Democritos/ICTP advanced school on HPC tools for e-Science

Fast Prototyping with Parallel Libraries

Luca Heltai - SISSA
luca.heltai@sissa.it

Review of MPI Material

- MPI_Send(b, nc, t, dst, tag, comm, ierr)

Send a message named tag containing nc elements of type t taken from the buffer b to the process identified by the rank dst in the communication environment comm. In case of error, store the error number in ierr.

Same for c, with the exception that a pointer to b is used, and the error is the return value.

Review of MPI Material

- MPI_Recv(b, nc, t, src, tag, comm, status, ierr)

Receive a message named tag which is AT MOST as big as nc elements of type t and store the result in the buffer b.
The message is accepted only by the process identified by the rank src in the communication environment comm.
Once the message is received, the vector status contains informations about the number of received objects, and in case of error, store the error number in ierr.
src can be MPI_ANYSOURCE and tag can be MPI_ANYTAG.

Same for c, with the exception that a pointer to b is used, and the error is the return value.

Review of MPI Material

- MPI_Bcast(b, nc, t, dst, root, comm, ierr)

Send a message containing nc elements of type t taken from the buffer b on the process identified by the rank root in the communication environment comm to all other processes in the same communication environment. In case of error, store the error number in ierr.

Same for c, with the exception that a pointer to b is used, and the error is the return value.

Review of MPI Material

- MPI_Scatter(s, ns, ts, d, nd, td, root, comm, ierr)

Subdivide the buffer s on the root process into npes (number of processes) sub-messages containing each ns elements of type ts and send each one of these sub-messages to all the processes in the communication environment comm (including root!) where it will be stored in the buffer d, which is made of AT MOST nd elements of type td.

In case of error, store the error number in ierr.

Same for c, with the exception that a pointer to b is used, and the error is the return value.

Review of MPI Material

- MPI_Gather(s, ns, ts, d, nd, td, root, comm, ierr)

Each process in the communication environment comm (including root!) sends ns elements of type ts to the root process which stores them in the buffer d, made of npes*nd elements of type td. It is required that nd >= ns, and there will be a gap of (nd-ns) elements between the messages received.

In case of error, store the error number in ierr.

Review of MPI Material

- MPI_Reduce(s, d, n, t, op, root, comm, ierr)

Each process in the communication environment comm (including root!) sends n elements of type t taken from the buffer s to the root process, which performs the operation op between each corresponding element of the messages received from the various processors, and stores the results in the buffer d.

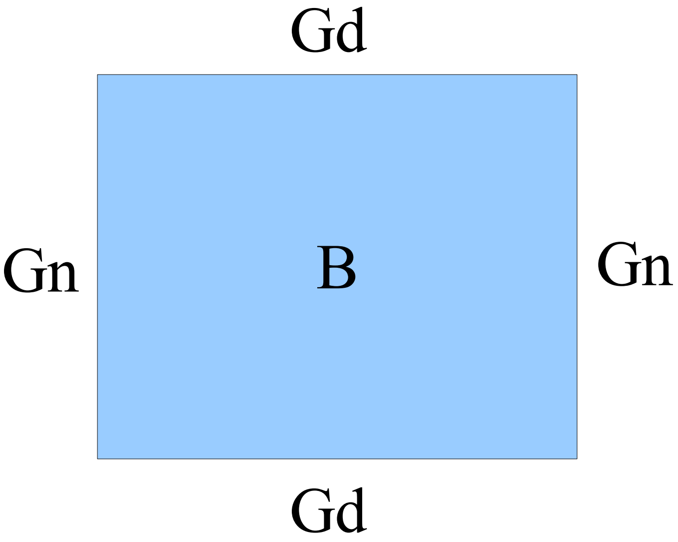
In case of error, store the error number in ierr.

