A Performance Analysis Of Parthenon/Phoebus

Ankush Jain

Mochi – Aug 10, 2022

Parthenon – Aug 11, 2022

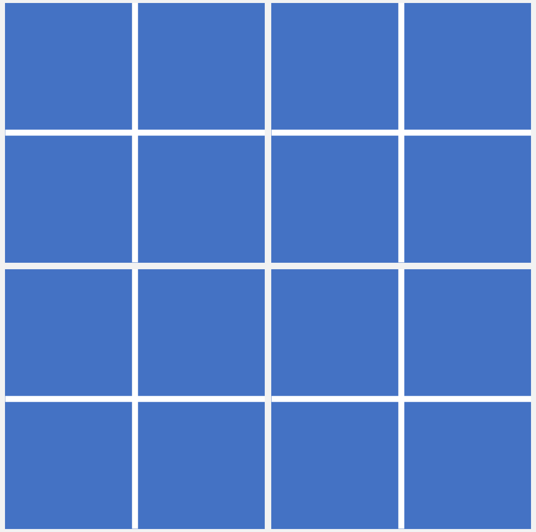
Goals

1. Understand why Parthenon performs the way it does
   - AMR: Adaptive Mesh Refinement
2. Have at least an intuitive model of how different inputs affect perf
3. Use these to identify promising avenues
   - For interesting load-balancing/optimizing problems
   - Potentially using in-network capabilities

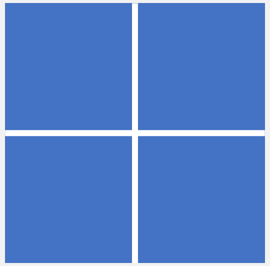
- *Where we are: advanced stages of 1. and 2. Early stages of 3.*

Parthenon - Block-Based AMR

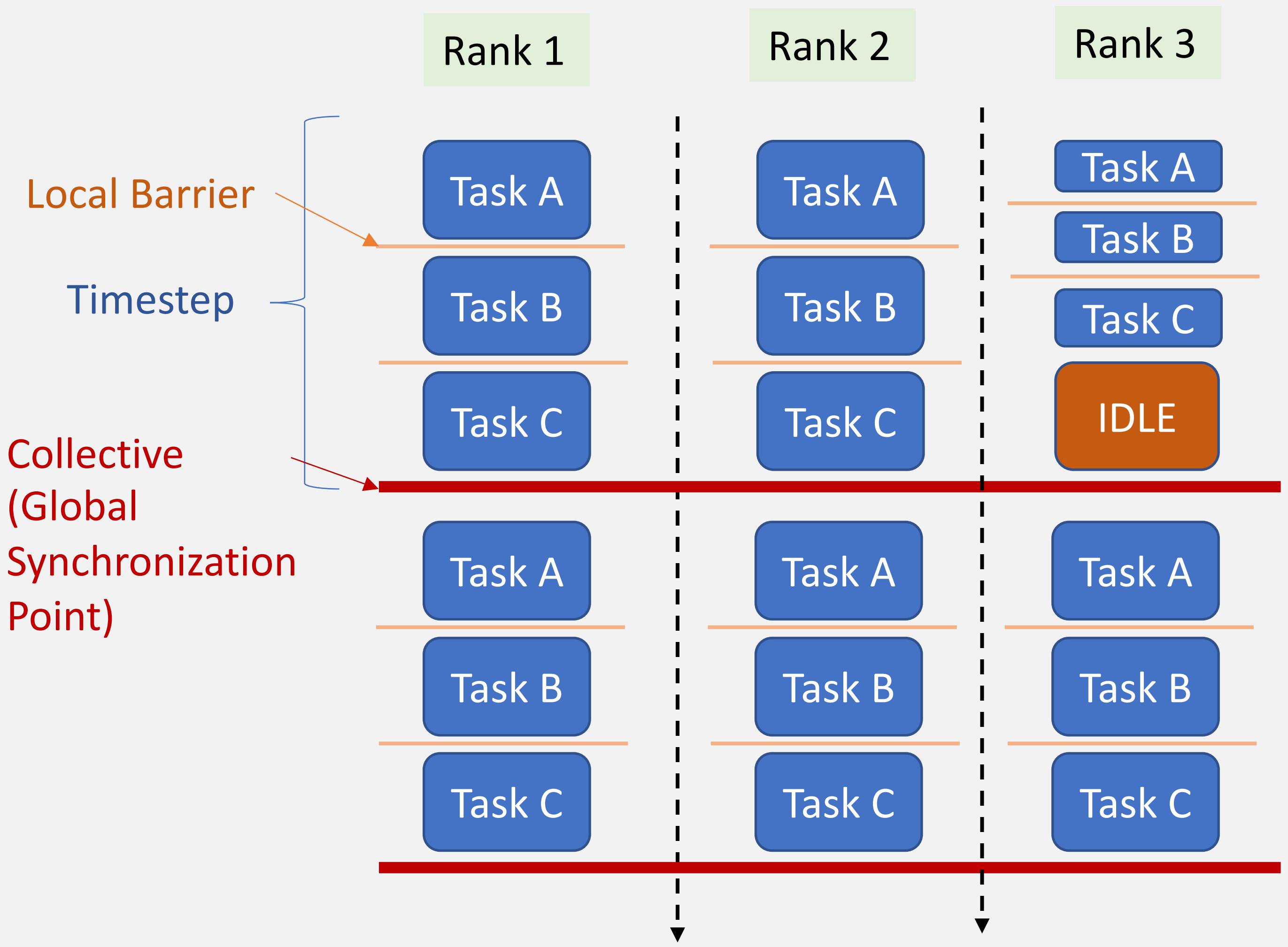
- A 3D mesh is simulated
  - Say 4x4x4 cells
- Divided into meshblocks
  - Uniform and contiguous
  - I.e. always 2x2x2
- Meshblocks are uniformly allocated to ranks
- Meshblocks get refined and derefined
  - New meshblocks are always the same size
  - i.e. 2x2x2



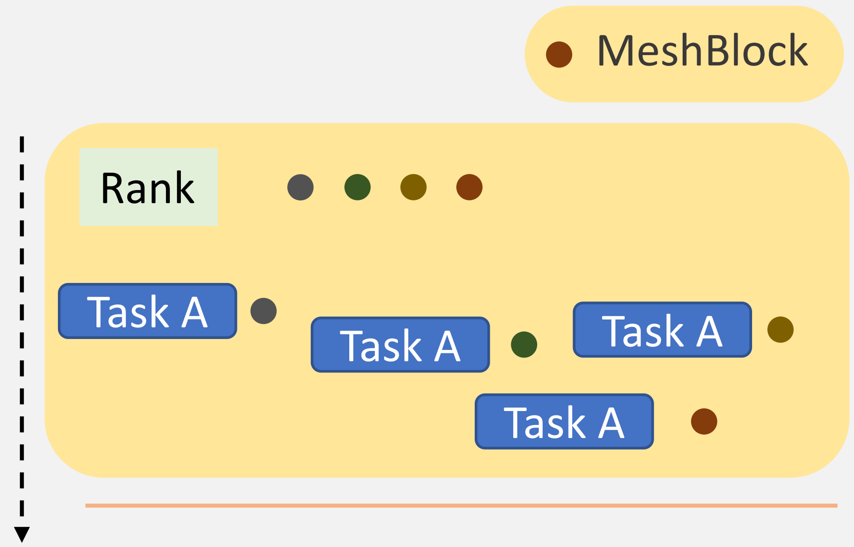
4x4 mesh
decomposed into
4X 2x2 meshblocks



