# MERCURY
# Optimized Software for Hybrid Simulation;
## from Pseudo-Dynamic to Hard Real Time

V. Saouma D.H. Kang G. Haussmann

University of Colorado, Boulder

September 6, 2010

## Outline I


1. Introduction
   - Global Context
   - Background
   - NRC/NEES
   - Mercury Overview
2. Analysis
   - Constitutive Models
   - Elements
     - Library
     - State Determination
   - Nonlinear Algorithms
3. Hybrid
   - Matlab
   - c++; General
   - c++; Hybrid Element
   - c++; Coordinate Transformation
4. Improvements
   - RTHS on a Shared Memory Computer

## Outline II


- RTHS on a Computer Cluster; Algorithm
- RTHS on a Computer Cluster; Results

5 Scripting

6 Interface
- Introduction
- Hybrid Pipe
- SCRAMNet Pipe
- Virtual Spring Pipe
- Environmental Variables

7 Xtras

8 Documentation
- Technical Manual
- User's manuals
- Validation Manual

9 Remarks
- Current Development
- Graphical Post Processor
- Applications

## Outline III


- Shake Table vs RTHS

10 Summary

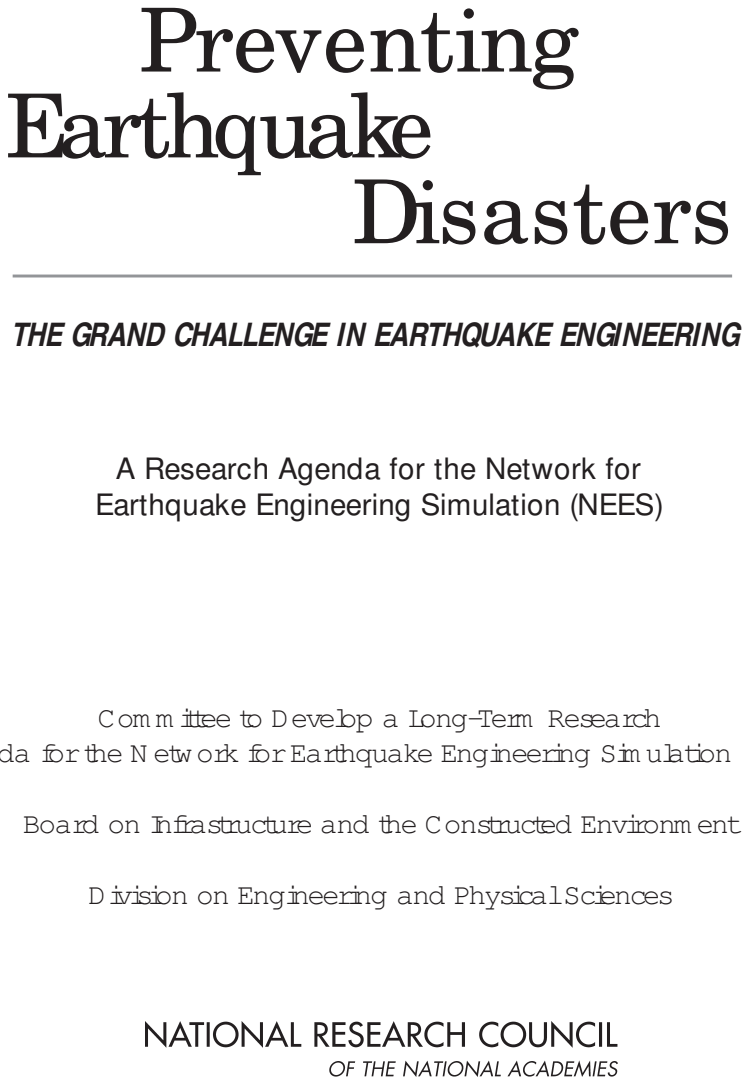
11 Credit

# Global Context

- Numerical simulation remains a major challenge to EQ engineering community.
- We can simulate the explosion of a nuclear bomb, but we can not (“exactly”) simulate seismic response of structures.
- Must rely on experiments of structural components, or systems to capture complex response.
- Finite element modeling capabilities are sophisticated, but not yet “perfect”.
- Many (hundreds of) laboratories
  - Are equipped with digitally controlled actuators/load frames (and LabView), and continue to operate under load/strain/stroke control only.
  - Could benefit from a full nonlinear finite element code which can be plugged in LabView/Simulink to drive their tests

# Background

- Existing software not ideally suited for single site pseudo dynamic (PsD) and RTHS.
- Software may require support of a “facilitator” (such as OpenFresco or SIMCOR) to interact with hardware. Essential for distributed hybrid simulation, a handicap for single site HS.
- The literature provides very little evidence of RTHS of structures (complexity of the numerical substructure, validation with shake table tests).
- Boulder embarked in a project to
  - Develop an optimized software for single site PsD and RTHS (not meant as an alternative to OpenSees, Sap2000, Midas, ...)
  - Perform RTHS of a reinforced concrete frame previously tested on a shake table, and compare results.




## National Research Council

One of the primary goals of NEES is to foster a movement toward integrated computer simulation and physical testing.

Holy Grail: replace all testing by numerical simulation. How can we achieve it?

1. Increase sophistication of real time hybrid simulation (RTHS).
2. Gradually replace/eliminate ST tests by RTHS.
3. Eliminate ST & RTHS

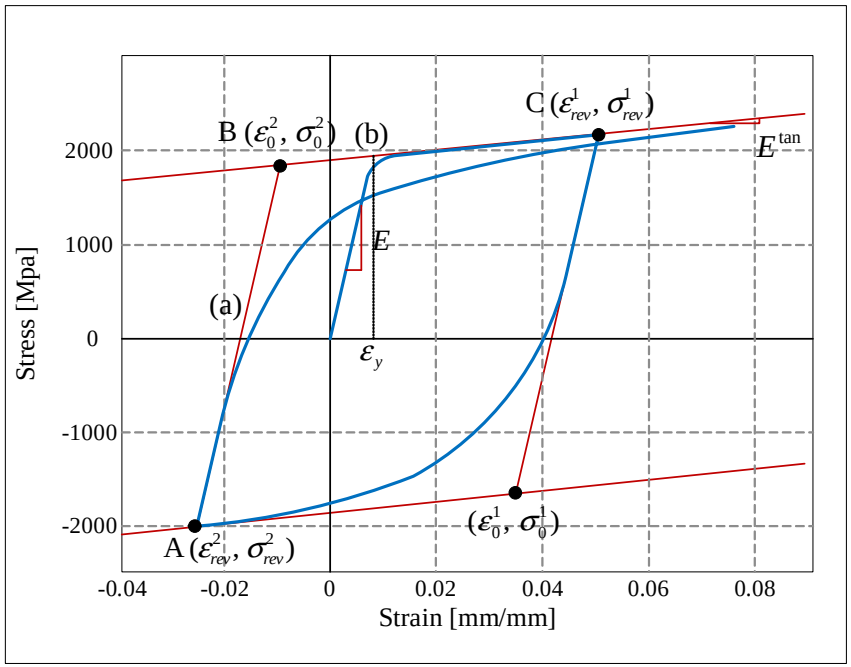
## Mercury Overview

- Two identical versions
  - Matlab (used for prototyping, and pedagogical purposes)
  - c++ for deployment
  - Students found it much easier to start with the Matlab version, and then run the c++ version.
- Has most of the key features likely to be required in a comprehensive nonlinear RTHS of steel or reinforced concrete structures.
- Runs within LabView, Simulink/xPC, or real time Linux.
- Optimized for speed and performance.
- “Battle tested”
  - c++ version with a complex RTHS simulation.
  - Matlab version in a new course (Nonlinear Structural Analysis)
- Extensive documentation, and validation.
- Will be supported by Boulder if you decide to adopt it.