An Example of Real Life Parallel Programming

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

Today's Example: Quasi Static Elastic Deformation

-div (C grad u) = 0	in B
u = g(t)	on Gd
C grad u = 0	on Gn

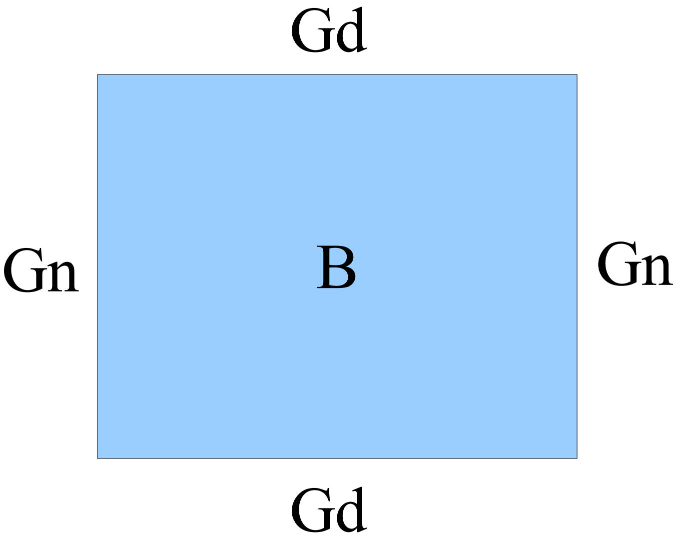


Variational Formulation

Vd = {v square integrable in B, with first derivative square integrable in B, such that v on Gd = 0}

Find u such that u on Gd = g(t) for t in [0,T] and such that

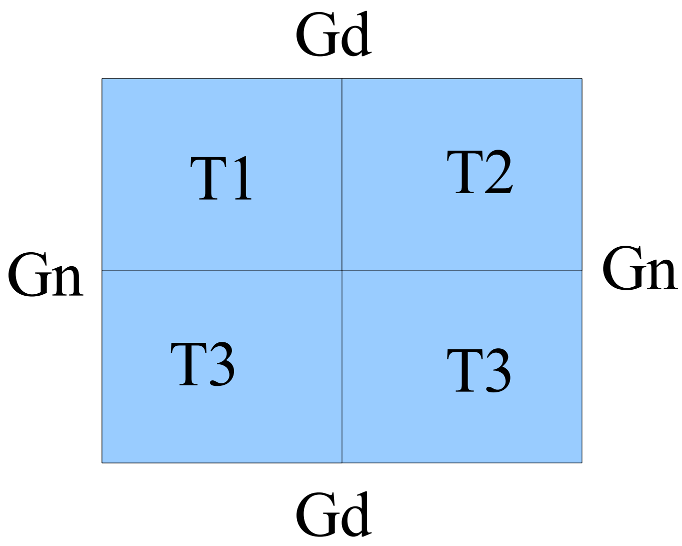
(C grad u, grad v) = 0 for each v in Vd



Finite Element Formulation

Vh = {v in Vd such that for each i, v on Ti is a bilinear function, and v is continuous on B}

Find uh such that uh on Gd = g(t) and that
(C grad uh, grad v) = 0 for each v in Vh



FEM: Reduce PDEs to Linear Systems of Equations

Vh = span { vi }, i=1,...,N

uh = ui vi (sum convention)

U = [u1, u2, ..., uN]'

FEM → solve   A U = F

Aij = (C grad vj, grad vi)

vi: piecewise bi-linear,

1 on node i, zero everywhere else...

FEM Requirements

- Subdivide the domain B in small “Elements” or “cells” Ti
- 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 ti
- Output the results U(ti) 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)
- The output of the solution (Deal.II)

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.

Deal.II Library

- A Finite Element Differential Equations Analysis Library
- Deal.II 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/

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/

PETSc

- Portable, Extensible Toolkit for Scientific Computation
- PETSc, 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

Gluing 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

Deal.II Wrappers to PETSc and MPI

- Deal.II has wrappers for various PETSc objects...

```
PETScWrappers::MPI::Vector vector;
PETScWrappers::MPI::SparseMatrix matrix;
```

- ...which need to be instructed on which parts are local and which are not...

```
vector.reinit(mpi_communicator, n_dofs, n_local_dofs);
matrix.reinit(mpi_communicator,
              n_dofs_n, n_dofs_m,
              n_local_dofs_n, n_local_dofs_m);
```

Wrappers - Hiding MPI

- Main goal of deal.II wrappers: hide MPI stuff....

  Example: Write access to an index of a vector

```
double & PETScWrappers::MPI::Vector::operator[](unsigned int index) {
     if(is_local(index)) {
          // this index is actually owned by this processor,
          // proceed as usual: Return the index of the local vector
          // in which the caller can write directly..
          return (local_vector[global_to_local[index]]);
     } else {
          // Delay the writing, passing the index of a local buffer
          // which will be syncronized later with the global vector
          out_of_scope_indices[n_out_of_scope_writes] = index;
          return (out_of_scope_values[n_out_of_scope_writes++]);
     }
}
```

Example Cont'ed

- Before we can use the vector, all data must be syncronized

```
void PETScWrappers::MPI::Vector::compress() {
     // Send the out of scope write attempts to the root process
     // which will distribute them accordingly...
     // First the sizes of the indices
     MPI_Gather(&n_out_of_scope_values, 1, MPI_UNSIGNED,
                out_of_scope_size_buffer, max_size_buffer,
                MPI_UNSIGNED, 0, mpi_communicator);
     // Now send the actual values
     MPI_Gather(&out_of_scope_values, value_buffer_size,
                MPI_DOUBLE, &out_of_scope_values_buffer,
                mpi_n_processes*value_buffer_size, MPI_DOUBLE,
                0, mpi_communicator);
     // Now do what's needed with this data...
     ...
}
```

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/

Now some details...

