# Mixed-Precision GPU-Multigrid Solvers with Strong Smoothers and Applications in CFD and CSM

Dominik Göddeke and Robert Strzodka

Institut für Angewandte Mathematik (LS3), TU Dortmund
Max Planck Institut Informatik, Saarbrücken

dominik.goeddeke@math.tu-dortmund.de
http://www.mathematik.tu-dortmund.de/~goeddeke

ENUMATH 2011 Mini-Symposium
Leicester, UK, September 7

technische universität dortmund
fakultät für mathematik
Max Planck Center for visual computing and communication

# Reprise: Hardware-oriented numerics

**Conflicting situations**

- Existing methods no longer hardware-compatible
- Neither want less numerical efficiency, nor less hardware efficiency

**Challenge: New algorithmic way of thinking**

- Balance these conflicting goals

**Consider short-term hardware details in actual implementations, but long-term hardware trends in the design of numerical schemes**

- Locality, locality, locality
- Commmunication-avoiding (-delaying) algorithms between all flavours of parallelism
- Multilevel methods, hardware-aware preconditioning

# Grid and Matrix Structures

# Flexibility $\leftrightarrow$ Performance

# Grid and matrix structures

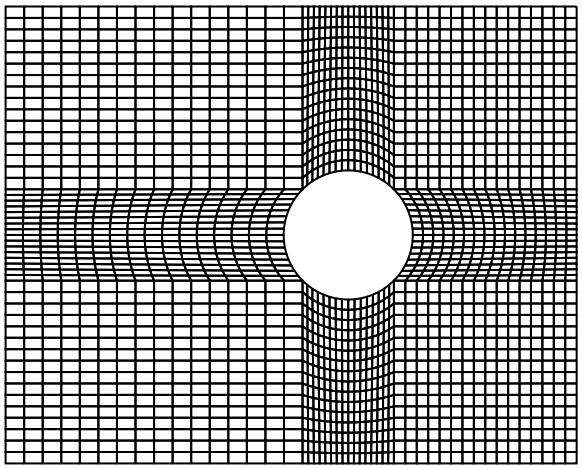
## General sparse matrices (unstructured grids)

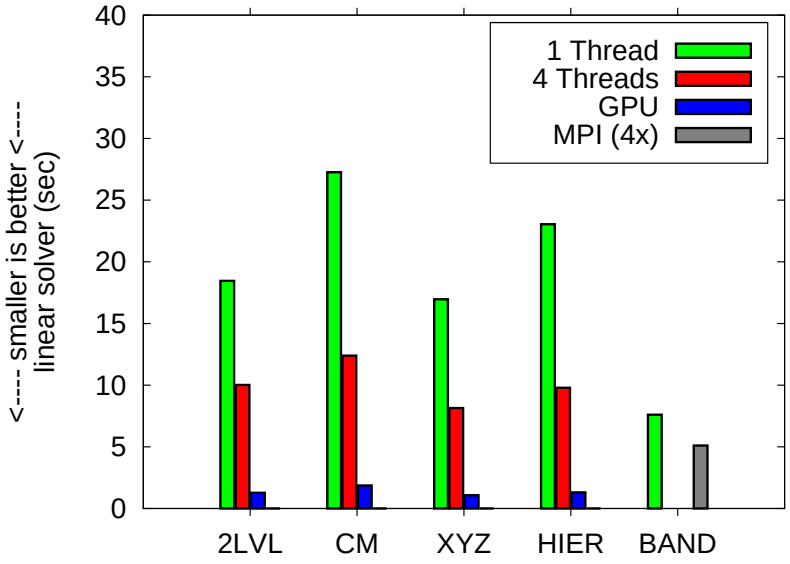
- CSR (and variants): General data structure for arbitrary grids
- Maximum flexibility, but during SpMV
  - Indirect, irregular memory accesses
  - Index overhead reduces already low arithm. intensity further
- Performance depends on nonzero pattern (grid numbering)

## Structured sparse matrices

- Example: Structured grids, suitable numbering $\Rightarrow$ band matrices
- Important: No stencils, fully variable coefficients
- Direct regular memory accesses, fast independent of mesh
- 'FEAST patches': Exploitation in the design of strong MG components

# Example: Poisson on unstructured mesh






- Nehalem vs. GT200, $\approx$ 2M bilinear FE, MG-JAC solver
- Unstructured formats highly numbering-dependent
- Multicore 2–3x over singlecore, GPU 8–12x over multicore
- Banded format (here: 8 'blocks') 2–3x faster than best unstructured layout and predictably on par with multicore

