High-performance Computing for Flows in Porous Media

Peter Bastian and Steffen Müthing

Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
Im Neuenheimer Feld 368, D-69120 Heidelberg

Universität Heidelberg | Zukunft. Seit 1386.

Heidelberg-Prag Workshop, April 29, 2015

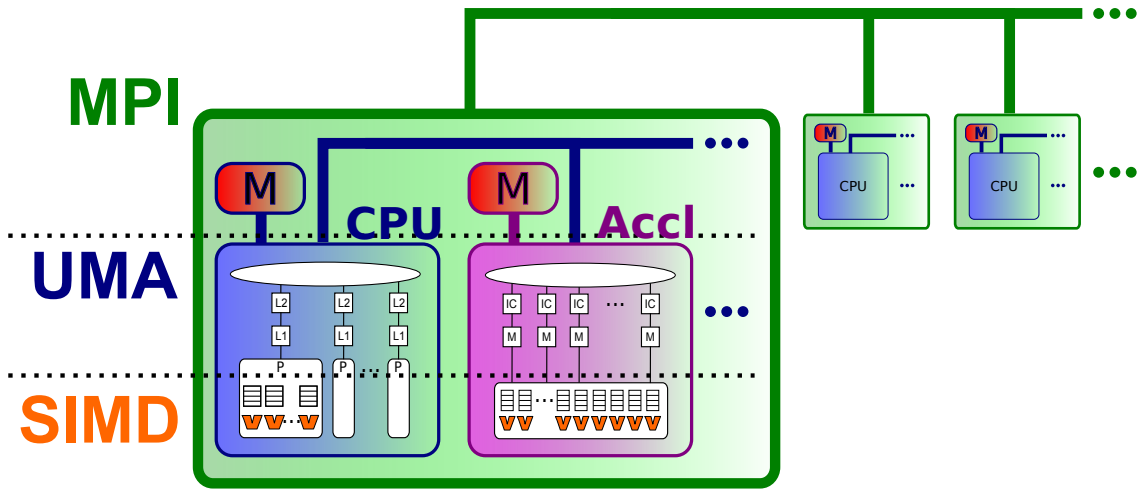
Contents

Trends in Computer Architecture

- Power wall
  - Power consumption is the limiting factor for exascale computing!
  - Tianhe 2: 33,8 PF (Linpack), 17.8 MW $\Rightarrow$ 500 MW
  - Clock rate stagnates but Moore's law is still valid
- Memory wall
  - Bandwidth not sufficient to sustain anywhere near peak performance
  - (Still) significant latency for main memory access
  - Memory access is energetically costly
  - Memory is hierarchical: register, cache, local, remote, ...
- ILP wall
  - Automatic extraction of instruction level parallelism stagnates
  - Revival of vectorization in form of SIMD instructions
- Other
  - Heterogeneous node architectures: CPU + coprocessors
  - Reduced reliability due to immense number of components?

EXA-SCALE $\Rightarrow 10^6$ **cores** $\times\ 10^3$ **threads** $\times\ 10^9$ **GFLOPS**

Typical Architecture



(Figure by Christian Engwer)

- Three levels of parallelism
  - Instruction level parallelism: SIMD vector units
  - Multithreading/shared memory: multiple sockets, multiple cores
  - Message passing: nodes connected by fast interconnect
- Coprocessor architectures
  - Triggered by economical and then power considerations
  - CPUs are catching up: E5-2699v3: 0.9TF@145W, K40: 1.4TF@235W
  - My opinion: coprocessor architectures will not last ...

What Does This Mean for PDE Numerics?

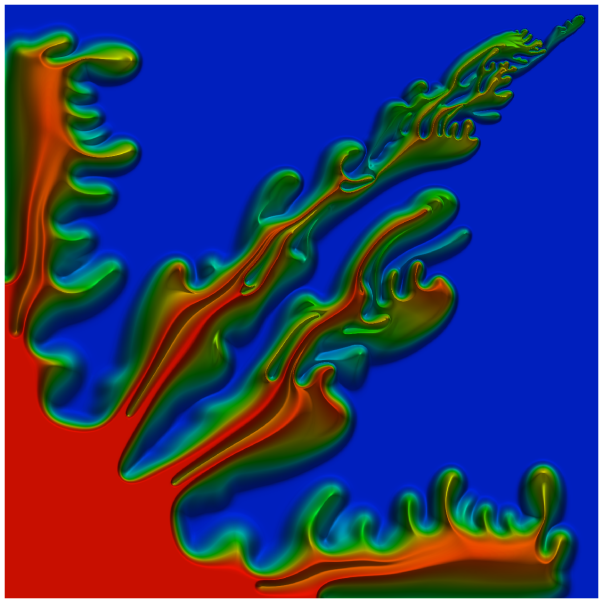
- Exploit hybrid parallelism:
  MPI + shared memory + SIMD,
  i.e. Intel TBB + auto vectorization
- Transparent accelerator support (concentrate on Intel Phi)
- FLOPS are for free, memory access is expensive
- Matrix-based methods
  - Robust preconditioners available (AMG)
  - Poor performance, bound by memory bandwidth
- Matrix-free methods
  - Restricted to explicit time-stepping or certain solvers, e.g. CG, GMG
  - Low-order FE schemes:
    - ★ Exploit problem structure: linear transformation, regular grids, constant coefficients, ...
    - ★ Vectorization over several elements
  - High-order FE schemes:
    - ★ Complexity reduction through tensor-product approaches
    - ★ Medium high order: Vectorization within one element
    - ★ Very high order: Several threads to one element

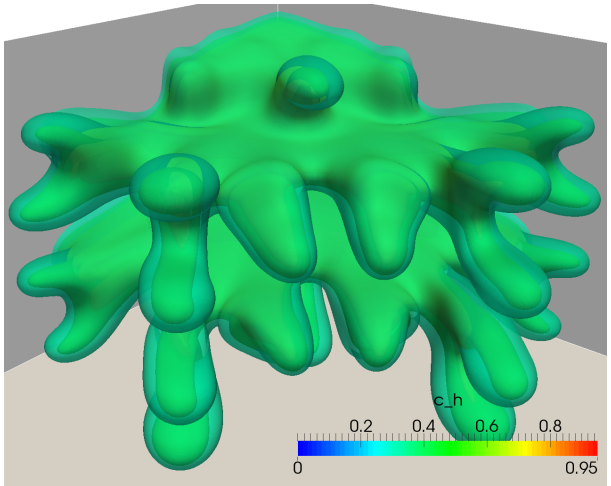
Unstable Flows in Porous Media

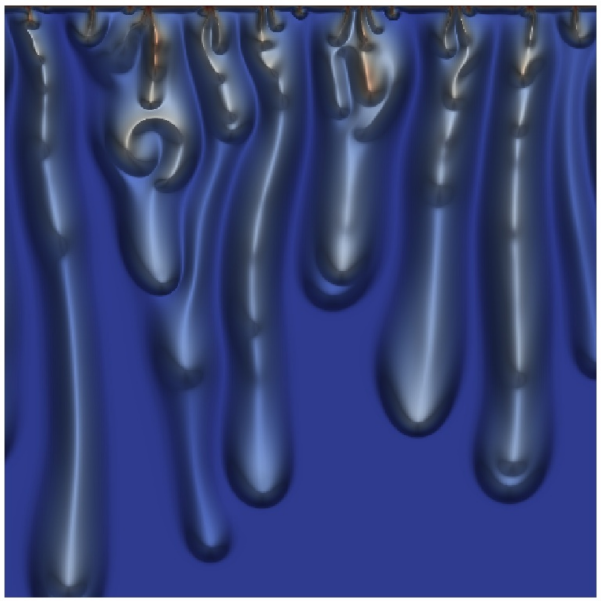
$$\nabla \cdot v = 0, \qquad v = -\frac{K}{\mu(c)}(\nabla p - \rho(c)g)$$

