Simulation and Animation
Using 294,912 Processor Cores


U. Rüde (LSS Erlangen, ruede@cs.fau.de)

Lehrstuhl für Informatik 10 (Systemsimulation)
Universität Erlangen-Nürnberg
www10.informatik.uni-erlangen.de

Kolloquium
Universität Passau
25. Oktober 2011
Overview

Motivation: How fast are computers today (and tomorrow)

How fast should our Solvers be

- **Scalable Parallel Multigrid** Algorithms for PDE
- Matrix-Free Multigrid FE solver: Hierarchical Hybrid Grids (HHG)

A Multi-Scale & Multi-Physics Simulation (with grain size resolution)

- Flow Simulation with **Lattice Boltzmann** Methods
- Software Architecture
- Performance Engineering on GPU clusters

Conclusions
Motivation
How much is ExaFlops?

- $10^6 = 1$ MegaFlops: Intel 486
  33MHz PC (~1989)
- $10^9 = 1$ GigaFlops: Intel Pentium III
  1GHz (~2000)
  - If every person on earth computes one operation every 6 seconds, all humans together have ~1 GigaFlops performance (less than a current laptop)
- $10^{12} = 1$ TeraFlops: HLRB-I
  1344 Proc., ~2000
- $10^{15} = 1$ PetaFlops
  - 122 400 Cores (Roadrunner, 2008)
  - 294 912 Cores (Jugene, Jülich, 1.44 $10^{14}$ Bytes Memory)
- If every person on earth runs a 486 PC, we all together have an aggregate Performance of 6 PetaFlops.
- ExaScale ($\sim10^{18}$ Flops) around 2018?

HLRB-I: 2 TFlops

HLRB-II: 63 TFlops
Example Peta-Scale System: Jugene @ Jülich

- PetaFlops = $10^{15}$ operations/second
- IBM Blue Gene
- Theoretical peak performance: 1.0027 Petaflop/s
- 294,912 cores
- 144 TBytes = $1.44 \times 10^{14}$
- #9 on TOP 500 List in Nov. 2010

For comparison: Current fast desktop PC is ~ 20,000 times slower

> 1,000,000 cores expected 2011

Exa-Scale System expected by 2018/19 ... likely with $\sim 10^9$ cores
What will Computers Look Like in 2020?

- **Super Computer (Heroic Computing)**
  - Cost: 200 Million €
  - Parallel Threads: $10^8 - 10^9$
  - $10^{18}$ FLOPS, Mem: $10^{15}-10^{17}$ Byte (1-100 PByte)
  - Power Consumption: 20 MW

- **Departmental Server (Mainstream Computing for R&D)**
  - Cost: 200 000 €
  - Parallel Threads: $10^5 - 10^6$
  - $10^{15}$ FLOPS, Mem: $10^{12}-10^{14}$ Byte (1-100 TByte)
  - Power Consumption: 20 KW

- **(mobile) Workstation (Computing for the Masses)**
  - ... scale down by another factor 100

But remember: Predictions are difficult ... especially those about the future.
What are the consequences?

- For the application developers “the free lunch is over”
  - Without explicitly parallel algorithms, the performance potential cannot be used any more

- For HPC
  - CPUs will have 2, 4, 8, 16, ..., 128, ..., ??? cores - maybe sooner than we are ready for this
  - We will have to deal with systems with millions of cores

- The memory wall grows higher
How Fast

should our simulations be

... and why they aren’t
How fast should our algorithms (multigrid) be on current architectures (single CPU)

**Assumptions:**
- Multigrid requires 27.5 Ops/unknown to solve an elliptic PDE (Hackbusch 85, Griebel ’89 for 2-D Poisson)
- A modern CPU delivers > 30 GFlops peak

**Consequence:**
- We should solve one million unknowns in less than 0.001 seconds
- \~ 1 ns per unknown

**Revised Assumptions:**
- Multigrid takes 300 Ops/unknown to solve your favorite PDE
- you can get 10\% of 30 Gflops performance

