HPC Programming

Message Passing Interface (MPI), Part I

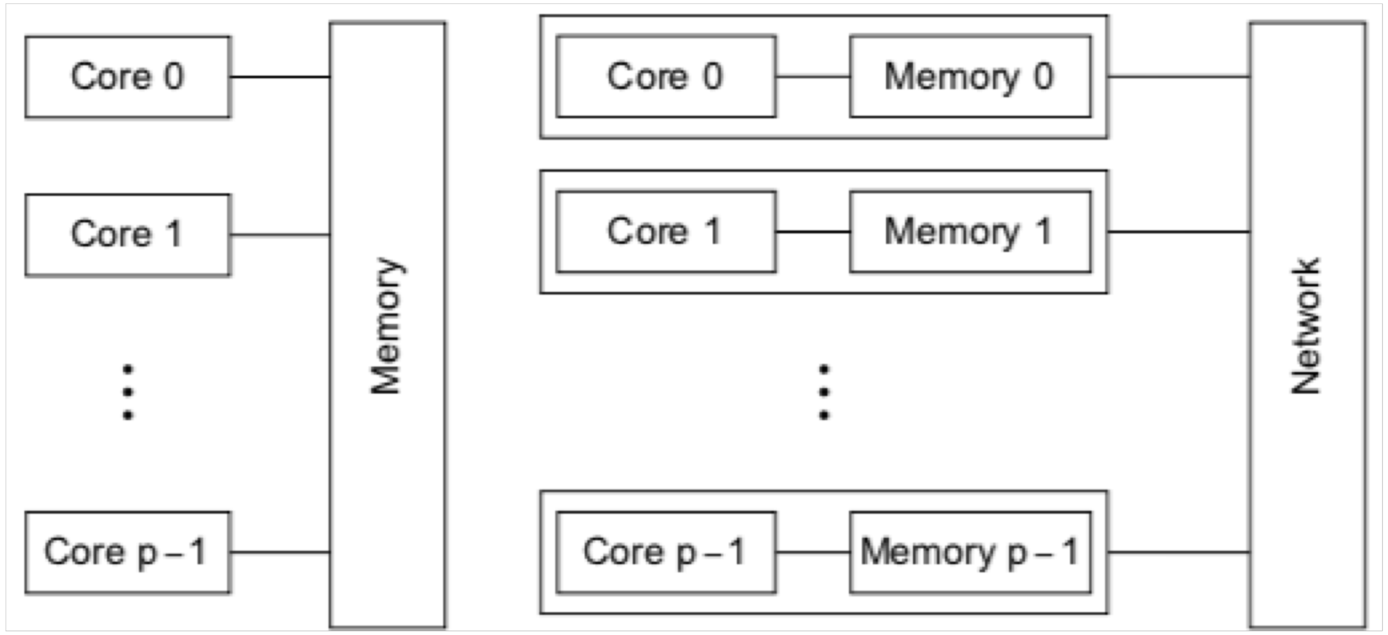
Peter-Bernd Otte, 20.11.2018

Shared System: Why two C extensions?

(a) Shared-Memory system:
- Each core can read/write each memory location
- Coordination of cores via shared-memory locations
- Use OpenMP
- Small projects. HIMster2: up to 32 cores/node

(b) Distributed-Memory system:
- Each core has private memory
- Cores explicitly sending messages for data exchange and coordination
- MPI
- Several nodes of a cluster

- Hybrid-Programming:
  - OpenMP+MPI



(a) (b)

Overview: Next lectures

- 4 lectures incl. hands on with MPI
  - 20., 27.11, 4., 11.12. (skip 18.12.)
- 1 lecture intro to Rust
  - https://en.wikipedia.org/wiki/Rust_(programming_language)
  - 8.1.2019
- 2 lectures on debugging
- Further lectures: open for your wishes

Worked out example: bandwidth (CPU: i7-4790)

