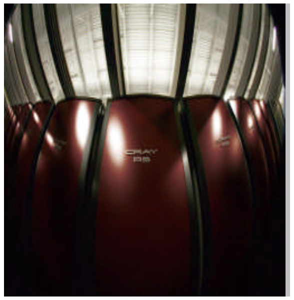
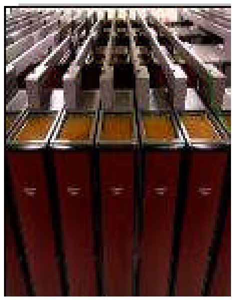




Sandia National Laboratories

Exceptional service in the national interest

Simulating Additive Manufacturing By Rethinking Simulation Structures

Jay Lofstead, John Mitchell

Enze Chen, Manisha Ganesh, Gavin St. John

gflofst@sandia.gov

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

SAND2020-3862 C

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Mod/Sim Typical Structure

- Examples:
  - CFD, heat transfer, molecular dynamics
- Compute on every element every timestep
- Mesh types vary, but most are 3D today
- Deploy across machine balancing compute vs. communication

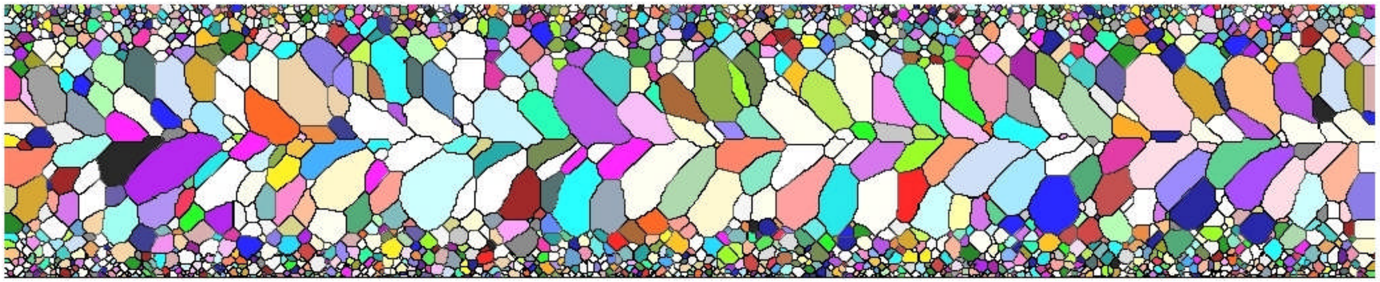
Another class of physical models

- Activity focused in a small subset of simulation domain
  - Welding
  - Additive manufacturing (small parts)
  - Spray
  - Pressure waves in material

- Domain too large to fit into memory
  - Additive manufacturing at scale
    - Satellite fuel tank cap 3D printed (1.16m diameter, 10 cm thick)
    - https://news.lockheedmartin.com/2018-07-11-Giant-Satellite-Fuel-Tank-Sets-New-Record-for-3-D-Printed-Space-Parts