$$\partial_t(\Phi c) + \nabla \cdot (cv - D(v)\nabla c) = 0$$

Density driven flow, miscible discplacement
Can exploit high resolution schemes







Contents

Discontinuous Galerkin (DG) Finite Element Methods

- Advantages
  - Potentially higher order of convergence
  - Higher-order methods beneficial even without sufficient regularity
  - Element-wise conservative
  - Full permeability tensors, discontinuous coefficients
  - Handles elliptic, parabolic and hyperbolic equations equally well
  - Unstructured grids, nonconforming refinement
  - Contiguous memory access
  - Block matrix structure: less indirect access, BLAS level 3

- Disadvantages
  - No monotonicity
  - Free parameter to tune
  - Higher number of DOFs compared to standard FE or FV
  - Linear systems are more difficult to solve

DG for Flow Equation

$$a_h(p,w;c) = \sum_{T\in\mathcal{T}_h}\int_T [\mu(c)^{-1}K(\nabla p - \rho(c)g)]\cdot\nabla w - \sum_{F\in\mathcal{F}_h^{iD}}\int_F \nu_F\cdot\{\mu(c)^{-1}K(\nabla p-\rho(c)g)\}_\omega [\![w]\!]$$

$$- \sum_{F\in\mathcal{F}_h^{iD}}\int_F \theta\nu_F\cdot\{\mu(c)^{-1}K\nabla w\}_\omega [\![p]\!] + \sum_{F\in\mathcal{F}_h^{iD}}\int_F \gamma_{F,f}[\![w]\!][\![p]\!]$$

$$l(w) = \sum_{T\in\mathcal{T}_h}\int_T q_f w - \sum_{F\in\mathcal{F}_h^{N}}\int_F j_f w \qquad \theta = 1 \text{ (SIPG)}, \ \theta = 0 \text{ (OBB)}, \ \theta = -1 \text{ (NIPG)}$$

SIPG + weighted averages (*Di Pietro, Ern, Guermond '2008*):

$$\delta_{Kn}^{\pm} = \nu_F^t K^{\pm}\nu_F/\mu(c)^{\pm}, \qquad \omega^- = \frac{\delta_{Kn}^+}{\delta_{Kn}^- + \delta_{Kn}^+}, \qquad \omega^+ = \frac{\delta_{Kn}^-}{\delta_{Kn}^- + \delta_{Kn}^+}.$$

Penalty term ($\langle\cdot,\cdot\rangle$ denotes harmonic average, k polynomial degree):

$$\gamma_{F,f} = \alpha\langle\delta_{Kn}^-,\delta_{Kn}^+\rangle M, \qquad M = k(k+n-1)\frac{|F|}{\min(|T^-(F)|,|T^+(F)|)}$$

Transport Equation

$$b_h(c,z;v) = \partial_t \int_\Omega \Phi c z + \sum_{T\in\mathcal{T}_h} \int_T (D\nabla c - vc)\cdot\nabla z + \sum_{F\in\mathcal{F}_h^{iD}} \int_F c^{\uparrow}\, \nu_F \cdot v\, [\![z]\!]$$

$$- \sum_{F\in\mathcal{F}_h^{iD}} \int_F \nu_F \cdot \{D\nabla c\}_\omega [\![z]\!] - \theta \sum_{F\in\mathcal{F}_h^{iD}} \int_F \nu_F \cdot \{D\nabla z\}_\omega [\![c]\!] + \sum_{F\in\mathcal{F}_h^{iD}} \int_F \gamma_{F,t} [\![z]\!][\![c]\!]$$

$$r(z) = \sum_{T\in\mathcal{T}_h} \int_T q_t z - \sum_{F\in\mathcal{F}_h^{Nn}} \int_F j_t z$$

$$m(c,z) = \int_\Omega \Phi c z$$

Upwinding, weighting and penalty parameter:

$$c^{\uparrow} = \begin{cases} c^- & \nu_F \cdot v \geq 0 \\ c^+ & \text{else} \end{cases}$$

$$\delta_{Dn}^{\pm} = \nu_F^t D^{\pm} \nu_F, \qquad \omega^{\pm} = \frac{\delta_{Dn}^{\mp}}{\delta_{Dn}^- + \delta_{Dn}^+}, \qquad \gamma_{F,t} = \langle \delta_{Dn}^-, \delta_{Dn}^+ \rangle M.$$

Decoupled Solution Algorithm

1: Set time t, substeps S
2: Initialize $c_h(t)$
3: **while** $t < t_{end}$ **do**
4: Solve flow equation $a_h(p_h, w; c_h(t)) = l(w)\ \forall w \in W_h^{k_1}$
5: Reconstruct Darcy velocity $\hat{v} \in \hat{V} \subset H(\mathrm{div}; \Omega)$ from p_h and data
6: Choose appropriate time step Δt, set $\tau = \Delta t / S$
7: **for** $i = 1$ to S **do**
8: Given $c_h(t + (i-1)\tau)$ compute $c_h(t + i\tau)$ by solving
9: ODE system $\partial_t m(c_h, z) + b_h(c_h, z; \hat{v}) = r(z)\ \forall z \in W_h^{k_2}$
10: **end for**
11: **end while**

- Line 4: May also use cell-centered finite volume scheme
- Line 5: Uses RT_0 or BDM_1 finite element spaces
- Line 6: Based on CFL criterion
- Line 9: May use explicit or implicit method; here: SSP2/3

AMG For DG

Idea: Subspace correction in continuous Galerkin subspace [*B., Blatt, Scheichl, NLAA, 2012*]:

$$E_{AMG4DG} = (I - B_{SG}A_{DG})^{\nu_2}(I - R^T B_{AMG} R A_{DG})(I - B_{SG}A_{DG})^{\nu_1}$$

- B_{SG}: Preconditioner for DG system, e.g. Block SSOR
- R given by $\varphi_i^{CG} = \sum_{j=1}^{n_{DG}} r_{ij}\varphi_j^{DG}$
- Compute $A_{CG} = RA_{DG}R^T$ via sparse matrix multiplication
- B_{AMG} is one V-cycle of AMG preconditioner applied to A_{CG}
- We use our agglomeration-based AMG
- For heterogeneous problems CG may be replaced by P_0
- Purely algebraic, except the definition of R

