Miniapplications: Vehicles for Co-design

Michael A. Heroux
Scalable Algorithms Department

Collaborators: Brian Barrett, Richard Barrett, Daniel Barnette, Erik Boman, Ron Brightwell, Paul Crozier, Doug Doerfler, Carter Edwards, Si Hammond, Paul Lin, Kevin Pedretti, Dylan Stark, Heidi Thornquist, Courtenay Vaughan, Alan Williams, Jim Willenbring, Michael Wolf
A Listing of Application Proxies

• Proxy App: Generic term for all types.
• Skeleton App:
  – Communication accurate, computation fake.
• Compact App:
  – A small version of a real app.
  – Attempting some tie to physics.
• Scalable Synthetic Compact Applications (SSCA):
  – DARPA HPCS.
  – Formal specification.
  – Code and detailed spec to allow re-write.
App Proxies (cont).

- HPC Challenge Benchmarks.
- NAS Parallel Benchmarks.
- SPEC.
- HPL: Really?
  - Yes: In the ’80s
  - Approximated:
    - Frontal solver, NASTRAN, ANSYS, more.
    - Multifrontal/Supernodal solver: First Gordon Bell.
  - Question: Why are DCA++, LSMS fastest apps?
  - Answer (?): HPL was first co-design vehicle… that never died!
• UHPC Challenge Problems:
  – Formal specification.
  – Math, kernel extraction.
  – Intended to be open source?

• Motifs, aka dwarves.
  – Really are patterns, not actionable.
  “Even as cartoon characters they are sketchy.”
    (John Lewis)

Question: Is there room for another approach?
Miniapps: Specs

• Size: O(1K) lines.
• Focus: Proxy for key app performance issue.
• Availability: Open Source.
• Scope of allowed change: Any and all.
• Intent: Co-design: From HW registers to app itself.
• Reference version developer & owner: Application team.
• Lifespan: Until it’s no longer useful.
Mantevo* Project

* Greek: augur, guess, predict, presage

- Multi-faceted application performance project.
- Started 5 years ago.
- Two types of packages:
  - Miniapps: Small, self-contained programs.
    - MiniFE/HPCCG: unstructured implicit FEM/FVM.
    - phdMesh: explicit FEM, contact detection.
    - MiniMD: MD Force computations.
    - MiniXyce: Circuit RC ladder.
    - MiniGhost: Data exchange pattern of CTH.
    - And many more coming online...
  - Minidrivers: Wrappers around Trilinos packages.
    - Beam: Intrepid+FEI+Trilinos solvers.
    - Epetra Benchmark Tests: Core Epetra kernels.
- Open Source (LGPL, not moving to New BSD)
- Staffing: Application & Library developers.
- Planning first official release of miniapps suite this year.
Goal: Develop scalable computing capabilities via:
- Application analysis.
- Application improvement.
- Computer system design.

Fixed timeline.

Countless design decisions.

Collaborative effort.

Pre-Mantevo:
- Work with each, large application.
- Application developers have conflicting demands:
  - Features,
  - performance.
- Application performance profiles have similarities.
Mantevo Effort

• Develop:
  – Mini apps, mini drivers.

• Goals:
  – Aid in system design decisions:
    • Proxies for real apps.
    • Easy to use, modify or completely rewrite, e.g., multicore studies.
  – Guide application and library developers:
    • Get first results in new situations: apps/libs know what to expect.
    • Better algorithms: Exploration of new approaches.
  – Predict performance of real applications in new situations.
  – New collaborations.

Results:
• Better-informed design decision.
• Broad dissemination of optimization techniques.
• Incorporation of external R&D results.
Didn’t give up on previous approach

Just added tools upstream
Examples
First Mantevo miniapp: HPCCG

• Glorified unstructured, distributed CG solve.
• SLOCCOUNT: 4091 SLOC (C++).
• Scalable (in z-dimension) to any processor count.
• Many targets:
  – Internode: MPI or not.
  – Intranode: Serial, OpenMP,
  – Scalar: float, double, complex
  – Int: 8, 16, 32, 64.
• Studied in numerous settings.
How could HPCCG really be a proxy?

• Simple logic experiment:
  – Many implicit apps spend 90+% of time in solver.
  – Solver is multi-level preconditioned Krylov method.
    • CG is (simple) Krylov method.
    • Preconditioner time dominated by smoother (GS, ILU)
      • GS, ILU similar to SpMV (except on multicore).
  – HPCCG is SpMV+CG.

