
TN-2007-16

# Using the OpenSees Interpreter on Parallel Computers

Frank McKenna [1]

Gregory L. Fenves [1]

[1]University of California, Berkeley

Last Modified: 2008-04-17 Version: 1.0

Acknowledgment: This work was supported by the George E. Brown, Jr. Network for Earthquake Engineering Simulation (NEES)

# 1 Introduction

Parallel compuation is becoming increasingly important for conducting realistic earthquake simulations of structural and geotecnical systems [1]. With the advent of multi-core processors it will shortly become the only means available for users to harness the full performance of even their personal computers. The objective of this document is to provide users with a an overall description of the parallel capabilities of the OpenSees interpreters, examples on how to run them on parallel machines, and an outline of how to build the executables of OpenSees for parallel machines. It assumes the user has a working knowledge of both OpenSees and the parallel computer platform on which they intend to run OpenSees.

# 2 The OpenSees Interpreter

OpenSees [2] is an object-oriented software framework for creating nonlinear finite element applications. The framework provides classes for modelling structural and geotechnical systems, classes to perform nonlinear analysis on the model, and classes to monitor the response of the model during the analysis. The framework is primarily written in C++ and provides classes for building finite element applications that can be run on both sequential and parallel computers.

The source code for OpenSees is open-source and is available from the OpenSees website (http://opensees.berkeley.edu) or from the OpenSees CVS repository using anonymous access: (:pserver:anonymous@opensees.berkeley.edu:/usr/local/cvs).

The OpenSees interpreter, an example of one type of application that can be built using the framework, is an extension of the Tcl scripting language [3]. The OpenSees interpreter adds commands to the basic Tcl interpreter for creating OpenSees objects (domain, analysis and recorder) and for invoking methods on those objects. Manuals and examples on using the OpenSees interpreter can be obtained from the OpeSees website user page (http://opensees.berkeley.edu/user/index.php)

# 3 Parallel Computers

A parallel computer is a computing machine which is composed of more than one processor and which allows processes to communicate with each other, either in the form of shared memory or message passing. There are many different forms of machines that can be classified as parallel computers. These include parallel supercomputer with many thousands of processors, local networks of workstations with tens to hundreds of processors, and even single multi-core processor personal computers with a few processors. To build an application that will run on many processors simultaneaously, the software makes calls to communication libraries, e.g. threads for shared memory and MPI for message passing. We limit our discussion here to message based parallel processing.

