Hardware-Oriented Finite Element Multigrid Solvers for PDEs

Dominik Göddeke

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

ASIM Workshop ‘Trends in CSE’, Garching, March 15 2011

technische universität dortmund

fakultät für mathematik

Motivation and Introduction

Hardware isn't our friend any more

- Power wall + memory wall + ILP wall = brick wall
- Pax MPI is over, we are in the middle of a paradigm shift
- On-chip parallelism grows exponentially
- Data movement cost gets prohibitively expensive
- Resources are heterogeneous (NUMA, multicores, GPUs, . . .)

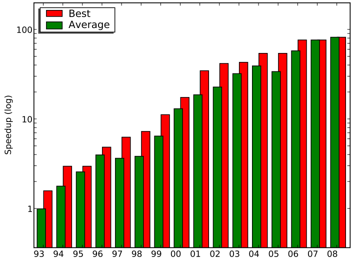
Challenges in CSE and HPC

- Existing codes just don't run faster automatically any more
- Compilers don't solve these problems, libraries are limited
- Traditional numerics is often contrary to these hardware trends
- *We have to take action*

Motivation and Introduction

Representative example: FeatFlow-benchmark 1993–2008

- Set of Navier-Stokes solvers based on FEM-multigrid
- Sequential, reasonably fast machines (data courtesy J. Hron)



- Good: 80x faster in 16 years without changes to source code
- But: 1 000x increase of peak performance in the same time
- Important: automatical speedup of conventional codes stagnates

Our Approach: Hardware-Oriented Numerics

Most important aspects of hardware paradigm shift

- Arithm. intensity: moving data gets prohibitively expensive
- Different levels of parallelism

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!

Talk Outline

Goal of this talk

- Thinking explicitly in parallel and of data movement is mandatory
- Unfortunately, there are many levels of parallism, each with its own communication characteristics
- Fortunately, parallelism is natural, we 'just' have to rediscover it
- Expressing parallelism in codes is a different story I won't talk about

Examples in my niche: Linear solvers for sparse systems

- Mixed precision iterative refinement (memory wall)
- Finite-element geom. multigrid for structured and unstructured grids
- Extracting fine-grained parallelism from inherently sequential ops
- Scale-out to (GPU-accelerated) clusters

Example #1:

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

	Data+Comp. in DP		Data in SP, Compute in DP		Data+Comp. in SP	
Level	L_2 Error	Red.	L_2 Error	Red.	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 $\approx f(\log(\text{cond}_2(\mathbf{A})), \log(\varepsilon_{\text{high}}/\varepsilon_{\text{low}}))$

New idea (Hardware-oriented numerics)

- Use this algorithm to improve time to solution and thus efficiency of linear system solves
- 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

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

- Preconditioner $\mathbf{C}_{\mathsf{SP}}$ 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)

Example #2:

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 (numbering of the grid points)

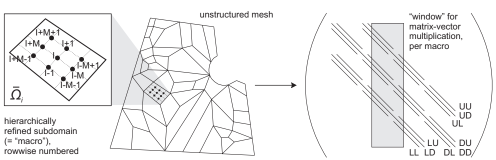
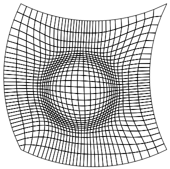
Structured matrices

- 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

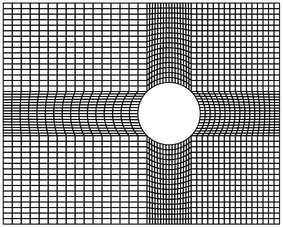
Combination of respective advantages

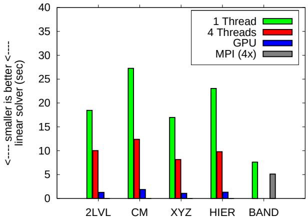
- Global macro-mesh: unstructured, flexible
- local micro-meshes: structured (logical TP-structure), fast
- Important: structured $\neq$ cartesian meshes!
- Reduce numerical linear algebra to sequences of operations on structured data (maximise locality)
- Developed for larger clusters (later), but generally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


I+M+1
I+M
I+1
I+M-1
I
I-M+1
I-1
I-M
I-M-1
$\bar{\Omega}_i$
hierarchically refined subdomain (= "macro"), rowwise numbered
unstructured mesh
"window" for matrix-vector multiplication, per macro
UU
UD
UL
LU
DU
LL
LD
DL
DD


