Fortran – What are the options for accelerated computing?

Hans Pabst (Intel)

Submitted Abstract

- This case study summarizes the evolution of an existing CUDA code base in DBCSR and CP2K and the introduction of HIP as well as an OpenCL based implementation. The offload interface in CP2K is already an evolution of DBCSR's offload interface and both embrace commonly supported primitives across stream programming models, i.e., the interface is only a handful of C functions (and Fortran binding).
- With the introduction of HIP, the idea was extended to limiting the kernel language to a well-supported (sub-)set of CUDA. For OpenCL this was helpful too, and work went into raising the limits of the existing implementation in DBCSR such as an improved auto-tuning infrastructure, generalized kernels and tuning parameters, supporting all vendors out of the box. The OpenCL backend was then fully reused within CP2K Molecular Dynamics application.
- The talk closes with a collection of results achieved on current HPC installations (DBCSR-MM and CP2K-DBM distributed block-sparse matrix multiplication).

What is accelerated computing?

- **FLOPS-intensive (#1)** as well as **memory-bound code (#2)** can run faster on specialized accelerators, namely GPUs.
- Actual acceleration is subject to the amount of time spent in code benefiting from these properties (**#1** and **#2**) – **Amdahl's law**.
- In case of "discrete" accelerators penalties for copying data to/from the device(s) apply – **Amdahl's law**.
- A typical range of speedup for one GPU over one CPU (multiple cores) is about "2..5..10x" (double precision arithmetic).

What is accelerated computing?

- **FLOPS-intensive (#1)** as well as **memory-bound code (#2)** can run faster on specialized accelerators, namely GPUs.
- Actual acceleration is subject to the amount of time spent in code benefiting from these properties (**#1** and **#2**) – **Amdahl's law**.
- In case of "discrete" accelerators penalties for copying data to/from the device(s) apply – **Amdahl's law**.
- A typical range of speedup for one GPU over one CPU (multiple cores) is about "2..5..10x" (double precision arithmetic).

Why accelerated computing?
... of course, we want to bend this rule of thumb to the max...

Options for accelerated computing

Use libraries with existing Fortran interface, e.g., (Sca-)LAPACK.

→ **Interface C libraries and C code (ISO_C_BINDING module).**

Develop a C *and* F interface for code for non-C languages.

Write pure Fortran code targeting GPUs directly: OpenMP offload.

→ **Write C/C++ code targeting GPUs (CUDA/HIP, SYCL, OpenCL).**

- This talk → **bold** options

Application Summary

Main point of CP2K is diversity, i.e., supported levels of theory, variety of simulations, and emerging systems.

- **Levels of theory**: DFTB, LDA, GGA, MP2, RPA, semi-empirical methods (AM1, PM3, PM6, RM1, MNDO), and classical force fields (AMBER, CHARMM, ...)
- **Simulations**: Metadynamics, Monte Carlo, Ehrenfest dynamics, vibrational analysis, core level spectroscopy, energy minimization, and transition state optimization using NEB or dimer method
- **Emerging**: molecular dynamics of biological systems, e.g., QM/MM simulation with GROMACS coupled with CP2K
- **Workloads**: representation of CP2K is not easy; like toolbox or "Swiss Army knife“
- **Acceleration**: see https://www.cp2k.org/gpu
  - Focus on library components supporting GPUs
  - Initial focus on CUDA, later OpenCL and HIP
- **Modern Fortran**: standard version rolls forward; F2018 currently.
  - Syntax (reformat) as well as conventions checks (AST based analysis), etc.
  - Unit- and Regression testing locally, public, and on cloud-based systems

A metaphorical depiction of how to improve upon the treatment of electron correlation by ascending from the Hartree world to the “heaven of chemical accuracy.”

Application Summary

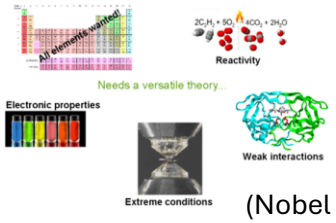
Main point of CP2K is diversity, i.e., supported levels of theory, variety of simulations, and emerging systems.