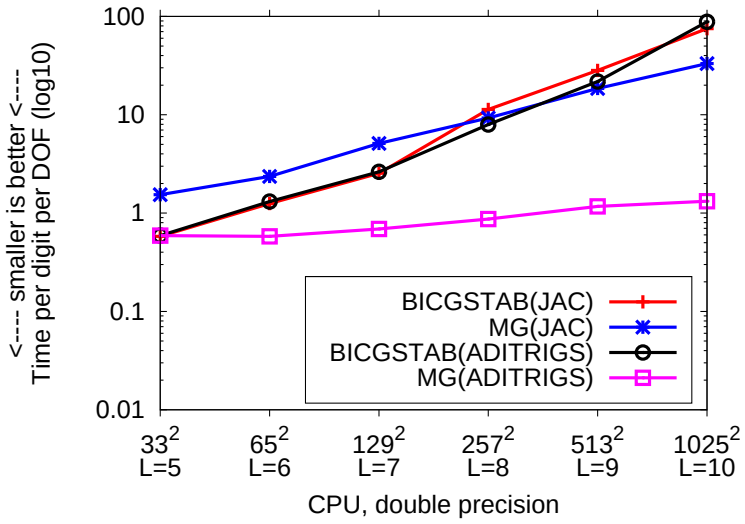
# Strong Smoothers

# Parallelising Inherently Sequential Operations

# Motivation: Why strong smoothers?

**Test case: Generalised Poisson problem with anisotropic diffusion**

- $-\nabla \cdot (\mathbf{G}\ \nabla \mathbf{u}) = \mathbf{f}$ on unit square (one FEAST patch)
- $\mathbf{G} = \mathbf{I}$: standard Poisson problem, $\mathbf{G} \neq \mathbf{I}$: arbitrarily challenging
- Example: $\mathbf{G}$ introduces anisotropic diffusion along some vector field




**Only multigrid with a strong smoother is competitive**

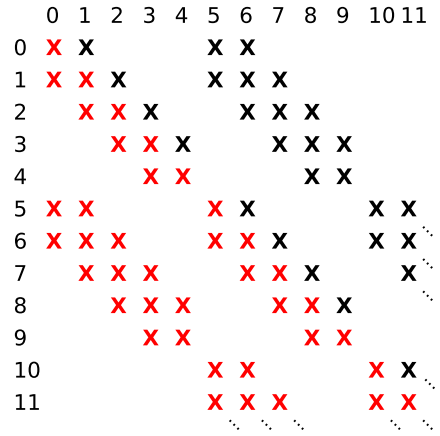
# Gauß-Seidel smoother

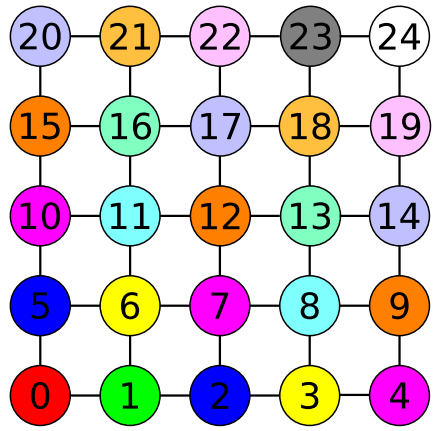
Disclaimer: Not necessarily a good smoother, but a good didactical example.

## Sequential algorithm

- Forward elimination, sequential dependencies between matrix rows
- Illustrative: Coupling to the left and bottom

## 1st idea: Classical wavefront-parallelisation (exact)



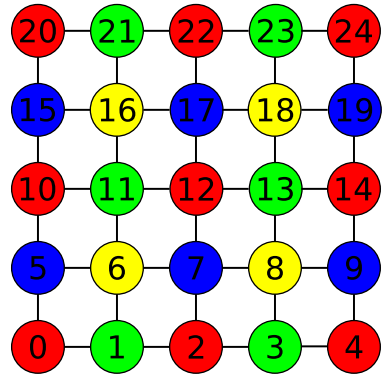




- Pro: Always works to resolve *explicit* dependencies
- Con: Irregular parallelism and access patterns, implementable?

