Mixed-Precision GPU-Multigrid Solvers with Strong Smoothers

Dominik Göddeke

Institut für Angewandte Mathematik (LS3)
TU Dortmund
dominik.goeddeke@math.tu-dortmund.de

ILAS 2011
Mini-Symposium: Parallel Computing in Numerical Linear Algebra
Braunschweig, August 24, 2011

technische universität dortmund
fakultät für mathematik

The Big Picture

Hardware evolution

- Memory wall: Data movement cost prohibitively expensive
- Power wall: Nuclear power plant for each machine (in the cloud)?
- ILP wall: 'Automagic' maximum resource utilisation?
- Memory wall + power wall + ILP wall = brick wall

Inevitable paradigm shift: Parallelism and heterogeneity

- In a single chip: singlecore $\rightarrow$ multicore, manycore, ...
- In a workstation (cluster node): NUMA, CPUs and GPUs, ...
- In a big cluster: different nodes, communication characteristics, ...

This is our problem as mathematicians

- Affects standard workstations and even laptops
- Compilers and (most often) libraries don't hide these issues from us

Consequences for Numerics

Parallelism is inevitable

- Impossible to exploit ever increasing peak performance
- Sequential codes even run slower on newer hardware (!)

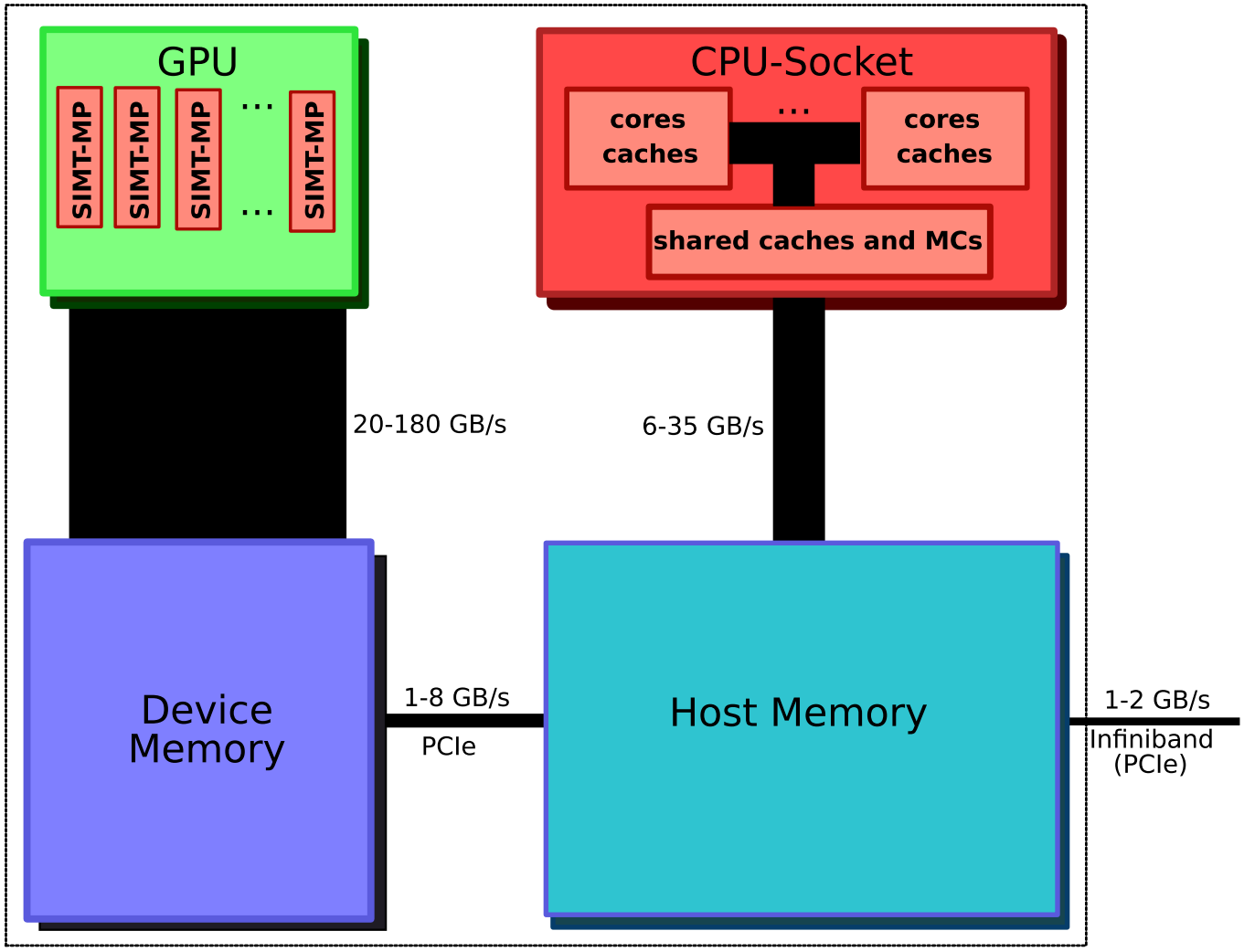
Challenges

- Technical: Compilers can't solve these problems, libraries are limited
- Numerical: Traditional methods often contrary to hardware trends
- Goal: Redesign existing numerical schemes (and invent new ones) to work well in the fine-grained parallel setting

GPUs ('manycore') are forerunners of this development

- 10 000s of simultaneously active threads
- Promises of significant speedups
