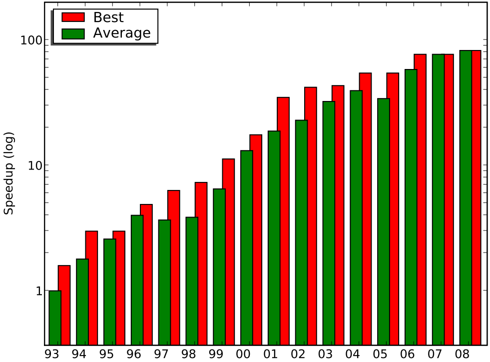
GPU Cluster Computing for Finite Element Applications

Dominik Göddeke, Hilmar Wobker,
Sven H.M. Buijssen and Stefan Turek

Applied Mathematics
TU Dortmund
dominik.goeddeke@math.tu-dortmund.de
http://www.mathematik.tu-dortmund.de/~goeddeke

38th SPEEDUP Workshop on High-Performance Computing
EPF Lausanne, Switzerland, September 7, 2009

The free ride is over



- FeatFlow benchmark 1993–2008 (single-threaded CFD code)
- 80x speedup in 16 years for free
- But: More than 1000x improvement in peak processor performance
- **Serial (legacy) codes no longer run faster automatically**

Outline

1. FEAST – hardware-oriented numerics
2. Precision and accuracy
3. Co-processor integration
4. Results
5. Conclusions

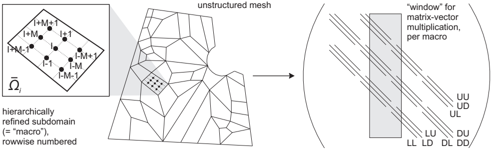
FEAST –

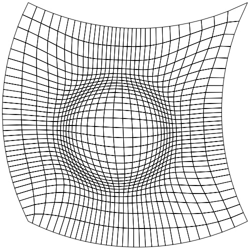
Hardware-oriented Numerics

Serial FEM: Data structures

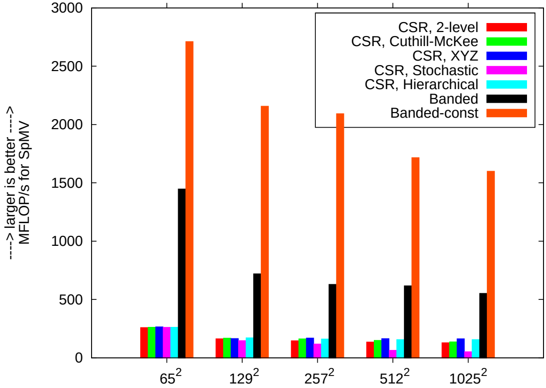
Fully adaptive grids
Maximum flexibility
'Stochastic' numbering
Unstructured sparse matrices
Indirect addressing, very slow.

Structured grids
Logical tensor product structure
Fixed banded matrix structure
Direct addressing (high perf.)
Not limited to const. operators



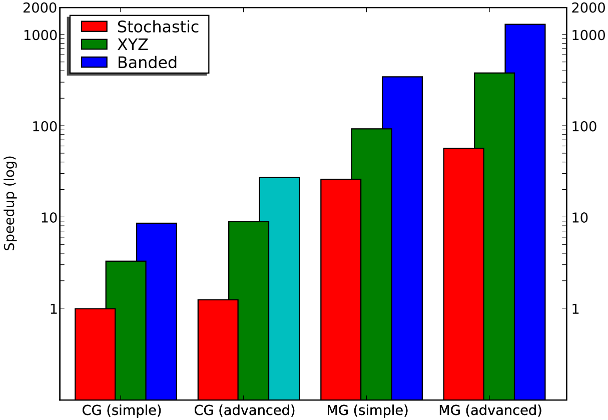


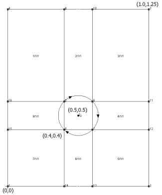
Example: SpMV on TP grid

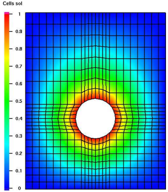


- Opteron 2214 dual-core, 2.2 GHz, 2x1 MB L2 cache, one thread
- 50 vs. 550 MFLOP/s for interesting large problem size
- Cache-aware implementation ⇒ 90% of memory throughput
- `const`: Stencil-based computation

Serial FEM: Solvers







More than 1300x faster due to hardware-oriented numerics

Parallel FEM: ScaRC

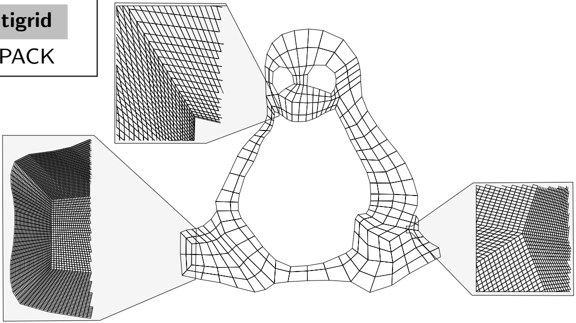
ScaRC – Scalable Recursive Clustering

- Unstructured macro mesh of tensor product 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