# Gauß-Seidel smoother

## 2nd idea: Decouple dependencies via multicolouring (inexact)

- Jacobi (red) – coupling to left (green) – coupling to bottom (blue) – coupling to left and bottom (yellow)



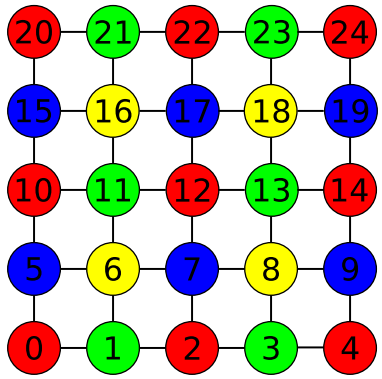

## Analysis

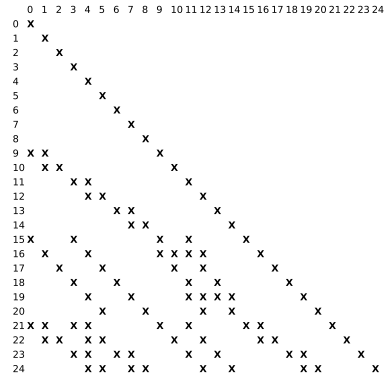
- Parallel efficiency: 4 sweeps with $\approx N/4$ parallel work each
- Regular data access, but checkerboard pattern challenging for SIMD/GPUs due to strided access
- Numerical efficiency: Sequential coupling only in last sweep

# Gauß-Seidel smoother

## 3rd idea: Multicolouring = renumbering

- After decoupling: 'Standard' update (left+bottom) is suboptimal
- Does not include all already available results



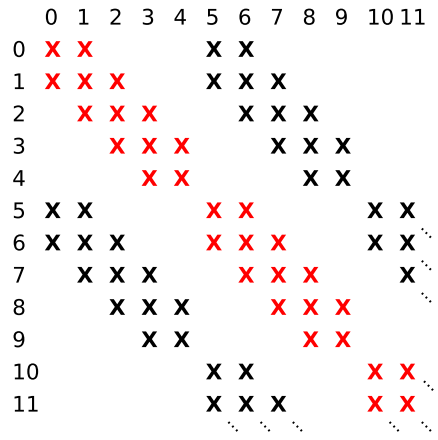



- Recoupling: Jacobi (red) – coupling to left and right (green) – top and bottom (blue) – all 8 neighbours (yellow)
- More computations that standard decoupling
- Experiments: Convergence rates of sequential variant recovered (in absence of preferred direction)

# Tridiagonal smoother (line relaxation)

## Starting point

- Good for 'line-wise' anisotropies
- *'Alternating Direction Implicit (ADI)'* technique alternates rows and columns
- CPU implementation: Thomas-Algorithm (inherently sequential)



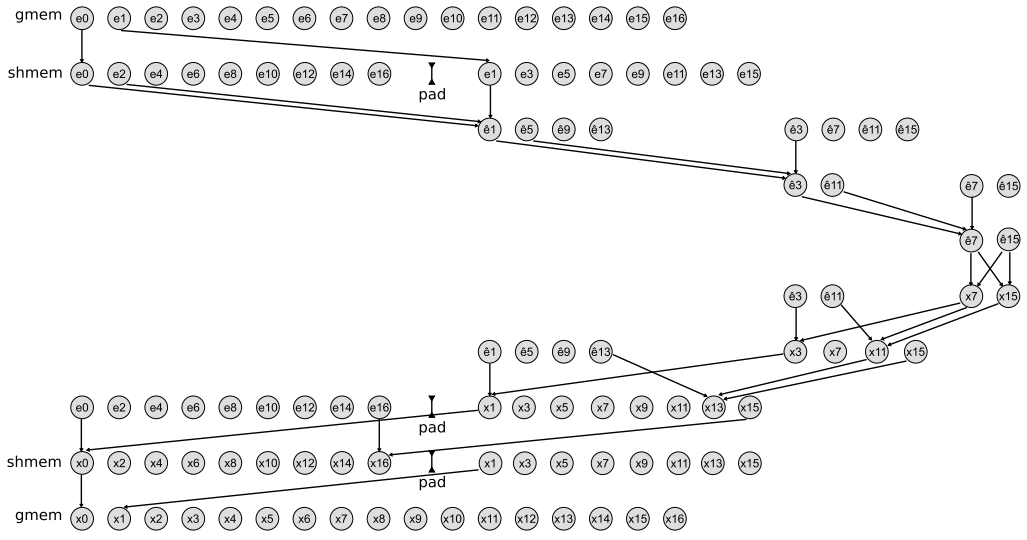

## Observations

- One independent tridiagonal system per mesh row
- $\Rightarrow$ top-level parallelisation across mesh rows
- Implicit coupling: Wavefront and colouring techniques not applicable

