Computing on GPU Clusters

Robert Strzodka (MPII), Dominik Göddeke (TUDo), Dominik Behr (AMD)

PPAM 2009 - Conference on Parallel Processing and Applied Mathematics
Wroclaw, Poland, September 13-16, 2009

www.gpgpu.org/ppam2009
Overview

• **Hardware**
  - Which hardware to use?
  - Consumer hardware?
  - AMD FireStream and NVIDIA Tesla
  - Guidelines

• **Case study: FEAST**
Which hardware to use?

• **Considerations:**
  – Workstation form factor vs. dense cluster installations
  – How many GPUs can you cram into one motherboard?
  – Energy requirements
  – Cooling
  – Lifetime and reliability (24/7 „gaming“?)

• **Two main choices**
  – NVIDIA Tesla and AMD FireStream
  – Consumer boards are great for development, but...
Consumer hardware?

• **Professional compute boards are more expensive**
  – 3-5x compared to high-end gaming hardware

• **Benefits**
  – Designed to run 24/7
  – Much more rigorous testing by the manufacturers
  – Much more memory (up to 4 GB)
  – Extended warranty
  – Enterprise-level support

• **Drawback**
  – Slightly downclocked memory to improve stability
  – No ECC memory (not a problem in my experience)
Availability

• **My advice**
  – Develop on mid-range consumer card (easier not to overlook bottlenecks)
  – Deploy to high-end clusters once it runs

• **HPC GPU clusters are becoming available**
  – Teragrid (Lincoln)
  – National supercomputing centers
    – Jülich, Stuttgart, Heidelberg in Germany
  – For research projects
    – Often just a question of asking nicely
    – Same application rules as for other Top500 clusters
AMD FireStream

• **AMD FireStream 9270, 9250, 9170**

• **High-end model specs**
  – 1.2 TFLOP/s (single), 240 GFLOP/s (double) peak floating point
  – 2 GB GDDR5
  – 108.8 GB/s memory bandwidth
  – 160 watts typical, <220 watts peak
  – Dual-width full length PCIe x16 Gen2

• **More information**
  – http://developer.amd.com/gpu/ATIStreamSDK/Pages/default.aspx
NVIDIA Tesla C1060

• **Specs**
  - 933 GFLOP/s (single), 78 GFLOP/s (double) peak floating point
  - 4 GB GDDR3
  - 102 GB/s memory bandwidth
  - 187.8 watts peak
  - Dual-width full length PCIe x16 Gen2

• **More information**

• **Personal supercomputer offering**
  - Preconfigured high-end workstation with up to 4 C1060s
NVIDIA Tesla S1070

• **Essentially, 4 C1060s in one blade**
  – 1U standard form factor
  – Own power supply
  – Proprietary PCIe interconnect
  – Needs host system (2 x16 Gen2 PCIe for one S1070)
    – Potential bus contention, but each two GPUs get full bandwidth if transferring alone

• **More information**
Guidelines

• **Balance host and GPUs**
  – Rule of thumb: One CPU core per GPU
  – Don’t forget about host performance and host memory
    – Not all tasks are suitable for the GPU

• **At the very least, host does MPI**
  – Or SMP-threading in a workstation
  – GPUs don’t have network interconnects
  – Data travels over PCIe from GPU to host, over Infiniband to next host, and back to its GPU

• **These rules apply to workstations and dense clusters**
Overview

• Hardware

• Case study (I): FEAST
  – Parallel multilevel/multigrid finite element solvers
  – Minimally invasive co-processor integration
  – Applications: Linearized elasticity and fluid dynamics
  – Exemplary results
Mesh structure

**Fully adaptive grids**
- Maximum flexibility
- 'Stochastic' numbering
- Unstructured matrices
- Very slow

**Locally structured grids**
- Logical tensor product
- Fixed banded structure
- Direct addressing
- Fast

Unstructured macro mesh of tensor product subdomains
FEAST grids
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)
    - Additive horizontally, block-Jacobi / Schwarz smoother (DD)
  - Hide local irregularities by MGs within the Schwarz smoother
  - Embed in Krylov to alleviate block-Jacobi Character

```plaintext
global BiCGStab
preconditioned by

global multilevel (V 1+1)
additively smoothed by

for all Ω_j: local multigrid

coarse grid solver: UMFPACK
```
Vector-valued problems

