Tuning Your MPI Application Without Writing Code

SNUG TechTalk, 8 Feb 2012

Outline

- MPI Libraries
    - Eager vs Rendezvous, Collective Algorithms
- mpitune
- otpo
- Locality and Pinning

Inside an MPI Library

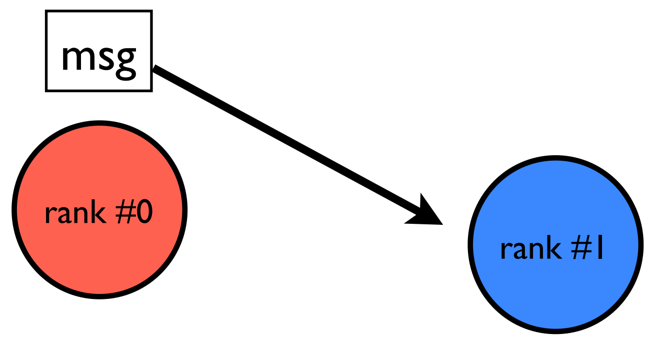
You send a message, a miracle occurs, and the message is received on the other side.

- Jeff Squyres, Cisco/OpenMPI, OpenMPI Mailing list, Jan 2012

Inside an MPI Library

- The MPI standard intentionally says nothing about *how* messages are sent between MPI tasks
- The implementation must decide
- Typically many behaviours, determined by threshold parameters.
- Parameters chosen for overall good performance - but your application may benefit from changing these.

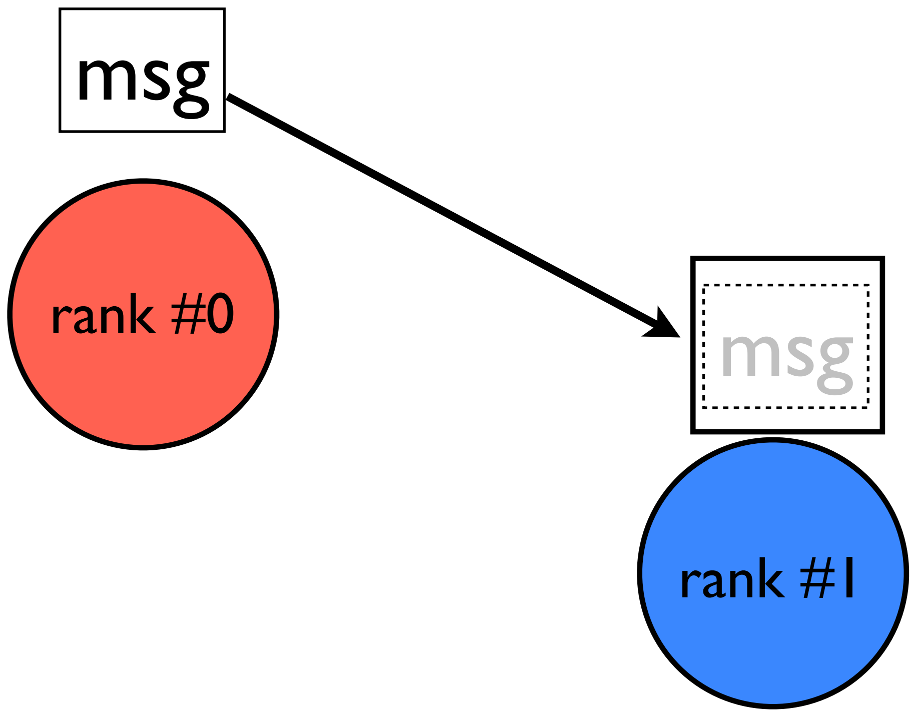
Point to Point



- Typically multiple protocols.
- None of this is in the standard; future implementations may use additional or different approaches entirely

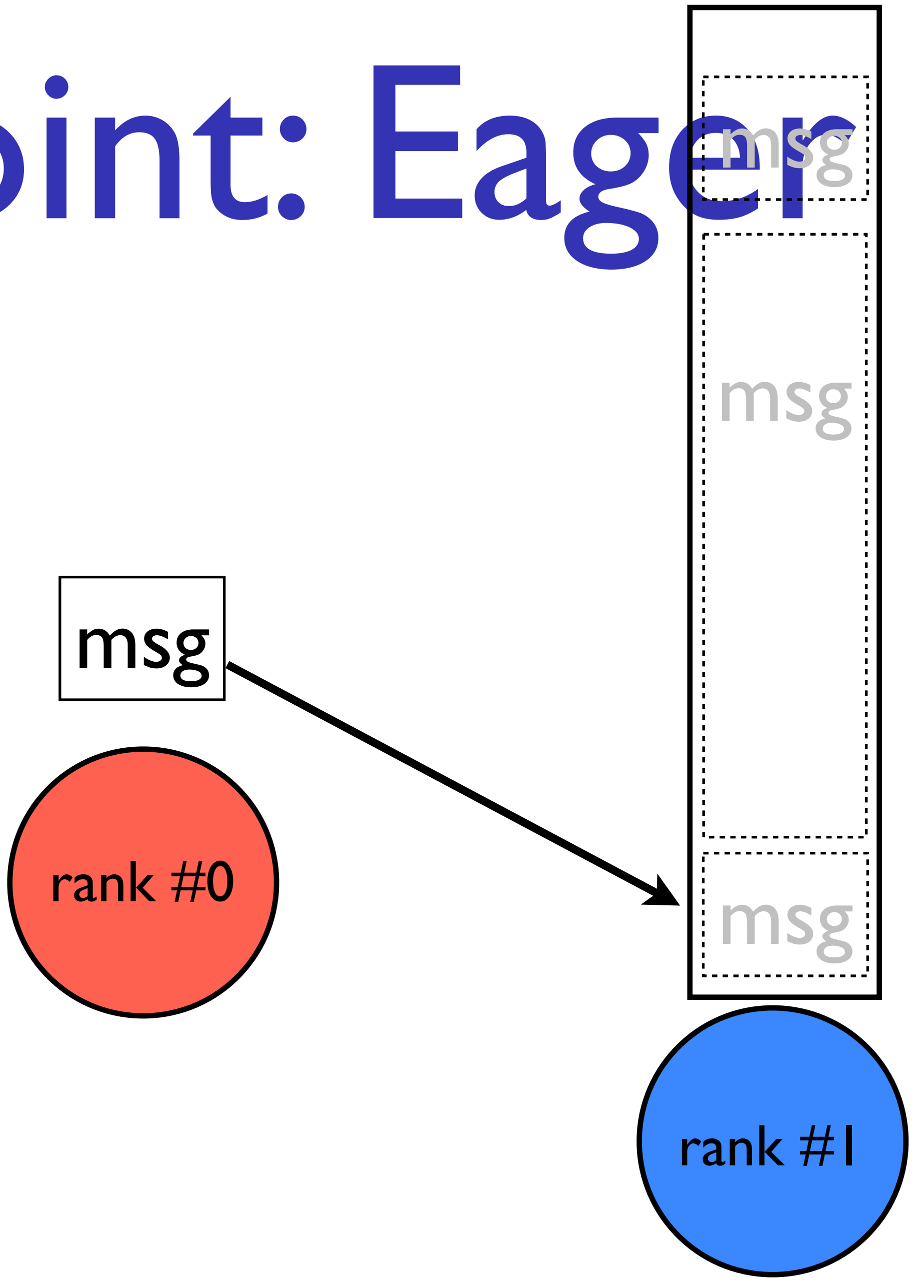
Point to Point: Eager

- Eager messages: sender plops message in MPI-defined system buffer on receive end.
- 1 transit of network



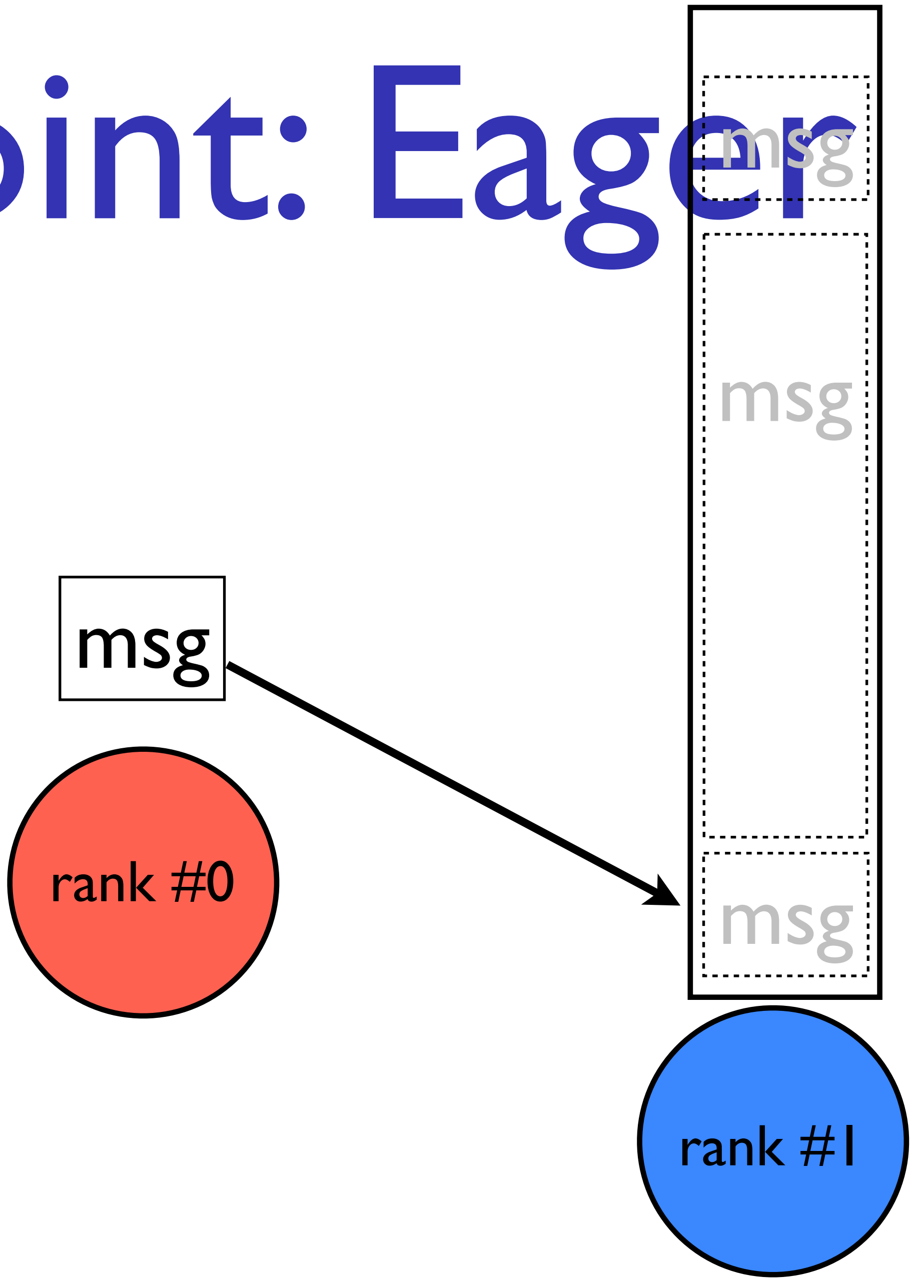
Point to Point: Eager

- But what if several messages pile up..
- And are quite large?
- (~100MB messages not uncommon in HPC)



