Finite Element Multigrid Solvers for PDE Problems on GPUs and GPU Clusters
Part 2: Applications on GPU Clusters

Dominik Göddeke and Robert Strzodka

Institut für Angewandte Mathematik (LS3), TU Dortmund

Integrative Scientific Computing, Max Planck Institut Informatik

INRIA summer school, June 8, 2011

technische universität dortmund

fakultät für mathematik

Max Planck Center for visual computing and communication

Introduction

Key topic of Robert's talk: Fine-grained parallelism within a single GPU

- Geometric multigrid solvers on GPUs
- Precision vs. accuracy
- Strong smoothers and preconditioners for ill-conditioned problems

This talk

- Combining fine-grained GPU parallelism with 'conventional' MPI-like parallelism
- Porting complex applications to GPU clusters: Rewrite or accelerate
- Case studies: Seismic wave propagation, solid mechanics and fluid dynamics

Existing codes

Common situation: Existing legacy codes

- Large existing code bases, often 100.000+ lines of code
- Well validated and tested, (often) sufficiently tuned
- Commonly not ready for hybrid architectures, often based on an 'MPI-only' approach

Applications vs. frameworks (toolboxes)

- One application to solve one particular problem repeatedly, with varying input data
- Common framework that many applications are build upon
- In our case, a Finite Element multigrid toolbox to numerically solve a wide range of PDE problems

Two general options to include accelerators

Rewrite everything for a new architecture

- Potentially best speedups
- But: Re-testing, re-tuning, re-evaluating, over and over again for each new architecture
- Well worth the effort in many cases
- First part of this talk: Case study in seismic wave propagation

Accelerate only crucial portions of a framework

- Potentially reduced speedups
- Changes under the hood and all applications automatically benefit
- Careful balancing of amount of code changes and expected benefits: *Minimally invasive integration*
- Second part of this talk: Case study for large-scale FEM-multigrid solvers at the core of PDE simulations

Case Study 1:

Seismic Wave Propagation on GPU Clusters

Introduction and Motivation

Acknowledgements

Collaboration with

- Dimitri Komatitsch: Université de Toulouse, Institut universitaire de France, CNRS & INRIA Sued-Oest MAGIQUE-3D, France
- Gordon Erlebacher: Florida State University, Tallahassee, USA
- David Michéa: Bureau de Recherches Géologiques et Minières, Orléans, France

Funding agencies

- French ANR grants NUMASIS, support by CNRS, IUF, INRIA
- German DFG and BMBF grants

Publications

- *High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster*, Journal of Computational Physics 229:7692-7714, Oct. 2010
- *Modeling the propagation of elastic waves using spectral elements on a cluster of 192 GPUs*, Computer Science – Research and Development 25(1-2):75-82, Special Issue International Supercomputing Conference (ISC'10), May/Jun. 2010

Seismic wave propagation

Application domains

- Earthquakes in sedimentary basins and at the scale of a continent
- Active acquisition experiments in the oil and gas industry

High practical relevance

- L'Aquila, Italy
- April 2009
- 5.8 Richter scale
- 260 dead
- 1.000 injured
- 26.000 homeless

Very efficient numerical and computational methods required!

Topography and sedimentary basins

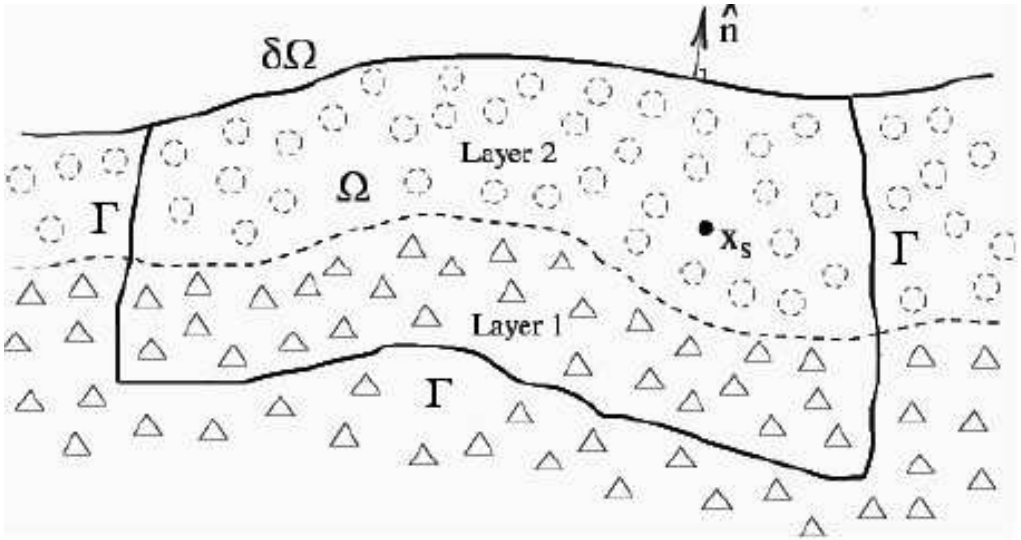
Topography needs to be honoured

- Densely populated areas are often located in sedimentary basins
- Surrounding mountains reflect seismic energy back and amplify it (think rigid Dirichlet boundary conditions)
- Seismic shaking thus much more pronounced in basins

High resolution requirements

Spatially varying resolution

- Local site effects and topography
- Discontinuities between heterogeneous sedimentary layers and faults in the Earth



High seismic frequencies need to be captured

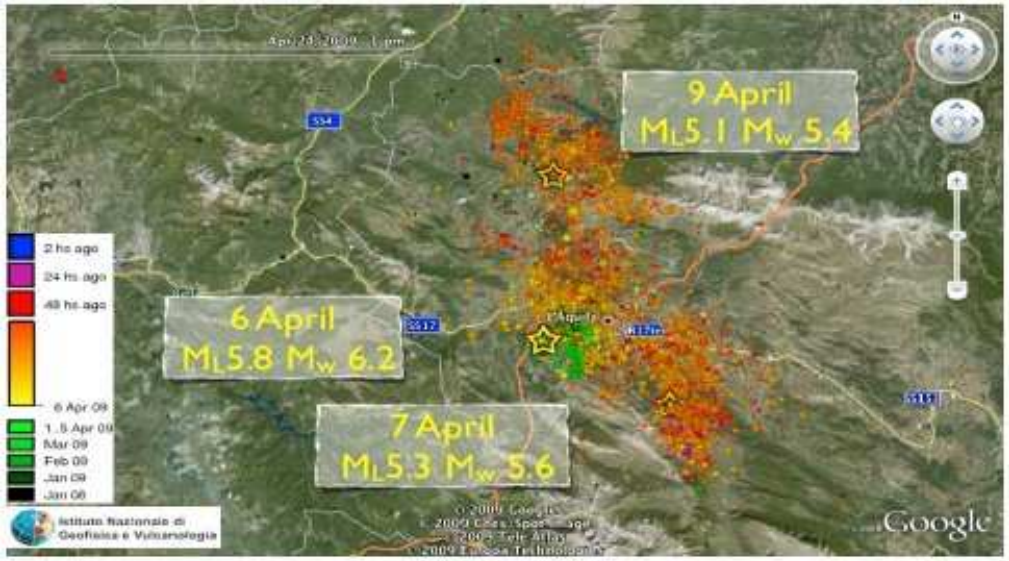
- High-order methods and finely-resolved discretisation in space and time required

Timing constraints: Aftershocks

Main shock typically followed by aftershocks

- Predict effect of aftershocks within a few hours after an earthquake
- Predict impact on existing faults (from previous earthquakes) that may break due to changed stress distribution in the area
- Finish simulation ahead of time of follow-up shaking to issue detailed warnings

L'Aquila earthquake aftershock predicion



Summary: Challenges

Conflicting numerical goals

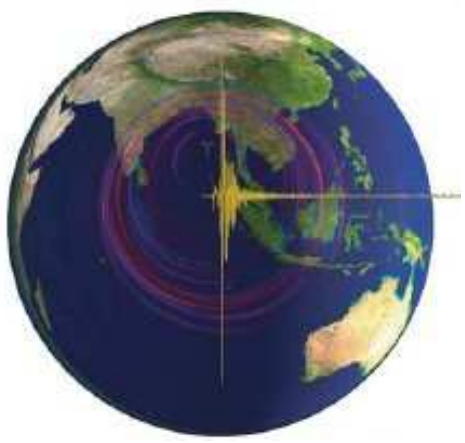
- High-order methods in space and time
- But: High flexibility and versatility required
- Must be efficiently parallelisable in a scalable way

Extremely high computational demands of typical runs

- 100s of processors and 1000s of GB worth of memory
- Several hours to complete (100.000s of time steps)

SPECFEM3D software package

- `http://www.geodynamics.org`
- Open source and widely used
- Accepts these challenges and implements good compromises
- Gordon Bell 2003, finalist 2008 (sustained 0.2 PFLOP/s)



Physical Model and

Numerical Solution Scheme

Physical model: Elastic waves

Model parameters

- Linear anisotropic elastic rheology for a heterogeneous solid part of the Earth mantle, full 3D simulation

Strong and weak form of the seismic wave equation

$$\rho\ddot{\mathbf{u}} = \boldsymbol{\nabla}\cdot\boldsymbol{\sigma} + f \qquad \boldsymbol{\sigma} = \mathbf{C} : \boldsymbol{\varepsilon} \qquad \boldsymbol{\varepsilon} = \frac{1}{2}\left(\boldsymbol{\nabla}\mathbf{u} + (\boldsymbol{\nabla}\mathbf{u})^{\mathsf{T}}\right)$$

$$\int_\Omega \rho\mathbf{w}\cdot\ddot{\mathbf{u}}\; d\Omega + \int_\Omega \boldsymbol{\nabla}\mathbf{w} : \mathbf{C} : \boldsymbol{\nabla}\mathbf{u}\; d\Omega = \int_\Omega \mathbf{w}\cdot f\; d\Omega$$

- Displacement $\mathbf{u}$, stress and strain tensors $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$
- Stiffness tensor $\mathbf{C}$ and density ρ (given spatially heterogeneous material parameters)
- Time derivative $\ddot{\mathbf{u}}$ (acceleration)
- External forces $\mathbf{f}$ (i.e., the seismic source), test function $\mathbf{w}$
- Boundary integral in weak form vanishes due to free surface B.C.

