Conduit

A Successful Strategy for Describing and Sharing Data In Situ

https://github.com/llnl/conduit

ISAV 2022

Sunday November 13th, 2022

Cyrus Harrison, Matt Larsen, Brian Ryujin, Adam Kunen, Arlie Capps, Justin Privitera, and other Conduit contributors.

LLNL-PRES-842096

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.
Lawrence Livermore National Security, LLC

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

Abstract

- Data representation and coupling between scientific libraries is a key challenge to building a vibrant ecosystem of HPC simulation tools. From bespoke data structures to hundreds of file-based data models, the myriad of possible choices involved both enables key features and blocks adoption of others. Connecting data between code bases requires agreeing on or adapting between data representations. While in some cases this process is trivial, for more complicated cases, adapting data becomes a costly barrier. Conduit was designed within this context to help meet the key challenge of sharing data across HPC simulation tools by providing a dynamic API to describe in-memory data. It supports coupling simulations and connecting simulations to analysis and I/O libraries.

- Conduit is an open-source project from Lawrence Livermore National Laboratory. It started in 2013 and has evolved through co-design with simulation applications and in situ tools since. Conduit is now an established part of LLNL's simulation data management strategy and has been adopted as the mesh-data interface for DOE community in situ visualization tools. While Conduit has been discussed briefly in prior in situ research, this paper provides a broader overview of Conduit, background on the evolution of the project, and details on recently added features relevant to in situ use cases.

Outline

- Intro
- Conduit Node
- Relay
- Blueprint
- Evolution
- Conclusion

Where we started:
Pain with sharing mesh data between simulations and tools

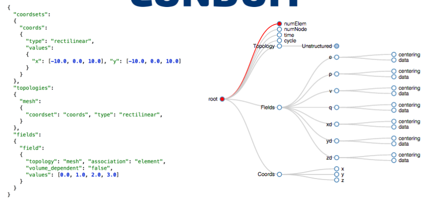
- Common approaches in the early 2010s:
  - Bespoke packed data arrays
  - Static file-based I/O Data Model APIs
- Both approaches were hard to extend:
  - Strict APIs lacked naming flexibly
  - Interpreting bespoke arrays was error prone
  - Data description was tied to file-based I/O

This pain motived us to create an `in-memory` first approach to data sharing

Conduit provides intuitive APIs for in-memory data description and exchange

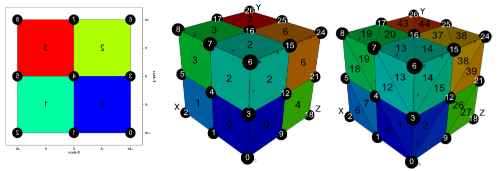
- **Provides an intuitive API for in-memory data description**
  - Enables *human-friendly* hierarchical data organization
  - Can describe in-memory arrays without copying
  - Provides C++, C, Python, and Fortran APIs
- **Provides common conventions for exchanging complex data**
  - Shared conventions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

CONDUIT



```
{
  "coordsets":
  {
    "coords":
    {
      "type": "rectilinear",
      "values":
      {
        "x": [-10.0, 0.0, 10.0], "y": [-10.0, 0.0, 10.0]
      }
    }
  },
  "topologies":
  {
    "mesh":
    {
      "coordset": "coords", "type": "rectilinear",
    }
  },
  "fields":
  {
    "field":
    {
      "topology": "mesh", "association": "element",
      "volume_dependent": "false",
      "values": [0.0, 1.0, 2.0, 3.0]
    }
  }
}
```

Hierarchical in-memory data description



Conventions for sharing in-memory mesh data

http://software.llnl.gov/conduit
http://github.com/llnl/conduit

Website and GitHub Repo

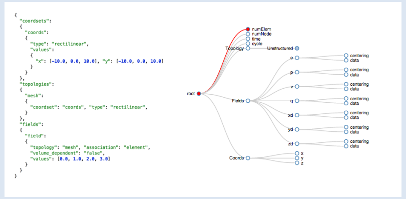
Projects are leveraging Conduit to support a wide range of capabilities

- Creating in-memory data stores
- Checkpoint restart of simulation data
- Conduit Node/YAML/JSON based:
  - Data APIs
  - User APIs
- Tree + Path-based I/O and partial I/O with:
  - YAML/JSON
  - HDF5
- Moving complex data with MPI
- Distributing work with MPI
- Sharing ad-hoc data between programs written in multiple languages (both in-memory and via files)
- Sharing simulation mesh data (both in-memory and via files)
- Transforming mesh data (topology changes, partitioning, flattening, etc)

Conduit's *Relay* and *Blueprint* libraries provide features built on top of Conduit's core data model

