Profiling - II

Lecture 24

April 15, 2024

Gprof

"gprof: A Call Graph Execution Profiler", by S. Graham, P. Kessler, M. McKusick; Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices, Vol. 17, No 6, pp. 120-126, June 1982.

- Compile with –g –pg flags
- gprof ./exe gmon.out > gprof.out

Sections in the mpiP report

- 31:@--- MPI Time (seconds) --
- 52:@--- Callsites: 43 --
- 99:@--- Aggregate Time (top twenty, descending, milliseconds) --------
- 123:@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
- 146:@--- Callsite Time statistics (all, milliseconds): 688 -------------------
- 923:@--- Callsite Message Sent statistics (all, sent bytes) ------------------
- 1268:@--- End of Report ---

Aggregate Time – Strong Scaling Comparison (mg)

```
@--- Aggregate Time (top twenty, descending, milliseconds)
Call          Site     Time   App%   MPI%    COV
Bcast           12     22.5   0.49  19.71   0.66
Send             9     21.1   0.46  18.47   0.10
Send            20     14.1   0.31  12.39   0.02
Send             1     13.4   0.29  11.74   0.32
```

Processes = 4

```
@--- Aggregate Time (top twenty, descending, milliseconds)
Call          Site     Time   App%   MPI%    COV
Barrier          7      149   2.24  21.84   0.01
Send             9      140   2.10  20.48   0.81
Send            21      123   1.84  17.94   0.87
Wait            26     58.8   0.88   8.60   0.09
```

Processes = 16

Aggregate Time – Strong Scaling Comparison (ft)

```
@--- Aggregate Time (top twenty, descending, milliseconds)
Call          Site     Time    App%    MPI%     COV
Alltoall         9      443    5.69   84.89    0.03
Bcast            7     43.3    0.56    8.29    0.00
Reduce           4     32.4    0.42    6.21    0.02
Barrier          5     1.57    0.02    0.30    1.16
```

Processes = 4

```
@--- Aggregate Time (top twenty, descending, milliseconds)
Call          Site     Time    App%    MPI%     COV
Alltoall         9 1.73e+03   16.22   91.43    0.05
Reduce           4     76.4    0.72    4.03    0.84
Comm_split      10     44.3    0.41    2.34    0.92
Bcast            7     24.8    0.23    1.31    0.48
```

Processes = 16

Aggregate Time – Data Scaling (cg on 16 processes)

```
@--- Aggregate Time (top twenty, descending, milliseconds)
Call            Site     Time   App%   MPI%    COV
Bcast              3      477  16.58  39.24   0.01
Wait              21      176   6.11  14.46   0.74
Send              10      162   5.64  13.34   0.85
Wait               6     89.6   3.11   7.37   0.13
```

Class = A (small problem)

```
@--- Aggregate Time (top twenty, descending, milliseconds)
Call            Site     Time   App%   MPI%    COV
Wait              21 1.03e+04   3.07  31.48   0.79
Send              15 8.84e+03   2.65  27.13   0.19
Send              12 8.44e+03   2.53  25.91   0.79
Wait              11      728   0.22   2.23   1.49
```

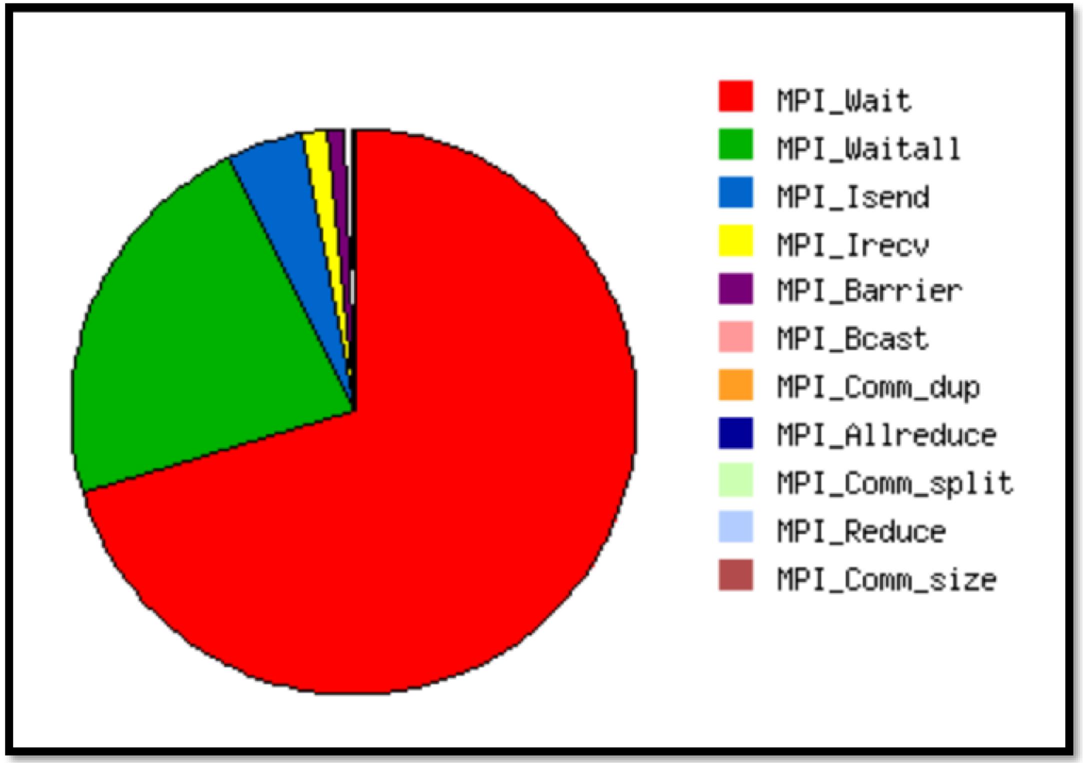
Class = C (large problem)

