UL HPC School 2017

PS4: HPC workflow with MPI Parallel/Distributed jobs (OSU Microbenchmarks, HPL)

UL High Performance Computing (HPC) Team

S. Varrette

University of Luxembourg (UL), Luxembourg
http://hpc.uni.lu

Shared Etherpad Notes:

https://goo.gl/kfHYkN

uni.lu
UNIVERSITÉ DU
LUXEMBOURG

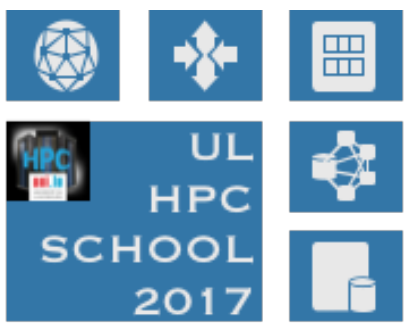
Latest versions available on Github:

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

UL HPC School: http://hpc.uni.lu/hpc-school/

PS4 tutorial sources:

https://github.com/ULHPC/tutorials/tree/devel/advanced/OSU_MicroBenchmarks



Summary

Main Objectives of this Session

- See how to use the MPI suit available on the UL HPC platform:
  - ↪ Intel MPI and the Intel MKL
  - ↪ OpenMPI
  - ↪ MVAPICH2
    - ✓ MPI-3 over OpenFabrics-IB, Omni-Path, OpenFabrics-iWARP, PSM, and TCP/IP
- Build and run MPI code (through the provided launcher scripts)
- Test case on reference parallel MPI benchmarks:
  - ↪ OSU micro-benchmarks:
    - ✓ measure the performances of various MPI operations
  - ↪ High-Performance Linpack (HPL)

MPI on UL HPC platform

- Rely on Environment Modules **once** on a computing node
- 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
```

Available MPI Modules

MPI suite	Module to load	`CC/CXX` (compilation)
Intel MPI	`module load toolchain/intel`	`mpiicc`
OpenMPI	`module load mpi/OpenMPI`	`mpicc`
MVAPICH2	`module load mpi/MVAPICH2`	`mpicc`
...		

Running MPI code (OAR)

- Normally and traditionally using `mpirun`
  - ↪ of course after loading the appropriate module

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

Running MPI code (OAR)

- Normally and traditionally using `mpirun`
  - ↪ of course after loading the appropriate module

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

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

Running MPI code (OAR)

- Normally and traditionally using `mpirun`
  - ↪ of course after loading the appropriate module

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

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

```
# MVAPICH2
(node)$> module load mpi/MVAPICH2
(node)$> mpirun -f $OAR_NODEFILE  ...
```

Running MPI code (SLURM)

- SLURM able to directly launch MPI tasks
  - ↪ ... and initialize of MPI communications
  - ↪ via Process Management Interface (PMI) [v2]

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

MPI launchers (OAR)

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

- Our launcher scripts on Github: https://github.com/ULHPC/launcher-scripts
  - ↪ see in particular our MPI generic launcher for OAR
  - ↪ Do not hesitate to contribute!

```
#! /bin/bash
#OAR -l nodes=2/core=1,walltime=1
#OAR -n MyMPIJob
# Prepare UL HPC modules
if [ -f  /etc/profile ]; then
.  /etc/profile
fi

module load toolchain/intel
mpirun -hostfile $OAR_NODEFILE /path/to/mpiprog <ARGS>
```

MPI launchers (SLURM)

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

Documentation

https://hpc.uni.lu/users/docs/slurm_launchers.html

```
#!/bin/bash -l
#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
srun -n $SLURM_NTASKS /path/to/mpiprog
```

Summary

HPC Interconnect Benchmarking

OSU Micro-Benchmarks Instructions

http://ulhpc-tutorials.readthedocs.io/en/latest/advanced/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 5.4 of the OSU micro-benchmarks**
- Focusing on (only) two one-sided benchmarks:
  - ↪ `osu_get_latency` - Latency Test
  - ↪ `osu_get_bw` - Bandwidth Test
- **Pre-requisites**:
  - ↪ clone ULHPC/tutorials and ULHPC/launcher-scripts repositories
  - ↪ Preparing your working directory