MPI Initialization and Main Routine

- Done through Deal.II wrappers to PETSc - MPI

```
 int main (int argc, char **argv)
{
       PetscMPIInitialize(&argc,&argv,0,0);
       {
          const dim = 3;  // The problem dimension: 1d, 2d or 3d.
         QuasiStaticElasticity::TopLevel<dim> elastic_problem;
         elastic_problem.run ();
       }
       PetscMPIFinalize();
}
```

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...

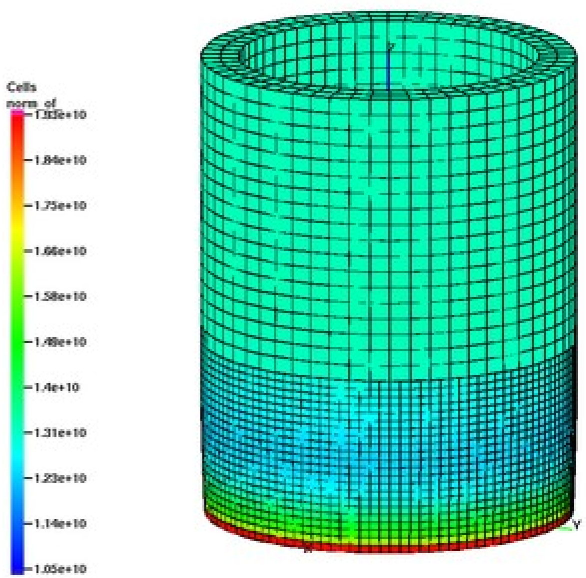
```
template <int dim>
  void TopLevel<dim>::do_timestep ()
  {
    present_time += present_timestep;
    ++timestep_no;

    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
  }
```

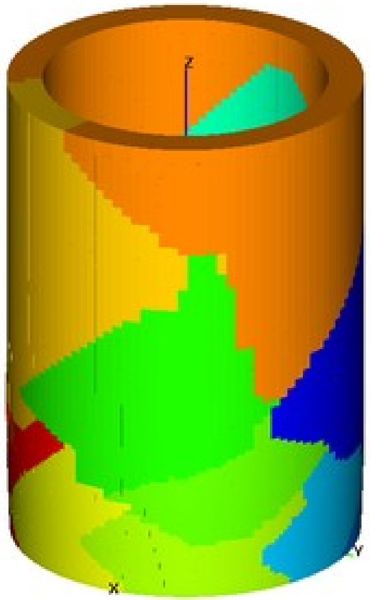
The Mesh Creation and Subdivision

```
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





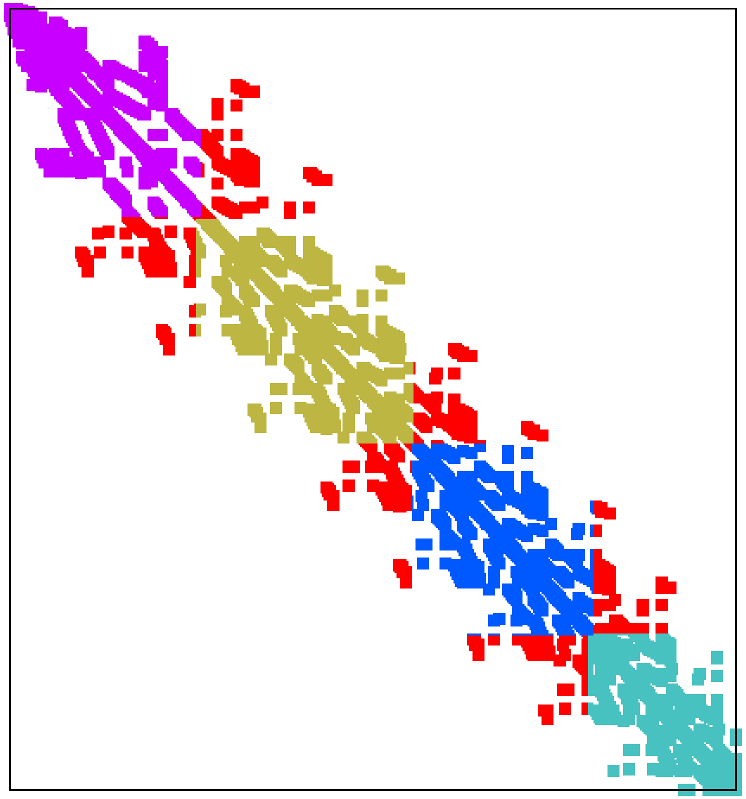
Assembling the System in Parallel...