Conduit

Implements interfaces to Conduit's in-memory data model

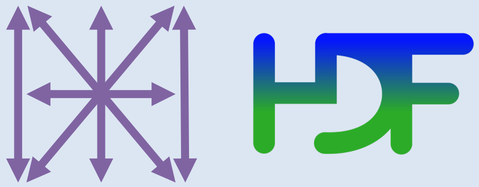


```
{
  "coordsets":
  {
    "coords":
    {
      "type": "rectilinear",
      "values":
      {
        "x": [-10.0, 0.0, 10.0], "y": [-10.0, 0.0, 10.0]
      }
    }
  },
  "topologies":
  {
    "mesh":
    {
      "coordset": "coords", "type": "rectilinear",
    }
  },
  "fields":
  {
    "field":
    {
      "topology": "mesh", "association": "element",
      "volume_dependent": "false",
      "values": [0.0, 1.0, 2.0, 3.0]
    }
  }
}
```

- Core Objects
- YAML/JSON parsing
- Basic I/O
- Basic transforms

Relay

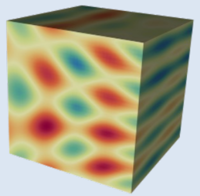
Provides advanced I/O features built on top of Conduit's data model

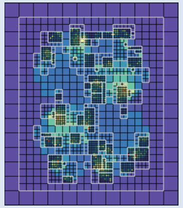


- File-based I/O: HDF5, Silo
- MPI
- WebSockets
- ZFP

Blueprint

Supports shared higher-level conventions for using Conduit to represent data



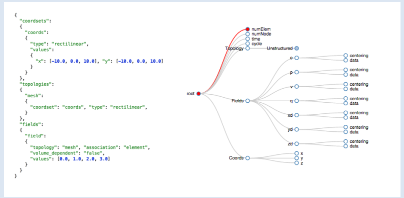


- Computational Meshes
- Multi-component Arrays
- One-to-many Relations
- Example Meshes
- Mesh Transforms

The heart of Conduit is a hierarchical variant type: *Node*

Conduit

Implements interfaces to Conduit's in-memory data model



```
{
  "coordsets":
  {
    "coords":
    {
      "type": "rectilinear",
      "values":
      {
        "x": [-10.0, 0.0, 10.0], "y": [-10.0, 0.0, 10.0]
      }
    }
  },
  "topologies":
  {
    "mesh":
    {
      "coordset": "coords", "type": "rectilinear",
    }
  },
  "fields":
  {
    "field":
    {
      "topology": "mesh", "association": "element",
      "volume_dependent": "false",
      "values": [0.0, 1.0, 2.0, 3.0]
    }
  }
}
```

- Core Objects
- YAML/JSON parsing
- Basic I/O
- Basic transforms

The heart of Conduit is a hierarchical variant type: *Node*

***conduit::Node* provides three core features:**

- An Array Representation
  - Bitwidth-style Type
  - Size, Offset, Stride, Element Bytes, Endianness
- Zero-Copy Support
  - Owned *`set()`* vs External Data *`set_external()`*
- A Tree-based Hierarchy
  - Dynamic *path-based* creation and access
  - *Compact* and *Contiguous* Tree Properties

The heart of Conduit is a hierarchical variant type: *Node*

A *conduit::Node* acts as one of following basic roles:

- *Object*: An ordered associative array mapping names to children
- *List*: An ordered list of unnamed children
- *Leaf*: Scalar or 1D Array of bitwidth-specified primitives:
  - *Signed Integers*: int8, int16, int32, int64
  - *Unsigned Integers*: uint8, uint16, uint32, uint64
  - *Floating Point Numbers*: float32, float64
  - *Strings*: char8_str
- *Empty*: No data

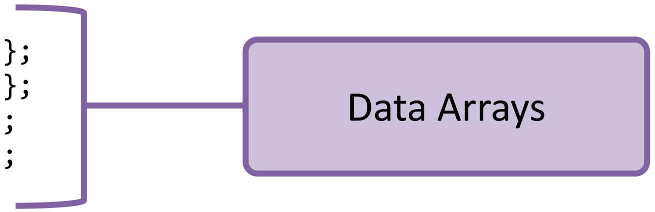
Experience with NumPy and JSON motivated Conduit's data model

The heart of Conduit is a hierarchical variant type: *Node*

Let's look at a simple C++ example expressing a hierarchy …

The heart of Conduit is a hierarchical variant type: *Node*

```
// define our data arrays
double x_coords[5] = { -1.0, 0.0, 0.0, 0.0, 1.0 };
double y_coords[5] = { 0.0, -1.0, 0.0, 1.0, 0.0 };
double z_coords[5] = { 0.0, 0.0, 1.0, 0.0, 0.0 };
int connectivity[8] = { 0, 1, 3, 2, 4, 3, 1, 2 };
double density[2] = { 1.0, 2.0};
```



The heart of Conduit is a hierarchical variant type: *Node*

```
// define our data arrays
double x_coords[5] = { -1.0, 0.0, 0.0, 0.0, 1.0 };
double y_coords[5] = { 0.0, -1.0, 0.0, 1.0, 0.0 };
double z_coords[5] = { 0.0, 0.0, 1.0, 0.0, 0.0 };
int connectivity[8] = { 0, 1, 3, 2, 4, 3, 1, 2 };
double density[2] = { 1.0, 2.0};

// create our tree
conduit::Node mesh;
```

