Matrix-free block-Jacobi preconditioners for higher-order DG methods

Peter Bastian[†], **Eike Hermann Müller**[*],
Steffen Müthing[†], Robert Scheichl[*]

[*]University of Bath, [†]University of Heidelberg

DUNE User meeting, Heidelberg, 28th Sep 2015

Introduction

Model problem: Linear elliptic operator

$$\begin{aligned} -\nabla\left(A\nabla u\right) + \boldsymbol{b}\cdot\nabla u + cu &= f(x) && \in \Omega = [0,1]^d \\ u &= g(x) && \in \partial\Omega_D \\ \boldsymbol{n}\cdot\nabla u &= j(x) && \in \partial\Omega_N = \partial\Omega \setminus \partial\Omega_D \end{aligned}$$

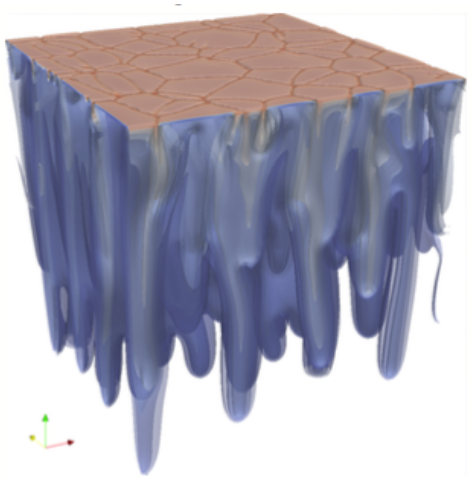
Discontinuous Galerkin discretisation:
$\Rightarrow$ Linear equation for vector of unknowns U

$$R[u] = 0 \Rightarrow JU + B = 0$$

Solve with **preconditioned Krylov-method**

$$\text{Krylov}(J, prec)$$

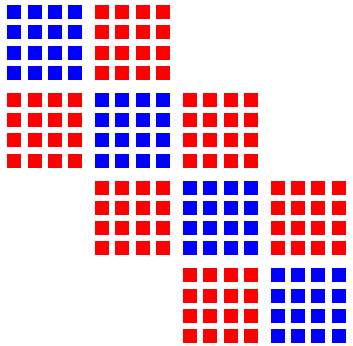
This talk: $A = \text{diag}(1,1,1)$, $\boldsymbol{b} = 0$, $c = 1$



Operator splitting in density-driven flow [Bastian et al.]

Introduction

Matrix has **block-structure**



$\Rightarrow$ **block-Jacobi preconditioner**

$$U \mapsto U + \omega D^{-1}(B - JU)$$

- Apply linear operator J matrix-free?
- Invert D iteratively, and apply D matrix-free?

Requirements:

1. D can be inverted in small # iterations
2. J and D can be applied efficiently ($\Rightarrow$ sum-factorisation)

block-Jacobi

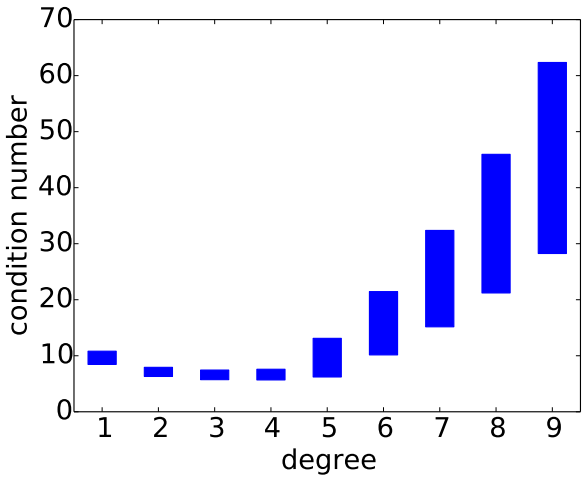
Algorithmic performance

Iteratively invert blocks to tolerance ϵ_{block}

$$J^{-1}: \quad CG(J, \text{prec} = D^{-1}; \epsilon)$$
$$D^{-1}: \quad CG(D, \text{prec} = D_{\text{diag}}^{-1}; \epsilon_{\text{block}})$$

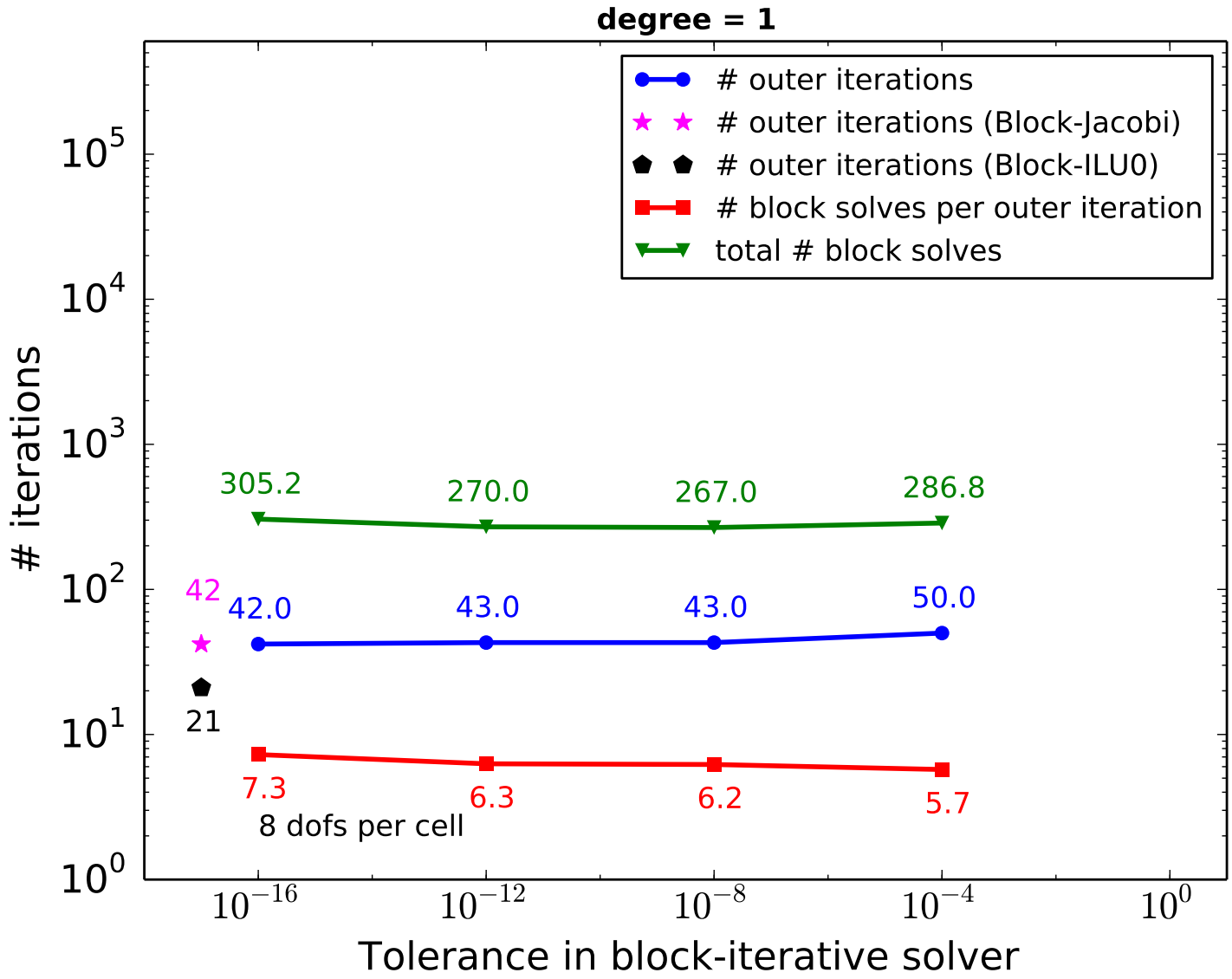
- # applications of D per outer iteration
- # outer CG iterations $n_{\text{outer}}(\epsilon_{\text{block}})$
- total # applications of D

