RADIUSS: Rapid Application Development via an Institutional Universal Software Stack
Build Tools
## Build Tools

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>License</th>
<th>Maturity (years)</th>
<th>Website</th>
<th>Repository</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spack</td>
<td>A flexible package manager for HPC</td>
<td>Apache-2 or MIT</td>
<td>~7</td>
<td>spack.io</td>
<td>github.com/spack/spack</td>
<td>Todd Gamblin</td>
</tr>
<tr>
<td>BLT</td>
<td>A streamlined CMake build system foundation for HPC software</td>
<td>BSD</td>
<td>~2</td>
<td>llnl-blt.readthedocs.io</td>
<td>github.com/LLNL/blt</td>
<td>Chris White</td>
</tr>
<tr>
<td>Shroud</td>
<td>Easily create Fortran, C and Python interfaces for C or C++ libraries</td>
<td>BSD</td>
<td>~3.5</td>
<td>shroud.readthedocs.io</td>
<td>github.com/LLNL/shroud</td>
<td>Lee Taylor</td>
</tr>
</tbody>
</table>
Spack
A flexible package manager for HPC

- Automates complex builds
  - Easily manage hundreds of dependencies, down to versions and build options
  - Easily test complex software with many compiler/MPI/BLAS combinations

- Easily share and leverage others’ work
  - Leverage a library of 4,000+ community-maintained package recipes
  - Leverage others’ internal/proprietary libraries with internal LLNL repositories
  - Allow other users and developers to easily use your software

- Broad use inside and outside the laboratories
  - ASC, LC, ENG, others at LLNL; codes at LANL, SNL, Fermi, ORNL, ANL, ECP
  - Nearly 3,000 worldwide users (per docs site), highly active community on GitHub

Spack users worldwide

ARES and its Dependencies

LLNL-developed
External Open Source

@spackpm
BLT
A streamlined CMake build system foundation for HPC software

- **Simple macros for complex tasks**
  - Create libraries, executables, and tests
  - Manages compiler flags across multiple compiler families
  - Unifies complexities of external dependencies into one easy to remember name

- **Batteries included**
  - Example configurations for most LC/Linux/OSX/Windows system and compiler families
  - Built-in support for:
    - HPC programming models
    - Code health
    - Documentation generation

- **Open source**
  - Leveraged by ALE3D, Ascent, Axom, CHAI, Conduit, FloBat, GeosX, Kripke, LEOS, MSLIB, RAJA, RAJA Perf Suite, Umpire, VBF Shaft, VTK-h
Shroud
Easily create Fortran, C, and Python interfaces for C or C++ libraries

- Generate wrappers with an annotated description of the C++ API
  - YAML input with C++ declarations for namespace, typedef, function, class, and struct
  - Annotations to provide semantic information: intent, dimension, ownership
  - Allows user control of generated names for functions and interfaces
  - Provides hooks to allow custom code to augment or replace generated wrapper

- Creates a Fortran idiomatic interface
  - Preserves object-oriented API
  - No need to be a Fortran expert to create Fortran wrapper
  - Uses C as lingua franca to access C++

- Use the same YAML file to create a Python module
  - Creates an extension module, no Python source code is created
  - Support for NumPy
Portable Execution and Memory Management
## Portable Execution and Memory Management

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>License</th>
<th>Maturity (years)</th>
<th>Website</th>
<th>Repository</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAJA</td>
<td>Loop-level abstractions to target machine-specific programming models and constructs</td>
<td>BSD</td>
<td>~5</td>
<td>software.llnl.gov/RAJA</td>
<td>github.com/LLNL/RAJA</td>
<td>Rich Hornung</td>
</tr>
<tr>
<td>CHAI</td>
<td>Optional add-on to RAJA for automating data motion between memory spaces</td>
<td>BSD</td>
<td>~4</td>
<td>software.llnl.gov/CHAI</td>
<td>github.com/LLNL/CHAI</td>
<td>David Beckingsale</td>
</tr>
<tr>
<td>Umpire</td>
<td>An application-focused API for memory management on NUMA &amp; GPU architectures</td>
<td>MIT</td>
<td>~3</td>
<td>software.llnl.gov/Umpire</td>
<td>github.com/LLNL/Umpire</td>
<td>David Beckingsale</td>
</tr>
<tr>
<td>LvArray</td>
<td>Array classes for high-performance simulation software</td>
<td>BSD</td>
<td>~2</td>
<td>lvarray.readthedocs.io</td>
<td>github.com/GEOSX/LvArray</td>
<td>Ben Corbett</td>
</tr>
</tbody>
</table>