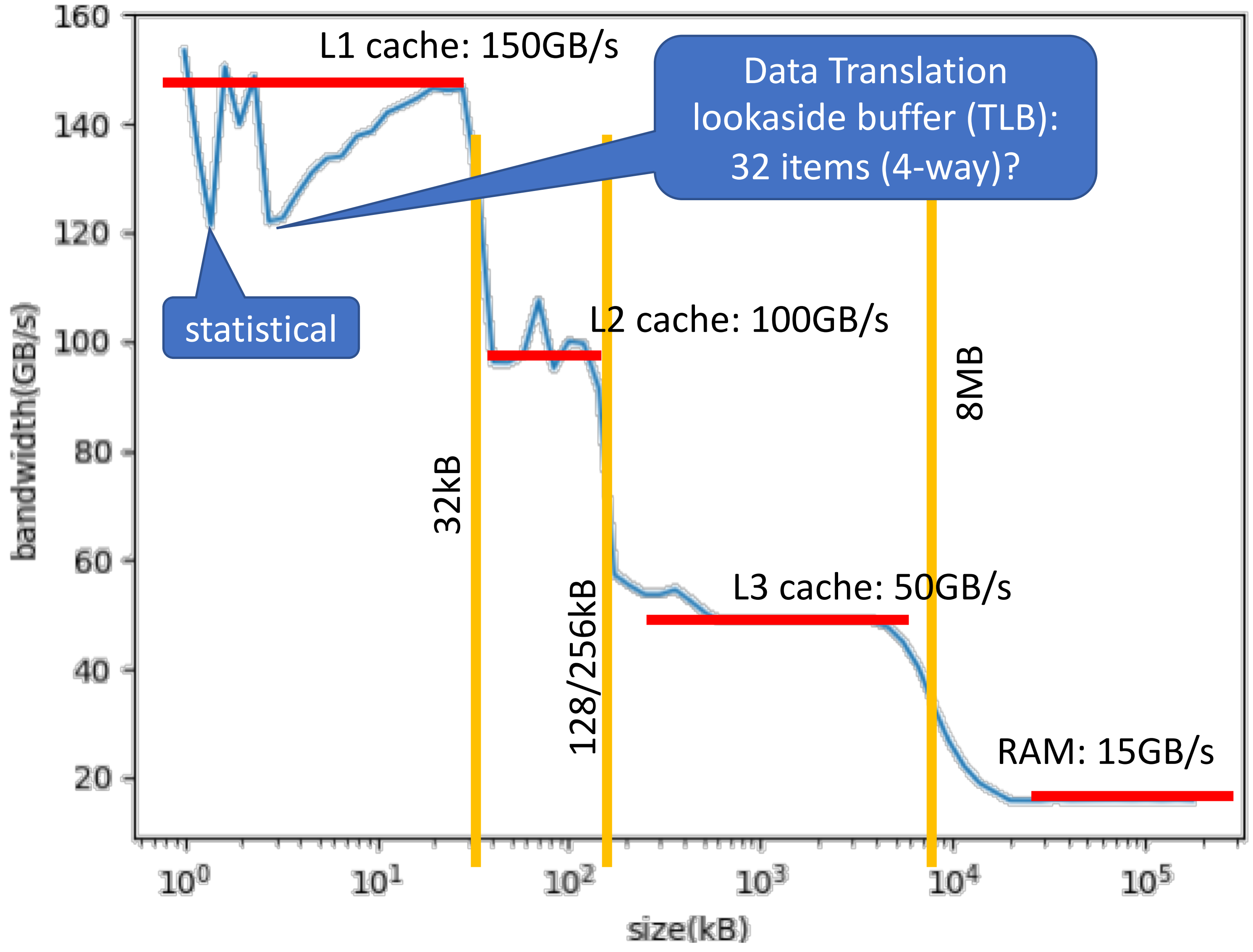
main routine:

```
int *mem;
for(int i=0;i<n;i+=16) {
    result+=mem[i];
}
```

cache line block size = 64 bytes = 16 ints

```
git clone –recursive https://github.com/realead/memmeter
bash run_test.sh band_width
```

CPU under test: i7-4790 CPU @ 3.90GHz (cat /proc/cpuinfo | grep MHz)



Worked out example: latency (CPU: i7-4790)