Compare to standard block-Jacobi D^{-1}

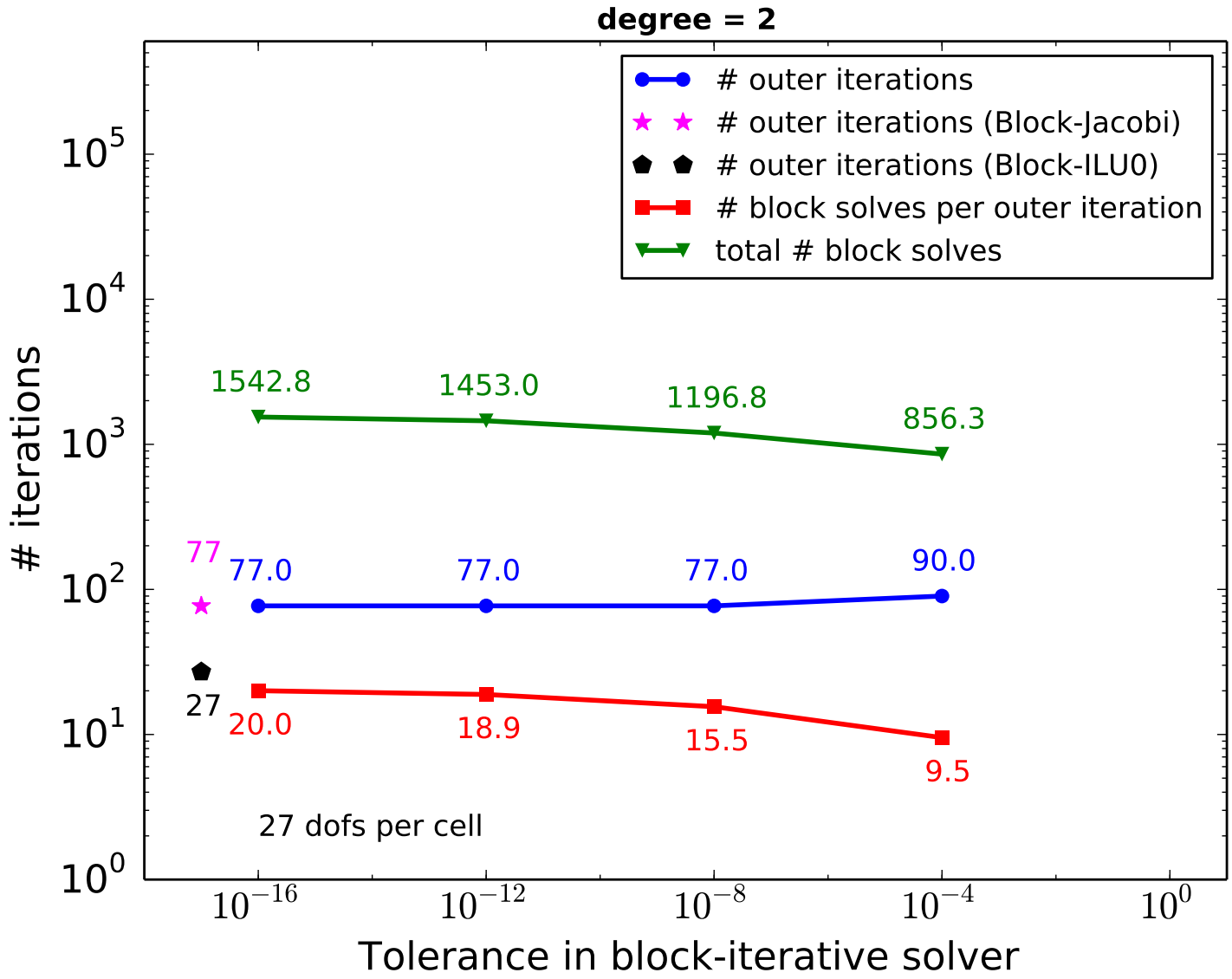


Condition number of D

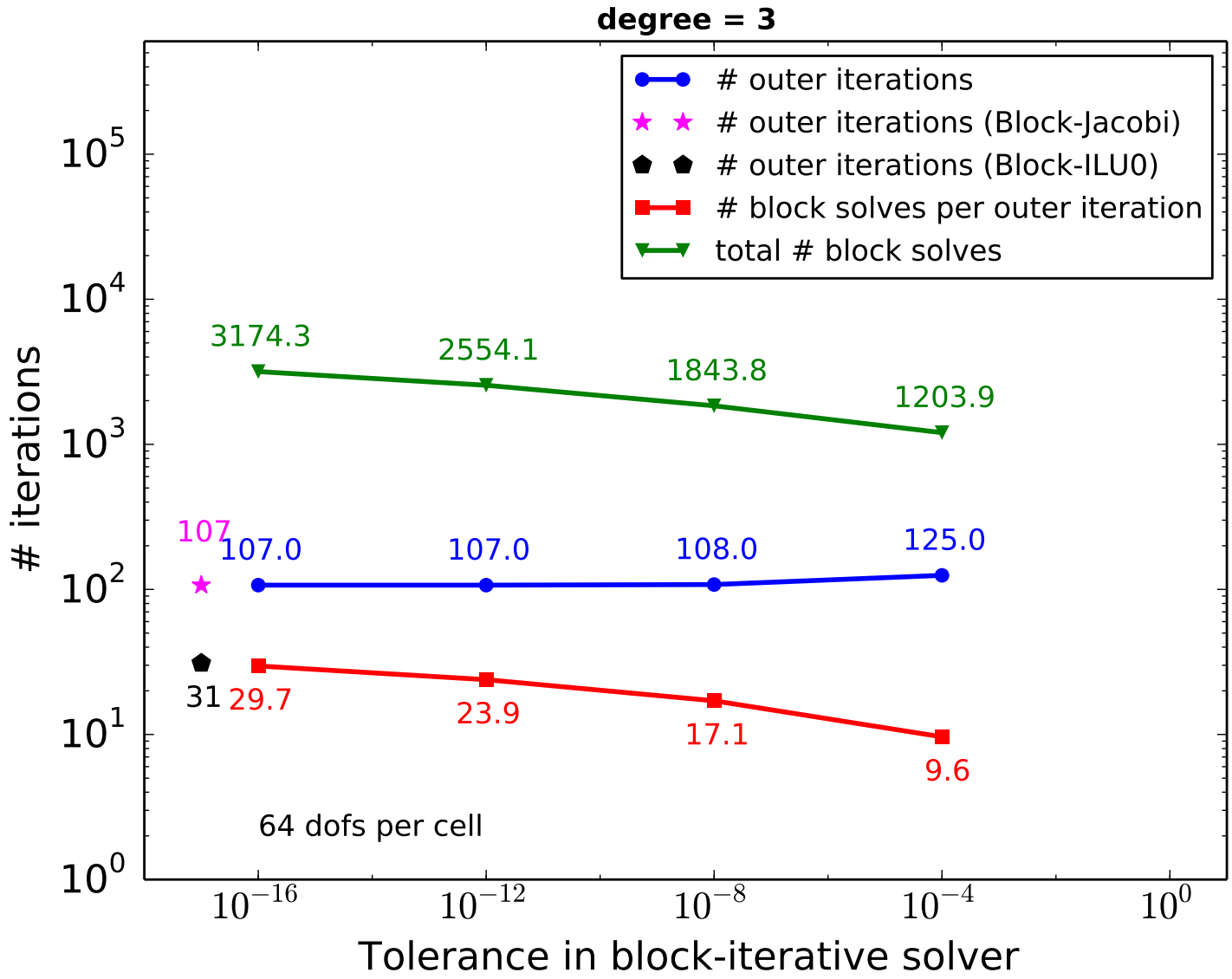
Number of iterations



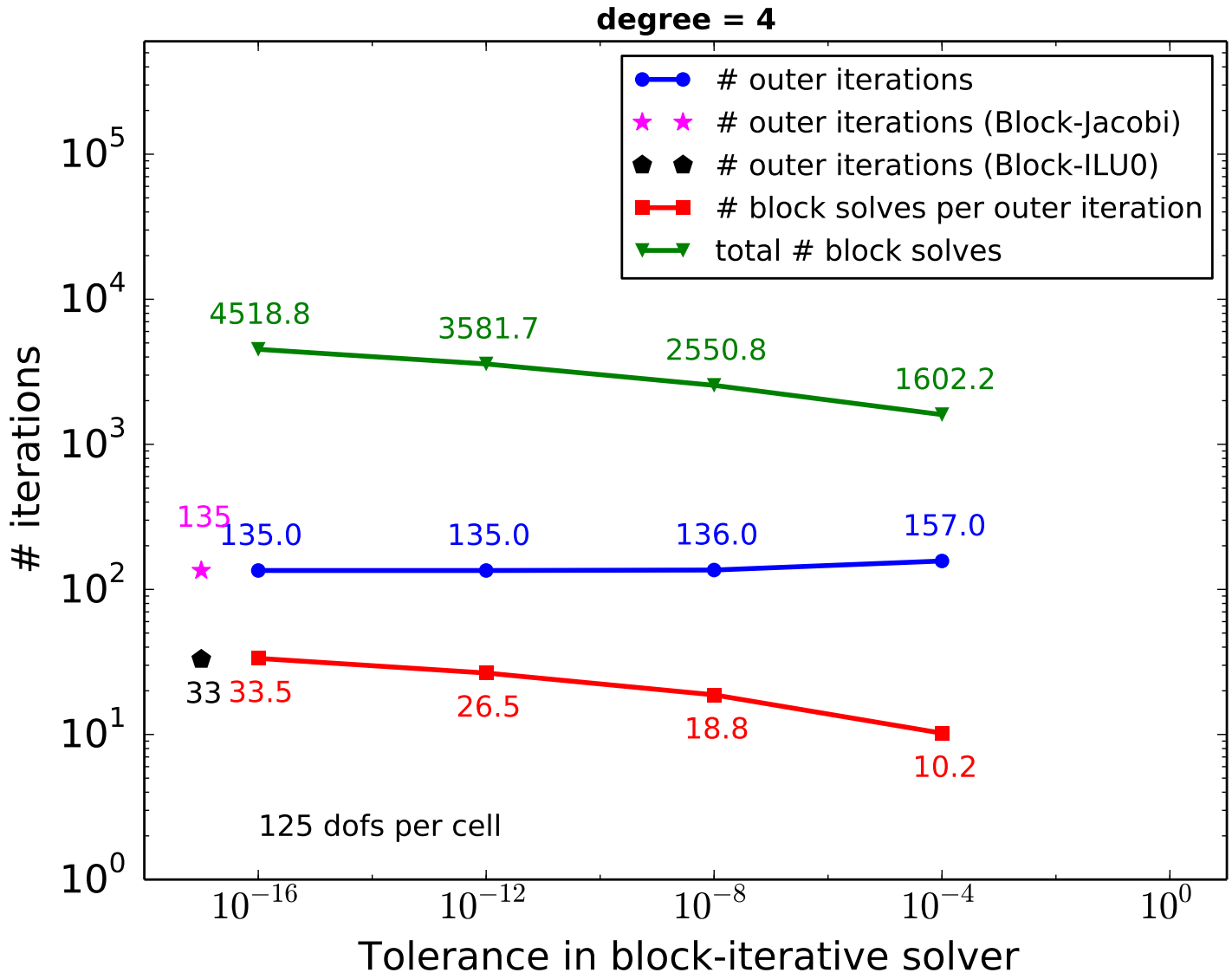
Number of iterations



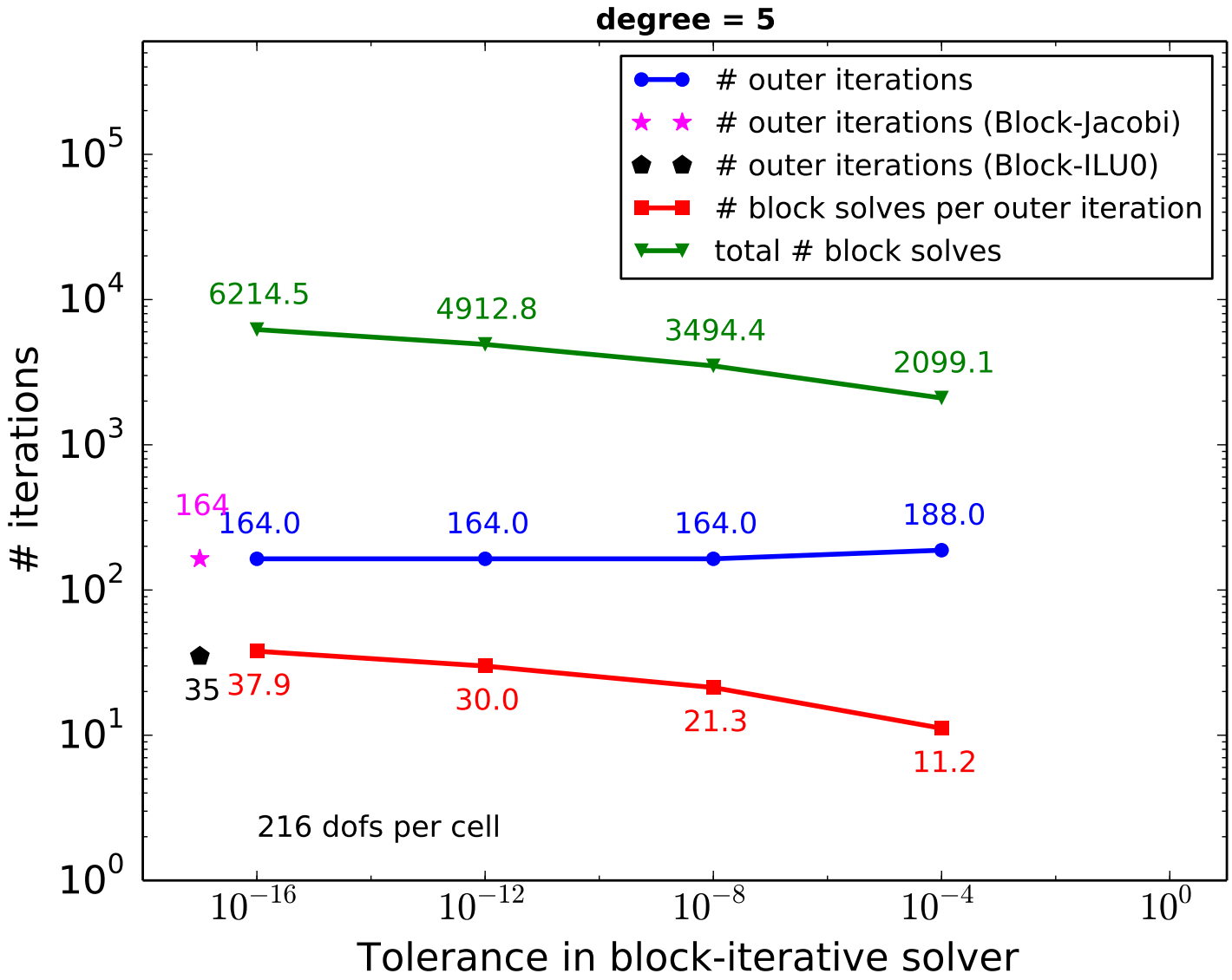
