Case study: GPU acceleration of parallel multigrid solvers

Dominik Göddeke

ARCS 2008 - Architecture of Computing Systems
GPGPU and CUDA Tutorials
Dresden, Germany, February 25 2008

Acknowledgements

- **Hilmar Wobker, Stefan Turek and the FEAST group**
  TU Dortmund

- **Robert Strzodka, Max Planck Institut Informatik**
  Saarbrücken

- **Patrick McCormick, Jamaludin Mohd-Yusof**
  Los Alamos National Labs

Outline

- Introduction
- **Starting point: FEAST - Finite Element Analysis and Solution Tools**
- **Co-processor integration**
- **Exemplary results**
- **Summary and conclusions**

Motivation

- **Co-processor hardware offers tremendous potential**
  - Both raw compute and bandwidth
  - Example: observed 345 GFLOP/s, 80 GB/s on NVIDIA's G80 GPU
  - Better than CPUs in Performance / Dollar and Performance / Watt

- **Examples**
  - GPUs
  - Cell Blades and Cell Accelerator Boards
  - ClearSpeed Adcance Accelerator Boards
  - ...

- **But:**
  - Different languages, different APIs, different compilers
  - No --march=gpu,cell,cpu in gcc ☺

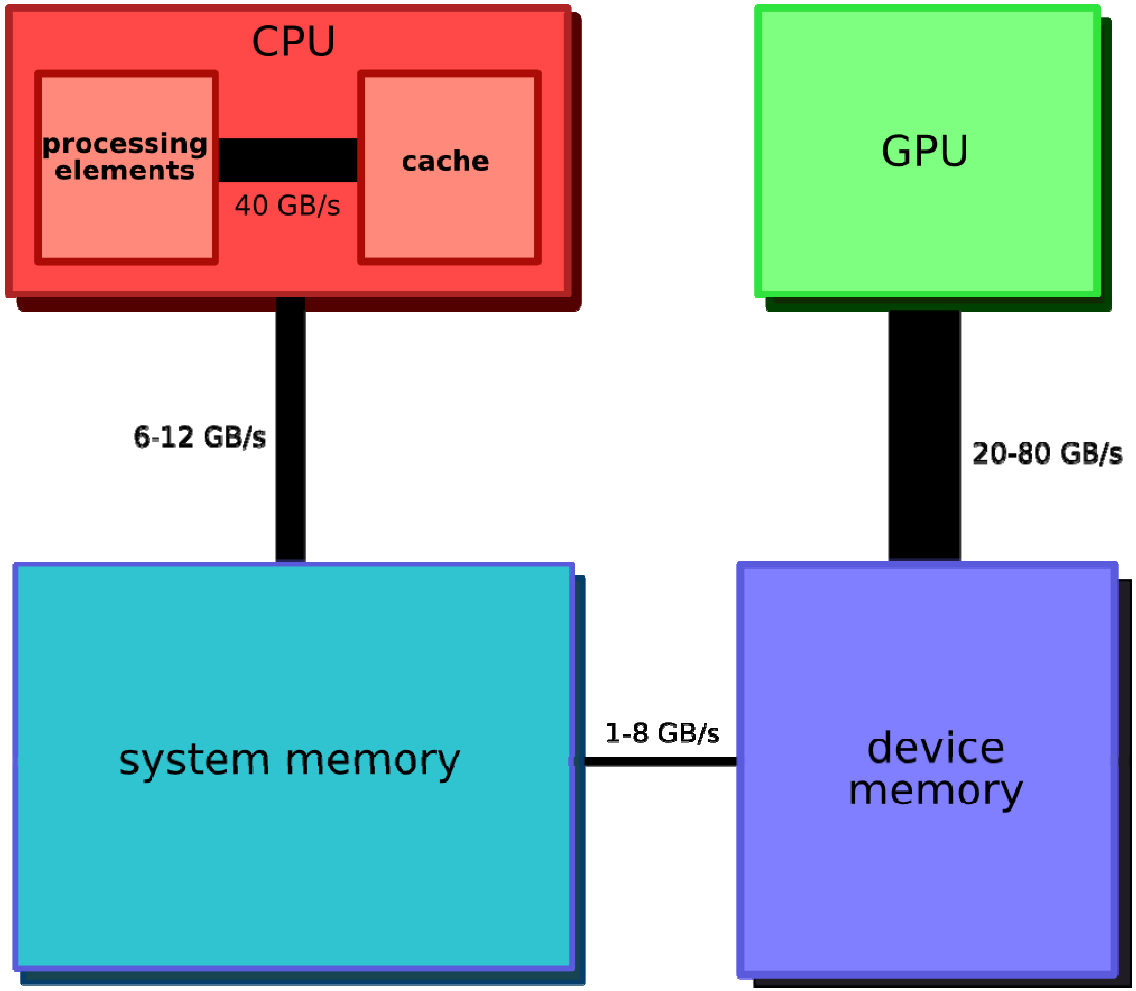
Motivation in case of FE software

- **Performance improvements**
  - Can only be achieved by ‘hardware oriented numerics’
  - Combine state-of-the-art numerical methodology with hardware-aware implementations
- **Paradigm change**
  - Frequency scaling is over, we now scale cores
  - No more automatic speedups for serial code
- **Numerical and algorithmic foundation research must go hand in hand**
  - To find a good balance between numerical and computational efficiency
  - Optimal technique for one architecture might not be optimal for a different one

Motivation (cont.)

- **Challenge of co-processor integration**
  - Significant reimplementations are prohibitive
  - In particular for large, established codes
- **Balance needed**
  - Actual hardware is evolving too rapidly
  - Integration should be reasonably future-proof
  - For several co-processor(s) and generations
- **Our approach: High level of abstraction**
  - Identify and isolate "accelerable" parts of a computation
  - Chunks must be large enough to amortise co-processor drawbacks
  - Encapsulate several co-processors under one interface

