
# Short introduction to MPI

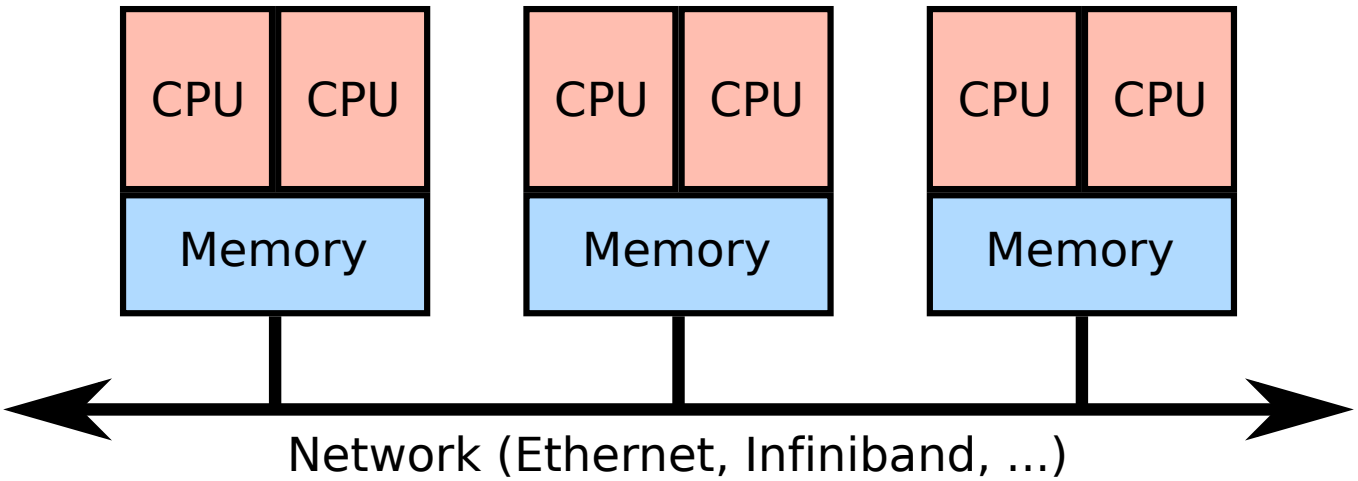
Simon Heybrock
simon.heybrock@esss.se

European Spallation Source

# Motivation

Computationally challenging problems:

- Exceedingly long computation runtime on a single computer
- More memory required than available on single computer



# MPI — message passing interface

MPI = standard, many implementations: OpenMPI, Intel MPI, MVAPICH, ...

- Communications library for processes on distributed hardware
- Bindings for C, C++ (boost), Python, ...
- Support for communication: typically Ethernet, Infiniband, shared memory

## Trivial example

```
$ echo Hi!
Hi!
$ mpirun -n 4 echo Hi!
Hi!
Hi!
Hi!
Hi!
```

# MPI basics (1/3)

- This is not a tutorial: Will omit many technical details here.

## MPI rank = linear index of process

```
1 int rank;
2 MPI_Comm_rank(&rank);
3 printf("Hi, I am rank %d!\n", rank);
```

# MPI basics (1/3)

- This is not a tutorial: Will omit many technical details here.

## MPI rank = linear index of process

```
1 int rank;
2 MPI_Comm_rank(&rank);
3 printf("Hi, I am rank %d!\n", rank);
```

```
$ mpirun -n 3 ./test
Hi, I am rank 1!
Hi, I am rank 0!
Hi, I am rank 2!
```

Omitted details: MPI initialization, communicators (think namespace for processes), MPI finalize

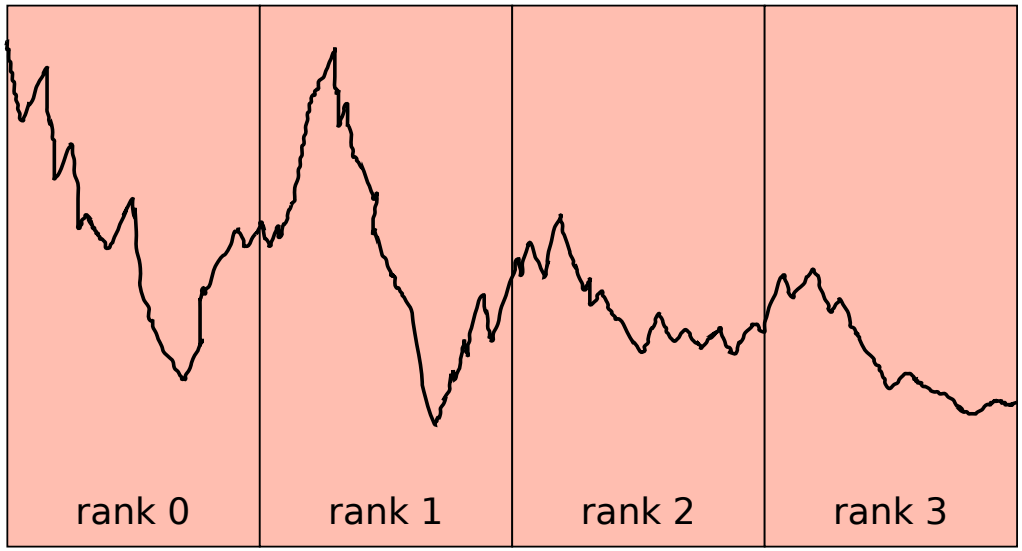
# MPI basics (2/3)

## Point-to-point communication

```
1 int size = sizeof(double);
2 MPI_Send(f[length-1], size, rank+1);
3 MPI_Recv(buffer, size, rank-1);
```

Omitted details: data types, message tags, status, blocking/non-blocking comms.

