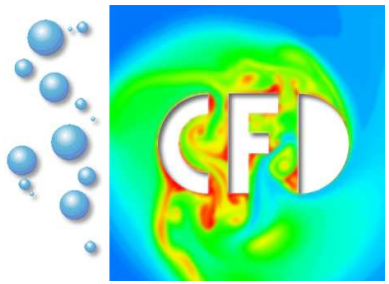
GPU Cluster Computing for Finite Element Applications

Dominik Göddeke and Stefan Turek

Applied Mathematics
TU Dortmund
dominik.goeddeke@math.tu-dortmund.de

Workshop: Experiences with the GPU and the Cell Processor
Delft, January 30, 2009



The big picture

Scientific computing faces a paradigm shift

ILP wall memory wall characteristic feature size

heat power consumption leaking voltage

Hardware evolves towards parallelism and heterogeneity

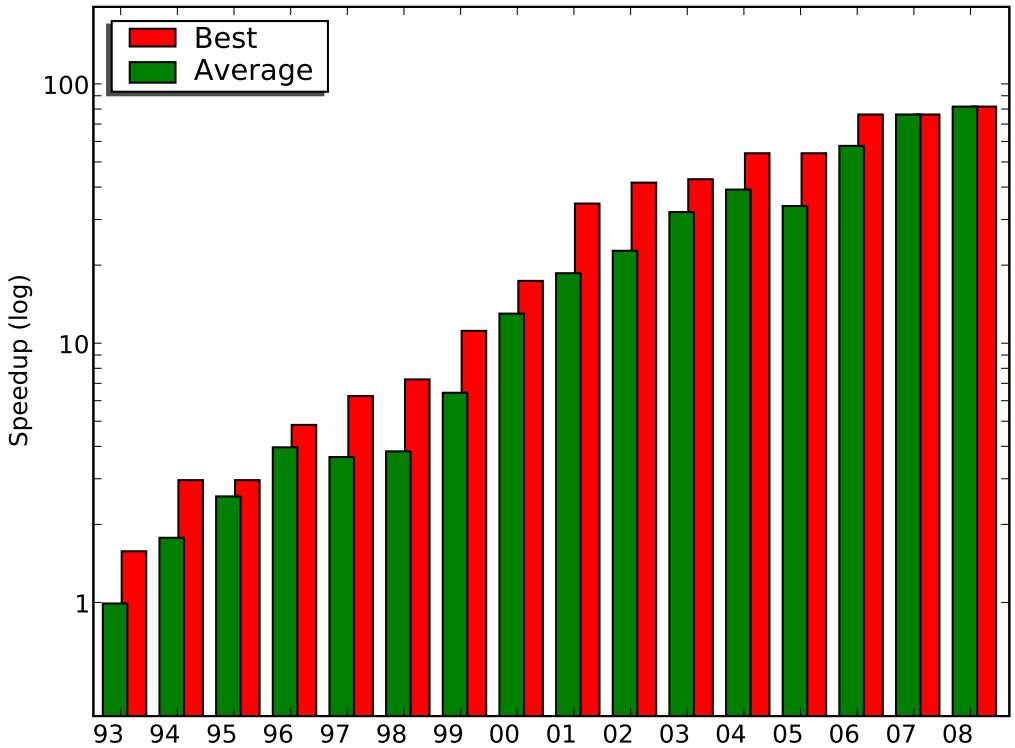
multi-core CPUs Cell BE processor GPUs

Frequency scaling is over, we now scale cores

We need to change the way we implement numerical codes now!

Example: FeatFlow 1993–2008

technische universität dortmund



- 80x speedup in 16 years for free
- But: More than 1000x improvement in peak processor performance
- And: Performance gain stagnates
- **Serial (legacy) codes no longer run faster automatically**

Outline

FEAST –

Hardware-oriented Numerics

Serial FEM: Data structures

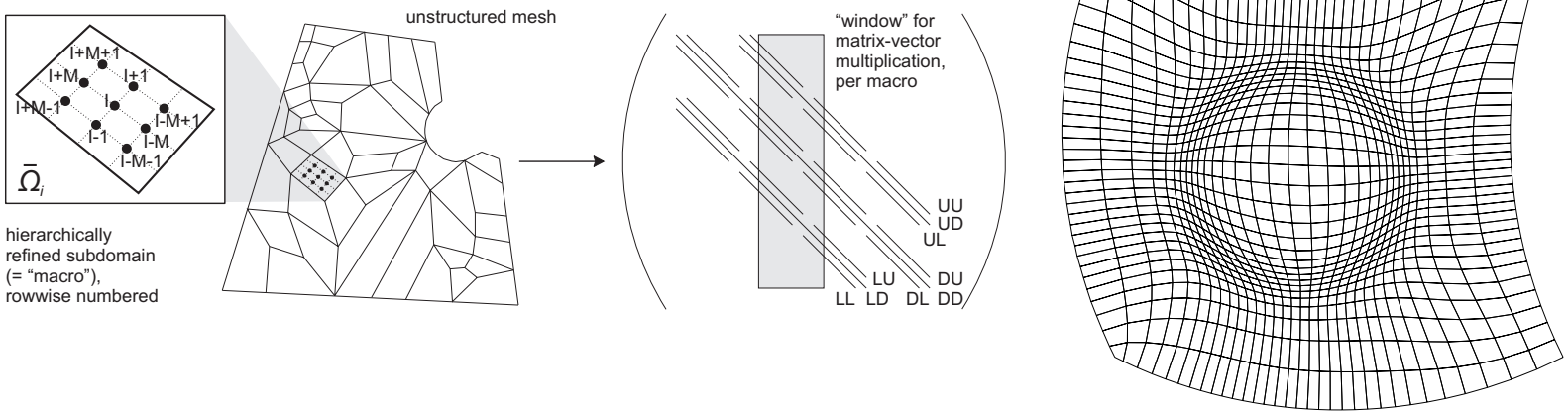
technische universität dortmund

Fully adaptive grids

Maximum flexibility
'Stochastic' numbering
Unstructured sparse matrices
Indirect addressing, very slow.