```
$> mkdir -p ~/git/ULHPC && cd  ~/git/ULHPC
$> git clone https://github.com/ULHPC/launcher-scripts.git
$> git clone https://github.com/ULHPC/tutorials.git
# Preparing  your working directory
$> mkdir -p ~/tutorials/OSU-MicroBenchmarks
$> cd ~/tutorials/OSU-MicroBenchmarks
 # Keep a symlink to the reference tutorial
$> ln -s ~/git/ULHPC/tutorials/advanced/OSU_MicroBenchmarks ref.ulhpc
```

Building the Benchmarks

Your Turn!

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

Compilation Best Practices

- **Try to build your software:**
  - ↪ within directories you own (i.e. under `$HOME`), **not** `/usr/local`
  - ↪ and within a dedicated build directory

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

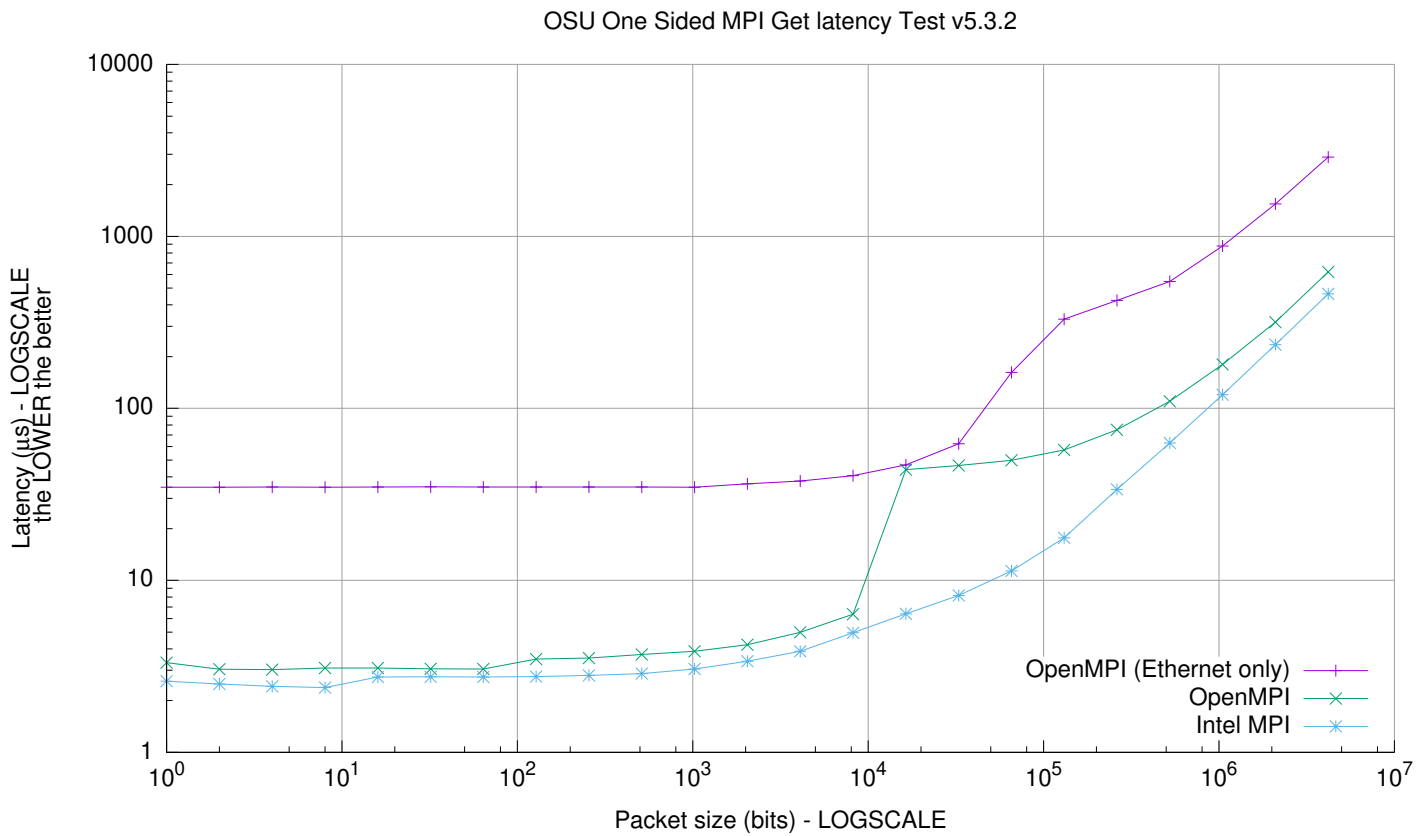
Your Turn!

- 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
```