**Consequence:** On your laptop you should
- solve one million unknowns in 0.1 second
- \~ 0.1 microsecond per unknown

**These predictions are unrealistic in current computational practice by several orders of magnitude**

**Gap between traditional math analysis and real life performance widens drastically**
Multigrid: V-Cycle

**Goal:** solve $A^h u^h = f^h$ using a hierarchy of grids

- **Relax on Residual**
  - $r^h = f^h - A^h u^h$

- **Restrict**
  - $r^H = I_h^H r^h$

- **Interpolate**
  - $e^h = I_H^h e^H$

- **Solve**
  - $A^H e^H = r^H$

by recursion
HHG refinement example

Input Grid
HHG Refinement example

Refinement Level one
HHG Refinement example

Refinement Level Two
HHG Refinement example

Structured Interior
HHG Refinement example

Structured Interior
HHG Refinement example

Edge Interior
HHG Refinement example

Edge Interior
HHG Parallel Update Algorithm

for each vertex do
    apply operation to vertex
end for

update vertex primary dependencies

for each edge do
    copy from vertex interior
    apply operation to edge
    copy to vertex halo
end for

update edge primary dependencies

for each element do
    copy from edge/vertex interiors
    apply operation to element
    copy to edge/vertex halos
end for

update secondary dependencies
## Multigrid-FE Scalability on Jugene

<table>
<thead>
<tr>
<th>Cores</th>
<th>Struct. Regions</th>
<th>Unknowns</th>
<th>CG</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1 536</td>
<td>534 776 319</td>
<td>15</td>
<td>5.64</td>
</tr>
<tr>
<td>256</td>
<td>3 072</td>
<td>1 070 599 167</td>
<td>20</td>
<td>5.66</td>
</tr>
<tr>
<td>512</td>
<td>6 144</td>
<td>2 142 244 863</td>
<td>25</td>
<td>5.69</td>
</tr>
<tr>
<td>1024</td>
<td>12 288</td>
<td>4 286 583 807</td>
<td>30</td>
<td>5.71</td>
</tr>
<tr>
<td>2048</td>
<td>24 576</td>
<td>8 577 357 823</td>
<td>45</td>
<td>5.75</td>
</tr>
<tr>
<td>4096</td>
<td>49 152</td>
<td>17 158 905 855</td>
<td>60</td>
<td>5.92</td>
</tr>
<tr>
<td>8192</td>
<td>98 304</td>
<td>34 326 194 175</td>
<td>70</td>
<td>5.86</td>
</tr>
<tr>
<td>16384</td>
<td>196 608</td>
<td>68 669 157 375</td>
<td>90</td>
<td>5.91</td>
</tr>
<tr>
<td>32768</td>
<td>393 216</td>
<td>137 355 083 775</td>
<td>105</td>
<td>6.17</td>
</tr>
<tr>
<td>65536</td>
<td>786 432</td>
<td>274 743 709 695</td>
<td>115</td>
<td>6.41</td>
</tr>
<tr>
<td>131072</td>
<td>1 572 864</td>
<td>549 554 511 871</td>
<td>145</td>
<td>6.42</td>
</tr>
<tr>
<td>262144</td>
<td>3 145 728</td>
<td>1 099 176 116 223</td>
<td>180</td>
<td>6.81</td>
</tr>
</tbody>
</table>

This runs only at roughly 7% of Jugene peak!
Pessimizing the Performance

- Bring loops in wrong order, ignore caches, randomize memory access, use many small MPI messages
  - $10^{12} \rightarrow 10^{11}$ unknowns
- Do not use a matrix-free implementation (keep in mind that a single multiplication with the mass and stiffness can easily cost 50 mem accesses per unknown):
  - $10^{11} \rightarrow 10^{10}$ unknowns
- Gain additional flexibility by using unoptimized unstructured grids (indirect mem access costs!)
  - $10^{10} \rightarrow 10^{9}$ unknowns
- Increase algorithmic overhead, e.g. permanently checking convergence, use the most expensive error estimator, etc. etc.
  - $10^{9} \rightarrow 10^{8}$ unknowns ( ... still a large system ... )
Computational Fluid Dynamics with the Lattice Boltzmann Method

Falling Drop with Turbulence Model (slow motion)
Walberla Software Framework
Computational Fluid Dynamics
Lattice Boltzmann Method

