High Performance
Computing &
Big Data Services

hpc.uni.lu

hpc@uni.lu

@ULHPC

LUXEMBOURG
LET'S MAKE IT HAPPEN

Uni.lu HPC School 2021

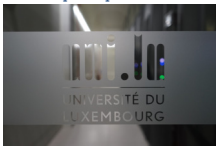
PS5: Scalable Science: Parallel computations with OpenMP/MPI

Uni.lu High Performance Computing (HPC) Team

Dr. S. Varrette

University of Luxembourg (UL), Luxembourg

http://hpc.uni.lu



Latest versions available on Github:

UL HPC tutorials: https://github.com/ULHPC/tutorials

UL HPC School: hpc.uni.lu/education/hpcschool

PS5 tutorial sources: ulhpc-tutorials.rtfd.io/en/latest/parallel/basics/

Summary

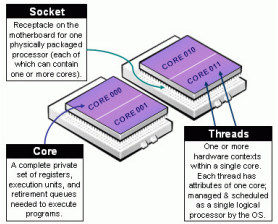
Main objectives of this session

- See how to run **threaded parallel OpenMP** programs
- See how to **use the MPI suits** available on the UL HPC platform:
  - ↪ Intel MPI and the Intel MKL
  - ↪ OpenMPI
- **Build** and **run** hybrid OpenMP/MPI programs
  - ↪ interactive and passive (through **launcher**) jobs

- **Test cases** on reference parallel and distributed **benchmarks**:
  - ↪ OSU micro-benchmarks:
    - ✓ measure the performances of various MPI operations

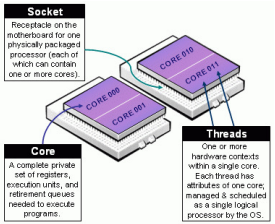
Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** `--ntasks-per-node <n>`



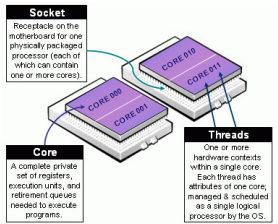
Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** `--ntasks-per-node <n>`
  - ↪ Slurm Socket = Physical Socket/**CPU**/Processor `--ntasks-per-socket <n>`



Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** `--ntasks-per-node <n>`
  - ↪ Slurm Socket = Physical Socket/**CPU**/Processor `--ntasks-per-socket <n>`
  - ↪ **Slurm CPU** = Physical **Core** `-c <#threads>`
    - ✓ Hyper-Threading (HT) Technology is **disabled** on all the compute nodes
    - ✓ #cores = #threads `-c <N>` ⟶ `OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}`
    - ✓ Total number of tasks: `${SLURM_NTASKS}` ⟶ `srun -n ${SLURM_NTASKS} [...]`



Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node — `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** — `--ntasks-per-node <n>`
  - ↪ Slurm Socket = Physical Socket/**CPU**/Processor — `--ntasks-per-socket <n>`
  - ↪ **Slurm CPU** = Physical **Core** — `-c <#threads>`
    - ✓ Hyper-Threading (HT) Technology is **disabled** on all the compute nodes
    - ✓ #cores = #threads `-c <N>` ⟶ `OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}`
    - ✓ Total number of tasks: `${SLURM_NTASKS}` ⟶ `srun -n ${SLURM_NTASKS} [...]`

- **Important: Always align resource specs with physical NUMA characteristics**
  - ↪ **Ex (AION): 16 cores per socket, 8 sockets ("physical" CPUs) per node** (128c/node)
  - ↪ `[-N <N>] --ntasks-per-node <8n> --ntasks-per-socket <n> -c <thread>`
    - ✓ **Total**: `<N>`×8×`<n>` tasks, each on `<thread>` threads
    - ✓ **Ensure** `<n>`×`<thread>`= 16 on aion
    - ✓ Ex: `-N 2 --ntasks-per-node 32 --ntasks-per-socket 4 -c 4` (**Total**: 64 tasks)

Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** `--ntasks-per-node <n>`
  - ↪ Slurm Socket = Physical Socket/**CPU**/Processor `--ntasks-per-socket <n>`
  - ↪ **Slurm CPU** = Physical **Core** `-c <#threads>`
    - ✓ Hyper-Threading (HT) Technology is **disabled** on all the compute nodes
    - ✓ #cores = #threads `-c <N>` ⟶ `OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}`
    - ✓ Total number of tasks: `${SLURM_NTASKS}` ⟶ `srun -n ${SLURM_NTASKS} [...]`

- **Important: Always align resource specs with physical NUMA characteristics**
  - ↪ **Ex (IRIS): 14 cores per socket, 2 sockets ("physical" CPUs) per node** (28c/node)
  - ↪ `[-N <N>] --ntasks-per-node <2n> --ntasks-per-socket <n> -c <thread>`
    - ✓ **Total**: `<N>`×2×`<n>` tasks, each on `<thread>` threads
    - ✓ **Ensure** `<n>`×`<thread>`= 14 on iris
    - ✓ Ex: `-N 2 --ntasks-per-node 4 --ntasks-per-socket 2 -c 7` (**Total**: 8 tasks)

Specific Resource Allocation

- **Beware of Slurm terminology** in Multicore Architecture!
  - ↪ Slurm Node = Physical node `-N <#nodes>`
    - ✓ **Advice**: explicit number of expected tasks **per node** `--ntasks-per-node <n>`
  - ↪ Slurm Socket = Physical Socket/**CPU**/Processor `--ntasks-per-socket <n>`
  - ↪ **Slurm CPU** = Physical **Core** `-c <#threads>`
    - ✓ Hyper-Threading (HT) Technology is **disabled** on all the compute nodes
    - ✓ #cores = #threads `-c <N>` ⟶ `OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}`
    - ✓ Total number of tasks: `${SLURM_NTASKS}` ⟶ `srun -n ${SLURM_NTASKS} [...]`

Hostname	Node type	#Nodes	#Socket	#Cores	RAM	Features
aion-[0001-0318]	Regular	318	8	128	256 GB	batch,epyc
iris-[001-108]	Regular	108	2	28	128 GB	batch,broadwell
iris-[109-168]	Regular	60	2	28	128 GB	batch,skylake
iris-[169-186]	Multi-GPU	18	2	28	768 GB	gpu,skylake,volta
iris-[191-196]	Multi-GPU	6	2	28	768 GB	gpu,skylake,volta32
iris-[187-190]	Large Memory	4	4	112	3072 GB	bigmem,skylake

- List available features: `sfeatures`

```
# OR: sinfo -o '%20N %.6D %.6c %15F %12P %f'
```

Summary

OpenMP

- OpenMP: Open Multi-Processing
  - ↪ popular parallel programming model for multi-threaded applications.
  - ↪ API for **multi-platform shared memory multiprocessing**
    - ✓ in C, C++, and Fortran
    - ✓ on most platforms, instruction set architectures and OS.
  - ↪ Parallelism accomplished **exclusively** through the use of threads.
    - ✓ **Thread**: smallest unit of processing that can be scheduled by an OS
  - ↪ **#threads $\simeq$ number of machine processors/cores**.
    - ✓ `OMP_NUM_THREADS` (if present): **initial max** number of threads;

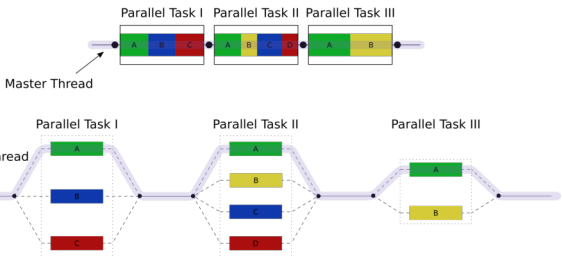
OpenMP

https://www.openmp.org/

- Reference website
- **Latest version: 5.2** (Nov 2021) – specifications

OpenMP Programming Model

- Explicit (not automatic) programming model
  - ↪ may mean taking a serial program & insert compiler directives... `#pragma omp [...]`
- **Fork-Join** model of parallel execution
  - ↪ **FORK**: **master** thread creates a **team** of parallel threads
  - ↪ **JOIN**: team threads complete statements in parallel regions
    - ✓ then they synchronize & terminate, leaving only the master thread.



