BISON Workshop

Implicit, parallel, fully-coupled nuclear fuel performance analysis

Fuels Modeling and Simulation Department
Idaho National Laboratory

Table of Contents

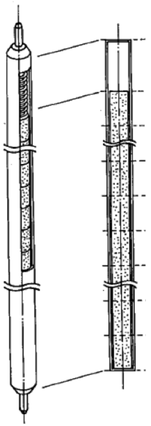
BISON Overview

BISON Team Members

- Rich Williamson
  - richard.williamson@inl.gov
- Steve Novascone
  - stephen.novascone@inl.gov
- Jason Hales
  - jason.hales@inl.gov
- Ben Spencer
  - benjamin.spencer@inl.gov
- Giovanni Pastore
  - giovanni.pastore@inl.gov
- Danielle Perez
  - danielle.perez@inl.gov
- Russell Gardner
  - russell.gardner@inl.gov
- Kyle Gamble
  - kyle.gamble@inl.gov
- Mudasar Zahoor
  - mudasar.zahoor@inl.gov
- Al Casagranda
  - albert.casagranda@inl.gov
- Wenfeng Liu
  - wenfeng.liu@anatech.com
- Ahn Mai
  - anh.mai@anatech.com
- Jack Galloway
  - jackg@lanl.gov
- Christopher Matthews
  - cmatthews@lanl.gov
- Cetin Unal
  - cu@lanl.gov

Fuel Behavior: Introduction

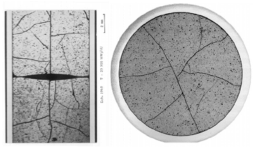
At beginning of life, a fuel element is quite simple...



Nakajima et al., Nuc. Eng. Des., **148**, 41 (1994)

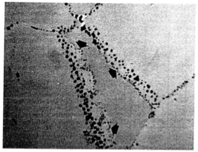
$\Longrightarrow$

but irradiation brings about substantial complexity...



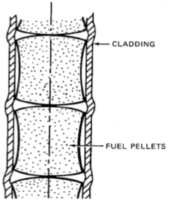
Michel et al., Eng. Frac. Mech., **75**, 3581 (2008)

Fuel Fracture



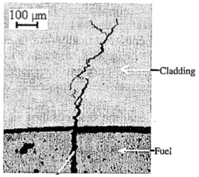
Olander, p. 323 (1978)

Fission Gas



Olander, p. 584 (1978)

Multidimensional Contact and Deformation

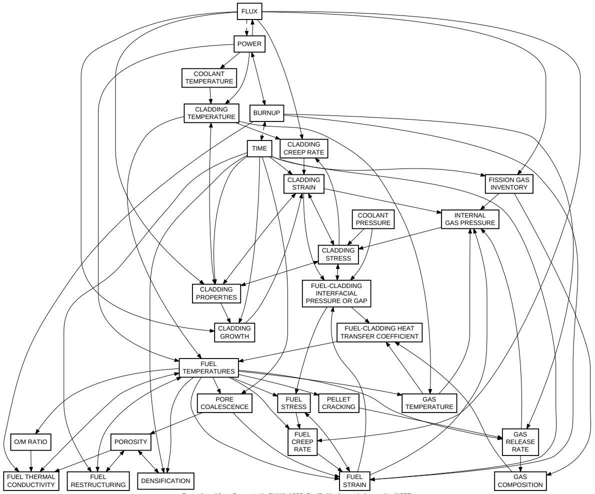


Bentejac et al., PCI Seminar (2004)

Stress Corrosion Cracking Cladding Failure

Fuel Behavior Modeling: Coupled Multiphysics

- Multiphysics
  - Fully-coupled nonlinear thermomechanics
  - Multiple species diffusion
  - Neutronics
  - Thermalhydraulics
  - Chemistry
- Multi-space scale
  - Important physics at atomistic and microstructural level
  - Practical engineering simulations require continuum level
- Multi-time scale
  - Steady operation ($\Delta t > 1$ week)
  - Power ramps/accidents ($\Delta t < 0.1$ s)



Reproduced from Beyer et al., BNWL-1898, Pacific Northwest Laboratories (1975)

BISON – Nuclear Fuel Performance Analysis

- BISON is a nuclear fuel performance analysis tool. It is used primarily for analysis of UO_2 fuel but has also been used to model TRISO fuel and both rod and plate metal fuel. BISON is built on top of MOOSE.

- BISON is implicit
  - *Large time steps*

- BISON runs in parallel
  - *Runs naturally on one or many processors*

- BISON is fully-coupled
  - *No operator split or staggered scheme necessary*
  - *All unknowns solved for simultaneously*

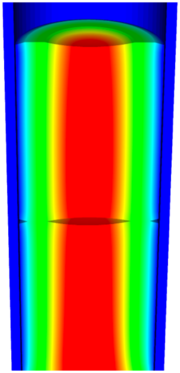
- BISON is under development – there is still much to do
  - *Fission gas release model continues to improve*
  - *Contact can be a challenge; friction needs improvement*
  - *Automatic time stepping needs improvement*
  - *Limited material models (UO_2/MOX and Zr4/HT9)*
  - *Documentation and validation is evolving*

BISON's Relationship to MOOSE

- Code too specific for MOOSE but useful for multiple applications is collected in libraries.
- BISON depends on:
- MOOSE Modules (solid mechanics, fluid dynamics, etc.) depends on:
- MOOSE (multiphysics application framework) depends on:
- libMesh (numerical PDE solution framework out of UT-Austin) depends on:
- PETSc, Exodus II, MPI, etc.

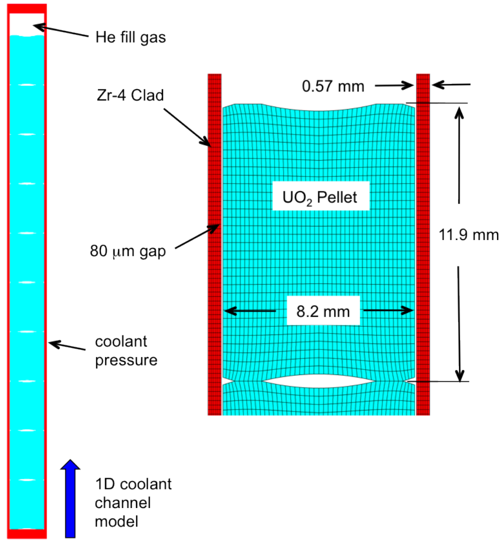
BISON LWR Capabilities

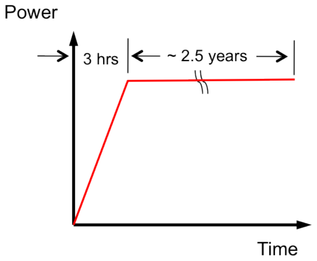
- General capabilities
  - 3D, 2D-RZ, 1D fully coupled thermomechanics
  - Large deformations
  - Parallel
  - Meso-scale informed
- Oxide Fuel Behavior
  - Temperature/burnup/porosity dependent material properties
  - Volumetric heat generation
  - Thermal, fission product swelling, and densification strains
  - Thermal and irradiation creep
  - Fuel fracture via relocation and smeared cracking
  - Fission gas release (2 stage)
    - transient release
    - grain growth/sweeping
    - athermal release


Temperature

- Gap/Plenum Behavior
  - Gap heat transfer with $k_g = f(T, n)$
  - Mechanical contact
  - Plenum pressure as a function of :
    - evolving gas volume (from mechanics)
    - gas mixture (FGR)
    - gas temperature approximation
- Cladding Behavior
  - Thermal and irradiation creep
  - Thermal expansion
  - Irradiation growth
  - Plasticity
  - Hydride Damage
- Coolant Channel
  - Closed channel thermal hydraulics with heat transfer coefficients

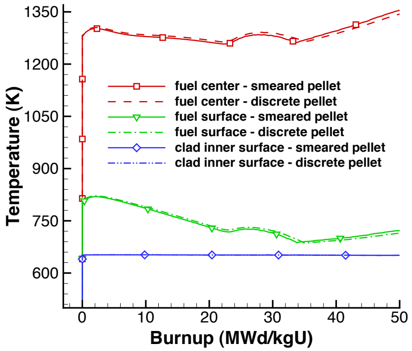
BISON Example – Axisymmetric LWR Fuel Rodlet

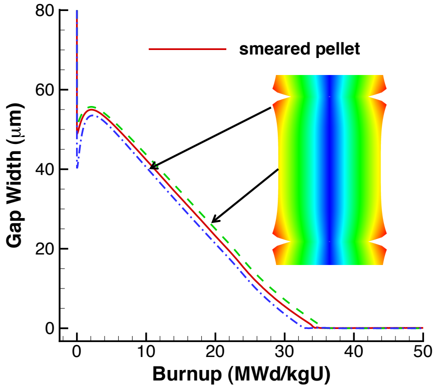




Linear average power	250 W/cm
Fast neutron flux	7.5×10^{17} n/m^{2}s
Coolant pressure	15.5 MPa
Coolant inlet temperature	580 K
Coolant inlet mass flux	3800 kg/m^2-s
Rod fill gas	helium
Fill gas initial pressure	2.0 MPa
Initial fuel density	95% theoretical
Fuel densification	1% theoretical
Burnup at full densification	5 MWd/kgU

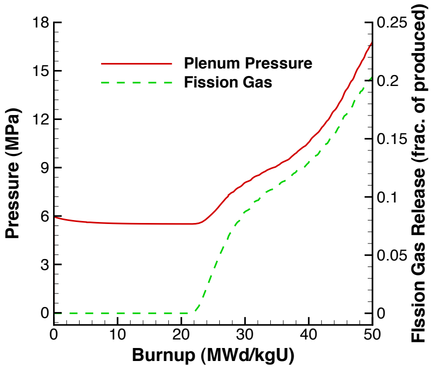
BISON Results – Axisymmetric LWR Fuel Rodlet

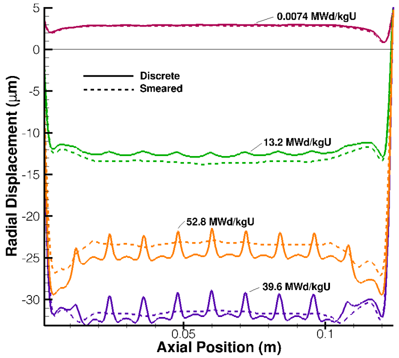




- Thermal expansion, fuel densification, clad creep-down, fission gas release, contact, and burnup dependent fuel thermal conductivity all affect fuel temperatures
- Hourglass shape of pellets is evident in gap closure histories

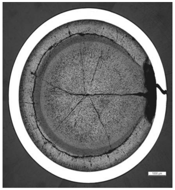
BISON Results – Axisymmetric LWR Fuel Rodlet

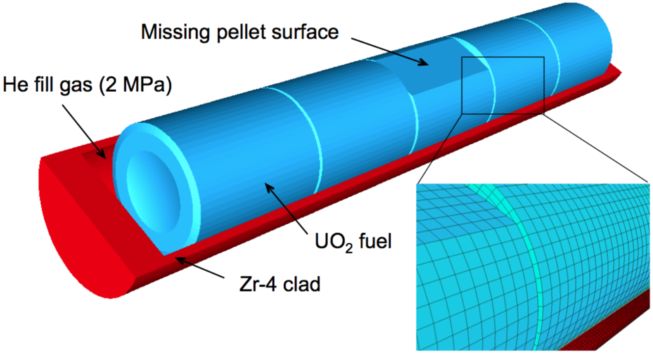




- Fission gas release begins at a burnup of 22 MWd/kgU and results in a strong increase in rod pressure
- Hourglass shape of pellets creates ridges in clad during PCMI

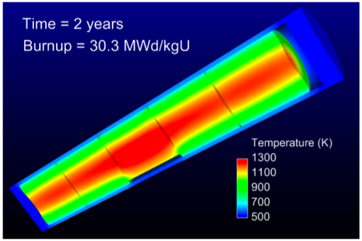
BISON Example – Missing Pellet Surface





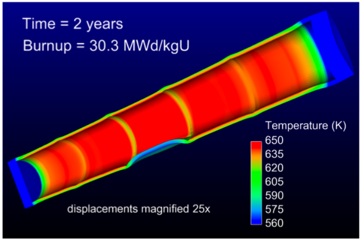
- High resolution 3D calculation (25,000 elements, 1.1×10^6 dof) run on 120 processors
- Simulation from fresh fuel state with a typical power history, followed by a late-life power ramp

BISON Results – Missing Pellet Surface

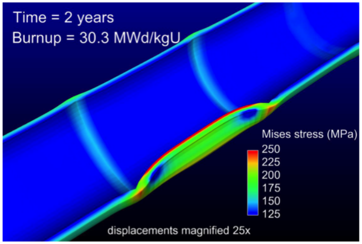


Fuel Temperature

- Missing pellet surface has a very significant effect on temperature and stress state in rod
- Model can be used to examine source of rod failures



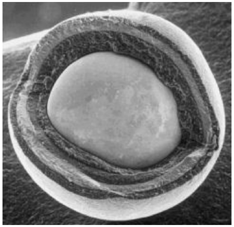
Clad Temperature

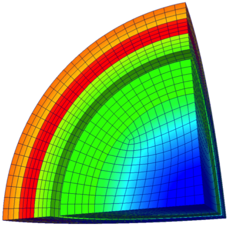


Clad Stress

BISON Coated-Particle Fuel Capabilities

- General capabilities
  - 3D, 2D-RZ, 1D fully coupled thermomechanics with species diffusion
  - Large deformation
  - Elasticity with thermal expansion
  - Steady and transient
  - Massively parallel
- Fuel Kernel
  - Temperature, burnup, porosity dependent conductivity
  - Solid and gaseous fission product swelling
  - Densification
  - Thermal/irradiation creep
  - Fission gas release
  - CO production
  - Radioactive decay



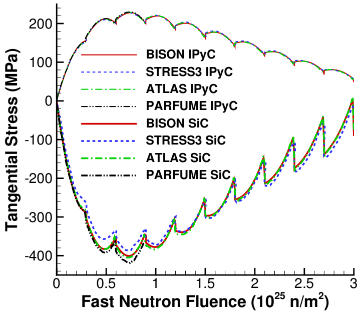


Tangential Stress

- Gap Behavior
  - Gap heat transfer with $k_g = f(T, n)$
  - Gap mass transfer
  - Mechanical contact
  - Plenum pressure as a function of :
    - evolving gas volume (from mechanics)
    - gas mixture (FGR and CO)
    - gas temperature approximation
- Silicon Carbide
  - Irradiation creep
- Pyrolytic Carbon
  - Anisotropic irradiation-induced strain
  - Irradiation creep

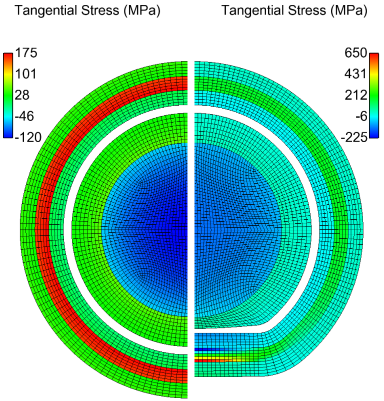
BISON Results – TRISO Particle

- Validated against PARFUME, ATLAS, STRESS3
- Code comparisons are excellent
- Run times of 1 s are typical



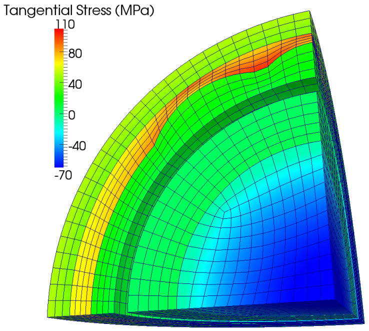
PBR cyclic particle temperature

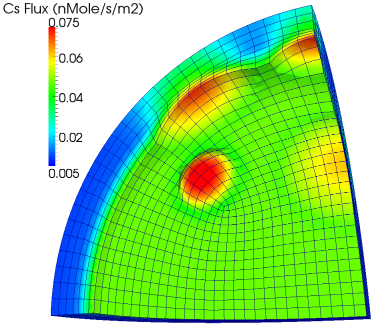
- Apherical particles are common
- Raises peak tensile stress by 4x
- Runs in a few minutes (8 procs)



BISON Results – 3D Simulation of Thinned SiC Layer

- Localized SiC thinning due to soot inclusions or fission product interaction
- 3D capability demonstrated on eighth-particle with random thinning





- Significantly higher tensile stress and cesium release; impossible to predict with state-of-the-art 1D or 2D analyses
- Typical run times of a few hours on 8 procs

Getting Started

www.inl.gov

Idaho National Laboratory

Getting started with BISON: Git

These instructions are meant to help users of both local machines and the INL High Performance Computing (HPC) get started with BISON. The installation and build processes are the same for both with the exception of the MOOSE package, which is prebuilt on HPC.
BISON and MOOSE use git for code management and distribution. For that reason you will need to learn a little bit about git; even if you are not planning on contributing to the code. You will also need access to the BISON repository and a license agreement. Instructions for this are below.

- Contact a member of the BISON team to get HPC access and a licensing agreement for your institution. This process may take a few weeks. BISON is housed at `https://hpcgitlab.inl.gov` which lives on the INL HPC, therefore HPC access is required. Continue on to the next step once HPC access is obtained.
- Install the MOOSE Package (Only needed if you ARE running on a local machine)
  - Goto `www.mooseframework.org`, click 'Getting Started' and follow all the instructions.
  - Make an account on `github.com`. It is free and required to contribute to MOOSE.

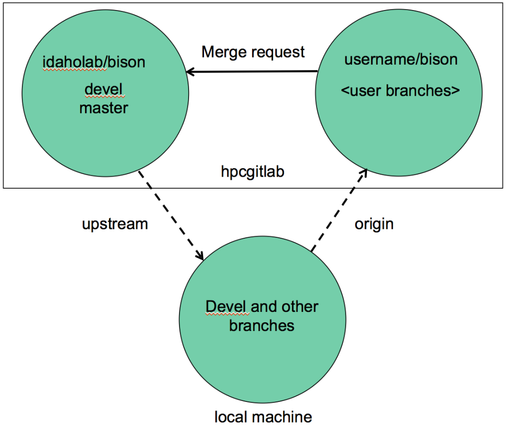
Getting Started with BISON: Git

Adding your SSH keys

- If you are working from outside the INL you will need to create a tunnel to INL using your HPC account and the instructions that should have been included. See the 'External Users' section for more help.
- If you can log in to `https://hpcgitlab.inl.gov` and then access the idaholab/bison project you may continue. Contact the BISON department for repository access if you are unable to access the project.
- Open the terminal on the machine that you intend to use and enter the following commands:
  - `> ssh-keygen -t rsa -C 'your_email'` Press 'Enter' when you are asked for a passphrase. You may enter a passphrase but you will need to enter it whenever you connect to the repository.
  - `> cat ~/.ssh/id_pub` and copy what is displayed
  - Log on to `https://hpcgitlab.inl.gov` using your HPC credentials and then paste the key to 'SSH Keys' under 'Profile Settings' on hpcgitlab.
  - Paste this key to your `github.com` account as well, in the same manner.
    - These steps are necessary for repository authentication.

Getting Started with BISON: Git

This is a pictorial diagram of the git environment. In this diagram ‘local machine’ represents either a physical, local machine or the HPC. The two upper circles represent space on `https://hpcgitlab.inl.gov`. The dotted arrows ‘upstream’ and ‘origin’ represent git remotes which are just aliases to paths.



Getting Started with BISON: Git

Fork BISON

- On hpcgitlab, search for ‘idaholab/bison’ and then click ‘Fork’. This makes a personal version of the BISON project denoted ‘username/bison’.
  - NOTE: Do not delete your fork. Recovery is a troublesome process.
- Click on ‘Members’ and add ‘moosetest’ as a ‘Reporter’.
  - This change is required for the testing system to work, which is necessary for contributing to the code. If you will not be contributing to the code, it is not required.
- If you wish to grant access to your fork to other BISON team members (helpful if you want them to review changes in your branch before submitting a merge request), add specific BISON team members as ‘Reporter’. This is optional and can be skipped if you will be running BISON but not changing the code.

Getting Started with BISON: Git

Load Modules

- HPC: `> module load use.moose moose-dev-gcc` will load the most up to date Moose modules. You may consider adding this command to your logon script.
- Local machine: As long as you installed the Moose package with the default settings the needed modules were added to your bash profile. You can double-check they are install by opening a terminal and typing `> module list`, this should list a bunch of MOOSE related modules.

Cloning BISON

- Create a directory on the computer that you are using. We suggest 'projects' for convention, but you can name it anything.
  - `> mkdir ~/projects`
- To clone BISON, enter the directory that was just made, `> cd ~/projects` and enter the following git commands:
  - `> git clone git@hpcgitlab.inl.gov:<username>/bison.git`
    - This copies your fork of BISON to your projects directory.
  - Enter the bison directory that was just created.
  - `> git submodule update --init`
    - This initializes any submodules that BISON depends on: in this case MOOSE.

Getting Started with BISON: Git

Some Git Cleanup

- In the bison directory enter the following:
  - `> git config user.name ‘your name’`
  - `> git config user.email ‘your email’`
    - These commands add your user information to your repository.
  - `> git remote add upstream git@hpcgitlab.inl.gov:idaholab/bison.git`
    - This adds a remote (path) to idaholab/bison. At this point if you type `> git remote show` you should see origin and upstream. Origin points to your fork of BISON and upstream points to idaholab/bison.
      The command `> git remote show <remote_name>` will display the path.

Getting Started with BISON: Building BISON

Build libMesh

- Open a terminal and go to `> ~/projects/bison/moose/scripts` and then run the `> update_and_rebuild_libmesh.sh` script. This script will download and install the latest version of libMesh.
- This is also the method used to update libMesh when updates are available. Notices about updates are sent to the moose-users and moose-announce mailing lists.

Build BISON

- Go to `> ~/projects/bison` and type `make -j#`, where # is the number of processors that you would like to use.
  - NOTE: Do not use more than 6 processors when building on the HPC. The login nodes have shared resources for all users and you will attract unwanted attention by using them all.
- After the build is completed run `> run_tests -j#` to run all of the BISON regression tests to make sure you have a clean build.

Getting Started with BISON: Building BISON

Running an Example Problem

- `> cd ~/bison/examples/2D-RZ_rodlet_10pellets`
- `> ../../bison-opt -i input.i` For serial processing
- `> mpiexec -n # ../../bison-opt -i input.i` For multiprocessor where # is the number of processors.

Updating BISON (In the bison directory)

- `> git pull --rebase upstream devel`
  - This command pulls the updates from the devel branch of idaholab/bison via the upstream remote and applies them to your current local branch.
- `> git submodule update`
  - This command applies the update to any submodules. Once again this is MOOSE.
- `> make cleanall`
  - This command deletes files that have dependencies. If your build has errors make sure you run this.
- `> make -j#`
  - This command build the BISON executable.

Contributing to BISON

Make an Issue

- Make an issue on hpcgitlab with a problem or feature that you are working on.
  - If you are making an issue about a bug or new feature that you would like please write details. For example: I received this 'error' using BISON 'version' on this 'machine' and this is what I have already 'tried'.

Working in Branches

- Update your version of BISON.
- Create a new branch in your personal repository to work in. We suggest naming it with the issue number that you just made as in the following: > `git checkout -b <feature_name>`
- Do work in the branch that was created.
- Add the work to be committed. There are a couple ways to add:
  - > `git add <filename>`
  - > `git add -i` (this opens the interactive window and is useful for picking multiple files to add).
- Enter > `git status` to make sure the files that you wanted are added and staged.

Contributing to BISON

Working in Branches cont.

- Commit the work using `> git commit`, this will open a vi window where you will make a commit message that references the issue number. You can also use `> git commit -m ‘commit_message’`.
  - NOTE: When referencing the issue number you need to have `#<your_number>` with no spaces. The testing system will fail you otherwise. Example: `#12 -> Good, # 12 -> Bad.`
- Push your commit to your bison fork with the following: `> git push origin <your_branch_name>`
- Create a merge request on hpcgitlab and verify that the changes that you are making are indeed intended.
- Stay involved.

Any and all information about git can be found here: `https://git-scm.com/doc` this guide was made to help you get started. It is NOT all encompassing.

Consider joining the user lists.

- Moose user lists at `mooseframework.org`
- Bison-users by emailing the BISON dept. Many questions can be or already have been answered here and you can interact with the MOOSE-BISON community.

External Users

- For external users there are a few additional steps to checking out the code. First request an HPC account. Once an HPC account has been generated a ssh tunnel will need to be set up to access GitLab. Add the following lines to your ~/.ssh/config file. Replace <USERNAME> with the username for your HPC account. NOTE: Type these commands, copy-paste from PDF can introduce metadata that fouls the process.

```
#Multiplex connections for less RSA typing
Host *
         ControlMaster auto
         ControlPath ~/.ssh/master-%r@%h:%p
# General Purpose HPC Machines
Host hpcsc flogin1 flogin2 falcon1 falcon2
         User <USERNAME>
         ProxyCommand ssh <USERNAME>@hpclogin.inl.gov netcat %h %p
#GitLab
Host hpcgitlab.inl.gov
         User <USERNAME>
         ProxyCommand nc -x localhost:5555 %h %p
#Forward license servers, webpages, and source control
Host hpclogin hpclogin.inl.gov
         User <USERNAME>
         HostName hpclogin.inl.gov
         LocalForward 8080 hpcweb:80
         LocalForward 4443 hpcsc:443
```

External Users

- Next create a tunnel into the HPC environment and leave it running while you require access to GitLab. If you close this window, you close the connection:

```
ssh -D 5555 username@hpclogin.inl.gov
```

- Then you have to adjust your socks proxy settings for your web browser to reflect the following settings localhost:5555 where localhost is the server name and 5555 is the port number.
- If you do not know how to do that, look up **Change socks proxy settings for <insert the name of your web browser here>** on google.com or some other search engine. Once that is complete you can login to the GitLab website.
- The rest of the steps for checking out the code are the same as for internal users given in the previous slides.

The BISON Wiki Page

In progress

MOOSE Modules and Thermomechanics Basics

www.inl.gov

Idaho National Laboratory

MOOSE as a Partial Differential Equation Solver

- We are interested in solving a set of partial differential equations (PDEs) that represent physical processes, such as heat transfer and solid mecahnics.
- MOOSE is a general solver that uses the finite element method (FEM) to solve arbitrary sets of PDEs for specific applications.
- FEM converts complex PDEs into a set of coupled algebraic equations which can be readily solved on a computer.
- FEM is applicable to a wide range of PDEs and can represent problems with arbitrary geometry.

FEM Vocabulary

The following list contains terms commonly used when discussing the finite element approach. These definitions are NOT COMPREHENSIVE. This list is just to get the conversation started.

- Domain - The space or geometry of your problem.
- Element - To obtain the approximate solution, the domain must be subdivided (discretized) into simpler smaller regions. These are called elements.
- Node - The points at which the elements are connected. We typically compute the value of primary solution variables (temperature, displacement) at nodes. Also where Dirichlet boundary conditions are applied.
- Boundary Condition - A constraint, or ‘load’ applied to the domain.
- Quadrature Point - One of the steps to finding the approximate solution to the PDE is integration. Quadrature points are where this integration happens. They are located within the elements.
- Test or Shape Function - Functions that help form the approximate solution to the PDE.

MOOSE Modules

- MOOSE Modules holds a collection of general physics capabilities.
- The purpose is to encapsulate common kernels, BCs and materials to keep them from being replicated in multiple codes.
- Examples include heat conduction, solid mechanics, and Navier-Stokes.
- No export controlled physics (e.g., neutronics) should be added to MOOSE Modules.
- Applications based on MOOSE can link in their needed modules.
- For example, BISON uses the Heat Conduction, Solid Mechanics, and Contact modules.

Modules: Heat Conduction

- MOOSE Modules' heat conduction routines are built to help solve

$$\rho C_p \frac{\partial T}{\partial t} - \nabla \cdot k \nabla T - q = 0$$

  where ρ is the mass density, C_p is the specific heat, T is the temperature, k is the thermal conductivity, and q is the volumetric heat generation rate.

