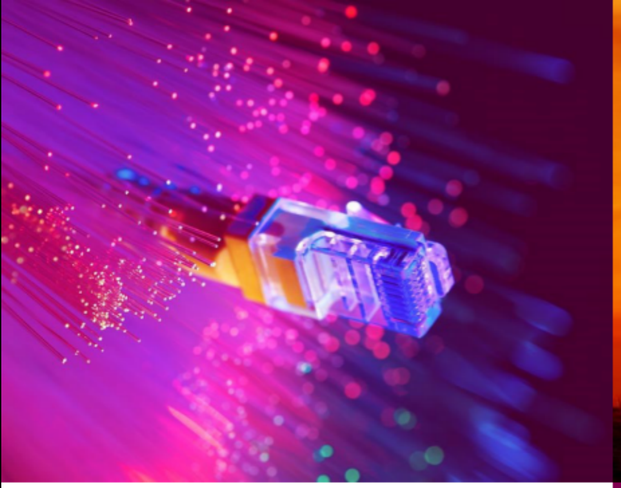


GPAW, GPUs, and LUMI

Martti Louhivuori, CSC - IT Center for Science
Jussi Enkovaara

GPAW 2021: Users and Developers Meeting, 2021-06-01

CSC – Finnish expertise in ICT for research, education and public administration

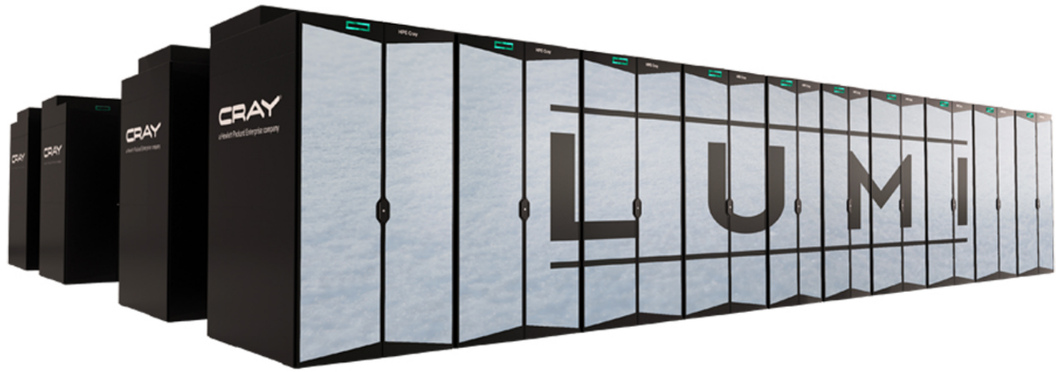
Outline

- LUMI supercomputer
- Brief history of GPAW with GPUs
- GPUs and DFT
- Current status
- Roadmap

LUMI - EuroHPC system of the North

- Pre-exascale system with AMD CPUs and GPUs
  - ~ 550 Pflop/s performance
- Half of the resources dedicated to consortium members
  - Finland, Belgium, Czechia, Denmark, Estonia, Iceland, Norway, Poland, Sweden, and Switzerland

https://www.lumi-supercomputer.eu



- Programming for LUMI
  - MPI between nodes / GPUs
  - HIP and OpenMP for GPUs
  - how to use Python with AMD GPUs?

GPAW and GPUs: history (1/2)

