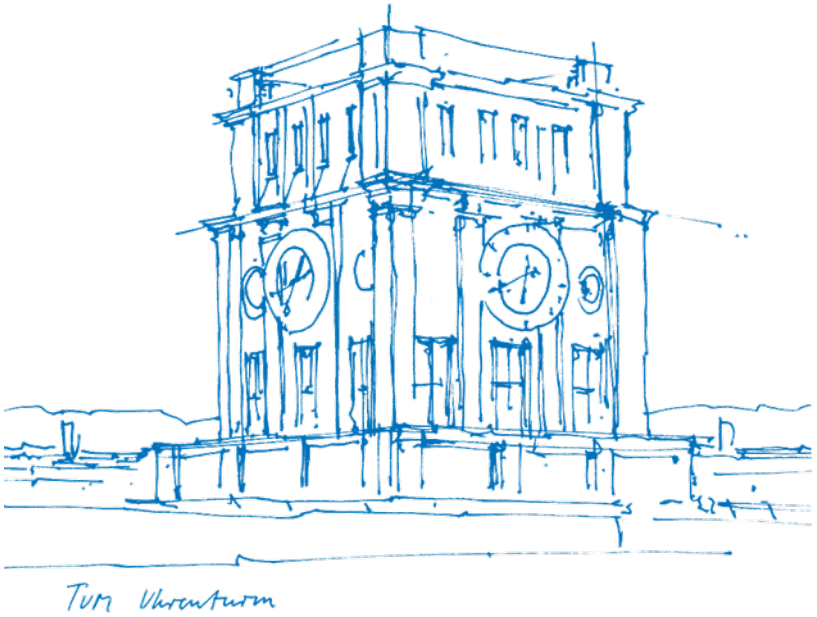
Parallel Programming Tutorial - Advanced MPI

M.Sc. Amir Raoofy
Technical University of Munich
11. Juni 2018



Organization

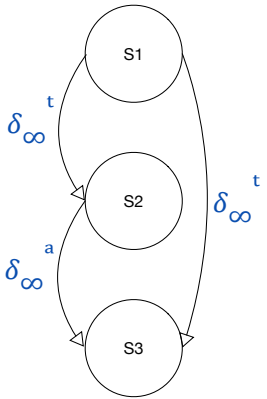
Organization

- The deadline for reverse_str is extended to tomorrow night. Please go to the Q/As tomorrow.
- The speedup limit is also relaxed to 8.0.
- The solution will be published and discussed next week on Wednesday.
- Next assignment will be published tonight.

Solution for Assignment 8

Solution for Loop Fusion

```
for (int i = 1; i < N; i++) {
   for (int j = 1; j < N; j++) {
S1:    a[i][j] = 2 * b[i][j];
S2:    d[i][j] = a[i][j] * c[i][j];
   }
}
for (int j = 1; j < N; j++) {
   for (int i = 1; i < N; i++) {
S3:    c[i][j - 1] = a[i][j - 1] - a[i][j + 1];
   }
}
```

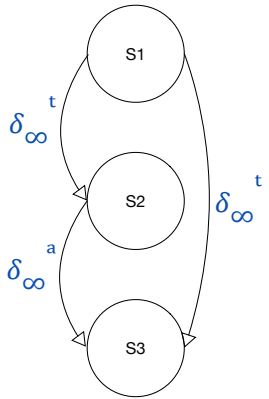


Solution

```
#pragma omp parallel for num_threads(num_threads)
  for (int i = 1; i < N; i++)
  {
    for (int j = 1; j < N; j++)
    {
      a[i][j] = 2 * b[i][j];
      d[i][j] = a[i][j] * c[i][j];
    }

    for (int j = 1; j < N; j++)
    {
      c[i][j - 1] = a[i][j - 1] - a[i][j + 1];
    }
  }
```