Example: Bandwidth in a CPU-GPU system

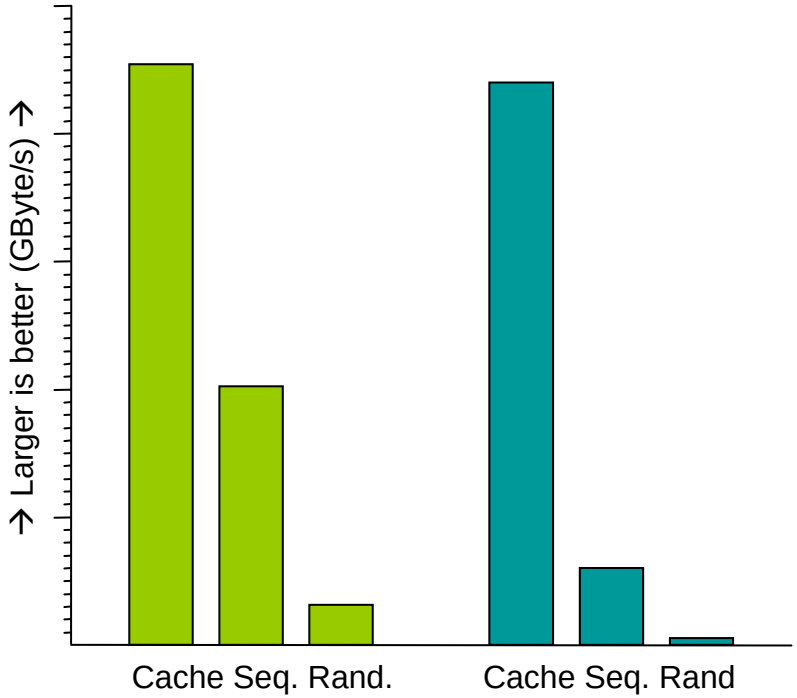


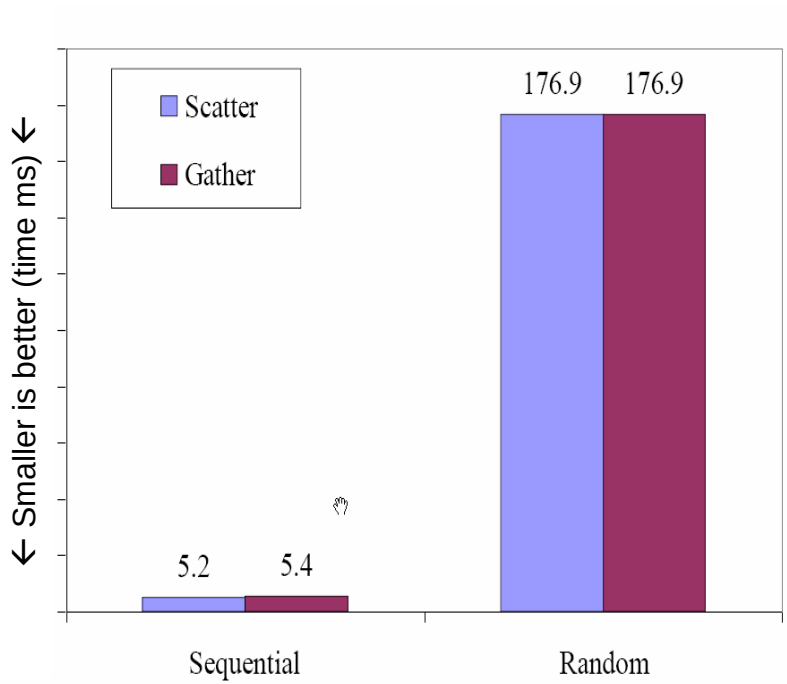
CPU in cache approx. GPU streaming

GPU connection to main memory (PCIe) can be a significant bottleneck.

Example: Data locality and performance

GPU memory access speed: Cached, sequential and random
(left diagram: only gather)





Charts courtesy Ian Buck and Naga Govindaraju

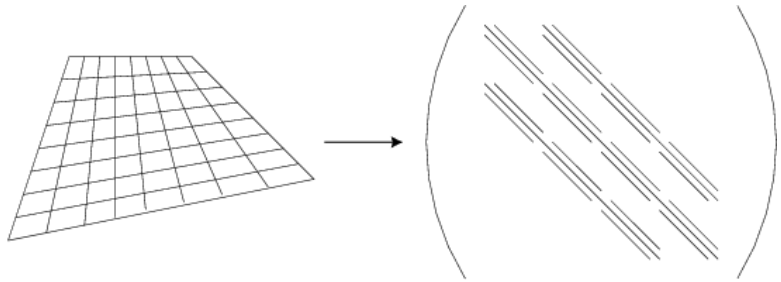
Co-processor evaluation

- **Evaluation strategy**
  - Perform experiments before attempting co-processor integration
  - Try to select relevant tests
  - Do not focus on microbenchmarks alone
  - Compute-bound codes on the CPU can be memory-bound on co-processors

- **Key tests for us**
  - Performance of matrix-vector multiplication typically a good estimate of overall performance
  - FE solvers are prone to suffer from reduced accuracy
  - Poisson equation as representative for scalar elliptic PDEs

Test 1 - Microbenchmarks

- **Model problem**
  - Poisson on unitsquare
  - Generalised tensorproduct mesh
  - Conforming bilinear elements (Q_1)
  - Band-structured matrix with 9 bands



- **Benchmark results**
  - Matrix-vector multiplication on Geforce 8800 GTX GPU
  - Arithmetic intensity: 17N flops, 19N loads/stores (extremely bandwidth-limited)
  - Sustained 18 GFLOP/s (5% peak)
  - Sustained 65-70 GByte/s (>80% peak)

- **So far: quite promising**

Test 2 - Accuracy

- **Test scenario**
  - Laplacian of analytic test function as RHS to Poisson equation
  - Multigrid solver, first in double then in single precision
  - Expect L_2 error reduction by factor of 4 per refinement step

	DOUBLE	REDUCTION	SINGLE	REDUCTION
3^2	5.208e-3		5.208e-3	
5^2	1.440e-3	3.62	1.440e-3	3.62
9^2	3.869e-4	3.72	3.869e-4	3.72
17^2	1.015e-4	3.81	1.015e-4	3.81
33^2	2.607e-5	3.89	2.611e-5	3.89
65^2	6.612e-6	3.94	6.464e-6	4.04
129^2	1.666e-6	3.97	1.656e-6	3.90
257^2	4.181e-7	3.98	5.927e-7	2.79
513^2	1.047e-7	3.99	2.803e-5	0.02
1025^2	2.620e-8	4.00	7.708e-5	0.36

- **Not promising at all, advanced techniques required**

Outline

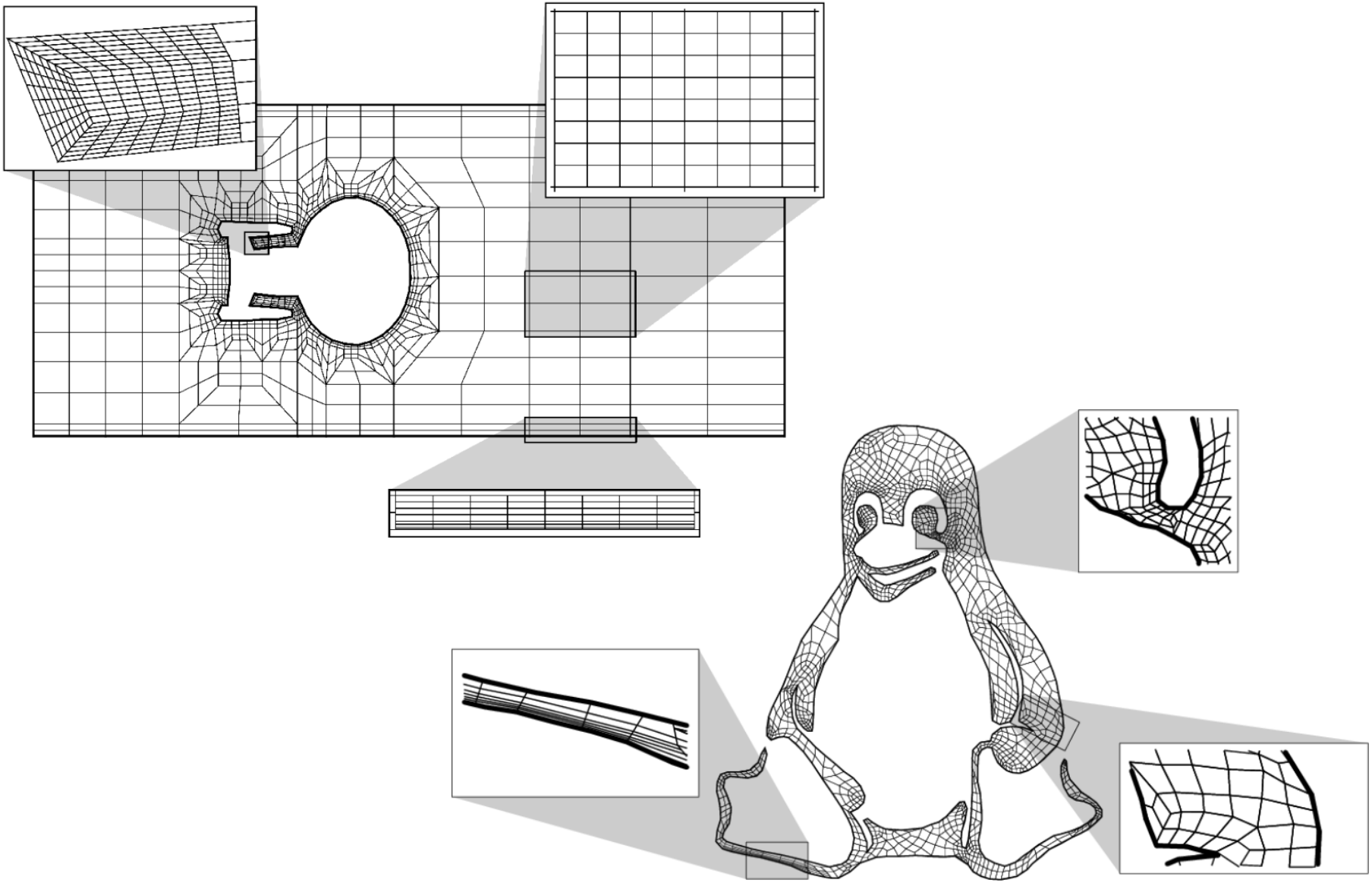
- Introduction
- Starting point: FEAST - Finite Element Analysis and Solution Tools
- Co-processor integration
- Exemplary results
- Summary and conclusions