# Constitutive Models; fiber elements

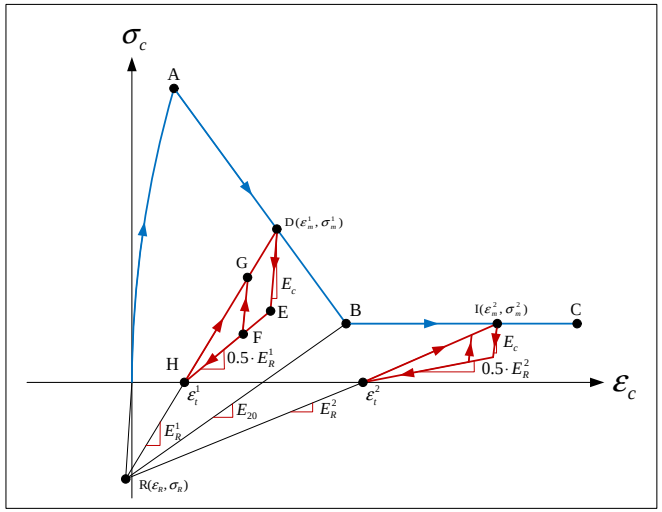
## Steel

- Models with isotropic and kinematic hardening
- Bilinear with isotropic hardening
- Modified Giuffre-Pinto

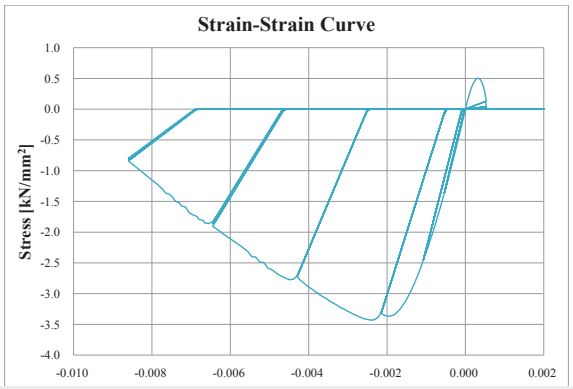



## Concrete

- Modified Kent and Park

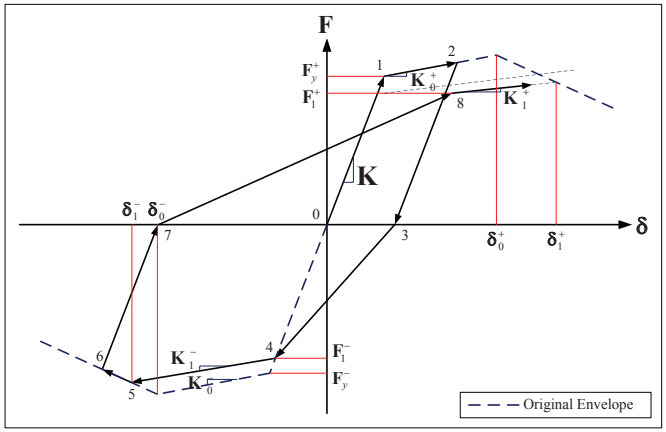



- Anisotropic damage model (LMT/Cachan)

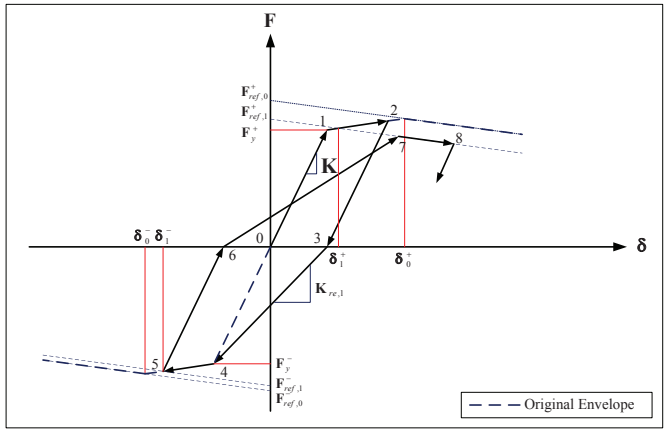


# Constitutive Models; Lumped Plasticity

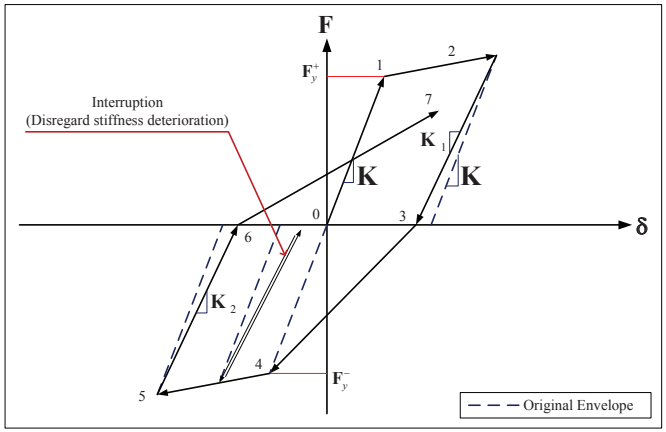
## Model of Ibarra, Medina and Krawinkler (2005)



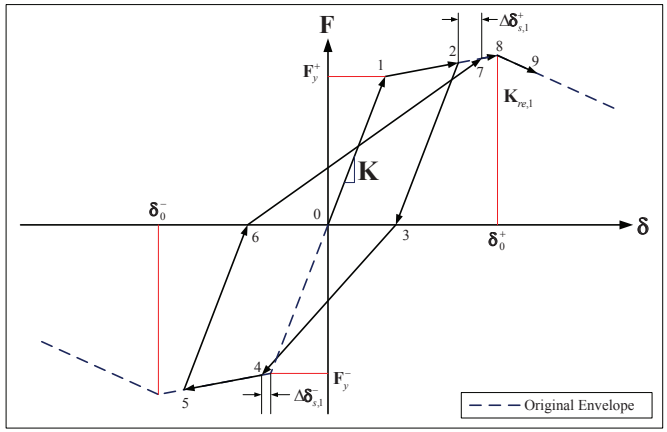

Basic strength deterioration




Post capping strength deterioration



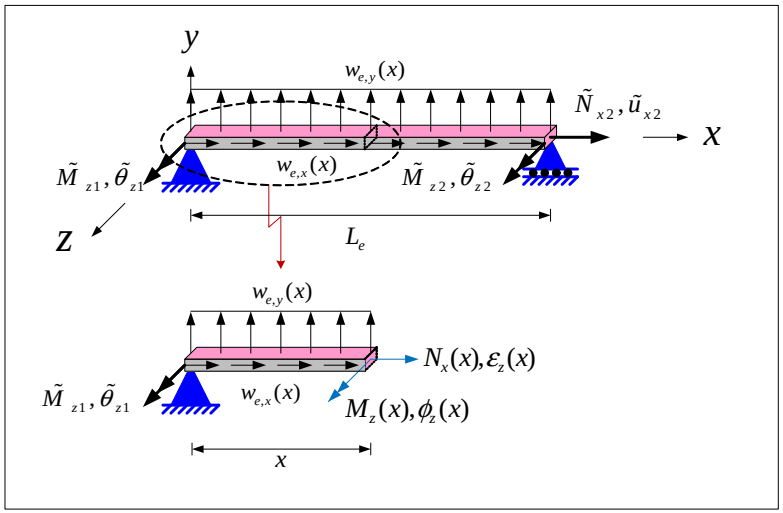

Unloading stiffness deterioration



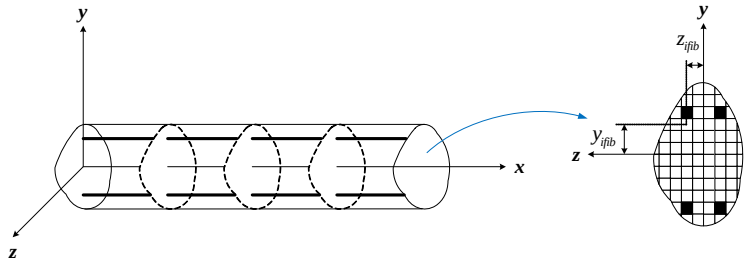

Accelerated reloading stiffness deterioration

# Element Library

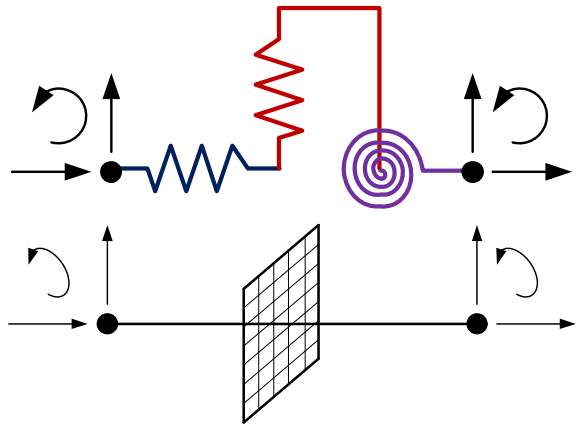
- 2D truss and beam-columns
- Stiffness Based
- Flexibility Based (with or without element iterations)