Number of iterations



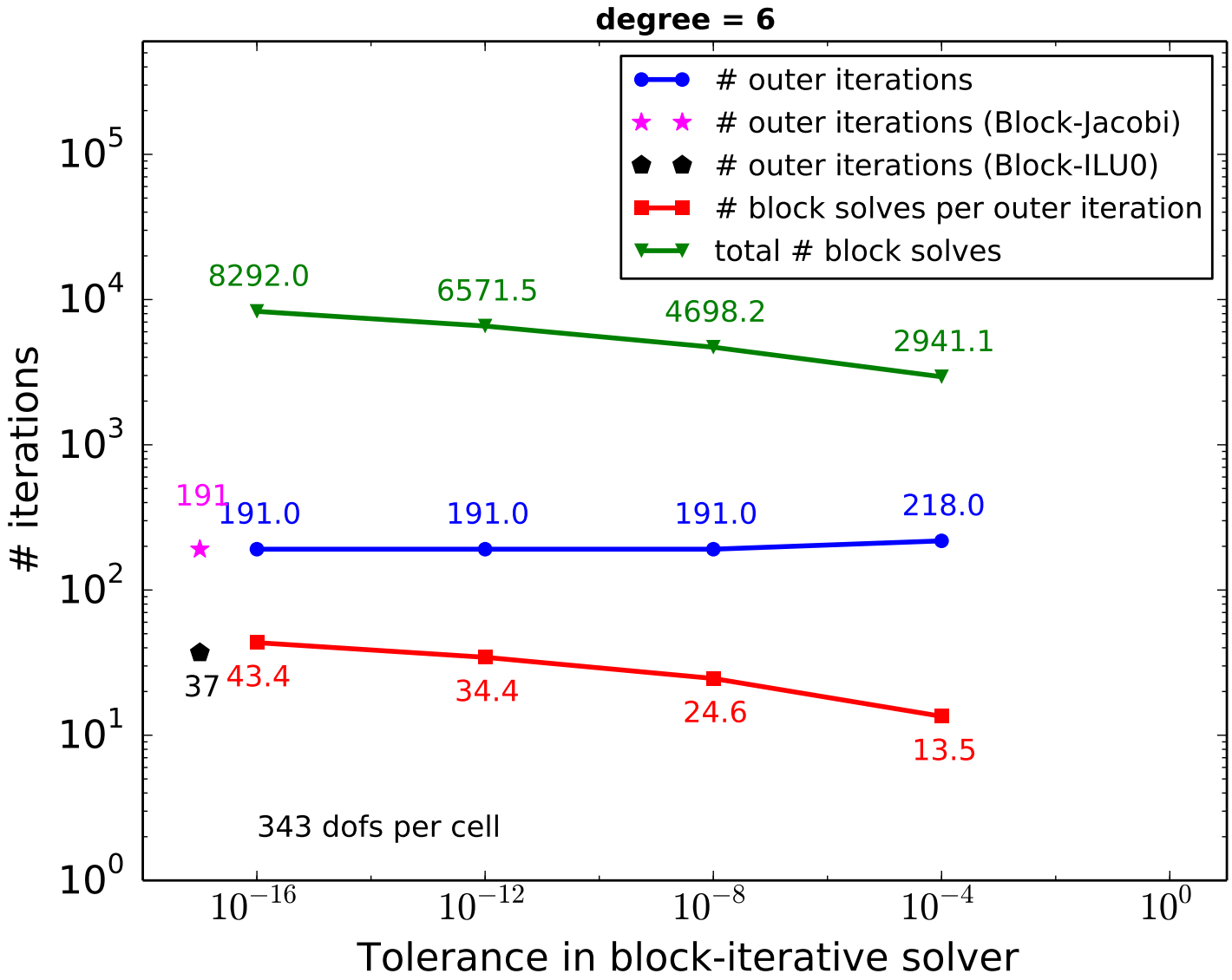
Number of iterations



Number of iterations



Number of iterations



Asymptotic cost estimates

Sum Factorisation (SF) + Matrix Free (MF)

see e.g. [Vos, Sherwin, Kirby (2009), Kronbichler, Kormann (2012)]

$$u(\hat{\boldsymbol{x}} = (\hat{x}_1, \dots, \hat{x}_d)) = \sum_{k=0}^{N} \hat{\phi}_k(\hat{\boldsymbol{x}}) U_k$$

$$= \sum_{k_1=0}^{p=n-1} \cdots \sum_{k_d=0}^{p=n-1} \hat{\psi}_{k_1}(\hat{x}_1) \dots \psi_{k_d}(\hat{x}_d) U_{k_1,\dots,k_d}$$