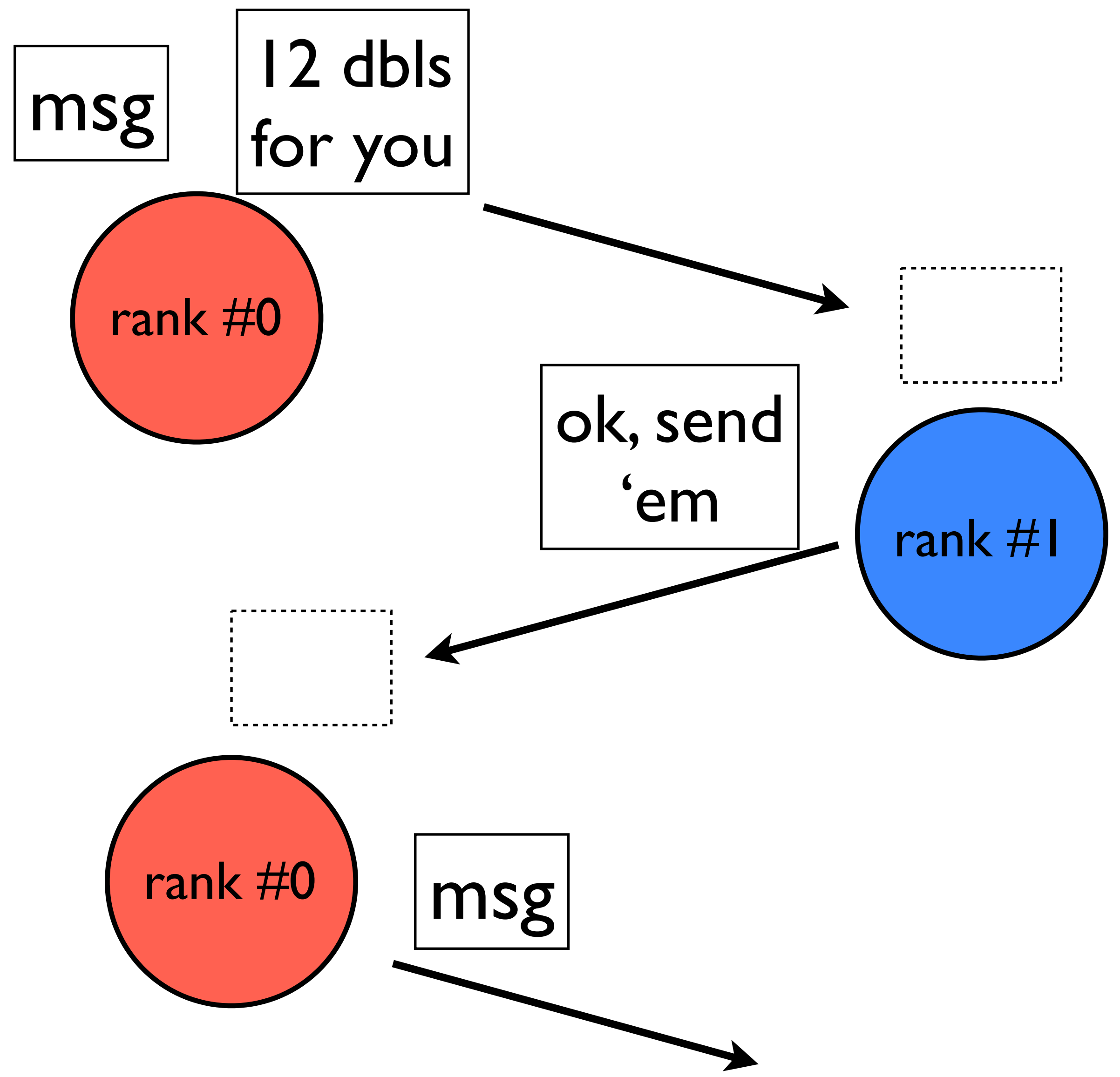
Point to Point: Eager

- Can't afford to dedicate large chunk of memory to receiveing data, "just in case"



Point to Point: Rendezvous

- Rendezvous protocol: 3-way handshake
- First message is just "envelope" - describes contents
- Small, fits in memory.



Eager vs. Rendezvous

- Eager: faster, lower latency, requires big buffers on receive
- Rendezvous: much lower memory overhead for big messages, much larger latency esp. on slow networks
- But Rendezvous doesn't save any memory for messages approximately the size of the envelope..

Eager vs. Rendezvous

- Send via Eager protocol for “small enough” messages
- Send via Rendezvous for “large” messages.
- “Eager Threshold” threshold tunable
- Typically one threshold per network type

Protocols

- OpenMPI, MPICH2, etc implement much more just these two protocols
- Also transport-specific protocols/policies
- Policies for fragment sizes to use for large messages, pipelining, etc.
- But eager vs. handshake good distinction to know

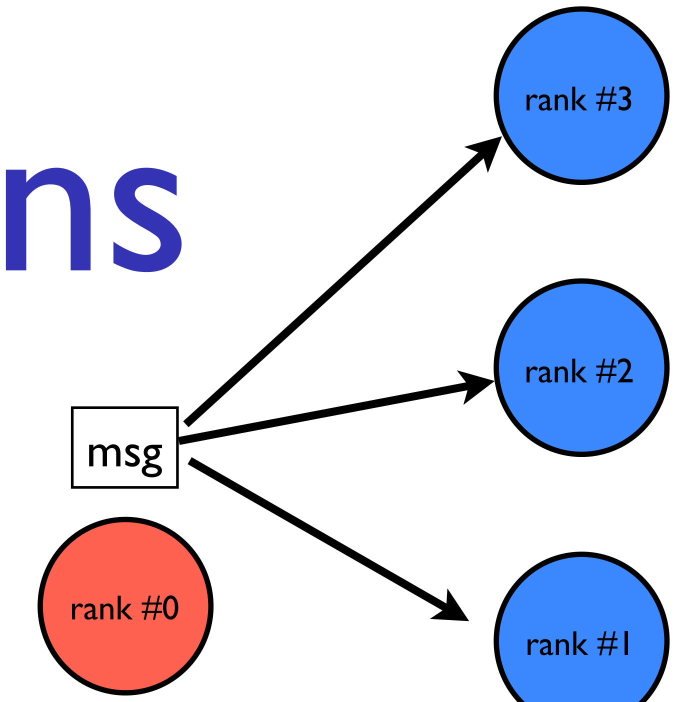
Setting eager thresholds

- OpenMPI:
  - --mca btl_sm_eager_limit [num] (default: 4k)
  - --mca btl_openib_eager_limit [num] (default: 12k)
  - --mca btl_tcp_eager_limit [num] (default: 64k)
  - Or: (eg)
    export OMPI_MCA_btl_sm_eager_limit=4096

Setting eager thresholds

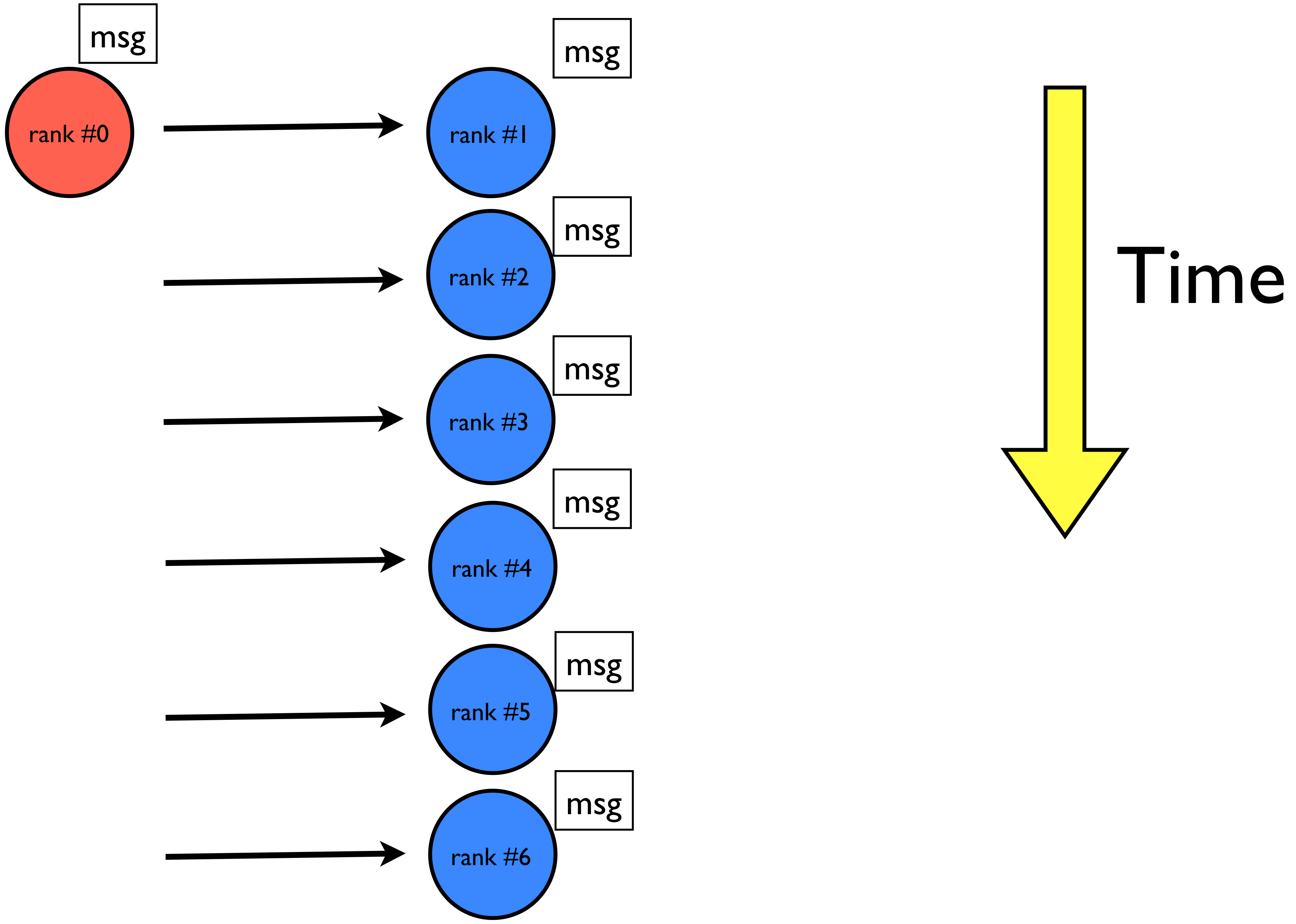
- IntelMPI:
  - -genv I_MPI_EAGER_THRESHOLD [num] (default: 256k)
  - -genv I_MPI_INTRANODE_EAGER_THRESHOLD [num] (default: 256k)
  - -genv I_MPI_RDMA_EAGER_THRESHOLD [num] (default: 16k)
  - Or: (eg) export I_MPI_EAGER_THRESHOLD=4096