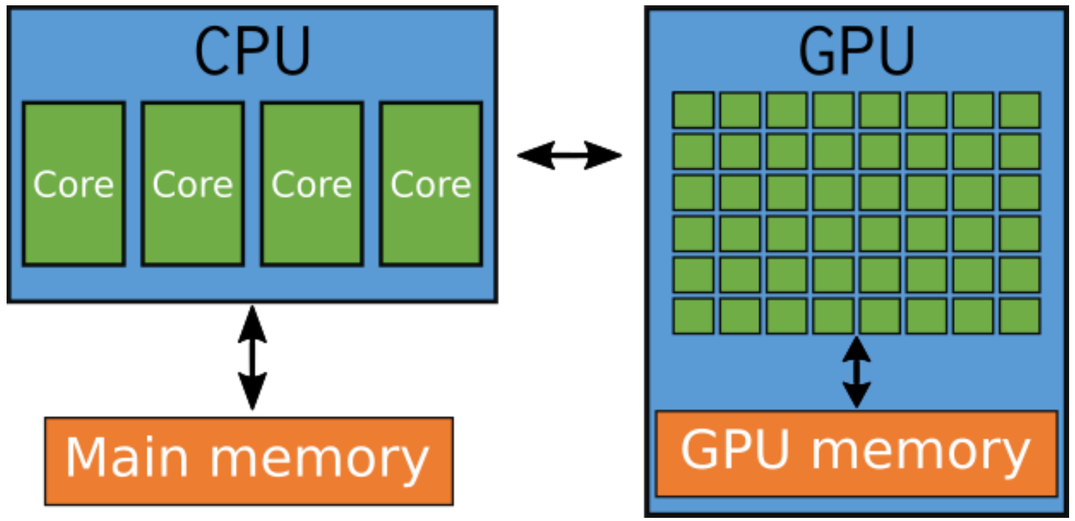
- Early proof-of-concept implementation for NVIDIA GPUs in 2012
  - ground state DFT and real-time TD-DFT with finite-difference basis
  - separate version for RPA with plane-waves
  - Hakala *et al.* in "Electronic Structure Calculations on Graphics Processing Units", Wiley (2016), https://doi.org/10.1002/9781118670712
- PyCUDA, cuBLAS, cuFFT, custom CUDA kernels
- Promising performance with factor of 4-8 speedup in best cases (CPU node vs. GPU node)

GPAW and GPUs: history (2/2)

- Code base diverged from the main branch quite a bit
  - proof-of-concept implementation had lots of quick and dirty hacks
  - fixes and features were pulled from other branches and patches
  - no proper unit tests for GPU functionality
  - active development stopped soon after publications
- Before development re-started, code didn't even work anymore on modern GPUs without applying a few small patches
- Lesson learned: try to always get new functionality to the main development branch!

What is a GPU?

- GPUs are extremely parallel processors with a limited instruction set
  - single core is not that fast, but there is a lot of them
  - thousands of threads needed for good utilisation
  - streamlined for number crunching
- Direct communication between GPUs is possible with some MPI implementations



- GPU needs a host CPU
  - computations are offloaded from CPU to GPU
  - data needs to be transferred between the memories

GPUs and DFT codes (1/2)

- At some system sizes, most algorithms in DFT become dominated by dense linear algebra (orthonormalisation, dense matrix diagonalisation, subspace rotation) ⇒ well suited for GPUs
- Transfer of data between CPUs and GPUs can easily become a bottleneck if not all computations are done on the GPUs
  - it may be useful to run even "inefficient" parts on the GPUs to avoid data transfers

GPUs and DFT codes (2/2)

- Most major DFT code packages start to have some GPU support
  - may be limited to specific features
  - VASP, CP2K, Abinit, Quantum Espresso, FHI-Aims, BigDFT, …
  - BerkeleyGW as Gordon Bell finalist in 2020
    - GW calculation of excited states of silicon divacancy, 2742 atoms and 10968 electrons using the whole Summit supercomputer (Oak Ridge, ~4600 nodes each with 2x Power9 + 6x V100 GPUs)

      https://cs.lbl.gov/news-media/news/2020/crdnersc-led-paper-a-gordon-bell-finalist-at-sc20/
- GPAW, re-booted GPU version
  - ground state DFT (and real-time TD-DFT?) with finite-difference basis
  - ported to be based on 21.1.0

GPAW/cuda: What's under the hood?