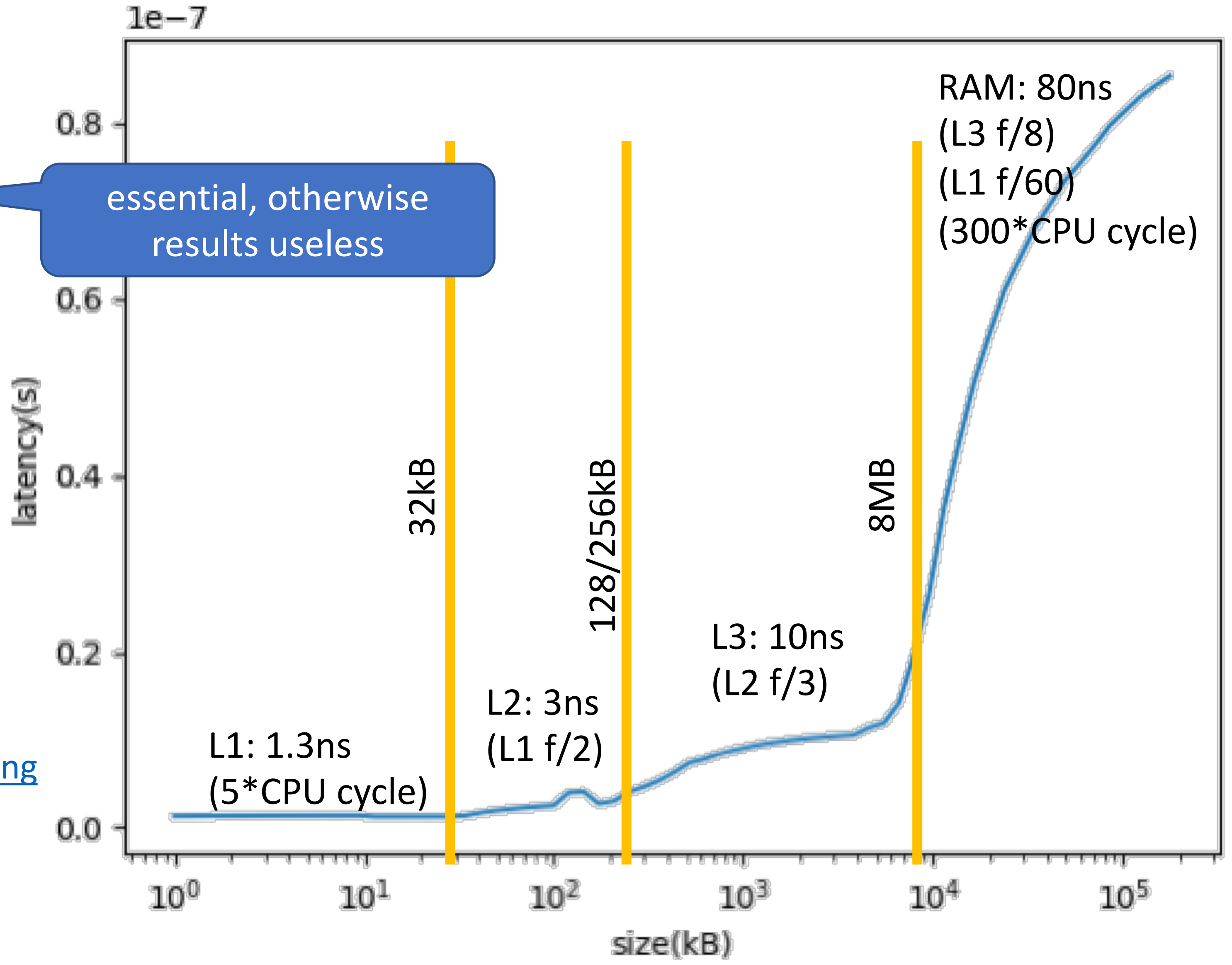
main routine:
int *mem[] = 0..steps, but randomly ordered to avoid cache prefetching

```
for(unsigned int
i=0;i<steps;i++){
   index=mem[index];
}
```