FEAST

- **FEAST – Finite Element Analysis and Solution Tools**
  - Under development at TU Dortmund in Stefan Turek‘s group
  - http://www.feast.tu-dortmund.de

- **Core features**
  - Separation of unstructured and structured data for optimised linear algebra components
  - Finite Element discretisations
  - Parallel generalised domain decomposition multigrid solvers
  - Scalar and vector-valued problems
  - applications in CFD and CSM

FEAST grids



Data structures

- **Cover domain by unstructured collection of subdomains**
  - Resolve complex geometries, boundary layers in fluid dynamics, etc.
- **Refine each subdomain independently and discretise using FEs**
  - Generalised tensorproduct fashion
  - Isotropic and anisotropic refinement combined with r/h/rh adaptivity
- **Performance**
  - Clear separation of globally unstructured and locally structured parts
  - Nonzero pattern of local FE matrices known a priori
  - Exploit spatial and temporal locality for tuned LA building blocks (Sparse Banded BLAS)

Parallel multigrid solvers

- **Contradictory properties**
  - Numerical vs. computational efficiency
  - Weak and strong scalability vs. numerical scalability
- **Parallel multigrid**
  - Strong recursive coupling optimal in serial codes
  - Usually relaxed to block-Jacobi due to high comm requirements
  - Degrades convergence rates in the presence of local anisotropies
- **Generalised DD/MG approach (ScaRC)**
  - Global MG, block-smoothed by local MGs
  - Hide anisotropies locally
  - Good scalability by design
  - Global operations realised via special local BCs and syncronisation across subdomain boundaries (no overlap!)

Vector-valued problems

- **Guiding idea**
  - Tune linear algebra and solver implementation once per architecture instead of over and over again per application

- **Block-structured systems**
  - Equation-wise ordering of the unknowns
  - $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ correspond to scalar elliptic operators

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

- **Examples**
  - Elasticity with compressible material
  - Stokes
  - Elasticity with (nearly) incompressible material

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

Vector-valued problems

- **Solution approach**
  - Illustrative example: Richardson iteration with block-Jacobi preconditioning

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

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \widetilde{\mathbf{A}}^{-1}(\mathbf{f} - \mathbf{A}\mathbf{u}^k)$$

$$\widetilde{\mathbf{A}}^{-1} = \begin{pmatrix} \mathbf{A}_{11}^{-1} & \mathbf{0} \\ 0 & \mathbf{A}_{22}^{-1} \end{pmatrix}$$

- **Real solvers**
  - Block-preconditioning with Gauss-Seidel, SOR etc.
  - Krylov-subspace methods, multigrid
  - Implemented as sequences of operations on individual blocks

Outline

- Introduction
- Starting point: FEAST - Finite Element Analysis and Solution Tools
- Co-processor integration
- Exemplary results
- Summary and conclusions

Solver example

- **First step: Identify accelerable parts of the solver**

- **Global BiCGStab**
  - Preconditioned by
- **Global MG**
  - Smoothed by
- **Local MGs per subdomain**

Solver example

- **First step: Identify accelerable parts of the solver**

- **Global BiCGStab**
  - Preconditioned by
- **Global MG**
  - Smoothed by
- **Local MGs per subdomain**

Realised as a series of local operations on each subdomain.

Solver example

- **First step: Identify accelerable parts of the solver**

- **Global BiCGStab**
  - Preconditioned by
- **Global MG**
  - Smoothed by
- **Local MGs per subdomain**

Realised as a series of local operations on each subdomain.

Typically one operation (defect calculation, grid transfers etc.) directly followed by neighbour communication (MPI).

Solver example

- **First step: Identify accelerable parts of the solver**

- **Global BiCGStab**
  - Preconditioned by
- **Global MG**
  - Smoothed by
- **Local MGs per subdomain**

Realised as a series of local operations on each subdomain.

Typically one operation (defect calculation, grid transfers etc.) directly followed by neighbour communication (MPI).

Poor acceleration potential due to PCIe bottleneck.

Solver example

- **First step: Identify accelerable parts of the solver**

- **Global BiCGStab**
  - Preconditioned by
- **Global MG**
  - Smoothed by
- **Local MGs per subdomain**

Realised as a series of local operations on each subdomain.

Typically one operation (defect calculation, grid transfers etc.) directly followed by neighbour communication (MPI).

1-2 full MG cycles with up to 1e6 unknowns.

Poor acceleration potential due to PCIe bottleneck.

Solver example

- **First step: Identify accelerable parts of the solver**

- **Global BiCGStab**
  - Preconditioned by
- **Global MG**
  - Smoothed by
- **Local MGs per subdomain**

Realised as a series of local operations on each subdomain.

Typically one operation (defect calculation, grid transfers etc.) directly followed by neighbour communication (MPI).

1-2 full MG cycles with up to 1e6 unknowns.

Poor acceleration potential due to PCIe bottleneck.

Good spatial locality.

Enough work for fine-grained parallelism.

Solver example

- **First step: Identify accelerable parts of the solver**

- **Global BiCGStab**
  - Preconditioned by
- **Global MG**
  - Smoothed by
- **Local MGs per subdomain**

Realised as a series of local operations on each subdomain.

Typically one operation (defect calculation, grid transfers etc.) directly followed by neighbour communication (MPI).

1-2 full MG cycles with up to 1e6 unknowns.

Poor acceleration potential due to PCIe bottleneck.

Good spatial locality.

Enough work for fine-grained parallelism.

Good acceleration potential.

Solver example

- **First step: Identify accelerable parts of the solver**

- **Global BiCGStab**
  - Preconditioned by
- **Global MG**
  - Smoothed by
- **Local MGs per subdomain**

Realised as a series of local operations on each subdomain.

Typically one operation (defect calculation, grid transfers etc.) directly followed by neighbour communication (MPI).

1-2 full MG cycles with up to 1e6 unknowns.

Poor acceleration potential due to PCIe bottleneck.