# Tridiagonal smoother (line relaxation)

## Cyclic reduction for tridiagonal systems

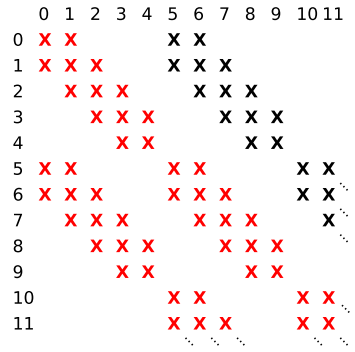
- Exact, stable (w/o pivoting) and cost-efficient
- Problem: Classical formulation parallelises computation but not memory accesses on GPUs (bank conflicts in shared memory)
- Developed a better formulation, 2-4x faster
- Index challenge, general idea: Recursive padding between odd and even indices on all levels



# Combined GS and TRIDI

## Starting point

- CPU implementation: Shift previous row to RHS and solve remaining tridiagonal system with Thomas-Algorithm
- Combined with ADI, this is the best general smoother (we have) for this matrix structure



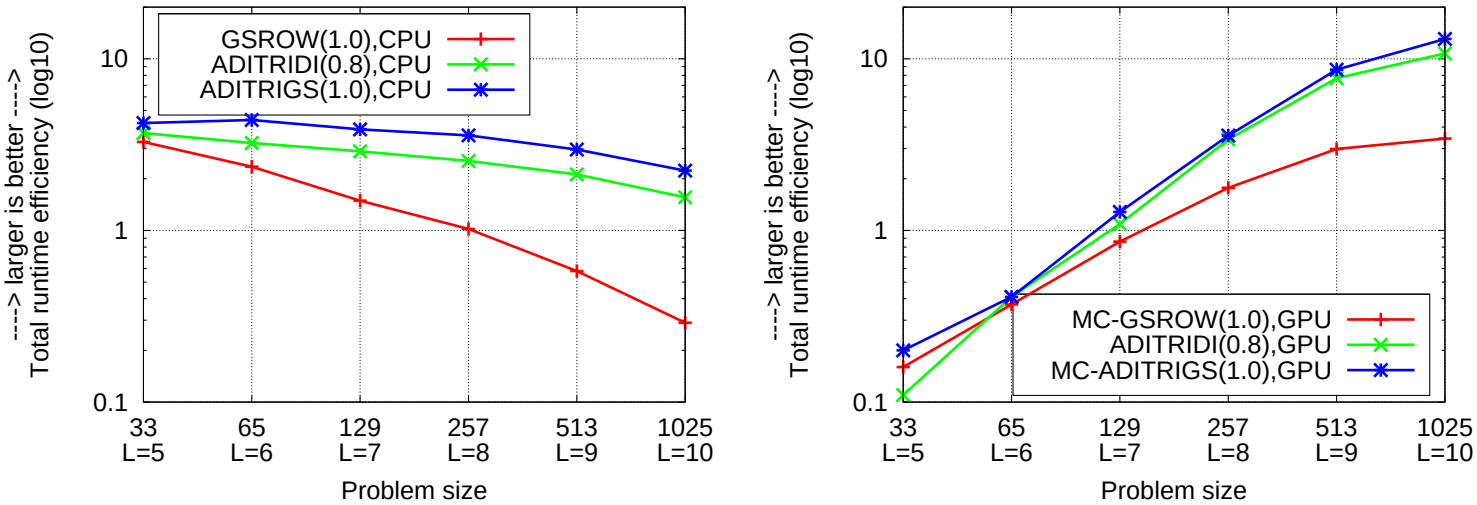

## Observations and implementation

- Difference to tridiagonal solvers: Mesh rows depend sequentially on each other
- Use colouring ($\#c \geq 2$) to decouple the dependencies between rows (more colours = more similar to sequential variant)