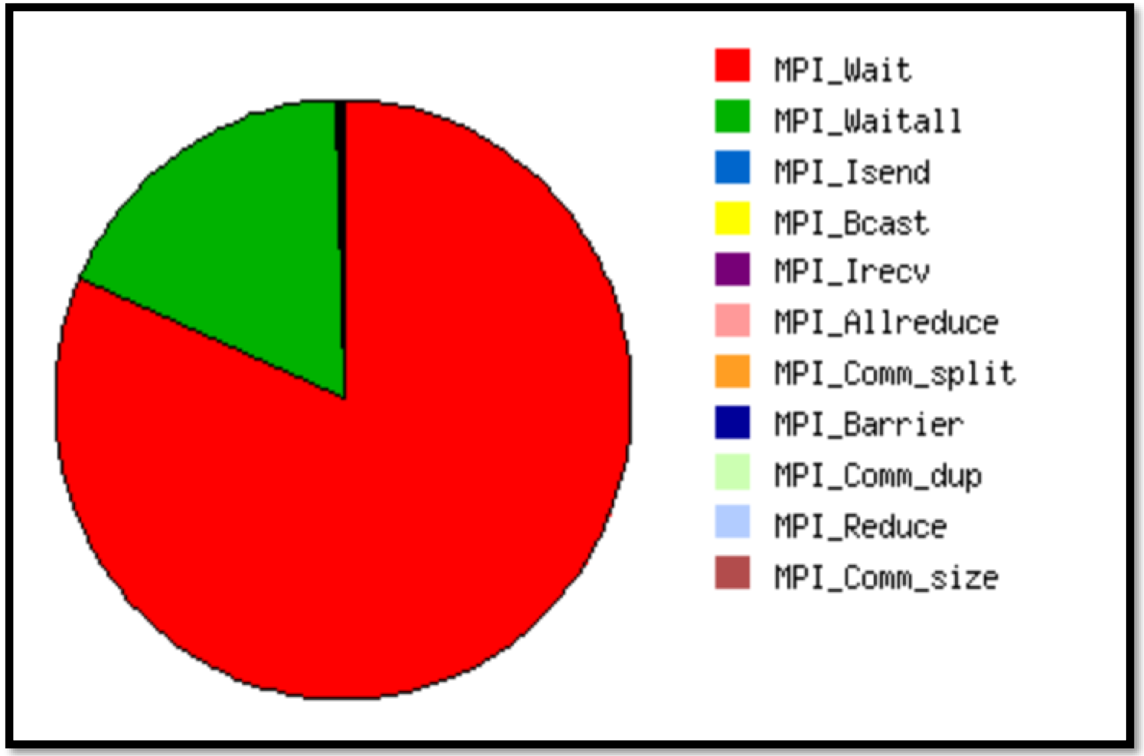
MPI Time vs. App Time (Class = A NPROCS=16)

@--- Aggregate Time (top twenty, descending, milliseconds)

Call	Site	Time	App%	MPI%	COV	
Bcast	3	477	16.58	39.24	0.01	cg
Allreduce	5	246	1.31	77.45	0.60	ep
Alltoall	9	1.73e+03	16.22	91.43	0.05	ft
Recv	3	3.78e+03	5.30	27.16	0.76	lu
Barrier	7	149	2.24	21.84	0.01	mg
Waitall	41	1.89e+03	2.13	20.52	0.17	sp