- 1000 Bubbles
  - 510x510x530 = $1.4 \cdot 10^8$ lattice cells
- 70,000 time steps
- 77 GB
- 64 processes
- 72 hours
- 4,608 core hours

Visualization
- 770 images
- Approx. 12,000 core hours for rendering

Best Paper Award for Stefan Donath (LSS Erlangen) at ParCFD, May 2009 (Moffett Field, USA)
Fluid-Structure Interaction with Free Surface and Moving Objects
Fluidization
Heterogenous CPU-GPU Simulation

Particles: 31250  Domain: 400x400x200  Timesteps: 400 000
Devices: 2 x M2070 + 1 Intel „Westmere“  Runtime: 17.5 h
What can we do with Exa-Scale Computers(2)?

Fluidized Bed

Even if we want

- to simulate a billion \((10^9)\) objects (particles): we can do a billion \((10^9)\) operations for each of them in each second
- a trillion \((10^{12})\) finite elements (finite volumes) to resolve the flow between particles: we can do a million \((10^6)\) operations for each of them in each second

Fluidized Bed

(movie: thanks to K.E. Wirth, Erlangen)
Fluid-Structure Interaction for particle laden flows
Domain Partitioning and Parallelization
Weak Scaling

Jugene
Blue Gene/P

Largest simulation to date:
8 Trillion ($10^{12}$) variables per time step

Densely packed particles
264 331 905 rigid spherical objects
The Lattice-Boltzmann-Method

- Discretization in squares or cubes (cells)
- 9 numbers per cell (or 19 in 3D)
  = number of particles traveling towards neighboring cells
- Repeat (many times)
  - stream
  - collide
The stream step

Move particle (numbers) into neighboring cells
The collide step

Compute new particle numbers according to the collisions
The Interface Between Liquid and Gas

- Volume-of-Fluids like approach
- Flag field: Compute only in fluid
- Special “free surface” conditions in interface cells
Performance Engineering for LBM on GPU Clusters
Tsubame 2

- 1408 compute nodes equipped with GPUs
- 3 NVIDIA Tesla M2050 per node
- Peak performance:
  - 2.2 PFlop/s
  - 633 TB/s memory bandwidth
- Total performance: 2.4 PFlops/s
- 5th in the TOP500 list
- Located at Tokyo Institute of Technology, Japan
- Collaboration with Prof. Takayuki Aoki
Pure LBM Performance on Tsubame 2

- MLUPS: Mega Lattice Updates per Seconds
- Pure LBM performance is limited by bandwidth
- Implementation in CUDA
- Scenario: Lid Driven Cavity

<table>
<thead>
<tr>
<th></th>
<th>NVIDIA Tesla M2050</th>
<th>Xeon X5670 „Westmere“</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 sockets</td>
<td>2 sockets</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12 cores</td>
<td>12 cores</td>
<td></td>
</tr>
<tr>
<td>Flops [ TFlop/s ]</td>
<td>1.0 / 0.5</td>
<td>0.25/0.13</td>
<td>x 4</td>
</tr>
<tr>
<td>Theoretical Peak</td>
<td>148</td>
<td>64</td>
<td>x 2-3</td>
</tr>
<tr>
<td>Bandwidth [ GB/s ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stream Copy</td>
<td>100(+ECC)/115(-ECC)</td>
<td>43</td>
<td>x 2-3</td>
</tr>
<tr>
<td>Bandwidth [ GB/s ]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Single GPU and CPU Node Performance

Performance estimates based in Stream bandwidth:

- CPU: 142 MLUPS (ECC, DP, -BC)
- GPU: 330 MLUPS (ECC, DP, -BC)

Resulting performance 75 % of estimate (+BC)

CPU Kernel:
- SSE Intrinsics
- Non-temporal stores
- Padding

GPU Kernel:
- Register usage optimized
- Memory layout: SoA
- Padding

Kernels implemented by Johannes Habich
3 GPU Overlapping Communication

Timeline

Comm

Outer Kernel

Inner Kernel

Cubic Domain Size per Device

Streaming 0

Streaming 1

Streaming 2
Weak Scaling Performance
Weak Scaling Performance
GPU & CPU: Heterogeneous LBM

Requirements:
- Different data structures
- Different kernels
- Common communication interface
- Load balancing

