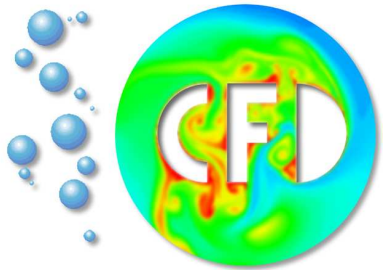
High Performance Computing for PDE
Some numerical aspects of Petascale Computing

S. Turek, D. Göddeke
with support by: Chr. Becker, S. Buijssen, M. Grajewski, H. Wobker

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

Algorithms and Architectures for Petascale Computing

Dagstuhl, February, 13th-17th 2006



Aim of this Talk

Overall Aim:

'High Performance Computing'

meets

'Hardware-Oriented Numerics for PDE'

Hardware-Oriented Numerics

What is:

Hardware-Oriented Numerics for PDE ?

It is more than "good Numerics" and "good Implementation" together with High Performance Computing techniques !

Critical quantity: 'Total Numerical Efficiency !'

Answer

What is the "Total Numerical Efficiency" for the computational simulation of PDE ?

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

Mathematical Key Technologies

'A posteriori error control/adaptive meshing'

'Iterative (parallel) solution strategies'

'Operator-splitting for coupled problems'

But: *How to achieve a high "Total Numerical Efficiency" ?*

For iterative solvers + adaptive discretizations ?

Example

Example: Fast Multigrid Solvers

'Optimized' versions for scalar PDE problems
($\approx$ Poisson problems) on general meshes
should require 100 - 1000 FLOPs per unknown

Problem size 10^6: Much less than 1 sec on PC!

Problem size 10^{12}: Less than 1 sec on PFLOP/s computer!