essential, otherwise results useless

```
bash run_test.sh latency
```

see:
https://en.wikipedia.org/wiki/Cache_prefetching



Introduction MPI

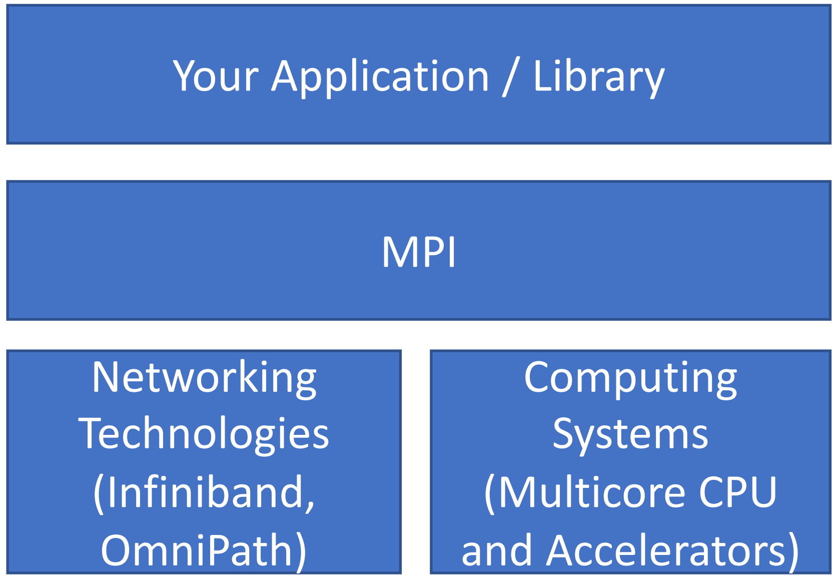
MPI: Getting Started (1)

- Message Passing Interface (MPI) by MPI Forum (mpi-forum.org)
  - underlying: Distributed Memory Model
- De facto standard in parallel computing
  - Developed by academia and industry since 1991
  - C and Fortran in version 3.1 (2008, MPI 4 in development)
  - Several well-tested and efficient implementations available

- Other attempts (will not be covered by this lecture):
  - PGAS (Partitioned Global Address Space):
    - parallel programming model: assumes global memory, logically partitioned and a portion of it is local to each process
    - Library based: Global Arrays, OpenSHMEM
    - Compile based: Unified Parallel C (UPC), Co-Array Fortran (CAF)
    - HPCS (High Productivity Computing Systems) PGAS, Language-based: Chapel (Cray), X10 (IBM)
  - PVM (Parallel Virtual Machine, last update 2011) for set of heterogenous machines

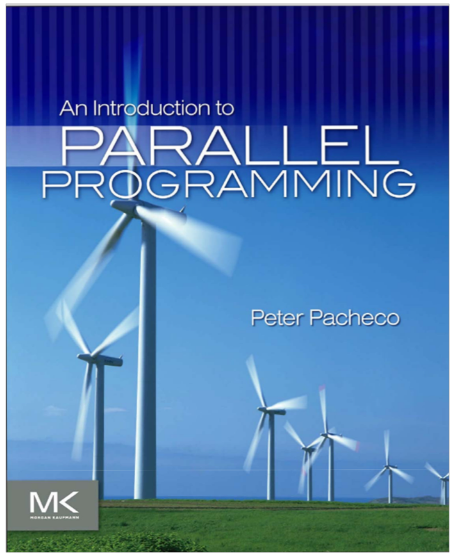
MPI: Getting Started (2)

- Major MPI Features
  - Point to point Two sided Communication
  - Collective Communication
  - One-sided Communication
  - Job Startup
  - Parallel I/O
- Why?
  - Abstract message and file I/O exchange
    - simplifies your programming
  - Overlap computation and communication:
    - e.g. Non-blocking collectives and sending/receiving
  - Resiliency:
    - integrated failure detection

