ExaDG: High-order discontinuous Galerkin for the exa-scale

Daniel Arndt[1] Niklas Fehn[3] Guido Kanschat[1]
Katharina Kormann[2] Benjamin Krank[3] Martin Kronbichler[3]
Wolfgang A. Wall[3] Julius Witte[1]

[1]Interdisziplinäres Institut für Wissenschaftliches Rechnen, Universität Heidelberg

[2]Max–Planck–Institut für Plasmaphysik

[3]Institute for Computational Mechanics, Technical University of Munich

March 20, 2017

Outline

Results operator evaluation

Multigrid scalability

Multigrid smoother

Applications

Generic operator evaluation framework

Operator evaluation by **fast integration** = matrix-vector products in solvers, nonlinear residuals, ...

- Integrated in `deal.II` finite element library (`www.dealii.org`)
- Target **generic weak forms** on quadrilaterals
- Mesh loop according to specific algorithm structure
  - Loop over quadrature points and cells/faces/vertices
  - Full support for mesh adaptivity with hanging nodes for L_2- and H^1-conforming elements, work in progress for H^{div}-, H^{curl}
  - **Parallelization**: MPI (domain decomposition), threads
  - **Vectorization**

Matrix-free algorithm for $v = Au$ with $A = \sum_K P_K^T A_K P_K$

$v = 0$
loop over cells:
- (i) Extract local vector values on cell: $u_K = P_K u$
- (ii) Apply operation locally on cell: $v_K = A_K u_K$ by fast integration
- (iii) Sum results from (ii) into the global solution vector: $v = v + P_K^T v_K$

Matrix-vector product for Laplacian

Examples:

- Laplacian for continuous elements

$$(Au)_j = \int_\Omega a(\mathbf{x}) \nabla_{\mathbf{x}} \phi_j \cdot \nabla_{\mathbf{x}} u^h d\mathbf{x} = \sum_{K=1}^{n_{\text{cells}}} \int_K a(\mathbf{x}) \nabla_{\mathbf{x}} \phi_j \cdot \nabla_{\mathbf{x}} u^h d\mathbf{x}$$

- DG Laplacian with symmetric interior penalty discretization

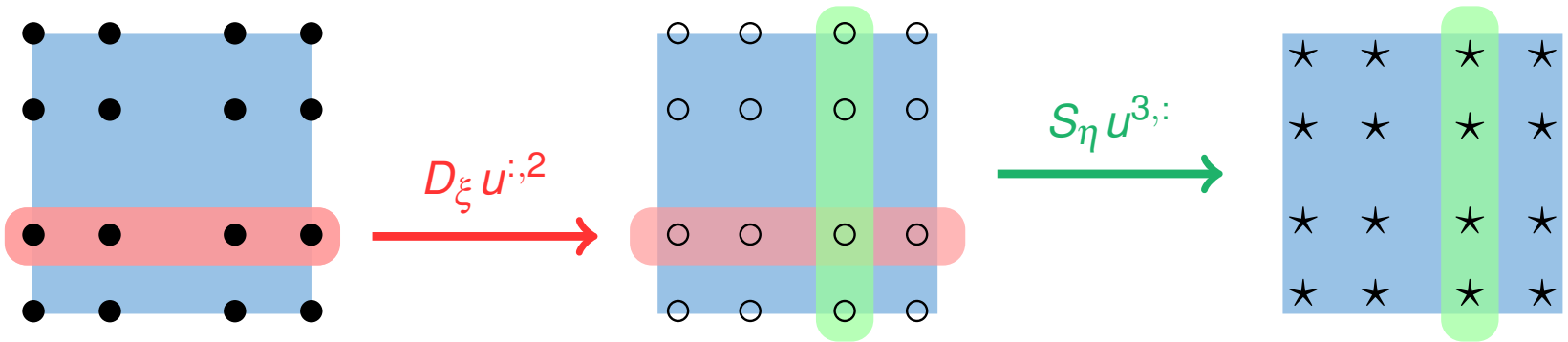
$$\begin{aligned}(Au)_j = \sum_{K=1}^{n_{\text{cells}}} \Bigg(& \int_K a \nabla_{\mathbf{x}} \phi_j \cdot \nabla_{\mathbf{x}} u^h d\mathbf{x} - \int_{\partial K} \phi_j \mathbf{n}^- \cdot \frac{a \nabla_{\mathbf{x}} u_h^- + a \nabla_{\mathbf{x}} u_h^+}{2} d\mathbf{s} \\ & + \int_{\partial K} a \frac{\nabla_{\mathbf{x}} \phi_j}{2} \cdot \mathbf{n}^- (u_h^- - u_h^+) d\mathbf{s} + \int_{\partial K} \phi_j a \sigma (u_h^- - u_h^+) d\mathbf{s} \Bigg)\end{aligned}$$