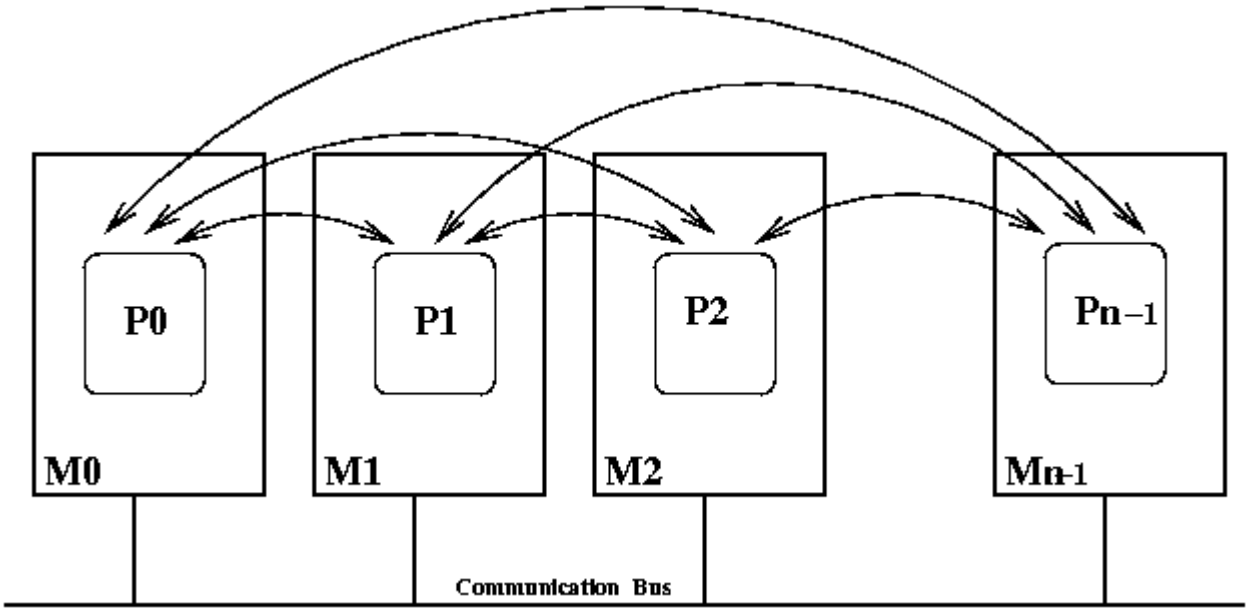


**Figure 1: Parallel Computation**

In a messaage passing system composed of 'n' processes, the processes are typically labelled P0 through PN-1, as shown in Figure 1. Each process can be viewed as running on its own machine, i.e. process P0 runs on machine M0. The processes communicate with each other by calling communication library routines that exist on the machines, these libraries providing a high level interface to the machines underlying communication bus, whose complexity varies greatly depending on the parallel machine. When running on parallel computers it is not the processor speed that generally governs cost of the machine, but the communication mechanism between the processors; the faster the communication the costlier the machine. When using parallel machines the user needs to have some understanding of the performance of each processor, speed and memory size, and the performance of the communication between machines. Sometimes, for example, for small jobs it is quicker to do the computation on a single machine, as communication costs will outweigh the benifit of multiple processors.

The OpenSees framework provides classes for interfacing with the most widely used message passing interface, MPI. For parallel machines with this library, two interpreters can be built from the OpenSees source code distribution:

1. **OpenSeesSP**: The first interpreter is for the analysis of very large models by users who have little understanding of parallel computing and who have an input file that is too large or takes too long to run on a sequential machine (single processor). This interpreter will process the same script that the OpenSees interpreter running on a sequential machine will process. There are no special commands for parallel processing, though there are additional options when it comes to choosing solvers. It will be referred to as the 'Single Parallel Interpreter' application.

2. **OpenSeesMP**: The second interpreter is for running many analyses with small to moderate sized models, i.e. parameter type studies, or for running more complicated scripts in which the

user specifies the subdomains, parallel numberer and parallel equation solver. When running on a parallel machine, each processor is running the same interpreter with the same main input script. The user has control at both the command line level and scripting level to specify the different work that each processor performs. It will be referred to as the 'Multiple Parallel Interpreter's' application.

# 4 OpenSeesSP: The Single OpenSees Parallel Interpreter

The simple parallel interpreter is intended for the single analysis of very large models. When running on a parallel machine, a single processor, p0, is running the main interpreter and processng commands from the main input script. The other processors are running ActorSubdomain objects [4]. As the model is built as model commands, e.g. element, node, loadPattern, are issued the domain is constructed on P0. This is as shown in Figure 2. On the first issuance of the analyze() command in the script the model is partitioned, that is the elements are split and distributed among all n-1 machines. After this the state and solving of the system of equations can be done in parallel, as shown in Figure 3. Whether or not the solution of the linear system of equations is done in parallel, depends on the choice of equation solver. Additional solvers may be present on the paralell machine the job is to be run on and if present, the appropriate solver needs to be set at the system command in the script. Also if recorders are specified in the scripts that use the –file option and are recording the information for more than one element or node, the column order of results stored in files from the Element and Node recorders will NOT be ordered as they are in single processor runs. The user's should use the –xml option when defining recorders to document each column's meta data.

