University at Buffalo The State University of New York

SCoRe group

Introduction to Parallel and Distributed Processing

MPI Collectives

Jaroslaw 'Jaric' Zola

http://www.jzola.org/

Collective Communication

- Exchange messages simultaneously between all processors
- Collectives always involve all ranks in a communicator
- Guaranteed not to interfere with point to point messages
- Highly optimized to leverage hardware properties

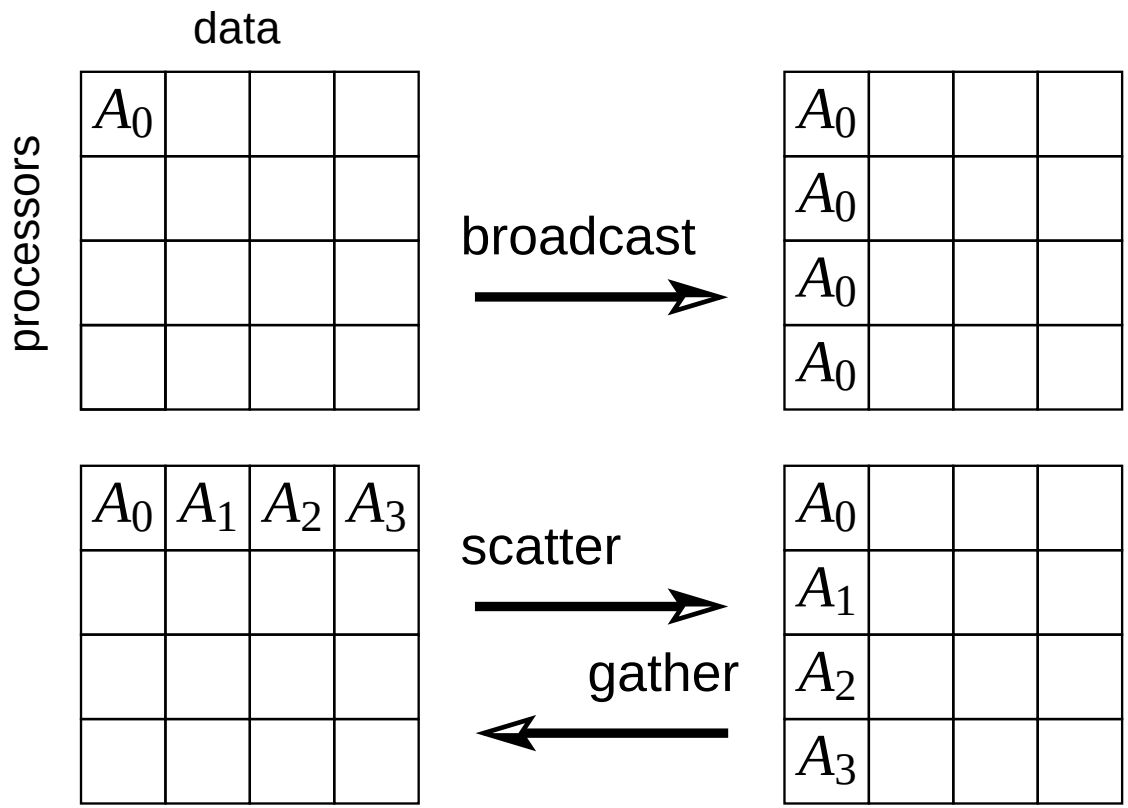
Barrier

- Simplest form of “communication” – caller blocked until all processors called

```
1  #include <iostream>
2  #include <mpi.h>
3
4  int main(int argc, char* argv[]) {
5    MPI_Init(&argc, &argv);
6    // ...
7
8    MPI_Barrier(MPI_COMM_WORLD);
9    double ts = MPI_Wtime();
10   // ...
11   MPI_Barrier(MPI_COMM_WORLD);
12   double tf = MPI_Wtime();
13
14   std::cout << (tf – ts) << std::endl;
15   return MPI_Finalize();
16 } // main
```

Broadcast/Scatter/Gather

- One to many, many to one communication pattern



Broadcast

```
1  #include <vector>
2  #include <mpi.h>
3
4  int main(int argc, char* argv[]) {
5      int rank;
6      MPI_Init(&argc, &argv);
7      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8
9      std::vector<int> x(4);
10     if (rank == 0) for (auto& v : x) std::cin >> v;
11
12     MPI_Bcast(x.data(), x.size(), MPI_INT, 0, MPI_COMM_WORLD);
13
14     return MPI_Finalize();
15 }
```

Cost of Broadcast

- At the algorithmic level, irrespective of network architecture:

$$T = (\tau + \mu m)\log(p) = O(\log(p))$$

  where τ latency (or more general startup time) and $\frac{1}{\mu}$ is network bandwidth, and m is message size

- In practice hardware may provide dedicated solution, network multicasts can be employed, etc.