- MOOSE Modules provides spherically symmetric 1D, axisymmetric 2D, and 3D formulations. Either first or second order elements may be used (QUAD4 or QUAD8 for RZ, HEX8 or HEX20 for 3D).

Modules: Heat Conduction

$$\rho C_p \frac{\partial T}{\partial t} - \nabla \cdot k \nabla T - q = 0$$

$$T|_{\partial\Omega_1} = g_1$$

$$\nabla T \cdot \hat{\boldsymbol{n}}|_{\partial\Omega_2} = g_2$$

- Multiply by test function, integrate

$$\left(\psi_i, \rho C_p \frac{\partial T}{\partial t}\right) - (\psi_i, \nabla \cdot k \nabla T) - (\psi_i, q) = 0$$

- Integrate by parts

$$\left(\psi_i, \rho C_p \frac{\partial T}{\partial t}\right) + (\nabla \psi_i, k \nabla T) - (\psi_i, q) - \langle \psi_i, g_2 \rangle = 0$$

- Jacobian

$$\left(\psi_i, \rho C_p \frac{\partial}{\partial T}\left(\frac{\partial T}{\partial t}\right) \phi_j\right) + (\nabla \psi_i, \nabla k \phi_j)$$

The Input File

- To solve these PDEs, we need to create an input file that contains all the necessary information.
- By default MOOSE uses a hierarchical, block-structured input file.
- Within each block, any number of name/value pairs can be listed.
- The syntax is completely customizable, or replaceable.
- To specify a simple problem, you will need to populate five or six top-level blocks.
- We will briefly cover a few of these blocks in this section and will illustrate the usage of the remaining blocks throughout this manual.

The Required Blocks of an Input File

- `[Mesh]` - the domain of the problem
- `[Variables]` - temperature and displacement
- `[Kernels]` - heat conduction, solid mechanics
- `[Materials]` - used by kernels, e.g. thermal conductivity
- `[BCs]` - specify Dirichlet or Neumann
- `[Executioner]` - steady state or transient
- `[Outputs]` - set options for how you want the output to look.

Example Input File

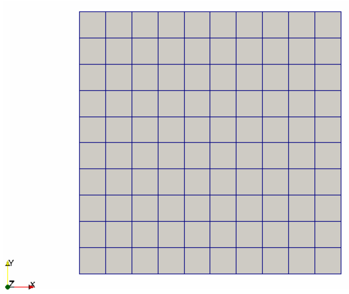
The following input file is an example of how to solve the heat conduction equation with a source term.

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 10
  ny = 10
[]

[Variables]
  [./temp]
  [../]
[]

[Kernels]
  [./heat_conduction]
    type = HeatConduction
    variable = temp
  [../]
  [./heat_source]
    type = HeatSource
    variable = temp
    value = 10000
  [../]
[]

[Materials]
  [./heat_conductor]
    type = HeatConductionMaterial
    thermal_conductivity = 1
    block = 0
  [../]
[]
```

```
[BCs]
  [./leftright]
    type = DirichletBC
    variable = temp
    boundary = 'left right'
    value = 200
  [../]
[]

[Executioner]
  type = Steady
  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]

[Outputs]
  exodus = true
  [./console]
    type = Console
    perf_log = true
  [../]
[]

[Postprocessors]
  [./peak_temp]
    type = NodalMaxValue
    variable = temp
  [../]
[]
```

Mesh Block

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 10
  ny = 10
[]
```



- The FEM mesh is defined in the `Mesh` block.
- A mesh can be read in from a file. There are many accepted formats (see the MOOSE manual). We typically use the exodus file format and create meshes with CUBIT.
- Simple meshes can also be generated within the input file. We'll use this approach for our first examples.
- The sides of a `GeneratedMesh` are named in a logical way (bottom, top, left, right, front, and back).

Variables Block

```
[Variables]
  [./temp]
  [../]
[]
```

- The primary or dependent variables in the PDEs (temperature, displacement) are defined in the `Variables` block.
- A user-selected unique name is assigned for each variable.

T

Kernels Block

```
[Kernels]
  [./heat_conduction]
    type = HeatConduction
    variable = temp
  [../]
  [./heat_source]
    type = HeatSource
    variable = temp
    value = 10000
  [../]
[]
```

- The kernels (individual terms in the PDEs being solved) are listed in the `Kernels` block.
- Each kernel is assigned a specific variable (in this case, temp or temperature).

$$-\nabla \cdot k \nabla T - q = 0$$

Materials Block

```
[Materials]
  [./heat_conductor]
    type = HeatConductionMaterial
    thermal_conductivity = 1
    block = 0
  [../]
```

$$k = 1$$

- Material properties are defined in the `Materials` block. Information from the materials block is used by some kernels.
- Here, thermal conductivity is defined to be used by the `HeatConduction` kernel.

Boundary Conditions (BCs) Block

```
[BCs]
  [./leftright]
    type = DirichletBC
    variable = temp
    boundary = 'left right'
    value = 200
  [../]
[]
```

Define temperature on boundary

- Boundary conditions are defined in the `BCs` block.
- Many types of boundary conditions can be applied.
- For this simple example, the temperature is set on the left and right sides of the domain.

Executioner Block

```
[Executioner]
  type = Steady
  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
[]
```

- The `Executioner` block defines how the problem is solved.
- You define here the type of solution (transient or steady state).
- The parameters `solve_type` and `petsc_options` will be discussed later.

Outputs Block

```
[Outputs]
  exodus = true
  [./console]
    type = Console
    perf_log = true
  [../]
[]
```

- The results you will output from your simulation are defined in the `Outputs` block.
- This includes defining the file type (exodus file here).
- Performance logs are also defined.

Postprocessors Block

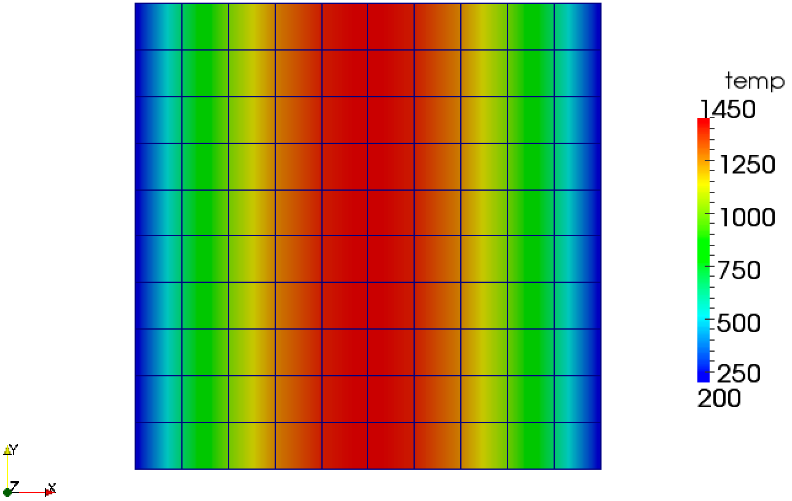
```
[Postprocessors]
  [./peak_temp]
    type = NodalMaxValue
    variable = temp
  [../]
[]
```

- Analysis results in the form of single scalar values are defined in the `Postprocessors` block.
- May operate on elements, nodes, or sides of the model.
- Examples include `NodalMaxValue`, `AverageElementSize`, and `SideAverageValue`.

Run Problem and Look at Results with Paraview

- The problems shown here can be run either with an application such as BISON that links in the heat_conduction module, or with the MOOSE combined module executable.
- To run with the MOOSE combined modules executable, run:
  ```
  ~/projects/moose/modules/combined/modules-opt -i
  heat_cond.i
  ```
- To run with an application (BISON example shown here), run:
  ```
  ~/projects/bison/bison-opt -i heat_cond.i
  ```
- These examples assume your code is in the `~/projects` directory. Substitute in an appropriate path if it is located elsewhere.

Heat Conduction with Source: Results



Modules: Solid Mechanics

- MOOSE Modules' solid mechanics routines are built to help solve

$$\nabla \cdot \boldsymbol{\sigma} + b = 0$$

  where $\boldsymbol{\sigma}$ is the stress and b is a body force.

- MOOSE Modules also supplies boundary conditions useful for solid mechanics (such as pressure).

- MOOSE Modules provides spherically symmetric 1D, axisymmetric 2D (typically linear), and 3D fully nonlinear formulations. Either first or second order elements may be used (QUAD4 or QUAD8 for RZ, HEX8 or HEX20 for 3D).

Modules: Solid Mechanics

$$\nabla \cdot \boldsymbol{\sigma} + b = 0$$
$$u|_{\partial\Omega_1} = g_1$$
$$\boldsymbol{\sigma} \cdot \hat{\boldsymbol{n}}|_{\partial\Omega_2} = g_2$$

- Multiply by test function, integrate

$$(\psi_i, \nabla \cdot \boldsymbol{\sigma}) + (\psi_i, b) = 0$$

- Integrate by parts

$$-(\nabla\psi_i, \boldsymbol{\sigma}) + (\psi_i, b) + \langle \psi_i, g_2 \rangle = 0$$

- $\boldsymbol{\sigma} = \boldsymbol{C}\epsilon$

$$-(\nabla\psi_i, \boldsymbol{C}\epsilon) + (\psi_i, b) + \langle \psi_i, g_2 \rangle = 0$$

Modules: Solid Mechanics: Spherically Symmetric 1D

- The 1D, 2D, and 3D classes have much in common.
- The calculation of the strain is of course different for the three formulations. However, they share material models.
- The spherically symmetric 1D strain is

$$\begin{aligned}\epsilon_{rr} &= u_{r,r} \\ \epsilon_{zz} &= u_r/r \\ \epsilon_{\theta\theta} &= u_r/r\end{aligned}$$

- The mesh for spherically symmetric 1D is defined such that the x coordinate corresponds to the radial direction.
- No displacement in the x (radial) direction must be explicitly enforced in the input file for nodes at x=0.

Modules: Solid Mechanics: Axisymmetric 2D

- The axisymmetric 2D strain is

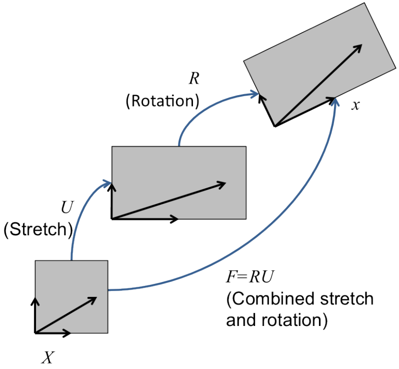
$$\begin{aligned}
\epsilon_{rr} &= u_{r,r} \\
\epsilon_{zz} &= u_{z,z} \\
\epsilon_{\theta\theta} &= u_r/r \\
\epsilon_{rz} &= (u_{r,z} + u_{z,r})/2
\end{aligned}$$

- The mesh for RZ is defined such that the x coordinate corresponds to the radial direction and the y coordinate with the axial direction.
- No displacement in the x (radial) direction must be explicitly enforced in the input file for nodes at x=0.

Modules: Solid Mechanics: Nonlinear 3D

$$F = \nabla x = \nabla u + I = RU$$

- The nonlinear kinematics formulation in MOOSE Modules accommodates both large strains and large rotations.
- The deformation gradient F can be viewed as the derivative of the current coordinates wrt the original coordinates. F can be decomposed into pure rotation R and pure stretch U.



Modules: Solid Mechanics: 3D

- We begin with a complete set of data for step n and seek the displacements and stresses at step $n+1$. We first compute an incremental deformation gradient,

$$\hat{\mathbf{F}} = \frac{\partial x^{n+1}}{\partial x^n}.$$

- With $\hat{\mathbf{F}}$, we next compute a strain increment that represents the rotation-free deformation from the configuration at n to the configuration at $n+1$. Following [1], we seek the stretching rate $\mathbf{D}$:

$$\begin{aligned}\mathbf{D} &= \frac{1}{\Delta t} \log(\hat{\mathbf{U}}) \\ &= \frac{1}{\Delta t} \log\left(\text{sqrt}\left(\hat{\mathbf{F}}^T\hat{\mathbf{F}}\right)\right) \\ &= \frac{1}{\Delta t}\log\left(\text{sqrt}\left(\hat{\mathbf{C}}\right)\right).\end{aligned}$$

- Here, $\hat{\mathbf{U}}$ is the incremental stretch tensor, and $\hat{\mathbf{C}}$ is the incremental Green deformation tensor. Through a Taylor series expansion, this can be determined in a straightforward, efficient manner.

Modules: Solid Mechanics: 3D

- $\mathbf{D}$ is passed to the constitutive model as an input for computing $\boldsymbol{\sigma}$ at $n+1$.
- The next step is computing the incremental rotation, $\hat{\mathbf{R}}$ where $\hat{\mathbf{F}} = \hat{\mathbf{R}}\hat{\mathbf{U}}$. Like for $\mathbf{D}$, an efficient algorithm exists for computing $\hat{\mathbf{R}}$. It is also possible to compute these quantities using an eigenvalue/eigenvector routine.
- With $\boldsymbol{\sigma}$ and $\hat{\mathbf{R}}$, we rotate the stress to the current configuration.

$$\begin{aligned}
\hat{\mathbf{F}} &= f(x) \\
\mathbf{D} &= f(\hat{\mathbf{F}}) \\
\hat{\mathbf{R}} &= f(\hat{\mathbf{F}}) \\
\Delta\boldsymbol{\sigma} &= f(\mathbf{D}, \boldsymbol{\sigma}_{\mathbf{n}}) \\
\boldsymbol{\sigma}_{n+1} &= \hat{\mathbf{R}}(\boldsymbol{\sigma}_n + \Delta\boldsymbol{\sigma})\hat{\mathbf{R}}^T
\end{aligned}$$

Modules: Solid Mechanics: Material Models

- The material models for 1D, axisymmetric 2D, and 3D are formulated in an incremental fashion (think hypo-elastic).
- Thus, the stress at the new step is the old stress plus a stress increment:

$$\boldsymbol{\sigma}_{n+1} = \boldsymbol{\sigma}_n + \Delta\boldsymbol{\sigma}.$$

- The incremental formulation is particularly useful for plasticity and creep models.

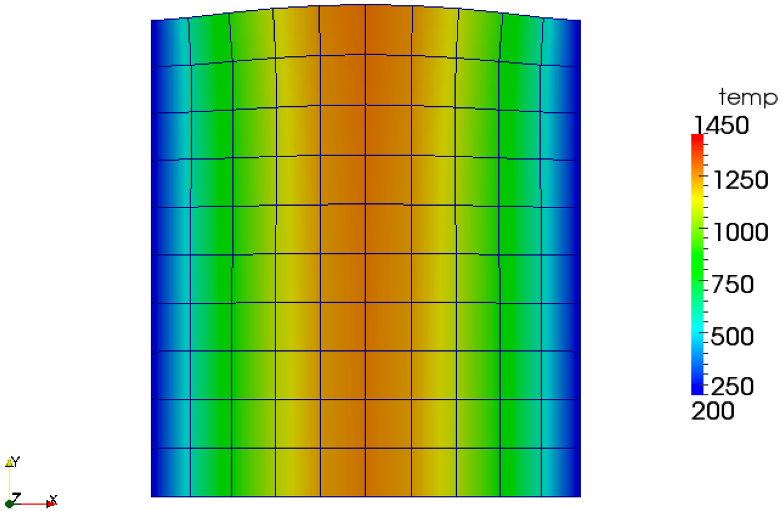
Let’s add some more physics... Solid Mechanics!

The following blocks have to be added or modified to our input file if we want to include solid mechanics behaviior.

```
[Variables]
  [./temp]
  [../]
  [./disp_x]
  [../]
  [./disp_y]
  [../]
[]

[SolidMechanics]
  [./solid]
    disp_x = disp_x
    disp_y = disp_y
    temp = temp
  [../]
[]

[Materials]
  [./heat_conductor]
    type = HeatConductionMaterial
    thermal_conductivity = 1
    block = 0
  [../]
  [./constant]
    type = LinearIsotropicMaterial
    block = 0
    youngs_modulus = 1e6
    poissons_ratio = .3
    thermal_expansion = 1e-4
    t_ref = 200
    disp_x = disp_x
    disp_y = disp_y
    temp = temp
  [../]
[]
```

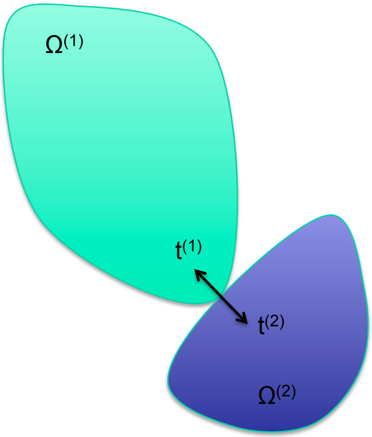
```
[BCs]
  [./leftright_temp]
    type = DirichletBC
    variable = temp
    boundary = 'left right'
    value = 200
  [../]
  [./leftright_disp_x]
    type = DirichletBC
    variable = disp_x
    boundary = 'left right'
    value = 0
  [../]
  [./bottom_disp_y]
    type = DirichletBC
    variable = disp_y
    boundary = bottom
    value = bottom
  [../]
[]
```

Heat Conduction + Solid Mechanics: Results



Modules: Contact: Finite Element Contact Basics

- A contact capability in a solid mechanics finite element code prevents the penetration of one domain into another or part of one domain into itself.



Modules: Contact: Required Capabilites

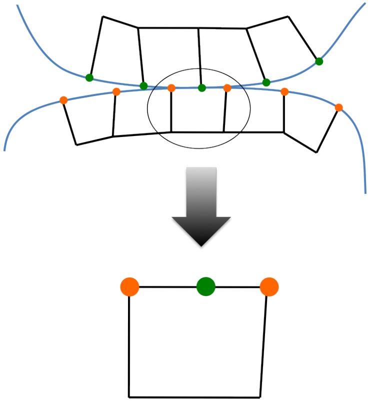
A Necessary but Insufficient List:

- Search
  - Exterior identification
  - Nearby nodes
    - Capture box
    - Binary search, e.g.
  - Contact existence
    - More geometric work
    - Penetration point
- Enforcement
  - Formulation of contact force
  - Formulation of Jacobian
  - Interaction with other capabilities (e.g., kinematic boundary conditions)

Development, testing, and application testing of contact require many, many man-months of effort. In fact, it is probably man years.

Modules: Contact: Overview

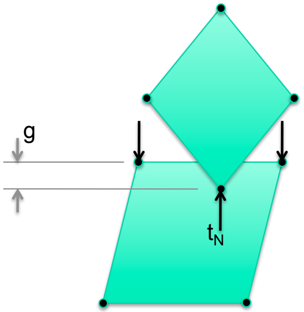
- In node-face contact, nodes (green) may not penetrate faces (defined by orange nodes).
- Forces must be determined to push against the two contacting bodies.
- No force should be applied where the bodies are not in contact.
- The contact forces must increase from zero as the bodies first come into contact.



Modules: Contact: Constraints

- $g \leq 0$; the gap (penetration distance) must be non-positive
- $t_N \geq 0$; the contact force must push bodies apart
- $t_N g = 0$; the contact force must be zero if the bodies are not in contact
- $t_N \dot{g} = 0$; the contact force must be zero when constraints are formed and released
- The gap in the normal direction for constraint i is (n is the normal, N denotes normal direction, d_s is position of the slave node, d_c is position of the contact point, and G is a matrix):

$$g_N^i = n^i(d^i(t) - d_c^i(t))$$
$$g_N^i = G_N^i(d(t))$$



Modules: Contact: Contact Options

- `formulation`: `kinematic` or `penalty`
  - Kinematic is more accurate but also harder to solve.
- `model`: `frictionless`, `glued`, or `coulomb`
  - Frictionless enforces the normal constraint and allows nodes to come out of contact if they are in tension. Glued ties nodes where they come into contact with no release. Coulomb is frictional contact with release.
- `friction_coefficient`
  - Coulomb friction coefficient.
- `penalty`
  - The penalty stiffness to be used in the constraint.
- `master`
  - The surface corresponding to the faces in the constraint.
- `slave`
  - The surface corresponding to the nodes in the constraint.
- `normal_smoothing_distance`
  - Distance from face edge in parametric coordinates over which to smooth the normal. Helps with convergence. Try 0.1.
- `tension_release`
  - The tension value which will allow nodes to be released. Defaults to zero.

Even more physics... CONTACT

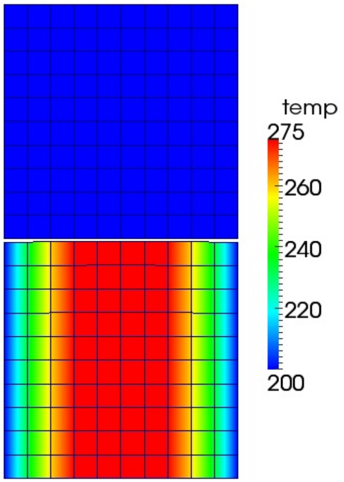
The following blocks have to be added or modified to our input file if we want to include the effects of mechanical contact.

```
[Mesh]
  file = contact.e
  displacements = 'disp_x disp_y'
[]

[Functions]
  [./source]
  type = PiecewiseLinear
  x = '0 1'
  y = '0 1'
  [../]
[]

[Kernels]
.
  [./heat_source]
    type = HeatSource
    variable = temp
    value = 1500
    function = source
    block = 2
  [../]
.
[Contact]
  [./mechanical]
    master = 1
    slave = 7
    disp_x = disp_x
    disp_y = disp_y
    penalty = 1e7
    tangential_tolerance = 0.1
  [../]
[]
```

```
[BCs]
.
.
  [./bottom_disp_y]
    type = DirichletBC
    variable = disp_y
    boundary = 3
    value = 0
  [../]
  [./bottom_disp_y_upper]
    type = DirichletBC
    variable = disp_y
    boundary = '5 6 8'
    value = 0
  [../]
.
.
[]
[Executioner]
  type = Transient
  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_hypre_type'
  petsc_options_value = 'hypre boomeramg'
  dt = 0.1
  dtmin = 0.01
  num_steps = 10
  nl_rel_tol = 1e-8
  nl_abs_tol = 1e-8
[]
```

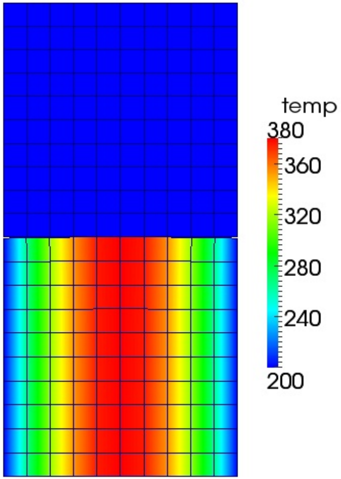
Heat Conduction + Mechanics + Contact: Results

- q = 600
- Bottom block heats and expands upward, but is not yet in contact
- Blocks do not communicate thermally (no gap heat transfer)



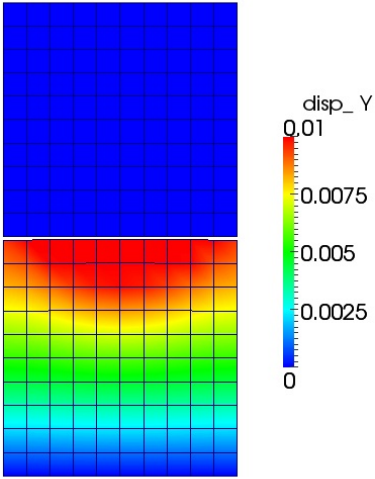
Heat Conduction + Mechanics + Contact: Results

- q = 1500
- Further heating and upward expansion brings blocks into contact, first at the center where the bottom block is hottest
- Still, blocks do not communicate thermally (no gap heat transfer)



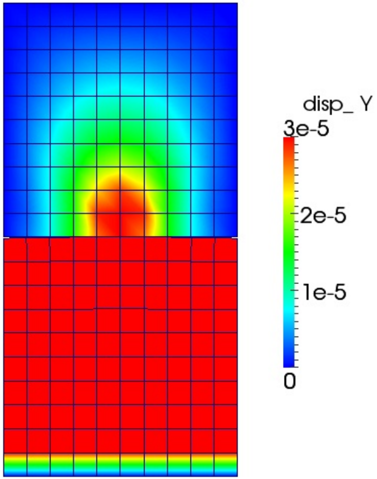
Heat Conduction + Mechanics + Contact: Results

- q = 600
- Bottom block heats and expands upward, but is not yet in contact
- Vertical displacement plots show curvature of top surface



Heat Conduction + Mechanics + Contact: Results

- q = 1500
- Contour scale is set to show displacement in top block resulting from mechanical contact



Modules: Heat Conduction: Gap Heat Transfer

- The principle is that the heat leaving one body must equal that entering another. For bodies *i* and *j* with heat transfer surface Γ:

$$\int_{\Gamma_i} h\Delta T dA_i = \int_{\Gamma_j} h\Delta T dA_j$$

- Gap heat transfer is modeled using the relation,

$$h_{gap} = h_g + h_s + h_r$$

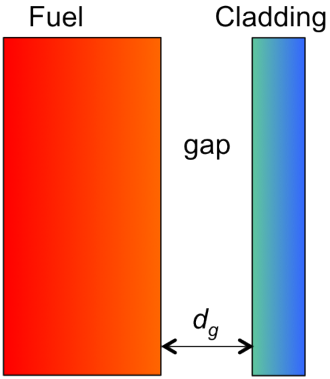
  where h_{gap} is the total conductance across the gap, h_g is the gas conductance, h_s is the increased conductance due to solid-solid contact, and h_r is the conductance due to radiant heat transfer.
- In MOOSE Modules, only the gas conductance h_g is active by default.
- The form of h_g in MOOSE Modules is

$$h_g = \frac{k_g}{d_g}$$

  where k_g is the conductivity in the gap and d_g is the gap distance.

Adding Thermal Contact

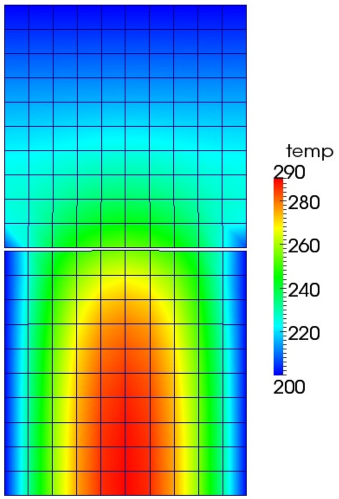
```
[ThermalContact]
  [./thermal_contact]
    type = GapHeatTransfer
    variable = temp
    master = 1
    slave = 7
    gap_conductivity = 1
  [../]
[]
```



$$h = \frac{k_g}{d_g}$$

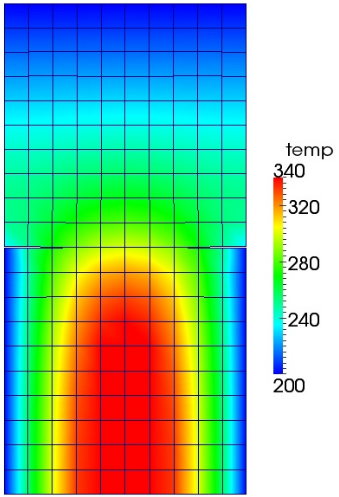
Heat Conduction + Mechanics + Contact + Thermal Contact: Results

- q = 750
- Heat tranfer occurs through the gap medium prior to mechanical contact



Heat Conduction + Mechanics + Contact + Thermal Contact: Results

- q = 1330
- Combined thermal and mechanical contact



Fuels Specific Models

www.inl.gov

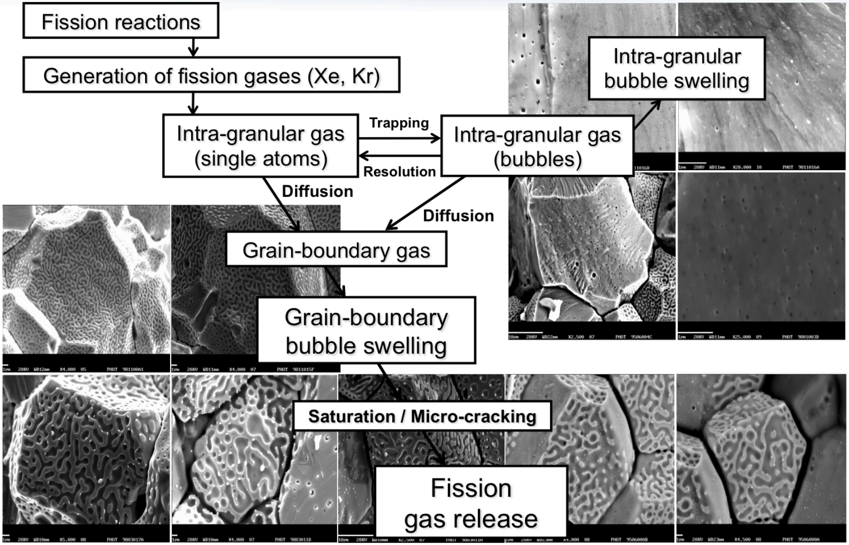
Idaho National Laboratory

Fuels Specific Models

- BISON consists, in addition to the capability in MOOSE, of material models specific to nuclear fuels:
  - Fission gas release
  - Material models that are functions of irradiation
    - creep
    - thermal conductivity
    - relocation
  - Other models that capture fuel behavior, like radial and axial power profiles
  - Gap heat transfer in LWR fuel

- This section highlights some of these models.