Example

Poisson on unstructured domain





- Nehalem vs. GT200, ≈ 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

Example #3:

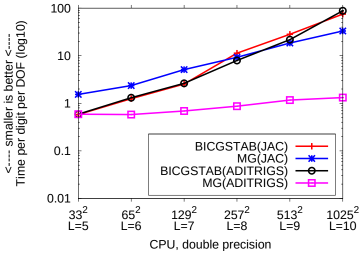
Parallelising Inherently Sequential Operations

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

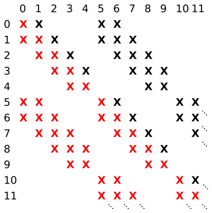
Gauß-Seidel Smoother

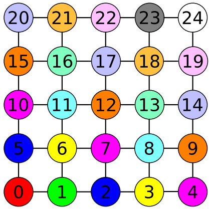
Disclaimer: Not necessarily a good smoother, but a good didactical example.

Sequential algorithm

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

1st idea: classical wavefront-parallelisation (exact)



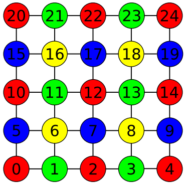


- Pro: always works to resolve *explicit* dependencies
- Con: irregular parallelism and access patterns, implementable?

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)



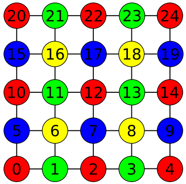
Analysis

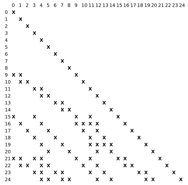
- Parallel efficiency: 4 sweeps with $\approx N/4$ parallelel work each
- Regular data access, but checkerboard pattern challenging for SIMD/GPUs due to strided access
- Numerical efficiency: sequential coupling only in last sweep

Gauß-Seidel Smoother

3rd idea: multicolouring = renumbering

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



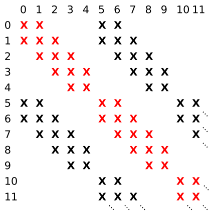


- Recoupling: Jacobi (red) – coupling to left and right (green) – top and bottom (blue) – all 8 neighbours (yellow)
- More computations that standard decoupling
- Experiments: convergence rates of sequential variant recovered (in absence of preferred direction)

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)



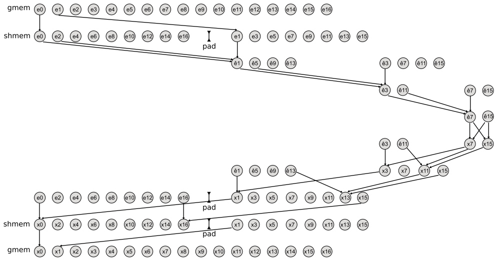
Observations

- 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

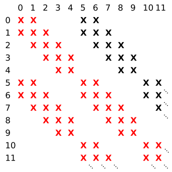
- Exact, stable (w/o pivoting) and cost-efficient
- Problem: classical formulation parallelises computation but not memory accesses on GPUs (bank conflicts in shared memory)
- Developed a better formulation, 2-4x faster
- Index nightmare, general idea: recursive padding between odd and even indices on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



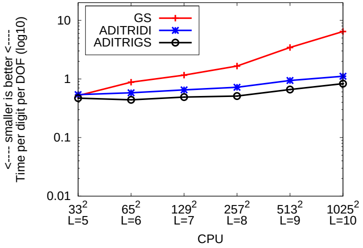
Observations and implementation

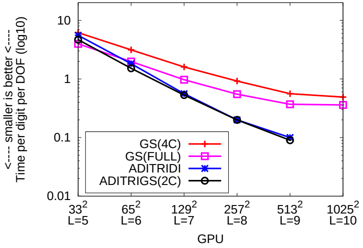
- Difference to tridiagonal solvers: mesh rows depend sequentially on each other
- Use colouring ($\#c \geq 2$) to decouple the dependencies between rows (more colours = more similar to sequential variant)

Evaluation: Total Efficiency on CPU and GPU

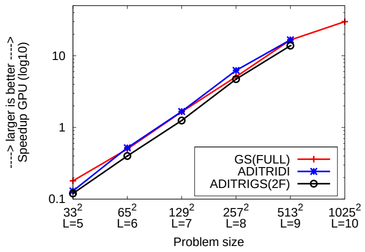
Test problem: generalised Poisson with anisotropic diffusion

- 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)

Example #4:

Scalable Multigrid Solvers on Heterogeneous Clusters

Robust coarse-grained parallel ScaRC solvers
GPU acceleration of CSM and CFD solvers

Coarse-Grained Parallel Multigrid

Goals

- Parallel efficiency: strong and weak scalability
- Numerical scalability, i.e. convergence rates independent of N
- Robust for different partitionings, anisotropies, etc.