- Focus of this talk: iterative solvers for sparse systems

GPUs and the Memory Wall Problem



Mixed Precision Iterative Refinement

Combatting the memory wall problem

Motivation

Switching from double to single precision (DP→SP)

- 2x effective memory bandwidth, 2x effective cache size
- At least 2x compute speed (often 4–12x)

Problem: Condition number

- For all problems in this talk: $\text{cond}_2(\mathbf{A}) \sim h_{\text{min}}^{-2}$
- Theory for linear system $\mathbf{Ax} = \mathbf{b}$

$$\text{cond}_2(\mathbf{A}) \sim 10^s; \frac{\|\mathbf{A} + \delta\mathbf{A}\|}{\|\mathbf{A}\|}, \frac{\|\mathbf{b} + \delta\mathbf{b}\|}{\|\mathbf{b}\|} \sim 10^{-k} (k > s) \quad \Rightarrow \quad \frac{\|\mathbf{x} + \delta\mathbf{x}\|}{\|\mathbf{x}\|} \sim 10^{s-k}$$

In our setting

- Truncation error in 7–8th digit increased by s digits

Numerical Example

Poisson problem on unit square

- Simple yet fundamental
- $\text{cond}_2(\mathbf{A}) \approx 10^5$ for $L = 10$ (1M bilinear FE, regular grid)
- Condition number usually much higher: anisotropies in grid and operator

Level	Data+Comp. in DP L_2 Error	Red.	Data in SP, Compute in DP L_2 Error	Red.	Data+Comp. in SP L_2 Error	Red.
5	1.1102363E-3	4.00	1.1102371E-3	4.00	1.1111655E-3	4.00
6	2.7752805E-4	4.00	2.7756739E-4	4.00	2.8704684E-4	3.87
7	6.9380072E-5	4.00	6.9419428E-5	4.00	1.2881795E-4	2.23
8	1.7344901E-5	4.00	1.7384278E-5	3.99	4.2133101E-4	0.31
9	4.3362353E-6	4.00	4.3757082E-6	3.97	2.1034461E-3	0.20
10	1.0841285E-6	4.00	1.1239630E-6	3.89	8.8208778E-3	0.24

$\Rightarrow$ **Single precision insufficient for moderate problem sizes already**