Numerical methods overview

Finite Differences

- Easy to implement, but difficult for boundary conditions, surface waves, and to capture nontrivial topography

Boundary Elements, Boundary Integrals

- Good for homogeneous layers, expensive in 3D

Spectral and pseudo-spectral methods

- Optimal accuracy, but difficult for boundary conditions and complex domains, difficult to parallelise

Finite Elements

- Optimal flexibility and error analysis framework, but may lead to huge sparse ill-conditioned linear systems

Spectral element method (SEM)

Designed as a compromise between conflicting goals

- 'Hybrid' approach: Combines accuracy of pseudo-spectral methods with geometric flexibility of Finite Element methods
- Parallelises moderately easy

Cover domain with large, curvilinear hexahedral 'spectral' elements

- Edges honours topography and interior discontinuities (geological layers and faults)
- Mesh is unstructured in the Finite Element sense

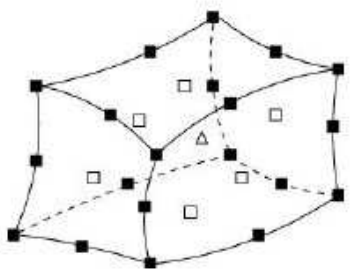
Use high-order interpolation

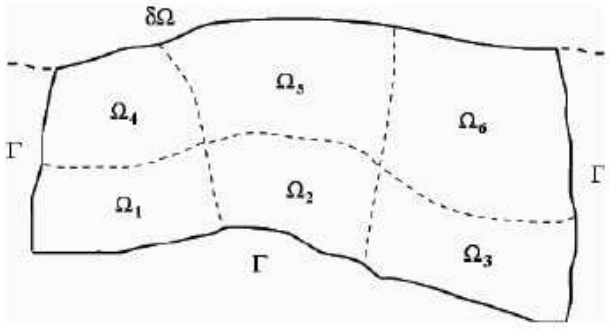
- To represent physical fields in each element
- Sufficiently smooth transition between elements

SEM for the seismic wave equation

Represent fields in each element by Lagrange interpolation

- Degree 4–10, 4 is a good compromise between accuracy and speed
- Use Gauß-Lobotto-Legendre control points (rather than just Gauß or Lagrange points)
- Degree+1 GLL points per spatial dimension per element (so 125 for degree 4 in 3D)
- Physical fields represented as triple products of Lagrange basis polynomials





SEM for the seismic wave equation

Clever trick: Use GLL points as cubature points as well

- To evaluate integrals in the weak form
- Lagrange polynomials combined with GLL quadrature yields strictly diagonal mass matrix

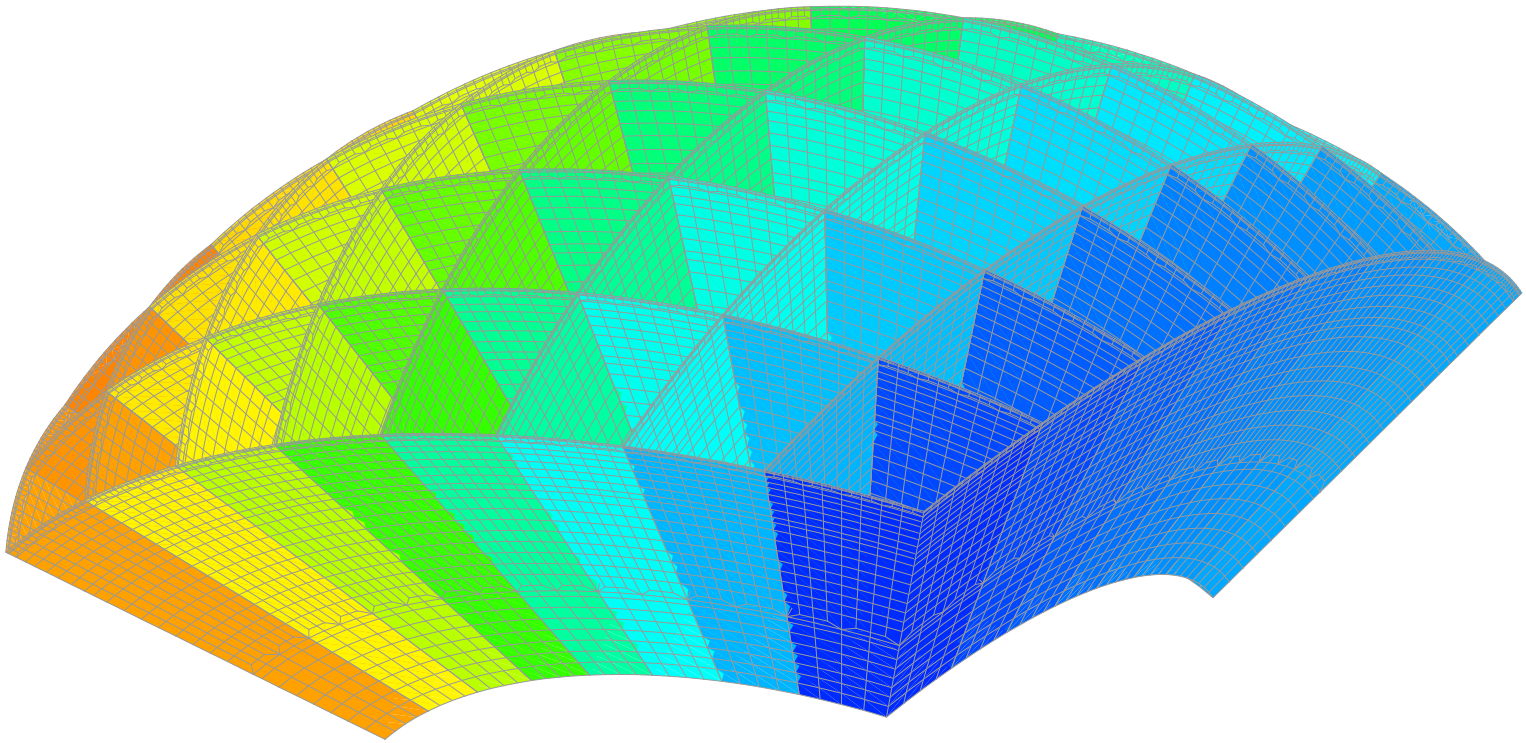
Important consequence: Algorithm significantly simplified

- Explicit time stepping schemes become feasible
- In our case: Second order centred finite difference Newmark time integration
- Solving of linear systems becomes trivial

Meshing

Block-structured mesh

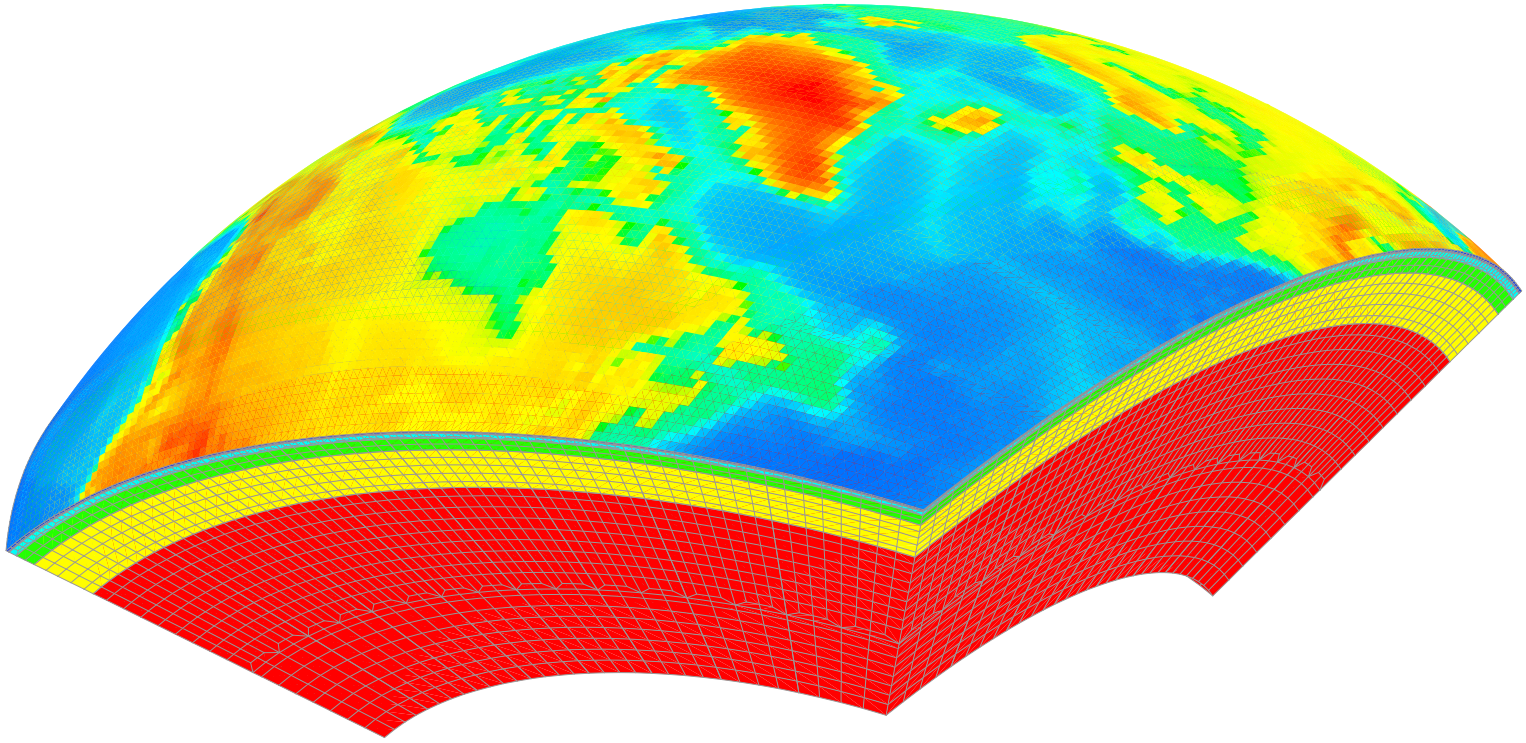
- Blocks are called slices
- Each slice is unstructured, but all are topologically identical
- Work per timestep per slice is identical $\Rightarrow$ load balancing



Meshing

Block-structured mesh

- Blocks are called slices
- Each slice is unstructured, but all are topologically identical
- Work per timestep per slice is identical $\Rightarrow$ load balancing



Solution algorithm

Problem to be solved in algebraic notation

- $\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f}$
- Mass matrix $\mathbf{M}$, stiffness matrix $\mathbf{K}$
- Displacement vector $\mathbf{u}$, sources $\mathbf{f}$, velocity $\mathbf{v} = \dot{\mathbf{u}}$, acceleration $\mathbf{a} = \ddot{\mathbf{u}}$

Three main steps in each iteration of the Newmark time loop

- Step 1: Update global displacement vector and second half-step of velocity vector using the acceleration vector from the last time step

$$\begin{aligned} \mathbf{u} &= \mathbf{u} + \Delta t \mathbf{v} + \frac{\Delta t}{2}\mathbf{a} \\ \mathbf{v} &= \mathbf{v} + \frac{\Delta t}{2}\mathbf{a} \end{aligned}$$

Solution algorithm

Problem to be solved in algebraic notation

- $\mathbf{M\ddot{u} + Ku = f}$

Three main steps in each iteration of the Newmark time loop

- Step 2: Compute new $\mathbf{Ku}$ and $\mathbf{M}$ to obtain intermediate acceleration vector (the tricky bit, called 'SEM assembly')
- Step 3: Finish computation of acceleration vector and compute new velocity vector for half the timestep (cannot be merged into steps 2 and 1 because of data dependencies)

$$\begin{aligned} \mathbf{a} &= \mathbf{M}^{-1}\mathbf{a} \\ \mathbf{v} &= \mathbf{v} + \frac{\Delta t}{2}\mathbf{a} \end{aligned}$$

SEM assembly

Most demanding step in the algorithm

- Measurements indicate up to 88% of the total runtime
- Employ 'assembly by elements' technique
- For each element, assembly process comprises two stages

First stage: All local computations

- Gather values corresponding to element GLL points from global displacement vector using global-to-local mapping
- Multiply with derivative matrix of Lagrange polynomials
- Perform numerical integration with discrete Jacobian to obtain local gradient of displacement
- Compute local components of stress tensor, multiply with Jacobian and dot with test function
- Combine everything into local acceleration vector

SEM assembly

First stage continued

- Lots and lots of computations
- Essentially straightforward computations, involves mostly many small matrix products for the three x, y, z-cutplanes
- Benefit of only doing per-element work: Cache-friendly, high data reuse, high arithmetic intensity, manual unrolling of matrix products possible, etc.