Execution Model: Tasks and Barriers



Asynchronicity (example):



- **Tasks**: Compute and/or Network kernels
  - (*per meshblock*)
- A task schedule forms a **timestep**
- Limited amounts of **asynchronicity**

Workload: Actual Task Schedule

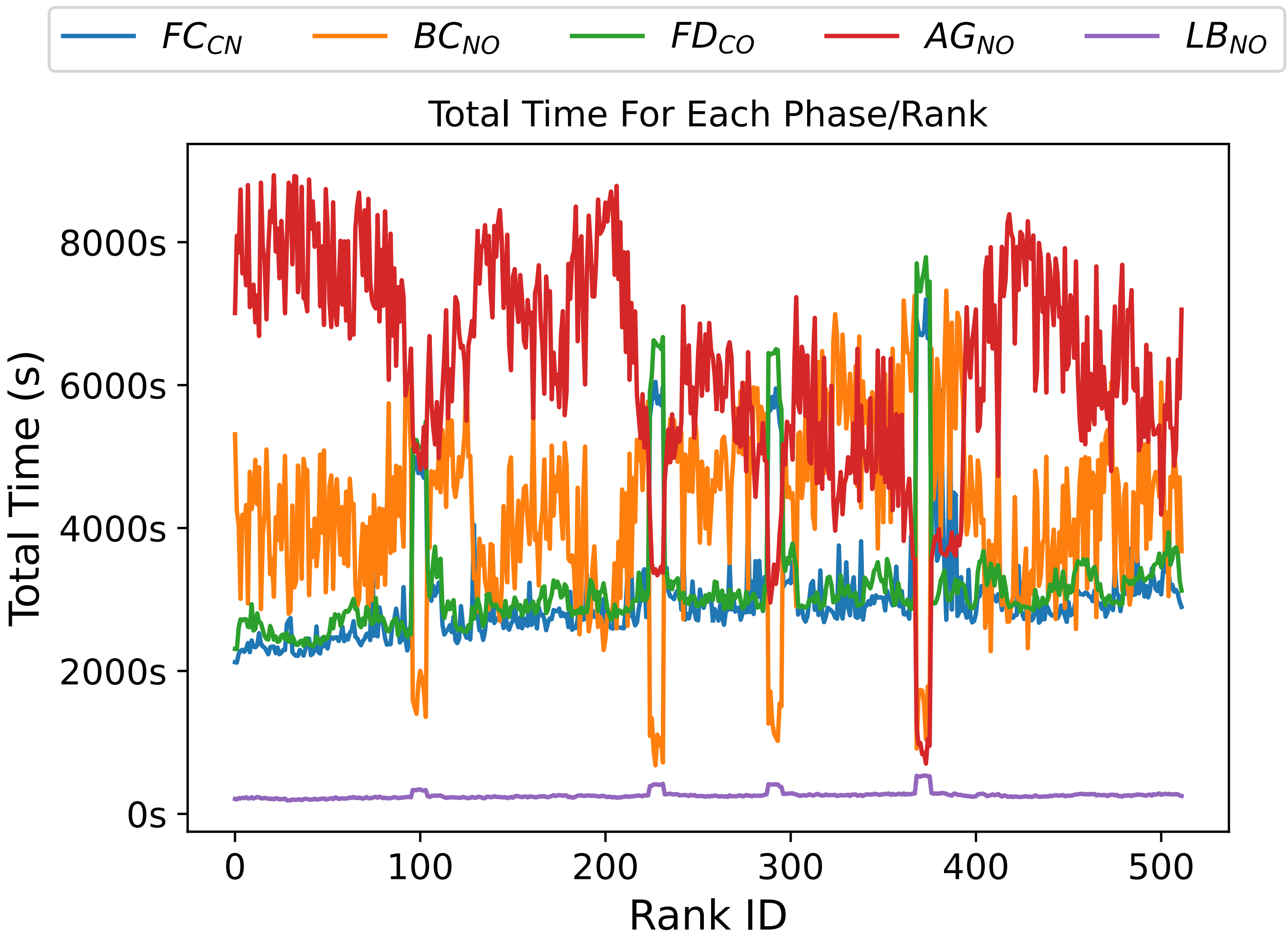
*Subscripts. **CN**: Compute + Network Task. **NO**: Network Only. **CO**: Compute Only.*

FC_{CN}	Flux Correction	Some messages exchanged (MPI_Send/Wait). Some compute.
BC_{NO}	Boundary Communication	O(100K) messages exchanged every timestep (at 512 ranks)
FD_{CO}	FillDerived (Compute Kernel)	A **non-uniform** compute kernel per meshblock
AG_{NO}	AllGather (MPI Collective)	A collective (+ global sync point). Gathers load info etc.
LB_{NO}	Load Balancing	Only runs if load info gathered indicates variance > threshold

There's another collective/synchronization pt at the end of timestep, but it's not as interesting. (First collective takes care of stragglers created by this timestep. Not much work happens after)

Some rule of thumb numbers

- **Deck**: *Blast Wave 3D /Phoebus/Parthenon*
- Mesh = 128^3, Meshblock = 16^3
  - Initial Meshblock Count: 512
  - 512 CPU-only ranks. 32 Wolf nodes.
  - MPI over PSM.
- 30,000 timesteps approx, 4.5 hours.
  - 500ms/timestep on average
  - (Timestep gets slower over time with refinement)



Rough total time breakdown:
(No need to remember this)

FC: 2500s
BC: 4000s
FD: 2500s
AG: 8000s
LB: 250s

Total: 17,250s for 30,000 timesteps

Takeaways:

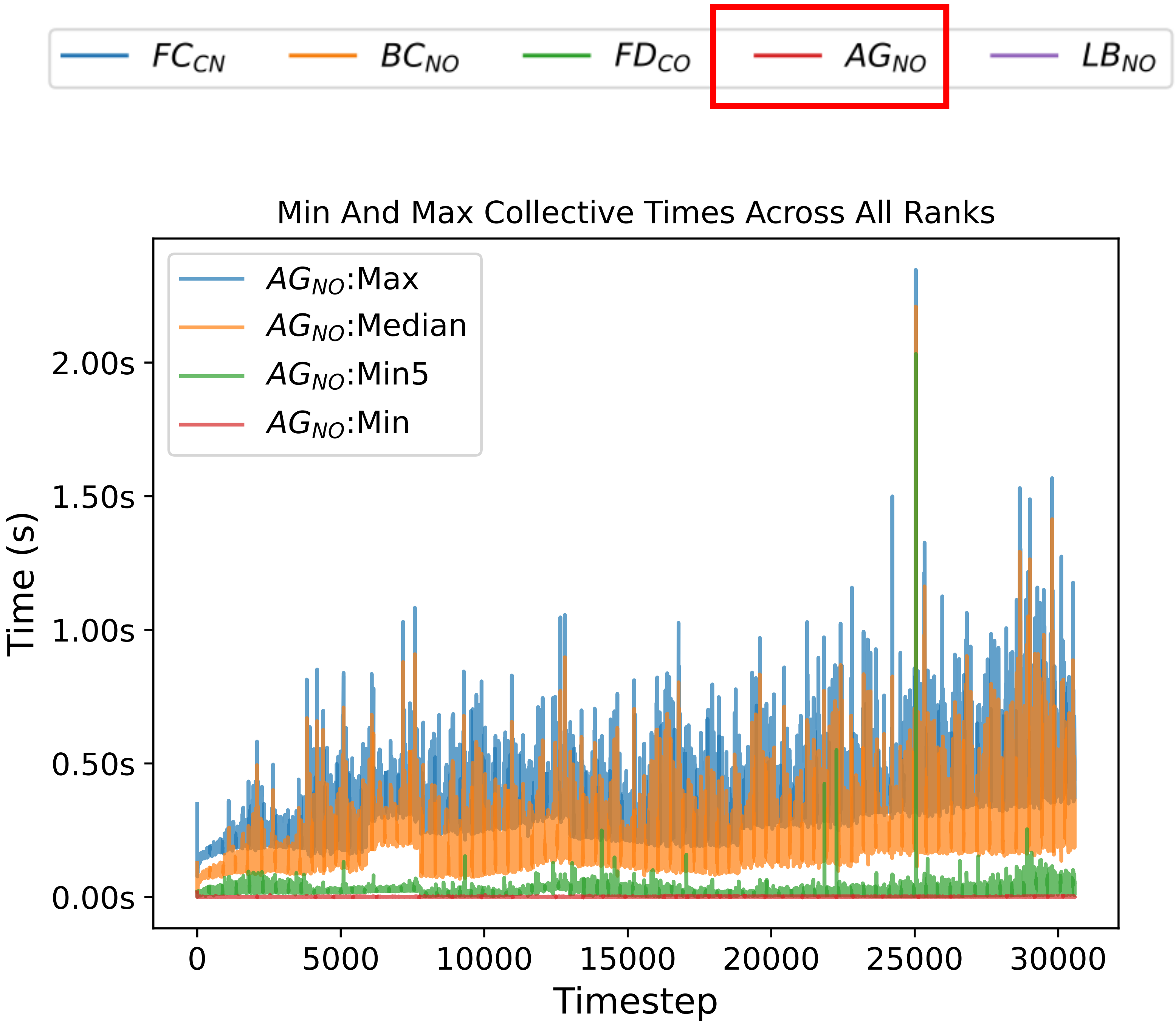
1. LoadBalancing time is negligible
   (Nothing about quality)
2. FC_{CN} and FD_{CO} variance not much
   (At least in aggregate)
3. *BoundaryComm* has a lot of variance
4. *AllGather* has a lot of variance

Scoping Problem Before Proceeding

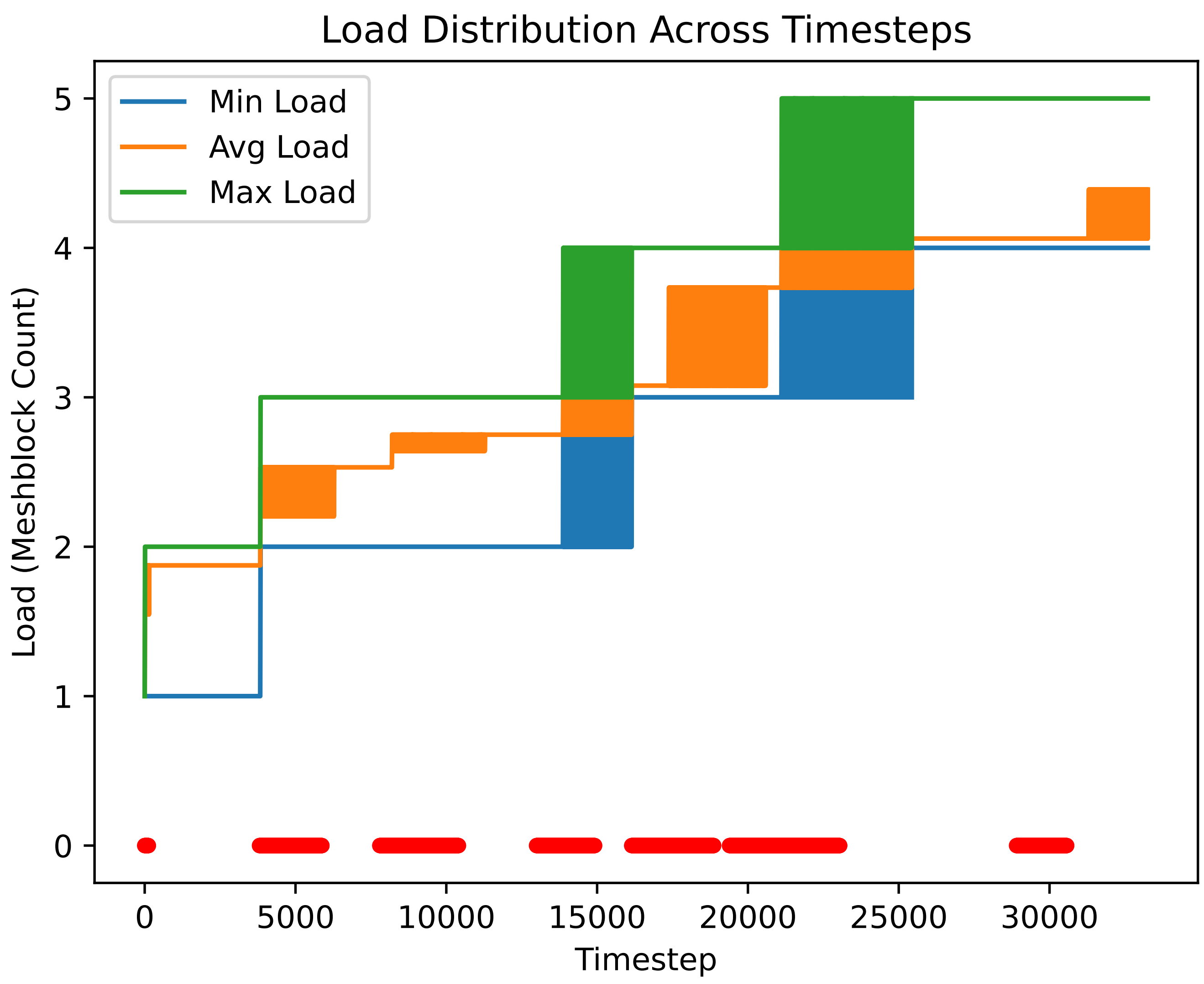
- Not concerned with: *compute kernel implementation, caching, prefetching, data layout etc*
- Concerned with: *load balancing, communication, scheduling*
  - i.e. aspects that can be solved with better approaches along these dimensions

AllGather takes 8000s of 17000s. Why!?