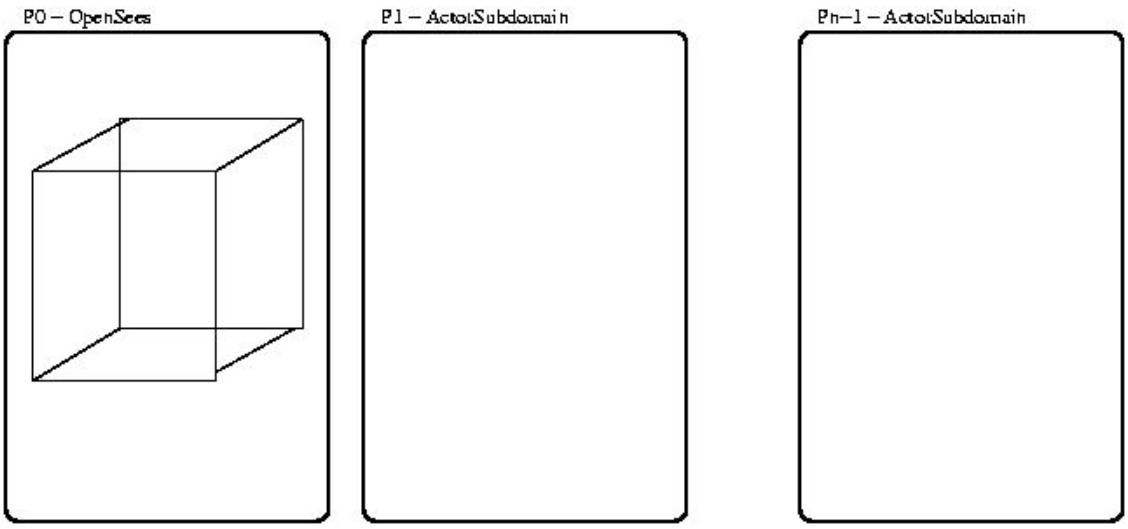


**Figure 2: Single Parallel Interpreter before analyze()**

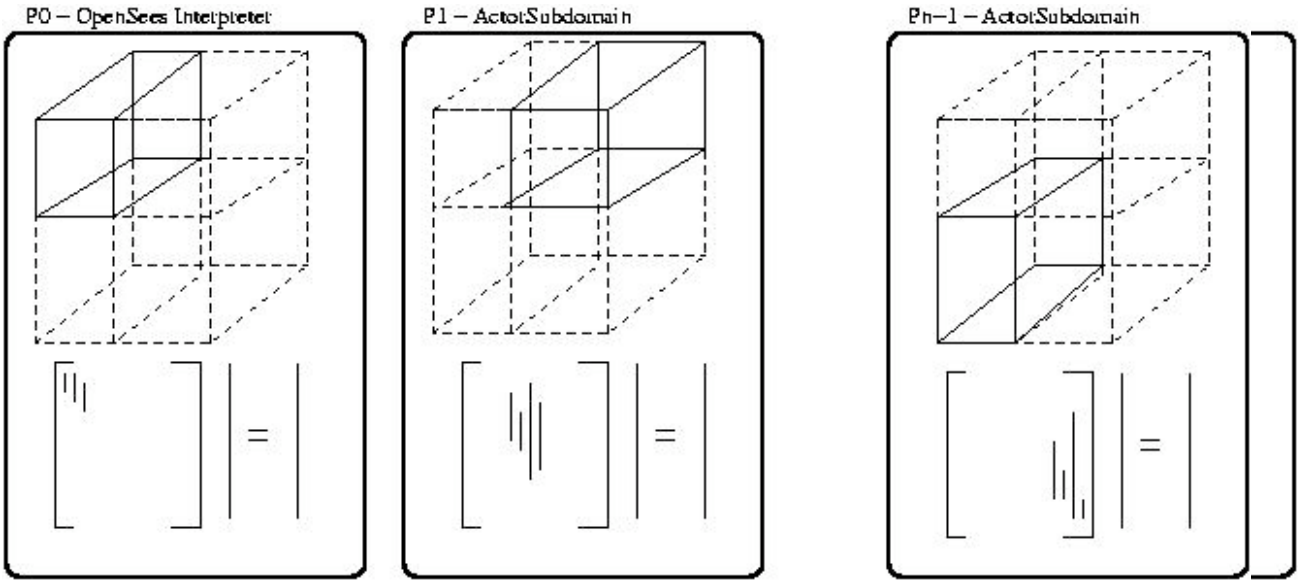


**Figure 3: Single Parallel Interpreter after analyze()**

## 4.1 Additional Solvers for Parallel Processing

A number of additional solvers may be available for parallel processing on the local parallel machine. These include Mumps, Petsc, and SuperLU. Of these, SuperLU will always be available, Petsc will probably be available for your system, and Mumps you will have to install because of copyright restrictions. Of the three, Mumps typically will perform the best and should be the one that is tried first. If you specify SparseGEN in the script, when running on a parallel machine you will automatically be switched to the parallel version of SuperLU. To use the additional solvers, the system command is altered to include the parallel solvers.

```
system Mumps
system Petsc
```

