How to run on LUMI

Rasmus Kronberg | Running GROMACS efficiently on LUMI workshop 2024

Introduction

- When you login to LUMI, you end up on one of the **shared login nodes**
  - Intended for management tasks, e.g. compiling software, preparing and submitting jobs, moving data and **light** pre-/post-processing tasks
  - *Do not run heavy tasks on the login nodes, these will be killed without warning!*
- `ssh <username>@lumi.csc.fi`
- ...or open “Login node shell” at `www.lumi.csc.fi`

Slurm

- LUMI uses the Slurm resource management system for scheduling batch jobs
- Available partitions (i.e. groups of nodes with similar resources/limits):
  - Allocatable by node (exclusive access)
    - `standard` (LUMI-C)
    - `standard-g` (LUMI-G)
  - Allocatable by resources (shared access)
    - `small`, `debug` (LUMI-C)
    - `small-g`, `dev-g` (LUMI-G)
    - `largemem` (LUMI-D)

LUMI

Submitting batch jobs

- Use `sbatch job.sh` to submit batch jobs
  - `job.sh` is your batch job script containing resource requests and commands to run
  - By default, stdout and stderr are directed to a file `slurm-<jobid>.out`
- Use `squeue --me` to list your submitted jobs
- Please submit jobs from your project's `/scratch` directory!

Simple batch job script for GROMACS

```
#!/bin/bash
#SBATCH --partition=small-g                          # Partition name
#SBATCH --account=project_465000934                  # Project for billing
#SBATCH --reservation=gromacs_wednesday              # Reservation name
#SBATCH --time=00:10:00                              # Run time (d-hh:mm:ss)
#SBATCH --nodes=1                                    # Total number of nodes
#SBATCH --gpus-per-node=1                            # Number of GPUs per node
#SBATCH --ntasks-per-node=1                          # Total number of MPI tasks per node
#SBATCH --cpus-per-task=7                            # Number of threads per task

module use /appl/local/csc/modulefiles               # We use CSC's local module tree
module load gromacs/2023.3-gpu                       # and load GROMACS version 2023.3

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}        # Set number of OpenMP threads

srun gmx_mpi mdrun ...                               # Launch application
```

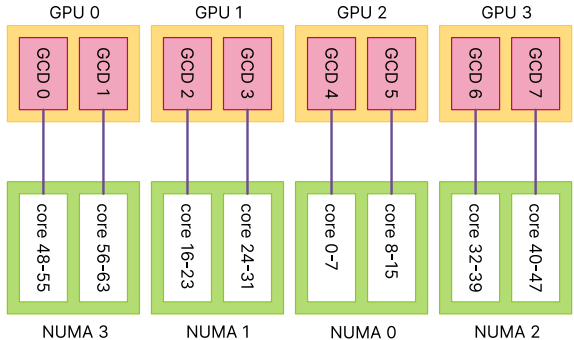
- **Note!** `srun` is the only parallel launcher available on LUMI

LUMI

Improving multi-GPU performance: Architecture recap

- Compute nodes use non-uniform memory access (NUMA) design
- 4 NUMA domains per CPU containing 2 CCDs with 8 cores each
- Memory in the local NUMA node can be accessed faster
- GPUs linked to specific NUMA nodes on LUMI-G
- If we have **exclusive access** to a node, we can use Slurm to *bind tasks to resources* for optimal performance
  - Requires `#SBATCH --exclusive` or run in `standard-g` partition
  - **Important! Do not use unless your job is really utilizing all the reserved resources!**

CPU–GPU links



- LUMI-G has “low-noise” mode activated
  - One CPU core is reserved for the OS to reduce jitter
  - For a more balanced layout, first core of each CCD is disabled, so **only 56 cores are available for GPU jobs** (7 cores per GCD)!

