High Performance Computing for PDE
Towards 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

January 30, 2007
Aim of this Talk

Overall Aim:

‘High Performance Computing’

meets

‘Hardware-Oriented Numerics for PDE’
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!’**
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: Fast Multigrid Solvers

‘Optimized’ versions for scalar PDE problems
(≈ 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’)

\[
\text{DO 10 IROW=1,N} \\
\text{DO 10 ICOL=KLD(IROW),KLD(IROW+1)-1} \\
10 \text{ Y(IROW)=DA(ICOL)*X(KCOL(ICOL))+Y(IROW)}
\]

**Sparse Banded Matrix-Vector techniques**
Generalized Tensorproduct Meshes
Generalized Tensorproduct Meshes
Generalized Tensorproduct Meshes
Sparse MV multiplication in (sequential) FEATFLOW:

<table>
<thead>
<tr>
<th>Computer</th>
<th>#Unknowns</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC 21264</td>
<td>8,320</td>
</tr>
<tr>
<td>(667 MHz)</td>
<td>33,280</td>
</tr>
<tr>
<td>‘EV67’</td>
<td>133,120</td>
</tr>
<tr>
<td></td>
<td>532,480</td>
</tr>
<tr>
<td></td>
<td>2,129,920</td>
</tr>
<tr>
<td></td>
<td>8,519,680</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CM</th>
<th>TL</th>
<th>STO</th>
<th>ILU-CM</th>
<th>ILU-TL</th>
<th>ILU-STO</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC 21264</td>
<td>147</td>
<td>136</td>
<td>116</td>
<td>90</td>
<td>76</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>125</td>
<td>105</td>
<td>100</td>
<td>86</td>
<td>73</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>81</td>
<td>71</td>
<td>58</td>
<td>81</td>
<td>52</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>51</td>
<td>21</td>
<td>40</td>
<td>35</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>47</td>
<td>13</td>
<td>38</td>
<td>30</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>45</td>
<td>10</td>
<td>36</td>
<td>30</td>
<td>11</td>
</tr>
</tbody>
</table>
Single Processor Performance (II)

‘Generalized Tensorproduct’ meshes

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

**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 (III)

#### Vectorization

<table>
<thead>
<tr>
<th>2D case</th>
<th>NEQ</th>
<th>ROW (STO)</th>
<th>SBB-V</th>
<th>SBB-C</th>
<th>MGTRI-V</th>
<th>MGTRI-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEC SX-8 (2000 MHz)</td>
<td>$65^2$</td>
<td>5070 (1521)</td>
<td>3611</td>
<td>3768</td>
<td>1112</td>
<td>1061</td>
</tr>
<tr>
<td>‘Vector’</td>
<td>$257^2$</td>
<td>5283 (1321)</td>
<td>6278</td>
<td>8363</td>
<td>1535</td>
<td>1543</td>
</tr>
<tr>
<td></td>
<td>$1025^2$</td>
<td>5603 (1293)</td>
<td>7977</td>
<td>15970</td>
<td>1918</td>
<td>2053</td>
</tr>
</tbody>
</table>

### Necessary: Development of ‘new’ methods

↑

”Cyclic Reduction” preconditioner

”SPA1” preconditioner (∼ pure MV multiplication)
‘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 (I)

‘Complex (anisotropic) ASMO3D configuration’

‘(Moderate) mesh anisotropies (AR = 20)’

‘Problems due to communication’

‘Problems due to Pressure Poisson multigrid solver’
Parallel Performance (II)

![Bar chart showing parallel efficiency for different systems and process counts.]

- **Alphacluster**
- **Cray T3E-1200**
- **Linuxcluster**
- **Sun Enterprise 3500**

<table>
<thead>
<tr>
<th>#Prozesse</th>
<th>1 P.</th>
<th>2 P.</th>
<th>4 P.</th>
<th>8 P.</th>
<th>16 P.</th>
<th>32 P.</th>
<th>64 P.</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Comm.</td>
<td>10%</td>
<td>24%</td>
<td>36%</td>
<td>45%</td>
<td>47%</td>
<td>55%</td>
<td>56%</td>
</tr>
<tr>
<td>#PPP-IT</td>
<td>2.2</td>
<td>3.0</td>
<td>3.9</td>
<td>4.9</td>
<td>5.2</td>
<td>5.7</td>
<td>6.2</td>
</tr>
</tbody>
</table>

**Summary and Conclusions**

- **HPC components**
- **Hardware-oriented Numerics**
- **FEM coprocessors: GPUs, FPGAs**
'Special requirements for numerical and algorithmic approaches in correspondance to modern hardware!'

⇒

‘Hardware-Oriented Numerics for PDE’

⇒

FEAST Project
FEAST Solution Strategy

- ScaRC approach: 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: Smooth outer global multigrid with local multigrid on the refined macros.
- Low communication overhead.
(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

Cell multicore processor, 7 synergetic processing units @ 3.2 GHz, 218 GFLOP/s
memory clocked @ 3.2 GHz

Graphics Processors: 128 parallel scalar processors @ 1.35GHz, 900 MHz GDDR3 memory (86.4 GB/s), ≈ 500 GFLOP/s
Motivation

- We want to solve large systems that arise from FEM discretisations fast 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 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 exemplary on Graphics Processors (soon: CELL Processor)
Benchmarks: FEM building blocks

Typical performance of FEM building blocks \texttt{SAXPY\_C}, \texttt{SAXPY\_V} (variable coefficients), \texttt{MV\_V} (9-point-stencil, $Q_1$ elements), \texttt{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, we see $\approx 95\%$ peak memory bandwidth
- Comparable to in-cache performance on CPU for large vectors and matrices, but for large problem sizes!
- Open question: How to make them compute-bound?
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

\[-\Delta u = f \text{ in some domain } \Omega \subset \mathbb{R}^2 \text{ with Dirichlet BCs}\]

Discretised with bilinear conforming Finite Elements.

<table>
<thead>
<tr>
<th>Level</th>
<th>Cycles</th>
<th>single precision</th>
<th>double precision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Error</td>
<td>Reduction</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2.391E-3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5.950E-4</td>
<td>4.02</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1.493E-4</td>
<td>3.98</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3.750E-5</td>
<td>3.98</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1.021E-5</td>
<td>3.67</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>6.691E-6</td>
<td>1.53</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2.012E-5</td>
<td>0.33</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>7.904E-5</td>
<td>0.25</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>3.593E-4</td>
<td>0.22</td>
</tr>
</tbody>
</table>
Mixed Precision Iterative Refinement

- Single precision computation insufficient for required result 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 wrt. solvers, degrees of anisotropy in the discretisation and matrix condition.
Results

- Poisson on unit square, regular refinement, conforming bilinear $Q_1$ elements. Multigrid solver with Jacobi smoother
- CPU: Athlon X2 4400+
- GPU: GeForce 7800 GTX, mixed precision iterative refinement

<table>
<thead>
<tr>
<th>$N$</th>
<th>CPU time</th>
<th>CPU error</th>
<th>GPU time</th>
<th>GPU error</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>$127^2$</td>
<td>0.28</td>
<td>1.666003670E-6</td>
<td>0.26</td>
<td>1.666003655E-6</td>
<td>1.08</td>
</tr>
<tr>
<td>$257^2$</td>
<td>0.69</td>
<td>4.181054493E-7</td>
<td>0.33</td>
<td>4.181054014E-7</td>
<td>2.09</td>
</tr>
<tr>
<td>$513^2$</td>
<td>1.95</td>
<td>1.047283071E-7</td>
<td>0.56</td>
<td>1.047281043E-7</td>
<td>3.48</td>
</tr>
<tr>
<td>$1025^2$</td>
<td>7.07</td>
<td>2.620418265E-8</td>
<td>1.69</td>
<td>2.620376988E-8</td>
<td>4.18</td>
</tr>
</tbody>
</table>

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

- FEAST: Under development since 1999, 100K+ lines of code, tuned data structures, adaptions for clusters (MPI) and NEC vector machines.
- Consequence: Full rewrite to incorporate GPUs is out of question!
- Goal: Minimally invasive integration.
- Large-scale solution scheme: global MG smoothed by many local MGs
- GPU backend adds new smoother, while FEAST maintains all global data structures.
- Data flow example: Outer MG calls smoother, matrix and current defect are duplicated into GPU memory, smoothing is performed independently, correction term is read back to the CPU.
Preliminary GPU cluster results

- Joint work with colleagues from Stanford and Los Alamos.
- Cluster with 16 compute nodes and 1 master node.
- Dual Intel EM64T 3.4 GHz, NVIDIA Quadro FX4500 PCIe graphics card.
- Fully connected via Infiniband.
- Two test cases:
  - Homogeneous domains: pure CPU vs. pure GPU vs. one CPU and one GPU per node
  - Heterogeneous domains: CPU treats few unstructured subdomains and GPU treats many structured subdomains (each does what it's best at)
Test Case I

### Overall Time in sec

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Overall Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0g_2c8m_Cn_DQ</td>
<td></td>
</tr>
<tr>
<td>0g_2c8m_Cn_LiDO</td>
<td></td>
</tr>
<tr>
<td>1g8m_0c_Cn_DQ</td>
<td></td>
</tr>
<tr>
<td>1g7m_1c1m_Cn_DQ</td>
<td></td>
</tr>
</tbody>
</table>

#### Degrees of Freedom (DOFs)

- 16Mi, C=8 (64 L9 mac)
- 32Mi, C=16 (64 L9 mac)
- 64Mi, C=8 (64 L10 mac)
- 128Mi, C=16 (64 L10 mac)

<--- smaller is better <---

### Summary and Conclusions
Test Case II

The graph shows the overall time in seconds for different configurations of degrees of freedom (DOFs). The x-axis represents the number of DOFs, and the y-axis represents the overall time in seconds. The configurations are color-coded for easier identification:

- Red for 0g_1c12m_Cn_DQ_L9
- Green for 1g8m_1c4m_Cn_DQ_L9
- Blue for 0g_2c12m_Cn_DQ_L10
- Purple for 1g8m_1c4m_Cn_DQ_L10

The graph indicates that smaller is better, aiming for the shortest overall time. The configurations ranging from 6Mi to 192Mi are presented, with corresponding values for C=2 to C=16.

HPC components
- Hardware-oriented Numerics
- FEM coprocessors: GPUs, FPGAs

Summary and Conclusions
Interesting perspectives:
- Inexpensive upgrade of commodity clusters.
- Potential to accelerate production codes.
- But: Maintaining two code lines on the solver and data structure level, not on the application level.

Paradigm shift to data parallelism: Multicores, Cell BE etc., so start learning now: The first honest attempt at petascale computing, the IBM Roadrunner at LANL, will contain multi-GPUs, Cells, Opterons and will in general be a massively parallel hybrid machine.
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” !

HPC components
Hardware-oriented Numerics
FEM coprocessors: GPUs, FPGAs
Summary and Conclusions