Collective Communications



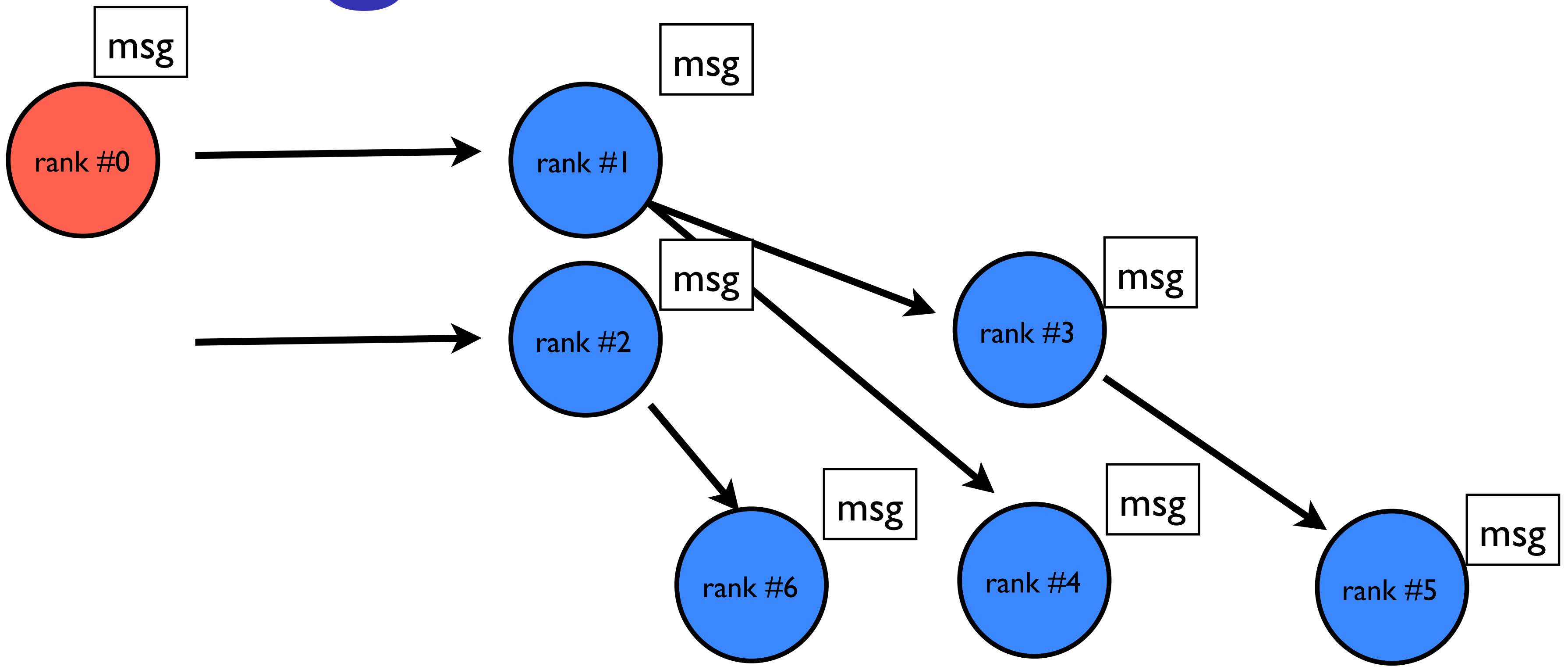
- Broadcast, Allreduce,...
- All the different ways above to send each individual message;
- *plus* decisions about which messages to send!

Linear:

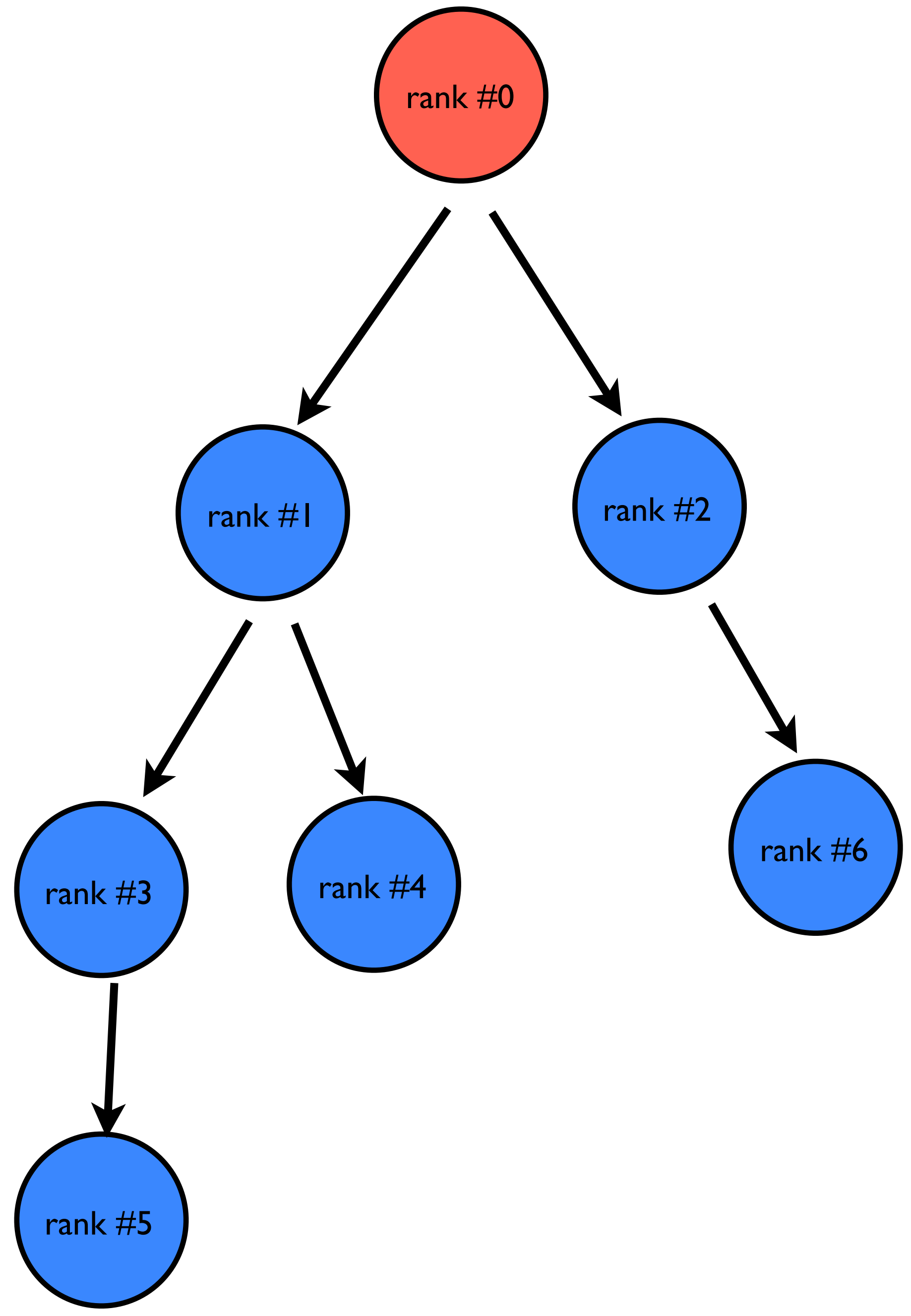


msg
rank #0
msg
rank #1
msg
rank #2
msg
rank #3
msg
rank #4
msg
rank #5
msg
rank #6
Time

Logarithmic Tree



Logarithmic Tree



Linear vs. Logarithmic

- Hierarchical tree obviously scales much better
  - lg(P) steps vs. P
- But for small P, linear actually faster - lower overhead.

Other considerations

- Modern clusters are hierarchial:
  - many cores in a node
  - many nodes on a switch
  - many switches in a cluster
- Modern MPI implementations have many collective algorithms, chosen depending on P, size of message, fabric...

