Miniapplications: Vehicles for Co-design

SOS 15, Engelberg, Switzerland
Michael A. Heroux
Scalable Algorithms Department

Collaborators: Brian Barrett, Richard Barrett, Erik Boman, Ron Brightwell, Paul Crozier, Doug Doerfler, Carter Edwards, Kevin Pedretti, Heidi Thornquist, Alan Williams, Michael Wolf
A Listing of Application Proxies

• 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.
- Developer & owner: Application team.
- Lifespan: Until it’s no longer useful.
Mantevo* Project

* Greek: augur, guess, predict, presage

- Multi-faceted application performance project.
- Started 4 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.
    - CTH-Comm: Data exchange pattern of CTH.
  - Minidrivers: Wrappers around Trilinos packages.
    - Beam: Intrepid+FEI+Trilinos solvers.
    - Epetra Benchmark Tests: Core Epetra kernels.
    - Dana Knoll working on new one.
- Open Source (LGPL)
- Staffing: Application & Library developers.
Background

- 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.
Data Placement on NUMA

- Memory Intensive computations: Page placement has huge impact.
- Most systems: First touch.
- Application data objects:
  - Phase 1: Construction phase, e.g., finite element assembly.
  - Phase 2: Use phase, e.g., linear solve.
- Problem: First touch difficult to control in phase 1.
- Idea: Page migration.
Data placement experiments

• MiniApp: HPCCG
• Construct sparse linear system, solve with CG.
• Two modes:
  – Data placed by assembly, not migrated for NUMA
  – Data migrated using parallel access pattern of CG.
• 1 hour of effort to modify code.
• Results on dual socket quad-core Nehalem system.
• Migrate-on-next-touch:
  – RT/OS feature.
Weak Scaling Problem

- MPI and conditioned data approach comparable.
- Non-conditioned very poor scaling.
• Rewrites of HPCCG:
  – Pthreads, OpenMP, Chapel, qthreads…

• MiniFE:
  – Prototype of Kokkos Node API.
  – Prototype of pipeline and task graph node parallelism.

• Skeleton app of miniapp!

• Performance comparisons of different platforms:
  – All.
TBB Pipeline for FE assembly

Serial Filter
Launch elem-data from mesh

Parallel Filter
Compute stiffnesses & loads

Several Serial Filters in series
Assemble rows of stiffness into global matrix

Element-stiffness matrices computed in parallel

Global Matrix

Assemble Rows
0,1,2

Assemble Rows
3,4,5

Assemble Rows
6,7,8

Each assembly filter assembles certain rows from a stiffness, then passes it on to the next assembly filter

Work done in MiniFE: Courtesy of Alan Williams
Preconditioners for Scalable Multicore Systems

- Observe: Iteration count increases with number of subdomains.
- With scalable threaded smoothers (LU, ILU, Gauss-Seidel):
  - Solve with fewer, larger subdomains.
  - Better kernel scaling (threads vs. MPI processes).
  - Better convergence, More robust.
- Exascale Potential: Tiled, pipelined implementation.
- Three efforts:
  - Level-scheduled triangular sweeps (ILU solve, Gauss-Seidel).
  - Decomposition by partitioning
  - Multithreaded direct factorization

Emerging Abstract Machine Model: Thread team

- Multiple threads.
- Fast barrier.
- Shared, fast access memory pool.
- Required to address the constraints of global SIMT.
- Example: Nvidia SM
- X86 more vague, emerging more clearly in future.
- Prototyped in variant of HPCCG.
Managing Miniapp Data
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

```bash
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/time[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: 0.163538
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

...
Emerging value: Broad Distribution
The Sentinel Dynamic
Validation

Are Miniapps Predictive?
Does MiniFE Predict Charon Behavior?
Processor Ranking: 8 MPI tasks; 31k DOF/core

- Charon steady-state drift-diffusion BJT
- Nehalem (Intel 11.0.081 –O2 –xsse4.2; all cores of dual-socket quadcore)
- 12-core Magny-Cours (Intel 11.0.081 –O2; one socket, 4 MPI tasks/die)
- Barcelona (Intel 11.1.064 –O2; use two sockets out of the quad-socket)
- 2D Charon (3 DOF/node) vs. 3D MiniFE; match DOF/core and NNZ in matrix row
- Charon LS w/o or w/ ps: GMRES linear solve without/with ML precond setup time
- Try to compare MiniFE “assembling FE”+”imposing BC” time with Charon equivalent

<table>
<thead>
<tr>
<th></th>
<th>CG</th>
<th>FE assem+BC</th>
<th></th>
<th>LS w/o ps</th>
<th>LS w/ ps</th>
<th>Mat+RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nehalem</td>
<td>Nehalem</td>
<td>1</td>
<td>Nehalem</td>
<td>Nehalem</td>
<td>Nehalem</td>
</tr>
<tr>
<td>2</td>
<td>MC(1.7)</td>
<td>MC(1.7)</td>
<td>2</td>
<td>MC(1.7)</td>
<td>MC(1.8)</td>
<td>MC(1.46)</td>
</tr>
<tr>
<td>3</td>
<td>Barc(2.7)</td>
<td>Barc(1.8)</td>
<td>3</td>
<td>Barc(2.8)</td>
<td>Barc(2.5)</td>
<td>Barc(1.52)</td>
</tr>
</tbody>
</table>

Number in parenthesis is factor greater than #1 time
MiniFE Predict Charon? Multicore Efficiency Dual-Socket 12-core Magny-Cours: 124k DOF/core

- Charon steady-state drift-diffusion BJT; Intel 11.0.081 –O2
- Weak scaling study with 124k DOF/core
- 2D Charon (3 DOF/node) vs. 3D MiniFE; match DOF/core and NNZ in matrix row
- Efficiency: ratio of 4-core time to n-core time (expressed as percentage)
- Charon LS w/o or w/ ps: GMRES linear solve without/with ML precond setup time
- 100 Krylov iterations for both MiniFE and Charon (100 per Newton step)

<table>
<thead>
<tr>
<th>MiniFE</th>
<th>Charon</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cores</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>24</td>
</tr>
</tbody>
</table>
Miniapps Predictive?

• First results are good:
  – No misleading trends.
• Careful calibration required: Apples to apples.
• Big plus: Ease of porting.
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.
HPCCG

pHPCCG

Beam

phdMesh

Epetra Kernels

Proleko

Nx = 75
Ny = 50
Nz = 50

Start

HPCCG Results:
(nx=75, ny=50, nz=50)
Total MFLOPS = 348.46
Press Start to run again.
Next Target App: CTH

• CTH:
  – Multi-material, large deformation, shock physics.
  – Used through DOE complex, heavily used by DOD.

• Each time step:
  – 2D face exchanges (19 times in each of 3 dims).
  – 1 face exchange: 40 arrays.
  – 100x100x100 local problem: 3.2 MB per face.

• Future systems (e.g. Cray Cielo):
  – Higher network injection rates.

• Goal: Study different comm algorithms to exploit rates.
Miniapp: 2D face exchange with simple 27-pt computation.

• Explore spectrum of comm algorithms:
  – Standard approach as baseline.
  – Transmit each variable as soon as available.
  – Transmit as soon as any 2D slide is available.

• Introduce dialable load imbalance.

• Results?
  – See Richard Barrett’s paper, submission to SC’11.
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)