- **Levels of theory**: DFTB, LDA, GGA, MP2, RPA, semi-empirical methods (AM1, PM3, PM6, RM1, MNDO), and classical force fields (AMBER, CHARMM, ...)
- **Simulations**: Metadynamics, Monte Carlo, Ehrenfest dynamics, vibrational analysis, core level spectroscopy, energy minimization, and transition state optimization using NEB or dimer method
- **Emerging**: molecular dynamics of biological systems, e.g., QM/MM simulation with GROMACS coupled with CP2K
- **Workloads**: representation of CP2K is not easy; like toolbox or "Swiss Army knife“
- **Acceleration**: see https://www.cp2k.org/gpu
  - Focus on library components supporting GPUs
  - Initial focus on CUDA, later OpenCL and HIP
- **Modern Fortran**: standard version rolls forward; F2018 currently.
  - Syntax (reformat) as well as conventions checks (AST based analysis), etc.
  - Unit- and Regression testing locally, public, and on cloud-based systems

A metaphorical depiction of how to improve upon the treatment of electron correlation by ascending from the Hartree world to the “heaven of chemical accuracy.”

CP2K and DBCSR (Small Matrix Multiplications)

CP2K implements* Density Functional Theory (DFT)
* among many other methods

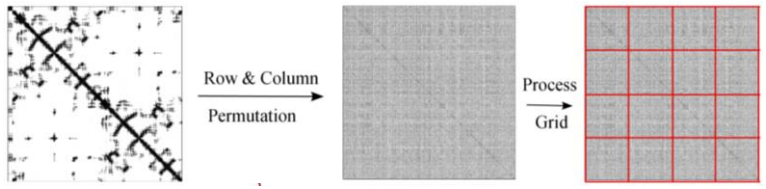


$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V_s(\vec{r})\right]\phi_i(\vec{r}) = \epsilon_i\phi_i(\vec{r})$$

$$V_s(\vec{r}) = V(\vec{r}) + \int \frac{e^2 n_s(\vec{r}')}{|\vec{r}-\vec{r}'|}\mathrm{d}^3r' + V_{\mathrm{XC}}[n_s(\vec{r})]$$

(Nobel Prize 1998: Walter Kohn and John A. Pople)

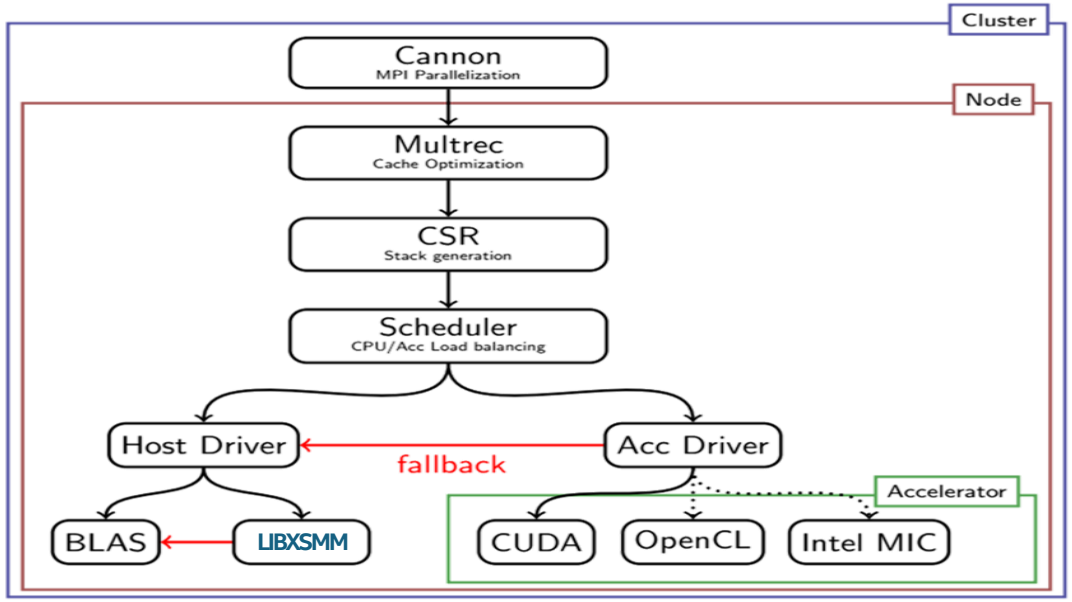
- DFT can be seen as general Eigenvalue problem, which is solved using the Self-Consistent Field (SCF) iterative method
- Sparsity can be exploited, and ends up with small dense blocks of natural structure (atoms)