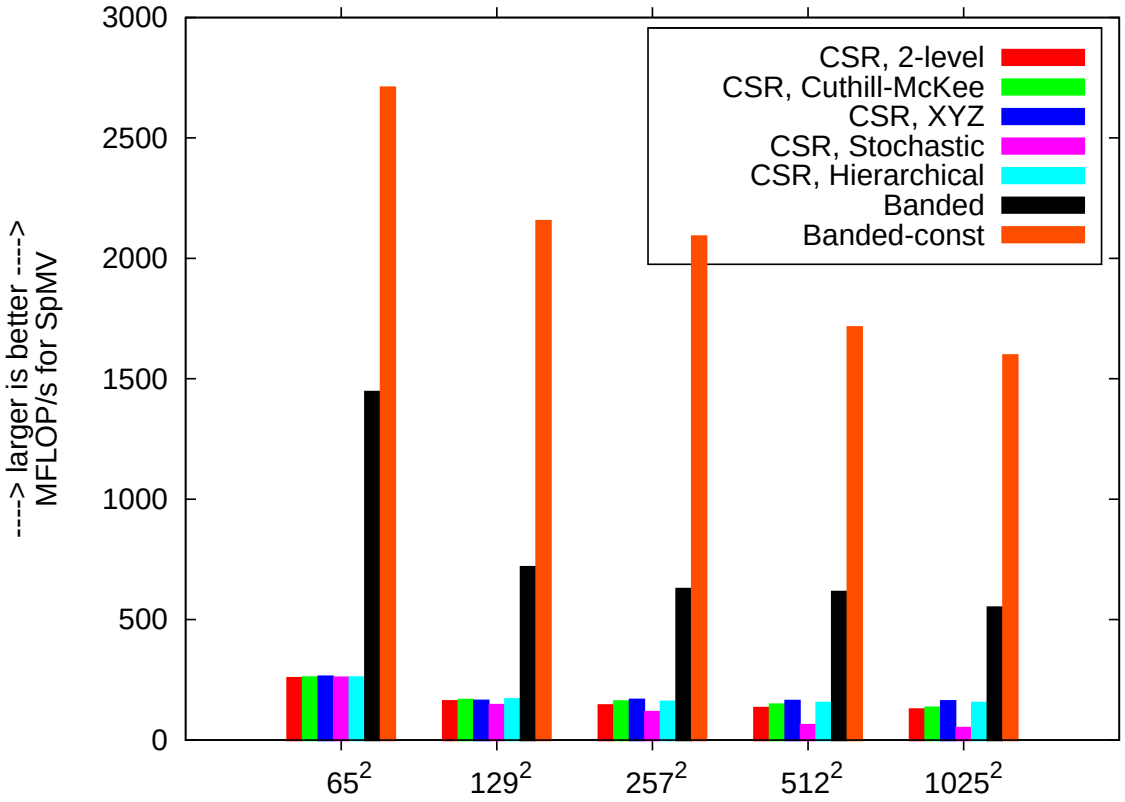
Structured grids

Logical tensorproduct structure
Fixed banded matrix structure
Direct addressing (high perf.)
Not limited to const. operators



Example: SpMV on TP grid

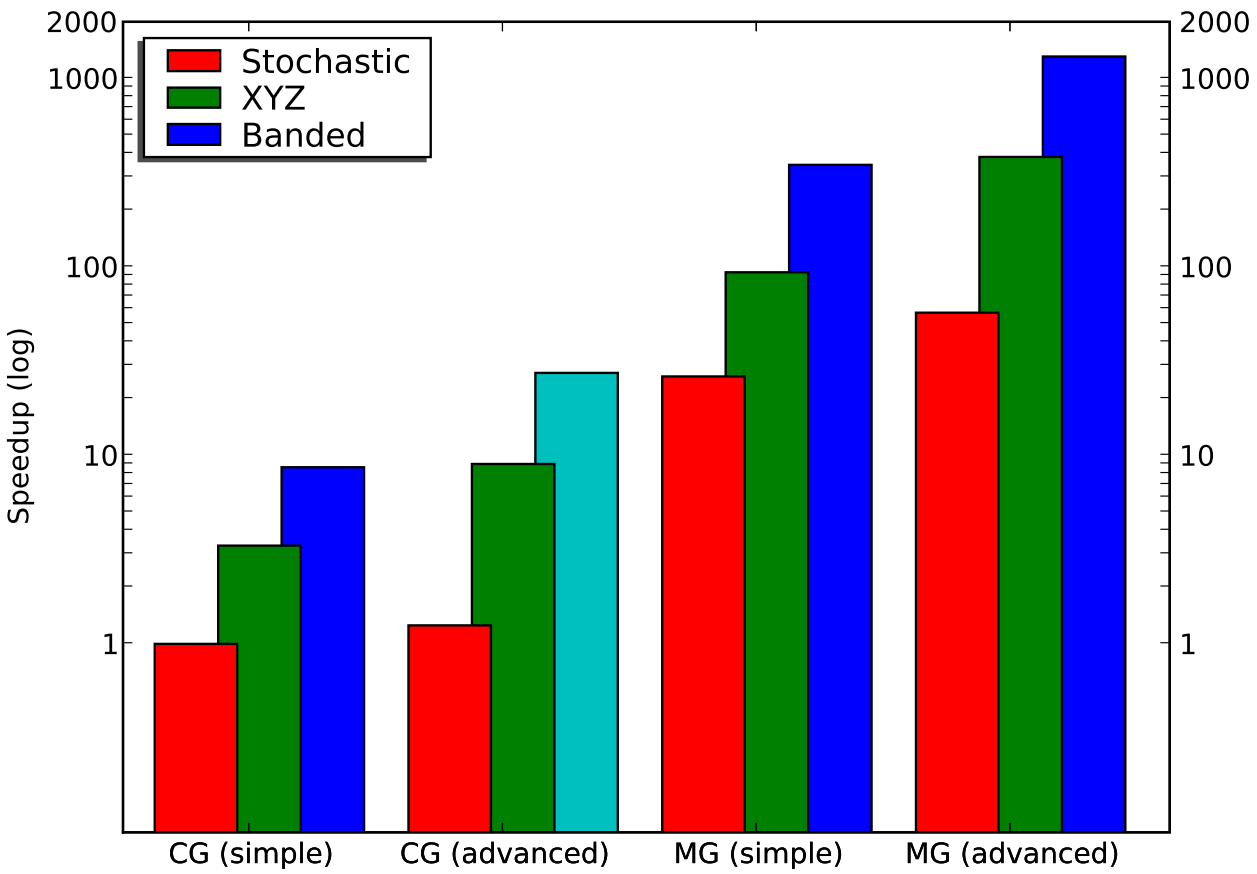
technische universität dortmund

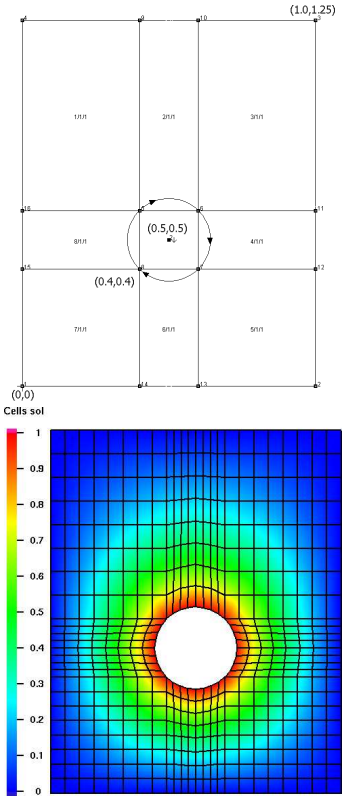


- Opteron 2214, 2.2 GHz, 2x1 MB L2 cache, one thread
- 50 vs. 550 MFLOP/s for interesting large problem size
- Cache-aware implementation $\Rightarrow$ 90% of memory throughput
- `const`: Stencil-based computation

