LUMI Hackathon -- GENE-X Summary Report

Jordy Trilaksono
Max Planck Institute for Plasma Physics

Dr. Mou Lin
Max Planck Compute and Data Facility

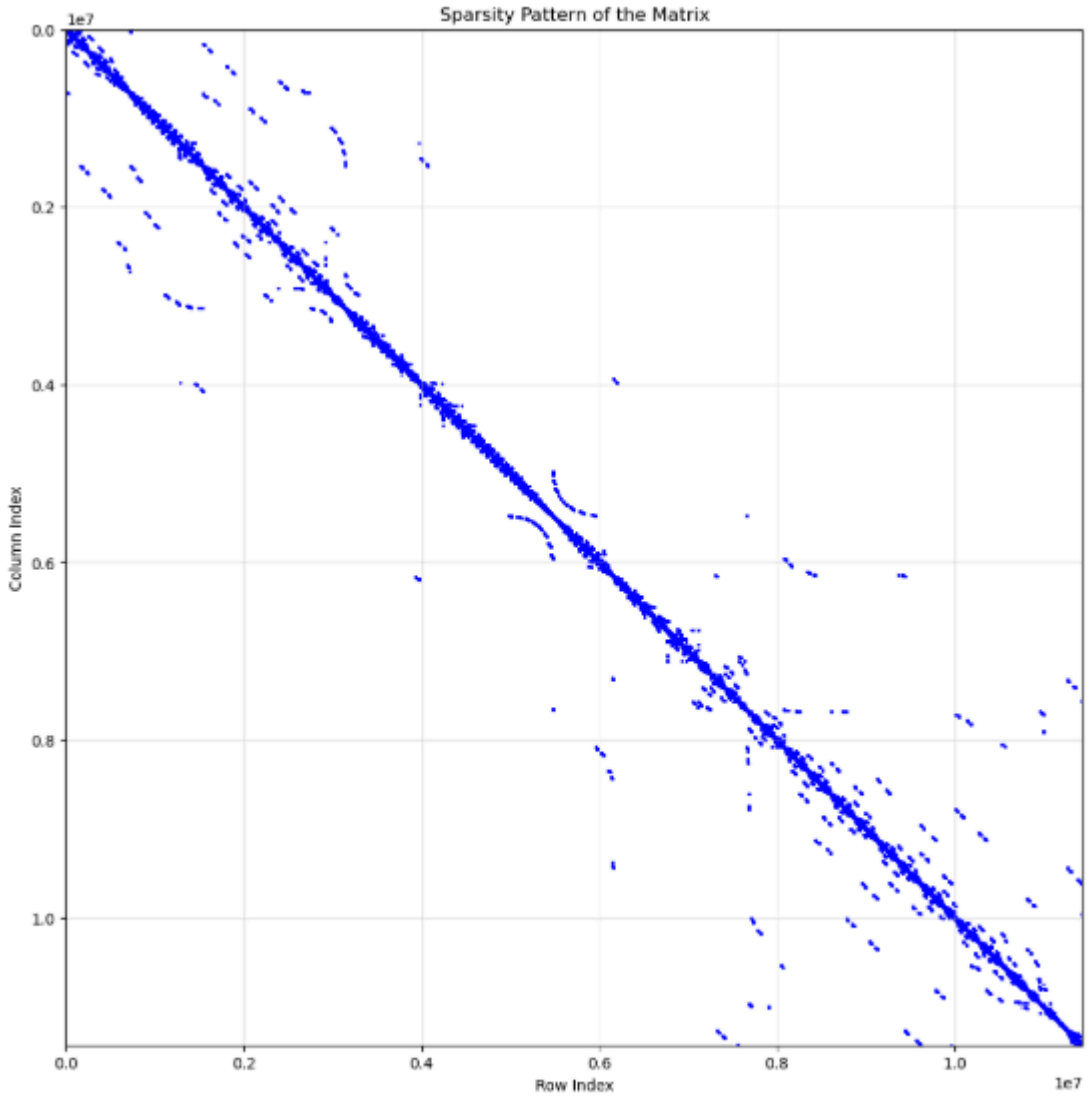
Date: May 16, 2025
Place: Oslo

Goals for Hackathon

- GENE-X
  1. Build GENE-X on LUMI with OpenMP offload
  2. Build GENE-X (OpenMP offload) and PAccX (HIP)
  3. Test GENE-X on LUMI-G with OpenMP offload
  4. Do a benchmark: 4-node or 8-node
  5. Debug and profile to identify coupling issue with PAccX
  6. Debug and profile to identify load imbalance
  7. Do a benchmark again: 4-node or 8-node
- PARALLAX/PAccX
  1. Improve the matrix update of the helmholtz solver, which is taking place on CPU when using rocalution.
  2. Further improve the solve part of the helmholtz solver using rocalution library.
  3. Parallelize the helmholtz solver with rocalution.
- Final goal: couple them together and do final benchmark

PARALLAX/PAccX with ROCalution

- Solving Helmhotlz equations on the 2D cross section of ITER Tokamak.
- A problem of $Ax = b$, A is a sparse matrix in CSR format (n_row = 11437831, nnz = 56789255).
- Solve it with rocALUTION in parallel.