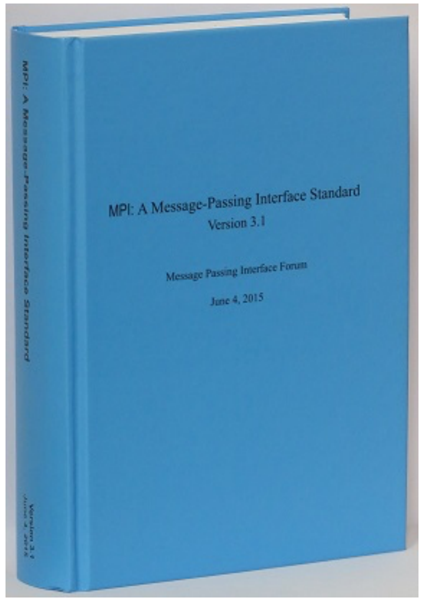

Your Application / Library
MPI
Networking Technologies (Infiniband, OmniPath)
Computing Systems (Multicore CPU and Accelerators)

MPI: Getting Started (3)

- Good reads:
  - An Introduction to Parallel Programming by Peter Pacheco



  - MPI-3.1 Standard



MPI: A Message-Passing Interface Standard
Version 3.1

Message Passing Interface Forum

June 4, 2015

MPI: Fast lane to “Hello World”

- Compilation of a MPI program:
  - mpicc
  - wrapper script for the C compiler
- Executing a MPI program:
  - mpirun -n [NumberOfRanks] ./Executable
  - (machine dependent: mpiexec, srun, …)
- But: to have MPI available on HIMster2: first load corresponding module
  - module load mpi/OpenMPI/3.1.0-GCC-6.3.0
  - On head node and on compute node, put in submit script
  - → see next slide

MPI libraries on HIMster2

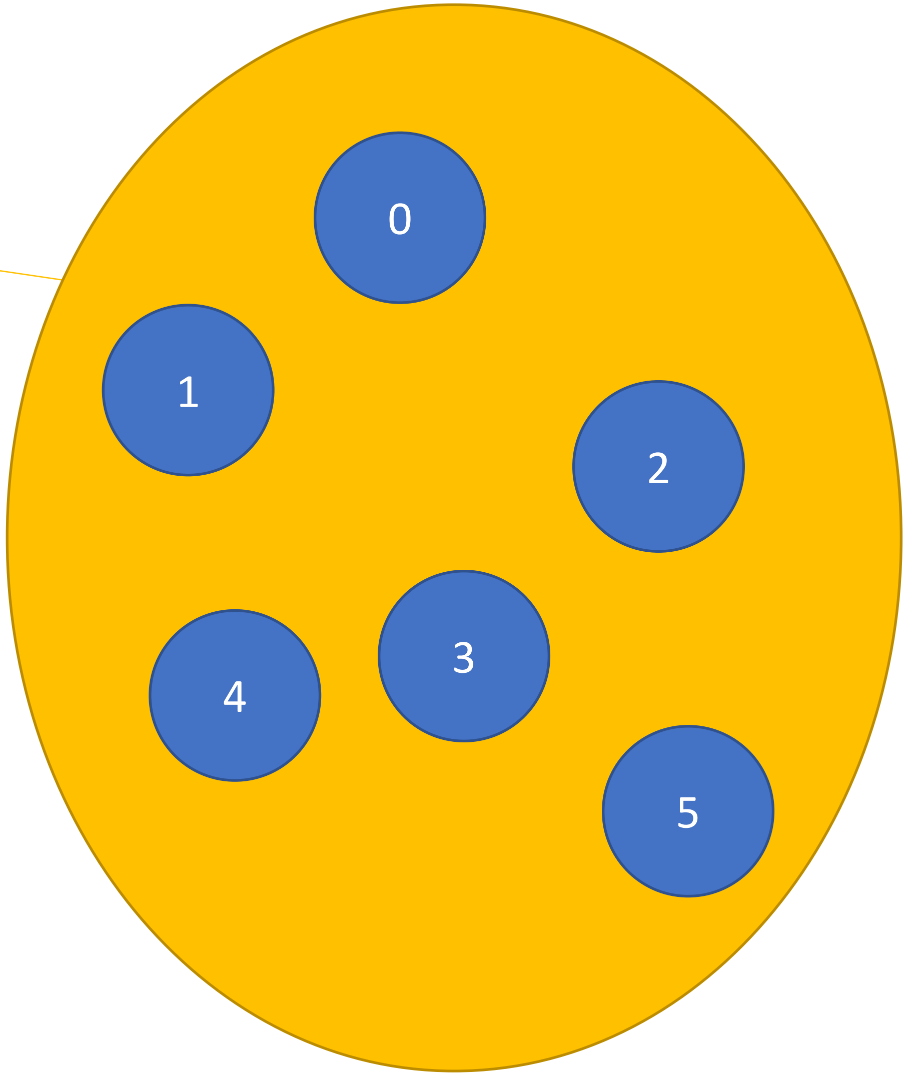
```
$ module load mpi/
mpi/impi/2017.2.174-iccifort-2017.2.174-GCC-6.3.0  mpi/OpenMPI/2.1.2-GCC-6.3.0
mpi/impi/2018.0.128-iccifort-2018.0.128-GCC-6.3.0  mpi/OpenMPI/3.0.0-GCC-6.3.0
mpi/impi/2018.1.163-iccifort-2018.1.163-GCC-6.3.0  mpi/OpenMPI/3.0.1-GCC-6.3.0
mpi/impi/2018.2.199-iccifort-2018.2.199-GCC-6.3.0  mpi/OpenMPI/3.0.1-GCC-6.4.0
mpi/impi/2018.3.222-iccifort-2018.3.222-GCC-6.3.0  mpi/OpenMPI/3.0.1-GCC-8.1.0
mpi/MVAPICH2/2.2-GCC-6.3.0                         mpi/OpenMPI/3.0.1-iccifort-2018.2.199-GCC-6.3.0
mpi/OpenMPI/1.10.4-GCC-6.3.0                       mpi/OpenMPI/3.1.0-GCC-6.3.0
mpi/OpenMPI/2.0.2-GCC-6.3.0                        mpi/OpenMPI/3.1.1-iccifort-2018.3.222-GCC-6.3.0
```