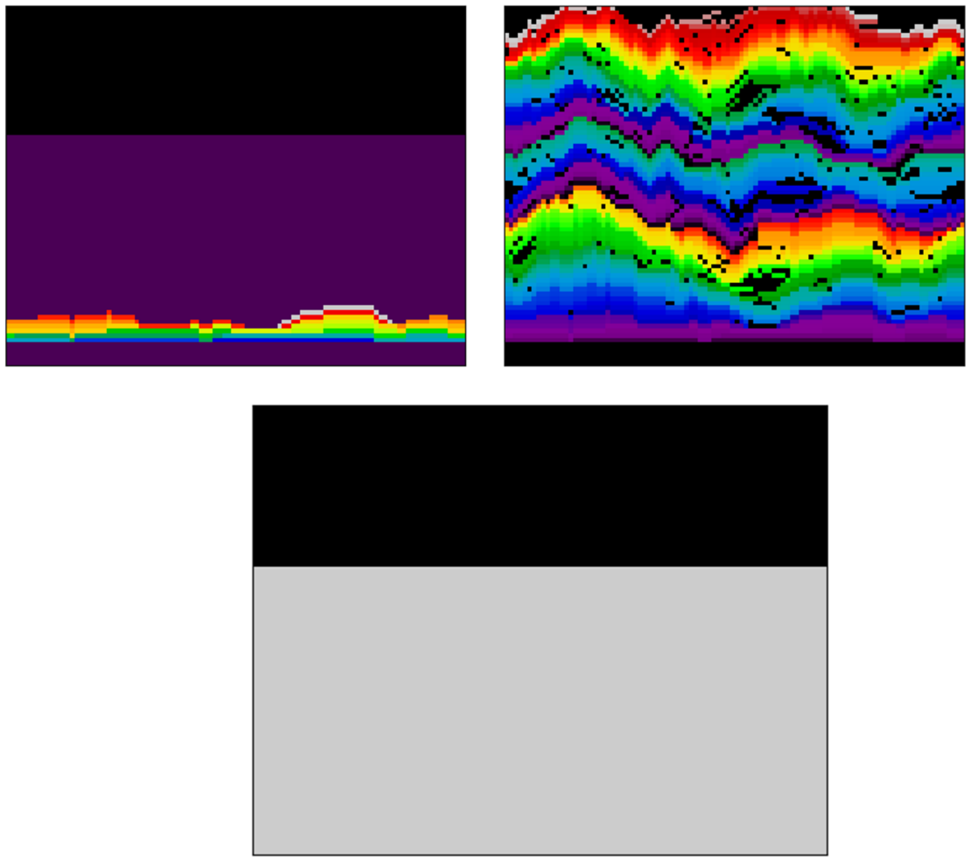
Welding example

- 1 site per micron
- Move heat affected zone across powder or piece edges to melt and form new piece



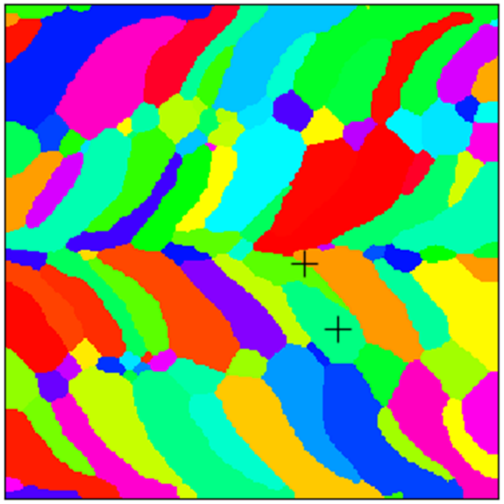
Spray accumulation model

- Spray a material (e.g., paint) and model the accumulation



Additive Manufacturing (single layer)

- Similar to welding, but sweeping and ultimately multi-layer
- Move heat affected zone across powder or piece edges to melt and form new piece



AM demo simulation

What these share

- In all cases, the area that changes after a computation round is highly localized
- Total area can be large leading to 1% or less use of allocated compute resources
- IO costs would dominate and storage space used would be overwhelming with vast unchanging data between timesteps

New Approach

- Idea: Only compute on part of domain that will be affected "soon"
  - Split simulation run into a series of computational volumes and write data progressively

- Reduce computation footprint by 99%+
- Reduce computation time by eliminating IO and reducing communication costs

- Challenges:
  - How to we make the compute approach work?
  - How do we store data to enable analysis later?

Can’t Current Models Work?

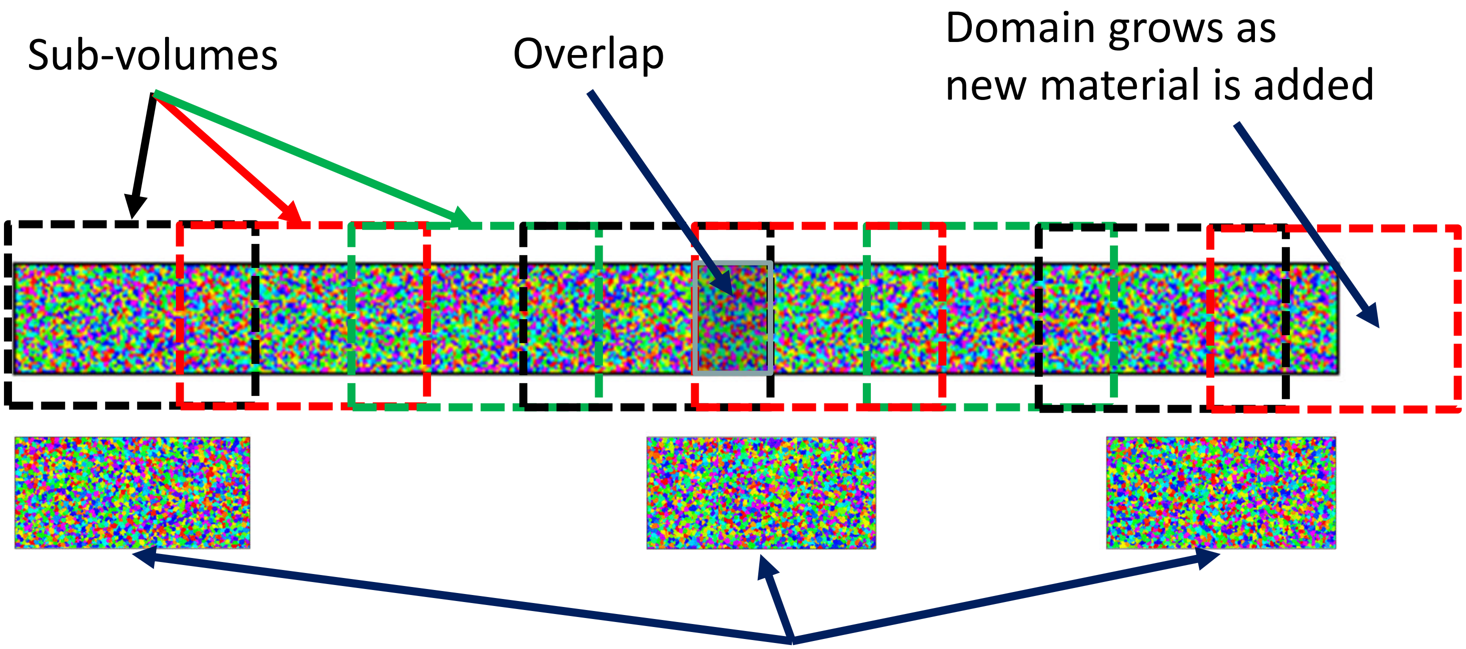
- Existing compute models can—with a little help
- Existing IO models cannot since they depend on whole domains being written
  - HDF5, NetCDF, PnetCDF, ADIOS, and others all have this “feature”

