CS 5220

Distributed memory

David Bindel

2024-09-26

MPI programming

Ideas from last time

- MPI = Message Passing Interface
- Single Program Multiple Data
  - Decide what to do based on *rank*
- `Send` and `Recv` are point-to-point primitives

Another approach

- Send/recv is one-to-one communication
- An alternative is one-to-many (and vice-versa):
  - *Broadcast* to distribute data from one process
  - *Reduce* to combine data from all processors
  - Operations are called by all processes in communicator

Broadcast and reduce

```
MPI_Bcast(buffer, count, datatype,
          root, comm);
MPI_Reduce(sendbuf, recvbuf, count, datatype,
           op, root, comm);
```

- buffer is copied from root to others
- recvbuf receives result only at root
- op $\in \{$ `MPI_MAX`, `MPI_SUM`, ...$\}$

Example: basic Monte Carlo

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char** argv) {
    int nproc, myid, ntrials = atoi(argv[1]);
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
    MPI_Bcast(&ntrials, 1, MPI_INT,
              0, MPI_COMM_WORLD);
    run_mc(myid, nproc, ntrials);
    MPI_Finalize();
    return 0;
}
```

Example: basic Monte Carlo

Let `sum[0]` = $\sum_i X_i$ and `sum[1]` = $\sum_i X_i^2$.