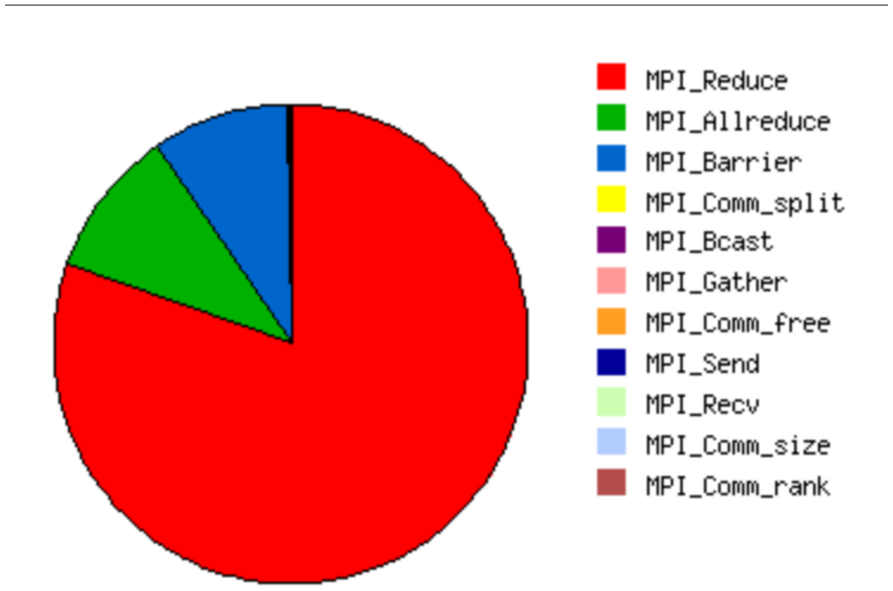
IPM Profiles

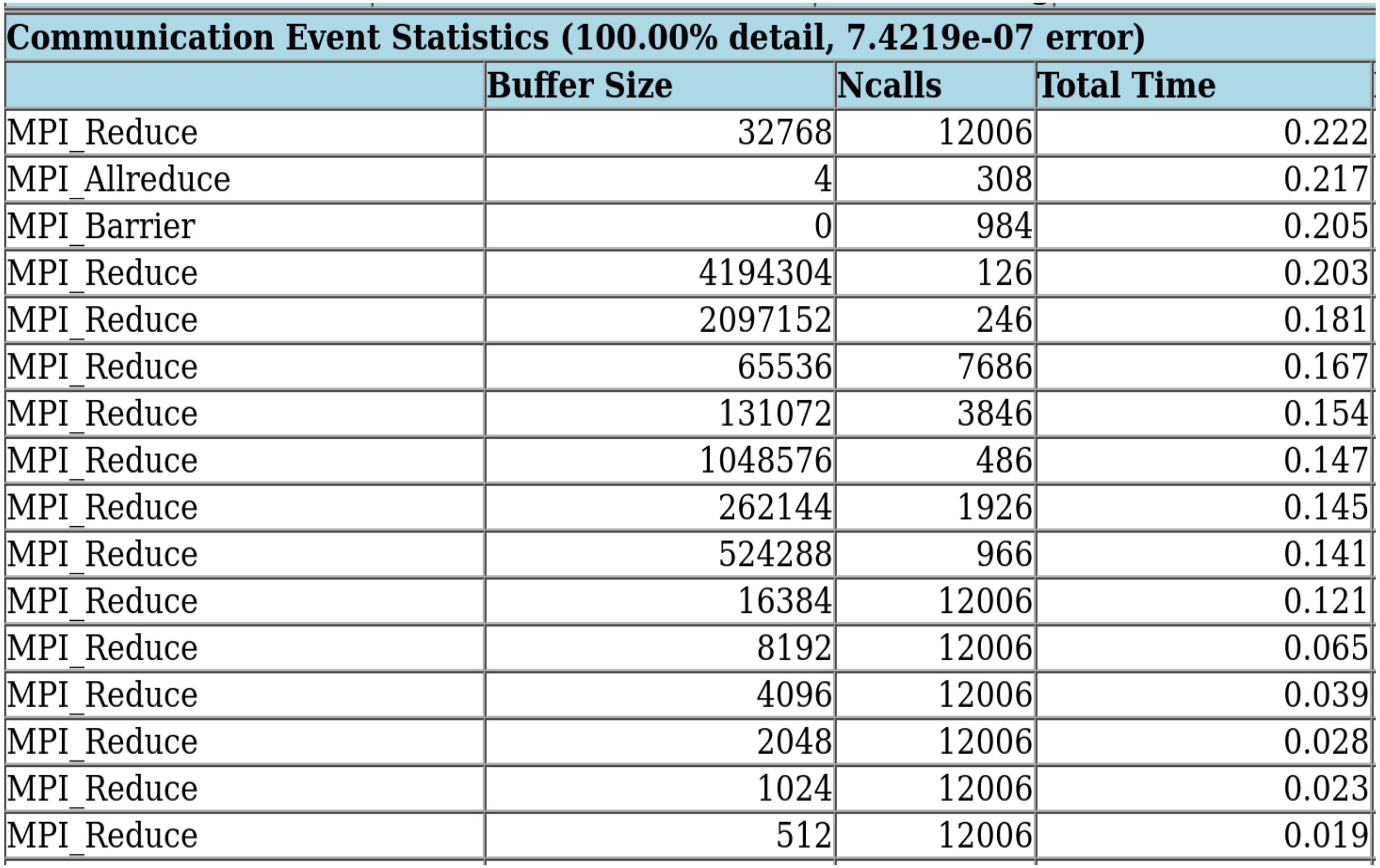




Configuration difference?