## 4.2 Domain Decomposition Example

In this example we perform domain decomposition analysis. The script is exactly the same as a regular OpenSees script that runs on a single processor machine.

---

```
# source in the model and analysis procedures
source model.tcl

# record the forces for all elements
recorder Element -xml eleForces.out –ele all forces

# perform gravity analysis
```

```
system ProfileSPD
constraints Transformation
numberer RCM
test NormDispIncr 1.0e-12 10 3
algorithm Newton
integrator LoadControl 0.1

analysis Static

set ok [analyze 10]
```

To run this example the user would start the OpenSees interpreter from the command line with a command such as:

```
mpirun -np 4 OpenSeesSingleParallelInterpreter main1.tcl
```

# 5 OpenSeesMP: The Multiple Parallel OpenSees Interpreter Application

When running as a job on a parallel computer with this interpreter, each process is running a slightly modified version of the basic OpenSees interpreter. This interpreter allows each process to determine the number of other processes running as part of the parallel job, numP, and the unique process id number for that process, np. This is as shown in Figure 4. In addition, if the command line option -par has been specified by the user to perform a parametric study, the processes execute using a modulo approach so that the parameter study is performed in parallel with each set of parameters being processed only once.

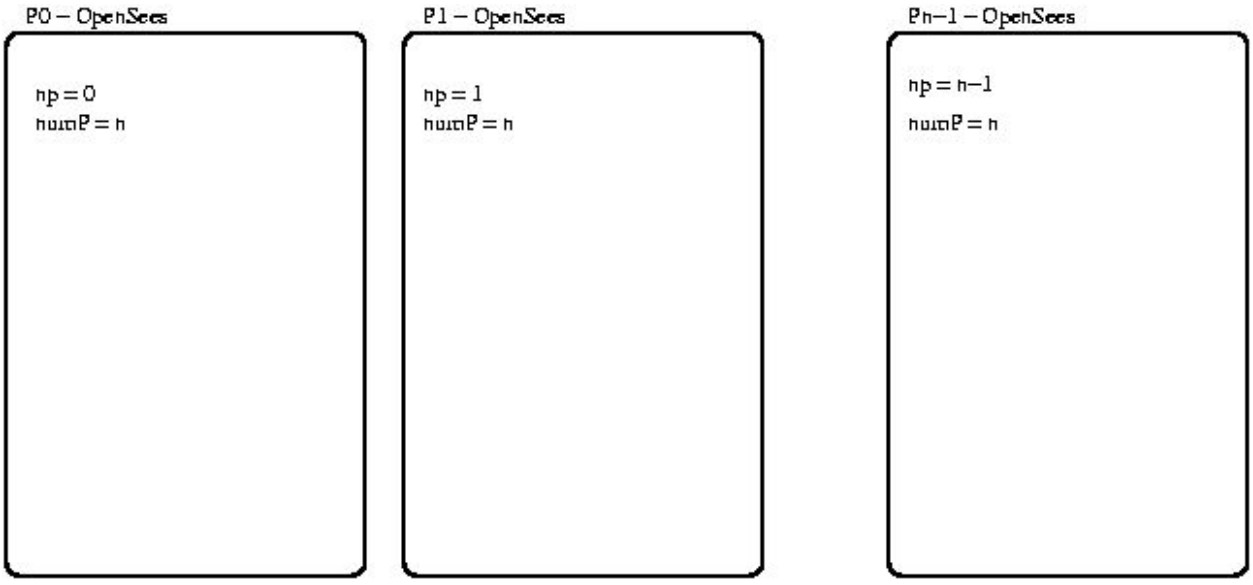


**Figure 4: Multiple Parallel Interpreters**

## 5.1 Additional Commands

This new OpenSees interpreter can handle all existing OpenSees commands. Additional commands and command line arguments, not found in the basic interpreter, are provided.