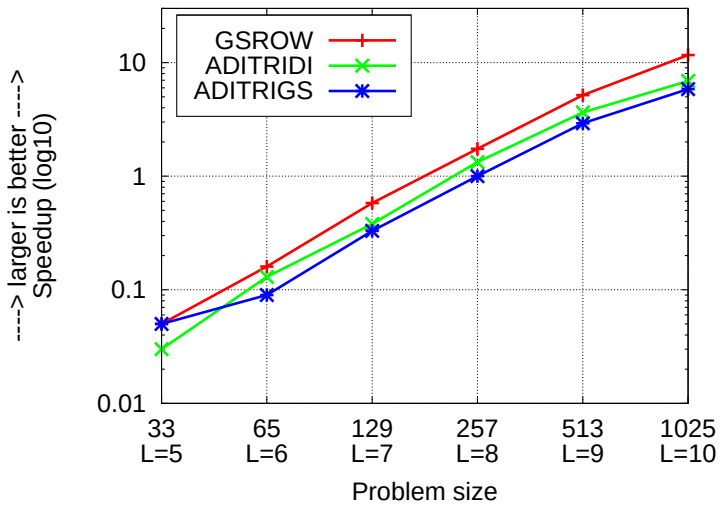
# Evaluation: Total efficiency on CPU and GPU

**Test problem: Generalised Poisson with anisotropic diffusion**

- Total efficiency: $(\mu s$ per unknown per digit$)^{-1}$
- Mixed precision iterative refinement multigrid solver
- Intel Westmere vs. NVIDIA Fermi



# Speedup GPU vs. CPU




## Summary: Smoother parallelisation

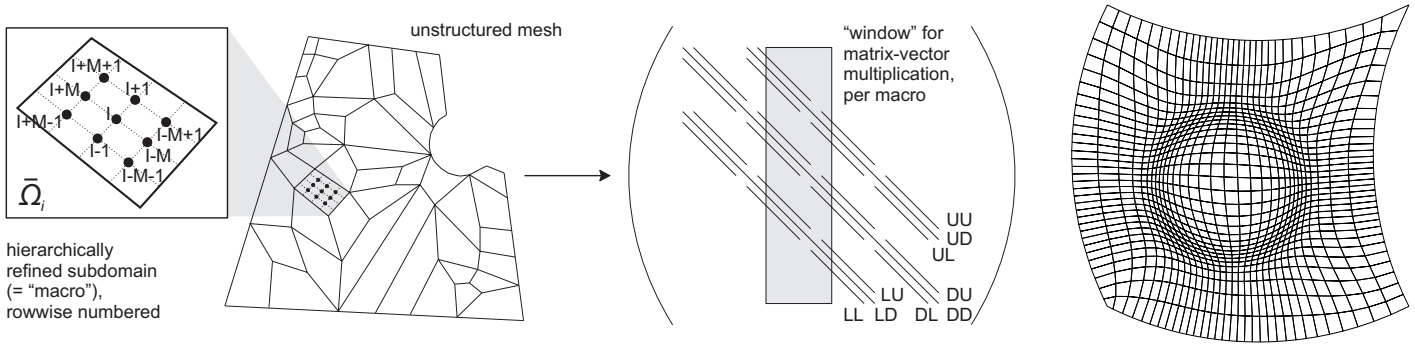
- Factor 10-30 (dep. on precision and smoother selection) speedup over already highly tuned CPU implementation
- Same numerical capabilities on CPU and GPU
- Balancing of numerical and parallel efficiency (hardware-oriented numerics)

# CSM and CFD on GPU-Accelerated Clusters

# ScaRC Solvers in FEAST

## Combination of structured and unstructured advantages

- Global macro-mesh: Unstructured, flexible, complex domains
- Local micro-meshes: Structured (logical TP-structure), fast
- Important: Structured $\neq$ simple meshes!




## Hybrid multilevel domain decomposition method

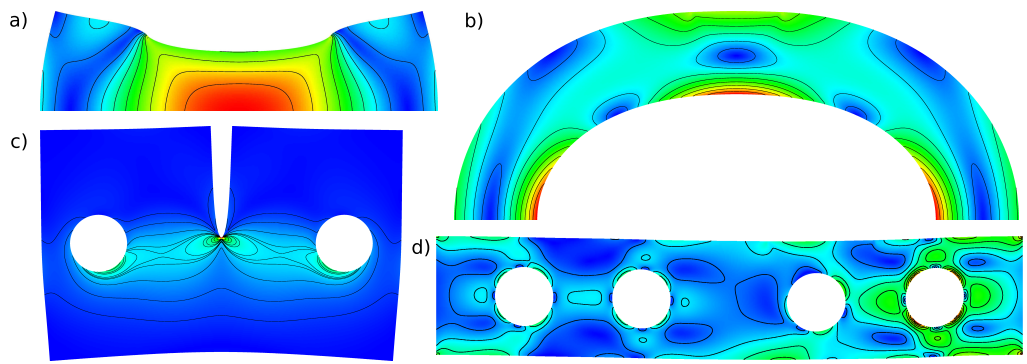
- Multiplicative between levels, global coarse grid problem (MG-like)
- Additive horizontally: block-Jacobi / Schwarz smoother (DD-like)
- Local GPU-accelerated MG hides local irregularities