Compute volume integral element by element

$$\begin{aligned}(A_K u_K)_j &= \int_K a \nabla_{\mathbf{x}} \phi_j \cdot \nabla_{\mathbf{x}} u^h d\mathbf{x} \approx \sum_q w_q \det J_q a \nabla_{\mathbf{x}} \phi_j \cdot \nabla_{\mathbf{x}} u^h \Big|_{\mathbf{x}=\mathbf{x}_q} \\ &= \sum_q \nabla_\xi \phi_j J_q^{-1} (a w_q \det J_q) J_q^{-\mathrm{T}} \sum_i \nabla_\xi \phi_i u_{K,i} \Big|_{\mathbf{x}=\mathbf{x}_q}, \quad j = 1, \ldots, \text{cell\_dofs}\end{aligned}$$

Fast computation of integrals

- Fast integration routines for quadrilaterals/hexahedra
- Evaluation of tensor product shape functions by sum factorization



Vector values u_K on nodes

$\frac{\partial}{\partial\xi} u^h$ on quadrature points

$$\frac{\partial}{\partial\xi} u^h(\xi_q, \eta_q)\big|_{\text{q\_points}} = (D_\xi \otimes S_\eta)\mathbf{u}_K$$

Dense matrix-matrix multiplication

$$S_\eta \mathbf{U}_K D_\xi^T$$

Evaluation cost:

$\mathcal{O}(2d(k+1)^{d+1})$ (degree k, dimension d)

Naive evaluation: $\mathcal{O}(d(k+1)^{2d})$

- Techniques established in spectral elements
- Similar techniques as used by Exa-DUNE

Result 1: Extension to throughput architectures

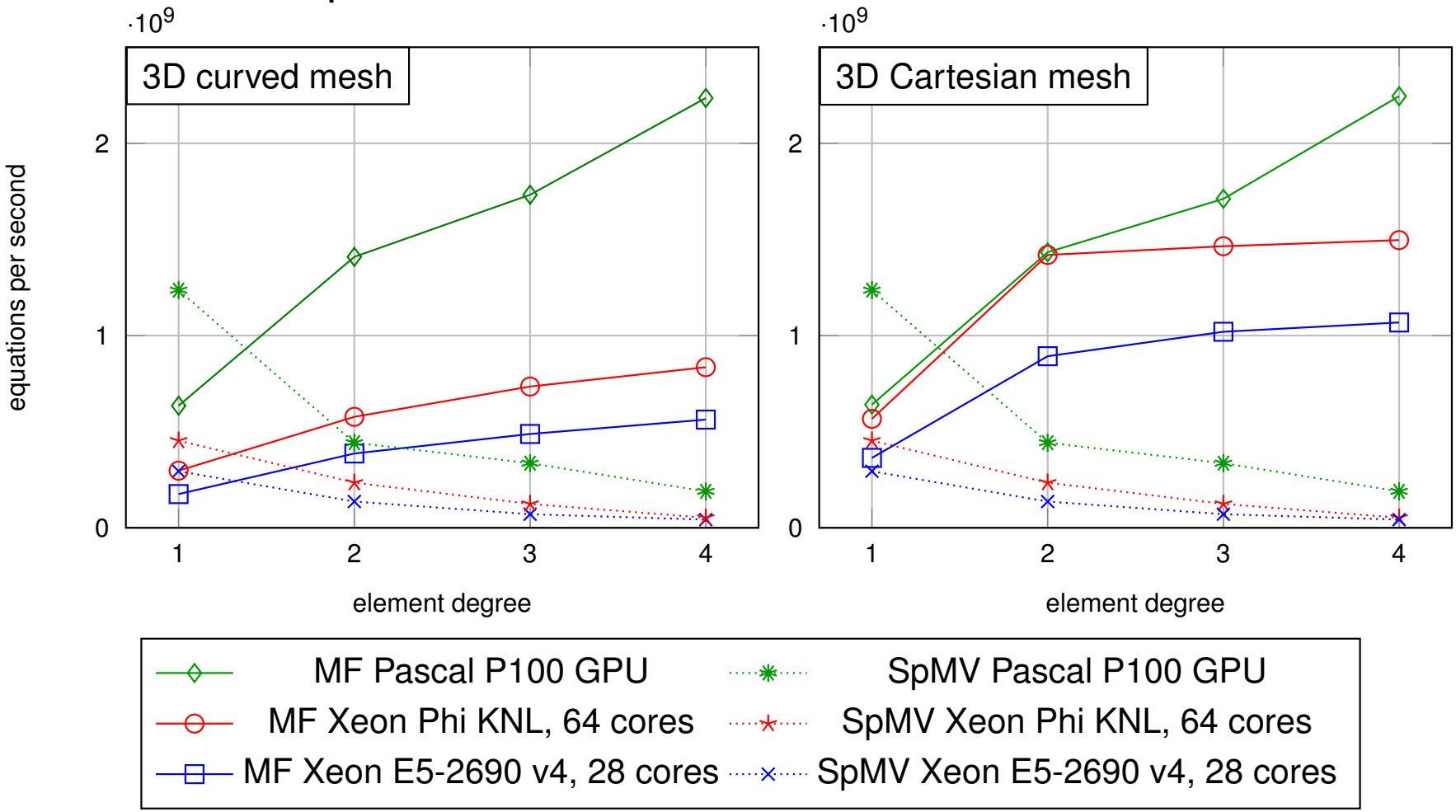
In 2016: Extension of algorithms from classical CPUs to **throughput architectures**

