Uni.lu HPC School 2019

PS5a: Scalable Science I: 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:	http://hpc.uni.lu/hpc-school/
PS5a 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
  - ↪ MVAPICH2
- **Build** and **run** OpenMP and/or MPI code
  - ↪ interactive and passive (through **launcher**) jobs

- **Test cases** on reference parallel and distributed **benchmarks**:
  - ↪ OSU micro-benchmarks:
    - ✓ measure the performances of various MPI operations
  - ↪ High-Performance Linpack (HPL)
  - ↪ Hybrid High Performance Conjugate Gradients (HPCG)

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;

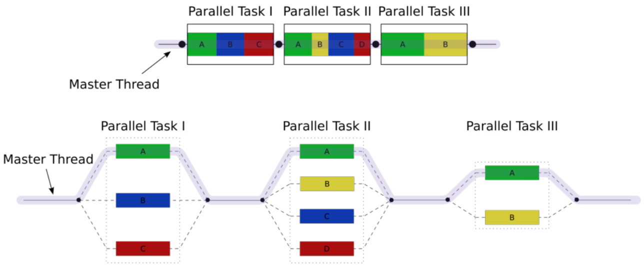
OpenMP

`https://www.openmp.org/`

- Reference website
- **Latest version: 5.0** (Nov 2018) – specifications

OpenMP Programming Model

- Explicit (not automatic) programming model
  - ↪ may mean taking a serial program & insert compiler directives. . . .
- **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.



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
`toolchain/intel`	`icc -qopenmp [...]`
`toolchain/foss`	`gcc -fopenmp [...]`

Reservation

- You will probably want to compile/test within an **interactive job**
  ↪ **Ex**: 4 threads (core) on 1 node

```
# SLURM -- Iris cluster
(access)$> srun -p interactive --ntasks-per-node=1 -c 4 --pty bash

# OAR --  gaia, chaos cluster
(access)$> oarsub -I -l nodes=1/core=4,walltime=4
```

OpenMP with OAR

- Reservation has to match the required number of OpenMP threads:
  ↪ **Ex**: `oarsub -l nodes=1/core=4[...]`

```
#!/bin/bash -l
#OAR -l nodes=1/core=4,walltime=1

export OMP_NUM_THREADS=$(cat $OAR_NODEFILE| wc -l)

# Use the RESIF build modules of the UL HPC platform
if [ -f  /etc/profile ]; then
   .  /etc/profile
fi
# Load the {intel | foss} toolchain and whatever module(s) you need
module purge
module load toolchain/intel # or toolchain/foss

path/to/your/openmp_program
```

OpenMP with Slurm

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

```
#!/bin/bash -l
#SBATCH --ntasks-per-node=1     # more explicit than '-n 1'
#SBATCH -c 28        # number of CPU cores / OpenMP threads per task
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch
if [ -f  /etc/profile ]; then
   .  /etc/profile
fi
# Load the {intel | foss} toolchain and whatever module(s) you need
module purge
module load toolchain/intel # or toolchain/foss
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
srun /path/to/your/openmp_program
```

Hands-on 1: Parallel OpenMP jobs

Your Turn!

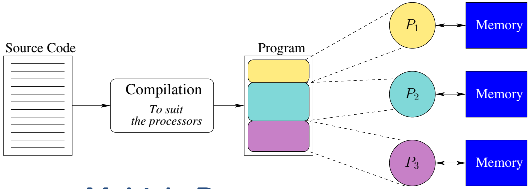
Hands-on 1

ulhpc-tutorials.rtfd.io/en/latest/parallel/basics/#parallel-openmp-jobs

- **Reserve** an **interactive** job to launch 4 OpenMP threads
- Check and **compile** `src/hello_openmp.c`
  - ↪ against both toolchains `bin/[intel_]hello_openmp`
- **execute** the generated binaries
  - ↪ set `$OMP_NUM_THREADS`
- prepare a **launcher script** `runs/launcher.OpenMP.sh`
- repeat on a more serious program `src/matrix_mult_openmp.c`
  - ↪ `bin/[intel_]matrix_mult_openmp`
- (eventually) OpenMP DataRaceBench suite