Adjusting algorithms

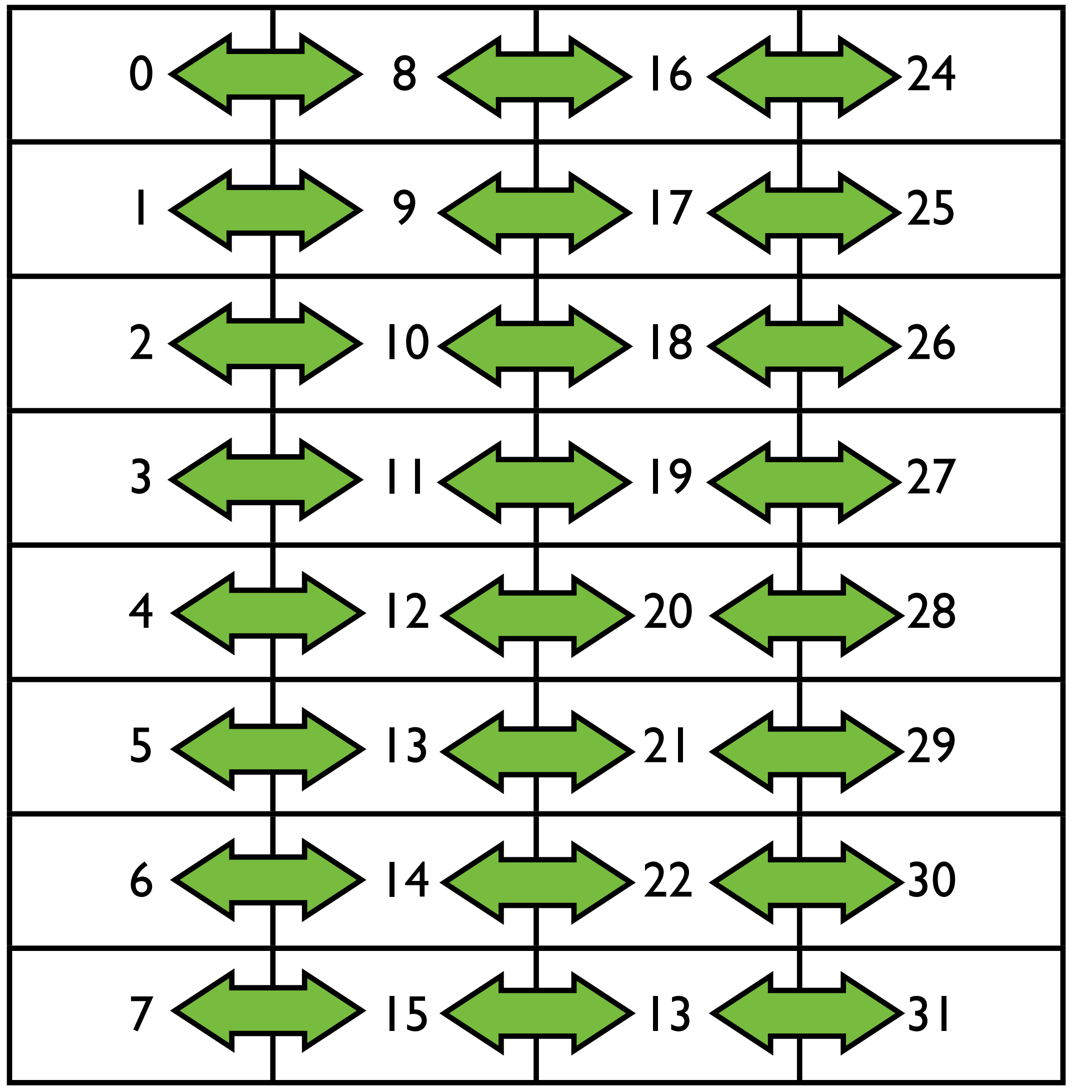
- IntelMPI
  - I_MPI_ADJUST_ALLREDUCE [num] (eg)
    - choose algorithm #[num]
- OpenMPI
  - --mca coll (many)
  - ompi_info --param coll all

Utilities to test parameters for you

- mpitune (IntelMPI)
- otpo (OpenMPI: not nearly as full featured, mainly for sysadmins)

Analysis.c

- Example program
- 256^2 x 32 array
- Pipeline data across rows, auto-correlation
- Allreduce answers



```
MPI_Dims_create(size, 2, dims);   /* eg, an 8x4 grid on 4 nodes */

int left  = (rank - dims[0] + size) % size;
int right = (rank + dims[0]) % size;

int rows = (problemsize + (row/dims[0]))/dims[0];
int cols = (problemsize + (col/dims[1]))/dims[1];

int ndata = problemdepth*rows*cols;
double *data = malloc(ndata*sizeof(double));
double *extdata = malloc(ndata*sizeof(double));
double *result  = malloc(ndata*sizeof(double));
/* ... */

for (int iter=0; iter<5; iter++) {
    /* ... */

    /* calculate on local data */

    /* get external data and calculate on it */
    for (int i=1; i<dims[1]; i++) {
        MPI_Sendrecv(data, ndata, MPI_DOUBLE, (rank + i*dims[0])%size, i,
                     extdata, ndata, MPI_DOUBLE, MPI_ANY_SOURCE, i,
                     MPI_COMM_WORLD, &status);

        /* do something with data */
    }

   /* get some local max */
    MPI_Allreduce(&locmaxres, &maxres, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);
}

MPI_Finalize();
```

Analysis.c

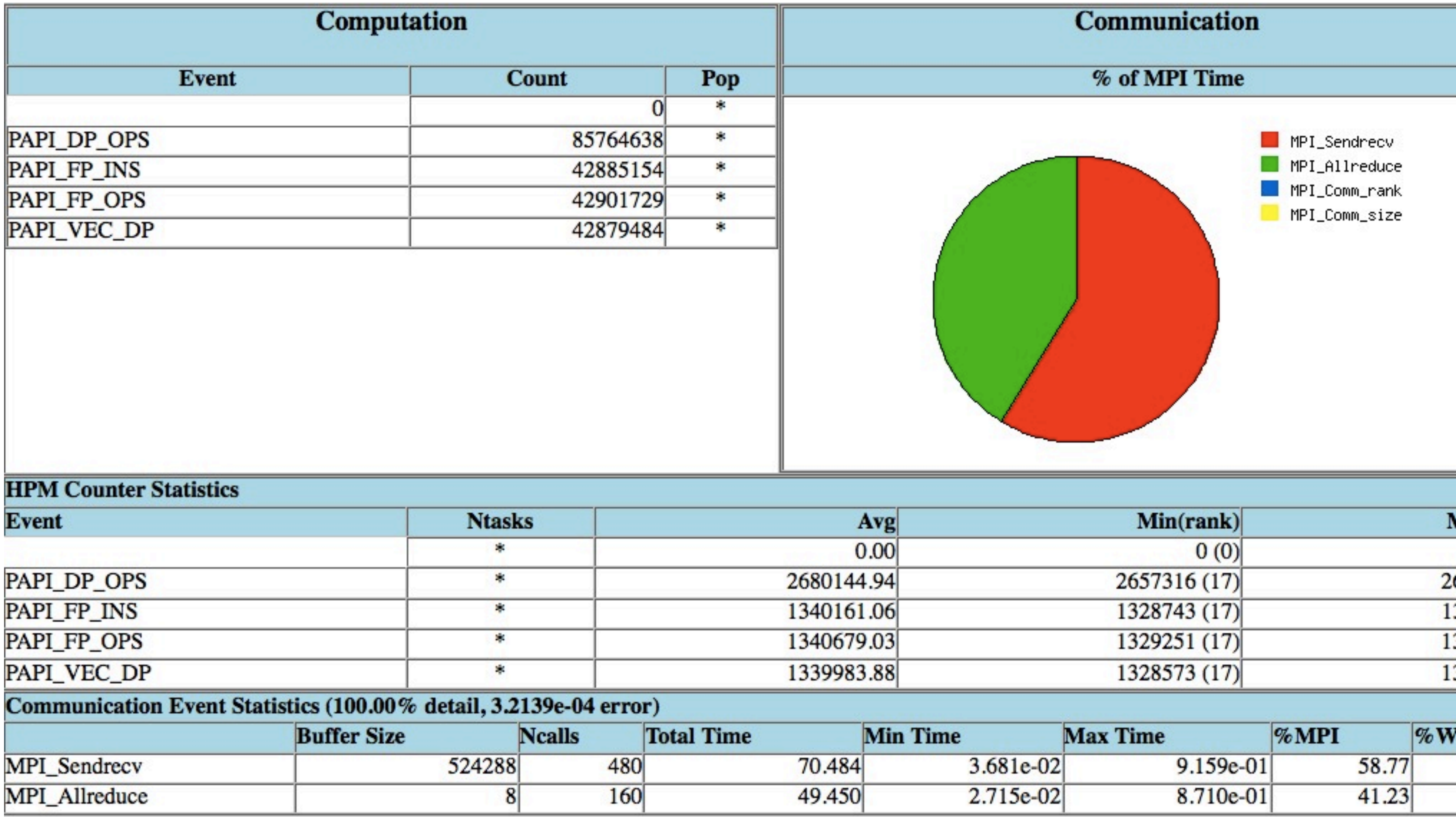
- Run for 5 iterations with IPM
- module avail ipm
- mpicc .... -L${SCINET_IPM_LIB} -lipm
- Can't improve performance if you don't measure it...
- Start with Intel MPI library defaults

Analysis.c

```
$ mpirun -genv I_MPI_FABRICS shm:tcp
    -np 32 ./analysis

$ ipm_parse -html ljdursi.*
```

```
##IPMv0.983####################################################################
#
# command : ./analysis  (completed)
# host    : gpc-f104n043/x86_64_Linux      mpi_tasks : 32 on 4 nodes
# start   : 02/06/12/08:17:44              wallclock : 3.769331 sec
# stop    : 02/06/12/08:17:47              %comm     : 99.43
# gbytes  : 3.56665e+00 total              gflop/sec : 1.13818e-02 total
#
###############################################################################
# region  : *       [ntasks] =     32
#
#                          [total]        <avg>          min            max
# entries                       32            1            1              1
# wallclock                120.527      3.76648      3.75933        3.76933
# user                     94.4216      2.95068      1.71474        3.36049
# system                   27.3028     0.853214     0.443932        2.08168
# mpi                      119.934      3.74795      3.74724        3.74866
# %comm                                 99.4327      99.4332        99.6855
# gflop/sec              0.0113818  0.000355681  0.000352649    0.000357786
# gbytes                   3.56665     0.111458     0.111458       0.111458
#
# PAPI_FP_OPS          4.29017e+07  1.34068e+06  1.32925e+06    1.34861e+06
# PAPI_FP_INS          4.28852e+07  1.34016e+06  1.32874e+06    1.34812e+06
# PAPI_DP_OPS          8.57646e+07  2.68014e+06  2.65732e+06    2.69606e+06
# PAPI_VEC_DP          4.28795e+07  1.33998e+06  1.32857e+06    1.34794e+06
#
#                           [time]      [calls]       <%mpi>        <%wall>
# MPI_Sendrecv             70.4843          480        58.77          58.48
# MPI_Allreduce            49.4501          160        41.23          41.03
# MPI_Comm_rank        1.01876e-05           32         0.00           0.00
# MPI_Comm_size        7.42101e-06           32         0.00           0.00
###############################################################################
```