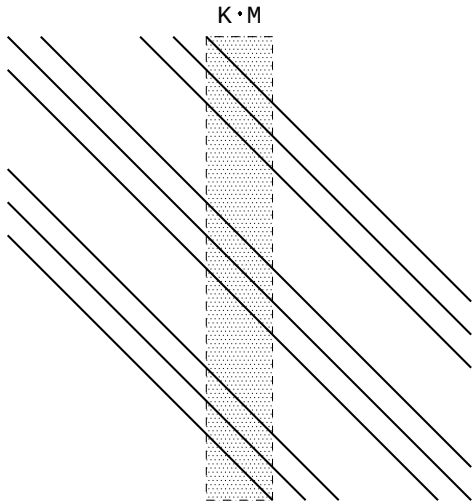
'Criterion' for Petascale Computing

Main Component: 'Sparse' MV Application

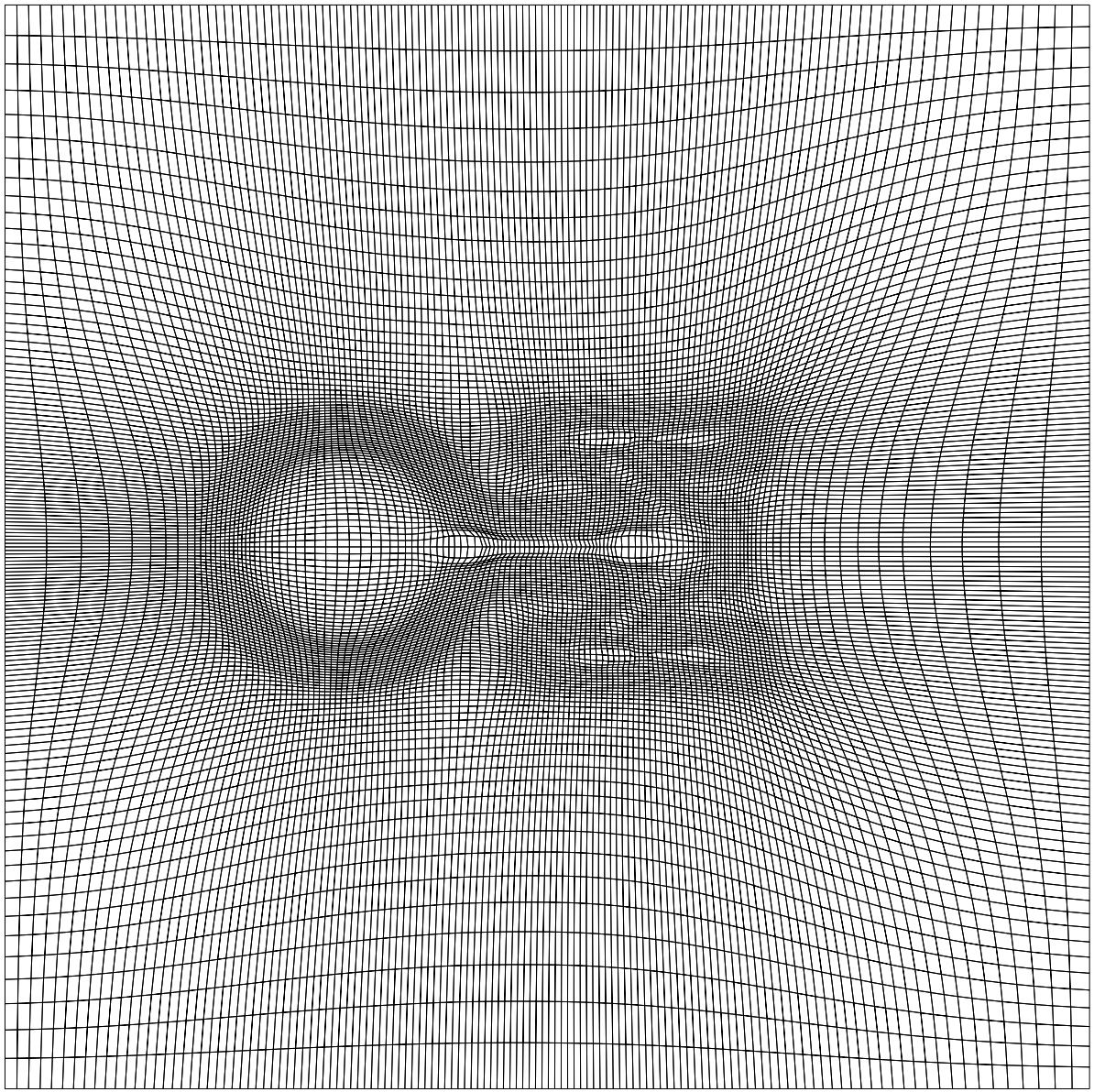
SPARSE **Matrix-Vector techniques** ('indexed DAXPY')