Addressing Computation

- Rethink as a series of small problems
- Build “glue” for overlapping areas and to initialize unvisited areas
- Make traversal algorithm and scripting that also handles resilience

Illustrative Example (SPPARKS)

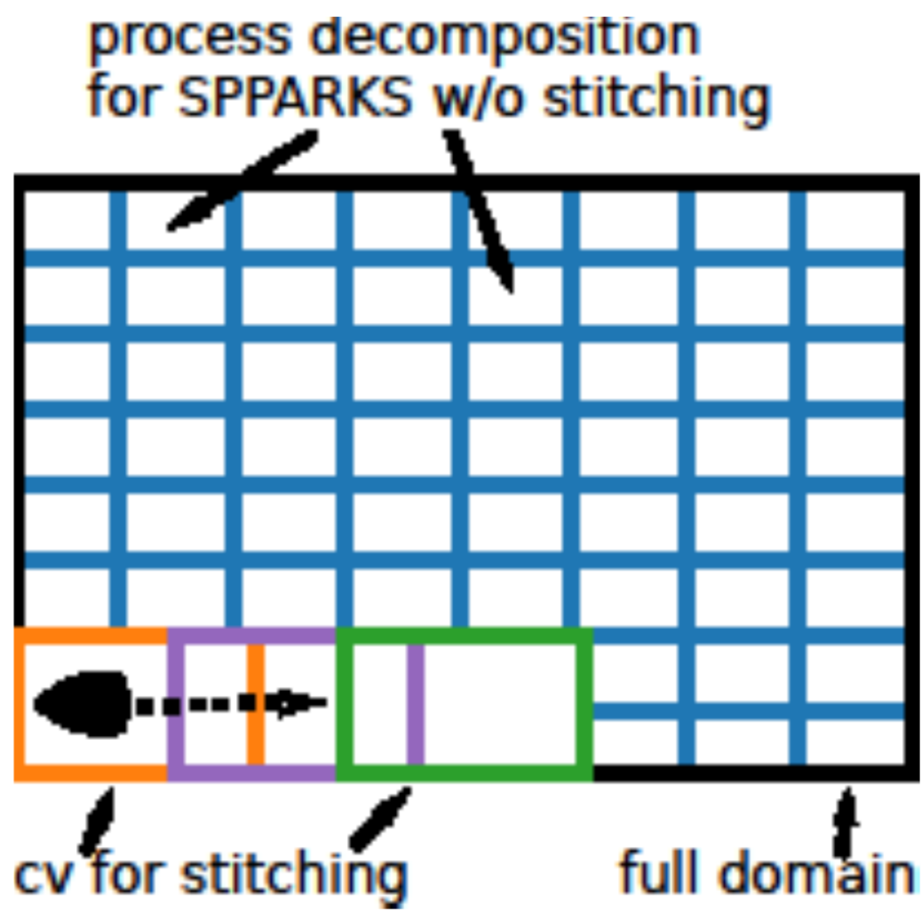
Grain growth across a large domain is simulated using a series of smaller overlapping sub-volumes.



Post-process, visualize and analyze on arbitrary sub-volumes and arbitrary times

Illustrative Example

- Orange/Purple/Green vs. Blue



Addressing IO

- Rethink how IO should work:
  - Lazy
    - Only track what has been seen so far (i.e., we don’t care about the size of the simulation domain)
  - Minimal
    - Only write was has changed since last output
  - Eventually Consistent
    - Rely on the output to eventually “make sense”
  - Construct arbitrary requested domain on demand

- Reading specifies an arbitrary region and a time; Stitch-IO assembles (‘stitches’) the region state together from various pieces using the newest for every point

Stitch-IO – A New IO Approach

- Stitch-IO changes the rules
  - Does not require global domain setup
  - Offers support for combining data from multiple outputs into a single blob
    - Selecting a region that goes behind the active region will get previously completed data even though it has not been written during the latest output
  - Uses floating point numbers (with absolute and relative tolerances) for identifying a time epoch
  - Uses a standard format for easy, direct access from other tools (SQLite)
  - Supports writing at both the current, new time and older (existing or not) times without ill effect.
  - Simultaneous writing and reading is assumed and fully supported.