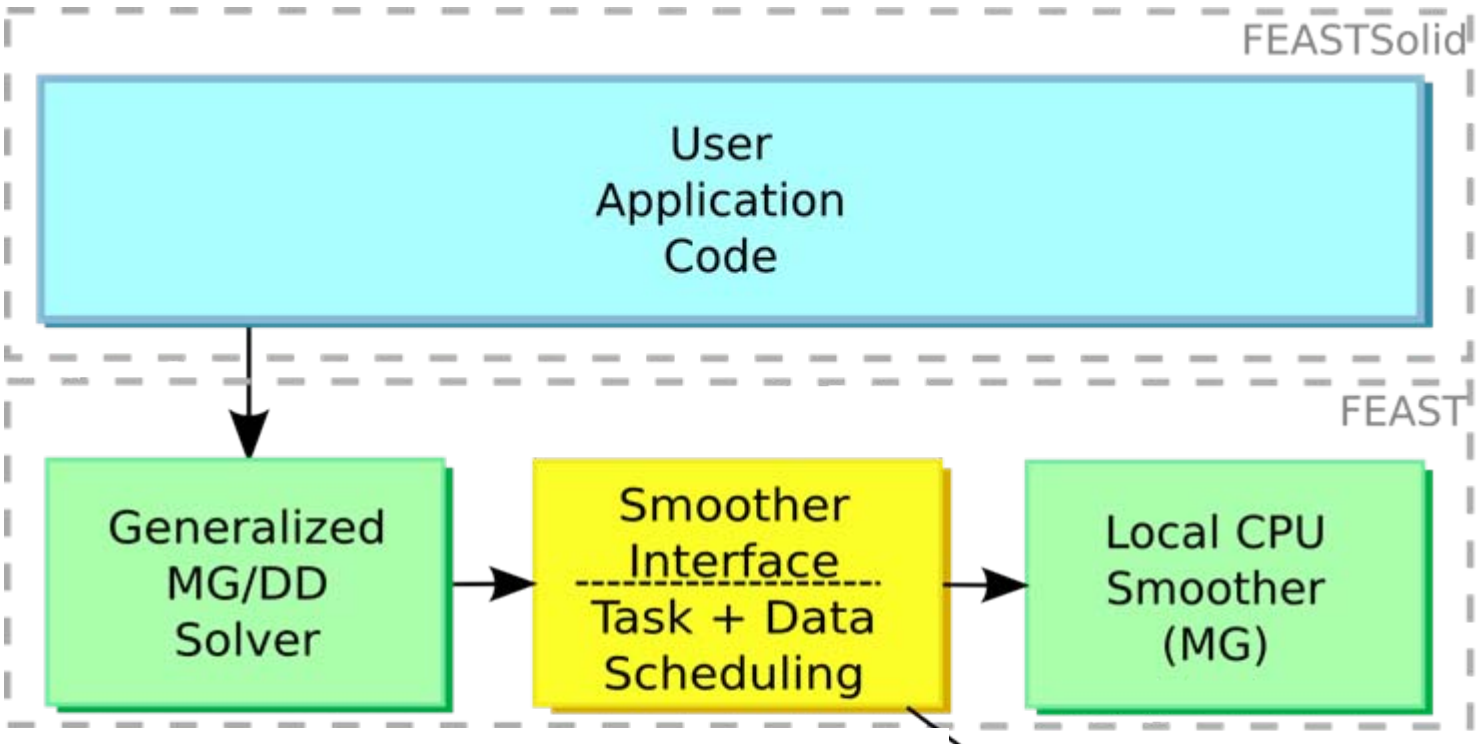
Good spatial locality.

Enough work for fine-grained parallelism.

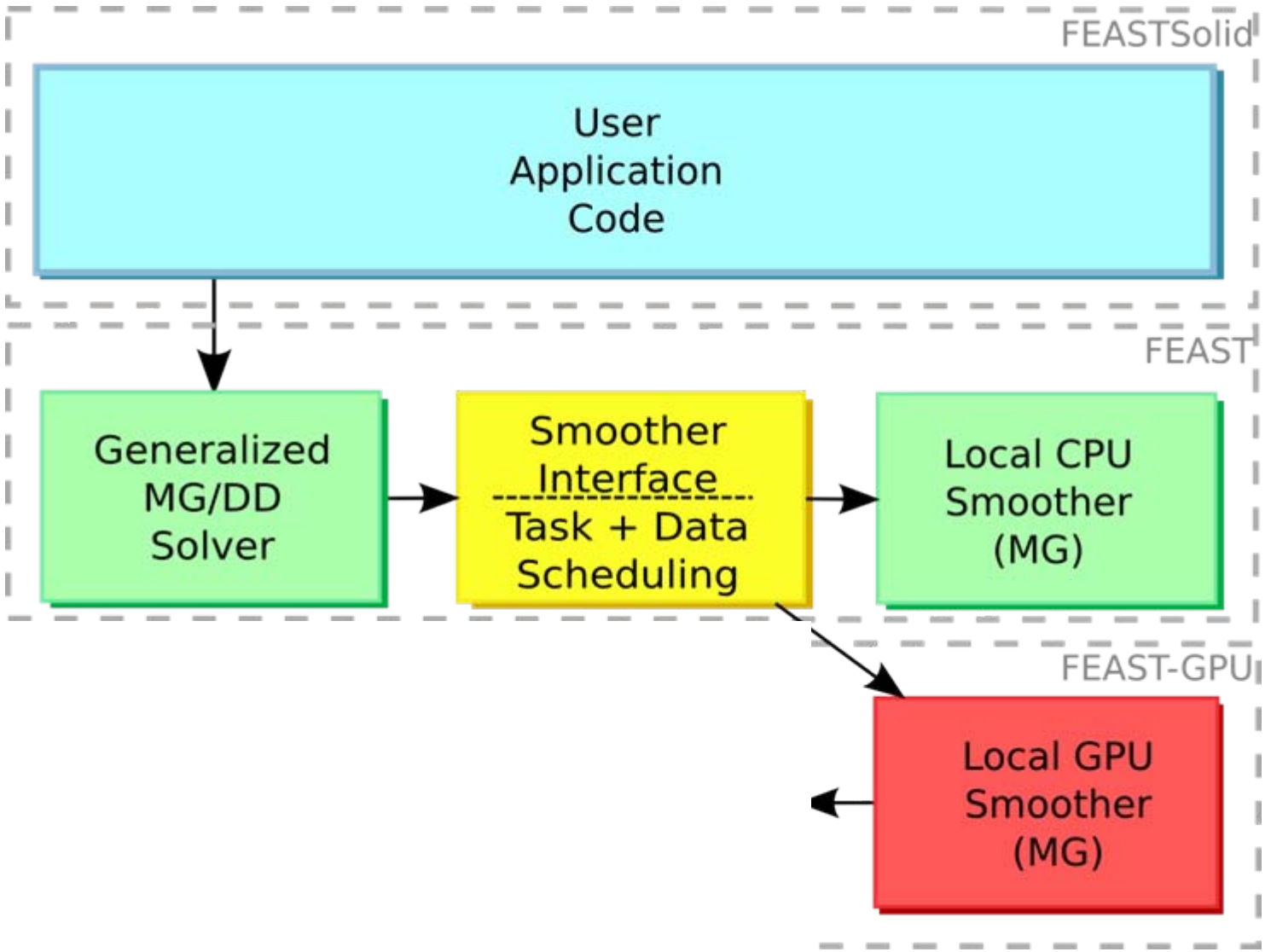
Good acceleration potential.

Only accelerate scalar solvers.

