UCHPC

UnConventional High Performance Computing for PDE

S. Turek
with support by the FEAST Group

Institut für Angewandte Mathematik, TU Dortmund
http://www.mathematik.uni-dortmund.de/LS3
http://www.featflow.de

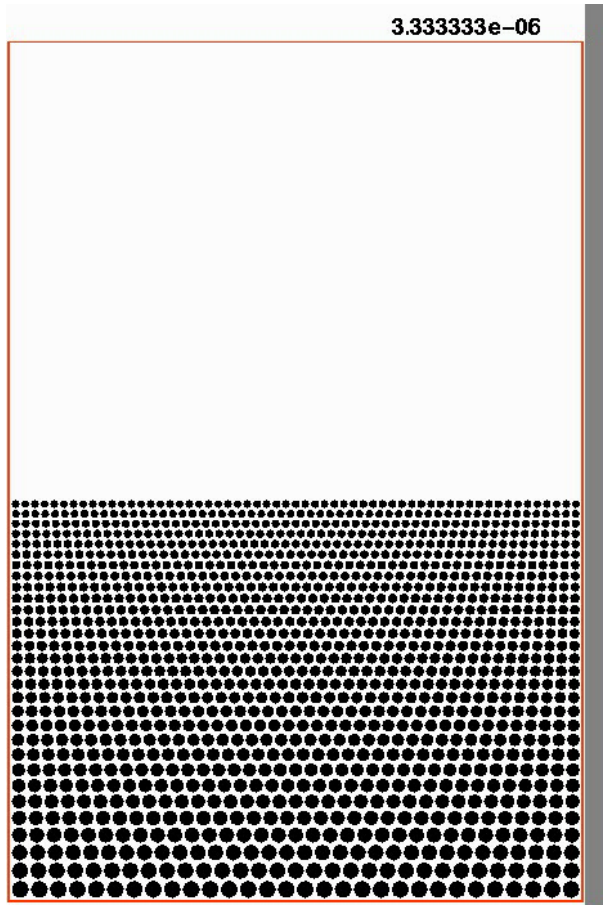
Motivation

- The ‘free ride’ is over, HPC faces a paradigm shift:
  - memory wall (in particular for sparse Linear Algebra problems)
  - physical barriers (heat, power consumption, leaking voltage)
  - **applications no longer run faster automatically on newer hardware**
- Special (heterogeneous) hardware has to be taken into account:
  - multi-core commodity CPUs - Cell BE processor - graphics cards (GPUs)
  - HPC accelerators (e.g. ClearSpeed) - reconfigurable hardware (FPGAs)
- *Rethink the way we design algorithms for these architectures when Virtual Labs for realistic applications (**Multiphase**-CFD, CSM) shall be created!*

Why Multiphase Problems

- Very flexible and efficient tools are required for a large class of important applications in chemical engineering, life science, medicine, biomechanics and more…..
- **However:** Huge dynamical systems

Special HPC Techniques required



Aim of this Talk

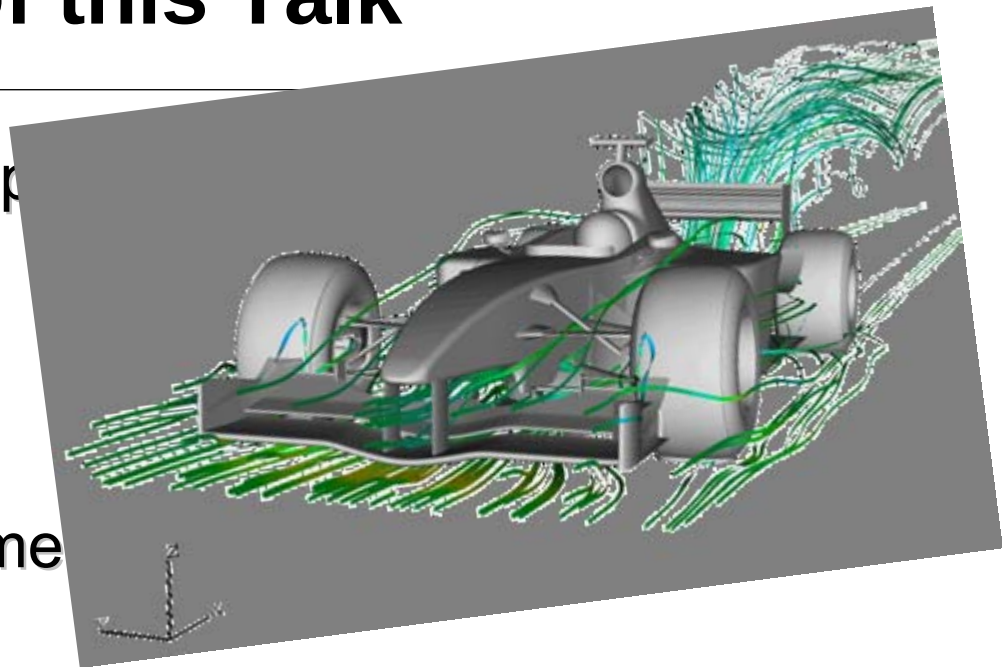
- ’High Performance Comp

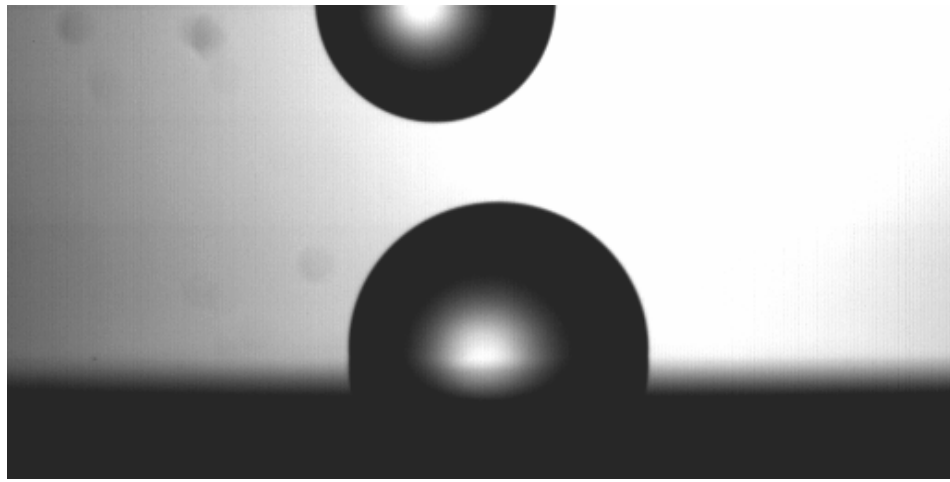
meets

- ’Hardware-Oriented Nume

on

- Unconventional Hardware





Hardware-Oriented Numerics

- What is:

  „Hardware-Oriented Numerics“ for PDE?

- It is more than *„good Numerics“* and *„good Implementation“* on High Performance Computers

- Critical quantity: **„Total Numerical Efficiency”**

Total Numerical Efficiency for PDE

- ’**High** (guaranteed) **accuracy** for user-specific quantities with minimal #d.o.f. (~ *N*) via **fast and robust solvers** – for a wide class of parameter variations – with **optimal** (~ *O*(*N*)) **numerical complexity** while exploiting a significant percentage of the **available huge sequential/ parallel GFLOP/s rates** at the same time’

- *Is it easy to achieve a high* **„Total Numerical Efficiency“**?

Example: Fast PDE Solvers

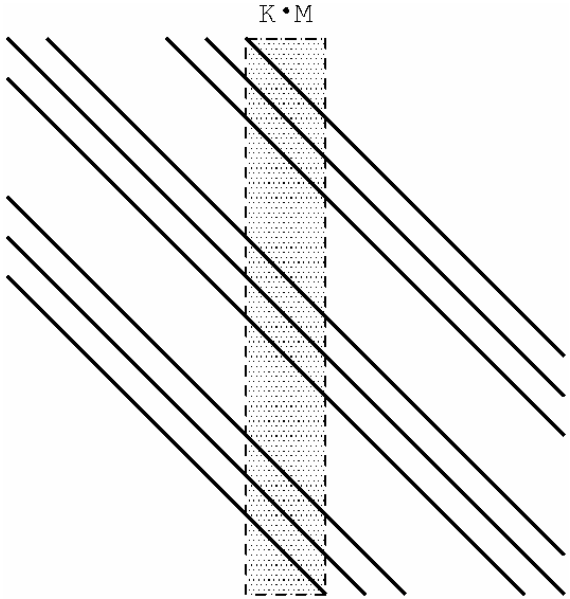
- **Fast Multilevel Methods as general philosophy**
  - ’Optimized’ versions for scalar PDE problems (≈ Poisson problems) on general meshes should require 100-1000 FLOPs per unknown (in contrast to LAPACK for dense matrices)

- Problem size 10^6: Much less than 1 sec on PC (???)
- Problem size 10^{12}: Less than 1 sec on PFLOP/s computer