**Technical Contact**

David Beckingsale
**RAJA**

Loop-level abstractions to target machine-specific programming models and constructs

- Provides a portable API for loop execution
- Powerful “kernel” API to express nested, multi-dimensional loops
- Other portable features
  - Reductions, scans, sorts, atomics, and multi-dimensional data views
- Supports multiple back-end targets: OpenMP, CUDA, AMD, …
- Easy to integrate into existing applications
  - Loop bodies remain generally unchanged
  - Can be adopted incrementally, one loop at a time
- Open source
  - Used by ASC and ATMD applications and libraries, and ECP projects: SAMRAI, MFEM, SUNDIALS, hypre, SW4, GEOS-X, ExaSGD, Alpine, etc.

```c
for (int i = 0; i < N; ++i) {
    a[i] += c * b[i];
}
```

A simple C-style loop

```c
forall<EXEC_POL>(RangeSegment(0, N), [=] (int i) {
    a[i] += c * b[i];
});
```

Same loop using RAJA

Loop execution defined by “execution policy”:
EXEC_POL can be seq_exec, openmp_exec, cuda_exec, etc.
Umpire
An application-focused API for memory management on NUMA and GPU architectures

- Simple and unified API to a wide range of memory resources:
  - DDR
  - NVIDIA GPU memory
    - Constant memory
  - AMD GPU memory
  - NUMA support
- Provides high-performance "strategies" for customizing data allocation:
  - Memory pools, buffers, CUDA memory advice
- "Operations" to copy, move, set data on any memory resource
- Open source
  - Underpins CHAI
  - Used by LLNL ASC and ATDM applications, SW4, SAMRAI, MFEM

```cpp
auto allocator = rm.getAllocator("DEVICE");
double* data = allocator.allocate(1024);
alculator.deallocate(data);
```
CHAI
Optional add-on to RAJA for automating data transfers between memory spaces

- Array-like object with automatic data migration
- Provides “unified memory” without any special system support
- Integrates with RAJA
  - Could be used with other programming models
- Uses Umpire, and behavior can be customized using different Umpire “Allocators”
- Open source
  - Used in LLNL ASC applications
  - Works with Umpire & RAJA

```cpp
chai::ManagedArray<double> data(100);

RAJA::forall<cuda_exec>(
    RangeSegment(0, 100), [=] (int i) {
        data[i] = i;
    }
);

RAJA::forall<seq_exec>(
    RangeSegment(0, 100), [=] (int i) {
        printf("data[%g] = %f\n", i, data[i]);
    }
);
```

CHAI arrays can be used on CPU or GPU, data migrates without user intervention
LvArray
Containers for use in high-performance simulation software

- Containers
  - A multi-dimensional array with a customizable memory layout and slicing.
  - A sorted unique list of values.
  - A jagged two-dimensional array.
  - A compressed row storage matrix and sparsity pattern.
- All containers support customizable allocation behavior and work on device.
- Integrates with RAJA and optionally CHAI.
- Open source
  - BSD license
  - Used by GEOSX ECP project

LvArray::Array<double, 2> x(10, 11);
forall<POLICY1>(x.size(0), [x=x.toView()](int i) {
    for(int j = 0; j < x.size(1); ++j )
        x(i, j) = foo(i, j);
});

LvArray::Array<double, 2> sums(x.size(0));
forall<POLICY2>(x.size(0),
    [x=x.toViewConst(), sums=sums.toView()](int i) {
    for(double value : x[i])
        sums[i] += value;
    });

sums.move(LvArray::MemorySpace::CPU);
std::cout << sums << std::endl;

When using CHAI POLICY1 and POLICY2 can be any RAJA policy and the data will migrate appropriately.
Application CS Infrastructure
# Application CS Infrastructure

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>License</th>
<th>Maturity (years)</th>
<th>Website</th>
<th>Repository</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axom</td>
<td>Flexible software infrastructure for the development of multi-physics applications and computational tools</td>
<td>BSD</td>
<td>~5</td>
<td>software.llnl.gov/axom</td>
<td>github.com/LLNL/axom</td>
<td>Rich Hornung</td>
</tr>
</tbody>
</table>

