Parallel Programming

pauldj@aices.rwth-aachen.de

Preliminaries

- Distributed-memory architecture
- Back in the days: Node $\equiv$ CPU $\equiv$ process
- Nowadays: NODE $\rightarrow$ CPUs $\rightarrow$ multi cores $\rightarrow$ many processes
- Assumption: fully connected topology
- Assumption: each process can symultaneously send and receive
- Assumption: messages in opposite directions do not cause a conflict

What is “MPI”?

- A **library**, not a language, not a program.

"Minimal" MPI

`MPI_Init(...)`	MPI Initialization
`MPI_Comm_size(...)`	How many processes are there?
`MPI_Comm_rank(...)`	What rank am I?
`MPI_Send(...)`	Send data to another process
`MPI_Recv(...)`	Receive data from another process
`MPI_Finalize( )`	MPI termination

What is “MPI”?

- A **library**, not a language, not a program.
- In fact, it’s the specification of a library, not the actual implementation.
- MPI defines the interface, the functionality and the semantics of functions that deliver a message passing mechanism.
- Idea: clear separation between data communication and application.
- Both open-source and proprietary implementations.
- De-facto standard for distributed-memory parallelism.
- `www.mpi-forum.org`

`int MPI_Init( ...)`

- `MPI_Init(&argc, &argv);`
- First MPI function
- Args not specified; an implementation might use them
- Query: `MPI_Initialized`

`int MPI_Finalize()`

- Last MPI function
- No arguments
- Query: `MPI_Finalized`

`int MPI_Comm_size(MPI_Comm comm, int *size)`

- Returns the number of processes in the communicator `comm`
- Communicator: for now `MPI_COMM_WORLD` $\equiv$ "everybody"

`int MPI_Comm_rank(MPI_Comm comm, int *rank)`

- Returns the rank of the calling process within the communicator
- The rank is THE unique process identifier!
- NOTE: each process (rank) can be multi-threaded.

Send $\leftrightarrow$ Recv

- Objective: data movement
- `MPI_Send` and `MPI_Recv` must be matched
- Blocking communication

Necessary information:

`Send`	`Recv`
`dest`	`source`
`*buffer`	`*target`
`size`	`size`
`datatype`	`datatype`
`tag`	`tag`
`comm`	`comm`

`int MPI_Send(*buffer, count, datatype, dest, tag, comm)`

- `*buffer` is an address!
- `count` is indispensible; so is `datatype`
- `dest` is a rank (in `comm`)
- `tag` is an integer

`int MPI_Recv(*target, count, datatype, source, tag, comm, *status)`

- `*target, count, datatype` as for the `Send`
- `source` is either a rank (in `comm`) or `MPI_ANY_SOURCE`
- `tag` is either an integer or `MPI_ANY_TAG`
- `*status` on exit, contains info about the message

History

Before 1994

- Before MPI, no standards
- Different computers, different needs $\Rightarrow$ **many** message passing environments
- N-cube, P4, PICL, PVM, ISIS, Express, Zipcode; Intel NX, IBM EUI, IBM CCL, ...
- **A lot of duplication!**
- **No portability whatsoever**

History

- **[1992]** First “MPI Forum” meeting (Supercomputing ’92)
- **[1993–94]** Seven “MPI Forum” meetings. Working on the MPI standard.
- **[1994]** Release of the first MPI standard: MPI-1
- **[1995]** First implementations of MPI: MPICH, LAM MPI, . . .
- **[1998]** Release of the second MPI standard: MPI-2.
  More than 100 new functions!
- **[2002]** Complete implementations of MPI-2.
  Dynamic process management, 1-sided communication, MPI-I/O
- **[2012]** Release of MPI-3.
  Non-blocking collectives, sparse collectives, . . .

Thanks to Jesper Larsson Träff (TU Wien).

Collective Communication

Broadcast	$\leftrightarrow$	Reduce
Scatter	$\leftrightarrow$	Gather
Allgather	$\leftrightarrow$	Reduce-scatter
Allreduce		
Alltoall		

References

- "Collective Communication: Theory, Practice, and Experience", Chan, Heimlich, Purkayastha, van de Geijn. (FLAME working note #22)
- Collective Communications in MPI

  http://www.mcs.anl.gov/research/projects/mpi/tutorial/gropp/node72.html

Reduction

```
MPI_Datatype:
MPI_CHAR, MPI_INT, MPI_UNSIGNED, MPI_FLOAT, MPI_DOUBLE, ...

MPI_Op:
MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND, ...,

MPI_Op_create( &function, commute, &new_op );
MPI_Op_Free( &op );
```