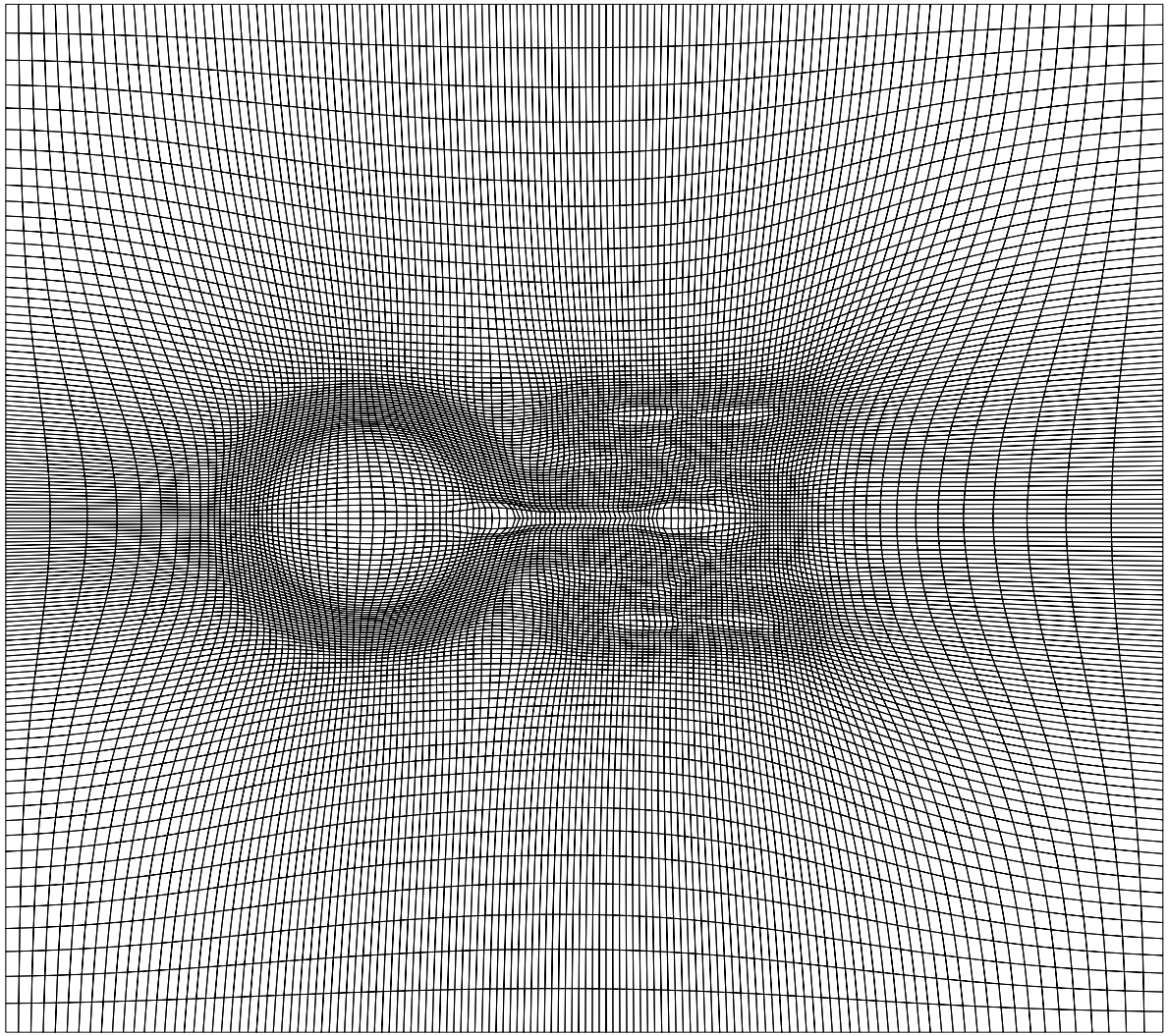
- **’Criterion’ for HPC/Petascale Computing**

Main Component: ‘Sparse‘ MV Application

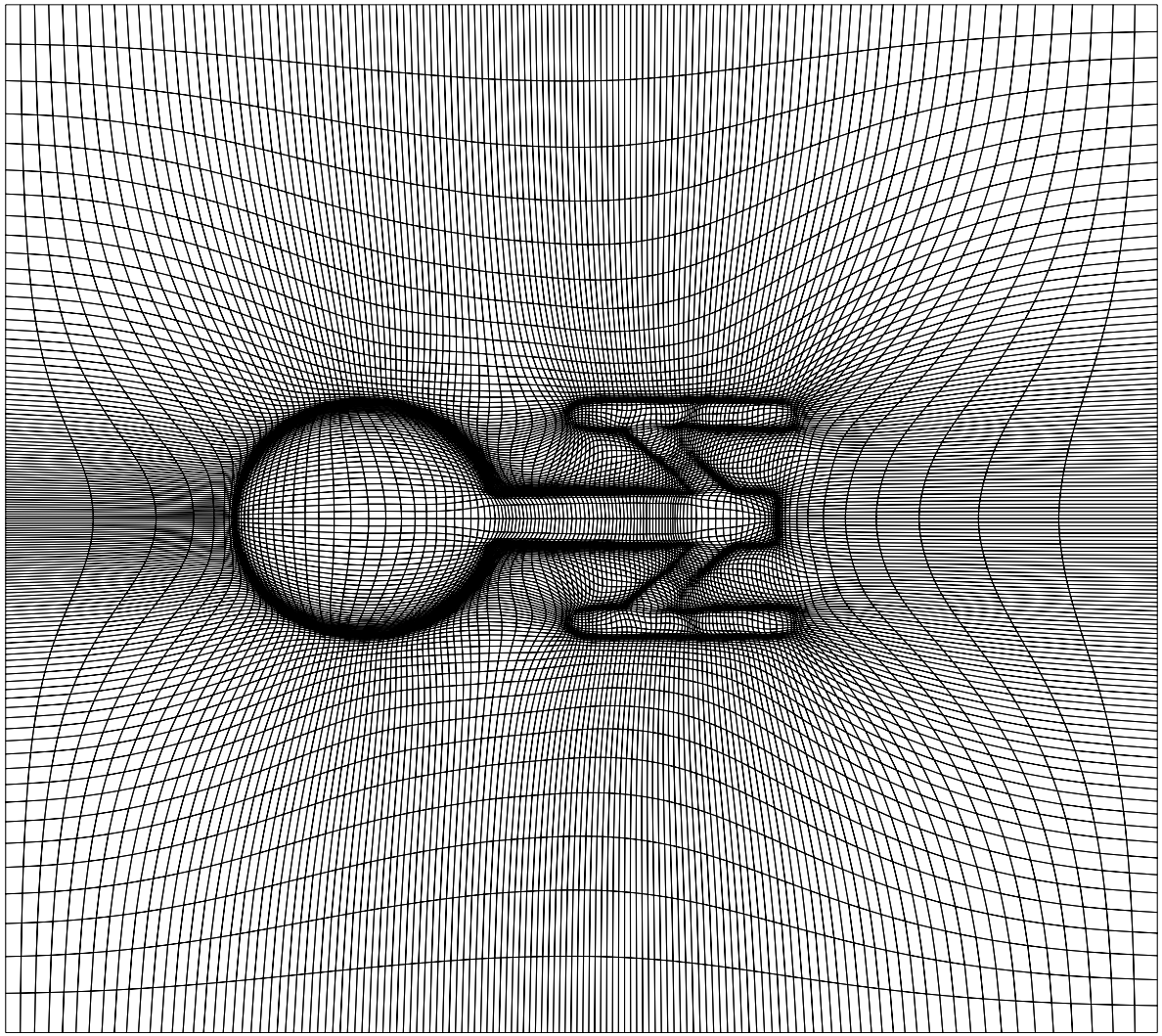
- Sparse **Matrix-Vector techniques** (‚indexed DAXPY‘)
  - DO 10 IROW=1,N
  - DO 10 ICOL=KLD(IROW),KLD(IROW+1)-1
  - 10 Y(IROW)=DA(ICOL)*X(KCOL(ICOL))+Y(IROW)
- Sparse Banded **MV techniques** on **generalized TP grids**



