



**The Abdus Salam  
International Centre for Theoretical Physics**



**1967-11**

**Advanced School in High Performance and GRID Computing**

***3 - 14 November 2008***

**Mathematical Libraries.  
Part II**

HELTAI Luca  
S.I.S.S.A.  
*International School for Advanced Studies*  
*Via Beirut 2-4*  
*34014 Trieste*  
*ITALY*

**Advanced School in  
High Performance  
and GRID Computing**




# **Mathematical Libraries Part II: Working with PDEs**

**Luca Heltai**

**SISSA, Trieste**



# Outline

- The Tools
    - General PDEs Libraries
    - Mesh Generation
    - External Solvers
    - Visualization Libraries
  - The Example
    - Mathematical Background
    - Analysis of the Example
    - Conclusions
- 

# Main Tasks to Solve a PDE\*

- Choose a Numerical Method!
  - FEM – BEM – FD –
- Discretize the Domain
- Generate a Linear(ized) System
- Solve the Linear System
- Visualize the Results

\* PDE := Partial Differential Equation

# Generic PDEs Libraries

- FEM: Deal.II, FreeFem, OOFem, Netlib ...
- FD: Samrai, ...
- BEM: Deal.II, BEMLib, ...

For a complete (almost...) list of scientific applications and Libraries on linux, you can take a look at:

<http://sal.jyu.fi/>

# FEM - BEM: Deal.II

Deal.II (**Finite Element Differential Equations Analysis Library**) is a C++ program library targeted at the computational solution of partial differential equations using adaptive finite elements. It uses state-of-the-art programming techniques to offer you a modern interface to the complex data structures and algorithms required.

<http://www.dealii.org/>

# FD - Structured FEM: SAMRAI

The SAMRAI (**Structured Adaptive Mesh Refinement Application Infrastructure**) library is the code base in CASC for exploring application, numerical, parallel computing, and software issues associated with SAMR.

SAMRAI development has been funded primarily by the DOE Advanced Simulation and Computing (ASC) Program. This investment is complemented by collaborative efforts supported by the Laboratory-Directed Research and Development (LDRD) program.

<https://computation.llnl.gov/casc/SAMRAI/index.html>

# Mesh Generation Tools

- Gmsh:

Gmsh is an automatic 3D finite element grid generator with a built-in CAD engine and post-processor. Its design goal is to provide a simple meshing tool for academic problems with parametric input and advanced visualization capabilities.

Gmsh is built around four modules: geometry, mesh, solver and post-processing. The specification of any input to these modules is done either interactively using the graphical user interface or in ASCII text files using Gmsh's own scripting language.

<http://www.geuz.org/gmsh/>



# Mesh Generation Tools

- Cubit:

CUBIT is an full-featured software toolkit for robust generation of two- and three-dimensional finite element meshes (grids) and geometry preparation. Its main goal is to reduce the time to generate meshes, particularly large hex meshes of complicated, interlocking assemblies.

It is a solid-modeler based preprocessor that meshes volumes and surfaces for finite element analysis. Mesh generation algorithms include quadrilateral and triangular paving, 2D and 3D mapping, hex sweeping and multi-sweeping, tet meshing, and various special purpose primitives. CUBIT contains many algorithms for controlling and automating much of the meshing process, such as automatic scheme selection, interval matching, sweep grouping and sweep verification, and also includes state-of-the-art smoothing algorithms.

<http://cubit.sandia.gov/>

# Helper Libraries

- Metis / ParMetis:

Family of Multilevel Partitioning Algorithms

METIS is a family of programs for **partitioning unstructured graphs and hypergraphs** and computing fill-reducing orderings of sparse matrices.

The underlying algorithms used by METIS are based on the state-of-the-art multilevel paradigm that has been shown to produce high quality results and scale to very large problems.

<http://glaros.dtc.umn.edu/gkhome/views/metis/>

# Helper Libraries

- PETSc:

PETSc (**Portable, Extensible Toolkit for Scientific Computation**), pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations.

It employs the MPI standard for all message-passing communication.

<http://www-unix.mcs.anl.gov/petsc/petsc-as>

# Helper Libraries

- Trilinos:

The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems.

- Framework & Tools
- Software Engineering Technologies and Integration
- Discretizations
- Meshes, Geometry, & Load Balancing
- Scalable Linear Algebra
- Linear & Eigen Solvers
- Embedded Nonlinear Analysis Tools

<http://trilinos.sandia.gov/>

# Visualization and Post Processing

- Visit:

VisIt is a free interactive parallel visualization and graphical analysis tool for viewing scientific data on Unix and PC platforms.