99% of time spent in communications



Computation		
Event	Count	Pop
	0	*
PAPI_DP_OPS	85764638	*
PAPI_FP_INS	42885154	*
PAPI_FP_OPS	42901729	*
PAPI_VEC_DP	42879484	*

Communication

% of MPI Time

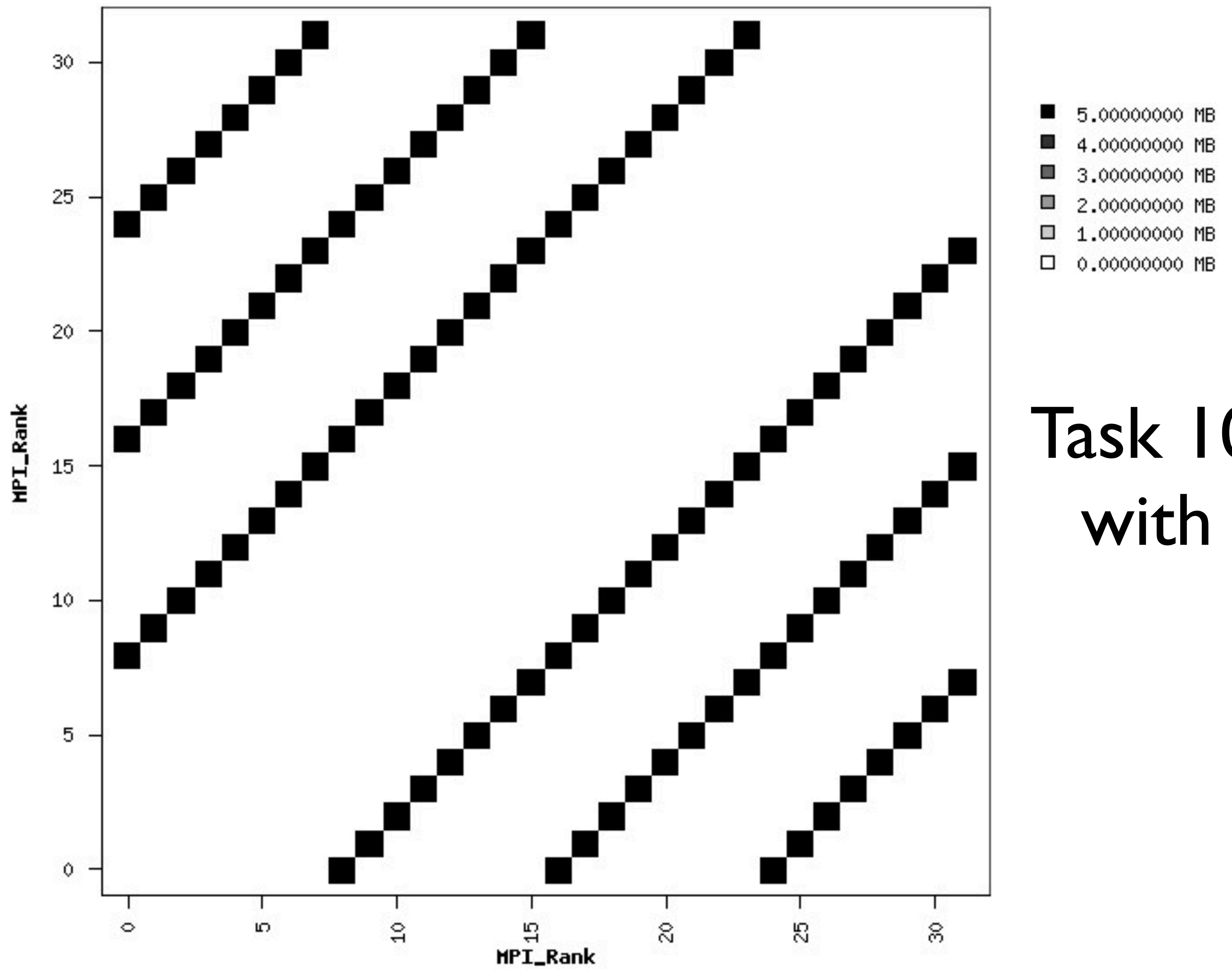
HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)
	*	0.00	0 (0)
PAPI_DP_OPS	*	2680144.94	2657316 (17)
PAPI_FP_INS	*	1340161.06	1328743 (17)
PAPI_FP_OPS	*	1340679.03	1329251 (17)
PAPI_VEC_DP	*	1339983.88	1328573 (17)

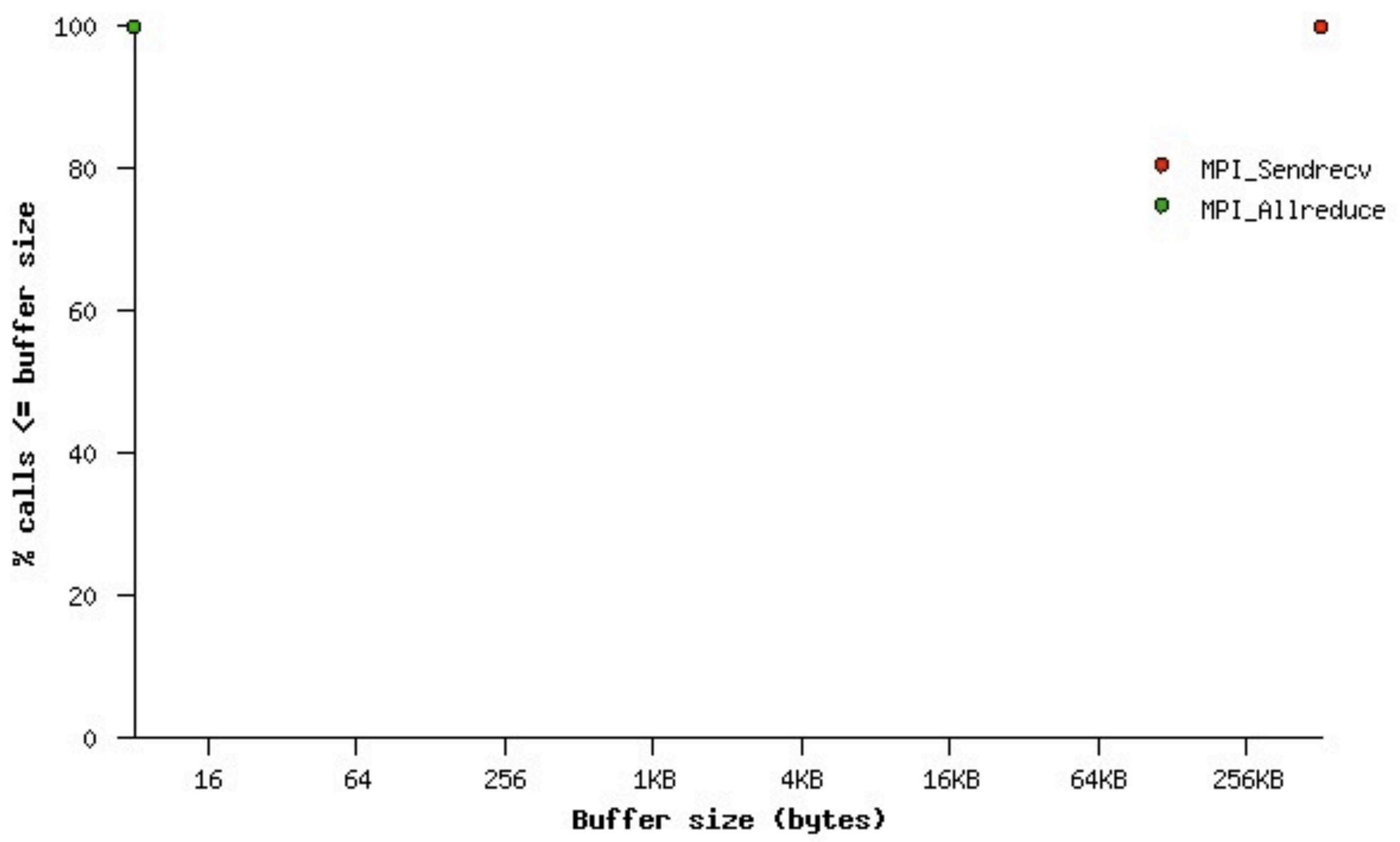
Communication Event Statistics (100.00% detail, 3.2139e-04 error)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI
MPI_Sendrecv	524288	480	70.484	3.681e-02	9.159e-01	58.77
MPI_Allreduce	8	160	49.450	2.715e-02	8.710e-01	41.23

