Architecture Aware Multigrid

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

joint work with

and many more students

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

SIAM CSE 09
March 2009
Overview

- Motivation:
  - Towards PetaScale and Beyond

- Efficient Parallel Multigrid Software
  - Data Local Iterative Methods (DiMe)
  - MultiCore MultiGrid on the IBM-Cell Processor
  - Hierarchical Hybrid Grids (HHG)
  - Parallel Expression Templates for PDE (ParExPDE)

- Conclusions
Part I
Towards PetaScale and Beyond
The Multigrid Promise

Assumptions:
- Multigrid requires 27.5 Ops/unknown to solve an elliptic PDE (Griebel 89 for Poisson)
- A modern laptop CPU delivers ~10 GFlops peak

Consequence:
- We should solve one million unknowns in 0.00275 seconds
- ~ 3 ns per unknown

Revised Assumptions:
- Multigrid takes 500 Ops/unknown to solve your favorite PDE
- you can get 5% of 10 Gflops performance

Consequence: On your laptop you should
- solve one million unknowns in 1.0 second
- ~ 1 microsecond per unknown

Also consider: A banded Gaussian elimination on the Sony Play Station (IBM Cell processor) will need about 15 seconds for 1000 x 1000 unknowns
Trends in Computer Architecture

- On Chip Parallelism
  - instruction level
  - multicore
- Off Chip parallelism
- Limits to clock rate
- Limits to memory bandwidth and latency
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
Part II

DiMe
Cache-Aware Multigrid
DiMe - Project

Data Local Iterative Methods (DFG 1996-2007) for the
Efficient Solution of Partial Differential Equations
www10.informatik.uni-erlangen.de/de/Research/Projects/DiME/

Single core optimization

- Started jointly with Linda Stals in 1996!
- work by M. Kowarschik, J. Treibig, F. Hülseemann, in collaboration with A. Bode, TUM
- Cache-optimizations for sparse matrix/stencil codes (1996-2007)
- Also used in Lattice-Boltzmann CFD
## V(2,2) cycle - bottom line
(old results)

<table>
<thead>
<tr>
<th>Mflops</th>
<th>For what</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Standard 5-pt. Operator</td>
</tr>
<tr>
<td>56</td>
<td>Cache optimized (loop orderings, data merging, simple blocking)</td>
</tr>
<tr>
<td>150</td>
<td>Constant coeff. + skewed blocking + padding</td>
</tr>
<tr>
<td>220</td>
<td>Eliminating rhs if 0 everywhere but boundary</td>
</tr>
</tbody>
</table>
Part III

Multicore Architectures

Multigrid on STI Cell
The STI Cell Broadband Engine

Element Interconnect Bus
- 4 rings
- up to 204.8 GB/s

Synergistic Processor Element
Memory Flow Controller
- "interface controller" of the SPU
- communication
- DMA transfers

Local Store
- "main memory" of SPU
- 256 kB only
- 16 B/s bandwidth
- 128 B/s if all 8 banks accessed concurrently

Power Processor Element
- simplified PowerPC
- similar to PPC970
- 2x SMT, but in order

@3.2GHz

Synergistic Processor Unit
- SIMD vector processor
- no direct connection to the outside

BEI

Broadband Engine Interface
- coherent connection to 2nd CPU
- IO devices
Multigrid on Cell Processor

- Work done by Daniel Ritter, H. Köstler, M. Stürmer
  - in collaboration with M. Bolten

- A Fast Multigrid Solver for Molecular Dynamics on the Cell Broadband Engine
  - will be reported in Copper Mountain 2009
Performance Results for a 3D Jacobi Smoother

Figure 6: Floating Point and Memory Performance of Jacobi Smoother with 128 byte Alignment
Part IV

Hierarchical Hybrid Grids
Parallel High Performance FE Multigrid

- Parallelize „plain vanilla“ multigrid
  - tune single core performance first
  - partition domain
  - parallelize all operations on all grids
  - use clever data structures

- Do not worry (so much) about Coarse Grids
  - idle processors?
  - short messages?
  - sequential dependency in grid hierarchy?

- Why we do not use conventional domain decomposition
  - DD without coarse grid does not scale (algorithmically) and is suboptimal for large problems/many processors
  - DD with coarse grids may be as efficient as multigrid but is as difficult to parallelize (the difficulty is in parallelizing the coarse grid)
Hierarchical Hybrid Grids (HHG)

Joint work with

Frank Hülsemann (now EDF), Ben Bergen (now Los Alamos), T. Gradl (still Erlangen)

HHG Goal: Ultimate Parallel FE Performance!