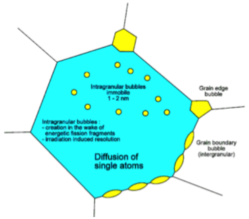
Fission Gas Behavior



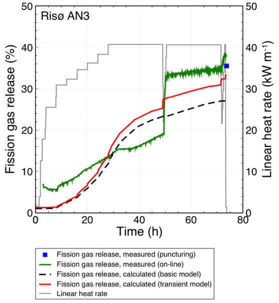
Fission reactions
Generation of fission gases (Xe, Kr)
Intra-granular gas (single atoms)
Trapping
Resolution
Intra-granular gas (bubbles)
Intra-granular bubble swelling
Diffusion
Diffusion
Grain-boundary gas
Grain-boundary bubble swelling
Saturation / Micro-cracking
Fission gas release

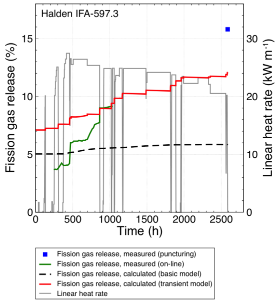
BISON Fission Gas Model

- Physics-based model which describes the different stages of fission gas behavior
  - Gas generation
  - Intra-granular diffusion to grain boundaries
  - Bubble development at grain boundaries and associated fuel swelling
  - Fission gas release due to grain boundary saturation
  - Fission gas release due to micro-cracking


Intragranular bubbles
immobile
1 - 2 nm
Grain edge bubble
Intragranular bubbles :
- creation in the wake of energetic fission fragments
- irradiation induced resolution
Grain boundary bubble (intergranular)
Diffusion of single atoms

- Current results are state-of-the-art or better





Material Models that Depend on Irradiation or Power

- Zirconium, mechanics example

$$\dot{\epsilon}_{ir} = C_0 \Phi^{C_1} {\sigma_m}^{C_2} \tag{1}$$

  where $\dot{\epsilon}_{ir}$ is the effective irradiation creep rate (1/s), Φ is the fast neutron flux (n/m^2-s), σ_m is the effective (Mises) stress (MPa), and C_0, C_1, and C_2 are material constants.

- UO_2, thermal conductivity example

$$\mathrm{k}_{95} = \mathrm{k}_{\mathrm{phonon}} + \mathrm{k}_{\mathrm{electronic}} \tag{2}$$

  The terms in Equation 2 are functions of burnup and temperature.

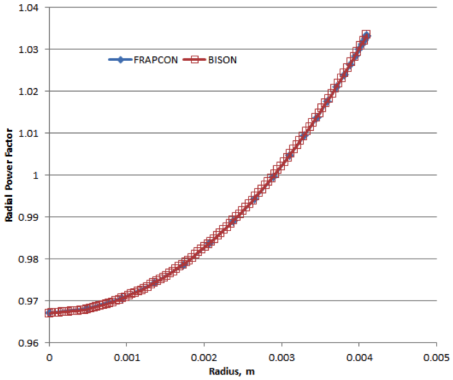
- Relocation

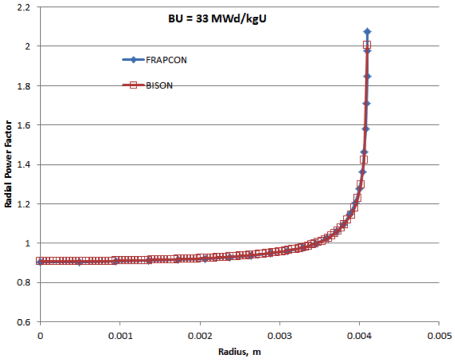
$$\left(\frac{\Delta D}{D_o}\right)_{REL} = 0.80Q\left(\frac{G_o}{D_o}\right)\left(0.005Bu^{0.3} - 0.20D_o + 0.3\right) \tag{3}$$

  This relocation model is a function of power(Q), as-fabricated pellet diameter(D_o), as-fabricated gap thickness (G_o), and burnup.

Power Profiles

- Radial power profile example





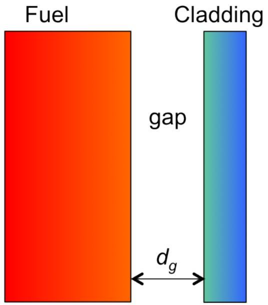
- Don’t forget the axial profile

LWR Gap Heat Transfer

- In BISON h_g and h_s are described using the form proposed by Ross and Stoute [2]. h_g is defined as

$$h_g = \frac{k_g(T_g)}{d_g + C_r(r_1 + r_2) + g_1 + g_2}$$

where k_g is the conductivity of the gas in the gap, d_g is the gap width, C_r is a roughness coefficient, r_1 and r_2 are roughnesses of the surfaces, and g_1 and g_2 are jump distances, which become important for small gap widths and low gas pressures. The jump distances provide a reduction in gap conductance when the mean free path of the gas molecules is significant in comparison to the gap width, and the continuum approximation is no longer valid. The gas temperature (T_g) is the average of the two surfaces.

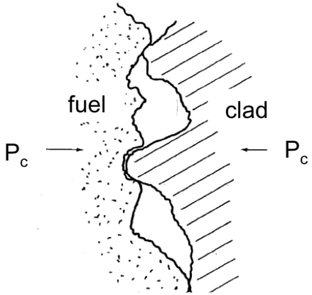


LWR Gap Heat Transfer

- h_s is defined as

$$h_s = C_s \frac{2k_1 k_2}{k_1 + k_2} \frac{P_c}{\delta^{1/2} H}$$

where C_s is an empirical constant, k_1 and k_2 are the thermal conductivities of the two materials, P_c is the contact pressure, δ is the average gas film thickness (approximated as $0.8(r_1 + r_2)$), and H is the Meyer hardness of the softer material.



BISON Gap Heat Transfer (continued)

- In BISON h_r is computed using a diffusion approximation. Based on the Stefan-Boltzmann law,

$$q_r = \sigma F_e(T_1^4 - T_2^4) \approx h_r(T_1 - T_2)$$

where σ is the Stefan-Boltzmann constant, F_e is an emissivity function, and T_1 and T_2 are the temperatures of the radiating surfaces.

- The radiant conductance is approximated as

$$h_r \approx \sigma F_e(T_1^4 - T_2^4)/(T_1 - T_2)$$

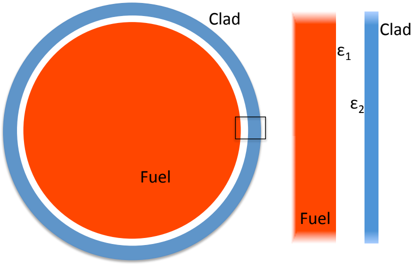
which can be reduced to

$$h_r = \sigma F_e(T_1^2 + T_2^2)(T_1 + T_2).$$

For infinite parallel plates,

$$F_e = 1/(1/\epsilon_1 + 1/\epsilon_2 - 1)$$

where ϵ_1 and ϵ_2 are the emissivities of the radiating surfaces. This is the specific function implemented in BISON.

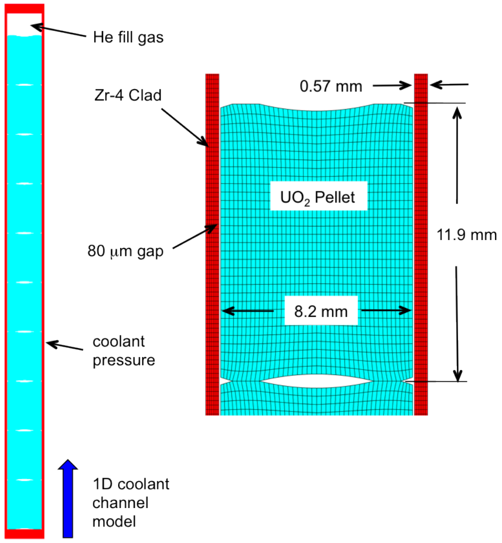


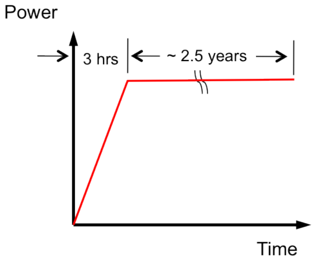
Example Problem

www.inl.gov

Idaho National Laboratory

Axisymmetric 10 Pellet LWR Fuel Rodlet





Linear average power	250 W/cm
Fast neutron flux	7.5x10^{17} n/m^{2}s
Coolant pressure	15.5 MPa
Coolant inlet temperature	580 K
Coolant inlet mass flux	3800 kg/m^2-s
Rod fill gas	helium
Fill gas initial pressure	2.0 MPa
Initial fuel density	95% theoretical
Fuel densification	1% theoretical
Burnup at full densification	5 MWd/kgU

Overview of the Example Problem

- In this section, we will review the syntax found in an example problem.
- The full input file is at `bison/examples/2D-RZ_rodlet_10pellets/inputQuad8.i`.
- The input file syntax consists of blocks where the PDEs you are solving are defined.

High-level Description of the Input File

- The input file is the place where you specify all the terms of your PDE(s) and supporting information to solve them. This is a different mindset than that used when running other simulation software.
- Things you define here are:
  - Mesh
  - Global Parameters (density and FE specification)
  - Coordinate system (RZ)
  - Physics kernels (individual terms in the PDE you're solving)
  - Source term (power)
  - Boundary Conditions (convection coefficient and displacement BCs)
  - Material models (UO_2 and Zr)
  - AuxKernels - Auxillary equations that you want to solve that may be used as input to Kernels or BCs or just for visualization
  - Post Processors (plenum pressure and average power)
- What's the minimum input file content requirement for running a problem? That depends on what PDE(s) you are solving and the information needed to support those solves.

BISON Conventions

- BISON uses several empirical models that were developed with a certain set of units.
- BISON converts from the input units to the units needed by each empirical model.
- The input units for BISON are:
  - meter, kilogram, second, kelvin, mole
- BISON uses FIMA (fissions per initial metal atom) to describe burnup.
- The coordinate convention for LWR analysis is that the rod axis corresponds to the y-axis in the global coordinate system. For axisymmetric RZ analyses, this implies that the r-direction (radial direction) corresponds to the x-axis and the z-direction corresponds to the y-axis.

Common Kernels

Kernels often found in a BISON input file include

- HeatConduction
  - Gradient term in heat conduction equation
- HeatConductionTimeDerivative
  - Time term in heat conduction equation
- NeutronHeatSource
  - Source term in heat conduction equation
- ArrheniusDiffusion
  - Arrhenius equation for mass diffusion
- HeatSource
  - General source term for heat conduction or mass diffusion. For example, this could be used as an alternative to NeutronHeatSource.
- Gravity
- Decay
  - Sink term for mass diffusion or heat conduction

Common AuxKernels

AuxKernels often found in a BISON input file include

- FastNeutronFluxAux
  - Compute fast flux based on power
- FastNeutronFluenceAux
  - Compute fast fluence based on fast flux
- MaterialTensorAux
  - Compute volume-averaged stress and strain
- FissionRateAux
  - Compute fission rate based on power
  - Not used if the Burnup block is used
- BurnupAux
  - Compute burnup based on fission rate
  - *This is not the same as the Burnup block*
  - Not used if the Burnup block is used

Common Materials

Materials often found in a BISON input include

- ThermalFuel
  - Compute thermal conductivity and specific heat for fuel
- HeatConductionMaterial
  - Set thermal conductivity and specific heat for a general material
- Density
  - Compute density, which may change due to deformation
- CreepUO2
  - Creep model for fuel
- MechZry
  - Primary and secondary thermal and irradiation creep model for clad
- VSwellingUO2
  - Densification and solid and gaseous swelling for fuel clad
- RelocationUO2
  - Relocation model for fuel
- Sifgrs
  - Fission gas release model. Also has option for calculating gaseous swelling.

Note that some of these models are empirical and have a limited ranges of applicability.

Common BCs

BCs often found in a BISON input file include

- DirichletBC and PresetBC
  - Set dirichlet BCs
- FunctionDirichletBC and FunctionPresetBC
  - Set dirichlet BCs based on a function
- NeumannBC
  - Set gradient of a variable
- Pressure
  - Set pressure on a surface
  - Note that this block requires subblocks
- PlenumPressure
  - Set pressure on interior of clad, exterior of fuel
  - Note that this block requires subblocks

Common Postprocessors

Postprocessors often found in a BISON input file include

- SideAverageValue
  - Compute the area-weighted average of a variable
- InternalVolume
  - Compute the volume of a closed sideset
- SideFluxIntegral
  - Integrated flux over an area
- TimestepSize
  - Report the time step size
- ElementIntegralPower
  - Total power by integrating the fission rate over all elements
- FunctionValuePostprocessor
  - Value of a time-varying function

Other Common Blocks

Other blocks often found in a BISON input file include

- Burnup
  - Compute the fission rate and burnup including the radial power profile effect
  - Note that this block requires subblocks
- Contact
  - Enforce mechanical contact constraints
  - Note that this block requires subblocks
- ThermalContact
  - Enforce gap heat transfer
  - Note that this block requires subblocks
- SolidMechanics
  - Divergence of stress in Cauchy's equation
  - SolidMechanics appears as its own block outside of Kernels
- CoolantChannel
  - Compute a convective boundary condition for the clad
  - Note that this block requires subblocks
- Executioner
  - Specify solver options and time stepping controls
- Outputs
  - Specify output options

Overview of Input File Format

```
# It is common to refer to a section of
#   the input file within an opening/closing
#   pair of square brackets as a 'block'.

[CategoryA]
  [./name1]
    type = Type1
    param1 = a_string_param
    param2 = '1 3 4' # a list
  [../]
  [./name2]
    type = Type2
    param = 3.14
  [../]
[]

[CategoryB]
  [./name3]
    type = Type3
    param = false
  [../]
[]
```

Input files for MOOSE applications follow the basic pattern here. On any line, any text following a '#' is considered a comment. Major categories (such as Kernels, BCs, Materials, etc.) are identified within square brackets and are closed with empty square brackets. Each specific instance of a given category begins with '[./<name>]' and ends with '[../]'. Parameters are given as key/value pairs separated with an equal sign. If the value is a list, the list is enclosed in single quotes.

Input Syntax

If you have questions about input syntax or what options are available for parameters.

Type: `> ~/projects/bison/bison-opt --dump <Parameter_in_question>`

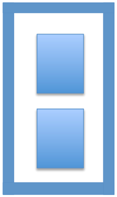
Example: `> ~/projects/bison/bison-opt --dump Postprocessors`

Blocks, sidesets, and nodesets

Some conventions to keep in mind as you look at the input file and consider blocks, sidesets, and nodesets in reference to BCs, source terms, and material models.

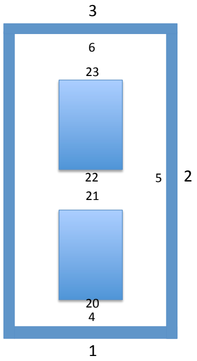
Blocks

Block 1: Cladding
Block 2: Liner (if present)
Block 3+: Fuel
Use 'clad', 'liner', 'pellet_type_1', 'pellet_type_2', etc.



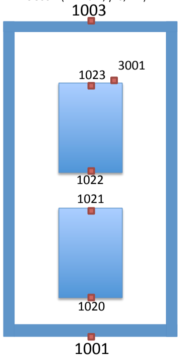
Sidesets

Sideset 7: Cladding Interior
Sideset 8: All pellet exteriors
Sideset 9: Union of 7 & 8
Sideset 10: Outer Radial Surface of Pellets
Sideset 11: Top Pellet Top
Sideset 12: Centerline (for RZ)
Sideset 13: Inner Radial Surface of Pellets



Nodesets

NS 1004: All central nodesets
NS 1005: All central pellet nodesets
NS 2000: Bottom Center Meso
NS 2001: Bottom Outer Meso
NS 2002: Middle Center Meso
NS 2003: Middle Outer Meso
NS 3000: Top Center Pellet
NS 3002: (x=<max>, y=0, z=*)



The GlobalParams Block

```
[GlobalParams]
  density = 10431.0
  disp_x = disp_x
  disp_y = disp_y
  order = SECOND
  family = LAGRANGE
  # J/fission
  energy_per_fission = 3.2e-11
[]
```

The `GlobalParams` block sets parameters that can be used by any other block. We can set parameters here instead of setting them many times in the remainder of the file. Here, we set the value of density, displacement solution variables, and variable order and family. The `Variables` block did not contain any lines for order and family. The `GlobalParams` contains them, and so they are not needed in the `Variables` block. The `GlobalParams` block is often listed first, but can occur anywhere within the input file.

The Problem Block

```
[Problem]
  coord_type = RZ
[]
```

This block needs to be included for an axisymmetric analysis. It tells BISON that all of the boundary conditions, kernels, and material models should be evaluated in axisymmetric coordinates. Similarly coord_type can be set to RSPHERICAL to specify a 1D spherically symmetric analysis.

The Mesh Block

```
[Mesh]
  file = quad8Medium10_rz.e
  displacements = 'disp_x disp_y'
  patch_size = 10
[]
```

Mesh parameters are defined in this block. The following parameters are defined in this particular block:

- `file`: defines the name of the finite element mesh file
- `displacements`: lists the names of the displacement variables (needed for large displacement, contact)
- `patch_size`: used by contact to define the number of nearest neighbor nodes

MOOSE also includes capability to build simple meshes (e.g., smeared pellet) from within the input file.

The Variables Block

```
[Variables]
  [./disp_x]
  [../]

  [./disp_y]
  [../]

  [./temp]
    # set initial temp to fill gas
 temperature, usually 20C
    initial_condition = 293
  [../]
[]
```

Dependent variables and initial conditions are examples of parameters defined in the variables block. Notice there are sub-blocks, whose names correspond to the dependent variables.

The AuxVariables Block

```
[AuxVariables]
  [./fast_neutron_flux]
    block = clad
  [../]

  [./fast_neutron_fluence]
    block = clad
  [../]
[]
```

What are AuxVariables? They are variables in addition to the dependent variables that allow explicit calculations. These can be used by kernels, boundary conditions, and material properties. AuxVariables are written to the output file. You can define two types of AuxVariables ... Element (constant monomial) or Nodal (linear Lagrange). AuxVariables have old states, just like the dependent variables. Some parameters you can set in this block are `order` (e.g. linear), `family` (e.g. Lagrange), and `block`. The `block` parameter specifies which blocks of the finite element mesh have the named AuxVariable.

The AuxVariables Block - continued

```
[./stress_xx]
  order = CONSTANT
  family = MONOMIAL
[../]
```

If you want to make contour plots of stress in the x direction, you have to include this sub-block. When you use `order = CONSTANT` and `family = MONOMIAL`, you will get an average of the integration point values at each element. If you use `order = FIRST`, `family = LAGRANGE`, values at nodes are written to the output file. `CONSTANT MONOMIAL` is the appropriate choice for stresses.

The Functions Block

```
[Functions]
  [./power_profile]
    type = PiecewiseLinearFile
    data_file = powerhistory.csv
    scale_factor = 1
  [../]
  [./axial_peaking_factors]
    type = PiecewiseBilinear
    data_file = peakingfactors.csv
    scale_factor = 1
    axis = 1 # (0,1,2) => (x,y,z)
  [../]
  [./pressure_ramp]
    type = PiecewiseLinear
    x = '-200 0'
    y = '   0 1'
  [../]
  [./q]
    type = CompositeFunction
    functions = 'power_history
                 axial_peaking_factors'
  [../]
[]
```

Functions can be used to define inputs to the simulation, such as power history, power factors, and pressure boundary conditions, to name a few. Notice some of the parameters you can specify, such as `type` (defines the function type), `data_file` (specifies the name of a data file), or `value` (sets the value of the function).

The Kernels Block ... and a note about Actions

```
[SolidMechanics]
  [./solid]
    disp_r = disp_x
    disp_z = disp_y
    temp = temp
  [../]
[]
[Kernels]
# for stress equilibrium equation
  [./gravity]
    type = Gravity
    variable = disp_y
    value = -9.81
  [../]
# for heat conduction equation
  [./heat]
    type = HeatConduction
    variable = temp
  [../]
# for heat conduction equation
  [./heat_ie]
    type = HeatConductionTimeDerivative
    variable = temp
  [../]
[]
```

Here's an example of a Kernel block. Notice that there's a block above the Kernel block called SolidMechanics. The SolidMechanics block defines the kernels relative to stress divergence. It has its own block to cut down on redundancy in the input file. Such a block is built using Actions from MOOSE. Future development work includes removing the requirement to specify kernels for a typical fuels problem (because they are the same every time) and only specifing the unique parameters for a particular simulation.

Kernels ... continued

```
[Kernels]
  [./gravity]
    type = Gravity
    variable = disp_y
    value = -9.81
  [../]
  [./heat]
    type = HeatConduction
    variable = temp
  [../]
  [./heat_ie]
    type = HeatConductionTimeDerivative
    variable = temp
  [../]
  [./heat_source]
     type = NeutronHeatSource
     variable = temp
     block = pellet_type_1
     fission_rate = fission_rate
  [../]
[]
```

Recall that kernels are used to define the various terms in the PDE set that you are solving. Each kernel is a term in the PDE. For example, the kernel `gravity` is the body force term in the stress divergence equation, `heat` is the gradient term in the heat conduction equation, `heat_ie` is the time term, and `heat_source` is the source term. Note that `heat_source` is coupled to `fission_rate` (an Aux Variable) and applied only in block 2 (the fuel). The required parameters are `type` (the actual name of the kernel) and `variable` (temperature or displacement).

Burnup Block

```
[Burnup]
  [./burnup]
    block = pellet_type_1
    rod_ave_lin_pow = power_history
    axial_power_profile = axial_peaking_factors
    num_radial = 80
    num_axial = 11
    a_lower = 2.26e-3
    a_upper = 1.2086e-1
    fuel_inner_radius = 0
    fuel_outer_radius = .0041
    # ratio of actual volume to cylinder volume:
    fuel_volume_ratio = 0.987775
    #N235 = N235
    #N238 = N238
    #N239 = N239
    #N240 = N240
    #N241 = N241
    #N242 = N242
    RPF = RPF
  [../]
[]
```

The Burnup block supplies input for computing burnup and fission rate based on the power history, the axial profile, and a radial power factor. The radial power factor is computed at grid points independent of the finite element mesh. The density of these points can be controlled with the `num_radial` and `num_axial` parameters. The parameters `N235` through `RPF` are optional and specify that the associated isotope concentrations and radial power factor value should be written to the results file.

AuxKernels Block

```
[AuxKernels]
  [./fast_neutron_flux]
    type = FastNeutronFluxAux
    variable = fast_neutron_flux
    block = clad
    rod_ave_lin_pow = power_profile
    axial_power_profile = axial_peaking_factors
    factor = 3e13
    execute_on = timestep_begin
  [../]

  [./fast_neutron_fluence]
    type = FastNeutronFluenceAux
    variable = fast_neutron_fluence
    block = clad
    fast_neutron_flux = fast_neutron_flux
    execute_on = timestep_begin
  [../]
[]
```

Here's an example of an AuxKernel block. Only fast neutron flux and fluence are shown here, but others may be used. Remember that every AuxVariable requires a corresponding AuxKernel. Important parameters include `type` (the kind of AuxKernel), `variable` (the AuxVariable to use), `block` (gives which mesh block the AuxKernel acts on), and `execute_on`, which specifies at which point in the algorithm the AuxKernel executes. Other inputs include functions (e.g., to define the rod average linear power) and constants required for particular AuxKernels.

AuxKernels Block ... continued

```
[AuxKernels]
  [./fast_neutron_flux]
    type = FastNeutronFluxAux
    variable = fast_neutron_flux
    block = clad
    rod_ave_lin_pow = power_profile
    axial_power_profile = axial_peaking_factors
    factor = 3e13
    execute_on = timestep_begin
  [../]

  [./fast_neutron_fluence]
    type = FastNeutronFluenceAux
    variable = fast_neutron_fluence
    block = clad
    fast_neutron_flux = fast_neutron_flux
    execute_on = timestep_begin
  [../]
[]
```

Also note that one of the parameters in the `fast_neutron_fluence` block is `fast_neutron_flux`. So, the AuxVariable `fast_neutron_fluence` is a function of the AuxVariable `fast_neutron_flux`.

AuxKernels Block ... continued ... again

```
[./vonmises]
  type = MaterialTensorAux
  tensor = stress
  variable = vonmises
  quantity = vonmises
[../]
```

Recall that AuxVariables/AuxKernels are also used to write results to the output file. If this block is included in the AuxKernels block, and `vonmises` is defined in the AuxVariables block, then von Mises stress will be available to visualize in the output file.

Contact

```
[Contact]
  [./pellet_clad_mechanical]
    master = 5
    slave = 10
    disp_x = disp_x
    disp_y = disp_y
    system = constraint
    penalty = 1e7
   [../]
[]
[ThermalContact]
  [./thermal_contact]
    type = GapHeatTransferLWR
    variable = temp
    master = 5
    slave = 10
    initial_moles = initial_moles
    gas_released = fis_gas_released
    contact_pressure = contact_pressure
    quadrature = true
  [../]
[]
```

The Contact block defines mechanical contact between the fuel (side set 10, slave) and the clad (side set 5, master). The displacement solution variables are parameters. `penalty` is the stiffness of a constraint that prevents the surfaces from penetrating by applying a normal force along the contact surface.

Contact ... continued

```
[Contact]
  [./pellet_clad_mechanical]
    master = 5
    slave = 10
    disp_x = disp_x
    disp_y = disp_y
    system = constraint
    penalty = 1e7
   [../]
[]
[ThermalContact]
  [./thermal_contact]
    type = GapHeatTransferLWR
    variable = temp
    master = 5
    slave = 10
    initial_moles = initial_moles
    gas_released = fis_gas_released
    quadrature = true
  [../]
[]
```

The thermal contact block specifies contact between the fuel (side set 10) and the clad (side set 5). The temperature variable must be given. `initial_moles` couples to a Postprocessor that supplies the initial plenum/gap gas mass, and `gas_released` couples to a Postprocessor that supplies the fission gas addition to the gap thermal conductance equation. More on Postprocessors later.

Boundary Conditions Block

```
[BCs]
  [./Pressure]
    [./coolantPressure]
      boundary = '1 2 3'
      factor = 15.5e6
      function = pressure_ramp
    [../]
  [../]
[]
```

This block describes the coolant pressure on the outside of the clad. Notice the parameter `function` that refers to a function defined in the Functions block. The parameter `factor` scales the pressure ramp function to complete the specification of pressure on the clad exterior.

