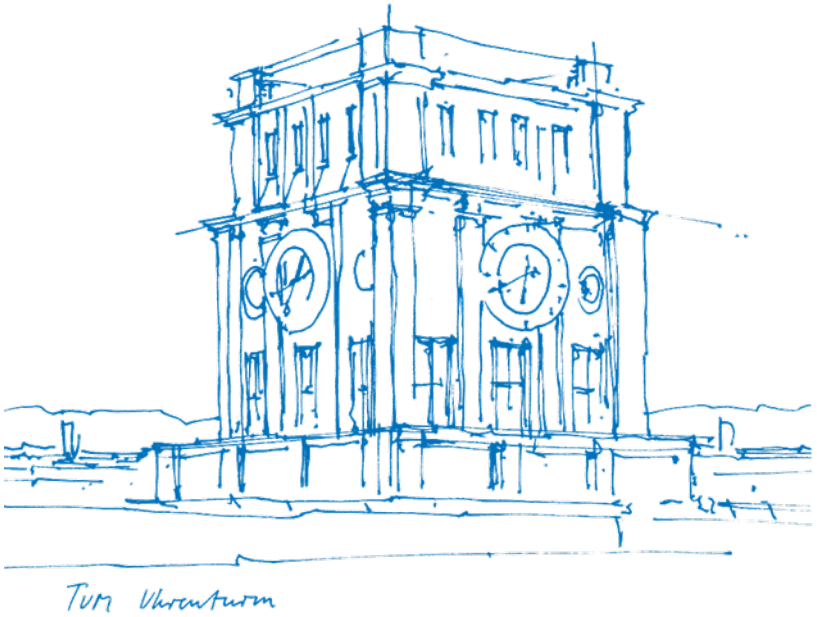
Parallel Programming Tutorial – More on MPI

Bengisu Elis, M.Sc.
Philipp Czerner
Hasan Ashraf
Chair for Computer Architecture and Parallel Systems (Prof. Schulz)
Technical University of Munich
3 July 2019



Organization

Organization

- Assignment 8 will be published on 3rd July - deadline on 9th July at 23:59
- Assingment 9 will be published on 10th July - deadline on 16th July 23:59
- The solutions will be published and discussed during tutorial session on 17th July .
- Please prepare your questions or list of topics you would like to have explained again and send me before 17th via e-mail.
- On 17th July we will vote and pick the most requested questions and topics.
- We will cover these questions and topics on the Q&A - exam preperation session (24th July)

Recap - Blocking communication

Circular communication, dead-lock free code

```
1  int main (int argc, char* argv[])
2  {
3    int rank, size, buf;
4
5    MPI_Init(&argc, &argv); /* starts MPI */
6    MPI_Comm_rank(MPI_COMM_WORLD, &rank); /* process id */
7    MPI_Comm_size(MPI_COMM_WORLD, &size); /* number processes */
8    buf = rank;
9
10   if (rank==0){
11     MPI_Recv(&buf, 1, MPI_INT, (rank+size-1)%size, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
12     MPI_Send(&buf, 1, MPI_INT, (rank+1)%size, 0, MPI_COMM_WORLD);
13   }
14   else{
15     MPI_Send(&buf, 1, MPI_INT, (rank+1)%size, 0, MPI_COMM_WORLD);
16     MPI_Recv(&buf, 1, MPI_INT, (rank+size-1)%size, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
17   }
18
19   MPI_Finalize();
20   return 0;
21 }
```

Circular communication (cont.)

```c
#include <stdio.h>
#include <mpi.h>

int main (int argc, char* argv[])
{
  int rank, size, buf;

  MPI_Init(&argc, &argv); /* starts MPI */
  MPI_Comm_rank(MPI_COMM_WORLD, &rank); /* process id */
  MPI_Comm_size(MPI_COMM_WORLD, &size); /* number processes */
  buf=rank;

  MPI_Sendrecv(&buf, 1, MPI_INT, (rank+1)%size, 0,
               &buf, 1, MPI_INT, (rank+size-1)%size, 0,
              MPI_COMM_WORLD, MPI_STATUS_IGNORE);

  MPI_Finalize();
  return 0;
}
```

