HIGH PERFORMANCE COMPUTING OPERATIONAL REVIEW (HPCOR) WORKSHOP RECAP

Nick Romero

Catalyst Team Lead

Argonne Leadership Computing Facility

Acknowledgements

Steering Committee

Katherine Riley, ANL, Chair
Katie Antypas, LBNL
Scott Futral, LLNL
Richard Gerber, LBNL
Dave Goodwin, SC
Barbara Helland, SC
Thuc Hoang, NNSA
Paul Messina, ANL
Joel Stevenson, SNL
Tjerk Straatsma, ORNL
Tim Williams, ANL
Cornell Wright, LANL

Argonne Leadership Computing Facility (ALCF) is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

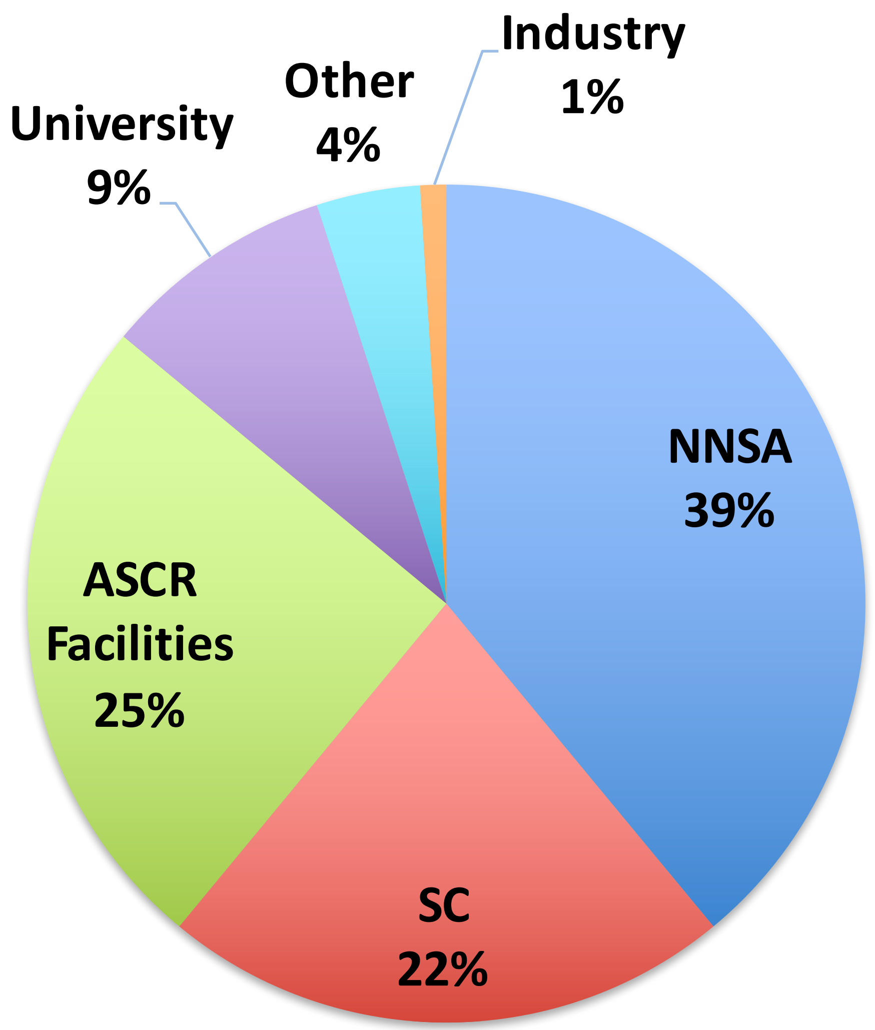
HPCOR 2015

- Application developers
- Compute facility staff
- Vendors
- Library/tool developers
- Program managers

