Introduction to HPC
Part 2: MPI

lsxm74*

I. INTRODUCTION

This report described the implementation of tree reduction in MPI, with four function:

$$MPI\_SUM,\ \ MPI\_PROD,\ \ MPI\_MIN,\ \ MPI\_MAX$$

It run at specific communicators whose size is a power of two, such like: $1(2^0),\ 2(2^1),\ 4(2^2),\ ...$ and compared with stander ring reduce and builtin algorithm 'MPI_Allreduce', it done by benchmarking on the computer notes of Hamilton system. At end paper shows and analyse some data of experiments.

II. ALGORITHM AND IMPLEMENTATION OF TREE REDUCTION

A. Tree reduction

During the stander ring reduction, if we need a item pass all the ring of P processes, we need sent item $(P-1)$ times, for each time it sent we need a bit of time so the time it need will grow linearly as ring contains more processes. However with tree reduction it will sufficiently reduce the times of sending item by combine the partial reductions pairwise in a tree as below Fig 1 shows with 8 processes of tree reduction.

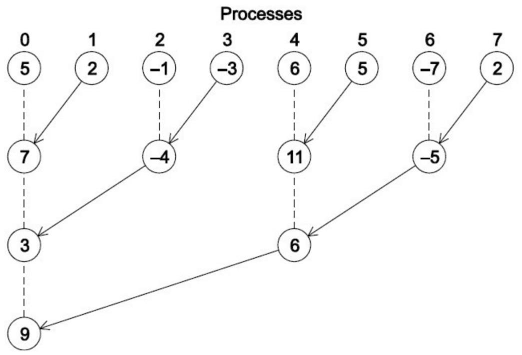


Fig. 1: 8 processes of tree reduction.

as you can see, it only sent item 7 times for 8 processes to get results, which clearly much less than ring reduction. so if we have huge numbers of processes we should use tree reductions rather than ring reduction.

B. Implementation

To help for implementation the processes of sending item can be rearrange like Fig 2, as mentioned before the size of communicators always will be a power of 2, so we can easily paired two processes by split into two groups with first half and second half of total processes and only second half will sent item.

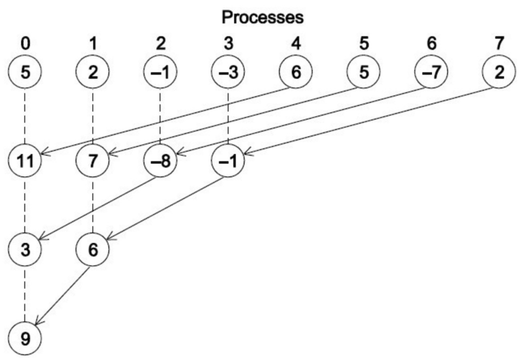


Fig. 2: The way of coding

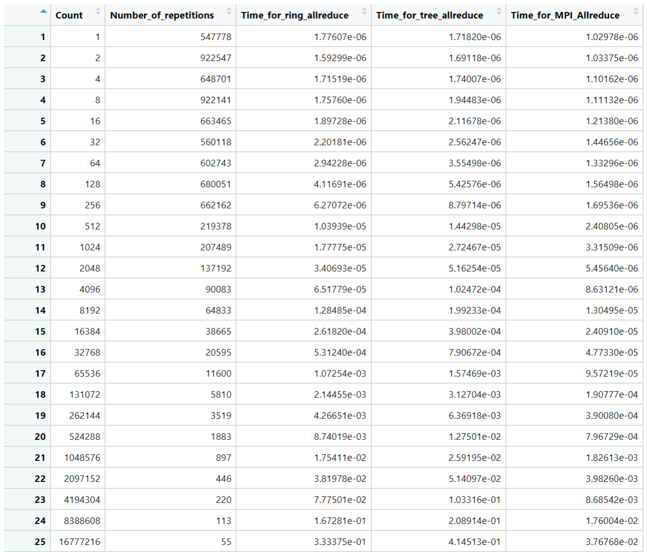
To get result back to all processes we just need to reverse what we do before, send result back up the tree with same logic. The part of code to the algorithm as follows:

```
        while (remain != 1){
          half = remain/2;
          if (rank<half){
            ierr = MPI_Recv(&temps2, count, MPI_INT, rank+half, 0, comm, MPI_STATUS_IGNORE);CHKERR(ierr);
            for (c = 0; c < count; c++){
              //do the calculations as different input operator}
          }else if(rank>=half && rank < remain){
            ierr = MPI_Send(&temps1, count, MPI_INT, rank-half, 0, comm);CHKERR(ierr);
            remain = half;
            break;
            }
          remain = half;
        }
        while (remain!=size){
            //reverse what we do
        }
```

It use remain to determine how many processes are left, start with total number of processes and at end of each round reduce to half.