Second stage: Perform actual assembly

- Accumulate (scatter out) per-element contributions of *shared* GLL points on vertices, edges, faces into global acceleration vector
- Note: structurally identical to FEM assembly, 'just' different cubature points

GPU Implementation and MPI Parallelisation

GPU implementation issues

Steps 1 and 3 are essentially trivial

- One kernel each
- Only involve uniquely numbered global data, `axpy`-type computations
- Block/thread decomposition in CUDA can be optimised as usual
- More importantly: Memory access automatically fully coalesced into minimal amount of transactions
- Optimal bandwidth utilisation

Step 2 is tricky

- Lots of optimisations, resulting in only one kernel
- Looking at the two stages separately (separated by `__syncthreads()`)
- One CUDA thread for each of the 125 cubature points, waste three threads to end up with one thread block of 128 threads per element

GPU implementation issues

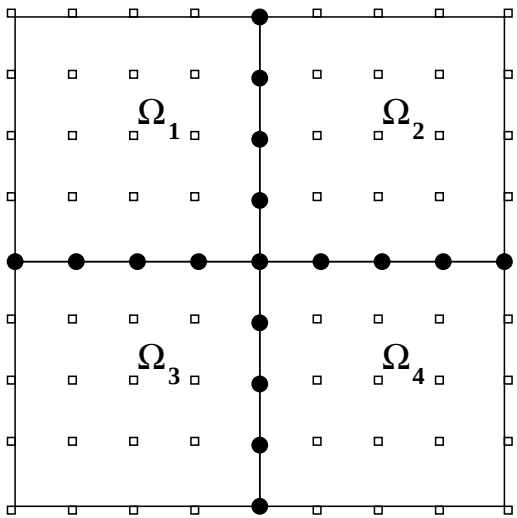
First stage: Local computations in each element

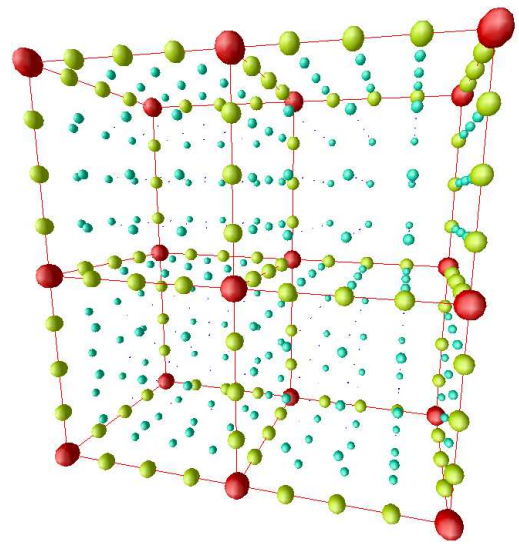
- Use shared memory for all computations
- Data layout is bank-conflict free
- Mesh is unstructured, thus indirect addressing in reading (and writing) global acceleration vector
- Cannot be fully coalesced, so use texture cache (work done on pre-Fermi hardware)
- Lots of computations inbetween global memory accesses
- Manually (and painfully) tune register and shared memory pressure so that two blocks (=elements) are concurrently active
- Together: unstructured memory accesses not too much of an issue
- Store small 5x5 derivative matrices in constant memory so that each half-warp can access the same constant in one cycle

GPU implementation issues

Second stage: Assemble of local contributions into global acceleration vector

- Shared grid points $\Rightarrow$ summation must be atomic
- 2D and 3D examples below
- Note that generally, cubature points are not evenly spaced (curvilinear mesh)

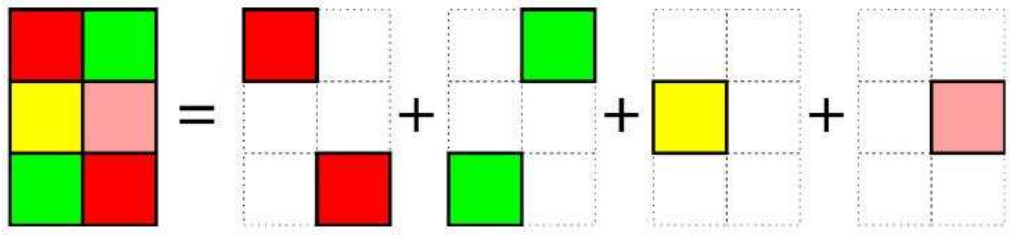




GPU implementation issues

Atomics are bad, solution: Multicolouring

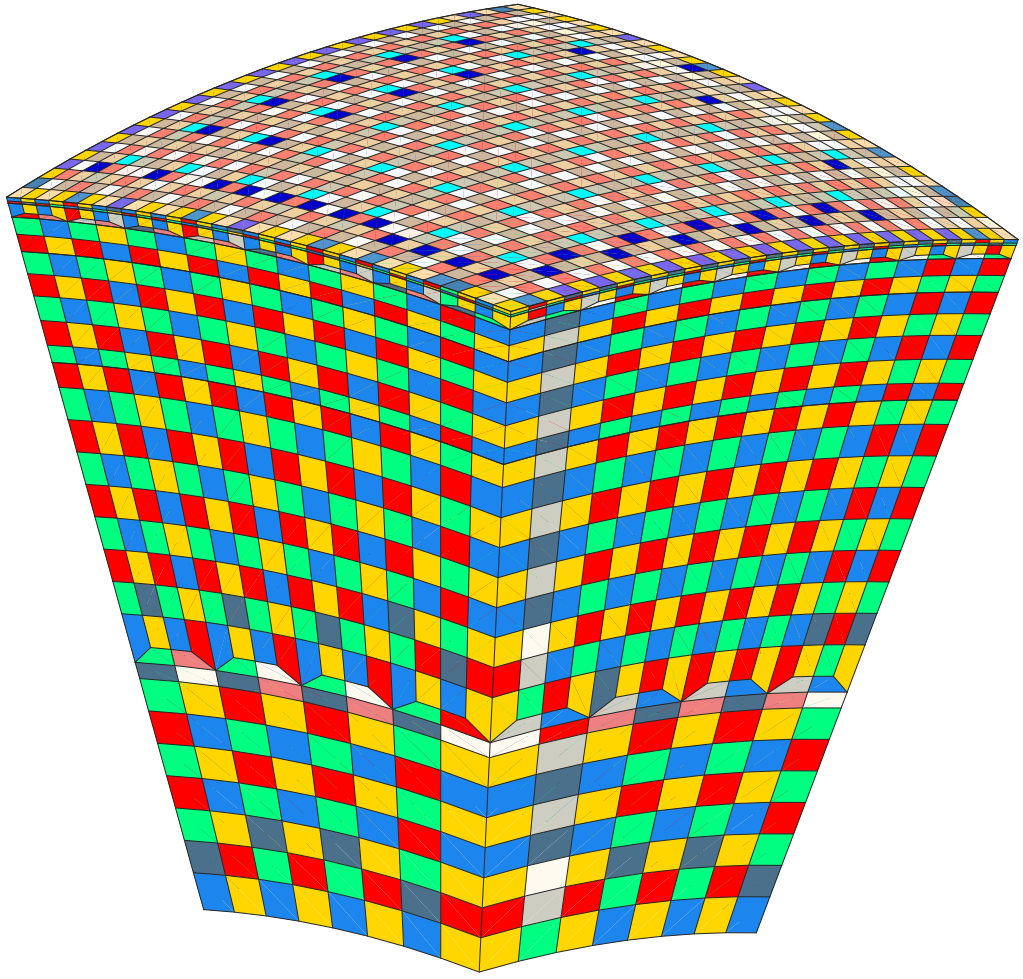
- Colour *elements* so that elements with the same colour have no common grid points and can be computed in parallel
- Sequential sweep over all colours



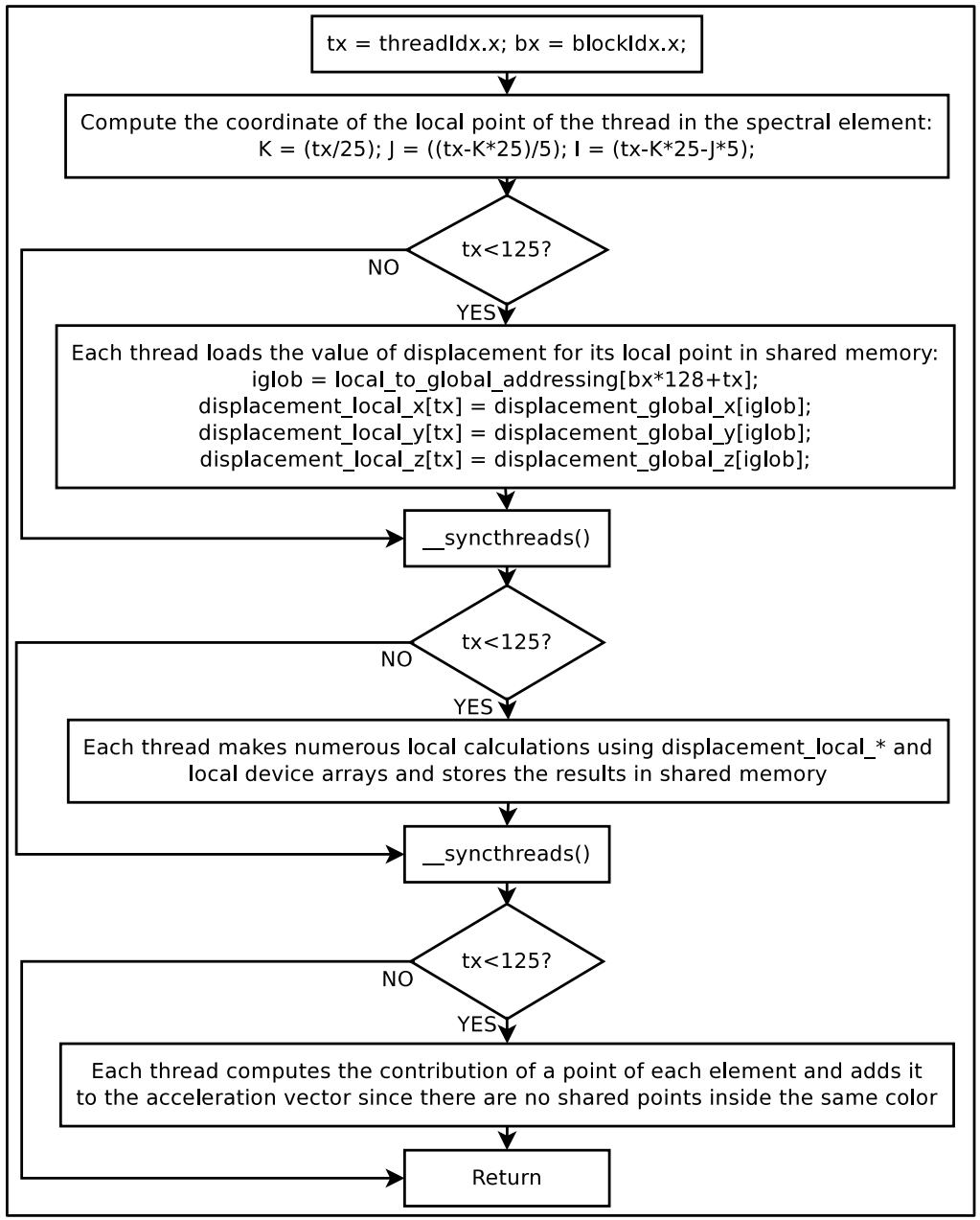
- Not to be confused with Gauß-Seidel multicolouring in Robert's talk: No loss of numerical functionality except FP noise
- Colouring is static (because mesh is static) and can be precomputed during mesh generation on the CPU
- Simple greedy algorithm to determine colouring, gives reasonably balanced results