Please direct detailed technical questions to the Axom developer team:

axom-dev@llnl.gov
Application CS Infrastructure (Axom)

- Motivated by LLNL ASC next-generation code planning
  - Core infrastructure for the LLNL ATDM code
  - Used across the LLNL ASC code portfolio
- The report (at right) contains 50 recommendations spanning
  - Software architecture and design
  - Software processes and tools
  - Software sharing and integration
  - Performance and portability
  - Co-design, external interactions, research
- In development for 5+ years
- Open source
Application CS Infrastructure (Axom)

Mesh-aware data schema

Hierarchical key-value in-memory datastore

Parallel file I/O & burst buffer support

Surface queries & spatial acceleration data structures

Unified inter/intra-package message logging & parallel filtering

Mesh data model
Application CS Infrastructure (Axom)

- Examples of Axom application support
  - Centralized, hierarchical simulation data management
  - Parallel file I/O for checkpoint-restart and visualization
  - Access to in-situ visualization and analysis tools
  - Shaping in arbitrary, complex material geometries
  - Immersed boundaries, interfaces
  - Building blocks for particle-based algorithms
  - Integrated cross-package parallel message logging
### Application CS Infrastructure (Axom)

<table>
<thead>
<tr>
<th>Axom Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sidre</strong></td>
<td>In-core hierarchical key-value data management, plus parallel file I/O (restart, viz. files), support for heterogeneous memory systems, etc.</td>
</tr>
<tr>
<td><strong>Quest</strong></td>
<td>Spatial point/surface queries; in-out, signed distance, point containment, point-in-cell, etc.</td>
</tr>
<tr>
<td><strong>Primal</strong></td>
<td>Geometric primitives (point, vector, triangle, etc.) and operations (distance, intersection, closest point, etc.)</td>
</tr>
<tr>
<td><strong>Spin</strong></td>
<td>Spatial acceleration data structures; octree, kd-tree, R-tree, BVH, etc.</td>
</tr>
<tr>
<td><strong>Mint</strong></td>
<td>Mesh data model; structured, unstructured, particles.</td>
</tr>
<tr>
<td><strong>Slam</strong></td>
<td>Set, relation, map abstractions.</td>
</tr>
<tr>
<td><strong>Slic/Lumberjack</strong></td>
<td>Unified/shared inter-package message streams, parallel logging, and filtering.</td>
</tr>
</tbody>
</table>

All Axom components provide native interfaces for C++, C, and Fortran (Python in the works).
Math + Physics Libraries
# Math + Physics Libraries

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>License</th>
<th>Maturity (years)</th>
<th>Website</th>
<th>Repository</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFEM</td>
<td>Unstructured high-order finite element library</td>
<td>BSD</td>
<td>~15</td>
<td>mfem.org</td>
<td>github.com/mfem</td>
<td>Tzanio Kolev</td>
</tr>
<tr>
<td>hypre</td>
<td>Preconditioners and solvers for large-scale matrices</td>
<td>Apache-2 or MIT</td>
<td>~20</td>
<td><a href="http://www.llnl.gov/casc/hypre">www.llnl.gov/casc/hypre</a></td>
<td>github.com/hypre-space</td>
<td>Rob Falgout</td>
</tr>
<tr>
<td>SUNDIALS</td>
<td>Nonlinear and differential/algebraic equation solvers</td>
<td>BSD</td>
<td>~20</td>
<td><a href="http://www.llnl.gov/casc/sundials">www.llnl.gov/casc/sundials</a></td>
<td>github.com/LLNL/sundials</td>
<td>Carol Woodward</td>
</tr>
<tr>
<td>XBraid</td>
<td>Lightweight support for multigrid Parallel-in-Time</td>
<td>LGPL-2.1</td>
<td>~5</td>
<td><a href="http://www.llnl.gov/casc/xbraid">www.llnl.gov/casc/xbraid</a></td>
<td>github.com/xbraid</td>
<td>Rob Falgout</td>
</tr>
</tbody>
</table>
MFEM
Lightweight, scalable C++ library for finite element methods