Another solution for Loop Fusion

```
for (int i = 1; i < N; i++) {
   for (int j = 1; j < N; j++) {
S1:    a[i][j] = 2 * b[i][j];
S2:    d[i][j] = a[i][j] * c[i][j];
   }
}
for (int j = 1; j < N; j++) {
   for (int i = 1; i < N; i++) {
S3:    c[i][j - 1] = a[i][j - 1] - a[i][j + 1];
   }
}
```



Solution

```
#pragma omp parallel for num_threads(num_threads)
for (int i = 1; i < N; i++) {
  int j = 1;
  a[i][j] = 2 * b[i][j];
  d[i][j] = a[i][j] * c[i][j];
  c[i][j - 1] = a[i][j - 1] - 2 * b[i][j + 1];

  for (j = 2; j < N - 1; j++) {
     a[i][j] = 2 * b[i][j];
     d[i][j] = a[i][j] * c[i][j];
     c[i][j - 1] = 2 * b[i][j - 1] - 2 * b[i][j + 1];
  }

  j = N - 1;
  a[i][j] = 2 * b[i][j];
  d[i][j] = a[i][j] * c[i][j];
  c[i][j - 1] =  2 * b[i][j - 1] - a[i][j + 1];
}
```

Blocking communication

Circular communication, dead-lock free code

```
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;

  if (rank==0){
    MPI_Recv(&buf, 1, MPI_INT, (rank+size-1)%size, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    MPI_Send(&buf, 1, MPI_INT, (rank+1)%size, 0, MPI_COMM_WORLD);
  }
  else{
    MPI_Send(&buf, 1, MPI_INT, (rank+1)%size, 0, MPI_COMM_WORLD);
    MPI_Recv(&buf, 1, MPI_INT, (rank+size-1)%size, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  }

  MPI_Finalize();
  return 0;
}
```

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

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

  MPI_Finalize();
  return 0;
}
```

Non-blocking communication

Circular communication using MPI_ISend/IRecv, Does this work?

```
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_Request req[2];

  MPI_Isend(&buf, 1, MPI_INT, (rank+1)%size, 0, MPI_COMM_WORLD, &req[0]);
  MPI_Irecv(&buf, 1, MPI_INT, (rank+size-1)%size, 0, MPI_COMM_WORLD, &req[1]);

  MPI_Finalize();
  return 0;
}
```

Circular communication with MPI_Waitall, Does this work?

```
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_Request req[2];

  MPI_Isend(&buf, 1, MPI_INT, (rank+1)%size, 0, MPI_COMM_WORLD, &req[0]);
  MPI_Irecv(&buf, 1, MPI_INT, (rank+size-1)%size, 0, MPI_COMM_WORLD, &req[1]);

  MPI_Waitall(2, &req, MPI_STATUS_IGNORE);

  MPI_Finalize();
  return 0;
}
```

Circular communication, non-blocking version

```
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_Request req[2];

  MPI_Isend(&rank, 1, MPI_INT, (rank+1)%size, 0, MPI_COMM_WORLD, &req[0]);
  MPI_Irecv(&buf, 1, MPI_INT, (rank+size-1)%size, 0, MPI_COMM_WORLD, &req[1]);

  MPI_Waitall(2, &req, MPI_STATUS_IGNORE);