Circular communication using MPI_Sendrecv_replace

```
1  int main (int argc, char* argv[])
2  {
3    int rank, size, buf;
4
5    MPI_Init(&argc, &argv); /* starts MPI */
6    MPI_Comm_rank(MPI_COMM_WORLD, &rank); /* process id */
7    MPI_Comm_size(MPI_COMM_WORLD, &size); /* number processes */
8    buf=rank;
9
10   MPI_Sendrecv_replace(&buf, 1, MPI_INT, (rank+1)%size, 0,
11                        (rank+size-1)%size, 0,
12                       MPI_COMM_WORLD, MPI_STATUS_IGNORE);
13
14   MPI_Finalize();
15   return 0;
16 }
```

Non-blocking communication

Circular communication using MPI_ISend/IRecv, Does this work?

```
1  int main (int argc, char* argv[])
2  {
3    int rank, size, buf;
4
5    MPI_Init(&argc, &argv); /* starts MPI */
6    MPI_Comm_rank(MPI_COMM_WORLD, &rank); /* process id */
7    MPI_Comm_size(MPI_COMM_WORLD, &size); /* number processes */
8    buf = rank;
9
10   MPI_Request req[2];
11
12   MPI_Isend(&buf, 1, MPI_INT, (rank+1)%size, 0, MPI_COMM_WORLD, &req[0]);
13   MPI_Irecv(&buf, 1, MPI_INT, (rank+size-1)%size, 0, MPI_COMM_WORLD, &req[1]);
14
15   MPI_Finalize();
16   return 0;
17 }
```

Circular communication with MPI_Waitall, Does this work?

```
1  int main (int argc, char* argv[])
2  {
3    int rank, size, buf;
4
5    MPI_Init(&argc, &argv); /* starts MPI */
6    MPI_Comm_rank(MPI_COMM_WORLD, &rank); /* process id */
7    MPI_Comm_size(MPI_COMM_WORLD, &size); /* number processes */
8    buf = rank;
9
10   MPI_Request req[2];
11
12   MPI_Isend(&buf, 1, MPI_INT, (rank+1)%size, 0, MPI_COMM_WORLD, &req[0]);
13   MPI_Irecv(&buf, 1, MPI_INT, (rank+size-1)%size, 0, MPI_COMM_WORLD, &req[1]);
14
15   MPI_Waitall(2, req, MPI_STATUS_IGNORE);
16
17   MPI_Finalize();
18   return 0;
19 }
```