Summary

SPMD Programming model

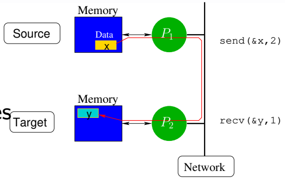


- SPMD: **Simple Program, Multiple Data**
  - ↪ same programs for each processors
    - ✓ executed at independent points
  - ↪ processes identified by a rank
    - ✓ each process knows the piece of code he works on
    - ✓ common in master-worker computations

```
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: 3.1** (June 2015) – specifications

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`

MPI Suite	Version	`module load...`	Compiler
Intel MPI	18.0.1	`toolchain/intel`	C: `mpiicc`; C++: `mpiicpc`
OpenMPI	2.1.3	`mpi/OpenMPI`	C: `mpicc`; C++: `mpic++`
MVAPICH2	2.3a	`mpi/MVAPICH2`	C: `mpicc`; C++: `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`

MPI Suite	Version	`module load...`	Compiler
Intel MPI	18.0.1	`toolchain/intel`	C: `mpiicc`; C++: `mpiicpc`
OpenMPI	2.1.3	`mpi/OpenMPI`	C: `mpicc`; C++: `mpic++`
MVAPICH2	2.3a	`mpi/MVAPICH2`	C: `mpicc`; C++: `mpic++`

MPI Suite	Compilation command (in C)
Intel MPI	`mpiicc -Wall [-qopenmp] [-xhost] -O2 [...]`
OpenMPI	`mpicc  -Wall [-fopenmp] -O2 [...]`
MVAPICH2	`mpicc  -Wall [-fopenmp] -O2 [...]`

Reservation

- You will probably want to compile/test within an **interactive job**
  ↪ **Ex**: 1 core on two **different** nodes

```
# SLURM -- Iris cluster
(access)$> srun -p interactive -N 2 --ntasks-per-node 1 --pty bash

# OAR --  gaia, chaos cluster
(access)$> oarsub -I -l nodes=2/core=1,walltime=4
```

- Then you can search for MPI suites available

```
(node)$> module avail mpi
```

MPI with OAR

- Reservation has to match the required number of MPI processes:
  ↪ **Ex**: `oarsub -l nodes=2/core=12[...]`
- Running MPI code using traditional `mpirun`

```
# Intel MPI
$> module load toolchain/intel
# ONLY on moonshot node have no IB card: export I_MPI_FABRICS=tcp
$> mpirun -hostfile $OAR_NODEFILE ...
```

MPI with OAR

- Reservation has to match the required number of MPI processes:
  ↪ **Ex**: `oarsub -l nodes=2/core=12[...]`
- Running MPI code using traditional `mpirun`

```
# Intel MPI
$> module load toolchain/intel
# ONLY on moonshot node have no IB card: export I_MPI_FABRICS=tcp
$> mpirun -hostfile $OAR_NODEFILE ...
```

```
# OpenMPI
$> module load mpi/OpenMPI
$> mpirun -hostfile $OAR_NODEFILE -x PATH -x LD_LIBRARY_PATH ...
```

MPI with OAR

- Reservation has to match the required number of MPI processes:
  ↪ **Ex**: `oarsub -l nodes=2/core=12[...]`
- Running MPI code using traditional `mpirun`

```
# Intel MPI
$> module load toolchain/intel
# ONLY on moonshot node have no IB card: export I_MPI_FABRICS=tcp
$> mpirun -hostfile $OAR_NODEFILE ...
```

```
# OpenMPI
$> module load mpi/OpenMPI
$> mpirun -hostfile $OAR_NODEFILE -x PATH -x LD_LIBRARY_PATH ...
```

```
# MVAPICH2
$> module load mpi/MVAPICH2
$>-launcher ssh -launcher-exec /usr/bin/oarsh -f $OAR_NODEFILE ...
```

MPI launchers (OAR) Github