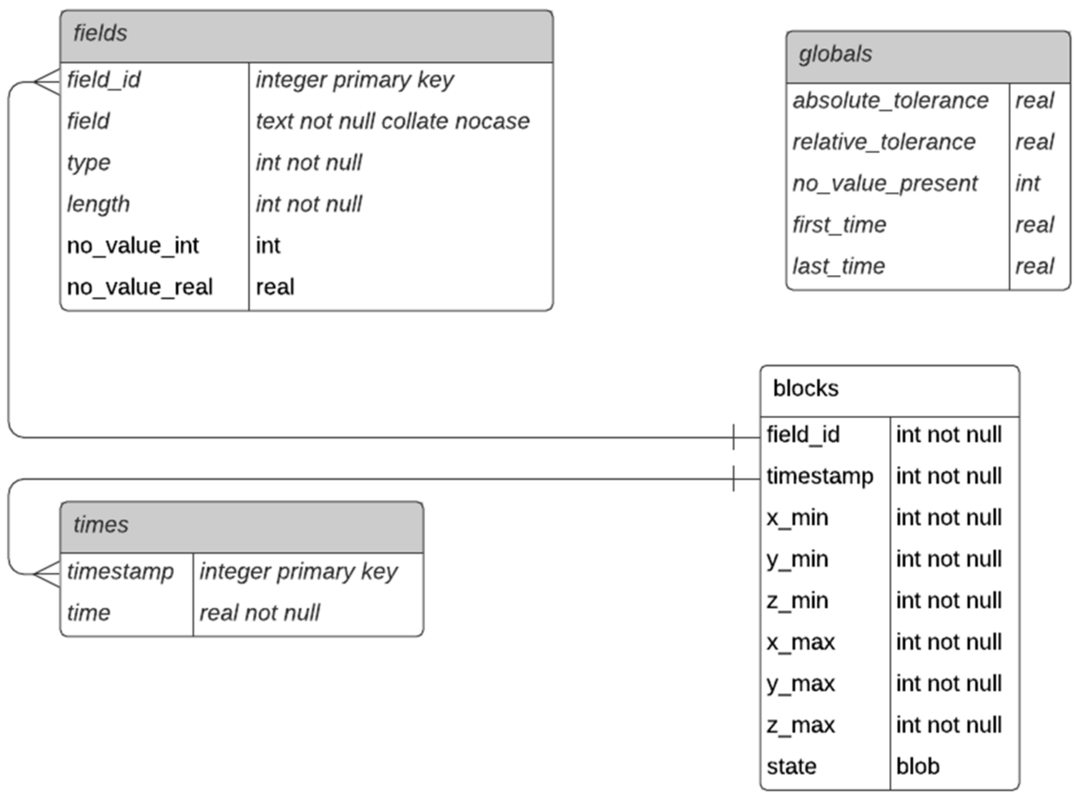
Stitch-IO – A New IO Approach

- Written in C with a full capability Python module interface
- Has both a serial (single process) and parallel (MPI-based) interface for C and Python.
  - The only difference is passing a communicator to the `open' command. Calls are invisibly parallel and efficient if can be optimized.

- Play along at home:
- https://github.com/gflofst/Stitch-IO
- You can build on a basic Linux setup for most clusters trivially with late model Python3, numpy, and a C compiler (and MPI for parallel builds).

Stitch-IO Schema

- SQLite storage format opens easy direct data access
  - Extend functionality without changing library using Python native API



Example Application Use

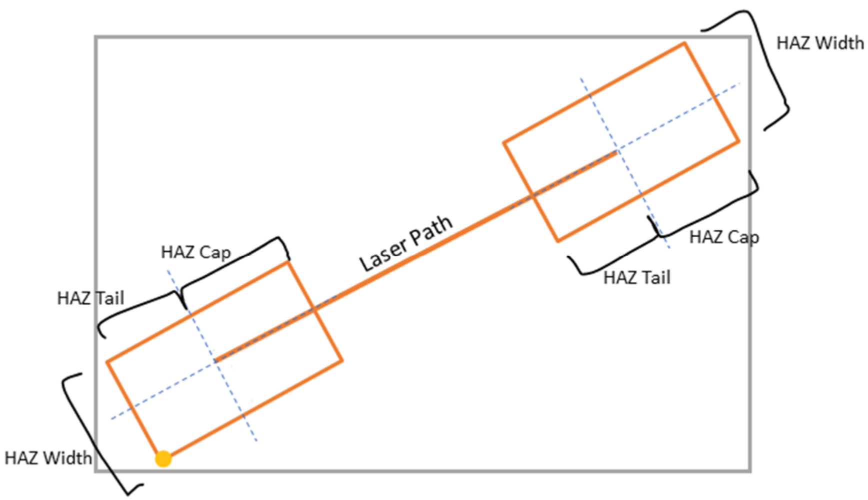
Digital twin creation for Additive Manufacturing