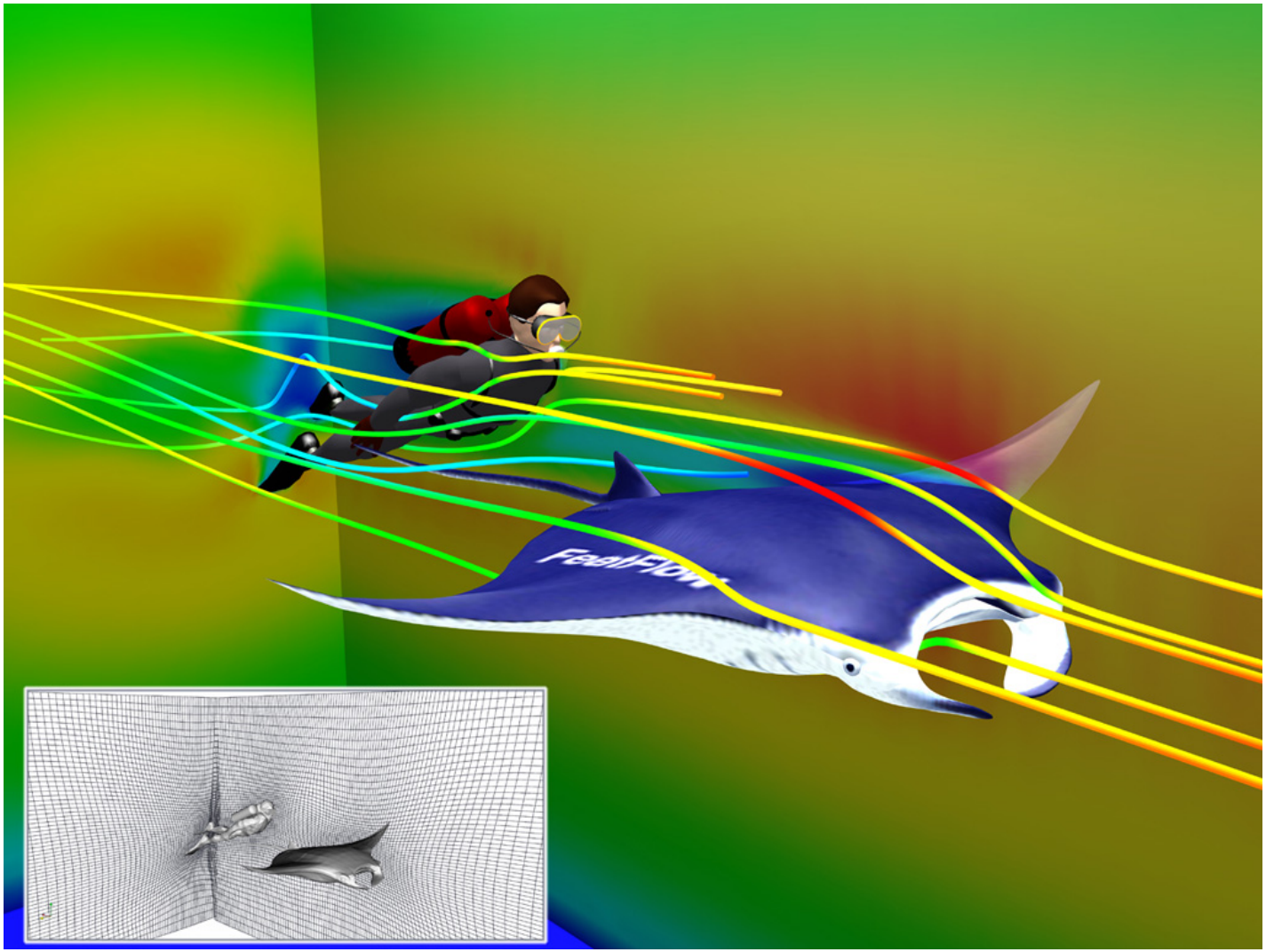
Generalized Tensorproduct Meshes



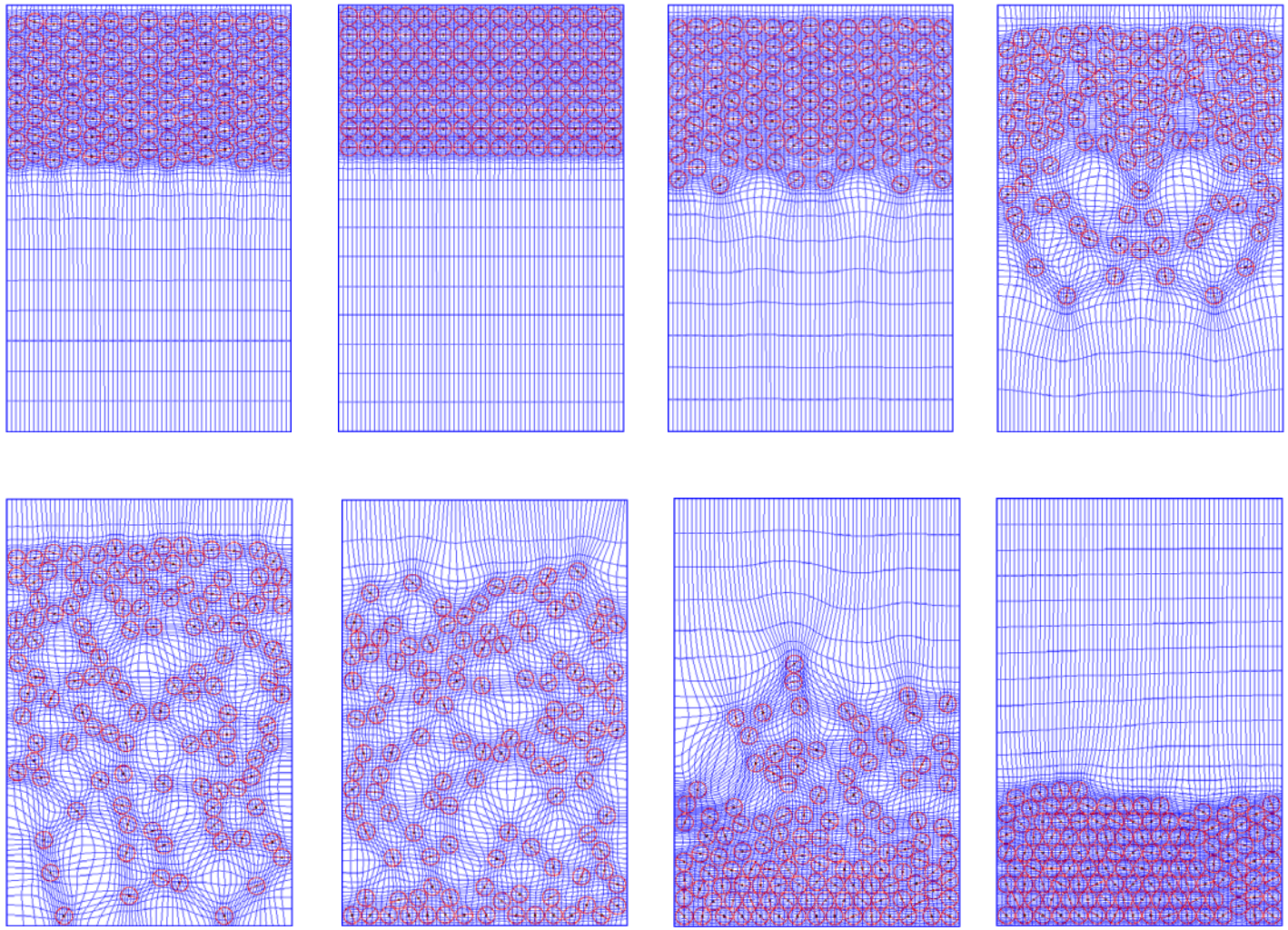
Generalized Tensorproduct Meshes



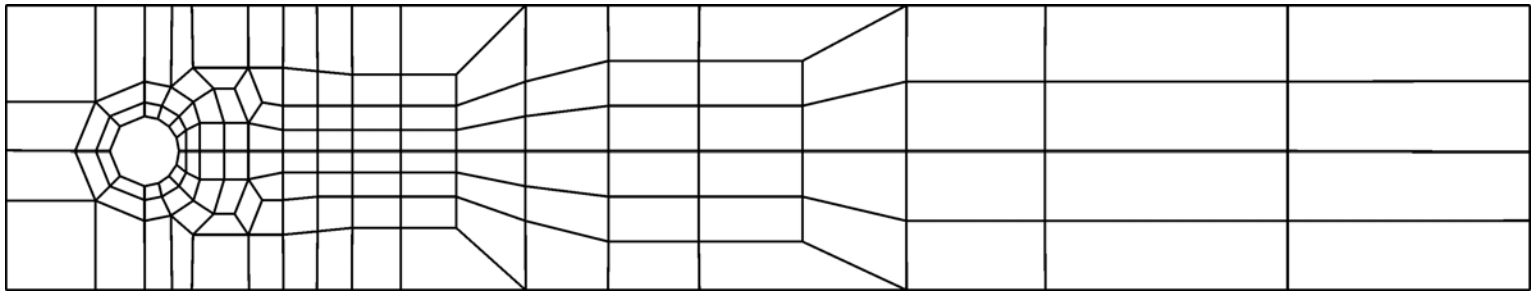
Generalized Tensorproduct Meshes



Generalized Tensorproduct Meshes



Single Processor Performance



- **Sparse MV** multiplication in (sequential) **FEATFLOW**:

Computer	# NEL	Q1-CM	Q1-TL	Q1-STO	Q2-CM	Q2-TL	Q2-STO
AMD Opteron (2600 MHz) ‚Opt. 252‘	520	322	322	**322**	335	349	**327**
	2,080	304	302	**304**	223	227	**193**
	8,320	241	241	**222**	210	202	**165**
	33,280	180	174	**151**	198	169	**91**
	133,120	173	152	**85**	192	157	**58**
	532,480	160	140	**59**	187	154	**51**

Only 50 MFLOP/s?!

Single Processor Performance

- **'Generalized Tensorproduct' meshes**

2D case	NEQ	**ROW (STO)**	**SBB-V**	**SBB-C**	**MGTRI-V**	**MGTRI-C**
AMD Opteron (2600 MHz) ‚Opt. 252'	65^2	2172 (633)	1806	3334	1541	2086
	257^2	574 (150)	627	2353	751	1423
	1025^2	**300 (64)**	**570**	**1774**	**538**	**943**
IBM POWER4 (1700 MHz) **‚JUMP'**	65^2	1521 (845)	2064	3612	906	1071
	257^2	943 (244)	896	2896	711	962
	1025^2	**343 (51)**	**456**	**1916**	**438**	**718**

- Sparse **MV techniques** (ROW/STO)
  - MFLOP/s rates vs. 'Peak Performance', problem size + numbering ???
  - **Local Adaptivity !!!**
- Sparse Banded BLAS **MV techniques** (SBB) + **MGTRI**
  - 'Supercomputing' (up to 4 GFLOP/s) vs. FEM for complex domains ???