A new -par option is provided in the command line when starting the interpreter. The syntax is:

**OpenSees mainFile? -par parName1 parFile1 -par parName2 parFile2 ..**

When this option is used, the script will be executed the combination of all the parameters found in the parameter files. The processor that each script runs on with a particular parameter set is determined using a modulo approach, by assigning a unique number to each parameter set, dividing by the total number of processors and using the modulo to assign it to a process.

A number of additional commands are provided to allow each running process to determine the processor it is running on and the number of processors that the user started on the parallel machine and to allow the interprocess communication at the scripting level. These are:

**getNP**: returns the total number of processors assigned to the user for this job.
**getPID**: return a unique processor number in range 0 through value returned in [expr [getNP] - 1]
**send –pid $pid $data**: to send the data from local process to process whose process id is given by the variable pid. Pid must be in range [0 through [expr [getNP] -1]
**recv –pid $pid variableName**: to receive data from a remote process and set the variable named variableName to be equal to that data. Pid must be in set {0,..[expr [getNP] -1, ANY}. If the value of $pid is ANY, the process can receive data from any process.
**barrier**: causes all processes to wait until all process reach this point in the code.

Using these commands it is possible for the user to perform their own domain decomposition analysis (see Figure 3). The getNP and getPID allow the user to specify which nodes and elements are created on which processor. The user however, must specify a parallel solver and numberer if this is their intent using a modified system and numberer command described next.

## 5.2 Modified Commands

system: A number of parallel system-of-equations can be used. Of these only Petsc, Mumps and DistributedSuperLU will actually perform the solving of the equations in parallel. The rest of the solvers will gather the system of equation on P0 and solve it there sequentially. When doing domain-decomposition using this interpreter a paralell system MUST be specified.

**system ParallelProfileSPD**

**system Mumps**
**system Petsc**

numberer: When doing domain-decomposition using this interpreter the user MUST specify a parallel numberer.

**numberer ParallelPlain**
**numberer ParallelRCM**

## 5.3 Parameter Study Examples

In the next two sections we will provide an example of a parameter study of subjecting a model to multiple earthquake exciatations. The study will be done in the first case using the command line –par option, and in the second example using the additional comands found in th interpreter.

To demonstrate the new interpreter consider the example of determing the drift ratios in a building model subject to a list of earthquakes from the PEER ground motion database. The list of records to run are assumed to be found in a file peerRecords.txt, which for this example holds 7200 lines, of which the first 10 are shown here:

CHICHI/TAP036-N.AT2
ABRUZZO/ATI-WE.AT2
ABRUZZO/GCN-NS.AT2
ABRUZZO/GCN-WE.AT2
ABRUZZO/ISE-NS.AT2
ABRUZZO/ISE-WE.AT2
ABRUZZO/PON-NS.AT2
ABRUZZO/PON-WE.AT2
ABRUZZO/ROC-NS.AT2
ABRUZZO/ROC-WE.AT2

The modeling and analysis details are unimportant for this example, so will be left out. Their details are assumed to be contained in the files model.tcl, gravity.tcl, and analysis.tcl. The gravity.tcl and analysis.tcl files contain the procedures doGravity and doDynamic.