Mixed Precision Iterative Refinement

Iterative refinement

- Established algorithm to provably guarantee accuracy of computed results (within given precision)
  - High precision: $\mathbf{d} = \mathbf{b} - \mathbf{A}\mathbf{x}$ (cheap)
  - Low precision: $\mathbf{c} = \mathbf{A}^{-1}\mathbf{d}$ (expensive)
  - High precision: $\mathbf{x} = \mathbf{x} + \mathbf{c}$ (cheap) and iterate (expensive?)
- Convergence to high precision accuracy if $\mathbf{A}$ *'not too ill-conditioned'*
- Theory: Number of iterations linear in $\log(\text{cond}_2(\mathbf{A}))$ and $\log(\varepsilon_{\text{high}}/\varepsilon_{\text{low}})$

New idea (hardware-oriented numerics)

- Use this algorithm to improve time to solution and thus efficiency of linear system solvers
- Goal: Result accuracy of high precision with speed of low precision floating point format

Iterative Refinement for Large Sparse Systems

Refinement procedure not immediately applicable

- 'Exact' solution using 'sparse LU' techniques too expensive
- Convergence of iterative methods not guaranteed in single precision

Solution

- Interpretation as a preconditioned mixed precision defect correction iteration

$$\mathbf{x}_{\mathrm{DP}}^{(k+1)} = \mathbf{x}_{\mathrm{DP}}^{(k)} + \mathbf{C}_{\mathrm{SP}}^{-1}(\mathbf{b}_{\mathrm{DP}} - \mathbf{A}_{\mathrm{DP}}\mathbf{x}_{\mathrm{DP}}^{(k)})$$

- Preconditioner $\mathbf{C}_{\mathrm{SP}}$ in single precision:
'Gain digit(s)' or 1-3 MG cycles instead of exact solution

Results (MG and Krylov for Poisson problem)

- Speedup at least 1.7x (often more) without loss in accuracy
- Asymptotic optimal speedup is 2x (bandwidth limited)

Grid- and Matrix Structures

Flexibility $\leftrightarrow$ Performance

Grid- and Matrix Structures

General sparce matrices (on 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 (mesh numbering)

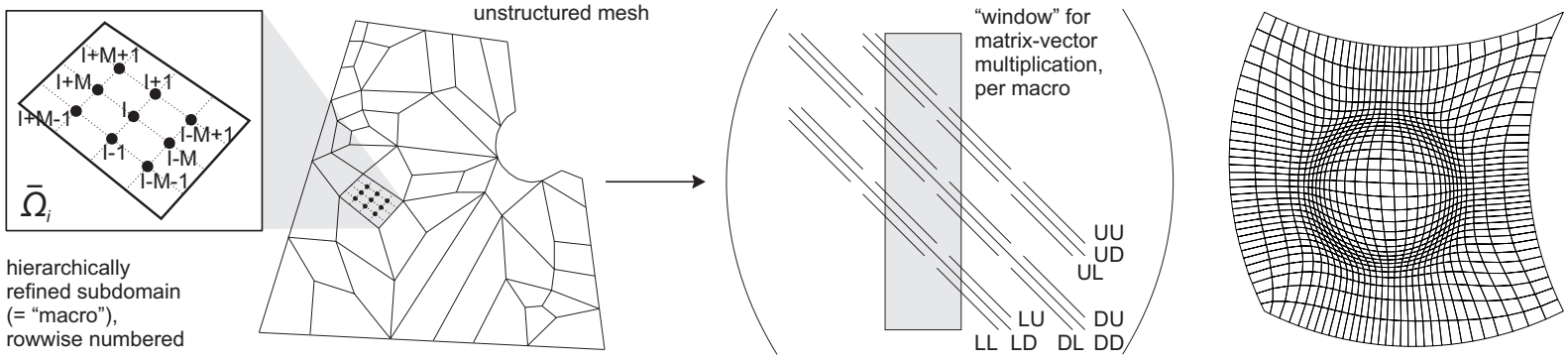
Structured matrices

- Example: structured grids, suitable numbering $\Rightarrow$ band matrices
- Important: no stencils, fully variable coefficients
- direct regular memory accesses (fast), mesh-independent performance
- Structure exploitation in the design of MG components (later)