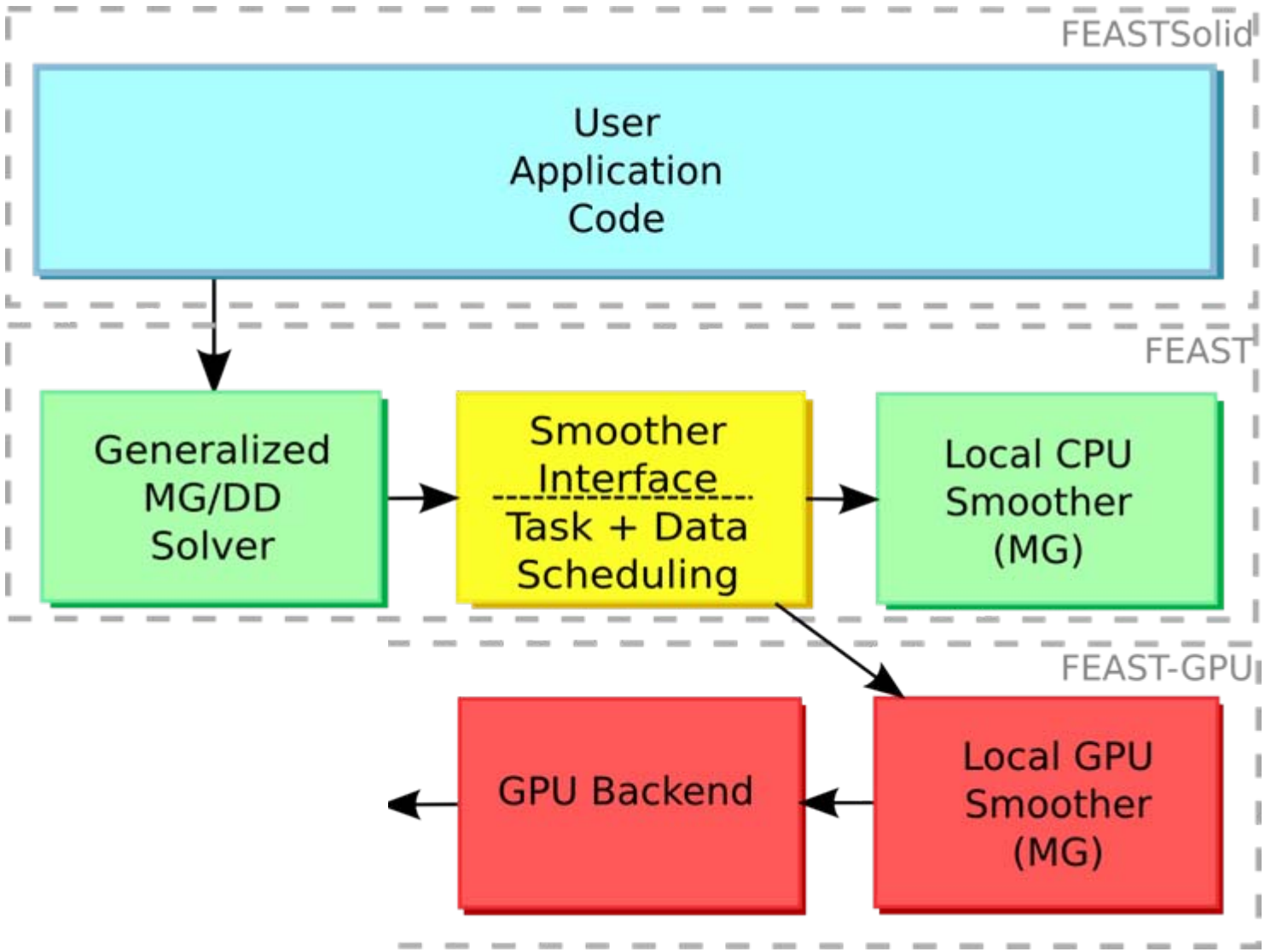
Integration overview



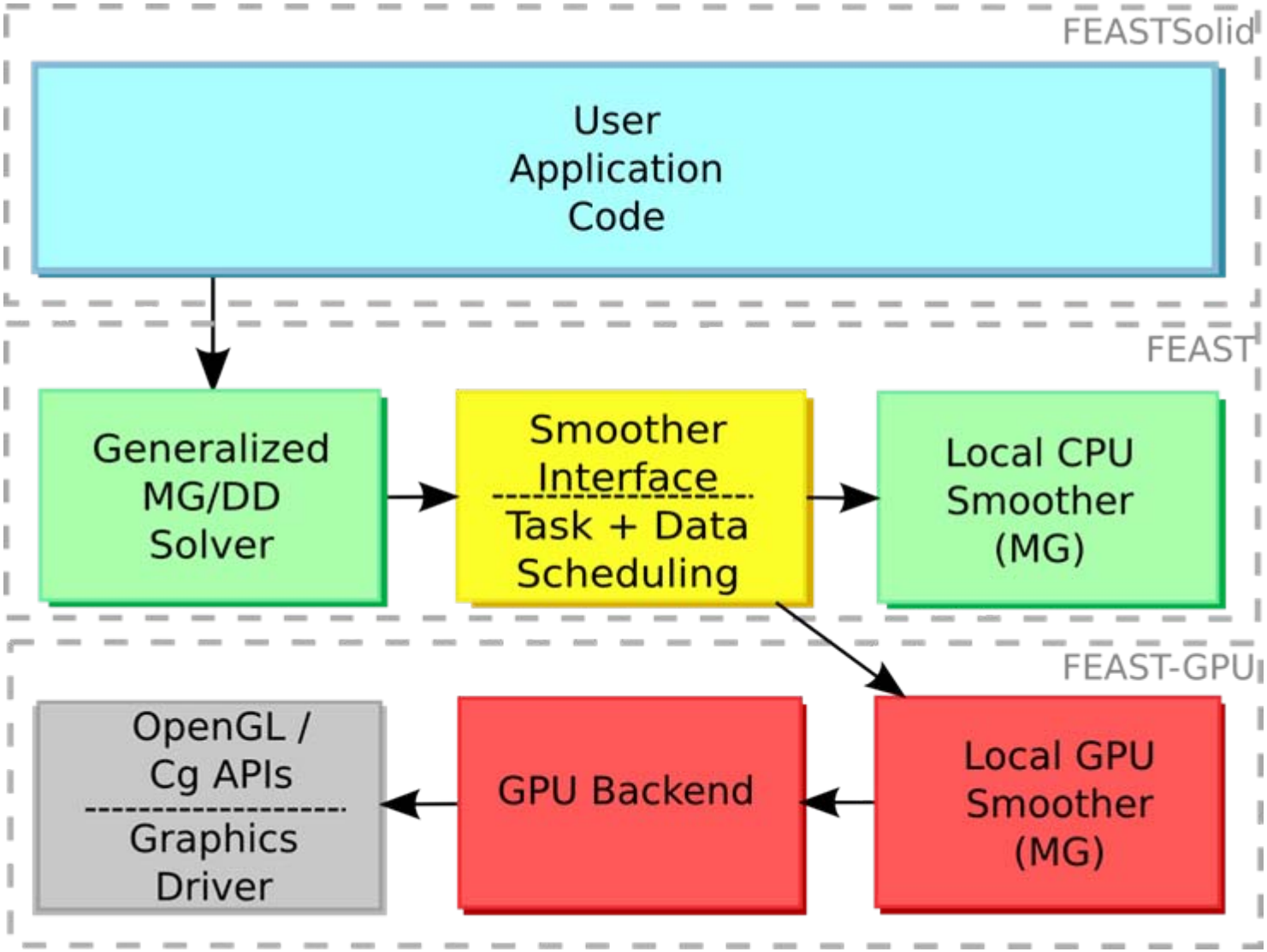
Integration overview



Integration overview



Integration overview



Integration summary

- **Isolate suitable parts**
  - Balance acceleration potential and acceleration effort
- **Diverge code paths as late as possible**
  - Local MG solver
  - Same interface for several co-processors
- **Important benefit of this minimally invasive approach: No changes to application code**
  - Co-processor code can be developed and tuned on a single node
  - Entire MPI communication infrastructure remains unchanged
  - But: prone to suffer from Amdahl's Law (discussed later)

Integration challenges

- **The usual perils and pitfalls in parallel computing**
  - Heterogeneity complicates load balancing
  - Heterogeneity complicates assigning jobs to specific resources
  - Don't want to leave the CPU idle while the co-processor computes

- **GPU-specific issues**
  - Building GPU clusters (density, power supplies, cooling)
  - Maintenance slightly increased
  - Our experience: MTBF and MTBM not affected (for professional-level cards)

Integration challenges

- **Data transfers to and from device**
  - Model: huge L3 cache with prefetching
  - Automatic prefetching: Pass control over all matrix data in a preprocessing stage
  - Manual prefetching: Complicated as older GPUs (pre-CUDA) don't support async. transfers

- **Precision vs. accuracy**
  - Double precision needed, but only at crucial stages of the computation
  - Mixed precision iterative refinement approach
  - Outer solver: high precision
  - Inner solver: low precision
  - Accuracy not affected (see results in a minute)

Paper: Göddeke et al., Using GPUs to improve multigrid solver performance on a cluster, accepted for publication in "Computational Science and Engineering", 2007