- Recent CPUs (FMA) are doing very well

* Pictures adapted from Speedup 2012 (Joost VandeVondele)

Distributed **B**locked **C**ompressed **S**parse **R**ow
Distributed **B**locked **C**annon **S**parse **R**ecursive



- DBCSR* library is ubiquitously used by many algorithms in CP2K (not only for DFT)
- DBCSR generates matrix batches (“stacks”) of ~1K..30K small matrix multiplication (accumulation): C += A * B

* CP2K’s sparse matrix library: https://dbcsr.cp2k.org/

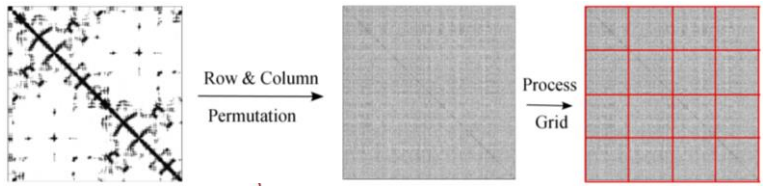
CP2K and DBCSR (Small Matrix Multiplications)

CP2K implements* Density Functional Theory (DFT)
* among many other methods

Many methods in CP2K rely on a SpBLAS-like functionality (DBCSR or DBM) and produce batches of small matrix multiplications.

(Nobel Prize 1998: Walter Kohn and John A. Pople)

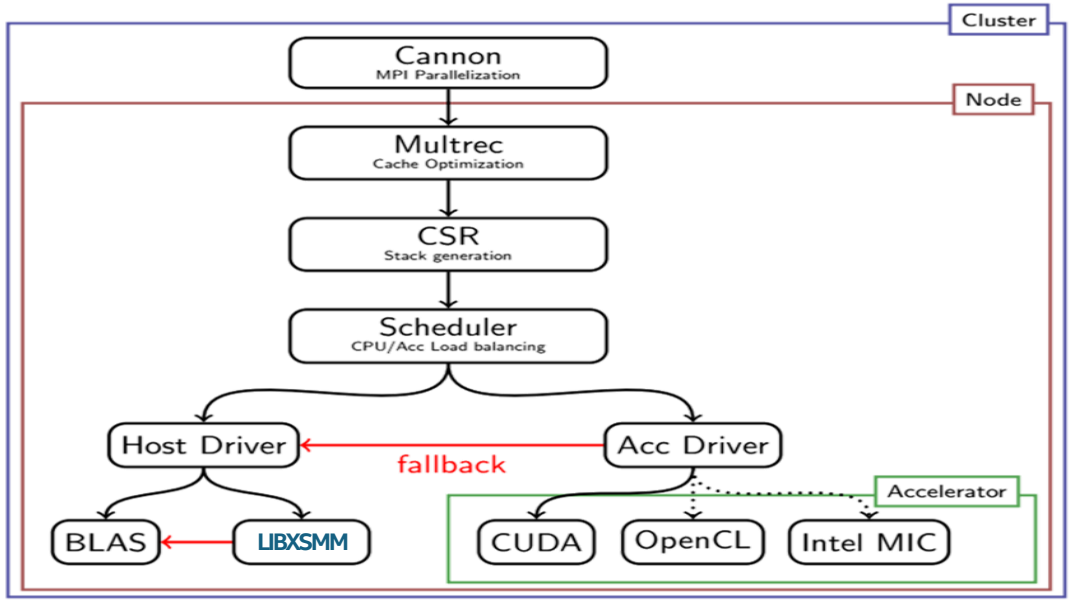
- DFT can be seen as general Eigenvalue problem, which is solved using the Self-Consistent Field (SCF) iterative method
- Sparsity can be exploited, and ends up with small dense blocks of natural structure (atoms)