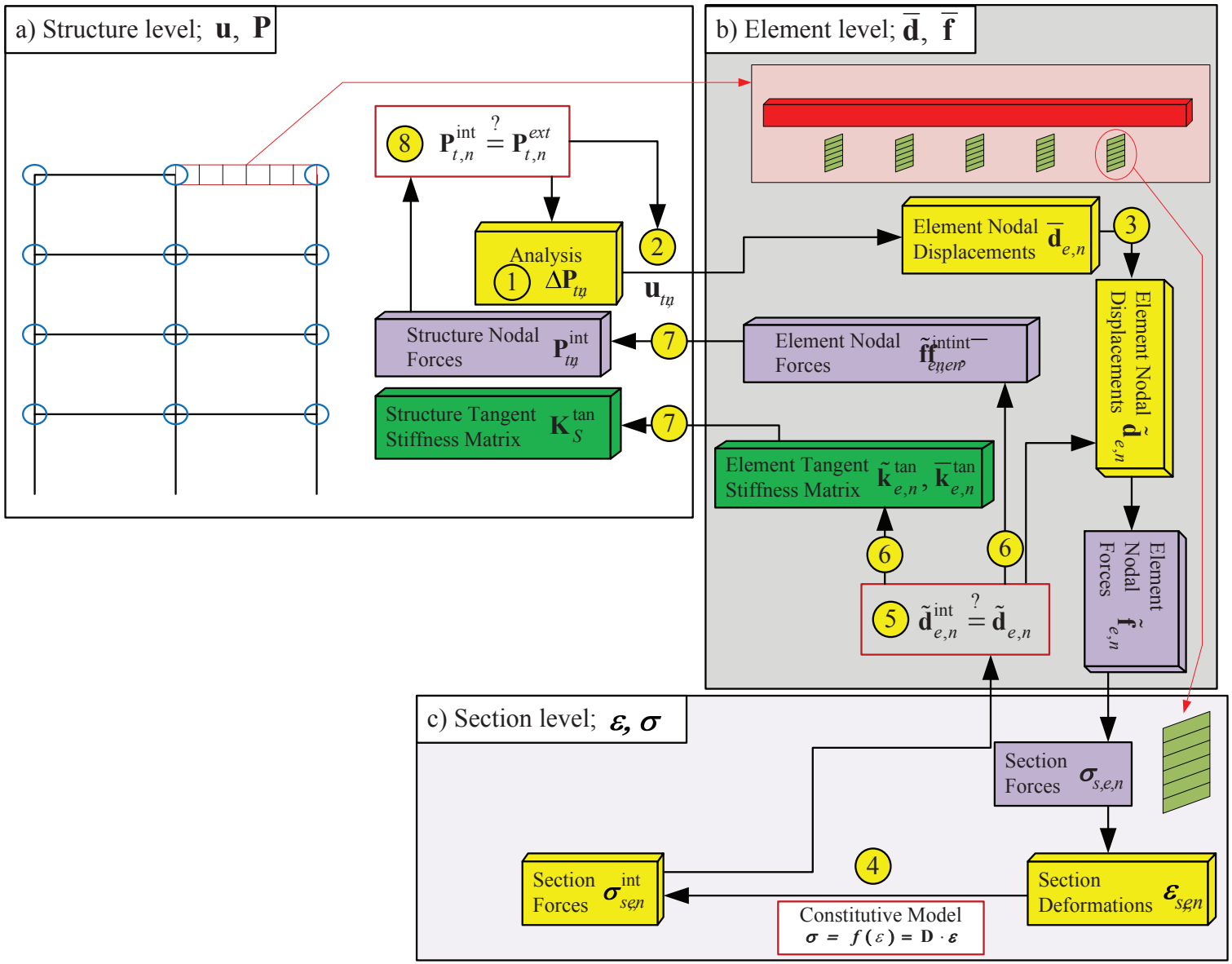

- Layered (fiber) sections




- Zero length elements and sections



# State Determination



# Nonlinear Analysis

## Solution Algorithms

- Linear static
- Eigenvalue analysis
- Initial stiffness
- Newton-Raphson
- Modified Newton-Raphson

**Mixed** User can specify automatic change of solution algorithm if convergence fails

## Convergence Criteria

- Displacement norm
- Force norm
- Energy norm

## Integrators

### Static

- Load control
- Displacement control
- Arc length†

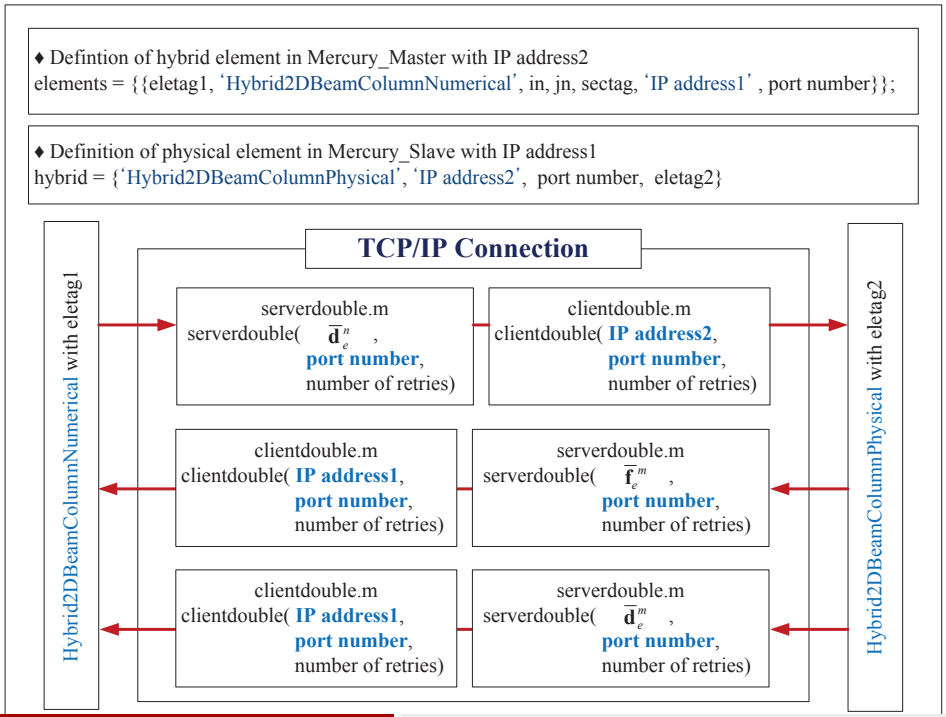
### Transient

- Newmark $\beta$
- Hilber-Hughes-Taylor $\alpha$
- Shing (IS, NR, or MNR)

# Hybrid Capabilities; Matlab

- Supports (non-real time) distributed hybrid element
- Communication through TCP/IP,
- "Physical element" can be another version of Mercury.
- Transfers nodal displacements `hybrid2DBeamColumnNumerical`
- Slave node (if Mercury) can return restoring forces and nodal displacements `hybrid2DBeamColumnPhysical`.