Serial FEM: Solvers

technische universität dortmund





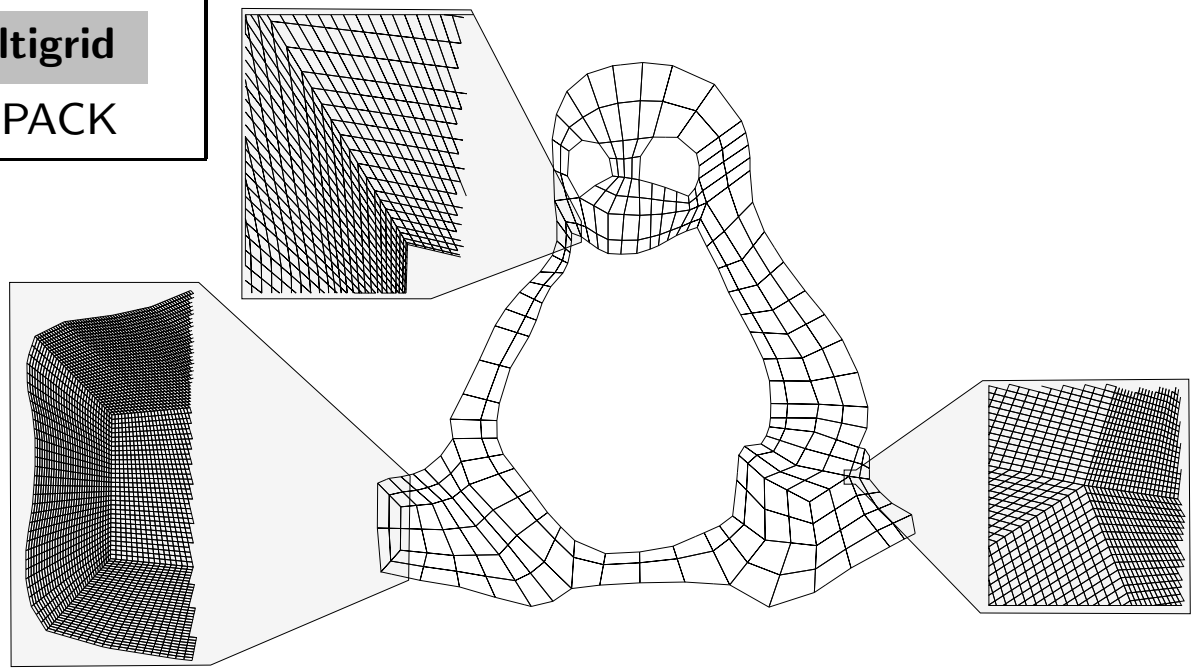
More than 1300x faster due to hardware-oriented numerics

ScaRC – Scalable Recursive Clustering

- Unstructured macro mesh of tensorproduct subdomains
- Minimal overlap by extended Dirichlet BCs
- Hybrid multilevel domain decomposition method
- Inspired by parallel MG ("best of both worlds")
  - Multiplicative vertically (between levels), global coarse grid problem (MG-like)
  - Additive horizontally: block-Jacobi / Schwarz smoother (DD-like)
- Hide local irregularities by MGs within the Schwarz smoother
- Embed in Krylov to alleviate Block-Jacobi character

Generic ScaRC solver template for scalar elliptic PDEs

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



Multivariate problems

technische universität dortmund

Block-structured systems

- Guiding idea: Tune scalar case once per architecture instead of over and over again per application
- Equation-wise ordering of the unknowns
- Block-wise treatment enables multivariate ScaRC solvers

Examples

- Linearised elasticity with compressible material
- Stokes
- Elasticity with (nearly) incompressible material
- Navier-Stokes with stabilisation

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

$\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ correspond to scalar elliptic operators

$\Rightarrow$ Tuned linear algebra (and tuned solvers)

Precision vs. accuracy

Mixed precision methods

Cool example

S.M. Rump, 1988:
Evaluating (with powers as multiplications)

$$(333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

for $x_0 = 77617$ and $y_0 = 33096$ gives

float s23e8	1.1726
double s52e11	1.17260394005318
long double s63e15	1.1726039400531786318

Not even the sign is correct:

Exact result $-0.8273\ldots$

Computational precision $\neq$ Result accuracy

Cool example

S.M. Rump, 1988:
Evaluating (with powers as multiplications)

$$(333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

for $x_0 = 77617$ and $y_0 = 33096$ gives

float s23e8	1.1726
double s52e11	1.17260394005318
long double s63e15	1.1726039400531786318

Not even the sign is correct:

Exact result $-0.8273\ldots$

Computational precision $\neq$ Result accuracy

Cool example

S.M. Rump, 1988:
Evaluating (with powers as multiplications)

$$(333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

for $x_0 = 77617$ and $y_0 = 33096$ gives

float s23e8	1.1726
double s52e11	1.17260394005318
long double s63e15	1.172603940053178631

Not even the sign is correct:

Exact result $-0.8273\ldots$

Computational precision $\neq$ Result accuracy

Cool example

S.M. Rump, 1988:
Evaluating (with powers as multiplications)

$$(333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

for $x_0 = 77617$ and $y_0 = 33096$ gives

float s23e8	1.1726
double s52e11	1.17260394005318
long double s63e15	1.172603940053178631

Not even the sign is correct:

Exact result $-0.8273\ldots$

Computational precision $\neq$ Result accuracy

Cool example

S.M. Rump, 1988:
Evaluating (with powers as multiplications)

$$(333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

for $x_0 = 77617$ and $y_0 = 33096$ gives

float s23e8	1.1726
double s52e11	1.17260394005318
long double s63e15	1.172603940053178631

Not even the sign is correct:

Exact result $-0.8273\ldots$

Computational precision $\neq$ Result accuracy

FEM example

technische universität dortmund

	single precision		double precision	
Level	Error	Reduction	Error	Reduction
2	2.391E-3		2.391E-3	
3	5.950E-4	4.02	5.950E-4	4.02
4	1.493E-4	3.98	1.493E-4	3.99
5	3.750E-5	3.98	3.728E-5	4.00
6	1.021E-5	3.67	9.304E-6	4.01
7	6.691E-6	1.53	2.323E-6	4.01
8	2.012E-5	0.33	5.801E-7	4.00
9	7.904E-5	0.25	1.449E-7	4.00
10	3.593E-4	0.22	3.626E-8	4.00

- Poisson $-\Delta \mathbf{u} = \mathbf{f}$ on $[0,1]^2$ with Dirichlet BCs, MG solver
- Bilinear conforming Finite Elements (Q_1) on cartesian mesh
- L_2 error against analytical reference solution
- Residuals indicate convergence, but results are completely off

Mixed precision motivation

technische universität dortmund

Bandwidth bound algorithms

- 64 bit = 1 double = 2 floats
- More variables per bandwidth (comp. intensity up)
- More variables per storage (data block size up)
- Applies to all memory levels:
  disc $\Rightarrow$ main $\Rightarrow$ device $\Rightarrow$ cache $\Rightarrow$ register

Compute bound algorithms

- 1 double multiplier $\approx$ 4 float multipliers (quadratic)
- 1 double adder $\approx$ 2 float adders (linear)
- Multipliers are much bigger than adders
  $\Rightarrow$ Quadrupled computational efficiency

