# deal.II on GPU

## Final presentation at Eurohack 19, Lugano, Switzerland

Peter Munch[123], Momme Allalen[4], Martin Kronbichler[12],
Paddy Ó Conbhuí[4], Prashanth Kanduri[5]

[1]deal.II developer

[2]Institute for Computational Mechanics, Technical University of Munich, Germany

[3]Institute of Materials Research, Materials Mechanics, Helmholtz-Zentrum Geesthacht, Germany

[4]CFD-Lab, Leibniz Supercomputing Centre, Germany

[4]Irish Centre for High-End Computing, Ireland

[5]CSCS/ ETH Zürich, Switzerland

October 4, 2019

Part 1:

# Motivation

# Goal

Goal:

Optimize the matrix-free deal.II[1] GPU implementation for CEED BP5[2].

[1] General purpose finite element library deal.II: `https://www.dealii.org/`.

[2] Solve Poisson problem for $p=5$ and $q=p+1$ with preconditioned conjugate gradient methods.

# Base

Given modules in deal.II:

- efficient matrix-free implementation
- GPU support with CUDA-aware MPI-support (with not satisfying performance results)
- standard preconditioned conjugate gradient method (next slide)

Kronbichler, Kormann, A generic interface for parallel finite element operator application. *Comput. Fluids* 63:135–147, 2012
Kronbichler, Kormann, Fast matrix-free evaluation of discontinuous Galerkin finite element operators. *arXiv:1711.03590*, 2017
M. Kronbichler, K. Ljungkvist. Multigrid for matrix-free high-order finite element computations on graphics processors, ACM Transactions on Parallel Computing, 2019
Daniel Arndt and Bruno Turcksin, Tutorial step-64, online available `https://www.dealii.org/developer/doxygen/deal.II/step_64.html`

# Preconditioned CG (PCG)

**Algorithm 1:** (Standard) preconditioned CG solver

**Data:** $\boldsymbol{A}$, $x_0$, $b$, $\boldsymbol{M}^{-1}$
**Result:** $x$

[startup];
**for** $k = 0,1,\ldots$ **do**
  $\underline{\boldsymbol{v}} \leftarrow \underline{\underline{\boldsymbol{A}}}\,\underline{\boldsymbol{p}}$;
  $\alpha \leftarrow \gamma/(\underline{\boldsymbol{p}}^T \underline{\boldsymbol{v}})$;
  $\underline{\boldsymbol{x}} \leftarrow \underline{\boldsymbol{x}} + \alpha \cdot \underline{\boldsymbol{p}}$;
  $r \leftarrow r - \alpha \cdot \underline{\boldsymbol{v}}$;
  **if** $||\underline{\boldsymbol{r}}||_2 < \varepsilon$ **then**
    **return**
  $\underline{\boldsymbol{v}} \leftarrow \underline{\underline{\boldsymbol{M}}}^{-1} r$;
  $\beta \leftarrow \gamma$;
  $\gamma \leftarrow \underline{\boldsymbol{r}}^T \underline{\boldsymbol{v}}$;
  $\beta \leftarrow \gamma/\beta$;
  $\underline{\boldsymbol{p}} \leftarrow \underline{\boldsymbol{v}} + \beta \cdot \underline{\boldsymbol{p}}$;

- $\underline{\underline{\boldsymbol{A}}}\,\underline{\boldsymbol{d}}$ evaluated matrix-free
- vectors live either on host or device
- CG algorithm can operate on both types of vectors
- CPU and GPU implementation of vector operations

- note: $\underline{\underline{\boldsymbol{M}}} := \mathrm{diag}(\underline{\underline{\boldsymbol{A}}})$ here!