Approach in FEAST

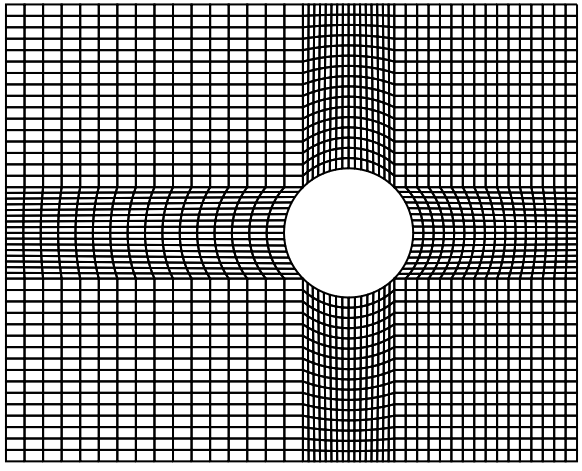
Combination of respective advantages

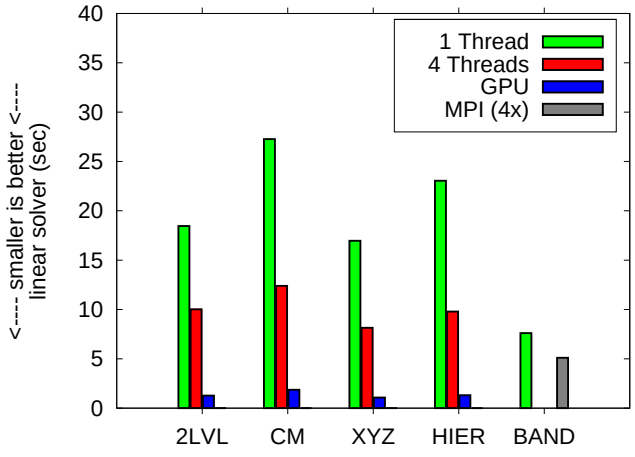
- Global macro-mesh: unstructured, flexible
- local micro-meshes: structured (logical TP-structure), fast
- Important: structured $\neq$ cartesian meshes!
- Reduce numerical linear algebra to sequences of operations on structured data (maximise locality)
- Developed for larger clusters, but generally useful



Example

Poisson on unstructured domain





- 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

Parallelising Inherently Sequential Operations

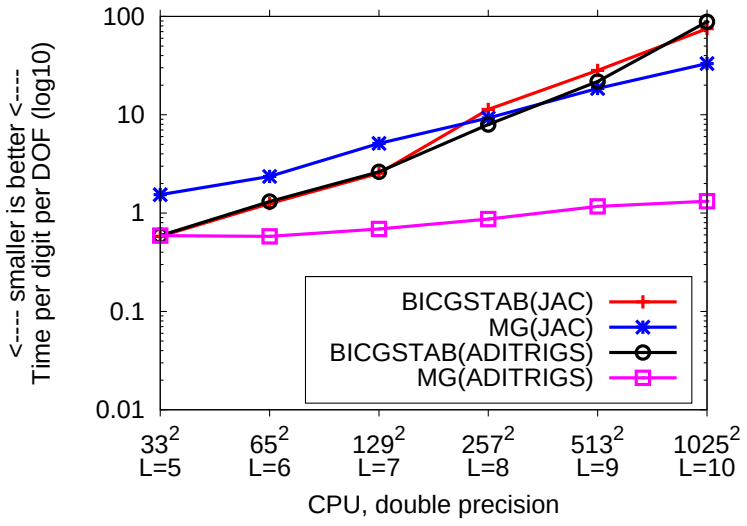
Multigrid with strong smoothers

Lots of parallelism available in inherently sequential operations

Motivation: Why Strong Smoothers?