  MPI_Finalize();
  return 0;
}
```

MPI collectives

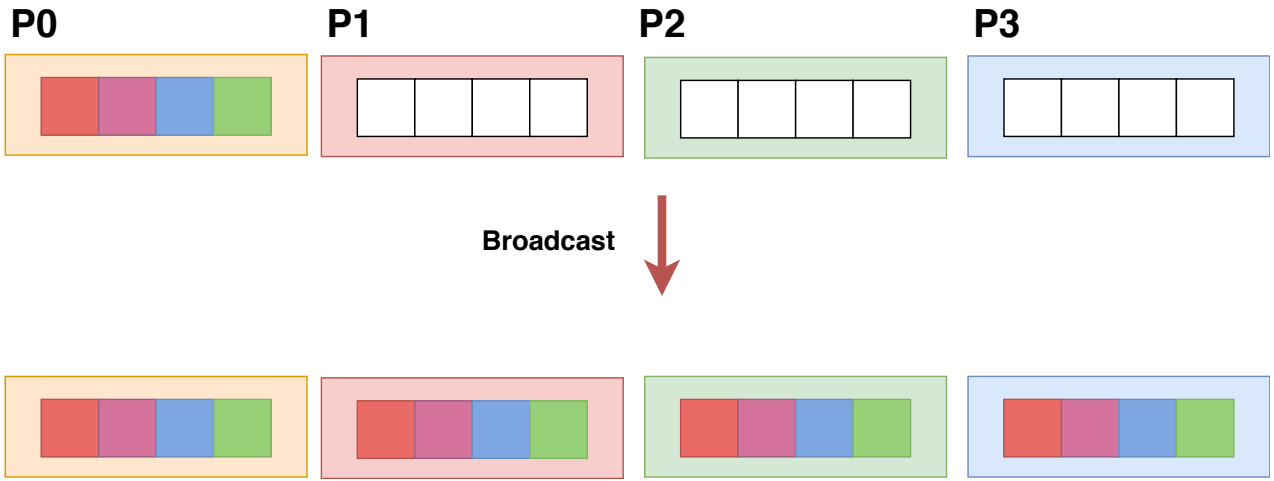
Collective operations

- 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

```
int main(int argc, char **argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int data[4];
    if (rank == 0) {data[0] = 0; data[1] = 1; data[2] = 2; data[3] = 3;}
    else {data[0] = 0; data[1] = 0; data[2] = 0; data[3] = 0;}

    MPI_Bcast(data, 4, MPI_INT, 0, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

