Users' Guide for 3D Finite Element Software
Citcom with Faults on Parallel Computers

Shijie Zhong

Seismological Laboratory
California Institute of Technology
June, 1997

E-mail: zhong@gps.caltech.edu

© COPYRIGHT 1997 by California Institute of Technology
All Rights Reserved

Distributed Freely For Non-commercial Purposes
Abstract. The 3D finite element software Citcom [Moresi and Solomatov, 1995; Moresi and Gurnis, 1996] has been modified to incorporate faults with 3D geometry. Citcom has been successfully ported to parallel computers with a message passing software MPI (e.g., Message Passing Interface). The code will work on both shared and distributed memory parallel computers. In this users' guide, the finite element techniques used to incorporate faults and the implementation of parallelizing Citcom with MPI are discussed. An example will be presented on how to use the Citcom software with faults on parallel computers.

1. Introduction

In models of the lithosphere, faults are important features of the dynamics. Since faults have complicated geometry and rheology, finite element methods are often used in constructing models [Melosh and Williams, 1989; Zhong and Gurnis, 1994]. In order to resolve the stress field near faults, high resolution is required. Consequently, fault modeling is computationally demanding. A fault algorithm has been incorporated into the 3D finite element software Citcom (a program written in C which stands for California Institute of Technology Convection in the Mantle). The software has been successfully ported to parallel computers making it feasible to model faults with high resolution [Zhong, Gurnis, and Moresi, 1997].

This manual consists of four sections: 1) the finite element techniques used to model faults; 2) the implementation of parallel computing used within Citcom; 3) how to use Citcom on parallel computers, including details of input files; 4) the limitations and precautions when using the current version of this software.

2. Modeling Faults in 3D Finite Element Models of Mantle Flows

The problem that we seek to solve in a fault model can be stated as: for a given driving force either in the form of boundary conditions or body forces, we seek the
resultant deformation and stress field throughout the medium with the presence of faults. The governing equations for the deformation and stress fields are derived from the conservation of mass and momentum. Assuming incompressibility, the momentum and continuity equations are, respectively:

\[ \sigma_{ij,j} + \rho g \delta_{iz} = 0, \]
\[ u_{i,i} = 0, \]  

(2)

where \( \sigma_{ij} \), \( u_i \), \( \rho \), and \( g \) are the stress tensor, the flow velocity, the density, and gravitational acceleration, respectively.

The finite element treatment of (1) and (2) can be found in Hughes [1987]. Constrained elements and matrix transformation are used to incorporate faults into 3D finite element models. The techniques in Citcom are similar to those used by Melosh and Williams [1989]; the current method is an extension to our previous penalty function method [Zhong and Gurnis, 1994]. The general treatment for such problems can be found in Cook [1981] and will be briefly described as following.

We assume that the fault plane consists of elemental boundaries (Fig. 1a); those elements with boundaries overlapping the fault are called constrained elements. There are two constraints on the fault: (1) velocities are continuous in the normal direction \( \hat{n} = (n_1,n_2,n_3) \), and (2) tangential velocities are coupled with a specified frictional stress (assumed to be zero for the sake of clarity in discussion). The vector \( \hat{n} \) is defined in a global coordinate system \( x, y, \) and \( z \) (Fig. 1). Define two tangential directional vectors \( \hat{t} = (t_1,t_2,t_3) \) and \( \hat{s} = (s_1,s_2,s_3) \) such that \( \hat{n} \), \( \hat{t} \), and \( \hat{s} \) form an orthogonal coordinate system (i.e., local coordinate system of the fault) (Fig. 1). We may define such a local coordinate system for each faulted node within constrained elements.

For each constrained element, elemental matrix equations in the global coordinate system are [Hughes, 1987]
where $K$ is a stiffness matrix with 24x24 entries (this is, for a 3D element with 8 nodes and trilinear shape functions); $F$ and $G^T$ are force and divergence vectors; $P$ is the pressure which is constant within the element; $U = (\cdot, \cdot, u_x^i, u_y^i, u_z^i, \cdot, \cdot) \cdot$ and $i$ is the local node index.