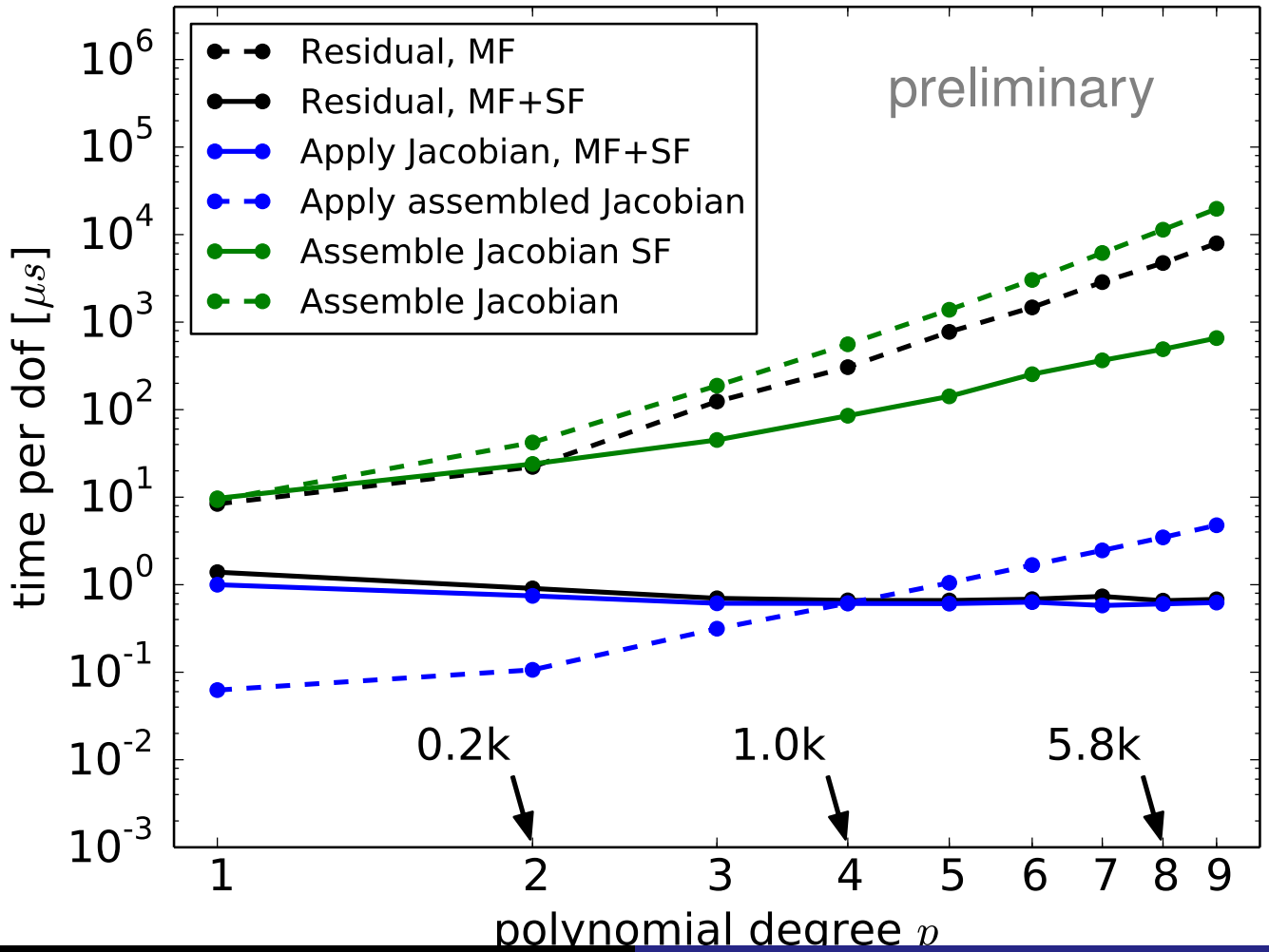
Method	FLOPs	memory references
ME	$2n^{2d} = \mathcal{O}(p^{2d})$	$n^{2d} = \mathcal{O}(p^{2d})$
MF	$\mathcal{O}(n^{2d} + m^d) = \mathcal{O}(p^{2d})$	$n^d = \mathcal{O}(p^d)$
MF + SF	$\mathcal{O}(d \cdot n^{d+1} + m^d) = \mathcal{O}(d \cdot p^{d+1})$	$n^d = \mathcal{O}(p^d)$

- What are the **constants**?
- How large is p? Is it $\mathcal{O}(10)$? Or $\mathcal{O}(100)$

Results

Full Operator application 8 grid cells, $d = 3$,

[single-threaded EXADUNE impl. P. Bastian, S. Müthing, dual core macbook]



Results

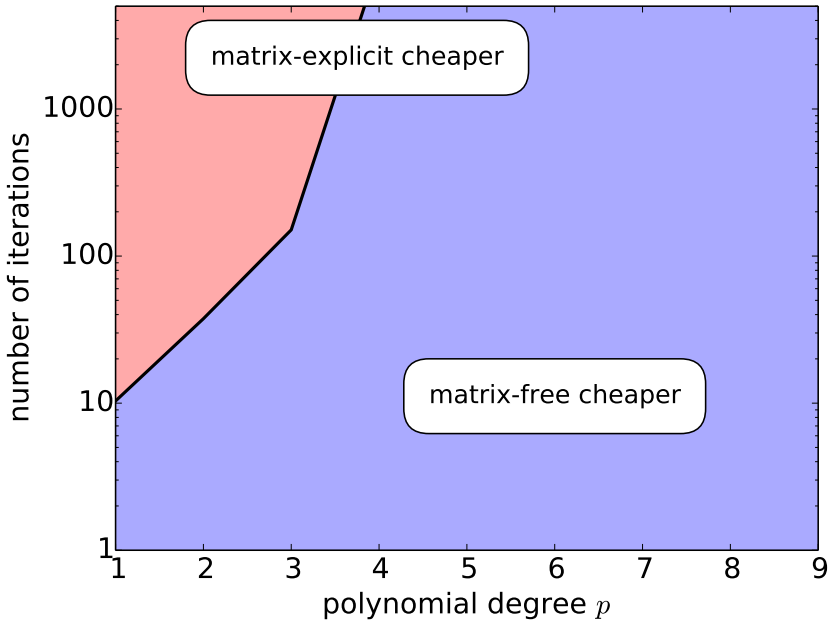
When does matrix-free pay off?