Test case: anisotropic diffusion in generalised Poisson problem

- $-\text{div}\ (\mathbf{G}\ \ \text{grad}\ \ \mathbf{u}) = \mathbf{f}$ 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

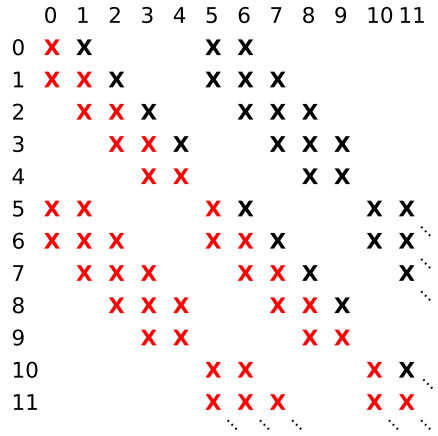
Gauß-Seidel Smoother

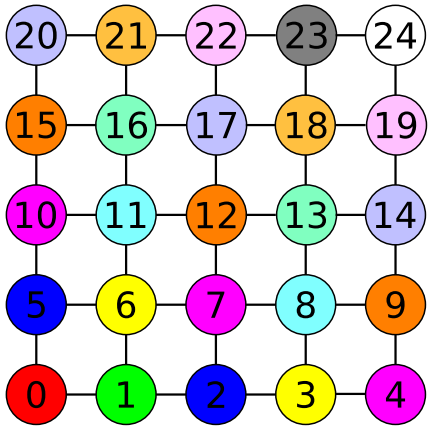
Disclaimer: Not necessarily a good smoother, but a good didactical example.

Sequential algorithm

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

1st idea: classical wavefront-parallelisation (exact)



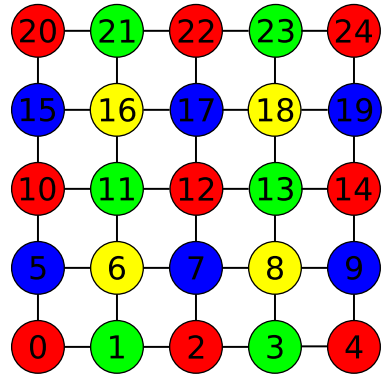


- Pro: always works to resolve *explicit* dependencies
- Con: irregular parallelism and access patterns, implementable?

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)



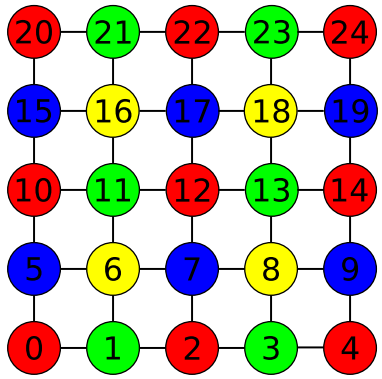
Analysis

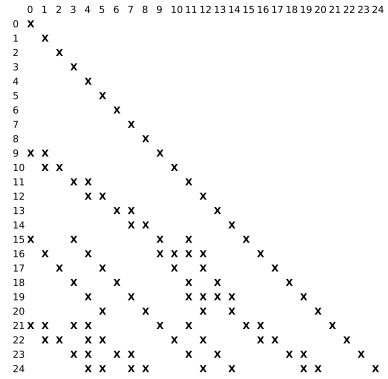
- Parallel efficiency: 4 sweeps with $\approx N/4$ parallel work each
- Regular data access, but checkerboard pattern challenging for SIMD/GPUs due to strided access
- Numerical efficiency: sequential coupling only in last sweep