- Recent CPUs (FMA) are doing very well

* Pictures adapted from Speedup 2012 (Joost VandeVondele)

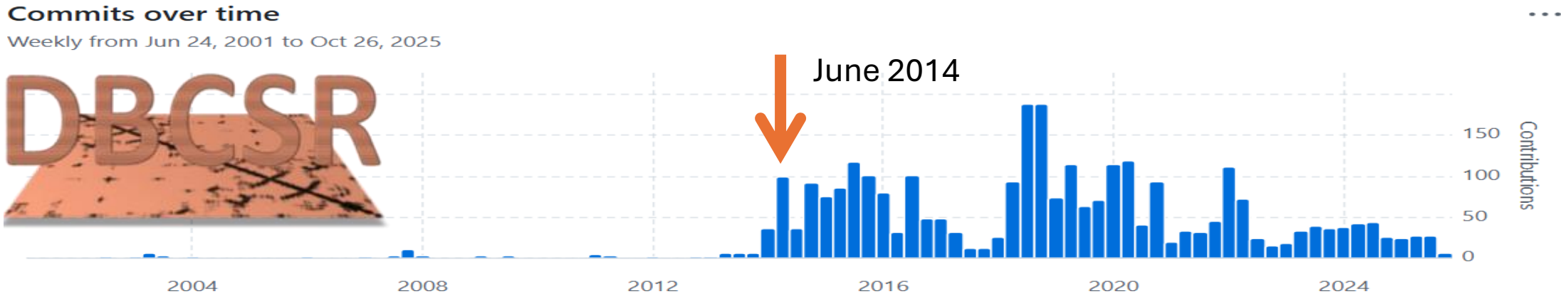
Distributed **B**locked **C**ompressed **S**parse **R**ow
Distributed **B**locked **C**annon **S**parse **R**ecursive

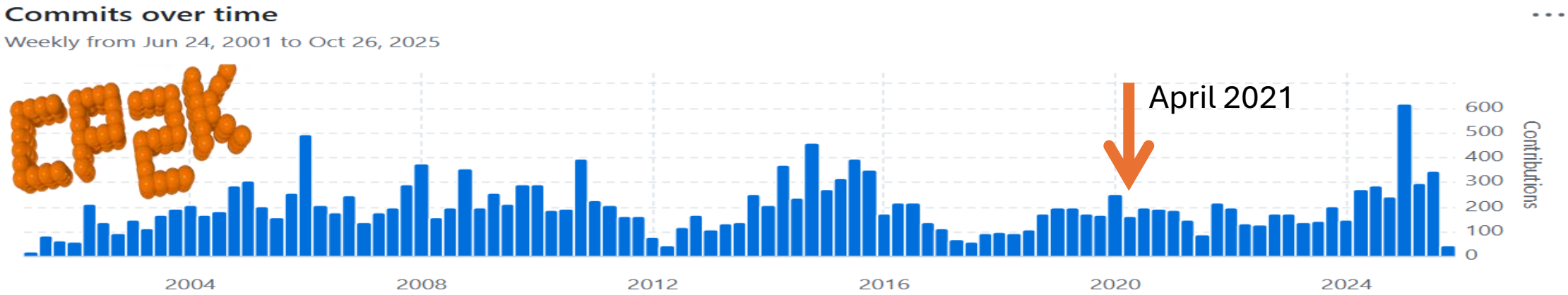


- DBCSR* library is ubiquitously used by many algorithms in CP2K (not only for DFT)
- DBCSR generates matrix batches (“stacks”) of ~1K..30K small matrix multiplication (accumulation): C += A * B

