Topology-Aware Collectives

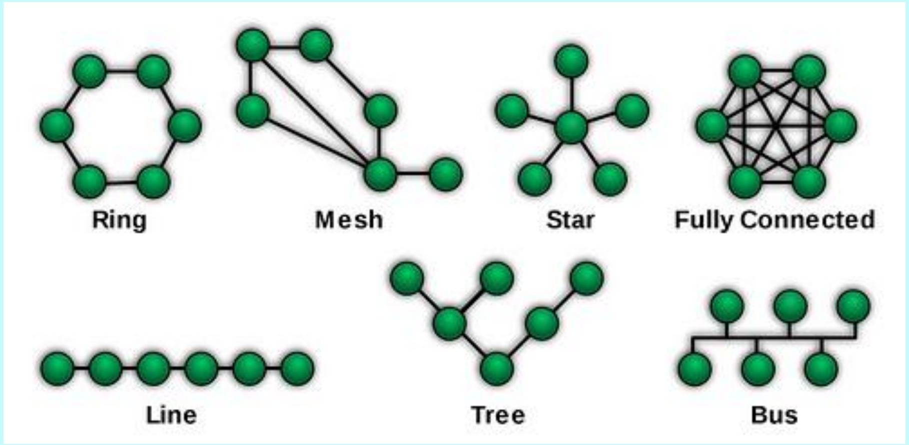
Rudranil Bhowmik(18111055)
Sanket Bodele(18111059)

MPI & Communication

- In MPI, The processes communicate with each other using MPI communication primitives.
- Types of communication
  - Point to point
  - collective
- Collective operations are more common in the real world and more communication intensive

Topology

- It is the layout of connections of various nodes
- Different Topology implementations affects the communication time between processes.
- Topology is hardwired and cannot be easily changed



Topology-Aware Collectives

- What if we can map more communicating processes to the nearest nodes which have low communication overhead.
- We can recurse on this step and find an optimal mapping.
- Such mapping based on the underlying toplogy is called topology-aware mapping.

Topology-Aware Collectives

Topology Awareness can be applied for collectives in two different ways.

A. By changing the communication pattern of the underlying algorithms
B. By changing the mapping of processes.

Previous Works

Seyed H. Mirsadeghi [1] in his paper proposed to use of distances among all cores using the hwloc library.Then he tries to map the processes according to the closest cores present at any given time.

They had a 30-35% reduction in execution time with our proposed rank reordering algorithms.

[1]http://post.queensu.ca/~pprl/papers/IPDRM-2016.pdf

Our work

In our work we

1. Try mapping out the distances layout using ping pong method.
2. Then use an algorithm to recursively plot closest processor wrt another.
3. Create a new communicator with these new ranks and run ring algorithm for MPI_AllGather.
4. Compare performance with non-topology aware ring algorithm.