```
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 Output Routine

Delicate Issue... I/O is not part of MPI standard, and Data Output may become the bottleneck of our parallel program (see Example 17 of Deal.II).

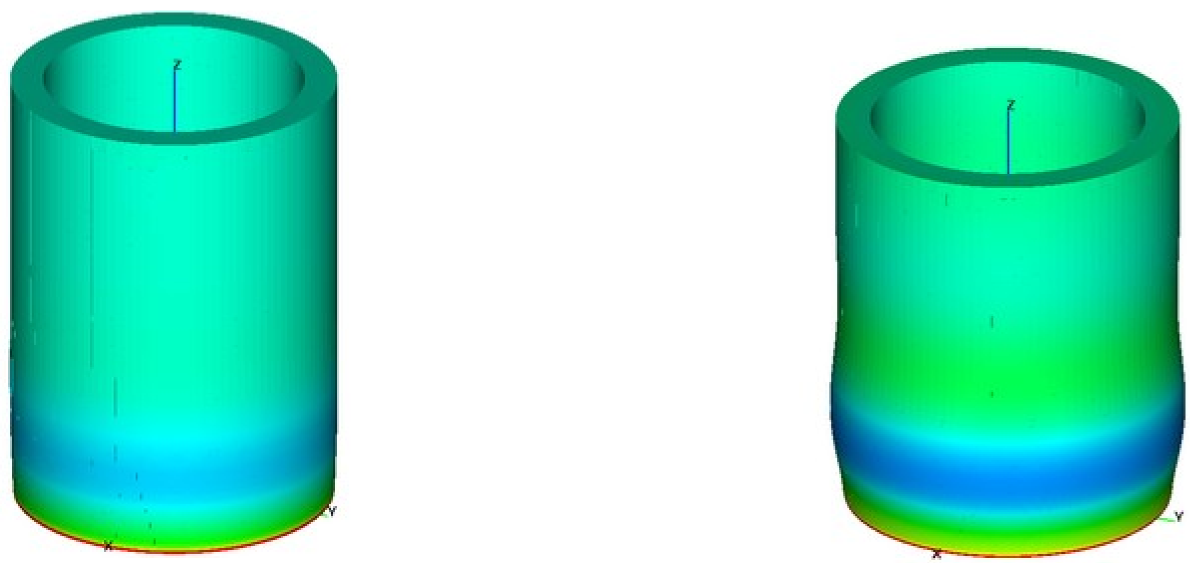
How do we do it?

- Each processor writes its own subset of the solution into a separate file
- An external program (Deal.II Example 19) merges the results into the desired format