Full operator application

n = Number of iterations

$$T^{ME}(n) = t_{assemble} + n \cdot t^{ME}_{apply}$$
$$T^{MF}(n) = n \cdot t^{MF}_{apply}$$

Crossover:
$$T^{ME}(n) = T^{MF}(n)$$

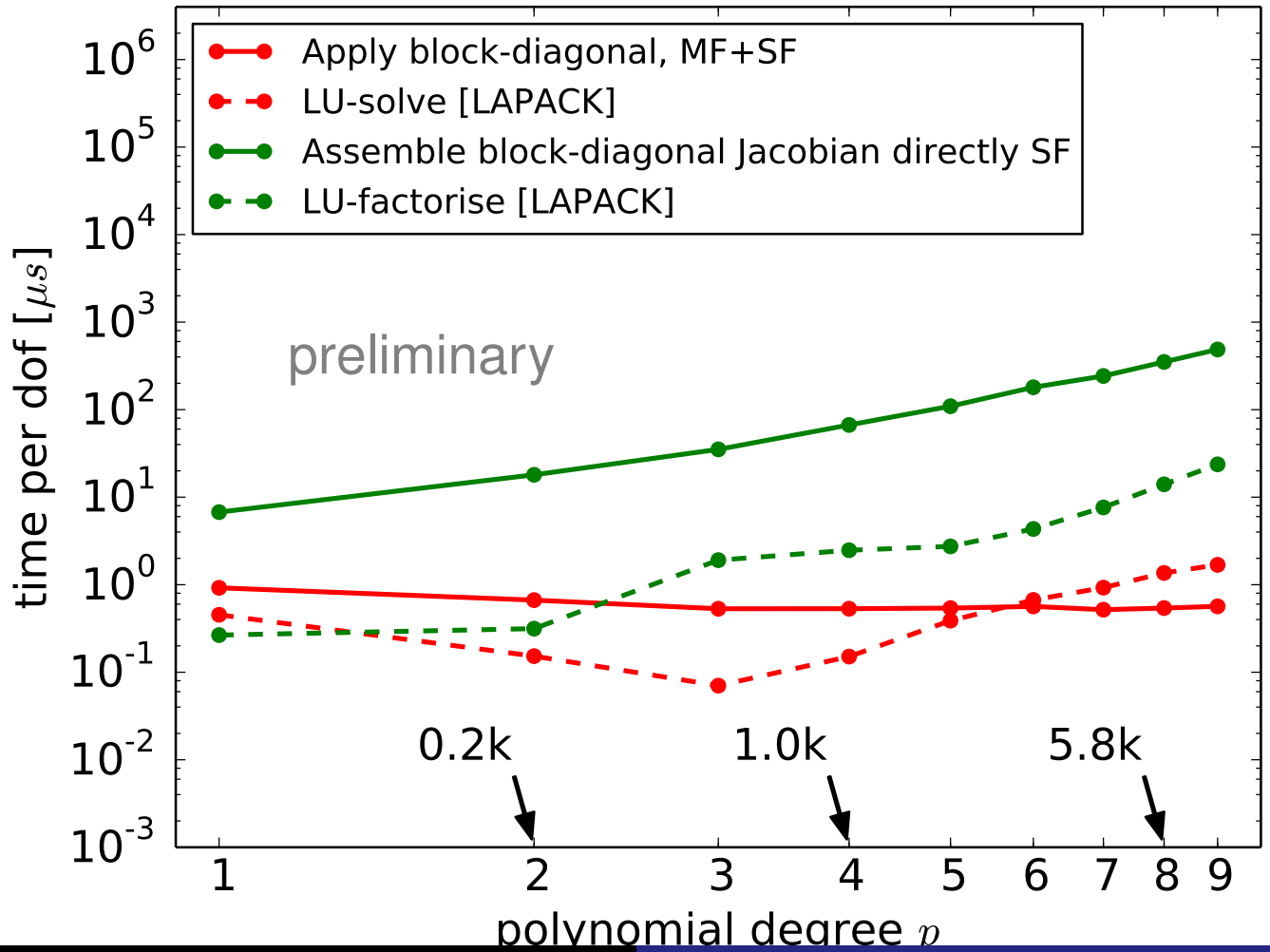


Sum-factorised assembly!

Results

Block-diagonal Operator application 8 grid cells, $d = 3$,

[single-threaded EXADUNE impl. P. Bastian, S. Müthing, dual core macbook]



Results

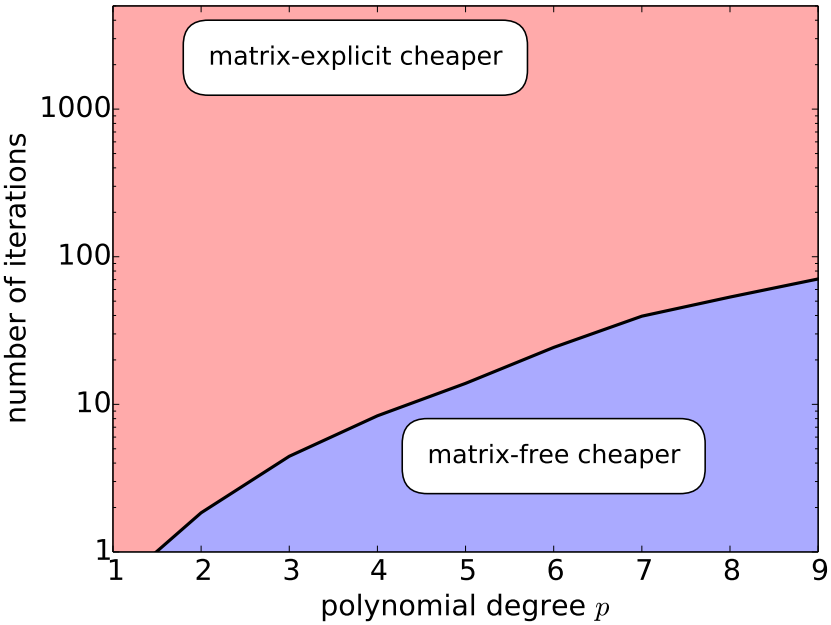
When does matrix-free pay off?

Block diagonal operator application

n = Number of iterations

$$T^{ME}(n) = t_{assemble} + t_{LU-factor} + n \cdot t^{ME}_{LU-solve}$$

$$T^{MF}(n, n_{\text{block}}) = n_{\text{block}} \cdot n \cdot t^{MF}_{apply}$$



Crossover:

$$T^{ME}(n) = T^{MF}(n, n_{\text{block}} = 20)$$

Sum-factorised assembly!