Contents

Exploiting Tensor Product Structure I

- Tensor product basis: $\phi_i(x) = \phi_{i_1,\ldots,i_d}(x_1,\ldots,x_d) = \theta_{i_d}(x_d)\ldots\theta_{i_1}(x_1)$
- Tensor product quadrature: $\Xi_j = (\xi_{j_1},\ldots,\xi_{j_d})^T$
- $i = (i_1,\ldots,i_d) \in I^d = \{1,..,m\}^d$, $j = (j_1,\ldots,j_d) \in J^d = \{1,..,n\}^d$

Evaluate finite element function on ref elem at quadrature points:

$$
\begin{aligned}
u_h(\Xi_i) &= \sum_{j\in J^d} c_j \phi_j(\Xi_i) && (i \in I^d)\\
&= \sum_{j_d\in J} \ldots \sum_{j_1\in J} \theta_{j_d}(\xi_{i_d})\ldots\theta_{j_1}(\xi_{i_1})\ \ c_{j_1,\ldots,j_d} && \text{(tensor product structure)}\\
&= \sum_{j_d\in J} \ldots \sum_{j_1\in J} A^{(d)}_{i_d,j_d}\ldots A^{(1)}_{i_1,j_1}\ \ c_{j_1,\ldots,j_d} && (A^{(k)} \in \mathbb{R}^{m\times n})
\end{aligned}
$$

All basis functions at all quadrature points as matrix-vector product:

$$
y = Ax = (A^{(d)} \otimes \ldots \otimes A^{(1)})\ x, \qquad A_{(i_1,\ldots,i_d),(j_1,\ldots,j_d)} = \Pi_{k=1}^{d} A^{(k)}_{i_k,j_k}
$$

Sum Factorization I

Extract common factors:

$$
\begin{aligned}
Y_{i_1,\dots,i_d} &= \sum_{j_d \in J} \dots \sum_{j_1 \in J} A^{(d)}_{i_d,j_d} \dots A^{(1)}_{i_1,j_1} X^{(0)}_{j_1,\dots,j_d} \\
&= \sum_{j_d \in J} \dots \sum_{j_2 \in J} A^{(d)}_{i_d,j_d} \dots A^{(2)}_{i_2,j_2} \underbrace{\left(\sum_{j_1 \in J} A^{(1)}_{i_1,j_1} X^{(0)}_{j_1,\dots,j_d} \right)}_{X^{(1)}_{j_2,\dots,j_d}} \\
&= \sum_{j_d \in J} \dots \sum_{j_3 \in J} A^{(d)}_{i_d,j_d} \dots A^{(3)}_{i_3,j_3} \underbrace{\left(\sum_{j_2 \in J} A^{(2)}_{i_2,j_2} X^{(1)}_{j_2,\dots,j_d} \right)}_{X^{(2)}_{j_3,\dots,j_d}} \\
&= \dots = \sum_{j_d \in J} A^{(d)}_{i_d,j_d} X^{(d-1)}_{j_d}
\end{aligned}
$$

Do this for all $(i_1, \dots, i_d)$ simultaneously

Sum Factorization II

- Algorithm by *Buis, Dyksen (1996)*
- For $k = 1, \dots d$ do („*dim by dim approach*")
  - compute **matrix-matrix product** $X^{(k)} = A^{(k)} X^{(k-1)}$
  - "rotate" $X^{(k)}$ to make j_{k+1} the row index (transposition in 2d)
- Overall complexity: $O(n^{d+1})$ instead of $O(n^{2d})$ $(n = m = k + 1)$ for the steps above!
- Problem: matrices are rather small, e.g. $(4 \times 4) \cdot (4 \times 16)$ for $\mathbb{Q}_3$ in 3d
- Finite element application by e.g. *Melenk, Gerdes, Schwab (2001)*, *Kronbichler, Korman (2012)*:
  - compute $\hat{p}_h$, $\nabla \hat{p}_h$ at quadrature points, $O(n^{d+1})$
  - compute coefficients, geometry factors at quad. points, $O(n^d)$
  - compute residuals $O(n^{d+1})$
- Can be applied to nonlinear problems on curved elements
- Only tensor product structure of basis functions and quadrature is needed

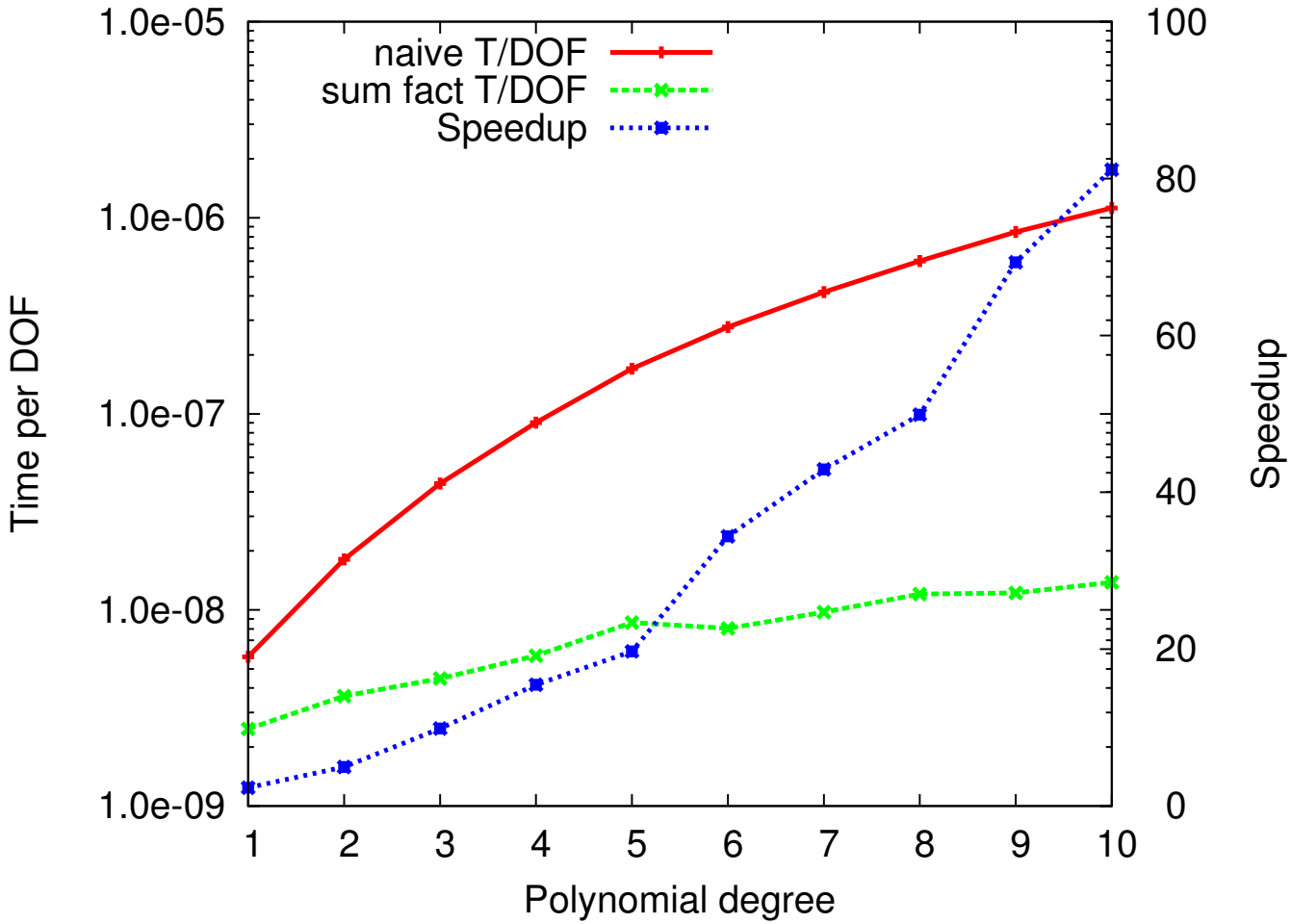
Sum Factorization III

- *Discontinuous Galerkin*: Effort for face terms dominates computation time for say $k < 10$
- Summary of complexities *per element* for $n = m = k + 1$ and $d = 3$

