LUMI

Scaling up GPU computing on LUMI

Case Study with GROMACS

Goals

GROMACS is a widely used molecular dynamics package, open-source and community-driven with mature support for GPU computing

This case study presents:

- How to work with the LUMI software stack
- Running a job on GPU partition (LUMI-G)
- Distributing work across available CPU cores and GPUs
- Correct binding to a GPU and CPU
- Difference between multi-process and multithreaded execution

Remarks on code's version

For this purpose unofficial branch of Gromacs is used

- HIP port from AMD
- The containerized version from the AMD Infinity Hub
  - https://www.amd.com/en/technologies/infinity-hub/gromacs
  - https://hub.docker.com/r/amdih/gromacs
- Source code from ROCm GitHub repository
  - https://github.com/ROCmSoftwarePlatform/Gromacs

The Benchmark Systems

- HECBioSim Benchmarks

  https://www.hecbiosim.ac.uk/access-hpc/benchmarks
- A free GROMACS benchmark set
  https://www.mpinat.mpg.de/grubmueller/bench

Using LUMI software stack

- I will start with the containerized version
- There are guidelines on execution on the AMD's page
  - https://www.amd.com/en/technologies/infinity-hub/gromacs
  - Including singularity which is supported container runtime on LUMI

- The container supports only single-node execution
- I would need to use custom version of the application to enable MPI (multi-node) execution
- There is a recipe for self-building in the "software library"
  - https://lumi-supercomputer.github.io/LUMI-EasyBuild-docs/g/GROMACS
  - GPU enabled version is marked with `-MPI-GPU` name prefix

Running a job on GPU partition

There are generic allocation options required to use GPU resources

- Either allocate resource access or define batch job with SLURM
  - Use `salloc` command to get on-demand access
  - Submit job defined with job script using `sbatch` command

- Define required resources (LUMI specific):
  - GPU partition with `--partition=small-g  standard-g dev-g`
  - Learn about allocation policy https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/partitions/
  - Define number of nodes with `--nodes=NN`
  - Define per-node resources: GPUs (`--gpus-per-node=NG`), tasks (`--ntasks-per-node=NT`), CPUs (`--cpus-per-task=NCPUS`)

Managing program execution

Resource and task distribution with SLURM

- Number of nodes
- Number of tasks
- Number of GPUs and CPUs
- Distribution of tasks
- Multithreading of taks
- CPUs and GPUs binding/mapping

Additional control with environmental variables

- Thread placement (multi-threading)
- GPU exposure (multi-gpu)
- GPU awareness in MPI (multi-node)
- NIC binding (multi-gpu, multi-node)
- Diagnostics on resource distribution

Features and runtime mode

- Use of GPU kernel offloading (application specific)
- Data decomposition (application specific)
- Multi-gpu management

Distributing work

- Remark: LUMI GPU nodes are running in the "low-noise" mode with first core reserved for OS and there 63 core available for user to allocate
- Starting set of SLURM directives (job script preamble)

```
#!/bin/bash                          shebang line
#SBATCH --exclusive                  required for task binding
#SBATCH --ntasks=1                   for single-node run
#SBATCH --ntasks-per-node=1          for the container multi-theaded run
#SBATCH --cpus-per-task=48           avoid using core 0
#SBATCH --gpus-per-node=8            for multi-gpu run
#SBATCH --partition small-g          select one of the GPU partitions
#SBATCH --account=project_XXXXXXXX   project account number
#SBATCH --hint=nomultithread         avoid hardware threads to be enabled
```

- Learn more on SLURM distribution modes using docs
  https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/slurm-quickstart/

Running the container

- Singularity is (HPC) container runtime on LUMI
- You can run docker containers directly from DockerHub
- Command line:

```
singularity exec \
docker://amdih/gromacs:2022.3.amd1_174 \
gmx mdrun \
-notunepme -noconfout \
-ntomp 6 -ntmpi 8 \
-pin on -pinoffset 1 \
-npme 1 -bonded cpu \
-nb gpu -pme gpu \
-gpu_id 0,1,2,3,4,5,6,7 \
-s benchmark.tpr
```