IMB Reduce (NPROCS = 4)

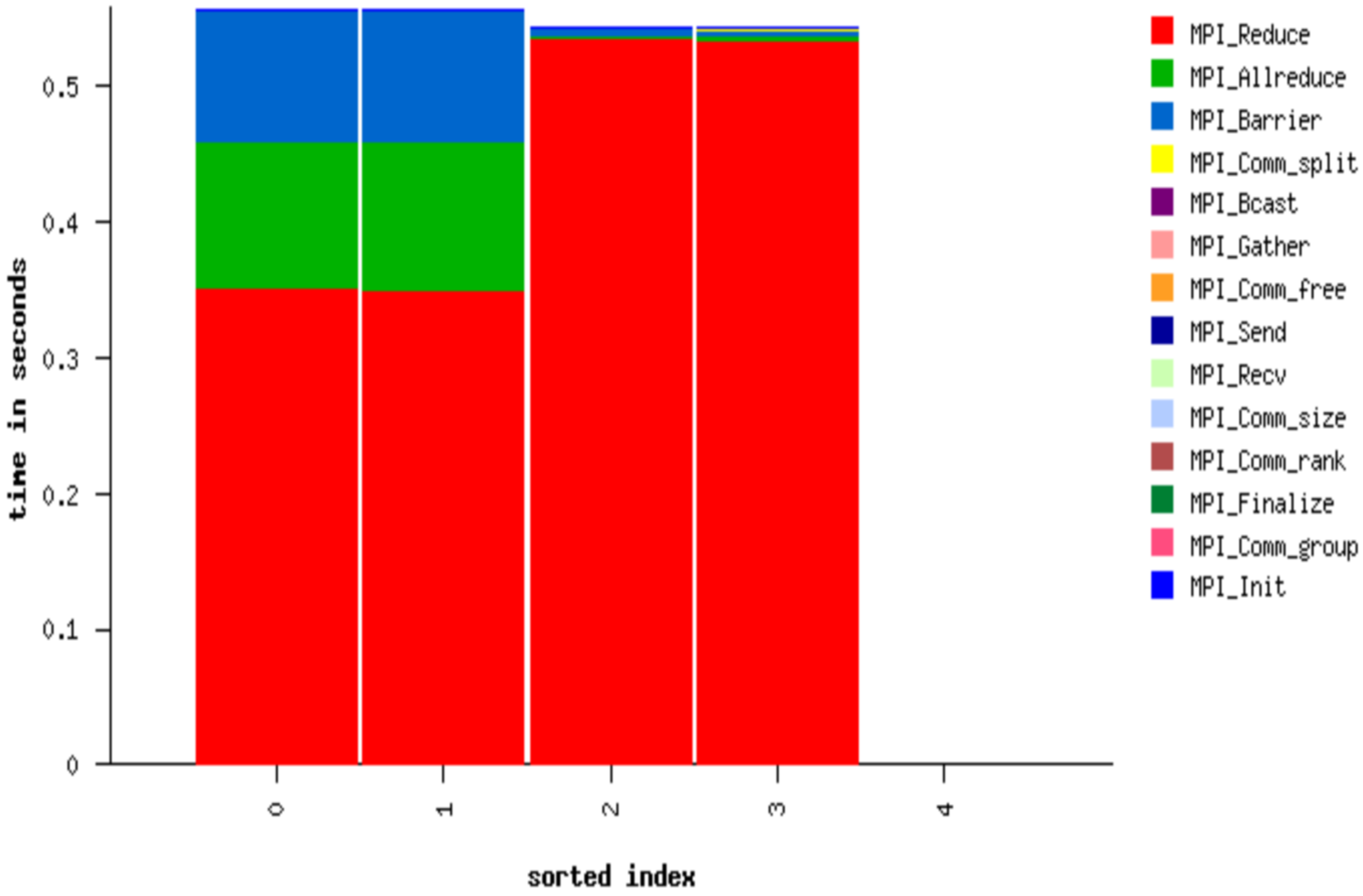




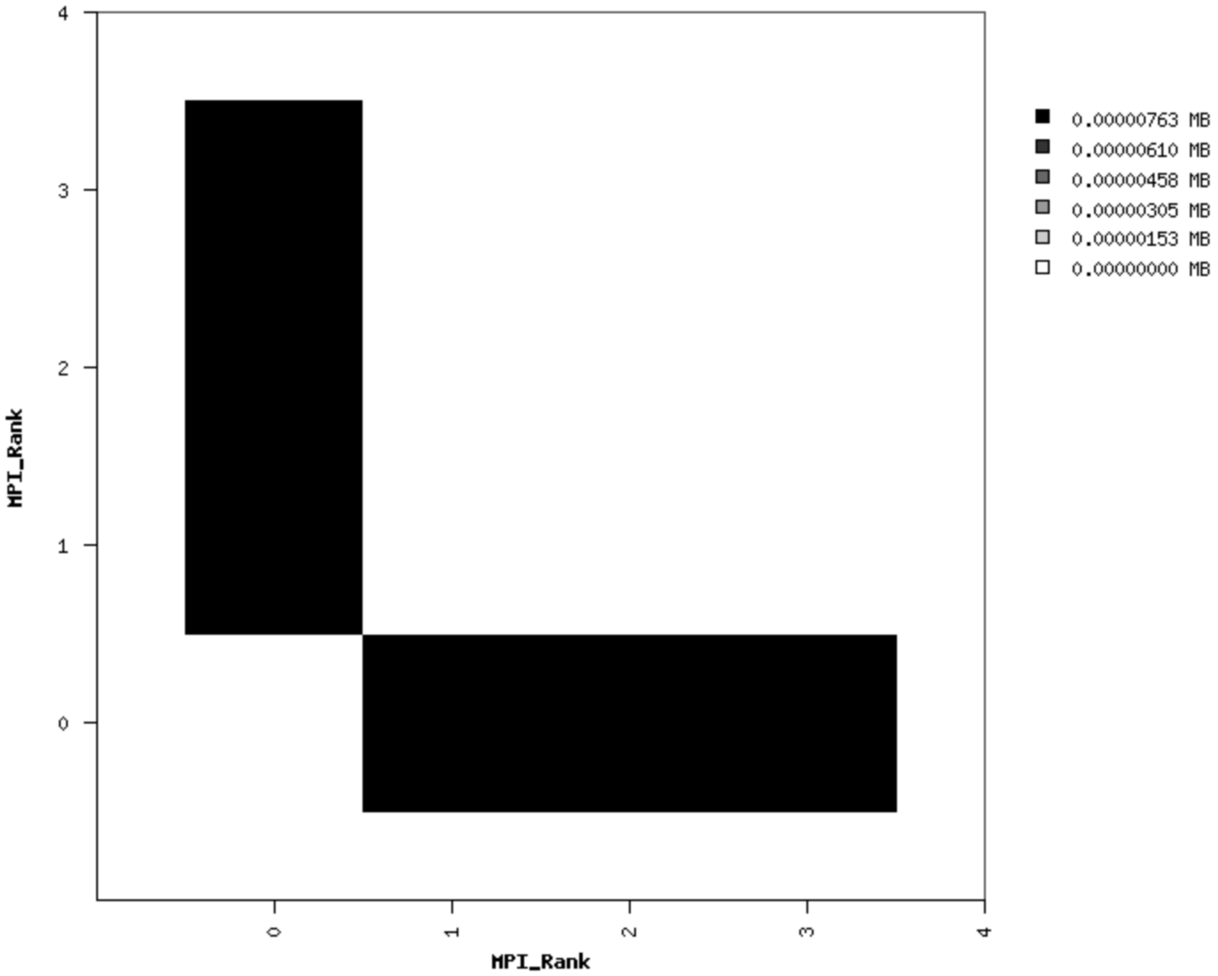
Communication Event Statistics (100.00% detail, 7.4219e-07 error)

	Buffer Size	Ncalls	Total Time
MPI_Reduce	32768	12006	0.222
MPI_Allreduce	4	308	0.217
MPI_Barrier	0	984	0.205
MPI_Reduce	4194304	126	0.203
MPI_Reduce	2097152	246	0.181
MPI_Reduce	65536	7686	0.167
MPI_Reduce	131072	3846	0.154
MPI_Reduce	1048576	486	0.147
MPI_Reduce	262144	1926	0.145
MPI_Reduce	524288	966	0.141
MPI_Reduce	16384	12006	0.121
MPI_Reduce	8192	12006	0.065
MPI_Reduce	4096	12006	0.039
MPI_Reduce	2048	12006	0.028
MPI_Reduce	1024	12006	0.023
MPI_Reduce	512	12006	0.019