Create a Node

The heart of Conduit is a hierarchical variant type: *Node*

```
// define our data arrays
double x_coords[5] = { -1.0, 0.0, 0.0, 0.0, 1.0 };
double y_coords[5] = { 0.0, -1.0, 0.0, 1.0, 0.0 };
double z_coords[5] = { 0.0, 0.0, 1.0, 0.0, 0.0 };
int connectivity[8] = { 0, 1, 3, 2, 4, 3, 1, 2 };
double density[2] = { 1.0, 2.0};

// create our tree
conduit::Node mesh;
mesh["coordsets/coords/type"] = "explicit";
```

Add data

Slashes in paths create a hierarchy

The heart of Conduit is a hierarchical variant type: *Node*

```cpp
// define our data arrays
double x_coords[5] = { -1.0, 0.0, 0.0, 0.0, 1.0 };
double y_coords[5] = { 0.0, -1.0, 0.0, 1.0, 0.0 };
double z_coords[5] = { 0.0, 0.0, 1.0, 0.0, 0.0 };
int connectivity[8] = { 0, 1, 3, 2, 4, 3, 1, 2 };
double density[2] = { 1.0, 2.0};

// create our tree
conduit::Node mesh;
mesh["coordsets/coords/type"] = "explicit";
mesh["coordsets/coords/values/x"].set_external(x_coords, 5);
mesh["coordsets/coords/values/y"].set_external(y_coords, 5);
mesh["coordsets/coords/values/z"].set_external(z_coords, 5);
```

Add data

set_external() zero copies existing data

The heart of Conduit is a hierarchical variant type: *Node*

```
// define our data arrays
double x_coords[5] = { -1.0, 0.0, 0.0, 0.0, 1.0 };
double y_coords[5] = { 0.0, -1.0, 0.0, 1.0, 0.0 };
double z_coords[5] = { 0.0, 0.0, 1.0, 0.0, 0.0 };
int connectivity[8] = { 0, 1, 3, 2, 4, 3, 1, 2 };
double density[2] = { 1.0, 2.0};
// create our tree
conduit::Node mesh;
mesh["coordsets/coords/type"] = "explicit";
mesh["coordsets/coords/values/x"].set_external(x_coords, 5);
mesh["coordsets/coords/values/y"].set_external(y_coords, 5);
mesh["coordsets/coords/values/z"].set_external(z_coords, 5);
mesh["topologies/topo/type"] = "unstructured";
mesh["topologies/topo/coordset"] = "coords";
mesh["topologies/mesh/elements/shape"] = "tet";
mesh["topologies/mesh/elements/connectivity"].set_external(connectivity, 8);
mesh["fields/density/association"] = "element";
mesh["fields/density/topology"] = ”topo";
mesh["fields/density/values"].set_external(density, 2);
// print our tree
std::cout << mesh.to_yaml() << std::endl;
```

The heart of Conduit is a hierarchical variant type: *Node*

```
coordsets:
  coords:
    type: "explicit"
    values:
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]
      y: [0.0, -1.0, 0.0, 1.0, 0.0]
      z: [0.0, 0.0, 1.0, 0.0, 0.0]
topologies:
  topo:
    type: "unstructured"
    coordset: "coords"
    elements:
      shape: "tet"
      connectivity: [0, 1, 3, 2, 4, 3, 1, 2]
fields:
  density:
    association: "element”
    topology: ”topo"
    values: [1.0, 2.0]
```

Example YAML Output

The heart of Conduit is a hierarchical variant type: *Node*

```
// examples of accessing data from leave Node
// exact type pointer access
double *          zs_ptr = mesh["coordsets/coords/values/x"].value();
// exact type array access
float64_array     zs_arr = mesh["coordsets/coords/values/x"].value();
// coerced type view access
float32_accessor ss_acc = mesh["coordsets/coords/values/x"].value();

for(int i = 0; i < zs_arr.dtype().number_of_elements(); i++)
{
   std::cout << " " << i << "]"
             << " ptr: " << zs_ptr[i]
             << " arr: " << zs_arr[i]
             << " acc: " << zs_acc[i] << std::endl;
}
```

```
[0] ptr: 0 arr: 0 acc: 0
[1] ptr: 0 arr: 0 acc: 0
[2] ptr: 1 arr: 1 acc: 1
[3] ptr: 0 arr: 0 acc: 0
[4] ptr: 0 arr: 0 acc: 0
```

C++ Node Leaf Access Example Output

Beyond setup and access methods, *Node* also provides methods to help you easily process and compare trees

- Inspect the Memory layout of
  - Leaves
  - Entire Trees
- Print Tree Summaries
- Calculate Leaf Summary Metrics
- Compact / Serialize Trees
- Iterate over Children
- Calculate Tree Differences (`diff` with adjustable tolerance)
- Parse and Create
  - YAML
  - JSON

The Conduit Relay library provides advanced I/O features built on top of Conduit’s data model