Tuning the environment

- Some of the GROMACS features need to be enabled with runtime environment variables (for GPU execution)

```
export GMX_GPU_DD_COMMS=true \
export GMX_GPU_PME_PP_COMMS=true \
export GMX_FORCE_UPDATE_DEFAULT_GPU=true
```

- Thread placement also can be managed with environment variables

```
export OMP_NUM_THREADS=6 \
export OMP_PROC_BIND=close \
export OMP_PLACES=cores
```

- Variable names need `SINGULARITYENV_` prefix to be exported to the container
- Thread placement is managed by GROMACS internally

Complete jobscript for a container

```
#!/bin/bash

#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=48
#SBATCH --time=0:15:0
#SBATCH --exclusive
#SBATCH --partition small-g
#SBATCH --account=project_465000XXX
#SBATCH --hint=nomultithread
#SBATCH --gpus-per-node=8

module load LUMI/22.08                                  # Use LUMI software stack

module load partition/G                                 # Use software components for GPU execution (ROCm)

export SINGULARITYENV_GMX_GPU_DD_COMMS=true

export SINGULARITYENV_GMX_GPU_PME_PP_COMMS=true

export SINGULARITYENV_GMX_FORCE_UPDATE_DEFAULT_GPU=true

export SINGULARITYENV_OMP_NUM_THREADS=6

export SINGULARITYENV_OMP_PROC_BIND=close

export SINGULARITYENV_OMP_PLACES=cores

srun singularity exec docker://amdih/gromacs:2022.3.amd1_174 \
gmx mdrun -notunepme -noconfout -resethway -ntomp 6 -ntmpi 8 -pin on -pinoffset 1 -npme 1 -bonded cpu -nb gpu -pme gpu -gpu_id 0,1,2,3,4,5,6,7  -s benchmark.tpr
```

- Launch program on allocated nodes with `srun` (the only launcher for LUMI)
- Submit with `sbatch job_script.sh`

Building native application

- I would need to use custom version of the application to enable MPI multi-node, multi-gpu execution
- There is ready to use recipe to build (self-install) version
- Code-base similar to the containerized version but with MPI enabled
  - Precisely commit `7bbbb2de208ac1a42c7144ac28ecc5fb0dc0dd4e` of the `develop_2022_amd` branch
  - Recipe name `GROMACS-2023-dev-cpeGNU-22.08-MPI-GPU`

Building native application cont.

- Simple steps for installation (on the system)
  - Select the Software stack version with: `` `module load LUMI/22.08` ``
  - Select target HW partition with: `` `module load partition/G` ``
  - Enable user based EasyBuild installation with: `` `module load EasyBuild-user` ``
  - Run installation with: `` `eb --robot --trace GROMACS-2023-dev-cpeGNU-22.08-MPI-GPU.eb` ``
- After completing these steps, software is available with `` `module load GROMACS/2023-dev-cpeGNU-22.08-MPI-GPU` ``

Running the MPI version

- Need to add tasks (multi-process) distribution
- Targeting one MPI rank per GPU and six threads per MPI task
- Add SLURM directives:

```
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=6
```

- Refine job script, adding software modules

```
module load LUMI/22.08
module load partition/G
module load GROMACS/2023-dev-cpeGNU-22.08-MPI-GPU
```

- Remove singularity executor, `tmpi` option and change executable name

```
srun gmx_mpi mdrun -nsteps 4000 -notunepme -noconfout -npme 1 -bonded cpu
-nb gpu -pme gpu -update gpu -ntomp 6 -gpu_id 0,1,2,3,4,5,6,7
```

Complete jobscript

```
#!/bin/bash

#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=6
#SBATCH --time=0:15:0
#SBATCH --exclusive
#SBATCH --partition small-g
#SBATCH --account=project_465000XXX
#SBATCH --hint=nomultithread
#SBATCH --gpus-per-node=8

module load LUMI/22.08
module load partition/G
module load GROMACS/2023-dev-cpeGNU-22.08-MPI-GPU

export GMX_GPU_DD_COMMS=true
export GMX_GPU_PME_PP_COMMS=true
export GMX_FORCE_UPDATE_DEFAULT_GPU=true

export SRUN_CPUS_PER_TASK=6
export OMP_NUM_THREADS=6
export OMP_PLACES=cores
export OMP_PROC_BIND=close

srun gmx_mpi mdrun -nsteps 4000 -notunepme -noconfout -npme 1 -bonded cpu -nb gpu -pme gpu -update gpu -ntomp 6 -gpu_id 0,1,2,3,4,5,6,7 \
-s benchmark.tpr
```