# Linearised elasticity

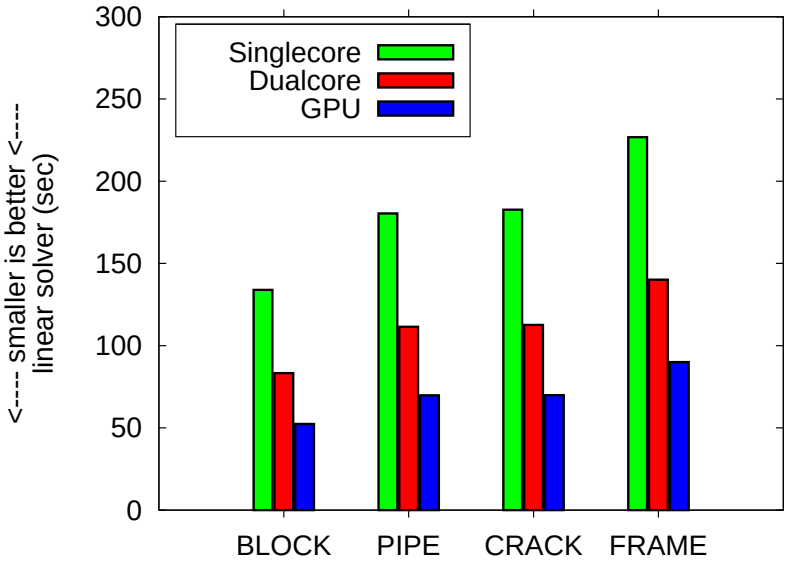
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \mathbf{f}$$

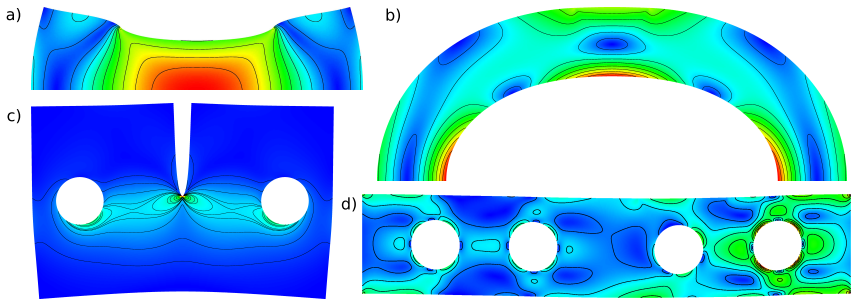
$$\begin{pmatrix} (2\mu+\lambda)\partial_{xx} + \mu\partial_{yy} & (\mu+\lambda)\partial_{xy} \\ (\mu+\lambda)\partial_{yx} & \mu\partial_{xx} + (2\mu+\lambda)\partial_{yy} \end{pmatrix}$$

**global multivariate BiCGStab**
block-preconditioned by
**Global multivariate multilevel** (V 1+1)
additively smoothed (block GS) by
for all $\Omega_i$: solve $\mathbf{A}_{11}\mathbf{c}_1 = \mathbf{d}_1$ by
**local scalar multigrid**
update RHS: $\mathbf{d}_2 = \mathbf{d}_2 - \mathbf{A}_{21}\mathbf{c}_1$
for all $\Omega_i$: solve $\mathbf{A}_{22}\mathbf{c}_2 = \mathbf{d}_2$ by
**local scalar multigrid**
coarse grid solver: UMFPACK



# Speedup




- USC cluster in Los Alamos, 16 dualcore nodes (Opteron Santa Rosa, Quadro FX5600)
- Problem size 128 M DOF
- Dualcore 1.6x faster than singlecore (memory wall)
- GPU 2.6x faster than singlecore, 1.6x than dualcore