Mixed precision schemes

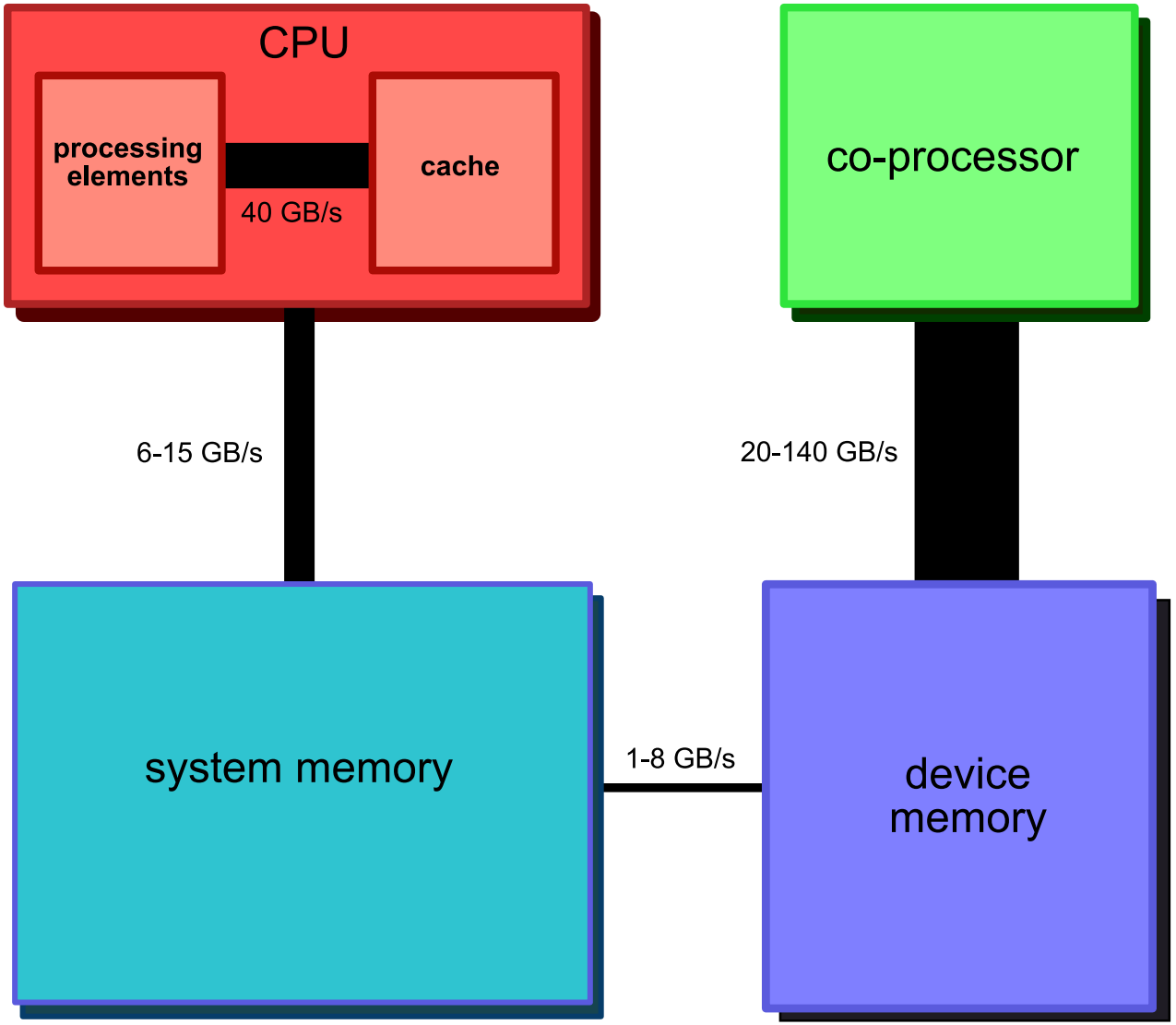
Mixed precision iterative refinement to solve $Ax = b$

Compute	$\mathbf{d}$	$=$	$\mathbf{b} - A\mathbf{x}$	in high precision
Solve	$A\mathbf{c}$	$=$	$\mathbf{d}$	approximately in low precision
Update	$\mathbf{x}$	$=$	$\mathbf{x} + \mathbf{c}$	in high precision and iterate

- Low precision solution is used as preconditioner in a high precision iterative method
- A is small and dense: Compute and apply LU factorisation in low precision
- A is large and sparse: **Approximately** solve $A\mathbf{c} = \mathbf{d}$ with an iterative method itself