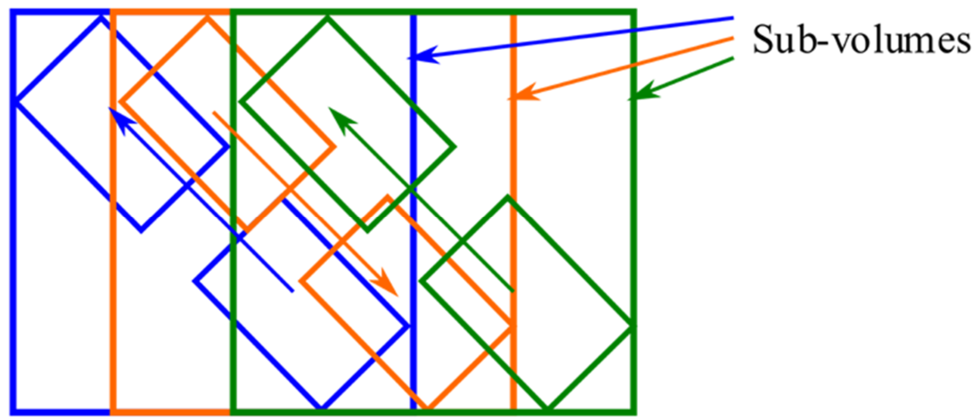
- Use a .STL file as source
- Use Slic3r to generate g-code to drive simulation
  - Fixup the g-code into an execution script
  - Run the simulation
- Use AM machine slicer to generate machine specific g-code
  - Run the AM machine with the g-code

- Use CT Scanner to get internal images of physical part to compare against simulation
- Use simulation results to test other physics

A series of computation volumes

- Limit simulation domain in memory to just a small part that we will compute over.