```
void run_mc(int myid, int nproc, int ntrials) {
    double sums[2] = {0,0};
    double my_sums[2] = {0,0};
    /* ... run ntrials local experiments ... */
    MPI_Reduce(my_sums, sums, 2, MPI_DOUBLE,
               MPI_SUM, 0, MPI_COMM_WORLD);
    if (myid == 0) {
        int N = nproc*ntrials;
        double EX = sums[0]/N;
        double EX2 = sums[1]/N;
        printf("Mean: %g; err: %g\n",
               EX, sqrt((EX*EX-EX2)/N));
    }
}
```

Collective operations

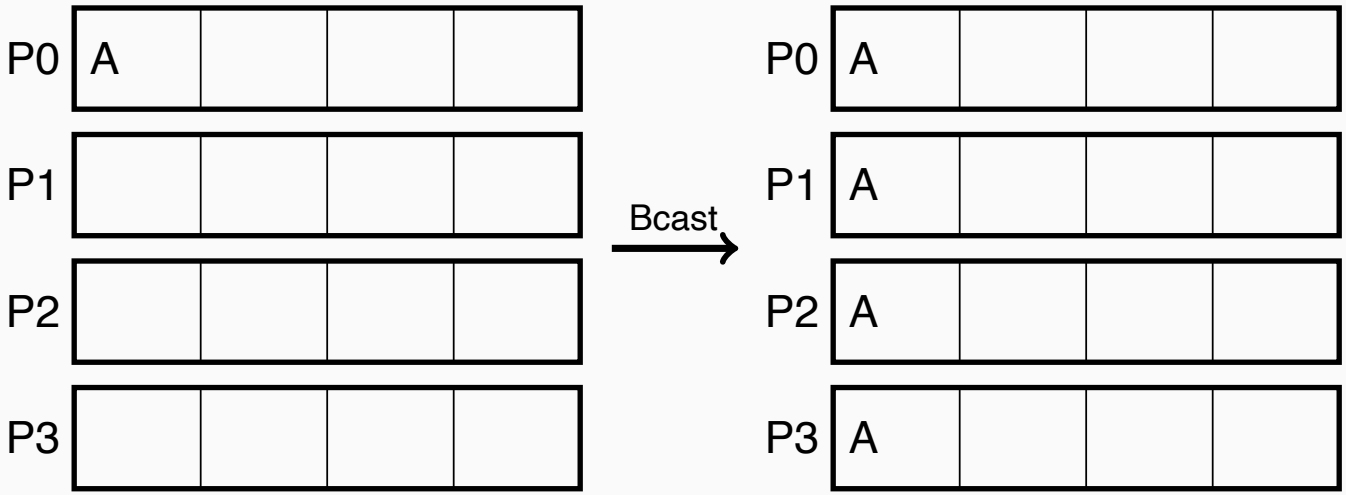
- Involve all processes in communicator