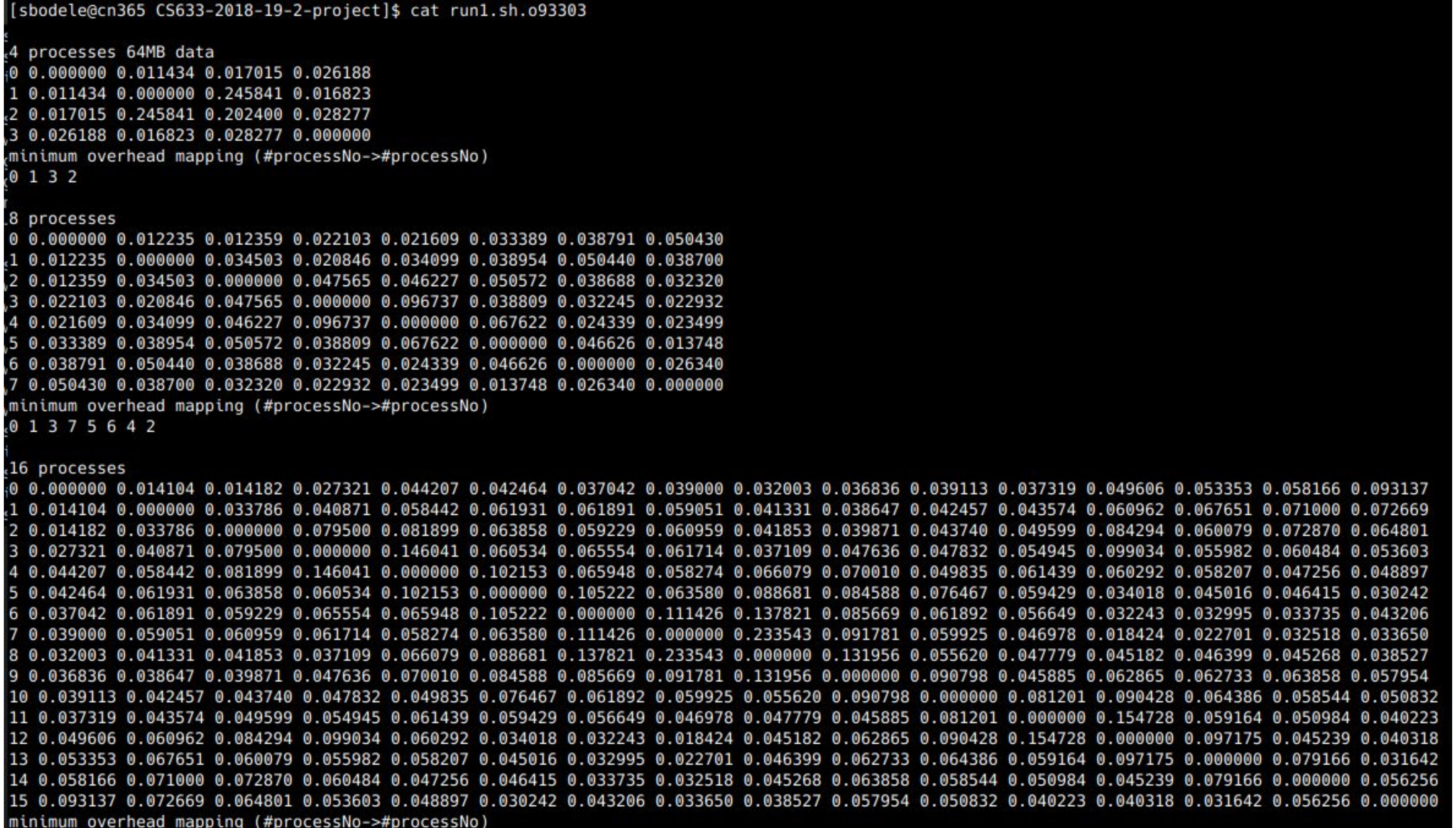
Algorithm for Mapping Heuristics

Input : Number of processes p, and mapping of the physical topology

Output: Mapping array M representing the new rank for each process

STEPS

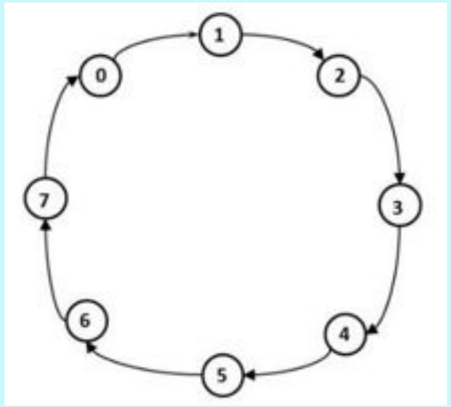
```
1. Fix rank 0 on its current core and start by using it as the reference core
2.  while there exists more processes to map
3. Find the core closest to the reference core that has not been mapped
4. Map the process on the target core
5. Update the newly formed core as reference core and continue
```



```
[sbodele@cn365 CS633-2018-19-2-project]$ cat run1.sh.o93303

4 processes 64MB data
0 0.000000 0.011434 0.017015 0.026188
1 0.011434 0.000000 0.245841 0.016823
2 0.017015 0.245841 0.202400 0.028277
3 0.026188 0.016823 0.028277 0.000000
minimum overhead mapping (#processNo->#processNo)
0 1 3 2

8 processes
0 0.000000 0.012235 0.012359 0.022103 0.021609 0.033389 0.038791 0.050430
1 0.012235 0.000000 0.034503 0.020846 0.034099 0.038954 0.050440 0.038700
2 0.012359 0.034503 0.000000 0.047565 0.046227 0.050572 0.038688 0.032320
3 0.022103 0.020846 0.047565 0.000000 0.096737 0.038809 0.032245 0.022932
4 0.021609 0.034099 0.046227 0.096737 0.000000 0.067622 0.024339 0.023499
5 0.033389 0.038954 0.050572 0.038809 0.067622 0.000000 0.046626 0.013748
6 0.038791 0.050440 0.038688 0.032245 0.024339 0.046626 0.000000 0.026340
7 0.050430 0.038700 0.032320 0.022932 0.023499 0.013748 0.026340 0.000000
minimum overhead mapping (#processNo->#processNo)
0 1 3 7 5 6 4 2

16 processes
0 0.000000 0.014104 0.014182 0.027321 0.044207 0.042464 0.037042 0.039000 0.032003 0.036836 0.039113 0.037319 0.049606 0.053353 0.058166 0.093137
1 0.014104 0.000000 0.033786 0.040871 0.058442 0.061931 0.061891 0.059051 0.041331 0.038647 0.042457 0.043574 0.060962 0.067651 0.071000 0.072669
2 0.014182 0.033786 0.000000 0.079500 0.081899 0.063858 0.059229 0.060959 0.041853 0.039871 0.043740 0.049599 0.084294 0.060079 0.072870 0.064801
3 0.027321 0.040871 0.079500 0.000000 0.146041 0.060534 0.065554 0.061714 0.037109 0.047636 0.047832 0.054945 0.099034 0.055982 0.060484 0.053603