MPI - Collectives

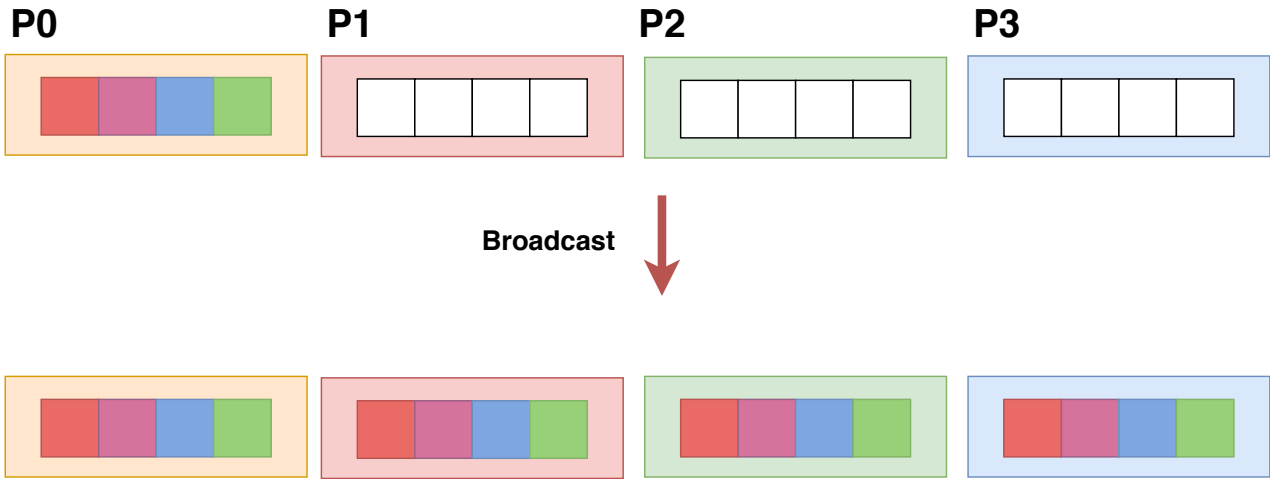
Collective operations

- Operations that are executed by all the processes in a communicator
- Types:
  - Synchronization
    - Barrier
  - Communication
    - Broadcast
    - Scatter
    - Gather
  - Reduction
    - Combine variables from different processes
- Help us in the implementation as they provide primitives for typical communication patterns

MPI_Bcast

```
1  int main(int argc, char **argv)
2  {
3      int rank, size;
4
5      MPI_Init(&argc, &argv);
6      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7      MPI_Comm_size(MPI_COMM_WORLD, &size);
8
9      int data[4];
10     if (rank == 0) {data[0] = 0; data[1] = 1; data[2] = 2; data[3] = 3;}
11     else {data[0] = 0; data[1] = 0; data[2] = 0; data[3] = 0;}
12
13     MPI_Bcast(data, 4, MPI_INT, 0, MPI_COMM_WORLD);
14
15     MPI_Finalize();
16 }
```