- *Collectives are expensive, but not that expensive*
- **Hypothesis**: Poor load balance
- **Data for:** wide variance in time spent by each rank at collective
  - Ranks that finish early wait for stragglers
- **Data against:** work allocation in terms of meshblocks
  - Meshblock allocation is reasonably well-balanced
- **Conclusions:**
  - 1. Meshblock count is a reasonable, but imperfect proxy for load per rank
  - 2. We possibly need to implement communication differently



FC_{CN}
BC_{NO}
FD_{CO}
AG_{NO}
LB_{NO}
Min And Max Collective Times Across All Ranks
AG_{NO}:Max
AG_{NO}:Median
AG_{NO}:Min5
AG_{NO}:Min
Time (s)
2.00s
1.50s
1.00s
0.50s
0.00s
0
5000
10000
15000
20000
25000
30000
Timestep



- Red dots: load-balancing event
- Meshblock distribution across ranks is not perfect, but as good as it gets
- When avg load is 2.7 meshblocks:
  - most ranks will get 3
  - some will get 2

Next: Analyze Phase-Wise Breakdown Of Time

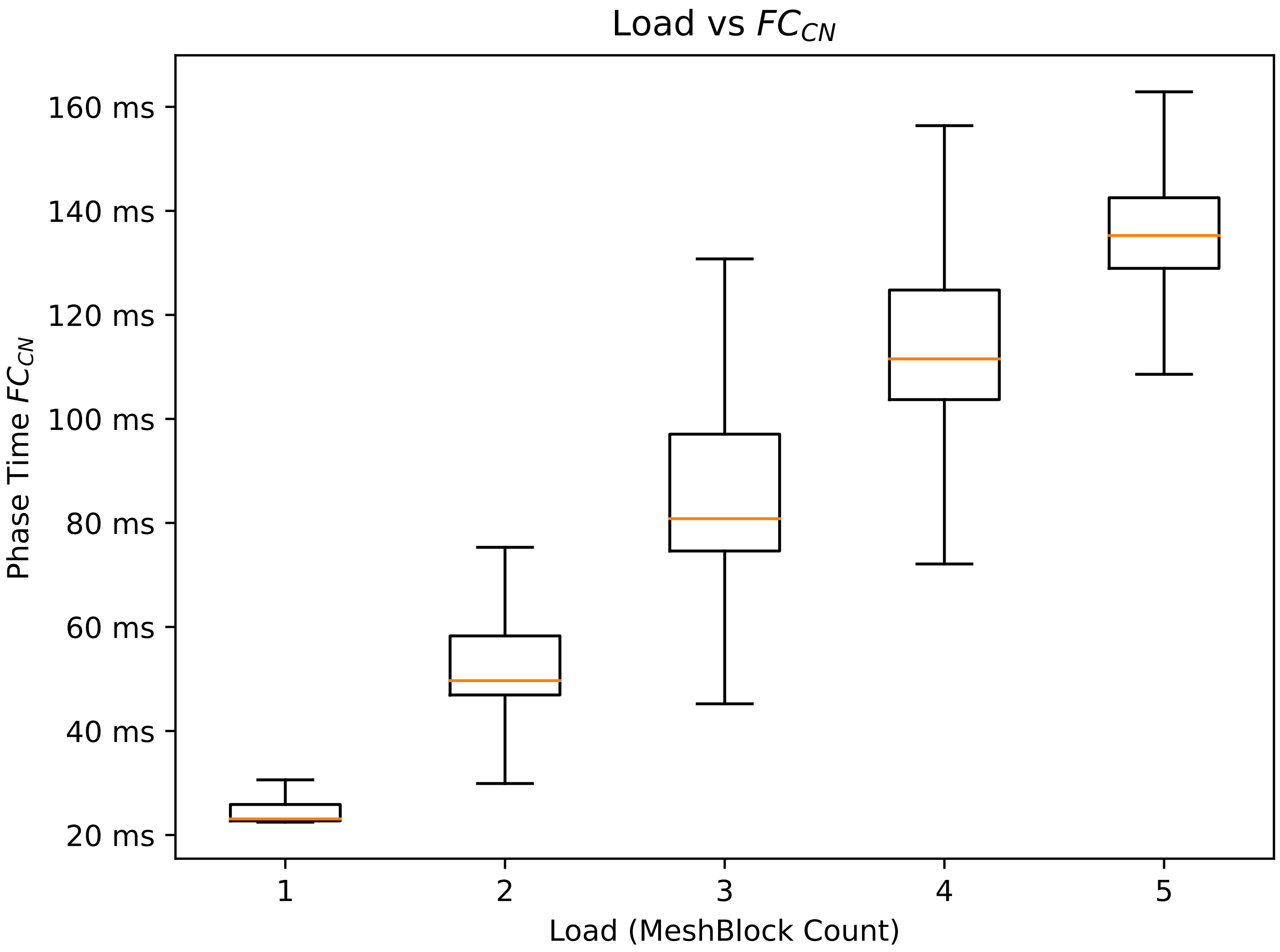
- Load reasonably balanced in terms of meshblock count
  - But meshblock count is an imperfect proxy for timestep time
  - Hence the stragglers
- Next:
  - Look at each phase (i.e. task) in a timestep separately
  - How well does meshblock count explain ‘phase time’?
  - What other factors do you need to explain ‘phase time’?

FluxCorrection

Total Time spent: 2500/17000s

Takeaway: for this task, meshblock count is *a useful but imperfect signal for load*

(Too much variance for a given meshblock count)