Single Processor Performance

- **Vectorization**

2D case	NEQ	ROW (STO)	SBB-V	SBB-C	MGTRI-V	MGTRI-C
NEC SX-8	65^2	5070 (1521)	3611	3768	1112	1061
(2000 MHz)	257^2	5283 (1321)	6278	8363	1535	1543
‚Vector‘	1025^2	**5603 (1293)**	**7977**	**15970**	**1918**	**2053**

- Necessary: Development of ’new’ methods

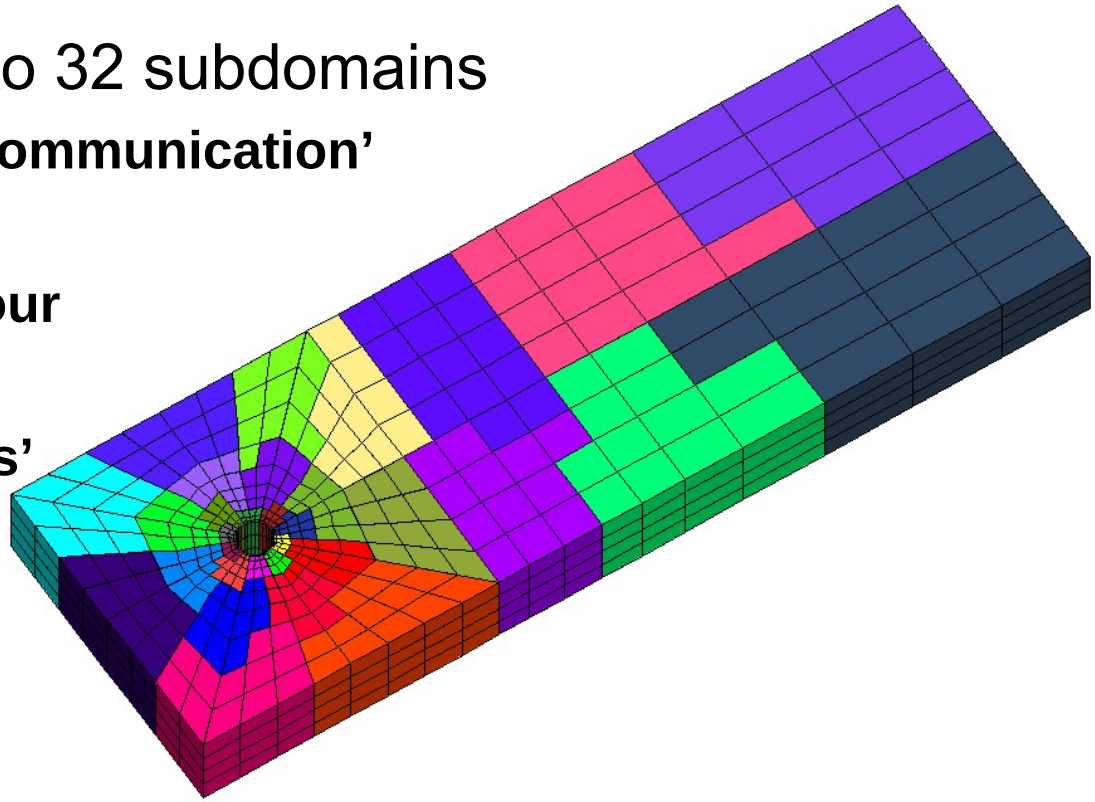
- **„Cyclic Reduction“ preconditioner**
- **„SPAI“ preconditioner (~ pure MV multiplication)**

Summary

- ’It is non-trivial to reach **Single Processor Peak Performance** with modern (= high numerical efficiency) PDE tools’
- ’Memory-intensive data/matrix/solver structures?’
- ’**Parallel Peak Performance** with modern Numerics even harder …’

Parallel Performance

- Partitioned mesh into 32 subdomains
  - **'Problems due to communication'**
  - **'Numerical behaviour vs. anisotropic meshes'**



	1 P.	2 P.	4 P.	8 P.	16 P.	32 P.	64 P.
%Comm.	10%	24%	36%	45%	47%	55%	56%
# PPP-IT	2.2	3.0	3.9	4.9	5.2	5.7	6.2

Summary

- ’Special requirements for numerical and algorithmic approaches in correspondance to modern hardware’

- **’Hardware-Oriented Numerics for PDE’**

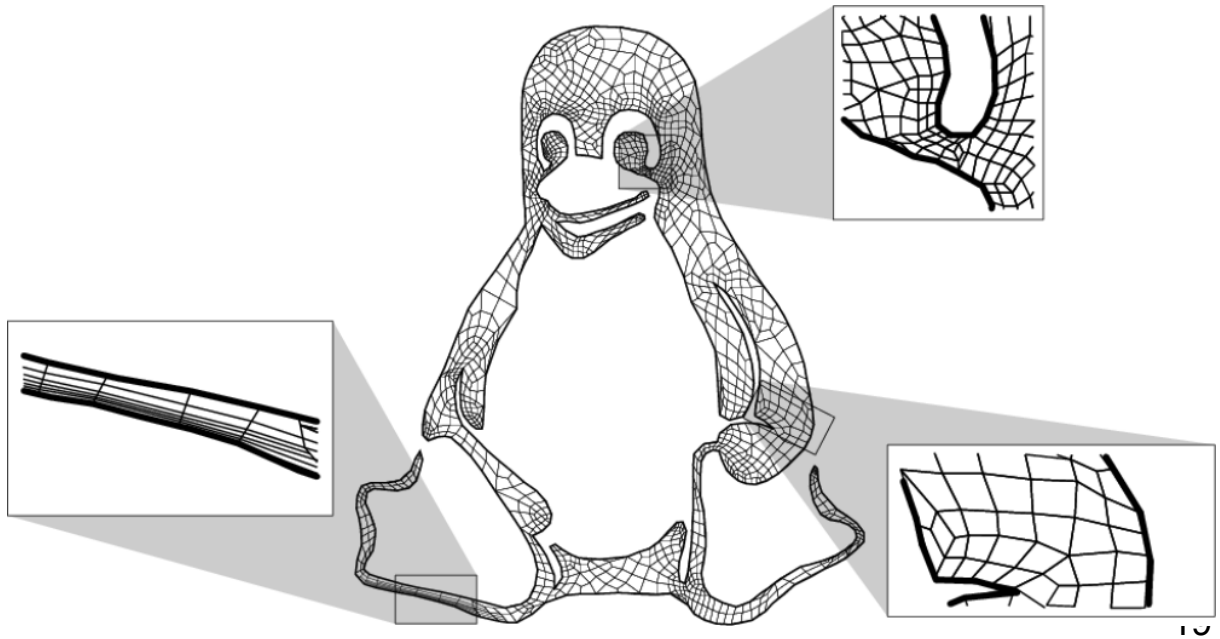
FEAST Project

Main Philosophy of FEAST

- ScaRC solver: Combine advantages of (parallel) domain decomposition and multigrid methods
- Exploit structured subdomains for high efficiency
- Hide anisotropies locally to increase robustness
- Globally unstructured – locally structured
- Recursive solution: Outer global multigrid smoother with local multigrid on the

  refined macros
- Low communication

  overhead



Open Algorithmic Problems

- **Adaptive remeshing?**
  - degree of macro-refinement and/or deformation?
  - h-p-r-refinement? 'When to do what' decision?
- **Load balancing?**
  - due to 'total CPU time per accuracy per processor'?
  - dynamical a posteriori process?
- **(Recursive) solver expert system?**
  - numerical + computational a priori knowledge?