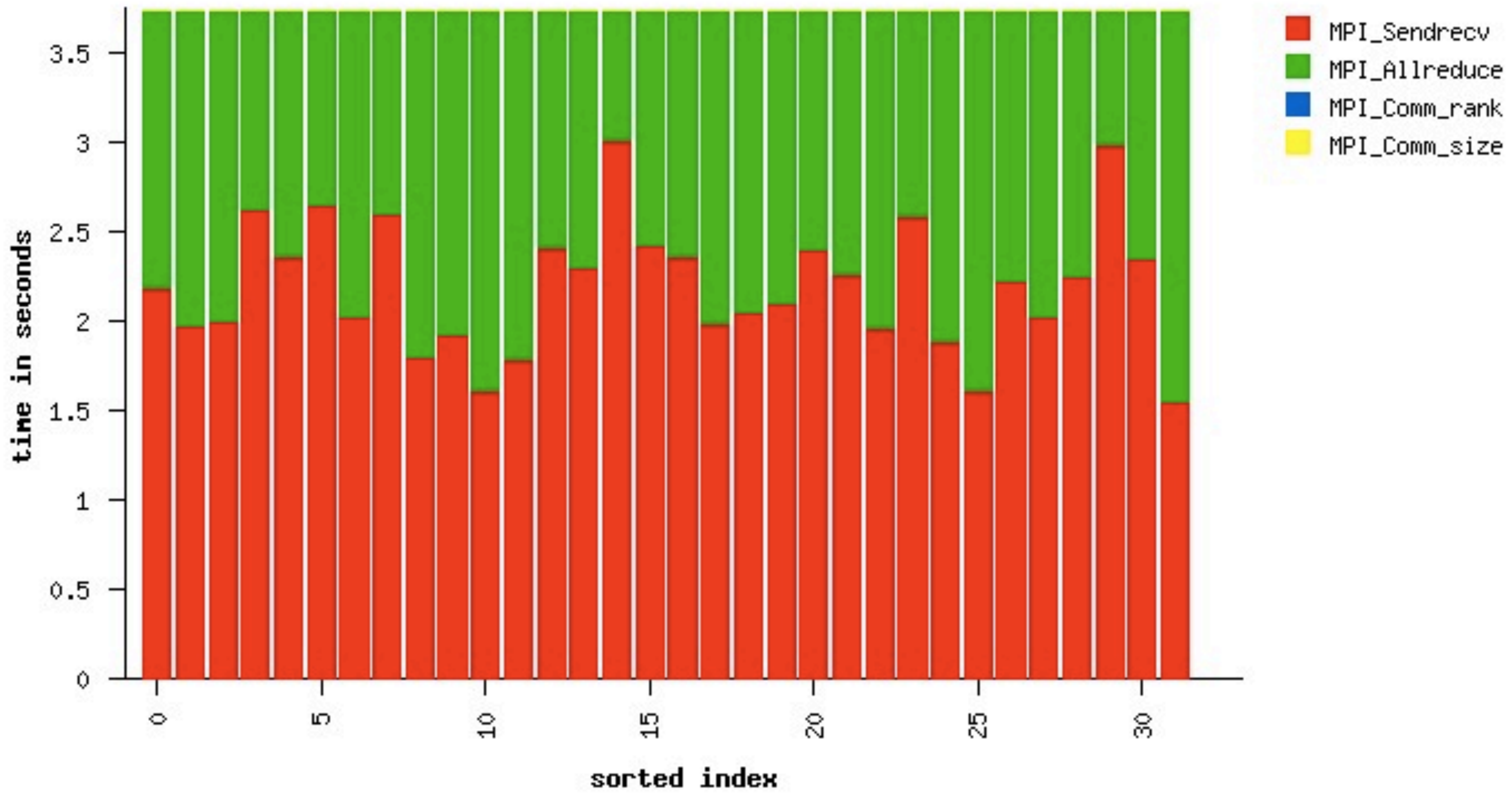
480 sendrecvs, 160 allreduces (across all procs)
sendrecv: 70s, 59% of MPI time.



Task 10 communicates with tasks 18, 26, 2; etc.



Only two message sizes:
Allreduce (single float)
Sendrecv (~32x32x64 floats)



About 3.75 s/task in MPI; a lot of allreduce time is likely due to load imbalance (communications)

mpitune

- IntelMPI utility
- Repeatedly (~couple dozen times, maybe more) runs your program while changing a handful of MPI parameters
- Have a *short* but realistically *sized* version of your probem for this!
- Can change the default bundle of parameters.

mpitune

```
$ mpitune -of analysis.conf
   --application \"mpiexec -genv
   I_MPI_FABRICS shm:tcp -n 32
   analysis-noipm\"

$ mpirun -genv I_MPI_FABRICS shm:tcp
      -tune analysis.conf
      -np 32 ./analysis

$ ipm_parse -html ljdursi.*
```

analysis.conf

```
-genv I_MPI_RDMA_SCALABLE_PROGRESS  0
-genv I_MPI_WAIT_MODE  1
-genv I_MPI_INTRANODE_EAGER_THRESHOLD  2097152
-genv I_MPI_RDMA_EAGER_THRESHOLD  3145728
-genv I_MPI_ADJUST_ALLREDUCE  '6:8-8'
```

Eager threshold increased
(2MB! But there's always a waiting receive)
RDMA not used here
Allreduce algorithm changed

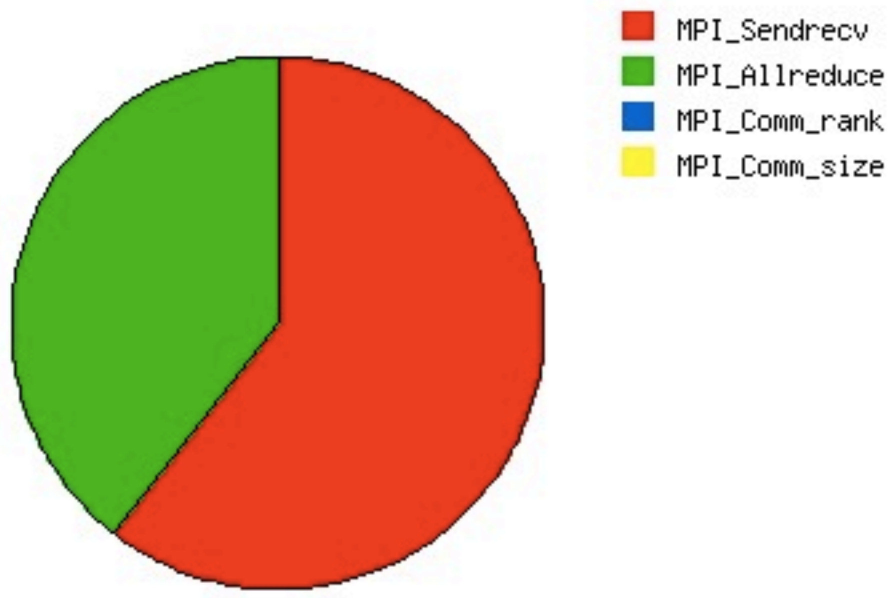
```
##IPMv0.983####################################################################
#
# command : ./analysis  (completed)
# host    : gpc-f104n043/x86_64_Linux      mpi_tasks : 32 on 4 nodes
# start   : 02/06/12/08:25:29              wallclock : 3.064447 sec
# stop    : 02/06/12/08:25:32              %comm     : 99.45
# gbytes  : 3.56665e+00 total              gflop/sec : 1.39936e-02 total
#
##############################################################################
# region  : *       [ntasks] =     32
#
#                           [total]        <avg>          min           max
# entries                        32            1            1             1
# wallclock                 98.0203      3.06313      3.06145       3.06445
# user                      75.0186      2.34433      1.61775       2.68559
# system                    24.1903     0.755947     0.413937       1.48877
# mpi                       97.5254      3.04767      3.04669       3.04825
# %comm                                99.4525      99.4511       99.5637
# gflop/sec               0.0139936  0.000437301  0.000433963   0.000440101
# gbytes                    3.56665     0.111458     0.111458      0.111458
#
# PAPI_FP_OPS           4.28828e+07  1.34009e+06  1.32986e+06   1.34867e+06
# PAPI_FP_INS           4.28661e+07  1.33957e+06  1.32935e+06   1.34815e+06
# PAPI_DP_OPS           8.57265e+07  2.67895e+06  2.65852e+06   2.69614e+06
# PAPI_VEC_DP           4.28604e+07  1.33939e+06  1.32917e+06   1.34798e+06
#
#                            [time]      [calls]        <%mpi>      <%wall>
# MPI_Sendrecv              59.0132          480        60.51         60.21
# MPI_Allreduce             38.5122          160        39.49         39.29
# MPI_Comm_rank         8.79297e-06           32         0.00          0.00
# MPI_Comm_size          7.6741e-06           32         0.00          0.00
###############################################################################
```

SendRecv: 59 (was 71); Allreduce 39 (was 50)

Computation		
Event	Count	Pop
	0	*
PAPI_DP_OPS	85726533	*
PAPI_FP_INS	42866118	*
PAPI_FP_OPS	42882785	*
PAPI_VEC_DP	42860415	*

Communication

% of MPI Time



HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Ma
	*	0.00	0 (0)	
PAPI_DP_OPS	*	2678954.16	2658521 (21)	2696
PAPI_FP_INS	*	1339566.19	1329347 (21)	1348
PAPI_FP_OPS	*	1340087.03	1329856 (21)	1348
PAPI_VEC_DP	*	1339387.97	1329174 (21)	1347

Communication Event Statistics (100.00% detail, -2.4647e-04 error)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wal
MPI_Sendrecv	524288	480	59.013	2.298e-02	6.650e-01	60.51	
MPI_Allreduce	8	160	38.512	2.292e-04	6.296e-01	39.49	

mpitune

- 20% improvement in runtime! (Extreme case)
- Can work very well for a code dominated by one (or very small number of) communications patters
- Need to find shortest-time case that exercises all of the communications patterns on real-sized problems.
- Works only with IntelMPI

otpo

- Part of OpenMPI suite of tools
- Mainly used for tuning OpenMPI as a whole for given cluster
- Runs well-established benchmarks (NAS, netpipe, Skapi)
- If your code looks like one of those, can be useful.

Locality

- Can use 'hostname' to find out what hosts are being used
- And with intel mpi, "-l" labels the output by each rank

```
$ mpirun -l -np 32 hostname | sort -n
0: gpc-f109n001
1: gpc-f109n001
2: gpc-f109n001
3: gpc-f109n001
4: gpc-f109n001
5: gpc-f109n001
6: gpc-f109n001
7: gpc-f109n001

8: gpc-f109n002
9: gpc-f109n002
10: gpc-f109n002
11: gpc-f109n002
12: gpc-f109n002
```

Locality

- OpenMPI: "--tag-output"
- [exe,rank]

```
$ mpirun --tag-output -np 32 hostname

[1,0]<stdout>:gpc-f109n001
[1,1]<stdout>:gpc-f109n001
[1,2]<stdout>:gpc-f109n001
[1,3]<stdout>:gpc-f109n001
[1,4]<stdout>:gpc-f109n001
[1,5]<stdout>:gpc-f109n001
[1,6]<stdout>:gpc-f109n001
[1,7]<stdout>:gpc-f109n001
[1,8]<stdout>:gpc-f109n002
[1,9]<stdout>:gpc-f109n002
[1,10]<stdout>:gpc-f109n002
[1,11]<stdout>:gpc-f109n002
[1,12]<stdout>:gpc-f109n002
[1,13]<stdout>:gpc-f109n002
[1,14]<stdout>:gpc-f109n002
....
```