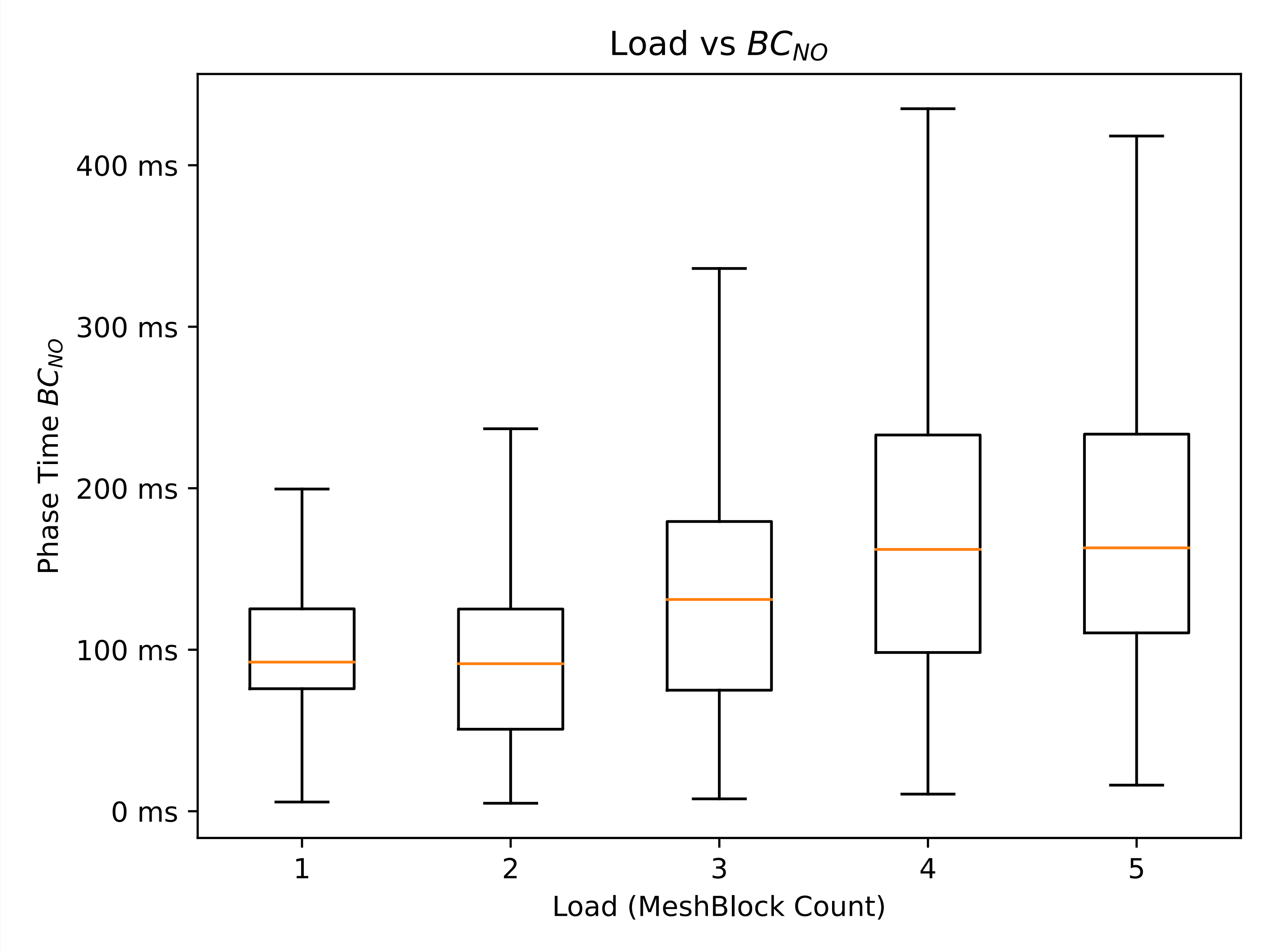
BoundaryComm

Total Time Spent: 4000/17000s

Takeaway: for *task latency*, meshblock count is *not a useful signal at all***.**

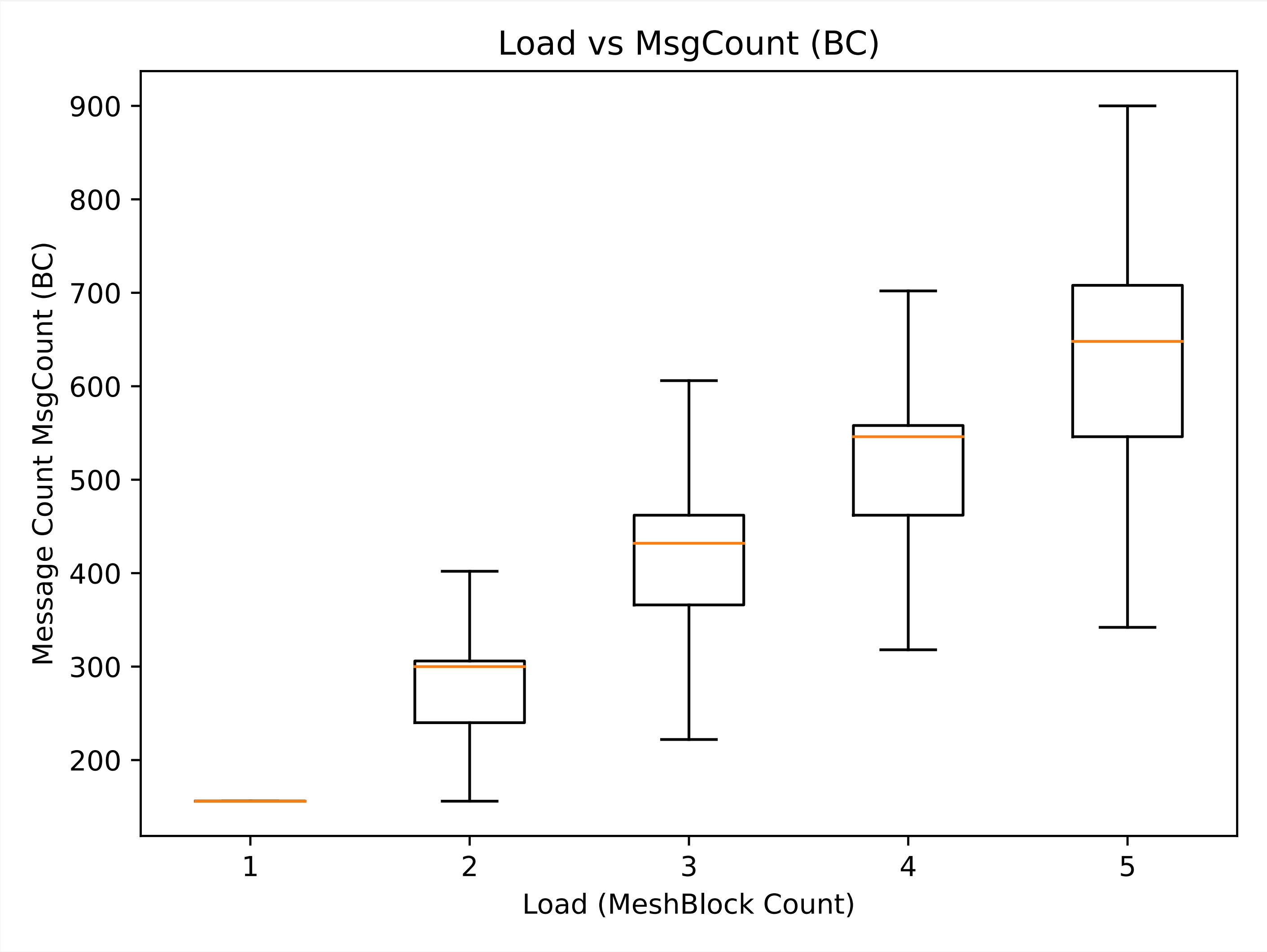
Next, let's break this down into:

1. **Load vs MessageCount**
2. **MessageCount vs Phase Time**



BoundaryComm

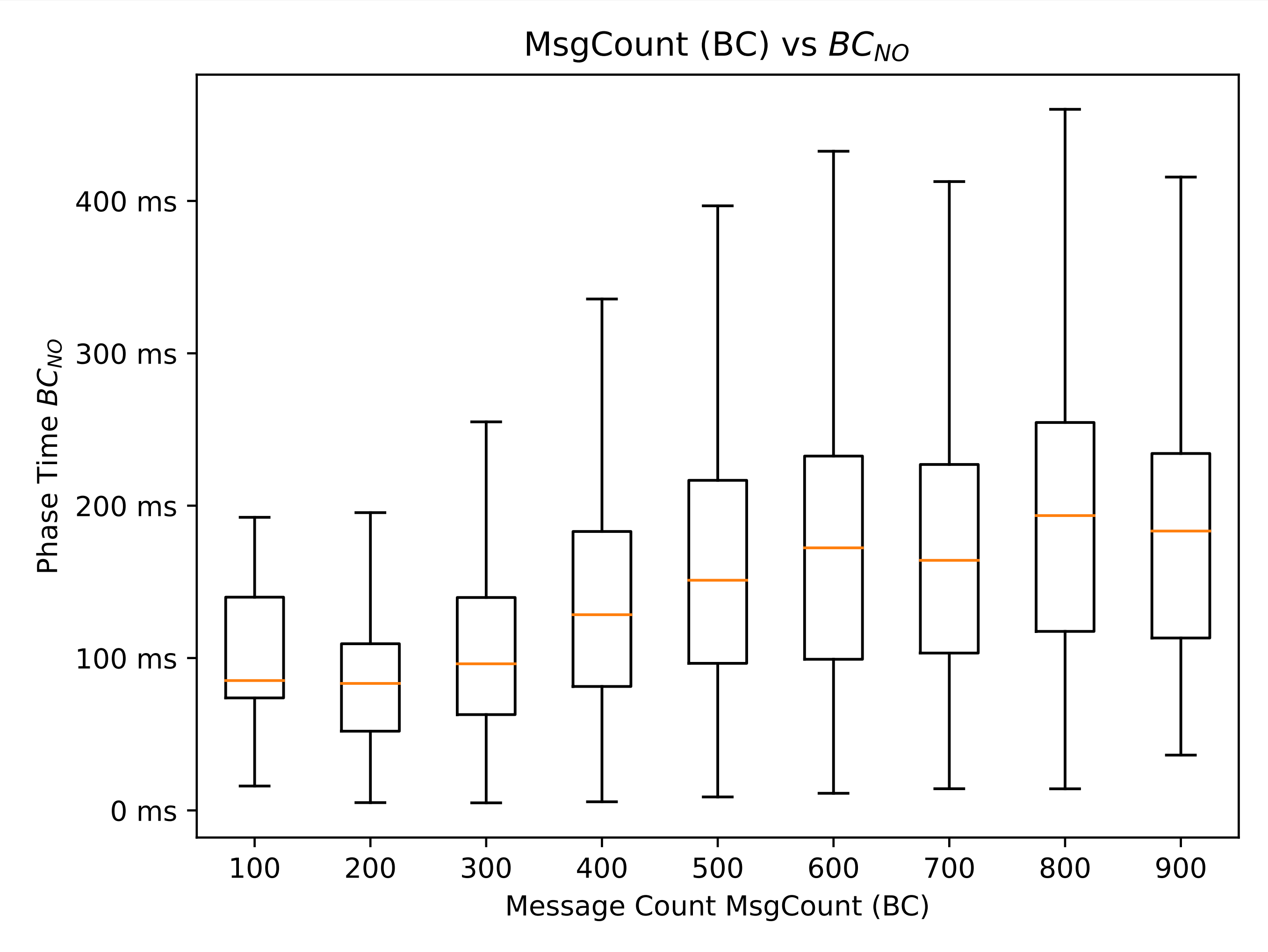
1. A lot of messages are exchanged (500/rank * 500 ranks = 250K/round)
2. MeshBlockCount is a good/imperfect signal for MsgCount



BoundaryComm

Takeaway:

1. MeshBlockCount is a useful and imperfect signal for MsgCount
2. *MsgCount is a poor signal for round latency*

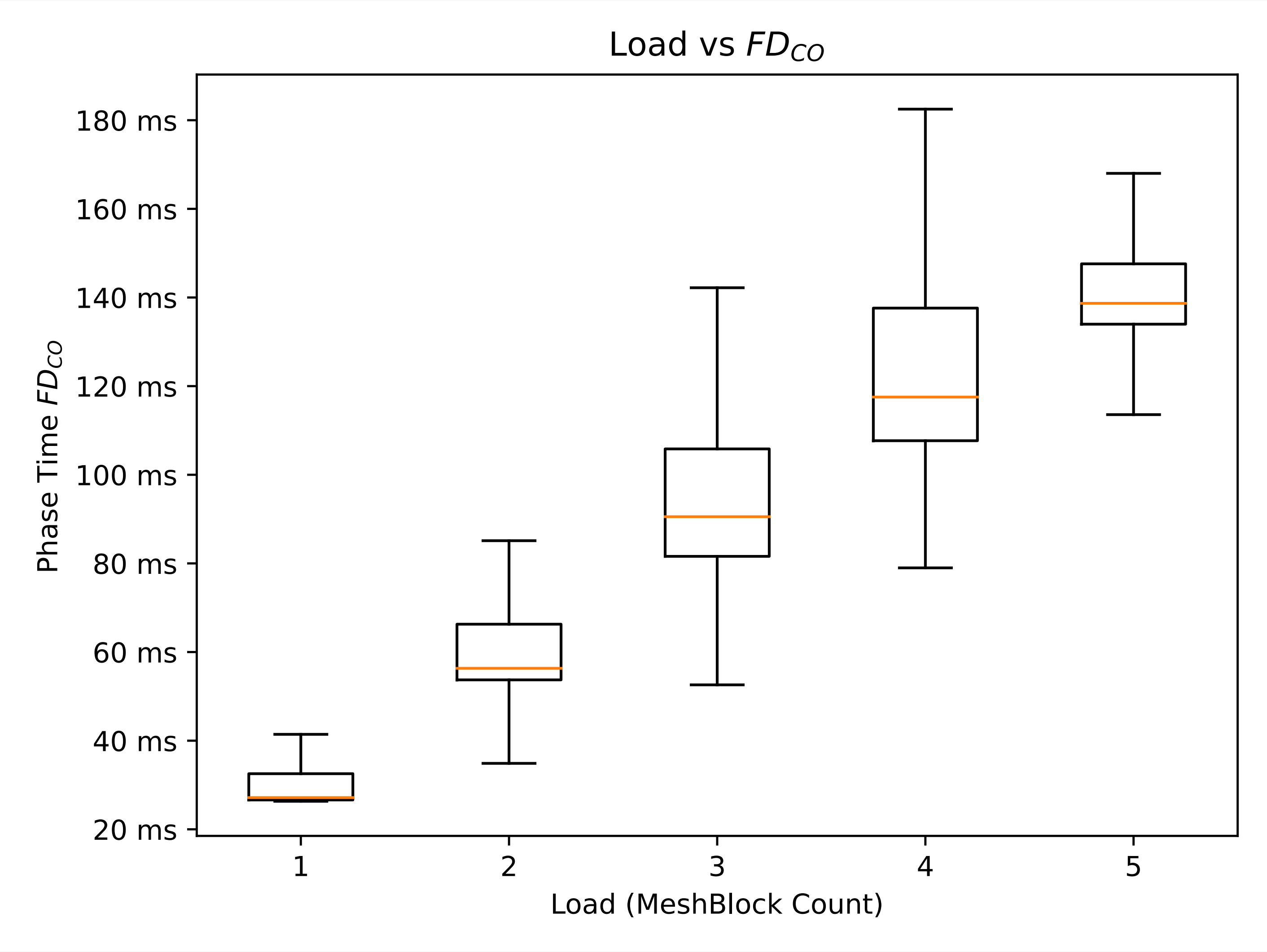


FillDerived

Total Time Spent:
2500/17000s

Takeaway: for this task, meshblock count is *a useful but imperfect signal for load*

(Too much variance for a given meshblock count)



MPI_AllGather

Total Time spent: 4000s to 8000s (out of 17000s)

Takeaway: *time spent in MPI_AllGather is inversely proportional to time spent in FC + BC + FD.*

Independently this phase is not interesting.