```
$> oarsub -S <scriptname>        # -S: interpret #OAR comments
```

```
#!/bin/bash -l
#OAR -l nodes=2/core=6,walltime=1
if [ -f  /etc/profile ]; then
   .  /etc/profile
fi
# Load the intel toolchain and whatever MPI module you need
module purge && module load toolchain/intel # or mpi/{OpenMPI|MVAPICH2}
# ONLY on moonshot node that have no IB card: export I_MPI_FABRICS=tcp

### Intel MPI
mpirun -hostfile $OAR_NODEFILE mpi_program
### OpenMPI
mpirun -hostfile $OAR_NODEFILE -x PATH -x LD_LIBRARY_PATH mpi_program
### MVAPICH2
mpirun -launcher ssh -launcher-exec /usr/bin/oarsh \
       -f $OAR_NODEFILE mpi_program
```

MPI with Slurm

sbatch/srun 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) [v2]

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

MPI launcher (Slurm) Docs

```
$> sbatch [-p batch] <scriptname>
```

```
#!/bin/bash -l
#SBATCH -N 2                      # Use 2 nodes
#SBATCH --ntasks-per-node=28  # Number of MPI process per node
#SBATCH -c 1      # threads per MPI process (1 unless hybrid code)
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

if [ -f  /etc/profile ]; then
   .  /etc/profile
fi
# Load the intel toolchain and whatever MPI module you need
module purge
module load toolchain/intel # or mpi/{OpenMPI|MVAPICH2}
# export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun -n $SLURM_NTASKS /path/to/your/mpi_program
```

Hands-on 2: MPI jobs

Your Turn!

Hands-on 2

ulhpc-tutorials.rtfd.io/en/latest/parallel/basics/#paralleldistributed-mpi-jobs

- **Reserve** an **interactive** job to launch 6 MPI processes
  - ↪ across two nodes (2x3), for 30 minutes
- Check and **compile** `src/hello_mpi.c`
  - ↪ `bin/{openmpi,intel,mvapich2}_hello_mpi`
- **execute** the generated binaries
- prepare a **launcher script** `runs/launcher.MPI.sh`
- repeat on a more serious program `src/matrix_mult_mpi.c`
  - ↪ `bin/{openmpi,intel,mvapich2}_matrix_mult_mpi`

Summary

Hybrid OpenMP+MPI Programs

MPI Suite	`module load...`	Compilation command (C)
Intel MPI	`toolchain/intel`	`mpiicc -Wall -qopenmp [-xhost] -O2 [...]`
OpenMPI	`mpi/OpenMPI`	`mpicc  -Wall -fopenmp -O2 [...]`
MVAPICH2	`mpi/MVAPICH2`	`mpicc  -Wall -fopenmp -O2 [...]`

- adapt (and share) `OMP_NUM_THREADS` environment variable
- **(Slurm only)**: adapt `-c <T>`: number of OpenMP threads
- **(OAR only)**: you have to take the following elements into account:
  - ↪ compute accurately MPI processes per node `<PPN>`
  - ↪ pass it to `mpirun`:
    - ✓ OpenMPI: `mpirun -npernode <PPN> -np <N>`
    - ✓ Intel MPI: `mpirun -perhost <PPN>  -np <N>`
    - ✓ MVAPICH2: `mpirun -ppn <PPN>     -np <N>`
  - ↪ (**Intel MPI only**): `I_MPI_PIN_DOMAIN=omp`
  - ↪ (**MVAPICH2 only**) `MV2_ENABLE_AFFINITY=0`

Hybrid OpenMP+MPI with OAR

```
### Intel MPI
mpirun -perhost ${NPERNODE:=1} -np ${NP} \
       -genv OMP_NUM_THREADS=${OMP_NUM_THREADS} \
       -genv I_MPI_PIN_DOMAIN=omp \
       -hostfile $OAR_NODEFILE  hybrid_program
```

Hybrid OpenMP+MPI with OAR