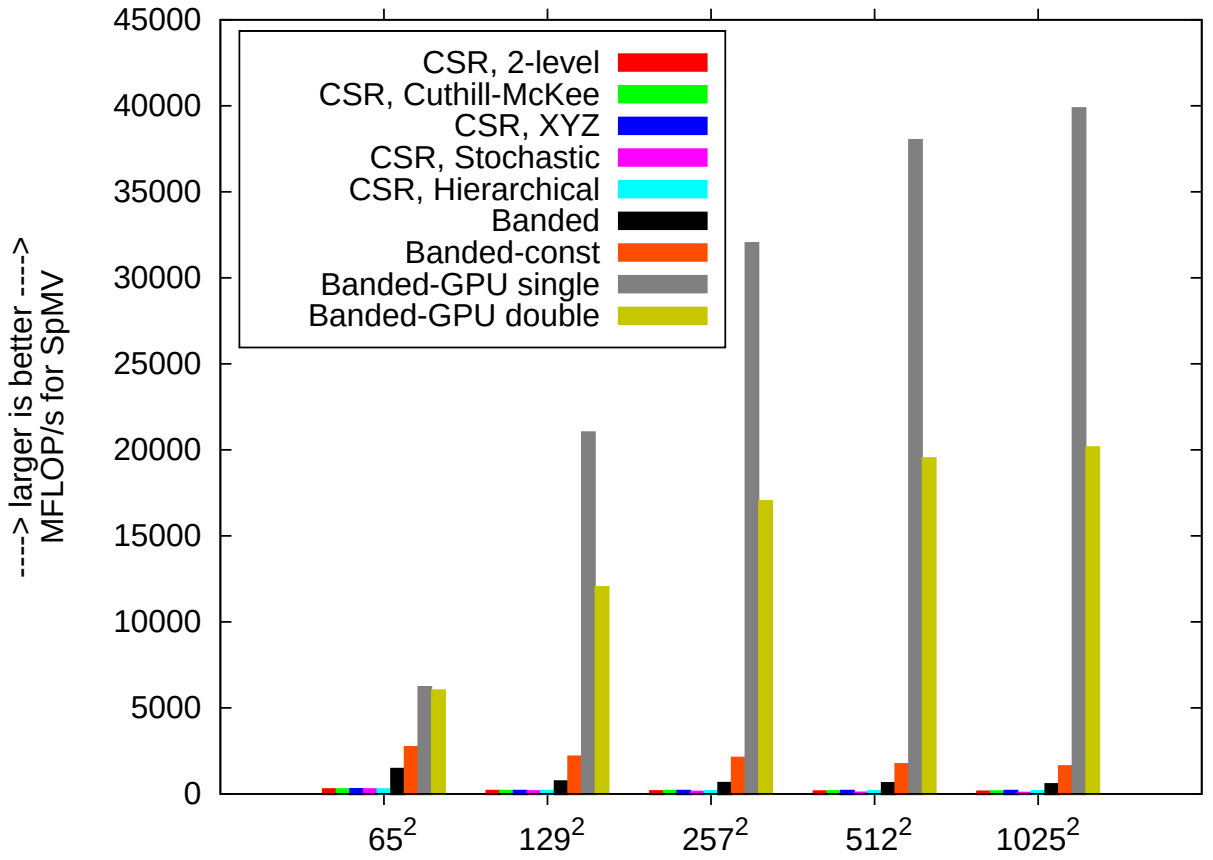
Co-processor integration into FEAST

Bandwidth in a CPU/GPU node



Example: SpMV on TP grid

technische universität dortmund



40 GFLOP/s, 140 GB/s with CUDA on GeForce GTX 280
Note: only 13 GFLOP/s on 8800 GTX (remainder of this talk)

Example: Multigrid on TP grid

technische universität dortmund

	Core2Duo (double)		GTX 280 (mixed)		
Level	time(s)	MFLOP/s	time(s)	MFLOP/s	speedup
7	0.021	1405	0.009	2788	2.3x
8	0.094	1114	0.012	8086	7.8x
9	0.453	886	0.026	15179	17.4x
10	1.962	805	0.073	21406	26.9x

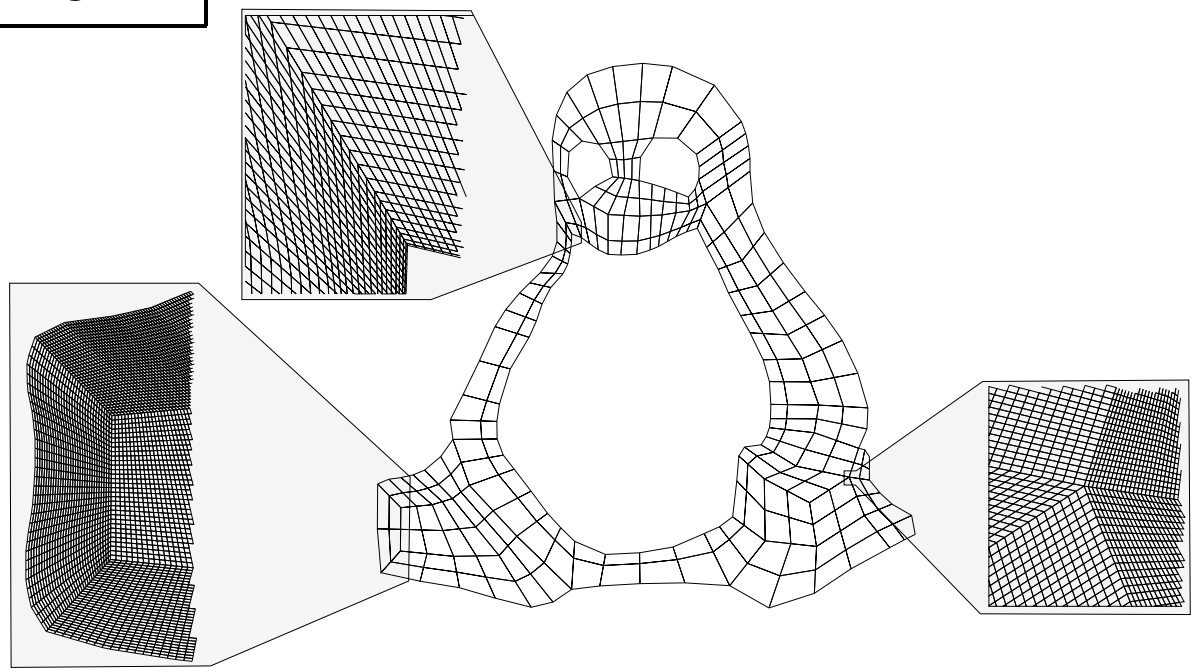
- Poisson on unitsquare, Dirichlet BCs, *not only a matrix stencil*
- 1M DOF, multigrid, FE-accurate in less than 0.1 seconds!
- 27x faster than CPU
- 1.7x faster than pure double on GPU
- 8800 GTX (double correction on CPU): 0.44 seconds on level 10

Co-processor integration

technische universität dortmund

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

All outer work: CPU, double
local MGs: Co-processor, single
$\Rightarrow$ Co-processor is preconditioner!
Ratio of inner work?



Minimally invasive integration

technische universität dortmund

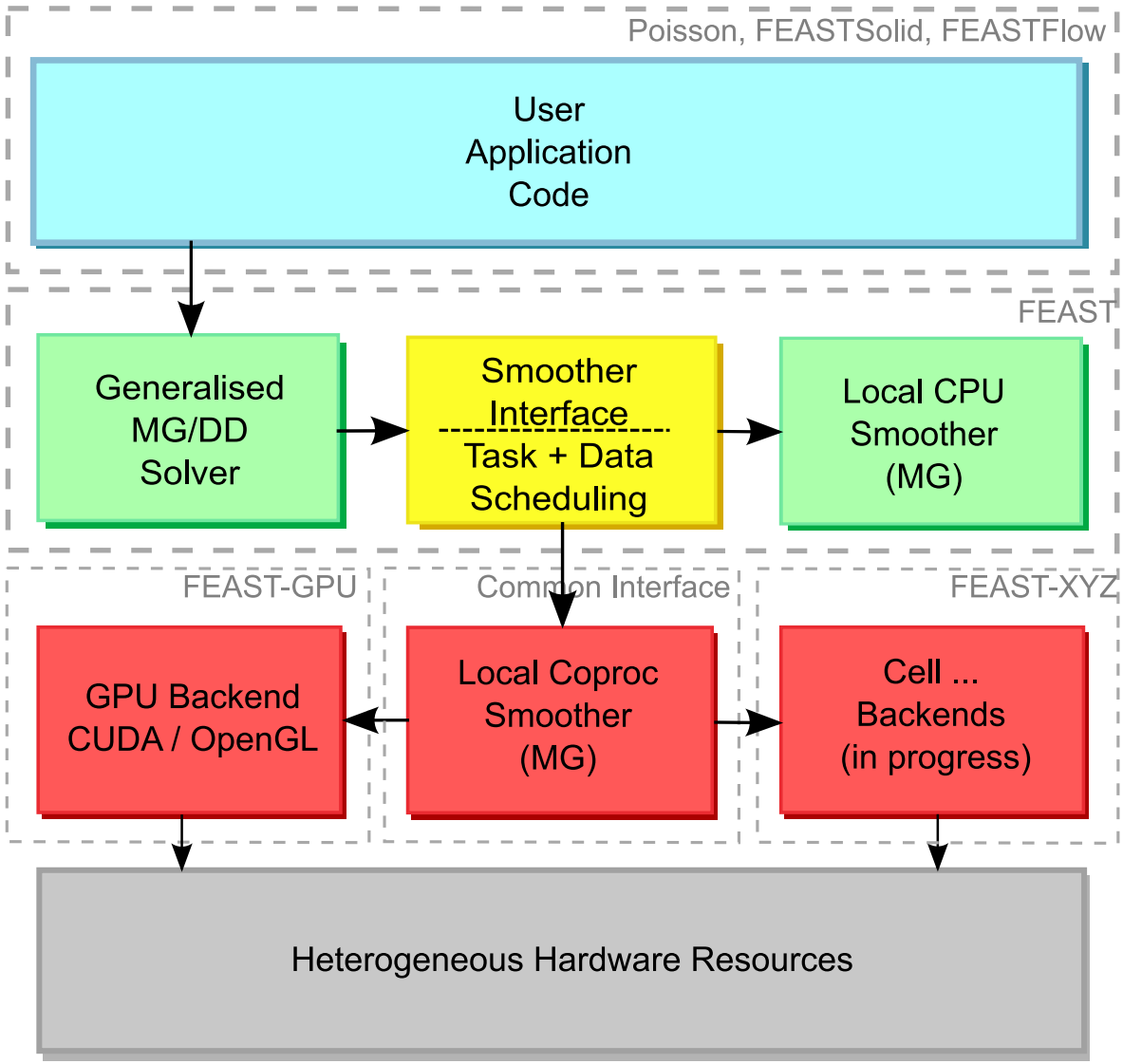
General approach

- Balance acceleration potential and integration effort
- Diverge code paths as late as possible
- No changes to application code!