For "debugging" (testing coordinate transformation) and teaching

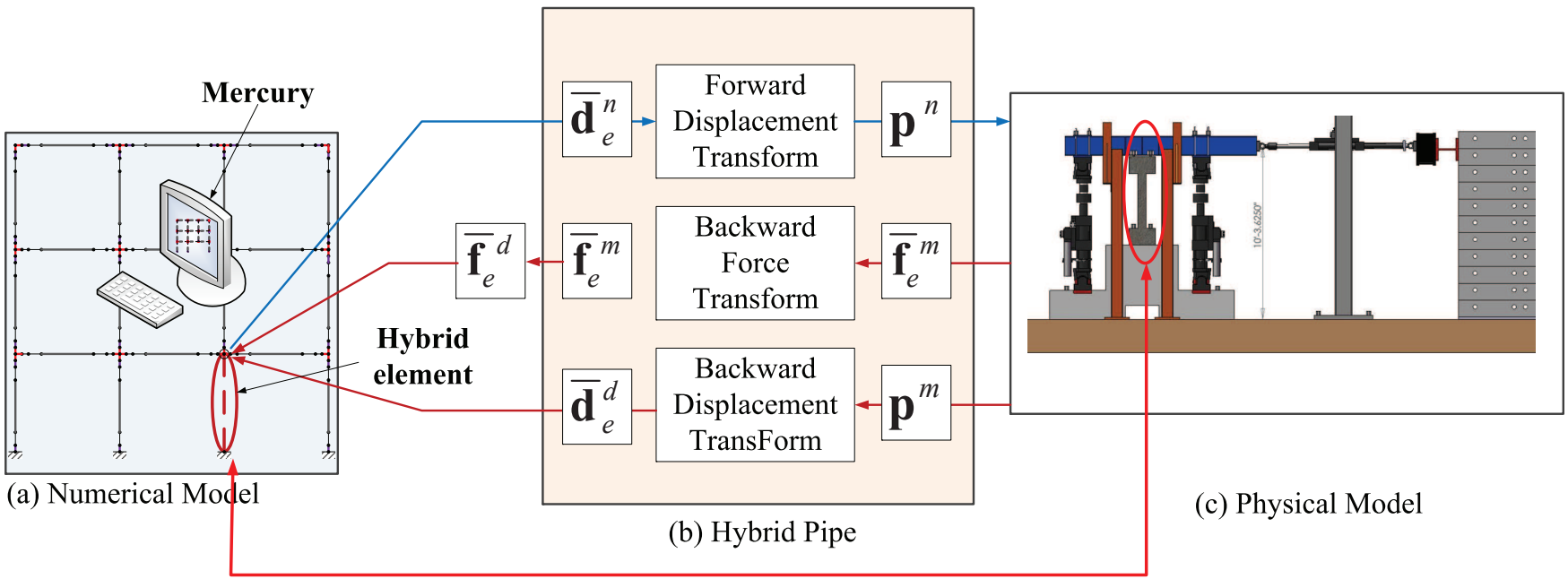


# Hybrid Capabilities; c++

- *Hybrid Element* converts numerical values into physical space values
- *transfer interface* shuttles data between Mercury and the hosting application that controls the physical test.
- Hybrid functionality has been exposed to a command-line program, National Instruments LabView, and Mathworks MATLAB/Simulink.
- Transfer interface can be used to embed Mercury within other applications
- `Environment` assigns a platform on which Mercury runs.
- Mercury performs timestep/increment iteration internally vs. embedded analysis (Mercury performs a single step when triggered by the hosting application)

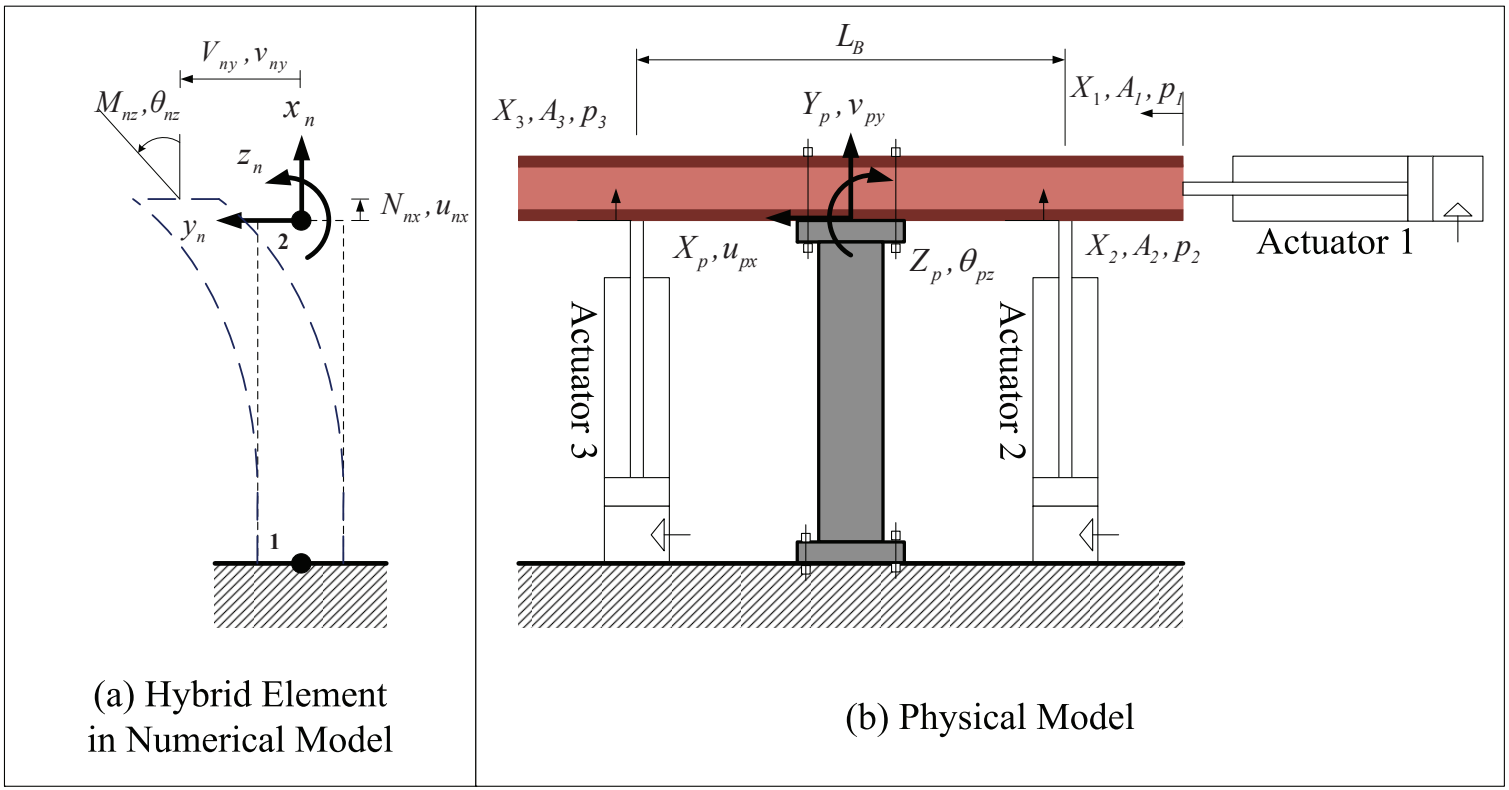
# Hybrid Element

- Appears as an element like any other element in the numerical model computes restoring forces ($\bar{\mathbf{f}}_e^{int}$) in terms of nodal displacements ($\bar{\mathbf{d}}_e^{n}$).
- Uses physical measurements instead of computation to produce element restoring forces.
- Measured quantities include all forces produced by the specimen: stiffness, damping, and inertial forces.
- For nonlinear solution user must provide reasonable estimates of initial and tangent stiffness.



# Coordinate Transformation

- Element nodal displacements in structural model are converted into actuator displacements ($\mathbf{p}^n$) which are applied to the physical specimen
- measured element restoring forces ($\bar{\mathbf{f}}_e^m$) and element nodal displacements ($\bar{\mathbf{d}}_e^m$) from the specimen are then converted into desired element restoring forces ($\bar{\mathbf{f}}_e^d$) and element nodal displacements ($\bar{\mathbf{d}}_e^d$) used in the numerical model