Operation	CG	DG (moderate k)
Matrix-free operator evaluation	n^4	n^3
Assembled operator evaluation	n^6	n^6
Sum factorized matrix assemby	n^7	n^6
Naive matrix assemby	n^9	n^9
Matrix-free point Jacobi application	n^4	n^3
Block SSOR preconditioner	n^9	n^9

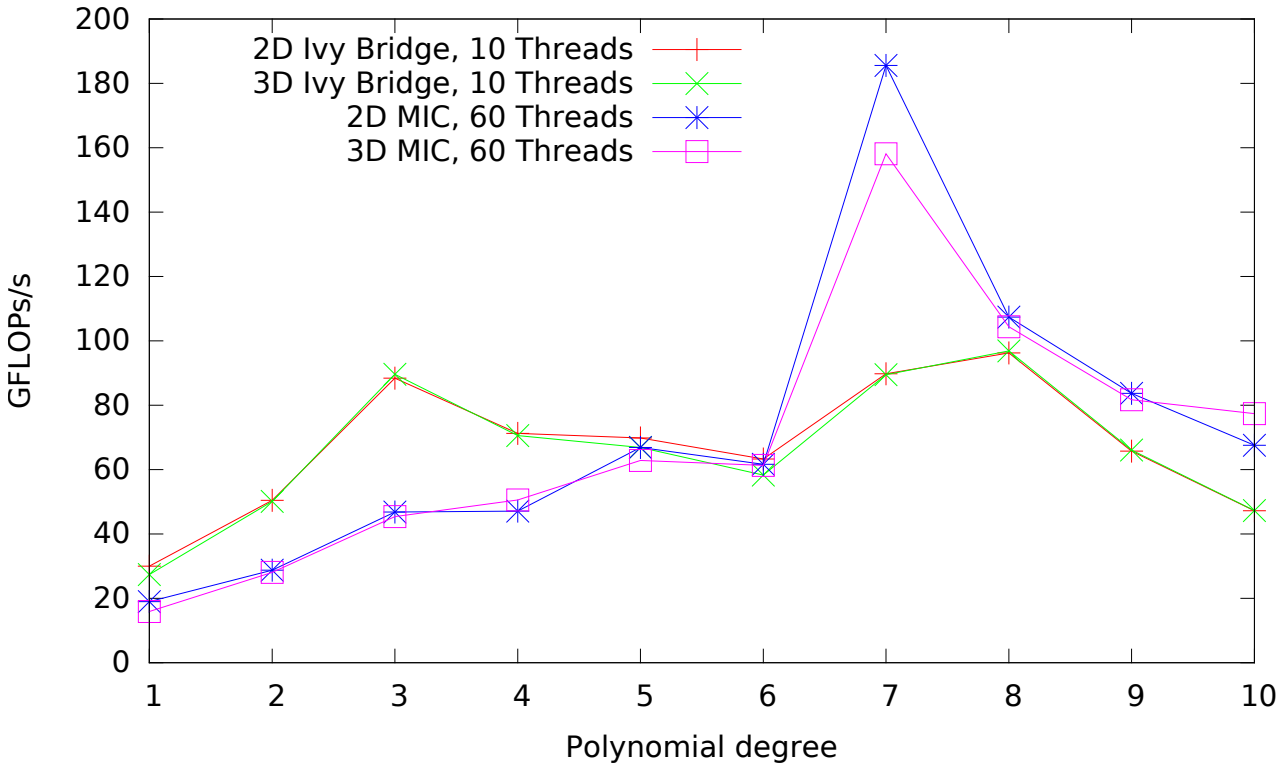
- Storage needed per element is reduced since only $1d$ basis functions need to be evaluated

Sum Factorization for Basis Evaluation in 3D



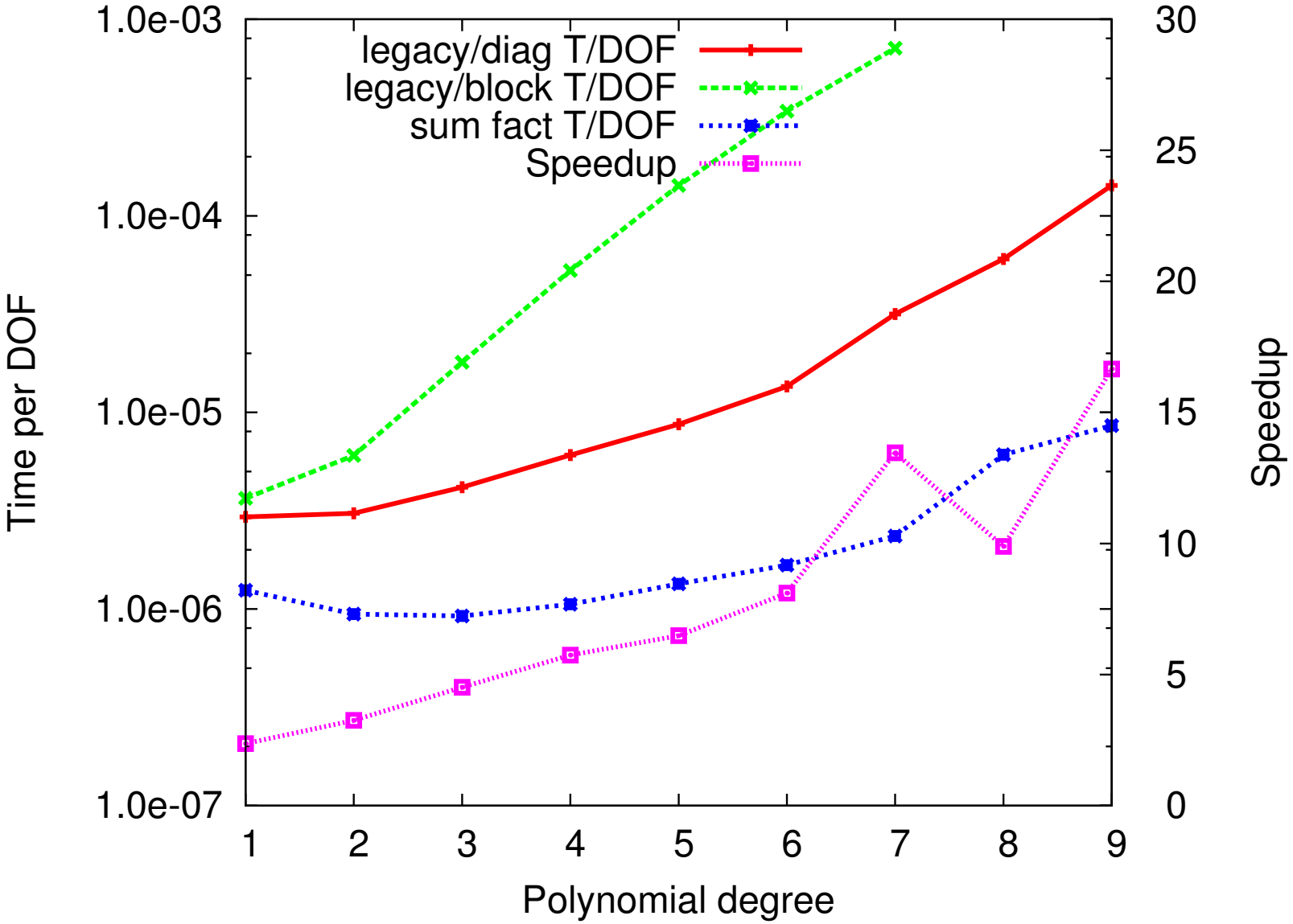
Except $\mathbb{Q}_1$ in 2d, sum factorization is always faster than naive version