Level	#unknowns	Computer.	63p		127p	
			sec	MF/s	sec	MF/s
8	90.317.056	Power4	170,4	5.252	116,4	7.698
		NEC	156,7	5.376	91,9	9.167
		Opteron	65,2	13.412	40,5	21.586
9	361.120.256	Power4	-	-	269,6	11.912
		NEC	222,8	13.511	128,0	23.519
		Opteron	192,6	16.299	104,7	29.974
10	**1.444.185.088**	Power4	-	-	-	-
		NEC	595,0	22.328	364,9	36.415
		Opteron	-	-	615,6	25.637

(Preliminary) Conclusions

- **Numerical efficiency?**
  → OK
- **Parallel efficiency?**
  → (OK)
- **Single processor efficiency?**
  → almost OK for CPU
- **„Peak“ efficiency?**
  → NO
  → **Special GPU/CELL/FPGA-based FEM Co-Processors**

UnConventional HPC

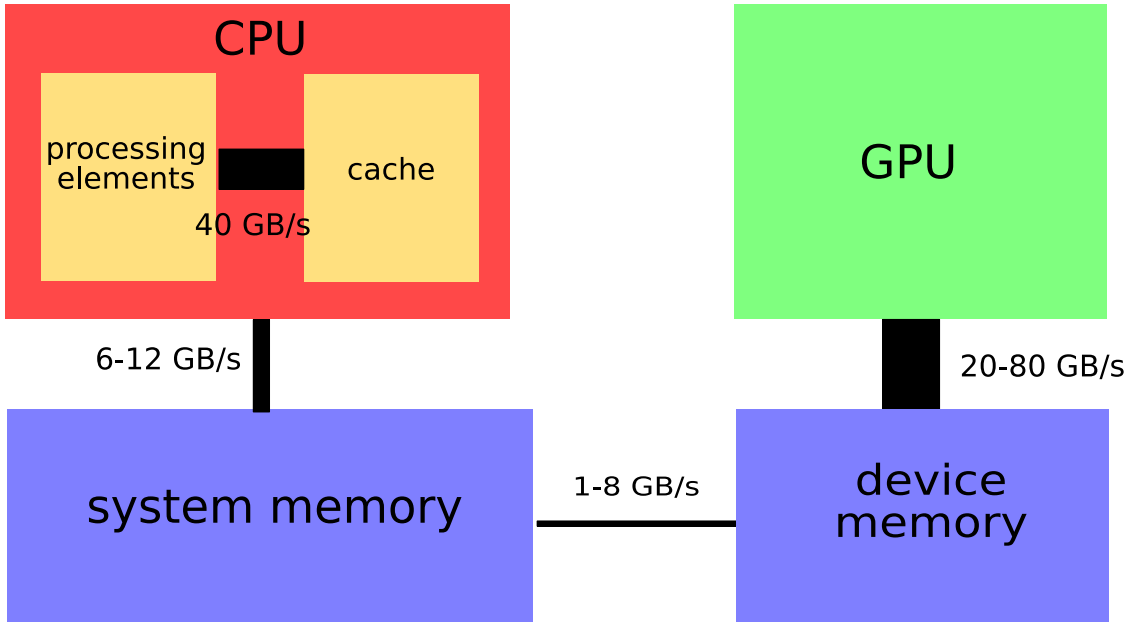
- Cell multicore processor (PS3), 7 synergetic processing units @ 3.2 GHz, 218 GFLOP/s Memory clocked @ 3.2 GHz

- CellGraphics Processor: 128 parallel Scalar processors @ 1.35 GHz, 900 MHz GDDR3 memory (86.4 GB/s) ≈ 500 GFLOP/s

UnConventional High Performance Computing (UCHPC)

Bandwidth, Bandwidth, Bandwidth…

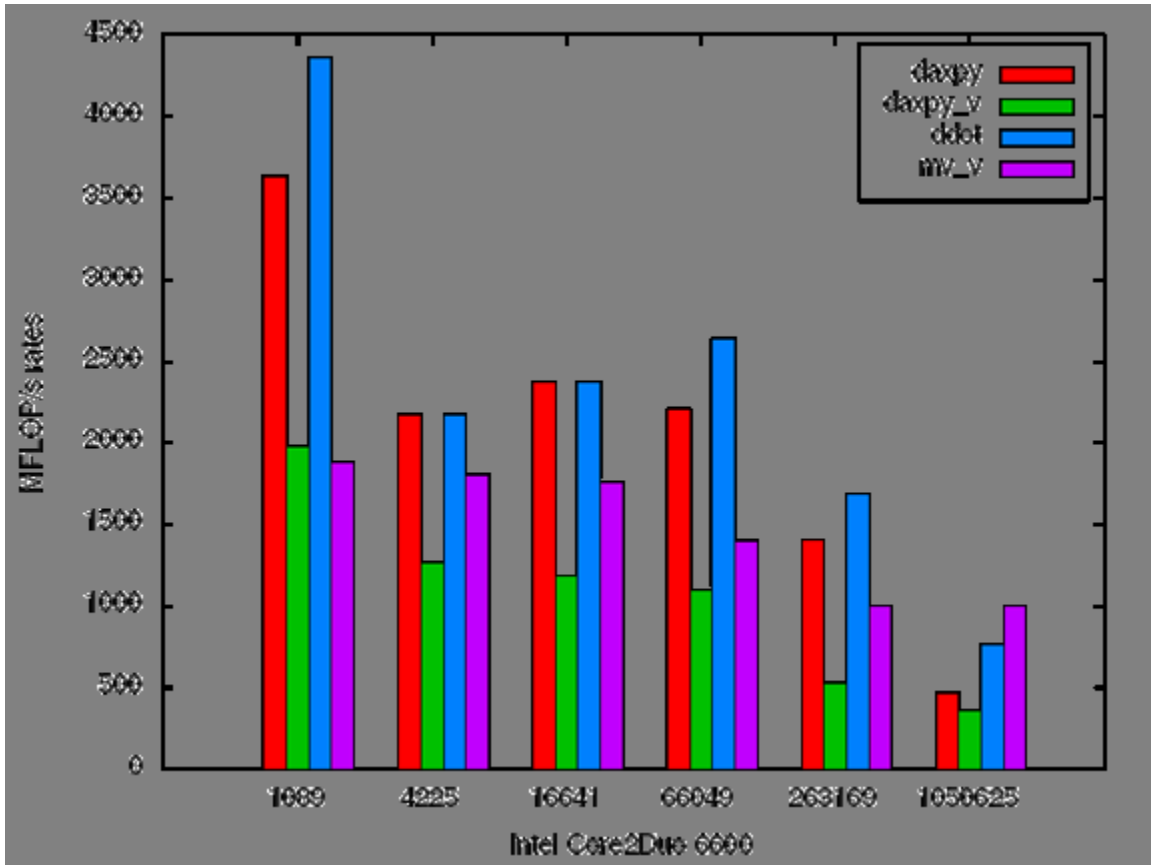
- FEM codes are 95% memory bounded, **bandwidth** is the crucial factor for performance



- GPUs offer superior bandwidth, are readily available, fast, cheap, ..., in short, seem like an ideal candidate to improve commodity based clusters

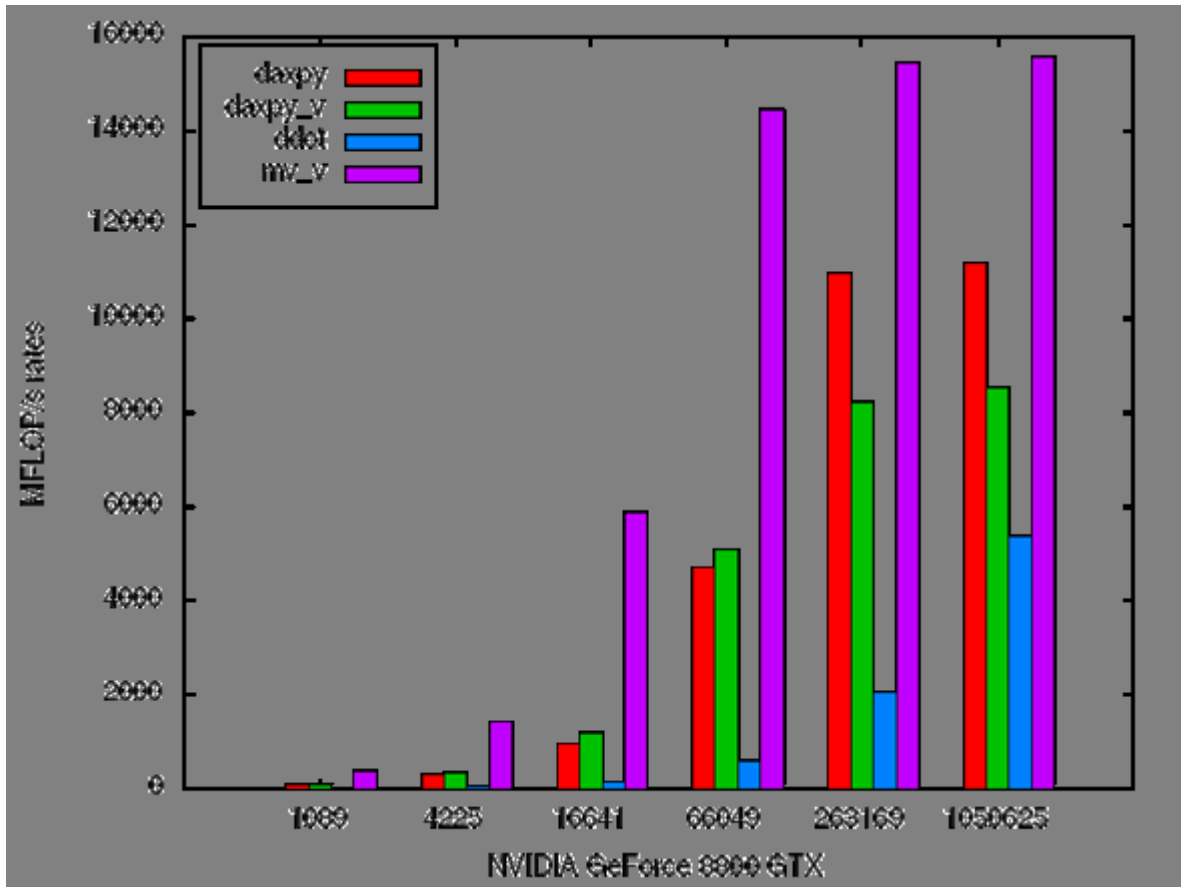
Benchmarks: FEM Building Blocks

- Typical performance of FEM building blocks SAXPY_C, SAXPY_V (variable coefficients), MV_V (9-point-stencil, Q1 elements), DOT on Intel Core2Duo (SBBLAS) and GeForce 8800 GTX