GPU implementation issues

Coloured elements in one mesh slice



Summary of assembly process



Mapping to MPI clusters

Mapping and partitioning

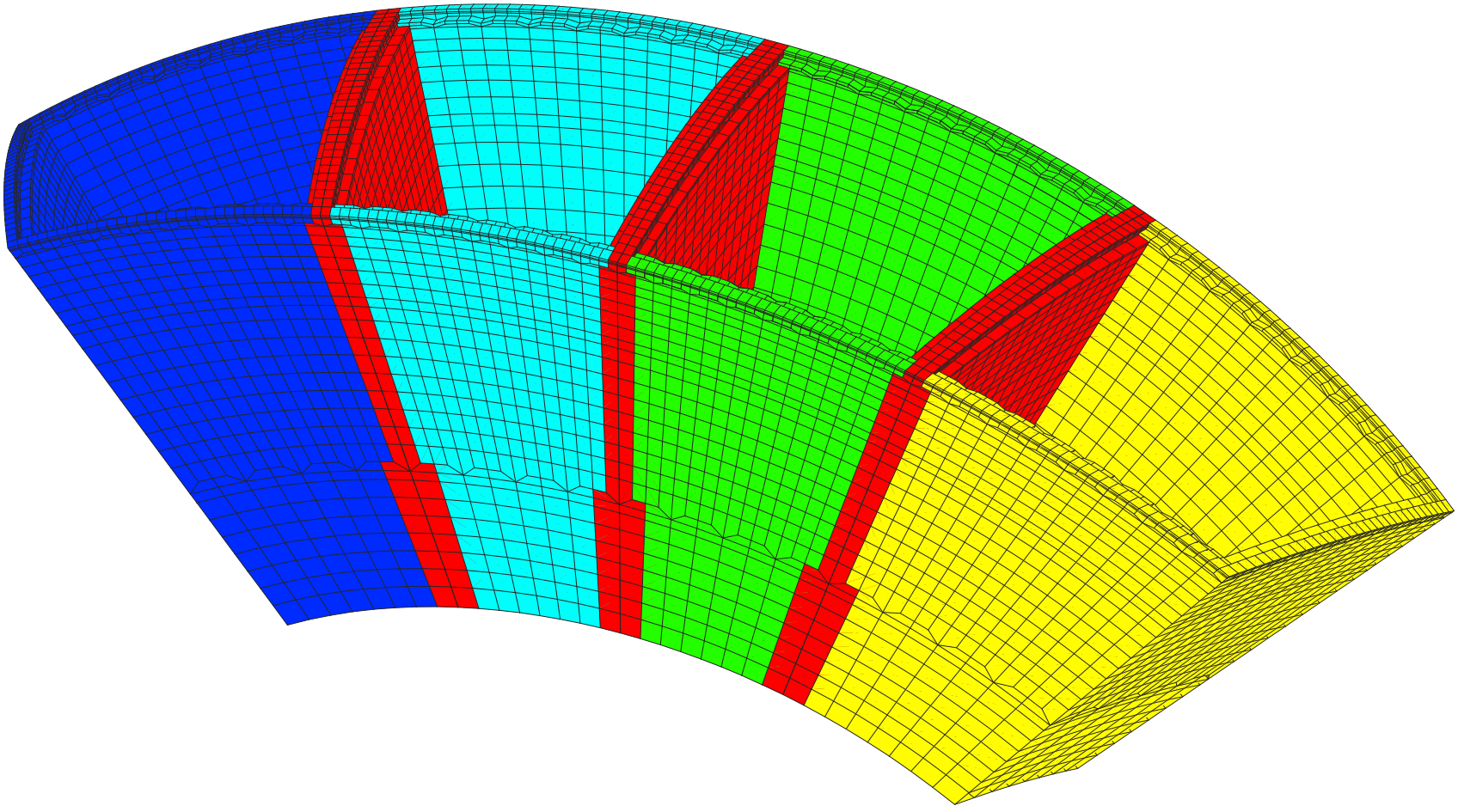
- One mesh slice (100K–500K elements) associated with one MPI rank
- Slice size always chosen to fill up available memory per CPU/GPU
- Relevant scenario: weak scaling

Overlap computation with communication (non-blocking MPI)

- Separate outer (shared) from inner elements
- Compute outer elements, send asyncroneously
- Compute inner elements, receive asyncroneously, `MPI_Wait()`
- Classical surface-to-volume issue: Balanced ratio (full overlap) of outer and inner elements if slice is large enough

Mapping to MPI clusters

Overlap computation with communication (non-blocking MPI)



GPU cluster challenges

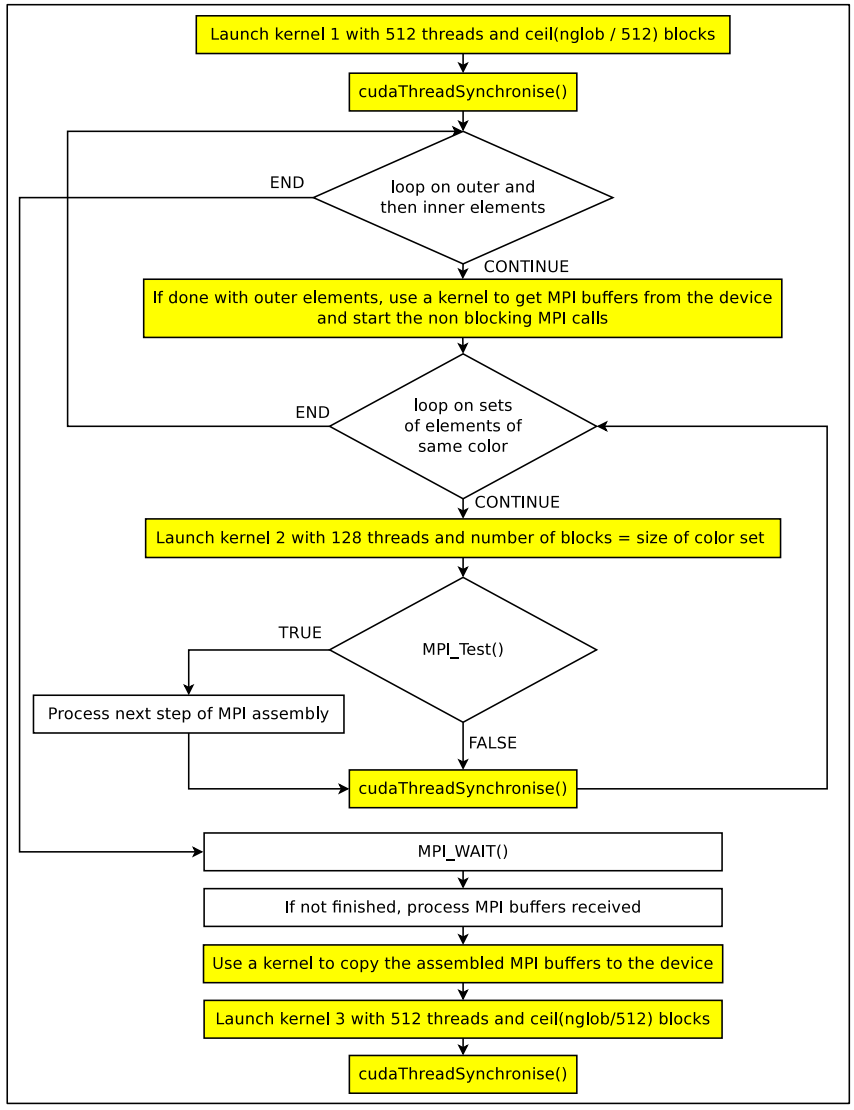
Problem: PCIe bottleneck

- MPI buffers and communication remain on CPU (story may be slightly different with new CUDA 4.x features on Fermi)
- PCIe adds extra latency and bandwidth bottleneck

Four approaches to alleviate bottleneck

- Transfer data for each cut plane separately from GPU into MPI buffer and vice versa
- Asynchroneous copy (`cudaMemcpyAsync()`, since kernel launches are async. anyway)
- Memory mapping (CUDA zero copy)
- Merge all cut planes on GPU, transfer in bulk to CPU, extract on CPU and send over interconnect to neighbours (so basically, online compression)
- Observation: Ordered by increasing performance!

Summary GPU+MPI implementation



GPU cluster challenges

'Problem': GPUs are too fast

- GPUs need higher ratio of inner to outer elements to achieve effective overlap of communication and computation
- Consequently, only GPUs with a sufficient amount of memory make sense for us
- Speedup is higher than amount of CPU cores in each node
- Hybrid CPU/GPU computations give low return on invested additional programming effort
- Hybrid OpenMP/CUDA implementation tricky to balance
- Ideal cluster for this kind of application: One GPU per CPU core
- Anyway, MPI implementations do a good job at shared memory communication (if local problems are large enough, cf. Mike Heroux's talk yesterday)

In the following

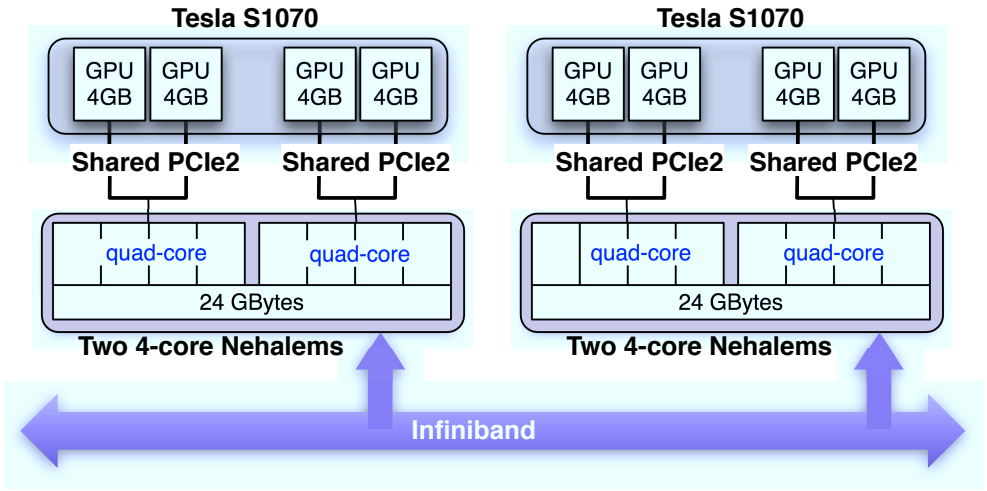
- Only CPU-only or GPU-only computations