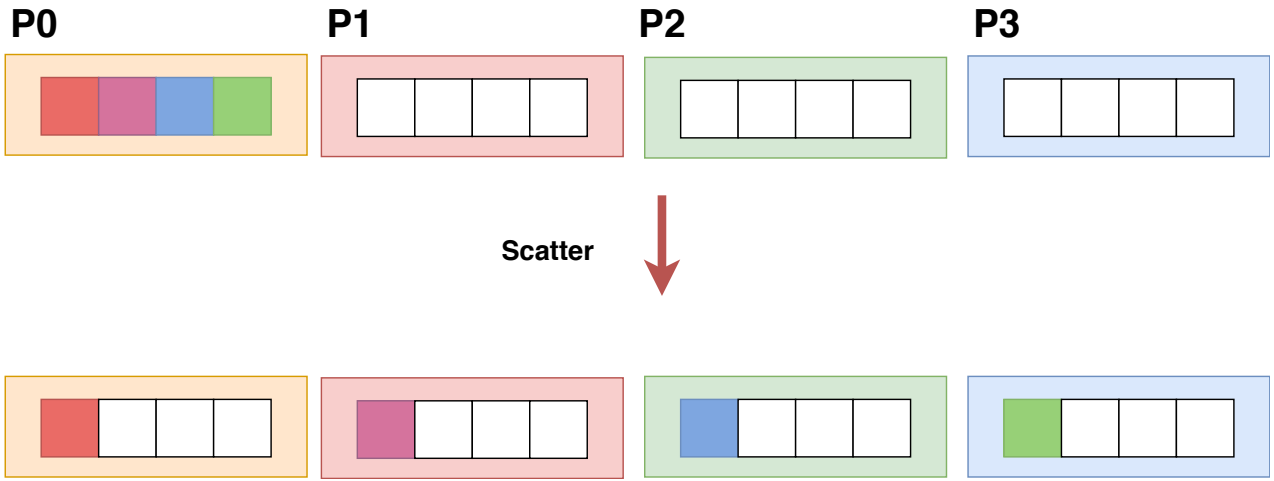
Broadcast, one to all



MPI_Scatter

```
1  int main(int argc, char **argv)
2  {
3      int rank, size;
4
5      MPI_Init(&argc, &argv);
6      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7      MPI_Comm_size(MPI_COMM_WORLD, &size);
8
9      int data[4];
10     if (rank == 0) {data[0] = 0; data[1] = 1; data[2] = 2; data[3] = 3;}
11     else {data[0] = 0; data[1] = 0; data[2] = 0; data[3] = 0;}
12
13     MPI_Scatter(data, 1, MPI_INT, data, 1, MPI_INT, 0, MPI_COMM_WORLD);
14
15     MPI_Finalize();
16 }
```

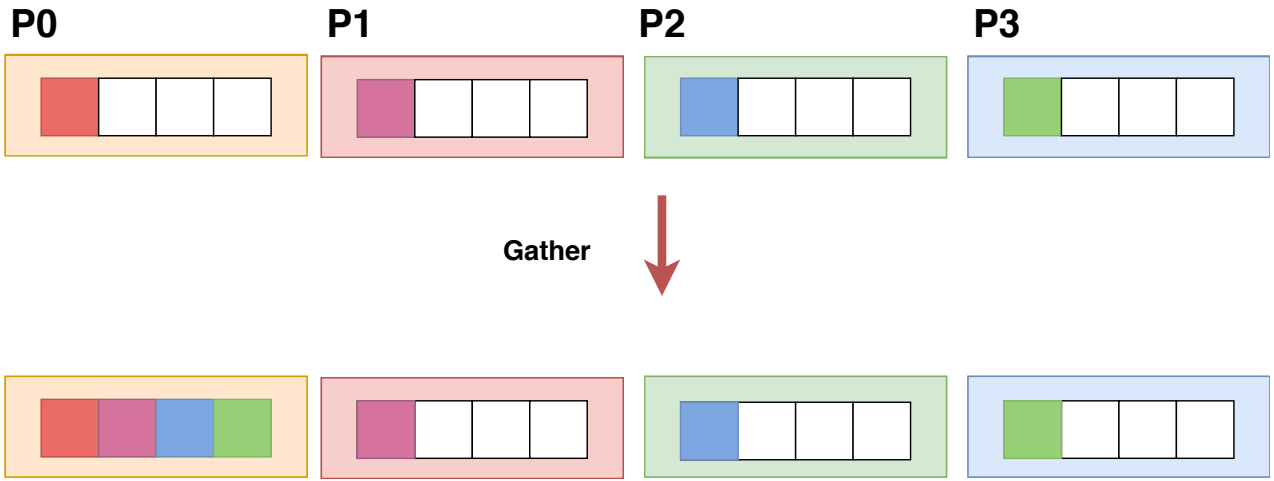
Scatter, one to all



MPI_Gather

```
1  int main(int argc, char **argv)
2  {
3      int rank, size;
4
5      MPI_Init(&argc, &argv);
6      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7      MPI_Comm_size(MPI_COMM_WORLD, &size);
8
9      int data[4];
10     data[0] = rank; data[1] = 0; data[2] = 0; data[3] = 0;
11
12     MPI_Gather(data, 1, MPI_INT, data, 1, MPI_INT, 0, MPI_COMM_WORLD);
13
14     MPI_Finalize();
15 }
```