Relay

Provides advanced I/O features built on top of Conduit’s data model

- File-based I/O: HDF5, Silo
- MPI
- WebSockets
- ZFP

The Conduit Relay library provides advanced I/O features built on top of Conduit's data model

Relay HDF5 I/O

- *Node* round trip to/from HDF5 files
- Underpins checkpoint restart I/O and data extracts
- *Unbound* and *Handle* based interfaces
- Supports path-based partial I/O

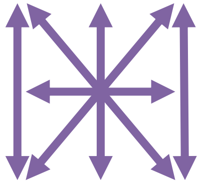
```
// save a Node tree to hdf5
conduit::relay::io::save(mesh,"mydata.hdf5");

// load hdf5 data into a Node tree
conduit::Node mydata;
conduit::relay::io::load("mydata.hdf5”, mydata);
```

Relay HDF5 I/O Example

The Conduit Relay library provides advanced I/O features built on top of Conduit's data model

Relay MPI

- *Node* based MPI functions
  - `send(), recv()`
  - `isend(), irecv()`
  - `gather(), all_gather()`
  - `reduce(), all_reduce()`
  - `broadcast()`
- *Schema-aware* and *Data-only* options



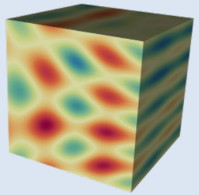
```
int tag = 0;
if(my_rank == 0)
{
  // send a Node tree to rank 1
  conduit::relay::mpi::send_using_schema(mesh, 1,
                                              tag,
                                              MPI_COMM_WORLD);

}
else if(my_rank == 1)
{
  // receive a Node tree from  rank 0
  conduit::relay::mpi::recv_using_schema(mesh, 0,
                                              tag,
                                              MPI_COMM_WORLD);
}
```

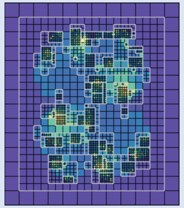
Relay MPI send/recv Example

The Conduit Blueprint library provides tools to share common flavors of data with Conduit

Blueprint

Supports shared higher-level conventions for using Conduit to represent data





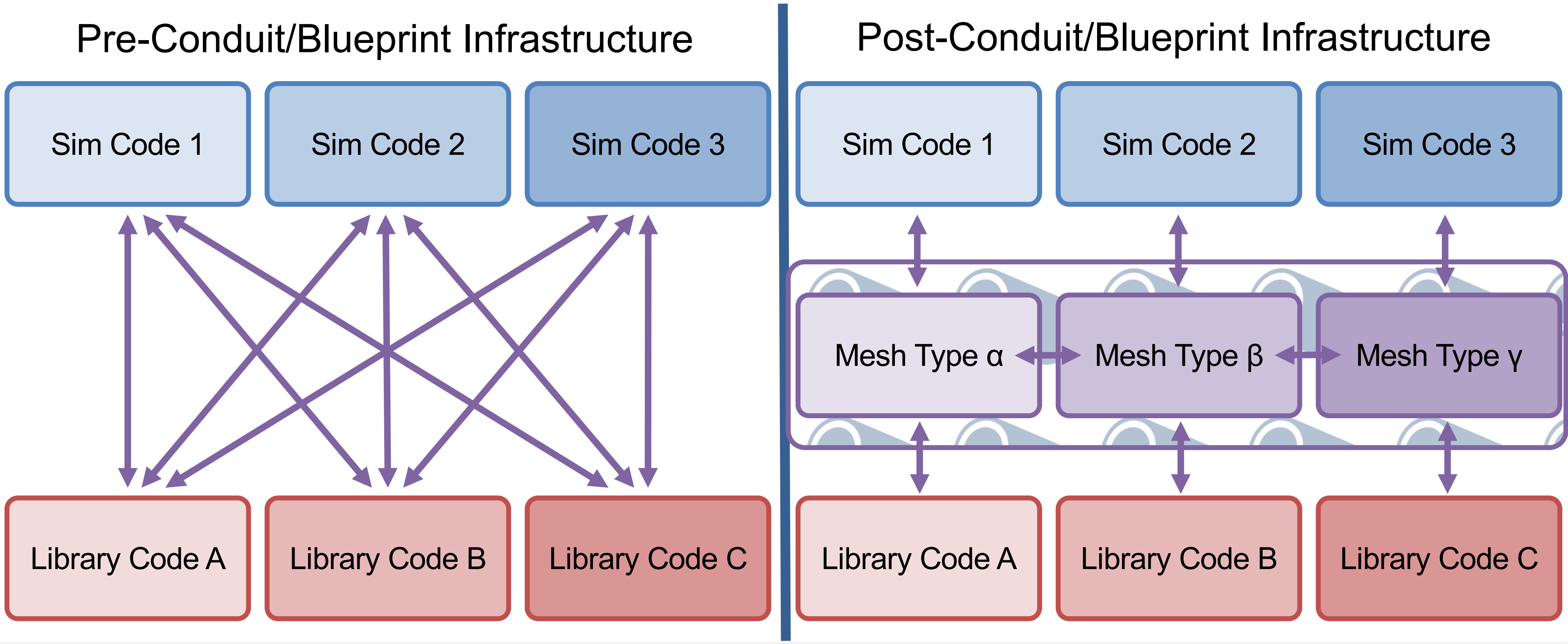
- Computational Meshes
- Multi-component Arrays
- One-to-many Relations
- Example Meshes
- Mesh Transforms