Some Results

Testbed

Titane: Bull Novascale R422 E1 GPU cluster

- Installed at CCRT/CEA/GENCI, Bruyères-le-Châtel, France
- 48 nodes



CPU reference code is heavily optimised

- Cooperation with Barcelona Supercomputing Center
- Extensive cache optimisation using ParaVer

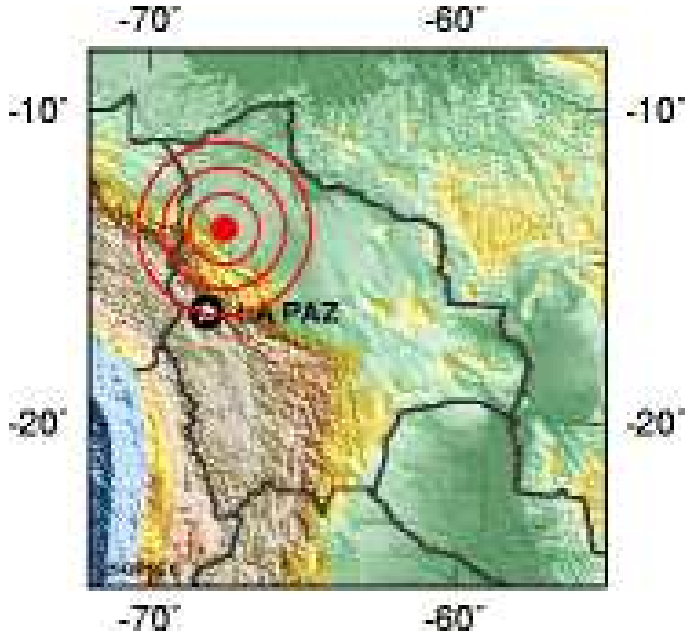
Numerical validation

Single vs. double precision

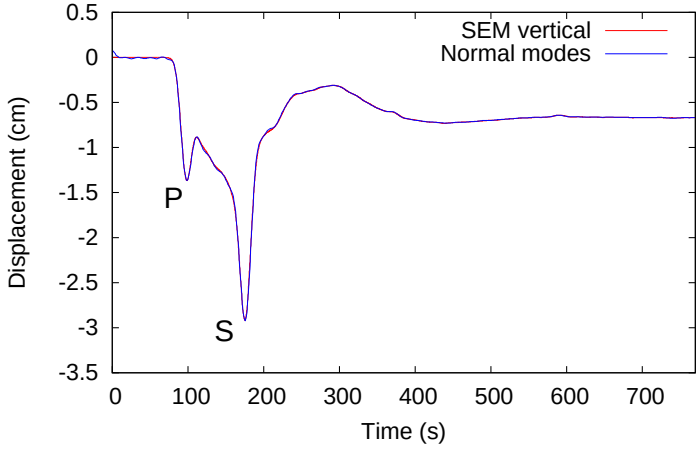
- Single precision is sufficient for this problem class
- So use single precision on CPU and GPU for a fair comparison
- Same results between single and double except minimal floating point noise

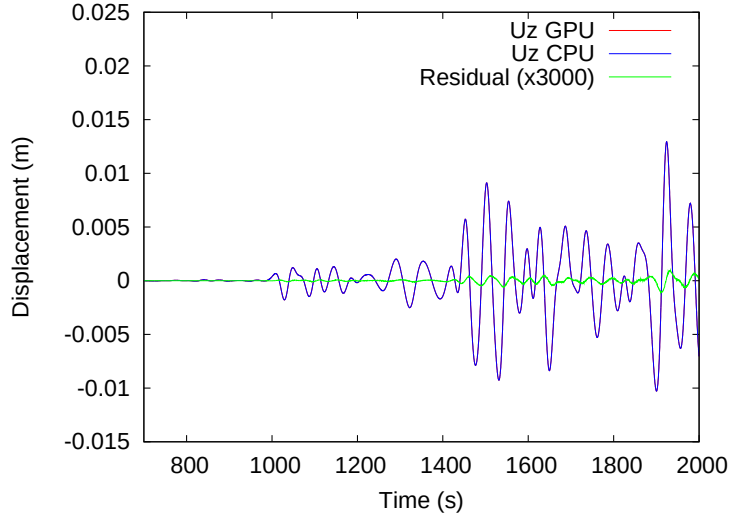
Application to a real earthquake

- Bolivia 1994, $M_w = 8.2$
- Lead to a static offset (permanent displacement) several 100 km wide
- Reference data from BANJO sensor array and quasi-analytical solution computed via summation of normal modes from sensor data



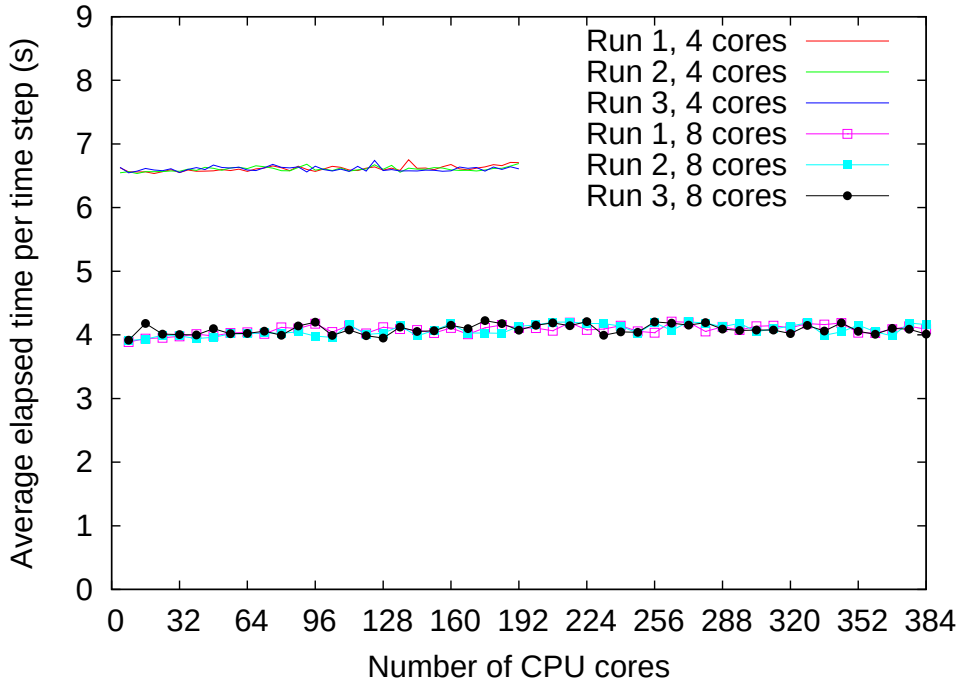
Numerical validation





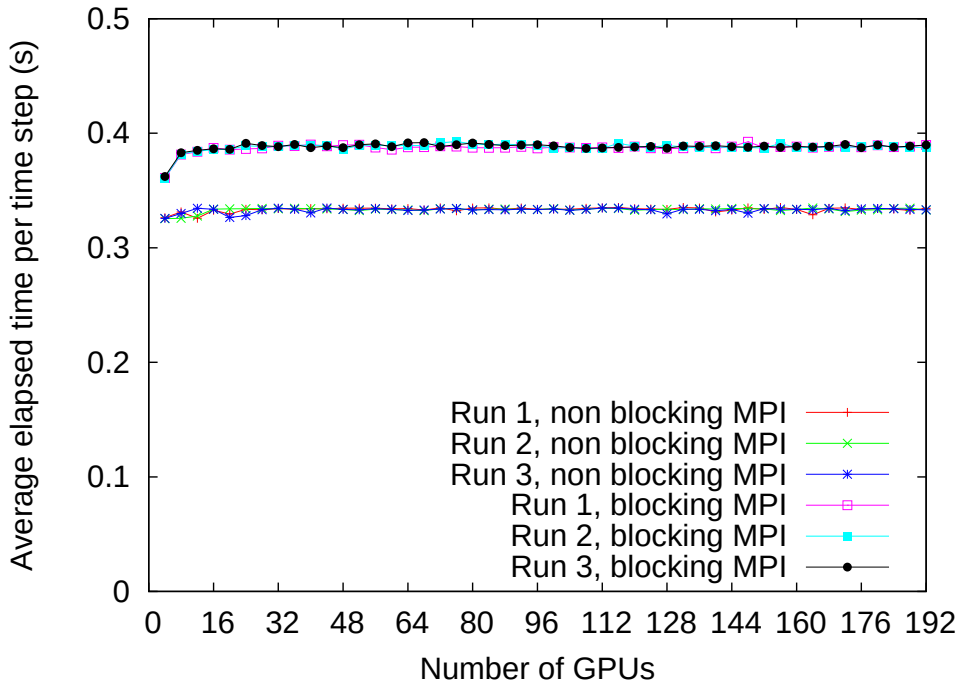
- Pressure and shear waves are accurately computed
- Static offsets are reproduced
- No difference between CPU and GPU solution
- Amplification shows that only differences are floating point noise

CPU weak and strong scaling



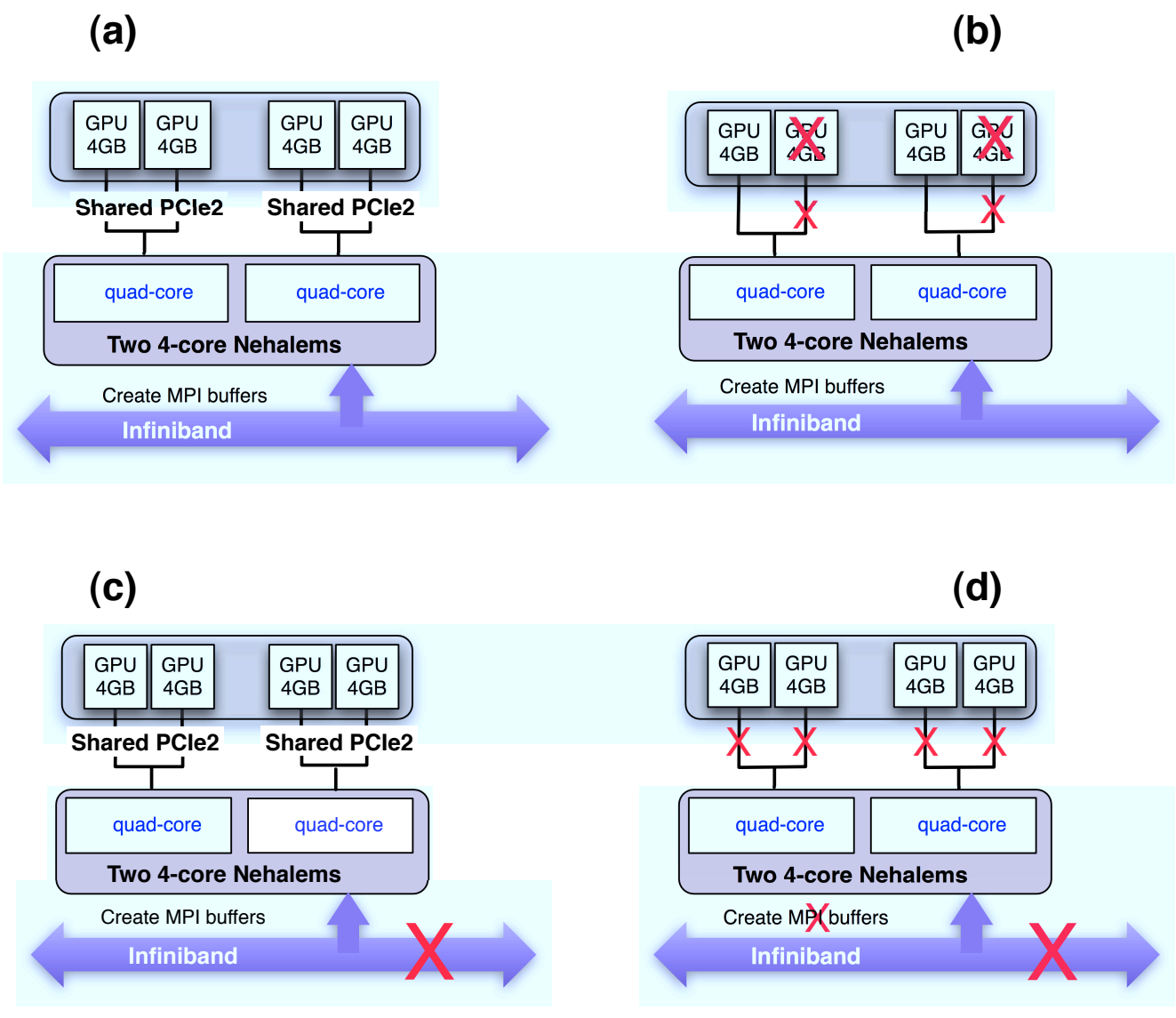
- Constant problem size per node (4x3.6 or 8x1.8 GB)
- Weak scaling excellent up to full machine (17 billion unknowns)
- 4-core version actually uses 2+2 cores per node (process pinning)
- Strong scaling only 60% due to memory bus and network contention