Floating-Point Performance: Auto vectorizer, threaded

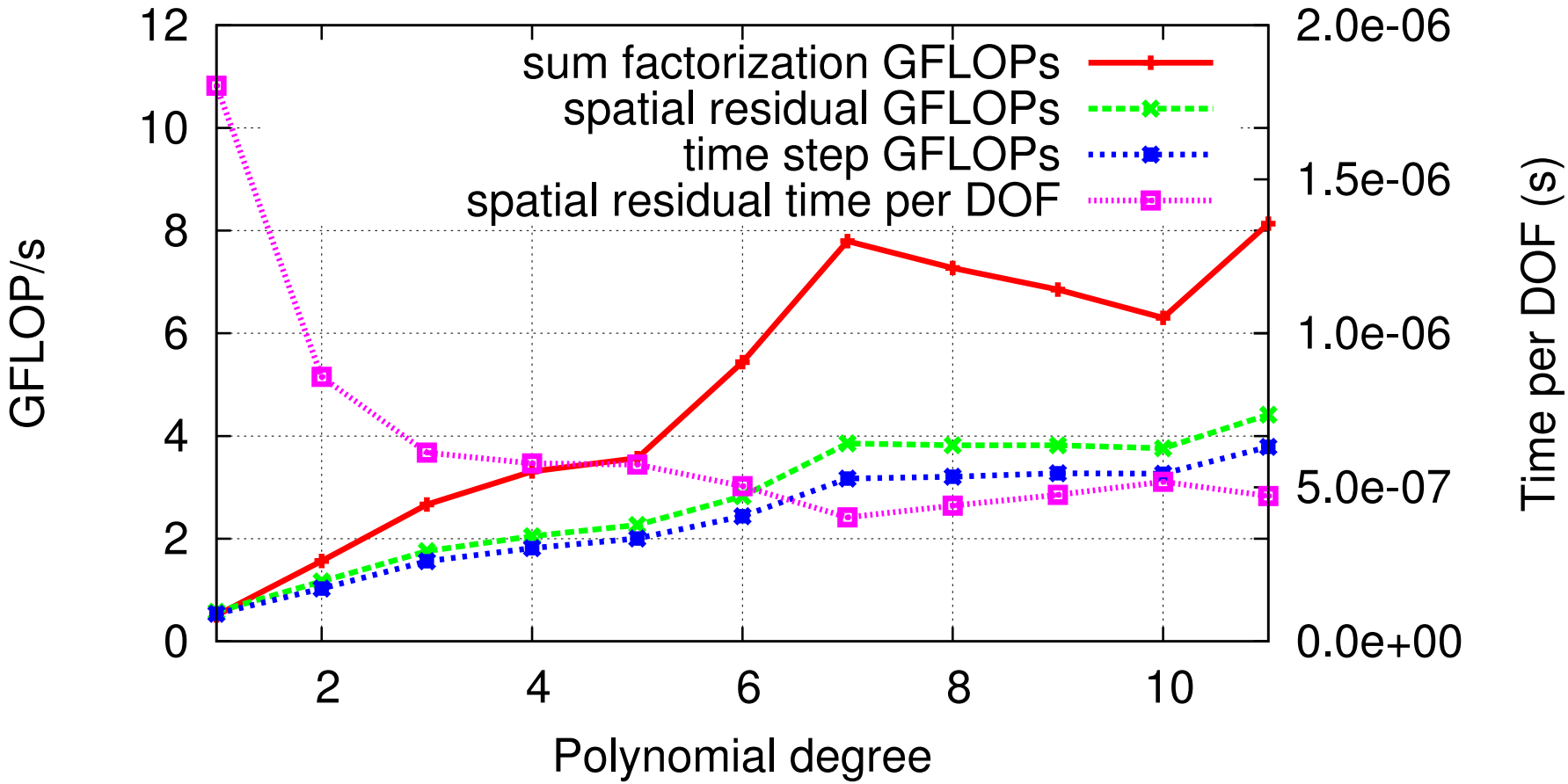


- Each tread runs sum factorization kernel in 3D
- icc with auto vectorization / SIMD hints
- Intel MIC (61 cores) vs. Xeon E5-2680v2 (Ivy Bridge, 10 cores)

Complete Explicit Time Step 3d



GFLOP Rates Per Core



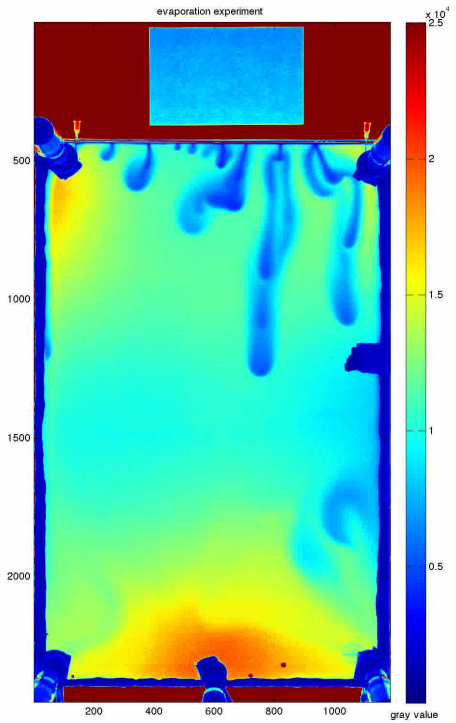
Xeon E5-2680v2 10 core processor (22 GFLOPS peak / core)