Gauß-Seidel Smoother

3rd idea: multicolouring = renumbering

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



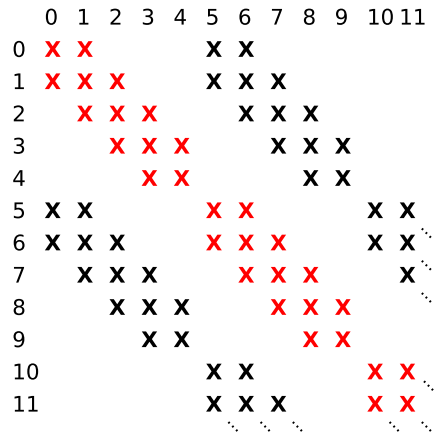


- Recoupling: Jacobi (red) – coupling to left and right (green) – top and bottom (blue) – all 8 neighbours (yellow)
- More computations that standard decoupling
- Experiments: convergence rates of sequential variant recovered (in absence of preferred direction)

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)



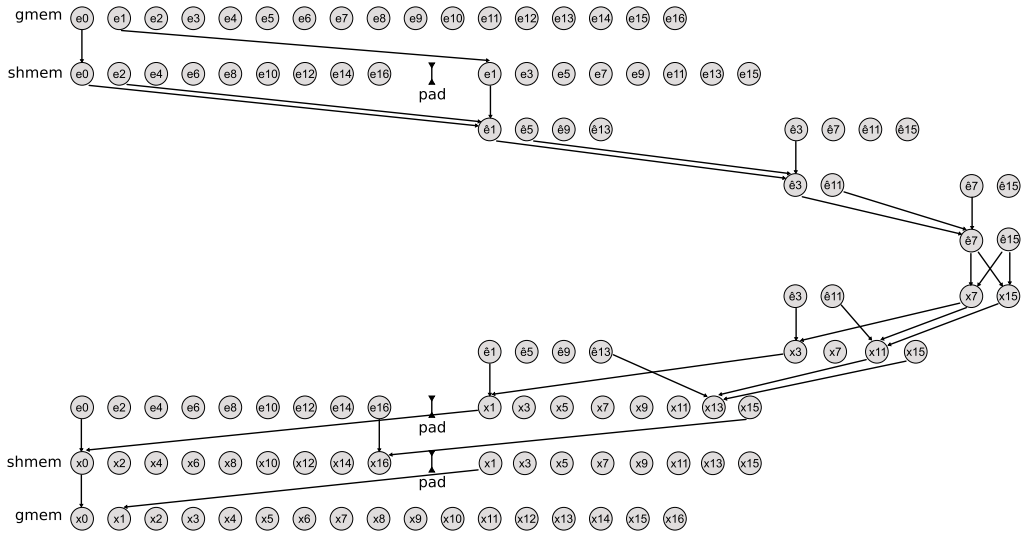
Observations

- 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

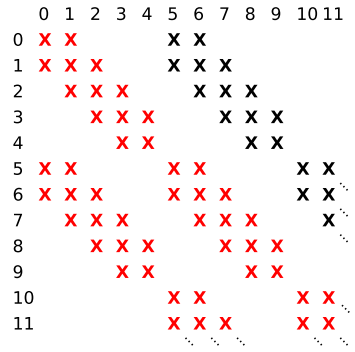
- Exact, stable (w/o pivoting) and cost-efficient
- Problem: classical formulation parallelises computation but not memory accesses on GPUs (bank conflicts in shared memory)
- Developed a better formulation, 2-4x faster
- Index nightmare, general idea: recursive padding between odd and even indices on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



Smoother Parallelisation: 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 know) for this matrix structure