• **Block-structured systems**
  - Guiding idea: Tune scalar case once per architecture instead of over and over again per application
  - Equation-wise ordering of the unknowns
  - Block-wise treatment enables multivariate ScaRC solvers

• **Examples**
  - Linearized elasticity with compressible material (2x2 blocks)
  - Saddle point problems: Stokes, elasticity with (nearly) incompressible material, Navier-Stokes with stabilization (3x3 blocks, 3 zeros for Stokes)
  - Diagonal blocks correspond to scalar elliptic operators

\[
\begin{pmatrix}
A_{11} & A_{12} \\
A_{12}^T & A_{22}
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2
\end{pmatrix}
= f
\begin{pmatrix}
A_{11} & 0 & B_1 \\
0 & A_{22} & B_2 \\
B_1^T & B_2^T & 0
\end{pmatrix}
\begin{pmatrix}
v_1 \\
v_2 \\
p
\end{pmatrix}
= f
\begin{pmatrix}
A_{11} & A_{12} & B_1 \\
A_{12}^T & A_{22} & B_2 \\
B_1^T & B_2^T & C_c
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
p
\end{pmatrix}
= f
\]
Project goals

• **Challenge of co-processor integration**
  – Significant reimplementations are prohibitive
  – In particular for large, established codes

• **Balance needed**
  – Actual hardware is evolving too rapidly
  – Integration should be reasonably future-proof
  – For several co-processor(s) and generations

• **Our approach: High level of abstraction**
  – *Minimally invasive co-processor integration*
  – Identify and isolate "accelerable" parts of a computation
  – Chunks must be large enough to amortise co-processor overhead
  – Encapsulate several co-processors under one interface
Project goals

• **This project is not about...**
  – Mapping a single application to one GPU
  – Mapping a single application to a GPU cluster
  – Reporting „the optimal speedup“

• **But about...**
  – Integrating several co-processors
  – Into an existing large-scale software package
  – *Without modifying application code*

• **Ultimate goal**
  – Hybridization of resources
  – Automatic scheduling across cores and devices
Bandwidth in a CPU-GPU system

1-2 GB/s
Infiniband to next node

6-30 GB/s

system memory

20-160 GB/s

device memory

40 GB/s

co-processor

Bandwidth in a CPU-GPU system

processing elements

CPU

cache
Co-processor evaluation

- **Evaluation strategy**
  - Perform experiments before attempting co-processor integration
  - Try to select relevant tests
  - Do not focus on microbenchmarks alone
  - Compute-bound codes on the CPU can be memory-bound on co-processors

- **Key tests for us**
  - Performance of matrix-vector multiplication typically a good estimate of overall performance
  - Performance of multigrid solver on „one subdomain“
  - FE solvers are prone to suffer from reduced accuracy (high condition number)
  - Poisson equation as representative for scalar elliptic PDEs
Test 1 - Microbenchmarks

• **Model problem**
  – Poisson on unit square, generalized tensor product mesh
  – *not just a matrix stencil!*
  – Conforming bilinear quadrilateral elements (Q₁), up to 1M DOF

• **Benchmark results: d=b-Ax**
  – GeForce GTX 280
    – Single precision: 46 GFLOP/s (5% peak), 50x speedup
      – ~860 MFLOP/s on Opteron 2214 single-threaded
      – ~2 GFLOP/s on PlayStation 3 => not worth the effort
    – Double precision: 17 GFLOP/s (22% peak), 40x speedup
      – ~440 MFLOP/s on Opteron 2214 single-threaded
      – ~600 MFLOPs on PlayStation 3
  – Double precision implementation cannot exploit on-chip texture cache

• **So far: quite promising (but we ruled out the Cell 😊)**
### Test 2 - Accuracy

Laplacian of analytic test function as RHS to Poisson equation
Multigrid solver, first in double then in single precision
Expect L2 error reduction by a factor of 4 per refinement step