```
      DO 10 IROW=1,N
      DO 10 ICOL=KLD(IROW),KLD(IROW+1)-1
10 Y(IROW)=DA(ICOL)*X(KCOL(ICOL))+Y(IROW)
```

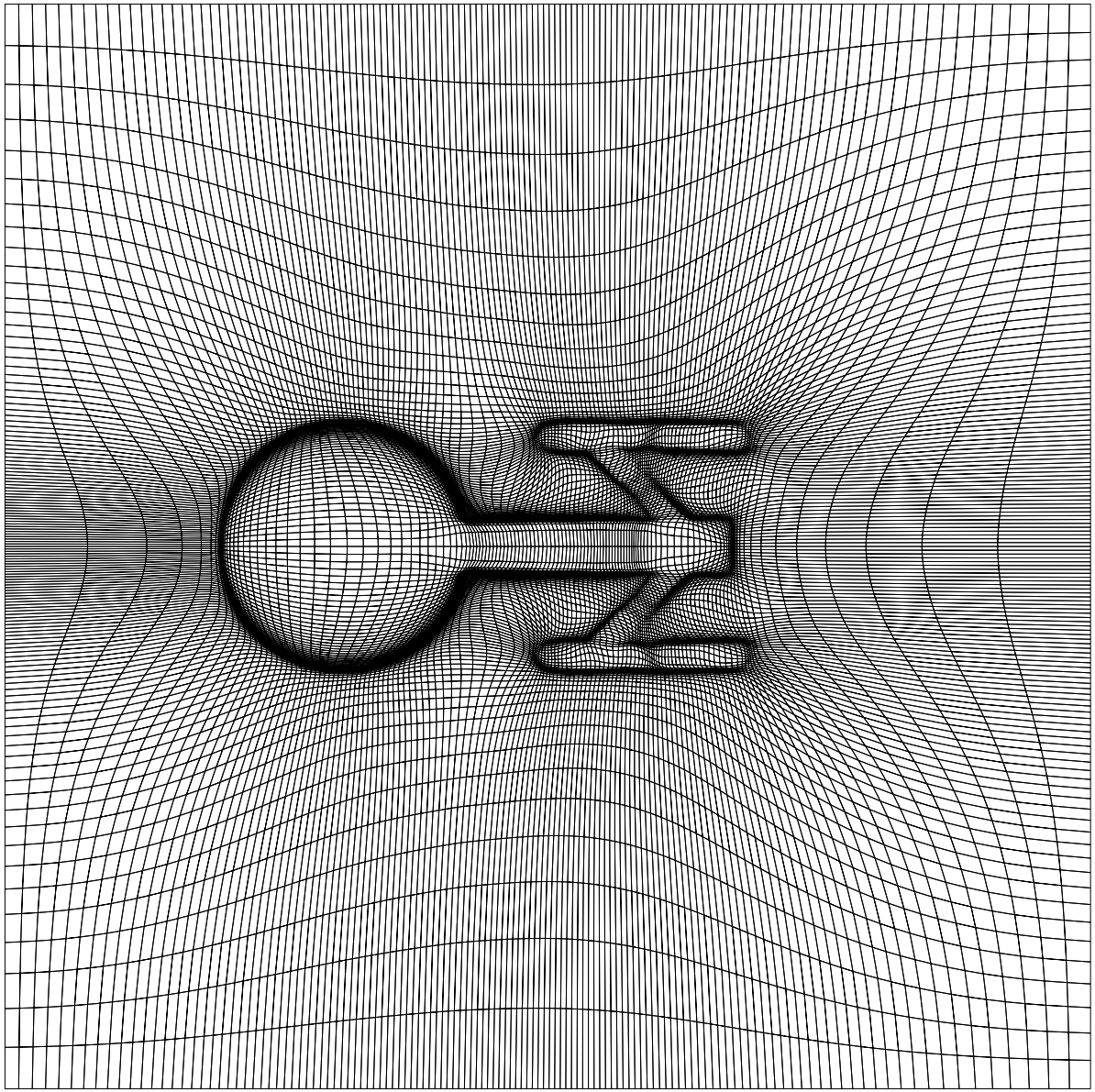
SPARSE BANDED **Matrix-Vector techniques**