- Best practices for scientific software architecture
- Especially those that increase portability and performance
- Consider impact over next 10 years

www.orau.gov/hpcor2015

About 70 Attendees



33 White Papers

- Language
- Lines of code
- Primary methods
- Types of problems/domains/science application problems
- Scale of resources commonly used for production runs
- Supercomputers regularly used
- Libraries/tools for prototyping
- Libraries/tools for production science campaigns
- Describe efforts to develop code (application, library, etc.) portable across diverse architectures.
- Where were the abstractions?
- How much code re-use was possible? If something was not possible, please describe why.
- What successes have you had with performant code across difference architectures? Were the same algorithms applicable at all across the architectures?
- What approaches did you reject and why? What was the leading contender rejected?
- What is your greatest fear going to exascale for application portability and functionality?

Three Breakouts

1. Application Architecture
   - ◎ Data structures/data movement?
   - ◎ Abstractions/parallelism?

2. Libraries and Tools
   - ◎ Most commonly used parts of tools and libraries?
   - ◎ Portability, performance, sustainability?

3. Software Engineering
   - ◎ Large changes to scientific codes?
   - ◎ Advance apps with lots of inertia?

- Best practices
- Failures
- Niche/emerging solutions
- Opportunities

Best Practices

- Two layers of abstraction:
  - inter-node
  - intra-node – swap for different architectures
- Virtually all applications use high-level libraries ➔ *libraries work. Use libraries.**
- Wide adoption of libs that perform well
  - FFTW, SuperLU, ScaLAPACK, …
- Portable, well-defined interfaces that work:
  - BLAS, MPI, HDF5, PAPI, …
- Lightweight tools are effective

- Porting codes: center of excellence approach has worked well
- Testing
  - Modernization/legacy apps: introduce abstractions/facades and **tests** using those interfaces
    - Later, change/specialize code beneath facades
    - Pay back *technical debt*
  - Use *science* as incentive/motivation: quality/reproducibility of results
- Training: 3-day mandatory software engineering training (*including managers)*

Niche/Emerging Solutions

- Future C++ standard language features for portability
- Avoid dependence on outside libraries*
- Frameworks like RAJA and Kokkos
- Domain specific software stacks
  - USQCD stack – solvers, I/O, operations, comm.
  - MOOSE – nuclear fuels/materials modeling
  - FenICS – differential equations solution by finite element methods
  - SPIRAL – DSP algorithms (autogenerate platform-tuned)
  - TCE – tensor contraction engine (NWChem)

- Lightweight, automated profiling tools that work at scale: HPM
- Vendor tools are used and useful
- Test driven development
- Multiplatform, multi-center automated build and test
- Provenance: data, versions, library versions, compiler versions
  - Workflows to manage

Failures

- Using vendor proprietary code & libraries
- OpenCL
- Portability via two code branches (Common! Good?)
- Number and variety of libraries is growing beyond our ability to support them
- Sustainability, long-term support lacking
- Libraries don’t necessarily interoperate (different programming/threading models)
- Inconsistent software environments across HPC centers

- Hero codes (single developer, long-lived code)
- Big bang code merges
- Translation from prototype-like codes (Matlab by scientist) to compiled C++ code
  - Scientist unable to work with it
- Unclear/unspecified support models for: community compilers, community libraries, ….

Opportunities

- Develop tools that will enable portability
- Train application engineers
- Career opportunities for staff with crossover expertise
- OpenMP 4.x – not demonstrated yet
- Provide access and support to tools and library developers at HPC centers
- Common base software environment across HPC Centers
- Performance portability: encourage investment, adoption, & guidance

- DOE investment in standards committees (OpenMP, C++, …)
- Facilities provide policies and mechanisms for automatic build and test: "auto-login"
- Develop best practices—knowledge base, training
  - OpenMP
  - Memory hierarchy

Tips & Quips

"Valley of death" between research & production

Motivate SWE practices with success stories—better than horror stories.

Users should be able to run the tests, not just developers.

When developing new algorithm, develop on at least two architectures.

Realize, accept the *full cost* of software development.

Common fear, looking ahead:

Loss of, lack of funding for, developers—to get software to production and keep it in production.