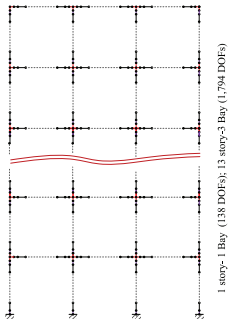


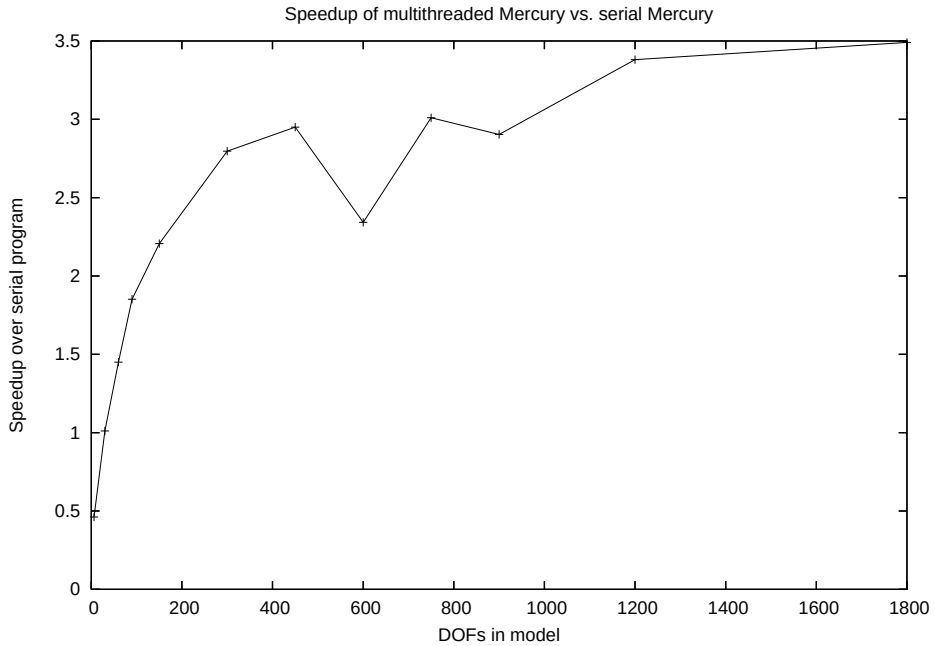
# RTHS on a Shared Memory Computer

Multithreading based on Intel MKL library

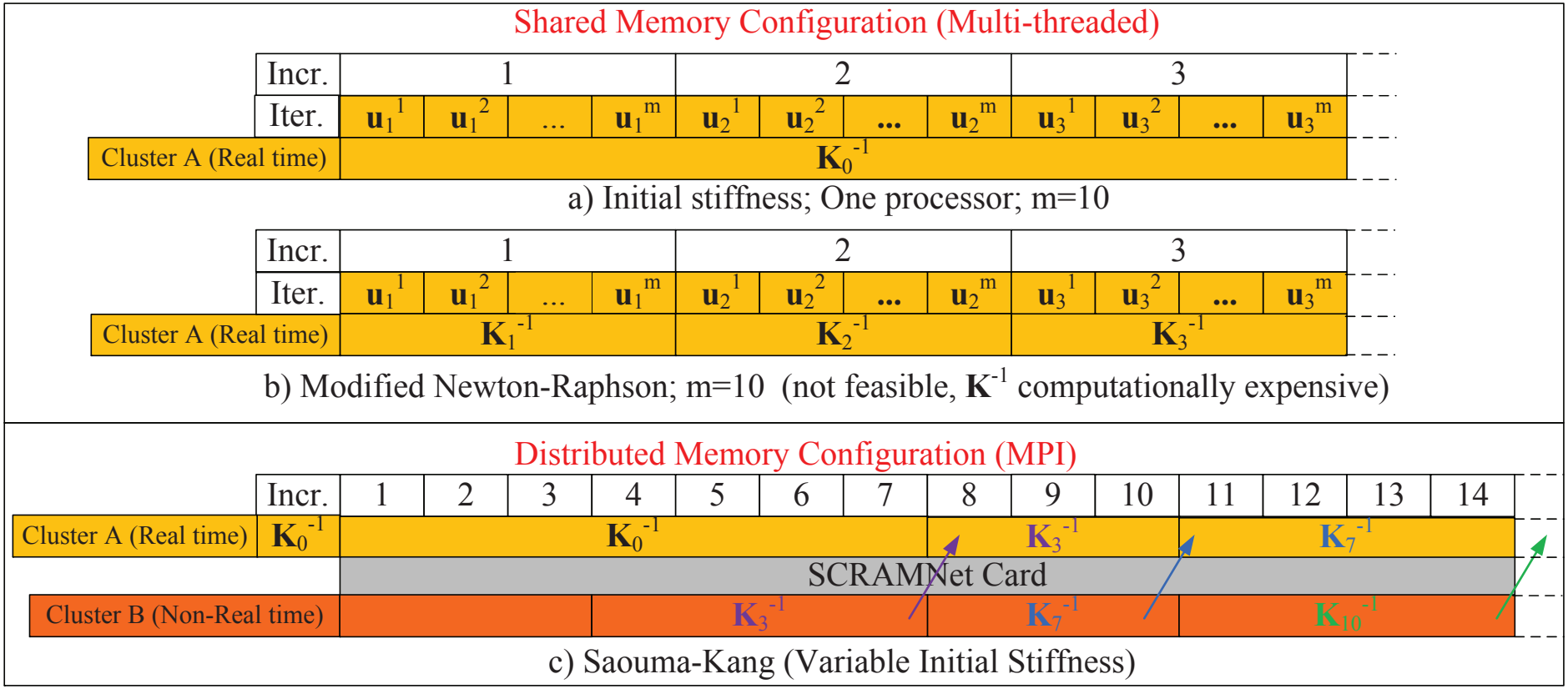
- Linear Solver: PARDISO (thread-safe, high-performance, robust, memory efficient software for large sparse symmetric matrices).
- Force Recovery: is multi-threaded for parallel operations.

Test Problem to Assess benefit of multi-threading analysing a R/C frame of increasing size.



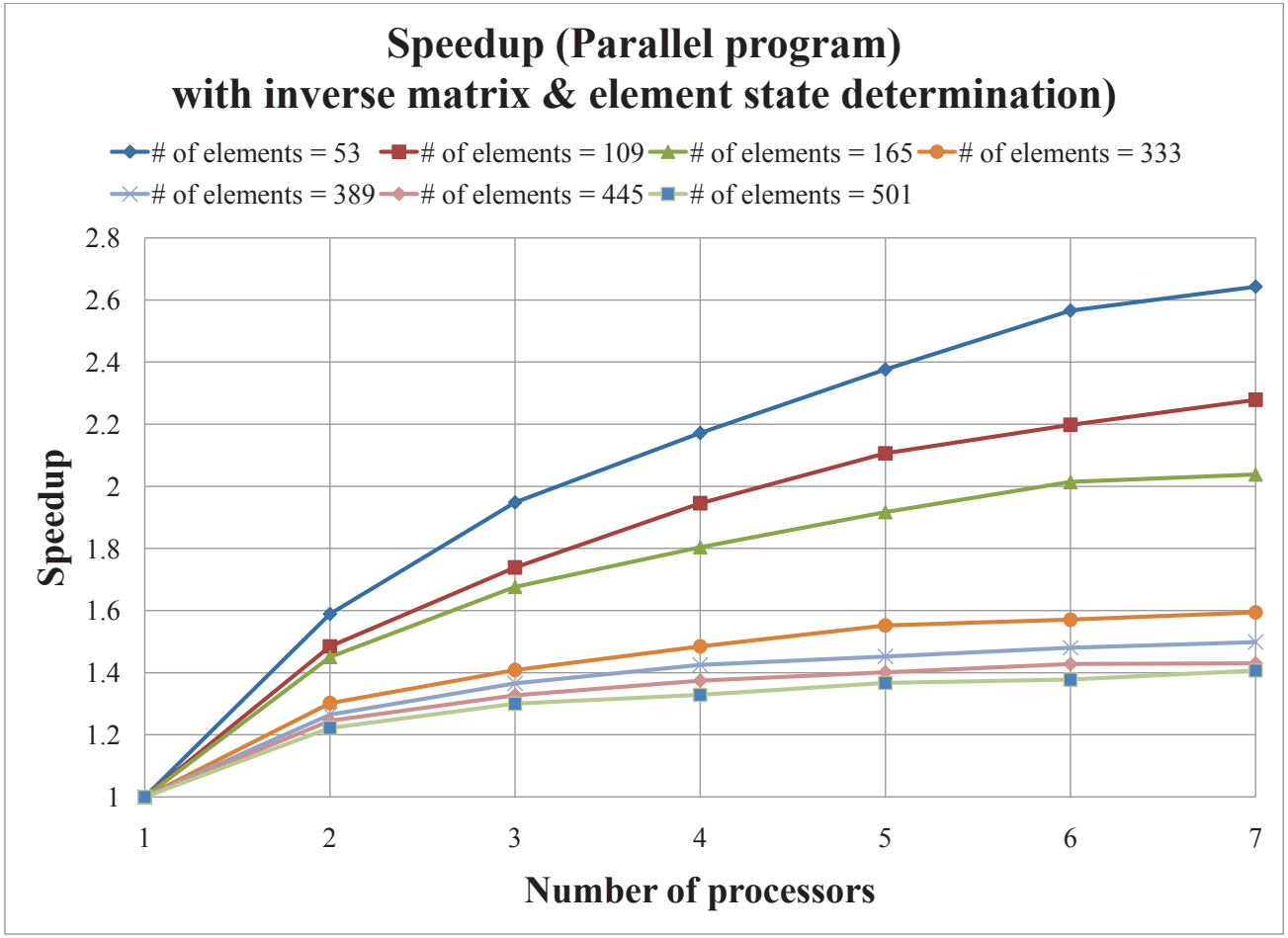



# RTHS on a Computer Cluster;Algorithm




Two clusters: A (real time) dedicated to hybrid simulation (multiple CPU for force recoveries); B (not real time) dedicated to evaluation of $\mathbf{K}_t^{-1}$. Swap inverse tangent matrix when completed.