- Evaluation and development on **Intel Xeon Phi Knights Landing**[1]
  - New intrinsics, gather + scatter
  - Loop parallelization
- Evaluation and development on **NVIDIA GPUs**
  - Separate, more restricted code
  - Similar interfaces that allow switching by changing typedefs
  - Collaboration with Karl Ljungkvist, Uppsala University
  - Evaluation on NVIDIA P100 (Pascal)
  - Extension to multi-GPU setup with NVLink-type interconnects ongoing
  - Extension to DG planned

[1] Kronbichler, Kormann, Pasichynk, Allalen, "Fast Matrix-Free Discontinuous Galerkin Kernels on Modern Computer Architectures", accepted paper for ISC17

CPU vs Phi vs GPU

Evaluation of Laplacian for continuous $\mathcal{Q}_1$ to $\mathcal{Q}_4$ elements

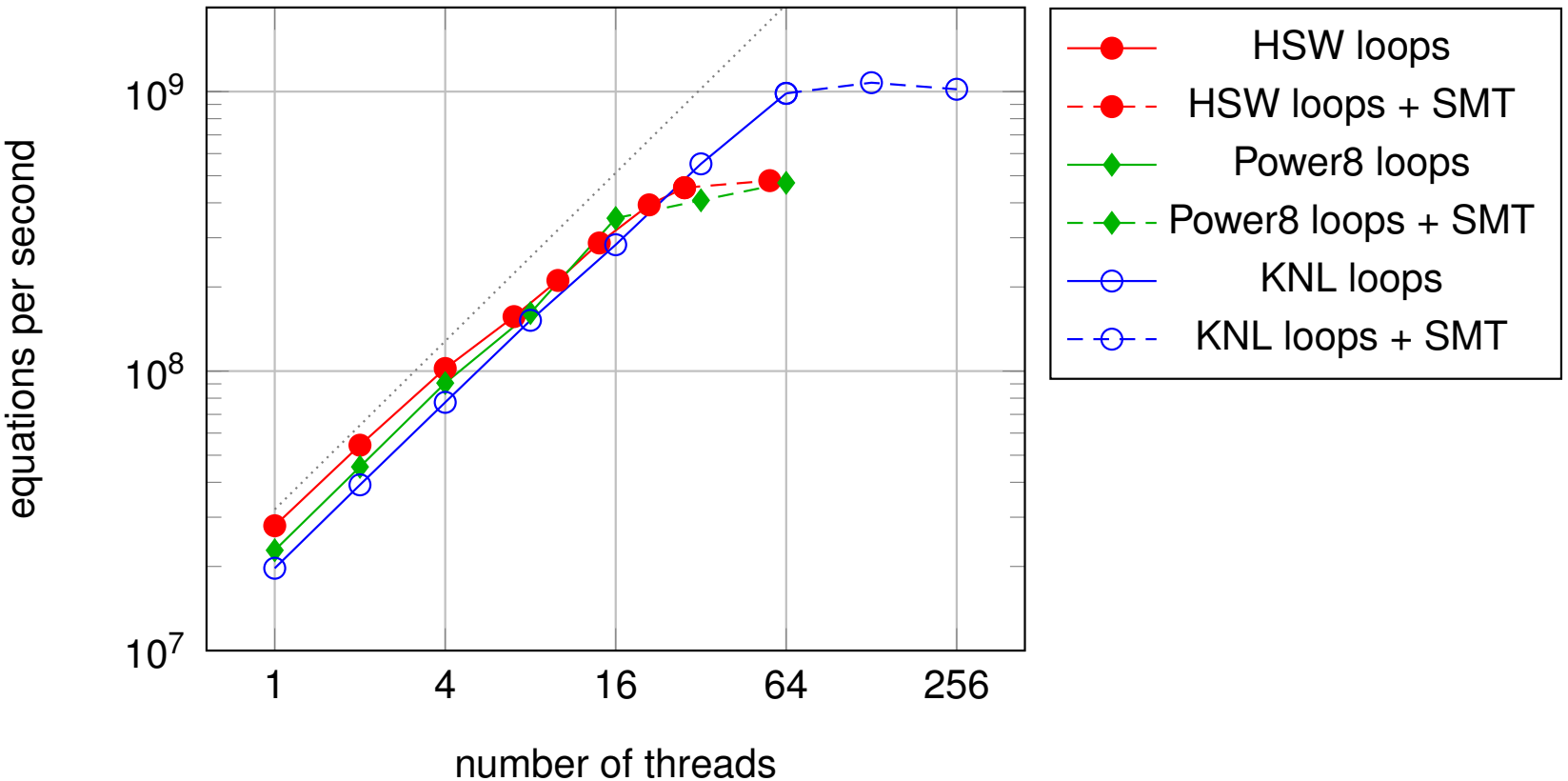


P100 2.3× **faster** than KNL, **KNL** 1.4× **faster** than 2-socket **Broadwell** on curved mesh (370 GB/s vs. 160 GB/s vs. 105 GB/s mem-bw)

Access to KNL and P100 kindly provided by LRZ, Garching with support from KONWIHR

CPU vs Phi vs Power8 on DG

Laplacian discretized by discontinuous $\mathcal{Q}_4$ elements, 32.8m DoFs, comparison of shared-memory parallelizations[1]: **KNL** 2.4× **faster**

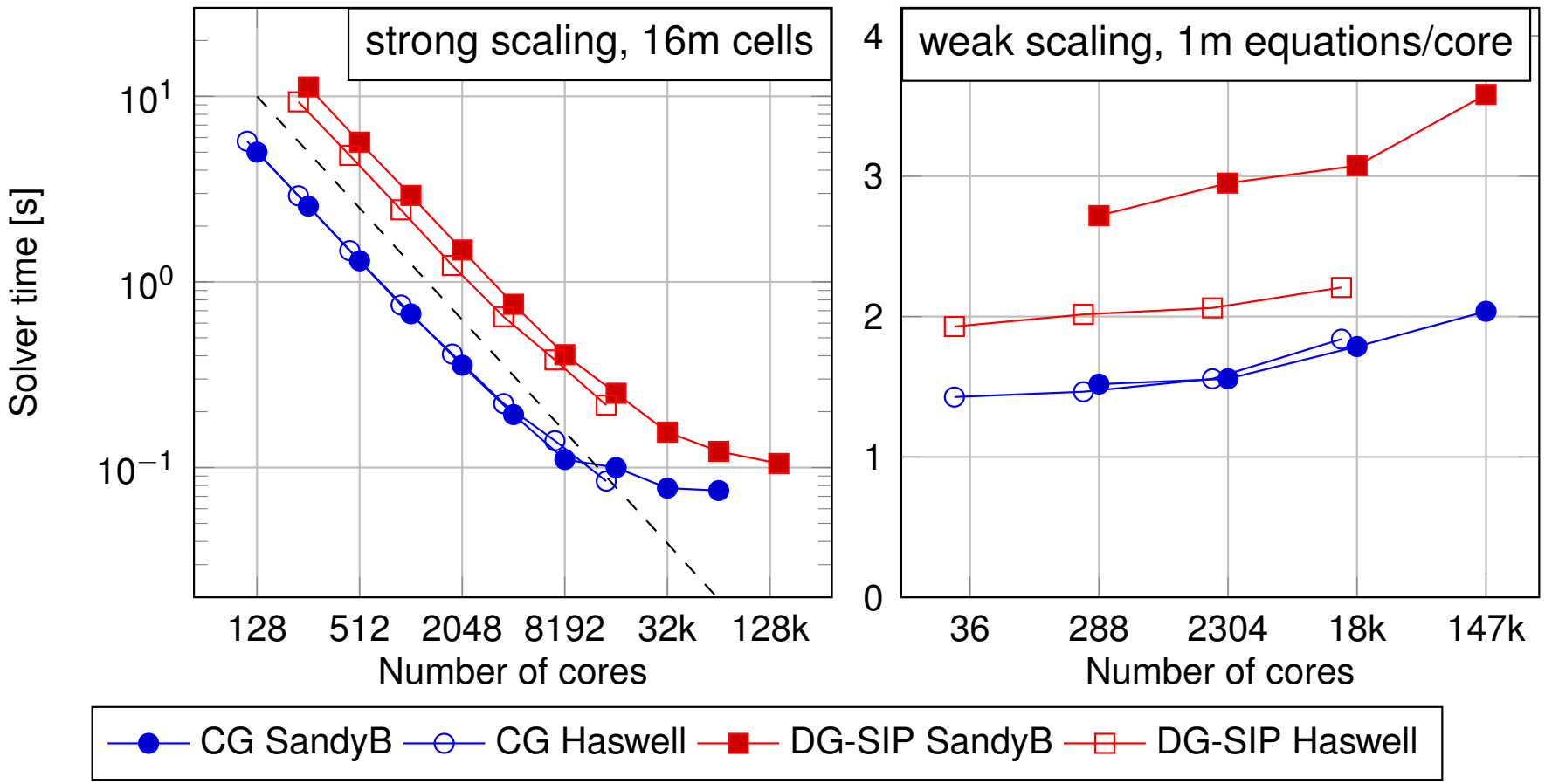


Access to KNL and Power8 kindly provided by LRZ, Garching with support from KONWIHR