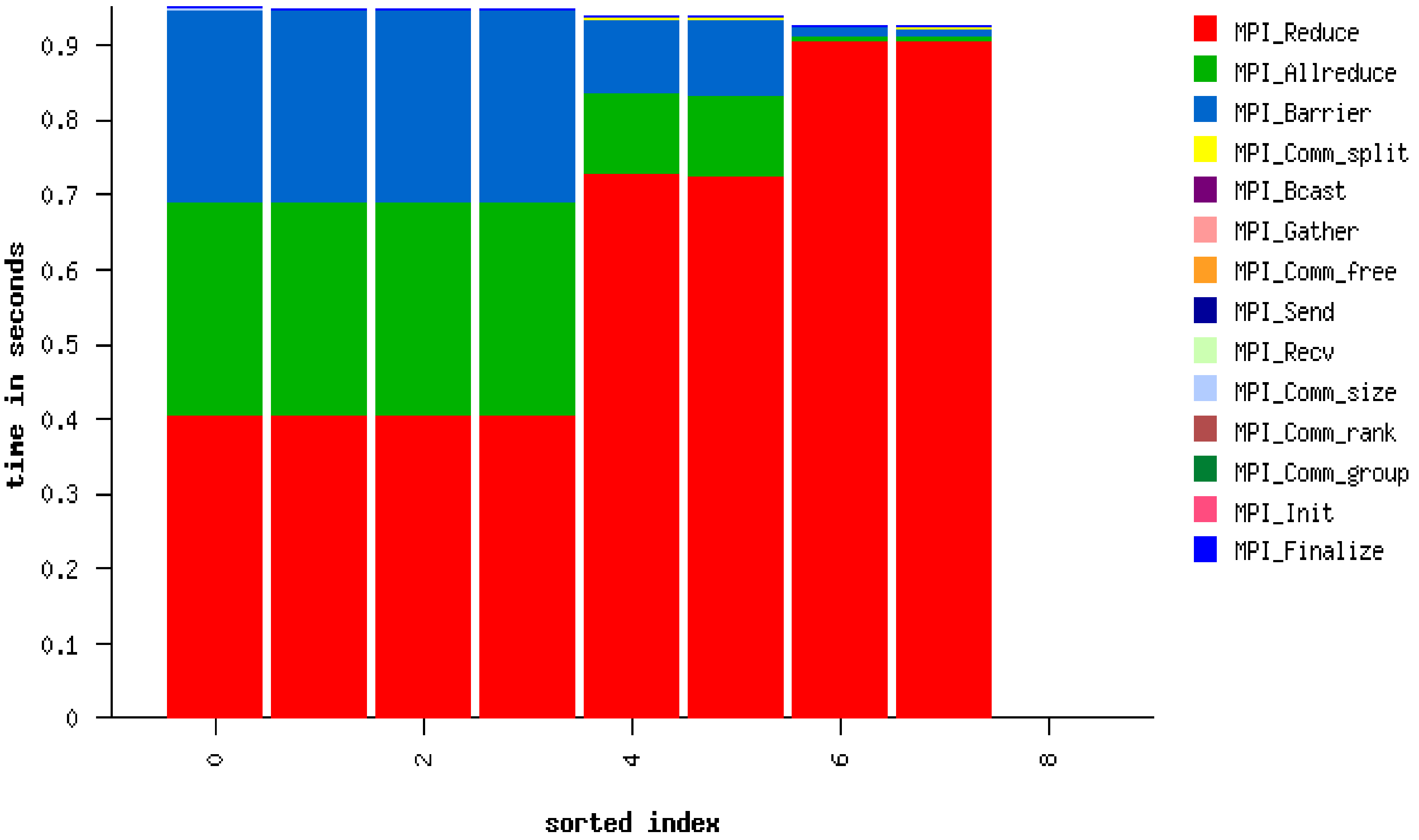
IMB Reduce (NPROCS = 4)