Outline

- Introduction
- Starting point: FEAST - Finite Element Analysis and Solution Tools
- Co-processor integration
- Exemplary results
- Summary and conclusions

Linearised Elasticity

- **Computational Solid Mechanics Application FEASTsolid**
  - Fundamental model problem: elastic, compressible material
  - Small deformations, static loading process
  - Hooke's material law

- **Lamé equation**

$$\begin{aligned} -2\mu\ \mathbf{div}\,\boldsymbol{\varepsilon}(\boldsymbol{u}) - \lambda\ \mathbf{grad}\,\mathrm{div}\,\boldsymbol{u} &= \boldsymbol{f}, & \boldsymbol{x} &\in \Omega \\ \boldsymbol{u} &= \boldsymbol{g}, & \boldsymbol{x} &\in \Gamma_{\mathrm{D}} \\ \boldsymbol{\sigma}(\boldsymbol{u}) \cdot \boldsymbol{n} &= \boldsymbol{t}, & \boldsymbol{x} &\in \Gamma_{\mathrm{N}} \end{aligned}$$

- **Separate displacement ordering**

$$-\begin{pmatrix} (2\mu+\lambda)\partial_{xx} + \mu\partial_{yy} & (\mu+\lambda)\partial_{xy} \\ (\mu+\lambda)\partial_{yx} & \mu\partial_{xx} + (2\mu+\lambda)\partial_{yy} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$$

Discretisation and solver details

- **Discretisation of reordered Lamé equation**
  - block-structured system

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

- **Coupling of K_{11}, K_{21}, K_{22}**
  - Block Gauß-Seidel smoothing of global multigrid solver
  - $\mathbf{K}_{11}\,\mathbf{u}_1 = \mathbf{f}_1$ and $\mathbf{K}_{22}\,\mathbf{u}_2 = \mathbf{f}_2$ correspond to scalar elliptic equations and solvers for them can be accelerated

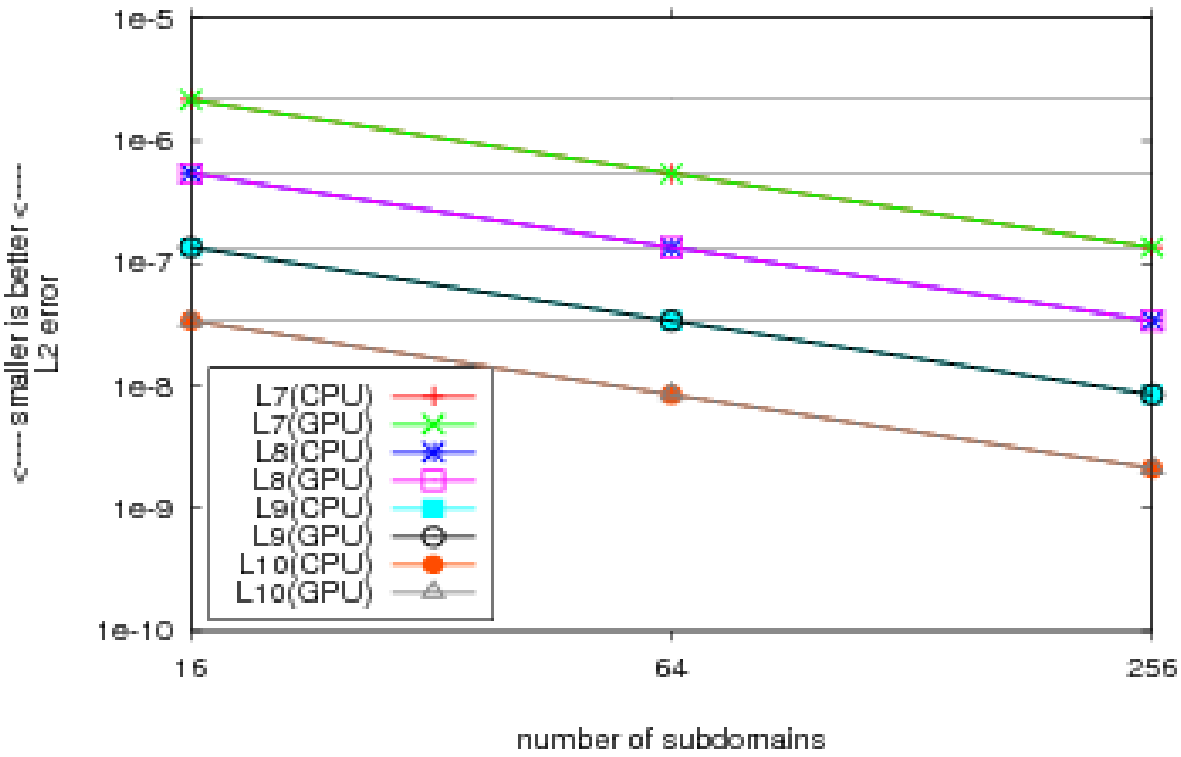
- **Solver specialisation**
  - Global BiCGStab (vector-valued) preconditioned by
  - Global multigrid (vector-valued) block-GS-smoothed by
  - Local multigrids (scalar) per subdomain

Test goals

- **Accuracy**
  - Evaluate impact of reduced precision
  - Analytic reference solution
  - Global anisotropies to worsen condition numbers
- **Scalability**
  - Here: only weak scalability
- **Speedup**
  - Exemplarily for some test scenarios
  - Detailed analysis and understanding of speedup components

Accuracy

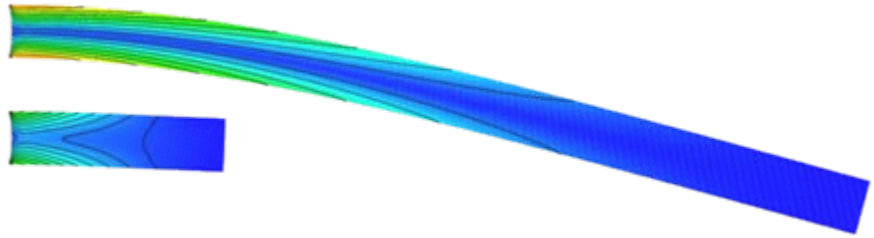
- L_2 error against analytically known reference solution



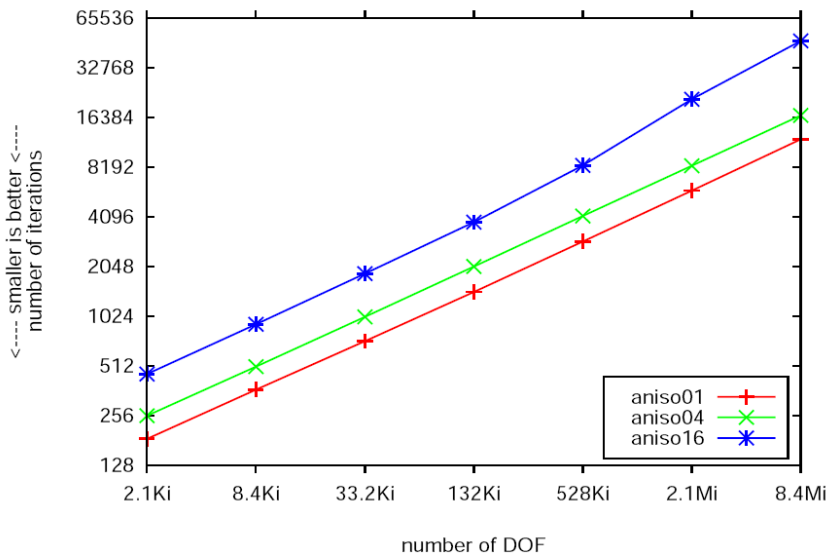
- Same results for CPU and GPU
  - expected error reduction independent of refinement and subdomain distribution

Ill-conditioned systems

- **Test case: Cantilever beam**
  - Global anisotropy and ratio of fixed Dirichlet and free Neumann BCs proportional to # of processors