# RTHS on a Computer Cluster; Results



# Scripting Interface: LUA

- Embedded in code
- MATLAB like syntax (cell arrays), and can easily define functions and variables
- can use alphanumeric (as opposed to just numeric) tags

## Listing 1: Example

```
barslipsectionf = 'BSColDFSection'; barslipspringf = 'BSColDFSS';
columnsection = 'ColDSection'; barslipspring  = 'BSColDSS';
barslipsection = 'BSColDSection'; columnrigidsection = 'ColRigidSection';
if (colfloor==1) then
node1 = columnnodename(colbay,colfloor,1);
node2 = columnnodename(colbay,colfloor,2);
barslipbottom  = {string.format('columnbsb_%d_%d',colbay,colfloor),
'InterfaceElement2D', node1, node2,
{ {barslipspringf, {1,0,0} } }, { {barslipsectionf}}, {0,1,0},{-1,0,0} }
plasticcolumn1 = { string.format('columnpl1_%d_%d',colbay,colfloor),
'StiffnessBased2DBeamColumn', node2, node3, {columnsection, nIp_stif} }
flexcolumn = { string.format('columnflx_%d_%d',colbay,colfloor),
'FlexibilityBased2DBeamColumn', node3, node4, {columnsection,nIp_flex},
flexparams }
plasticcolumn2 = { string.format('columnpl2_%d_%d',colbay,colfloor),
'StiffnessBased2DBeamColumn',node4, node5, {columnsection, nIp_stif} }
barsliptop     = { string.format('columnbst_%d_%d',colbay,colfloor),
'InterfaceElement2D', node5, node6,
```

# Interface and Coupling to Hosting Application

- Embedded in code
- MATLAB like syntax
- can easily define functions and variables
- Embedded version of Mercury which runs within another application (the hosting application)
- C/C++ code connects embedded Mercury to hosting application (LabView,Simulnk, etc.) which can
  - Transfer data quantities between embedded Mercury and external application
  - Allows the hosting application to 'trigger' embedded Mercury to perform a single timestep or increment
- Input files used for command-line based Mercury can be used by embedded Mercury with minimal changes

# Hybrid Pipe

- Abstract interface which connects Mercury hybrid element to a physical specimen
- Different hybrid pipes must be used/written depending on specific laboratory hardware
- Example: SCRAMNETpipe is specifically designed to work with MTS SCRAMNet-based actuators
- Laboratory hardware differences are contained in the Hybrid Pipe code, without affecting other code
- Two labs with differing actuator setups could use the same input file by changing the Hybrid Pipe used

# SCRAMNet Pipe

- Hybrid Pipe specifically written to work with SCRAMNet-driven MTS actuators
- Displacements and Forces of a hybrid element are written/read from the proper SCRAMNet memory locations
- Timestepping is synchronized from a SCRAMNet interrupt, locking Mercury's timestep rate
- Useful starting point for writing new Hybrid Pipe classes to handle different hardware

# Virtual Spring Pipe

- A Hybrid pipe that simulates a physical specimen
- Useful for testing and verifying simulation setup without actually running actuators
- Simple spring response: $F = Kd$
- Can delay the response by N timesteps, to simulate actuator lag for testing purposes

# Transient Recorders

[fragile]

`transientrecorder` will accumulate and record displacements and forces for a list of nodes and write them only after simulation is completed in order to avoid interrupting the real-time simulation.

One should specify:

- Number of values to record
- Number of DOFs in the model
- Filename to save the data into
- Names/numbers of all the nodes to record

# Environmental Variables

Describes which platform Mercury is running on. There are several possible values:

1. `CommandLine`
2. `CommandLineTriggered`
3. `LabView`
4. `Simulink`

For the `CommandLine` environment, one should use the solve method to perform analysis, specifying the number of timesteps to use. For any other environment, the user should set some variables describing the model, analysis and loading to use; the environment will properly trigger Mercury for the specified number of timesteps.

```
if (Environment == "CommandLine") then
    print("Error: run mercury with the --external option")
    transientanalysis:solve(30000)
else
    CommandLineTriggeredHook = transientanalysis
    LabViewHook = transientanalysis
    SimulinkHook = transientanalysis
    ModelRef = model
    LoadingRef = earthquakeloading
    numsteps = 30001
    rampsteps = 1000 --[[ramp up initial values]]
end
```

## Special Features

### Speed

- Shared memory: multithreaded with Intel MKL library
- Distributed memory: Two groups of processors (MPI)
  - Hybrid simulation in real time, element force recovery spread in a cluster of $n$ processors
  - Continuous update of tangent stiffness matrix in a “parallel” non real time computer cluster; Initial stiffness matrix is continuously overwritten

**Refinements of Shing’s Method** to allow $10n$ iterations per increment of 0.01 sec.

**Pushover Analysis** Allows application of displacement over multiple dof following application of vertical loads (single analysis).

**High performance concrete model** (Developed by LMT/Cachan)

# Technical Manual

Over 250 pages of detailed technical documentation for element formulations, constitutive models, integration schemes, hybrid elements.

Draft

With the section tangent flexibility matrices at end of the last convergence in structural level given by

$$\mathbf{c}_{s,e,n}^{tan,k=1,j=0}(x) = \mathbf{c}_{s,e,n-1}^{tan}(x)$$

the linearization of the section force-deformation relation yields the incremental section deformation vectors.

$$\delta\boldsymbol{\varepsilon}_{s,e,n}^{k=1,j=1}(x) = \mathbf{c}_{s,e,n}^{tan,k=1,j=0}(x) \cdot \delta\boldsymbol{\sigma}_{s,e,n}^{k=1,j=1}(x)$$

The section deformation vectors are updated to the state that corresponds to point **B** in Fig. 3.13(b), and the updated section deformation vector will be given by

$$\boldsymbol{\varepsilon}_{s,e,n}^{k=1,j=1}(x) = \boldsymbol{\varepsilon}_{s,e,n}^{k=1,j=0}(x) + \delta\boldsymbol{\varepsilon}_{s,e,n}^{k=1,j=1}(x)$$

For the sake of simplicity we will assume that the section force-deformation relation is explicitly known, then the section deformation vectors $\boldsymbol{\varepsilon}_{s,e,n}^{k=1,j=1}(x)$ will correspond to internal section force vectors $\boldsymbol{\sigma}_{s,e,n}^{int,k=1,j=1}(x)$ and updated section tangent flexibility matrices $\mathbf{c}_{s,e,n}^{tan,k=1,j=1}(x)$ in Fig. 3.13(b) can be defined.

The residual section force vectors are then determined

$$\boldsymbol{\sigma}_{s,e,n}^{R,k=1,j=1}(x) = \boldsymbol{\sigma}_{s,e,n}^{k=1,j=1}(x) - \boldsymbol{\sigma}_{s,e,n}^{int,k=1,j=1}(x)$$

and are transformed into residual section deformation vectors $\boldsymbol{\varepsilon}_{s,e,n}^{R,k=1,j=1}(x)$

$$\boldsymbol{\varepsilon}_{s,e,n}^{R,k=1,j=1}(x) = \mathbf{c}_{s,e,n}^{tan,k=1,j=1}(x) \cdot \boldsymbol{\sigma}_{s,e,n}^{R,k=1,j=1}(x)$$

The residual section deformation vectors are thus the linear approximation of the deformation error made in the linearization of the section force-deformation relation (Fig. 3.13(b)). While any suitable section flexibility matrix can be used to calculate the residual section deformation vector, the section tangent flexibility matrices offer the fastest convergence rate.

The residual section deformation vectors are integrated along the element using the complimentary principle of virtual work to obtain the residual element nodal displacement vector.

$$\tilde{\mathbf{d}}_{e,n}^{R,k=1,j=1} = \int_0^{L_e} \mathbf{N}_{f,e}(x)^T \cdot \boldsymbol{\varepsilon}_{s,e,n}^{R,k=1,j=1}(x)\mathrm{d}x$$

At this stage the first iteration ($j = 1$) is completed. The final element and section states for $j = 1$ correspond to point **B** in Fig. 3.13. The residual section deformation vectors $\boldsymbol{\varepsilon}_{s,e,n}^{R,k=1,j=1}(x)$ and the residual element nodal displacement vector $\tilde{\mathbf{d}}_{e,n}^{R,k=1,j=1}$ were determined in the first iteration, but the corresponding element nodal displacement vector have not yet been updated. Instead, they constitute the starting point of the remaining steps within iteration loop $j$.