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>Double Reduction</th>
<th>Single Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3^2$</td>
<td>5.208e-3</td>
<td>5.208e-3</td>
</tr>
<tr>
<td>$5^2$</td>
<td>1.440e-3</td>
<td>3.62</td>
</tr>
<tr>
<td>$9^2$</td>
<td>3.869e-4</td>
<td>3.72</td>
</tr>
<tr>
<td>$17^2$</td>
<td>1.015e-4</td>
<td>3.81</td>
</tr>
<tr>
<td>$33^2$</td>
<td>2.607e-5</td>
<td>3.89</td>
</tr>
<tr>
<td>$65^2$</td>
<td>6.612e-6</td>
<td>3.94</td>
</tr>
<tr>
<td>$129^2$</td>
<td>1.666e-6</td>
<td>3.97</td>
</tr>
<tr>
<td>$257^2$</td>
<td>4.181e-7</td>
<td>3.98</td>
</tr>
<tr>
<td>$513^2$</td>
<td>1.047e-7</td>
<td>3.99</td>
</tr>
<tr>
<td>$1025^2$</td>
<td>2.620e-8</td>
<td>4.00</td>
</tr>
</tbody>
</table>

Not promising at all, advanced techniques required
Test 3: Mixed precision multigrid

<table>
<thead>
<tr>
<th>Level</th>
<th>Core2Duo (double)</th>
<th>GTX 280 (mixed)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time(s)</td>
<td>MFLOP/s</td>
</tr>
<tr>
<td>7</td>
<td>0.021</td>
<td>1405</td>
</tr>
<tr>
<td>8</td>
<td>0.094</td>
<td>1114</td>
</tr>
<tr>
<td>9</td>
<td>0.453</td>
<td>886</td>
</tr>
<tr>
<td>10</td>
<td>1.962</td>
<td>805</td>
</tr>
</tbody>
</table>

Same model problem as before
Mixed precision computes exactly the same solution as pure double
1M DOF, FE accurate in less than 0.1 seconds!
27x faster than CPU
1.7x faster than pure double on GPU
8800 GTX (correction loop on CPU): 0.44 seconds for L=10
Solver analysis

- Identify accelerable parts of the solver

- Global BiCGStab
  - Preconditioned by

- Global MG
  - Smoothed by

- Local MGs per subdomain

Realised as a series of local operations on each subdomain.
Typically one operation (defect calculation, grid transfers etc.) directly followed by neighbor communication (MPI).

1-2 full MG cycles with up to 1e6 unknowns.
Poor acceleration potential due to PCIe bottleneck.

Good spatial locality.
Enough work for fine-grained parallelism.
Good acceleration potential.
Only accelerate scalar solvers.
Integration overview

**global BiCGStab**
preconditioned by
**global multilevel** ($V$ 1+1)
additively smoothed by
for all $\Omega_i$: **local multigrid**

coarse grid solver: UMFPACK
Integration summary

- **Isolate suitable parts**
  - Balance acceleration potential and acceleration effort

- **Diverge code paths as late as possible**
  - Local scalar 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
  - But: prone to suffer from Amdahl's Law (discussed later)
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

- **Data transfers to and from device**
  - Device memory model: huge L3 cache with prefetching
  - Automatic prefetching: Backend copies all matrix data in preprocessing
  - Manual prefetching: Round-robin for devices with small memory

- **Precision vs. accuracy**
  - Double precision needed, but only at crucial stages of the computation
  - Mixed precision iterative refinement approach
  - Outer solver: high precision
  - Inner solver: low precision
  - Accuracy not affected (see results in a minute)
Test goals

• **Accuracy and impact of reduced precision**

• **Scalability and Speedup**
  – Detailed analysis and understanding of speedup components

• **For simplicity**
  – Detailed study only for linearized elasticity
  – Speedup analysis for both applications

• **Navier-Stokes is a stress test of our minimally invasive approach**
  – „only“ the local scalar velocity „solves“ can be accelerated
  – Goal: push our approach to its limits
Linearized elasticity

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

$$(2\mu + \lambda)\partial_{xx} + \mu\partial_{yy} \quad (\mu + \lambda)\partial_{xy}$$

$$(\mu + \lambda)\partial_{yx} \quad \mu\partial_{xx} + (2\mu + \lambda)\partial_{yy}$$

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

- for all $\Omega_i$: solve $A_{11}c_1 = d_1$ by local scalar multigrid
- update RHS: $d_2 = d_2 - A_{21}c_1$
- for all $\Omega_i$: solve $A_{22}c_2 = d_2$ by local scalar multigrid
- coarse grid solver: UMFPACK
Accuracy