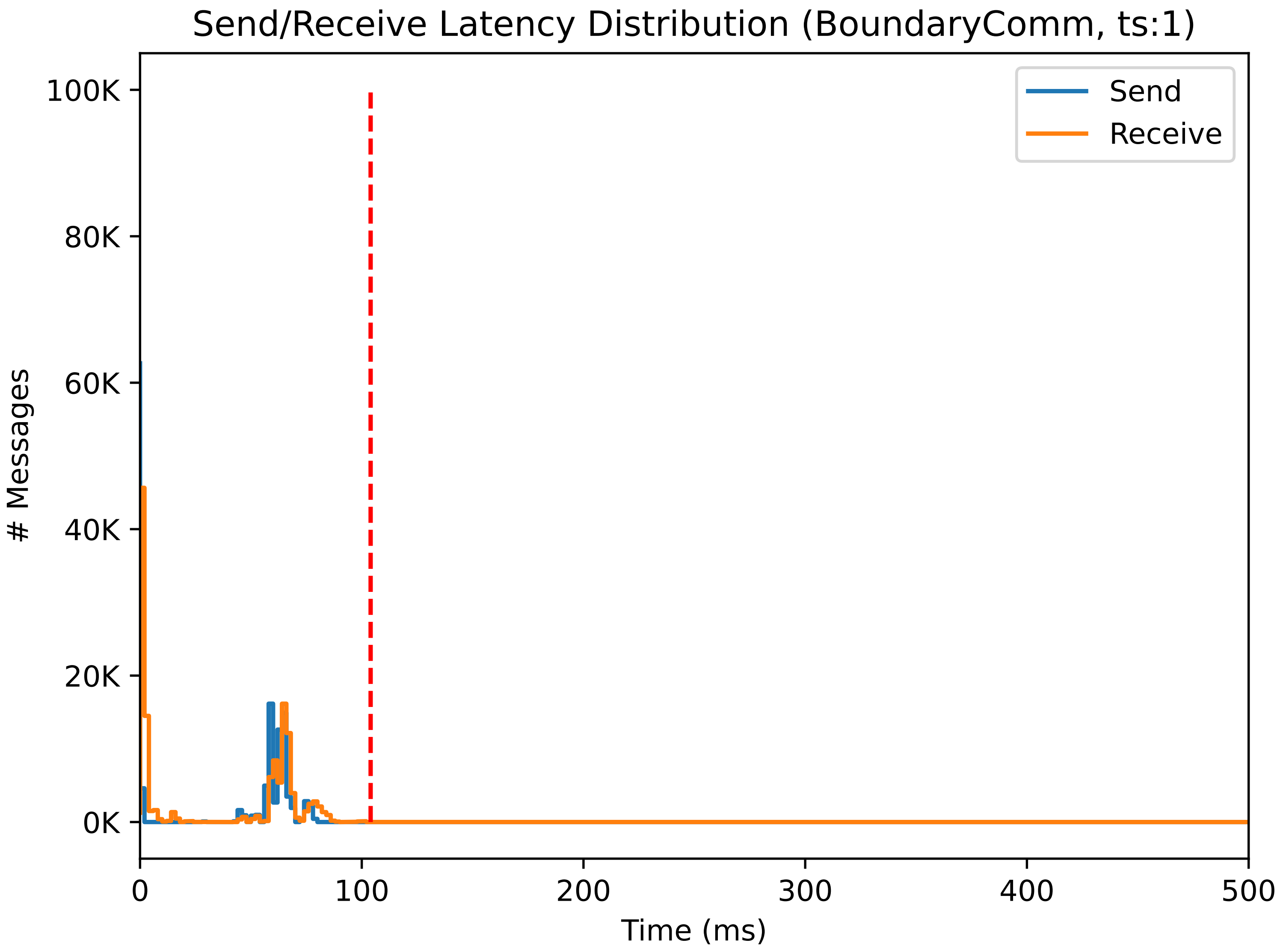
LoadBalancing

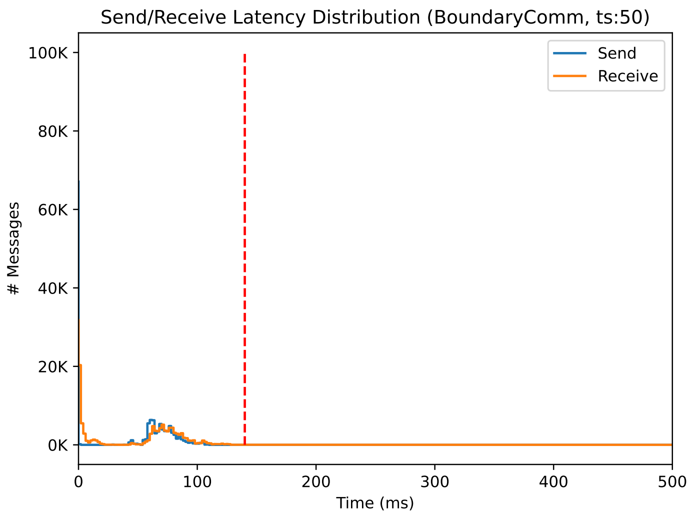
Total time spent: 250s-500s out of 17000s.

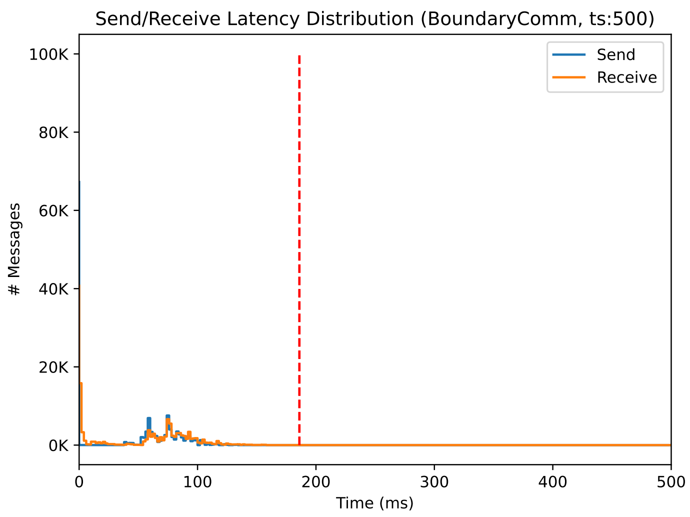
This phase is too minor to be interesting in its own right.