LUMI

Binding tasks to resources

- Slurm uses *hexadecimal masks* for custom selection of which CPU cores tasks should bind to
  - Bits are ordered from **right to left**
  - Each task needs a mask
- Example: Single mask for 7 cores out of 8 (disabling core number 0)

```
76543210    # core numbers
11111110    # binary mask (0 = exclude, 1 = include)
fe          # hexadecimal value
```

- This would be the correct mask for CCD 0

Binding tasks to resources

- CCD 1:
  - Binary mask: `1111111000000000` (16 bits)
  - Hexadecimal value: `fe00`
- CCD 2:
  - Binary mask: `111111100000000000000000` (24 bits)
  - Hexadecimal value: `fe0000`
- ...and so on, yielding the complete mask:

```
srun --cpu-bind=mask_cpu:fe,fe00,\                          # cores 1-7, 9-15
                         fe0000,fe000000,\                  # cores 17-23, 25-31
                         fe00000000,fe0000000000,\          # cores 33-39, 41-47
                         fe000000000000,fe00000000000000    # cores 49-55, 57-63
```

Multi-GPU runs

- Remember that there's **no direct correspondence between CCD order and GCD numbering:**

	GCD 0	GCD 1	GCD 2	GCD 3	GCD 4	GCD 5	GCD 6	GCD 7
Cores	49–55	57–63	17–23	25–31	1–7	9–15	33–39	41–47

- To account for this, we **expose a single GCD to each task and reorder the CPU mask so that the task and GCD IDs match**
  - **Note!** the lowest task ID on each node is mapped to the first mask specified in the list (see next slide)
- To enable GPU-aware MPI, add `export MPICH_GPU_SUPPORT_ENABLED=1`

A complete example for GROMACS (full GPU node)

```
#!/bin/bash
#SBATCH --partition=standard-g
#SBATCH --account=<project>
#SBATCH --time=00:10:00
#SBATCH --nodes=1
#SBATCH --gpus-per-node=8
#SBATCH --ntasks-per-node=8

export OMP_NUM_THREADS=7
export MPICH_GPU_SUPPORT_ENABLED=1
export GMX_ENABLE_DIRECT_GPU_COMM=1
export GMX_FORCE_GPU_AWARE_MPI=1

cat << EOF > select_gpu
#!/bin/bash
export ROCR_VISIBLE_DEVICES=\$SLURM_LOCALID
exec \$*
EOF

chmod +x ./select_gpu

CPU_BIND="mask_cpu:fe000000000000,fe00000000000000"
CPU_BIND="${CPU_BIND},fe0000,fe000000"
CPU_BIND="${CPU_BIND},fe,fe00"
CPU_BIND="${CPU_BIND},fe00000000,fe0000000000"

srun --cpu-bind=${CPU_BIND} ./select_gpu gmx_mpi ...
```

- **Note!** if requesting more tasks than GCDs, one needs to ensure that `ROCR_VISIBLE_DEVICES` is not assigned a too large value
- To keep the exercise job scripts simple, most of this magic is hidden in a script `lumi-affinity.sh` that we source
- However, it's important to remember that these steps are very important for optimal multi-GPU performance

Monitoring GPU utilization

- It is not possible to `ssh` to compute nodes on LUMI
- You can, however, start an **interactive shell** on a compute node where you have a job running using `srun`:

```
srun --interactive --pty --jobid=<jobid> $SHELL
```

- `rocm-smi -u` can then be used to monitor the GPU use
- Alternatively, replace `$SHELL` with `rocm-smi -u` to avoid having to start a shell on the compute node in the first place

Take-home messages

- Due to LUMI's CPU–GPU linking and low-noise mode, a custom binding is important to maximize performance of multi-GPU runs
  - Requires an **exclusive job allocation**, so ensure that your system is large enough to utilize all resources!
  - Alternatively, run multiple independent simulations that share the allocated resources (e.g. GROMACS `-multidir`)
- Slurm uses *hexadecimal bitmasks* to bind tasks to resources
  - A bit cumbersome, so **use the ready-made templates**!
- See:
  - **GROMACS batch script templates:** `docs.csc.fi/apps/gromacs/`
  - **Running jobs on LUMI:** `docs.lumi-supercomputer.eu/runjobs/`