Controlling Number of Threads

- **Environmental variable**: exploit slurm reservation `sbatch -c <threads>[...]`
  - ↪ `export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}`
  - ↪ Max number of theads (`omp_get_max_threads()`)
- **Run-Time Library Routines** provides useful functions to query the threads.

Primitive	Description
`omp_get_num_threads()`	returns number of threads within the team in the parallel region.
`omp_get_thread_num()`	returns unique thread id number from the team of treads.
`omp_in_parallel()`	returns **TRUE** if placed within the parallel region; otherwise returns **FALSE**.
`omp_get_num_procs()`	returns the number of processors that are available to the program

Parallel Region Example – C++

```cpp
#include <iostream>
#include <omp.h>
int main(int argc, char * argv[]){
  int tid;
  std::cout << "Master/Max num threads: " << omp_get_max_threads() << std::endl;
  // Fork a team of threads giving them their own copies of variable 'tid'
#pragma omp parallel private(tid)
  {
    tid = omp_get_thread_num(); // Thread ID or rank
#pragma omp critical
    {
      std::cout << "Thread #" << tid << " out of " << omp_get_num_threads() << std::endl;
    }
  }
  std::cout << "End parallel region" << std::endl;
  return 0;
}
```

OpenMP on UL HPC platform

- Rely on Environment Modules **once** on a computing node
- Comes part of the `intel` or the `foss` toolchains modules

Toolchain	Compilation command (C)	Compilation command (C++)
`toolchain/intel`	`icc -qopenmp [...]`	`icpc -qopenmp [...]`
`toolchain/foss`	`gcc -fopenmp [...]`	`g++ -fopenmp [...]`

```
# Interactive job with 4 threads (-c <threads>). Example for C++ compilation
$ si -c 4
(node)$ export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}
(node)$ module load toolchain/foss
(node)$ g++ -fopenmp -Wall   main.cpp -o ${ULHPC_CLUSTER}_main_foss
(node)$ module load toolchain/intel
(node)$ icpc -qopenmp -Wall  main.cpp -o ${ULHPC_CLUSTER}_main_intel
```

OpenMP with Slurm (docs)

`sbatch/srun/salloc` **option**	**Description**
`--ntasks-per-node=1`	**single** task per node
`-c <thread>`	number of OpenMP threads

```
#!/bin/bash -l    # Single node, threaded (pthreads/OpenMP) application launcher
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 128              # Aion: use all 128 cores, set 28 on iris
#SBATCH -p batch
print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
module load toolchain/foss    # or toolchain/intel

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}
srun /path/to/your/openmp.app
```

Practical Session on OpenMP

Your Turn!

Hands-on Pre-requisites

▶ url ◀ | github | src

- Access to ULHPC facility `ssh`
- Clone/Pull `ULHPC/tutorials` repository `~/git/github.com/ULHPC/tutorials`
- Prepare dedicated directory `~/tutorials/OpenMP-MPI` for this session

```
(access)$> mkdir -p ~/tutorials/OpenMP-MPI
(access)$> cd ~/tutorials/OpenMP-MPI
# create a symbolic link to the reference material
(access)$> ln -s ~/git/github.com/ULHPC/tutorials/parallel/basics ref.d
```