### *i. Example using Command Line Arguments*

In this example, the ability of the OpenSees interpreter to automatically perform the parallelization of the parameter study based on the usage of the -par option in the command line will be demonstrated. The program will be run on a user specified number of processors. Each processor will see the same input file, main.tcl, and will be started with the additional command line arguments **-par gMotion**

**records.txt**, where each time a process executes main.ctl, the variable gMotion will be one of the lines of the records.txt file. The main script, main1.tcl, is as follows:

```
# source in the model and analysis procedures
source model.tcl

# do gravity analysis
source analysis.tcl

set ok [doGravity]

source model.tcl
source analysis.tcl

set ok [doGravity]

loadConst -time 0.0;

if {$ok == 0} {
  set gMotionList [split $gMotion "/"]
  set gMotionDir [lindex $gMotionList end-1]
  set gMotionNameInclAT2 [lindex $gMotionList end]
  set gMotionName [string range $gMotionNameInclAT2 0 end-4 ]

  set Gaccel "PeerDatabase $gMotionDir $gMotionName -accel $G -dT dT -nPts nPts"
  pattern UniformExcitation 2 1 -accel $Gaccel

  recorder EnvelopeNode -file $gMotionDir$gMotionName.out -node 3 4 -dof 1 2 3 disp

  doDynamic [expr $dT*$nPts] $dT

  if {$ok == 0} {
   puts "$gMotionDir $gMotionName OK"
  } else {
   puts "$gMotionDir $gMotionName FAILED"
  }
}
wipe
```



To run this example, the user would start the OpenSees interpreter from the command line using the -par option with a command such as:

```
mpirun -np 1024 OpenSeesManyParallelInterpreters main1.tcl -par gMotion records.txt
```

### *ii. Example using Additional Commands*

This example demonstrates the ability of the OpenSees interpreter to perform parallelization based on the processor id on which it runs and the number of processors involved in the parallel simulation. The program will be run on a user specified number of processes. Each processor will see the same input file, main.tcl. Each proceses when it starts will determine its unique id and the number of processes involved in the computation. They will each open the records.txt file and run through the list of records, only proceessing a record if the line number modulo the processor id is equal to 0. This guarantees that each record is handled only once and that each process handles roughly the same number of records. The main script, main2.tcl, for this example is as follows:

---

```
set pid [getPID]
set np [getNP]
set recordsFileID [open "peerRecords.txt" r]
set count 0;

foreach gMotion [split [read $recordsFileID] \n] {
  if {[expr $count % $np] == $pid} {

  source model.tcl
  source analysis.tcl

  set ok [doGravity]

  loadConst -time 0.0

  set gMotionList [split $gMotion "/"]
  set gMotionDir [lindex $gMotionList end-1]
  set gMotionNameInclAT2 [lindex $gMotionList end]
  set gMotionName [string range $gMotionNameInclAT2 0 end-4 ]

  set Gaccel "PeerDatabase $gMotionDir $gMotionName -accel 384.4 -dT dT -nPts nPts"
  pattern UniformExcitation 2 1 -accel $Gaccel
```



```
  recorder EnvelopeNode -file $gMotionDir$gMotionName.out -node 3 4 -dof 1 2 3 disp

  doDynamic [expr $dT*$nPts] $dT

  if {$ok == 0} {
    puts "$gMotionDir $gMotionName OK"
  } else {
    puts "$gMotionDir $gMotionName FAILED"
  }
  wipe
  }

  incr count 1;
}
```

To run this example the user would start the OpenSees interpreter running on the parallel machine with a command such as:

```
mpirun -np 1024 OpenSeesManyParallelInterpreters main2.tcl
```

### *iii. Differences between the Examples and Performance Issues*

Internally there is very little difference between the two examples. The C++ code that is built into the interpreter is handling the foreach statement that must be programmed explicitly in the second example once the process id and number of processors are known. The scripting code for the first is simpler, but while the second example is harder to program, it allows the user more control.