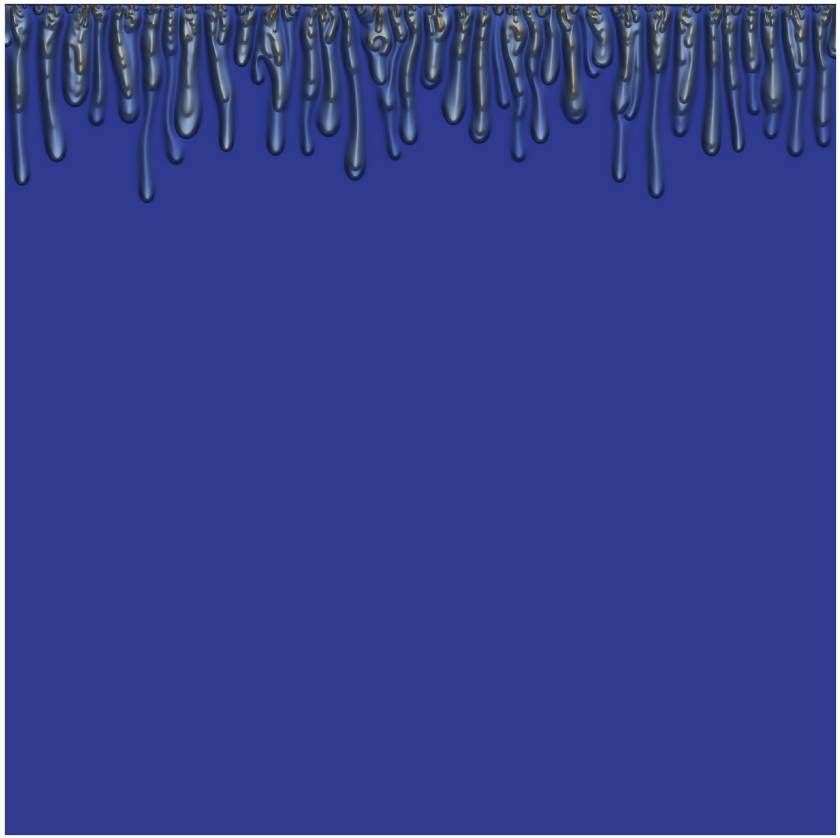
Residual: 4 GFLOPS

Sum factorization alone: 8 GFLOPS

Contents

2d Density Driven Flow





Experiment by Philip Kreyenberg (IUP, Heidelberg)