4 0.044207 0.058442 0.081899 0.146041 0.000000 0.102153 0.065948 0.058274 0.066079 0.070010 0.049835 0.061439 0.060292 0.058207 0.047256 0.048897
5 0.042464 0.061931 0.063858 0.060534 0.102153 0.000000 0.105222 0.063580 0.088681 0.084588 0.076467 0.059429 0.034018 0.045016 0.046415 0.030242
6 0.037042 0.061891 0.059229 0.065554 0.065948 0.105222 0.000000 0.111426 0.137821 0.085669 0.061892 0.056649 0.032243 0.032995 0.033735 0.043206
7 0.039000 0.059051 0.060959 0.061714 0.058274 0.063580 0.111426 0.000000 0.233543 0.091781 0.059925 0.046978 0.018424 0.022701 0.032518 0.033650
8 0.032003 0.041331 0.041853 0.037109 0.066079 0.088681 0.137821 0.233543 0.000000 0.131956 0.055620 0.047779 0.045182 0.046399 0.045268 0.038527
9 0.036836 0.038647 0.039871 0.047636 0.070010 0.084588 0.085669 0.091781 0.131956 0.000000 0.090798 0.045885 0.062865 0.062733 0.063858 0.057954
10 0.039113 0.042457 0.043740 0.047832 0.049835 0.076467 0.061892 0.059925 0.055620 0.090798 0.000000 0.081201 0.090428 0.064386 0.058544 0.050832
11 0.037319 0.043574 0.049599 0.054945 0.061439 0.059429 0.056649 0.046978 0.047779 0.045885 0.081201 0.000000 0.154728 0.059164 0.050984 0.040223
12 0.049606 0.060962 0.084294 0.099034 0.060292 0.034018 0.032243 0.018424 0.045182 0.062865 0.090428 0.154728 0.000000 0.097175 0.045239 0.040318
13 0.053353 0.067651 0.060079 0.055982 0.058207 0.045016 0.032995 0.022701 0.046399 0.062733 0.064386 0.059164 0.097175 0.000000 0.079166 0.031642
14 0.058166 0.071000 0.072870 0.060484 0.047256 0.046415 0.033735 0.032518 0.045268 0.063858 0.058544 0.050984 0.045239 0.079166 0.000000 0.056256
15 0.093137 0.072669 0.064801 0.053603 0.048897 0.030242 0.043206 0.033650 0.038527 0.057954 0.050832 0.040223 0.040318 0.031642 0.056256 0.000000
minimum overhead mapping (#processNo->#processNo)
```

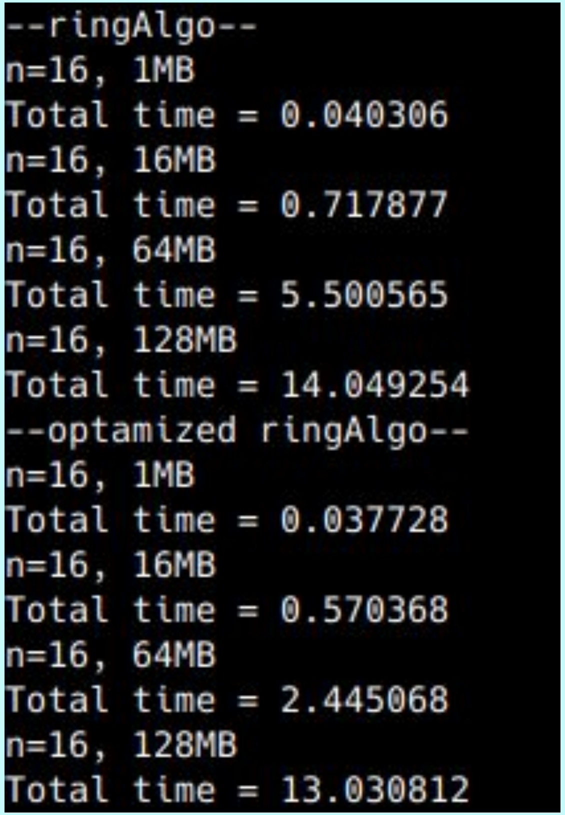
Application on gatherall(ring communication pattern)

Here a single process communicates with a single process for the entire lifetime so mapping this according to the heuristic should work.

