Numerical Linear Algebra II: 02 Introduction to MPI

Jakub Kruzik
jakub.kruzik@vsb.cz

September 22, 2025 Ostrava

Strong and Weak Scaling

- **Strong scaling:** Fix total problem size, increase number of processes.
  - Goal: reduce time-to-solution.
  - Limited by communication overheads.
- **Weak scaling:** Increase problem size proportionally to number of processes.
  - Goal: keep time-to-solution constant.
  - Limited by load balance and communication growth.

See `https://hpc-wiki.info/hpc/Scaling`

Amdahl's Law

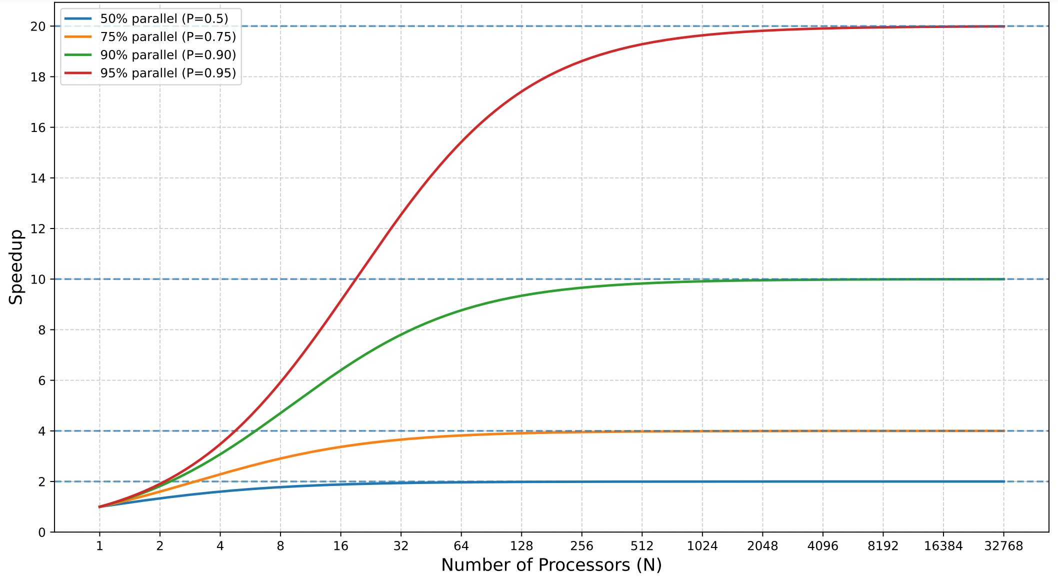
Speedup is limited by the fraction of the serial part of the software that is not amenable to parallelization:

$$\text{speedup} = 1/\left(s + \frac{p}{N}\right),$$

where p is proportion of execution time that can be parallelized, $s = 1 - p$ is the proportion of the serial part, and N is the number of processors (cores).

$$\text{speedup} \xrightarrow[N \to \infty]{} \frac{1}{s}$$

Amdahl's Law



50% parallel (P=0.5)
75% parallel (P=0.75)
90% parallel (P=0.90)
95% parallel (P=0.95)
Speedup
0
2
4
6
8
10
12
14
16
18
20
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
Number of Processors (N)

Message Passing Interface (MPI)

- Processes with separate memory spaces.
- Communication happens explicitly via messages.
- Able to run on clusters (of nodes (individual computers))
- Contrast with threads (e.g., OpenMP):
  - Threads share memory (care should be taken when reading/writing memory)
  - Can run only on a single node
- Mixing parallelism models is possible (MPI+X)

Communicators, Rank, and Size

- A communicator defines a group of processes that can talk to each other.
- Each process has a unique ID within the communicator: **rank**.
- The total number of processes is called **size**.

```
MPI_COMM_WORLD
// Contains all of the processes
MPI_COMM_SELF
// Contains only the calling process
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Comm_size(MPI_Comm comm, int *size)
```

Point-to-Point Communication

C interface:

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm);

int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm,
             MPI_Status *status);
```

Python (mpi4py):