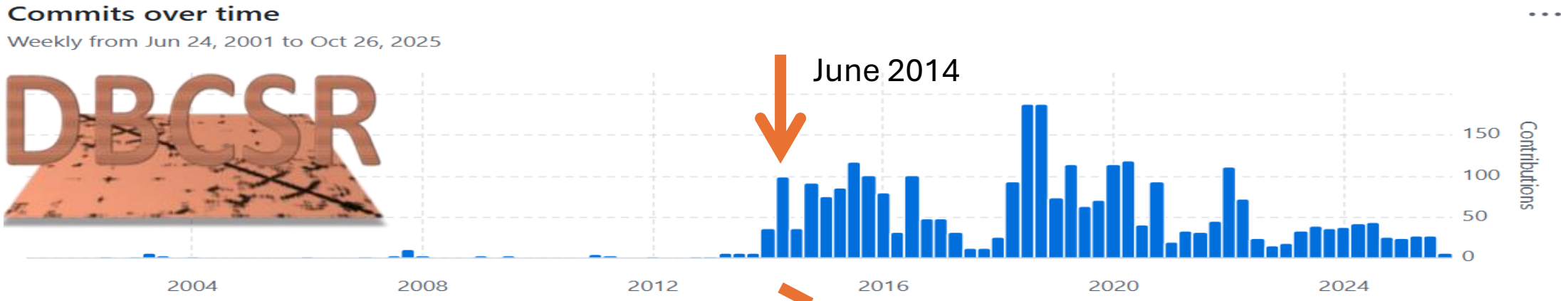
* CP2K’s sparse matrix library: https://dbcsr.cp2k.org/

Introduction of GPU Acceleration

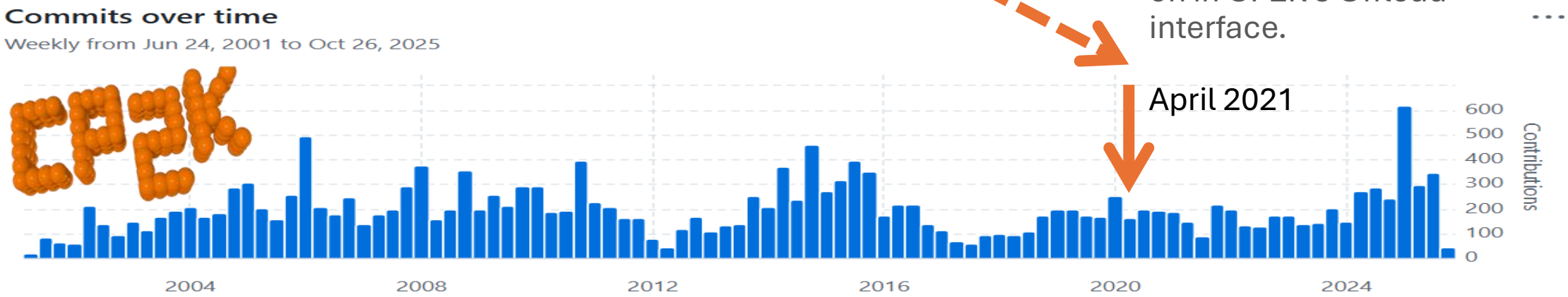




Introduction of GPU Acceleration



Idea of DBCSR's ACC interface was carried on in CP2K's Offload interface.



Offload Interface

offloadMemset[Async]

offloadMemcpy[Async]HtoD

offloadMemcpy[Async]DtoH

offloadMemcpyAsyncDtoD

offloadDeviceSynchronize

offloadStreamCreate/Destroy

offloadStreamSynchronize

offloadEventCreate/Destroy

offloadEventSynchronize

offloadEventRecord

offloadStreamWaitEvent

offloadEventQuery

offloadMalloc/Host

offloadFree/Host

... and some more

ACC interface (DBCSR)

- Very similar, but some differences
- Only base pointers for device memory (offsets are separate)