Components of the HPC simulation ecosystem implement and leverage a wide range of mesh data structures and APIs

- A variety of simulation codes leverage their own bespoke in-memory mesh data models.
- Other tools leverage a range of mesh-focused toolkits, frameworks, and APIs including: VTK, VTK-m, MFEM, and SAMRAI
- A wide set of powerful analysis tools are mesh agnostic (NumPy, PyTorch, etc) and recasting mesh data into these tools is a challenge
- *A single full-fledged API will never cover all use cases across the ecosystem*

The Mesh Blueprint is a strategy to describe and adapt mesh data between APIs

We are using Mesh Blueprint to simplify interfaces and support shared modular development



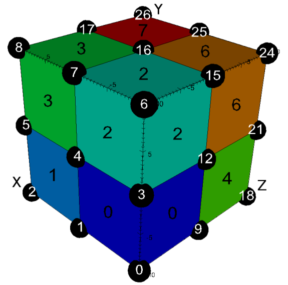
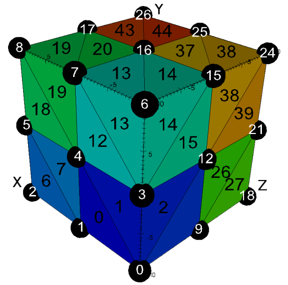
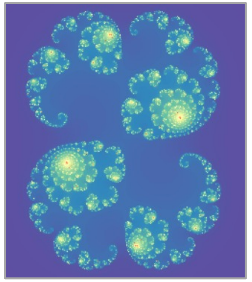
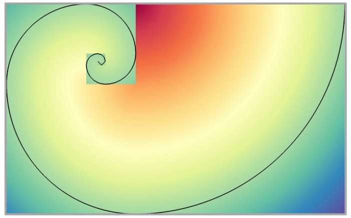
Blueprint is a strategy to share our investments and lower barriers to build and adopt new capabilities

- Blueprint interfaces connect simulation codes to a wide set of capabilities:
  - More Physics Packages
  - Visualization and Data Analysis
  - Mesh Transformations
  - I/O support
- Multiple codes and tools leverage Blueprint transforms
- Blueprint simplifies composing features across the ecosystem

Open-source and incremental adoption are also key aspects of this strategy

The Conduit Blueprint library facilitates using Mesh Blueprint data via three important capabilities

- Methods which verify if data conforms to blessed conventions at runtime
  - Provides detailed information for non-conforming data
- Methods that transform conforming data, including:
  - Coordinate Set and Topology transforms (e.g. Implicit Uniform to Explicit Coordinates)
  - Memory layout transforms (e.g. Contiguous to Interleaved to array layouts)
- Methods that generate mesh examples, which cover the menu of supported meshes

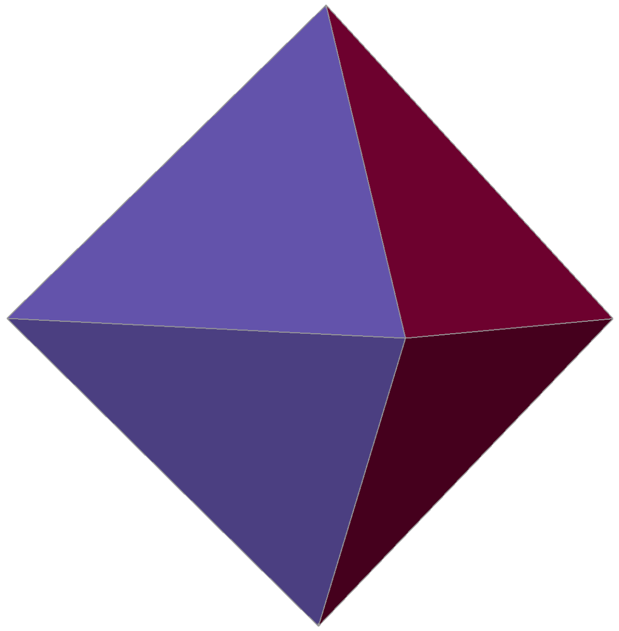





Mesh Examples from the Blueprint Library

Non-shocking Revelation:
Our prior example was a valid Blueprint Mesh

```
coordsets:
  coords:
    type: "explicit"
    values:
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]
      y: [0.0, -1.0, 0.0, 1.0, 0.0]
      z: [0.0, 0.0, 1.0, 0.0, 0.0]
topologies:
  topo:
    type: "unstructured"
    coordset: "coords"
    elements:
      shape: "tet"
      connectivity: [0, 1, 3, 2, 4, 3, 1, 2]
fields:
  density:
    association: "element”
    topology: ”topo"
    values: [1.0, 2.0]
```