```
### Intel MPI
mpirun -perhost ${NPERNODE:=1} -np ${NP} \
       -genv OMP_NUM_THREADS=${OMP_NUM_THREADS} \
       -genv I_MPI_PIN_DOMAIN=omp \
       -hostfile $OAR_NODEFILE  hybrid_program
```

```
### OpenMPI
mpirun -npernode ${NPERNODE:=1} -np ${NP} \
       -x OMP_NUM_THREADS -x PATH-x LD_LIBRARY_PATH \
       -hostfile $OAR_NODEFILE  hybrid_program
```

Hybrid OpenMP+MPI with OAR

```
### Intel MPI
mpirun -perhost ${NPERNODE:=1} -np ${NP} \
       -genv OMP_NUM_THREADS=${OMP_NUM_THREADS} \
       -genv I_MPI_PIN_DOMAIN=omp \
       -hostfile $OAR_NODEFILE  hybrid_program
```

```
### OpenMPI
mpirun -npernode ${NPERNODE:=1} -np ${NP} \
       -x OMP_NUM_THREADS -x PATH-x LD_LIBRARY_PATH \
       -hostfile $OAR_NODEFILE  hybrid_program
```

```
### MVAPICH2
export MV2_ENABLE_AFFINITY=0
mpirun -ppn ${NPERNODE:=1} -np ${NP} \
       -genv OMP_NUM_THREADS=${OMP_NUM_THREADS} \
       -launcher ssh -launcher-exec /usr/bin/oarsh \
       -f $MACHINEFILE  hybrid_program
```

Hybrid OpenMP+MPI with Slurm

```
#!/bin/bash -l
#SBATCH -N 2                   # Use 2 nodes
#SBATCH --ntasks-per-node=1    # Number of MPI process per node
#SBATCH -c 4              # Number of OpenMP threads per MPI process
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

if [ -f  /etc/profile ]; then
   .  /etc/profile
fi
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

# Load the intel toolchain and whatever MPI module you need
module purge
module load toolchain/intel    # or mpi/{OpenMPI|MVAPICH2}
# export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun -n $SLURM_NTASKS /path/to/your/hybrid_program
```

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 Instructions

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

- **Pre-requisites**: get an interactive job for compilation

```
### Iris cluster
(access)$> srun -p interactive -N 2 --ntasks-per-node 1 --pty bash
# aliased/short version: 'si -N 2 --ntasks-per-node 1'
# best-effort mode :    append '--qos qos-best-effort'
# within a reservation: append '--reservation <name>'
```

```
### Gaia, chaos cluster
(access)$> oarsub -I -l nodes=2/core=1,walltime=4
# best-effort mode :    append '-t besteffort'
# within a reservation: append '-t inner=<containerID>'
```

OSU micro-benchmarks

http://mvapich.cse.ohio-state.edu/benchmarks/

- We will build **version 5.5 of the** **OSU micro-benchmarks**
- Focusing on (only) two one-sided benchmarks:
  - ↪ `osu_get_latency` - Latency Test
  - ↪ `osu_get_bw` - Bandwidth Test

Building the Benchmarks

Your Turn!

Hands-on 4

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

- Get the sources
- Uncompress them
- Compilation based on
  - ↪ **Intel MPI**
  - ↪ **Open MPI**
  - ↪ **Open MPI** over Ethernet interface
    - ✓ highlight performance drops compared to Infiniband

Building the Benchmarks

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

- Based on Autotools/Automake
  - ↪ **Rely on `--prefix=$(pwd)`** to state where to install
  - ↪ sometimes being in a separate build directory raise issues!
    - ✓ Ex: *osu_util.h: No such file or directory* (missing header)
    - ✓ then you have to play with `CFLAGS=-I<path>` (or `LDPATH`)

```
$> mkdir <builddir> && cd <builddir>
$> ../src/configure [CC=<compiler>] --prefix=$(pwd)
$> make && make install
```

Building the Benchmarks

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

- Based on Autotools/Automake
  - ↪ **Rely on `--prefix=$(pwd)`** to state where to install
  - ↪ sometimes being in a separate build directory raise issues!
    - ✓ Ex: *osu_util.h: No such file or directory* (missing header)
    - ✓ then you have to play with `CFLAGS=-I<path>` (or `LDPATH`)