- unstructured adaptive refinement grids with
  - regular substructures for
  - efficiency
  - superconvergence effects
HHG for Parallelization

- Use regular HHG patches for partitioning the domain
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
Adaptivity in HHG (with conforming meshes)
HLRB-II: SGI Altix 4700 at the Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften
No. 10 in in TOP-500 of June 2007

- Shared memory architecture
  - 9728 CPU Cores (Itanium2 Montecito Dual Core)
  - 39 TBytes RAM
  - NUMAlink network
  - 62.3 TFLOP/s Peak Performance

- Our testing ground for scalability experiments
<table>
<thead>
<tr>
<th>#Proc</th>
<th>#unkn. x 10^6</th>
<th>Ph.1: sec</th>
<th>Ph. 2: sec</th>
<th>Time to sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>134.2</td>
<td>3.16</td>
<td>6.38*</td>
<td>37.9</td>
</tr>
<tr>
<td>8</td>
<td>268.4</td>
<td>3.27</td>
<td>6.67*</td>
<td>39.3</td>
</tr>
<tr>
<td>16</td>
<td>536.9</td>
<td>3.35</td>
<td>6.75*</td>
<td>40.3</td>
</tr>
<tr>
<td>32</td>
<td>1,073.7</td>
<td>3.38</td>
<td>6.80*</td>
<td>40.6</td>
</tr>
<tr>
<td>64</td>
<td>2,147.5</td>
<td>3.53</td>
<td>4.92</td>
<td>42.3</td>
</tr>
<tr>
<td>128</td>
<td>4,295.0</td>
<td>3.60</td>
<td>7.06*</td>
<td>43.2</td>
</tr>
<tr>
<td>252</td>
<td>8,455.7</td>
<td>3.87</td>
<td>7.39*</td>
<td>46.4</td>
</tr>
<tr>
<td>504</td>
<td>16,911.4</td>
<td>3.96</td>
<td>5.44</td>
<td>47.6</td>
</tr>
<tr>
<td>2040</td>
<td>68,451.0</td>
<td>4.92</td>
<td>5.60</td>
<td>59.0</td>
</tr>
<tr>
<td>3825</td>
<td>128,345.7</td>
<td>6.90</td>
<td></td>
<td>82.8</td>
</tr>
<tr>
<td>4080</td>
<td>136,902.0</td>
<td></td>
<td>5.68</td>
<td></td>
</tr>
<tr>
<td>6102</td>
<td>205,353.1</td>
<td></td>
<td>6.33</td>
<td></td>
</tr>
<tr>
<td>8152</td>
<td>273,535.7</td>
<td></td>
<td>7.43*</td>
<td></td>
</tr>
<tr>
<td>9170</td>
<td>307,694.1</td>
<td>7.75*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Parallel scalability of scalar elliptic problem discretized by tetrahedral finite elements.

Times for 12 V(2,2) cycles on SGI Altix: Itanium-2 1.6 GHz.

Largest problem solved to date: $3.07 \times 10^{11}$ DOFS on 9170 Procs: 7.8 s per V(2,2) cycle

Part V
Towards User Friendly Scalable FE Software

Performance Results
ParExPDE
(Parallel Expression Templates for PDE)

- work done by C. Freundl
- A library for the user friendly, rapid development of numerical PDE solvers on parallel (super-)computers
- Provides a high level and intuitive user interface without compromising on efficiency
- Regularly refined hexahedral grids
- Support for multigrid hierarchies
C++ Expression Templates

- Encapsulation of arithmetic expressions
  - tree-like structure
  - C++ template constructs
- Evaluation of expression at compile time
- Avoid unnecessary copying and temporary objects

Elegance & Performance
C++ Expression Templates

Evaluation of an expression:

```cpp
template <class T>
void Vector::operator=(Expr<T>& expr) {
    for (int i = 0; i < _size; i++)
        _values[i] = expr.valueAt(i);
}
```
C++ Expression Templates

\[
z = a \times x + b
\]

C++ Compiler
(Template instantiation, Inlining)

```cpp
for (int i = 0; i < z._size; i++) {
    z._values[i] = 
    a * x._values[i] + b._values[i];
}
```

Subsequent compiler optimisations can be applied
Program code of a V-cycle:

```c
for (int l = 0; l < nlevels - 1; l++) {
    for (int s = 0; s < npre; s++) {
        u = u + (f - laplace(u)) / Diag(laplace) | interior_points;
    }
    r = f - laplace(u) | interior_points;
    r.doRestrict();
    f.levelDown();
    f = r;
    u.levelDown();
    u = 0.0;
}
```
ParExPDE: Serial Performance

- AMD Opteron 848:
  - Rpeak = 4.4 GFLOPS
  - Memory bandwidth: 5.3 GB/s
  - Machine balance: 0.1506

- Jacobi smoother (constant coefficients):
  - 28 loads, 1 store
  - 56 floating point operations
  - Loop balance: 0.5179

⇒ maximum achievable performance:

\[
\frac{0.1506}{0.5179} \times 4.4 \text{ GFLOPS} = 1280 \text{ MFLOPS}
\]
ParExPDE: Serial Performance

- Implementation of Jacobi smoother with ParExPDE
- Intel C++ compiler 9.1
- Carefully chosen optimisation flags

→ Code performs with up to 990 MFLOPS

- Excellent performance for pure C++ code
ParExPDE: Parallel Performance

- Strong scaleup of Jacobi smoother on LSS cluster (210 hexahedrons of size $100^3$)
ParExPDE: Parallel Performance

- Weak scaleup of MG V(2,2) solver on HLRB 2
  \( \approx 1.7 \cdot 10^7 \) unknowns per processor
Part VI

Conclusions
An HPC Tutorial!

Getting Supercomputer Performance is 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: disguise HitMe within “clever data structures” that introduce a lot of overhead

- 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 (such as multigrid)

  the more primitive the algorithm
  the easier to maximize your MachoFlops!