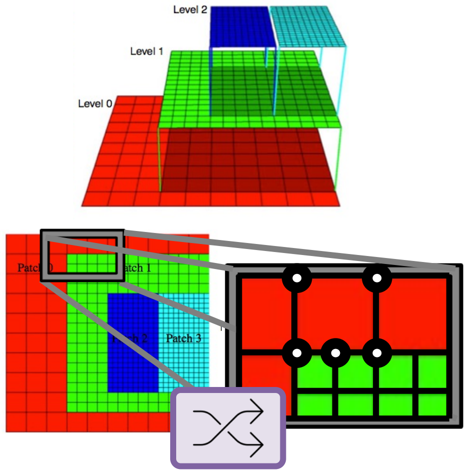
Example YAML Output

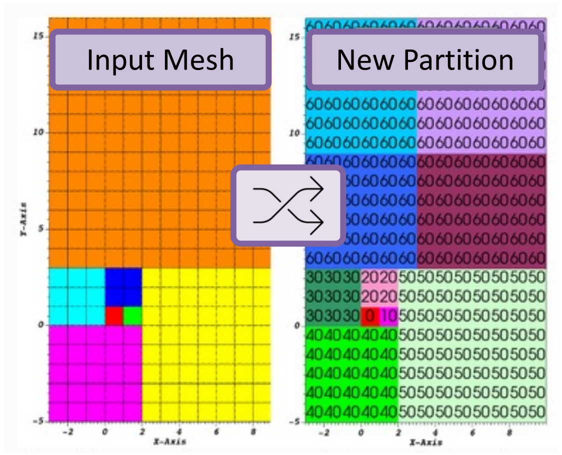


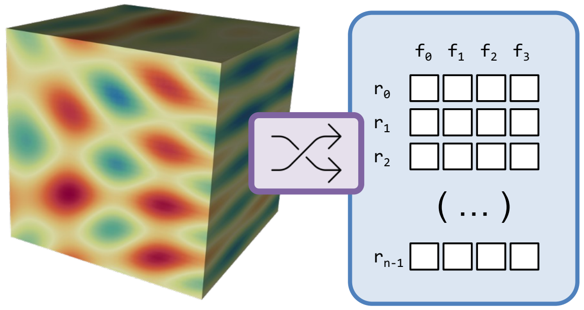
An unstructured tet mesh

Conduit Blueprint provides a range of Mesh Representation Transforms

- Transforms adapt mesh data for use across components of the ecosystem







Mesh Topology Transforms

N-to-M Mesh Partitioning

Mesh Flattening

Partitioning and Flattening were implemented via a LLNL + Intelligent Light Contract

Conduit evolved through collaboration and co-design of simulation and visualization tools over many years

Prototyping

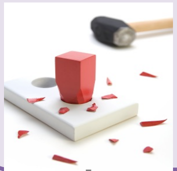
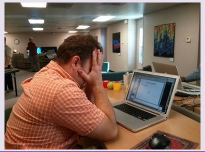
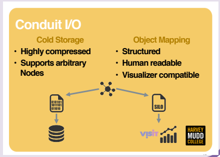
- 2011 Initial Brainstorming
- 2013 Creation at LLNL Hackathon
- 2014 – 2015 Harvey Mudd Clinic
- 2015 "Strawman"

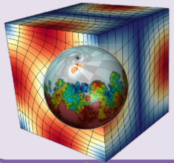
Adoption in Production Applications

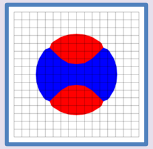
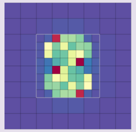
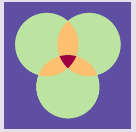
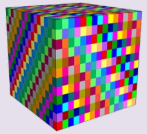
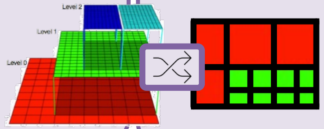
- In-memory Datastore and HDF5 Check-point Restart I/O for Simulations
- Ascent Project
- In situ Mesh Overlay, Multi-material, AMR, and Distributed Memory Mesh Info
- Broadening adoption

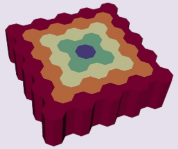
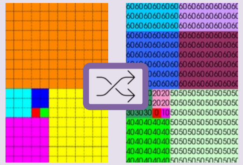
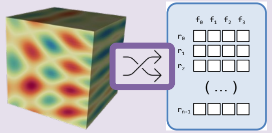
Investing in Shared Transforms

- Partitioning, Load-balancing, and Flattening
- Polytopal support, AMR to Polytopal
- Long-tail Blueprint support