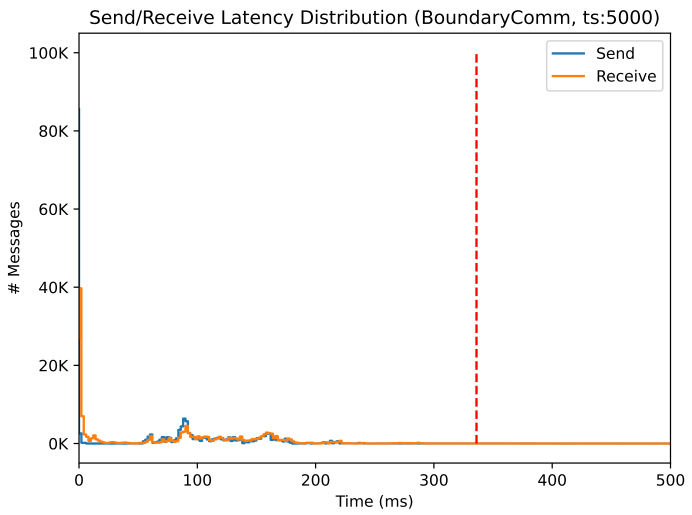
What’s Going On With BoundaryComm

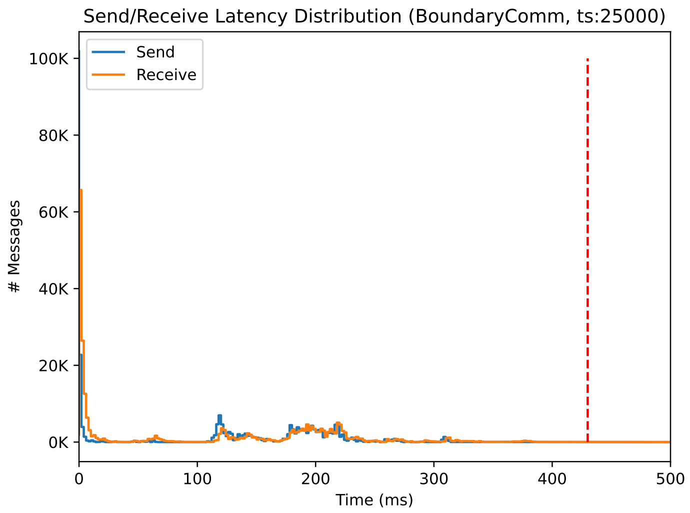
- O(100,000) messages are exchanged every timestep.
- **Tail latency** pattern exists
- Gets worse over time. (More meshblocks, => more messages)
- The set of all asynchronously exchanged messages forms a collective.
    - *Can we optimize/schedule them collectively?*


Send/Receive Latency Distribution (BoundaryComm, ts:1)
Send
Receive
100K
80K
60K
40K
20K
0K
Messages
0
100
200
300
400
500
Time (ms)


Send/Receive Latency Distribution (BoundaryComm, ts:50)
Send
Receive
Messages
100K
80K
60K
40K
20K
0K
0
100
200
300
400
500
Time (ms)

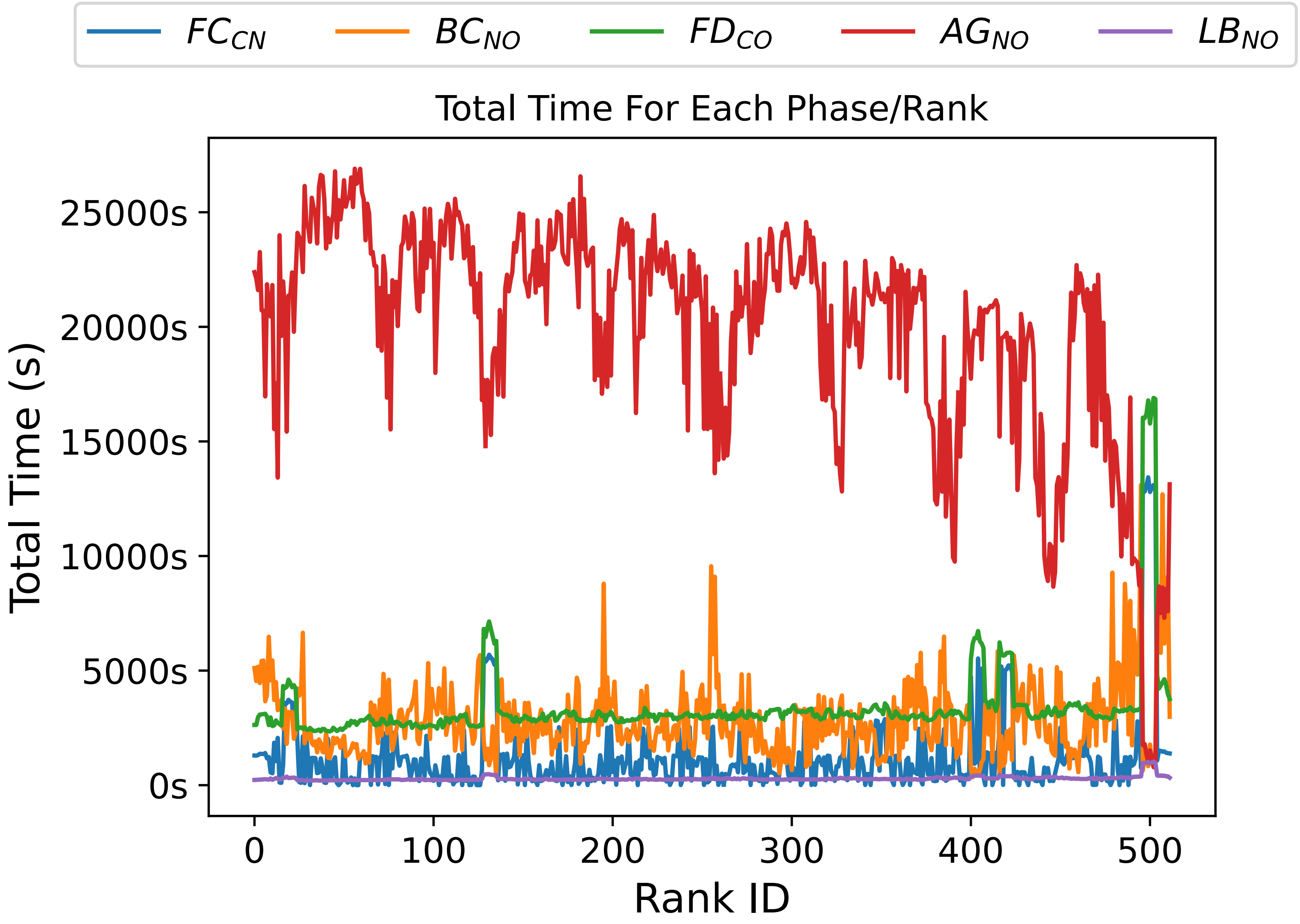

Send/Receive Latency Distribution (BoundaryComm, ts:500)
Send
Receive
Messages
100K
80K
60K
40K
20K
0K
0
100
200
300
400
500
Time (ms)


Send/Receive Latency Distribution (BoundaryComm, ts:5000)
Send
Receive
Messages
100K
80K
60K
40K
20K
0K
0
100
200
300
400
500
Time (ms)


Send/Receive Latency Distribution (BoundaryComm, ts:25000)
Send
Receive
Messages
100K
80K
60K
40K
20K
0K
0
100
200
300
400
500
Time (ms)

FC: 2500 s -> 1300 s
BC: 4000 s -> 5000 s
FD: 2500 s -> 2500 s
AG: 4000-800s -> 15000->25000s
LB: same

Total: 17000s -> 36000s



Next Steps

- Fairly good idea of the execution model and challenges
  - Significant load imbalance exists
  - Mesh block count a poor proxy for load balancing
  - TODO: impact of timing-based load-balancing?
- Can do: Microbenchmarks to quantify impact of different approaches
  - Lots of MPI_Sends + MPI_Recvs – with similar distributions.
- Possible steps forward:
  - Network-based monitoring of all these statistics in realtime
  - Network-based load-balancing services
  - Optimize asynchronous messages as a collective operation