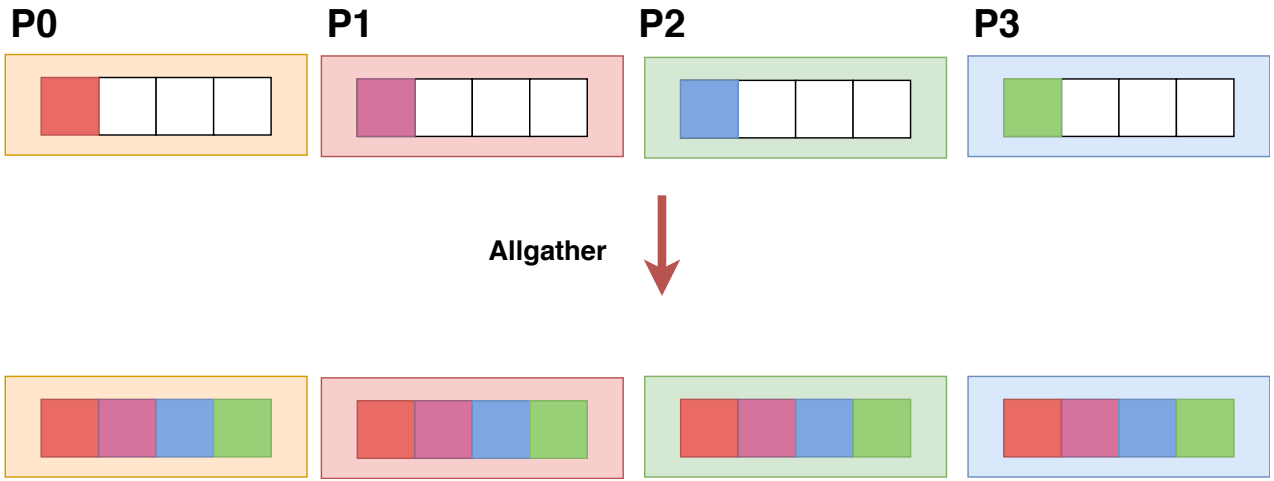
Gather, all to one



MPI_Allgather

```
1  int main(int argc, char **argv)
2  {
3      int rank, size;
4
5      MPI_Init(&argc, &argv);
6      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7      MPI_Comm_size(MPI_COMM_WORLD, &size);
8
9      int data[4];
10     data[0] = rank; data[1] = 0; data[2] = 0; data[3] = 0;
11
12     MPI_Allgather(data, 1, MPI_INT, data, 1, MPI_INT, MPI_COMM_WORLD);
13
14     MPI_Finalize();
15 }
```