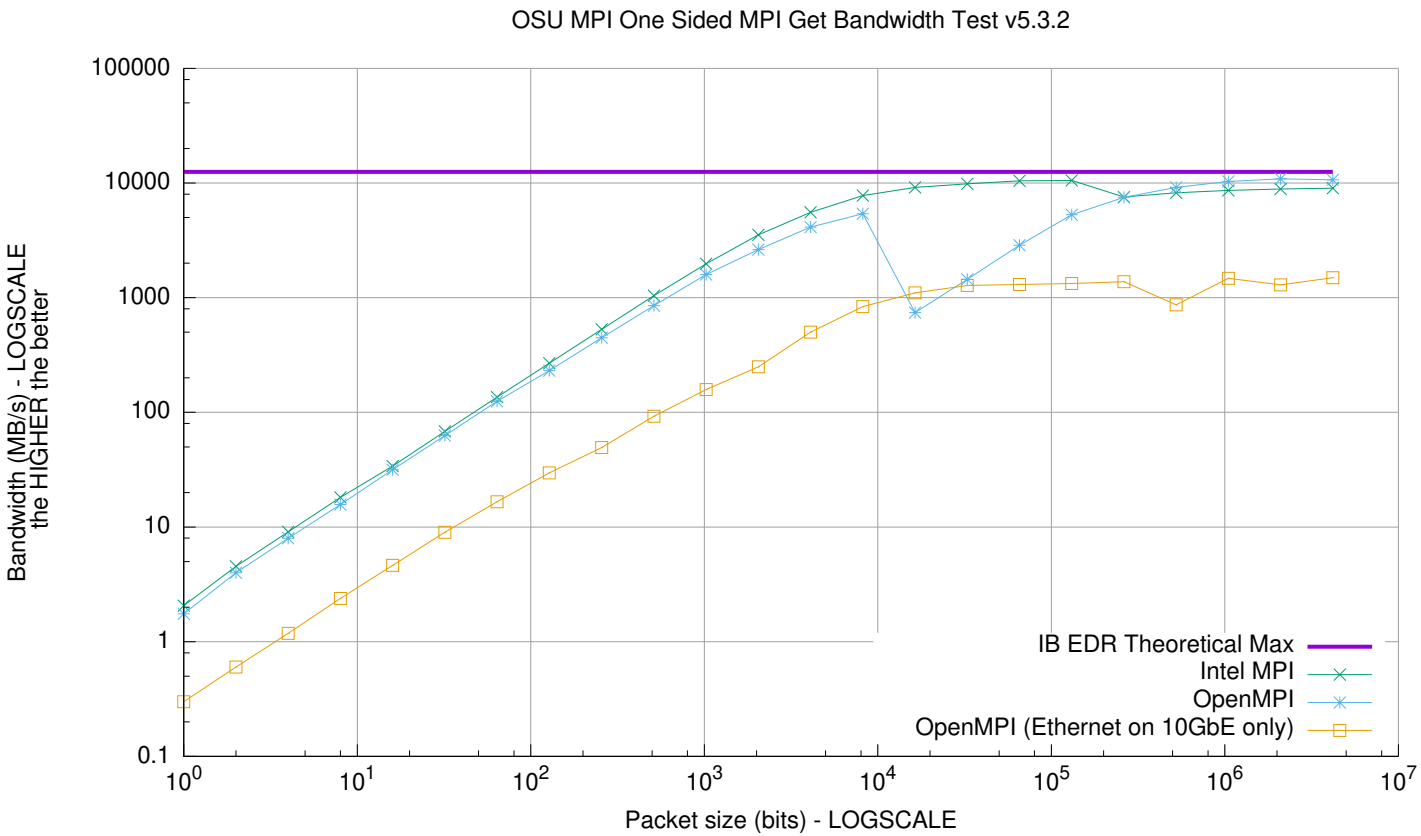
Interconnect Performances

- Based on OSU Micro-benchmarks



Interconnect Performances

- Based on OSU Micro-benchmarks



Summary

High-Performance Linpack (HPL)

HPL Instructions

http://ulhpc-tutorials.readthedocs.io/en/latest/advanced/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

```
$> mkdir -p ~/git/ULHPC && cd  ~/git/ULHPC
$> git clone https://github.com/ULHPC/launcher-scripts.git
$> git clone https://github.com/ULHPC/tutorials.git
# Preparing  your working directory
$> mkdir -p ~/tutorials/HPL &&  cd ~/tutorials/HPL
# Keep a symlink to the reference tutorial
$> ln -s ~/git/ULHPC/tutorials/advanced/HPL ref.ulhpc.d
```

Building HPL

Your Turn!