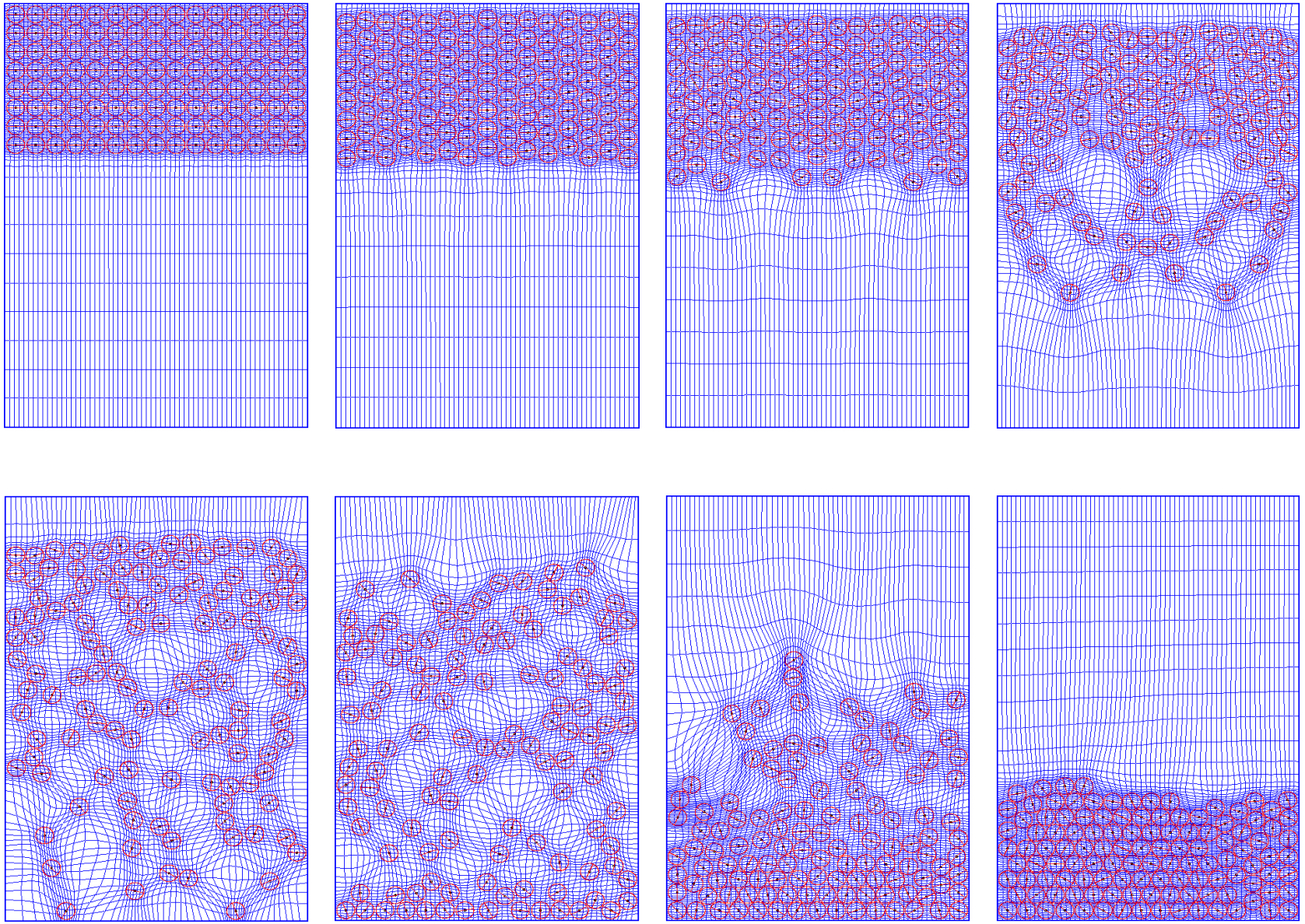
Generalized Tensorproduct Meshes



Generalized Tensorproduct Meshes



Generalized Tensorproduct Meshes



Single Processor Performance (I)

'Generalized Tensorproduct' meshes

2D case	NEQ	ROW (STO)	SBB-V	SBB-C	MGTRI-V	MGTRI-C
Sun V20z	65^2	2172 (633)	1806	3334	1541	2086
(2600 MHz)	257^2	574 (150)	627	2353	751	1423
'**Opteron**'	1025^2	**300 (64)**	**570**	**1774**	**538**	**943**
IBM POWER4	65^2	1521 (845)	2064	3612	906	1071
(1700 MHz)	257^2	943 (244)	896	2896	711	962
'**JUMP**'	1025^2	**343 (51)**	**456**	**1916**	**438**	**718**

SPARSE **MV techniques (**STO/ROW**)**

MFLOP/s rates vs. 'Peak Performance', problem size + numbering ???

Local Adaptivity !!!

SPARSE BANDED **MV techniques (**SBB**) + MGTRI**

'Supercomputing' (up to 4 GFLOP/s) vs. FEM for complex domains ???

Single Processor Performance (II)

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

$\Uparrow$

"Cyclic Reduction" preconditioner
"SPAI" preconditioner ($\sim$ pure MV multiplication)

Summary (I)

'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...'

Summary (II)

'Special requirements for numerical and algorithmic approaches in correspondance to modern hardware!'

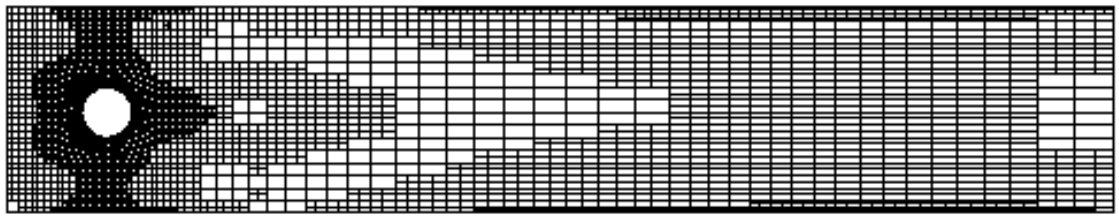
'Hardware-Oriented Numerics for PDE'

FEAST Project

(Some) Numerical Techniques

I) *Patch-oriented h-p-r adaptivity*

'Many' local TP grids (SBB) *with arbitrary spacing*
'Few' unstructured grid parts (SPARSE)



II) *Generalized MG-DD solver:* ScaRC

Exploit locally 'regular' structures **(efficiency)**
Recursive 'clustering' of anisotropies **(robustness)**
'Strong local solvers improve global convergence !'

'Exploit locally regular structures !!!'

Open Numerical 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 ?