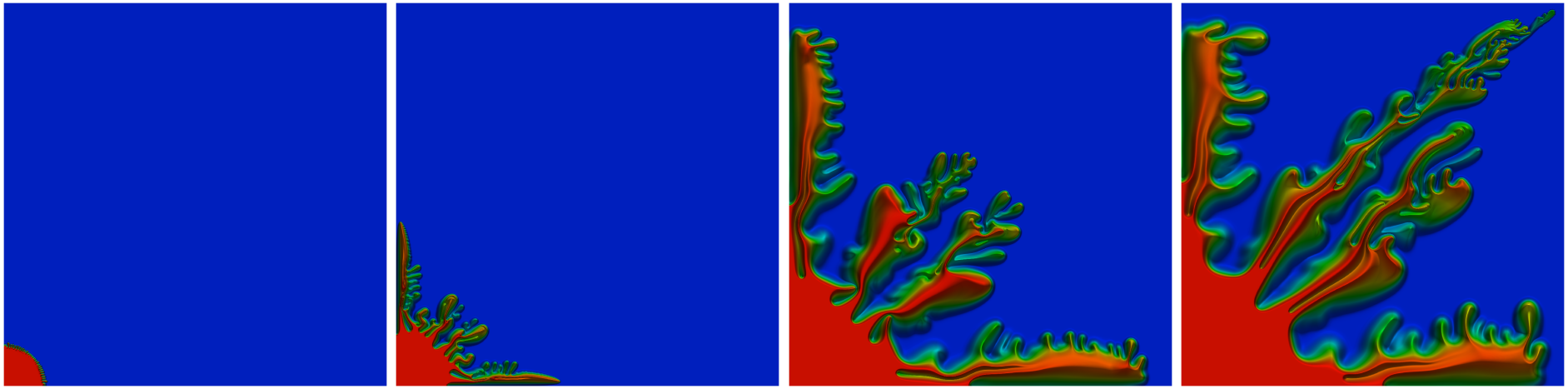
$\mathrm{Ra} = 8000$

25 cores of Xeon E5-2680v2

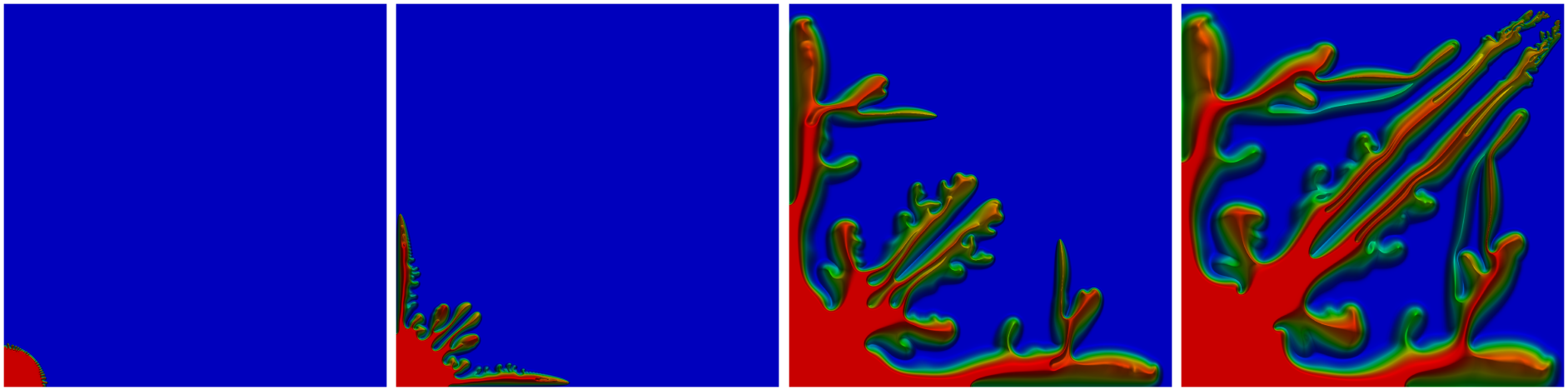
1200^2, Q_2, $1.3 \cdot 10^7$ DOF

2d Miscible Displacement

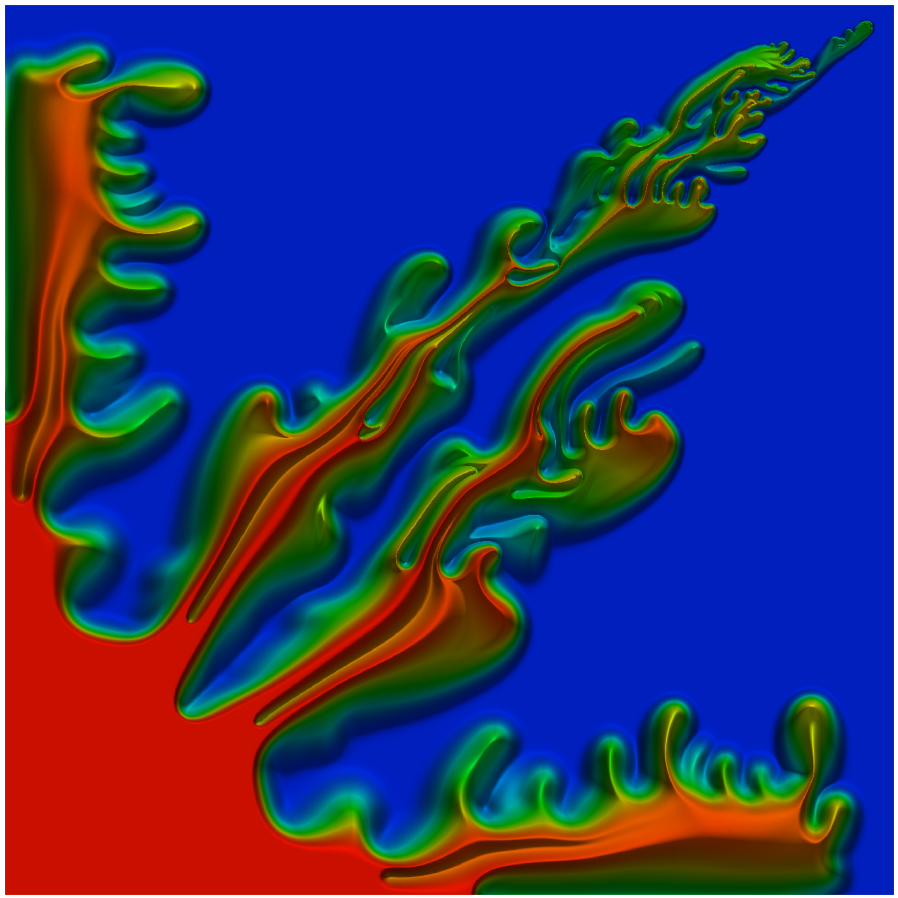
DG(Q_2)+BDM$_1$+DG(Q_3)+SSP3 — $9 \cdot 10^6$ DOF, 180000 time steps — 40 cores, 19 h

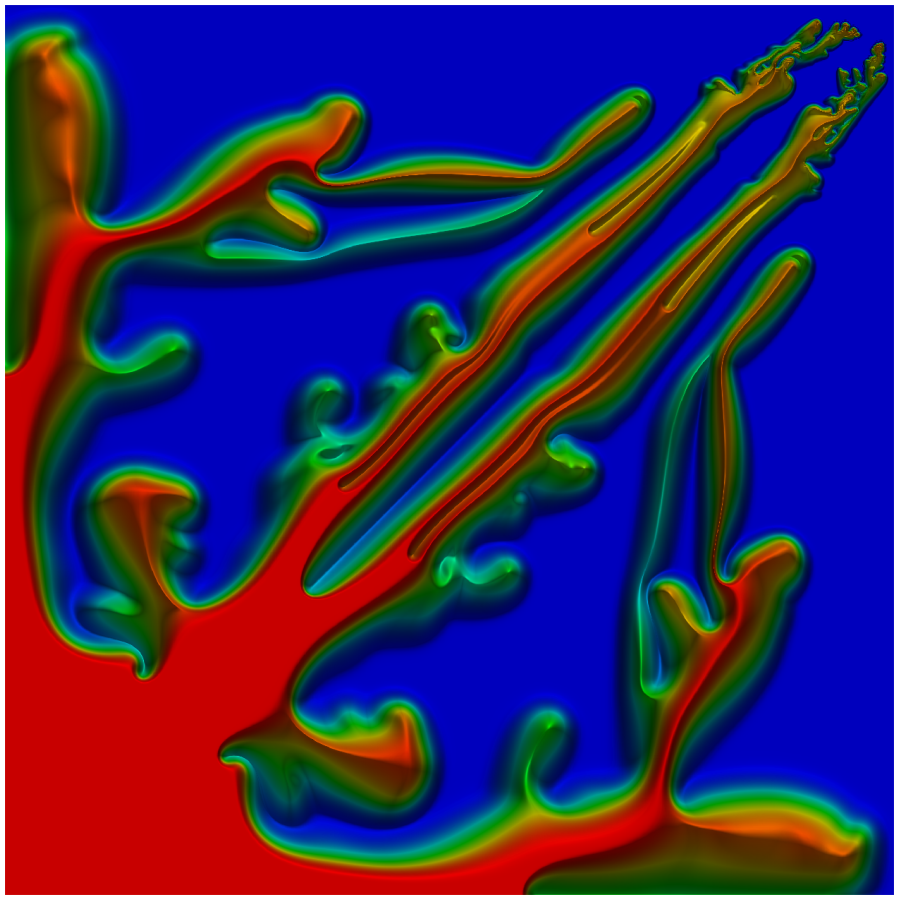


CCFV+RT$_0$+DG(Q_3)+SSP3 — $6.21 \cdot 10^6$ DOF, 180000 time steps — 40 cores, 19 h