III. Experiment of different reductions

By benchmarking, we run our reductions at Hamilton with range of processes from 2 to 128 through range of message counts from 1 to 2^{24}. As the results we get many table of times for each reduction to complete calculations such as Fig 3 for 4 processes:



	Count	Number_of_repetitions	Time_for_ring_allreduce	Time_for_tree_allreduce	Time_for_MPI_Allreduce
1	1	547778	1.77607e-06	1.71820e-06	1.02978e-06
2	2	922547	1.59299e-06	1.69118e-06	1.03375e-06
3	4	648701	1.71519e-06	1.74007e-06	1.10162e-06
4	8	922141	1.75760e-06	1.94483e-06	1.11132e-06
5	16	663465	1.89728e-06	2.11678e-06	1.21380e-06
6	32	560118	2.20181e-06	2.56247e-06	1.44656e-06
7	64	602743	2.94228e-06	3.55498e-06	1.33296e-06
8	128	680051	4.11691e-06	5.42576e-06	1.56498e-06
9	256	662162	6.27072e-06	8.79714e-06	1.69536e-06
10	512	219378	1.03939e-05	1.44298e-05	2.40805e-06
11	1024	207489	1.77775e-05	2.72467e-05	3.31509e-06
12	2048	137192	3.40693e-05	5.16254e-05	5.45640e-06
13	4096	90083	6.51779e-05	1.02472e-04	8.63121e-06
14	8192	64833	1.28485e-04	1.99233e-04	1.30495e-05
15	16384	38665	2.61820e-04	3.98002e-04	2.40910e-05
16	32768	20595	5.31240e-04	7.90672e-04	4.77330e-05
17	65536	11600	1.07254e-03	1.57469e-03	9.57219e-05
18	131072	5810	2.14455e-03	3.12704e-03	1.90777e-04
19	262144	3519	4.26651e-03	6.36918e-03	3.90080e-04
20	524288	1883	8.74019e-03	1.27501e-02	7.96729e-04
21	1048576	897	1.75411e-02	2.59195e-02	1.82613e-03
22	2097152	446	3.81978e-02	5.14097e-02	3.98260e-03
23	4194304	220	7.77501e-02	1.03316e-01	8.68542e-03
24	8388608	113	1.67281e-01	2.08914e-01	1.76004e-02
25	16777216	55	3.33375e-01	4.14513e-01	3.76768e-02

Fig. 3: 4 processes of reduction.

with all that data we can plot some graphs to compare each reduction.