Most important challenge

- Minimising communication between cluster nodes
- Concepts for strong smoothers so far not applicable (shared memory) due to high communication cost and synchronisation overhead
- Insufficient parallel work on coarse levels

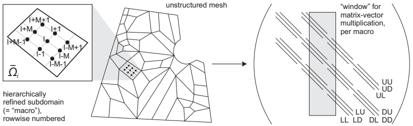
Our approach: Scalable Recursive Clustering (ScaRC)

- Under development at TU Dortmund

ScaRC: Concepts

ScaRC for scalar systems

- Hybrid multilevel domain decomposition method
- Minimal overlap by extended Dirichlet BCs
- Inspired by parallel MG (‘best of both worlds’)
  - Multiplicative between levels, global coarse grid problem (MG-like)
  - Additive horizontally: block-Jacobi / Schwarz smoother (DD-like)
- Schwarz smoother encapsulates local irregularities
  - Robust and fast multigrid (‘gain a digit’), strong smoothers
  - Maximum exploitation of local structure


I+M+1
I+M
I+1
I+M-1
I
I-M+1
I-1
I-M
I-M-1
$\bar{\Omega}_i$
hierarchically refined subdomain (= “macro”), rowwise numbered
unstructured mesh
“window” for matrix-vector multiplication, per macro
UU
UD
UL
LL
LD
LU
DL
DU
DD

global BiCGStab
preconditioned by
  global multilevel (V 1+1)
  additively smoothed by
    for all Ω_i: **local multigrid**
  coarse grid solver: UMFPACK

ScaRC for Multivariate Problems

Block-structured systems

- Guiding idea: tune scalar case once per architecture instead of over and over again per application
- Blocks correspond to scalar subequations, coupling via special preconditioners
- Block-wise treatment enables *multivariate ScaRC solvers*

Examples (2D case)

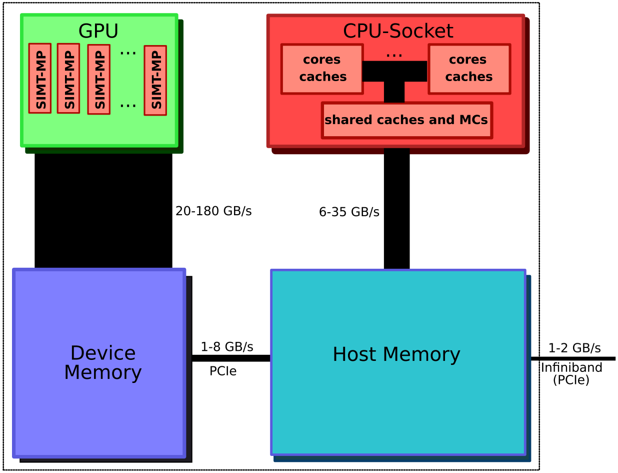
- Linearised elasticity with compressible material (2x2 blocks)
- Saddle point problems: Stokes (3x3 with zero blocks), elasticity with (nearly) incompressible material, Navier-Stokes with stabilisation (3x3 blocks)

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \mathbf{f}, \quad \begin{pmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{B}_1 \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^\mathsf{T} & \mathbf{B}_2^\mathsf{T} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{p} \end{pmatrix} = \mathbf{f}, \quad \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{B}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^\mathsf{T} & \mathbf{B}_2^\mathsf{T} & \mathbf{C}_C \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{p} \end{pmatrix} = \mathbf{f}$$

$\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ correspond to scalar (elliptic) operators
$\Rightarrow$ Tuned linear algebra **and** tuned solvers

Minimal-Invasive GPU Integration

Motivation: bandwidths in a hybrid CPU/GPU node

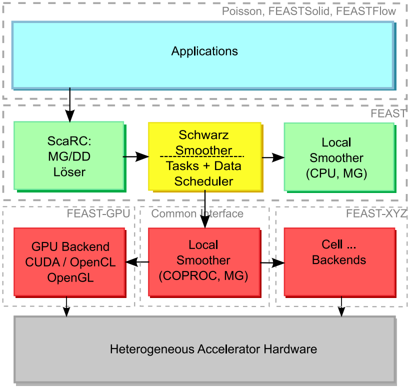


Minimally Invasive Integration

Concept: locality

- GPUs as accelerators of the most time-consuming component
- CPUs: outer MLDD solver
- No changes to applications!