```
comm.Send([buf, MPI.INT], dest=rank, tag=tag)
comm.Recv([buf, MPI.INT], source=rank, tag=tag)
```

Must match in source, destination, and tags.

Collective Communication

- Involve all processes in a communicator.
- Simplify common communication patterns.

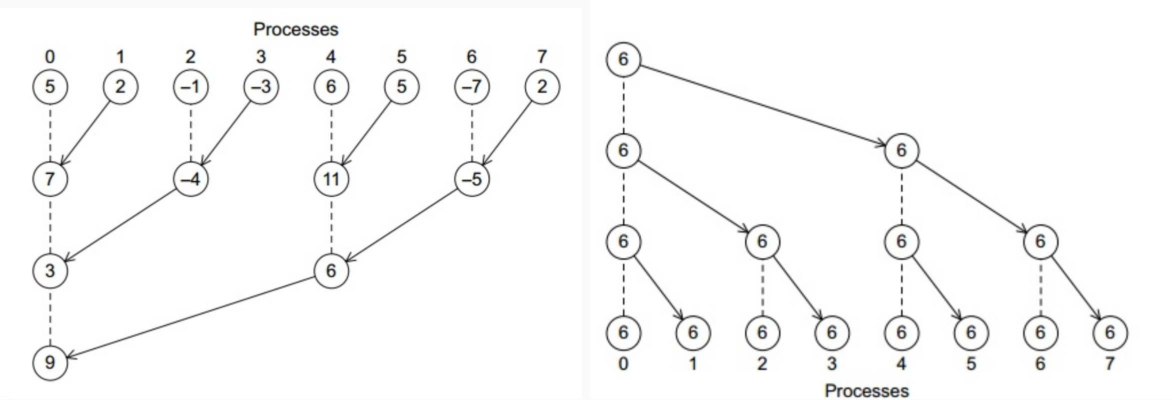
Collective Data Distribution

```
// Broadcast data from root to all processes in comm.
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
              int root, MPI_Comm comm);
// Distribute chunks of an array from root to all processes in comm.
int MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                void *recvbuf, int recvcount,
                MPI_Datatype recvtype, int root, MPI_Comm comm);
// Collect data from all processes in comm to root.
int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
               void *recvbuf, int recvcount, MPI_Datatype recvtype,
               int root, MPI_Comm comm);
// All processes gather data from everyone in comm
int MPI_Allgather(const void *sendbuf, int sendcount, MPI_Datatype sendtype
                  void *recvbuf, int recvcount, MPI_Datatype recvtype,
                  MPI_Comm comm);
```

Collective Reductions and Barrier

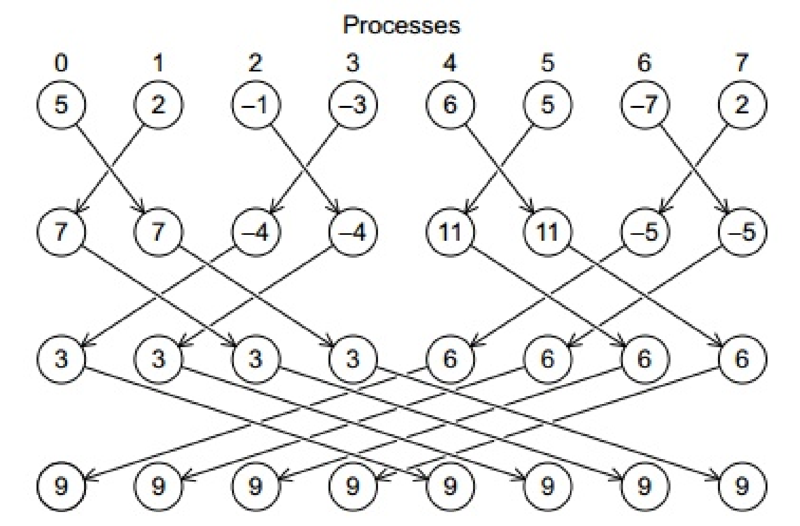
```
// Combine values from all processes, deliver result to root.
int MPI_Reduce(const void *sendbuf, void *recvbuf, int count,
               MPI_Datatype datatype, MPI_Op op, int root,
               MPI_Comm comm);
// Like Reduce, but result to all processes.
int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count,
                  MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);
// Computes partial reductions across processes.
int MPI_Scan(const void *sendbuf, void *recvbuf, int count,
             MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);
//MPI_Op = [MPI_MIN, MPI_MAX, MPI_SUM, MPI_PROD,...]

// Synchronize all processes in the communicator.
int MPI_Barrier(MPI_Comm comm);
```

Possible Implementation of Allreduce Sum: Reduction Tree + Broadcast Tree



Source: P. S. Pacheco, ”An Introduction to Parallel Programming”, 2011.

Possible Implementation of Allreduce Sum: Butterfly



Source: P. S. Pacheco, "An Introduction to Parallel Programming", 2011.

Non-Blocking Communication

- MPI supports asynchronous operations.
- `MPI_Isend`, `MPI_Irecv` return immediately.
- Progress is checked with `MPI_Wait` or `MPI_Test`.
- Enables overlap of computation and communication.

Summary

- MPI: distributed memory, message-based communication.
- Processes are grouped in a communicator and are identified by their rank.
- Supports point-to-point, collective, and non-blocking operations.
- Many more features: IO, one-sided communication, dynamic process management,...