Node Setup:
- 1 MPI Process per Core
- 1 MPI Process per Socket
- 1 MPI Process per Node
Heterogeneous CPU & GPU Performance

on TinyGPU @ RRZE: 2 x Tesla M2070
CPU vs GPU: LBM Implementation Effort

- Subjective evaluation
- Valid for the pure LBM implementation
- Partly for stencil based methods

<table>
<thead>
<tr>
<th>Difficulty</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Intra Node</td>
<td>★</td>
<td>★★★</td>
</tr>
<tr>
<td>Inter Node</td>
<td>★★★</td>
<td>★★★★</td>
</tr>
</tbody>
</table>
Conclusions

*Exa-Scale will be Easy!*

- If parallel efficiency is bad, choose a slower serial algorithm
  - it is probably easier to parallelize
  - and will make your speedups look much more impressive

- Introduce the “CrunchMe” variable for getting high Flops rates
  - advanced method: disguise CrunchMe by using an inefficient (but compute-intensive!) algorithm from the start

- Introduce the “HitMe” variable to get good cache hit rates
  - advanced version: Implement HitMe in the “Hash-Brown Lookaside Table for the Multi-Threatened Cash-Filling Clouded Tree” data structure
  - ... impressing yourself and others

- Never cite “time-to-solution”
  - who cares whether you solve a real-life problem anyway
  - it is the MachoFlops that interest the people who pay for your research

- Never waste your time by trying to use a complicated algorithm in parallel
  - Use Primitive Algorithm => Easy to Maximize your MachoFlops
  - A few million CPU hours can easily save you days of reading in boring math books
Current Research Challenges!

- Think about possible new modeling paradigms!
  - can we do process scale direct simulation fully resolving the level of
    • pores?
    • grains?
    • fibers?
    • molecules?
    • atoms?

- Performance? We do not have good metrics!
  - $O(N^q)$ complexity versus time to solution
  - or $O(h^p)$ convergence versus accuracy per MWh

- We will be drowned by the Tsunami of Parallelism

- Validation?
  - rigorous theory vs. experimental validation

- „Abstract“ numerical algorithms vs. high quality software

- Monolithic models vs. coupling a variety of „algorithmic building blocks“
Acknowledgements

Collaborators

- In Erlangen: WTM, LSE, LTM, LSTM, LGDV, RRZE, LME, Neurozentrum, Radiologie, Applied Mathematics, Theoretical Physics, etc. Especially for foams: C. Körner (WTM) & A. Delgado (LSTM)
- International: Utah, Technion, Constanta, Ghent, Boulder, München, CAS, Zürich, Delhi, Tokyo, ...

Dissertationen Projects

- N. Thürey, T. Pohl, S. Donath, S. Bogner, C. Godenschwager (LBM, free surfaces, 2-and 3-phase flows)
- M. Kowarschik, J. Treibig, M. Stürmer, J. Habich (architecture aware algorithms)
- K. Iglberger, T. Preclik, K. Pickel (rigid body dynamics)
- J. Götz, C. Feichtinger, F. Schornbaum (Massively parallel LBM software, suspensions)
- C. Mihoubi, D. Bartuschat (Complex geometries, parallel LBM)

(Long Term) Guests in summer/fall 2009-11:

- Dr. S. Ganguly, Prof. S. Chakraborty, IIT Kharagpur (Humboldt) - Electroosmotic Flows
- Prof. V. Buwa, IIT Delhi (Humboldt) - Gas-Fluid-Solid flows
- Felipe Aristizabal, McGill Univ., Canada (LBM with Brownian Motion)
- Prof. Popa, Constanta, Romania (DAAD) Numerical Linear Algebra
- Prof. Steve Roberts, Prof. Linda Stals, Australian National University
- Prof. Hanke, Prof. Oppelstrup, Prof. Edsberg, KTH Stockholm (DAAD), Mathematical Modelling

~30 Diplom-/Master- Thesis, ~35 Bachelor Thesis

Funding by KONWIHR, DAAD, DFG, BMBF, EU, Elitenetzwerk Bayern
Thank you for your attention!

Questions?

Slides, reports, thesis, animations available for download at:
www10.informatik.uni-erlangen.de