Full solve

Performance in linear solver

CG + block-Jacobi solve, $2 \times 2 \times 2$ grid, $\epsilon = 10^{-5}$, $\epsilon_{\text{block}} = 10^{-4}$

	Matrix-explicit			Matrix-free		
Order	# its	t_{total}	t_{iter}	# its	t_{total}	t_{iter}
2	10	0.0038	0.00038	10	0.078	0.00078
4	19	0.27	0.014	20	0.92	0.046
6	27	6.5	0.24	28	4.6	0.16
8	34	**99.0**	**2.9**	35	**14.8**	**0.42**

- single-threaded
- explicit block-Jacobi LU-factorises in every iteration

Conclusion

Summary and outlook

- Test in **full solver**
- Test on "proper" **manycore CPUs**: Haswell, $\mathcal{O}(10)$ cores, AVX2 (256bit, 4 double vectors)
  $\Rightarrow$ should favour matrix-free implementation by factor $\mathcal{O}(10\times)$
- **Higher order multigrid**
  [P. Bastian, M. Blatt, R. Scheichl, Num. Lin. Alg. with Appl, 19 (2) (2012)]
  - iterative block-smoother on fine level
  - AMG for low-order problem on coarse levels
- **EXADUNE applications** (e.g. multiphase flow)
- Intermediate approach:
  Assemble matrix at $\mathcal{O}(p^d)$ quad. points $\Rightarrow$ still $\mathcal{O}(d \cdot p^{d+1})$

Sum factorisation

Local operator application, volume term

$$U \mapsto J^{(e)} U = B^T W^{(e)} B U$$

$$B_{q,k} \equiv \hat{\phi}_k(\hat{\boldsymbol{x}}_q) = \hat{\psi}_{k_1}(\hat{x}_{1,q_1}) \dots \psi_{k_d}(\hat{x}_{d,q_d}) = \tilde{B}_{q_1,k_1} \dots \tilde{B}_{q_d,k_d}$$

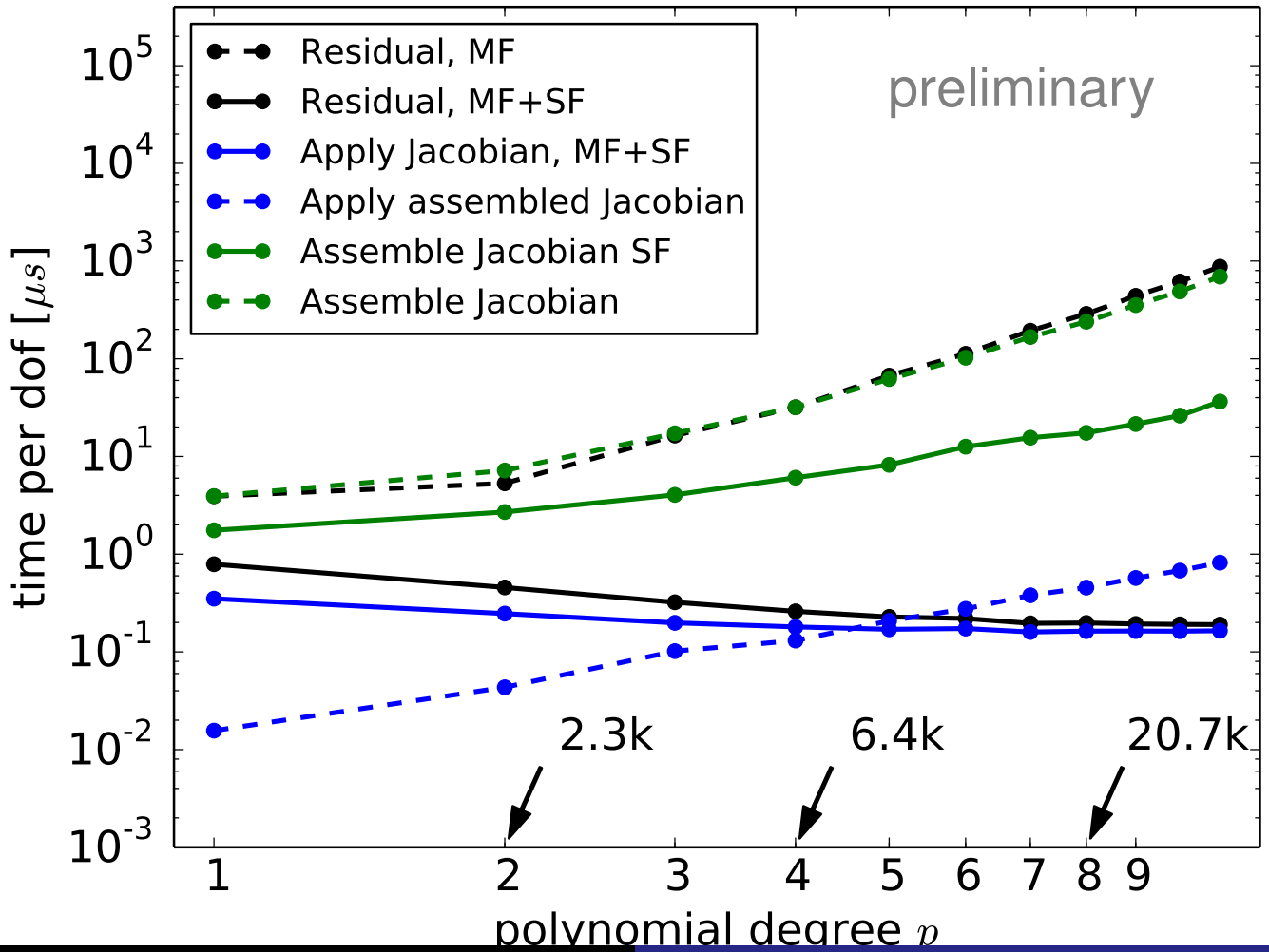
Quadrature points $q_i = 0, \dots, m-1 = 2p$, basis functions $k_i = 0, \dots, n-1 = p$
Operator dependent term $W^{(e)}$

- **standard** multiply n^d-vector by $m^d \times n^d$-matrix $\mathcal{O}(p^{2d})$
- **sum factorisation** $\mathcal{O}(d \cdot p^{d+1})$
  - multiply n-vectors by $m \times n$-matrix $\mathcal{O}(n \cdot m)$
  - ... for each of the d dimensions
  - ... n^{d-1} vectors in each dimension

Results

Full Operator application 256 grid cells, $d = 2$,

[TBB-threaded EXADUNE impl. P. Bastian, S. Müthing, dual core macbook]



Results

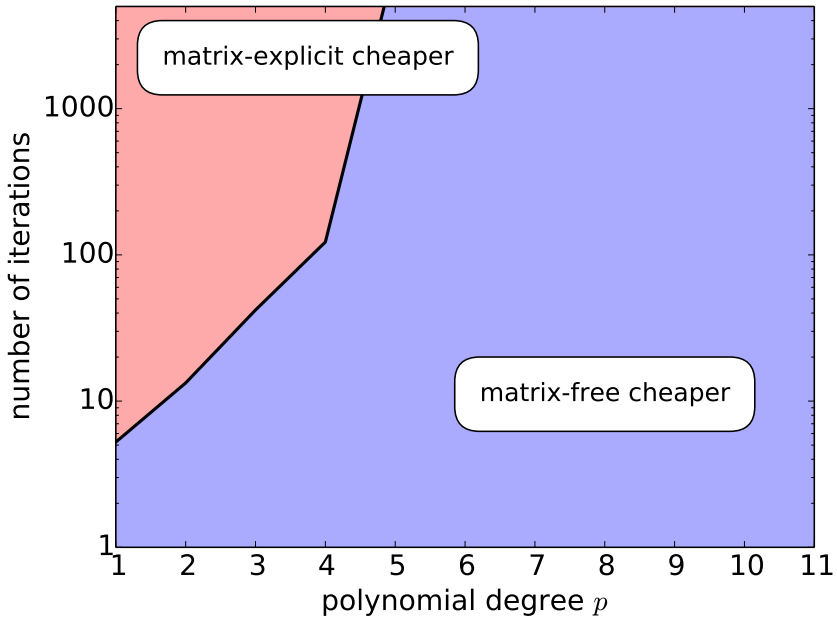
When does matrix-free pay off?