Challenges

- Heterogeneous task assignment to maximise throughput
- Limited device memory (modeled as huge L3 cache)
- Building dense accelerated clusters

Integration summary



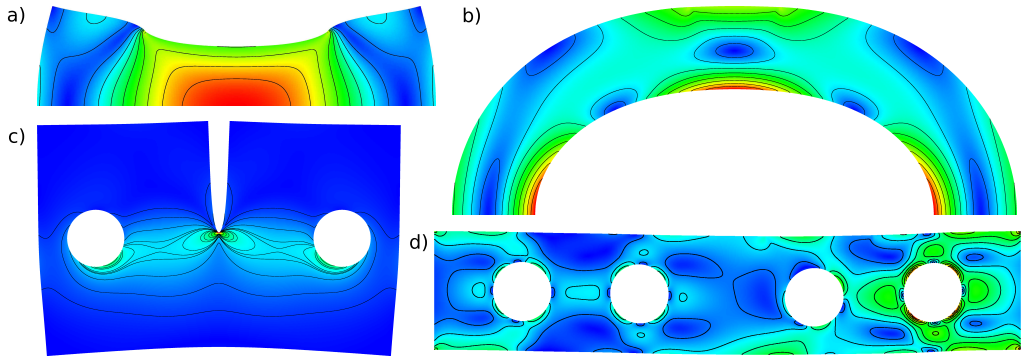
Some results

Linearised elasticity

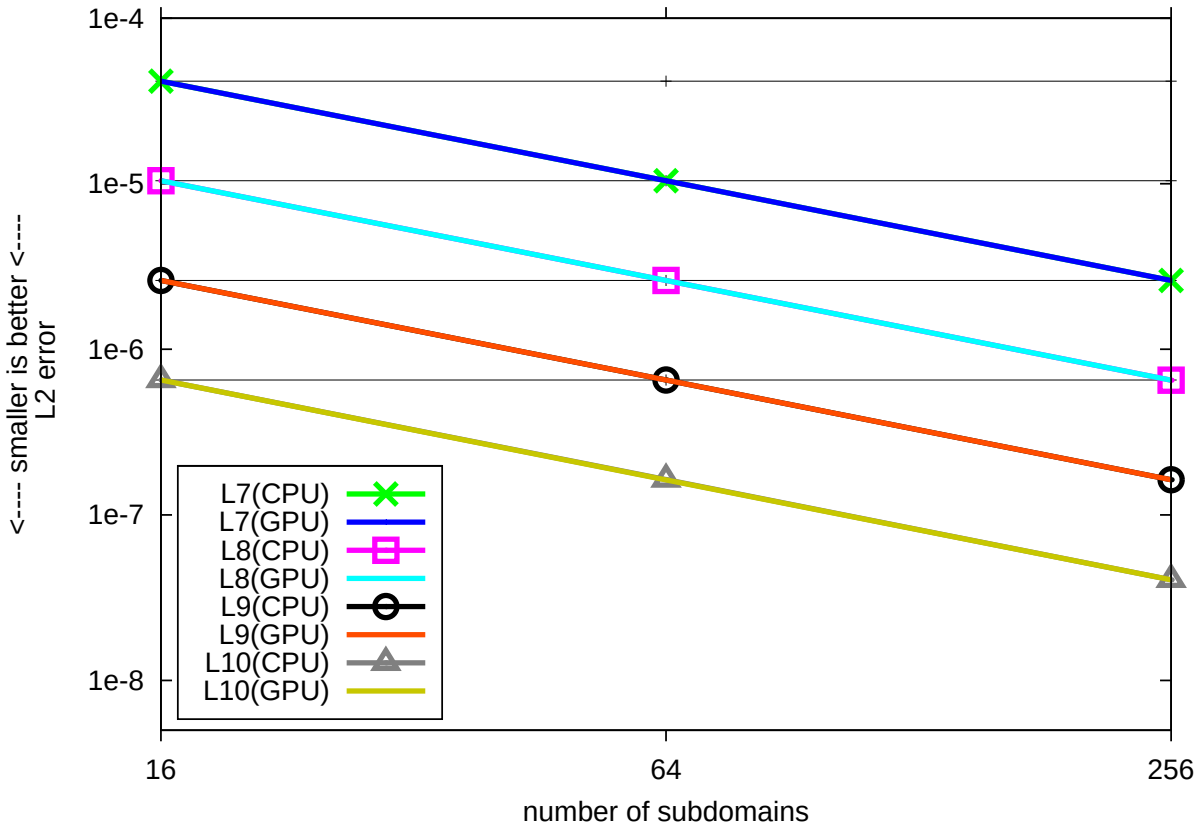
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \mathbf{f}$$

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

global multivariate BiCGStab
block-preconditioned by
Global multivariate multigrid (V 1+1)
additively smoothed (block GS) by
for all Ω_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 Ω_i: solve $\mathbf{A}_{22}\mathbf{c}_2 = \mathbf{d}_2$ by
local scalar multigrid
coarse grid solver: UMFPACK

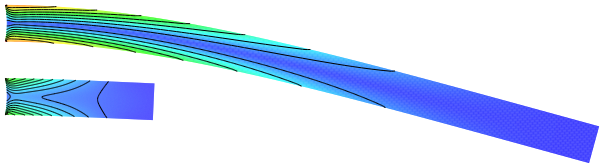


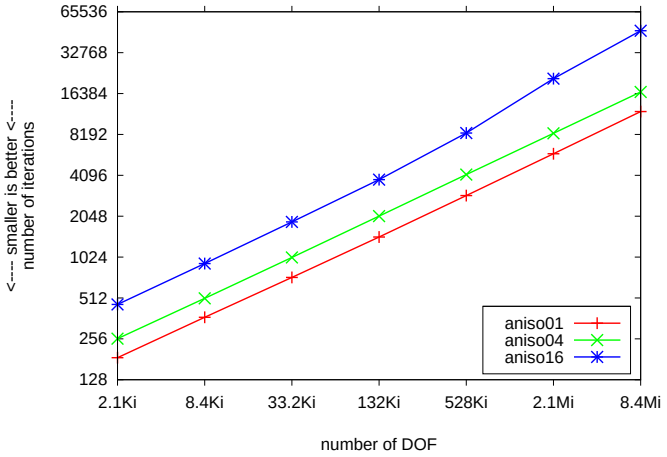
Accuracy (I)