MPI: Basics

- `int MPI_Init(int *argc_p, char ***argv_p)`
    - tells MPI system to setup (eg allocate storage for message buffers, decides which process gets which rank, creates MPI_COMM_WORLD communicator)
    - MPI_Init(Null, Null) just fine for beginning.
    - no other MPI function call before this
- `int MPI_Finalize(void)`
    - tells MPI system: we are done using MPI, free resources allocated for MPI
    - no MPI functions after this call, also no MPI_Init!

MPI: Communicators

- MPI Communicator
  = group of processes that can send messages to each other.
- All processes are in `MPI_COMM_WORLD` communicator
  - Defining sub groups → see future lecture
- Number of members in communicator with
```
int MPI_Comm_size (
      MPI_Comm comm /*in*/,
      int *comm_size_p /*out*/)
```
- Get rank of sub_process with
```
int MPI_Comm_rank (
      MPI_Comm comm /*in */,
      int * my_rank_p /*out*/)
```



Single Program, Multiple Data (SPMD)

- Standard MPI programming:
  - Write single executable
  - behaviour depends on its rank
    - eg rank=0: message collecting master,
      ranks>0: computing
  - Number of ranks from 1 to O(10^4) on Himster2
    - O(10^6) on extreme machines
  - called “Single Program, multiple Data”
- ⇔ Multiple-Program Multiple-Data (MPMD)
  - even mixture of different software possible with MPI:
    Fortran and C executable communicating fine

```
MPI_Comm_rank(MPI COMM WORLD, &my_rank);

if (my_rank == 0) {
  ...
} else {
  ...
}
```

MPI: MPI_Send

- Sending a message to another receiving rank
- Syntax:

```
int MPI_Send(
      void          *msg_buf_p    /*in*/,
      int           msg_size      /*in*/,
      MPI_Datatype  msg_type      /*in*/,
      int           dest          /*in*/,
      int           tag           /*in*/,
      MPI_Comm      communicator  /*in*/);
```

defines contents of message

defines destination of message

- dest = receiving rank (defined in communicator)
- tag to distinguish messages
- defines the “communication universe”,
  all processes are in: `MPI_COMM_WORLD`