• Can’t be accept results blindly.
  – App ownership of miniapp important here.
Validation

Are Miniapps Predictive?
Charon Complexity

• SLOCCOUNT (tool from David A. Wheeler).
  – Charon physics: 191,877 SLOC.
  – Charon + nevada framework 414,885 SLOC
  – Charon_TPL 4,022,296 SLOC

• Library dependencies:
  – 25 Trilinos package.
  – 15 other TPLs.

• Requires “heroic effort” to build.
• MPI-only, no intranode parallelism.
• Export controlled.
• Stats courtesy of Roger Pawlowski.
MiniFE Complexity

• SLOCCOUNT:
  – Main code: 6,469 SLOC
  – Optional libraries (from Trilinos): 37,040 SLOC

• Easy to build:
  – Multiple targets:
    • Internode: MPI or not.
    • Intranode: Serial, Pthreads, OpenMP, TBB, CUDA.
  – Dialable properties:
    • Compute load imbalance.
    • Communication imbalance.
    • Data types: float, double, mixed.

• Open source.
• Stats: Courtesy of me.
Does MiniFE Predict Charon Behavior?

MPI Traffic: 16, 64, 256 MPI Processes; 31k DOF/core

- 2D Charon steady-state drift-diffusion BJT
- Muzia Cray XE6 (dual-socket 2.4GHz 8-core Magny-Cours); use all cores
- Gcc 4.5.2 compiler (-O2); MPI profiling via mpiP
- 2D Charon (3 DOF/node) vs. 3D MiniFE; match DOF/core and NNZ/(matrix row)
- Charon: ML preconditioned GMRES linear solver

<table>
<thead>
<tr>
<th>cores</th>
<th># messages</th>
<th>Total bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>24.3</td>
<td>5.8</td>
</tr>
<tr>
<td>64</td>
<td>28.9</td>
<td>6.8</td>
</tr>
<tr>
<td>256</td>
<td>25.4</td>
<td>5.9</td>
</tr>
</tbody>
</table>
Does MiniFE Predict 3D Charon Behavior? Cache and %Peak Performance, Single Node XE6; 31k DOF/core

- 3D Charon steady-state BJT
- Single compute node of muzia (Cray XE6; dual-socket 2.4GHz 8-core Magny-Cours)
  - Single MPI task
  - 16 MPI processes (use all 16 cores)
- Krylov solver: compare MiniFE CG with 3D Charon; start with Charon case closest to MiniFE and progress to regular Charon case
  - 1 DOF/node with BiCGSTAB; ILU(0)ov=0 preconditioner (“ILU/bicg”)
  - 3 DOF/node with BiCGSTAB; ILU(0)ov=0 preconditioner (“ILU/bicg”)
  - 3 DOF/node with GMRES; ILU(0)ov=0 preconditioner (“ILU/GMR”)
  - 3 DOF/node with GMRES; ML multigrid preconditioner with ILU(0)ov=0 smoother (“ML/GMR”)
- 100 iterations for minife vs. 1 Newton step for Charon (100 iterations/Newton step)
- Gcc 4.5.2 compiler (-O2); PAPI 4.1.3
- Try to compare MiniFE “assembling FE”+”imposing BC” time with Charon equivalent
- Note to Mike: can be quite a bit of scatter for L3 data between multiple runs
Krylov Solve: Single MPI Process Cache and %Peak Performance

- 31k DOF

<table>
<thead>
<tr>
<th></th>
<th>MiniFE</th>
<th>3D Charon (Aztec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DOF/node</td>
<td>1 DOF/node</td>
<td>3 DOF/node</td>
</tr>
<tr>
<td>CG</td>
<td>ILU/bicg</td>
<td>ILU/bicg</td>
</tr>
<tr>
<td>L1 HR</td>
<td>99.7</td>
<td>99.1</td>
</tr>
<tr>
<td>L2 HR</td>
<td>95</td>
<td>83</td>
</tr>
<tr>
<td>L3 HR</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>%peakDP</td>
<td>6.9</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Abbreviations: HR (hit rate), %peakDP (% peak double precision)
Prec Setup Plus Krylov Solve: 1-MPI Process Cache and % Peak Performance

- 31k DOF

<table>
<thead>
<tr>
<th></th>
<th>MiniFE</th>
<th>3D Charon (Preconditioner Setup + Aztec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 DOF/node</td>
<td>1 DOF/node</td>
</tr>
<tr>
<td>CG</td>
<td>99.7</td>
<td>99.3</td>
</tr>
<tr>
<td>ILU/bicg</td>
<td>95</td>
<td>83</td>
</tr>
<tr>
<td>ILU/GMR</td>
<td>17</td>
<td>25</td>
</tr>
<tr>
<td>ML/GMR</td>
<td>%peakDP</td>
<td>6.9</td>
</tr>
</tbody>
</table>