Parallel FEM: Solver template

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

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
- Saddle point problems: Stokes, elasticity with (nearly) incompressible material, Navier-Stokes with stabilisation

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

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

Precision vs. accuracy

Mixed precision methods

Cool example

S.M. Rump (1988), updated by Loh and Walster (2002) for IEEE-754 round-to-nearest: 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

single precision (s23e8)	1.172604
double precision (s52e11)	1.1726039400531786
quad precision (s112e15)	$1.1726039400531786318588349045201838$

Not even the sign is correct:

Exact result $-0.8273\ldots$

Computational precision $\neq$ Result accuracy

Cancellation promotes small round-off errors, impossible to avoid a priori

Cool example

S.M. Rump (1988), updated by Loh and Walster (2002) for IEEE-754 round-to-nearest: 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

single precision (s23e8)	1.172604
double precision (s52e11)	1.1726039400531786
quad precision (s112e15)	$1.1726039400531786318588349045201838$

Not even the sign is correct:

Exact result $-0.8273\ldots$

Computational precision $\neq$ Result accuracy

Cancellation promotes small round-off errors, impossible to avoid a priori

Cool example

S.M. Rump (1988), updated by Loh and Walster (2002) for IEEE-754 round-to-nearest: 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

single precision (s23e8)	1.172604
double precision (s52e11)	1.1726039400531786
quad precision (s112e15)	$1.1726039400531786318588349045201838$

Not even the sign is correct:

Exact result $-0.8273\ldots$

Computational precision $\neq$ Result accuracy

Cancellation promotes small round-off errors, impossible to avoid a priori

Cool example

S.M. Rump (1988), updated by Loh and Walster (2002) for IEEE-754 round-to-nearest: 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

single precision (s23e8)	1.172604
double precision (s52e11)	1.1726039400531786
quad precision (s112e15)	$1.1726039400531786318588349045201838$

Not even the sign is correct:

Exact result $-0.8273\dots$

Computational precision $\neq$ Result accuracy

Cancellation promotes small round-off errors, impossible to avoid a priori

Cool example

S.M. Rump (1988), updated by Loh and Walster (2002) for IEEE-754 round-to-nearest: 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

single precision (s23e8)	1.172604
double precision (s52e11)	1.1726039400531786
quad precision (s112e15)	$1.1726039400531786318588349045201838$

Not even the sign is correct:

Exact result $-0.8273\ldots$

Computational precision $\neq$ Result accuracy

Cancellation promotes small round-off errors, impossible to avoid a priori

FEM example

	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 quadrilateral elements (Q_1) on cartesian mesh
- L_2 error against analytical reference solution
- Residuals indicate convergence, but results are completely off

Mixed precision motivation

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

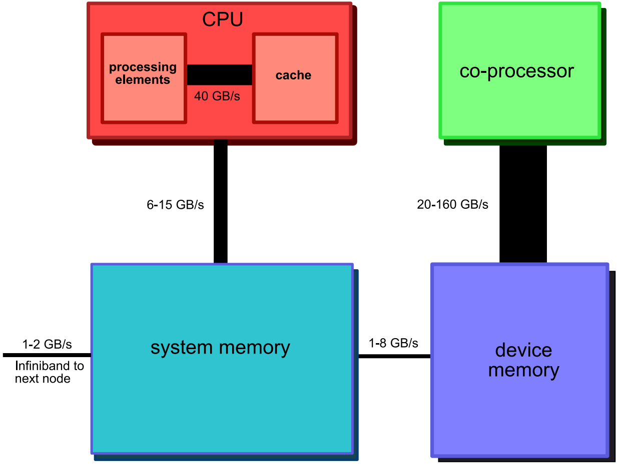
Mixed precision iterative refinement to solve $A\mathbf{x} = \mathbf{b}$

$$
\begin{array}{|rrcll|}
\hline
\text{Compute} & \mathbf{d} & = & \mathbf{b} - A\mathbf{x} & \text{in high precision} \\
\text{Solve} & A\mathbf{c} & = & \mathbf{d} & \text{approximately in low precision} \\
\text{Update} & \mathbf{x} & = & \mathbf{x} + \mathbf{c} & \text{in high precision and iterate} \\
\hline
\end{array}
$$

