Proxy Applications: Vehicles for Co-design and Collaboration

PSAAP II Kick-off meeting

Albuquerque, Dec 10, 2013

Rob Neely
The role of proxy applications in exascale co-design

- **Co-design**: A collaboration between vendors, hardware architects, system software developers, domain scientists, computer scientists, and applied mathematicians **working together** to make informed evaluations about **hardware and software** components necessary for a successful transition to exascale.

- **Proxy applications**: The “language of co-design”. They are simplified representations of algorithms, data motion patterns, and coding styles used to do early **evaluation of trade-offs** in the hardware and software design space.
Co-design Provides a Formal Methodology for us to Work With the Community

What is Co-design?
- Deep collaboration… (See previous slide)

How Does Co-design Work?
- Hardware vendors provide access to future roadmaps
- Researchers develop new software methodologies
- Application developers provide “proxy applications” for open study of software requirements

How is this different from past practice?
- Time-frame (5-10 years)
- Co-dependence on successful HPC strategy

What are the challenges of co-design?
- Unclassified proxy applications are required
- Deep NDA and trust with multiple vendors
(One of) the difficulties of co-design

Co-design gets more difficult the further you get from open collaboration and the closer you get to the “truth” (particularly with competing vendors and national security applications)

• ASC : Involve staff with clearances in co-design efforts and developing proxy apps
• Vendor : Limit number of lab staff engaging in multiple “deep NDA” discussions
ASC NSApp Co-design Project

- ExMatEx, CESAR, and ExaCT are ASCR-funded co-design centers
  - 2 of 3 led at NNSA labs
  - Lots of opportunity for cross-fertilization between NNSA and Office of Science

- Co-design centers are designed to be application-centric
  - NNSA already has the applications, but they’re typically not open (export controlled / classified)

- NNSA/ASC co-design effort - NSApp CDP
  - National Security Applications (LANL / LLNL / SNL)
  - See paper at codesign.llnl.gov
ASC multi-physics applications are complex, expensive to maintain and evolve, and daunting to rewrite. Proxy apps to the rescue!

- Support broad ranges of applications
- > 10 packages, 10-30+ third party libraries
- Many different spatial, temporal scales
- Multi-language (C++, C, Fortran90, Python)
- Variety of parallelism approaches
- Long life-time projects with >1 million lines of code
- 15+ years of development by large teams (10 – 20+ FTEs)
- Steerable / interactive interfaces
- Algorithms tuned for minimal turn-around time (vs. max compute efficiency)
- How future physics model improvements will impact compute balance between packages is unknown

The difficulty is compounded by continuing to deliver the programmatic mission while addressing the challenges of next generation advanced architectures.
Full Re-implementation of Large Codes Is a Decadal Process

Research and conception of idea 1-5y

Proof of concept product 1-3y

Hardened product (suitable for prototyping) 2-4y

First Users / Feedback 1-3y

Re-implementation application 4-6y

Community buy-in & “standards” 2-4y

Full-featured 1-5y

Validated for Mission 1-3y

Production use until retirement

Research = 6 – 15 years to fruition

Rewrite = 7 – 16 years to replacement
Evolve or Rewrite? This is a fundamental question we’re addressing

**Evolve existing code bases**
- Gain experience with massive scaling (Sequoia / BlueGeneQ)
- Expose fine-grained concurrency
- Accelerator directives
- Application controlled resilience and power management
- Leverage validated code base

**Undertake new “from scratch” rewrite**
- Evaluate and gain experience with new programming models
- “Harden” them beyond research prototype phase
- Determine degree of rewrite needed (if any)

At a minimum, we know we’re going to have to do this. But will it take us far enough?