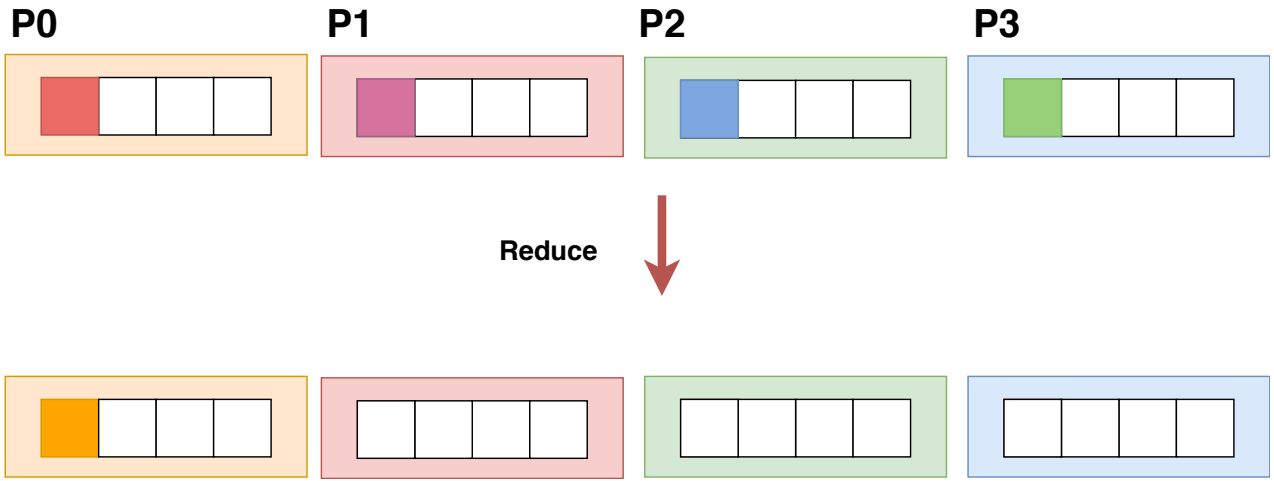
Allgather, all to all



MPI_Reduce

```
1  int main(int argc, char **argv)
2  {
3      int rank, size;
4
5      MPI_Init(&argc, &argv);
6      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7      MPI_Comm_size(MPI_COMM_WORLD, &size);
8
9      int local_data=1, global_data=0;
10
11     MPI_Reduce(&local_data, &global_data, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
12
13     MPI_Finalize();
14 }
```

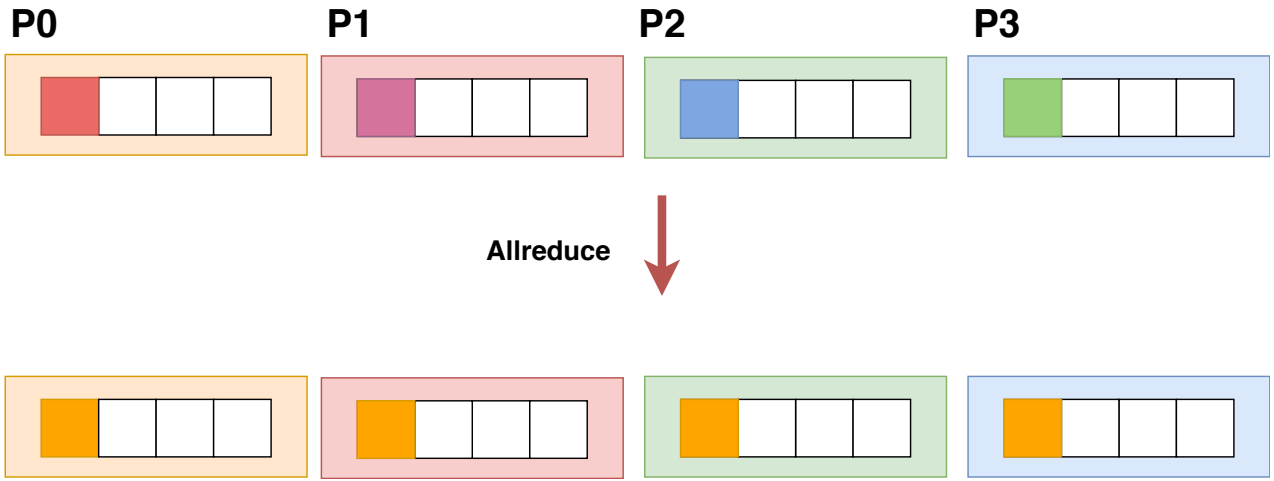
Reduce, all to one



MPI_Allreduce

```
1  int main(int argc, char **argv)
2  {
3      int rank, size;
4
5      MPI_Init(&argc, &argv);
6      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7      MPI_Comm_size(MPI_COMM_WORLD, &size);
8
9      int local_data=1, global_data=0;
10
11     MPI_Allreduce(&local_data, &global_data, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
12
13     MPI_Finalize();
14 }
```