Hands-on: Parallel OpenMP jobs

Hands-on: Work with OpenMP examples

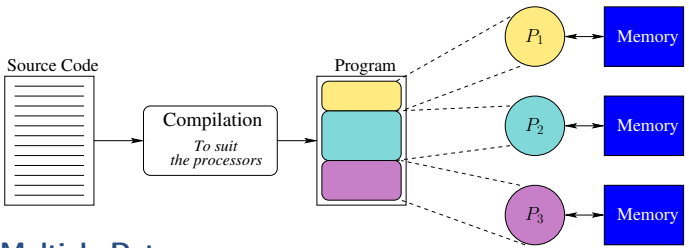
▶ url ◀ | github | src

- **Reserve** an **interactive** job to launch 4 OpenMP threads
- Check and **compile** `src/hello_openmp.c`
  - ↪ against both toolchains `bin/${ULHPC_CLUSTER}_[intel_]hello_openmp`
- **execute** the generated binaries
  - ↪ set `$OMP_NUM_THREADS` `export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}`
- prepare a **launcher script** `runs/launcher.OpenMP.sh`
  - ↪ see pthreads/OpenMP template Launcher from the ULHPC Technical docs
- repeat on a more serious program `src/matrix_mult_openmp.c`
  - ↪ `bin/[intel_]matrix_mult_openmp`

- **Note**: More exercise can be referred **here**

Summary

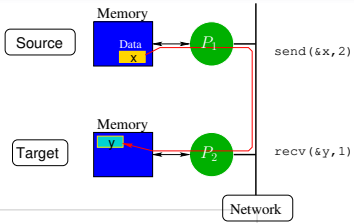
SPMD Programming model



- SPMD: **Simple Program, Multiple Data**
  - ↪ same programs for each processors
    - ✓ executed at independent points
  - ↪ processes identified by a rank
    - ✓ each process knows the piece of code he works on
    - ✓ common in master-worker computations

```
if (my_rank == 0) { /* master */
   //... load input and dispatch ...
} else { /* workers */
   //... wait for data and compute ...
}
```

MPI (Message Passing Interface)



- **Message Passing Model**:
  - ↪ each "processor" runs a process
  - ↪ processes communicate by exchanging messages
    - ✓ *analogy*: **mail**

Message Passing Interface (MPI) Standard

- **Goal**:
  - ↪ **portable**, **efficient** & **flexible** standard for message passing
  - ↪ industry standard
- Reference website `https://www.mpi-forum.org/`
- **Latest version: 4.0** (June 2021) – specifications

Example of MPI program (C)

```
#include <stdio.h>
#include <unistd.h>

int main (int argc, char *argv[]) {
   int id;   // process rank
   int p;    // number of processes
   char hostname[128];
   gethostname(hostname,128);









   return 0;
}
```

Example of MPI program (C)

```
#include <stdio.h>
#include <unistd.h>
#include <mpi.h>
int main (int argc, char *argv[]) {
   int id;   // process rank
   int p;    // number of processes
   char hostname[128];
   gethostname(hostname,128);









   return 0;
}
```

Example of MPI program (C)

```c
#include <stdio.h>
#include <unistd.h>
#include <mpi.h>
int main (int argc, char *argv[]) {
   int id;   // process rank
   int p;    // number of processes
   char hostname[128];
   gethostname(hostname,128);
   MPI_Init(&argc, &argv); // Has to be called first and once









   return 0;
}
```

Example of MPI program (C)

```
#include <stdio.h>
#include <unistd.h>
#include <mpi.h>
int main (int argc, char *argv[]) {
   int id;   // process rank
   int p;    // number of processes
   char hostname[128];
   gethostname(hostname,128);
   MPI_Init(&argc, &argv); // Has to be called first and once




   MPI_Finalize(); // Has to be called last and once
   return 0;
}
```

Example of MPI program (C)

```
#include <stdio.h>
#include <unistd.h>
#include <mpi.h>
int main (int argc, char *argv[]) {
   int id;   // process rank
   int p;    // number of processes
   char hostname[128];
   gethostname(hostname,128);
   MPI_Init(&argc, &argv); // Has to be called first and once
   MPI_Comm_rank(MPI_COMM_WORLD, &id);









   MPI_Finalize(); // Has to be called last and once
   return 0;
}
```