Abbreviations: HR (hit rate)
MPI Message Count (Cray XE6)  
Weak Scaling; 31k DOF/core

- Gcc 4.5.2 compiler (-O2); MPI profiling via mpiP  
- Preconditioner setup insignificant for ILU, but not ML (PS=prec setup)  
- ML: 4 level W-cyc (need to keep number of levels constant)

Ratio of (Charon Aztec)/(MiniFE CG) for # messages:

<table>
<thead>
<tr>
<th>cores</th>
<th>3D Charon (Aztec)</th>
<th>PS+Aztec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 DOF/node</td>
<td>3 DOF/node</td>
</tr>
<tr>
<td>ILU/bicg</td>
<td>ILU/bicg</td>
<td>ILU/GMR</td>
</tr>
<tr>
<td>16</td>
<td>2.3</td>
<td>2.3</td>
</tr>
<tr>
<td>64</td>
<td>2.3</td>
<td>2.2</td>
</tr>
<tr>
<td>256</td>
<td>2.4</td>
<td>2.4</td>
</tr>
</tbody>
</table>
• Gcc 4.5.2 compiler (-O2); MPI profiling via mpiP
• Preconditioner setup insignificant for ILU, but not ML (PS=prec setup)
• ML: 4 level W-cyc (need to keep number of levels constant)

Ratio of (Charon Aztec)/(MiniFE CG) for total bytes:

<table>
<thead>
<tr>
<th>cores</th>
<th>3D Charon (Aztec)</th>
<th>PS+Aztec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 DOF/node</td>
<td>3 DOF/node</td>
</tr>
<tr>
<td></td>
<td>ILU/bicg</td>
<td>ILU/bicg</td>
</tr>
<tr>
<td>16</td>
<td>1.9</td>
<td>3.1</td>
</tr>
<tr>
<td>64</td>
<td>2.3</td>
<td>3.5</td>
</tr>
<tr>
<td>256</td>
<td>2.4</td>
<td>3.5</td>
</tr>
</tbody>
</table>
MPI Message Count (Cray XE6)  
Strong Scaling; Total DOF ~2 million

- Gcc 4.5.2 compiler (-O2); MPI profiling via mpiP
- Total DOF: miniFE 1.953 million; Charon 1 DOF/node 1.980 million; Charon 3 DOF/node 1.942 million (~122,000 DOF/core for 16-core, ~31,000 DOF/core for 64-core, ~7500 DOF/core for 256-core)
- Preconditioner setup insignificant for ILU, but not ML (PS=prec setup)
- ML: 4 level W-cyc (need to keep number of levels constant)

Ratio of (Charon Aztec)/(MiniFE CG) for # messages:

<table>
<thead>
<tr>
<th>cores</th>
<th>3D Charon (Aztec)</th>
<th>PS+Aztec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 DOF/node</td>
<td>3 DOF/node</td>
</tr>
<tr>
<td></td>
<td>ILU/bicg</td>
<td>ILU/bicg</td>
</tr>
<tr>
<td>16</td>
<td>2.4</td>
<td>2.3</td>
</tr>
<tr>
<td>64</td>
<td>2.3</td>
<td>2.2</td>
</tr>
<tr>
<td>256</td>
<td>2.4</td>
<td>2.4</td>
</tr>
</tbody>
</table>
**MPI Message Bytes (Cray XE6)**  
**Strong Scaling; Total DOF ~2 million**

- Gcc 4.5.2 compiler (-O2); MPI profiling via mpiP
- Total DOF: miniFE 1.953 million; Charon 1 DOF/node 1.980 million; Charon 3 DOF/node 1.942 million (~122,000 DOF/core for 16-core, ~31,000 DOF/core for 64-core, ~7500 DOF/core for 256-core)
- Preconditioner setup insignificant for ILU, but not ML (PS=prec setup)
- ML: 4 level W-cyc (need to keep number of levels constant)

Ratio of (Charon Aztec)/(MiniFE CG) for MPI message bytes:

<table>
<thead>
<tr>
<th>cores</th>
<th>3D Charon (Aztec)</th>
<th>PS+Aztec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 DOF/node</td>
<td>3 DOF/node</td>
</tr>
<tr>
<td></td>
<td>ILU/bicg</td>
<td>ILU/bicg</td>
</tr>
<tr>
<td>16</td>
<td>2.0</td>
<td>2.8</td>
</tr>
<tr>
<td>64</td>
<td>2.4</td>
<td>3.5</td>
</tr>
<tr>
<td>256</td>
<td>2.2</td>
<td>3.3</td>
</tr>
</tbody>
</table>
Some MiniFE/Charon Observations

• MiniFE behavior correlates well to Charon’s for:
  – Cache behavior.
  – Compiler, processor ranking (not shown).
  – Message volume.
  – Scaling for additive Schwarz preconditioners.