For parallel computers where the file system can be a bottleneck when it comes to performance, this control is crucial to achieving optimum performance. If many small jobs all access the file system at once, the performance of the running job can be severely affected by the file system. There are a number of things the user can do in the script to reduce the demand on the file system. For example, files that are going to be read multiple times by the same process should be copied to the local disk of a processor (/scratch typically on Unix machines) before opening and reading the files. Another example would be to use 'reset' instead of ‘wipe’; source model.tcl', if the same example is going to be run over and over again in a parameter study. We can rewrite the second example to use less file resources. The new example, main3.tcl, makes use of a fast /scratch directory that each local process has access to and uses the reset and remove commands so that the model only has to be created once.

```
set pid [getPID]
set np [getNP]
```

```
source model.tcl
source analysis.tcl

file copy peerRecords.txt /scratch/peerRecords.txt.$pid

set recordsFileID [open "/scratch/peerRecords.txt.$pid" r]

set count 0;

foreach gMotion [split [read $recordsFileID] \n] {
 if {[expr $count % $np] == $pid} {

  set ok [doGravity]

  loadConst -time 0.0

  set gMotionList [split $gMotion "/"]
  set gMotionDir [lindex $gMotionList end-1]
  set gMotionNameInclAT2 [lindex $gMotionList end]
  set gMotionName [string range $gMotionNameInclAT2 0 end-4 ]

  set Gaccel "PeerDatabase $gMotionDir $gMotionName -accel 384.4 -dT dT -nPts nPts"
  pattern UniformExcitation 2 1 -accel $Gaccel

  recorder EnvelopeNode -file $gMotionDir$gMotionName.out -node 3 4 -dof 1 2 3 disp

  doDynamic [expr $dT*$nPts] $dT

  if {$ok == 0} {
   puts "$gMotionDir $gMotionName OK"
  } else {
   puts "$gMotionDir $gMotionName FAILED"
  }

  # revert to start by removing load patterns, recorders & resetting
  remove loadPattern 1
  remove loadPattern 2
  remove recorders
  reset
 }
 incr count 1;
}
```



## 5.4 Domain Decomposition Example

In this example, we perform domain decomposition analysis for a large model. The nodes, elements and loads created on each processor depend on the process id, pid. For the analysis, a parallel solver and dof-numberer are used.

```
# source in the model and analysis procedures
set pid [getPID]
set np [getNP]

# source in model and build model based on np and pid
source modelP.tcl
doModel {$pid $np}

# perform gravity analysis
system ParallelBandGeneral
constraints Transformation
numberer ParallelRCM
test NormDispIncr 1.0e-12 10 3
algorithm Newton
integrator LoadControl 0.1

analysis Static

set ok [analyze 10]
return $ok
```

# 6 Running the Interpreters on Windows Machines

Prebuilt binaries of the two interpreters can be found on the OpenSees website http://OpenSees.berkeley.edu/OpenSees/parallel/parallel.php . These binaries require that the MPICH2 windows packages is installed on your machine. The MPICH2 distribution includes mpiexec, a binary to run the applications, and smpd, a service that runs under windows to create processes for mpiexec. When instaling under Windows Vista User Acess Control (UAC) needs to be turned off, this will require a restart. This is required so that smpd service is set to run at startup in the install. You can turn it on again after installing the package. If you do not do this, before running mpiexec yoi will need to open a



terminal and run smpd using ‚smpd –d 0‘ option. The first time you run mpiexec in a terminal you will be prompted for the passphrase. You should set the Path enviroment variable to pint to the MPICH2 install directory, which by default is c:\Program Files\MPICH2\bin.

To run an application with n processes on a tcl script you simply type

**mpiexec –np n applicationName scriptName**

For running the code on a network of parallel windows machines MPICH2 needs to be installed on all machines. In addition the user runing the application must have an account on all machines and must be registered to run on all machines. Also the OpenSees applications needs to be installed on all machines and in the same directory locations on all machines. If running OpenSeesMP the input scripts must be on all machines.

# 7 Running the Interpreters on Intel Mac Machines

Prebuilt binaries of the two interpreters can be found on the OpenSees website http://OpenSees.berkeley.edu/OpenSees/parallel/parallel.php . These binaries require that the OpenMPI Macintosh distribution is installed on your machine. The OpenMPI distribution includes mpiexec, a binary to run the applications. The default installtion locataion of OpenMPI is /usr/local. You should include the directory /usr/local/bin in your PATH envionment variable.

To run an application with n processes on a tcl script you simply type

**mpiexec –np n applicationName scritName**

# 8 Obtaining and Building the Interpreters

There are a number of steps to be performed to obtain and build this parallel version of the OpenSees interpreter. These steps will be outlined assuming the user is building the interpreter on a Unix-based machine.