The presence of residual element nodal displacement vector $\tilde{\mathbf{d}}_{e,n}^{R,k=1,j=1}$ will violate compatibility, since elements sharing a common node would now have different element nodal displacement vector. In order to restore the inter-element compatibility, corrective force vector



Draft

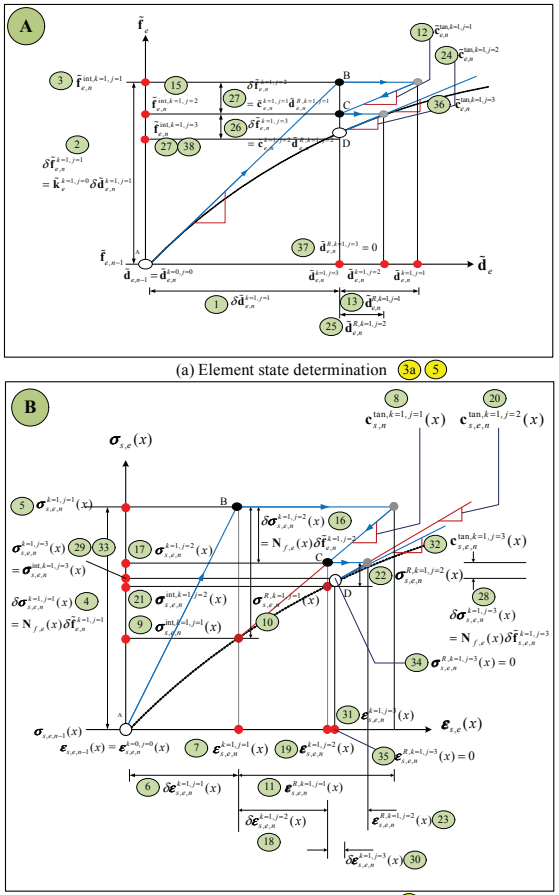



Fig. 3.13 Element and section state determinations for flexibility-based 2D beam-column element with Newton-Raphson iteration loop in element level

# User's Manuals

One for Matlab version, and one for c++

Draft

- `Tol`: Convergence criteria on the residuals

Example:
`FlexibilityBased2DBeamColumn` in Sec. A.4, then we could have: `StrMiter = 20; EleMiter = 50; Convergence = 'ForceNorm'; ConvergenceEle = 'EnergyNorm'; Tolerance = 1.0e-8`

### A.3 Geometry Block

The geometry block defines nodal coordinates and their constraints assuming a right handed coordinate system.

#### A.3.1 Nodal coordinates

The `nodcoord` assigns coordinates of nodes.

| | | | |
|---|---|---|---|
| `nodcoord` | = | { | $\texttt{nodtag}_1,\ x_1,\ y_1\ [z_1]$ ; |
| | | | ... ; |
| | | | $\texttt{nodtag}_i,\ x_i,\ y_i\ [z_i]$ ; |
| | | | ... ; |
| | | | $\texttt{nodtag}_n,\ x_n,\ y_n\ [z_n]$ } |

**for example:**

```
Node  =  {  1,  0.0,  0.0  ;
            2,  1.0,  3.0  ;
            3,  2.0,  0.0  }
```

#### A.3.2 Boundary condition

The `constraint` command assigns boundary conditions to the nodes. Each node has to have as many constraint as d.o.f's per node.

| | | | |
|---|---|---|---|
| `constraint` | = | { | $\texttt{nodtag}^1,\ id_1^1,\ id_2^1\ [\ id_3^1,\ id_4^1,\ id_5^1,\ id_6^1]$ ; |
| | | | ... ; |
| | | | $\texttt{nodtag}^i,\ id_1^i,\ id_2^i\ [\ id_3^i,\ id_4^i,\ id_5^i,\ id_6^i]$ ; |
| | | | ... ; |
| | | | $\texttt{nodtag}^n,\ id_1^n,\ id_2^n\ [\ id_3^n,\ id_4^n,\ id_5^n,\ id_6^n]$ } |

Where 0 corresponds to a free dof, and 1 to a fixed one. For example:

```
constraint  =  {  3,  1,  1  ;
                  5,  1,  0  }
```

### A.4 Element Block

The `elements` command defines element type, nodal connectivity, and basic sectional information. These may vary with the element type.



## Chapter 1

# MERCURY USER'S MANUAL FOR C++

This document[1] describes the input for C++ version of Mercury. The C++ version uses the Lua scripting language[2] (analogous to TCL in OpenSees).

### 1.1 nodes

The "nodes" command defines nodal coordinates, and nodal masses if material densities are not in materials.

| | | | |
|---|---|---|---|
| `nodes` | = | { | { $\texttt{nodtag}_1,\ x_1,\ y_1\ [, z_1]\ [,\text{'mass'},\ mx_1,\ my_1\ [, mz_1]]$ }; |
| | | | { ... }; |
| | | | { $\texttt{nodtag}_i,\ x_i,\ y_i\ [, z_i]\ [,\text{'mass'},\ mx_i,\ my_i\ [, mz_i]]$ }; |
| | | | { ... }; |
| | | | { $\texttt{nodtag}_n,\ x_n,\ y_n\ [, z_n]\ [,\text{'mass'},\ mx_n,\ my_n\ [, mz_n]]$ } }; |

- $\texttt{nodtag}_i$: Tag of the $i^{th}$ node
- $x_i$, $y_i$, and $z_i$ are node coordinates of node $i$ at each global coordinate.
- $mx_i$, $my_i$, and $mz_i$ are mass quantities of node $i$ at each global coordinate.

### 1.2 elements

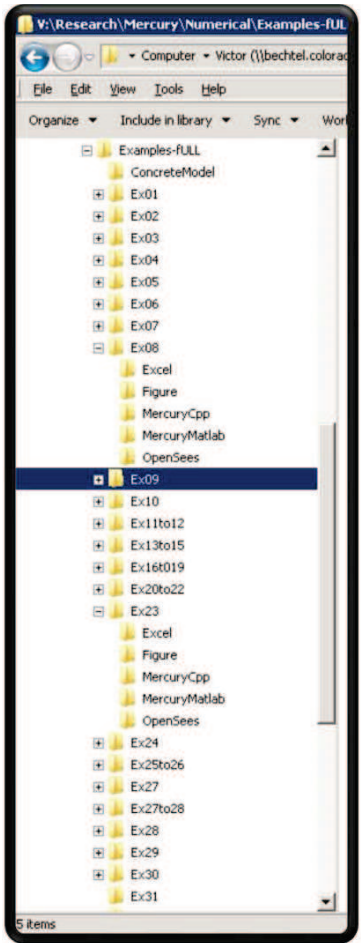
The "elements" command defines element type, nodal connectivity, and basic section information. These may vary with the element type.

| | | | |
|---|---|---|---|
| `elements` | = | { | { Definition of $\text{element}_1$ }; |
| | | | { ... }; |
| | | | { Definition of $\text{element}_i$ }; |
| | | | { ... }; |
| | | | { Definition of $\text{element}_n$ } }; |

[1] In this preliminary version of Mercury, no attempt has been made to simplify (generate/automate) data entry, and there is not (yet) a mesh generator for the program. Those are simple future developments.
[2] http://www.lua.org/

# Validation Manual

Over 30 examples used for validations. Each problem stored in a separate folder containing: a) Matlab input file; b)c++ (lua) input file; c) OpenSees (tcl) input file; and d) Excel/visio files for results