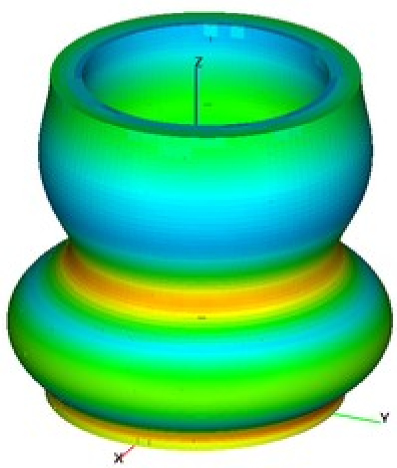
The Output Files

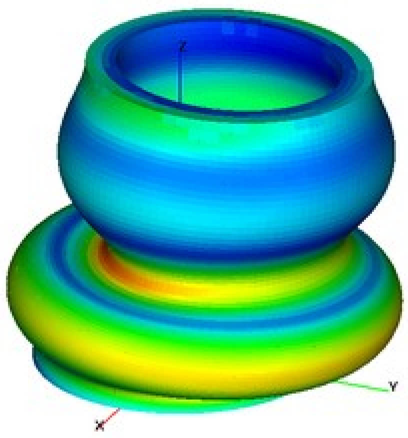
```
-rw-r--r--   1 heltai users 1.3M Feb 21 00:54 solution-0001.0000-000.d2
-rw-r--r--   1 heltai users 1.3M Feb 21 00:54 solution-0001.0000-001.d2
-rw-r--r--   1 heltai users 1.3M Feb 21 00:54 solution-0001.0000-002.d2
-rw-r--r--   1 heltai users 1.3M Feb 21 00:54 solution-0001.0000-003.d2
-rw-r--r--   1 heltai users 1.4M Feb 21 00:54 solution-0001.0000-004.d2
-rw-r--r--   1 heltai users 1.3M Feb 21 00:54 solution-0001.0000-005.d2
-rw-r--r--   1 heltai users 1.3M Feb 21 00:54 solution-0001.0000-006.d2
-rw-r--r--   1 heltai users 1.3M Feb 21 00:54 solution-0001.0000-007.d2
-rw-r--r--   1 heltai users 2.5M Feb 21 00:55 solution-0002.0000-000.d2
-rw-r--r--   1 heltai users 2.5M Feb 21 00:55 solution-0002.0000-001.d2
-rw-r--r--   1 heltai users 2.5M Feb 21 00:55 solution-0002.0000-002.d2
-rw-r--r--   1 heltai users 2.7M Feb 21 00:55 solution-0002.0000-003.d2
-rw-r--r--   1 heltai users 1.4M Feb 21 00:54 solution-0002.0000-004.d2
-rw-r--r--   1 heltai users 1.3M Feb 21 00:54 solution-0002.0000-005.d2
-rw-r--r--   1 heltai users 1.3M Feb 21 00:54 solution-0002.0000-006.d2
-rw-r--r--   1 heltai users 1.3M Feb 21 00:54 solution-0002.0000-007.d2
```