New languages and a clean slate approach are compelling, but can we manage the risk?
LLNL ASC Integrated Code Strategy – Four “pillars” capture our exploration space

1) Evolve current ICs to run effectively on Sequoia (0 – 4 year scope)
2) Develop modular packages (evolve or rewrite) for Sequoia and beyond (0 – 8 years)
3) Revolutionary approaches as basis for next generation code (3 – 15 years)
4) New programming models and parallelization strategies (5 – 20+ years)

Our Challenges are not tied to the pursuit of exascale computing. Future architectures at extreme scale (100’s of Pf) are just as demanding.
A suite of proxy applications should span a wide design space

- **Algorithms**
  - Mesh-based, particles, solvers, structural, CFD, etc...
  - Discretization methods

- **Breadth**
  - Targeted to a specific purpose
  - Generally representative and multi-purpose

- **Audience**
  - Hardware designers
  - Software designers

- **Modernness**
  - Extraction of existing application
  - Exploration of future application

- One must consider a broad range of proxy applications to even begin to cover the design space of an ASC application
- Proxy apps are more than just a benchmark. They are meant to be modified – perhaps dramatically
- Interoperability of software solutions is key
The $65,536 dollar question

- What is the right balance between too big to understand, and too simple to be representative?

Small [O(1k loc)]

**Pros** – Easy to pick up and learn

**Cons** – Hard to draw general conclusions from

Larger [O(25-75k loc)]

**Pros** – Likely more representative of real applications

**Cons** – Some benefits are lost to added complexity and size

Understand: Small proxy applications are often gross approximations of reality. Be careful of the conclusions you draw!
Sample LLNL success stories with proxy apps

- **LULESH**
  - Ported to 8+ programming languages – allowed an unprecedented comparison. (IPDPS Best Paper 2012)
  - Helped influence development of Chapel, Liszt, Charm++, …
  - Used in ExMatEx co-design center to great effect in FastForward interactions

- **LCALS**
  - Allowed rapid identification of optimization issues with multiple vendor compiler teams.

- **AMG / UMT / MCB / LULESH / SNAP**
  - Repurposed as TN8/CORAL benchmarks (NNSA 2015-17 large scale procurements)
  - Co-design has influenced changes in how we do procurement benchmarking

- **Lassen**
  - First (?) example of Charm++ used as a library component

Numerous examples of proxy apps giving students and new hires a jump start
AMG2013: Algebraic Multi-grid

Description
- Derived from BoomerAMG in LLNL’s hypre solver library
- Representative of implicit solves performed in large unstructured applications

Characteristics / Uses
- Stresses memory bandwidth
- Irregular communication patterns and memory accesses
- Fine-grained threading
- Acceleration

Language(s) / Size
- C (~70k loc)
- MPI
- OpenMP
MCB: Monte Carlo

Description

- Little or no real physics – just a “particle pusher”
- Object-oriented design

Characteristics / Uses

- Low floating point intensity, large amount of branching
- Irregular memory and communication patterns
- Could be used for exploration of alternatives to MPI (e.g. PGAS), transactional memory,

Language(s) / Size

- C++ (~13k loc)
- MPI
- OpenMP
**Lassen: Front tracking**

**Description**
- Front-tracking algorithm used to propagate wave-fronts and pre-calculate arrival times
- Work is isolated to a narrow region around the front

**Characteristics / Uses**
- Highly load-imbalanced (1D front in a 2D mesh)
- Similar issues to sweep-based algorithms
- Studying task-based parallelism or dynamic load balancing
- Interoperability of MPI and other prog. models (e.g. Charm++)

**Language(s) / Size**
- C++ (3,500 loc)
- Charm++
- MPI
LULESH: Lagrangian Hydrodynamics

Description
- Unstructured mesh
- Small in size – about 12 representative kernels
- Ported to numerous different programming models
- Version 2.0 released in 2013

Characteristics / Uses
- Unstructured mesh data structures (indirection)
- Analysis of thread overheads in OpenMP
- Port to various programming models

Language(s) / Size
- C++ (5k loc)
- MPI / OpenMP
- A++, Chapel, CUDA, OpenACC, Loci, Liszt, Charm++ versions available
LCALS: Loop Kernel Suite

**Description**
- Livermore Compiler Analysis Loop Suite
- Based on classic “Livermore Loops” from the 70’s-80’s
- ~30 representative loop kernels
- Easy framework to extract individual loops, add new ones

**Characteristics / Uses**
- Primarily designed to work with compiler vendors on optimizations (e.g. SIMD)
- Multiple versions: “Raw”, “Lambda/RAJA”, cilk plus, …
- Some loops large enough to study threading / runtimes

**Language(s) / Size**
- C++ (4k loc)
- OpenMP
Mulard: Multigroup Rad Diffusion

Description
- Includes some simpler mini-apps (Duckling, Hatchling) as well
- Built on top of MFEM library

Characteristics / Uses
- Coupled implicit solve methods
- Use of abstractions in design of finite-element based apps
- Ability to modify spatial ordering of the mesh
- Useful to study acceleration, transactional memory, ...