MPI: MPI_Recv

- Receiving a message from another rank
- Syntax:

```
int MPI_Recv(
      void            *msg_buf_p     /*in*/,
      int             msg_size       /*in*/,
      MPI_Datatype    msg_type       /*in*/,
      int             source         /*in*/,
      int             tag            /*in*/,
      MPI_Comm        communicator   /*in*/,
      MPI_Status      *status_p      /*out*/);
```

defines contents of message

defines destination of message

- source = sender rank (defined in communicator). To accept all: MPI_ANY_SOURCE
- tag to distinguish messages. To accept from all: MPI_ANY_TAG
- defines the “communication universe”, no wildcard available, all processes are in: `MPI_COMM_WORLD`
- `stauts_p` to retrieve error information, or: `MPI_STATUS_IGNORE`

MPI: Make a match

- rank s calls: `MPI Send(send_buf, send_buf_size, send_type, dest, send_tag, send_comm);`

- rank q calls: `MPI Recv(recv_buf, recv_buf_size, recv_type, src, recv_tag, recv_comm, &status);`

- All 5 “green” parameters need to match to get message successfully through.
  - all mandatory to be equal, except recv_buf_size >= send_buf_size

MPI: Elementary datatypes

- C types can't be passed → use MPI datatypes
- Advantage: interoperability with other software and hardware

MPI datatype	C equivalent
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	char

Set up your workbench

- Connect 2 times via SSH to Mogon2 / HIMster2
  1. Use the first SSH connection for editing (gedit, vi, vim, nano, geany) and
     module load mpi/OpenMPI/3.1.1-GCC-7.3.0
     compiling: mpicc -o ExecutableName SourceFileName.c
  2. Use the second connection for the interactive execution on the nodes (no execution on the head node!):
     salloc -p parallel -N 1 --time=01:30:00 -A m2_himkurs --reservation=himkurs -C skylake
     module load mpi/OpenMPI/3.1.1-GCC-7.3.0
     mpirun -n 2 ./ExecutableName
- Download the files via: wget https://www.hi-mainz.de/fileadmin/user_upload/IT/lectures/WiSe2018/HPC/files/05.zip && unzip 05.zip

Hints:

- If the reservation with salloc –p parallel fails, try:
  - salloc -p devel -n 4 -A m2_him_exp
- The reserved resources with salloc can’t be overwritten with mpirun
  - Resources(salloc) => Resources(mpirun)
- Possible to check reservation with: squeue -u USERNAME

Backup: Mogon 1

- Access to head node of Mogon1
    - No computational intense execution allowed, only in case HIMster2 is down

```
ssh <username>@mogon.zdv.uni-mainz.de
module load mpi/OpenMPI/3.1.1-GCC-7.3.0
mpicc -o execname source.c
mpirun -n 2 execname
```

Exercise 1: Getting started

Learning objectives:

- First use of MPI

Steps:

1. Download the skeleton from lecture webpage:
   - wget https://www.hi-mainz.de/fileadmin/user_upload/IT/lectures/WiSe2018/HPC/files/MPI-01.zip && unzip MPI-01.zip
2. Compile the program and run it with different numbers of ranks (number of cores). Count the number of lines printed and compare.
3. Each rank shall print out its process number.
4. Bonus:
   - Can the output be sorted according to the rank number?
   - How could this done?

Exercise 2: Sending first messages

Learning objectives:

- First use of messaging

Steps:

1. Download the skeleton from lecture webpage:
   - wget https://www.hi-mainz.de/fileadmin/user_upload/IT/lectures/WiSe2018/HPC/files/MPI-02.zip && unzip MPI-02.zip
2. Insert the correct MPI_Send and MPI_Recv procedures to enable the sending of messages:
   - rank 0 shall receive from all other sub-programs a string. Print out this string.
   - All other ranks (rank i>0) shall send a string to rank 0.

4. Bonus:
   - Change your program to send two decimal values instead of a string (sending my_rank and comm_size).