If the local node $i$ falls on the fault plane, we will use velocities in the local coordinate system for this node. Velocities in the local coordinate system for this faulted node can be found through the following transformation:

\[
KU + G^T P = F, \quad (3)
\]

\[
GU = 0, \quad (4)
\]
where $U' = (\ldots, u_n^l, u_t^l, u_s^l, \ldots)^T$ is the new velocity vector; $\Gamma_i$ is a transformational matrix consisting of 24x24 entries with only nonzero entries in eight 3x3 submatrices on the diagonal of $\Gamma_i$. Seven of the eight 3x3 submatrices are unit matrices, and the only nonunit submatrix, $T_i$, is associated with node $i$. $T_i$ can be written as

$$T_i = \begin{bmatrix} n_1 & n_2 & n_3 \\ t_1 & t_2 & t_3 \\ s_1 & s_2 & s_3 \end{bmatrix}. \quad (6)$$

Applying the transformation to equations (3) and (4) leads to

$$\Gamma_i K_i T_i U' + \Gamma_i G T_i P = \Gamma_i F, \quad (7)$$

$$G T_i U' = 0. \quad (8)$$

If we define $K' = \Gamma_i K_i T_i^T$, which remains symmetric and positive definite, $G' = G T_i^T$, and $F' = \Gamma_i F$, then the forms of (7) and (8) are identical to (3) and (4). If there are other faulted nodes within this element, we consecutively apply the above transformation to these nodes such that velocities associated with each faulted node in the $U'$ vector are based on the local coordinate system for that node.

Fault constraints are enforced when elemental matrix equations are assembled into global matrix equations. For each faulted node in the global matrix equations, we assign five degrees of freedom: one normal velocity, $u_n^l$, and four tangential velocities, $u_t^l$, $u_s^l$, $r u_t^l$, and $r u_s^l$, such that the first two are for the left side of the fault and the last two are for the right (Fig. 1b). $\mu l^l$ and $\mu s^l$, and $r \mu l^l$ and $r \mu s^l$ are used for the constrained elements on the left and right sides of the fault, respectively. After the global equations are solved, we obtain the velocity field in which the velocities on faulted nodes are based on the local coordinate system for each faulted node. We can apply the reverse transformation of (5):
\[ U = \Gamma_i^T U', \text{ for each faulted node to retrieve the velocities in the global coordinate system} \]
\[ (\Gamma_i \text{ is an orthogonal matrix and } \Gamma_i^T \Gamma_i = I). \]

3. Parallelization of 3D Finite Element Code Citcom With MPI

The message passing software MPI [Snir et. al., 1996] is used to parallelize the finite element code Citcom. Since MPI is widely supported on parallel computers including shared and distributed memory machines, the parallel version of Citcom has been found to be quite portable. Citcom uses a predictor-corrector SUPG algorithm for solving the time-dependent energy equation and a two-level Uzawa algorithm for the momentum equation. The original version of Citcom supports two different momentum equation solvers: multigrid and conjugate-gradient. Since the current multigrid solver only works for regular grids and cannot deal with the complicated geometries associated with faults, only the conjugate-gradient solver can be used. Therefore, only the conjugate-gradient solver is parallelized.

The SUPG and two-level Uzawa algorithms implemented in Citcom display a significant locality and are suitable for parallelization. Computations of the stiffness matrix, force vectors, and the divergence, gradient, and Laplacian operators can all be performed at an element level and do not involve interprocessor communication. Communications are needed in assembling force vectors and residue terms and in computing the searching step in the conjugate gradient method. Communications only involve information on the boundaries between each of the computational domains. This will be illustrated with an example of assembling force vectors. The following loop is over the total number of elements for the computational domain for a single processor; within the loop, computed elemental force vectors elt_f are assembled into the global force vector F.

```plaintext
for el = 1, nel        # nel is the number of elements for this computational domain
...
for n = 1, nnpe       # nnpe is the number of nodes per element
  for i = 1, dims     # dims is the number of dimensions
```
\[ F(id(ien(el,n),i)) = F(id(ien(el,n),i)) + elt_f(n,i) \]

# \( F \) and \( elt_f \) are global and elemental force vectors, \( id \) and \( ien \) are id and node arrays.

However, because the contributions from elements on other side of the boundary nodes have not been included (since those elements are in another processor), the assembly for \( F \) is incomplete for the boundaries nodes (Fig. 2). Therefore, in order to complete the assembly, the following operations with interprocessor communications are needed.