Users can quickly generate visualizations from their data, animate them through time, manipulate them, and save the resulting images for presentations. VisIt contains a rich set of visualization features so that you can view your data in a variety of ways. It can be used to visualize scalar and vector fields defined on two- and three-dimensional (2D and 3D) structured and unstructured meshes. VisIt was designed to handle very large data set sizes in the terascale range and yet can also handle small data sets in the kilobyte range.

<https://wci.llnl.gov/codes/visit/home.html>

# Visualization and Post Processing

- Paraview:

ParaView is an open-source, multi-platform data analysis and visualization application. ParaView users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities.

ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of terascale as well as on laptops for smaller data.

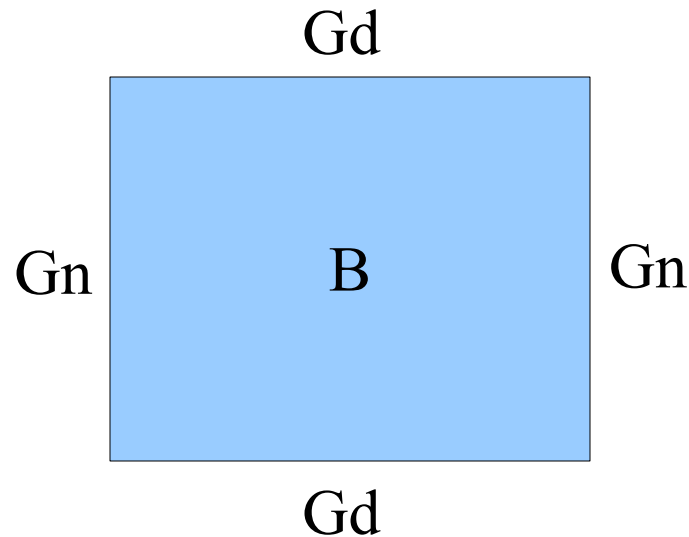
<http://www.paraview.org/>

# **An Example of Fast Prototyping: Parallel Programming for PDEs Using Deal.II + External Libs**

- The mathematical background
- The tools (C++, Deal.II, PETSc, Metis...)
- Analysis of an example
- Conclusions

# Today's Example: Quasi Static Elastic Deformation

$$\begin{aligned} -\operatorname{div} ( C \operatorname{grad} u ) &= 0 && \text{in } B \\ u &= g(t) && \text{on } G_d \\ C \operatorname{grad} u &= 0 && \text{on } G_n \end{aligned}$$



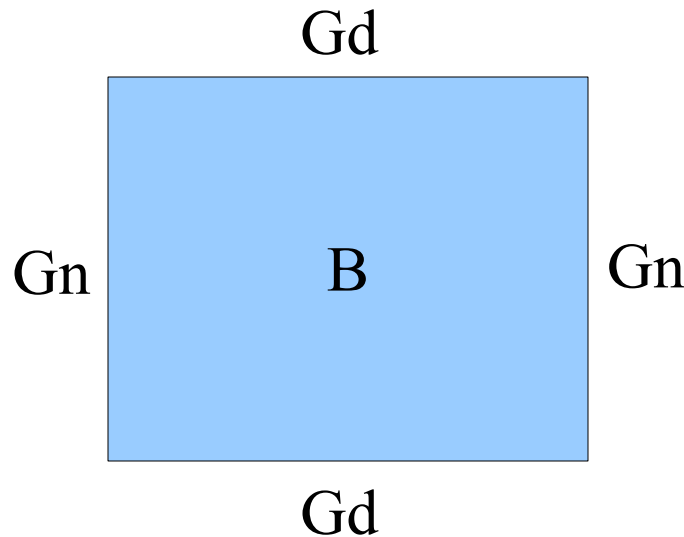


# Variational Formulation

$V_d = \{v \text{ square integrable in } B, \text{ with first derivative square integrable in } B, \text{ such that } v \text{ on } G_d = 0\}$

Find  $u$  such that  $u \text{ on } G_d = g(t)$  for  $t$  in  $[0, T]$  and such that