- Basic classes:
  - Synchronization (e.g. barrier)
  - Data movement (e.g. broadcast)
  - Computation (e.g. reduce)

Barrier

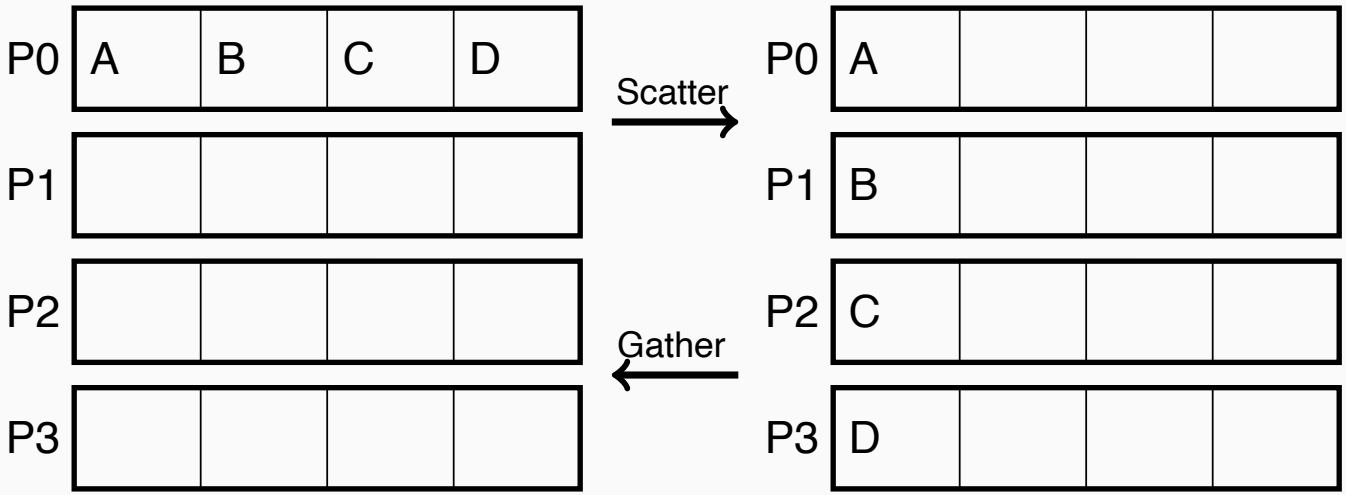
```
MPI_Barrier(comm);
```

Not much more to say. Not needed that often.

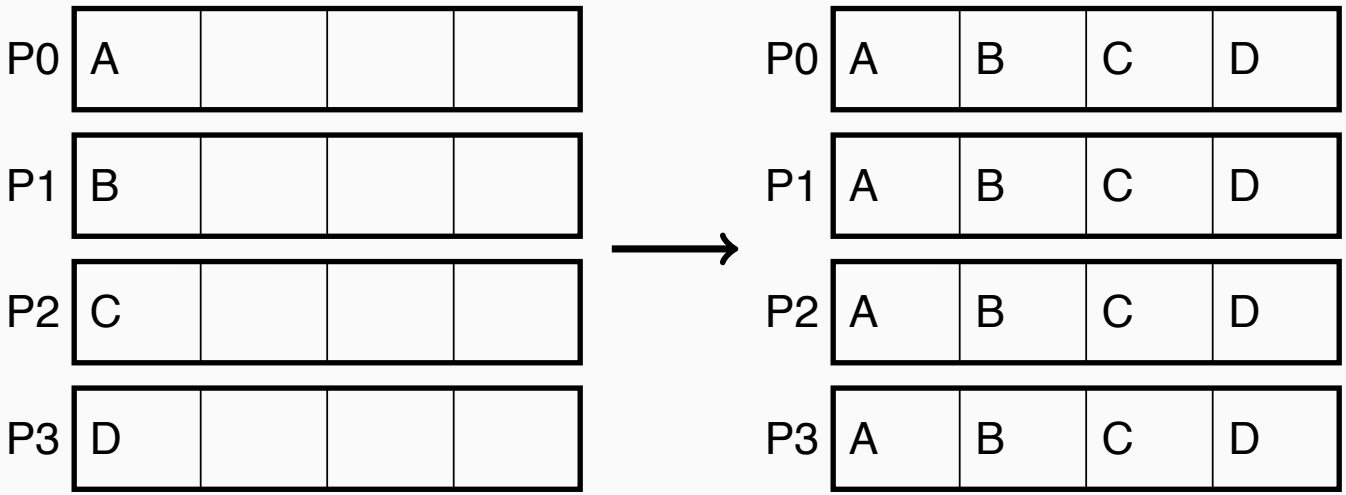
Broadcast



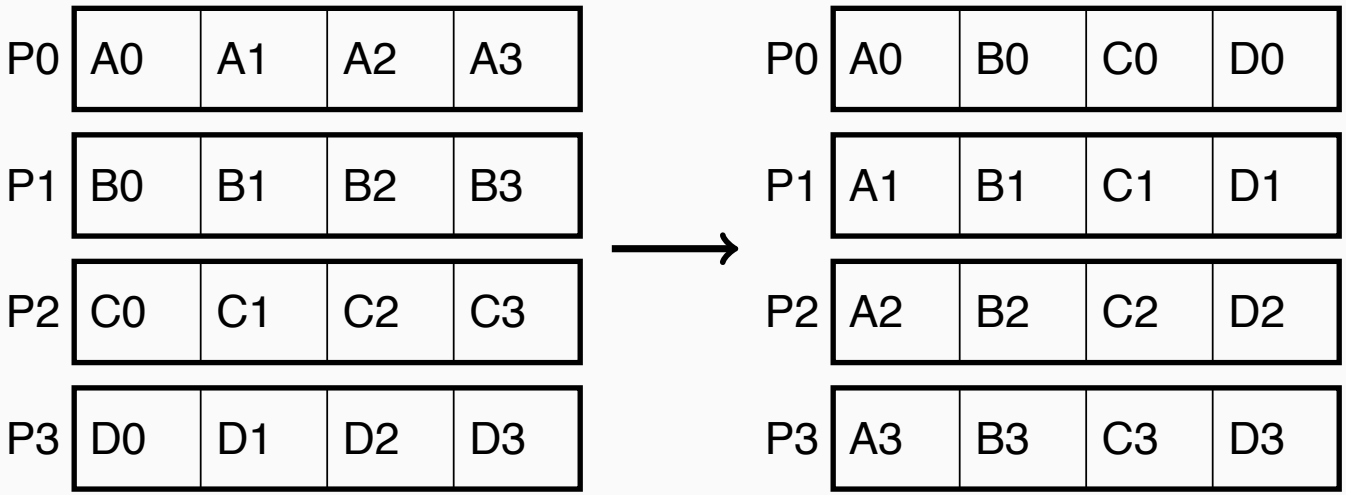
Scatter/gather



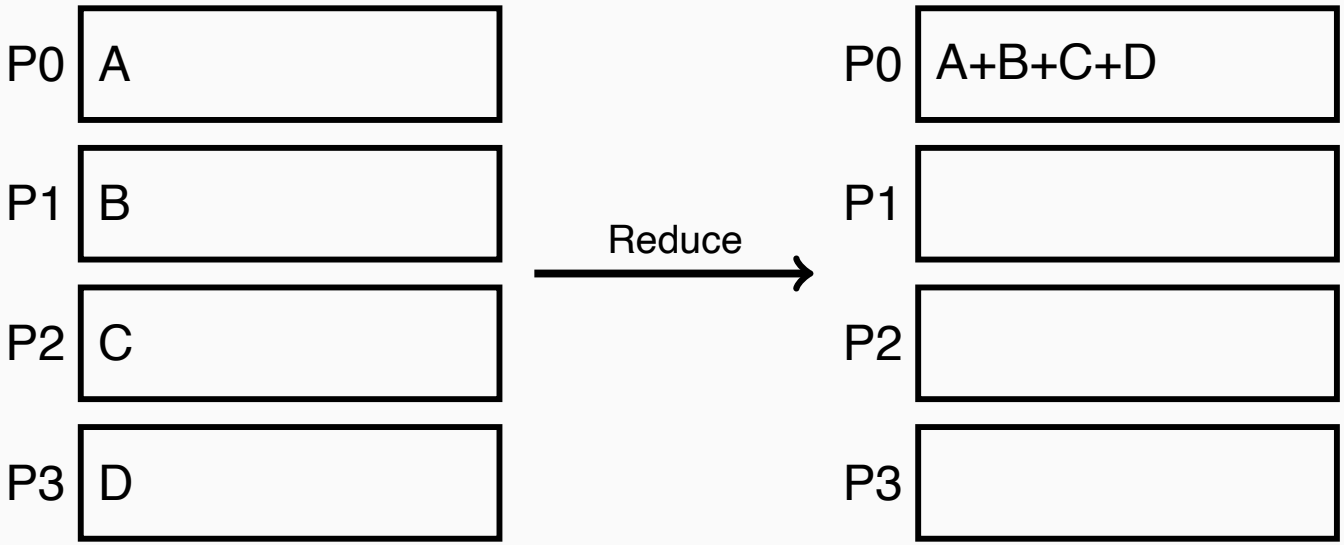
Allgather



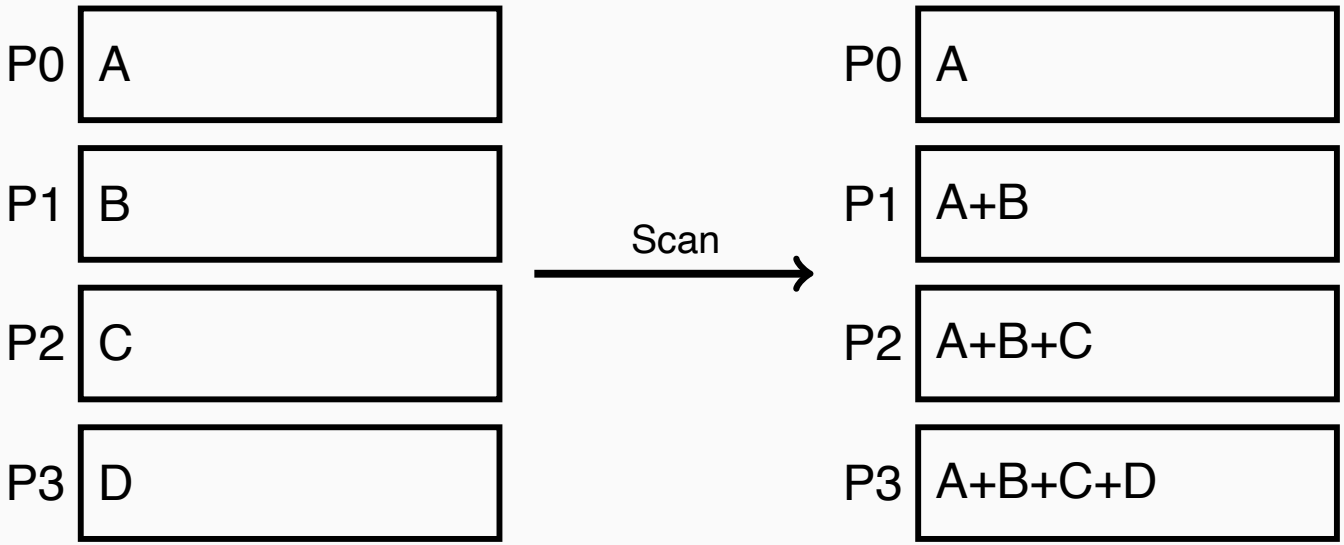
Alltoall


P0
A0
A1
A2
A3
P1
B0
B1
B2
B3
P2
C0
C1
C2
C3
P3
D0
D1
D2
D3
P0
A0
B0
C0
D0
P1
A1
B1
C1
D1
P2
A2
B2
C2
D2
P3
A3
B3
C3
D3

Reduce



Scan



The kitchen sink

- In addition to above, have vector variants (v suffix), more All variants (`Allreduce`), `Reduce_scatter`, ...
- MPI3 adds one-sided communication (put/get)
- MPI is *not* a small library!
- But a small number of calls goes a long way
  - `Init`/`Finalize`
  - `Get_comm_rank`, `Get_comm_size`
  - `Send`/`Recv` variants and `Wait`
  - `Allreduce`, `Allgather`, `Bcast`

Demo time

Let's look at how these work in practice:

https://github.com/cs5220-f24/demos/