- **Same results for CPU and GPU**
- L_2 error against analytically prescribed displacements
- Tests on 32 nodes, 512 M DOF

Accuracy (II)

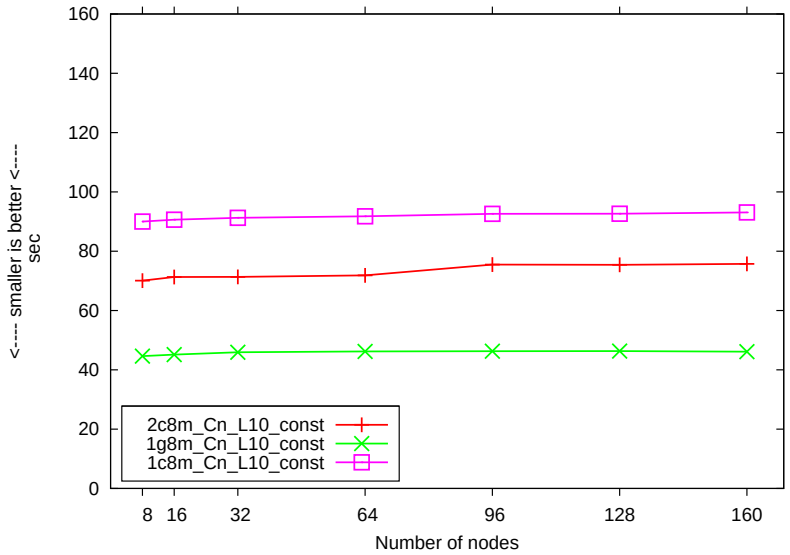


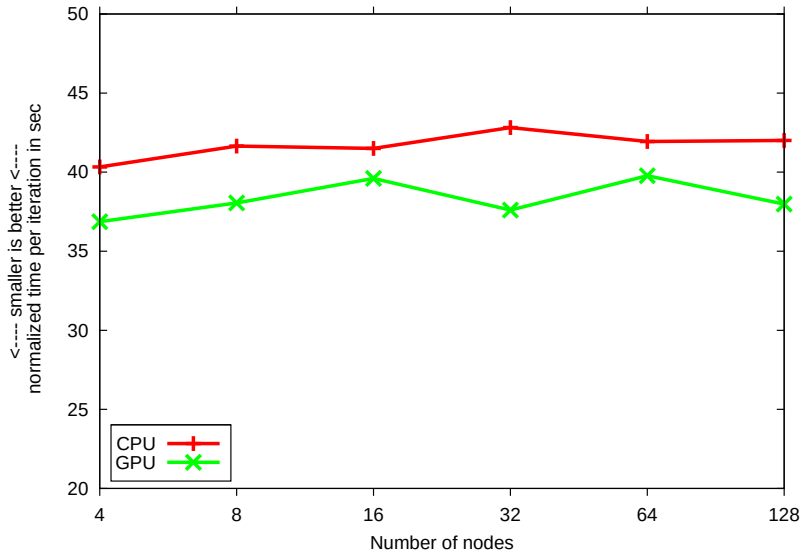


Cantilever beam, aniso 1:1, 1:4, 1:16
Hard, ill-conditioned CSM test
CG solver: no doubling of iterations
GPU-ScaRC solver: same results as CPU

aniso04	Iterations		Volume		y-Displacement	
refinement L	CPU	GPU	CPU	GPU	CPU	GPU
8	4	4	1.6087641E-3	1.6087641E-3	-2.8083499E-3	-2.8083499E-3
9	4	4	1.6087641E-3	1.6087641E-3	-2.8083628E-3	-2.8083628E-3
10	4.5	4.5	1.6087641E-3	1.6087641E-3	-2.8083667E-3	-2.8083667E-3
aniso16						
8	6	6	6.7176398E-3	6.7176398E-3	-6.6216232E-2	-6.6216232E-2
9	**6**	**5.5**	6.7176427E-3	6.7176427E-3	-6.6216551E-2	-6.6216552E-2
10	5.5	5.5	6.7176516E-3	6.7176516E-3	-6.6217501E-2	-6.6217502E-2

Scalability

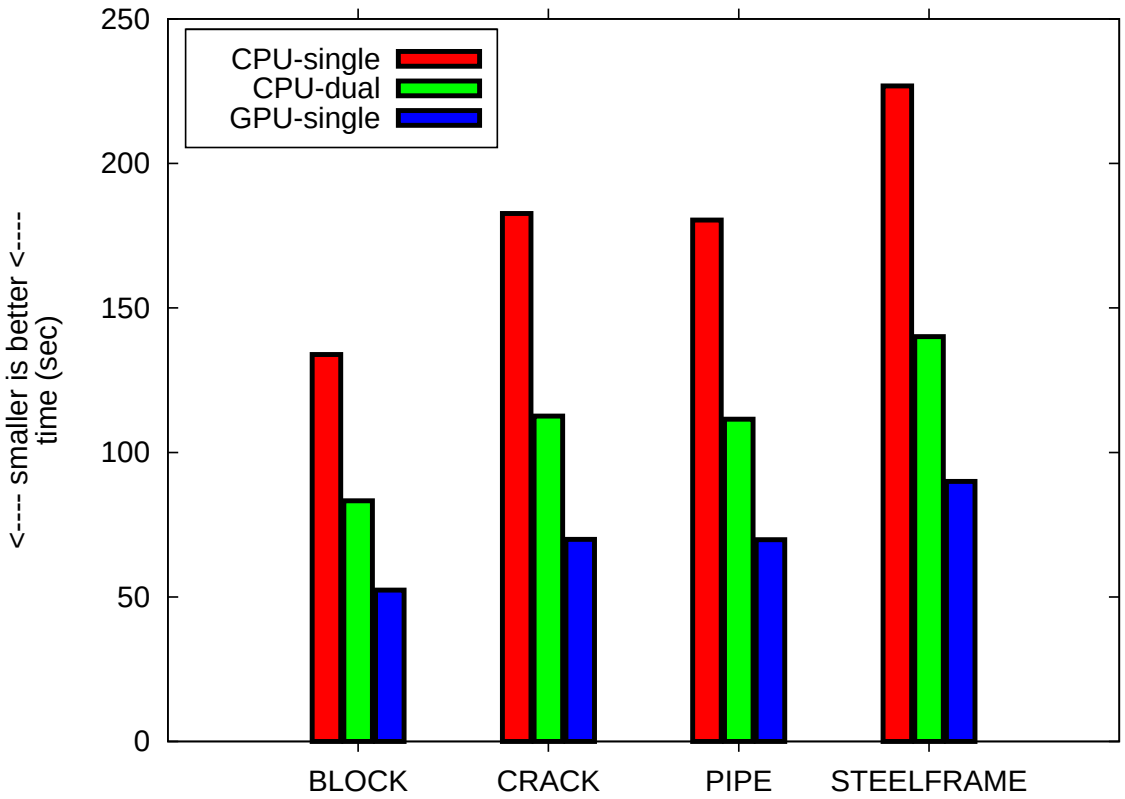




- Old cluster, dual Xeon EM64T, one Quadro 1400 per node
- Poisson problem (left): up to 1.3 B DOF, 160 nodes
- Elasticity (right): up to 1 B DOF, 128 nodes