- **Illustration of ill-conditioning**
  - Plain Conjugate gradient solver
  - Anisotropies of 1:1, 1:4 and 1:16
  - Plot: Number of iterations for increasing problem size, logscale
  - Aniso16 does not even exhibit doubling of iterations any more



Ill-conditioned systems

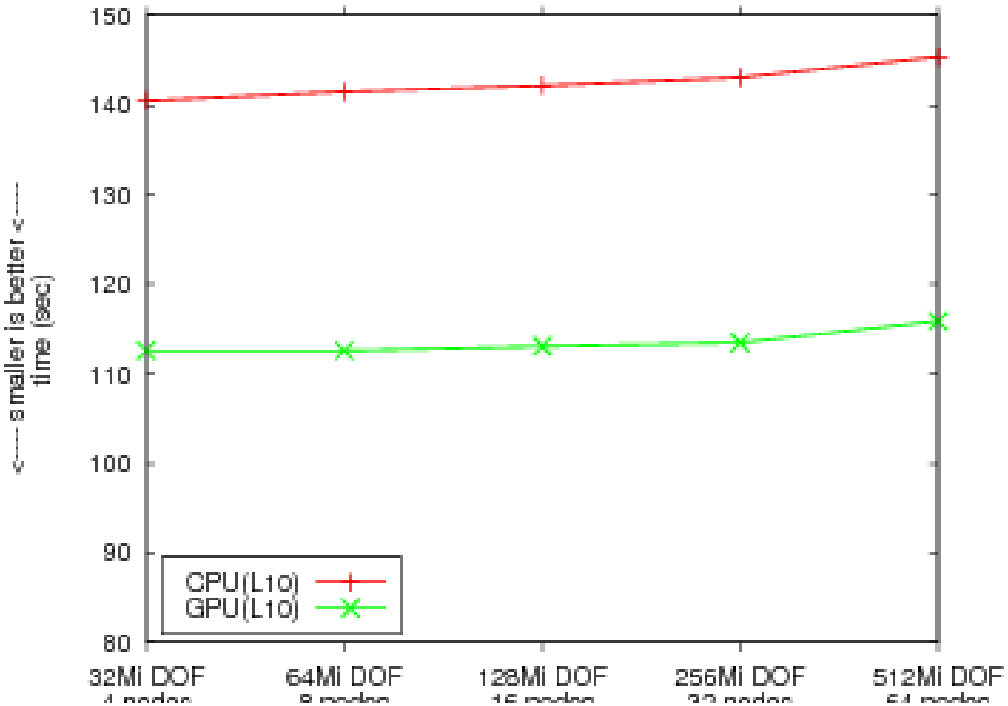
- **CPU-GPU comparison**

aniso04 refinement L	Iterations CPU	Iterations GPU	Volume CPU	Volume GPU	y-Displacement CPU	y-Displacement 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**1E-2**	-6.621655**2E-2**
10	5.5	5.5	6.7176516E-3	6.7176516E-3	-6.621750**1E-2**	-6.621750**2E-2**

- **Same solution for GPU and CPU**
  - Volume of deformed body
  - Displacement of reference point at tip of the beam
  - Same number of iterations until convergence

Weak scalability

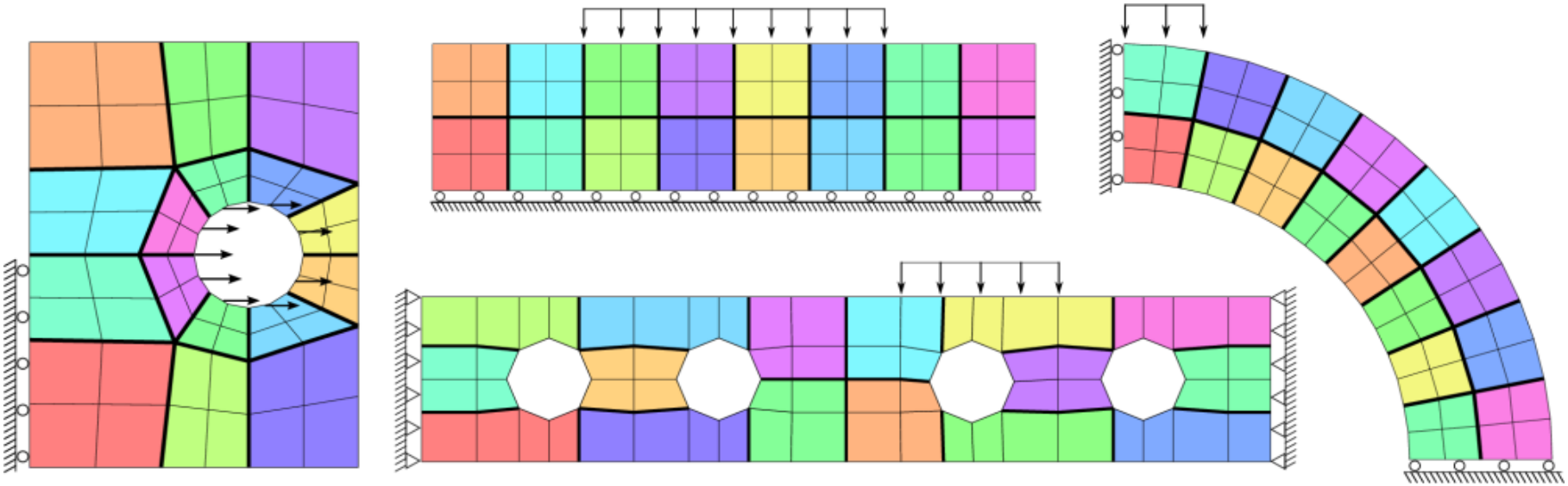
- **Good scalability**
  - original and accelerated CSM solver
  - Infiniband, Xeon EM64T, 3.4GHz, outdated Quadro 1400 GPU



- **More results**
  - Poisson problem for 1.3 billion unknowns in less than 50 seconds on 160 outdated GPUs (Quadro 1400)

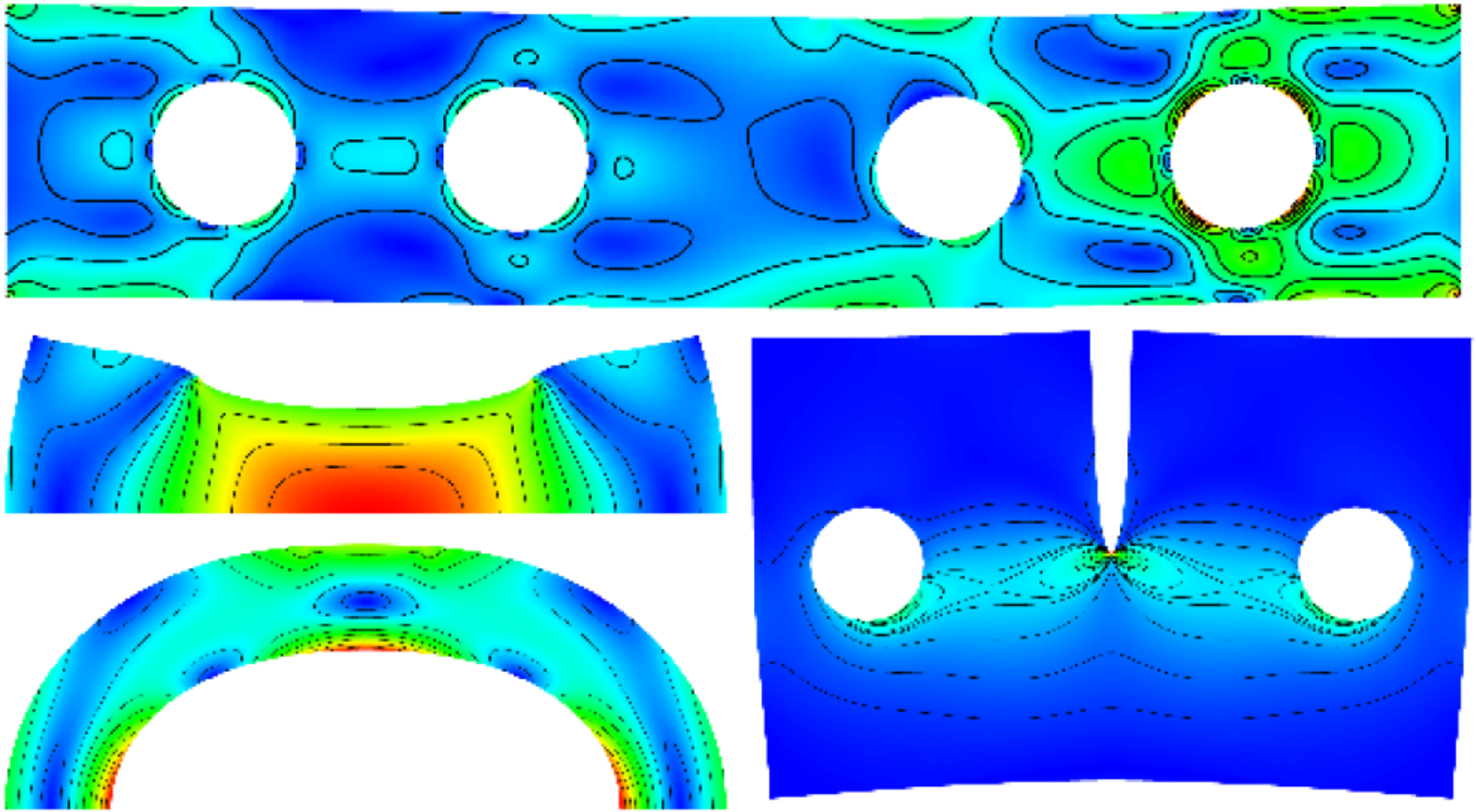
Paper: Göddeke et al., Exploring weak scalability for FEM calculations on a GPU-enhanced cluster, Parallel Computing 33(10-11), 685-699, 2007