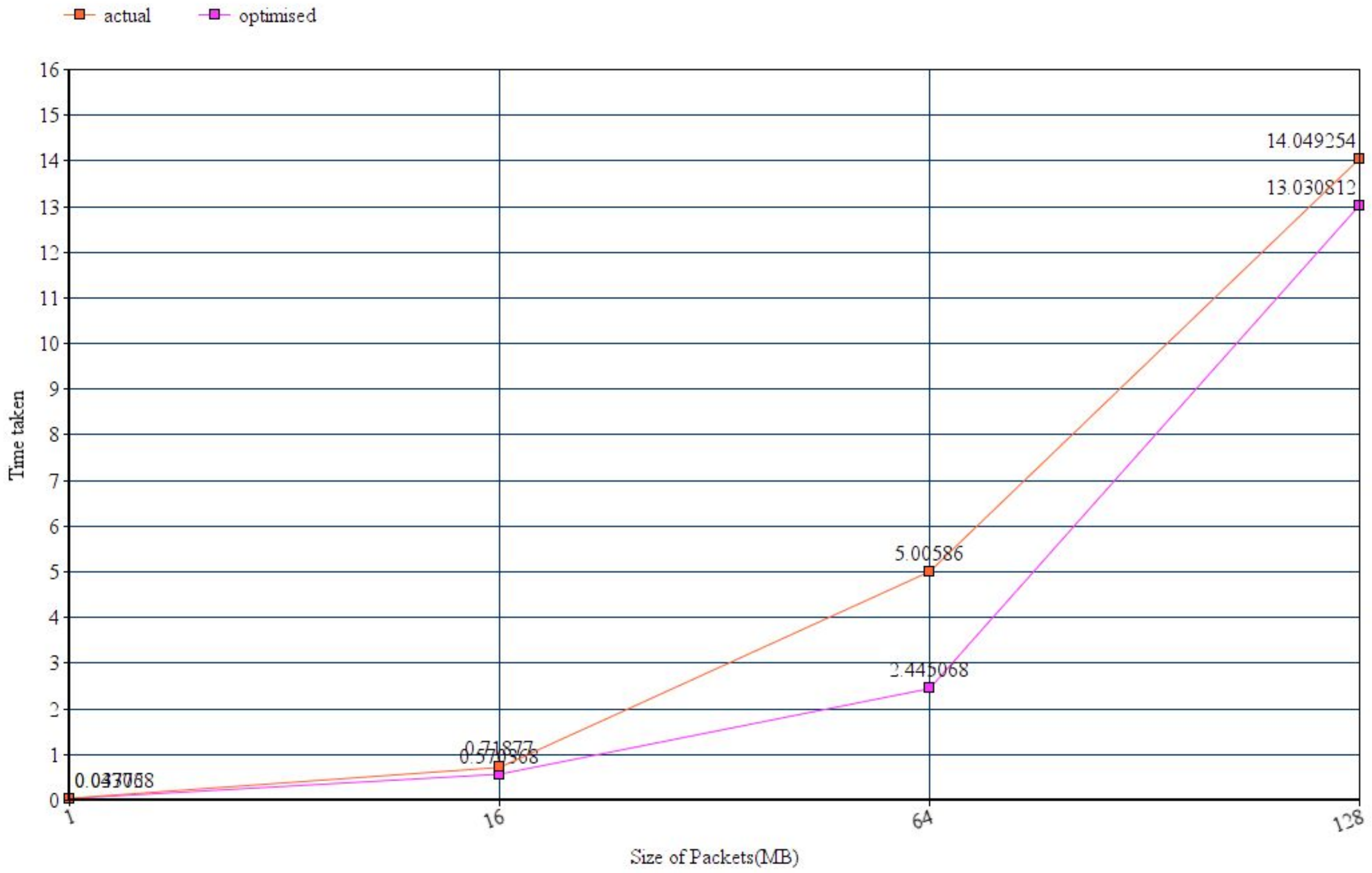


Results

- Since we don’t apply it on a standard algorithm we get a huge variation of result with 9%-35% improvement in different stages of the application.



```
--ringAlgo--
n=16, 1MB
Total time = 0.040306
n=16, 16MB
Total time = 0.717877
n=16, 64MB
Total time = 5.500565
n=16, 128MB
Total time = 14.049254
--optamized ringAlgo--
n=16, 1MB
Total time = 0.037728
n=16, 16MB
Total time = 0.570368
n=16, 64MB
Total time = 2.445068
n=16, 128MB
Total time = 13.030812
```


actual
optimised
Time taken
Size of Packets(MB)
14.049254
13.030812
5.00586
2.445068
0.71877
0.570568
0.043768
1
16
64
128
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Future Work

- Application of gatherall/broadcast using recursive doubling.
- Considering Data Sizes being transferred in stages we need to op