Original matrix

- Build customised rocALUTION with MPI support on LUMI (native one within ROCM does not support multi-GPU).

```
cmake .. -DSUPPORT_HIP=ON \
         -DSUPPORT_MPI=ON \
         -DROCM_PATH=${ROCM_PATH} \
         -DAMDGPU_TARGETS="gfx90a:xnack-;gfx90a:xnack+" \
         -DCMAKE_CXX_COMPILER=CC \
         -DCMAKE_C_COMPILER=cc \
         -DCMAKE_INSTALL_PREFIX=${INSTALL_PATH}
```

- Option 2: use the existing ones on LUMI (credit to Samuel)
- Add the MPI-enabled rocalution lib folder to the CMAKE_PREFIX_PATH to use the right rocALUTION.

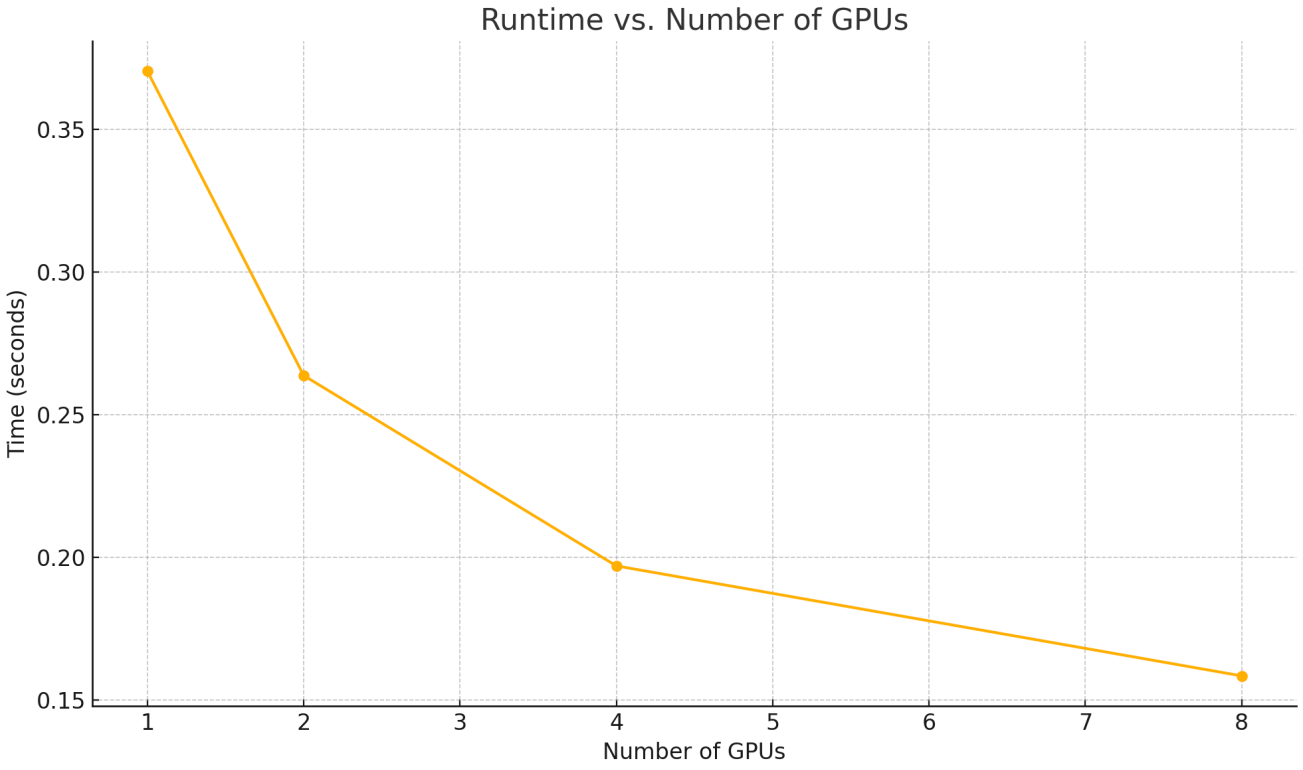
```
module use /pfs/lustrep3/scratch/project_462000394/amd-sw/modules
module load rocalution/3.2.2-rocm-6.0.3
export CMAKE_PREFIX_PATH=/pfs/lustrep3/scratch/project_462000394/amd-sw/rocalution/rocalution-3.2.2-for-rocm-6.0.3/lib/cmake/rocalution:$CMAKE_PREFIX_PATH
```

- **Version matters!**: in our mini-benchmark, rocm 6.0.3 yields the best result (as warned by LUMI).
- Using rocm 6.2.4 makes GPU-aware MPI ineffective (10% slower than rocm 6.0.3).