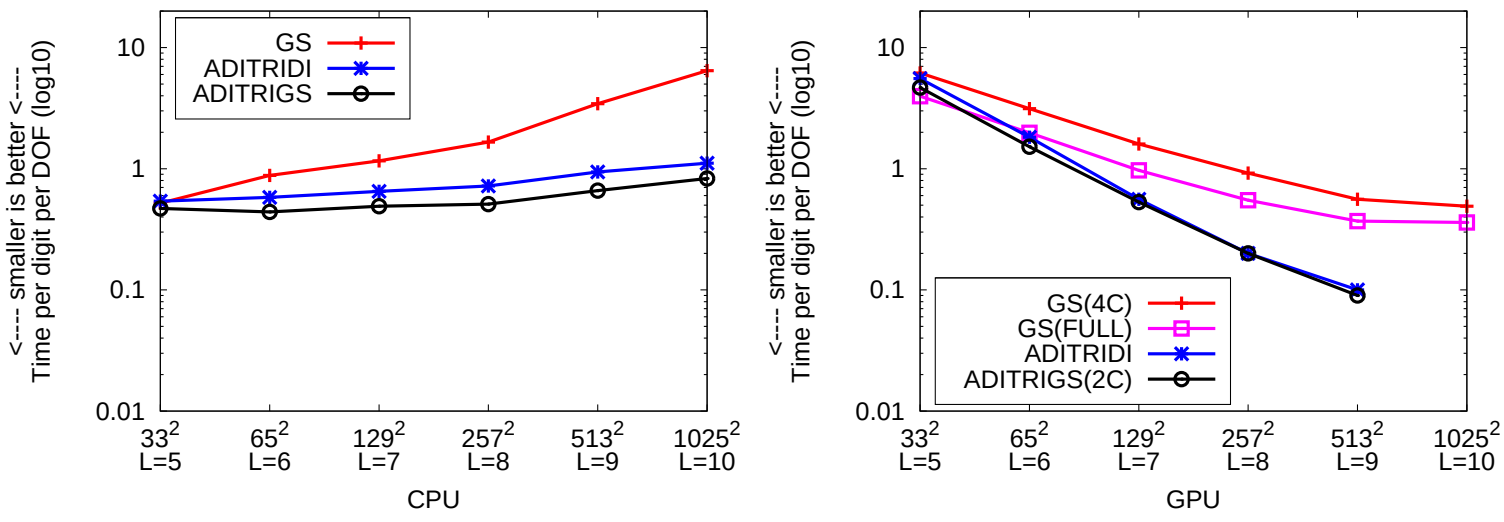
Observations and implementation

- 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)

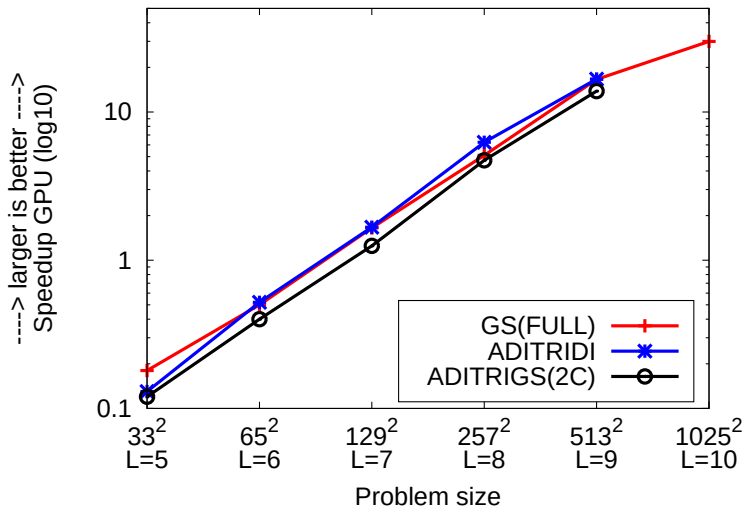
Evaluation: Total Efficiency on CPU and GPU

Test problem: generalised Poisson with anisotropic diffusion

- Total efficiency: time per unknown per digit (μs)
- Mixed precision iterative refinement multigrid solver