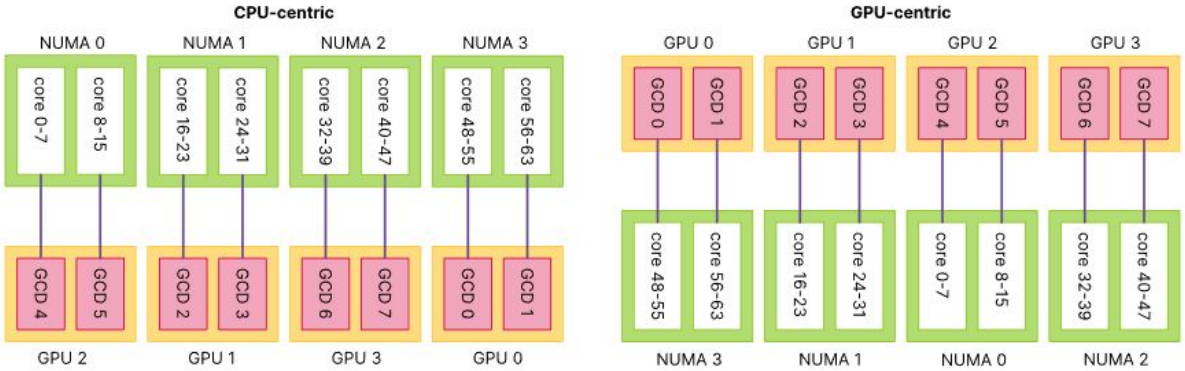
- Submit with `sbatch job_script.sh`

Failure

- Expected performance with this run mode is very low (comparing with the multi-threaded container)
- GROMACS is doing good job with thread placement (binding to CPUs) in the multi-threaded mode (tMPI plus OpenMP)
- How to improve?

Binding to GPU to CPUs

- Proper binding the NUMA node to the GPU is crucial for achieving optimal performance
- It may be beneficial to avoid each first core of the NUMA region while they are used for GPU management



Source https://docs.lumi-supercomputer.eu/hardware/lumig/

Binding to GPU to CPUs cont.

- To allocate specific CPU cores to job tasks (processes) there are SLURM's options:
  - `--cpu-bind=map_cpu:MAPPING`
  - `--cpu-bind=mask_cpu:MASK`
- I need to bind CPUs using "GPU-centric" policy with custom NUMA region order

Understanding bitmasks

- In this case I decided to mask every first (and last) core of the NUMA region
- Mask first and last core of the CCD (8 cores) to accommodate 6 threads
  - binary mask: `0111110`
  - hexadecimal mask: `0x7e`
- Combining masks for NUMA regions (CCDs)
  - `mask_cpu:7e,7e00,7e0000,7e000000,7e00000000,7e0000000000,7e000000000000,7e000000000000`
- Correct order:
  - `48-55 , 56-63 , 16-23 , 24-31 , 0-7 , 8-15, 32-39 , 40-47`
  - `mask_cpu:7e000000000000,7e000000000000,7e0000,7e000000,7e,7e00,7e00000000,7e0000000000`

Complete jobscript

```
#!/bin/bash

#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=6
#SBATCH --time=0:15:0
#SBATCH --exclusive
#SBATCH --partition small-g
#SBATCH --account=project_465000XXX
#SBATCH --hint=nomultithread
#SBATCH --gpus-per-node=8

module load LUMI/22.08
module load partition/G
module load GROMACS/2023-dev-cpeGNU-22.08-MPI-GPU

export GMX_GPU_DD_COMMS=true
export GMX_GPU_PME_PP_COMMS=true
export GMX_FORCE_UPDATE_DEFAULT_GPU=true

export SRUN_CPUS_PER_TASK=6
export OMP_NUM_THREADS=6
export OMP_PLACES=cores
export OMP_PROC_BIND=close

export SLURM_CPU_BIND="mask_cpu:7e000000000000,7e00000000000000,7e0000,7e000000,7e,7e00,7e00000000,7e0000000000"

srun gmx_mpi mdrun -nsteps 4000 -notunepme -noconfout -npme 1 -bonded cpu -nb gpu -pme gpu -update gpu -ntomp 6 -gpu_id 0,1,2,3,4,5,6,7 \
-s benchmark.tpr
```