Example: compute numeric derivative $f'[i] = (f[i+1] - f[i-1])/\Delta$



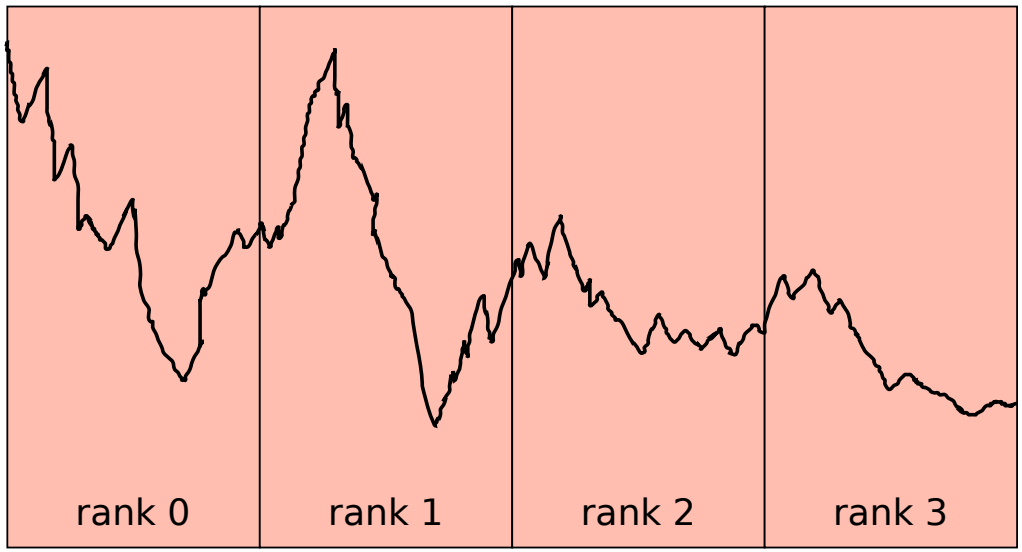
# MPI basics (3/3)

## Collective communication

```
1 int root_rank = 0;
2 MPI_Gather(local_result, buffer, root_rank);
3 if(rank == root_rank)
4     // sum all local results (now in buffer)
```

Omitted details: data types, message tags, status, blocking/non-blocking comms.

Example: compute numeric integral



# So far so good?

## Gist

- MPI basics relatively simple
  - In many cases 10 functions from MPI library enough
  - MPI-2 and MPI-3 can get quite complex, often not necessary
- Hard part is to figure out good way to map problem onto distributed architecture in an efficient way

## Things to consider when writing an MPI program

- Network latencies
- Network bandwidth
- Programming effort
  - if you need to rewrite your program from scratch with MPI library calls in almost every line, something is wrong
- Load balance, ...

# Scalability of a computational problem

$N$ = number of computers (cores, processors, nodes in a cluster)

# Scalability of a computational problem

$N$ = number of computers (cores, processors, nodes in a cluster)

## "Trivial scaling": $N \sim$ number of problems

User measured 1000 samples on instrument X; reduction script.

- All reductions independent
- Can use 1 computer or $N$ computers
- $T \sim 1/N$

Difficulty: trivial

# Scalability of a computational problem

$N$ = number of computers (cores, processors, nodes in a cluster)

## "Trivial scaling": $N \sim$ number of problems

User measured 1000 samples on instrument X; reduction script.
Difficulty: trivial

## Weak scaling: $N \sim$ problem size

User measures on instrument X (10.000 pixels), instrument Y (100.000 pixels), and instrument Z (1.000.000 pixels)

- Run reduction for X on 1 computer
- Run reduction for Y on 10 computers
- Run reduction for Z on 100 computers

Difficulty: relatively easy (but limited)

# Scalability of a computational problem

$N$ = number of computers (cores, processors, nodes in a cluster)

## "Trivial scaling": $N \sim$ number of problems

User measured 1000 samples on instrument X; reduction script.
Difficulty: trivial

## Weak scaling: $N \sim$ problem size

User measures on instrument X (10.000 pixels), instrument Y (100.000 pixels), and instrument Z (1.000.000 pixels)
Difficulty: relatively easy (but limited)

## Strong scaling: fixed problem size, increase $N$

Users measures on instrument X

- On a single computer the reduction takes 10 days
- Try to run reduction on more and more computers

Difficulty: relatively hard (often quite limited)

# Summary

Thank you for your attention!
Questions?