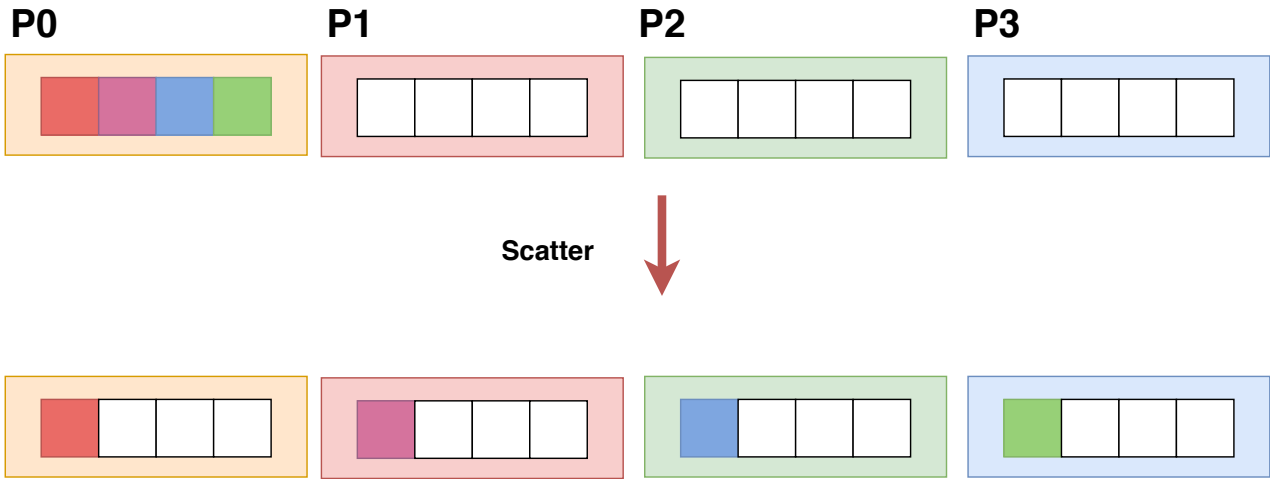
Broadcast, one to all



MPI_Scatter

```
int main(int argc, char **argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int data[4];
    if (rank == 0) {data[0] = 0; data[1] = 1; data[2] = 2; data[3] = 3;}
    else {data[0] = 0; data[1] = 0; data[2] = 0; data[3] = 0;}

    MPI_Scatter(data, 1, MPI_INT, data, 1, MPI_INT, 0, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

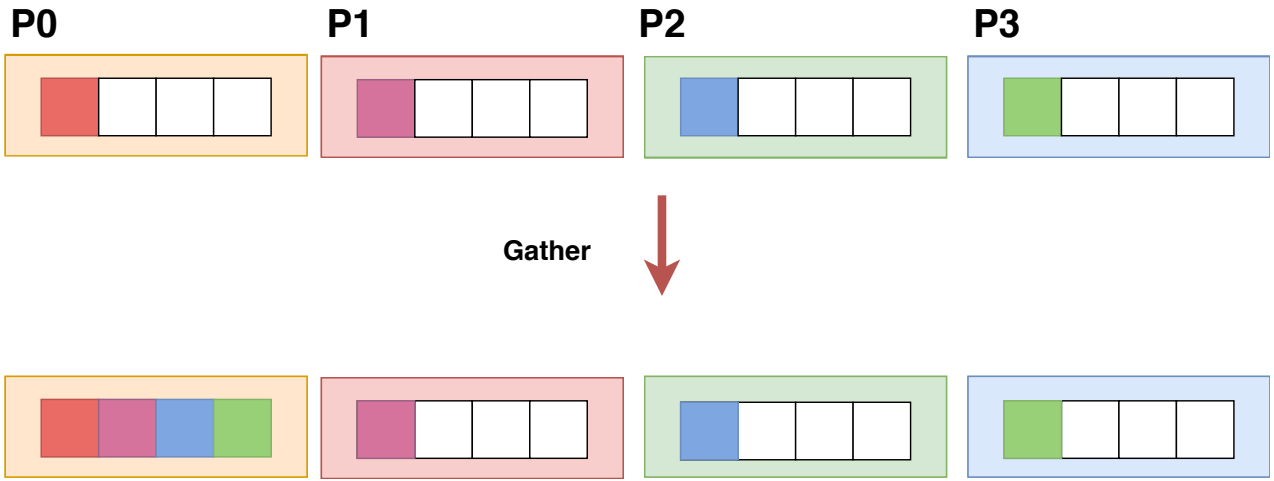
Scatter, one to all



MPI_Gather

```
int main(int argc, char **argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int data[4];
    data[0] = rank; data[1] = 0; data[2] = 0; data[3] = 0;

    MPI_Gather(data, 1, MPI_INT, data, 1, MPI_INT, 0, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