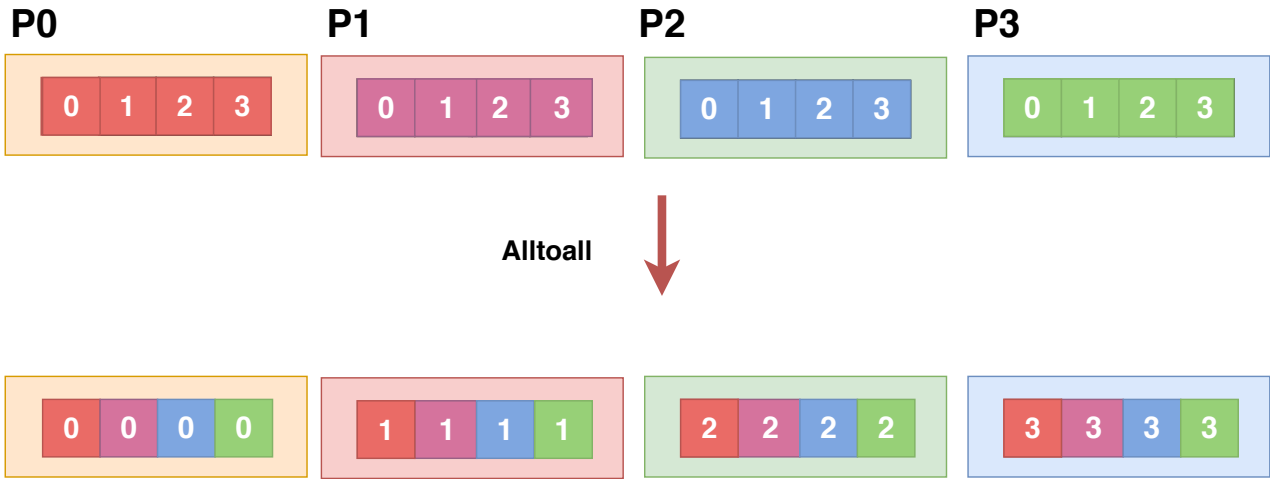
Allreduce, all to all



MPI_Alltoall

```
1  int main(int argc, char **argv)
2  {
3      int rank, size;
4
5      MPI_Init(&argc, &argv);
6      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7      MPI_Comm_size(MPI_COMM_WORLD, &size);
8
9      int send_data[4] = {0,1,2,3};
10     int recv_data[4] = {0,0,0,0};
11
12     MPI_Alltoall(send_data, 1, MPI_INT, recv_data, 1, MPI_INT, MPI_COMM_WORLD);
13
14     MPI_Finalize();
15 }
```