Absolute speedup

technische universität dortmund



- 16 nodes, Opteron 2214 X2, Quadro 5600, OpenGL
- Problem size 128 M DOF
- Dualcore 1.6x faster than singlecore
- GPU 2.6x faster than singlecore, 1.6x than dual

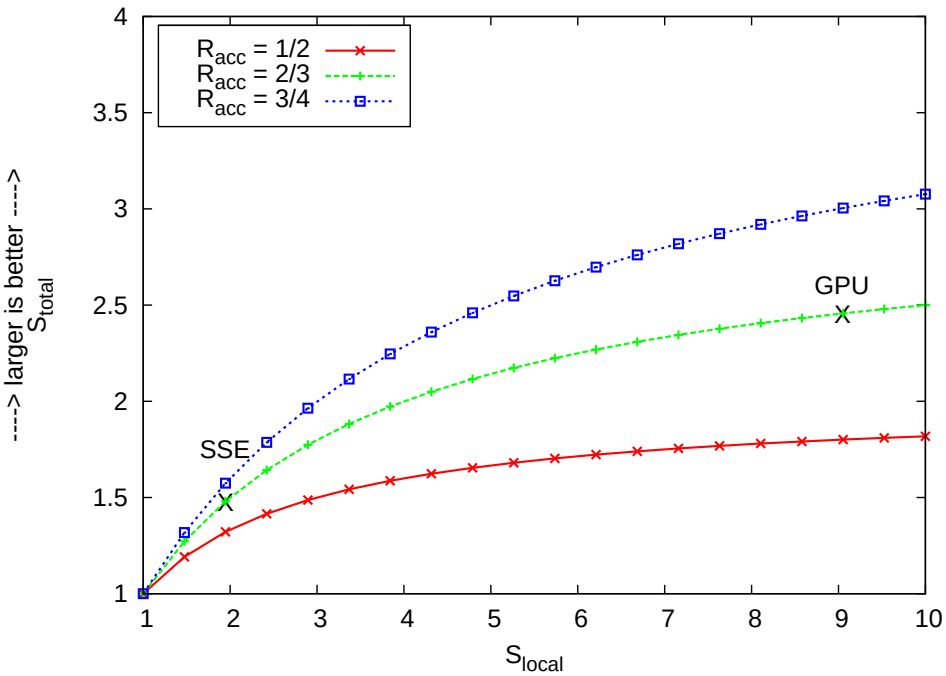
Acceleration analysis

technische universität dortmund

Speedup analysis

- Addition of GPUs increases resources
- $\Rightarrow$ Correct model: strong scalability inside each node
- Accelerable fraction of the elasticity solver: 2/3
- Remaining time spent in MPI and the outer solver

Local speedup: 9x
Global speedup: 2.6x
Theoretical limit: 3x



Stokes

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{B}_1 \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^T & \mathbf{B}_2^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{p} \end{pmatrix} = \mathbf{f}$$

- 18.8 M DOF
- 3-node OpteronX2 2214 cluster
- one 8800 GTX per node, CUDA

Acceleration potential:	75%
Local speedup:	11.5x
Global speedup:	3.8x
Theoretical limit:	4x

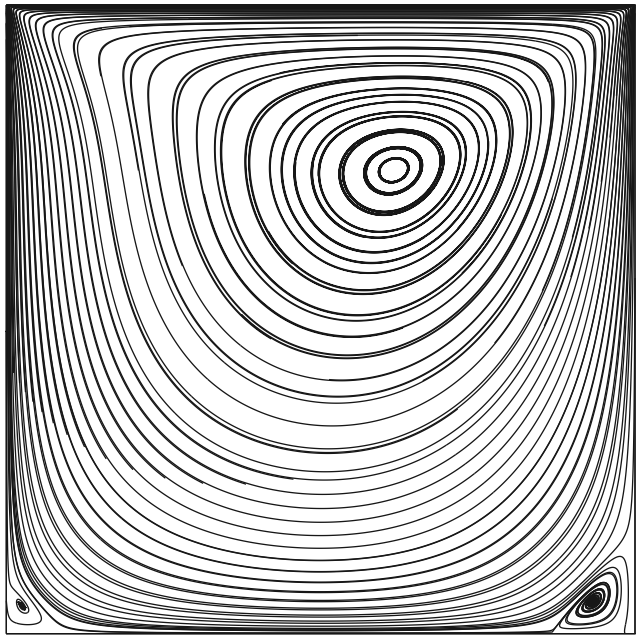
global multivariate BiCGStab
block-preconditioned (gain one digit) by :
velocity:
solve for $\mathbf{v}_1$ ($\mathbf{A}_{11}\mathbf{c}_1 = \mathbf{d}_1$) by
global scalar multigrid (V 1+1),
add. smoothed by
for all Ω_i: **local scalar multigrid**
coarse grid solver: UMFPACK
solve for $\mathbf{v}_2$ ($\mathbf{A}_{22}\mathbf{c}_2 = \mathbf{d}_2$) by
global scalar multigrid (V 1+1),
add. smoothed by
for all Ω_i: **local scalar multigrid**
coarse grid solver: UMFPACK
pressure: Schur complement
$\mathbf{c}_3 = \mathbf{M}_p^{-1}(\mathbf{d}_3 + \mathbf{B}_1^T\mathbf{c}_1 + \mathbf{B}_2^T\mathbf{c}_2)$

Navier-Stokes

technische universität dortmund

Driven cavity simulation

- Full assembly in each iteration
- Linear solver same as for Stokes
- W/out GPUs (% of total time):
  - 23% assembly
  - 77% solver
  - 70% linear solver
  - 54% local solvers
- With GPUs (% of total time):
  - 45% assembly
  - 55% solver
  - 37% linear solver
  - 9% local solvers
- Local speedup: 12.1x
- Total speedup: 2.1x (theoretical maximum: 2.17x)



Conclusions

Conclusions

- Hardware-oriented numerics to prevent existing codes being worthless in a few years
- Mixed precision schemes
- GPUs as local preconditioners in large-scale parallel FEM packages
- Not limited to GPUs, applicable to all kinds of hardware accelerators
- Minimally invasive approach, no changes to application code
- Excellent local acceleration
- Global acceleration limited by 'sequential' part (Amdahl's law, strong scaling)

Acknowledgements

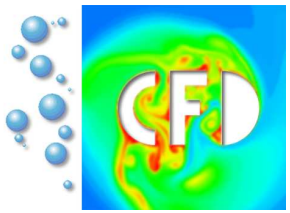
technische universität dortmund

Collaborative work with:

Hilmar Wobker, Sven Buijssen, Stefan Turek and the FEAST group (Dortmund)

Robert Strzodka (Max Planck Institut Informatik)

Jamaludin Mohd-Yusof, Patrick McCormick (Los Alamos National Laboratory)



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

Supported by Deutsche Forschungsgemeinschaft, project TU 102/22-1