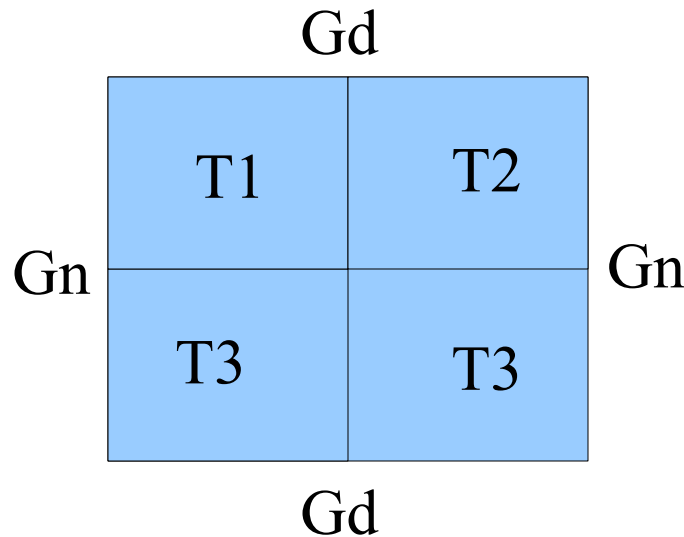
$(C \operatorname{grad} u, \operatorname{grad} v) = 0$  for each  $v$  in  $V_d$



# Finite Element Formulation

$V_h = \{v \text{ in } V_d \text{ such that for each } i, v \text{ on } T_i \text{ is}$   
a bilinear function, and  $v$  is continuous on  $B\}$

Find  $u_h$  such that  $u_h$  on  $G_d = g(t)$  and that  
 $(C \text{ grad } u_h, \text{grad } v) = 0$  for each  $v$  in  $V_h$



# FEM: Reduce PDEs to Linear Systems of Equations

$$\begin{aligned} V_h &= \text{span} \{ v_i \}, i=1, \dots, N \\ u_h &= u_i v_i \text{ (sum convention)} \\ U &= [u_1, u_2, \dots, u_N]^T \end{aligned}$$

$$\text{FEM} \longrightarrow \text{solve } A U = F$$

$$A_{ij} = (C \text{ grad } v_j, \text{ grad } v_i)$$

$v_i$ : piecewise bi-linear,  
1 on node  $i$ , zero everywhere else...

# FEM Requirements

- Subdivide the domain  $B$  in small “Elements” or “cells”  $T_i$
- Assemble the (Sparse) Matrix  $A$  and the right hand side  $F(t)$
- Solve the Linear system  $A U = F(t)$  for a discrete set of times  $t_i$
- Output the results  $U(t_i)$  in a suitable format

# Parallelization of the Program

What can we parallelize?

- The creation of the domain mesh (not implemented so far)
- The domain mesh itself (**Metis**)
- The assembly of the system Matrix (**Deal.II**)
- The solution of the system (**PETSc-Trilinos**)
- The output of the solution (**Deal.II-Visit**)

# Domain Decomposition Paradigm

- The domain is partitioned using **METIS**, an external tool, wrapped into a **Deal.II** function call
- Each subdomain is taken care of by one processor and all the data is distributed using wrappers for **PETSc** Vectors and Matrices
- Each processor only sees a fraction of the entire problem
- Global communication is taken care of by the **PETSc** linear algebra pack.

# **Gluings Things Together...**

- Deal.II provides wrappers for each of the mentioned libraries
- The user needs only to be aware of the domain decomposition techniques, and all the “dirty” MPI messaging is done transparently in the background

# Reference of the Example Program

- Example 18 of the Deal.II library generates one file of output for EACH node and for EACH time step
- Example19 of the Deal.II library glues together the output files relative to the same time step

<http://www.dealii.org/>



# Master Program: Written Using the Deal.II libraries.

- Cycle through the time steps...

```
template <int dim>
void TopLevel<dim>::run ()
{
    present_time = 0;
    present_timestep = 1;
    end_time = 10;
    timestep_no = 0;

    while (present_time < end_time)
        do_timestep ();
}
```

# The Single Time Step...

```
template <int dim>

void TopLevel<dim>::do_timestep ()

{

    present_time += present_timestep;

    ++timestep_no;


    create_mesh();                // Serial    - Divided in subdomains
    assemble_system ();           // Parallel - Subdomain wise
    solve_linear_system ();       // Parallel  - Using PETSc

    output_results ();            // Parallel  - Subdomain wise

    move_mesh ();                 // Parallel  - Subdomain wise

}
```