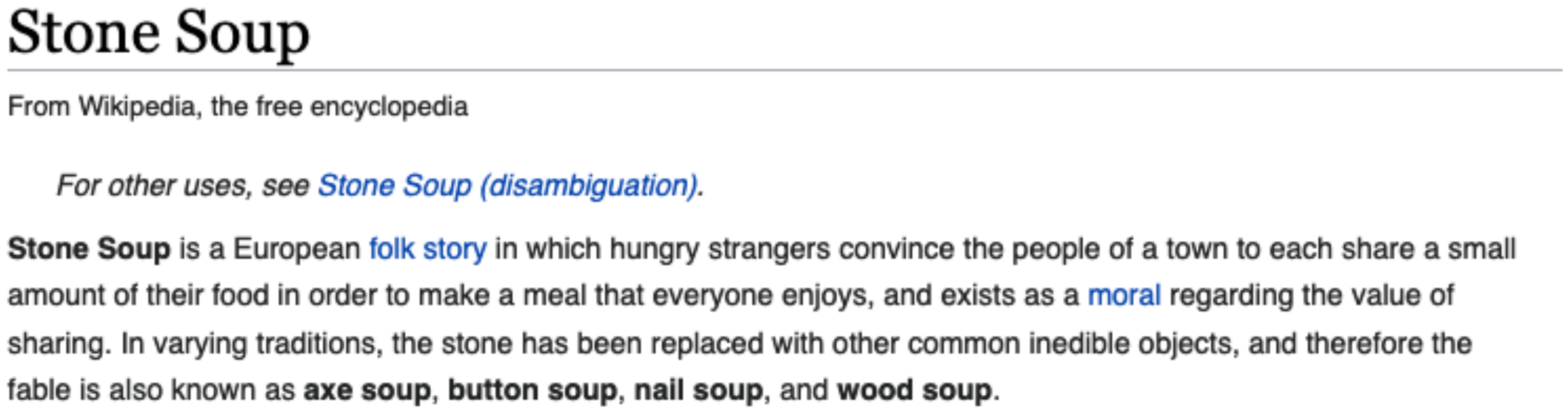


2011 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022

Conduit evolved through "Stone Soup" co-design



Stone Soup

From Wikipedia, the free encyclopedia

For other uses, see Stone Soup (disambiguation).

Stone Soup is a European folk story in which hungry strangers convince the people of a town to each share a small amount of their food in order to make a meal that everyone enjoys, and exists as a moral regarding the value of sharing. In varying traditions, the stone has been replaced with other common inedible objects, and therefore the fable is also known as **axe soup**, **button soup**, **nail soup**, and **wood soup**.

Conduit evolved through “Stone Soup + Dogfooding” co-design

We built Conduit to simplify data description and sharing across software components

- Conduit focuses on in-memory description and other use cases are built on that foundation

- Conduit simplifies writing HPC simulation and data analysis software
  - Streamlined I/O and MPI communication Relay APIs

- Conduit Blueprint connects codes to a rich ecosystem of mesh aware tools
  - Conduit Mesh Transforms
  - Blueprint interfaces to Ascent, Catalyst, Sensei, and VisIt

- Conduit Github: https://github.com/llnl/conduit

- Cyrus Contact Email: cyrush@llnl.gov

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.
Lawrence Livermore National Security, LLC

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

Lawrence Livermore National Laboratory

Backup Slides

References (Presentations and Related Papers)

- “Conduit: A Successful Strategy for Describing and Sharing Data In Situ”
  SC22 ISAV22 Workshop, Nov 2022
- “The Conduit Mesh Blueprint: Drafting a New Way to Share Simulation Meshes”
  DOE Computer Graphics Forum Talk, April 2019
- “The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman”
  In Proceedings of ISAV 2017 (SC17) Workshop, Denver CO, November 2017
- SciPy 2016 talk on Conduit (https://youtu.be/3_GKjeRUPKg)
- “Strawman: A Batch In Situ Visualization and Analysis Infrastructure for Multi-Physics Simulation Codes”
  In Proceedings of ISAV 2015 (SC15)Workshop, Austin TX, November 2015

References (Tutorials)

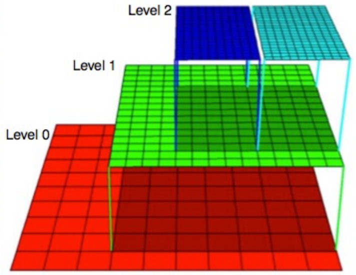
- Conduit User Tutorials (C++ & Python)
  https://llnl-conduit.readthedocs.io/en/latest/conduit.html

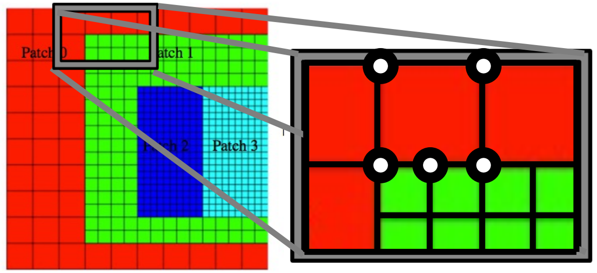
- Ascent Conduit Intro (C++ & Python)
  https://ascent.readthedocs.io/en/latest/Tutorial_Intro_Conduit_Basics.html

Conduit Blueprint provides a range of Mesh Representation Transforms

Use Cases:

- Transform a uniform mesh into an an unstructured mesh, so you can run an algorithm written for only unstructured meshes on this data
- Transform a polyhedral mesh into an unstructured tetrahedral mesh for visualization
- Transform a block structured AMR mesh to a polyhedral mesh, so it can be connected another package that only runs on polyhedra.