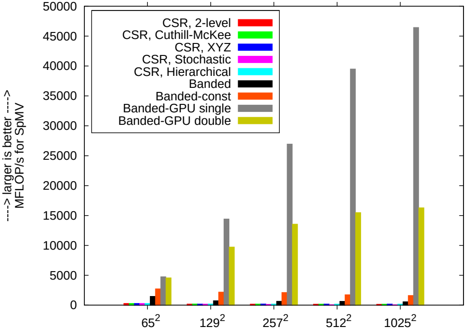
- Low precision solution is used as preconditioner in a high precision iterative method
- A is small and dense: Compute and apply LU factorisation in low precision
- A is large and sparse: **Approximately** solve $A\mathbf{c} = \mathbf{d}$ with an iterative method itself

Co-processor integration into FEAST

Bandwidth in a CPU/GPU node



Example: SpMV on TP grid



- Sufficiently tuned CUDA implementation of band-MV
- NVIDIA GeForce GTX 280
- 46.5 GFLOP/s (compare 1 GFLOP/s on Opteron 2214)
- 16.2 GFLOP/s vs. 550 MFLOP/s in double
- PlayStation 3: 3 GFLOP/s single precision

Example: Multigrid on TP grid

	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

Minimally invasive integration

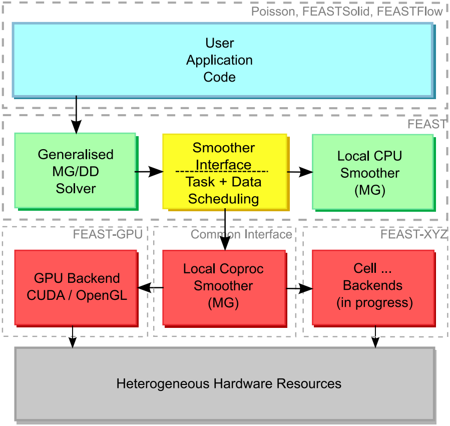
global BiCGStab
preconditioned by
global multilevel (V 1+1)
additively smoothed by
for all Ω_i: **local multigrid**
coarse grid solver: UMFPACK

All outer work: CPU, double

Local MGs: GPU, single

GPU performs smoothing or preconditioning

Not limited to GPUs



Minimally invasive integration

General approach

- Balance acceleration potential and integration effort
- Accelerate many different applications built on top of one central FE and solver toolkit
- Diverge code paths as late as possible
- No changes to application code!
- Retain all functionality
- Do not sacrifice accuracy

Challenges

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

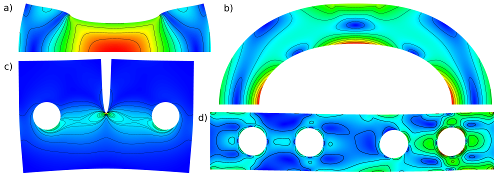
Some results

Linearised elasticity

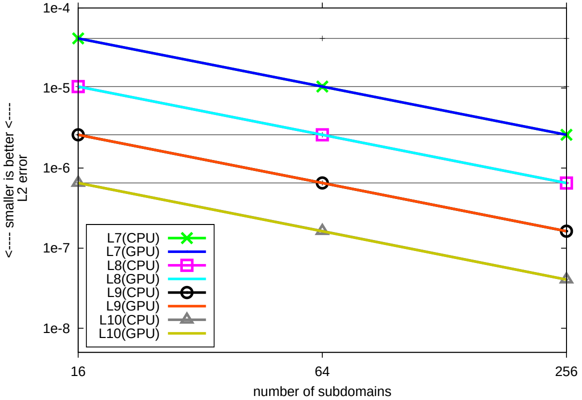
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \mathbf{f}$$

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

global multivariate BiCGStab
block-preconditioned by
Global multivariate multilevel (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

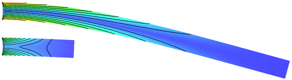


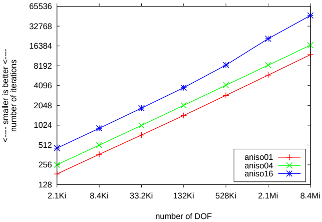
Accuracy (I)



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

Accuracy (II)

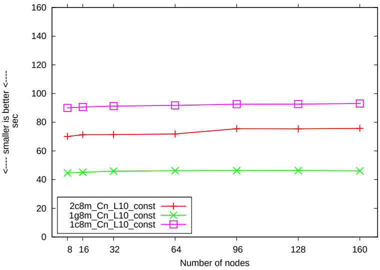


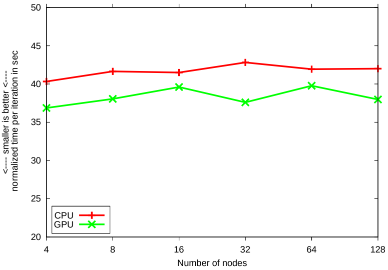


Cantilever beam, aniso 1:1, 1:4, 1:16
Hard, ill-conditioned CSM test
CG solver: no doubling of iterations
GPU-ScaRC solver: same results as CPU