- Supports arbitrary high-order discretizations and meshes for a wide variety of applications
- Flexible discretizations on unstructured grids
  - Triangular, quadrilateral, tetrahedral and hexahedral meshes.
  - Local conforming and non-conforming refinement.
  - Bilinear/linear forms for variety of methods: Galerkin, DG, DPG, …
- High-order and scalable
  - Arbitrary-order H1, H(curl), H(div)- and L2 elements. Arbitrary order curvilinear meshes.
  - MPI scalable to millions of cores and GPU-accelerated. Enables application development on wide variety of platforms: from laptops to exascale machines.
- Built-in solvers and visualization
  - Integrated with: HYPRE, RAJA, UMPIRE, SUNDIALS, PETSc, SUPERLU, …
  - Accurate and flexible visualization with VisIt and GLVis
- Open source
  - BSD license with thousands of downloads/year worldwide.
  - Available on GitHub. Part of ECP’s CEED co-design center.
**Hypre**

Highly scalable multilevel solvers and preconditioners

- **Conceptual linear system interfaces**
  - Provides natural “views” of the linear system: structured, semi-structured, finite element, linear algebraic
  - Enables more efficient data storage schemes and kernels

- **Scalable preconditioners and solvers**
  - Structured and unstructured algebraic multigrid (including constant coefficient)
  - Maxwell solvers, H-div solvers, and more
  - Demonstrated scalability beyond 1M cores

- **Integrated with other math libraries**
  - SUNDIALS, PETSc, Trilinos

- **Unique, user-friendly interfaces**

- **Open source**
  - Used worldwide in a vast range of applications
  - Available on GitHub, Apache-2 or MIT license
SUNDIALS
Adaptive time integrators for ODEs and DAEs and efficient nonlinear solvers

- **ODE integrators:**
  - CVODE(S): variable order and step BDF (stiff) and Adams (non-stiff)
  - ARKode: variable step implicit, explicit, and additive IMEX Runge-Kutta

- **DAE integrators:** IDA(S) - variable order and step BDF integrators

- **Sensitivity analysis (SA):** CVODES and IDAS provide forward and adjoint SA

- **Nonlinear solvers:** KINSOL - Newton-Krylov, Picard, and accelerated fixed point

- **Modular design**
  - Written in C with interfaces to Fortran
  - Users can supply own data structures and solvers
  - Optional use structures: serial, MPI, threaded, CUDA, RAJA, hypre, & PETSc
  - Encapsulated parallelism

- **Open source**
  - Freely available (BSD License) from LLNL site, GitHub, and Spack
  - CMake-based portable build system
  - Can be used from MFEM, PETSc, and deal.II

- **Supported by extensive documentation, a sundials-users email list, and an active user community**

- **Used by thousands worldwide in applications from research and industry**
XBr aid
Parallel-in-time multigrid solver software

- Speeds up existing application codes by creating concurrency in the time dimension
- Unique non-intrusive approach
  - Builds as much as possible on existing codes and technologies
  - Converges to same solution as sequential code
- Demonstrated effectiveness and potential
  - Tech: Implicit, explicit, multistep, multistage, adaptivity in time and space, moving meshes, spatial coarsening, low storage approach
  - Apps: Linear/nonlinear diffusion, fluids (shocks), power grid (discontinuities), elasticity, optimization, …
  - Codes: Strand2D, Cart3D, LifeV, CHeart, GridDyn, …
- Leverages spatial multigrid research and experience
  - Extensive work developing scalable multigrid methods in hypre
- Open source
  - Available on GitHub, LGPL-2.1
SAMRAI
Structured adaptive mesh refinement applications infrastructure

- Object-oriented library, scalable and flexible for use in many applications
- Full support of AMR infrastructure
  - Multi-level dynamic gridding of AMR mesh
  - Transparent parallel communication (MPI)
  - Load balancing
  - Data type for common mesh centerings (cell, node, face, ...)
  - Data transfer operations (copy, coarsen, refine, time interpolation)
- Flexibility provided to applications
  - Applications provide numerical kernels to operate on distributed patches
  - Users may define and own their own data structures
  - Works on different geometries (Cartesian, staggered, multiblock, etc.)
  - Applications choose when and where to use SAMRAI data structures
  - Interfaces to solver libraries included (hypre, SUNDIALS, PETSc)
  - VisIt visualization and HDF5 checkpoint/restart supported
- Open source
  - LGPL 2.1 license, available on GitHub
Performance and Workflow Tools
# Performance and Workflow Tools