Full operator application

n = Number of iterations

$$T^{ME}(n) = t_{assemble} + n \cdot t^{ME}_{apply}$$
$$T^{MF}(n) = n \cdot t^{MF}_{apply}$$

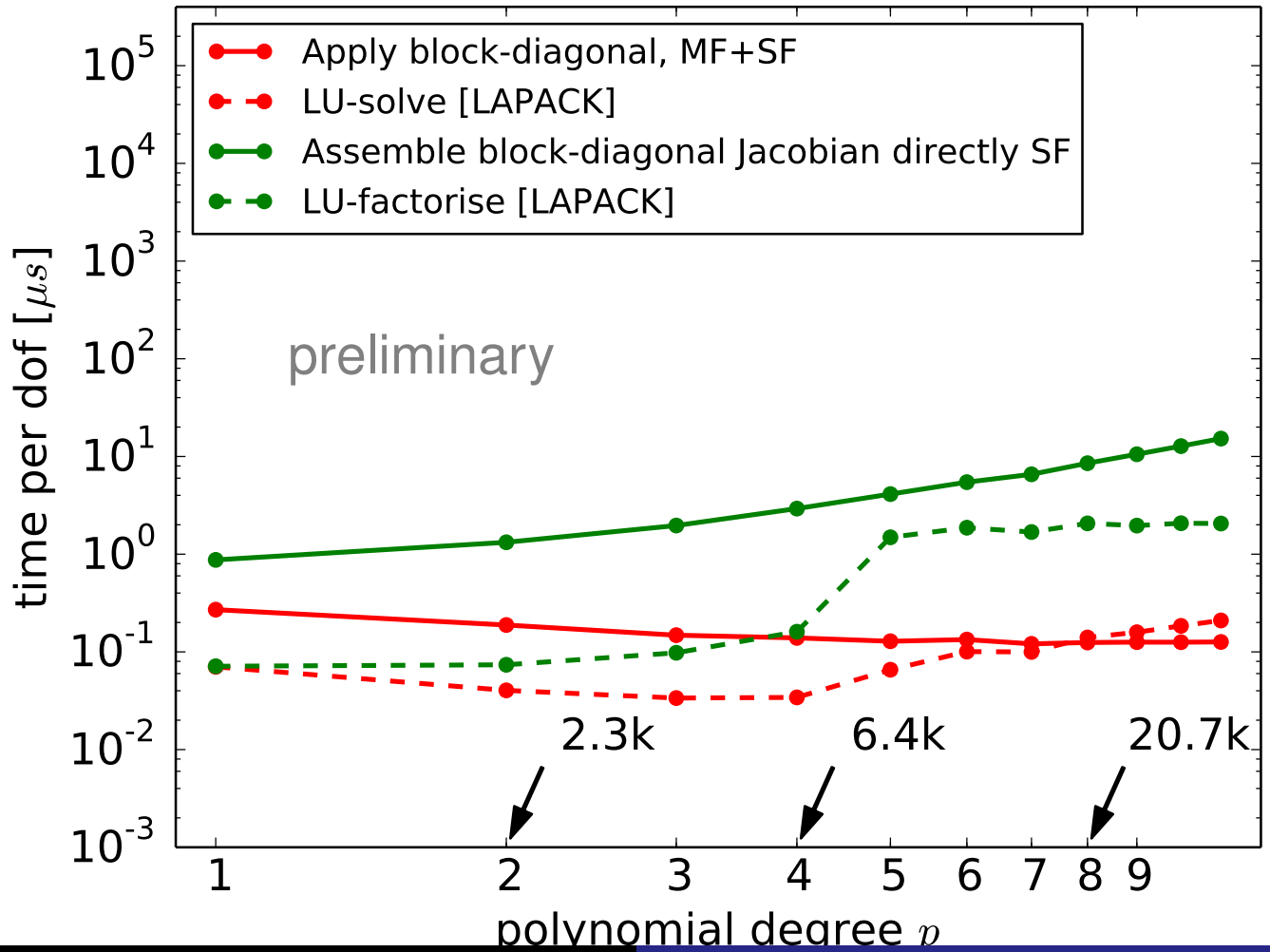
Crossover:
$$T^{ME}(n) = T^{MF}(n)$$



Results

Block-diagonal Operator application 256 grid cells, $d = 2$,

[TBB-threaded EXADUNE impl. P. Bastian, S. Müthing, dual core macbook]



Results

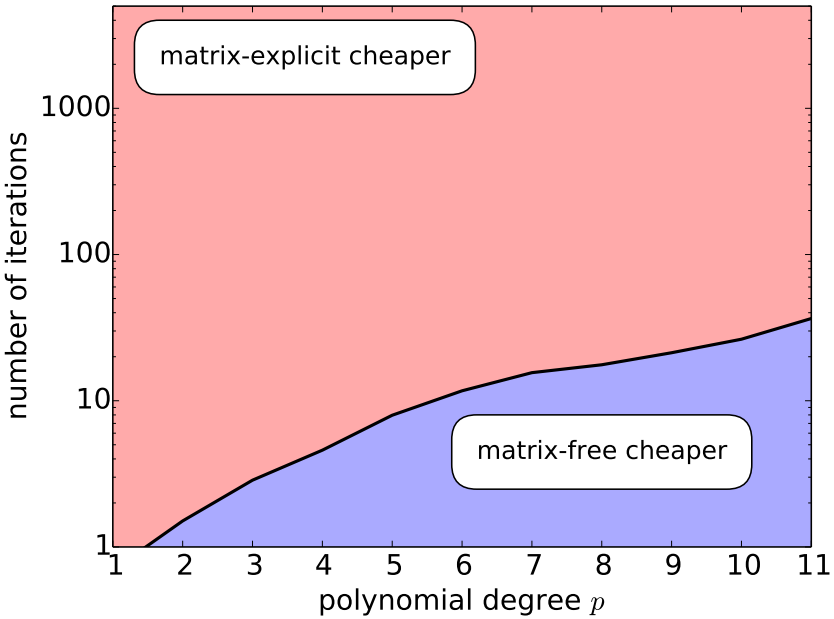
When does matrix-free pay off?

Block diagonal operator application

n = Number of iterations

$$T^{ME}(n) = t_{assemble} + t_{LU-factor} + n \cdot t^{ME}_{LU-solve}$$

$$T^{MF}(n, n_{\text{block}}) = n_{\text{block}} \cdot n \cdot t^{MF}_{apply}$$



Crossover:

$$T^{ME}(n) = T^{MF}(n, n_{\text{block}} = 10)$$