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

```
$> 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)
    - ✓ $\texttt{N} = \alpha\sqrt{\#nodes * RAM * 1024}$ 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

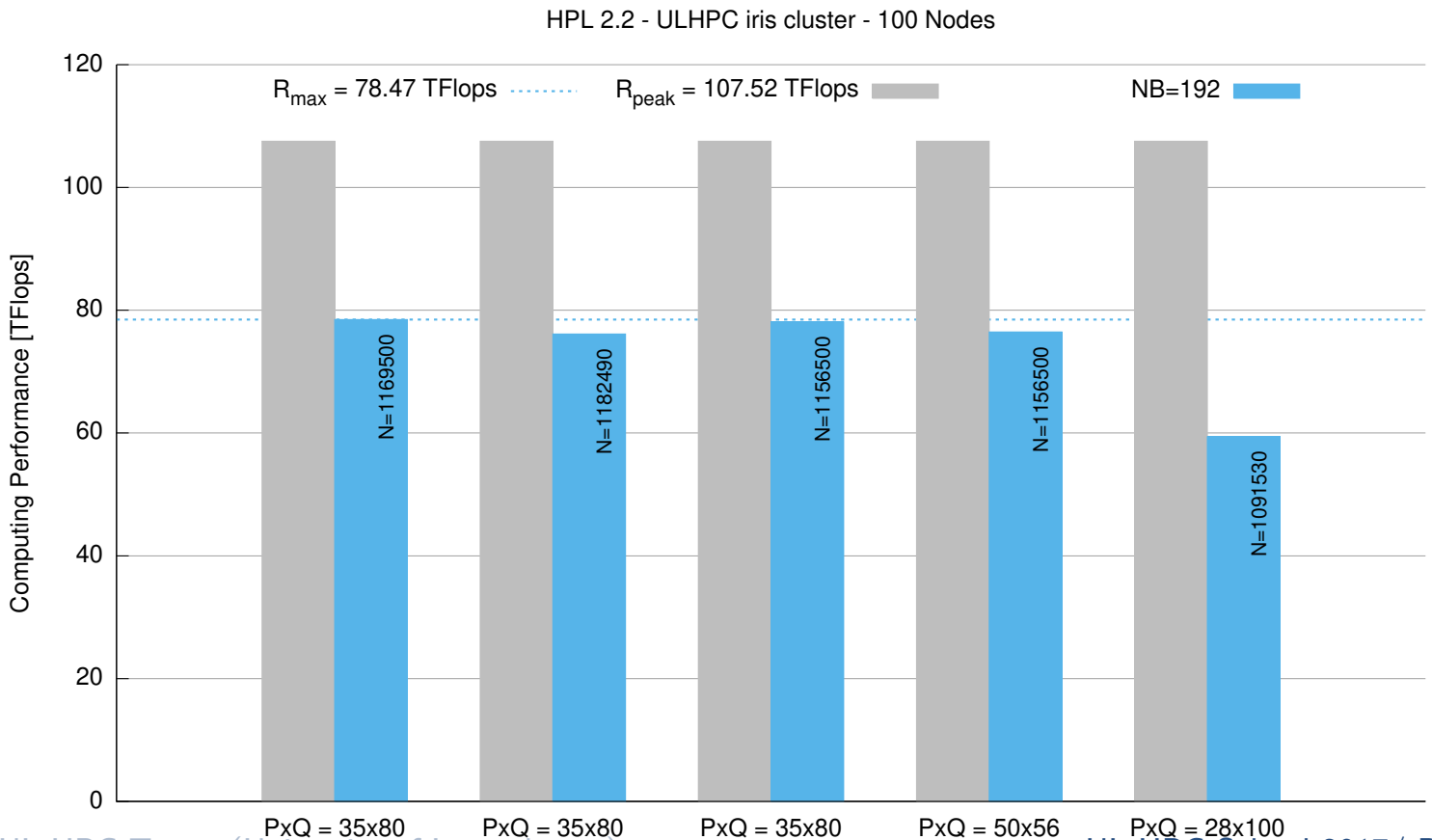
```
$> cd ~/tutorials/HPL/runs
$> cp ../ref.ulhpc.d/HPL.dat .
### On iris
$> sbatch ./launcher-HPL.intel.sh
### On gaia, chaos
$> oarsub -S ./launcher-HPL.intel.sh
```

- 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



Questions?

http://hpc.uni.lu

High Performance Computing @ UL
Prof. Pascal Bouvry
Dr. Sebastien Varrette & the UL HPC Team
(V. Plugaru, S. Peter, H. Cartiaux & C. Parisot)
University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu