UnConventional High Performance Computing (UCHPC) for Finite Element Simulations

- Towards LIDO³ -

S. Turek, Chr. Becker, S. Buijssen, D. Göddeke, M. Köster, H. Wobker  
(FEAST Group)

Institut für Angewandte Mathematik, TU Dortmund

http://www.mathematik.tu-dortmund.de/LS3 
http://www.featflow.de 
http://www.feast.tu-dortmund.de
3 Observations

- The ‘free ride’ is over, paradigm shift in HPC:
  - physical barriers (heat, power consumption, leaking voltage)
  - memory wall (in particular for sparse Linear Algebra problems)
  - applications no longer run faster automatically on newer hardware

- Heterogeneous hardware: commodity CPUs plus co-processors
  - graphics cards (GPU)
  - CELL BE processor
  - HPC accelerators (e.g. ClearSpeed)
  - reconfigurable hardware (FPGA)

- Finite Element Methods (FEM) and Multigrid solvers: most flexible, efficient and accurate simulation tools for PDEs nowadays.
Aim of this Talk

High Performance Computing meets Hardware-oriented Numerics on Unconventional Hardware for Finite Element Multigrid Methods
What is ‘Hardware-Oriented Numerics’?

- It is more than ‘good Numerics‘ and ‘good Implementation’ on High Performance Computers
- Critical quantity: ‘Total Numerical Efficiency’
‘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 numerical complexity (~ O(N)) while exploiting a significant percentage of the available huge sequential/parallel GFLOP/s rates at the same time’

Is it easy to achieve high ‘Total Numerical Efficiency’?

FEM Multigrid solvers with a posteriori error control for adaptive meshing are a candidate
Example: Fast Poisson Solvers

- ‘Optimized’ Multigrid methods for scalar PDE problems (≈Poisson problems) on general meshes should require ca. 1000 FLOPs per unknown (in contrast to LAPACK for dense matrices with $O(N^3)$ FLOPs)

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

⇒ More realistic (and much harder) ‘Criterion’ for Petascale Computing in Technical Simulations
Main Component: ‘Sparse’ MV

- Sparse **Matrix-Vector techniques** (‘indexed DAXPY’) on general unstructured grids
  
  ```
  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 Grids
Generalized Tensorproduct Grids
Generalized Tensorproduct Grids

...with appropriate Fictitious Boundary techniques in FEATFLOW....
Observation I: Sparse MV on TP Grid

- Opteron X2 2214 (1MB C$, 2.2 GHz)
- vs.
- Xeon E5450 (6MB C$, 3 GHz, LiDO2)
- One thread, 3 cores idle ("best case" test for memory bound FEM)
- Production runs expected to be much slower, but not asymptotically
- Banded-const: constant coefficients (stencil), fully in-cache
Observation II: Sparse MV (again)

<table>
<thead>
<tr>
<th>Numbering</th>
<th>4K DOF</th>
<th>66K DOF</th>
<th>1M DOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stochastic</td>
<td>500</td>
<td>364</td>
<td>95</td>
</tr>
<tr>
<td>Hierarchical</td>
<td>536</td>
<td>445</td>
<td>418</td>
</tr>
<tr>
<td>Banded</td>
<td>3285</td>
<td>2219</td>
<td>687</td>
</tr>
<tr>
<td>Stencil (const)</td>
<td>5720</td>
<td>5094</td>
<td>2415</td>
</tr>
</tbody>
</table>

In realistic scenarios, MFLOP/s rates are

- often poor, and
- problem size dependent
Observation III: (Sparse) CFD

Speed-up of 100x for free in 10 years

Stagnation for standard simulation tools on conventional hardware
Observation IV: Parallel Performance

- Mesh partitioned into 32 subdomains
  - Problems due to communication
  - Numerical behavior vs. anisotropic meshes

<table>
<thead>
<tr>
<th></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>
First Summary

- It is (almost) impossible to come close to **Single Processor Peak Performance** with modern (= high numerical efficiency) simulation tools.

- **Parallel Peak Performance** with modern Numerics even harder, already for moderate processor numbers.

*Hardware-oriented Numerics (HwoN)*
Example for HwoN

FEM for 8 Mill. unknowns on general domain, 1 CPU, Poisson Problem in 2D

Dramatic improvement *(factor 1000)* due to **better Numerics** AND **better data structures/ algorithms** on 1 CPU
FEAST – Realization of HwoN

- **ScaRC solver**: Combine advantages of (parallel) domain decomposition and multigrid methods
- Cascaded multigrid scheme
- Hide anisotropies locally to increase robustness
- Globally unstructured – locally structured
- Low communication overhead

**FEAST applications:**
- FEASTFLOW (CFD)
- FEASTSOLID (CSM)
- FEASTLBM (CFD)

→ SkaLB Project
SkaLB Project

- BMBF-Initiative “HPC-Software für skalierbare Parallelrechner”
- “SkaLB - Lattice-Boltzmann-Methoden für skalierbare Multi-Physik-Anwendungen”
- Partners: Braunschweig, Erlangen, Stuttgart, Dortmund (1.8 MEuro)
  - Lehrstuhl für Angewandte Mathematik und Numerik (LS3)
  - ITMC
  - IANUS
- Industry: Intel, Cray, IBM, BASF, Sulzer, hhpberlin, HP
(Preliminary) State-of-the-Art

- Numerical efficiency?
  → OK

- Parallel efficiency?
  → OK (tested up to 256 CPUs on NEC and commodity clusters)
  → More than 10,000 CPUs???

- Single processor efficiency?
  → OK (for CPU)

- ‘Peak’ efficiency?
  → NO
  → Special *unconventional* FEM Co-Processors
- CELL multicore processor (PS3), 7 synergistic processing units @ 3.2 GHz, 218 GFLOP/s, Memory @ 3.2 GHz

- GPU (NVIDIA GTX 285): 240 cores @ 1.476 GHz, 1.242 GHz memory bus (160 GB/s) ≈ 1.06 TFLOP/s
Why are GPUs and CELLs so fast?

**CPUs** minimise latency of individual operations (cache hierarchy to combat memory wall problem)

**GPUs** and **CELLs** maximise throughput over latency and exploit data-parallelism (more “ALU-efficient“ and parallel memory system)
Bandwidth in a CPU/GPU Node
Example: Sparse MV on TP Grid

40 GFLOP/s, 140 GB/s on GeForce GTX 280
0.7 (1.4) GFLOP/s on 1 core of LiDO2
Example: Multigrid on TP Grid
Example: Multigrid on TP Grid

<table>
<thead>
<tr>
<th>Level</th>
<th>LiDO2 (double)</th>
<th>GTX 280 (mixed)</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time(s)</td>
<td>MFLOP/s</td>
<td>time(s)</td>
</tr>
<tr>
<td>7</td>
<td>0.021</td>
<td>1405</td>
<td>0.009</td>
</tr>
<tr>
<td>8</td>
<td>0.094</td>
<td>1114</td>
<td>0.012</td>
</tr>
<tr>
<td>9</td>
<td>0.453</td>
<td>886</td>
<td>0.026</td>
</tr>
<tr>
<td>10</td>
<td>1.962</td>
<td>805</td>
<td>0.073</td>
</tr>
</tbody>
</table>

- 1M unknowns in less than 0.1 seconds!
- 27x faster than CPU

Promising results, attempt to integrate GPUs as FEM Co-Processors
Design Goals

Include GPUs into FEAST

- without
  - changes to application codes FEASTFLOW / FEASTSOLID
  - fundamental re-design of FEAST
  - 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 compiler switch
Integration Principles

- 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 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
Show-Case: FEASTSOLID

\[
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2
\end{pmatrix} = f
\]

\[
\begin{pmatrix}
(2\mu + \lambda)\partial_{xx} + \mu\partial_{yy} & (\mu + \lambda)\partial_{xy} \\
(\mu + \lambda)\partial_{yx} & \mu\partial_{xx} + (2\mu + \lambda)\partial_{yy}
\end{pmatrix}
\]
Mixed Precision Approach

<table>
<thead>
<tr>
<th>Level</th>
<th>single precision Error</th>
<th>Reduction</th>
<th>double precision Error</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.391E-3</td>
<td></td>
<td>2.391E-3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5.950E-4</td>
<td>4.02</td>
<td>5.950E-4</td>
<td>4.02</td>
</tr>
<tr>
<td>4</td>
<td>1.493E-4</td>
<td>3.98</td>
<td>1.493E-4</td>
<td>3.99</td>
</tr>
<tr>
<td>5</td>
<td>3.750E-5</td>
<td>3.98</td>
<td>3.728E-5</td>
<td>4.00</td>
</tr>
<tr>
<td>6</td>
<td>1.021E-5</td>
<td>3.67</td>
<td>9.304E-6</td>
<td>4.01</td>
</tr>
<tr>
<td>7</td>
<td>6.691E-6</td>
<td>1.53</td>
<td>2.323E-6</td>
<td>4.01</td>
</tr>
<tr>
<td>8</td>
<td>2.012E-5</td>
<td>0.33</td>
<td>5.801E-7</td>
<td>4.00</td>
</tr>
<tr>
<td>9</td>
<td>7.904E-5</td>
<td>0.25</td>
<td>1.449E-7</td>
<td>4.00</td>
</tr>
<tr>
<td>10</td>
<td>3.593E-4</td>
<td>0.22</td>
<td>3.626E-8</td>
<td>4.00</td>
</tr>
</tbody>
</table>

- Poisson problem with bilinear Finite Elements (Q1)
- Mixed precision solver: double precision Richardson, preconditioned with single precision MG („gain one digit“)
- Same results as entirely in double precision
Resulting Accuracy

- $L_2$ error against reference solution

- Same results for CPU and GPU
  - expected error reduction independent of refinement and subdomain distribution
(Weak) Scalability

- Outdated cluster, dual Xeon EM64T
- 1 NVIDIA Quadro FX 1400 per node
  (one generation behind the Xeons, 20 GB/s BW)
- Poisson problem (left): up to 1.3B DOF, 160 nodes
- Elasticity (right): up to 1B DOF, 128 nodes
Realistic Absolute Speedup

- 16 nodes, Opteron X2 2214
- NVIDIA Quadro FX 5600 (76 GB/s BW), OpenGL
- Problem size 128 M DOF
- Dualcore 1.6x faster than singlecore
- GPU 2.6x faster than singlecore, 1.6x than dual
Speedup Analysis

- Speedups in 'time to solution' for one GPU: 2.6x vs. Singlecore, 1.6x vs. Dualcore

- Amdahl's Law is lurking
  - Local speedup of 9x and 5.5x by the GPU
  - 2/3 of the solver accelerable => theoretical upper bound 3x

- Future work
  - Three-way parallelism in our system:
    - coarse-grained (MPI)
    - medium-grained (heterogeneous resources within the node)
    - fine-grained (compute cores in the GPU)
  - Better interplay of resources within the node
  - Adapt **Hardware-oriented Numerics** to increase accelerable part
There is a Huge Potential for the Future …

However:

- High Performance Computing has to consider recent and future hardware trends, particularly for heterogeneous multicore architectures and massively parallel systems!