Scatter

```
1  #include <vector>
2  #include <mpi.h>
3
4  int main(int argc, char* argv[]) {
5    int rank, size;
6    MPI_Init(&argc, &argv);
7    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8    MPI_Comm_size(MPI_COMM_WORLD, &size);
9
10   int x;
11   std::vector<int> buf(size);
12   if (rank == 0) for (auto& b : buf) b = rand();
13
14   MPI_Scatter(buf.data(), 1, MPI_INT,
15               &x, 1, MPI_INT, 0, MPI_COMM_WORLD);
16
17   return MPI_Finalize();
18 } // main
```

Cost of Scatter

- Story almost the same as broadcast

$$T = \tau \log(p) + \mu m(p-1)$$

Gather

```
1  #include <vector>
2  #include <mpi.h>
3
4  int main(int argc, char* argv[]) {
5    int rank, size;
6    MPI_Init(&argc, &argv);
7    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8    MPI_Comm_size(MPI_COMM_WORLD, &size);
9
10   if (rank == 0) {
11     std::vector<int> buf(size);
12     MPI_Gather(&rank, 1, MPI_INT,
13                buf.data(), 1, MPI_INT, 0, MPI_COMM_WORLD);
14   } else {
15     MPI_Gather(&rank, 1, MPI_INT,
16                0, 1, MPI_INT, 0, MPI_COMM_WORLD);
17   }
18
19   return MPI_Finalize();
20 } // main
```

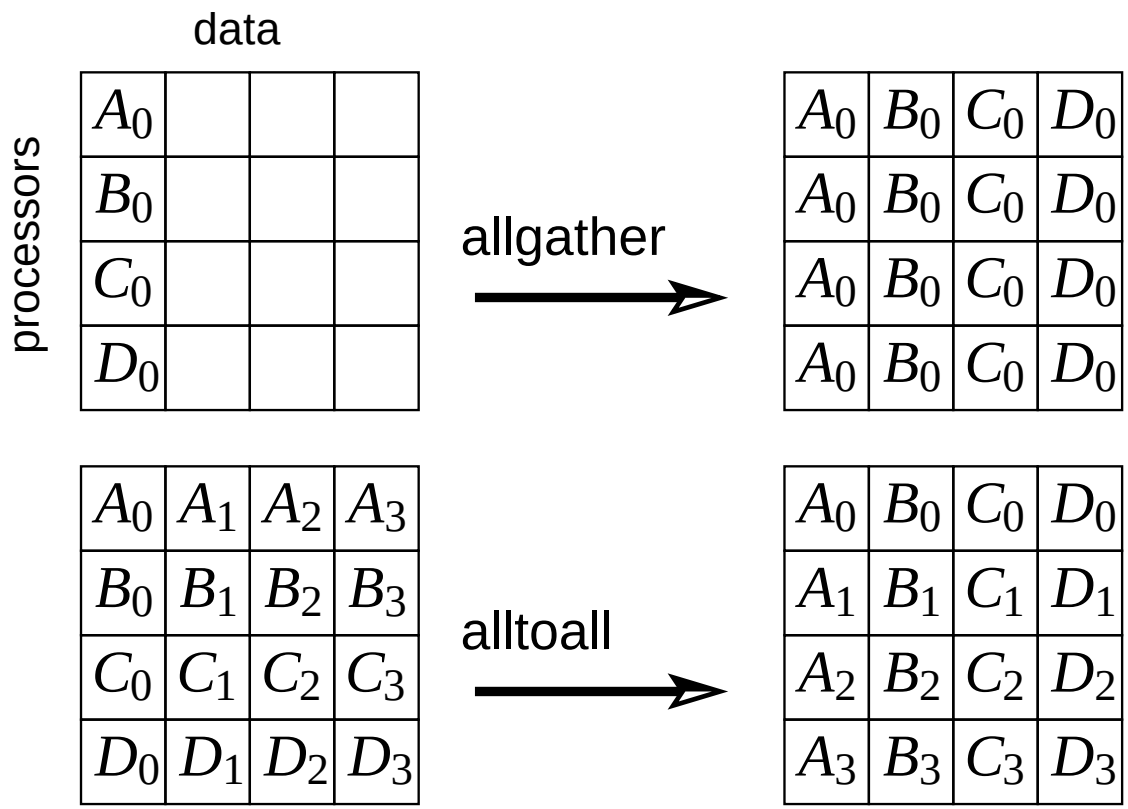
Reduce/Scan

- Two magic primitives, recall the need for associative operator
- MPI provides set of predefined ops (e.g. `MPI_SUM`)
- New operators can be defined via `MPI_Op_create`

```
1  #include <mpi.h>
2
3  int main(int argc, char* argv[]) {
4    int rank;
5    MPI_Init(&argc, &argv);
6    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7
8    int x[2] = {rank, rank + 1};
9    int s[2];
10
11   MPI_Scan(x, s, 2, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
12
13   return MPI_Finalize();
14 } // main
```

Allgather/Alltoall

- Everybody has a message for everybody



Allgather

```
1  #include <vector>
2  #include <mpi.h>
3
4  int main(int argc, char* argv[]) {
5    int rank, size;
6    MPI_Init(&argc, &argv);
7    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8    MPI_Comm_size(MPI_COMM_WORLD, &size);
9
10   std::vector<int> buf(size);
11   MPI_Allgather(&rank, 1, MPI_INT,
12                 buf.data(), 1, MPI_INT, 0, MPI_COMM_WORLD);
13
14   return MPI_Finalize();
15 } // main
```

Alltoall

```
1  #include <vector>
2  #include <mpi.h>
3
4  int main(int argc, char* argv[]) {
5    int rank, size;
6    MPI_Init(&argc, &argv);
7    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
8    MPI_Comm_size(MPI_COMM_WORLD, &size);
9
10   std::vector<int> send(size);
11   std::vector<int> recv(size);
12   for (int i = 0; i < size; ++i) send[i] = rank;
13
14   MPI_Alltoall(send.data(), 1, MPI_INT,
15                recv.data(), 1, MPI_INT, MPI_COMM_WORLD);
16
17   return MPI_Finalize();
18 } // main
```

Cost of Alltoall

- Depends on architecture, but overall always the most expensive operation
- Avoid if possible

For Fun

- Assess latency and bandwidth of your interconnect as experienced by MPI.