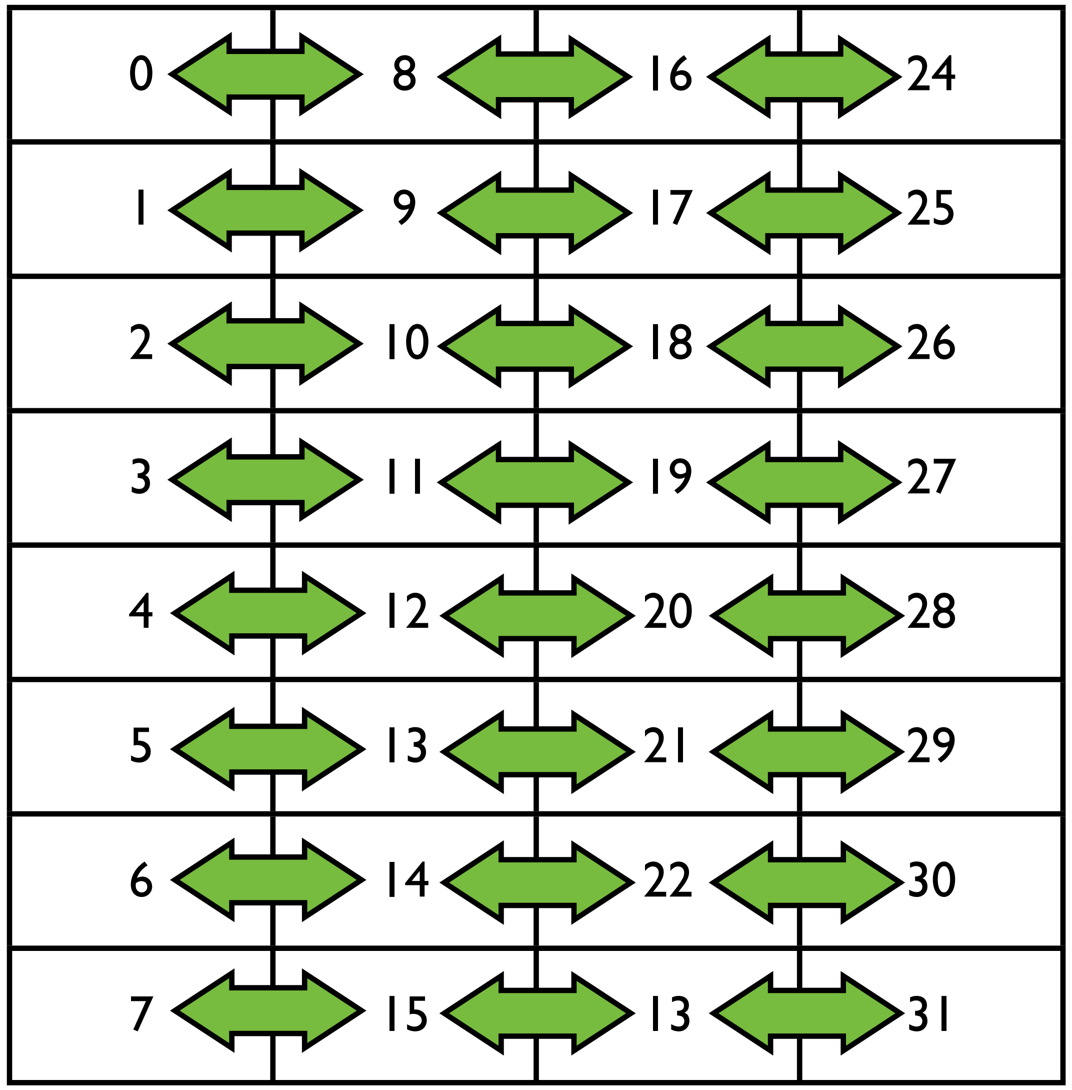
Locality

- OpenMPI: "--display-map"
- At start of job, lays out the ranks on each host

```
mpirun -display-map -np 32 hostname

========================   JOB MAP   ===========

Data for node: Name: gpc-f109n001   Num procs: 8
    Process OMPI jobid: [932,1] Process rank: 0
    Process OMPI jobid: [932,1] Process rank: 1
    Process OMPI jobid: [932,1] Process rank: 2
    Process OMPI jobid: [932,1] Process rank: 3
    Process OMPI jobid: [932,1] Process rank: 4
    Process OMPI jobid: [932,1] Process rank: 5
    Process OMPI jobid: [932,1] Process rank: 6
    Process OMPI jobid: [932,1] Process rank: 7

Data for node: Name: gpc-f109n002   Num procs: 8
    Process OMPI jobid: [932,1] Process rank: 8
    Process OMPI jobid: [932,1] Process rank: 9
    Process OMPI jobid: [932,1] Process rank: 10
    Process OMPI jobid: [932,1] Process rank: 11
    Process OMPI jobid: [932,1] Process rank: 12
    Process OMPI jobid: [932,1] Process rank: 13
    Process OMPI jobid: [932,1] Process rank: 14
    Process OMPI jobid: [932,1] Process rank: 15

....
==================================================
```

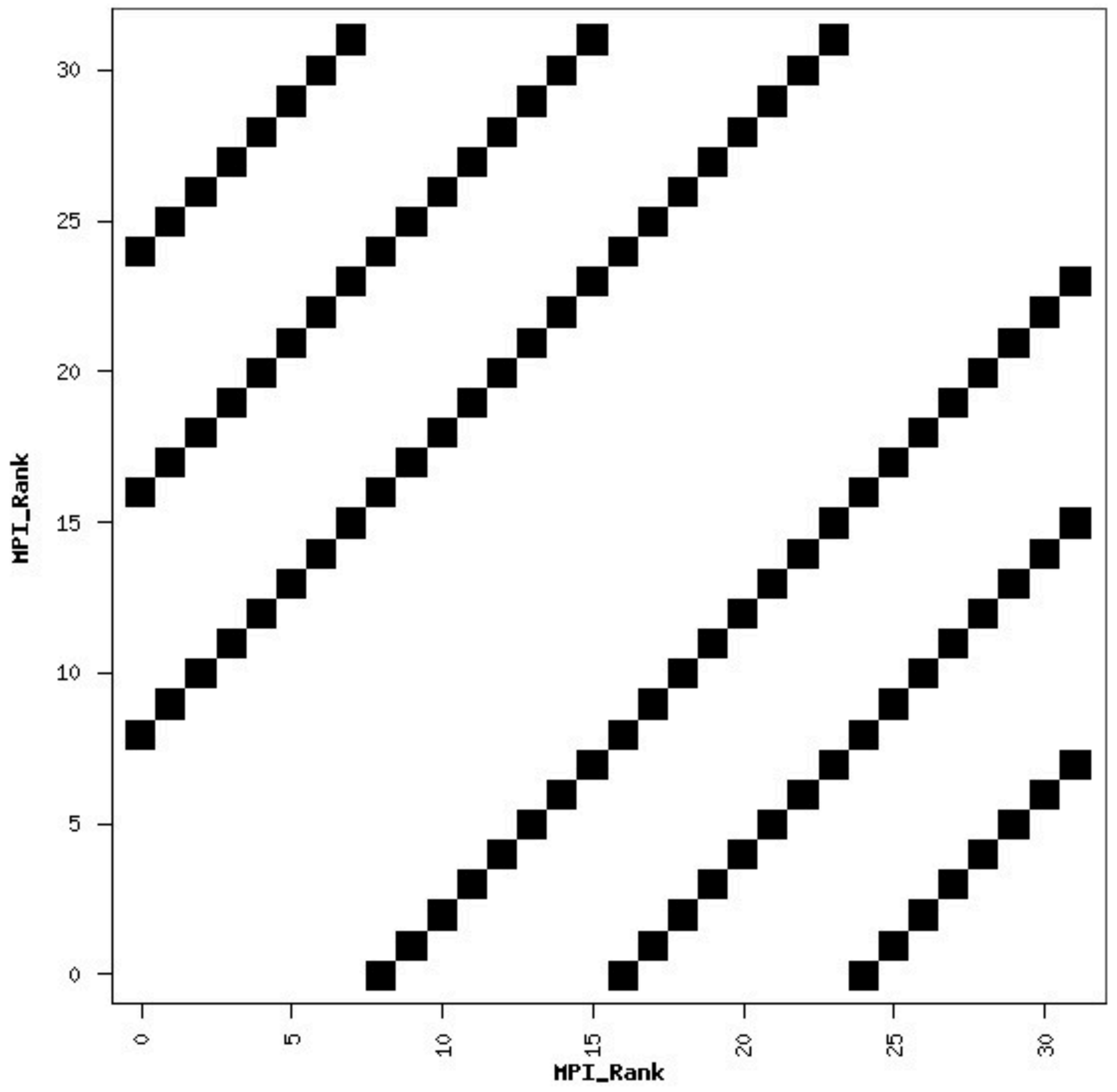
Locality

- Note the setup for our analysis routine
- Almost all the communications going off-node
- Off-node always slower than on.



Locality

- Note the setup for our analysis routine
- Almost all the communications going off-node
- Off-node always slower than on.