- The combination of ‘Hardware-oriented Numerics’ and special ‘Data Structures/Algorithms’ and ‘Unconventional Hardware’ has to be used!

…or most of existing (academic/commercial) FEM software will be ‘worthless’ in a few years!

Let’s start with LIDO³ at the UAMR…..
Acknowledgements

- FEAST Group + LIDO Team (TU Dortmund)
- Robert Strzodka (Max Planck Center, Max Planck Institut Informatik)
- Jamaludin Mohd-Yusof, Patrick McCormick (Los Alamos National Laboratories)
Generalized Tensorproduct
Meshes

….dynamic CFD problems…..
Solver Structure

ScaRC -- Scalable Recursive Clustering

- Minimal overlap by extended Dirichlet BCs
- Hybrid multilevel domain decomposition method
- Inspired by parallel MG ("best of both worlds")
  - Multiplicative vertically (between levels), global coarse grid problem (MG-like)
  - Additive horizontally: block-Jacobi / Schwarz smoother (DD-like)
- Hide local irregularities by MGs within the Schwarz smoother
- Embed in Krylov to alleviate Block-Jacobi character
Show-Case: FEASTSolid

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

\[
\begin{pmatrix}
A_{11} & A_{12} & B_1 \\
A_{21} & A_{22} & B_2 \\
B_1 & B_2 & C
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
p
\end{pmatrix}
= 
\begin{pmatrix}
f_1 \\
f_2 \\
g
\end{pmatrix}
\]

- ★ 4-node cluster
- ★ Opteron X2 2214
- ★ GeForce 8800 GTX (90 GB/s BW), CUDA
- ★ Driven cavity and channel flow around a cylinder

**fixed point iteration**
- solving linearised subproblems with
  - **global BiCGStab** (reduce initial residual by 1 digit)
  - Block-Schurcomplement preconditioner
    - 1) approx. solve for velocities with
      - **global MG** (V1+0), additively smoothed by
        - for all $\Omega_i$: solve for $u_1$ with
        - **local MG**
          - for all $\Omega_i$: solve for $u_2$ with
  - **local MG**
    - 2) update RHS: $d_3 = -d_3 + B(c_1, c_2)$
    - 3) scale $c_3 = (M_p^L)d_3$
Navier-Stokes results

**Speedup analysis**

<table>
<thead>
<tr>
<th></th>
<th>( R_{acc} )</th>
<th>( S_{local} )</th>
<th>( S_{total} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L9</td>
<td>L10</td>
<td>L9</td>
</tr>
<tr>
<td>DC Re100</td>
<td>41%</td>
<td>46%</td>
<td>6x</td>
</tr>
<tr>
<td>DC Re250</td>
<td>56%</td>
<td>58%</td>
<td>5.5x</td>
</tr>
<tr>
<td>Channel flow</td>
<td>60%</td>
<td>–</td>
<td>6x</td>
</tr>
</tbody>
</table>

**Important consequence:**
Ratio between assembly and linear solve changes significantly

<table>
<thead>
<tr>
<th></th>
<th>DC Re100</th>
<th>DC Re250</th>
<th>Channel flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain accel.</td>
<td>plain</td>
<td>accel.</td>
<td></td>
</tr>
<tr>
<td>29:71</td>
<td>11:89</td>
<td>25:75</td>
<td>13:87</td>
</tr>
<tr>
<td>50:48</td>
<td>26:74</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Acceleration analysis

Speedup analysis
★ Addition of GPUs increases resources
★ ⇒ Correct model: strong scalability inside each node
★ Accelerable fraction of the elasticity solver: 2/3
★ Remaining time spent in MPI and the outer solver

Accelerable fraction $R_{acc}$: 66%
Local speedup $S_{local}$: 9x
Total speedup $S_{total}$: 2.6x
Theoretical limit $S_{max}$: 3x
Minimally invasive integration

global BiCGStab
preconditioned by

global multilevel $(V \, 1+1)$

additively smoothed by

for all $\Omega_i$: local multigrid

coarse grid solver: UMFPACK

- All outer work: CPU, double
- Local MGs: GPU, single
- GPU is preconditioner
- Applicable to many co-processors
Grid Structures

Fully adaptive grids

- Maximum flexibility
- 'Stochastic' numbering
- Unstructured sparse matrices
- Indirect addressing (very slow)

Locally structured grids

- Logical tensor product
- Fixed banded matrix structure
- Direct addressing (fast)
- r-adaptivity

Unstructured macro mesh of tensor product subdomains