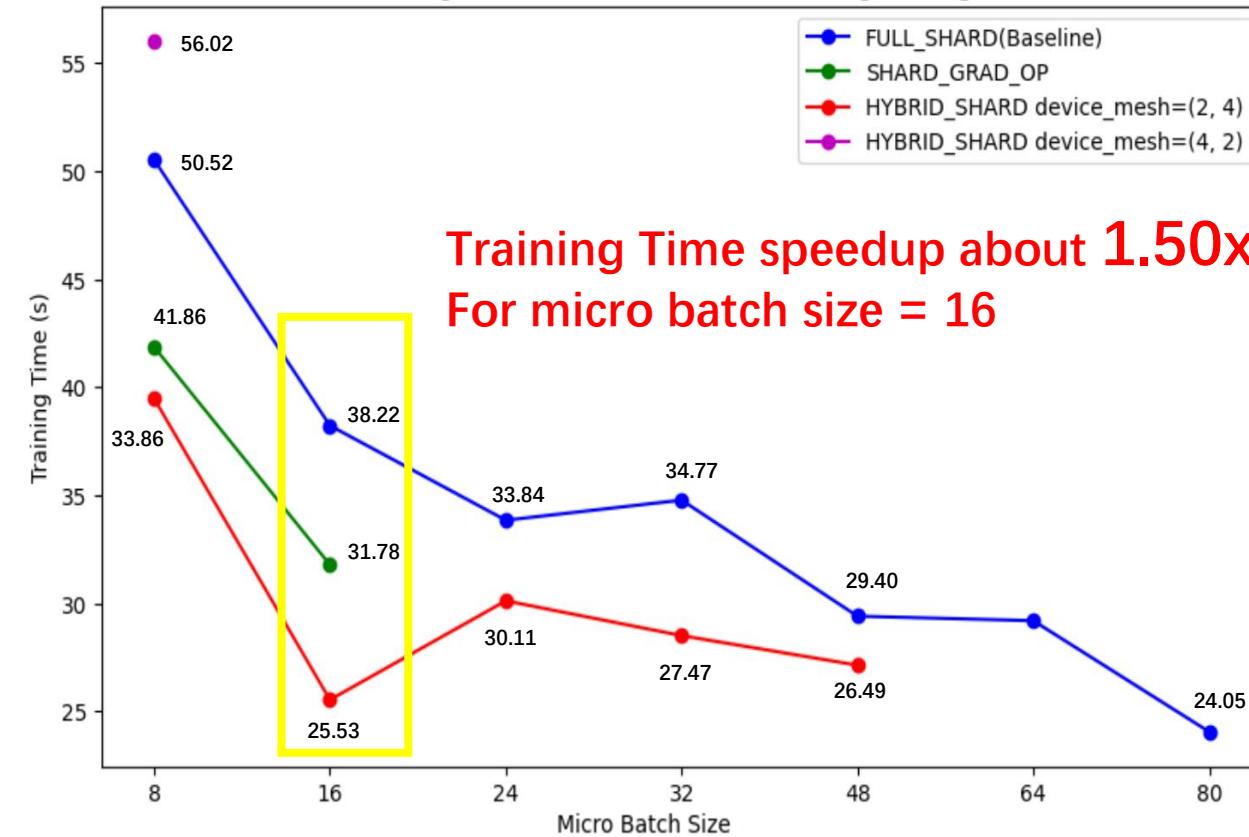
- memory utilization ↑
- communication cost ↑

litgpt src: finetune/full.py#L100

[What if the batch size is increased simultaneously?](#)