- Submit with `sbatch job_script.sh`

Problems with a performance

- Again performance with this run mode is not comparable to the multi-threaded container
- You can observe large overheads associated with GPU communication
- How to improve?

Enabling GPU-aware MPI

- To improve performance you need to enable direct communication between GPUs binded to the different MPI tasks
- GROMACS is not able to detect if MPI on LUMI is GPU aware
- This is controlled with environmental variables:

```
export GMX_ENABLE_DIRECT_GPU_COMM=true
export GMX_FORCE_CUDA_AWARE_MPI=true (GROMACS 2022)
```

- MPI also needs to be enabled with GPU awareness:

```
export MPICH_GPU_SUPPORT_ENABLED=1
export MPICH_GPU_IPC_ENABLED=1
```

Complete jobscript

```
#!/bin/bash

#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=0:15:0
#SBATCH --exclusive
#SBATCH --partition small-g
#SBATCH --account=project_465000XXX
#SBATCH --hint=nomultithread
#SBATCH --gpus-per-node=8

module load LUMI/22.08
module load partition/G
module load GROMACS/2023-dev-cpeGNU-22.08-MPI-GPU

export GMX_GPU_DD_COMMS=true
export GMX_GPU_PME_PP_COMMS=true
export GMX_FORCE_UPDATE_DEFAULT_GPU=true
export GMX_ENABLE_DIRECT_GPU_COMM=true
export GMX_FORCE_CUDA_AWARE_MPI=true

export MPICH_GPU_SUPPORT_ENABLED=1
export MPICH_GPU_IPC_ENABLED=1

export OMP_NUM_THREADS=6
export OMP_PLACES=cores
export OMP_PROC_BIND=close

export SLURM_CPU_BIND="mask_cpu:7e000000000000,7e00000000000000,7e0000,7e000000,7e,7e00,7e00000000,7e0000000000"

srun gmx_mpi mdrun -nsteps 4000 -notunepme -noconfout -npme 1 -bonded cpu -nb gpu -pme gpu -update gpu -ntomp 6 -gpu_id 0,1,2,3,4,5,6,7 \
-s benchmark.tpr
```

Still Problems with a performance

- Some of the features allowing efficient GPU usage are not implemented in this code version
- How to update code version?

Upgrading code version

- Upgrade your build to more recent version of GROMACS
- Create custom EasyBuild recipe by adding specific commit from the `develop_2023_amd` branch (change commit hash)
- Reinstall GROMACS with the new recipe
  - Use `--force` and `--force-download=all` options for EasyBuild
- Update `GMX_FORCE_CUDA_AWARE_MPI` variable with `GMX_FORCE_GPU_AWARE_MPI`

Conclusion

- Containerized version provide easy to start, performance friendly approach
- Building custom version with a multi-node support requires care
- Correct, non-default CPU and GPU binding make huge difference in performance
- GPU-awareness of MPI is not enabled by default
- You still need to update to the latest GROMACS release (2023) to get best performance on multiple nodes!

LUMI

Maciej Szpindler

m.szpindler@cyfronet.pl

Follow us

Twitter: @LUMIhpc

LinkedIn: LUMI supercomputer

YouTube: LUMI supercomputer

www.lumi-supercomputer.eu

contact@lumi-supercomputer.eu

EuroHPC Joint Undertaking

The acquisition and operation of the EuroHPC supercomputer is funded jointly by the EuroHPC Joint Undertaking, through the European Union's Connecting Europe Facility and the Horizon 2020 research and innovation programme, as well as the of Participating States FI, BE, CH, CZ, DK, EE, IS, NO, PL, SE.

Leverage from the EU 2014–2020

European Union
European Regional Development Fund

REGIONAL COUNCIL OF KAINUU

EURO