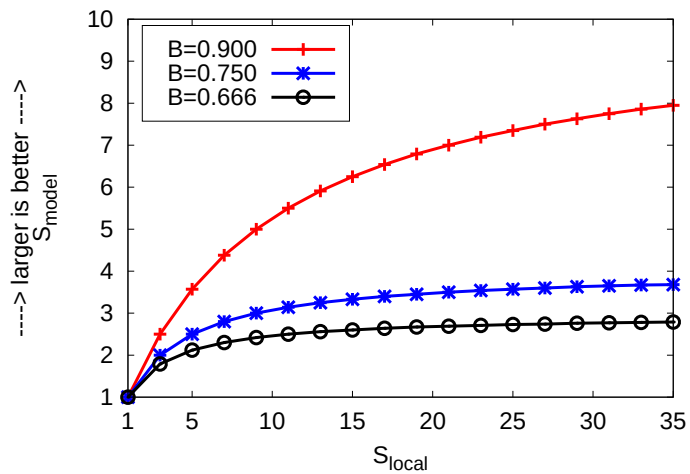
# Speedup analysis

## Theoretical model of expected speedup

- Integration of GPUs increases resources
- Correct model: Strong scaling within each node
- Acceleration potential of the elasticity solver: $R_{\text{acc}} = 2/3$ (remaining time in MPI and the outer solver)
- $S_{\text{max}} = \frac{1}{1-R_{\text{acc}}}$ $\qquad$ $S_{\text{model}} = \frac{1}{(1-R_{\text{acc}})+(R_{\text{acc}}/S_{\text{local}})}$

## This example

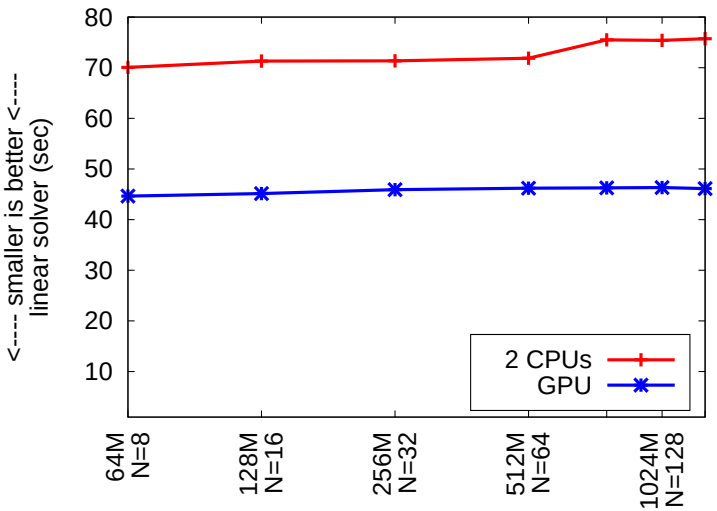
| | |
|---|---|
| Accelerable fraction $R_{\text{acc}}$ | 66% |
| Local speedup $S_{\text{local}}$ | 9x |
| Modeled speedup $S_{\text{model}}$ | 2.5x |
| Measured speedup $S_{\text{total}}$ | 2.6x |
| Upper bound $S_{\text{max}}$ | 3x |

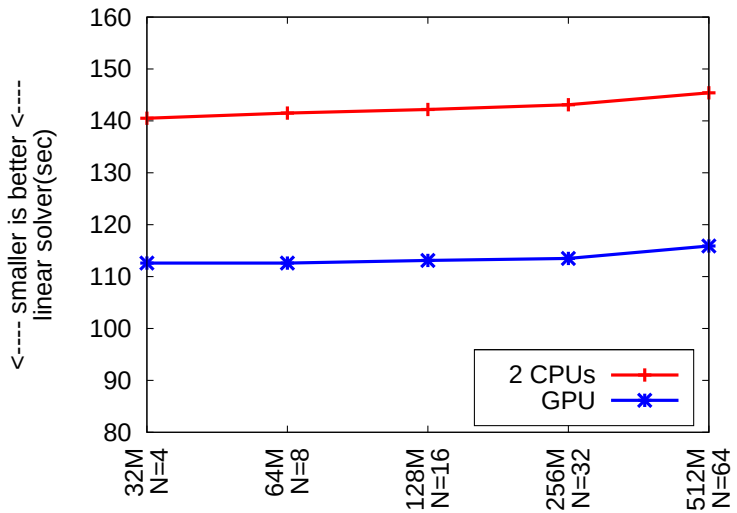


# Weak scalability

## Simultaneous doubling of problem size and resources

- Left: Poisson, 160 dual Xeon / FX1400 nodes, max. 1.3 B DOF
- Right: Linearised elasticity, 64 nodes, max. 0.5 B DOF