ISO_C_BINDING and Stream Programming

- Introduce a concise set of C functions for memory allocation, data transfer, stream and event creation, synchronization, etc.
- Binding acts as boundary between science code (Fortran), and targeting/optimizing for accelerators
- Stream programming as a library-based model (beside of writing actual kernels) takes place in Fortran code (much like MPI)

```
FUNCTION offload_get_device_count() &
  RESULT(count)

  INTEGER :: count

  INTERFACE
    FUNCTION get_device_count_c() &
      BIND(C, name="offload_get_device_count")
    IMPORT :: C_INT

    INTEGER(KIND=C_INT) :: get_device_count_c
    END FUNCTION get_device_count_c
  END INTERFACE

  count = get_device_count_c()

END FUNCTION offload_get_device_count
```

OpenCL Overview

- All GPU vendors agree on supporting OpenCL (industry standard)
  - Less or more hidden inside of promoted packages, e.g., Intel oneAPI delivers OpenCL, and so does Nvidia CUDA and AMD ROCm.
  - Delivered CL/cl.h can be old (use Linux distro's "opencl-headers").
- OpenCL can be compiled "ahead-of-time" (AOT; not common)
- OpenCL most often used "just-in-time" (JIT)
  - This contributes to "needs more code" (boiler plate)
  - Advantage: no offline compiler needed
- Language: C99/C11 (and C++ via extension)

Batched Multiplication of Small Matrices (SMMs)

IN: array of tasks, array of A- and B-data, **IN/OUT**: array of C-data

CP2K DBM

- A single/unified kernel for all combinations of M, N, and K
- Top-level control-flow for cases like “larger M”
- Much leaner code base

DBCSR LIBSMM

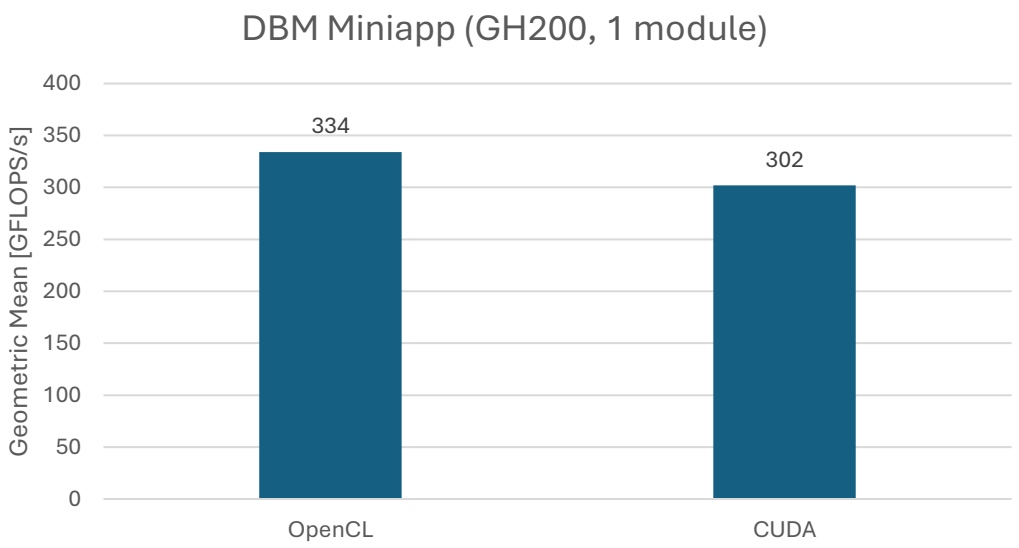
- A single kernel for every combination of M, N, and K
- GPU specific (auto-)tuned parameters