- **'Optimality' of the mesh, resp., discretization ?**
  → *w.r.t. number of unknowns or total CPU time ?*

(Preliminary) Conclusions

- **Numerical efficiency ?**
  → *OK*
- **Parallel efficiency ?**
  → *(OK)*
- **Single processor efficiency ?**
  → *almost OK for CPU*
- **"Peak" efficiency ?**
  → *NO*
  → *Special GPU/FPGA-based FEM co-processors*

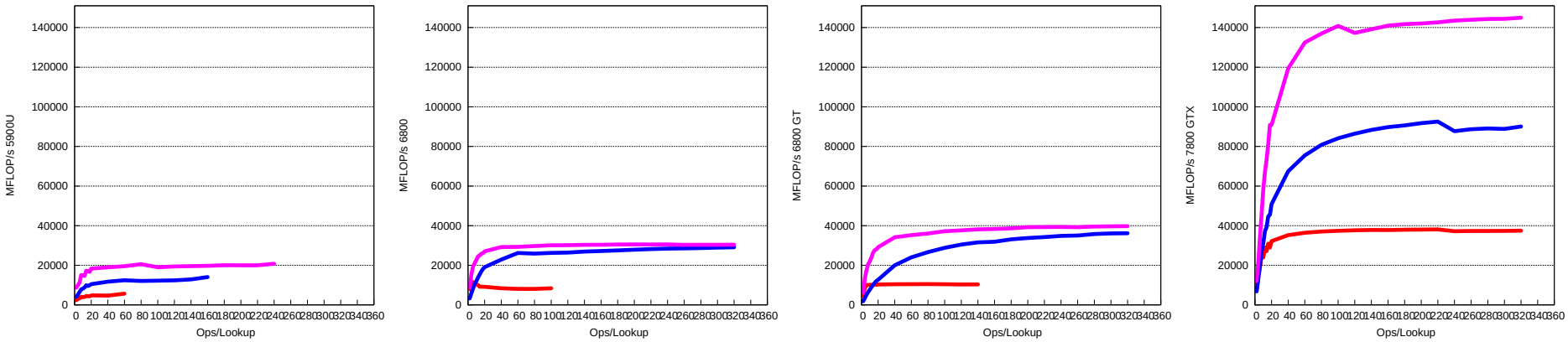
Aim: Numerics on Sony PlayStation 3

Hot off the press:
Cell multicore processor, 7 synergetic processing units @ 3.2 GHz, 218 GFLOP/s
RSX graphics processor, 6+24 pipelines, 1.8 TFLOP/s, memory clocked @ 3.2 GHz, announced for 2006

Graphics Processors: 8+24 parallel pipelines @ 430MHz, 600 MHz GDDR3 memory (40 GB/s), capable of performing 24*2 vec4 multiply-adds simultaneously in the fragment pipe ($\approx$ 160 GFLOP/s) alone, 2-level SIMD.

Benchmarks: Peak performance

Measured peak performance with R32, RGBA32 and RGBA16 floating point formats: GeForce 5900U, 6800, 6800GT, 7800 GTX, 2004–2006



"Moore's Law cubed, 2000 – ???"

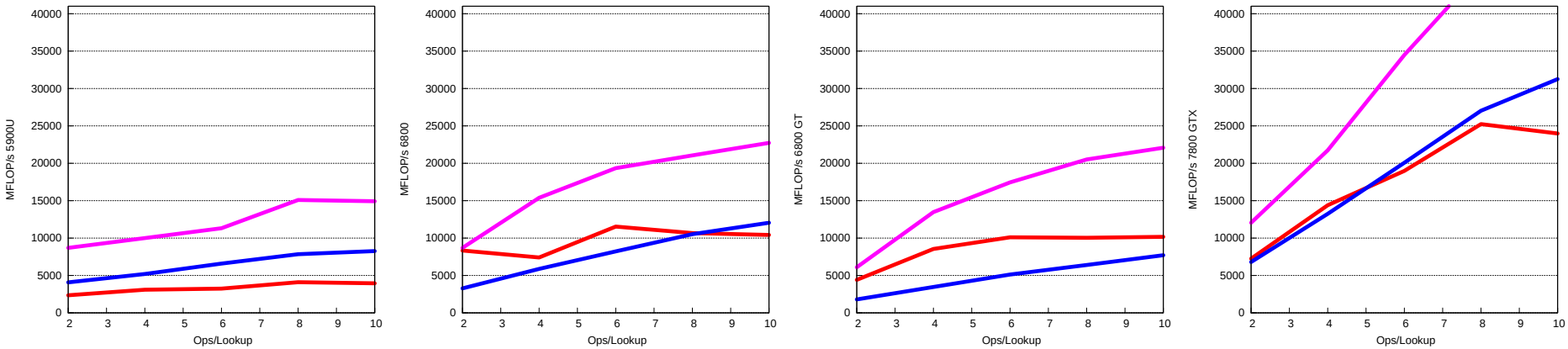
- Plots are just tendency for compute-bound calculation, not necessarily based on optimal implementation.
- Need high arithmetic intensity for optimal performance