# Reformulated (merged) PCG

**Algorithm 2:** Cache friendly CG preconditioned by a diagonal

**Data:** $\boldsymbol{A}$, $x_0$, $b$, $\boldsymbol{M}^{-1}$
**Result:** $x$

[startup];
**for** $k = 0,1,\ldots$ **do**
  $\underline{\boldsymbol{r}} \leftarrow \underline{\boldsymbol{r}} - \alpha \underline{\boldsymbol{v}} \quad \underline{\boldsymbol{x}} \leftarrow \underline{\boldsymbol{x}} + \alpha \underline{\boldsymbol{p}} \quad \underline{\boldsymbol{p}} \leftarrow \underline{\underline{\boldsymbol{M}}}^{-1} \underline{\boldsymbol{r}} + \beta \underline{\boldsymbol{p}}$ ; // pre
  $\underline{\boldsymbol{v}} \leftarrow \underline{\underline{\boldsymbol{A}}} \underline{\boldsymbol{p}}$ ; // vmult
  $a \leftarrow \underline{\boldsymbol{p}}^T \underline{\boldsymbol{v}} \quad b \leftarrow \underline{\boldsymbol{r}}^T \underline{\boldsymbol{r}} \quad c \leftarrow \underline{\boldsymbol{r}}^T \underline{\boldsymbol{v}} \quad d \leftarrow \underline{\boldsymbol{v}}^T \underline{\boldsymbol{v}} \quad e \leftarrow \underline{\boldsymbol{r}}^T \underline{\underline{\boldsymbol{M}}}^{-1} \underline{\boldsymbol{r}} \quad f \leftarrow \underline{\boldsymbol{r}}^T \underline{\underline{\boldsymbol{M}}}^{-1} \underline{\boldsymbol{v}} \quad g \leftarrow \underline{\boldsymbol{v}}^T \underline{\underline{\boldsymbol{M}}}^{-1} \underline{\boldsymbol{v}}$ ; // post
  $\alpha \leftarrow e/a$;
  **if** $\sqrt{b + 2\alpha c + \alpha^2 d} < \varepsilon$ **then**
    $\underline{\boldsymbol{x}} \leftarrow \underline{\boldsymbol{x}} + \alpha \underline{\boldsymbol{p}}$;
    **return**
  $\beta \leftarrow (e - 2\alpha f + \alpha^2 g)/e$;

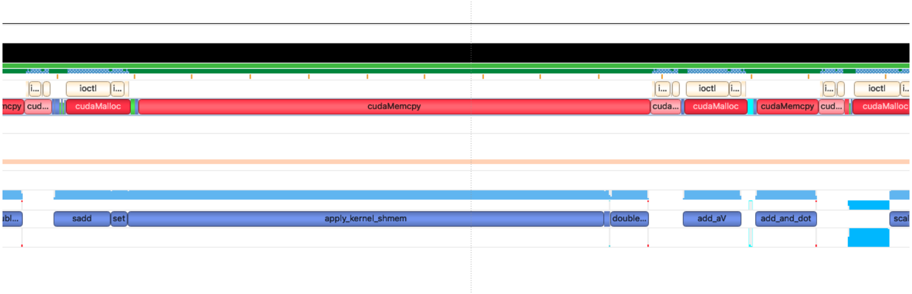
Observation: Works very well for CPUs!

Part 2:

# Optimization steps

# Base (single node/single GPU)

Profile of a single iteration (with NVIDIA Nsight):

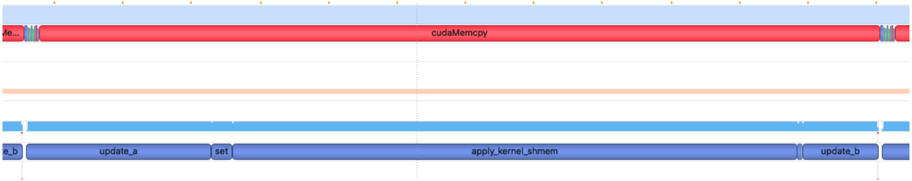



Observations:

- + schoolbook-like: easy to understand; working for any preconditioners
- - (too) many kernel calls and redundant memory loads

# Reformulated algorithm (single node/single GPU)

## Profile of a single iteration (with NVIDIA Nsight):



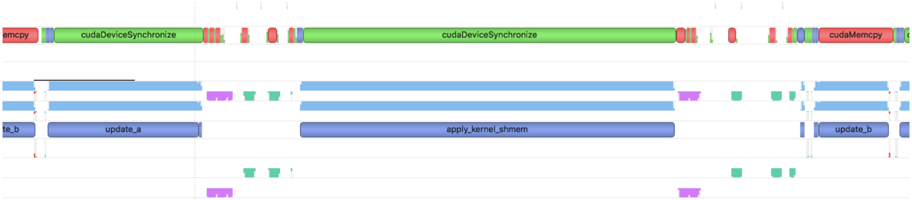

## Observations:

+ 3 kernel calls[3]
- limited to diagonal preconditioner

[3] We eliminated "set" and merged into "update_a" in a next step (not shown here) with reduced memory access.

# Reformulated algorithm (multi node/single GPU)

## Profile of a single iteration (with NVIDIA Nsight):



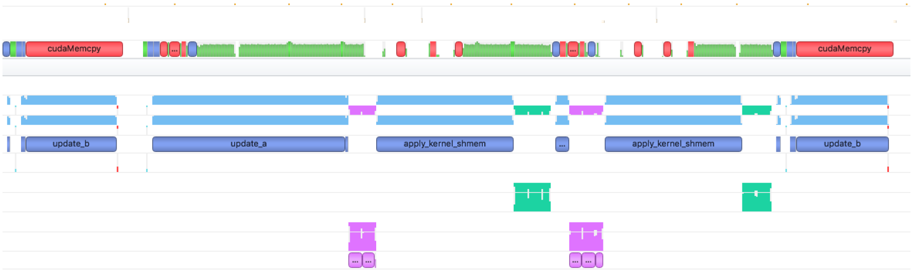

**”update ghost values”** **”computation”** **”compress”**

## Observations:

- No overlap of communication and computation.