Fig. 2. (a) Domain decomposition and interprocessor communication. \( p(i,j) \) indicates processor index. (b) Incomplete assembly for nodes on boundaries of computational subdomain (open circle) prior to interprocessor communication.
for n = 1, node_bd  # node_bd is the number of boundary nodes for one side
    for i = 1, dims
        Send_F(n,i) = F(id(node_b(n),i))  # node_b is the boundary nodes array

send_recv (Send_F, Recv_F, adjacent_proc)
    # send Send_F to and receive Recv_F from the adjacent processor for this side

for n = 1, node_bd
    for i = 1, dims
        F(id(node_b(n),i)) = F(id(node_b(n),i)) + Recv_F(n,i)
        # add Recv_F to F for boundary nodes.

The above operations are only for one side of the computational domain, and similar operations are needed for other sides of the domain.

Fig. 3. Efficiency and CPU time versus the number of processors for a benchmark case on the Intel Paragon at Caltech. Benchmark is done for a mesh 160x64x80. The thick lines are the efficiency; thin solid lines are for CPU times; thin dashed lines are the CPU times for ideal speedup. In computing the efficiency, the efficiency for the smallest number of processors (limited by the memory size of each processor) is 1.0.
The parallelized Citcom uses a single program-multiple data programming model (SPMD) and MPI for interprocessor communications. The design of an efficient communication strategy is crucial in any parallel computing application. In the approach used here, each processor communicates with its neighbors in one direction (e.g., x direction) first, and after this type of communication is finished for all the processors, each processor exchanges information in the other directions (Fig. 2). Each processor needs at most 4 pairs of send and receive operations for 2D problems (6 pairs for 3D problems). Although not obvious, the correct order of interprocessor communications is important. This interprocessor communication scheme is both efficient and stable. For 3D problems, the parallelized Citcom achieves better than 91% efficiency on the Intel Paragon supercomputer at Caltech (Fig. 3). This efficiency is reached if there are more than 3200 trilinear finite elements for each processor; for 3D Cartesian problems, 7000 elements can be fit into each processor of the Intel Paragon. The benchmarks are performed with the numbers of iterations fixed for the conjugate gradient solver and the timings are only for the set of procedures which are most intensive in terms of computation and communications.

4. Structure and Usage of Parallelized Citcom With Faults

4.1. Structure

The original version of Citcom was written in modules. These modules form the basis upon which other functions, such as faults and parallel computing, are built. This users' guide does not intend to explain the structure of Citcom in detail, which can be found in The Citcom Users' Manual, available from L. Moresi. This guide will only show the gross structure with an emphasis on those routines associated with faults and parallelization. A flow chart of the overall code structure is shown in Fig. 4.

As described in Fig. 4, the code can be divided into four parts: the initial stage, the momentum equation solver, energy equation solver, and the output. The initial stage reads in all the necessary information and establishes the foundations for finite element analysis.
and parallel computing. For the initial stage, most modules are included in Instructions.c, Parallel_related.c, Construct_arrays.c, and Faults_mesh.c. The modules for the momentum equation solver are included in Stokes_flow_solver.c and Element_calculations.c. The modules for the energy equation solver are in Advection_diffusion.c. Output.c is for the output.

Fig. 4. Simplified flow chart for the parallelized Citcom with faults.
The two additions to the original version of Citcom are faults and parallel computing. The modules for faults are included in Faults_mesh.c and Faults_related.c. In Faults_mesh.c, a global mesh is generated based on input fault geometry; faulted nodes and elements are marked and grouped; normal vector and transformational matrix for each faulted nodes are computed. Faults_related.c includes modules to perform the matrix transformation for elemental stiffness matrix, divergence and force vectors, and velocity vectors for each faulted nodes. Since fault planes are required on elemental boundaries, as discussed in the previous sections, it is difficult to write a mesh routine that can universally be applied to arbitrary fault planes. Consequently, the routines in generating the grid with faulted nodes and elements, which are in Faults_mesh.c, will depend on input fault geometry. In the following example, Faults_mesh.c is only for faults with one geometry. If user would like to use a different fault geometry, they will have to modify some of the routines contained in Faults_mesh.c. Generally speaking, Faults_related.c is less dependent on fault geometry, as long as the faulted nodes are adequately marked and transformational matrices are correctly computed in Faults_mesh.c.

The modules for parallel computing are in Parallel_related.c and Global_operations.c. In Parallel_related.c, MPI functions like initialization and termination are included; the global grid is decomposed into subcomputational domains, one domain for each processor (CPU). The communication routes for each processor are also established. All the different kinds of routines for exchanging information between adjacent processors are included in Parallel_related.c. Global_operations.c mainly includes modules for computing global quantities such as norms of global residues.