$$\Rightarrow \quad > 8 \text{ ops per read!}$$

- Galoppo et al. (Supercomputing 05): Single precision LINPACK performance approximately 2x ATLAS.

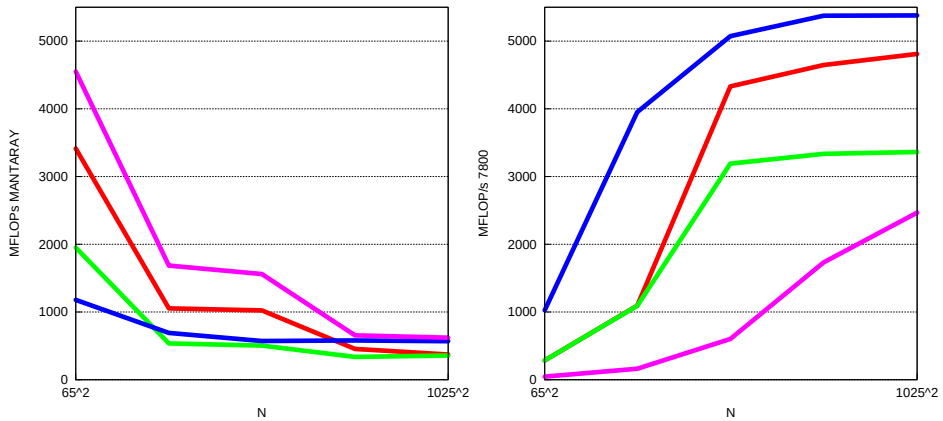
Benchmarks: Peak performance

Realistic range of 2–10 operations per memory access. Zoomed view of previous slide with R32, RGBA32 and RGBA16 floating point formats: GeForce 5900U, 6800, 6800GT, 7800 GTX.