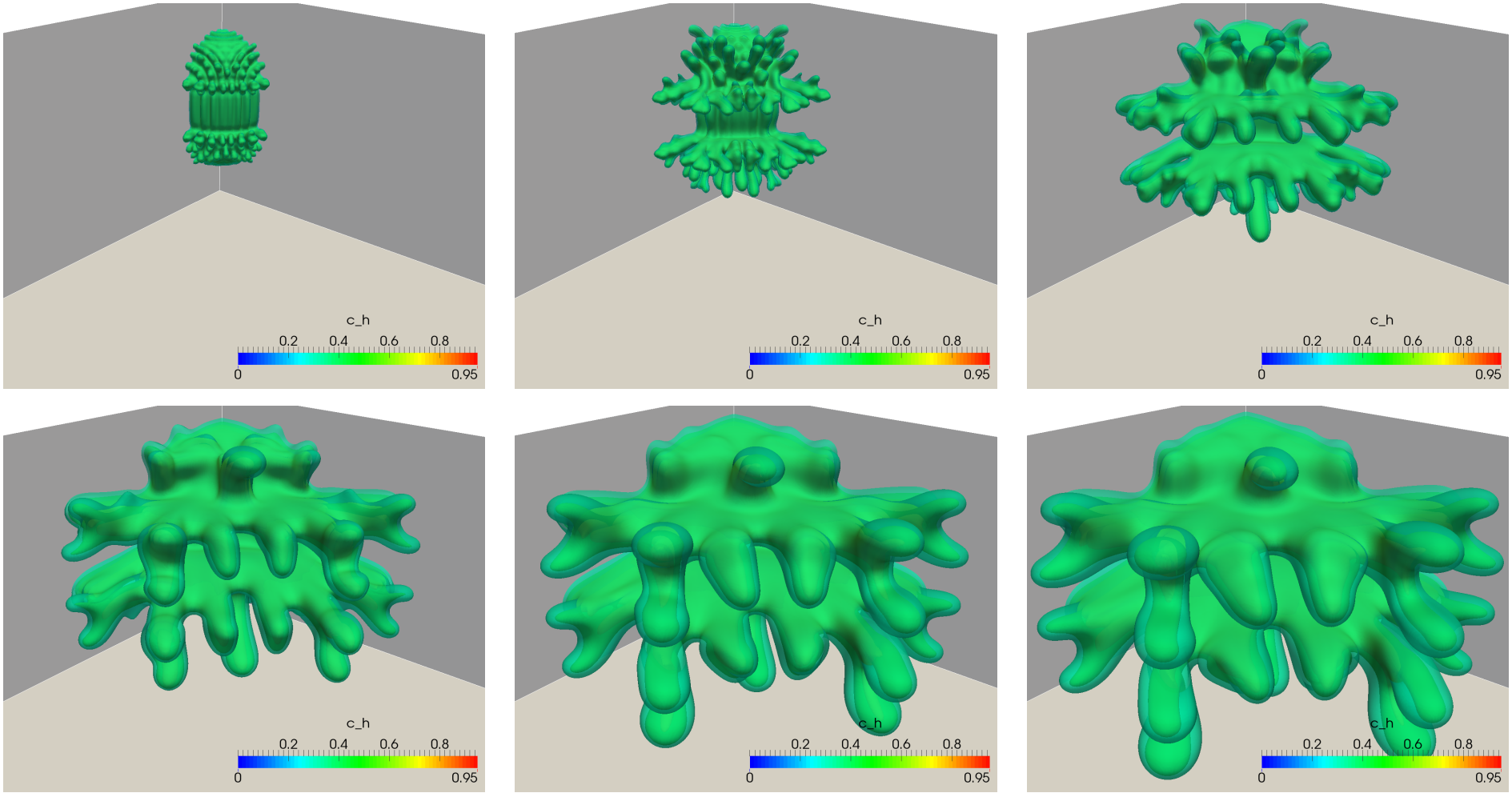
2d Miscible Displacement – Movies





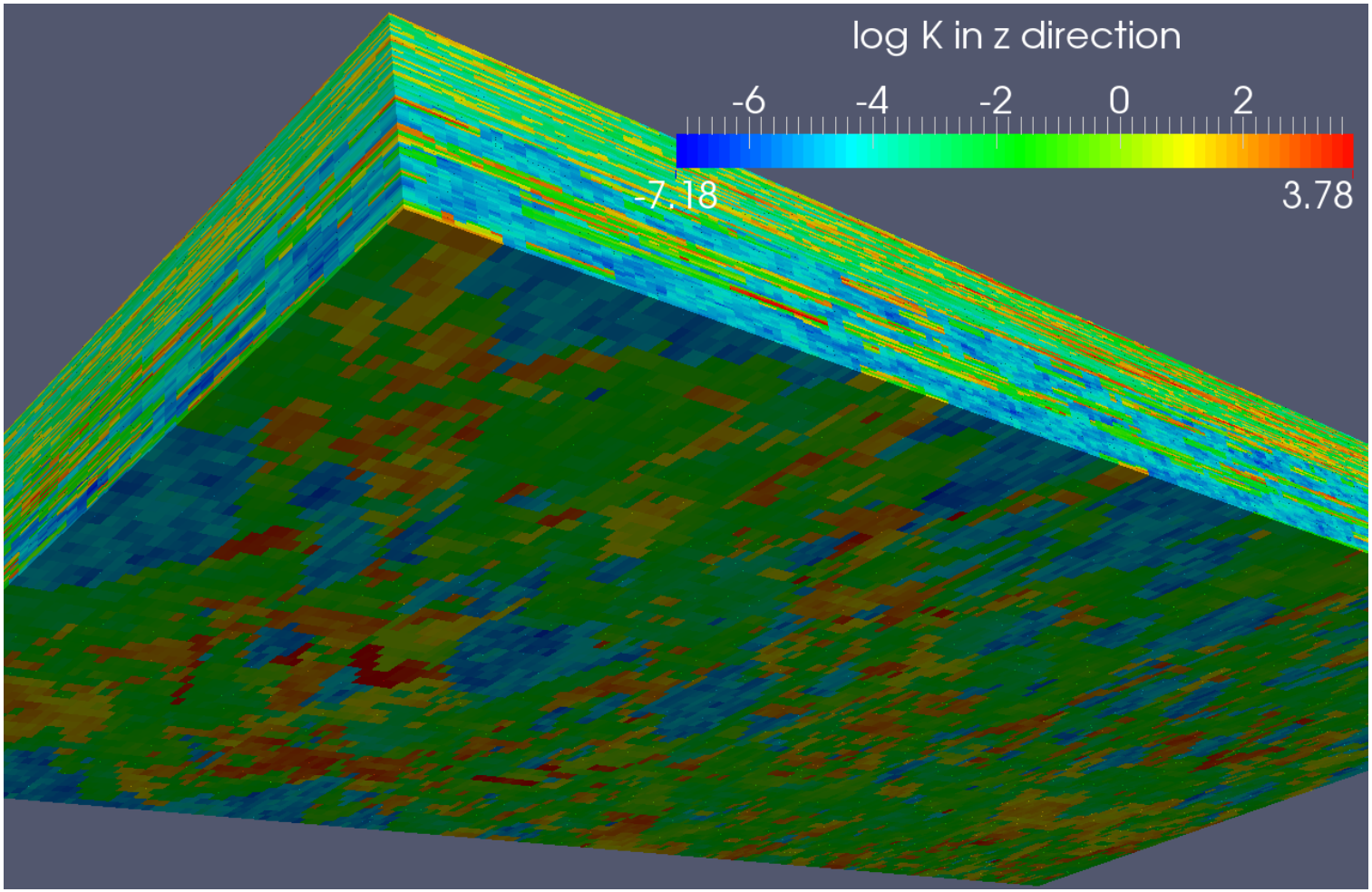
3d Miscible Displacement

DG(Q_1)+BDM$_1$+DG(Q_2)+SSP2 — 259 · 10^6 DOF, 46000 time steps — 100 cores, 3 days



Contents

SPE10 Test Problem I



Domain $1200 \times 2200 \times 170$ ft^3, $60 \times 220 \times 85$ mesh (1.1 Mio cells)
Anisotropic, diagonal permeability tensor, 11 orders of magnitude variation

SPE10 Test Problem II

Solve elliptic problem $-\nabla \cdot (K_{SPE10} \nabla u) = 0;\ u = g$ on $\partial\Omega_{SPE10}$

DG(Q_1), hybrid DG/AMG-P_0 and Block-SSOR preconditioner

Discretization	DG-Q_1	DG-Q_1
Preconditioner	AMG-DG(2,2,V)	BSSOR(1)
DOF $\cdot 10^6$	9.0	9.0
#IT(10^{-6})	55	2637
TIT(s)	8.90	1.90
$T_{solve}(s)$	490	5010

$\rightarrow$ Combine matrix-based and matrix-free

Summary

- All methods presented in this talk have been implemented in the DUNE software framework (`www.dune-project.org`)
- SF based DG implementation gives huge savings for matrix-free and matrix based approaches
- SF can handle variable coeffs, non-affine geometry, nonlinear problems
- Good SIMD performance is achieved for larger polynomial degrees
- Future work
  - Code generation or intrinsics might be necessary to achieve still better performance
  - Hybrid (matrix-free/matrix-based) preconditioners for implicit DG formulations
  - **Disclaimer**: Use moderate polynomial degree in practice!

Thank you for your attention!