Speedup GPU vs. CPU

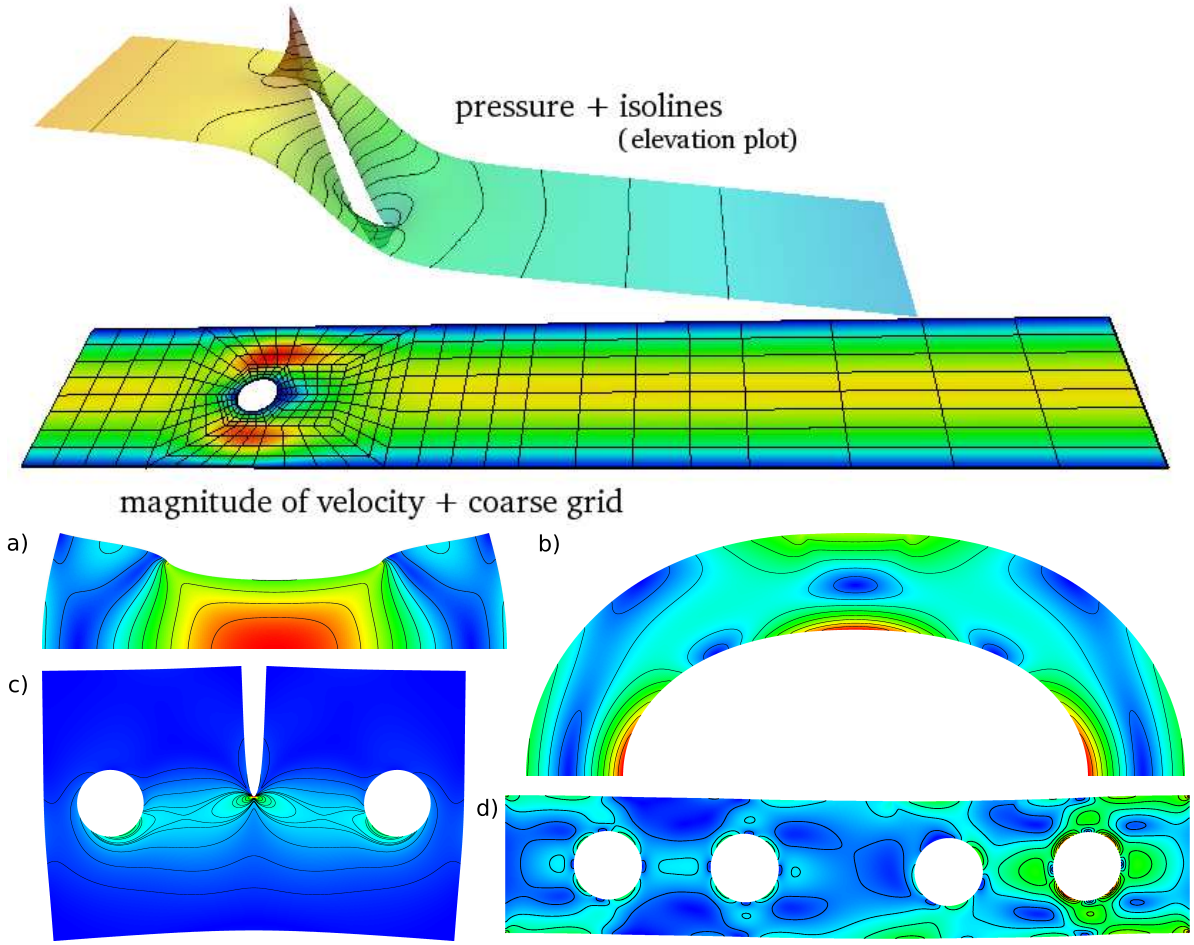


Summary: smoother parallelisation

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

Extension to Heterogeneous Clusters

FEAST on Heterogeneous Clusters



Summary and Conclusions

Summary

Hardware

- Paradigm shift: Heterogeneity, parallelism and specialisation
- Locality and parallelism on many levels
  - In one GPU (fine-granular)
  - In a compute node between heterogeneous resources (medium-granular)
  - In big clusters between compute nodes (coarse-granular)

Hardware-oriented numerics

- Design new numerical methods ‘matching’ the hardware
- Only way to achieve future-proof continuous scaling

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

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