## Results

- No loss of weak scalability despite local acceleration
- 1.3 billion unknowns (no stencil!) on 160 GPUs in less than 50 s

# Stationary laminar flow (Navier-Stokes)

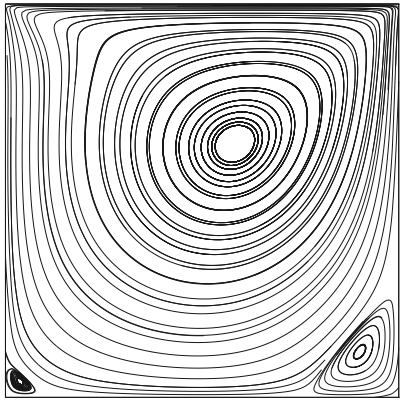
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{B}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^T & \mathbf{B}_2^T & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{g} \end{pmatrix}$$

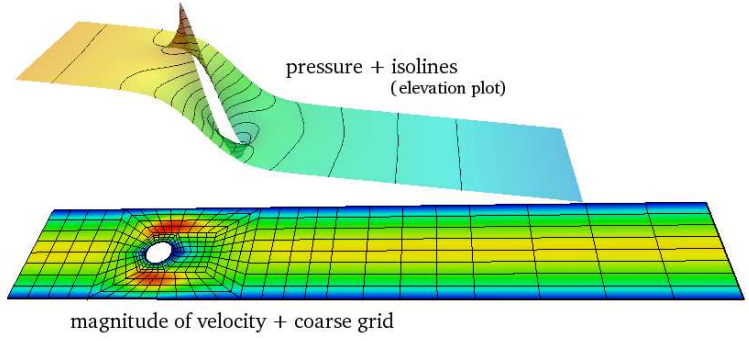
**fixed point iteration**
assemble linearised subproblems and solve with
**global BiCGStab** (reduce initial residual by 1 digit)
Block-Schurcomplement preconditioner
1) approx. solve for velocities with
**global MG** (V 1+0), additively smoothed by

for all $\Omega_i$: solve for $\mathbf{u}_1$ with
**local MG**

for all $\Omega_i$: solve for $\mathbf{u}_2$ with
**local MG**

2) update RHS: $\mathbf{d}_3 = -\mathbf{d}_3 + \mathbf{B}^T(\mathbf{c}_1, \mathbf{c}_2)^T$
3) scale $\mathbf{c}_3 = (\mathbf{M}_p^L)^{-1}\mathbf{d}_3$





# Stationary laminar flow (Navier-Stokes)

## Solver configuration

- Driven cavity: Jacobi smoother sufficient
- Channel flow: ADI-TRIDI smoother required

## Speedup analysis

| | $R_{\text{acc}}$ | | $S_{\text{local}}$ | | $S_{\text{total}}$ | |
|---|---|---|---|---|---|---|
| | L9 | L10 | L9 | L10 | L9 | L10 |
| DC Re250 | 52% | 62% | 9.1x | 24.5x | 1.63x | 2.71x |
| Channel flow | 48% | – | 12.5x | – | 1.76x | – |

## FE assembly vs. linear solver, max. problem size

| DC Re250 | | Channel | |
|---|---|---|---|
| CPU | GPU | CPU | GPU |
| 12:88 | 31:67 | 38:59 | **68:28** |

# Summary

# Summary

**Grid and data layouts**

- ScaRC approach: locally structured, globally unstructured

**GPU computing**

- Parallelising numerically strong recursive smoothers
- More than an order of magnitude speedup

**Scale-out to larger clusters**

- Minimally invasive integration
- Good speedup despite 'Amdahl's Law'
- Excellent weak scalability
- One GPU code to accelerate CSM and CFD applications built on top of ScaRC

# Acknowledgements


## Collaborative work with

- FEAST group (TU Dortmund): Ch. Becker, S.H.M. Buijssen, M. Geveler, D. Göddeke, M. Köster, D. Ribbrock, Th. Rohkämper, S. Turek, H. Wobker, P. Zajac
- Robert Strzodka (Max Planck Institut Informatik)
- Jamaludin Mohd-Yusof, Patrick McCormick (Los Alamos National Laboratory)

## Supported by

- DFG: TU 102/22-1, TU 102/22-2
- BMBF: *HPC Software für skalierbare Parallelrechner*: SKALB project 01IH08003D


## Papers

`http://www.mathematik.tu-dortmund.de/~goeddeke`