• Does not correlate:
  – Multilevel preconditioners:
    • Working on aggregate approach with MiniFE.
  – Message count:
    • Uncovered an anomaly in MPI message counting.
• Careful calibration required: Apples to apples.
• First results are good:
  – No misleading trends.
• Known modeling weakness:
  – Multi-level preconditioner scaling trends.
  – Next approach: Use MiniFE at multiple sizes for aggregate time.
• MiniFE & Nvidia:
  – Register usage requirements:
    • Show register storage size imbalance.
  – Strategies for data structures.
Managing Miniapp Data
Data Management

Common Look-and-Feel: YAML

- Input parameters:
  - Command line.
  - YAML file.
- Output:
  - YAML.
  - Embeds input parameters.
  - Output file can be input.
- Data parsing and collection:
  - Email list submission of YAML file.
  - CoPylot: Digests email, populates database.
- Common YAML data functions across all miniapps.

YAML ain’t a Markup Language
- de facto standard format
- Human readable
- Convertible to/from XML, others

```php
$currentElement->get("performance_summary")->add("total","\"");
$currentElement->get("performance_summary")->get("total")->add("time",times[0]);
$currentElement->get("performance_summary")->get("total")->add("flops",3.0*fnops);
$currentElement->get("performance_summary")->get("total")->add("mflops",3.0*fnops/times[0]/1.0E6);
```
YAML Output File Excerpts

beefy.109% ./miniFE.x nx=30 ny=30 nz=30
creating/filling mesh...0.00031209s, total time: 0.00031209

generating matrix structure...0.0196991s, total time: 0.0200112

assembling FE data...

get-nodes: 0.0035727
compute-elems: 0.090822
sum-in: 0.0277233

0.125864s, total time: 0.145875

imposing Dirichlet BC...0.0176551s, total time: 0.16353

making matrix indices local...8.10623e-06s, total time: 0.163538

Starting CG solver ...

Initial Residual = 182.699

Iteration = 5   Residual = 43.6016
Iteration = 10  Residual = 6.13924
Iteration = 15  Residual = 0.949901
Iteration = 20  Residual = 0.131992
Iteration = 25  Residual = 0.0196088

Platform:
hostname: beefy.cs.csbsju.edu
kernel name: 'Linux'
kernel release: '2.6.34.7-66.fc13.x86_64'
processor: 'x86_64'

Build:
CXX: '/usr/lib64/openmpi/bin/mpicxx'
compiler version: 'g++ (GCC) 4.4.5 20101112 (Red Hat 4.4.5-2)'
CXXFLAGS: '-O3'

using MPI: yes
Threading: none

Run Date/Time: 2011-03-14, 22-30-26

Rows-per-proc Load Imbalance:
Largest (from avg, %): 0
Std Dev (%): 0

Total:
Total CG Time: 0.065695
Total CG Flops: 9.45762e+07
Total CG Mflops: 1439.63
Time per iteration: 0.0013139
Total Program Time: 0.237604
Emerging value: Broad Distribution
The Sentinel Dynamic
HPCCG

pHPCCG

Beam

phdMesh

Epetra Kernels

Prolego

HPCCG Results:
(nx=75, ny=50, nz=50)
Total MFLOPS = 348.46
Press Start to run again.
Plans for 2012

• Flesh out minidrivers
• Release the Suite of miniapps
• Set up mantevo.org
• Encourage community interaction.
Summary

• Miniapps:
  – In many ways similar to other efforts.
  – Two important distinctions:
    • App team develops and owns.
    • Miniapp retired when no longer useful.
  – Some strengths:
    • Completely open process: LGPL, validation.
    • Highly collaborative.

• Challenges:
  – Engaging already-busy apps developers.
  – Keeping miniapps relevant over time (to avoid premature retirement).

• Mantevo site: http://software.sandia.gov/mantevo
• Soon: mantevo.org (website up, not populated)