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>License</th>
<th>Maturity (years)</th>
<th>Website</th>
<th>Repository</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maestro WF</td>
<td>A tool and library for specifying and conducting general workflows</td>
<td>MIT</td>
<td>~2.5</td>
<td><a href="https://maestrowf.readthedocs.io">maestrowf.readthedocs.io</a></td>
<td><a href="https://github.com/LLNL/maestrowf">github.com/LLNL/maestrowf</a></td>
<td>Frank Di Natale</td>
</tr>
<tr>
<td>Spindle</td>
<td>Library loading and program start-up at scale</td>
<td>LGPL-2.1</td>
<td>~6</td>
<td><a href="http://computing.llnl.gov/projects/spindle">computing.llnl.gov/projects/spindle</a></td>
<td><a href="https://github.com/hpc/spindle">github.com/hpc/spindle</a></td>
<td>Matthew LeGendre</td>
</tr>
<tr>
<td>LBANN</td>
<td>Machine learning training and inference at extreme scale</td>
<td>Apache-2</td>
<td>~5.5</td>
<td><a href="https://lbann.readthedocs.io">lbann.readthedocs.io</a></td>
<td><a href="https://github.com/LLNL/lbann">github.com/LLNL/lbann</a></td>
<td>Brian Van Essen</td>
</tr>
</tbody>
</table>
Caliper
A library for always-on performance monitoring

- Add simple annotations to source code
  - Physics regions, Key loops, other semantics

```c
// Mark the "initialization" phase
CALI_MARK_BEGIN("initialization");
int count = 4;
double t = 0.0, delta_t = 1e-6;
CALI_MARK_END("initialization");

// Mark the loop
CALI_CXX_MARK_LOOP_BEGIN(mainloop, "main loop");
for (int i = 0; i < count; ++i) {
    // Mark each loop iteration
    CALI_CXX_MARK_LOOP_ITERATION(mainloop, i);

    // A Caliper snapshot taken at this point will contain
    // "function"="main", "loop"="main loop", "iteration=main loop"=<i>

    // ...
}
CALI_CXX_MARK_LOOP_END(mainloop);
```

- Link code with Caliper library from C++, C, or Fortran
- Attach arbitrary performance measurement tools to your regions
- Leave Caliper in and always have performance data available
SPOT
Performance analysis and history tracking

- Collect performance results from arbitrary application runs, track performance across users and history
- Integrate performance analysis tools into applications
  - Annotate code regions with Caliper
  - Control performance collection through command line or input deck
  - Store history of performance data and visualize through web interfaces
- Caliper interfaces with applications
  - Annotation interface puts labels on code and data regions
  - Variety of metrics (time, memory bandwidth, MPI usage, etc.) are collected and reported against annotation labels.
  - More reliable than traditional performance tools.
- SPOT visualizes history of Caliper-collected runs
  - Any application run can report performance data to SPOT.
  - Track how performance changes with code releases and across systems
  - Explore performance data to identify issues
- Under active development & integrated into several large codes
Flux

Next-generation resource management and scheduling framework to address emerging challenges

- **Workflow challenges**
  - Modern Workflows are increasingly difficult to schedule
  - Cancer Moonshot Pilot2, Machine Learning LDRD Strategic Initiative, …

- **Resource challenges**
  - Changes in resource types are equally challenging
  - GPGPUs, Burst buffers, under-provisioned PFS BW, …

- Fully hierarchical approach for job throughput/co-scheduling
- Graph-based resource model for resource challenges
- Rich APIs for workflow communication and coordination
- Consistent APIs for workflow portability and reproducibility
MaestroWF
A standard framework to make simulation studies easier to manage, run, and expand

- **Consistent study specification definition**
  - Specify multi-step workflows in a human-readable and self-documenting YAML specification.
  - Studies can be linear or parameterized, are easily shareable between users, and can be software generated.
  - Easily repeat studies simply by launching an existing specification.

- **Lightweight workflow automation and monitoring**
  - Studies are parsed, expanded based on parameters, and monitored automatically.
  - Workflows are expanded into DAGs, with workflow steps being launched as their dependencies allow them.