Example of MPI program (C)

```c
#include <stdio.h>
#include <unistd.h>
#include <mpi.h>
int main (int argc, char *argv[]) {
   int id;   // process rank
   int p;    // number of processes
   char hostname[128];
   gethostname(hostname,128);
   MPI_Init(&argc, &argv); // Has to be called first and once
   MPI_Comm_rank(MPI_COMM_WORLD, &id);
   MPI_Comm_size(MPI_COMM_WORLD, &p);









   MPI_Finalize(); // Has to be called last and once
   return 0;
}
```

Example of MPI program (C)

```c
#include <stdio.h>
#include <unistd.h>
#include <mpi.h>
int main (int argc, char *argv[]) {
   int id;   // process rank
   int p;    // number of processes
   char hostname[128];
   gethostname(hostname,128);
   MPI_Init(&argc, &argv); // Has to be called first and once
   MPI_Comm_rank(MPI_COMM_WORLD, &id);
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   if (id == 0) { /* master */
       printf("I am the master: %s\n",hostname);
       fflush(stdout);
   } else { /* worker */
       printf("I am a worker: %s (rank %d/%d)\n",hostname,id,p-1);
       fflush(stdout);
   }
   MPI_Finalize(); // Has to be called last and once
   return 0;
}
```

Example of MPI program (C)

```c
#include <stdio.h>
#include <unistd.h>
#include <mpi.h>
int main (int argc, char *argv[]) {
   int id;   // process rank
   int p;    // number of processes
   char hostname[128];
   gethostname(hostname,128);
   MPI_Init(&argc, &argv); // Has to be called first and once
   MPI_Comm_rank(MPI_COMM_WORLD, &id);
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   if (id == 0) { /* master */
       printf("I am the master: %s\n",hostname);
       fflush(stdout);
   } else { /* worker */
       printf("I am a worker: %s (rank %d/%d)\n",hostname,id,p-1);
       fflush(stdout);
   }
   MPI_Finalize(); // Has to be called last and once
   return 0;
}
```

Example of MPI program (C)

```c
#include <stdio.h>
#include <unistd.h>
#include <mpi.h>
int main (int argc, char *argv[]) {
   int id;   // process rank
   int p;    // number of processes
   char hostname[128];
   gethostname(hostname,128);
   MPI_Init(&argc, &argv); // Has to be called first and once
   MPI_Comm_rank(MPI_COMM_WORLD, &id);
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   if (id == 0) { /* master */
       printf("I am the master: %s\n",hostname);
       fflush(stdout);
   } else { /* worker */
       printf("I am a worker: %s (rank %d/%d)\n",hostname,id,p-1);
       fflush(stdout);
   }
   MPI_Finalize(); // Has to be called last and once
   return 0;
}
```

MPI on UL HPC platform

- Rely on Environment Modules **once** on a computing node
  - ↪ then you can search for MPI suites available: `module avail mpi`
    - ✓ except for Intel MPI that comes part of the `intel` toolchain modules
  - ↪ Official Slurm guide for Open MPI

MPI Suite	`module load...`	**Compiler (C)**	**Compiler (C++)**
Intel MPI	`toolchain/intel`	`mpiicc [...]`	`mpiicpc [...]`
OpenMPI	`mpi/OpenMPI`	`mpicc [...]`	`mpic++ [...]`

MPI on UL HPC platform

- Rely on Environment Modules **once** on a computing node
  - ↪ then you can search for MPI suites available: `module avail mpi`
    - ✓ except for Intel MPI that comes part of the `intel` toolchain modules
  - ↪ Official Slurm guide for Open MPI

MPI Suite	`module load...`	**Compiler (C)**	**Compiler (C++)**
Intel MPI	`toolchain/intel`	`mpiicc [...]`	`mpiicpc [...]`
OpenMPI	`mpi/OpenMPI`	`mpicc [...]`	`mpic++ [...]`