# The Mesh Creation and Subdivision

```
template <int dim>

void TopLevel<dim>::create_mesh ()

{

    const double inner_radius = 0.8,

               outer_radius = 1;

    // Internal deal.II function

    GridGenerator::cylinder_shell (triangulation,

                                   inner_radius, outer_radius);

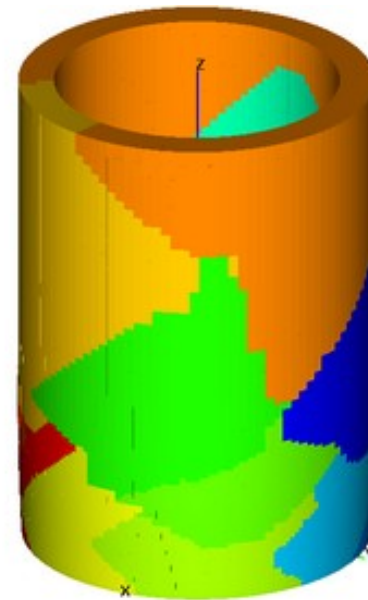
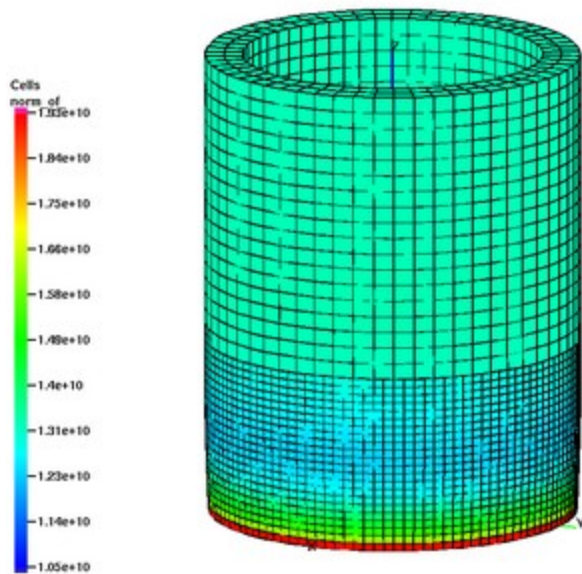
    // Wrapper to the METIS library

    GridTools::partition_triangulation (n_mpi_processes, triangulation);

}
```

# The Actual Mesh

- Generated through Deal.II subroutines...
- ...and subdivided with METIS



# Assembling the System in Parallel...

```
template <int dim>

void TopLevel<dim>::assemble_system ()

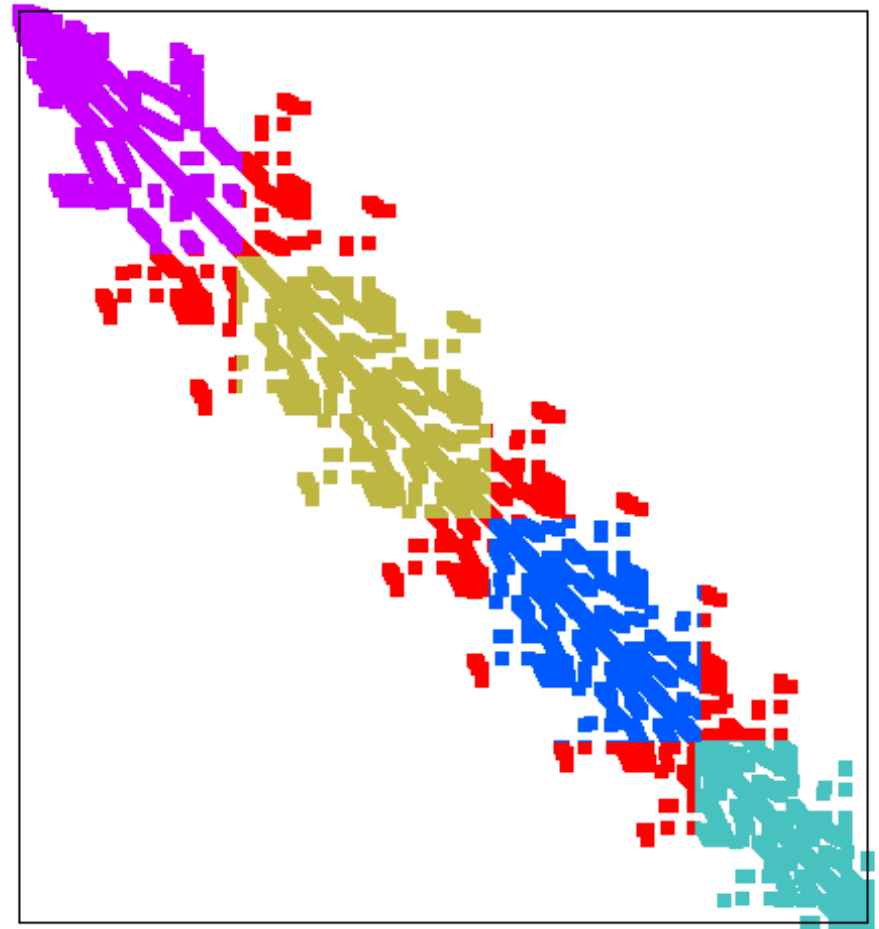
{

    ...
    for (cell = triangulation.begin();
         cell != triangulation.end();
         ++cell)
        if (cell->subdomain_id() == this_mpi_process)
        {
            // Here we assemble the local contribution of this cell
            // and copy it to the Matrix A in the appropriate places
            ...
        }
    A.compress(); // Make sure that the data is coherent
}
```

- The passage of informations to A is done through MPI\_Send
- This passage is transparent to the deal.II library...