[1] Kronbichler, Kormann, Pasichynk, Allalen, "Fast Matrix-Free Discontinuous Galerkin Kernels on Modern Computer Architectures", accepted paper for ISC17

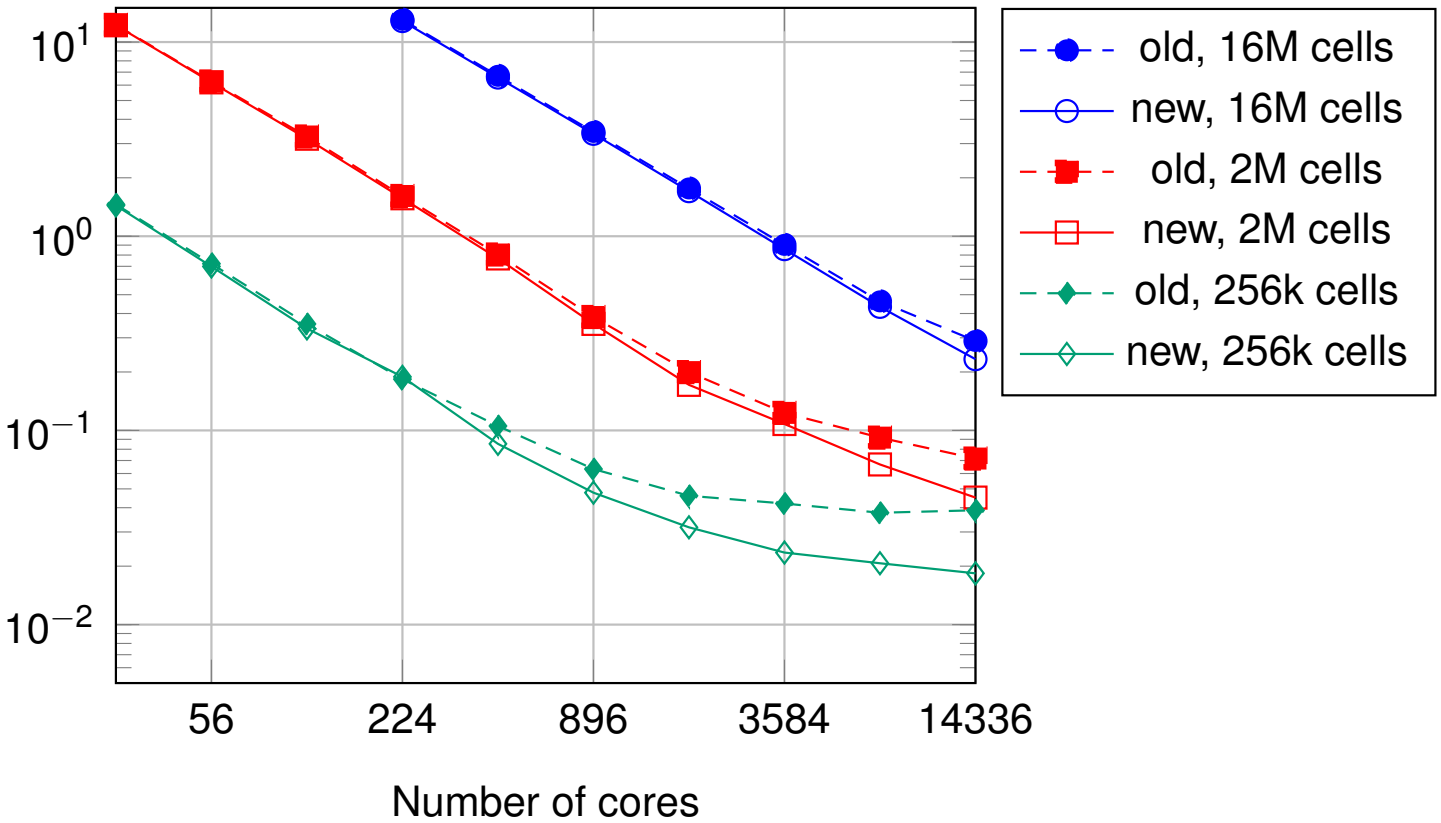
Result 2: Improvements to multigrid scalability