MPI Suite	**Example Compilation command**
Intel MPI	`{mpiicc/mpiicpc} -Wall [-qopenmp] [-xhost] -O2 [...]`
OpenMPI	`{mpicc/mpic++} -Wall [-fopenmp] -O2 [...]`

MPI with Slurm

sbatch/srun/salloc option	Description
-N <N>	number of **distributed** nodes
--ntasks-per-node=<N>	number of **MPI processes** per node
-c 1	(default) set a **single** thread per MPI process
-c <T>	number of OpenMP threads for hybrid runs

- SLURM able to directly launch MPI tasks (**recommended**)
  - ↪ ... and initialize of MPI communications
  - ↪ via Process Management Interface (PMI) and PMIx

```
$> srun -n $SLURM_NTASKS /path/to/mpiprog
```

OpenMPI Launcher Example (docs)

```
#!/bin/bash -l      # OpenMPI Launcher
#SBATCH -N 2
#SBATCH --ntasks-per-node 128    # MPI processes per node - use 28 on iris
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch

print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
module load toolchain/foss
module load mpi/OpenMPI
OPTS=$*

srun -n $SLURM_NTASKS /path/to/your/openmpi.app ${OPTS}
```

Intel MPI Launcher Example (docs)

```
#!/bin/bash -l      # Intel MPI Launcher
#SBATCH -N 2
#SBATCH --ntasks-per-node 128    # MPI processes per node - use 28 on iris
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch

print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
module load toolchain/intel
OPTS=$*

srun -n $SLURM_NTASKS /path/to/your/intelmpi.app ${OPTS}
```

Hands-on: MPI jobs

Your Turn!

Hands-on: MPI

▶ url ◀ | github | src

- **Reserve** an **interactive** job to launch 6 MPI processes
  ↪ across two nodes (2x3), for 30 minutes
- Check and **compile** `src/hello_mpi.c`
  ↪ `bin/${ULHPC_CLUSTER}_{openmpi,intel}_hello_mpi`
- **execute** the generated binaries
- prepare a **launcher script** `runs/launcher.MPI.sh`
  ↪ see MPI template Launcher from the ULHPC Technical docs
- repeat on a more serious program `src/matrix_mult_mpi.c`
  ↪ `bin/{openmpi,intel}_${ULHPC_CLUSTER}_matrix_mult_mpi`

Summary

Hybrid OpenMP+MPI Programs

- **Take the best of both worlds!!!**
  - ↪ distributed (**inter-socket/nodes**) **MPI** processes
  - ↪ local (**intra-socket**) multi-**threads** acceleration of each MPI process by **OpenMP**
- Rely on Environment Modules **once** on a computing node

MPI Suite	**Example Compilation command**
Intel MPI (`ml toolchain/intel`)	`{mpiicc/mpiicpc} -qopenmp -Wall [-xhost] -O2 [...]`
OpenMPI (`ml mpi/OpenMPI`)	`{mpicc/mpic++}  -fopenmp -Wall -O2 [...]`

- adapt `OMP_NUM_THREADS` environment variable from slurm reservation (with default value)
  - ↪ `export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}`
  - ↪ adapt `-c <T>` accordingly: number of OpenMP threads
  - ↪ (**Intel MPI only**): you may want to set `I_MPI_PIN_DOMAIN=omp`

Hybrid OpenMP+MPI with Slurm

`sbatch/srun/salloc` **option**	**Description**
`-N <N>`	number of **distributed** nodes
`--ntasks-per-node=<N>`	number of **MPI processes** per node
`-c <T>`	number of OpenMP threads for hybrid runs

- **Important: Always align resource specs with physical NUMA characteristics**
  - ↪ **Ex (AION): 16 cores per socket, 8 sockets ("physical" CPUs) per node** (128c/node)
  - ↪ `[-N <N>] --ntasks-per-node <8n> --ntasks-per-socket <n> -c <thread>`
    - ✓ **Total**: `<N>`×8×`<n>` tasks, each on `<thread>` threads
    - ✓ **Ensure** `<n>`×`<thread>`= 16 on aion
    - ✓ Ex: `-N 2 --ntasks-per-node 32 --ntasks-per-socket 4 -c 4` (**Total**: 64 tasks)