```
$> mkdir <builddir> && cd <builddir>
$> ../src/configure [CC=<compiler>] --prefix=$(pwd)
$> make && make install
```

- Now common to have CMake based software

```
$> cmake ../src/
$> make && make install
```

Running the Benchmarks

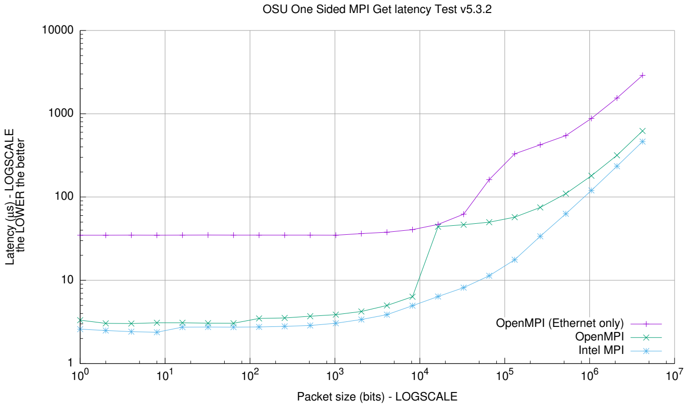
Your Turn!

- Build directory:
  `libexec/osu-micro-benchmarks/mpi/one-sided/`
- Prepare a batch launcher
  ↪ copy and adapt the default SLURM launcher
- Run it in batch mode

```
$> cd ~/tutorials/OSU-MicroBenchmarks/runs
### On iris
$> sbatch ./launcher-OSU.intel.sh osu_get_bw
$> sbatch ./launcher-OSU.intel.sh osu_get_latency
### On gaia, chaos
$> oarsub -S ./launcher-OSU.intel.sh
```

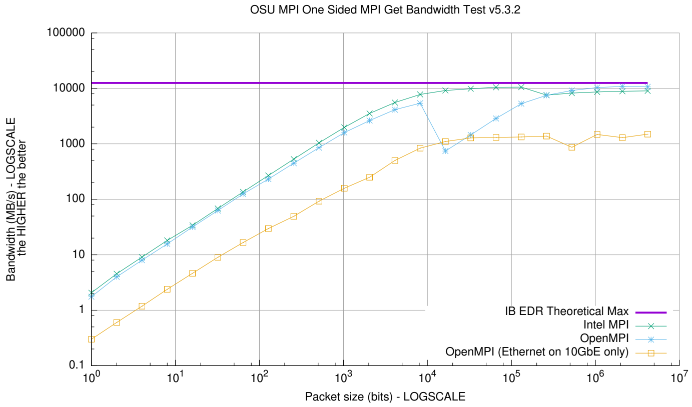
Interconnect Performances

- Based on OSU Micro-benchmarks



Interconnect Performances

- Based on OSU Micro-benchmarks



Summary

High-Performance Linpack (HPL)

HPL Instructions

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

- **Pre-requisites**: get an interactive job for compilation
  - ↪ **Q**: justify the difference with the job request made for OSU.

```
### Iris cluster
(access)$> srun -p interactive -n 14 --pty bash
# aliased/short version: 'si -n 14'
# best-effort mode :     append '--qos qos-best-effort'
# within a reservation: append '--reservation <name>'
```

```
### Gaia, chaos cluster
(access)$> oarsub -I -l nodes=1/core=6,walltime=4
# best-effort mode :     append '-t besteffort'
# within a reservation: append '-t inner=<containerID>'
```

High-Performance Linpack (HPL)

http://www.netlib.org/benchmark/hpl/

- Portable implem. of High-Performance Linpack (HPL) Benchmark
  - ↪ for Distributed-Memory Computers
  - ↪ *reference* benchmark for ranking the Top500 list
- We will build **version 2.2 of the HPL**
  - ↪ Focusing (only) on Intel MPI+MKL build
  - ↪ **Pre-requisites**:
    - ✓ clone ULHPC/tutorials and ULHPC/launcher-scripts repositories
    - ✓ preparing your working directory

Building HPL

Your Turn!

Hands-on 4

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

- Get the sources
- Uncompress them
- Compilation based on the **Intel MPI** suit
  $\hookrightarrow$ Prepare and adapt `src/hpl-2.2/Make.intel64`
- Compile it !

Building HPL