1) Obtain the Source Code for Tcl/Tk

Tcl/Tk may already be installed on the parallel computer you wish to run on. On unix machines issue the command which tclsh'. If tclsh is there, run it and check which version is there using the tcl_version variable, e.g. start the interpreter and type the following 'set a $tcl_version'.

If you do not have Tcl/Tk you will need to obtain the source code and build it, as when building OpenSees you will need to link to the tcl libraries. Ths source code and installation instructions can be found at http://www.tcl.tk/software/tcltk/

2) Obtain the Source Code for OpenSees



The latest and most up-to-date version of the source code can be obtained from the the OpenSees CVS repository using the following two commands:

cvs -d :pserver:anonymous@opensees.berkeley.edu:/usr/local/cvs
at prompt type: anonymous

cvs -d :pserver:anonymous@opensees.berkeley.edu:/usr/local/cvs co OpenSees

The latest stable version of the code can be obtained from the OpenSees web-site:

http://opensees.berkeley.edu/OpenSees/developer/download.php

3) Create the Makefile.def

The next and most complicated part of the process involves creating a Makefile.def file. Example Makefile.def files can be found in the OpenSees/MAKES directory. Currently there are examples in this directory for parallel computers: one is for the Datastar machine at SDSC (Makefile.def.DATASTAR), another is for a linux IA-64 cluster (Makefile.def.TERAGRID) at SDSC, and a third for a linux cluster at Makefile.def.LINUX_CLUSTER for a cluster at UC Davis. Copy the file closest to your parallel machine set-up to OpenSees/Makefile.def. Now you need to open this file and edit certain lines. These will be edited assuming that OpenSees is to be found at /home/fmckenna/OpenSees and tcl/tk in /home/fmckenna/tcl, and that the compiler to use is gcc. Additional flags may be required to use mpi depending on your system.

```
HOME = /home/fmckenna
TCL_LIBRARY = /home/fmckenna/tcl/lib/libtcl.X
TCL_INCLUDES = /home/fmckenna/tcl/include
CC++  = gcc
CC    = gcc
FC    = g77
C++FLAGS = -02
CFLAGS  = -02
FFLAGS  = -02
```

One other variable that needs to be set is PROGRAMMING_MODE. The value depends on which interpreter you plan on building.

4) For OpenSeesSP.



PROGRAMMING_MODE = PARALLEL

5) For OpenSeesMP.

PROGRAMMING_MODE = PARALLEL_INTERPRETERS

6) Create Directories for the OpenSees .exe and libraries

In the HOME directory given above create directories lib and bin.

7) Build the .exe

In the HOME/OpenSees directory type 'make'.

Datastar users should note that this Makefile.def has been set up for 64 bit compilation. As such, as so as not to be stopped by a bunch of errors, you need to set the OBJECT_MODE enviroment variable to 64, e.g. type

setenv OBJECT_MODE 64.

# 9 Acknowledgments

The development of OpenSees has been supported by the Pacific Earthquake Engineering Research Center under grant no. EEC-9701568 from the National Science Foundation. Integration of OpenSees with the NEES information technology program (NEESit) has been supported by NSF under grant no. CMS-0402490 from the NEES Consortium to the University of California, San Diego Supercomputer Center. The support of Dr. Ahmed Elgamal and Dr. Lelli Van Den Einde from NEESit is greatly appreciated. Significant contributors to OpenSees include, among many others, Dr. Michael H. Scott, Dr. Filip C. Filippou, and Dr. Boris Jeremic.

# 10 References

[1] Fenves, G.L., "A Vision for Computational Simulation in Earthquake Engineering", Community Workshop on Computational Simulation and Visualization Environment for the Network for Earthquake Engineering Simulation (NEES), Sponsored by NSF, Roddis, K., editor, Lawrence, KS, 2003.
[2] Open System for Earthquake Engineering Simulation, http://opensees.berkeley.edu/
[3] Ousterhout. J., Tcl and the Tk Toolkit, Addison-Wesley, 1994.
[4] McKenna, F., Object-oriented Finite Element Programming: Frameworks for Analysis, Algorithms and Parallel computing, Ph.D. Thesis, University of California, Berkeley, CA, 1997.