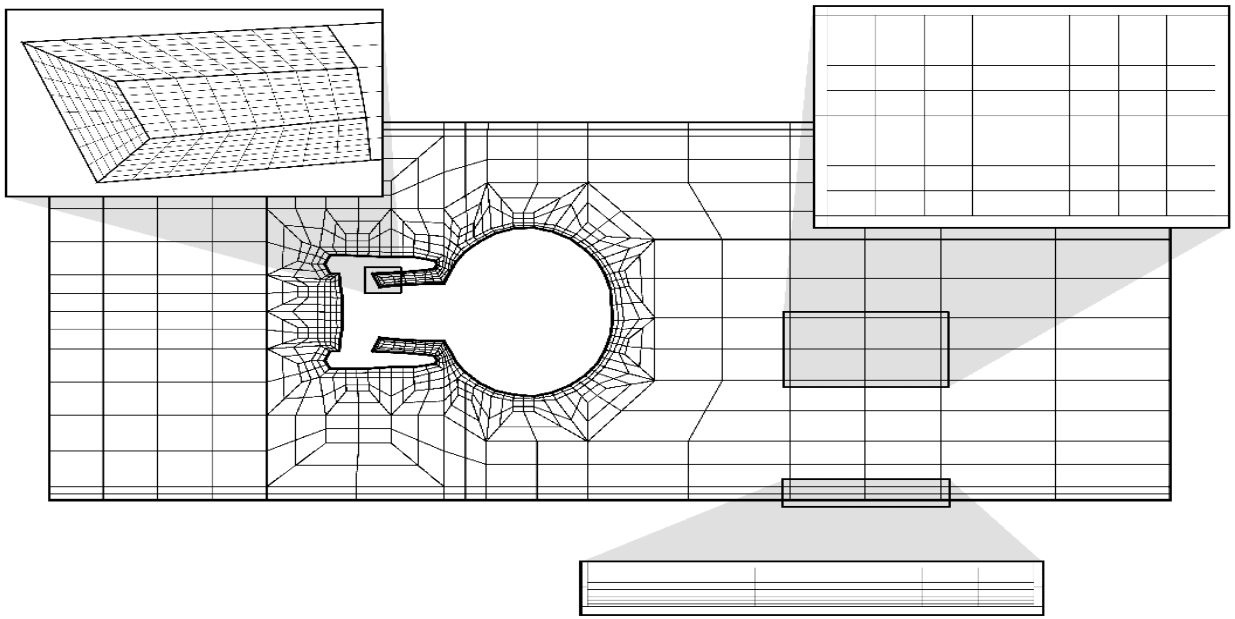
Benchmarks: FEM building blocks

Typical performance of FEM building blocks `SAXPY_C`, `SAXPY_V` (variable coefficients), `MV_V` (9-point-stencil, Q_1 elements), `DOT` on Opteron 244 (SBBLAS) and GeForce 7800 GTX, $N = 65^2 \ldots 1025^2$:



- Basic linear algebra operations on banded matrices are typically memory-bound.
- But: Comparable to in-cache performance on CPU for large vectors and matrices!
- Open question: How to make them compute-bound?

Prototypical realization of FEAST-GPU



Goal: Outer (parallel) ScaRC solver (CG or MG) offloads local problems to GPU co-processor(s) for preconditioning.

Challenges:

- different programming paradigm (parallel array processing)
- cross-language cross-API programming and optimization (OpenGL, no GPU-(SB)BLAS (yet?))
- numerical and computational efficiency/flexibility
- high accuracy (GPUs only support single precision)

Mixed Precision Iterative Refinement

- Collaboration with R. Strzodka (Stanford) and H. Müller (Dortmund).
- Native single precision (32 bit) insufficient for large problem sizes.
- Approach 1: Native-pair arithmetic (emulating double float by combining two singles): Doubles memory requirements, increases op count quadratically.
- Approach 2: Mixed precision **iterative refinement**.

Calculate local defect on CPU in high precision.

Smooth defect (= local problems) on co-processor in low precision.

Update local CPU solution with smoothed defect.

Iterate.

Issues and questions:
PCI/E transfer rate bidirectional: $\approx$ 1–2 GB/s. Error and convergence control; tradeoff inner workload vs. (asynchronous) transfer overhead; restart frequencies, choice of inner solver,...

Proof of Concept

- Poisson problem on unit square, $N = 127^2 \ldots 1025^2$ unknowns.
- (Unpreconditioned) Conjugate Gradient 'solver'.
- CPU: Opteron 244, FEAST, all compiler optimizations, double precision.
- GPU: GeForce 7800 GTX, double-single, restart after 2 gained digits.
- Time until solution, including all necessary transfers.

N	Iters FEAST	Iters GPU-CPU	Time FEAST	Time GPU-CPU	speedup
127^2	151	412 (5)	0.27s	0.03s	9.0
257^2	303	861 (5)	2.32s	0.79s	2.3
513^2	601	2256 (5)	19.47s	4.41s	4.4
1025^2	1191	4500 (6)	146.90s	29.75s	4.9

Accuracy: Same error as double precision FEAST solver compared to analytically known reference solution.

Discussion

- GPU-CPU solver outperforms CPU-SCARC by a factor of 5 despite unoptimized "proof of concept" implementation and an increase in iterations by 3.7.
- High accumulated iteration count not caused by lack of precision, but by CG sensitivity to restarts, known information about descend directions is lost upon restart: $\Rightarrow$ Better solvers!
- Evaluation of several mixed precision multigrid solvers as inner preconditioners: No overall increase in iterations (submitted to IJPEDS).
- Pipelined mixed precision CG tuned for FPGAs and various user-defined low-precision floating point formats: Substantially less increase in iterations (submitted to FCCM).

Current and future work

- GPU-based multigrid for non-cartesian meshes.
- Robust multigrid smoothers for highly deformed meshes: ILU(k), ADITRIGS, ...
- Dynamic loadbalancing, globally between cluster nodes and locally between CPUs and their co-processor(s).
- Many more potential co-processor platforms appearing on the horizon (cf. Uwe Küster's talk): CELL processor, Ageia PhysX chip, ClearSpeed accelerator boards, ...

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 and definition of 'Total Numerical Efficiency'
→ Design, analysis and realization of hardware-oriented Numerics
→ Identification and realization of hardware-optimized basic components
→ CPU-GPU clusters as 'building blocks'

Do not forget Terascale tools for "daily life" !