# Overlapping communication and computation (multi node/single GPU)

Profile of a single iteration (with NVIDIA Nsight):



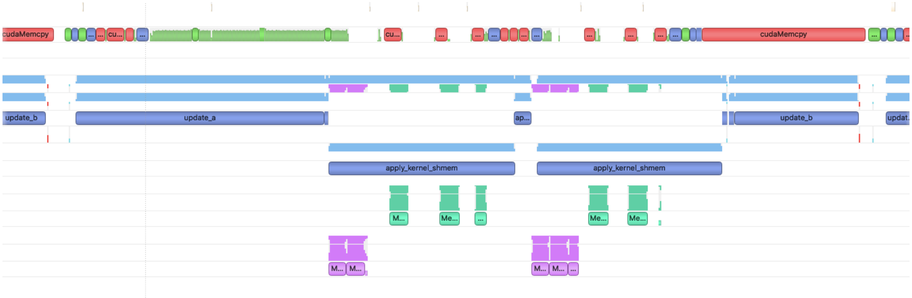

**”update ghost values”** **”compress”**

Split up matrix-vector multiplication into three parts → Observations:

+ Overlap of (actual) communication and computation.
- MPI (MPI_Isend, MPI_Waitall) memcopies data (DtoH and HtoD), which blocks communication.

# Overlapping MPI memcopy with computation (multi node/single GPU)

Profile of a single iteration (with NVIDIA Nsight):




Move first and last part of matrix-vector multiplication on to a new streams → Observations:

+ Implicit synchronization (by default stream).
+ Perfect overlap.

# Running multiple processes on a node

- work in progress
- hope: better utilization of the GPU and the network card

# Final code

The final code can be found online:

- `https://github.com/peterrum/deal-and-ceed-on-gpu`
- `https://github.com/peterrum/dealii/tree/dealii-on-gpu`

Part 3:

# Results

# Intel Skylake vs NVIDIA Volta (p/q=4/5) vs Pascal (p/q=5/6)

**Achieved throughput:**

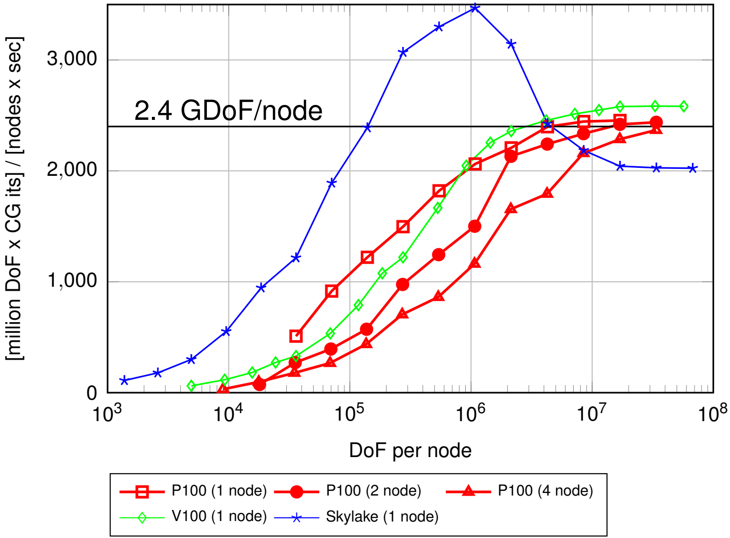
- **Pascal (4×): 7.6 GDoF**

**Total peak performance:**

- **Pascal: 4.7 TFlop/s**
- Volta: 7.8 TFlop/s
- Skylake: 4.2 TFlop/s

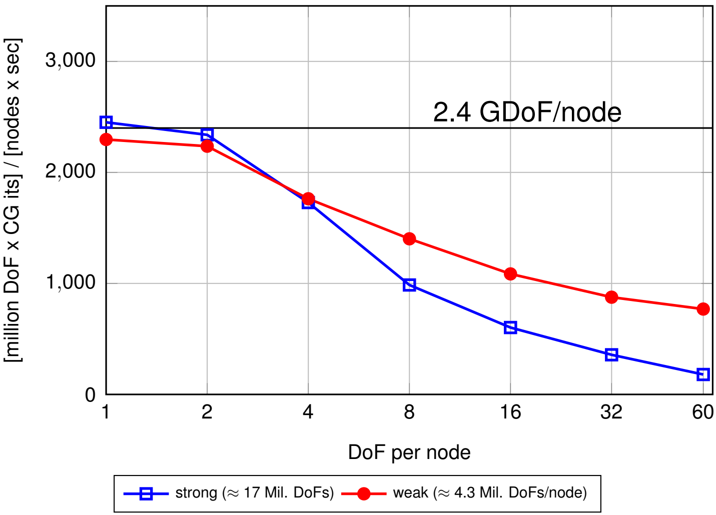
**Maximum memory bw:**

- **Pascal: 732 GB/s**
- Volta: 900 GB/s
- Skylake: 191 GB/s




*Note: Skylake/Pascal run merged PCG. Volta runs standard implementations!*

# Scaling with up to 60 P100 nodes on Piz Daint



# Variants of BP5 on Pascal (p/q=5/6)

Variant 1:

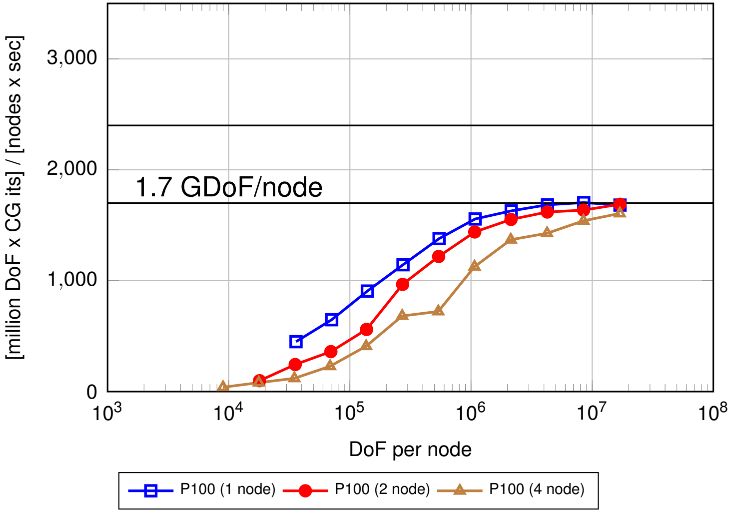
- do not merge coefficient

$$(J_q^{-1}|J_q|w_q J_q^{-T})$$

  in quadrature points
- instead: load $J_q$, $|J_q|$
- i.e.: load 10 vs. 6 doubles
- (additional work)

**Observation:**

- **Performance drop higher than expected!**



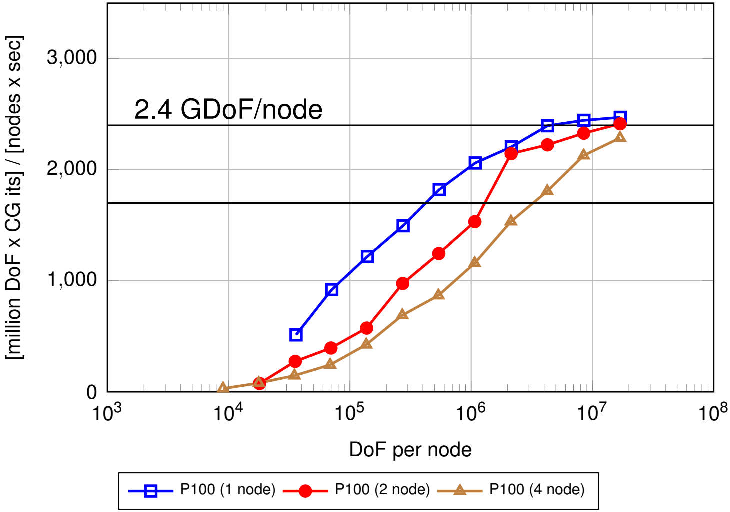
# Variants of BP5 on Pascal (p/q=5/6) (cont.)

Variant 2:

- integer in Gauss quadrature points
- requires interpolation
- additional $2 \cdot dim$ sum-factorization sweeps

**Observation:**

- **Basis transformation comes for free!**



Part 4:

# Conclusions and lessons learned

# Conclusions and lessons learned

- GPU programming is fun (although we did not have to do much), except the redundant conversion of block- and thread-id
- similar optimization strategies work for CPU and GPU
(e.g. minimize data access overlapping communication and computation)
- given a heavily optimized FEM code (matrix-free exploiting caches and SIMD), the usage of GPUs will not lead to a significant speed-up

## Outlook

- Step-by-step integration of the new features in deal.II.
- Extend the functionality of MatrixFree (GPU) so that it matches MatrixFree (CPU).
  - geometric multigrid and hybrid multigrid methods[4]
  - discontinuous Galerkin methods
  - over-integration $p+1 \ll q$
- Investigation of compute nodes with multiple GPUs.
- Build a minimal build configuration for GPU-only runs.

[4] P. Munch, An efficient hybrid multigrid solver for high-order discontinuous Galerkin methods, Master's thesis, 2018 (online available: `https://mediatum.ub.tum.de/node?id=1514962`)

Thanks to the organizers and the mentors!