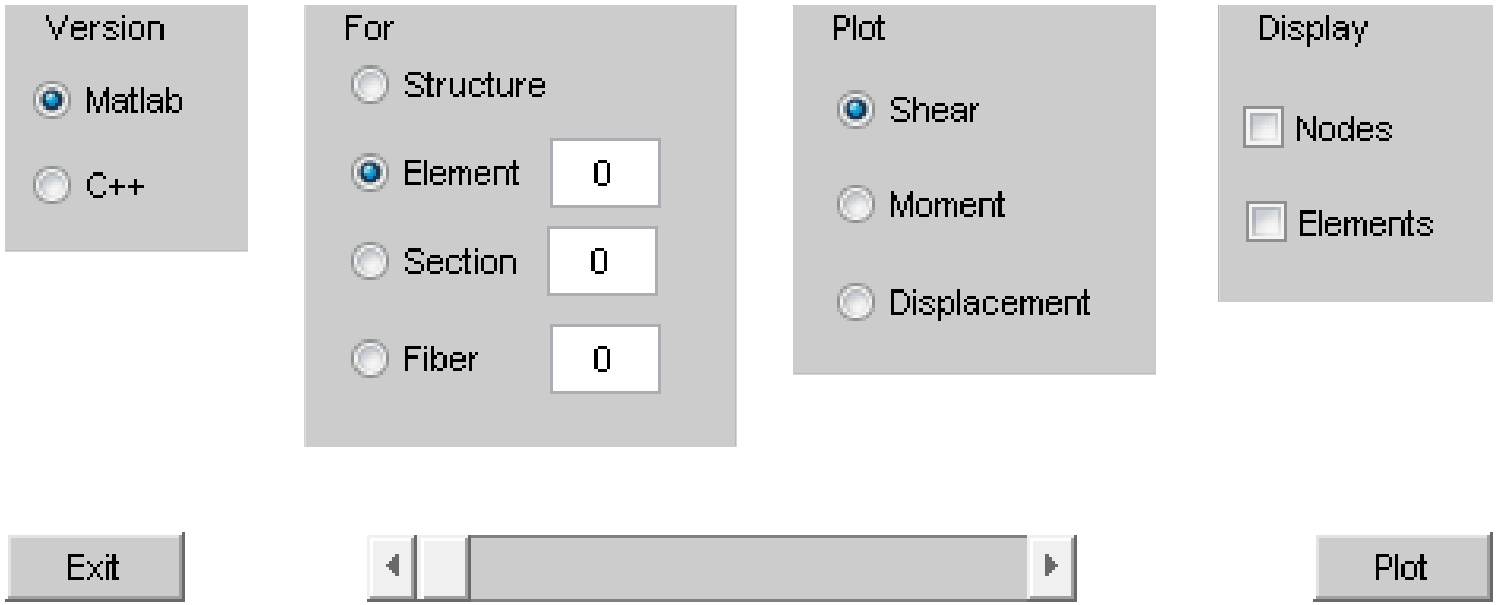

| Example | Algorithm | Element | Section | Material | Analysis 1 | Analysis 2 |
|---|---|---|---|---|---|---|
| Ex01 | Linear | Simple truss | General | Elastic | Static | Newmark beta |
| Ex02 | Linear | Simple truss | General | Elastic | Load control | HHT |
| Ex03 | Linear | Simple truss | General | Elastic | Load control | |
| Ex04 | Linear | Simple truss | General | Elastic | Disp control | |
| Ex05 | Mixed | Simple truss | General | Elastic and Hardening | Static | Newmark beta |
| Ex06 | Mixed | Simple truss | General | Elastic and Hardening | Load control | HHT |
| Ex07 | Mixed | Simple truss | General | Elastic and Hardening | Newmark beta | |
| Ex08 | Mixed | Simple truss | General | Elastic and Hardening | Disp control | HHT |
| Ex09 | Mixed | Simple truss | General | Elastic and Bilinear | Load control | Newmark beta |
| Ex10 | Mixed | Simple truss | General | Elastic and ModGMP | Load control | HHT |
| Ex11 | Mixed | Simple truss | General | Damage2 | Cyclic disp control | |
| Ex12 | Mixed | Simple truss | General | ModKP | Cyclic disp control | |
| Ex13 | Mixed | Simple truss | General | Hardening | Cyclic disp control | |
| Ex14 | Mixed | Simple truss | General | Hardening | Cyclic disp control | |
| Ex15 | Mixed | Simple truss | General | Hardening | Cyclic disp control | |
| Ex16 | Mixed | Simple truss | General | Bilinear | Cyclic disp control | |
| Ex17 | Mixed | Simple truss | General | Bilinear | Cyclic disp control | |
| Ex18 | Mixed | Simple truss | General | ModGMP | Cyclic disp control | |
| Ex19 | Mixed | Simple truss | General | ModGMP | Cyclic disp control | |
| Ex20 | Mixed | SBC | Layer | Hardening | Cyclic disp control | HHT |
| Ex21 | Mixed | FBC1 | Layer | Hardening | Cyclic disp control | Shing |
| Ex22 | Mixed | FBC2 | Layer | Hardening | Cyclic disp control | |
| Ex23 | Mixed | SBC and zero-length | General | Elastic and Bilinear | Load control | |
| Ex24 | Mixed | SBC and zero-length section | General | Elastic and Bilinear | Load control | |
| Ex25 | Mixed | SBC | Layer | Hardening | Multiple disp control | |
| Ex26 | Mixed | FBC1 | Layer | Hardening | Multiple disp control | |
| Ex27 | Mixed | Ghanoumn single column | Layer | ModGMP and ModKP | | HHT |
| Ex28 | Mixed | Ghanoumn single column | Layer | ModGMP and Damage2 | | HHT |
| Ex29 | Mixed | Ghannoum 3 bay 3 floor | Layer | Mixed materials | | HHT/Shing |

# Current Development

- Timoshenko beam
- Graphical (OpenGL based) post-processor
- Geometric nonlinearity
- Limit elements
- Online tutorials
- Mesh generator
- Translator from Matlab to Lua
- Improved documentation for integration with hybrid simulation

# Graphical Post Processor

A Graphical Post Processor (written in Matlab) is under current development. It will support both the Matlab and the C++ versions, and will provide a GUI through which user could visualize various results.



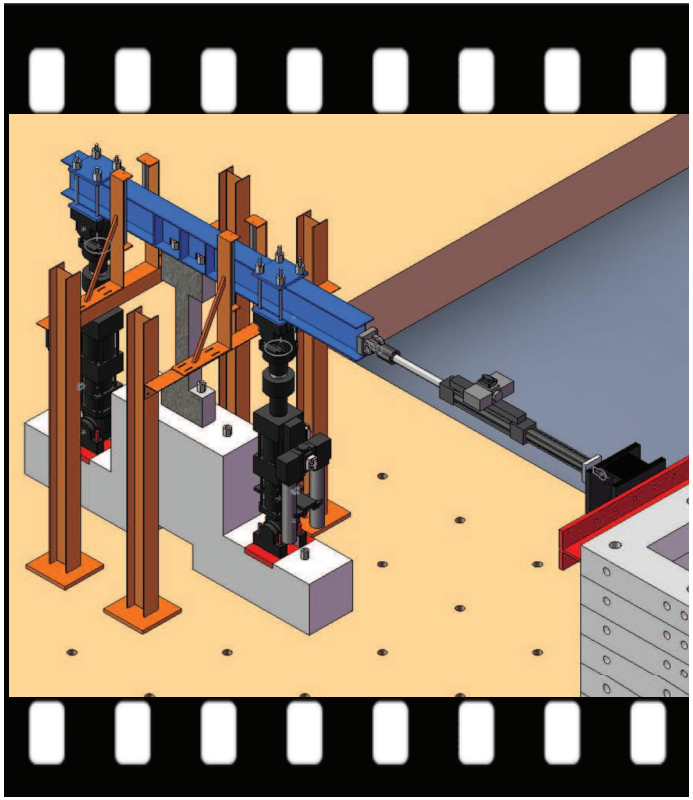
# Applications

Mercury has been “battle tested" through two major endeavors:

**Research** Real time Hybrid Simulation of a reinforced concrete frame with non-ductile columns, and comparison with shake table test results.

**Education** in a new course “Nonlinear Structural Analysis” in which students first used the Matlab version to gently acquaint themselves with basic concepts (including modifying the code), and then with the c++ version for a comprehensive project on seismic rehabilitation.

# Comparison between Shake Table and RTHS; The Video



Click to Start

# Summary

## Mercury

- Is a full nonlinear structural dynamic finite element code which can be embedded inside LabView for pseudo-dynamic tests, or real time hybrid simulation.
- Is more than a software for hybrid dynamic nonlinear analysis of civil engineering structures.
- It is a concept of hardware in the loop which can be adapted to other disciplines such as automotive and aerospace engineering.

# Credit


| | |
|---|---|
| Prof. Victor Saouma | Project Director |
| Dr. Dae-Hang Kang | Development of Mercury Matlab, c++ |
| Dr. Gary Haussmann | Development of Mercury c++ &Hybrid capabilities |
| LMT/Cachan | Anisotropic model |
| | |
| State of Colorado | Financial support |