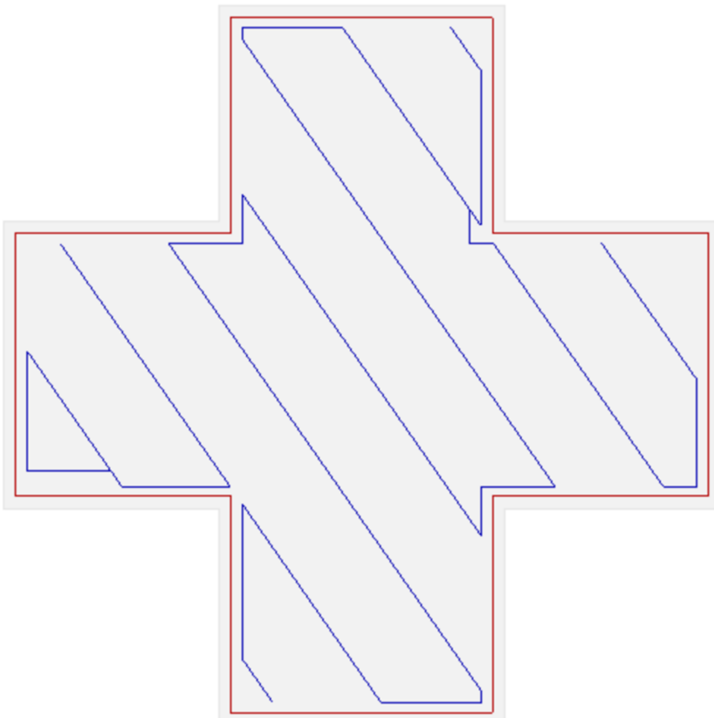


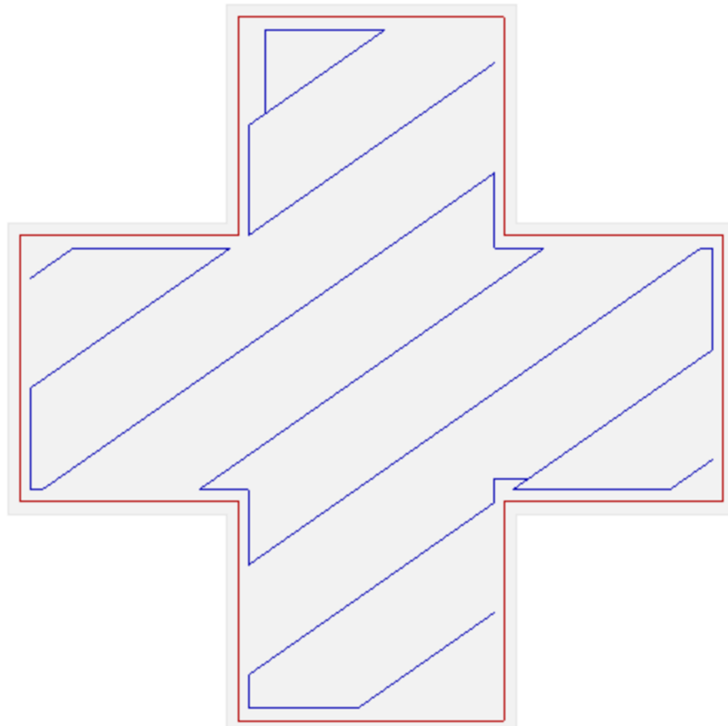


- HAZ = Heat Affected Zone
- Laser Path is the AM machine laser path

Slicing

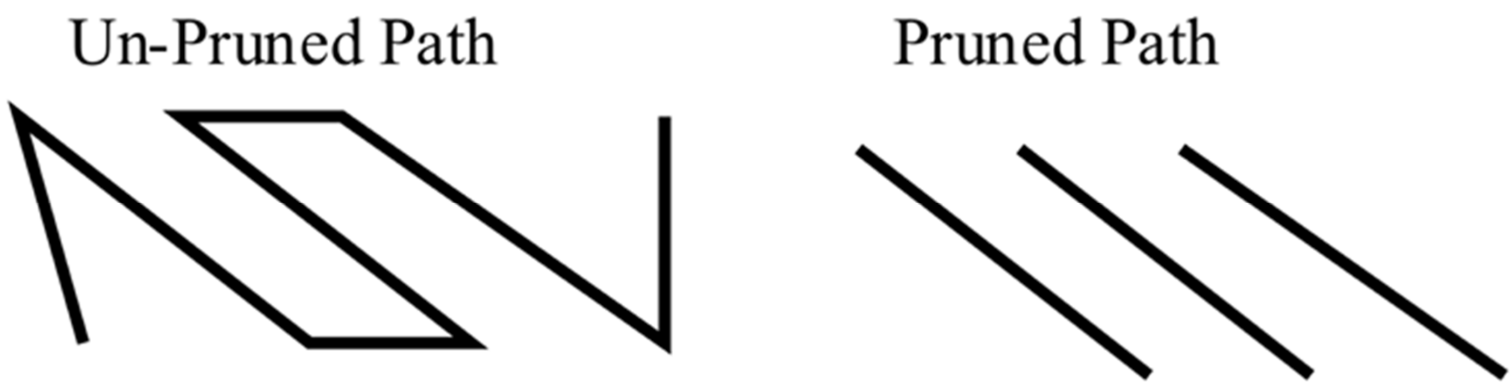
- Slic3r used to generate paths
- Alternating layers. Red outline is perimeter, blue is laser path with a fill angle of 35 degrees.





Path Pruning

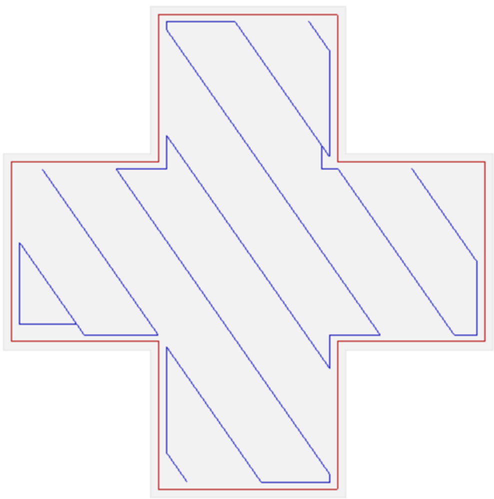
- Get rid of places where the laser is turned off just to move to a new starting position