Hybrid OpenMP+MPI with Slurm

`sbatch/srun/salloc` **option**	**Description**
`-N <N>`	number of **distributed** nodes
`--ntasks-per-node=<N>`	number of **MPI processes** per node
`-c <T>`	number of OpenMP threads for hybrid runs

- **Important: Always align resource specs with physical NUMA characteristics**
  - ↪ **Ex (IRIS): 14 cores per socket, 2 sockets ("physical" CPUs) per node** (28c/node)
  - ↪ `[-N <N>] --ntasks-per-node <2n> --ntasks-per-socket <n> -c <thread>`
    - ✓ **Total**: `<N>`×2×`<n>` tasks, each on `<thread>` threads
    - ✓ **Ensure** `<n>`×`<thread>`= 14 on iris
    - ✓ Ex: `-N 2 --ntasks-per-node 4 --ntasks-per-socket 2 -c 7` (**Total**: 8 tasks)

Hybrid OpenMP+MPI Launcher Example (docs)

```
#!/bin/bash -l      # Multi-node hybrid application IntelMPI+OpenMP launcher
#SBATCH -N 2
#SBATCH --ntasks-per-node   8     # MPI processes per node - use 2 on iris
#SBATCH --ntasks-per-socket 1     # MPI processes per [virtual] processor
#SBATCH -c 16                     # OpenMP threads per MPI process - use 14 on iris
#SBATCH --time=0-01:00:00
#SBATCH -p batch

print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
module load mpi/OpenMPI    # or toolchain/intel
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}
OPTS=$*

srun -n $SLURM_NTASKS /path/to/your/parallel-hybrid-app ${OPTS}
```

Hands-on 3: Hybrid OpenMP+MPI

Your Turn!

Hands-on 3

ulhpc-tutorials.rtfd.io/en/latest/parallel/basics/#hybrid-openmpmpi-programs

- **Reserve** an **interactive** job to launch
  - ↪ 2 MPI processes (on 2 nodes) / 4 OpenMP threads
- Check and **compile** `src/hello_hybrid.c`
  - ↪ `bin/{openmpi,intel,mvapich2}_hello_hybrid`
- **execute** the generated binaries
  - ↪ set `$OMP_NUM_THREADS`
  - ↪ compute `$NPERHOST`
- prepare a **launcher script** `runs/launcher.hybrid.sh`

Summary

HPC Interconnect Benchmarking

OSU Micro-Benchmarks Tutorial

ulhpc-tutorials.rtfd.io/en/latest/parallel/mpi/OSU_MicroBenchmarks/

- **Objective**: build and run **OSU Micro-benchmarks 5.8** for each considered MPI suit
  - ↪ Reference (Infiniband) interconnect benchmark from MVAPICH team
  - ↪ Focusing on (only) two **one-sided MPI benchmarks**:
    - ✓ `osu_get_latency` - Latency Test
    - ✓ `osu_get_bw` - Bandwidth Test

Hands-on Pre-requisites

▶ url ◀ | github | src

- Clone/Pull `ULHPC/tutorials` repository `~/git/github.com/ULHPC/tutorials`
- Prepare dedicated directory `~/tutorials/OSU-MicroBenchmarks` for this session

Practical Session: OSU Micro-Benchmarks (MPI)

Your Turn!