Boundary Conditions Block ... continued

```
[BCs]
  [./PlenumPressure]
    [./plenumPressure]
      boundary = 9
      initial_pressure = 2.0e6
      startup_time = 0
      R = 8.3143
      output_initial_moles = initial_moles
      temperature = ave_temp_interior
      volume = gas_volume
      material_input = fis_gas_released
      output = plenum_pressure
      displacements = 'disp_x disp_y'
    [../]
  [../]
[]
```

The plenum pressure block defines the pressure on the inside of the clad and outside of the fuel using the ideal gas law. `initial_pressure` sets the value of the fill gas pressure. `R` is the universal gas constant. The other parameters are links to Postprocessors that provide input to this boundary condition. The parameter `output` writes the magnitude of the plenum pressure to a Postprocessor called `plenum_pressure`.

Coolant Channel Block

```
[CoolantChannel]
  [./convective_clad_surface]
    boundary = '1 2 3'
    variable = temp
    inlet_temperature = 580       # K
    inlet_pressure    = 15.5e6    # Pa
    inlet_massflux    = 3800      # kg/m^2-sec
    rod_diameter      = 0.948e-2 # m
    rod_pitch         = 1.26e-2  # m
    linear_heat_rate  = power_profile
    axial_power_profile = axial_peaking_factors
  [../]
[]
```

This block specifies the convection of the coolant on the outside of the clad. `boundary` defines the side sets where the BC is applied. In this BC, inlet temperature, pressure, and mass flux are required. Rod diameter and pitch are also needed. Finally, the time varying power profile and axially varying factors are given.

Material Properties/Models Block

```
[Materials]
  [./fission_gas_release]
    type = Sifgrs
    block = pellet_type_1
    temp = temp
# coupling to fission_rate aux variable
    fission_rate = fission_rate
    gbs_model = true
  [../]
```

There are many thermal and solid mechanics material models and properties that can be defined here for the fuel and clad. The thermal models for UO_2 are functions of temperature and burnup. The solid mechanics models specify features such as elasticity, creep, volumetric strains, fuel relocation, clad growth, and smeared cracking. The block shown here is for the fission gas release model, which is the Sifgrs model. Note that it is coupled to the AuxVariables `fission_rate` and the solution variable `temp`.

Material Properties/Models Block ... continued

```
[./clad_solid_mechanics]
  type = MechZry
  block = clad
  disp_r = disp_x
  disp_z = disp_y
  temp = temp
  fast_neutron_flux = fast_neutron_flux
  fast_neutron_fluence = fast_neutron_fluence
  youngs_modulus = 7.5e10
  poissons_ratio = 0.3
  thermal_expansion = 5.0e-6
  output_iteration_info = false
  model_thermal_expansion = false
  model_irradiation_growth = true
  stress_free_temperature = 295.0
[../]
```

Here's an example of a solid mechanics block. This is a creep model for the clad, which calculates creep and elastic strains. Notice how the primary solution variables (displacements and temperature) are parameters, which are fully coupled. Also note that `fast_neutron_flux` and `fast_neutron_fluence` are parameters, which are defined via the AuxVariable/AuxKernel system. The material MechZry has an option to compute anisotropic thermal strain. In this example, it is set to false, so isotropic thermal strain is computed.

Material Properties/Models Block ... continued

```
[./fuel_solid_mechanics_swelling]
  type = VSwellingUO2
  block = pellet_type_1
  temp = temp
  burnup = burnup
[../]
[./fuel_creep]
  type = CreepUO2
  block = pellet_type_1
  disp_r = disp_x
  disp_z = disp_y
  temp = temp
  fission_rate = fission_rate
  youngs_modulus = 2.e11
  poissons_ratio = .345
  thermal_expansion = 10e-6
  grain_radius = 10.0e-6
  oxy_to_metal_ratio = 2.0
  max_its = 10
  output_iteration_info = false
  stress_free_temperature = 295.0
[../]
```

Shown in this block are two solid mechanics models, each applied to the fuel (block = pellet_type_1). The first is a volumetric swelling and densification model that is a function of temperature and burnup. The second is a fuel creep model.

Material Properties/Models Block ... continued

```
[./fuel_relocation]
  type = RelocationUO2
  block = pellet_type_1
  burnup = burnup
  diameter = 0.0082
  q = q
  gap = 160e-6 # diametral gap
  burnup_relocation_stop = 1.e20
[../]
```

Here, we define relocation, which is applied to the fuel (block = pellet_type_1). Note the fuel pellet `diameter` and `diametral_gap` are parameters here and that they are currently specified independent of the mesh. This model depends on burnup and the function `q`. The relocation strain will cease when time reaches the value defined by `burnup_relocation_stop`.

Material Properties/Models Block ... continued

```
[./fuel_thermal]
  type = ThermalFuel
  block = pellet_type_1
  temp = temp
  burnup = burnup
  model = 4
[../]
```

This material block describes a thermal model for the fuel that is a function of temperature and burnup. Options include

0. Duriez-Lucuta
1. Amaya-Lucuta
2. Fink-Lucuta
3. Halden
4. NFIR w/wo Gd
5. Modified NFI

Material Properties/Models Block ... continued

```
[./clad_density]
  type = Density
  block = clad
  density = 6551.0
  disp_r = disp_x
  disp_z = disp_y
[../]
[./fuel_density]
  type = Density
  block = pellet_type_1
  disp_r = disp_x
  disp_z = disp_y
[../]
```

These material blocks give the density for the clad and fuel. The density will be updated due to deformation as the simulation progresses. Note that the density parameter for the fuel will be taken from the GlobalParams block.

The Executioner

```
[Executioner]
  type = Transient
  solve_type = PJFNK
  # PETSC options
  petsc_options_iname = <more petsc options>
  petsc_options_value = <more petsc options>
  line_search = none
  # controls for linear iterations
  l_max_its = 100
  l_tol = 8e-3
  # controls for nonlinear iterations
  nl_max_its = 15
  nl_rel_tol = 1e-4
  nl_abs_tol = 1e-10
  # time control
  start_time = -200
  end_time = 8.0e7
  num_steps = 5000
  dtmax = 2e6
  dtmin = 1
  # adaptive timestepping options
  [./TimeStepper]
    type = IterationAdaptiveDT
    dt = 2e2
    optimal_iterations = 6
    iteration_window = 2
    linear_iteration_ratio = 100
  [../]
[]
```

This block contains solver and time stepping controls. The most important parameters are `start_time`, `dt` (time step), `end_time`, and the adaptive timestepping options. The start time is set to -200, which accounts for a cold zero to hot zero power (fabrication conditions to hot zero power reactor conditions).

The Postprocessors Block

```
[Postprocessors]
  [./ave_temp_interior]
     type = SideAverageValue
     boundary = 9
     variable = temp
     execute_on = linear
   [../]
[]
```

The `Postprocessors` block defines several quantities that are used throughout the simulation. This particular Postprocessor is computing the average temperature of the cladding and all pellet exteriors. The result of Postprocessor calculations is one scalar at every time throughout the simulation. Recall that `ave_temp_interior` is used in the plenum pressure BC. There are too many to list, but here are the names of a few that you will find in the example problem: `fission_gas_produced`, `fission_gas_released`, `pellet_volume`, and `average_clad_temperature`.

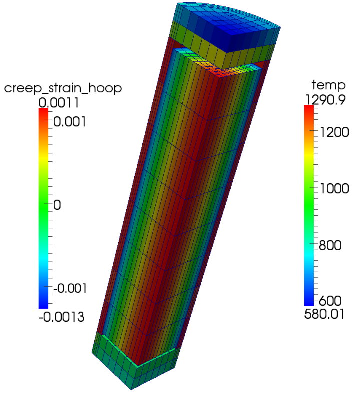
Output ... FINALLY!

```
[Outputs]
  file_base = medium_out
  exodus = true
  [./console]
    type = Console
    perf_log = true
    max_rows = 25
  [../]
[]
```

The parameter `file_base` specifies the prefix of the output file name. If not given, the output file base name will be the base name of the input file. `exodus = true` defines the type of output file. `perf_log = true` specifies whether or not the performance log should be printed.

Other Examples

- You will find other examples of running BISON at `bison/examples`.
- It may also be helpful to review assessment cases at `bison/assessment`.



Hoop Direction Creep Strain on Cladding, Temperature on Fuel

Mesh Generation

BISON Input Files

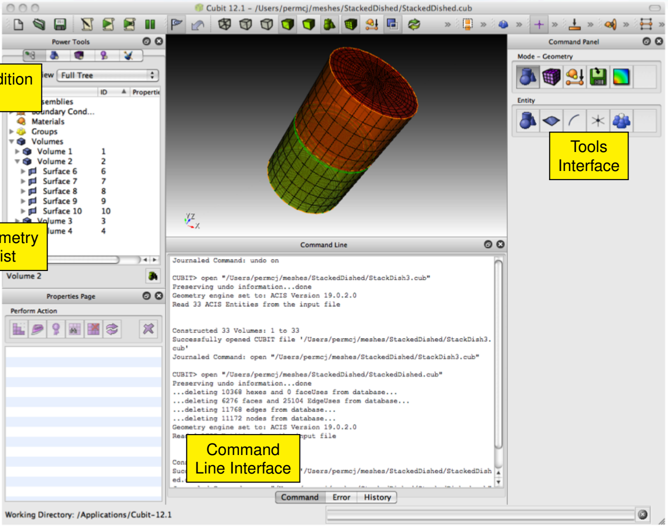
- BISON requires two files in order to run.
- The first of these is an input text file.
- The second is an input mesh file.
  - The default format is ExodusII [3].
  - This is a binary file format.
- The creation of the mesh file is the subject of this section.

How to Generate an ExodusII File

- CUBIT from Sandia National Laboratories (cubit.sandia.gov) [4].
  - Use CUBIT directly.
  - Use scripts to drive CUBIT. (This is the recommended option.)
- Smeared pellet mesh generator within BISON.
- Create an Abaqus file and import that into BISON instead.
- Output ExodusII from Patran or Ansys.

CUBIT can be licensed from Sandia (free for government use). See the website for details.

CUBIT Interface



```
Journaled Command: undo on

CUBIT> open "/Users/permcj/meshes/StackedDished/StackDish3.cub"
Preserving undo information...done
Geometry engine set to: ACIS Version 19.0.2.0
Read 33 ACIS Entities from the input file


Constructed 33 Volumes: 1 to 33
Successfully opened CUBIT file '/Users/permcj/meshes/StackedDished/StackDish3.
cub'
Journaled Command: open "/Users/permcj/meshes/StackedDished/StackDish3.cub"

CUBIT> open "/Users/permcj/meshes/StackedDished/StackedDished.cub"
Preserving undo information...done
...deleting 10368 hexes and 0 faceUses from database...
...deleting 6276 faces and 25104 EdgeUses from database...
...deleting 11768 edges from database...
...deleting 11172 nodes from database...
Geometry engine set to: ACIS Version 19.0.2.0
Rea                       nput file


Con
Suc                       '/Users/permcj/meshes/StackedDished/StackedDish
ed.
```

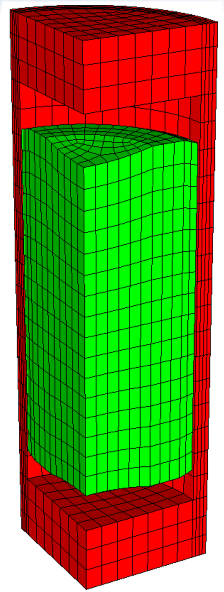
CUBIT Capabilities

- Generating a solid model
- Importing a solid model
- Automatically generating a mesh for simple geometries
- Creating 1D, 2D, or 3D meshes
- Assigning blocks, side sets, and node sets
- Being driven by a GUI, command line, journal file, or Python

BISON's Mesh Generation Scripts

- Shell and Python scripts for mostly-automatic fuel rod mesh generation are in `bison/tools/UO2`

- Relevant files are:
  - `mesh_script.sh`: Sets up environment variables. Calls `mesh_script.py` and `mesh_script_input.py`.
  - `mesh_script.py`: Main script. Interfaces with CUBIT. Handles both 2D and 3D geometries. User should not have to modify this file.
  - `mesh_script_input.py` Input file. Defines geometry and mesh parameters using Python dictionaries.



Input file review: Fuel

mesh_script_input.py

Defines pellet geometry.

```
#!/Usr/bin/env python2.5

# Pellet Type 1: Active Fuel
# Required parameters
Pellet1= {}
Pellet1['type'] = 'discrete'          # 'smeared' or 'discrete'
Pellet1['quantity'] = 1               # Number of pellets of this type
Pellet1['mesh_density'] = 'medium'    # Defines mesh density

Pellet1['outer_radius'] = 0.005205
Pellet1['inner_radius'] = 0.0
Pellet1['height'] = 0.07521
Pellet1['dish_spherical_radius'] = 0
Pellet1['dish_depth'] = 0
Pellet1['chamfer_width'] = 0
Pellet1['chamfer_height'] = 0
```

Some of these parameters exist only for discrete geometry.

```
# Pellet Collection
# This list defines the pellet in the fuel stack.
# First item is at the bottom the fuel stack.

pellets = [Pellet2, Pellet1, Pellet2]
```

Defines generated fuel stack

Input file review: Pellet stack

mesh_script_input.py

```
# Stack options
pellet_stack = {}
pellet_stack['default_parameters'] = False

pellet_stack['interface_merge'] = 'point'
pellet_stack['higher_order'] = True
pellet_stack['angle'] = 0

# Pellet stack default parameters:
#  pellet_stack['interface_merge'] = 'point'
#  pellet_stack['higher_order'] = False
#  pellet_stack['angle'] = 0
```

Parameters review

- `default_parameters` Use default parameters without considering below parameters

- `interface_merge`
  - `'point'` (Default) Common vertex (2D) or curve (3D)
  - `'none'` not merged
- `higher_order`
  - `False`: QUAD4 (2D) or HEX8 (3D).
  - `True`: QUAD8 (2D) or HEX27 (3D).
- `angle` 0: create a 2D rz geometry. > 0 create a 3D stack of the specified angle ($\leq 360°$)

Input file review: Clad

mesh_script_input.py

```
# Clad: Geometry of the clad
clad = {}
clad['mesh_density'] = 'medium'
clad['gap_width'] = 0.11e-3
clad['bot_gap_height'] = 5e-3
clad['clad_thickness'] = 0.815e-3
clad['top_bot_clad_height'] = 28.5e-3
clad['plenum_fuel_ratio'] = 0.1813

clad['with_liner'] = True
clad['liner_width'] = 0.076e-3
```

Defines clad geometric parameters.
Please note:

- `mesh_density` Clad mesh depends on fuel mesh.
- `clad_width` This parameter is the total width of the clad **including the liner**.

Input file review: Meshing parameters

mesh_script_input.py

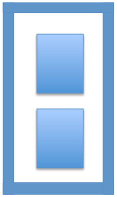
```
# Meshing parameters
mesh = {}
mesh['default_parameters'] = False

# Parameters of the mesh density 'medium'
medium = {}
medium['pellet_r_interval'] = 11
medium['pellet_z_interval'] = 3
medium['pellet_dish_interval'] = 6
medium['pellet_flat_top_interval'] = 3
medium['pellet_chamfer_interval'] = 2
medium['clad_radial_interval'] = 4
medium['clad_sleeve_scale_factor'] = 1
medium['cap_radial_interval'] = 4
medium['cap_vertical_interval'] = 3
medium['pellet_slices_interval'] = 16
medium['pellet_angular_interval'] = 12
medium['clad_angular_interval'] = 16
```

- Mesh parameters also stored in a dictionary
- The name of the dictionary must be the same as defined in the pellet type block (`mesh_density`)
- For a smeared pellet, the mesh density of the fuel is controlled by the parameters `pellet_r_interval` and `pellet_z_interval`. Other `pellet*` parameters are used with a discrete geometry.
- `clad_sleeve_scale_factor`
  - 1: same vertical density as the fuel
  - $>$ 1: higher density
  - $<$ 1: smaller density
  - Recommend $\leq$ 1

Output review: Boundary conditions

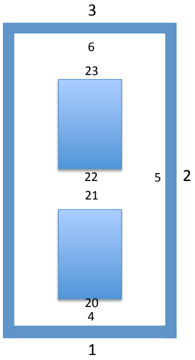
Blocks

Block 1: Cladding
Block 2: Liner (if present)
Block 3+: Fuel
Use 'clad', 'liner', 'pellet_type_1', 'pellet_type_2', etc.



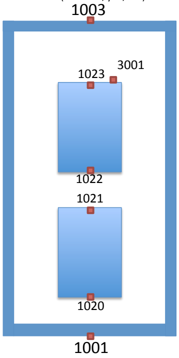
Sidesets

Sideset 7: Cladding Interior
Sideset 8: All pellet exteriors
Sideset 9: Union of 7 & 8
Sideset 10: Outer Radial Surface of Pellets
Sideset 11: Top Pellet Top
Sideset 12: Centerline (for RZ)
Sideset 13: Inner Radial Surface of Pellets



Nodesets

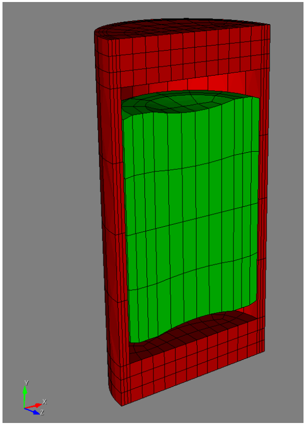
NS 1004: All central nodesets
NS 1005: All central pellet nodesets
NS 2000: Bottom Center Meso
NS 2001: Bottom Outer Meso
NS 2002: Middle Center Meso
NS 2003: Middle Outer Meso
NS 3000: Top Center Pellet
NS 3002: (x=<max>, y=0, z=*)



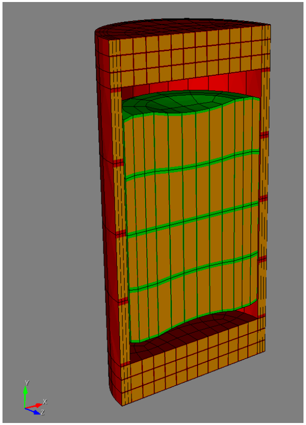
3D Boundary Conditions

- 180 Degree Model

3D Mesh



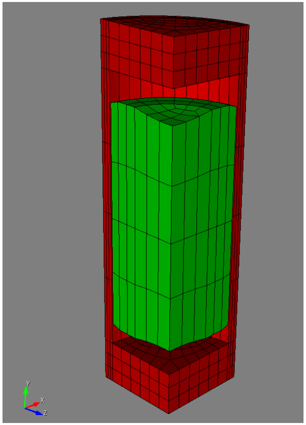
Sideset 99 Definition



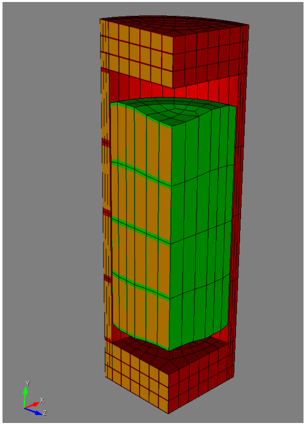
3D Boundary Conditions (cont.)

- 90 Degree Model

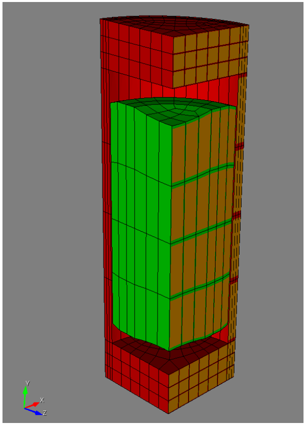
3D Mesh



Sideset 98 Definition

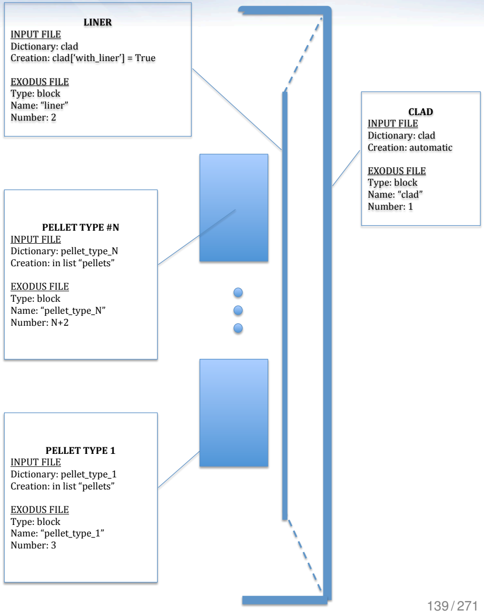


Sideset 99 Definition

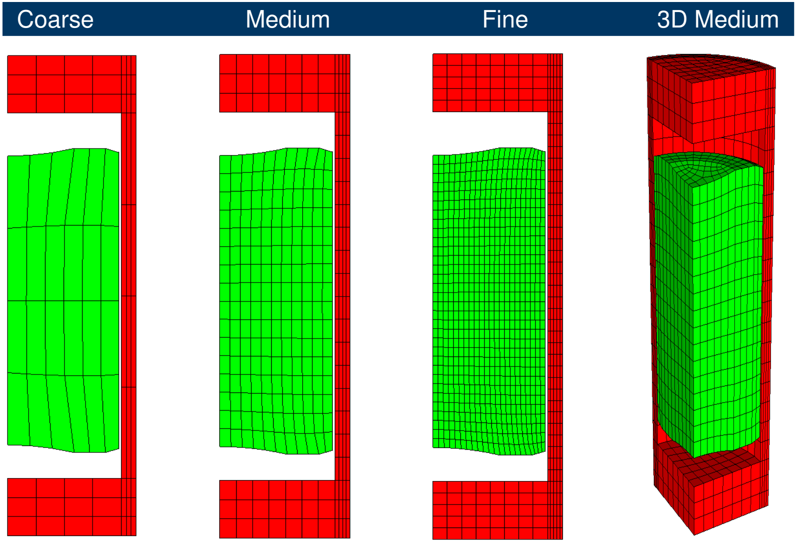


Mesh script: Wrap up

- Geometry and mesh parameters are defined in the input file for 2D or 3D geometry
- No interaction with the main script is required
- In the exodus file, blocks have these names: `clad`, `liner`, `pellet_type_#`



Mesh Examples



Running BISON

Running BISON

- This section walks through the steps of running BISON on a new problem.
- We will build upon the sections describing the input file and mesh generation.

Running BISON ... continued

Let’s assume that we want to run a problem similar to the example problem but with the following differences:

- Smeared fuel pellet mesh
- Coolant pressure of 16 MPa
- Rod averaged linear power of 21 kW/m
- Clad properties:
  - Thermal conductivity of 21.5 W/m/K
  - Specific heat capacity of 285 J/kg/K
  - Coefficient of thermal expansion of 6e-6 m/m/K
  - Density of 6560 kg/m^3
  - Young’s modulus of 99.3 GPa
  - Poisson’s ratio of 0.37
- First, copy the example problem to a new file:
  - `> mkdir bison/sandbox`
  - `> cd bison/sandbox`
  - `> cp ../examples/2D-RZ_rodlet_10pellets/inputQuad8.i myProblem.i`
- Now, in a text editor, we can change the input file.

Editing the Input File: Mesh

Before: `inputQuad8.i`

```
# Import mesh file
[Mesh]
  file = quad8Medium10_rz.e
  displacements = ’disp_x disp_y’
  patch_size = 1000 # For contact algorithm
[]
```

After: `myProblem.i`

```
# Import mesh file
[Mesh]
  file = coarse1_rz.e
  displacements = ’disp_x disp_y’
  patch_size = 1000
[]
```

Editing the Input File: Functions

Before: `inputQuad8.i`

```
# Define functions to control power, etc.
[Functions]

  [./power_history]
    type = PiecewiseLinearFile
    data_file = powerhistory.csv
    scale_factor = 1
  [../]

  [./axial_peaking_factors]
    type = PiecewiseBilinear
    data_file = peakingfactors.csv
    scale_factor = 1
    axis = 1 # (0,1,2) => (x,y,z)
  [../]

  [./pressure_ramp]
    type = PiecewiseLinear
    x = '-200 0'
    y = '    0 1'
  [../]

  [./q]
    type = CompositeFunction
    functions = 'power_history
                 axial_peaking_factors'
  [../]
[]
```

After: `myProblem.i`

```
# Define functions to control power, etc.
[Functions]

  [./power_history]
    type = PiecewiseLinear
    x = '0 1e4'
    y = '0 21000'
  [../]

  [./axial_peaking_factors]
    type = PiecewiseBilinear
    data_file = peakingfactors.csv
    scale_factor = 1
    axis = 1 # (0,1,2) => (x,y,z)
  [../]

  [./pressure_ramp]
    type = PiecewiseLinear
    x = '-200 0'
    y = '    0 1'
  [../]

  [./q]
    type = CompositeFunction
    functions = 'power_history
                 axial_peaking_factors'
  [../]
[]
```

Editing the Input File: Coolant Pressure

Before: `inputQuad8.i`

```
  [./Pressure]
#  apply coolant pressure on clad outer walls
    [./coolantPressure]
      boundary = '1 2 3'
      factor = 15.5e6
      function = pressure_ramp
    [../]
  [../]
```

After: `myProblem.i`

```
  [./Pressure]
#  apply coolant pressure on clad outer walls
    [./coolantPressure]
      boundary = '1 2 3'
      factor = 16e6
      function = pressure_ramp
    [../]
  [../]
```

Editing the Input File: Cladding

Before: `inputQuad8.i`

```
[./clad_thermal]
  type = HeatConductionMaterial
  block = clad
  thermal_conductivity = 16.0
  specific_heat = 330.0
[../]
```

After: `myProblem.i`

```
[./clad_thermal]
  type = HeatConductionMaterial
  block = clad
  thermal_conductivity = 21.5
  specific_heat = 285.0
[../]
```

Editing the Input File: Cladding Continued

Before: `inputQuad8.i`

```
  [./clad_solid_mechanics]
    type = MechZry
    block = clad
    disp_r = disp_x
    disp_z = disp_y
    temp = temp
    fast_neutron_flux = fast_neutron_flux
    fast_neutron_fluence = fast_neutron_fluence
    youngs_modulus = 7.5e10
    poissons_ratio = 0.3
    thermal_expansion = 5.0e-6
    output_iteration_info = false
    model_irradiation_growth = true
    model_thermal_expansion = false
    stress_free_temperature = 295.0
  [../]
```

After: `myProblem.i`

```
  [./clad_solid_mechanics]
    type = MechZry
    block = clad
    disp_r = disp_x
    disp_z = disp_y
    temp = temp
    fast_neutron_flux = fast_neutron_flux
    fast_neutron_fluence = fast_neutron_fluence
    youngs_modulus = 9.93e10
    poissons_ratio = 0.37
    thermal_expansion = 6.0e-6
    output_iteration_info = false
    model_irradiation_growth = true
    model_thermal_expansion = false
    stress_free_temperature = 295.0
  [../]
```