GPU weak scaling

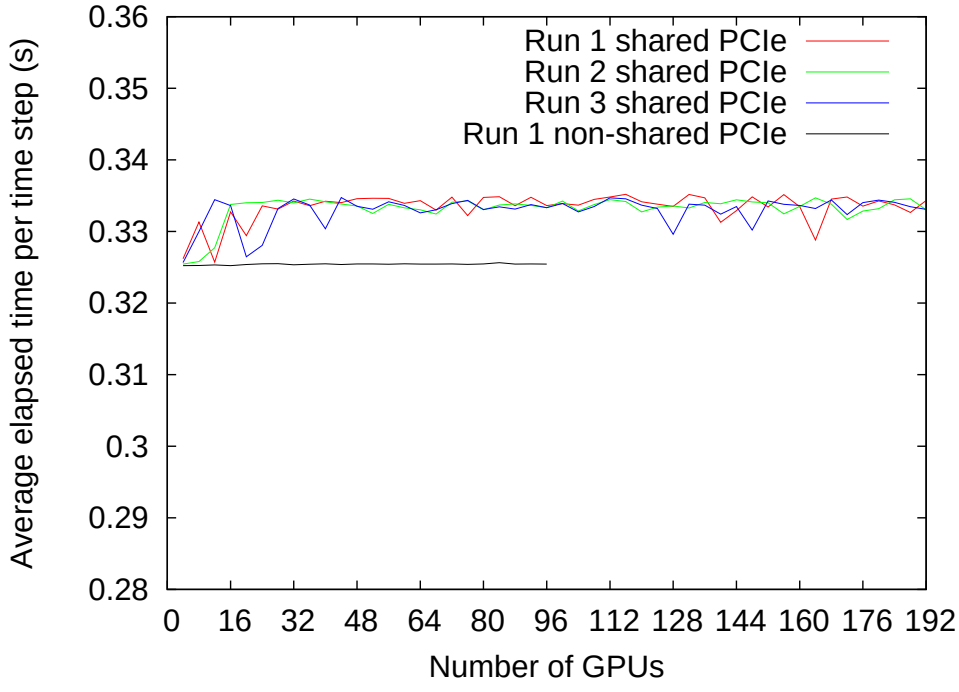


- Constant problem size per node (4x3.6 GB)
- Weak scaling excellent up to full machine (17 billion unknowns)
- Blocking MPI results in 20% slowdown

Detailed experiments

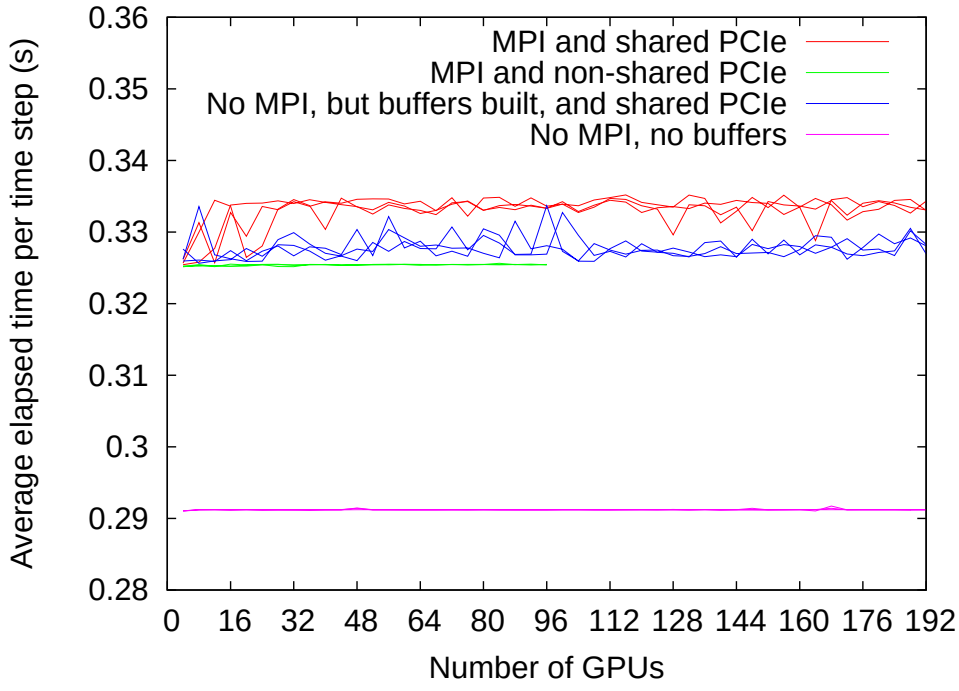


Effect of bus sharing



- Two GPUs share one PCIe bus in the Tesla S1070 architecture
- Potentially huge bottleneck?
- Results show that this is not the case (for this application)
- Introduces fluctuations and average slowdown of 3%

GPU performance breakdown



- Effect of overlapping (no MPI = replace send-receive by memset())
- Red vs. blue curve: Difference $\leq 2.8\%$, so excellent overlap
- Green vs. magenta: Total overhead of running this problem on a cluster is $\leq 12\%$ for building, processing and transmitting buffers

Summary

Summary

Excellent agreement with analytical and sensor data

- Double precision not necessary

Excellent weak scalability for full machine

- Up to 386 CPU cores and 192 GPUs
- Full CPU nodes suffer from memory bus and interconnect contention
- GPUs suffer minimally from PCIe bis sharing
- Very good overlap between computation and communication

GPU Speedup

- 25x serial
- 20.6x vs. half the cores, 12.9x vs. full nodes
- Common practice in geophysics is to load up the machine as much as possible
- GPUs are a good way to scale in the strong sense

Case Study 2:

FEAST - Finite Element Analysis and Solution Tools

Introduction and Motivation

Acknowledgements

Collaboration with

- Robert Strzodka
- FEAST group at TU Dortmund: S. Buijssen, H. Wobker, Ch. Becker, S. Turek, M. Geveler, P. Zajac, D. Ribbrock, Th. Rohkämper

Funding agencies

- German DFG and BMBF grants
- Max Planck Center for Visual Computing and Communication

Publications

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

Introduction and motivation

What happens if porting effort is too high?

- Code written in some obscure language
- Code simply too large
- Several application folks depending on one common framework (don't want to break their code and force fellow PhD students to start over)

High-level software design questions of interest

- Feasibility of partial acceleration?
- Interface design (smallest common denominator)?
- Return on investment (speedup, # of applications, coding effort)?
- GPU clusters as easy to use as conventional ones?
- Future-proof acceleration?

Introduction and motivation

Enter numerics (the more fun part)

- Existing methods often no longer hardware-compatible
- Neither want less numerical efficiency, nor less hardware efficiency
- Numerics is orthogonal dimension to pure performance and software design

Hardware-oriented numerics

- Balance these conflicting goals (want 'numerical scalability')
- Our niche: Finite Element based simulation of PDE problems
- Our prototypical implementation: FEAST

Consider short-term hardware details in actual implementations, but long-term hardware trends in the design of numerical schemes!

Grid and Matrix Structures

Flexibility $\leftrightarrow$ Performance

Grid and matrix structures

General sparse matrices (from 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)

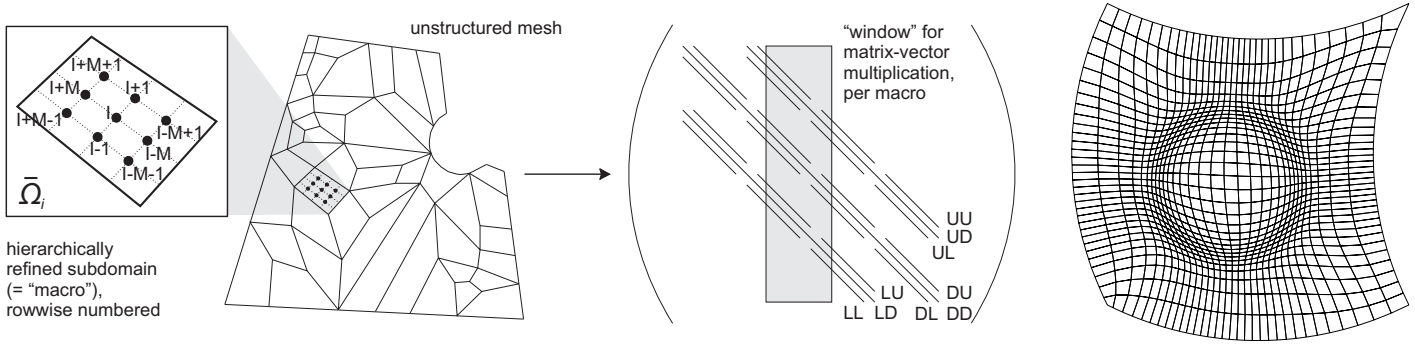
Structured matrices

- 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 (Robert)

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!
- Batch several of these into one MPI rank
- Reduce numerical linear algebra to sequences of operations on structured, local data (maximise locality intra- and inter-node)



Scalable Multigrid Solvers on GPU-enhanced Clusters

Coarse-grained parallel multigrid

Goals

- Parallel efficiency: strong and weak scalability
- Numerical scalability: convergence rates independent of problem size and partitioning (multigrid!)
- Robustness: anisotropies in mesh and differential operator (strong smoothers!)