L2 error against analytically known displacements

Same results for CPU and GPU

Expected error reduction independent of refinement and subdomain distribution

Up to 32 nodes, 512M DOF
Very ill-conditioned systems

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

<table>
<thead>
<tr>
<th>refinement L</th>
<th>Iterations</th>
<th>Volume</th>
<th>y-Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU</td>
<td>GPU</td>
<td>CPU</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
<td>1.6087641E-3</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>4</td>
<td>1.6087641E-3</td>
</tr>
<tr>
<td>10</td>
<td>4.5</td>
<td>4.5</td>
<td>1.6087641E-3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>aniso16</th>
<th>Iterations</th>
<th>Volume</th>
<th>y-Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU</td>
<td>GPU</td>
<td>CPU</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>6</td>
<td>6.7176398E-3</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>5.5</td>
<td>6.7176427E-3</td>
</tr>
<tr>
<td>10</td>
<td>5.5</td>
<td>5.5</td>
<td>6.7176516E-3</td>
</tr>
</tbody>
</table>
Weak scalability

Old cluster, dual Xeon EM64T 3.4 GHz, 1 MB L2 cache
one Quadro FX 1400 per node (20 GB/s bandwidth)
GPUs one generation behind the Xeons (ignore speedup)
Poisson (left): up to 1.3 B DOF, 160 nodes
Elasticity (right): up to 1 B DOF, 128 nodes
Absolute speedup

16 nodes, Opteron 2214 dualcore (1 MB L2 cache, 2.2 GHz)
Quadro 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 faster than dualcore
Speedup analysis

• Model
  – Addition of GPUs increases resources
  – Correct model: strong scalability inside each node
  – Detailed measurements: 2/3 of entire solver can be accelerated
  – Remaining time spent in MPI and outer solver

Accel. fraction $R_{acc}: 66\%$
Local speedup $S_{local}: 9x$
Global speedup $S_{total}: 2.6x$
Theoretical limit $S_{max}: 3x$
Stokes results

Opteron 2214, 4 nodes
GeForce 8800 GTX (86 GB/s)
CUDA backend
18.8 M DOF

Accel. fraction $R_{\text{acc}}$: 75%
Local speedup $S_{\text{local}}$: 11.5x
Global speedup $S_{\text{total}}$: 3.8x
Theoretical limit $S_{\text{max}}$: 4x

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

**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

2) update RHS: $d_3 = -d_3 + B^T(c_1,c_2)^T$
3) scale $c_3 = (M_p^L)^{-1}d_3$
Navier-Stokes test cases

Driven cavity and channel flow
Navier-Stokes results

Speedup analysis

<table>
<thead>
<tr>
<th></th>
<th>$R_{\text{acc}}$</th>
<th>$S_{\text{local}}$</th>
<th>$S_{\text{total}}$</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 solving changes

<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>29:71</td>
<td>50:48</td>
<td>13:87</td>
</tr>
<tr>
<td></td>
<td>11:89</td>
<td>25:75</td>
<td>26:74</td>
</tr>
</tbody>
</table>
Summary and conclusions

• Identify accelerable parts of established code
  – Balance acceleration potential and acceleration effort

• Minimally invasive integration without changes to application code
  – Instead, hardware acceleration available under the same interface
  – User gets acceleration for free, just a change in a parameter file

• Good speedups
  – Limited by Amdahl's Law
  – Future work needs to address 3-way parallelism
    – coarse-grained (MPI between nodes)
    – medium-grained (resources within the node)
    – fine-grained (compute cores in the GPUs)
  – Designing solvers with a higher acceleration potential
  – Accelerate parts of the outer solver on double precision GPUs
Acknowledgements

• Hilmar Wobker, Sven Buijssen, Christian Becker, Stefan Turek (TU Dortmund)

• Robert Strzodka (Max Planck Institut Informatik Saarbrücken)

• Patrick McCormick, Jamaludin Mohd-Yusof (Los Alamos National Laboratory)

• Supported in parts by
  • DFG: projects TU102/22-1, TU102/22-2, TU102/27-1, TU102/11-3
  • BMBF: HPC Software für skalierbare Parallelrechner, SKALB project, 01IH08003D / SKALB