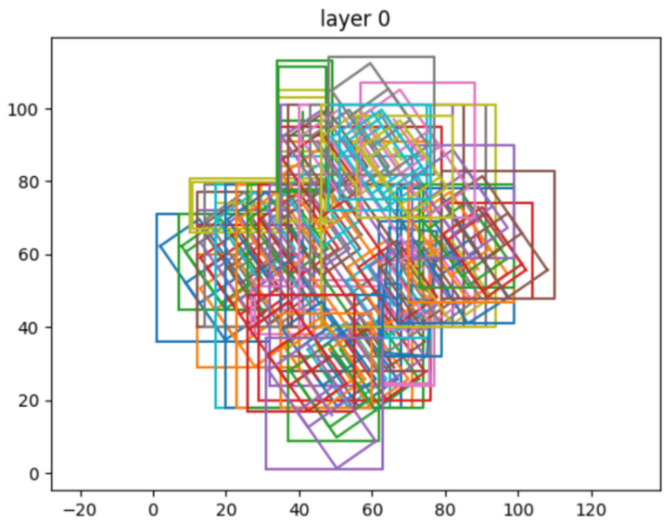


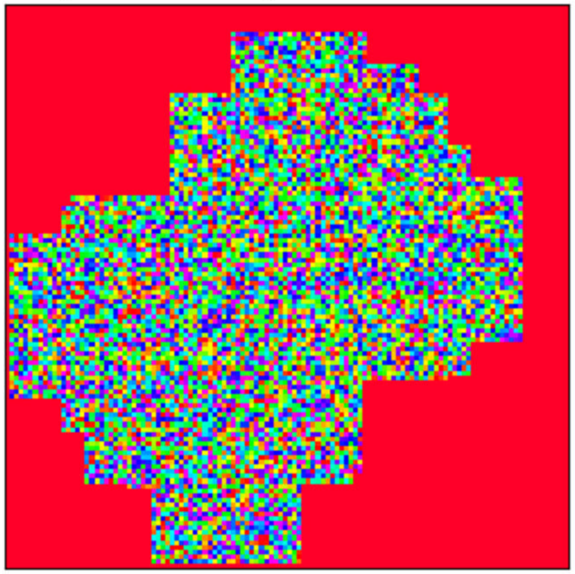
How Stitch-IO Reduces Computation

- Computational volumes limit total domain

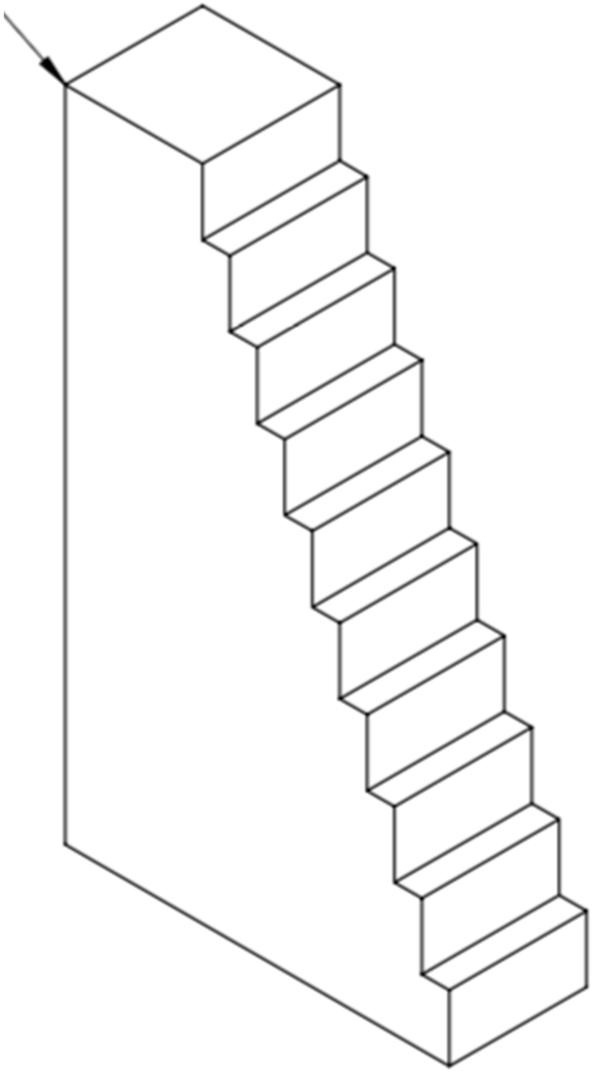
- Left: Geometry and Slic3r z-cut path lines
- Middle: Computational volumes bounding boxes
- Right: Simulated layer (red border is no data present)