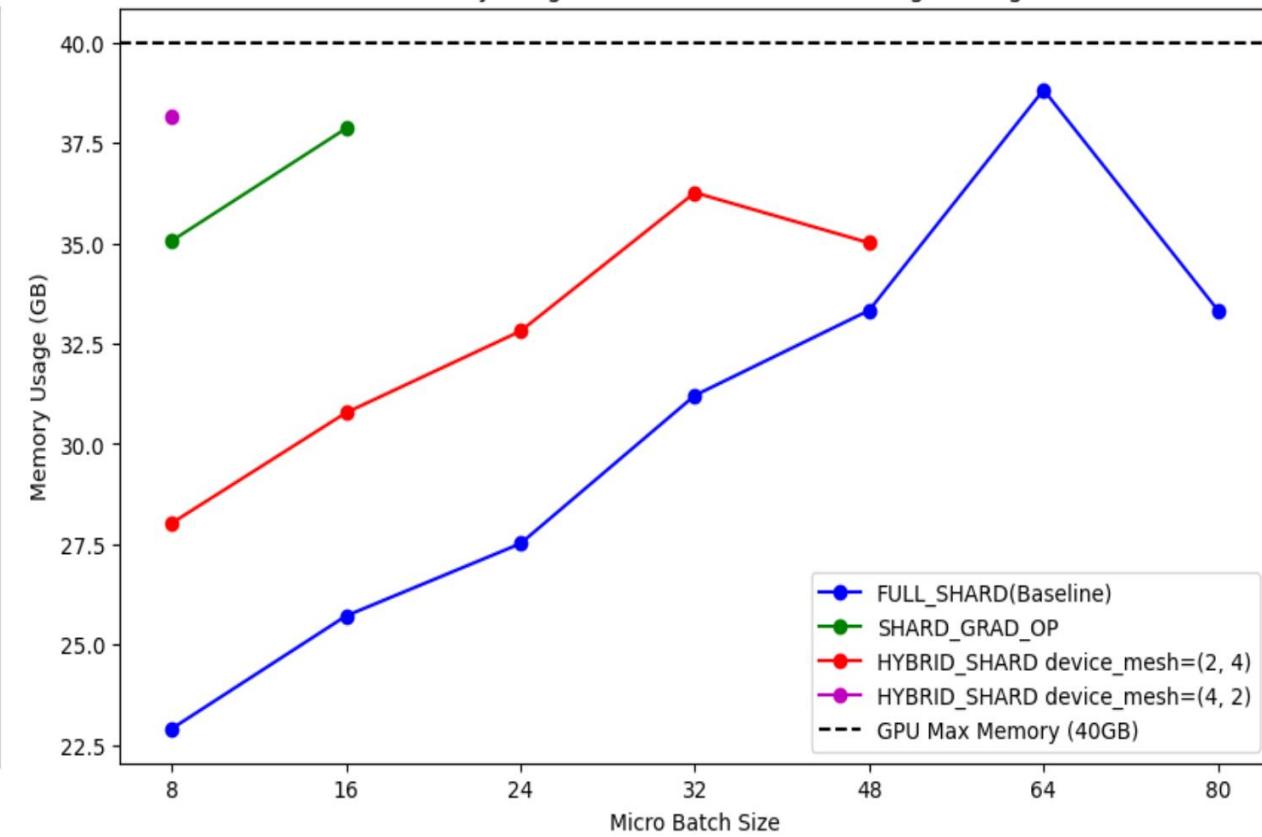
Optimization of FSDP Sharding Strategy and Parameters