Benchmarks: FEM Building Blocks

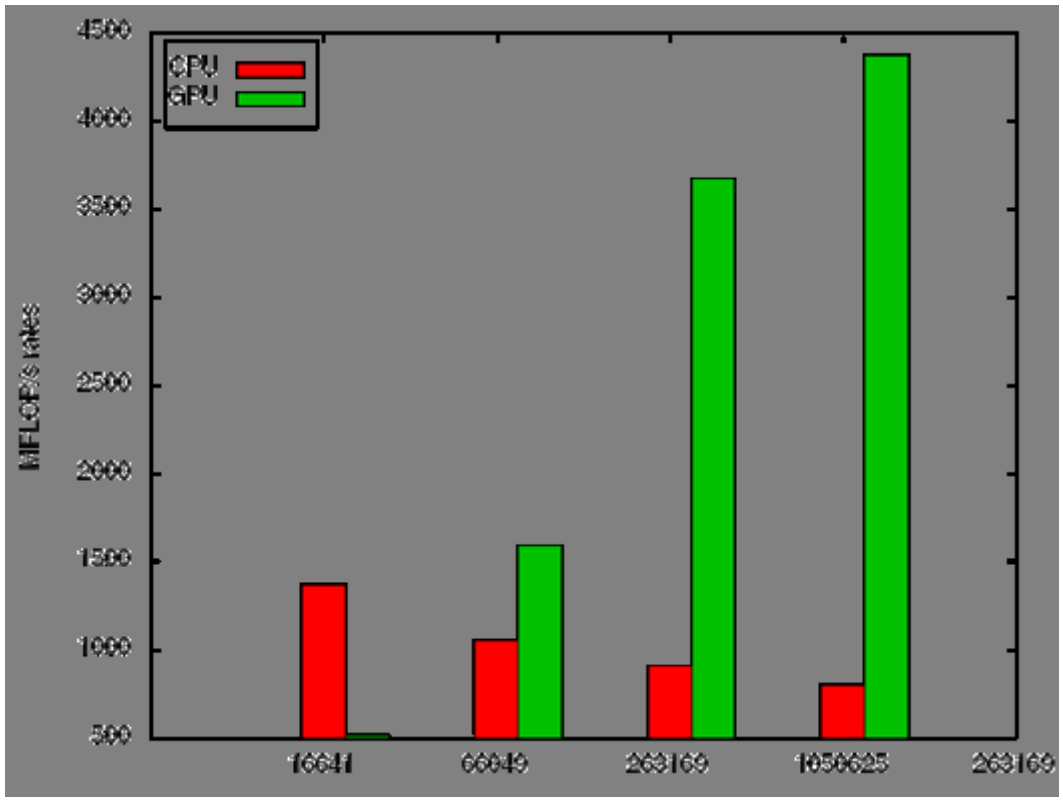
- Typical performance of FEM building blocks SAXPY_C, SAXPY_V (variable coefficients), MV_V (9-point-stencil, Q1 elements), DOT on Intel Core2Duo (SBBLAS) and GeForce 8800 GTX



Benchmarks: FEM Building Blocks

- Problems:
  - Basic linear algebra operations for sparse matrices are typically memory-bounded
  - GPU: Superior to in-cache performance of CPU for large vectors and matrices!
- Open question:

  How to use them?



Motivation

- We want to solve large systems that arise from FEM discretisations very efficiently on commodity clusters.
- CPUs are general-purpose and only achieve close-to-peak performance in-cache. CPUs devote most of the area to memory (hierarchies) and not to processing elements (PEs).
- Emerging parallel specialised chips (GPUs, CELL) are PE-dominated and provide potentially lots of FLOPS and huge memory bandwidth.
- **Goal:** Investigate how such designs can be used as numerical co-processors in scientific computing.

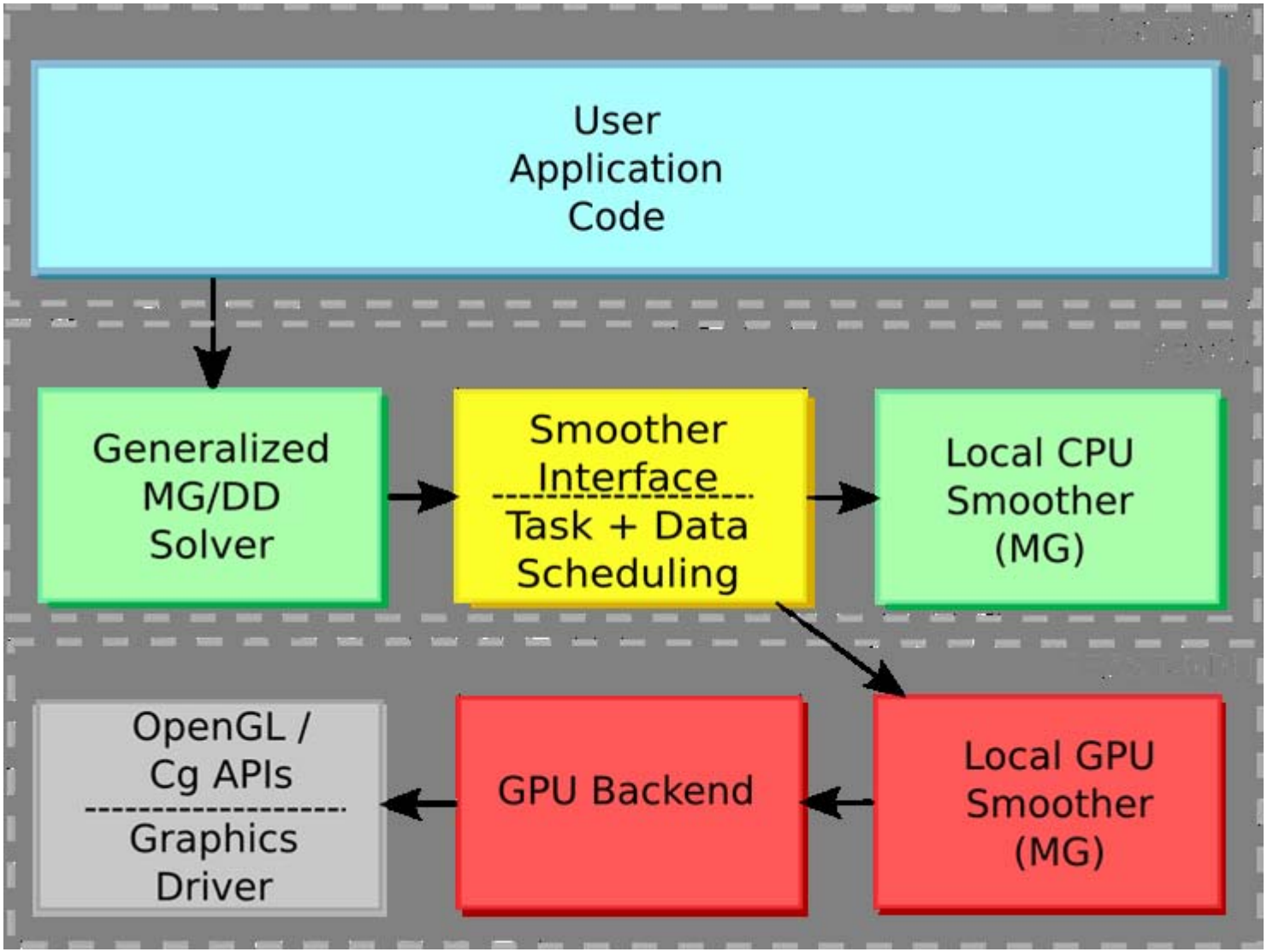
- **Focus exemplarily on GPUs (and: CELL Processor)**

Goals

- **Include GPUs into an existing FEM package**
  - ...without changes to application code built on top of the package,
  - ...without fundamental re-design of the package,
  - ...without sacrificing either functionality or accuracy,
  - ...but with noteworthy speedups,
  - ...a reasonable amount of generality w.r.t. other co-processors,
  - ...and additional benefits in terms of space/power/etc.

But: no –march=gpu, cell

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

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, cooling)
  - Maintenance slightly increased
- Precision vs. accuracy
  - Double precision needed, but only at crucial stages of the computation
  - Mixed precision iterative refinement approach
  - Accuracy not affected

GPU Limitations

- **Challenge:** Reformulate algorithms to the data-stream based programming paradigm!
  - PCIe bus between host system and GPU delivers up to 2 GB/s only
  - GPUs only provide quasi-IEEE 32-bit floating point storage and arithmetics. No double precision! Tests for Poisson equation in 2D $-\Delta u = f$ in some domain $\Omega \subset R^2$ with Dirichlet BCs Discretised with bilinear conforming Finite Elements.