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	-6.621655**1**E-2	-6.621655**2**E-2
10	5.5	5.5	6.7176516E-3	6.7176516E-3	-6.621750**1**E-2	-6.621750**2**E-2

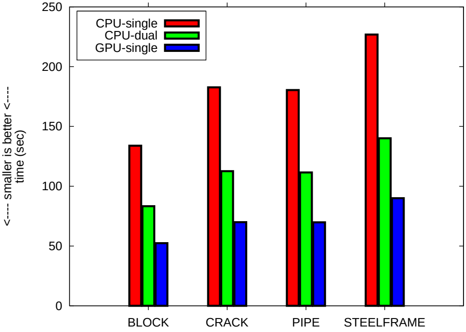
Weak scalability





- Outdated cluster, dual Xeon EM64T
- one NVIDIA Quadro FX 1400 per node (one generation behind the Xeons, 20 GB/s BW)
- Poisson problem (left): up to 1.3 B DOF, 160 nodes
- Elasticity (right): up to 1 B DOF, 128 nodes

Absolute performance



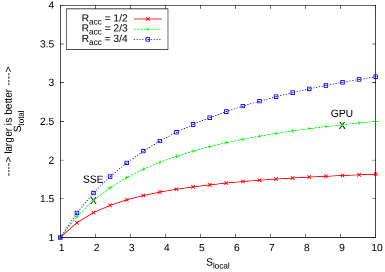
- 16 nodes, Opteron 2214 dualcore
- NVIDIA Quadro FX 5600 (76 GB/s BW), OpenGL
- Problem size 128 M DOF
- Dualcore 1.6x faster than singlecore
- GPU 2.6x faster than singlecore, 1.6x than dual

Acceleration analysis

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

Accelerable fraction R_{acc}:	66%
Local speedup S_{local}:	9x
Total speedup S_{total}:	2.6x
Theoretical limit S_{max}:	3x

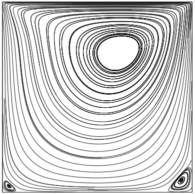


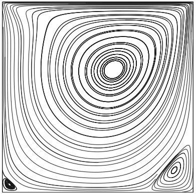
Stokes and Navier-Stokes

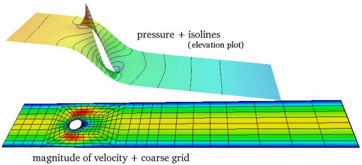
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{B}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^\mathsf{T} & \mathbf{B}_2^\mathsf{T} & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1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

- 4-node cluster
- Opteron 2214 dualcore
- GeForce 8800 GTX (86 GB/s BW), CUDA
- Driven cavity and channel flow around a cylinder

fixed point iteration
solving linearised subproblems 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 $\mathbf{c}_3 = (\mathbf{M}_p^\mathsf{L})^{-1}\mathbf{d}_3$







Stokes results

Setup

- Driven Cavity problem
- Remove convection part $\Rightarrow$ linear problem
- Measure runtime fractions of linear solver

Accelerable fraction R_{acc}:	75%
Local speedup S_{local}:	11.5x
Total speedup S_{total}:	3.8x
Theoretical limit S_{max}:	4x

Navier-Stokes results

Speedup analysis

	R_{acc}		S_{local}		S_{total}	
	L9	L10	L9	L10	L9	L10
DC Re100	41%	46%	6x	12x	1.4x	1.8x
DC Re250	56%	58%	5.5x	11.5x	1.9x	2.1x
Channel flow	60%	–	6x	–	1.9x	–

Important consequence: Ratio between assembly and linear solve changes significantly

DC Re100		DC Re250		Channel flow	
plain	accel.	plain	accel.	plain	accel.
29:71	50:48	11:89	25:75	13:87	26:74

Conclusions

Conclusions

- Hardware-oriented numerics prevents existing codes being worthless in a few years
- Mixed precision schemes exploit the available bandwidth without sacrificing accuracy
- GPUs as local preconditioners in a large-scale parallel FEM package
- 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
- Future work: Design solver schemes with higher acceleration potential without sacrificing numerical efficiency

Acknowledgements

Collaborative work with

FEAST group (TU Dortmund)

Robert Strzodka (Max Planck Institut Informatik)

Jamaludin Mohd-Yusof, Patrick McCormick (Los Alamos)

Supported by

DFG, projects TU 102/22-1, 22-2, 27-1, 11-3

BMBF, *HPC Software für skalierbare Parallelrechner*: SKALB project (01IH08003D / SKALB)