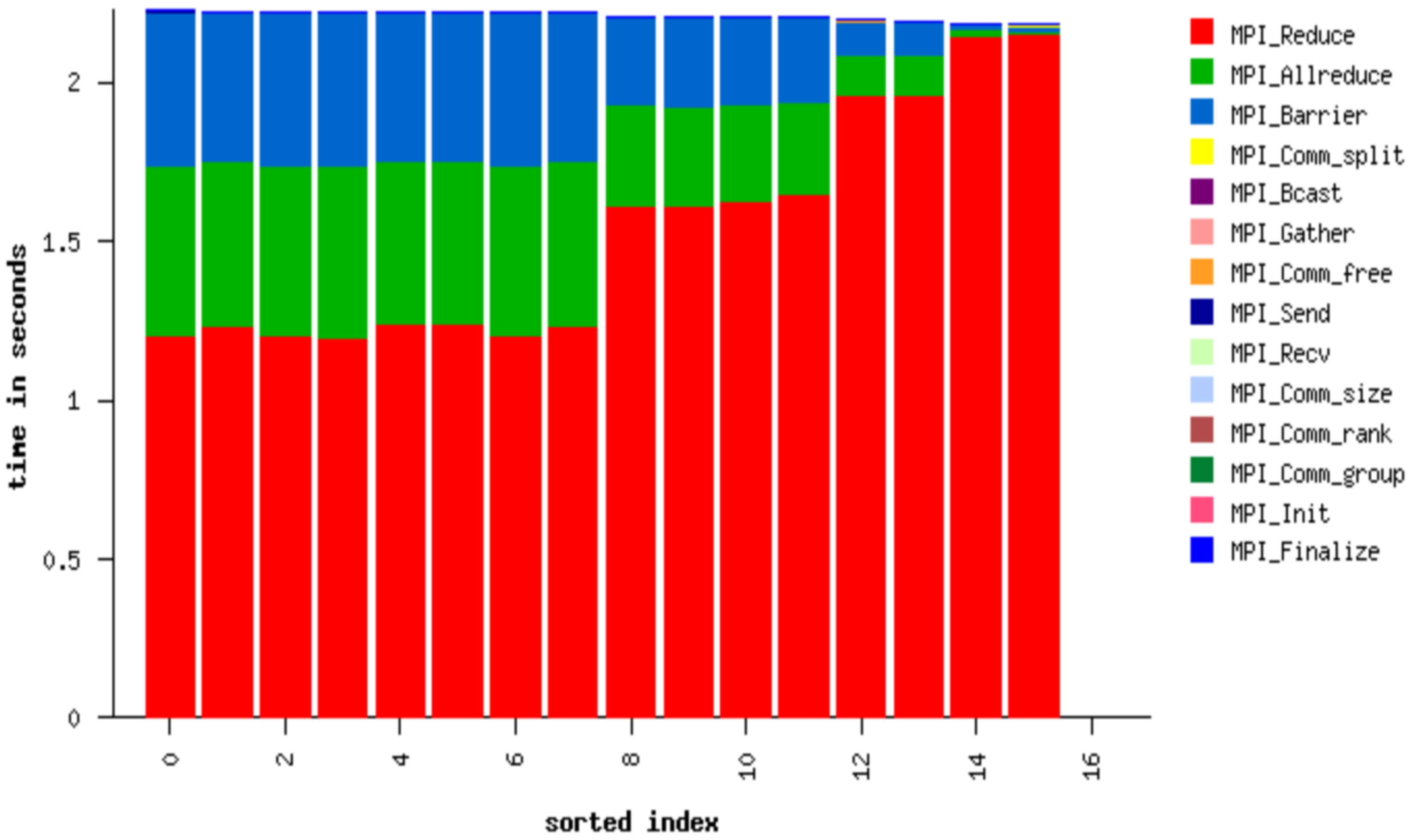
Communication Matrix (IMB Reduce, NPROCS=4)



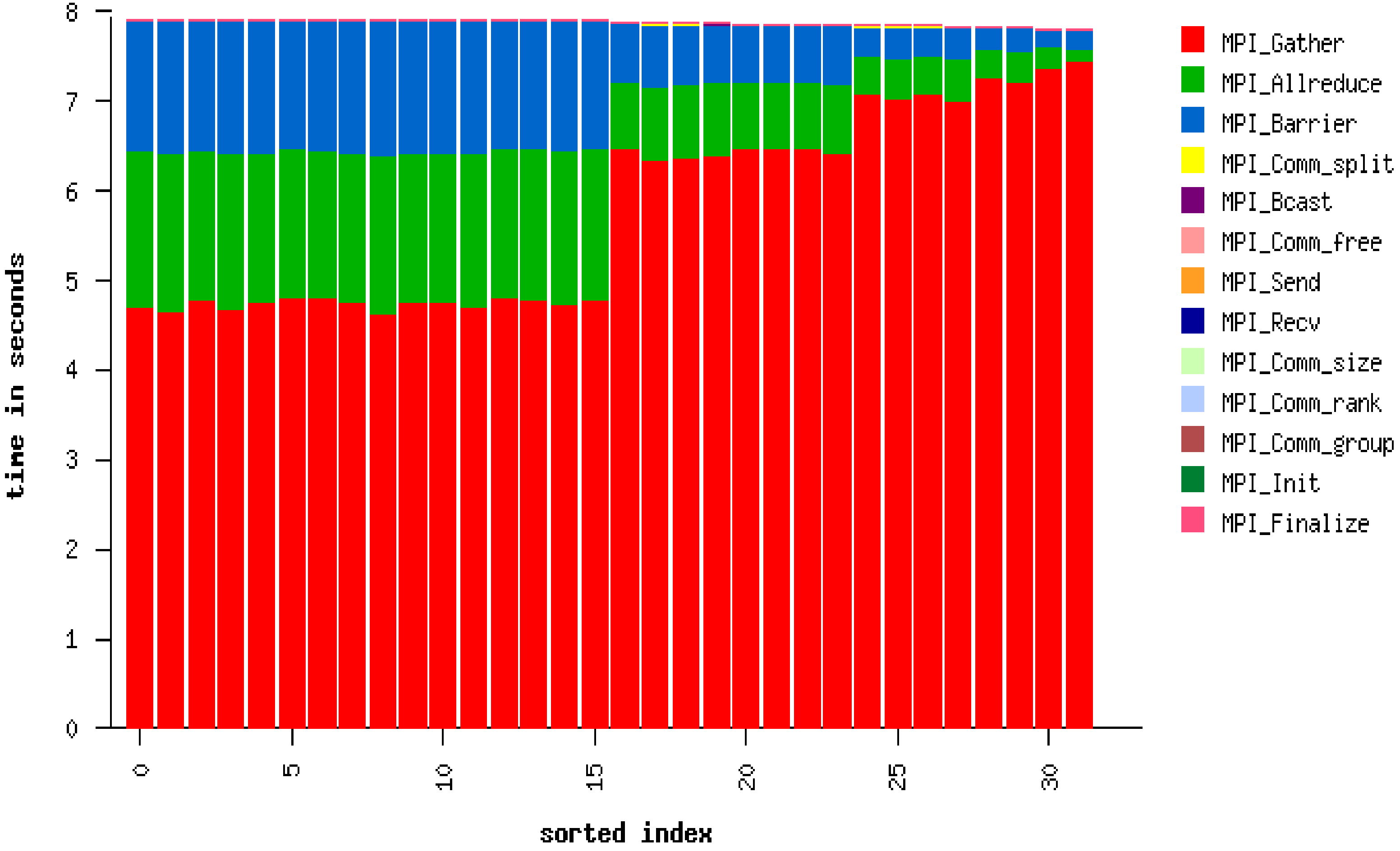
IMB Reduce (NPROCS = 8, 1 host)