Training Time for Different FSDP Sharding Strategies



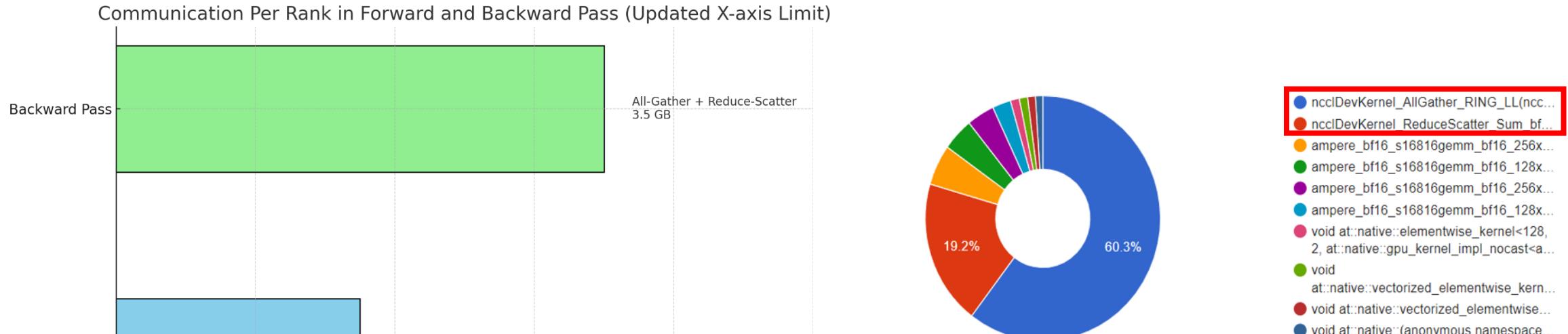
Training Time speedup about 1.50x
For micro batch size = 16