Copy Input Data Files

- The input file uses a PiecewiseBilinear function (`axial_peaking_factors`) that requires a comma separated value (csv) file. PiecewiseBilinear functions allow data lookup in a table.

  -
    ```
    > cp ../examples/2D-RZ*/peakingfactors.csv .
    ```
- The format of this csv file is as follows:

	coor_1	coor_2	coor_3	...	coor_M
time_1	factor11	factor12	factor13	...	factor1M
time_2	factor21	factor22	factor23	...	factor2M
⋮	⋮	⋮	⋮	⋱	⋮
time_N	factornN1	factorN2	factorN3	...	factorNM

Generate Smeared Pellet Mesh

- With the input file (`myProject.i`) complete and the csv file in place, all that remains is to generate the mesh file:
  - `> cp ../tools/UO2/mesh_script.sh .`
  - `> cp ../tools/UO2/mesh_script.py .`
  - `> cp ../tools/UO2/mesh_script_input.py coarse1_rz.py`
  - `> ./mesh_script.sh -i coarse1_rz.py`

Note, you'll also have to modify the burnup block and axial profile to account for the change in fuel height from 10 pellets to one pellet.

Run BISON

- We will now analyze this problem using BISON.
- We will use four processors with MPI (Message Passing Interface):
  - `> mpiexec -n 4 ../bison-opt -i myProblem.i`
- To run with a single processor:
  - `> ../bison-opt -i myProblem.i`

Postprocessing

www.inl.gov

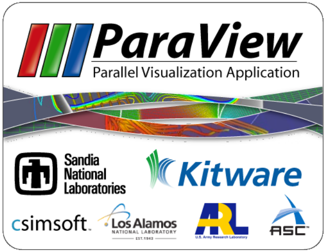
Idaho National Laboratory

Output Files

- Preferred output for MOOSE applications is ExodusII [3] binary format
- Several options for visualizing ExodusII files:
  - Paraview
    - Open-source general visualization tool
    - `http://www.paraview.org`
  - Ensight
    - Commercial general visualization tool
    - `http://www.ensight.com`
  - Peacock
    - MOOSE GUI has integrated postprocessor
    - Live update of results while model is running
    - Currently provides very basic postprocessing
  - Blot
    - Command-line visualization tool
    - Part of SEACAS suite of codes for working with Exodus files
    - Easily scripted, useful for generating x-y plots
    - `http://sourceforge.net/projects/seacas`
  - Patran
    - Commercial pre and post-processor, requires Exodus plugin
    - `http://mscsoftware.com`
  - VisIt
    - Open-source general visualization tool
    - `https://wci.llnl.gov/codes/visit`

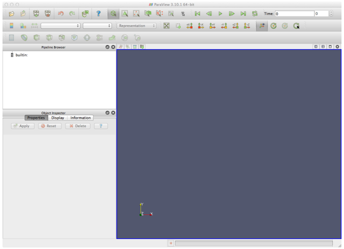
Paraview

- Open-source GUI-based visualization tool
- Provides readers for many data formats, including Exodus
- Targeted at visualization of very large data sets
  - Remote parallel rendering
  - Some behavior of the user interface driven by that emphasis.
  - Strong preference toward loading minimal data into memory.
- Thin GUI layer on top of VTK open-source visualization toolkit (Kitware).
  - Same software used for displaying graphics in Cubit
- Brief usage tutorial provided in following slides


ParaView
Parallel Visualization Application
Sandia National Laboratories
Kitware
csimsoft
Los Alamos NATIONAL LABORATORY EST. 1943
ARL
U.S. Army Research Laboratory
ASC

Opening Results File

- Option 1: GUI
  1. Click on Paraview icon, initial blank screen:

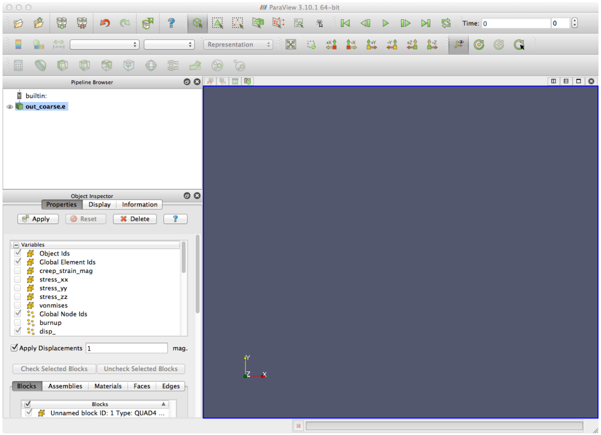


  2. File→Open, Select file, Click OK

- Option 2: Command Line
  1. Edit your `.bashrc` file to put `paraview` in your path:
     ```
     export PATH=$PATH:/Applications/ParaView\
     3.14.1.app/Contents/MacOS
     ```
  2. `paraview myfile.e`

After Opening File

After opening the file, you will likely see a blank screen because your model is still not loaded into memory:



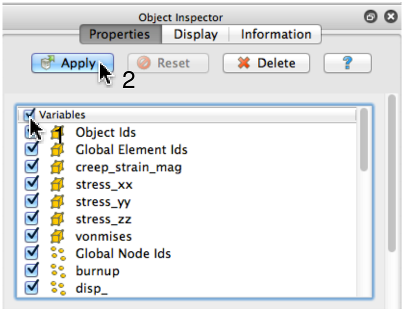
Variables
- [✓] Object Ids
- [✓] Global Element Ids
- [ ] creep_strain_mag
- [ ] stress_xx
- [ ] stress_yy
- [ ] stress_zz
- [ ] vonmises
- [✓] Global Node Ids
- [ ] burnup
- [✓] disp_

[✓] Apply Displacements 1 mag.

Blocks
- [✓] Unnamed block ID: 1 Type: QUAD4 ...

Loading Model and Variables into Memory

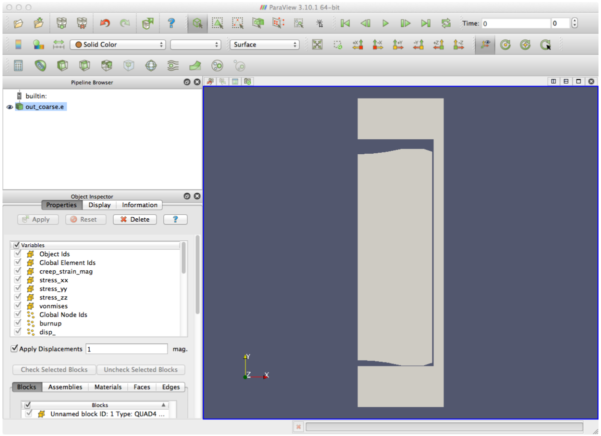
- Paraview is designed to handle extremely large data sets
- Avoids automatically performing expensive operations
  - "Apply" button must be pressed to initiate many operations
  - Minimal set of variables is loaded into memory by default
- To load model and all variables in memory:
  1. Click check box next to "Variables" to select all variables.
  2. Click "Apply" button to load the model with the selected variables.



- ✓ Object Ids
- ✓ Global Element Ids
- ✓ creep_strain_mag
- ✓ stress_xx
- ✓ stress_yy
- ✓ stress_zz
- ✓ vonmises
- ✓ Global Node Ids
- ✓ burnup
- ✓ disp_

After Opening File

- Now you should see your model loaded:



Variables
- Object Ids
- Global Element Ids
- creep_strain_mag
- stress_xx
- stress_yy
- stress_zz
- vonmises
- Global Node Ids
- burnup
- disp_

Apply Displacements 1 mag.

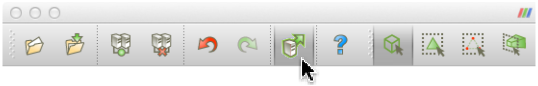
Check Selected Blocks | Uncheck Selected Blocks

Blocks | Assemblies | Materials | Faces | Edges

Blocks
- Unnamed block ID: 1 Type: QUAD4 ...

Auto Apply

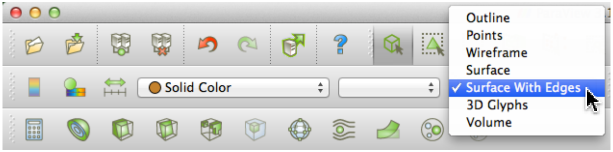
- Paraview provides option to apply parameters to changes automatically
- That option removes need to click “Apply” button
- To enable this, click on the “Apply changes to parameters automatically” toggle button in the toolbar:



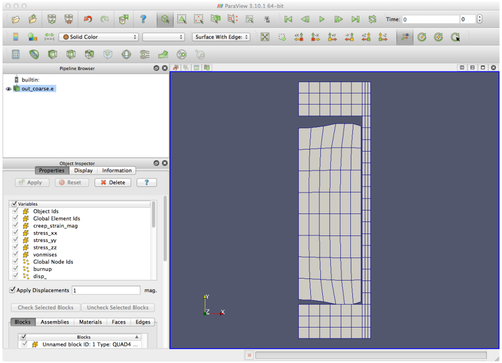
- This setting will persist the next time Paraview is launched
  - The next time you open a model, it will be loaded automatically with a minimal set of variables

Displaying Element Boundaries

- Switch display mode from “Surface” to “Surfaces with Edges”:

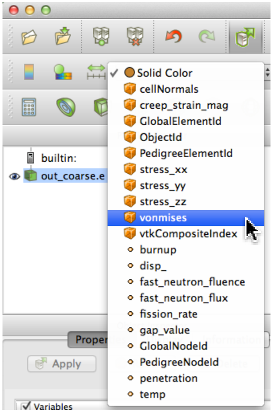


- Element boundaries are now displayed:



Contour Plots: Select Variable

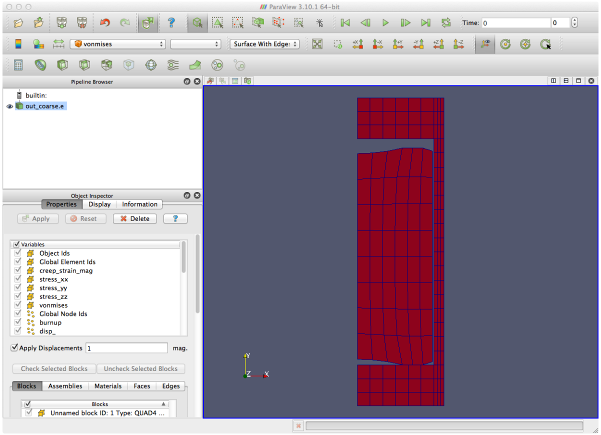
- Select “vonmises” as the variable to be used for the contour plot:



- Icons next to variable names indicate type (element or nodal)
- If a variable wasn’t loaded, it won’t be shown in this list

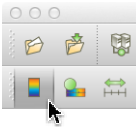
Contour Plots: Select Variable (cont.)

- You should now see the von Mises stress contour plot:

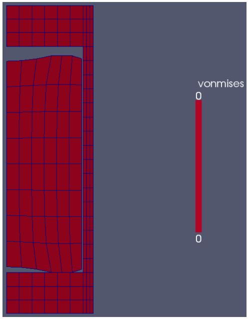


Contour Plots: Enable Legend

- Click on “Toggle Color Legend Visibility” toolbar button:

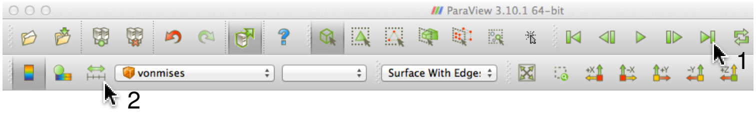


- Legend is now displayed with data range 0-0 because it has not been set

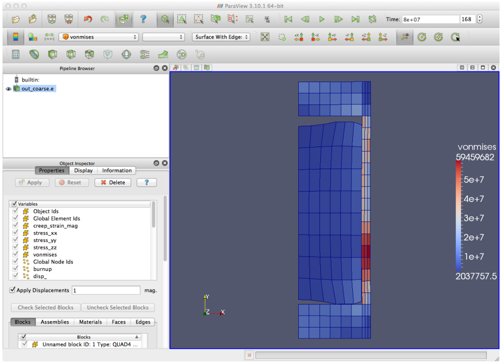


Contour Plots: Rescale to Data Range

- “Rescale to Data Range” button operates on current time step
- Go to last time step, then rescale for that time step:



- Resulting von Mises stress plot with range scaled for last timestep



Variables
- ✓ Object Ids
- ✓ Global Element Ids
- ✓ creep_strain_mag
- ✓ stress_xx
- ✓ stress_yy
- ✓ stress_zz
- ✓ vonmises
- ✓ Global Node Ids
- ✓ burnup
- ✓ disp_

✓ Apply Displacements: 1 mag.

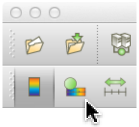
Check Selected Blocks | Uncheck Selected Blocks

Blocks | Assemblies | Materials | Faces | Edges

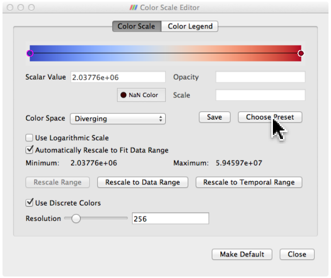
Blocks
- ✓ Unnamed block ID: 1 Type: QUAD4 ...

Contour Plots: Changing Color Map

- Click on “Edit color map” icon



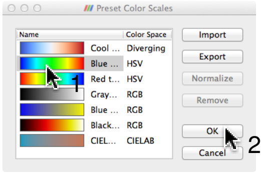
- This brings up the “Color Scale Editor” dialog box
- Click on “Choose Preset” to pick a different color map:



Scalar Value 2.03776e+06
Opacity
Scale
Color Space Diverging
☐ Use Logarithmic Scale
☑ Automatically Rescale to Fit Data Range
Minimum: 2.03776e+06 Maximum: 5.94597e+07
☑ Use Discrete Colors
Resolution 256

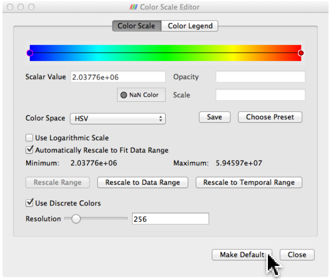
Contour Plots: Changing Color Map (cont.)

- Select desired color map in "Preset Color Scales" dialog and click "OK":


Preset Color Scales

Name	Color Space
Cool ...	Diverging
Blue ...	HSV
Red t...	HSV
Gray...	RGB
Blue ...	RGB
Black...	RGB
CIEL...	CIELAB

- You can make this the default by clicking "Make Default" in "Color Scale Editor":

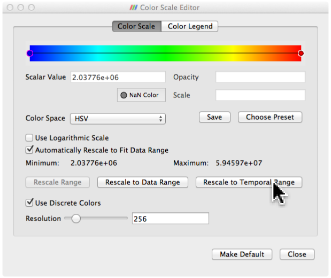

Color Scale Editor

Color Scale | Color Legend

Scalar Value 2.03776e+06
Opacity
NaN Color
Scale
Color Space HSV
Save | Choose Preset
☐ Use Logarithmic Scale
☑ Automatically Rescale to Fit Data Range
Minimum: 2.03776e+06 Maximum: 5.94597e+07
Rescale Range | Rescale to Data Range | Rescale to Temporal Range
☑ Use Discrete Colors
Resolution 256
Make Default | Close

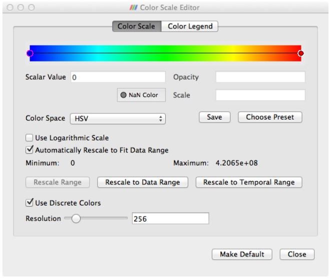
Contour Plots: Rescale to Temporal Range

- In "Color Scale Editor", option is provided to rescale contour bounds to minimum/maximum of every time step.
- Click on "Rescale to Temporal Range" button, and click "Yes" when warned about taking a long time
- Note that Minimum/Maximum have changed


Color Scale Editor

Scalar Value 2.03776e+06
Color Space HSV
☐ Use Logarithmic Scale
☑ Automatically Rescale to Fit Data Range
Minimum: 2.03776e+06 Maximum: 5.94597e+07
☑ Use Discrete Colors
Resolution 256

Before

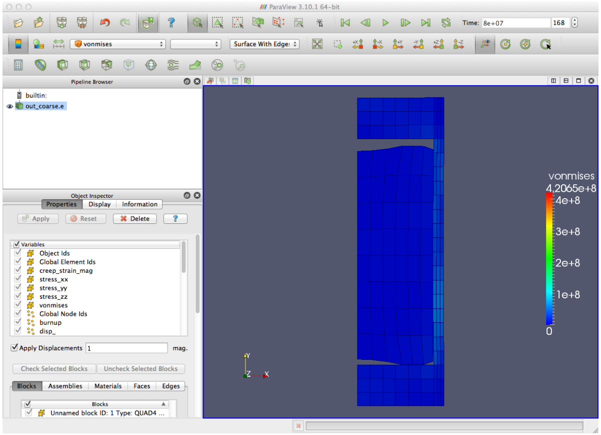

Color Scale Editor

Scalar Value 0
Color Space HSV
☐ Use Logarithmic Scale
☑ Automatically Rescale to Fit Data Range
Minimum: 0 Maximum: 4.2065e+08
☑ Use Discrete Colors
Resolution 256

After

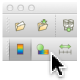
Contour Plots: Rescale to Temporal Range (cont.)

- End result when “Color Scale Editor” dialog is closed:

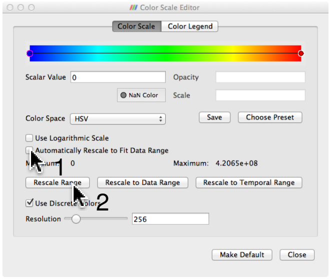


Contour Plots: Manually set Range

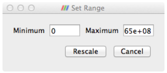
- Color scale range can also be set manually.
- Re-open “Color Scale Editor” by clicking on “Edit color map” again:



- Uncheck “Automatically Rescale to Fit Data Range” box, click on “Rescale Range”. Edit Minimum/Maximum in dialog box:


Color Scale Editor

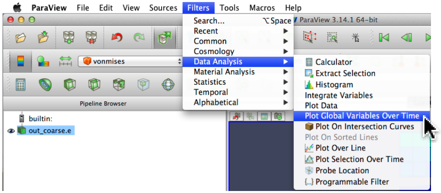
Scalar Value 0
Opacity
Scale
Color Space HSV
☐ Use Logarithmic Scale
☐ Automatically Rescale to Fit Data Range
Minimum: 0 Maximum: 4.2065e+08
☑ Use Discrete Colors
Resolution 256


Set Range

Minimum 0 Maximum 65e+08

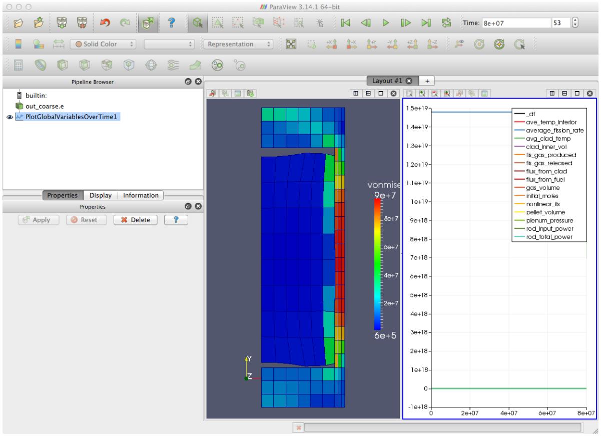
X-Y Plots

- Paraview can generate several types of x-y plots, including:
  - Global variables over time
  - Variables at selected nodes/elements over time
  - Spatial variation of variables over a line
- Procedure to generate all of these will be demonstrated
- Plotting global variables:
- Filters→Data Analysis→Plot Global Variables Over Time



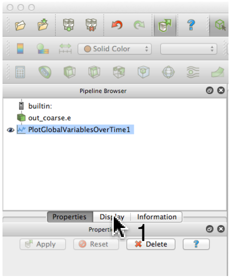
XY-Plots: Global Variables

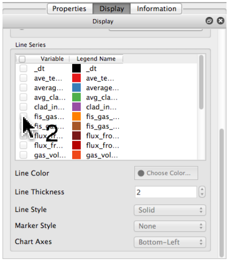
- The result is a split view showing both the model and an x-y plot with all global variables:



XY-Plots: Global Variables (cont.)

- To limit the set of variables plotted, click on the “Display” tab:
  1. Click on the “Display” tab:
  2. Select variables to plot from the list
     - Check the box to the left of “Variable” to toggle all variables
     - Select desired variables (“fis_gas_produced” and “fis_gas_released” in our case)



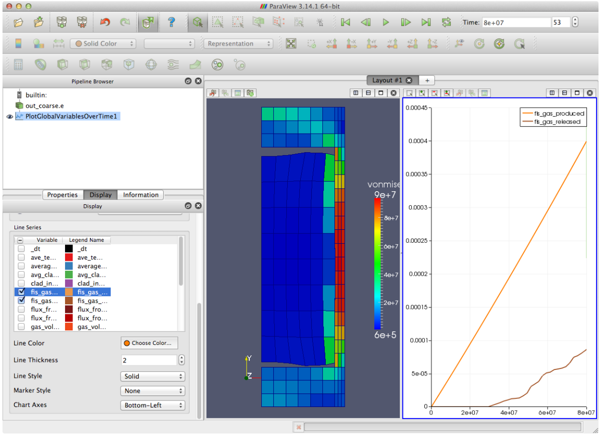


Variable	Legend Name
_dt	■ _dt
ave_te...	🟥 ave_te...
averag...	🟦 average...
avg_cla...	🟩 avg_cla...
clad_in...	🟪 clad_in...
fis_gas...	🟧 fis_gas_...
fis_g...	🟫 fis_gas_...
flux_f...	🟥 flux_fro...
flux_fr...	🟥 flux_fro...
gas_vol...	🟧 gas_vol...

Line Color: Choose Color...
Line Thickness: 2
Line Style: Solid
Marker Style: None
Chart Axes: Bottom-Left

XY-Plots: Global Variables (cont.)

- Now only fission gas variables are shown:



Display

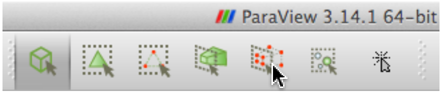
Line Series

	Variable		Legend Name
☐	_dt	■	_dt
☐	ave_te...	🟥	ave_te...
☐	averag...	🟦	average...
☐	avg_cla...	🟩	avg_cla...
☐	clad_in...	🟪	clad_in...
☑	fis_gas...	🟧	fis_gas_...
☑	fis_gas...	🟫	fis_gas_...
☐	flux_fr...	🟥	flux_fro...
☐	flux_fr...	🟥	flux_fro...
☐	gas_vol...	🟧	gas_vol...

Line Color: Choose Color...
Line Thickness: 2
Line Style: Solid
Marker Style: None
Chart Axes: Bottom-Left

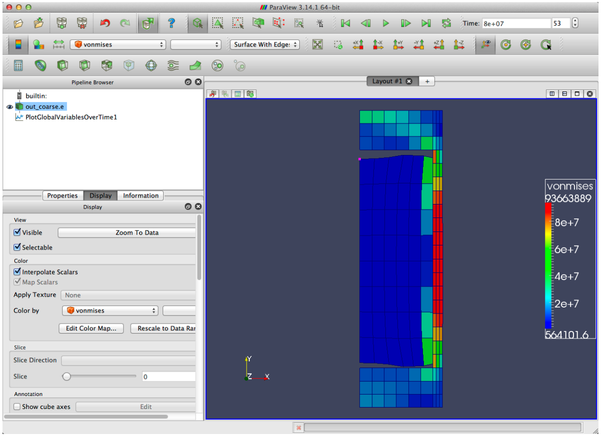
X-Y Plots: Ploting Data from a Selection

- Select the button called “Select Points Through”



XY-Plots: Selection (cont.)

- Now select the node of interest:



Display

View
☑ Visible — Zoom To Data
☑ Selectable

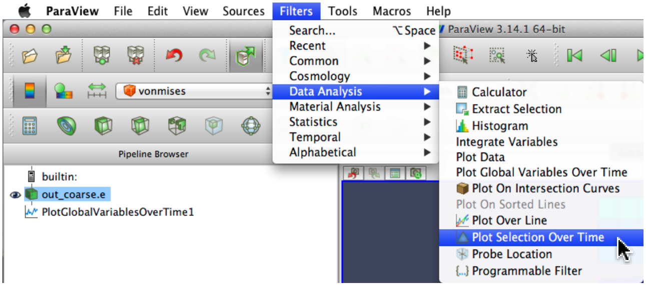
Color
☑ Interpolate Scalars
☑ Map Scalars
Apply Texture: None
Color by: vonmises
Edit Color Map... | Rescale to Data Range

Slice
Slice Direction
Slice: 0

Annotation
☐ Show cube axes — Edit

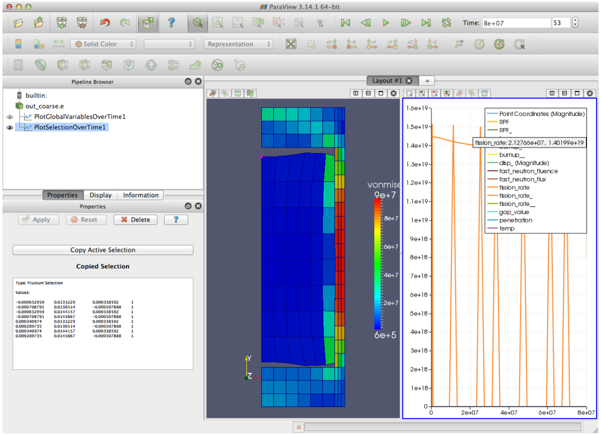
XY-Plots: Selection (cont.)

- Use Filters again:
- Filters→Data Analysis→Plot Selection Over Time



XY-Plots: Selection (cont.)

- Similar to what we saw in the global variables example, the result is a split view showing both the model and an x-y plot with all the field variables.

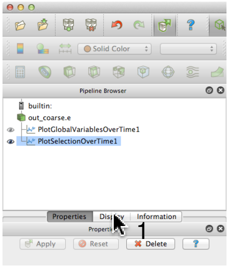


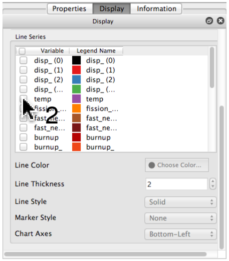
Copied Selection

```
Type: Frustum Selection

Values:

-0.000632959    0.0135229    0.000338592     1
-0.000708791    0.0136514    -0.000507888    1
-0.000632959    0.0144157    0.000338592     1
-0.000708791    0.0145667    -0.000507888    1
0.000340974     0.0135229    0.000338592     1
0.000289735     0.0136514    -0.000507888    1
0.000340974     0.0144157    0.000338592     1
0.000289735     0.0145667    -0.000507888    1
```

XY-Plots: Selection (cont.)

- To limit the set of variables plotted, click on the "Display" tab:
  1. Click on the "Display" tab:
  2. Select variables to plot from the list
     - Check the box to the left of "Variable" to toggle all variables
     - Select desired variables ("temp" in this example)



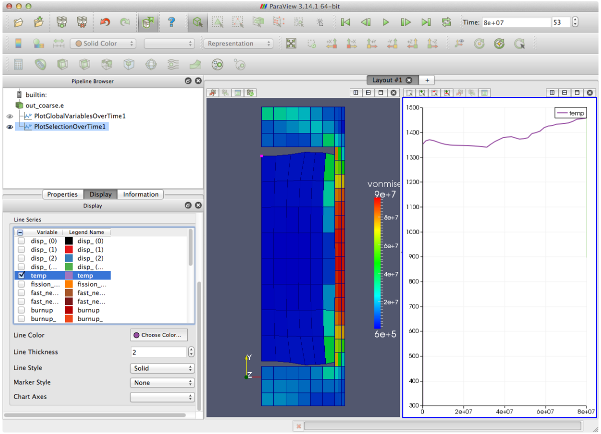


	Variable		Legend Name
☐	disp_ (0)	■	disp_ (0)
☐	disp_ (1)	🟥	disp_ (1)
☐	disp_ (2)	🟦	disp_ (2)
☐	disp_ (...	🟩	disp_ (...
☐	temp	🟪	temp
☐	fission_...	🟧	fission_...
☐	fast_ne...	🟫	fast_ne...
☐	fast_ne...	🟫	fast_ne...
☐	burnup	🟥	burnup
☐	burnup_	🟥	burnup_

Line Color: Choose Color...
Line Thickness: 2
Line Style: Solid
Marker Style: None
Chart Axes: Bottom-Left

XY-Plots: Selection (cont.)

- Now, only the temperature at the node you selected is plotted:



Display

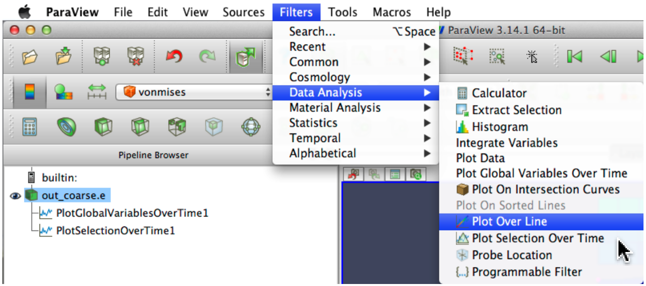
Line Series

Variable	Legend Name
☐ disp_ (0)	■ disp_ (0)
☐ disp_ (1)	🟥 disp_ (1)
☐ disp_ (2)	🟦 disp_ (2)
☐ disp_ (...	🟩 disp_ (...
☑ temp	🟪 temp
☐ fission_...	🟧 fission_...
☐ fast_ne...	🟫 fast_ne...
☐ fast_ne...	🟥 fast_ne...
☐ burnup	🟥 burnup
☐ burnup_	🟥 burnup_

Line Color: Choose Color...
Line Thickness: 2
Line Style: Solid
Marker Style: None
Chart Axes:

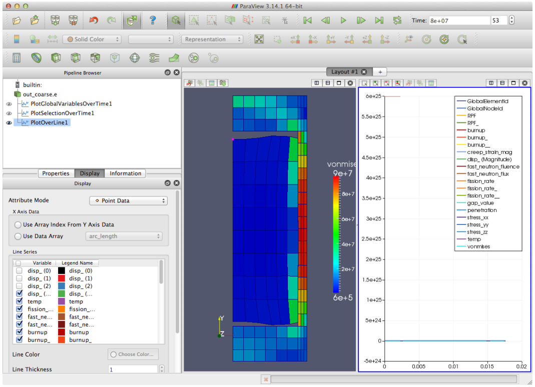
XY-Plots: Line Plots

- Line plots show field data on a line defined by you. This is a convenient way to plot the diameter of the clad as a function of axial position, for example.
- To make a Line Plot, yet again, use Filters (are you sensing a theme?):
- Filters→Data Analysis→Plot Line Over Line



XY-Plots: Line Plots (cont.)

- Similar to what we saw in the previous examples, the result is a split view showing both the model and an x-y *line plot* with all the field variables plotted on some line.
- At this point, you haven't defined the position of the line. What you're seeing here are the magnitudes of the field variables at Time = 8e+07 plotted against positions along paraview's default line position.



Pipeline Browser

- builtin:
- out_coarse.e
  - PlotGlobalVariablesOverTime1
  - PlotSelectionOverTime1
  - PlotOverLine1

Properties | Display | Information

Display

Attribute Mode: Point Data

X Axis Data

- ○ Use Array Index From Y Axis Data
- ○ Use Data Array: arc_length

Line Series

	Variable		Legend Name
☐	disp_ (0)	■	disp_ (0)
☐	disp_ (1)	🟥	disp_ (1)
☐	disp_ (2)	🟦	disp_ (2)
✓	disp_ (...	🟩	disp_ (...
✓	temp	🟪	temp
✓	fission_...	🟧	fission_...
✓	fast_ne...	🟫	fast_ne...
✓	fast_ne...	🟫	fast_ne...
✓	burnup	🟥	burnup
✓	burnup_	🟧	burnup_

Line Color: Choose Color...

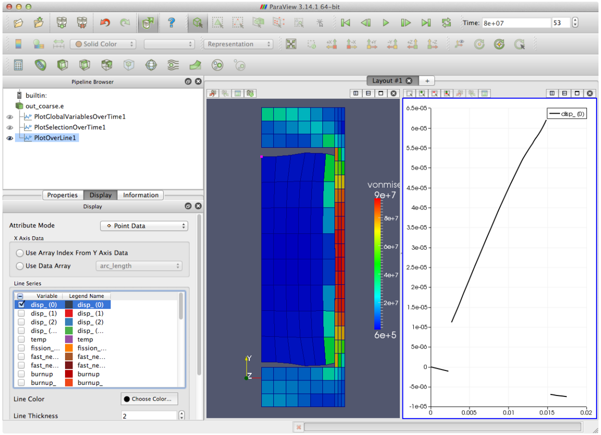
Line Thickness: 1

Legend:

- GlobalElementId
- GlobalNodeId
- RPF
- RPF_
- burnup
- burnup_
- burnup__
- creep_strain_mag
- disp_ (Magnitude)
- fast_neutron_fluence
- fast_neutron_flux
- fission_rate
- fission_rate_
- fission_rate__
- gap_value
- penetration
- stress_xx
- stress_yy
- stress_zz
- temp
- vonmises

XY-Plots: Line Plots (cont.)

- Check the box to the left of “Variable” to toggle all variables
- Select disp_0 (radial displacement)



Display

Attribute Mode: Point Data

X Axis Data
- Use Array Index From Y Axis Data
- Use Data Array: arc_length

Line Series

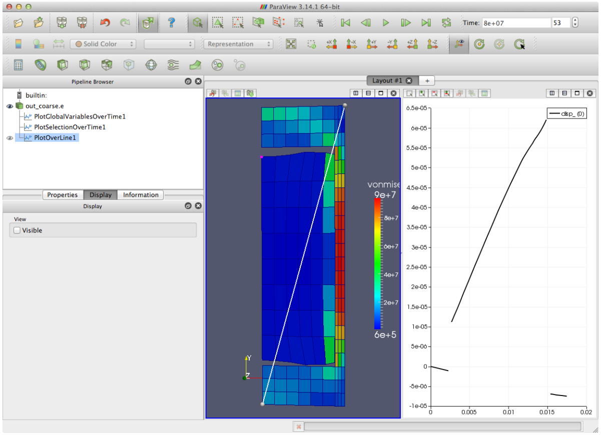
	Variable		Legend Name
✓	disp_ (0)	■	disp_ (0)
☐	disp_ (1)	🟥	disp_ (1)
☐	disp_ (2)	🟦	disp_ (2)
☐	disp_ (...	🟩	disp_ (...
☐	temp	🟪	temp
☐	fission_...	🟧	fission_...
☐	fast_ne...	🟫	fast_ne...
☐	fast_ne...	🟫	fast_ne...
☐	burnup	🟥	burnup
☐	burnup_	🟥	burnup_

Line Color: ● Choose Color...

Line Thickness: 2

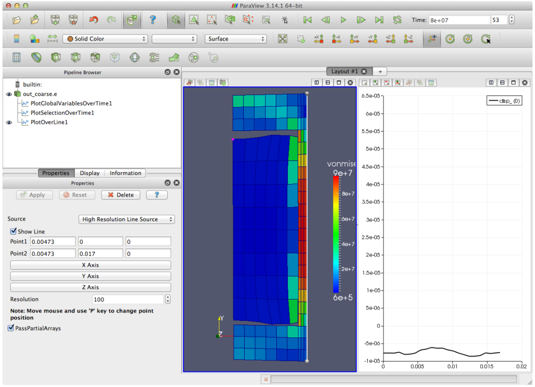
XY-Plots: Line Plots (cont.)

- Click on the window that contains the model image. Now, you can see the location of the line.
- In the right-hand window is a line plot of the radial displacement along that line.



XY-Plots: Line Plots (cont.)

- You can adjust the position of that line by clicking on the “Properties” tab and changing the values of the two points that define the line.
- We’ve selected points coincident with the lower right, and upper right points of the clad.
- The line plot now shows the radial displacement of the clad exterior as a function of axial position



builtin:
out_coarse.e
PlotGlobalVariablesOverTime1
PlotSelectionOverTime1
PlotOverLine1

Properties | Display | Information

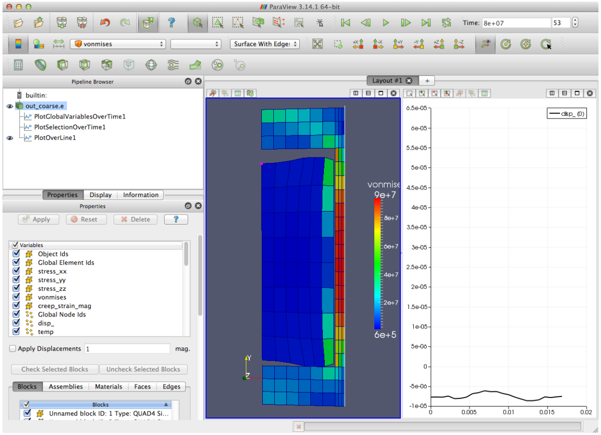
Properties

Apply | Reset | Delete | ?

Source: High Resolution Line Source
☑ Show Line
Point1: 0.00473 | 0 | 0
Point2: 0.00473 | 0.017 | 0
X Axis
Y Axis
Z Axis
Resolution: 100
Note: Move mouse and use 'P' key to change point position
☑ PassPartialArrays

XY-Plots: Line Plots (cont.)

- By selecting out_coarse.e in the Pipeline Browser window (upper left), then toggling the “Apply Displacements” box, the line plot switches from displaced mesh coordinates to original coordinates.



ParaView 3.14.1 64-bit

Time: 8e+07 53

vonmises

Surface With Edges

Pipeline Browser

- builtin:
  - out_coarse.e
    - PlotGlobalVariablesOverTime1
    - PlotSelectionOverTime1
    - PlotOverLine1

Properties | Display | Information

Apply | Reset | Delete | ?

- [✓] Variables
  - [✓] Object Ids
  - [✓] Global Element Ids
  - [✓] stress_xx
  - [✓] stress_yy
  - [✓] stress_zz
  - [✓] vonmises
  - [✓] creep_strain_mag
  - [✓] Global Node Ids
  - [✓] disp_
  - [✓] temp

[ ] Apply Displacements 1 mag.

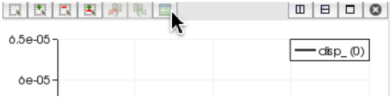
Check Selected Blocks | Uncheck Selected Blocks

Blocks | Assemblies | Materials | Faces | Edges

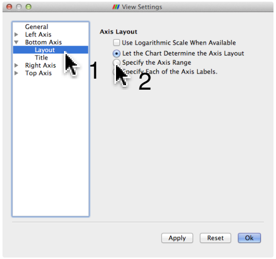
- [✓] Blocks
  - [✓] Unnamed block ID: 1 Type: QUAD4 Si...

XY-Plots: Modify Plot Axis Properties

- To modify the axes and other properties of x-y plots, click on the “Edit View Options” button at the top of the x-y plot pane:

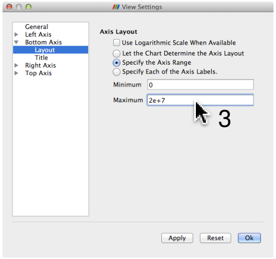


- To modify the x axis range, click on Bottom Axis→Layout, then on Specify the Axis Range, and enter new values



Axis Layout

- [ ] Use Logarithmic Scale When Available
- [✓ Let the Chart Determine the Axis Layout]
- Specify the Axis Range
- Specify Each of the Axis Labels.



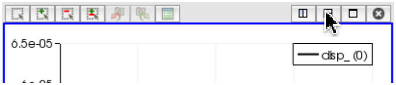
Axis Layout

- [ ] Use Logarithmic Scale When Available
- Let the Chart Determine the Axis Layout
- [✓ Specify the Axis Range]
- Specify Each of the Axis Labels.

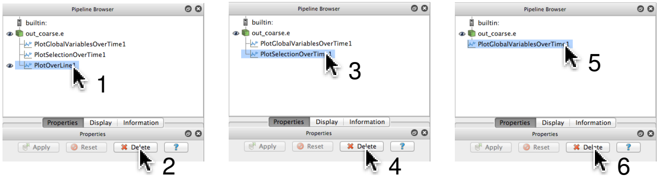
Minimum	0
Maximum	2e+7

More on Selections: Selection Inspector

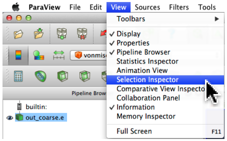
- First some cleanup: Close line chart view



- Delete plotting filters one at a time

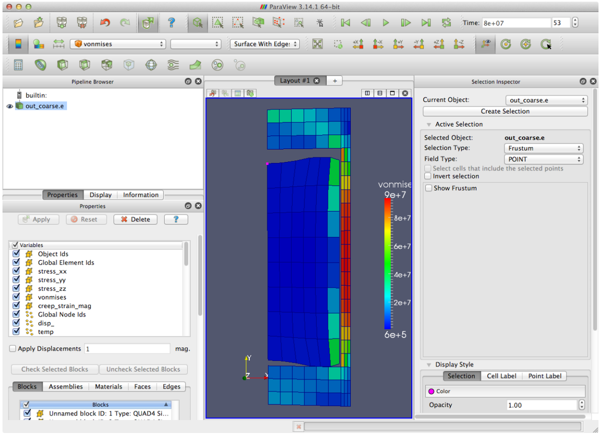


- Now, turn on the selection inspector. Click on View→Selection Inspector



Selection Inspector

- The Selection Inspector is now visible on the right side of the screen.
  - Provides tools for selecting sets of nodes and elements and controlling how they are displayed



Time: 8e+07 53

Apply Displacements 1

Current Object: out_coarse.e

Selected Object: out_coarse.e

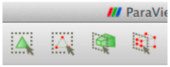
Selection Type: Frustum

Field Type: POINT

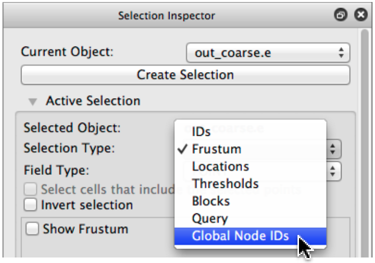
Opacity 1.00

Selection Inspector: Selection Types

- Multiple ways to select nodes/elements:
  - Selections by location
    - Elements and nodes can be selected on the surface or through the volume
    - Use the buttons on the top of the screen (shown previously):

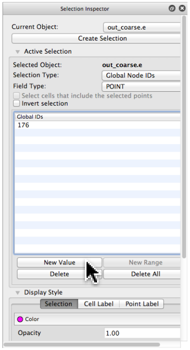


  - Selections by node or element ID
    - Useful if you know the ID of the node/element
    - Select the “Global Node IDs” option. These are the IDs used in the Exodus file. Note: Do not select the “IDs” option. It is an internal Paraview numbering scheme.



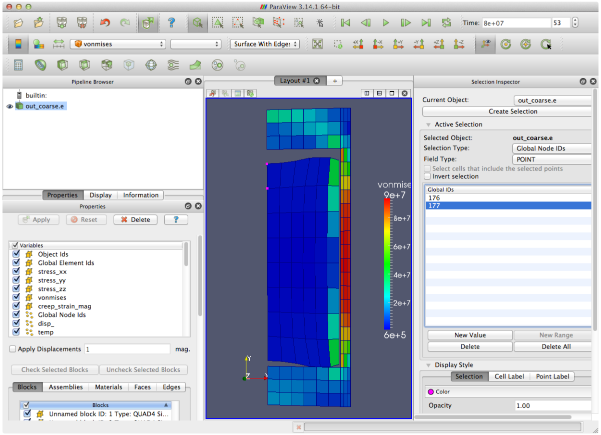
Selections: Selecting by Global ID

- The Selection Inspector allows the user to manipulate the list of global IDs
- Field Type can be POINT (node) or CELL (element)
- Nodes can be added or removed from the selection list
- Click “New Value” and enter the ID of another node (177 in our case):


Current Object: out_coarse.e
Selected Object: out_coarse.e
Selection Type: Global Node IDs
Field Type: POINT
Global IDs
176

Selections: Selecting by Global ID (cont.)

- Now nodes 176 and 177 are both included in the selection:



ParaView 3.14.1 64-bit

Time: 8e+07 53

vonmises

Surface With Edges

Pipeline Browser

builtin:

out_coarse.e

Properties Display Information

Apply Reset Delete

Variables
- ✓ Object Ids
- ✓ Global Element Ids
- ✓ stress_xx
- ✓ stress_yy
- ✓ stress_zz
- ✓ vonmises
- ✓ creep_strain_mag
- ✓ Global Node Ids
- ✓ disp_
- ✓ temp

Apply Displacements 1 mag.

Check Selected Blocks Uncheck Selected Blocks

Blocks Assemblies Materials Faces Edges

✓ Unnamed block ID: 1 Type: QUAD4 Si...

Selection Inspector

Current Object: out_coarse.e

Create Selection

Active Selection

Selected Object: out_coarse.e

Selection Type: Global Node IDs

Field Type: POINT

Select cells that include the selected points

Invert selection

Global IDs
176
177

New Value New Range

Delete Delete All

Display Style

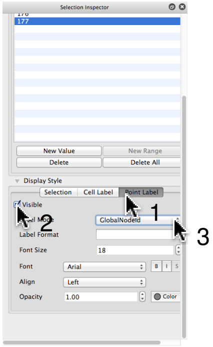
Selection Cell Label Point Label

● Color

Opacity 1.00

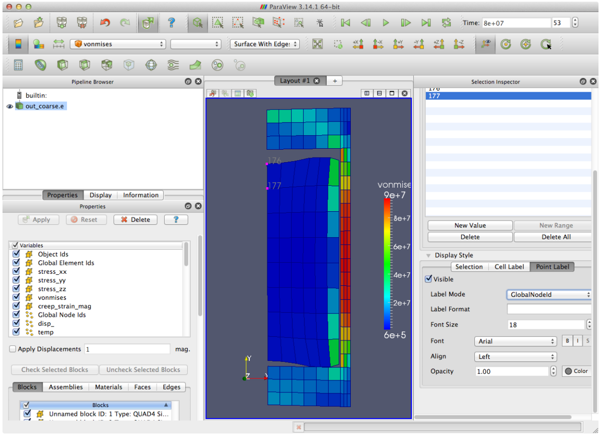
Selections: Labels on selected nodes/elements

- Selected nodes and elements can be labeled by ID or any variable
- To enable node ID labels:
  - Scroll to the bottom of the Selection Inspector
  - Click on the “Point Label” tab under “Display Style”
  - Click on the “Visible” checkbox
  - Make sure “GlobalNodeId” is selected for “Label Mode”



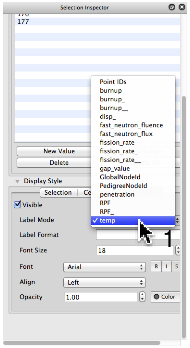
Selections: Labels on selected nodes/elements (cont.)

- Selected nodes are now labled by Exodus ID:

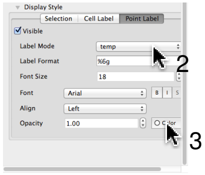


Selections: Labeling by variable value

- To label by temperature, select "temp" from the list of variables
- To improve formatting, use C format specifier (try `%6g`) in "Label Format"
- Change the color to white to improve visibility



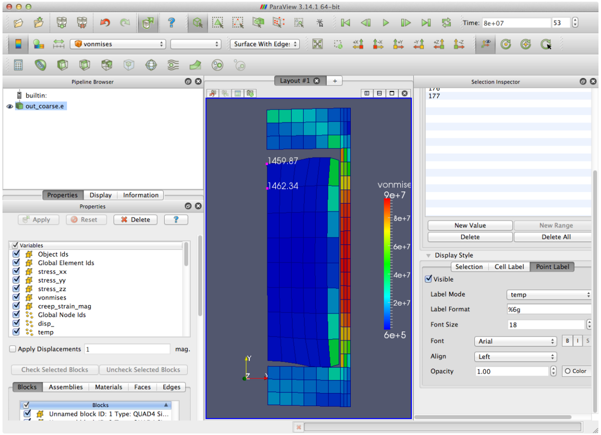
✓ Visible
Label Mode: temp
Label Format:
Font Size: 18
Font: Arial
Align: Left
Opacity: 1.00



✓ Visible
Label Mode: temp
Label Format: %6g
Font Size: 18
Font: Arial
Align: Left
Opacity: 1.00

Selections: Labeling by variable value (cont.)

- Nodal temperatures are now labeled and formatted:



Time: 8e+07 53

vonmises

Surface With Edge:

Variables: Object Ids, Global Element Ids, stress_xx, stress_yy, stress_zz, vonmises, creep_strain_mag, Global Node Ids, disp_, temp

Apply Displacements 1 mag.

Visible ✓

Label Mode: temp

Label Format: %6g

Font Size: 18

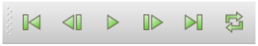
Font: Arial

Align: Left

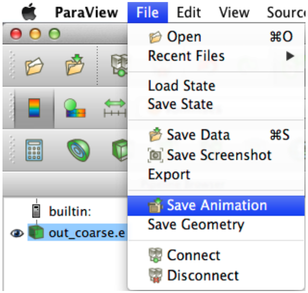
Opacity: 1.00

Animations

- Within Paraview, the model can be animated using the toolbar buttons:

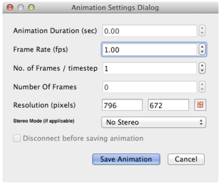


- Animations can also be saved by selecting File→Save Animation:



Animations: Saving animation

- The dialog box for saving animations offers a control to change the frame rate, but it does not work (as of version 3.14.1):



Animation Duration (sec)	0.00
Frame Rate (fps)	1.00
No. of Frames / timestep	1
Number Of Frames	0
Resolution (pixels)	796 672
Stereo Mode (if applicable)	No Stereo

Disconnect before saving animation

- Once you click on “Save Animation” you will be prompted to enter a file name and type.
- Options are AVI, JPEG, TIFF, and PNG
- AVI option allows you to directly save animation, but with limited control

Animations: Combining images into movie

- For more control and higher quality, save as image (PNG recommended) to create a sequence of images
  - Images are named as: basename.0001.png, basename.0002.png, ...
- Images can be combined together into a movie using a number of tools
- `ffmpeg` is an open-source command-line tool for doing this:
- To create a .mov file:

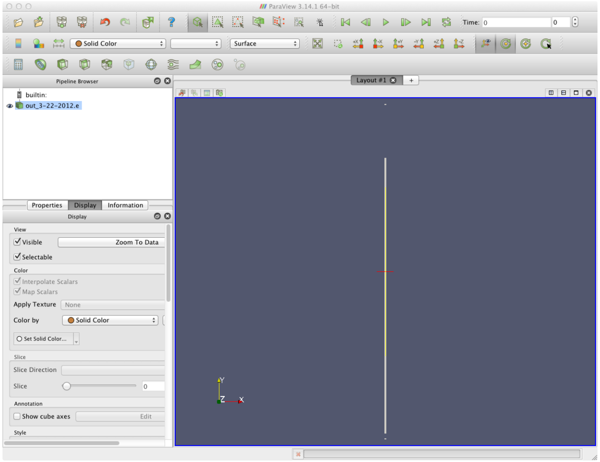
  ```
  ffmpeg -r 5 -i basename.%4d.png basename.mov
  ```

  where:

  `-r` controls the framerate
  `-i` specifies png file names (note C formatting for name sequence)
  `basnemane.mov` specifies the output file name

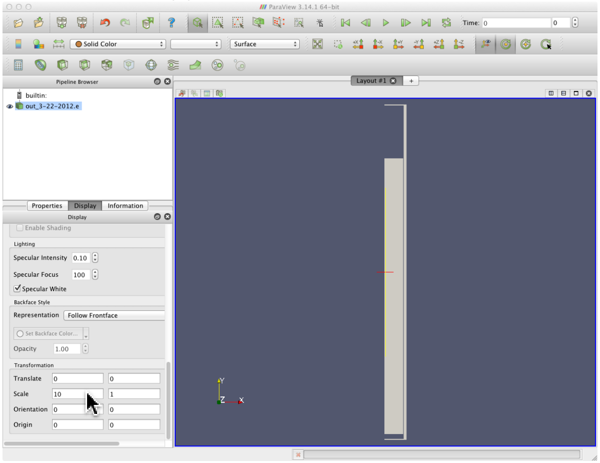
Scaling the model in one direction

- Often, fuel pin models are very long compared to the diameter.
- This can be visualized better by scaling the image in the radial direction.
- The following fuel pin model will be used to demonstrate this:



Scaling the model in one direction (cont.)

- Select the Display tab and scroll down to the Transformation section, where you can translate, scale, orient (rotate), or change the origin.
- Change the Scale section from 1 1 to 10 1 as shown
- This magnifies the radial (x) direction so that the model easier to see:



Display

Enable Shading

Lighting

Specular Intensity 0.10

Specular Focus 100

☑ Specular White

Backface Style

Representation Follow Frontface

Set Backface Color...

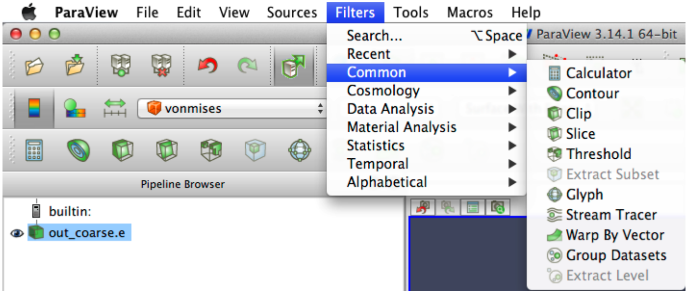
Opacity 1.00

Transformation

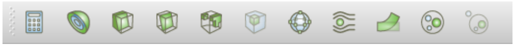
Translate	0	0
Scale	10	1
Orientation	0	0
Origin	0	0

Filters

- Paraview uses filters to control how data is displayed.
- Filters can be applied to other filters, and are shown in the Pipeline Browser
- Filters are applied to the item currently selected in the Pipeline Browser by picking the desired filter from the “Filters” menu:



- Alternatively, common filters can be accessed through the toolbar:

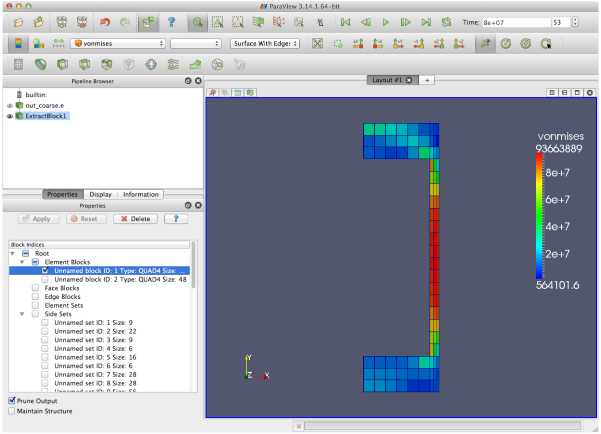


Filter Demonstration

- An example appliction of filters will be shown in the following slides
- Filters will be used to extract Blocks 1 (cladding) and 2 (fuel)
- Different filters will be applied to these extracted blocks to display them in different ways

Filters: Extracting Block 1

- Use the “Extract Block” Filter to display blocks in different ways.
- Select the model (out_coarse.e) in the Pipeline Browser, then select Filters→Alphabetical→Extract Block
- Under Properties for that filter, select Block 1 to display only that block:



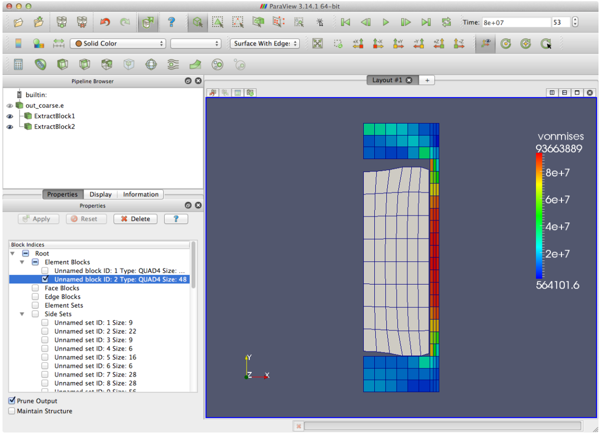
Pipeline Browser:
- builtin:
- out_coarse.e
- ExtractBlock1

Block Indices
- Root
  - Element Blocks
    - ☑ Unnamed block ID: 1 Type: QUAD4 Size: ...
    - ☐ Unnamed block ID: 2 Type: QUAD4 Size: 48
  - ☐ Face Blocks
  - ☐ Edge Blocks
  - ☐ Element Sets
  - ☐ Side Sets
    - ☐ Unnamed set ID: 1 Size: 9
    - ☐ Unnamed set ID: 2 Size: 22
    - ☐ Unnamed set ID: 3 Size: 9
    - ☐ Unnamed set ID: 4 Size: 6
    - ☐ Unnamed set ID: 5 Size: 16
    - ☐ Unnamed set ID: 6 Size: 6
    - ☐ Unnamed set ID: 7 Size: 28
    - ☐ Unnamed set ID: 8 Size: 28
    - ☐ Unnamed set ID: 9 Size: 56

☑ Prune Output
☐ Maintain Structure

Filters: Extracting Block 2

- Select the model (out_coarse.e) in the Pipeline Browser, then apply “Extract Block” again
- This time, select Block 2
- Now blocks 1 and 2 are both displayed, but with different properties:



Time: 8e+07 53

Solid Color

Surface With Edge:

builtin:
out_coarse.e
ExtractBlock1
ExtractBlock2

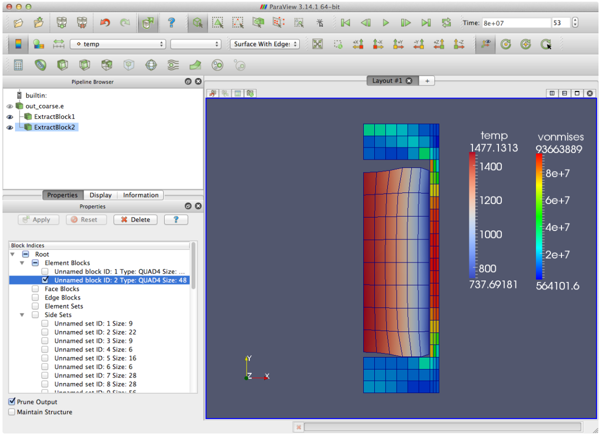
Block Indices
Root
Element Blocks
Unnamed block ID: 1 Type: QUAD4 Size: ...
☑ Unnamed block ID: 2 Type: QUAD4 Size: 48
Face Blocks
Edge Blocks
Element Sets
Side Sets
Unnamed set ID: 1 Size: 9
Unnamed set ID: 2 Size: 22
Unnamed set ID: 3 Size: 9
Unnamed set ID: 4 Size: 6
Unnamed set ID: 5 Size: 16
Unnamed set ID: 6 Size: 6
Unnamed set ID: 7 Size: 28
Unnamed set ID: 8 Size: 28

☑ Prune Output
Maintain Structure

vonmises 93663889 … 564101.6

Filters: Changing display properties in Block 2

- Select the “Extract Block 2” filter in the Pipeline Browser
- Select “temp” and edit the color map, picking a different color scale type
- Now separate variables are shown on the two blocks with different scales:


ParaView 3.14.1 64-bit

Time: 8e+07 53

temp

Surface With Edges

Pipeline Browser

builtin:
out_coarse.e
ExtractBlock1
ExtractBlock2

Properties | Display | Information

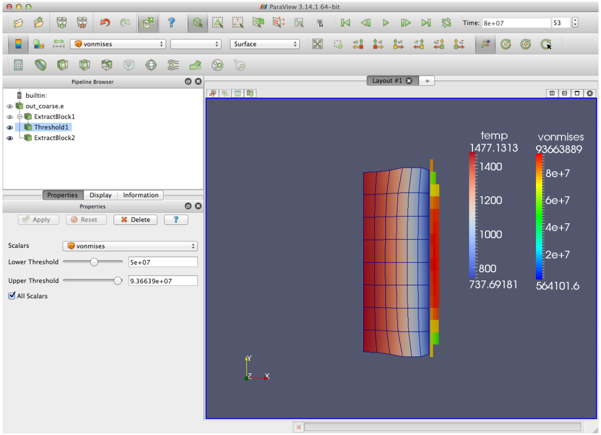
Apply Reset Delete ?

Block Indices
Root
Element Blocks
Unnamed block ID: 1 Type: QUAD4 Size: ...
Unnamed block ID: 2 Type: QUAD4 Size: 48
Face Blocks
Edge Blocks
Element Sets
Side Sets
Unnamed set ID: 1 Size: 9
Unnamed set ID: 2 Size: 22
Unnamed set ID: 3 Size: 9
Unnamed set ID: 4 Size: 6
Unnamed set ID: 5 Size: 16
Unnamed set ID: 6 Size: 6
Unnamed set ID: 7 Size: 28
Unnamed set ID: 8 Size: 28

Prune Output
Maintain Structure

Filters: Threshold filter

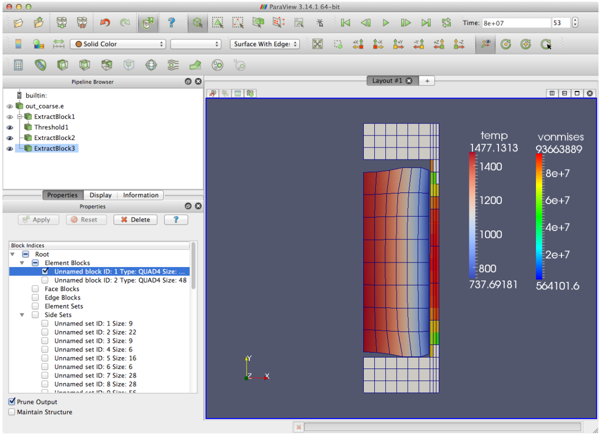
- The “Threshold” filter displays elements based on the value of a variable
- Select “ExtractBlock1”, then Filters→Common→Threshold
- Under the filter properties, select “vonmises” and provide a range
- Now only elements with high von Mises stress in the clad block are shown:



Scalars: vonmises
Lower Threshold: 5e+07
Upper Threshold: 9.36639e+07
☑ All Scalars

Filters: Extract Block 1 (again)

- Now display the rest of the elements in Block 1 with a wireframe
- Select the model (out_coarse.e) in the Pipeline Browser, then apply “Extract Block” again
- Under the filter properties, select Block 1
- “ExtractBlock1” and “ExtractBlock3” both display Block 1



Pipeline Browser

- builtin:
  - out_coarse.e
    - ExtractBlock1
      - Threshold1
    - ExtractBlock2
    - ExtractBlock3

Properties

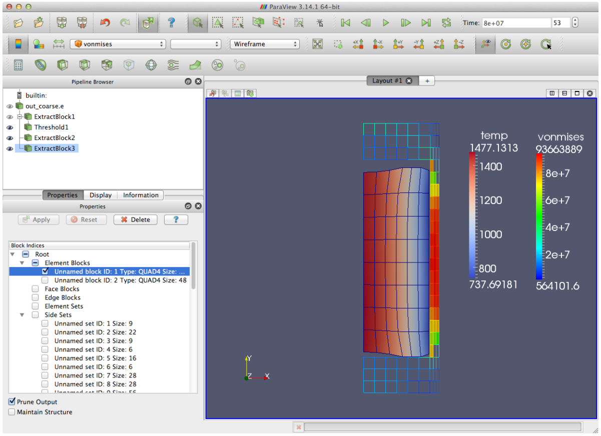
Block Indices

- Root
  - Element Blocks
    - [✓] Unnamed block ID: 1 Type: QUAD4 Size: ...
    - [ ] Unnamed block ID: 2 Type: QUAD4 Size: 48
  - [ ] Face Blocks
  - [ ] Edge Blocks
  - [ ] Element Sets
  - [ ] Side Sets
    - [ ] Unnamed set ID: 1 Size: 9
    - [ ] Unnamed set ID: 2 Size: 22
    - [ ] Unnamed set ID: 3 Size: 9
    - [ ] Unnamed set ID: 4 Size: 6
    - [ ] Unnamed set ID: 5 Size: 16
    - [ ] Unnamed set ID: 6 Size: 6
    - [ ] Unnamed set ID: 7 Size: 28
    - [ ] Unnamed set ID: 8 Size: 28

[✓] Prune Output
[ ] Maintain Structure

Filters: Display Block 1 as wireframe

- Select “ExtractBlock3” in the Pipeline Browser
- Select the “vonmises” variable
- Choose the Wireframe display mode

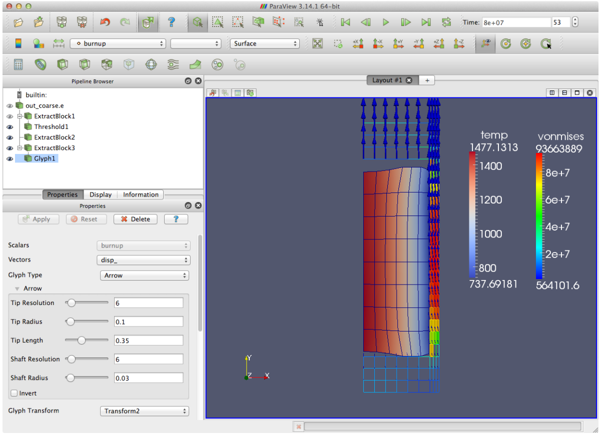


Block Indices
Root
Element Blocks
☑ Unnamed block ID: 1 Type: QUAD4 Size: ...
☐ Unnamed block ID: 2 Type: QUAD4 Size: 48
Face Blocks
Edge Blocks
Element Sets
Side Sets
Unnamed set ID: 1 Size: 9
Unnamed set ID: 2 Size: 22
Unnamed set ID: 3 Size: 9
Unnamed set ID: 4 Size: 6
Unnamed set ID: 5 Size: 16
Unnamed set ID: 6 Size: 6
Unnamed set ID: 7 Size: 28
Unnamed set ID: 8 Size: 28

☑ Prune Output
☐ Maintain Structure

Filters: Use Glyph filter to display arrows

- The Glyph filter can display a variety of symbols at nodes
- The symbols can be scaled based on variables
- Select “ExtractBlock3” in the Pipeline Browser
- Select the Arrow glyph type, and choose the disp_ vector variable



ParaView 3.14.1 64-bit

Time: 8e+07 53

burnup | Surface

Pipeline Browser

- builtin:
  - out_coarse.e
    - ExtractBlock1
      - Threshold1
    - ExtractBlock2
    - ExtractBlock3
      - Glyph1

Properties | Display | Information

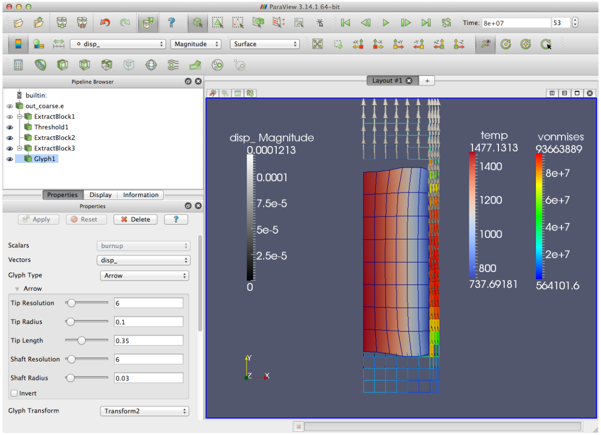
Properties

Apply | Reset | Delete | ?

Scalars	burnup
Vectors	disp_
Glyph Type	Arrow
Arrow	
Tip Resolution	6
Tip Radius	0.1
Tip Length	0.35
Shaft Resolution	6
Shaft Radius	0.03
☐ Invert	
Glyph Transform	Transform2

Filters: Color scale for arrows

- Select the “Glyph1” filter in the Pipeline Browser
- Select disp_ and Magnitude for the contour plot
- Display the legend and pick a different color scale type



Pipeline Browser

- builtin:
  - out_coarse.e
    - ExtractBlock1
      - Threshold1
    - ExtractBlock2
    - ExtractBlock3
      - Glyph1

Properties

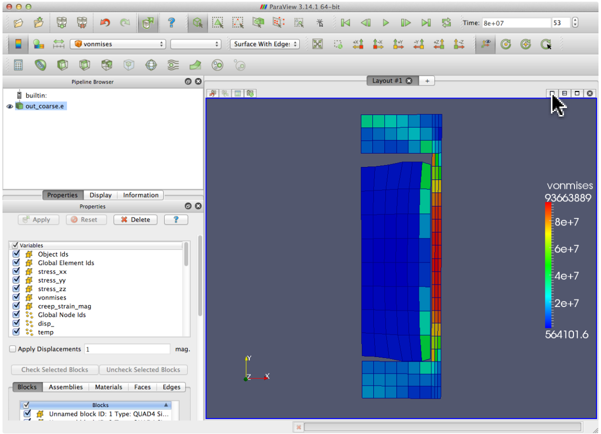
Scalars	burnup
Vectors	disp_
Glyph Type	Arrow

Arrow

Tip Resolution	6
Tip Radius	0.1
Tip Length	0.35
Shaft Resolution	6
Shaft Radius	0.03
Invert	☐
Glyph Transform	Transform2

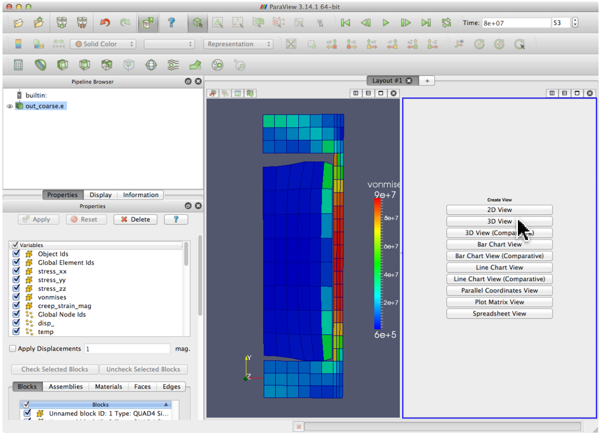
Linking Displays

- We can use the multiple windows feature to open two windows, each containing a results file, and link the two displays (or windows) together. Linking allows translation, zoom, and rotation operations in one display to occur in the other display.
- Start by selecting the vertically split window button as shown:



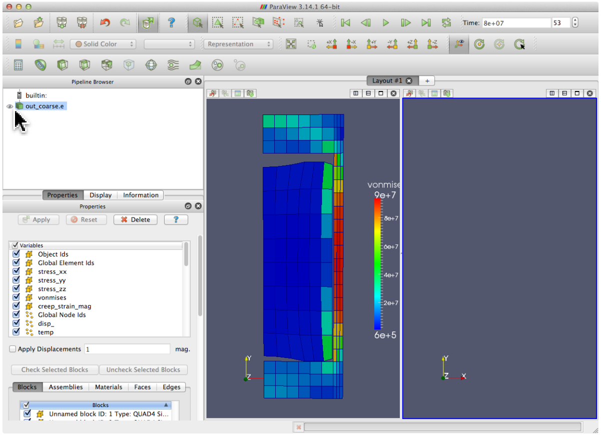
Linking Displays (cont.)

- Now, select “3D View” from the menu in the new display you just created



Linking Displays (cont.)

- A new display should apear showing only the axes.
- Select the “eyeball” in the Pipeline Browser to the left of the out_coarse.e results file.



Time: 8e+07 53

builtin:
out_coarse.e

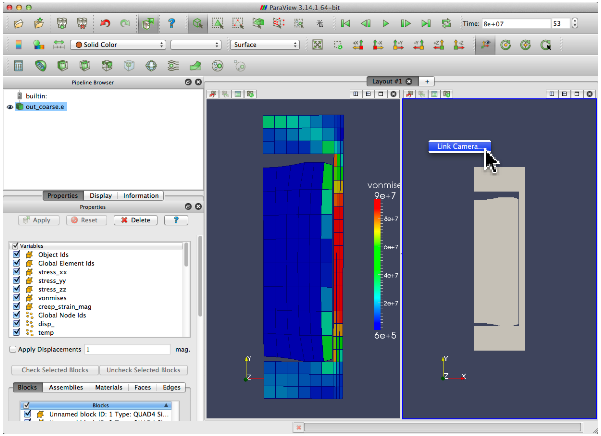
Variables
- ☑ Object Ids
- ☑ Global Element Ids
- ☑ stress_xx
- ☑ stress_yy
- ☑ stress_zz
- ☑ vonmises
- ☑ creep_strain_mag
- ☑ Global Node Ids
- ☑ disp_
- ☑ temp

☐ Apply Displacements 1 mag.

Blocks
- ☑ Unnamed block ID: 1 Type: QUAD4 Si...

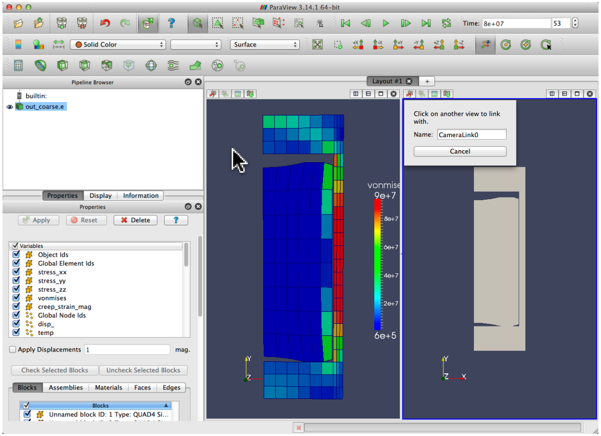
Linking Displays (cont.)

- Right click in the new display with the grey model and that will bring up a “Link Camera” option. Select that.



Linking Displays (cont.)

- Now a window with a message “Click on another view to link with.” should be visible. Select the original display (on left).



ParaView 3.14.1 64-bit

Time: 8e+07 53

Solid Color

Surface

Pipeline Browser

builtin:

out_coarse.e

Click on another view to link with.

Name: CameraLink0

Variables

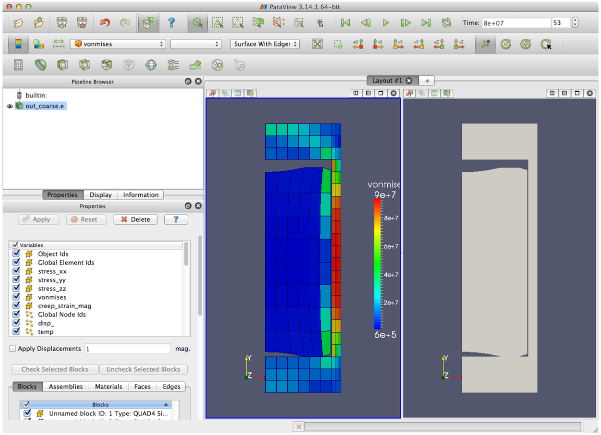
- ✓ Object Ids
- ✓ Global Element Ids
- ✓ stress_xx
- ✓ stress_yy
- ✓ stress_zz
- ✓ vonmises
- ✓ creep_strain_mag
- ✓ Global Node Ids
- ✓ disp_
- ✓ temp

Apply Displacements 1 mag.

Unnamed block ID: 1 Type: QUAD4 Si...

Linking Displays (cont.)

- The displays are linked! Try to rotate/zoom/translate in one window and watch the same action occur in the other. Also try displaying a different set of results in the new display (like temperature).



Variables
- ✓ Object Ids
- ✓ Global Element Ids
- ✓ stress_xx
- ✓ stress_yy
- ✓ stress_zz
- ✓ vonmises
- ✓ creep_strain_mag
- ✓ Global Node Ids
- ✓ disp_
- ✓ temp

Apply Displacements 1 mag.

Check Selected Blocks | Uncheck Selected Blocks

Blocks | Assemblies | Materials | Faces | Edges

Blocks
- ✓ Unnamed block ID: 1 Type: QUAD4 Si...

Best Practices and Solver Options

Best Practices Solver Options

- This section explains some of the best practices to help decrease the runtime of simulations and some of the solver options available to a MOOSE application user.
- We will review:
  - Best Practices
    - Auto Patch Size
    - Centroid Partitioning
    - Inexact Newton Method
  - Solver Options
    - The Jacobian-free Newton-Krylov method
    - PETSc options
    - Preconditioning
    - Time stepping options

Auto Patch Size

- The contact algorithm relies on a cached set of neighboring nodes for each of the nodes on the slave contact surface. The approximate Jacobian matrix (for the Newton method) is formed to be used as a preconditioner to the JFNK method, and this matrix gets nonzero entries added for each of X nearest master nodes, where X is the patch size. Thus, the patch size has a large effect on the memory used for the Jacobian, which in turn dominates the overall memory usage of BISON.
- In the past we would run with a fixed patch size that is large enough so that no slave node moves further up the y axis than the highest master node in its patch (during expansion of the fuel stack and cladding). Depending on the mesh resolution, this can require a patch size of 20, 40, or more.
- A capability has been added to dynamically adjust the patch as the contact surfaces move so that we can run with a much smaller patch size. To use this feature:

```
[Mesh]
  patch_size = 5 #something smaller than the fixed patch size
  patch_update_strategy = auto
[]
```

Centroid Partitioning

- For RZ fuel rod simulations centroid partitioning can help speed up run time by distributing nodes (in parallel simulations) by axial positions along the rod, which leads to better load balancing on longer rods.
- Without this option, it is possible that adjacent chunks of cladding and fuel end up on separate MPI processes increasing run time. To use centroid partitioning:

```
[Mesh]
  partitioner = centroid
  centroid_partitioner_direction = y
[]
```

Inexact Newton Method

- PETSc includes an Eisentatd-Walker algorithm that loosens the required linear tolerance when the Newton solve is still (assumed to be) far from the root. This reduces the number of linear iterations early in the solve where they are wasteful and adds more linear iterations as the Newton solver closes in on the root. These algorithms have been shown to increase robustness and efficiency for many PDEs. To use this algorithm:

```
[Executioner]
  petsc_options = '-snes_ksp_ew'
  l_tol = 1e-2 #<---------- l_tol is ignored when the EW algorithm is used.
[]
```

Jacobian Free Newton Krylov

- $\mathbf{J}(\mathbf{u}_n)\delta\mathbf{u}_{n+1} = -\mathbf{R}(\mathbf{u}_n)$ is a linear system of equations to solve during each Newton step.
- In a Krylov iterative method (such as GMRES) we have the representation:

$$\delta\mathbf{u}_{n+1,k} = a_0\mathbf{r}_0 + a_1\mathbf{J}\mathbf{r}_0 + a_2\mathbf{J}^2\mathbf{r}_0 + \cdots + a_k\mathbf{J}^k\mathbf{r}_0$$

- Note that $\mathbf{J}$ is never explicitly needed. Instead, only the action of $\mathbf{J}$ on a vector needs to be computed.
- This action can be approximated by:

$$\mathbf{J}\mathbf{v} \approx \frac{\mathbf{R}(\mathbf{u}+\epsilon\mathbf{v}) - \mathbf{R}(\mathbf{u})}{\epsilon}$$

- This form has many advantages:
  - No need to do analytic derivatives to form **J**
  - No time needed to compute **J** (just residual computations)
  - No space needed to store **J**

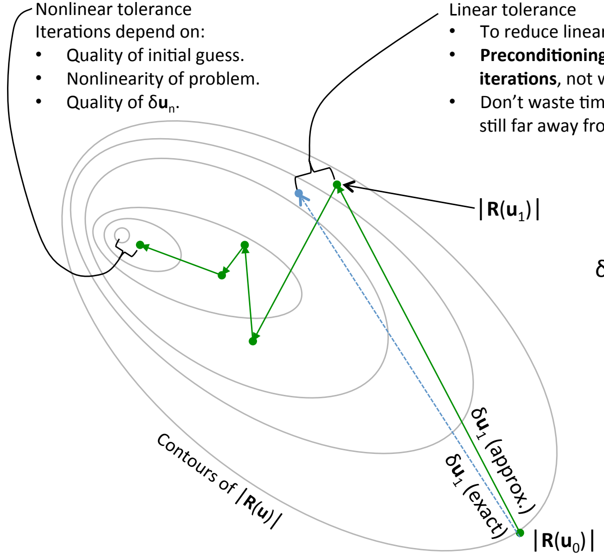
JFNK Overview

Nonlinear tolerance
Iterations depend on:

- Quality of initial guess.
- Nonlinearity of problem.
- Quality of $\delta\mathbf{u}_n$.

Linear tolerance

- To reduce linear error increase linear iterations.
- **Preconditioning improves convergence with linear iterations**, not with nonlinear iterations.
- Don't waste time with tight linear tolerance when still far away from root (try Eisenstadt-Walker).



$|\mathbf{R}(\mathbf{u}_1)|$
$\delta\mathbf{u}_1$ (approx.)
$\delta\mathbf{u}_1$ (exact)
$|\mathbf{R}(\mathbf{u}_0)|$
Contours of $|\mathbf{R}(\mathbf{u})|$

$$\delta\mathbf{u}_{n+1} = -\mathbf{J}^{-1}(\mathbf{u}_n)\mathbf{R}(\mathbf{u}_n)$$

PETSc Solver Options

- PETSc provides many options for the solver and preconditioner:
  `http://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf`
- Codes that use PETSc (e.g. MOOSE) can use a command-line interface to specify solver options. For example:

```
bison -i myfile.i -snes_ksp_ew -ksp_monitor -pc_type hypre
      -snes_linesearch_type basic -ksp_gmres_restart 100
```

- Those can also be specified in the input file using the following commands:

`petsc_options`	⇒ Command line options with no arguments
`petsc_options_iname`	⇒ Command line options with arguments
`petsc_options_value`	⇒ Arguments to options (in order)

- The equivalent input file syntax for the above example would be:

```
[Executioner]
  ...
  petsc_options = '-snes_ksp_ew -ksp_monitor'
  petsc_options_iname = '-pc_type -snes_linesearch_type -ksp_gmres_restart'
  petsc_options_value = 'hypre    basic                 100'
  ...
[]
```

PETSc-specific Options

- The following PETSc-specific options are still used:

Executioner/ petsc_options_iname	Executioner/ petsc_options_value	Description
-pc_type	ilu	Default for serial
	bjacobi	Default for parallel with ilu sub_pc_type
	asm	Additive Schwartz with ilu sub_pc_type
	lu	Full LU. . . serial only!
	hypre	Hypre. . . usually used with Multigrid
-sub_pc_type	ilu, lu, hypre	Can be used with bjacobi or asm
-pc_hypre_type	boomeramg	Used with hypre to use algebraic multigrid
-ksp_gmres_restart	#	Number of Krylov vectors to keep

- PETSc-specific options with no arguments:

Executioner/ petsc_options	Description
-snes_linesearch_monitor	Show progress of line searches
-snes_view	Show summary at end of each nonlinear solve
-info	Show tons of information during the solve
-snes_ksp_ew	use Eisenstadt-Walker inexact Newton

Recommended starting point for PETSc (no mechanical contact)

- Default preconditioning matrix (block diagonal), preconditioned JFNK.
- Use Hypre with algebraic multigrid and store 101 Krylov vectors.
- Use inexact Newton method.
- Turn off line searches for thermo-mechanics.

```
[Executioner]
...
  #Preconditioned JFNK (default)
  solve_type = 'PJFNK'
  petsc_options = '-snes_ksp_ew'
  petsc_options_iname = '-pc_type -pc_hypre_type -ksp_gmres_restart'
  petsc_options_value = 'hypre    boomeramg      101'
  line_search = 'none'

  l_max_its = 100
  l_tol = 1e-2 # ignored because of inexact Newton
  nl_max_its = 100
  nl_rel_tol = 1e-6
  nl_abs_tol = 1e-10
...
[]
```

Recommended starting point for PETSc (with mechanical contact)

- Default preconditioning matrix (block diagonal), preconditioned JFNK.
- Do not use multigrid or ASM (they do not work well with contact).
- Use a direct solver. Fastest is SuperLU_DIST in parallel.
- Turn off line searches for thermo-mechanics.