- CUDA kernels for many operations
- GPU algorithms implemented in C / Python
- Python interfaces to external GPU libraries

- On the Python side, pyCUDA is used to store and access data on the GPU
  - offers a (limited) Numpy-like array interface

GPAW/cuda: CUDA kernels

- Stencil operations: interpolate, restrict, relax, FD
  - different stencil sizes (3..11), different GPU HW (Fermi, Kepler)
- Other grid operations: cut, paste, paste+zero, translate
- Elementwise operations:
  - axpbyz, axpbz, fill, xypz, ax2py, negation
  - vectorised version of axpy, scal, ax2py, xypz
- Localized functions: add, integrate (reduce)
- TD-DFT: addition of linear fields

GPAW/cuda: Python

- Generic GPU stuff
  - device management: init, set device
  - array container: GPUArray
- Python interfaces to CUDA kernels
  - grid operations: cut, paste, interpolate, restrict, relax, FD
  - localized functions: add, integrate
  - elementwise operations
    - including multi-block axpy, scal, dotu, dotc
  - TD-DFT: addition of linear fields
- Python interfaces to cuBLAS
  - BLAS (cuBLAS): scal, gemm, gemv, axpy, syrk/herk, syr2k/her2k, dotc, dotu
  - hybrid BLAS (cuBLAS): gemm, syrk, syr2k

GPAW/cuda: Overview

GPU implementation

- stencil operations:
  - interpolate, restrict, relax, FD
- multi-GPU parallelization:
  - k-points, spins, domain decomposition
  - domain decomposition: sync and async boundary exchange
- TD-DFT using the same CUDA kernels etc.

Performance enhancements

- batching:
  - small grids combined into larger blocks
  - used in stencil operations and in several BLAS functions
- hybrid BLAS level 3 functions:
  - GEMM, SYRK, SYR2K
  - calculation decomposed to both GPU and CPU

GPAW/cuda: Non-optimised performance on Puhti

CPU (2x20 Xeon Gold 6230)

nodes	C6-6-10	Cu71	C60Pb	Si702
1	104.721	3050.768	-	-
2	76.875	1559.466	-	-
4	42.748	932.713	-	-
8	28.149	640.163	327.936	1702.476
16	-	-	206.629	1017.338
20	-	-	205.711	871.564

GPU (4x V100)

nodes	C6-6-10	Cu71	C60Pb	Si702
1	66.022	-	-	-
2	42.295	755.803	-	-
4	29.084	390.078	-	-
8	20.909	221.431	127.906	553.859
16	-	-	82.942	322.553
20	-	-	74.374	278.861

- ground state DFT, FD
  - C6-6-10: carbon nanotube
  - Cu71: copper filament
  - C60Pb: fullerenes & lead surface
  - Si702: silicon cluster
  - https://github.com/mlouhivu/gpaw-benchmarks/

GPU vs. CPU

	C6-6-10	Cu71	C60Pb	Si702
speed-up	~1.5x	~2.5x	~2.5x	~3x

GPAW/cuda: Challenges and warts (1/2)

1. Not all arrays are allocated on the GPU
    - unnecessary(?) synchronisations between GPUs and CPUs
        - operations on mixed CPU and GPU arrays
        - algorithms that use Numpy features that are not supported by pyCUDA
    - ugly passing of `cuda` parameter to decide if an array should be allocated on the GPU or the CPU side
        - nice for flexibility, though!
    - *solution*: allocate all (or at least most) arrays on the GPU?
    - *solution*: switch to (or implement) a better GPU array interface

GPAW/cuda: Challenges and warts (2/2)

2. Add-on nature of the design
    - GPU algorithms are used as an alternative to CPU ones
    - lot of checking and branching to call the correct functions
    - *solution*: deeper integration, e.g. a common GPAW array wrapper that transparently handles Numpy and GPUArray arrays and CPU/GPU functions
3. Lack of proper testing

   ⇒ try something new: things go boom!
4. No support for non-NVIDIA GPUs

Roadmap (1/2)

Clean integration with the CPU code (WIP)

- clean up and document code
- enable one to run the same code either on CPUs or GPUs
- CPU code path should be identical to master

https://gitlab.com/gpaw/gpaw/-/merge_requests/580

Roadmap (2/2)

HIP support

- HIPify CUDA kernels
- change to a GPU array interface that supports both CUDA and HIP
  - or implement a very light-weight interface to manually allocated GPU memory
  - e.g. cupy has experimental HIP support
- change from cuBLAS to hipBLAS

Extend functionality

- new modes: PW, LCAO
- new eigensolvers: Davidson, CG
- ...

Thanks!