GPU Memory Usage for Different FSDP Sharding Strategies



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

Total Parameters and Sharding Across GPUs

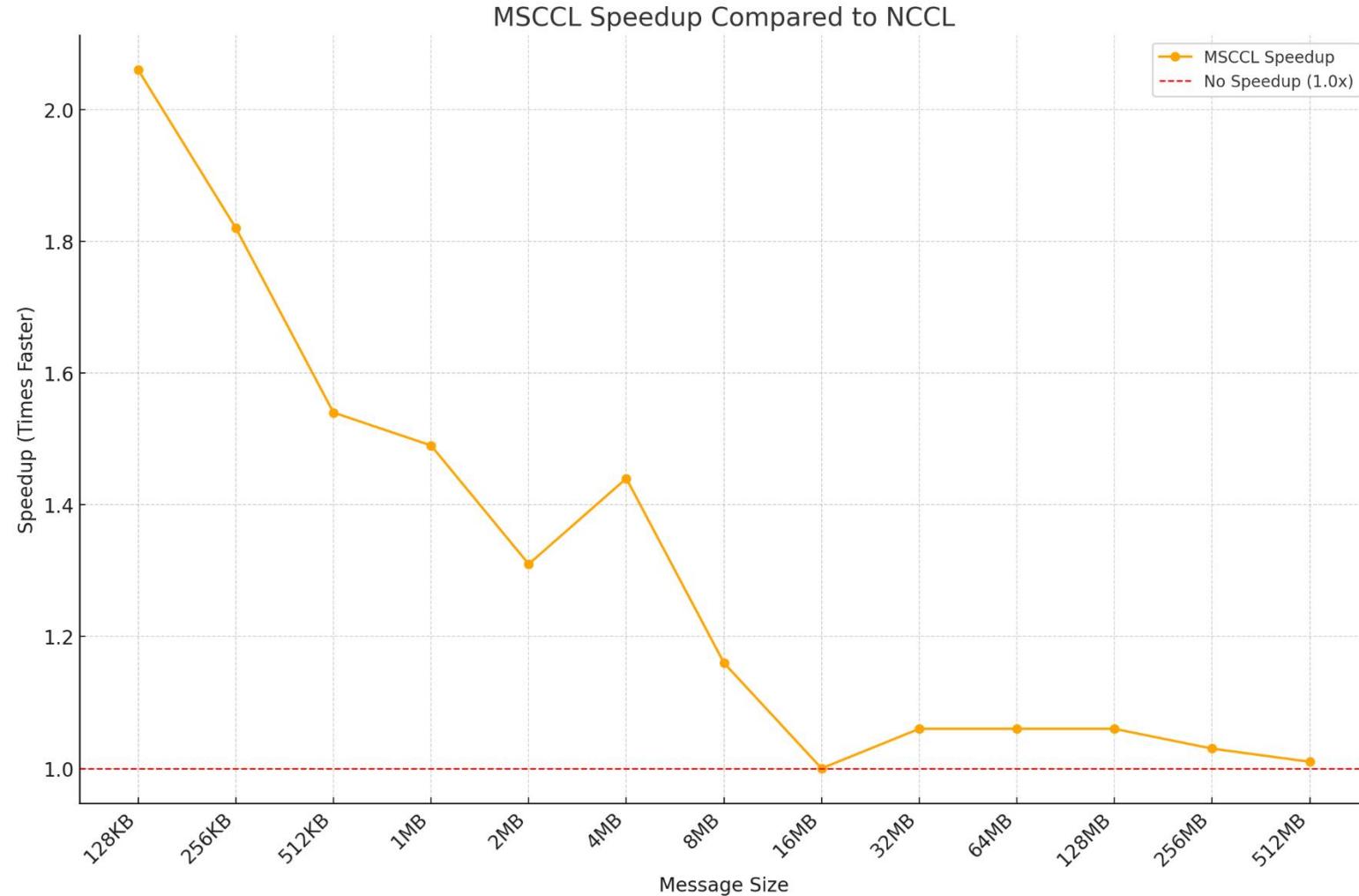


communication data size is large

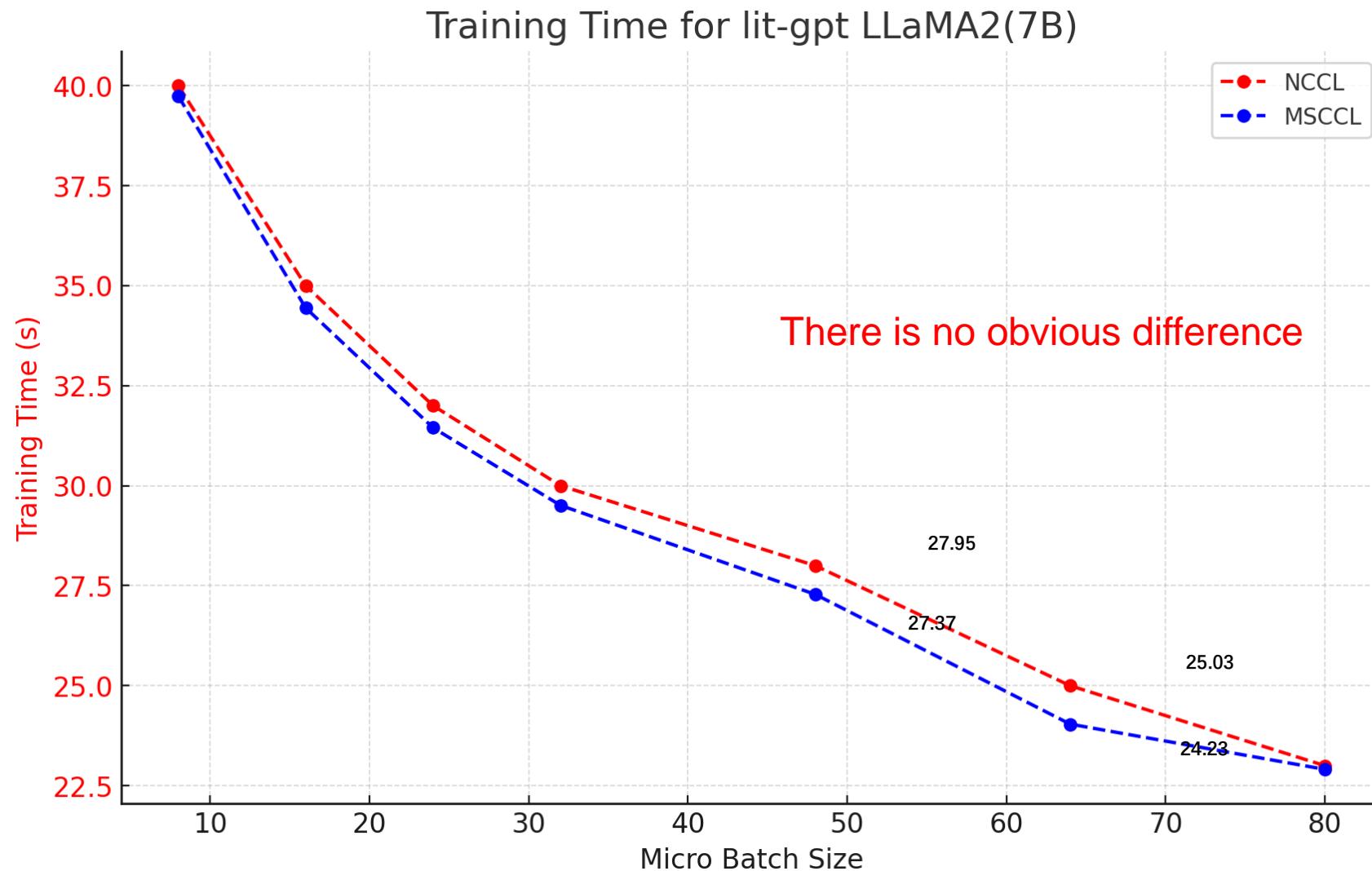


Reduce communication time

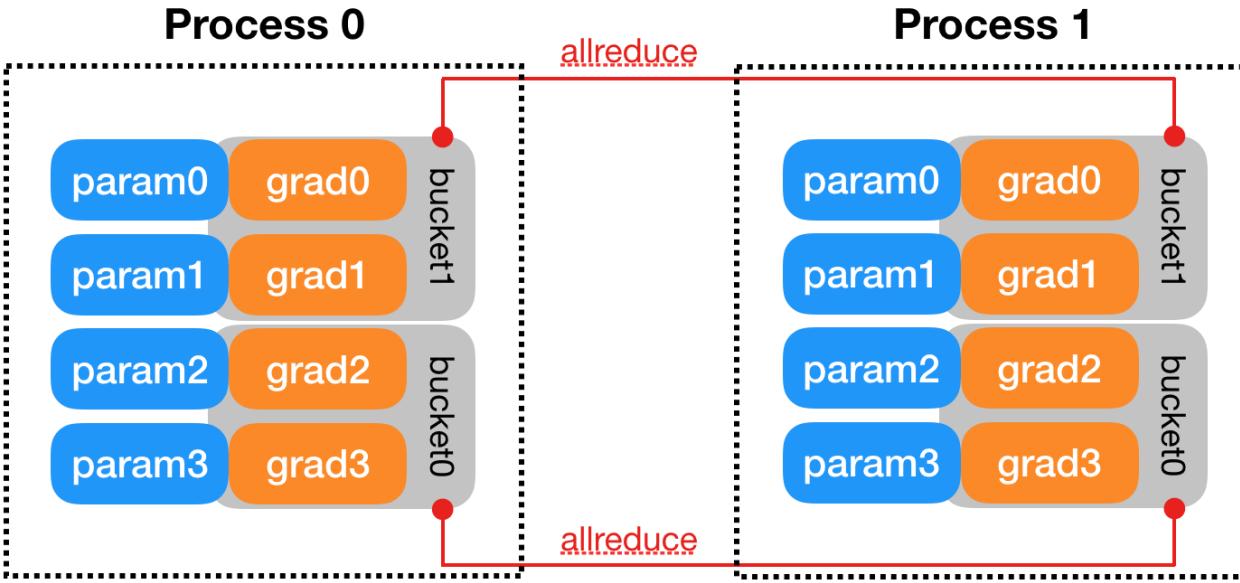
MSCCL Speedup Compared to NCCL



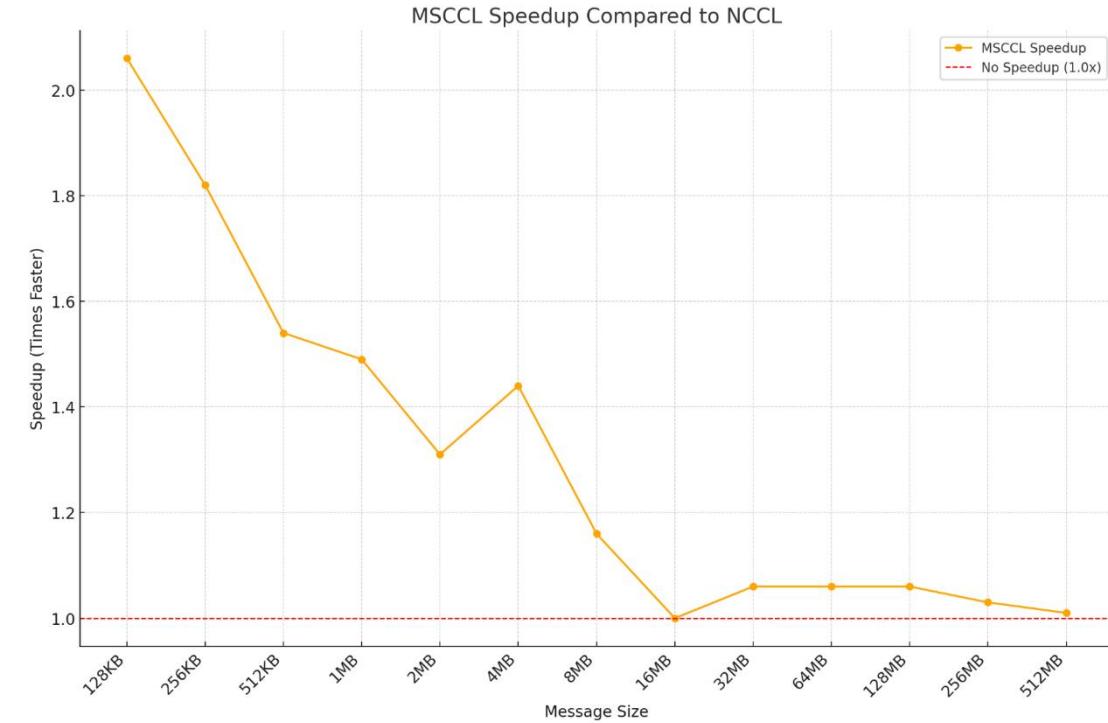
Result in NCCL and MSCCL



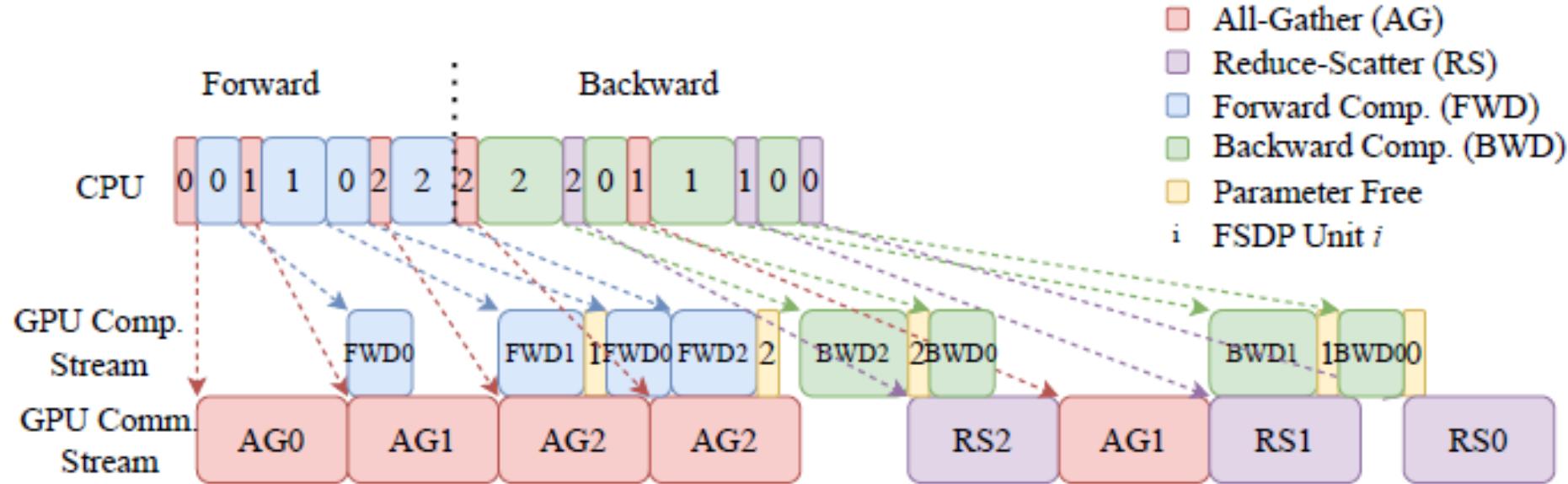
MSCCL Optimization at Default Bucket Size