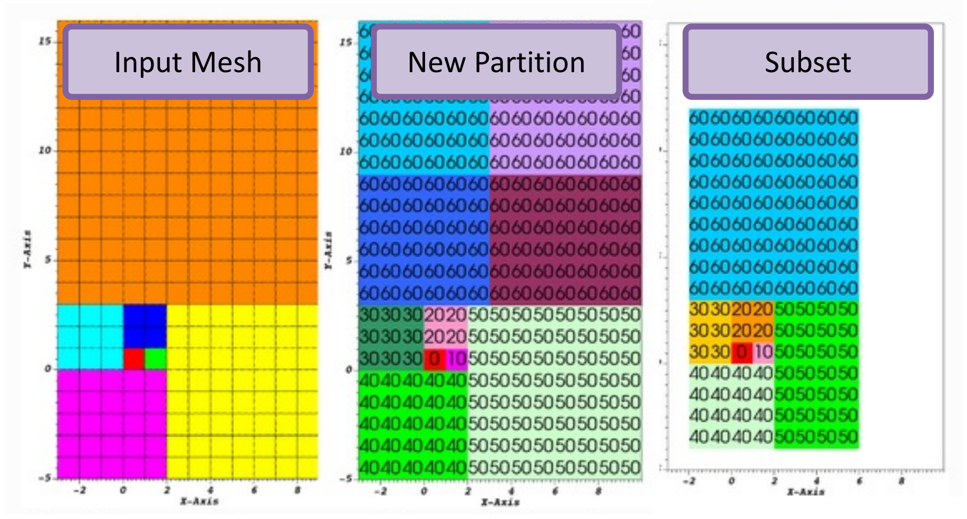




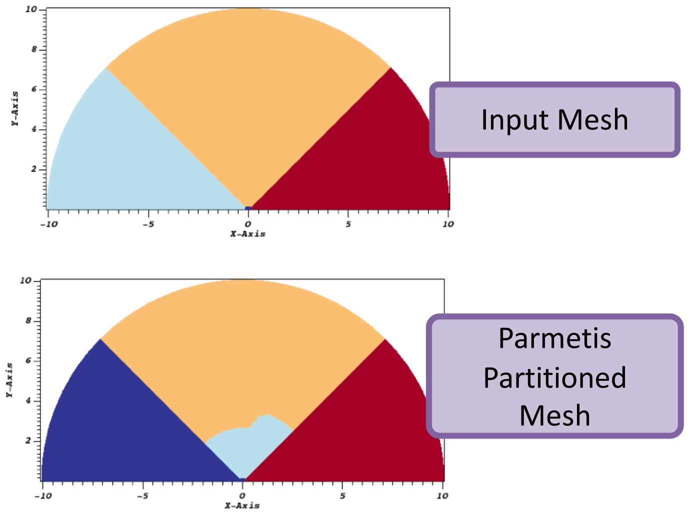
AMR to Polygonal Example

Conduit Blueprint provides general M-to-N mesh partitioning and subset selection tools

Use Cases: Split Domains, Fuse Domains, Load Balancing, Selections



`conduit::blueprint::mesh::partition()` Examples



Load Balancing Example

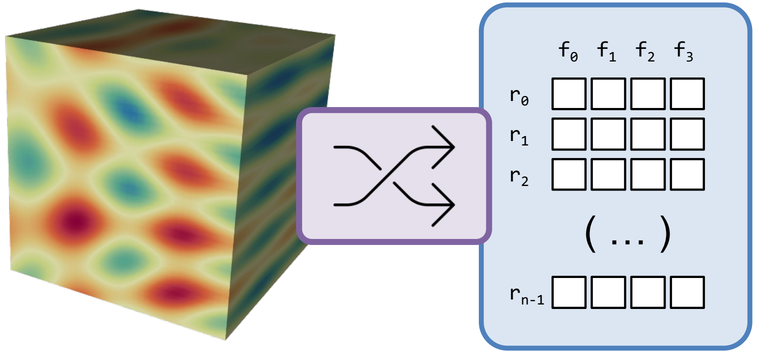
https://llnl-conduit.readthedocs.io/en/latest/blueprint_mesh_partition.html

Partitioning was implemented in a LLNL + Intelligent Light Contract

Conduit Blueprint provides a tool to flatten mesh field values into tables

Use Cases:

- Transform Mesh field data data into tables of per-element and per-vertex field values for further analysis
- Supports the common quandary:
  “I have a mesh, but I need a giant table with all my field values”
- Save tables to CSV or HDF5 for mesh agnostic tools (NumPy, PyTorch, etc) to digest



Flattening Example

Flattening was implemented in a LLNL + Intelligent Light Contract