Test configurations

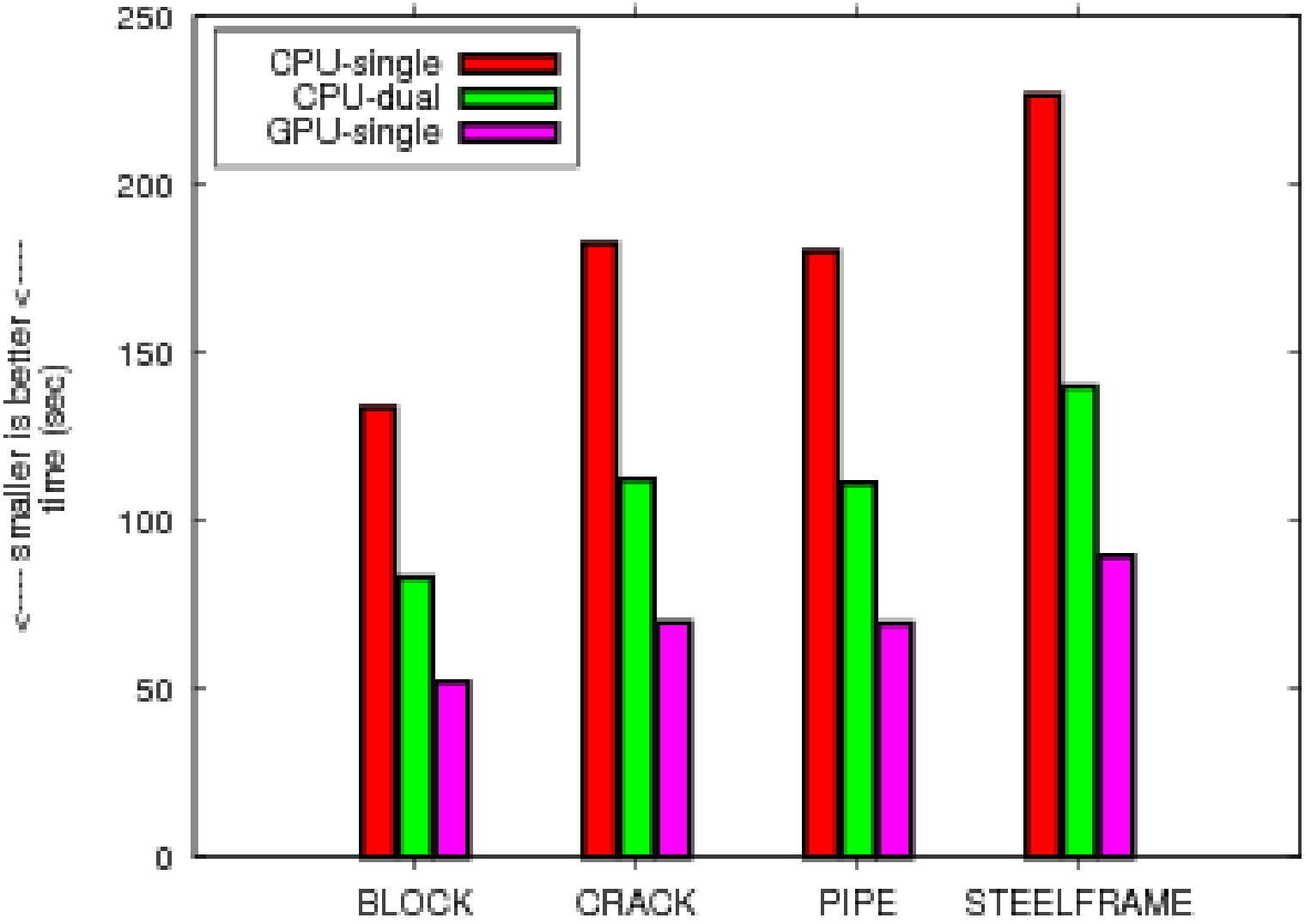


Test system with 16 nodes: dualcore Santa Rosa Opteron CPU, Quadro 5600 GPU, Infiniband

Displacements and von Mises stresses



Speedup


CPU-single
CPU-dual
GPU-single
<---- smaller is better <----
time (sec)
250
200
150
100
50
0
BLOCK
CRACK
PIPE
STEELFRAME

Speedup analysis

- **Speedups in 'time to solution' for one GPU**
  - 2.6x vs. singlecore, 1.6x vs. dualcore
- **Amdahl's Law is lurking**
  - Profiling: Local speedup of 9x and 5.5x by the GPU
  - Observation: 2/3 of the entire solver can be accelerated, so the theoretical upper bound for the speedup is 3x (2.6x is quite good!)
  - For dualcores, we expect 2.2x but see only 1.6x
  - Explanation: Overlapping communication / memory access and computation are giving the dualcore an 'unfair advantage'
- **Avenue for future work**
  - Exploit available resources within the node better
  - Three-way parallelism in our system: coarse-grained (MPI) - medium-grained (resources within the node) - fine-grained (compute cores in the GPU)

Outline

- Introduction
- Starting point: FEAST - Finite Element Analysis and Solution Tools
- Co-processor integration
- Exemplary results
- Summary and conclusions

Summary and conclusions

- **Identify accelerable parts of established code**
  - Balance acceleration potential and acceleration effort
- **Minimally invasive integration without changes to application code**
  - Instead, hardware acceleration available under the same interface
  - User gets acceleration for free, just a change in a parameter file
- **Good speedups**
  - Limited by Amdahl's Law
  - Future work needs to address 3-way parallelism

    coarse-grained (MPI between nodes)

    medium-grained (resources within the node)

    fine-grained (compute cores in the GPU)s

Acknowledgements

- **Hilmar Wobker, Stefan Turek and the FEAST group**
  TU Dortmund

- **Robert Strzodka, Max Planck Institut Informatik**
  Saarbrücken

- **Patrick McCormick, Jamaludin Mohd-Yusof**
  Los Alamos National Labs