global BiCGStab
preconditioned by
 global multilevel (V 1+1)
 additively smoothed by
  for all Ω_i: **local multigrid**
 coarse grid solver: UMFPACK

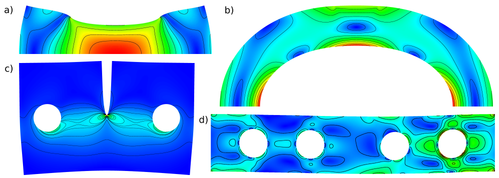


Example: Linearised Elasticity

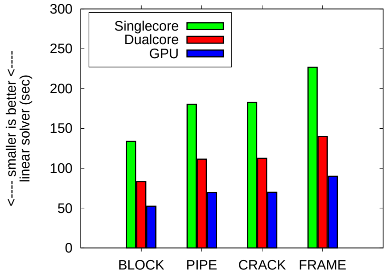
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \mathbf{f}$$

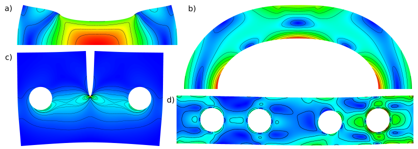
$$\begin{pmatrix} (2\mu+\lambda)\partial_{xx} + \mu\partial_{yy} & (\mu+\lambda)\partial_{xy} \\ (\mu+\lambda)\partial_{yx} & \mu\partial_{xx} + (2\mu+\lambda)\partial_{yy} \end{pmatrix}$$

```
**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 Linearised Elasticity





- 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

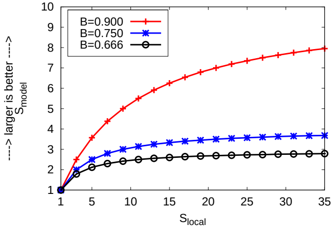
Speedup Analysis

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

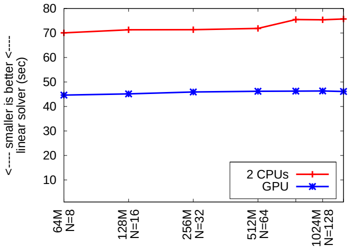
Accelerable fraction R_{acc}	66%
Local speedup S_{local}	9x
Modeled speedup S_{model}	2.5x
Measured speedup S_{total}	2.6x
Upper bound S_{max}	3x

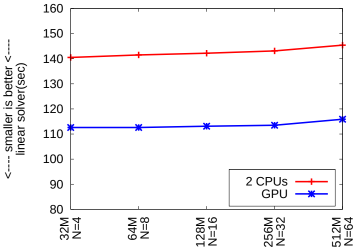


Weak Scalability

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





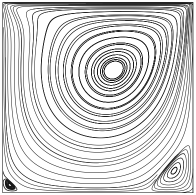
Results

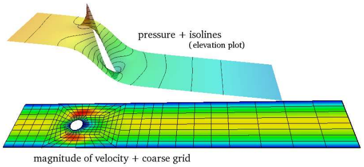
- No loss of weak scalability despite local acceleration
- 1.3 billion unknowns (no stencil!) on 160 GPUs in less than 50 s

Stationary Laminar Flow (Navier-Stokes)

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

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 Ω_i: solve for $\mathbf{u}_1$ with **local MG**
     - for all Ω_i: solve for $\mathbf{u}_2$ with **local MG**
  2) update RHS: $\mathbf{d}_3 = -\mathbf{d}_3 + \mathbf{B}^\mathsf{T}(\mathbf{c}_1, \mathbf{c}_2)^\mathsf{T}$
  3) scale $\mathbf{c}_3 = (\mathbf{M}_p^\mathsf{L})^{-1}\mathbf{d}_3$





Stationary Laminar Flow (Navier-Stokes)

Solver configuration

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

Speedup analysis

	R_{acc}		S_{local}		S_{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	–

Shift away from domination by linear solver (fraction of FE assembly and linear solver of total time, max. problem size)

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

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
- Four examples and approaches

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, TU 102/27-1, TU102/11-3
- BMBF: *HPC Software für skalierbare Parallelrechner*: SKALB project 01IH08003D

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