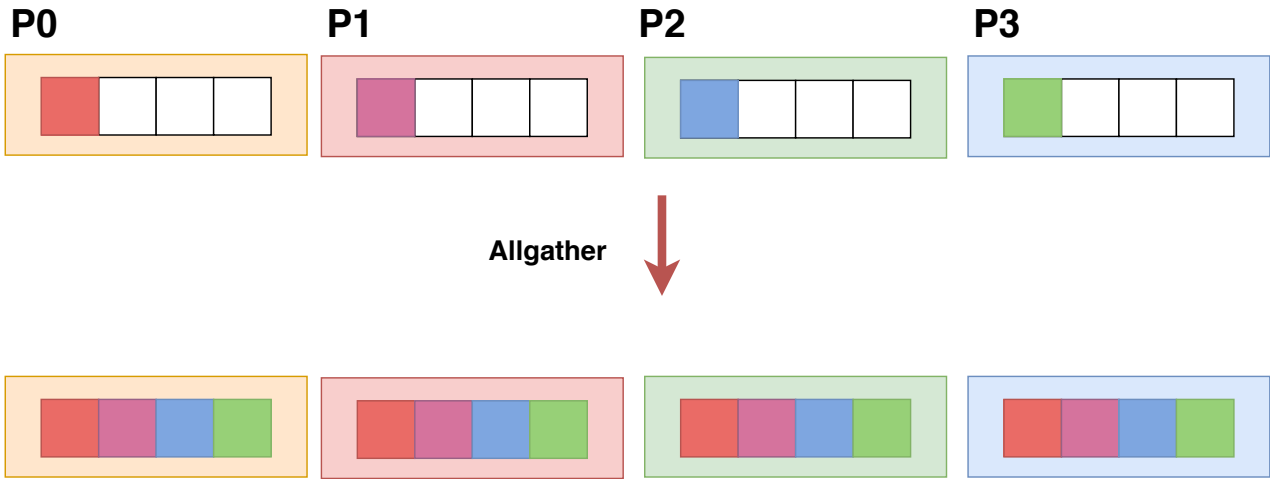
Gather, all to one



MPI_Allgather

```
int main(int argc, char **argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int data[4];
    data[0] = rank; data[1] = 0; data[2] = 0; data[3] = 0;

    MPI_Allgather(data, 1, MPI_INT, data, 1, MPI_INT, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

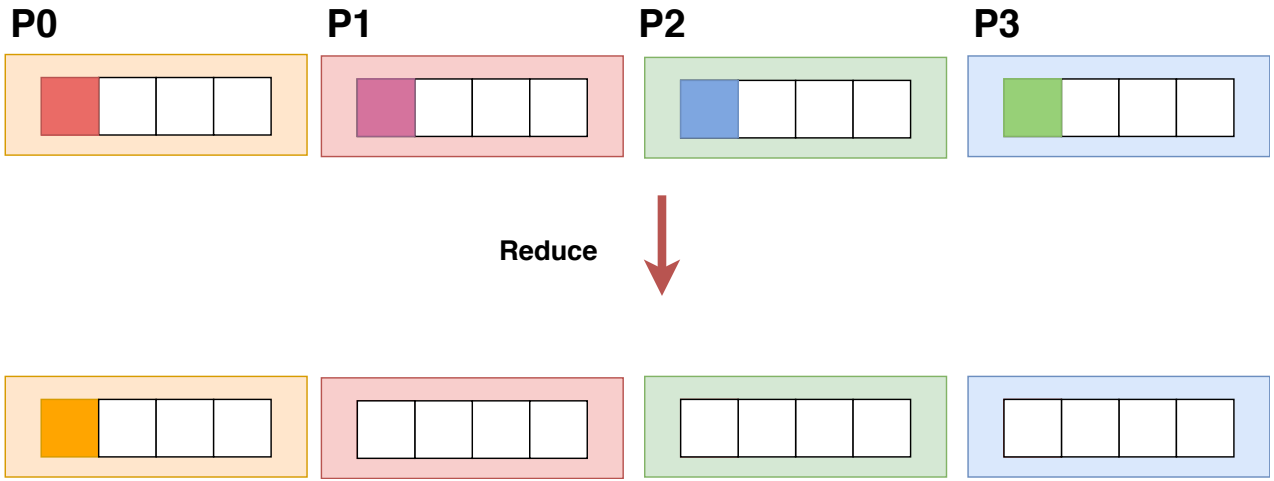
Allgather, all to all



MPI_Reduce

```
int main(int argc, char **argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int local_data=1, global_data=0;

    MPI_Reduce(&local_data, &global_data, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