Conjugate gradient solver for Laplacian on $\mathscr{Q}_3$ elements, preconditioned by **geometric multigrid** V-cycle with fast matrix-vector products, Chebyshev smoother of degree 5, on full SuperMUC phase 1 (SandyB) and up to 14k cores on SuperMUC phase 2 (Haswell)



Fixed bottlenecks in MG infrastructure

Experiments on SuperMUC phase 2 (Haswell) with discontinuous $\mathcal{Q}_3$ elements, 6 CG iterations with **geometric multigrid** V-cycle, Chebyshev smoother

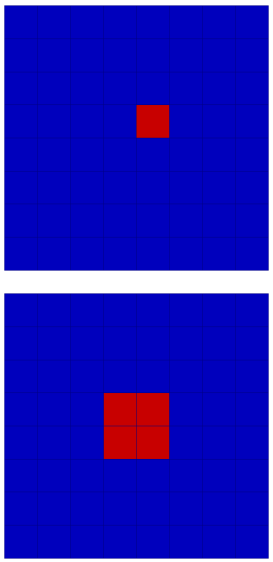


Improvements in smoother and level transfer infrastructure $\rightarrow$ **lower latency**

Result 3: Multigrid Smoother - Dictionary

Block Jacobi/additive Schwarz smoother

- Take the local structure of the problem into account
- Use local problems for preconditioning



$$R_l = \sum_{K \in \mathcal{T}_l} P_K A_K^{-1}$$

- Cell or vertex patches

Often not all the local inverses are different or at least they only differ slightly

- Idea: Store **only local inverses that differ** significantly
- Hope: Comparable results to block Jacobi smoothing
  Better run-time

$$\|A_i^{-1} A_j - Id\| < \text{tol} \cdot n^2 \quad \Longrightarrow \quad A_j^{-1} \approx A_i^{-1}.$$

Additive Schwarz Smoother - Results

- Variable-coefficient Laplacian with $a(\mathbf{x}) = \dfrac{1}{\alpha + \beta \|\mathbf{x}\|^2}$
- Discretization with discontinuous $\mathscr{Q}_2$ elements, $(0,1)^2$ Cartesian mesh, conjugate gradient solver preconditioned by V-cycle, block Jacobi relaxation parameter 0.7
- Test case: $\alpha = 0.05$, $\beta = 100$

BlockJacobi

separate ("exact") local cell matrix

SameDiagonal

use one cell and scale by coefficient at the midpoint of the local cell for the rest

Dictionary

as above, tolerance 0.1

	BlockJacobi		SameDiagonal		Dictionary	
L	n	time	n	time	n	time
2	11	0.00344	15	0.00433	12	0.00508
3	14	0.00561	29	0.0121	17	0.00716
4	14	0.0106	40	0.0298	16	0.0115
5	15	0.0298	43	0.0781	18	0.0314
6	15	0.098	39	0.228	18	0.099
7	16	0.422	37	0.805	19	0.332
8	16	2.54	37	3.28	20	1.44
9	16	11	38	13.5	21	6.9
10	17	54.3	37	55.8	22	32.2

Additive Schwarz Smoother - Results

- Variable-coefficient Laplacian with $a(\mathbf{x}) = \dfrac{1}{\alpha + \beta \|\mathbf{x}\|^2}$
- Discretization with discontinuous $\mathscr{Q}_2$ elements, $(0,1)^2$ Cartesian mesh, conjugate gradient solver preconditioned by V-cycle, block Jacobi relaxation parameter 0.7
- Test case: [1] $\alpha = 0.05$, $\beta = 100$, [2] $\alpha = 0.01$, $\beta = 1000$

BlockJacobi

separate ("exact") local cell matrix

SameDiagonal

use one cell and scale by coefficient at the midpoint of the local cell for the rest