Most important challenges

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

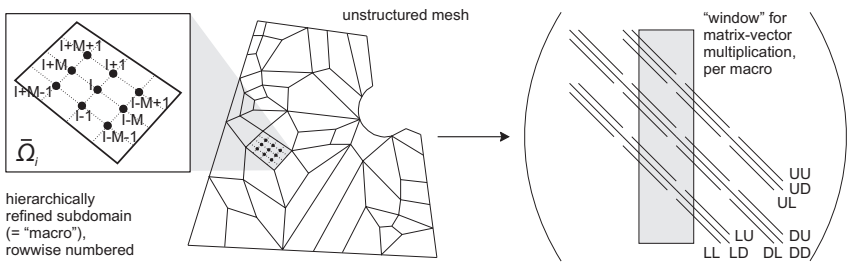
Our approach: Scalable Recursive Clustering (ScaRC)

- 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



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*

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

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

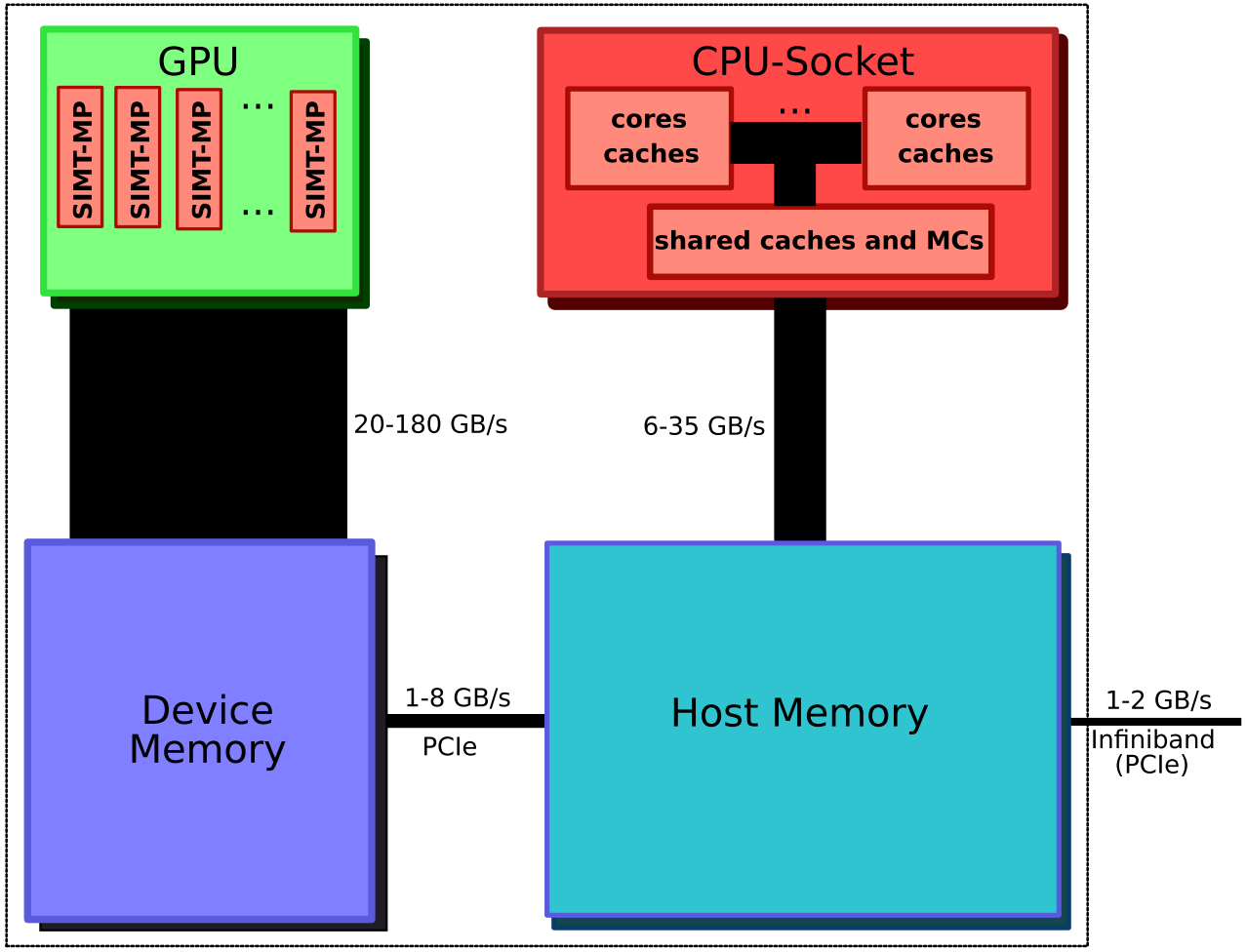
$\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ correspond to scalar (elliptic) operators

$\Rightarrow$ Tuned linear algebra **and** tuned solvers

Minimally Invasive Integration

Minimal invasive integration

Bandwidth distribution in a hybrid CPU/GPU node

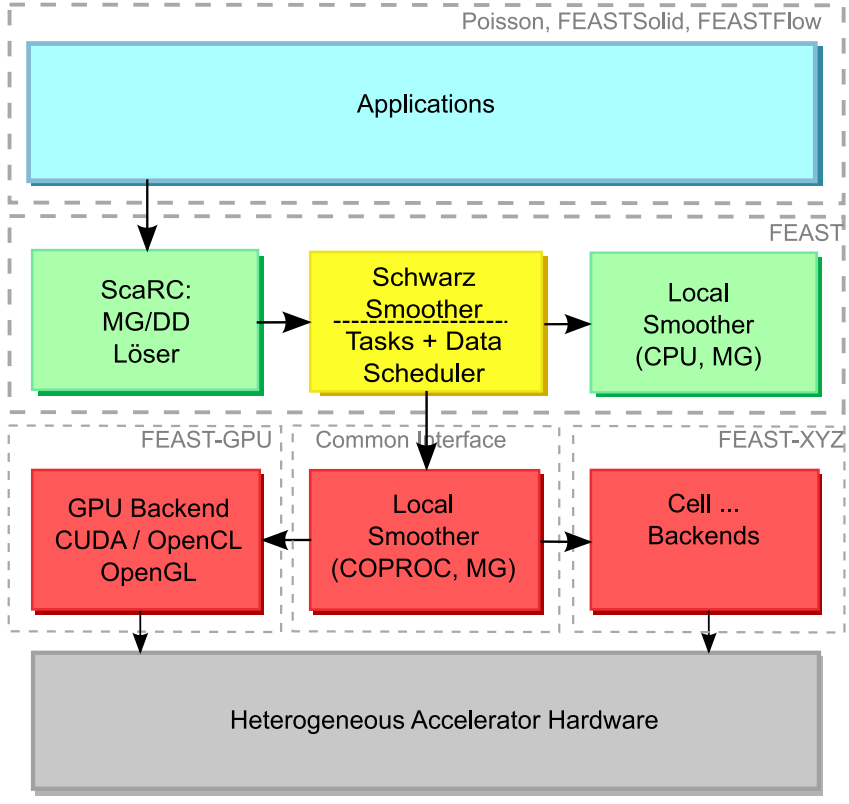


Minimally invasive integration

Guiding concept: locality

- Accelerators: most time-consuming inner component
- CPUs: outer MLDD solver (only hardware capable of MPI anyway)
- Employ mixed precision approach

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



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
- Develop on a single GPU and scale out later
- No changes to application code!
- Retain all functionality
- Do not sacrifice accuracy

Challenges

- Heterogeneous task assignment to maximise throughput
- Overlapping CPU and GPU computations, and transfers

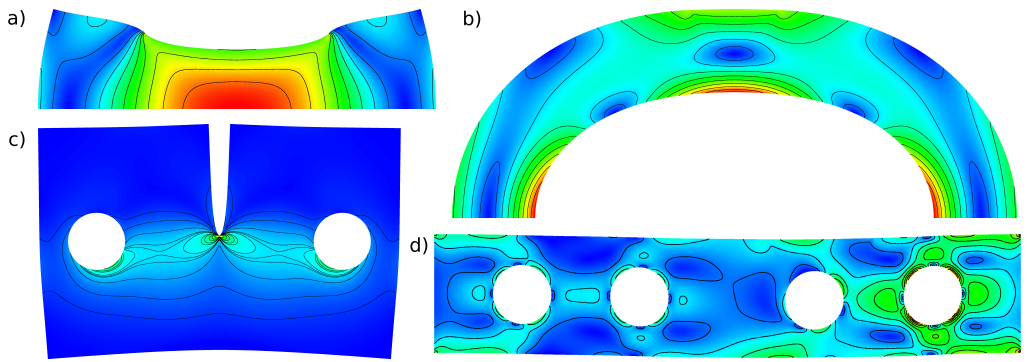
Example: Linearised Elasticity

Example: Linearised elasticity

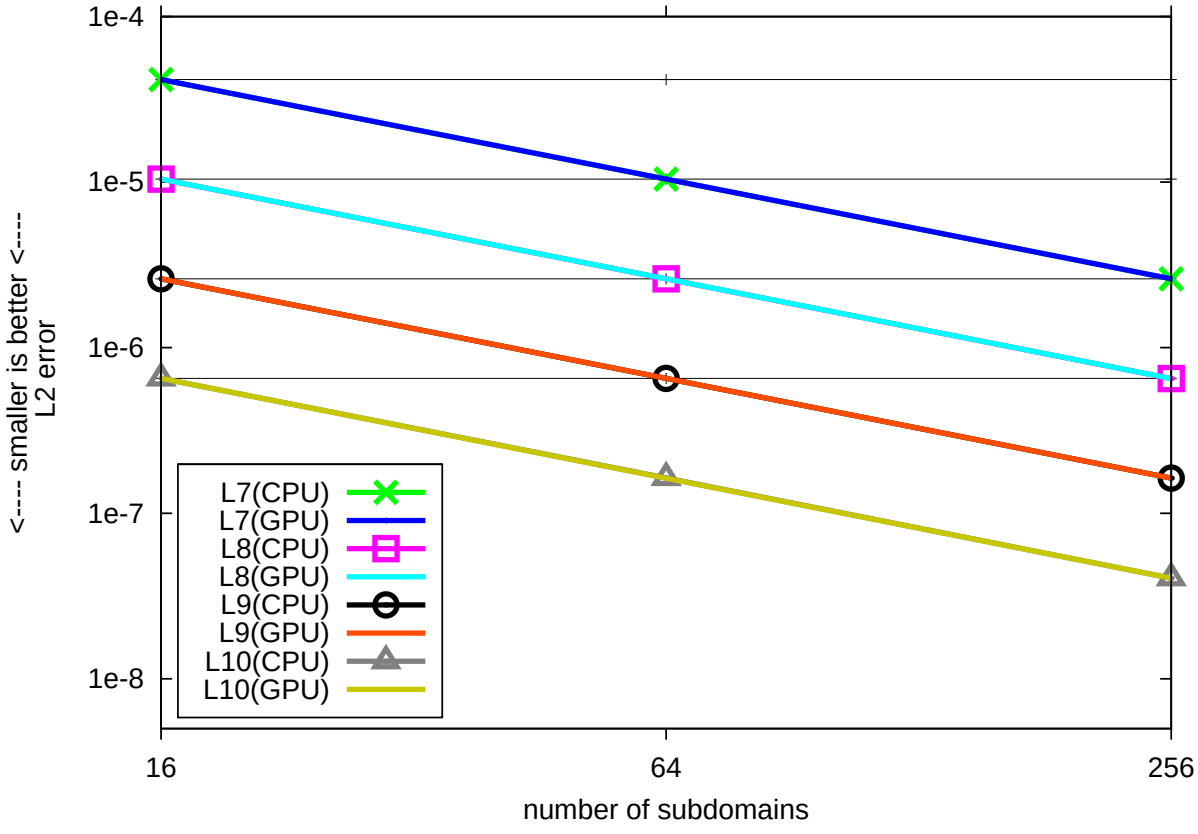
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \mathbf{f}$$