# The Sparsity of the Matrix

- The Generated Matrix is sparse (each node is coupled only with its neighbors)
- For large  $N$ , iterative solvers are more efficient



# The Solution of the Linear System

Done using wrappers to PETSc parallel Krilov Subspace Solvers

```
template <int dim>

void TopLevel<dim>::solve_linear_system ()

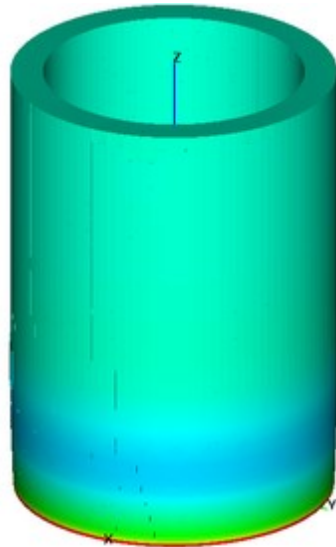
{

    // Conjugate Gradient solver
    PETScWrappers::SolverCG cg (mpi_communicator);
    // Additive Schwartz Preconditioner
    PETScWrappers::PreconditionBlockJacobi preconditioner(A);

    // Perform the solution - in PARALLEL
    cg.solve (A, soution, rhs, preconditioner);

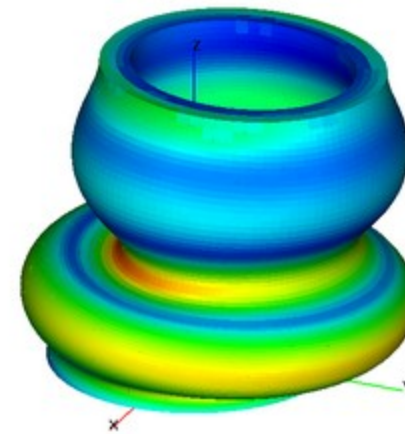
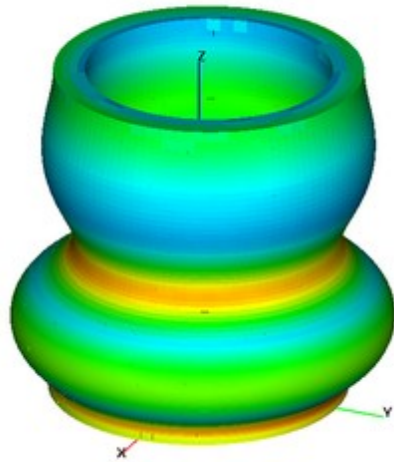
}
```

# The Final Result

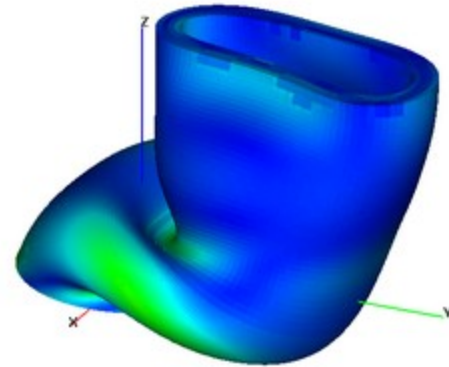
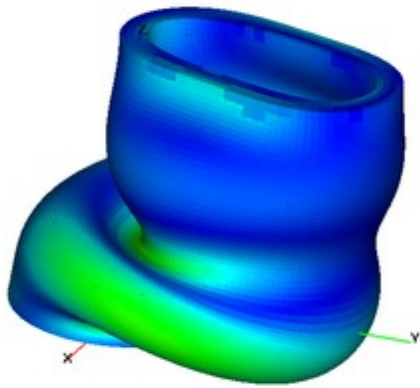