Round-Robin allocation

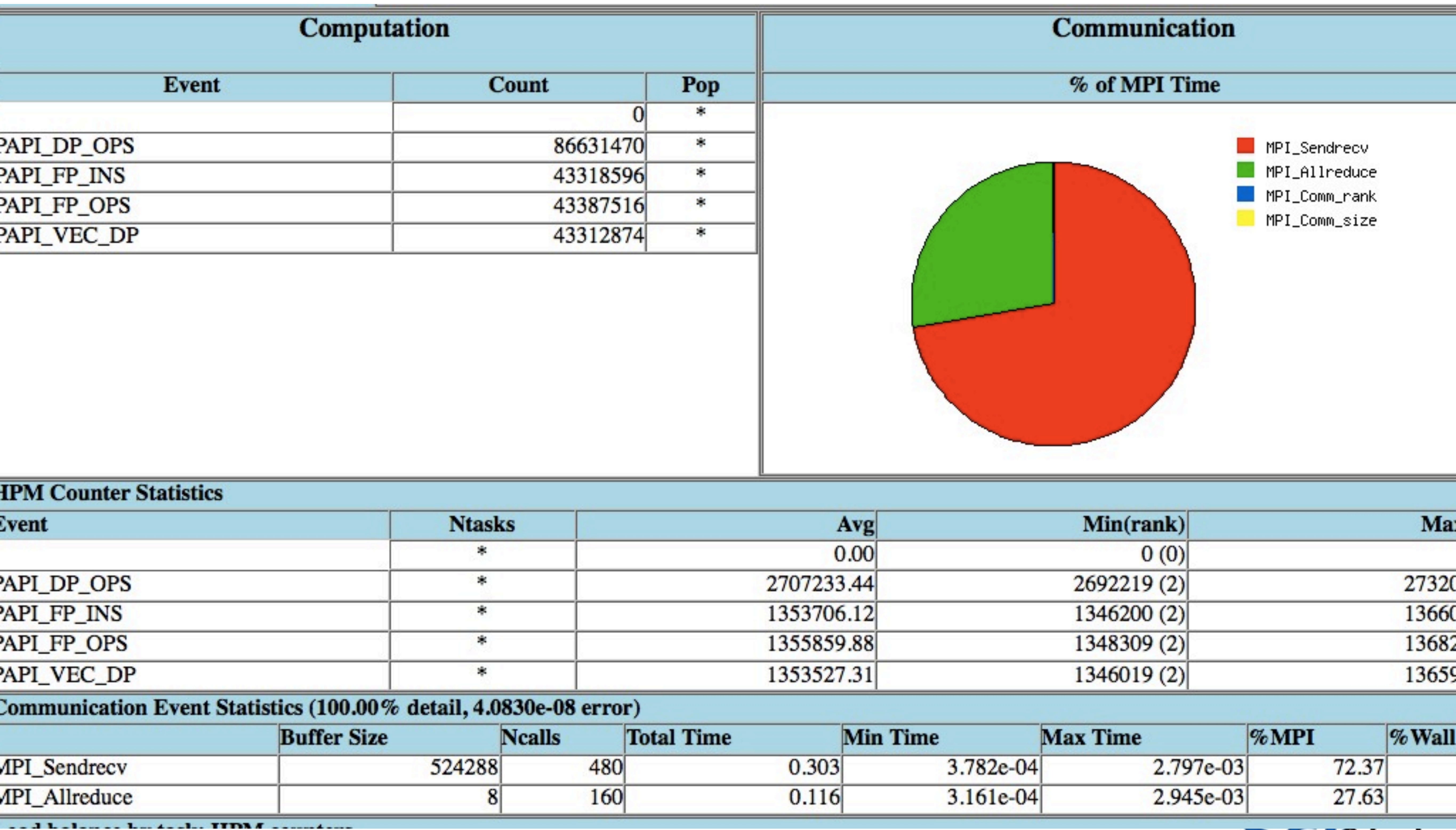
- Instead of filling up a node before next,
- Puts one rank on node, 2nd rank on 2nd node, etc.

```
$ mpirun -l -rr -np 32 hostname | sort -n
0: gpc-f109n001
1: gpc-f109n006
2: gpc-f109n005
3: gpc-f109n002
4: gpc-f109n001
5: gpc-f109n006
6: gpc-f109n005
7: gpc-f109n002
8: gpc-f109n001
9: gpc-f109n006
10: gpc-f109n005
11: gpc-f109n002
12: gpc-f109n001
```

Round-Robin allocation

- IntelMPI: -rr
- OpenMPI --bynode
- Other OpenMPI options: --bysocket, --bycore..

Run with -rr



Computation

Event	Count	Pop
	0	*
PAPI_DP_OPS	86631470	*
PAPI_FP_INS	43318596	*
PAPI_FP_OPS	43387516	*
PAPI_VEC_DP	43312874	*

Communication

% of MPI Time

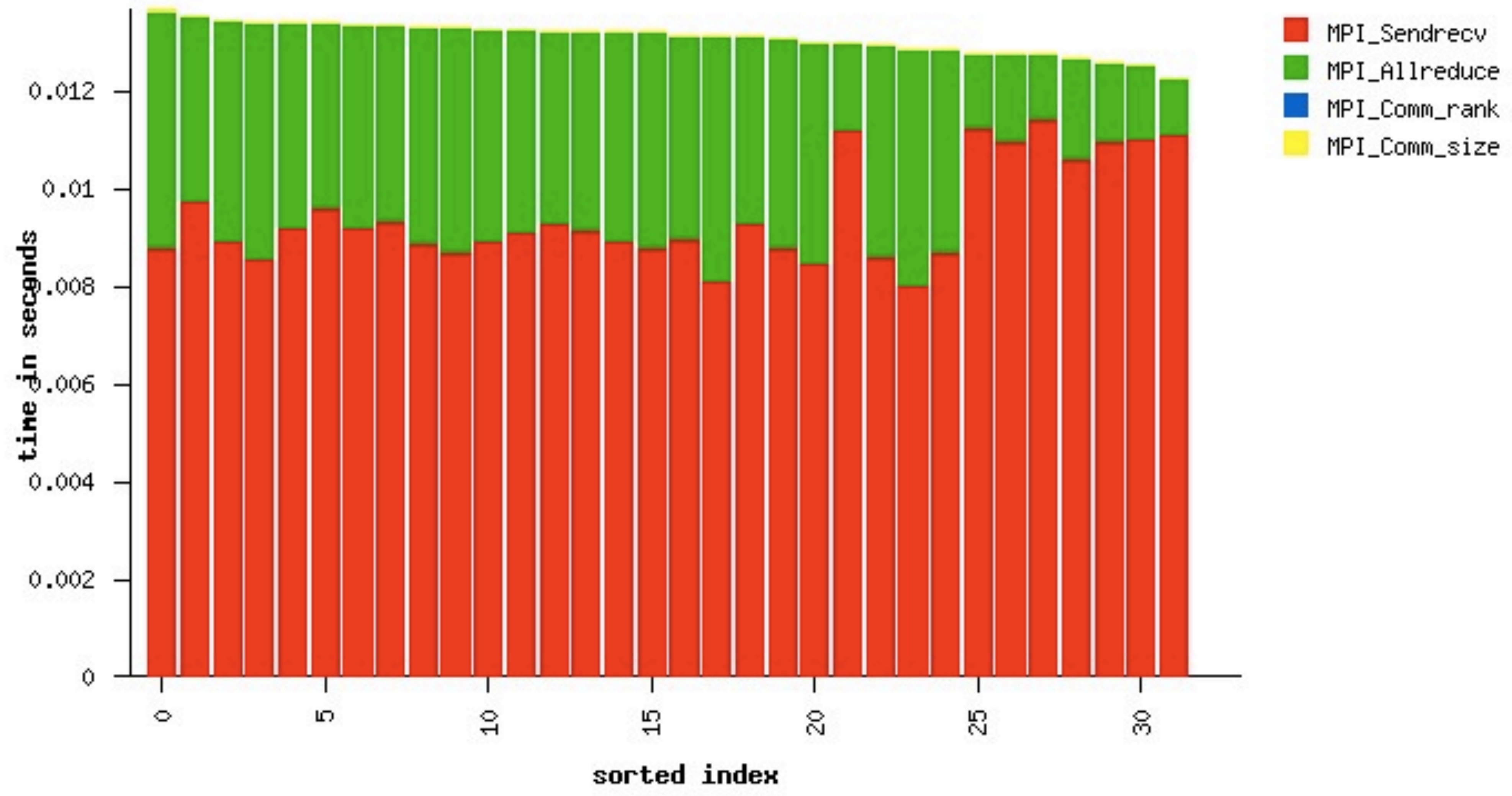
HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Ma
	*	0.00	0 (0)	
PAPI_DP_OPS	*	2707233.44	2692219 (2)	27320
PAPI_FP_INS	*	1353706.12	1346200 (2)	13660
PAPI_FP_OPS	*	1355859.88	1348309 (2)	13682
PAPI_VEC_DP	*	1353527.31	1346019 (2)	13659

Communication Event Statistics (100.00% detail, 4.0830e-08 error)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Sendrecv	524288	480	0.303	3.782e-04	2.797e-03	72.37	
MPI_Allreduce	8	160	0.116	3.161e-04	2.945e-03	27.63	

Load balance by task: HPM counters



Huge difference! By keeping most communications on-node, enormously reduce runtime.

Locality: -rr

- An admittedly exteme case, but an important point
- Layout of nodes for locality is extremely important.
- Could also fix this in the code by reordering (MPI_CART_CREATE)
- Even this case can be tuned, for improvements in allreduce

Hybrid MPI/OpenMP

- Locality is extremely important in the case of hybrid codes.
- Typically you want one MPI task per node (or per socket), and multiple threads per task.
- Want them to stay put; threads shouldn't move around within the node.

OpenMPI

- hwloc library implements binding
- Make sure you specify how many cores per rank:
- Default will just

Good:

```
gpc-f109n002-$ mpirun --display-map -cpus-per-rank 4 -np 8 hostname

 ========================   JOB MAP   ========================

 Data for node: Name: gpc-f109n002 Num procs: 2
   Process OMPI jobid: [61447,1] Process rank: 0
   Process OMPI jobid: [61447,1] Process rank: 1

 Data for node: Name: gpc-f109n003 Num procs: 2
   Process OMPI jobid: [61447,1] Process rank: 2
   Process OMPI jobid: [61447,1] Process rank: 3

 Data for node: Name: gpc-f109n004 Num procs: 2
   Process OMPI jobid: [61447,1] Process rank: 4
   Process OMPI jobid: [61447,1] Process rank: 5

 Data for node: Name: gpc-f109n005 Num procs: 2
   Process OMPI jobid: [61447,1] Process rank: 6
   Process OMPI jobid: [61447,1] Process rank: 7

 =============================================================
```

Bad:

```
$ mpirun --display-map -np 8 hostname

 ========================   JOB MAP   ========================

 Data for node: Name: gpc-f109n002 Num procs: 8
   Process OMPI jobid: [61470,1] Process rank: 0
   Process OMPI jobid: [61470,1] Process rank: 1
   Process OMPI jobid: [61470,1] Process rank: 2
   Process OMPI jobid: [61470,1] Process rank: 3
   Process OMPI jobid: [61470,1] Process rank: 4
   Process OMPI jobid: [61470,1] Process rank: 5
   Process OMPI jobid: [61470,1] Process rank: 6
   Process OMPI jobid: [61470,1] Process rank: 7

 =============================================================
```

IntelMPI

- Same deal; if you only want (say) 2 tasks per node, use -perhost 2
- -rr if you want them to be round-robined between nodes.

Specifying process maps

- If you have a specific process layout in mind, either MPI library will allow you to do that.
- With hybrid codes, in IntelMPI, best to export OMP_NUM_THREADS to the appropriate number, and then use I_MPI_PIN_DOMAIN=omp to keep threasds in right place

Conclusions

- Be aware of where your processes are communicating
- IPM is an invaluable tool for this!
- mpitune is worth using IntelMPI for