Dictionary

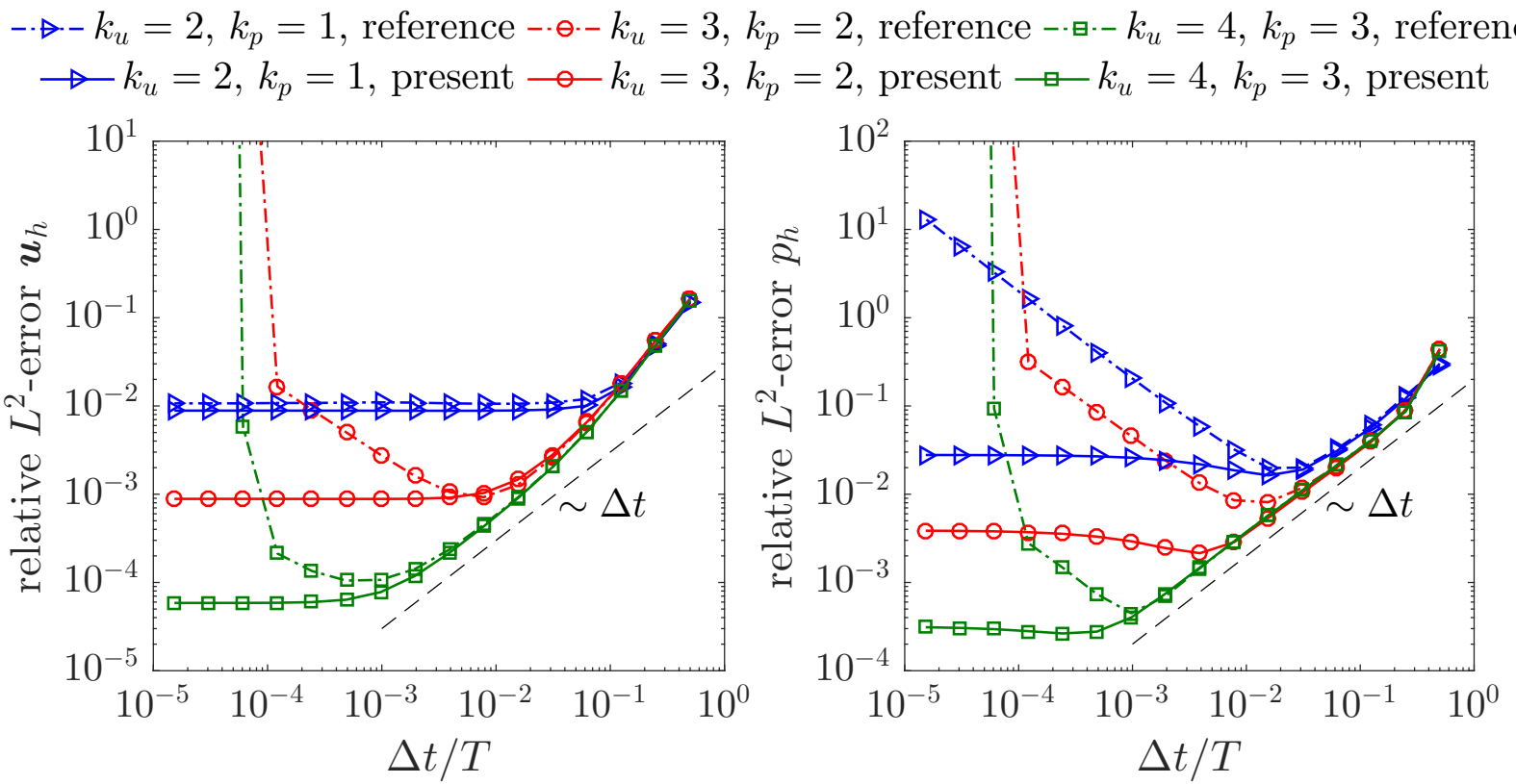
as above, tolerance 0.1

	BlockJacobi		SameDiagonal		Dictionary	
L	[1]	[2]	[1]	[2]	[1]	[2]
2	11	10	15	14	12	11
3	14	13	29	30	17	15
4	14	14	40	43	16	17
5	15	15	43	63	18	18
6	15	16	39	92	18	19
7	15	15	37	212	19	19
8	16	16	37	210	20	19
9	16	16	38	151	21	20
10	17	17	37	185	22	20

Result 4: Towards complex application codes

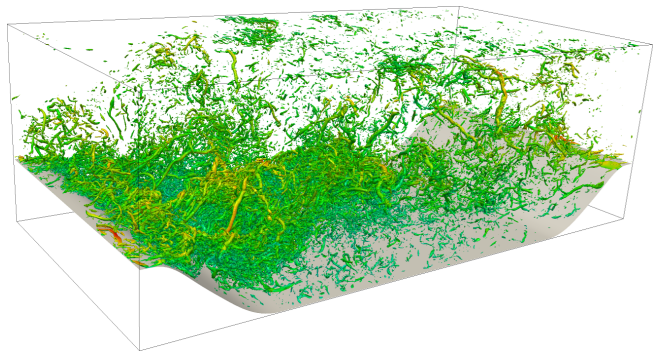
- Goal: Complex discontinuous Galerkin codes for fluid dynamics
- Step 1: Flexible incompressible Navier–Stokes solvers
  - Time stepping: fully coupled, velocity correction, pressure correction
  - Literature on DG with interior penalty discretization and splitting schemes unconvincing $\rightarrow$ needed own developments

Stability in limit of small time steps:

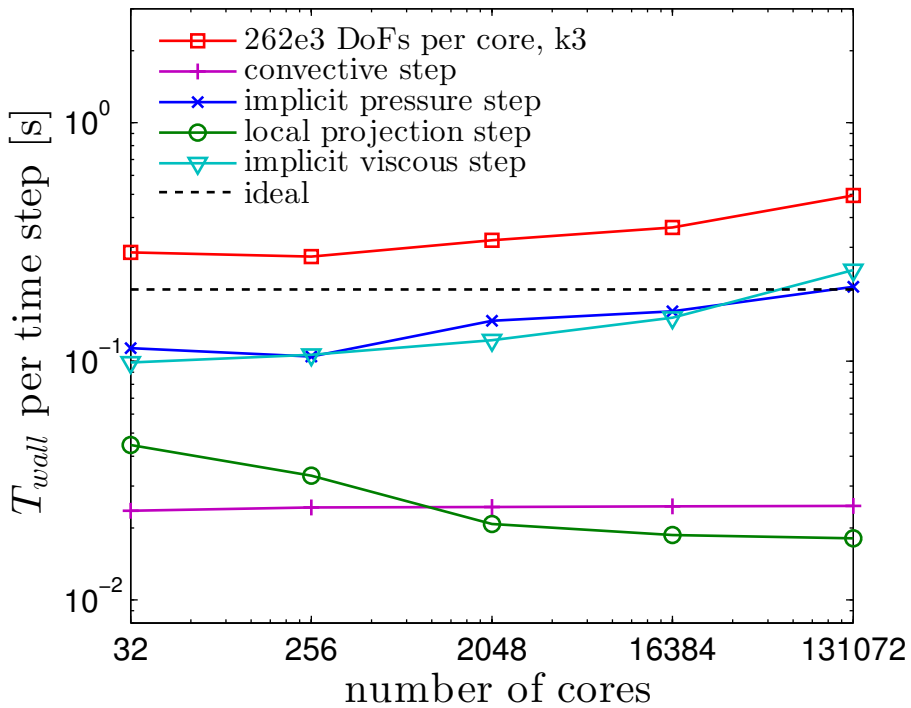


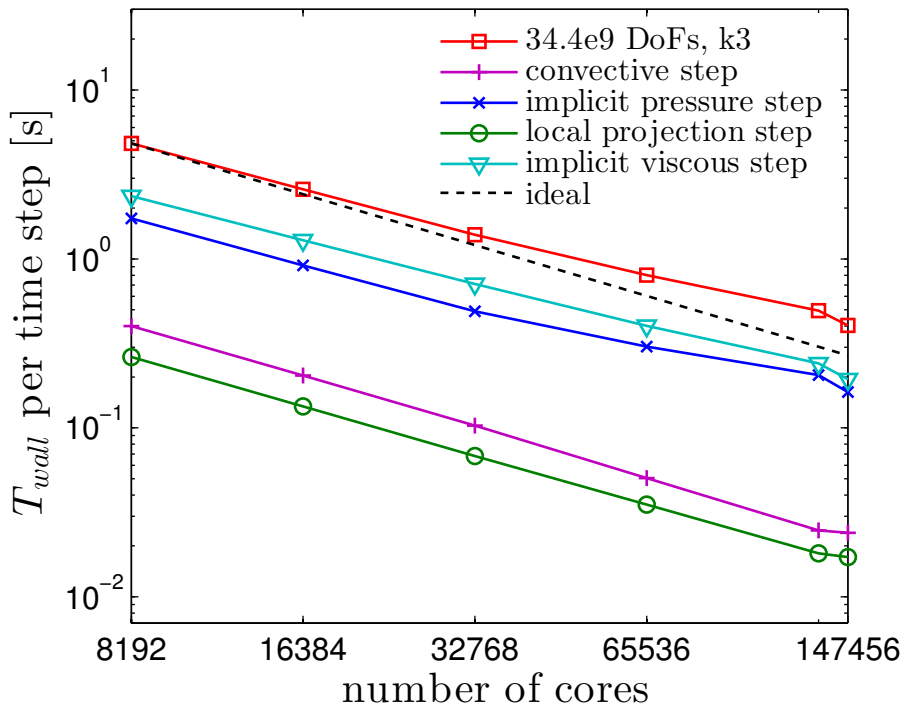
Result 4: Scalability of Navier–Stokes solvers

- Developed for turbulent flow settings, verified for turbulent channel flow and flow past periodic hills (see right)
- Weak and strong scaling on SuperMUC Phase 1 for channel flow



turbulent vortices visualized by Q-criterion





Outlook

- Better multigrid smoothers based on block Jacobi-like methods
  - Support for iterative solvers on elements (vectorized + optimized)
  - Tensor product inverses (extend concepts from spectral elements to DG)
- Robust solvers/smoothers for convection-dominated flows
- Flexibility of framework
  - Other elements than usual continuous and discontinuous ones
  - Enrichments
  - Local time stepping
- Development of domain-specific form language in C++ to facilitate use of tensor kernels