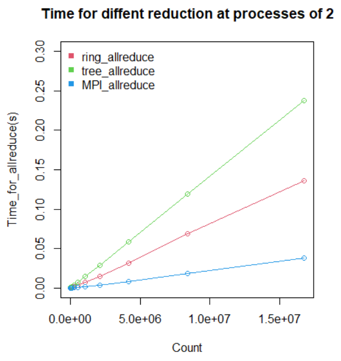


Fig. 4: Time at processes of 2

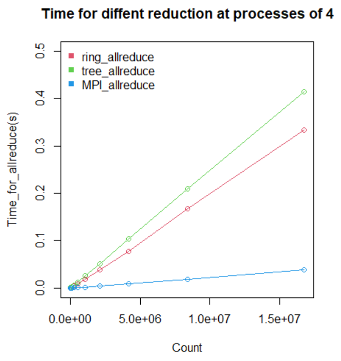


Fig. 5: Time at processes of 4

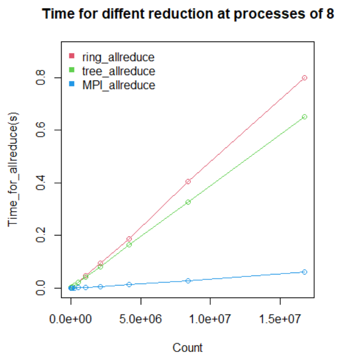


Fig. 6: Time at processes of 8

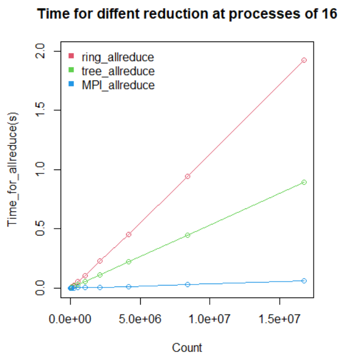


Fig. 7: Time at processes of 16

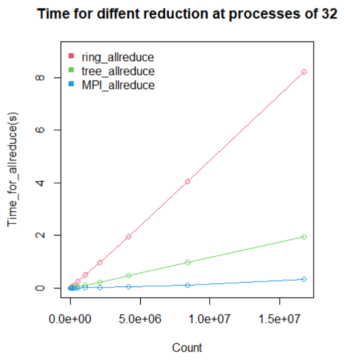


Fig. 8: Time at processes of 32

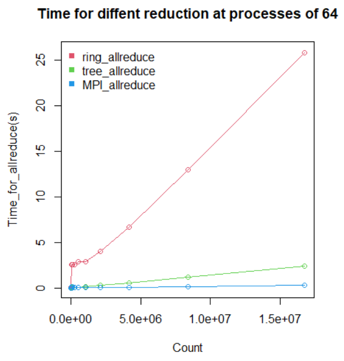


Fig. 9: Time at processes of 64

I choose to plot their Time vs. size of Count for different processes as it will clearly see which got less time and how they changed with growing Count. After plotting all graph there have some interesting observation that tree reduction is not always better than ring reduction, nevertheless, as we believe MPI all-reduction is always the fastest reduction as Fig 4 to 9 shows.

It can see that during processes of 2 and 4 the ring reduction is still better than our tree reduction, so only choose tree reduction when processes is great or equal than 8 if there is no MPI reduction. As number of processes growth bigger, the time differences between is become significantly great, it is matched with what we believed at beginning of paper.For the shape of graph, all three reduction is following linear model, some graph like Fig 9 shows some unusual trend but it should be accident during running at Hamilton. Overall it shows same scaling behaviour as we constructed in lectures.

IV. Other Findings

I also benchmarking the 128 processes but it not shows as before because it show different pattern with other graph as Fig 10 in next page, the MPI reduction suddenly grows at around 4000000 count and keep the level for rest of count, it did not agree with other graph I got, it may be some unknown problems with Hamilton when I run it or my code has some memory leaking that I do not know for now which is unlikely as other two reduction is behave normally, the memory leaking should effect all three reduction therefor it more like accident due to Hamilton. However it interesting that before MPI growing the tree reduction is using almost same time as MPI used. Cause the experiment stop at 128 processes, it may have some point in future that tree reduction will use less time than MPI, if it is true, it may because the degree of complexity for MPI as it build for all the situation and tree reduction is only implement for this four situation.

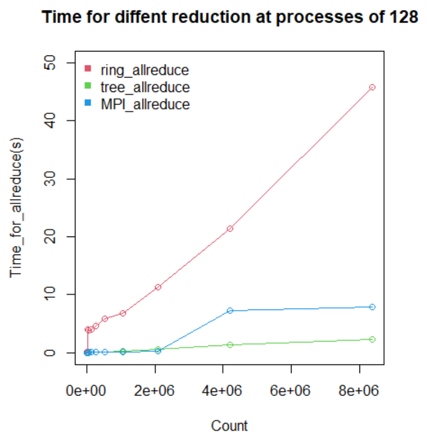


Fig. 10: 128 processes of reduction.

V. THINKING AND IMPROVEMENT

At first the code could be more efficiency and maybe faster. Secondly this tree reduction is only for the specific communicators whose size is a power of two, it could change to all type of communicators, the rough idea is still split processes into half, when number is odd, make first half one more than second. As code below, every rank which small than (half-1) will receive item from processes who bigger than half, and (half) term will do nothing just go to next level of tree, then at next level it return to even processes we can go as our implementation.

```
    if (rank<(half-1)){
        //do the calculations as different input operator}
    }else if(rank>half && rank < remain){
        //do the calculations
    }else {
      break
    }
  while (remain!=size){
      //reverse what we do
  }
```