```
[Executioner]
...
  solve_type = 'PJFNK'
  petsc_options_iname = '-pc_type -pc_factor_mat_solver_package -ksp_gmres_restart'
  petsc_options_value = ' lu       superlu_dist                      101'
  line_search = 'none'

  l_max_its = 100
  l_tol = 1e-2
  nl_max_its = 100
  nl_rel_tol = 1e-6
  nl_abs_tol = 1e-10
...
[]
```

- Note: BISON, MOOSE, and libmesh must be re-built from scratch to use SuperLU:

```
module load moose-dev-clang
module load petsc-3.4.3-superlu
cd my/bison/source/directory
make cleanall
./moose/scripts/update_and_rebuild_libmesh.sh
make
```

The Preconditioning Block

```
[Preconditioning]

  [./my_prec]
    type = SMP
    # SMP Options Go Here!
    # Override PETSc Options Here!
  [../]

  #[./other_prec]
    #type = PBP
    # PBP Options Go Here!
    # Override PETSc Options Here!
  #[../]
[]
```

- The `Preconditioning` block allows you to define which type of preconditioning matrix to build and what process to apply.
- You can define multiple blocks with different names, allowing you to quickly switch out preconditioning options.
- Each sub-block takes a `type` parameter to specify the type of preconditioning matrix.
- Within the sub-blocks you can also provide other options specific to that type of preconditioning matrix.
- You can also override PETSc options here.
- Only one block can be active at a time.

Example with specified preconditioning scheme

- Single matrix preconditioner: Fill in all off-diagonal blocks, preconditioned JFNK, solve linear system with LU

```
[Preconditioning]
  [./SMP]
    type = SMP
    full = true
    petsc_options_iname = '-pc_type'
    petsc_options_value = 'lu'
  [../]
[]
```

Some Common Options for All Executioners

- There a number of options that appear in the executioner block that control the solver.
- Here are a few common options:

Option	Typical value	Description
line_search	"none"	always use full Newton step
l_tol	10^{-2}	linear tolerance
l_max_its	50	maximum linear iterations
nl_rel_tol	10^{-6}	nonlinear relative tolerance
nl_abs_tol	10^{-10}	nonlinear absolute tolerance
nl_max_its	100	maximum nonlinear iterations

Transient Executioner and TimeStepper

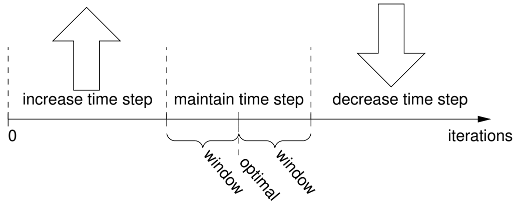
- Transient executioners are used to solve a nonlinear problem over several time steps.
- A TimeStepper is used in conjunction with a Transient Executioner to control the time step size.
- Several TimeStepper types available:
  - ConstantDT
  - FunctionDT
  - IterationAdaptiveDT
- Frequently used general timestepping options that go in Executioner block

num_steps	$\Rightarrow$ Maximum Number of Timesteps
start_time	$\Rightarrow$ Start Time of the Simulation
end_time	$\Rightarrow$ The End Time of the Simulation
dtmax	$\Rightarrow$ Maximum timestep size
dtmin	$\Rightarrow$ Mininum timestep size

IterationAdaptiveDT TimeStepper

- The IterationAdaptiveDT TimeStepper controls the time step based on the solution difficulty, measured by the number of nonlinear iterations required to solve the previous step.
- It can also adjust the solution time based on change in a time-dependent function (i.e. power history).
- This is the TimeStepper type most commonly used in BISON analyses
- Frequently used options:

dt	⇒ Initial Timestep size
optimal_iterations	⇒ Optimal number of nonlinear iterations
iteration_window	⇒ Window within which timestep held constant
timestep_limiting_function	⇒ Function whose change limits timestep size
force_step_every_function_point	⇒ Step at every point in a piecewise function



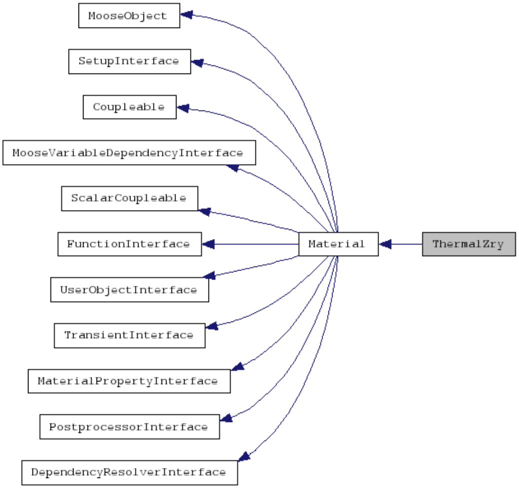
Adding a New Material Model to BISON

Material Models in BISON

- The modular, object-oriented structure of MOOSE makes it straightforward to add new material models to BISON.
- Recall from MOOSE training that material models compute parameters used in kernels.
  - For example, computing specific heat.
- Here are some examples of material and behavioral models in BISON:
  - Thermal models (compute thermal conductivity as a function of burnup, e.g.)
  - Constitutive models that are used with solid mechanics to compute stress (e. g. a creep model that is a function of burnup)
  - Models that compute strain (volumetric swelling and relocation models, e.g.)
  - Fission gas release models that calculate gas produced and released and may contribute to strain due to fission products.
- This section will demonstrate adding a new thermal material model.

Material Models in BISON ... continued

Recall the inheritance structure from MOOSE training.



A New Thermal Material Model

- Let's suppose we want to create a new model for the evolving thermal conductivity and specific heat of a fuel material (called ThermalMyMaterial in this example).
- Our model will depend on temperature and burnup.
- Our model will also depend on input parameters α and β.

Creating a New File

- The source files for nuclear-specific material models are in `bison/src/materials/`.
- The header files for these models are in `bison/include/materials/`.
- Looking in those directories, we see that `ThermalZry` in `bison` is a simple material model that can be used as a template. To make things simple, let's copy `ThermalZry.C` and `ThermalZry.h` to new files. From the project root directory:
  - `> cp bison/src/materials/ThermalZry.C bison/src/materials/ThermalMyMaterial.C`
  - `> cp bison/include/materials/ThermalZry.h bison/include/materials/ThermalMyMaterial.h`
- Using your favorite text editor, change every instance of the name `ThermalZry` in files `ThermalMyMaterial.C` and `ThermalMyMaterial.h` to the name `ThermalMyMaterial`.
- *Important! Be sure to change* `THERMALZRY_H` *to* `THERMALMYMATERIAL_H` *in the header (.h) file.*

Resulting Header File (ThermalMyMaterial.h) ...

Before:

```
#ifndef THERMALZRY_H
#define THERMALZRY_H

#include "Material.h"

//Forward Declarations
class ThermalZry;

template<>
InputParameters validParams<ThermalZry>();

/**
 * Temperature dependent thermal properties
 * of zirconium alloy
 */

class ThermalZry : public Material
{
public:
  ThermalZry(const std::string & name,
      InputParameters parameters);

protected:
  virtual void computeProperties();
   .
   .
   .
#endif //THERMALZRY_H
```

After:

```
#ifndef THERMALMYMATERIAL_H
#define THERMALMYMATERIAL_H

#include "Material.h"

//Forward Declarations
class ThermalMyMaterial;

template<>
InputParameters validParams<ThermalMyMaterial>();

/**
 * Temperature and burnup dependent thermal properties
 * of ThermalMyMaterial
 */

class ThermalMyMaterial : public Material
{
public:
  ThermalMyMaterial(const std::string & name,
      InputParameters parameters);

protected:
  virtual void computeProperties();
   .
   .
   .
#endif //THERMALMYMATERIAL_H
```

Including New Parameters in Header File

- Edit the header file to accommodate additional parameters:
  - Real number parameters α and β.
  - Burnup variable to couple with.
- Note that the convention is to start each member variable with an underscore.

Resulting Header File (`ThermalMyMaterial.h`) ...

Before:

```
  .
  .
  .
class ThermalMyMaterial : public Material
{
public:
  ThermalMyMaterial(const std::string & name,
      InputParameters parameters);

protected:
  virtual void computeProperties();

private:
  bool _has_temp;

  const VariableValue    & _temp;
  const VariableGradient & _grad_temp;

  MaterialProperty<Real> & _thermal_conductivity;
  MaterialProperty<Real> & _thermal_conductivity_dT;
  MaterialProperty<Real> & _specific_heat;
};

#endif //THERMALMYMATERIAL_H
```

After:

```
  .
  .
  .
class ThermalMyMaterial : public Material
{
public:
  ThermalMyMaterial(const std::string & name,
      InputParameters parameters);

protected:
  virtual void computeProperties();

private:
  bool _has_temp;

  const VariableValue    & _temp;
  const VariableGradient & _grad_temp;
  bool _has_burnup;
  const VariableValue & _burnup;

  MaterialProperty<Real> & _thermal_conductivity;
  MaterialProperty<Real> & _thermal_conductivity_dT;
  MaterialProperty<Real> & _specific_heat;

  const Real _alpha;
  const Real _beta;
};

#endif //THERMALMYMATERIAL_H
```

Including New Parameters in Source File

- Edit the source file to accommodate the parameters α and β, and the burnup variable.

- Two types of changes are required to get the parameters ready for use:
    1. Adding the parameters to the list of `InputParameters` to be parsed.
    2. Adding the parameters to the initialization list of `ThermalMyMaterial`.

Resulting Source File (ThermalMyMaterial.C) ...

For the parsing...

Before:

```
#include "ThermalMyMaterial.h"
  .
  .
  .
template<>
InputParameters validParams<ThermalMyMaterial>()
{
  InputParameters params = validParams<Material>();

  params.addCoupledVar("temp", "Coupled Temperature");

  return params;
}
  .
  .
  .
```

After:

```
#include "ThermalMyMaterial.h"
  .
  .
  .
template<>
InputParameters validParams<ThermalMyMaterial>()
{
  InputParameters params = validParams<Material>();

  params.addCoupledVar("temp", "Coupled Temperature");
  params.addCoupledVar("burnup", "Coupled Burnup");

  params.addRequiredParam<Real>("alpha",
    "The alpha parameter");
  params.addRequiredParam<Real>("beta",
    "The beta parameter");

  return params;
}
  .
  .
  .
```

Resulting Source File (`ThermalMyMaterial.C`) ...

For the initialization...

Before:

```
  .
  .
  .
ThermalMyMaterial::ThermalMyMaterial(
                   const
                   InputParameters & parameters)
  : Material(parameters),
  _has_temp(isCoupled("temp")),
  _temp(_has_temp ? coupledValue("temp") : _zero),
  _grad_temp(_has_temp ? coupledGradient("temp") :
                        _grad_zero),
  _thermal_conductivity(
    declareProperty<Real>("thermal_conductivity")),
  _thermal_conductivity_dT(
    declareProperty<Real>("thermal_conductivity_dT")),
  _specific_heat(
    declareProperty<Real>("specific_heat"))
{
}
  .
  .
  .
```

After:

```
  .
  .
  .
ThermalMyMaterial::ThermalMyMaterial(
                   const
                   InputParameters & parameters)
  : Material(parameters),
  _has_temp(isCoupled("temp")),
  _temp(_has_temp ? coupledValue("temp") : _zero),
  _grad_temp(_has_temp ? coupledGradient("temp") :
                        _grad_zero),
  _has_burnup(isCoupled("burnup")),
  _burnup(_has_burnup ?  coupledValue("burnup") :
   _zero),
  _thermal_conductivity(
    declareProperty<Real>("thermal_conductivity")),
  _thermal_conductivity_dT(
    declareProperty<Real>("thermal_conductivity_dT")),
  _specific_heat(
    declareProperty<Real>("specific_heat")),
  _alpha(getParam<Real>("alpha")),
  _beta(getParam<Real>("beta"))
{}
  .
  .
  .
```

Coding New Model Equations

- Edit the source file to compute the thermal conductivity and specific heat according to our new model.

- Let's suppose that our equations are:
  - $k = 5(T - 500)^{\alpha}(1 - Bu^3)$
  - $C = 50 + (T/1000)^{\beta}$

Resulting Source File (`ThermalMyMaterial.C`)

After:

```
  .
  .
  .
void
ThermalMyMaterial::computeProperties()
{

  for(unsigned int qp=0; qp<_qrule->n_points(); ++qp)
  {
    // Conductivity of irradiated MyMaterial: W/(m K)
    const Real b(1-std::pow(_burnup[qp],3));
    _thermal_conductivity[qp] = 5*std::pow(_temp[qp]-500,_alpha)*b;
    _thermal_conductivity_dT[qp] = 5*_alpha*std::pow(_temp[qp]-500,_alpha-1)*b;

    // Specific heat: J/(mol K)
    _specific_heat[qp] = 50 + std::pow(_temp[qp]/1000,_beta);

  }
  .
  .
  .
```

Registering Our Model with BISON

- The final code change is to tell BISON about our model.
- This is done by modifying `bison/src/base/BisonApp.C`.
- Recompile BISON to have your changes available for use.

Resulting Source File (`BisonApp.C`)

Before:

```
  .
  .
  .
#include "MesoThCond.h"
#include "RichUO2.h"
#include "ThermalUO2.h"
#include "ThermalUO2Meso.h"
  .
  .
  .
    registerMaterial(MesoThCond);
    registerMaterial(RichUO2);
    registerMaterial(ThermalUO2);
    registerMaterial(ThermalUO2Meso);
  .
  .
  .
```

After:

```
  .
  .
  .
#include "MesoThCond.h"
#include "RichUO2.h"
#include "ThermalMyMaterial.h"
#include "ThermalUO2.h"
#include "ThermalUO2Meso.h"
  .
  .
  .
    registerMaterial(MesoThCond);
    registerMaterial(RichUO2);
    registerMaterial(ThermalMyMaterial);
    registerMaterial(ThermalUO2);
    registerMaterial(ThermalUO2Meso);
  .
  .
  .
```

Running with Our New Model

```
[Materials]
  .
  .
  .
  [./new_model]
    type = ThermalMyMaterial
    block = 2
    temp = temp
    burnup = burnup
    alpha = 0.6
    beta = 0.333
  [../]
  .
  .
  .
[]
```

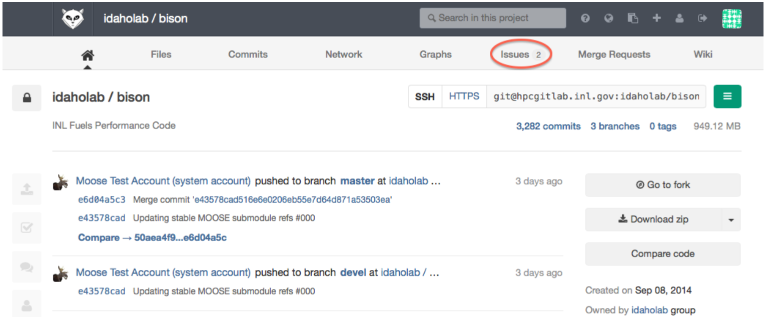
To run our new model, we simply include its description in the Materials section of our BISON input file.

Maintaining Our Model

- To keep up-to-date with the latest BISON/MOOSE changes, periodically run the following in the project root directory:
  - `git pull --rebase upstream devel`
  - `git submodule update`
- You are encouraged to commit your changes to the repository. If you commit your code
  - The BISON team assumes a level of ownership.
  - Others will be able to access and modify your code.
  - Those making changes that prevent your code from compiling must eliminate those errors before committing their changes.
  - You must also commit at least one regression test. This helps ensure that no one breaks your code.
- If you do not commit your code
  - No one else has access to it.
  - It is your responsibility to update your code if others' changes prevent your code from compiling.
- If your code has wide applicability (particularly if others are interested in it), you can improve the code by committing your work. Remember, though, that at least one regression test is required.

Committing Your New Material Model

- From the idaholab/bison GitLab page. Click Issues and then create a New Issue.



idaholab / bison

git@hpcgitlab.inl.gov:idaholab/bison

INL Fuels Performance Code

3,282 commits 3 branches 0 tags 949.12 MB

Moose Test Account (system account) pushed to branch master at idaholab ... 3 days ago

e6d04a5c3 Merge commit 'e43578cad516e6e0206eb55e7d64d871a53503ea'

e43578cad Updating stable MOOSE submodule refs #000

Compare → 50aea4f9...e6d04a5c

Moose Test Account (system account) pushed to branch devel at idaholab / ... 3 days ago

e43578cad Updating stable MOOSE submodule refs #000

Created on Sep 08, 2014

Owned by idaholab group

- Change directories to where you cloned your fork, and run:
  - `git checkout devel` (this ensures you are branching off of the devel branch)
- Decide on the name of your branch. (e.g. my_material_1729, where 1729 references your issue number). Now create your branch:
  - `git checkout -b your_branch_name devel`

Committing Your New Material Model

- Add the files you just created and modified to your local copy of the BISON repository.
  - from `bison/` type `git add src/materials/ThermalMyMaterial.C`
  - from `bison/` type `git add include/materials/ThermalMyMaterial.h`
  - from `bison/` type `git add src/base/BisonApp.C`
- If you type `git status` the screen should look like:
  - `Changes to be committed:` `(use "git reset HEAD <file>..." to unstage)`
    - `modified:   src/base/BisonApp.C`
    - `new file:   src/materials/ThermalMyMaterial.C`
    - `new file:   include/materials/ThermalMyMaterial.h`
- If you see a file you don't want to commit type:
  - `git checkout the_offending_file_name` (this will "revert" to that file's original status)
- Your code will be rejected if it has trailing whitespace. It is recommended to run a script to remove the trailing whitespace:
  - `./scripts/delete_trailing_whitespace.sh`
- If you are an Emacs user, add the following to your .emacs file:
  - `(add-hook 'before-save-hook 'delete-trailing-whitespace)`

Committing Your New Material Model

- Commit your changes locally.
  - `git commit -m "Some message which references #your_issue_number"`
- Make sure you are up to date by typing:
  - `git pull --rebase upstream devel`
  - `git submodule update`
- Compile and run the tests.
- Push the branch to your fork.
  - `git push origin your_branch_name`
- Go back to the GitLab website and create a Merge Request. Reference your issue number. The BISON team will now comment on your contributions and you'll have to address those comments.

Addressing BISON Team Comments

- Ensure you are up to date
  - `git pull --rebase upstream devel`
  - `git submodule update`
- Make your changes then compile and run the tests. Next amend your commit:
  - `git commit -a --amend` (this will bring up a vi editor to change the commit message. You can just type :wq)
  - `git push -f origin your_branch_name`
- Once the BISON team is satisfied with your contributions we will merge your changes. Once the changes are merged you can delete your branch.

Deleting your Branch

- Delete your branch locally.
  - `git branch -D your_branch_name`
- Delete branch on GitLab
  - `git push origin :your_branch_name`
- Delete references to GitLab branch in local repository
  - `git remote prune origin`
- Once your branch is deleted you have successfully committed to BISON.

Adding a Regression Test to BISON

Regression Test - Definition

- A *simple* simulation that you design for the purpose of testing a new feature. Regression tests generally have an analytical solution and are designed to verify that the software computes the correct result.

- For example, if you committed `ThermalMyMaterial` model from the “Adding a New Material Model to BISON” section, you would include a regression test.

- A regression test for `ThermalMyMaterial` would ensure the proper values of thermal conductivity are calculated.

- Regression tests usually consist of a single element and run in less that 2 seconds on 1 processor.

Adding Your New Regression Test

- Start by copying a similar regression test. These can be found in `bison/tests/`
- Let's add a regression test for `ThermalMyMaterial`
- We'll start with a similar test located at `bison/tests/thermalU3Si2/`, which tests a thermal model for Uranium Silicide. In this test, temperature of the single element domain is controlled via application of a Dirichlet boundary condition. Values for thermal conductivity and specific heat are recorded and checked against an analytical expression.
- Create a directory in `bison/tests/` called `ThermalMyMaterial`
  - in `bison/tests/` type `mkdir ThermalMyMaterial`
- Then, copy the input file, mesh, and a file called `tests` from `thermalU3Si2/` to the directory you just created
  - in `bison/tests/ThermalMyMaterial/` type `cp ../thermalU3Si2/thermalU3Si2.i ThermalMyMaterial.i`
  - then type `cp ../thermalU3Si2/1x1x1cube.e .`
  - then type `cp ../thermalU3Si2/tests .`

Adding Your New Regression Test... continued

The next step is to edit the file `ThermalMyMaterial.i`. Recall that this input file was copied from one used to test a thermal model for Uranium Silicide, which has models for thermal conductivity and specific heat, each of which depends on temperature. Thermal conductivity in our new model (`ThermalMyMaterial`) depends on temperature and burnup and specific heat depend on only temperature. So, we have to modify this input file so that `ThermalMyMaterial` is used, and we need to supply our new model with burnup as a variable.

Also note that this input file begins with a description of the test, references, and results from BISON and the corresponding analytical expressions for thermal conductivity and specific heat. This information is important and expected in an input file for a regression test. Other users and developers will look at this information to understand the model and the test.

Editing the input file (`ThermalMyMaterial`)

Before:

```
[Mesh]
  file = 1x1x1cube.e
[]


[Variables]
  [./T]
    order = FIRST
    family = LAGRANGE
    initial_condition = 300
# set initial T to 300 K
  [../]
[]
```

After:

```
[GlobalParams]
 density = 1200
[]


[Mesh]
  file = 1x1x1cube.e
[]


[Variables]
  [./T]
    order = FIRST
    family = LAGRANGE
    initial_condition = 500
# set initial T to 300 K
  [../]
[]
```

We added the `[GlobalParams]` block because we're going to add the burnup variable, which is a function of density. The parameter `density` now shows up in more than one place in the input file, so it's convenient and less error prone to include it in the `[GlobalParams]` block.

Editing the input file (`ThermalMyMaterial.i`)...

Before:

```
[AuxVariables]
  [./th_cond]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./cp]
    order = CONSTANT
    family = MONOMIAL
  [../]
[]
```

After:

```
[AuxVariables]
  [./fission_rate]
    order = FIRST
    family = LAGRANGE
  [../]
  [./burnup]
    order = FIRST
    family = LAGRANGE
  [../]
  [./th_cond]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./cp]
    order = CONSTANT
    family = MONOMIAL
  [../]
[]
```

Editing the input file (`ThermalMyMaterial.i`)...

Before:

```
[AuxKernels]
  [./th_cond]
    type = MaterialRealAux
    variable = th_cond
    property = thermal_conductivity
    block = 1
  [../]

  [./cp]
    type = MaterialRealAux
    variable = cp
    property = specific_heat
    block = 1
  [../]
[]
```

After:

```
[AuxKernels]
  [./fissionrate]
    type = FissionRateAux
    variable = fission_rate
    value = 1.183e19
    execute_on = timestep_begin
  [../]
  [./burnup]
    type = BurnupAux
    variable = burnup
    fission_rate = fission_rate
    execute_on = timestep_begin
  [../]
  [./th_cond]
    type = MaterialRealAux
    variable = th_cond
    property = thermal_conductivity
    block = 1
  [../]

  [./cp]
    type = MaterialRealAux
    variable = cp
    property = specific_heat
    block = 1
  [../]
[]
```

Editing the input file (`ThermalMyMaterial.i`)...

Before:

```
[Materials]

  [./fuel_thermalU3Si2]
    type = ThermalU3Si2
    block = 1
    temp = T
  [../]

  [./density]
    type = Density
    block = 1
    density = 1200.0
  [../]
[]
```

After:

```
[Materials]

  [./fuel_ThermalMyMaterial]
    type = ThermalMyMaterial
    block = 1
    temp = T
    burnup = burnup
    alpha = 0.6
    beta = 0.333
  [../]

  [./density]
    type = Density
    block = 1
  [../]
[]
```

Note how the parameter `density` is *not* included in the `[Materials]` block because we already included it in the `[GlobalParams]` block

Editing the input file (`ThermalMyMaterial.i`)...

Before:

```
[Postprocessors]
   [./avg_th_cond]
     type = ElementAverageValue
     block = 1
     variable = th_cond
   [../]
   [./avg_cp]
     type = ElementAverageValue
     block = 1
     variable = cp
   [../]
   [./average_fuel_T]
     type = ElementAverageValue
     output = both
     block = 1
     variable = T
   [../]
 []
```

After:

```
[Postprocessors]
   [./avg_th_cond]
     type = ElementAverageValue
     block = 1
     variable = th_cond
   [../]
   [./avg_cp]
     type = ElementAverageValue
     block = 1
     variable = cp
   [../]
   [./average_fuel_T]
     type = ElementAverageValue
     output = both
     block = 1
     variable = T
   [../]
   [./average_fuel_burnup]
     type = ElementAverageValue
     output = both
     block = 1
     variable = burnup
   [../]
 []
```

Editing the input file (`ThermalMyMaterial.i`)...

Before:

```
[Output]
  linear_residuals = true
  interval = 1
  output_initial = true
  exodus = true
  perf_log = true
[]
```

After:

```
[Output]
  linear_residuals = true
  postprocessor_csv = true
  interval = 1
  output_initial = true
  exodus = true
  perf_log = true
[]
```

Note the additional parameter `postprocessor_csv = true`. When you commit this test, you should comment out this parameter. If left enabled, this parameter will produce a csv file that contains the history of the Postprocessors defined in the input file. You don't want the regression test to reproduce this file every time it runs. However, before you commit this test, you could use this file to compare BISON calculations with, say, spread sheet calculations of equations for thermal conductivity and specific heat that we coded in `ThermalMyMaterial.C`. Then, include a summary of these calculations at the beginning of the regression test input file. See `bison/tests/thermalU3Si2.i` for an example. For the test we just created, thermal conductivity should be reported as a function of burnup and temperature. Specific heat should be reported as a function of temperature.

Editing the `tests` file

Before:

```
[Tests]
  [./U3Si2_test]
    type = 'Exodiff'
    input = 'thermalU3Si2.i'
    exodiff = 'thermalU3Si2_out.e'
  [../]
[]
```

After:

```
[Tests]
  [./ThermalMyMaterial]
    type = 'Exodiff'
    input = 'ThermalMyMaterial.i'
    exodiff = 'ThermalMyMaterial_out.e'
  [../]
[]
```

Finishing steps and `gold/`

- Run your simulation and verify that it runs as you expect.
- Provide documentation of the test in the input file.
- Create a directory called `gold/` in `bison/tests/ThermalMyMaterial/`
- Move the exodus output file to the `gold/` directory.
  - `> mv ThermalMyMaterial_out.e gold/`
- Run BISON tests and make sure the new test passes.
  - `>./run_tests -j4` from the `bison/` directory.
- Make sure that the output exodus file as well as any other extraneous files are deleted from the directory `bison/tests/ThermalMyMaterial/`.
- Add the new directory and commit it as outlined in the previous section.
- Done.

Additional Information

www.inl.gov

Idaho National Laboratory

Additional Information: Useful Git Commands

`> git clone git@hpcgitlab.inl.gov:<username>/bison.git` - Clones your bison fork to your computer. `> git pull --rebase upstream devel` - Updates local repository against the devel branch of the upstream remote.

`> git submodule update` - Updates your submodules.

`> git add <filename>` - Registers the file with git. Use -i for interactive mode. Use with -f to force add files that are in the .gitignore file.

`> git commit -m 'commit_message'` - Commits the file(s) that were added to the current branch. Use –amend to amend to the previous commit.

`> git checkout -b <branch_name>` - Creates a branch and checks it out.

`> git checkout -b <branch_name> <sha>` - Creates a branch with the specified sha. Good for checking out previous versions.

`> git branch -D <branch_name>` - Force deletes the branch.

`> git status`

`> git log`

`> git reset --hard HEAD` - Removes any staged files and puts the repository back to the head

`> git clean -dfx` - Deletes all files that are not registered with the idaholab/bison repository. This is useful if you have build issues or a corrupted repo.

Additional Information: PBS Submission Script

```
#!/bin/bash
#PBS -M <your_email>
#PBS -m abe
#PBS -N <jobname>
#PBS -l select=1:ncpus=24:mpiprocs=24
```

The above line requests the resources. select=<number of nodes>, ncps=<number of cpu/node> and mpiprocs=<number cpus actually running the problem>

```
#PBS -l place=scatter:excl
#PBS -l walltime=48:00:00
JOB_NUM=$PBS_JOBID%%\.*
log=<logname>
input_file=<filename>
cd $PBS_O_WORKDIR
module load use.moose moose-dev-gcc
date > $log
mpiexec <path_to_bison-opt> -i $input_file >> $log
date >> $log
```

References

www.inl.gov

Idaho National Laboratory

References

M. M. Rashid.
Incremental kinematics for finite element applications.
Internat. J. Numer. Methods Engrg., 36:3937–3956, 1993.

A. M. Ross and R. L. Stoute.
Heat transfer coefficient between UO_2 and Zircaloy-2.
Technical Report AECL-1552, Atomic Energy of Canada Limited, 1962.

L. Schoof and V. Yarberry.
EXODUS II: A finite element data model.
Technical Report SAND92-2137, Sandia National Laboratories, September 1996.

Sandia National Laboratories.
CUBIT: Geometry and mesh generation toolkit.
http://cubit.sandia.gov, 2008.