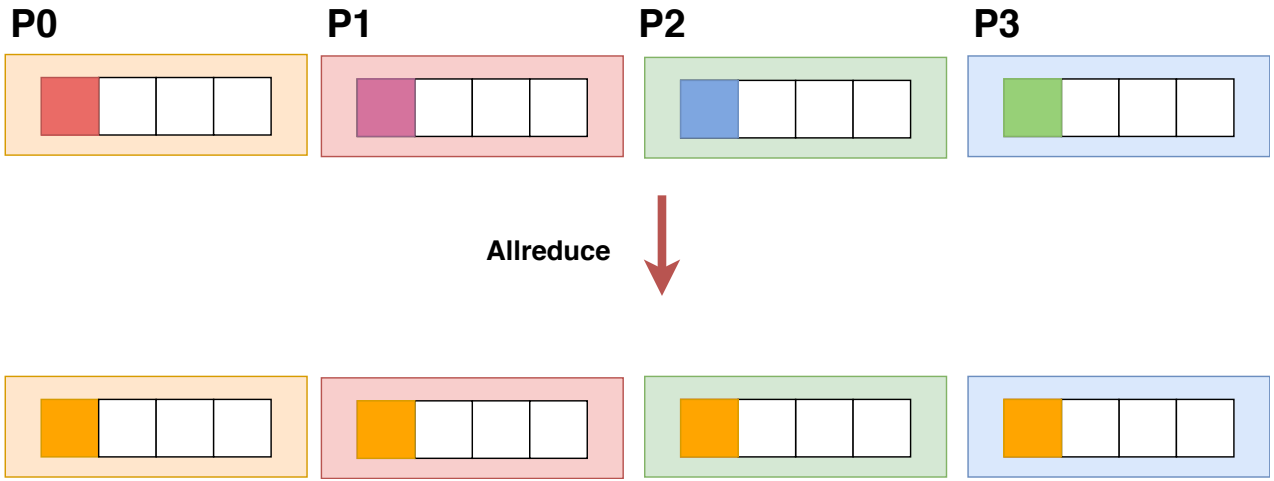
Reduce, all to one



MPI_Allreduce

```
int main(int argc, char **argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int local_data=1, global_data=0;

    MPI_Allreduce(&local_data, &global_data, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

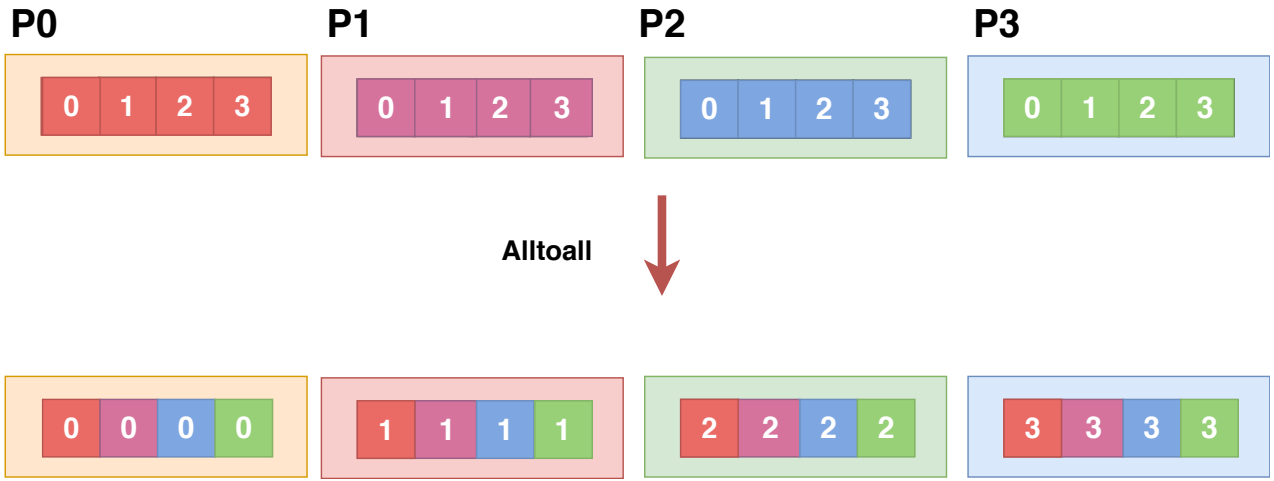
Allreduce, all to all



MPI_Alltoall

```
int main(int argc, char **argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int send_data[4] = {0,1,2,3};
    int recv_data[4] = {0,0,0,0};

    MPI_Alltoall(send_data, 1, MPI_INT, recv_data, 1, MPI_INT, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