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

global multivariate BiCGStab
block-preconditioned by
Global multivariate multilevel (V 1+1)
additively smoothed (block GS) by
for all 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

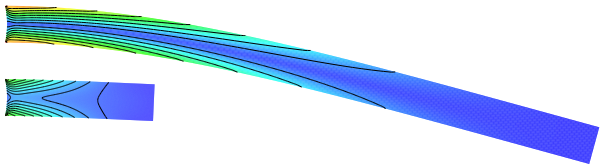


Accuracy

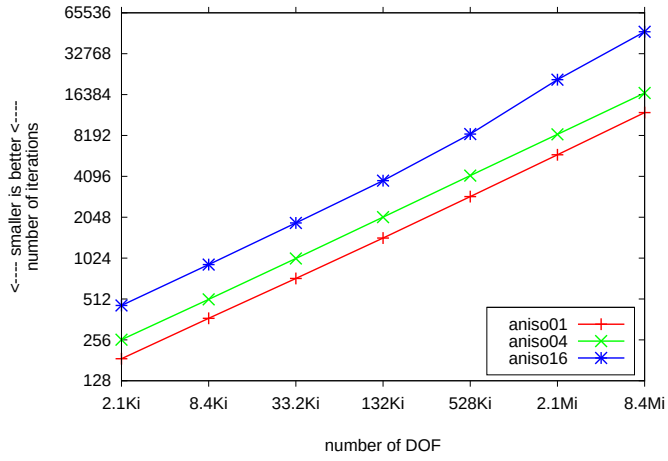


- **Same results for CPU and GPU**
- L_2 error against analytically prescribed displacements
- Tests on 32 nodes, 512 M DOF

Accuracy

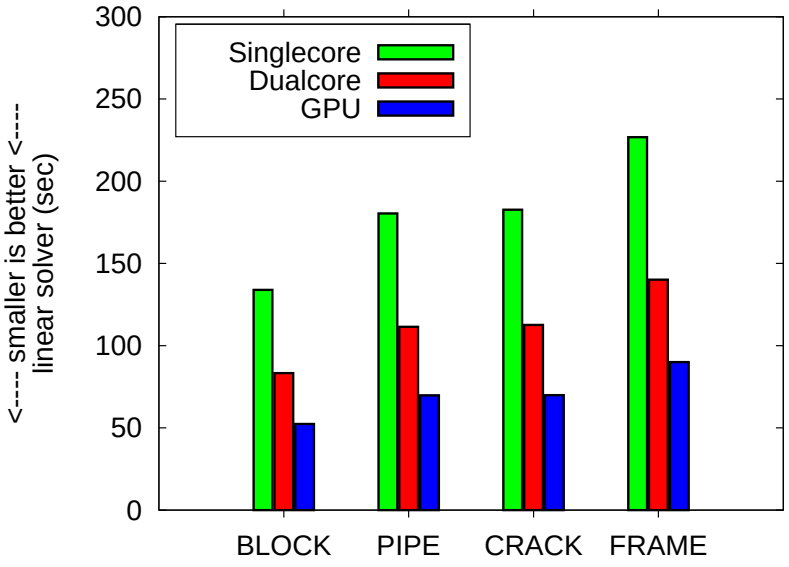


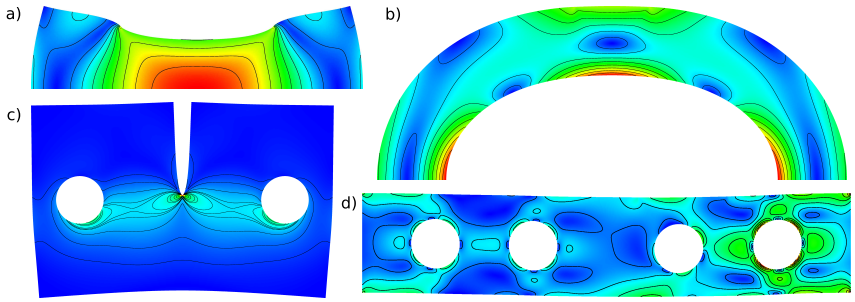
Cantilever beam, aniso 1:1, 1:4, 1:16
Hard, ill-conditioned CSM test
CG solver: no doubling of iterations
GPU-ScaRC solver: same results as CPU



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

Speedup





- 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

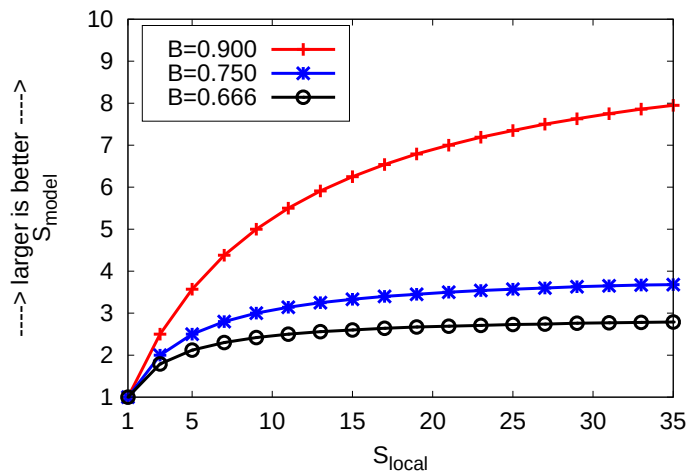
Speedup analysis

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

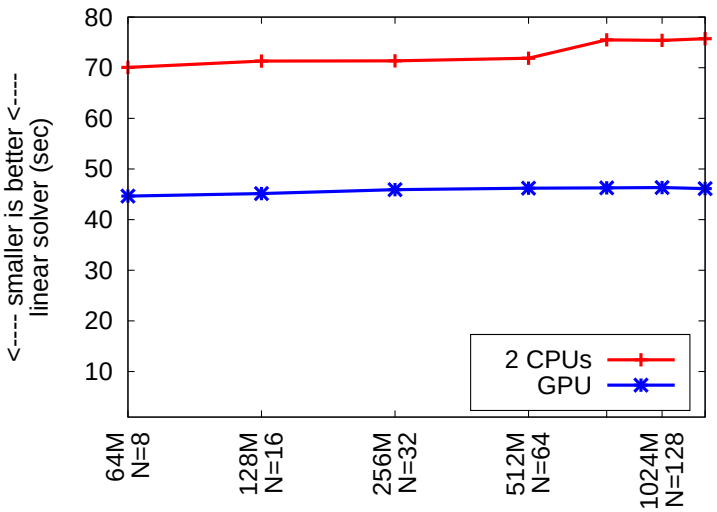
Accelerable fraction R_{acc}	66%
Local speedup S_{local}	9x
Modeled speedup S_{model}	2.5x
Measured speedup S_{total}	2.6x
Upper bound S_{max}	3x

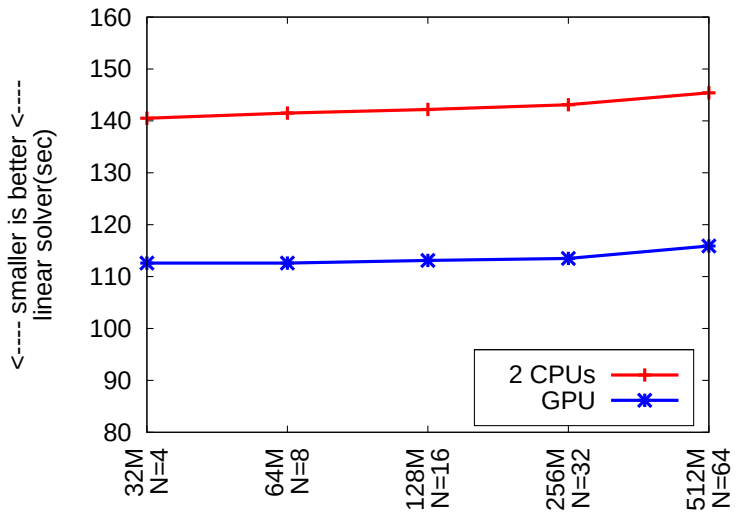


Weak scalability

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

Example:

Stationary Laminar Flow

(Navier-Stokes)

Stationary laminar flow (Navier-Stokes)

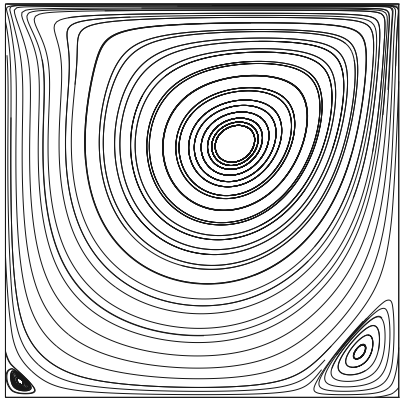
$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{B}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^T & \mathbf{B}_2^T & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2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

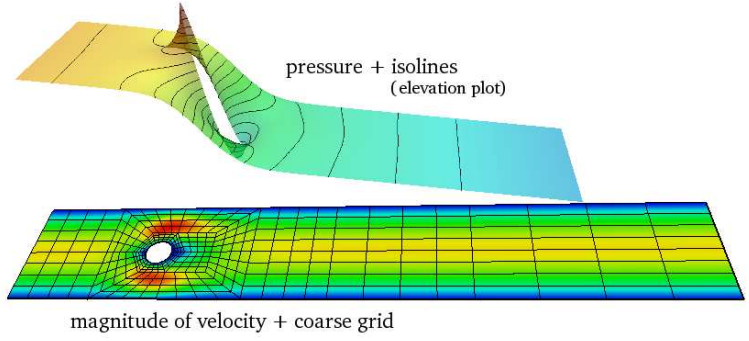
fixed point iteration
assemble linearised subproblems and solve with
global BiCGStab (reduce initial residual by 1 digit)
Block-Schurcomplement preconditioner
1) approx. solve for velocities with
global MG (V 1+0), additively smoothed by

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: $\mathbf{d}_3 = -\mathbf{d}_3 + \mathbf{B}^T(\mathbf{c}_1, \mathbf{c}_2)^T$
3) scale $\mathbf{c}_3 = (\mathbf{M}_p^L)^{-1}\mathbf{d}_3$





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

Summary

ScaRC solver scheme

- Beneficial on CPUs and GPUs
- Numerically and computationally future-proof (some odd ends still to be resolved)

Large-scale FEM solvers

- Finite Element PDE solvers
- Solid mechanics and fluid dynamics

Partial acceleration

- Very beneficial in the short term
- Amdahl's law limits achievable speedup
- Risk of losing long-term scalability?

Last slide

Bottom lines of the last 180 minutes

- Exploiting all four levels of parallelism (SIMD/SIMT $\rightarrow$ MPI)
- Parallelising seemingly sequential operations
- Optimisation for memory traffic and locality among levels
- Redesign of algorithms, balancing numerics and hardware
- Software engineering for new and legacy codes
- Scalability (weak, strong, numerical, future-proof)

More information

- `www.mpi-inf.mpg.de/~strzodka`
- `www.mathematik.tu-dortmund.de/~goeddeke`