4.2. Usage

This section shows how to compile and execute Citcom on a particular parallel computer (a SUN Enterprise 4000 which is a shared memory machine with 8 processors) and how to prepare input files and process output files. Compiling and executing the code
on different parallel computers should be very similar (makefiles for the Convex-HP Exemplar shared memory and Intel Paragon distributed memory parallel computers at Caltech are also provided).

4.2.1. Compiling and Executing Citcom on Parallel Computers

To compile the executable `citcom.mpi`, type

```
$ make citcom.mpi
```

in the directory where the makefile and source codes are located, where $ is the unix prompt. The makefile is similar to that for a uniprocessor machine except that information about the location of the MPI library must be provided.

To run the executable `citcom.mpi`, type

```
$ mpirun -np N citcom.mpi input
```

in the directory where `citcom.mpi` and file `input` are located. "mpirun -np N" notifies the computer that for this MPI job, the number of processors needed is N. Since in the input file `input`, the number of processors for each of x, y, and z dimensions are specified, the total number of processors requested in `input` must equal N.

4.2.2. Input Files

Input file `input` is required to run `citcom.mpi`. File `input` provides all the important information about a model run and a sample file is included in Appendix A. File `input` is divided into 8 sections: Output, Grid, Faults, Viscosity, Boundary conditions, Miscellaneous parameters, Dimensional parameters, and Solver, and each part will be explained in detail as following (a string of characters following # for a line serves as comments). This is not a comprehensive discussion of all the parameters within Citcom; again the user is referred to *The Citcom User's Manual*.

**Output:** The parameters listed under this section are

```
# Output:
  datafile="case1"
```
logfile="case1.log"
maxstep=1 # total timesteps
storage_spacing=1 # write data every ...
verbose=off

The first line gives the prefix for the output file names. The second line gives the file name for output of miscellaneous information about the model run. The third line specifies the total number of time steps for a time dependent model run. The fourth line gives the interval in time steps between each output. The fifth line specifies whether or not to print full information during the model run.

Grid: The parameters listed under this section are

# Grid:
Geometry=cart3d # cart2d,cart3d or axi, at the moment
nodex=97 nodez=33 nodey=49
dimenx=4.0 dimenz=1.0 dimeny=3.0
nprocx=3 nprocz=1 nprocy=2
nfz=6
nz_410=17
nz_lmantle=23
z_lith=0.0666666667
z_410=0.2733333333
z_lmantle=0.4466666667

The first line gives the geometry for this model (only 3D Cartesian geometry can be used for this version). The second and third lines specify the number of grid points and the nondimensional box size in x, z, and y dimensions. In the fourth line, nprocx, nprocz, and nprocy specify the number of processors in x, z, and y dimensions. The multiplication of nprocx, nprocz, and nprocy is the total number of processors for the model run and must be equal to the number of processors specified in the mpirun command. The next six lines are used for building a grid in the vertical (i.e., z) dimension and here I give a highly customized example. For this particular run, these six lines specify the nondimensional vertical coordinates for the base of lithosphere (100 km) and 410 km and 670 km boundaries and the vertical nodal indices at these locations (note: the total number of nodes in z dimension is 33).
**Fault:** The parameters listed under this section are

```
# Faults:
  faults=1 faults_comparison=0
  fault_coord_file=fault_coord_file3d.dat
```

The first line indicates whether faults are included. Parameter `faults` is 1 (0) if faults are included (excluded). The second parameter in this line `faults_comparison` is ignored if `faults=1`; if `faults=0`, parameter `faults_comparison` can be either 1 or 0. If `faults_comparison=1`, the grid will be identical to that with faults, and such a model may be used to compare with a case with faults. If `faults_comparison=0`, a uniform grid will be used. The second line gives the file that contains the fault geometry (this file will be discussed later).

**Viscosity:** The parameters listed under this section are

```
# Viscosity:
  TDEPV=on        rheol=1
  num_mat=6
  viscE=0.0,0.0,0.0,0.0,0.0,0.0
  viscT=0.0,0.0,0.0,0.0,0.0,0.0
  visc0=2.0e7,2.0e1,2.0e1,1.0e0,2.0e03,2.0e7
  VMIN=on visc_min=1.0e-3
  VMAX=on visc_max=1.0e03
  VISC_UPDATE=off
  SDEPV=off       sdepv_misfit=0.010
  sdepv_expt=3,3,3,3,3,3
```