```
$> cd ~/tutorials/HPL/src/hpl-2.2
$> cp setup/Make.Linux_Intel64 Make.intel64
$> vim Make.intel64
# [...] change TOPdir and MP{dir,inc,lib} (at least)
$> make arch=intel64 clean_arch_all
$> make arch=intel64
```

Preparing the HPL Benchmark Run

Your Turn!

- Build directory: `bin/intel64`
- Prepare a batch launcher
  - ↪ copy and adapt the default SLURM launcher
- Prepare an input `HPL.dat` file
  - ↪ use Tuning HPC Online for some default settings

- **Main HPL parameters constraints**
  - ↪ `PxQ = <nodes>*<cores> = $SLURM_NTASKS`
  - ↪ Problem size: `N` (to be as large as possible)
    - ✓ $\mathtt{N} = \alpha\sqrt{\#nodes * RAM * 1024}$ where RAM is expressed in GiB
  - ↪ NB: depends on processor architecture (Ex: Intel MKL notes)
    - ✓ `NB = 192 on iris cluster`

Example HPL.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
24650        Ns
1            # of NBs
192          NBs
0            PMAP process mapping (0=Row-,1=Column-major)
2            # of process grids (P x Q)
2 4          Ps
14 7         Qs
[...]
```

- Targeting 1 node in this case on 2 sets of parameters (`PxQ` = 28)

	N	NB	P	Q
Run 1	24650	192	2	14
Run 2	24650	192	4	7

HPL Benchmark [batch] Runs

- Adapt the default SLURM launcher
- Run it

```
$> cd ~/tutorials/HPL/runs
$> cp ../ref.ulhpc.d/HPL.dat .
### On iris
$> sbatch ./launcher-HPL.intel.sh
### On gaia, chaos
$> oarsub -S ./launcher-HPL.intel.sh
```

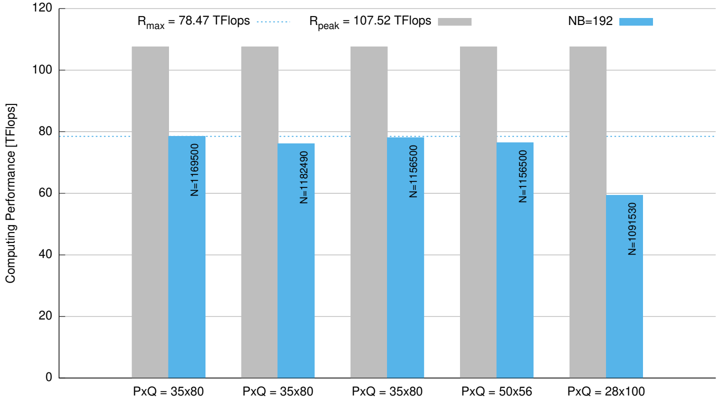
- Grab the HPL results from the output logs

```
# T/V      N        NB       P     Q    Time       Gflops
$> grep WR slurm-2758.out
WR11C2R4  24650    192      2    14    13.51     7.392e+02
WR11C2R4  24650    192      4     7    12.69     7.869e+02
```

Computing Performances / HPL

- Based on High-Performance Linpack (HPL)
  ↪ reference benchmark for Top 500

HPL 2.2 - ULHPC iris cluster - 100 Nodes



Thank you for your attention...

Questions?

http://hpc.uni.lu

High Performance Computing @ uni.lu

Prof. Pascal Bouvry
Dr. Sebastien Varrette
Valentin Plugaru
Sarah Peter
Hyacinthe Cartiaux
Clement Parisot
Dr. Fréderic Pinel
Dr. Emmanuel Kieffer

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