- 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.

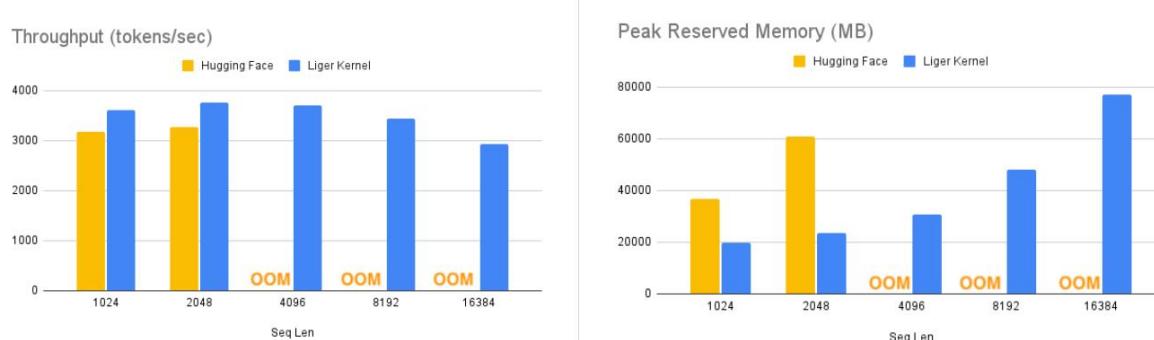
Faster Computational Operator

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:

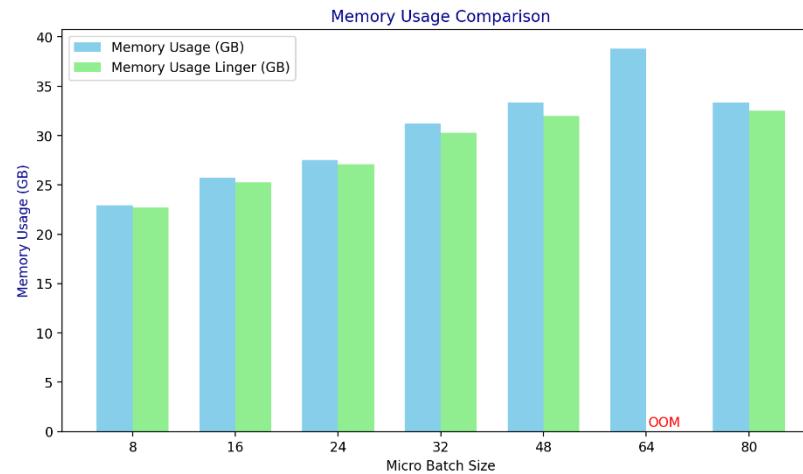
Looking forward to the integration or participated in development of litgpt train

Faster Computational Operator

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