Alltoall, all to all



Scatterv, one to all

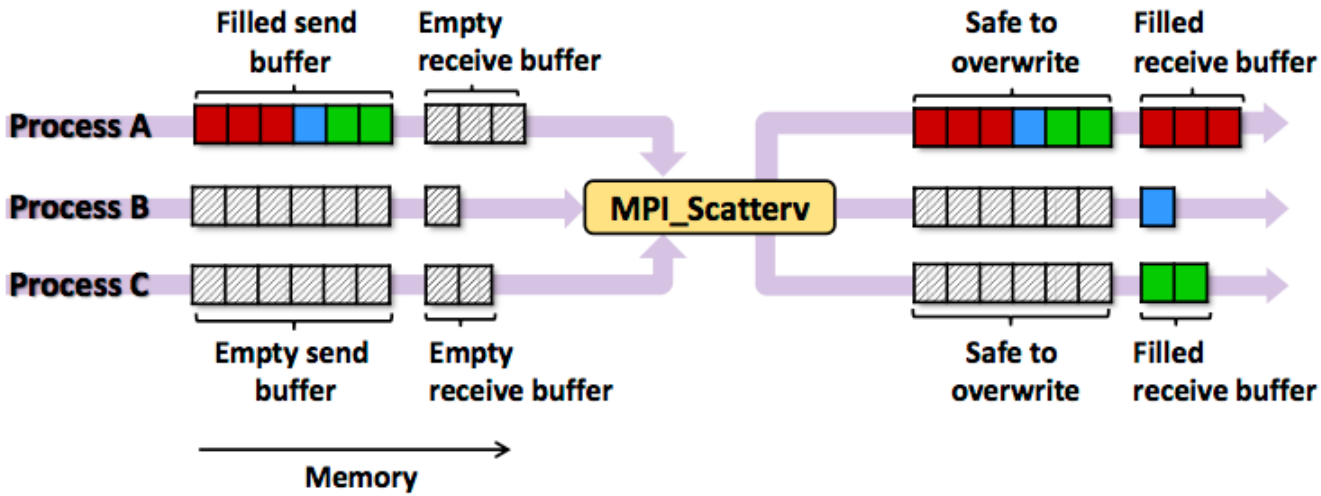


Abbildung: from SKIRT Docs

Gatherv, all to one

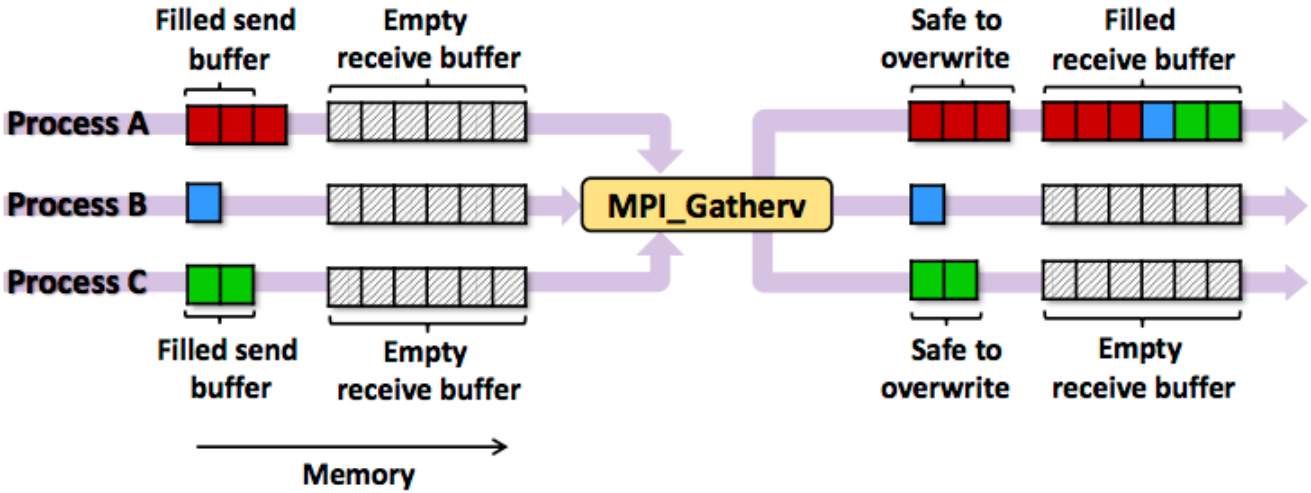


Abbildung: from SKIRT Docs