The Final Result

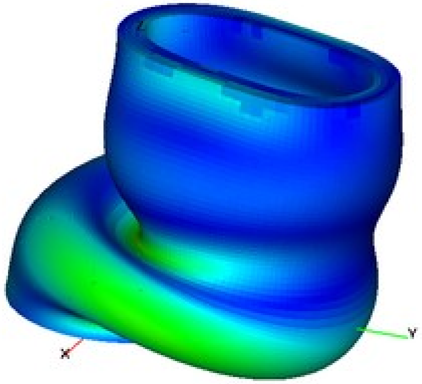


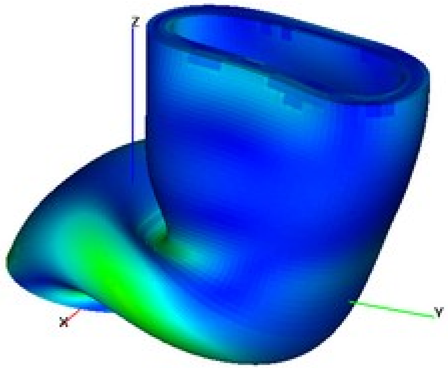
The Final Result - 2





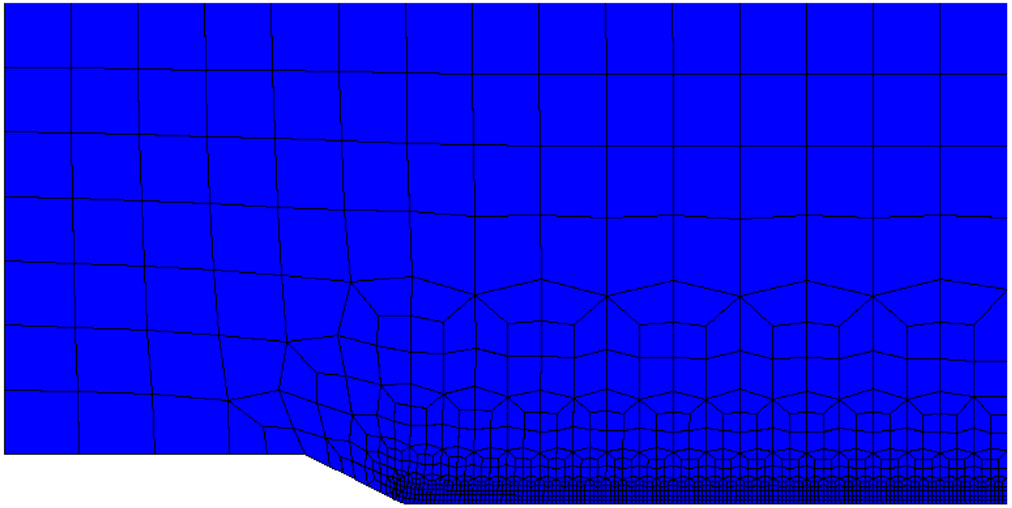
The Final Result - 3





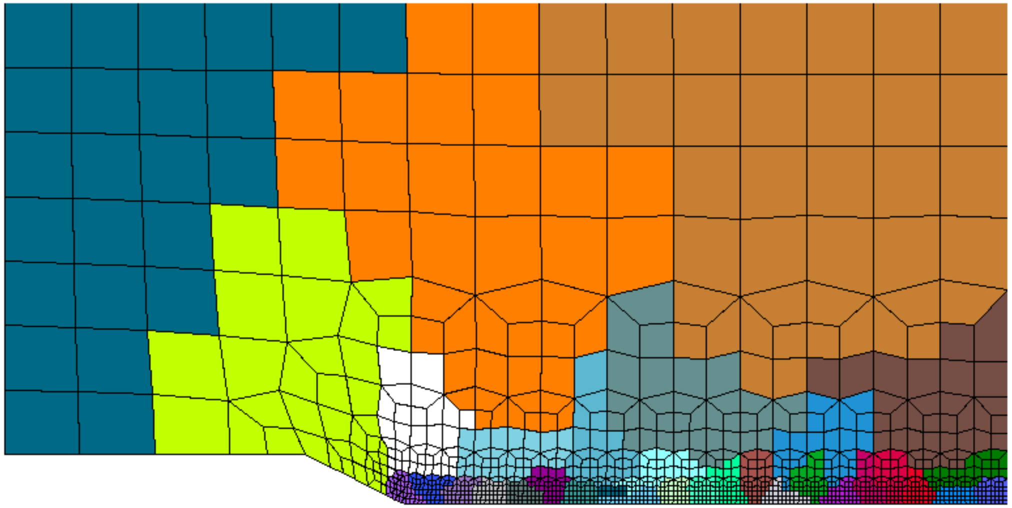
Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



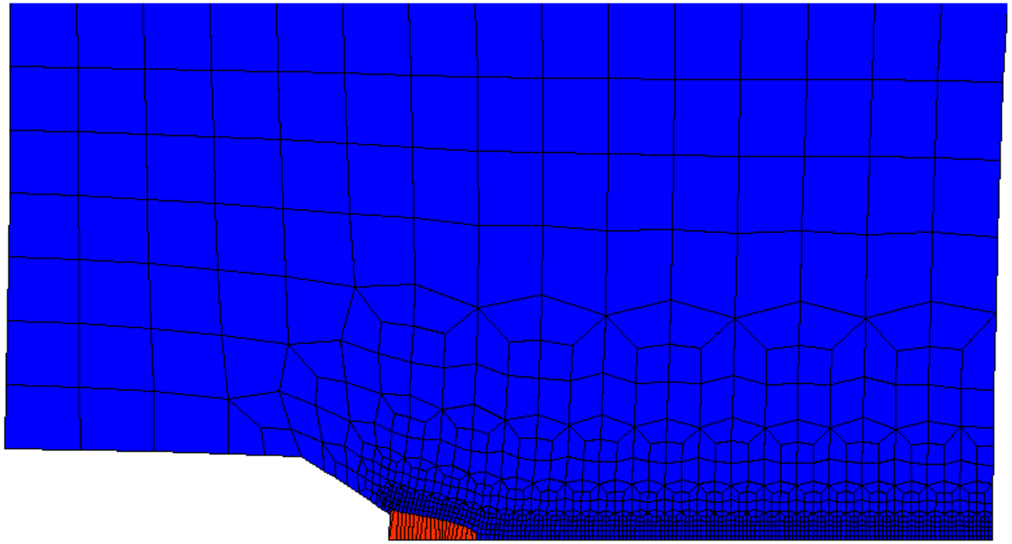
Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