# The Final Result - 2

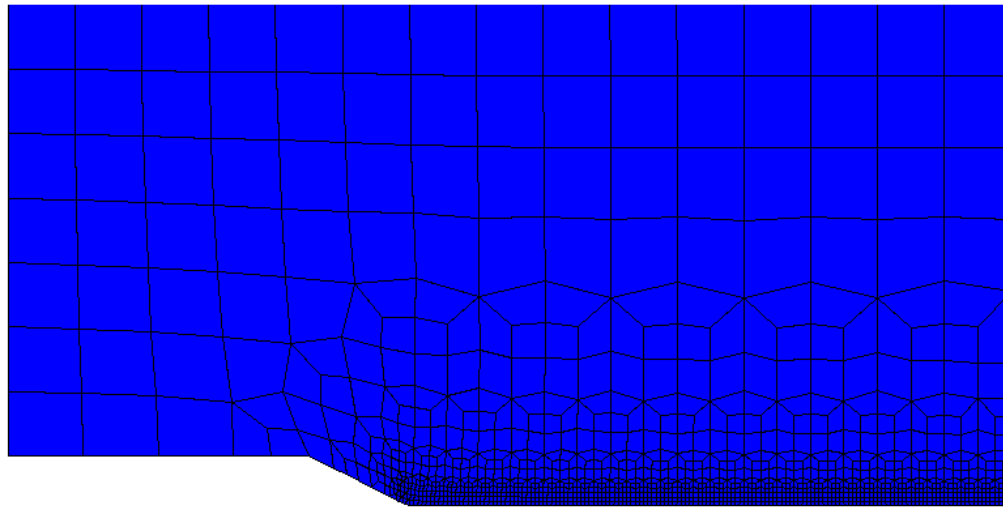


# The Final Result - 3



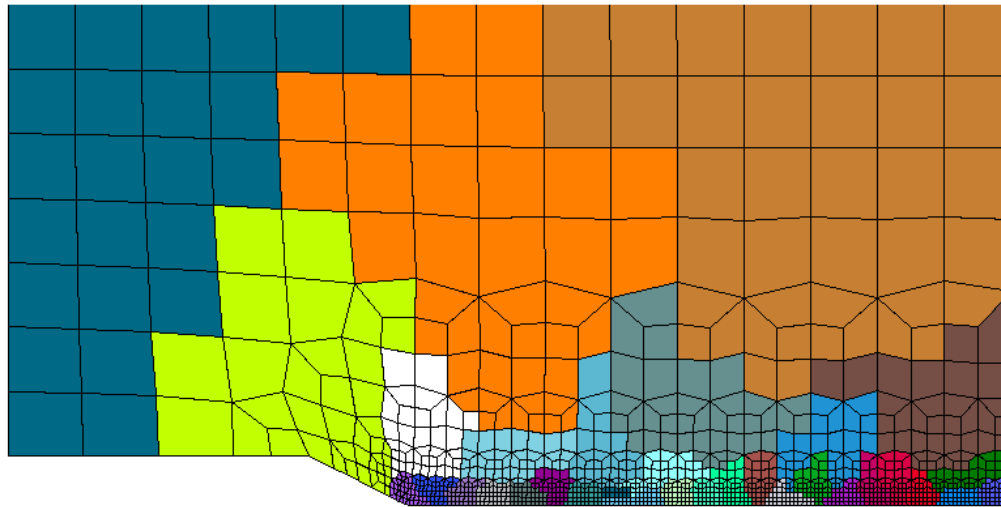
## Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



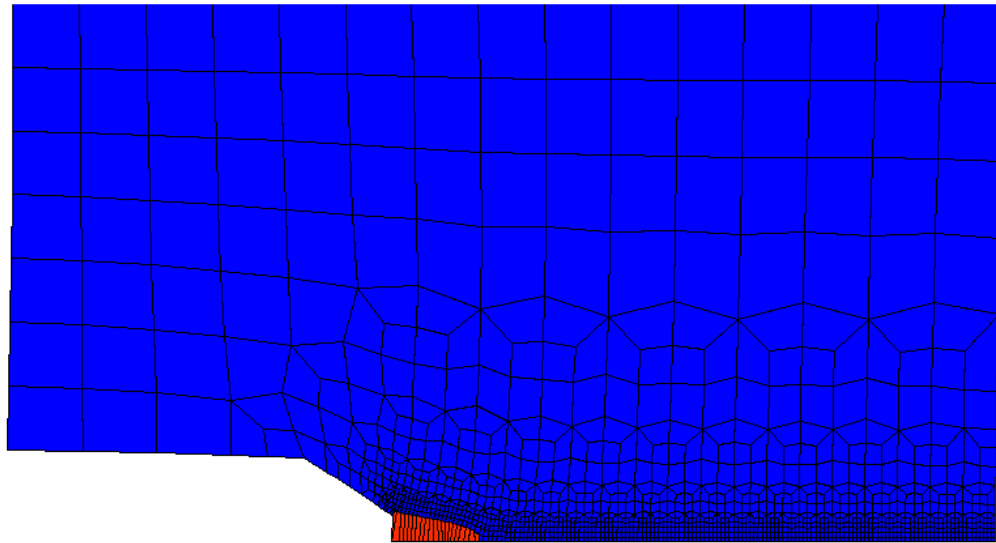
## Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