IMB Reduce (NPROCS = 16, 2 hosts)



IMB Gather (NPROCS=32, 4 hosts)



Darshan Internals

- Intercepts MPI-IO routines using PMPI interface
- Data recorded on each process at run time and then merged and stored during MPI_Finalize
- MPI_Wtime() collects timing information
- In-memory file record
  - Array of counters for I/O calls
  - Frequency count of common access sizes
- Dynamic linking at runtime
  - LD_PRELOAD – enables overriding
- Static linking at compile-time
  - Inserting wrapper functions
  - --wrap option
- Time Overhead
  - MPI_Wtime() call - 165 ns
  - Function wrapping - 14 ns *
- Memory overhead
  - File record 2 MB limit per process
  - Aggregate statistics beyond limit

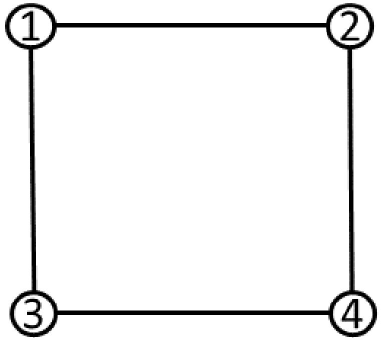
* "24/7 Characterization of Petascale I/O Workloads"

Darshan I/O Profiler

- cd io
- export DARSHAN_LOGPATH=darshan-logs
- mpiicc –o indepIO indepIO.c
- export LD_PRELOAD=../lib (path to libdarshan.so)
- qsub subindepIO.c
- mkdir $DARSHAN_LOGPATH/2024/4/15
- ls –t $DARSHAN_LOGPATH/2024/4/15 [Look for .darshan]
- ./darshan-parser <logfile> > parsed
- grep POSIX_F_FASTEST_RANK_TIME parsed
- grep MPIIO_F_FASTEST_RANK_TIME parsed
- grep MPIIO_F_SLOWEST_RANK_TIME parsed

Revision Q1

MPI_Bcast of 10 KB data (root=0) on the 2D mesh.
There are 8 processes placed on the 4 nodes.
Ranks 0 and 1 are placed on node 1, ranks 2 and 3 are placed on node 2 and so on.
Bandwidth of every link is 1 Gbps. Assume hop=0 between processes in a node. Assume XY routing policy (i.e. messages first traverse in x-dimension, followed by y-dimension).
Total time = 4 ms
Analyze and discuss the effective bandwidth, maximum #hops, and link contention with Bcast.



Revision Q2

Compare and contrast recursive doubling algorithm for MPI_Reduce on 8 processes for the following node allocations:

(a) Ranks 0 – 3 are on csews1, ranks 4 – 7 are on csews2

(b) Even ranks are on csews1, odd ranks are on csews2

Revision Q3: 3D domain decomposition

```
  //initialize
  for (int i=0; i<N; i++)
   for (int j=0; j<N; j++)
    for (int k=0; k<N; k++)
     data[i][j][k] = (rank+1) * (i+j+k);

  int xStart=________________________________,
yStart=________________________________,
zStart=________________________________;

  int xEnd=__________________________________,
yEnd=__________________________________,
zEnd=__________________________________;
```