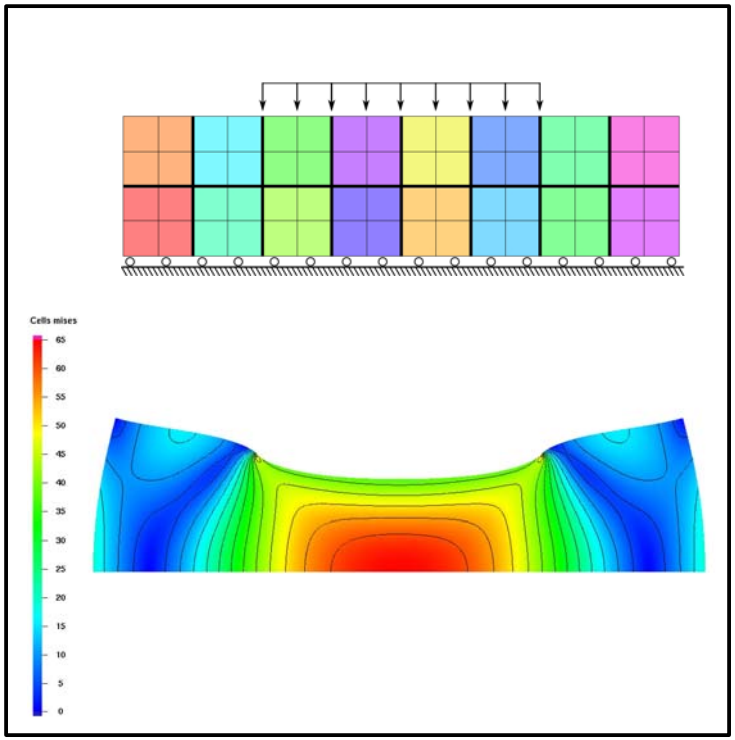
Level	single precision			double precision		
	Cycles	Error	Reduction	Cycles	Error	Reduction
2	1	2.391E-3		1	2.319E-3	
3	2	5.950E-4	4.02	2	5.950E-4	4.02
4	2	1.493E-4	3.98	2	1.493E-4	3.99
5	2	3.750E-5	3.98	2	3.728E-5	4.00
6	2	1.021E-5	3.67	2	9.304E-6	4.01
7	2	6.691E-6	1.53	2	2.323E-6	4.01
8	2	2.012E-5	0.33	2	5.801E-7	4.00
9	2	7.904E-5	0.25	2	1.449E-7	4.00
10	2	3.593E-4	0.22	2	3.626E-8	4.00

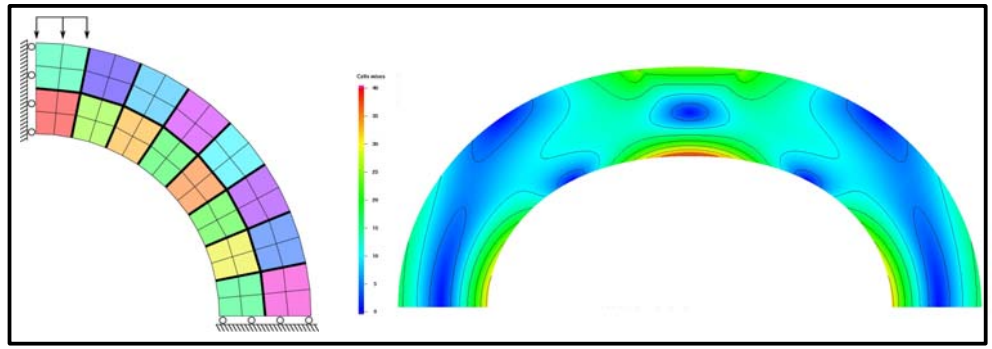
Mixed Precision Iterative Refinement

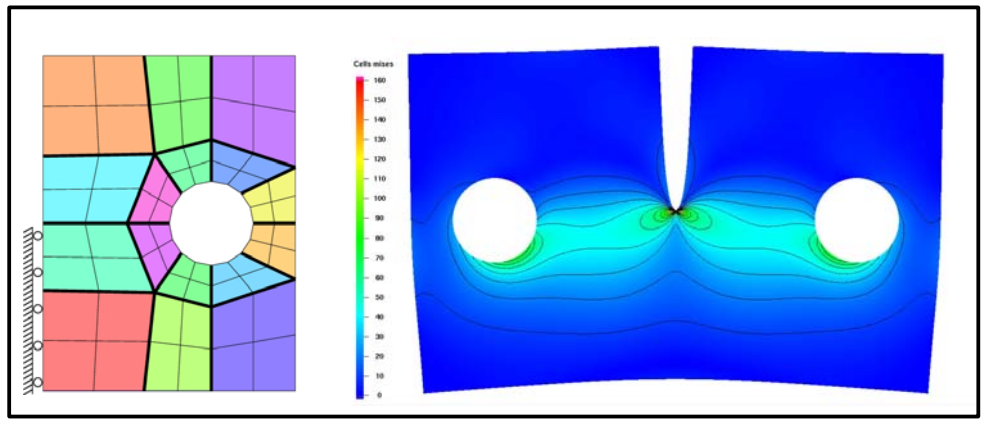
- Single precision computation insufficient for required accuracy, but:
  - High precision only necessary at few, crucial stages!
- Mixed precision iterative refinement approach to solve $Ax = b$:
  - Compute $d = b - Ax$ in high precision.
  - Solve $Ac = d$ approximately in low precision.
  - Update $x = x + c$ in high precision and iterate.
- Use arbitrary iterative inner solvers until "*few*" digits are gained locally.
- Fits naturally on target hardware: Few, high precision updates on the CPU and expensive low precision iterative solution on the GPU.
- Exhaustive experimental and theoretical foundation: very robust w.r.t. solvers, degrees of anisotropy in the discretisation and matrix condition.

Show-Case: FEAST-Solid

- Fundamental model problem:
  - solid body of elastic, compressible material (e.g. steel)
  - exposed to some external load