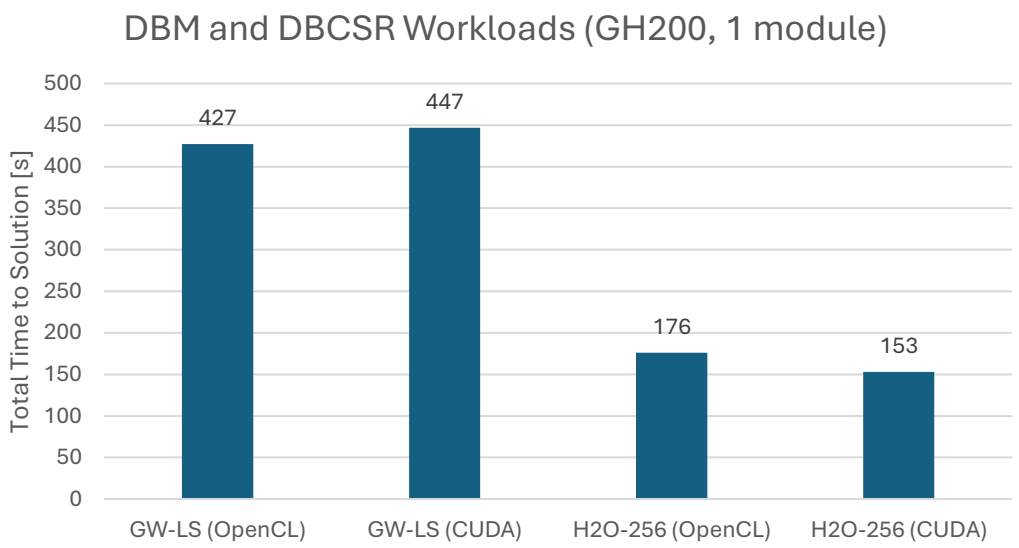
What is different from recent BLAS or batched GEMM?

- Array of tasks with a task structure { M, N, K, Offset A, B, C }
- C-offsets are duplicate indexes in general (data races!)

Performance Results

All code primarily compiled with GNU Fortran





- DBM Miniapp runs a collection of SMMs (different MxNxK); multiple MPI ranks per GPU.
- Two real/citable workloads of CP2K's collection exercising DBM and DBCSR.

Conclusion and Call to Action

- To write GPU kernels in Fortran directly, consider compilers with OpenMP 5 (and later). See also other talks at FortranCon’25.
- To adopt C/C++ backend and kernels, rely on ISO_C_BINDING
  - Consider CUDA/HIP, OpenCL or SYCL
- Appropriate choice can balance maintenance and total lines of code *and* achieve vendor portable performance
  - Stream programming: consider common subset
  - Directive based: consider OpenMP 5 as baseline

References

- CP2K Offload Interface
  https://github.com/cp2k/cp2k/blob/master/src/offload/offload_runtime.h
- CP2K DBM component
  https://github.com/cp2k/cp2k/tree/master/src/dbm
- DBCSR ACC Interface
  https://github.com/cp2k/dbcsr/blob/develop/src/acc/acc.h

Contact: hans [dot] pabst [at] intel.com

Backup

OpenCL Backend for CP2K and DBCSR

- Serves both CP2K's Offload as well as DBCSR's ACC interface
- OpenCL uses cl_mem type instead of raw device pointers
  - CUDA/HIP however internally implement a registry of device pointers
  - This was implemented for the OpenCL BE in CP2K/DBCSR
- Macros to deliver FP-atomics (normally via cl_ext_float_atomics and implemented by ARM, Intel and Qualcomm)
  - AMD: __builtin_amdgcn_global_atomic_fadd_f64 intrinsics
  - Nvidia: PTX inline ASM in OpenCL kernel is used

Plan: OpenCL BE will become an own project (LIBXSTREAM)

- Currently hosted in DBCSR's repository (DBCSR is mandatory in CP2K)

Application Features (GPU/general)

- DBCSR was developed as part of CP2K and later separated
  - GPU features include direct GPU-to-GPU communication
  - Sophisticated and extensive code base (Fortran)
  - Accelerator interface (ACC): CUDA/HIP and OpenCL
- CP2K offload interface to accelerate more parts of code base
  - Here “more parts” means with kernel code in CP2K repository, e.g., GRID integration and FFT offload
  - Distributed Block Matrix (DBM) to replace DBCSR in the future

- Note: Multi-GPU support is generally via MPI

Application Features (OpenCL only)

- Tuned parameters for DBCSR are not required, whereas CUDA/HIP code-path falls back to CPU with no suitable parameters (potentially after uploading data to GPU)
- CP2K's DBM component can act as LIBSMM
- DBCSR's LIBSMM can act as DBM
  - Batch of SMMs must be "pure" (fixed MxNxK)