- **Easy for users to specify and launch workflows**
  - Specifications being shareable allows existing studies to serve as templates for new ones (making both set up and knowledge sharing easier).
  - A study specification allows users to build standard infrastructure to generate the necessary YAML to run larger collections of studies.
Spindle
Scalable application start-up

- **Job launch not scalable with many libraries or Python**
  - Solves start-up issues from loading libraries and Python modules at scale
  - Nodes hammer shared file systems when searching and loading libraries
  - Impacts users across whole center

- **Spindle makes job launch scalable**
  - Single node loads libraries/python-modules.
  - Broadcasts libraries to other nodes over high-bandwidth communication network.
  - Run by: `% spindle srun -n 512 ./myapp`

- **Open source**
  - LGPL-2.1 with thousands of downloads/year worldwide
  - Available on GitHub

---

**Weak Scaling Dynamic with and without SPINDLE**

![Graph showing weak scaling dynamic with and without SPINDLE. The graph compares execution time in seconds against job size in nodes (12 processes/node). The red line represents Dynamic without SPINDLE, and the blue line represents Dynamic with SPINDLE.](image)
LBANN
Livermore Big Artificial Neural Network Toolkit

- **Distributed deep learning training and inference**
  - Optimize for strong and weak scaling network training
  - Train large networks quickly
  - Enable training on data samples or data sets too large for other frameworks (e.g., 3D data cubes, billion sample data sets)
  - Optimized distributed memory algorithm
  - Including spatially decomposed convolutions
  - Multi-level parallelism (model / data / ensemble)
  - Hydrogen GPU-accelerated distributed linear algebra library
  - Optimized asynchronous GPU-aware communication library

- **Utilize unique HPC resources at scale**
  - InfiniBand and next-generation interconnect
    - Low latency / high cross-section bandwidth
  - Tightly-coupled GPU accelerators
  - Node-local NVRAM
  - High bandwidth parallel file system

- **C++ / MPI + OpenMP / CUDA / ROCm / NCCL / cuDNN**

- **Open source under Apache license**
  - [github.com/LLNL/lbann](https://github.com/LLNL/lbann)
  - [github.com/LLNL/Elemental](https://github.com/LLNL/Elemental)
  - [github.com/LLNL/Aluminum](https://github.com/LLNL/Aluminum)
Data Management and Visualization
## Data Management and Visualization

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>License</th>
<th>Maturity (years)</th>
<th>Website</th>
<th>Repository</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conduit</td>
<td>Simplified data exchange for HPC simulations</td>
<td>BSD</td>
<td>~6</td>
<td>software.llnl.gov/conduit</td>
<td>github.com/llnl/conduit</td>
<td>Cyrus Harrison</td>
</tr>
<tr>
<td>Ascent</td>
<td>Flyweight in situ visualization and analysis for HPC simulations</td>
<td>BSD</td>
<td>~4</td>
<td>ascent-dav.org</td>
<td>github.com/alpine-dav/ascent</td>
<td>Matt Larsen</td>
</tr>
<tr>
<td>zfp</td>
<td>In-memory compression of floating-point arrays</td>
<td>BSD</td>
<td>~6</td>
<td>zfp.readthedocs.io</td>
<td>github.com/LLNL/zfp</td>
<td>Peter Lindstrom</td>
</tr>
<tr>
<td>SCR</td>
<td>Multilevel checkpointing support and burst buffer interface</td>
<td>BSD</td>
<td>~13</td>
<td>scr.readthedocs.io</td>
<td>github.com/LLNL/scr/</td>
<td>Kathryn Mohror</td>
</tr>
<tr>
<td>VisIt</td>
<td>Feature-rich mesh-based visualization and analysis platform</td>
<td>BSD</td>
<td>~20</td>
<td>visit.llnl.gov</td>
<td>github.com/visit-dav/visit</td>
<td>Cyrus Harrison</td>
</tr>
<tr>
<td>GLVis</td>
<td>Lightweight high order visualization for MFEM</td>
<td>LGPL-2.1</td>
<td>~11</td>
<td>glvis.org</td>
<td>github.com/GLVis/glvis</td>
<td>Tzanio Kolev</td>
</tr>
</tbody>
</table>

### Technical Contact

- **Cyrus Harrison**
Conduit
Simplified data exchange for HPC simulations

- Provides an intuitive API for in-memory data description
  - Enables *human-friendly* hierarchical data organization
  - Can describe in-memory arrays without copying
  - Provides C++, C, Python, and Fortran APIs

- Provides common conventions for exchanging complex data
  - Shared conventions for passing complex data (e.g., simulation meshes) enable modular interfaces across software libraries and simulation applications

- Provides easy to use I/O interfaces for moving and storing data
  - Enables use cases like binary checkpoint restart
  - Supports moving complex data with MPI (serialization)