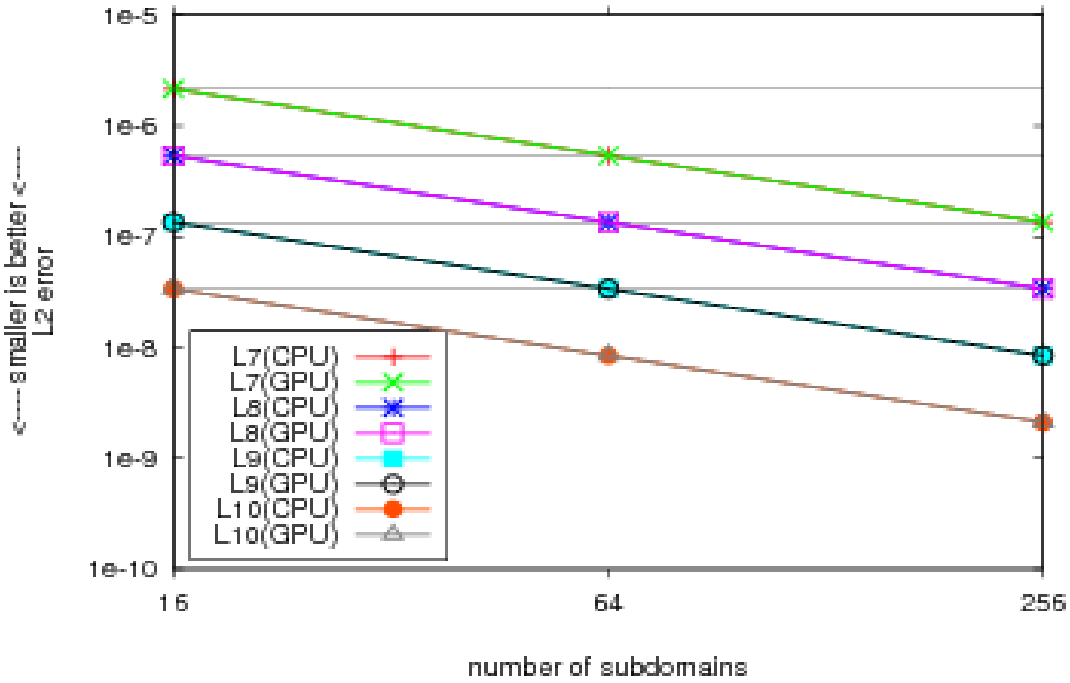


Test goals

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

Accuracy

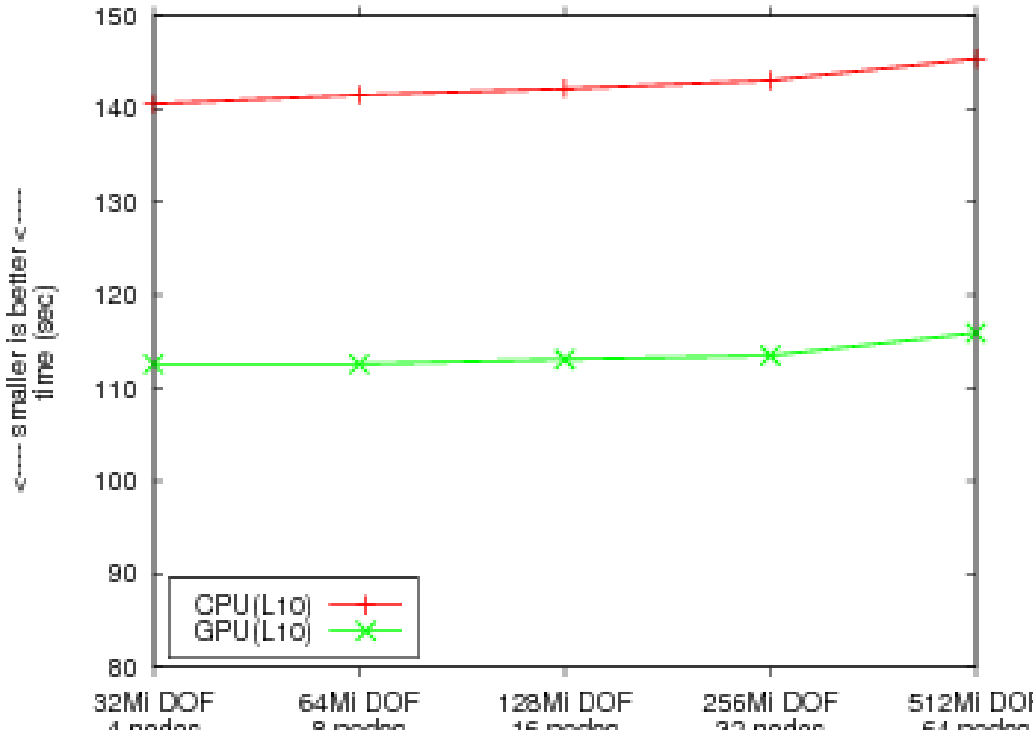
- **L_2 error against analytically known reference solution**



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

Weak scalability

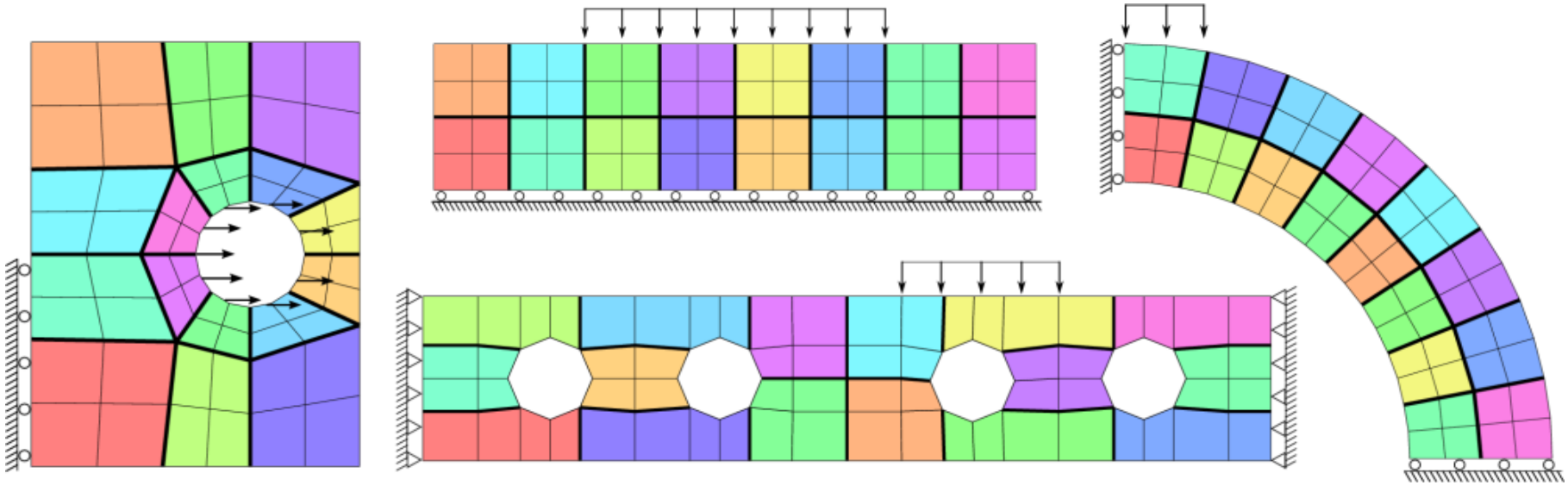
- Good scalability
  - original and accelerated CSM solver
  - Infiniband, Xeon EM64T, 3.4GHz, outdated Quadro 1400 GPU



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

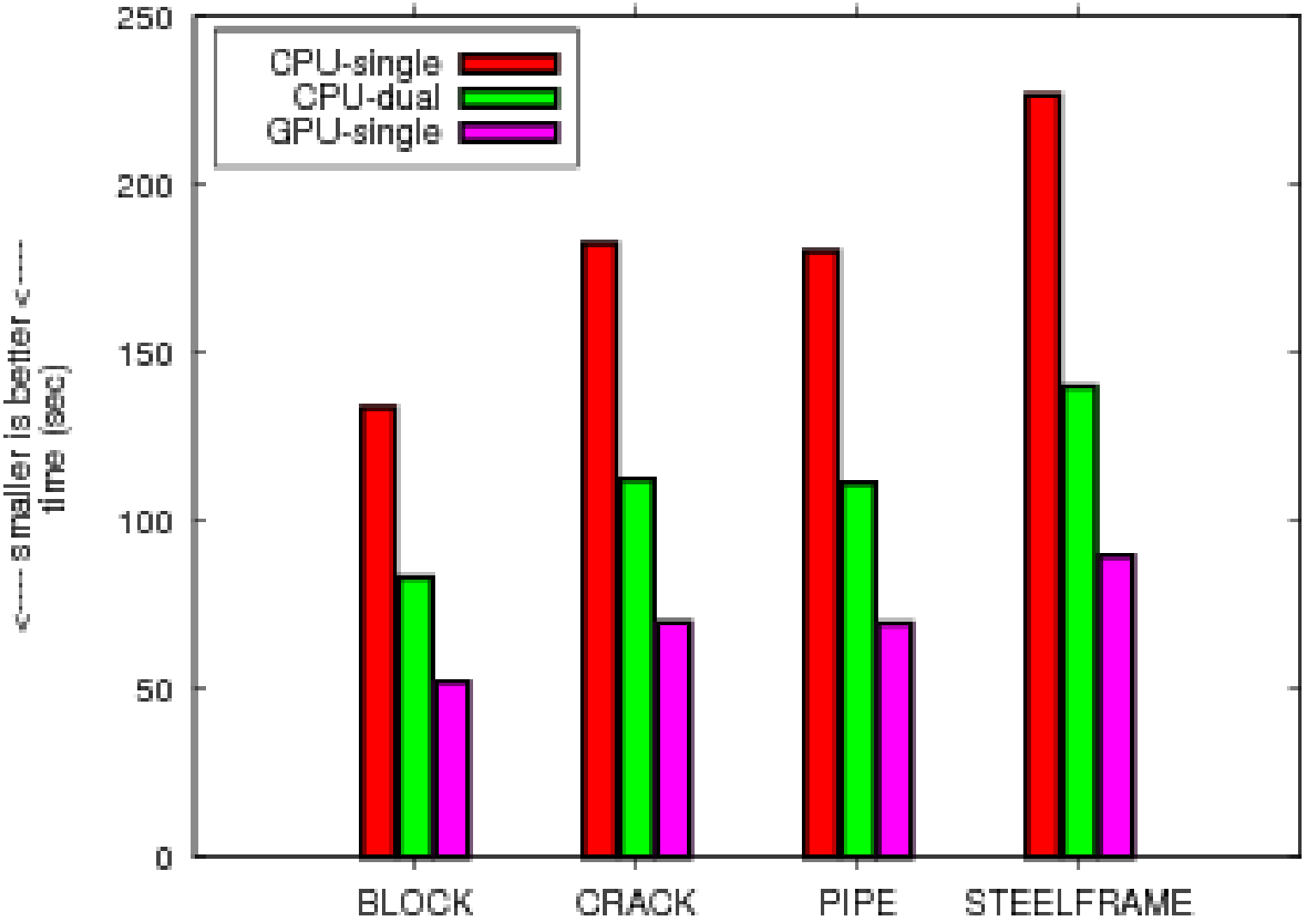
Paper: Göddeke et al., Exploring weak scalability for FEM calculations on a GPU-enhanced cluster, Parallel Computing 33(10-11), 685-699, 2007

Test configurations



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

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)

Summary

- **Goal: Include GPUs into an existing FEM package …**
  - ...without changes to application code built on top of the package,
    → **ok**
  - ...without fundamental refactoring of the package,
    → **only 1% of the code base has been touched**
  - ...without sacrificing either functionality or accuracy,
    → **identical errors and convergence behaviour**
  - ...but with noteworthy speedups,
    → **up to 4x for a reasonably challenging application**
  - ...with a reasonable amount of generality w.r.t. other co-processors,
    → **in progress**
  - ...and with additional benefits in terms of space/power/etc.
    → **promising preliminary results**

Conclusions

- ***There is a huge potential for the future …***
  - **But:** Numerics has to consider recent and future hardware trends!
  - **But:** Developing ’HPC-PDE software’ is more than the implementation of existing Numerics for PDE!
    - Understanding the essentials of ’Total Numerical Efficiency’
    - Design, analysis and realization of hardware-oriented Numerics
    - Identification and realization of hardware-optimized basic components
    - Heterogeneous CPU-GPU clusters as ’building blocks’