Efficient Large Scale Parallel Multigrid

Multigrid Applications with a Parallel Expression Template Based Library

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

EMG
October 2008
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
Why Multigrid has Failed to Deliver its 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
How much is a PetaFlops?

- $10^6 = 1 \text{ MegaFlops: Intel 486}$
  33MHz PC (~1989)
- $10^9 = 1 \text{ GigaFlops: Intel Pentium III}$
  1GHz (~2000)
  - If every person on earth does one operation every 6 seconds, all humans together have 1 GigaFlops performance (less than a current laptop)
- $10^{12} = 1 \text{ TeraFlops: HLRB-I}$
  1344 Proc., ~ 2000
- $10^{15} = 1 \text{ PetaFlops}$
  - 122400 Cores (Roadrunner, 2008)
  - If every person on earth runs a 486 PC, we all together have an aggregate Performance of 6 PetaFlops.
- ExaScale (~$10^{18}$ Flops) around 2020?

HLRB-I: 2 TFlops
HLRB-II: 63 TFlops
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\(^1\)

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

Memory Interface Controller
- Rambus eXtreme Data Rate
- 2 channels
- 25.6 GB/s\(^1\) bandwidth

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

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

Broadband Engine Interface
- coherent connection to 2nd CPU
- IO devices

\(^1\) @3.2GHz
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
13th Copper Mountain Conference on Multigrid Methods

March 22-28, 2009
Copper Mountain, Colorado, USA

• THEMES:
  – General scalable multigrid - multilevel, multiscale and heirarchial techniques,
  – algebraic & structured - Parallel and cache-aware implementation of multigrid
  – Coarsening and upscaling - MG as preconditioning
  – Applications - Systems of PDEs

• Student Paper Competition: travel & lodging assistance available to students & new PhDs

• Multigrid Tutorial: basic & advanced MG, AMG

• Organizers & Sponsors: Front Range Scientific, Inc., CASC at LLNL, University of Colorado, SIAM, DOE, NSF, IBM, LANL, LLNL, BOEING

• Deadlines:
  • Student Papers January 5, 2009
  • Author Abstracts January 15, 2009
  • Early Registration February 22, 2009
  • Guaranteed Lodging February 18, 2009

• INFORMATION:
  http://grandmaster.colorado.edu/~copper/2009
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
### Parallel scalability of scalar elliptic problem discretized by tetrahedral finite elements.

<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></td>
<td>7.75*</td>
<td></td>
</tr>
</tbody>
</table>

Part V
Towards User Friendly Scalable FE Software
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);
}
```

```cpp
z = a * x + y
ExprLiteral(a)
ExprVector(x)
ExprBinOp<OpMult>
ExprBinOp<OpAdd>
Expr<ExprBinOp<OpAdd>>
```
C++ Expression Templates

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

C++ Compiler
(Template instantiation, Inlining)

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

Subsequent compiler optimisations can be applied
Multigrid in ParExPDE

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:
  - $R_{peak} = 4.4$ 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

$\Rightarrow$ 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!
Talk is Over

Questions?