Language(s) / Size
- C++ (46k loc)
- OpenMP
LIP (not yet released)

**Description**
- 1D/2D tabular interpolation library
- Builds sets of tabular data for repeated lookups
- Basis for Equation-of-State calculations

**Characteristics / Uses**
- Designed to study memory hierarchies
- Sharing of data between MPI processes on same node
- Effective caching strategies for NUMA architectures
- Threading strategies

**Language(s) / Size**
- C (11k loc)
- MPI driver to spawn multiple copies (no communication)
LUAU3D (not yet released)

**Description**
- 3D Unstructure mesh advection
- Complex data motion – fluxing of material through element faces
- No physics (velocities), just mesh relaxation and material flux

**Characteristics / Uses**
- Useful counterpart to LULESH for understanding ALE (arbitrary lagrange eulerian)
- Irregular memory accesses, lots of p2p communication
- Element ordering, acceleration, …

**Language(s) / Size**
- C++ (26k loc)
- MPI
- OpenMP (in later versions?)
## LLNL ASC Proxy App Suite

<table>
<thead>
<tr>
<th>Proxy</th>
<th>Type</th>
<th>Lang</th>
<th>LOC</th>
<th>Description</th>
<th>Example uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lassen</td>
<td>Mini</td>
<td>C++, Charm++</td>
<td>3.5k</td>
<td>Front tracking through a 2D mesh. Work is concentrated at front, leading to high load imbalance.</td>
<td>Exploring task-based programming models and load balance strategies</td>
</tr>
<tr>
<td>LULESH</td>
<td>Mini</td>
<td>C++, MPI, OMP</td>
<td>5k</td>
<td>Lagrangian hydro on arbitrary connected hex mesh. 2.0 includes regions, and unified ser/par source.</td>
<td>New programming models and alternate languages. Overall performance characteristics of hydro</td>
</tr>
<tr>
<td>MCB</td>
<td>Mini</td>
<td>C++, MPI</td>
<td>13k</td>
<td>Monte Carlo Particle transport, Low floating point intensity. Lots of integer and branching</td>
<td>Threading, transactional memory, acceleration, irregular messaging, PGAS</td>
</tr>
<tr>
<td>LIP</td>
<td>Skel</td>
<td>C</td>
<td></td>
<td>Library for doing 2D interpolation on large tabular data</td>
<td>NUMA techniques. Sharing data between MPI tasks.</td>
</tr>
<tr>
<td>LUAU</td>
<td>Mini</td>
<td>C++, MPI</td>
<td></td>
<td>Multi-material advection on arbitrary connected hex mesh</td>
<td>Impact of memory indirection, bandwidth intensive.</td>
</tr>
</tbody>
</table>

http://codesign.llnl.gov
For more info: http://codesign.llnl.gov

- Get latest versions of LLNL proxy apps
- Contact developers
- References to other co-design efforts
How might this all relate to the PSAAP2 Centers?

- PSAAP centers have a unique opportunity to inject changes in NNSA code development processes
  - Exascale is driving disruptive changes.
  - Both sides (NNSA/Univ) must push exascale research concepts into complex applications

- Communicating CS concepts developed at your centers through simpler proxy apps is an effective method

- We invite you use our proxy apps
  - Developing a DSL or other abstraction? Test it on multiple proxy apps
  - Ask us questions – don’t assume anything!
  - Communicate back your changes to us (we learn from them as well)

- We expect to engage in co-design with you
  - We can be a conduit to the vendor community for you