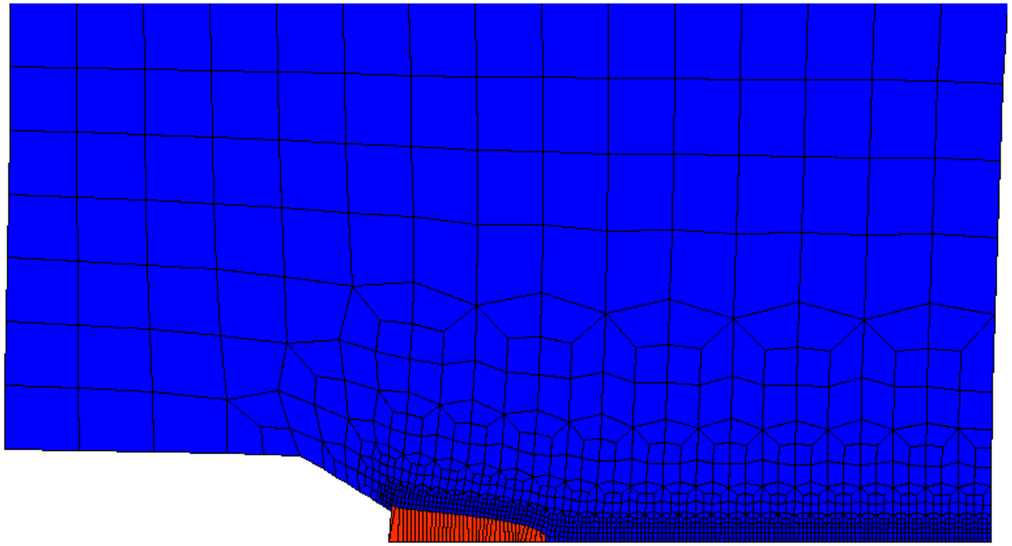
Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



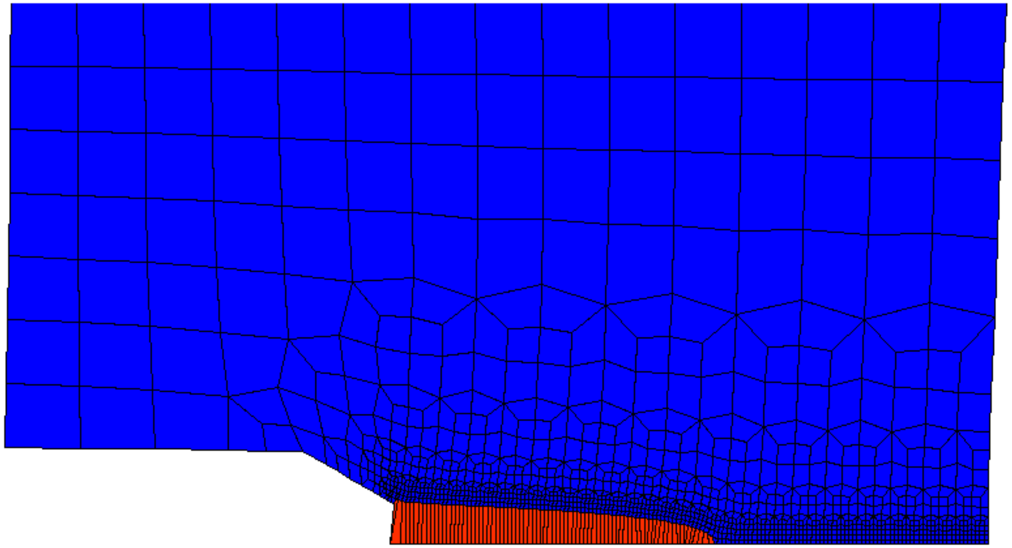
Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



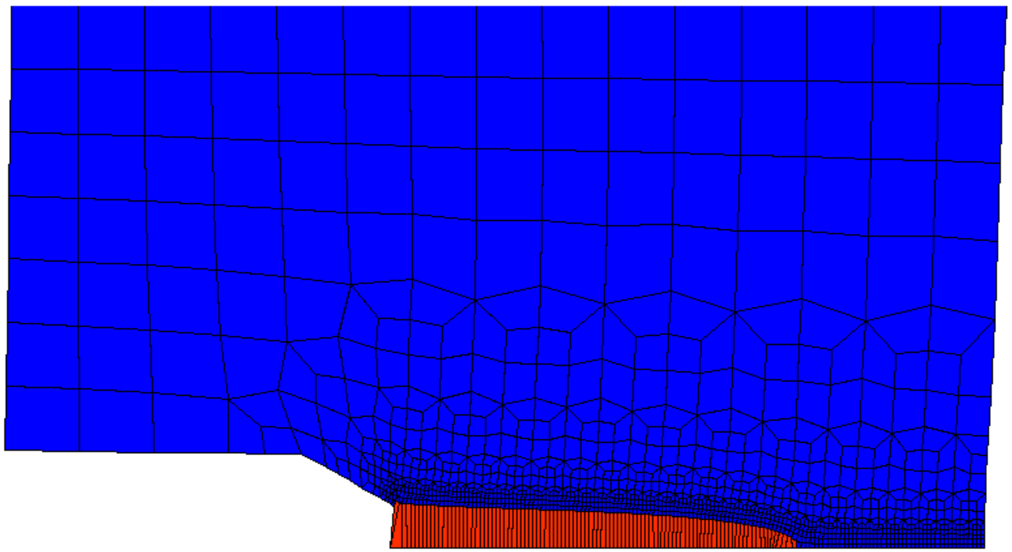
Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