Hands-on Build and Run OSU Micro-benchmarks

▶ url ◀ | github | src

- Fetch and uncompress the sources
- Compilation based on **Intel MPI** and **Open MPI**
  - ↪ Based on Autotools/Automake
  - ↪ **Rely on** `configure --prefix=$(pwd)` to state where to install
  - ↪ sometimes being in a separate build directory as recommended raise issues!
    - ✓ Ex: *osu_util.h: No such file or directory* (missing header)
    - ✓ then you have to play with `CFLAGS=-I<path>`
- Pepare a **launcher script** `runs/launcher.OSU.sh`
- Benchmark execution in interactive and passive
  - ↪ MPI application directory: `libexec/osu-micro-benchmarks/mpi/one-sided/`

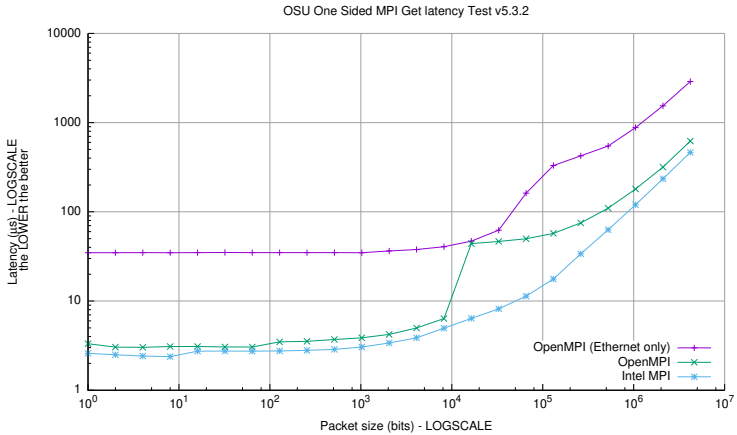
Build and Run OSU Micro-Benchmarks

```
$> configure --prefix=<path>; make && make install
```

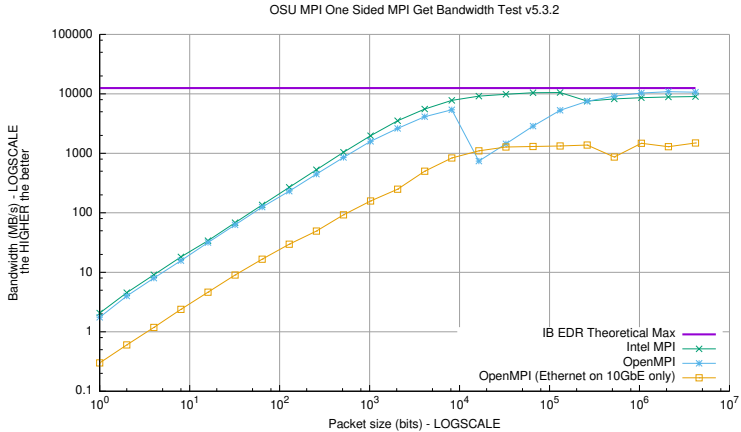
```
$ module load mpi/OpenMPI # or toolchain/intel
$ mkdir <builddir> && cd <builddir>
$ ../src/configure [CC=<compiler>] [CFLAGS=-I<srcdir>/util] --prefix=$(pwd)
$ make && make install
```

```
$ cd ~/tutorials/OSU-MicroBenchmarks/runs
$ sbatch ./launcher-OSU.intel.sh
$ sbatch ./launcher-OSU.intel.sh intel
```

Iris IB Network Perf. (OSU Micro-benchmarks)



Iris IB Network Perf. (OSU Micro-benchmarks)



Thank you for your attention...

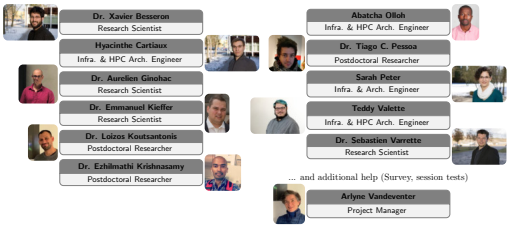
Questions?

ulhpc-tutorials.rtfd.io/en/latest/parallel/basics/

High Performance Computing @ Uni.lu

University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu

Uni.lu HPC School 2021 Contributors



hpc.uni.lu