- **Method matters!**: before we are using smooth aggregated algebric multigrid (SAAMG) for single GPU. To parallelise it, we need to use Ruge-Stüben AMG and other setup

```
p.SetCoarseningStrategy(CoarseningStrategy::PMIS);
p.SetInterpolationType(InterpolationType::ExtPI);
p.SetCoarsestLevel(20);
p.SetInterpolationFF1Limit(false);
```

- Runtime vs GPU No.



- The scaling is not ideal, at least it is there.
- Further profiling with rocprof and omniprof and omnitrace to further improve it.

Application Side: GENE-X

Mixed-Compiler Toolchains and Build Attempts

Toolchain	C	C++	Fortran	Build	Run
GNU	cc	CC	ftn	O	O
GNU-AMD	amdclang	amdclang++	ftn	O	Details below
CCE	cc	CC	ftn	O	X

GENE-X run with CCE toolchain on CPU fails very early on at a very trivial point in the code without giving so much useful error output. We came up with a set of compiler flags but perhaps we need to re-evaluate it.

We further investigated GENE-X with GNU-AMD toolchains. In short, the combanitation of `PrgEnv-gnu/8.5.0` and `rocm/6.2.2` seems to work with the addition of

```
export LIBRARY_PATH=$ROCM_PATH/llvm/lib:$LIBRARY_PATH
```

> Note: Unfortunately, mesh generation in PARALLAX, in the equilibrium geometry relevant to production runs, is known to be not working with gfortran/13 although it works with gfortran/12 and gfortran/14. Therefor here we're using different run case to test the code.

Here are the runtime status with different combination of GENE-X and PARALLAX backend.

GENE-X	PARALLAX	Run
CPU: OMP	CPU	O
CPU: OMP	GPU: HIP	Fails during solver matrix generation
GPU: OMPX	CPU	O
GPU: OMPX	GPU: HIP	Fails during solver matrix generation

- OMP: OpenMP on CPU and OMPX: OpenMP offload on GPU
- HIP issue: The spot where the runtime fails typically doesn't fail like that, especially when it runs on CPU.
- For OMPX-CPU run, the total time per timestep is still lower than that with OMP-CPU but we observe some speedups for the main 3 operators or compute kernels.

Operators	Speedup
Vlasov static	8.5X
Vlasov dynamic	11.8X
BGK collision	16.2X

These results were simply on 1 GPU. We didn't have enough time to try multiple GPUs but the full node run (8 MPI process) with OMP-CPU seems to be fine but at the end we got:

```
BLAS : Bad memory unallocation! :   50  0x14f828e3a000
BLAS : Bad memory unallocation! :   50  0x14a1ece3a000
BLAS : Bad memory unallocation! :   50  0x14bd6ce3a000
BLAS : Bad memory unallocation! :   50  0x145d98e3a000
```

Also in all runs that we have, at the end we got the following warning:

```
[CRAYBLAS_WARNING] Application linked against multiple cray-libsci libraries
```

Back with 1 GPU run, we encounter that Vlasov static operator, the largest and most complicated kernel in GENE-X fails in debug mode (-O0) but runs perfectly in release mode (-O3). Intrumenting the source code with the following allows us to have pinpoint verbose diagnostic of the OpenMP offload kernel:

```
extern "C" void __tgt_set_info_flag(uint32_t);
...
__tgt_set_info_flag(-1);

... kernel needs to be debugged ...

__tgt_set_info_flag(0);
```

The metrics are as follow:

Metrics	Debug mode	Release mode
#teams	660	660
#threads	256	256

Metrics	Debug mode	Release mode
ConstWGSize	256	256
lds_usage	9976B	8432B
#sgpr	108	106
#vgpr	118	338
#sgpr spill	2	50
#vgpr spill	0	0
#trip	133592960	133592960

GENE-X/PARALLAX ROCalution Integration Efforts

The integration here only allows ROCalution to use 1 GPU / MPI process, the same as GENE-X.

GENE-X	PARALLAX	Run
CPU: OMP	GPU: ROCalution	OK but slow
GPU: OMPX	GPU: ROCalution	Fails during solver matrix generation

The solver are faster but somehow rocalution affect the OpenMP runtime of GENE-X native kernels and significantly slow them down. Setting the following still results the same slowdown:

```
// Disable OpenMP thread affinity
rocalution::set_omp_affinity_rocalution(false);

// Disable OpenMP threading
rocalution::set_omp_threads_rocalution(1);
```