- Open source
  - Leveraged by Ascent, VisIt, and Axom

Hierarchical in-memory data description

Conventions for sharing in-memory mesh data
Ascent
Flyweight in-situ visualization and analysis for HPC simulations

- Ascent is an easy to use in-memory visualization and analysis library
  - Use cases: making pictures, transforming data, and capturing data
  - Young effort, yet already supports most common visualization operations
  - Provides a simple infrastructure to integrate custom analysis
  - Provides C++, C, Python, and Fortran APIs

- Uses a flyweight design targeted at next-generation HPC platforms
  - Efficient distributed-memory (MPI) and many-core (CUDA or OpenMP) execution
  - Has lower memory requirements then current tools
    - Demonstrated scaling: In situ filtering and ray tracing across 16,384 GPUs on LLNL's Sierra Cluster
    - Requires less dependencies than current tools (e.g., no OpenGL)

- Open source
  - Leverages Conduit, will also be released with Visit
Production end-user tool supporting scientific and engineering applications

- Use cases: data exploration, quantitative analysis, visual debugging, comparative analysis and generation of presentation graphics
  - Provides a rich feature set and a flexible data model suitable for many scientific domains
  - Includes more than 100 file format readers
  - Provides GUI and Python interfaces, extendable via C++ and Python

Provides parallel post-processing infrastructure that scales from desktops to massive HPC clusters

- Uses MPI for distributed-memory parallelism on HPC clusters
- Development underway to leverage on-node many-core (CUDA or OpenMP) parallelism

Open source

- Used as a platform to deploy research from the DOE visualization community
- Initially developed by LLNL to support ASC, now co-developed by several organizations
GLVis
Lightweight OpenGL tool for accurate and flexible interactive finite element visualization

- **Accurate visualization**
  - 1D/2D/3D, volume/surface, triangular/quad/tet/hex, low/high-order meshes
  - Arbitrary high-order, scalar and vector finite element and NURBS solutions
  - Visualization of parallel meshes and solutions

- **Lightweight and interactive**
  - Unlimited number of refinement and de-refinement levels
  - Support for antialiasing, accurate cutting planes, materials, lighting, and transparency
  - Processor and element shrinking for better visualization of 3D mesh interiors

- **Flexible server support**
  - Simultaneous visualization of multiple fields/meshes in separate GLVis windows
  - Local visualization for remote parallel runs with secure socket connections
  - Persistent visualization of time-evolving fields

- **Open source**
  - LGPL-2.1. Available on GitHub
  - Based on the MFEM finite element library
  - Used in MFEM, MARBL/BLAST, LiDO, and more
Scalable Checkpoint/Restart (SCR) Library
Enables fast, portable I/O to burst buffers across HPC systems

- **SCR provides fast, scalable I/O performance for LLNL applications**
  - SCR caches output data in node local storage like RAM disk or burst buffer, which can be as much as 1000x faster than the parallel file system
  - SCR hides the complexity of different burst buffer systems and storage architectures

- **Easy integration into application codes**
  - Simple wrapper API around existing checkpoint/restart code
  - Full featured scripting tools wrap existing job launch commands, e.g. srn → scr_srun

- **SCR now enables fast I/O for general output from applications**
  - SCR can now cache visualization dumps or other output to node local storage and drain data to the parallel file system in the background
  - Applications can output data more frequently without the overhead

- **Open source**
  - Available on GitHub with BSD license
ZFP
In-memory compression of floating-point and integer arrays

- Provides a conventional array interface for multidimensional scalar fields
  - Supports constant-time read & write random access to any array element
  - Hides complexity of (de)compression via C++ operator overloading
  - Provides efficient data access via iterators, views, proxy references and pointers
  - Supports thread safe access and STL algorithms

- Provides a simple API for (de)compression of whole arrays
  - Supports prescribed error tolerance or precision, exact storage, lossless compression
  - Supports OpenMP and CUDA parallel (de)compression at up to 150 GB/s throughput
  - Provides C++, C, Python, and Fortran APIs
  - Suitable for compressing checkpoints, viz dumps, MPI messages, CPU-GPU transfers

- Open source
  - BSD licensed and available via GitHub, Spack, and Fedora RPM
  - Supported by Intel IPP, HDF5, Silo, ADIOS, VTK-m, LEOS, E4S, …