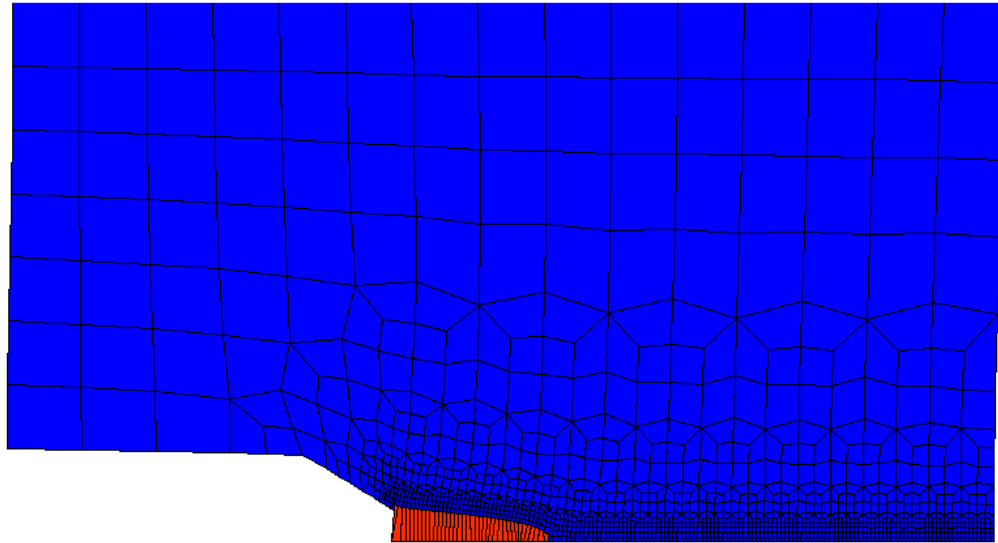
## Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



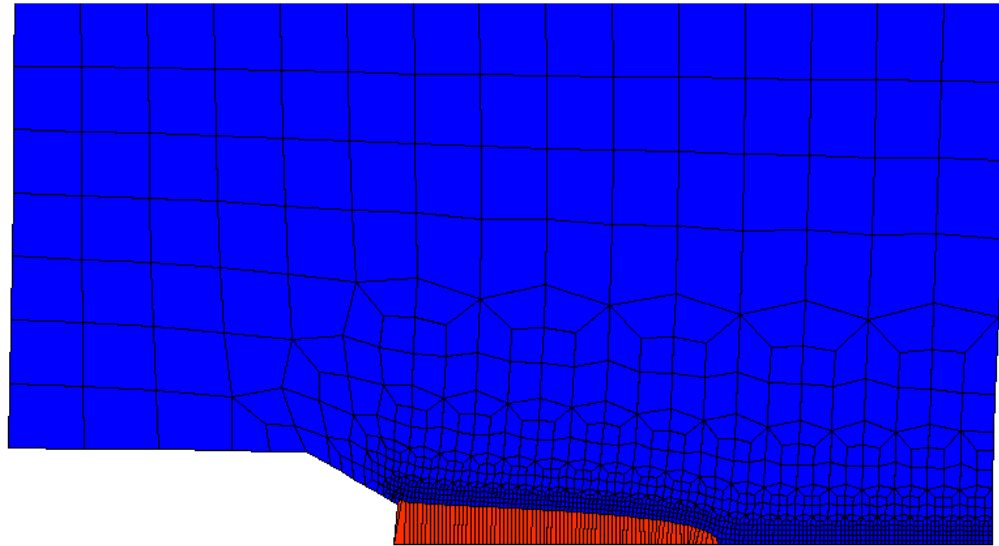
## Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



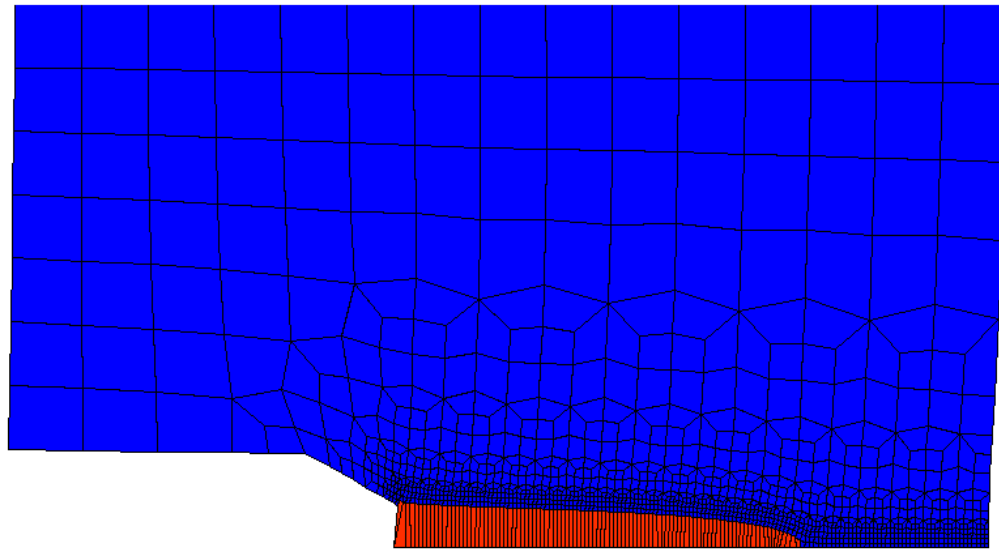
## Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