Alltoall, all to all



Scatterv, one to all

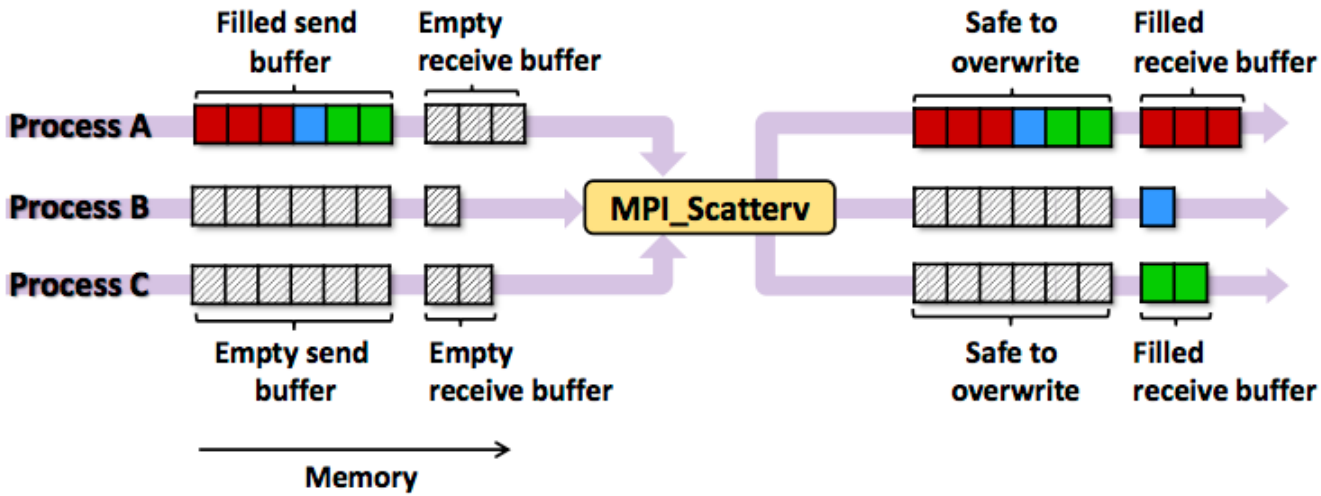


Abbildung: from SKIRT Docs

Gatherv, all to one

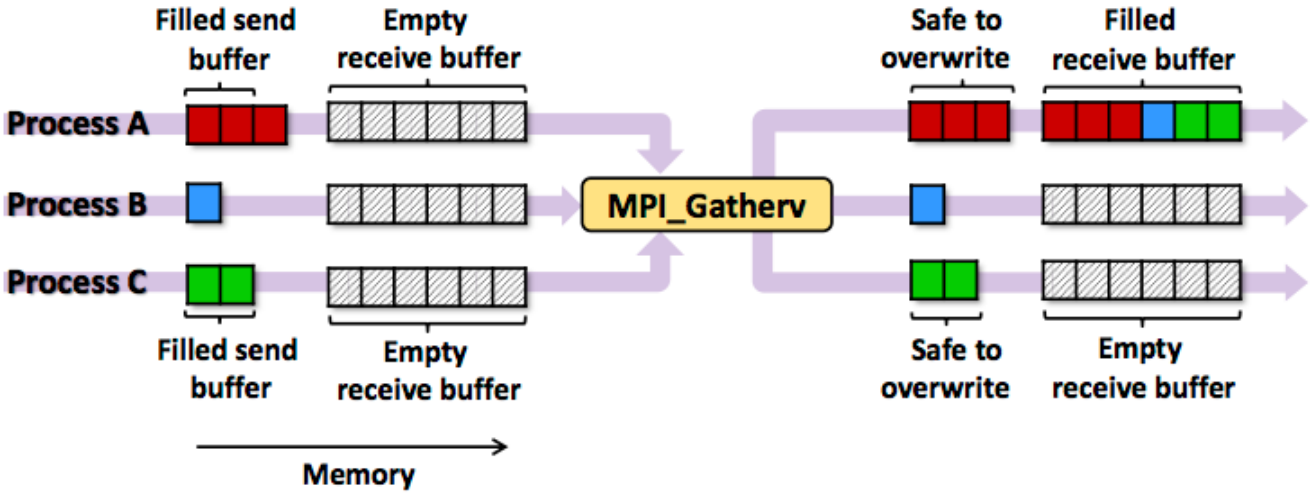


Abbildung: from SKIRT Docs

Assignment 8

Assignment 8

- Search for words in a large text file.
- Words are separated by spaces, semicolons so on.
- Use MPI_Send and MPI_Recv to parallelize the search.
- You should be able to achieve a minimum speedup of 5 using 16 processes.
- The server has 2 NUMA nodes.

Assignment 8

- Makefile
  - Makefile contains rules to build executables
  - available targets: parallel, sequential, unit_test, all (default), clean
  - 'mode=debug make [target]' to build debug version, use 'make clean' before

- Executable requires two extra parameters: a text file and a search string
- Run your code with

  `mpirun -np <num processes> <program name> <path to text file> <search string>`
- Run unit text with

  `mpirun -np <num processes> <unit test>`
- Unit test requires that the file treasure_island.txt be present in assignment directory.

Assignment 8

- WARNING: If you're running your code on lxhalle machines, all your processes might have rank 0.
- Make sure you are compiling and running your code with compatible implementations of mpicc and mpirun / mpiexec (OpenMPI / MPICH).
- For more details, see
  https://wiki.mpich.org/mpich/index.php/Frequently_Asked_Questions#Q:_All_my_processes_get_rank_0.