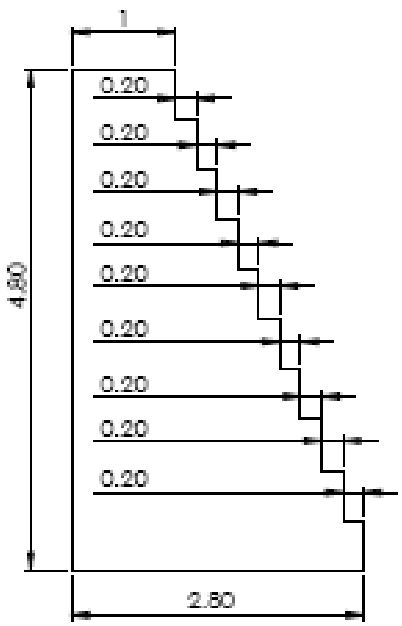


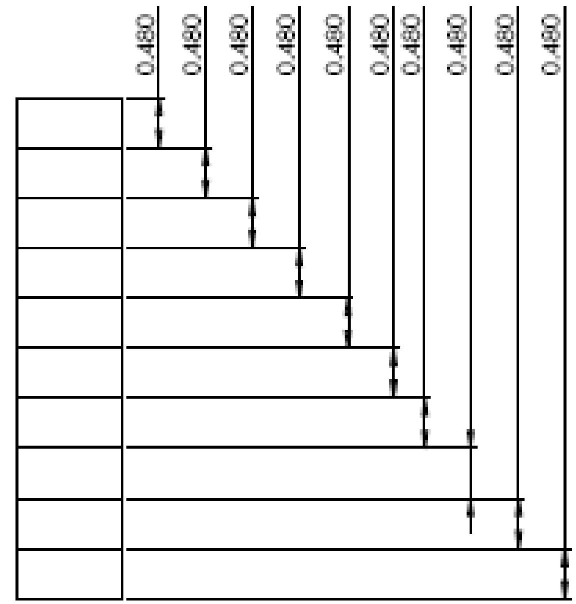




Staircase example

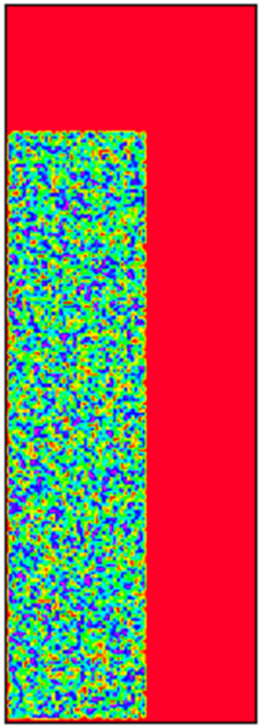


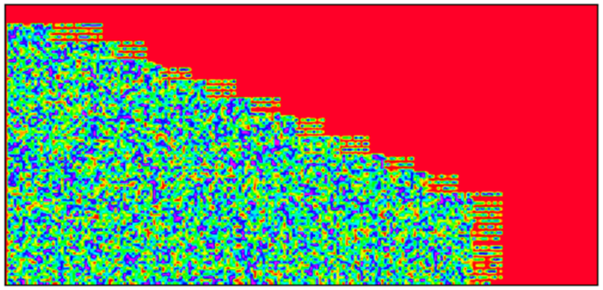




Forming the Staircase

- 4 different paths
  - (UDLR, RLUD, DURL, LRDU)
- Each colored dot is metallic powder. The larger colored areas are metal grains formed by the laser melt.
- Top graphic shows the laser path. It starts in each corner iteratively and follows a different path.
- Bottom graphic shows errors in the edges of a staircase causing weaknesses and the large color bars indicate large grain growth.





Stitch-CAD for Digital Twins

Simulate rather than build and destructively test

- From CAD file that can be used by the AM machine
    1. Use Slic3r to generate path (G-code)
    2. Split G-code into layers
    3. Prune each layer
    4. For each laser path
        1. Calculate a computational volume
        2. Initialize the computational volume
        3. Run simulation
    5. End for
- See microstructure generation identifying parameter issues that will generate parts with inadequacies.
- Iterate in simulation to find the best parameters

API Basics

- `from stitch.libstitch import libstitch`
- (rc, file_id) = libstitch.**open** (“filename”)
- rc = libstitch.**close** (file)
- (rc, new_time) = libstitch.**write_block** (file, field_id, timestamp, block, state)
  - Timestamp is a real
  - Block is 6 ints representing the (x,y,z)-(x,y,z) min max pairs
  - State is a linearization of the block (typical memory layout)
  - new_time is a flag to indicate if this time has been used before or not as a sanity check
- (rc, state, new_time) = libstitch.**read_block** (file, field_id, timestamp, block)
  - Same as write

API Basics

- (rc, field_id) = libstitch.**create_field** (file, ‘field_name’, 1, 1, -1)
  - Type 1 is 32-bit int (C has an enum)
  - Length 1 is single element (C has an enum)
  - Default value is -1
- (rc, field_id) = libstitch.**query_field** (file, ‘field_name’)
- (rc, times) = libstitch.**get_times** (file)
  - list of times used
- (rc, field_ids, labels, t, lengths, no_value_presents) = libstitch.**get_fields** (file)
  - The field attributes

API Basics

- rc = libstitch.**set_parameters** (file, abs_tol, rel_tol, nvp)
  - Optional as these are defaulted
- (rc, abs_tol, rel_tol, nvp, first_time, last_time) = libstitch.**get_parameters** (file)
  - Mainly to get the min and max time in the file
- rc = libstitch.**set_field_no_value_present** (file, field_id, nvp_val) – value to initialize areas with no value

Benefits and Challenges

- Move from 1000s process to 10s (cluster to a laptop)
- Radical data size reduction (about 1% or less losslessly)
- Wall clock time the same or smaller (less output time)
  - Spray model went from > 24 hours to 6-8 hours; latest changes may reduce that to 4 hours

- New simulations are pushing things hard (400+ million blocks)

- Open Source (LGPL) at https://github.com/gflofst/Stitch-IO
- Paper at IPDPS 2020

- Email: gflofst@sandia.gov

Other ideas for using Stitch-IO

- Any image data-based application

- CT scans
  - Explore regions across images

Future Work

- Continue to work on scalability with SPPARKS
- Additional application examples and other domains
- Working with any of you on new problems
- gflofst@sandia.gov