## Ex2: Damage in Brittle Materials

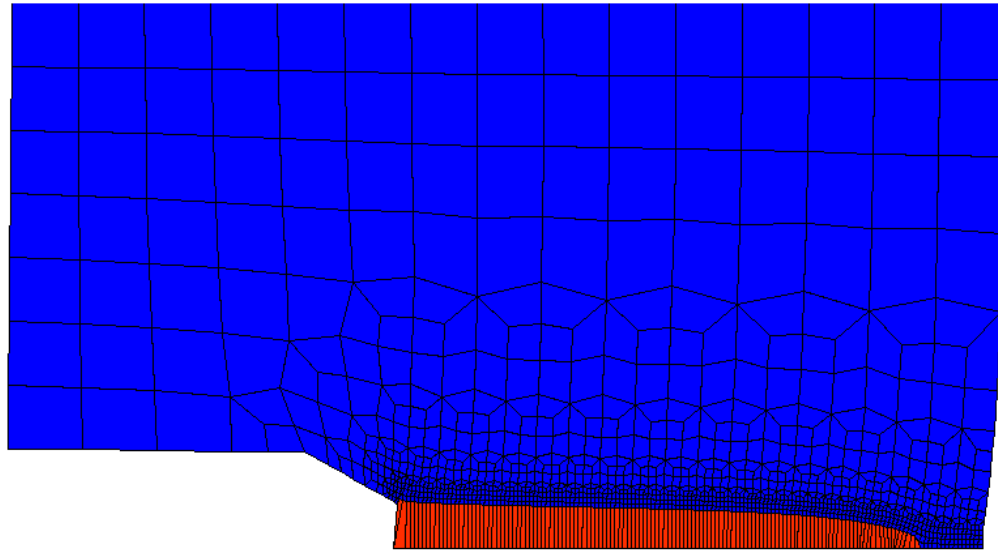
- Work done by Jonathan Pitt, PSU





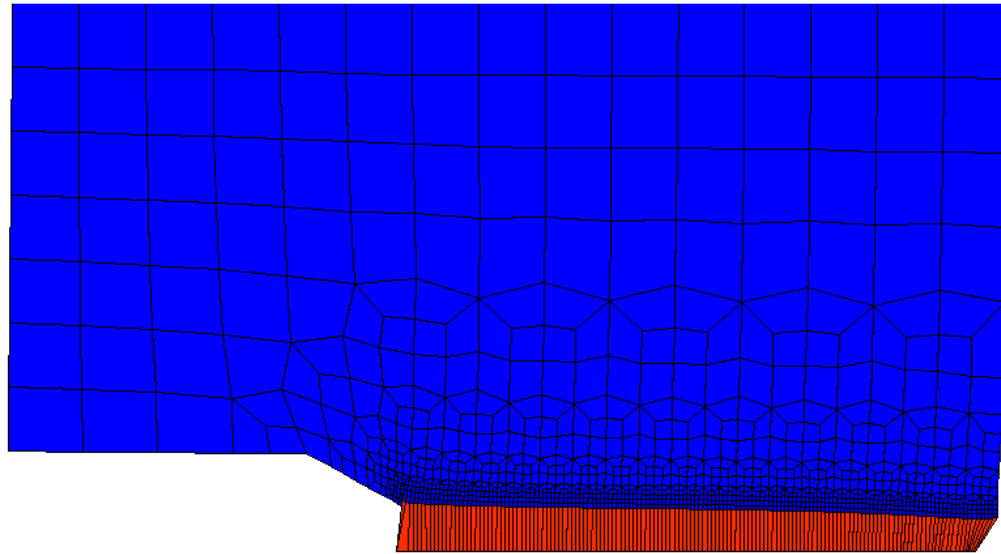
## Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



## Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



# Conclusions

- Fast Prototyping ↔ PreExisting Libraries
- Efficiency ↔ Parallel Libraries
- Customizability ↔ Flexible Libraries