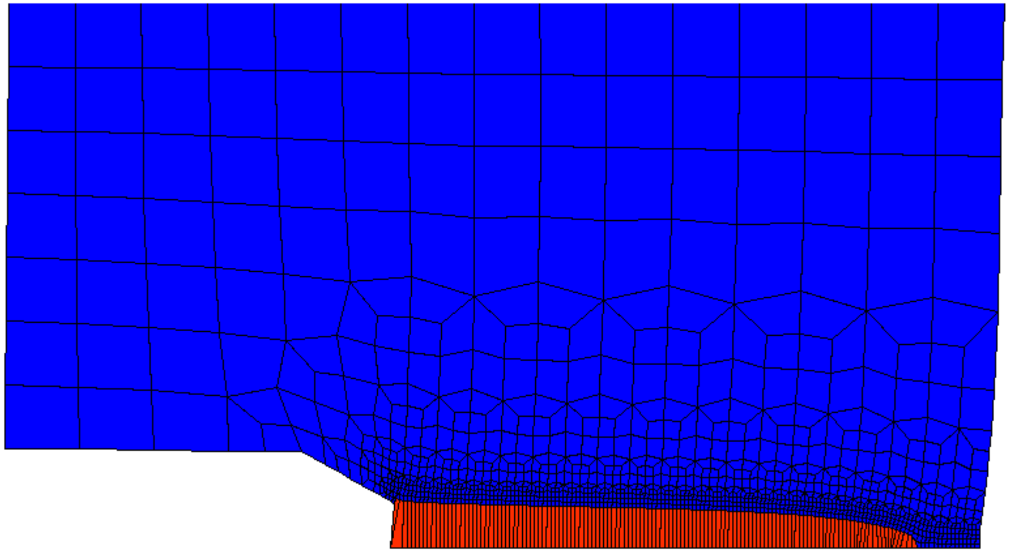
Ex2: Damage in Brittle Materials

- Work done by Jonathan Pitt, PSU



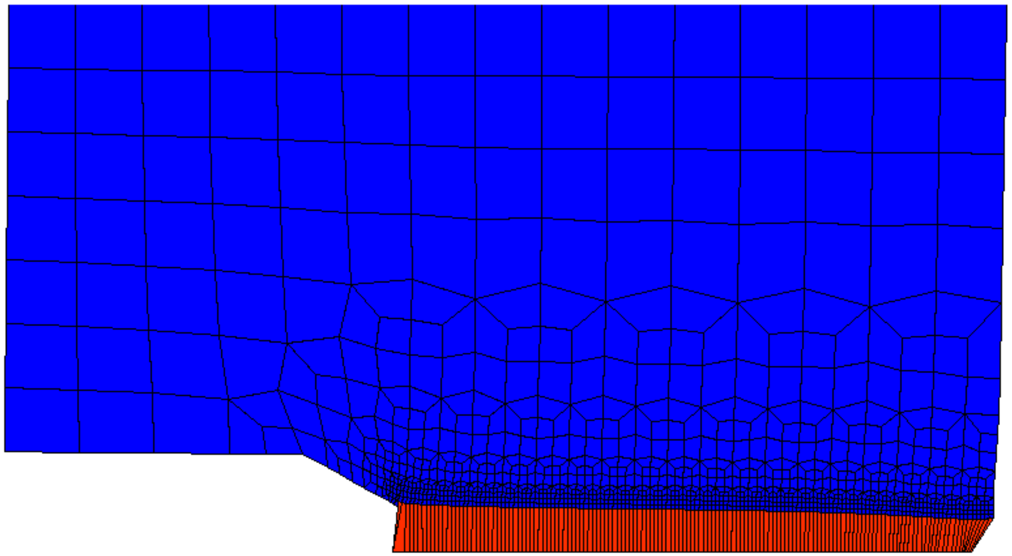
Ex2: Damage in Brittle Materials

- 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