In the first line, the parameter `TDEPV=on` is for temperature-dependent viscosity, and `rheol=1` specifies temperature dependence of viscosity which is given by

\[
\eta = A \exp\left[\frac{C_1}{\theta + C_2} - \frac{C_1}{1 + C_2}\right],
\]

where \(\theta\) is nondimensional temperature, \(C_1\) and \(C_2\) are two constants, and \(A\) is the preexponent. Users can define different temperature-dependence of viscosity in `Viscosity_structure.c`. The second line gives the number of material groups (i.e.,
lithosphere and mantle can have different material properties and material groups can be defined in `Construct_arrays.c`). The third, fourth, and fifth lines give \( C_1, C_2, \) and \( A \) in (9) for six material groups. The sixth and seventh lines specify the cutoff viscosities (i.e., minimum and maximum viscosities). The eighth line indicates whether or not the viscosity will be updated. The ninth line specifies whether stress-dependent viscosity (SDEPV) is used and what the tolerance of non-Newtonian iteration (\texttt{sdepv\_misfit}) is. The tenth line gives the exponent for a power law rheology for each material group.

**Boundary Conditions:** The parameters listed under this section are

```plaintext
# BOUNDARY CONDITIONS
  topvbc=0         # velocity boundary conditions top and bottom
  topvbxval=0.0  topvbyval=0.0
  botvbc=0
  botvbxval=0.0  botvbyval=0.0
  toptbc=1        # temperature bc's top and bottom
  toptbcval=0.0
  bottbc=1
  bottbcval=1.0
```

The current version of the code only supports isothermal and free slip boundary conditions for the top and bottom boundaries and reflecting boundary conditions for the sidewalls. The first two lines are for velocity boundary conditions on the top boundary. In the first line, parameter \texttt{topvbc=0} is for free slip top boundary; \texttt{topvbc=1} is for prescribed velocity top boundary (not yet implemented in parallel). If the top boundary has a prescribed velocity condition, the second line gives the velocities in x and y directions. The third and forth lines are for velocity boundary conditions on the bottom boundary. The fifth and sixth lines are for temperature boundary condition on the top boundary. In the fifth line, parameter \texttt{toptbc=1} indicates an isothermal boundary condition on the top, and in the sixth line, parameter \texttt{toptbcval} specifies the nondimensional temperature on the top, which is 0.0 here. Parameter \texttt{toptbc=0} would indicate a specified heat flow boundary condition with the heat flow given by parameter \texttt{toptbcval}. However, heat flow boundary
conditions are not tested for the current version of code. The seventh and eighth lines are for temperature boundary conditions on the bottom boundary.

*Miscellaneous Parameters:* The parameters listed under this section are

```plaintext
# Miscellaneous parameters
rayleigh=1.782e5        # Rayleigh number
Ra_670=0.0              #0.0% density jump
clapeyron670=-0.11313131        #-3.5MPa/K
transT670=1.0           # 1600 K
width670=0.025          # 37.5 km
Q0=0.0                  # Dimensionless internal heating rate
frictional=0
fric_force=0.0  # for 5 MPa
plate_velocity=0.03     # unit : m/year
```

The first line gives the thermal Rayleigh number. The second, third, forth and fifth lines specify the Rayleigh number, Clapeyron slope, transition temperature, and loop width for the 670-km phase change. The sixth line gives internal heating rate. In the seventh line, the first parameter frictional indicates whether or not a nonzero frictional force is applied on faults, and the second parameter fric_force gives the magnitude of fictional force. If frictional=0, zero frictional force is on faults. All the parameters from lines 1 to 7 are nondimensional numbers. The last line is the surface velocity in m/yr that is used for deriving a half-space cooling thermal structure for this model run.

*Dimensional Parameters:* The parameters listed under this section are

```plaintext
# Dimensional parameters
layerd=1500.0 # km
ReferenceT=1600.0
density=3300.0
thermdiff=1.0e-6
gravacc=10.0
thermexp=2.e-5
cp=1250
wdensity=0.0
```

The first line gives the thickness of box in kilometers. The reference temperature, density, thermal diffusivity, gravitational acceleration, thermal expansivity, specific heat
capacity, and density of overlying medium are given in second, third, forth, fifth, sixth, seventh, and eighth lines, respectively.

**Solver:** The parameters listed under this section are

```
# Solver
stokes_flow_only=1
Solver=cgrad node_assemble=1    # conjugate gradient
precond=on
aug_lagr=on
aug_number=5.0e3
vlowstep=1000           # Enough to get very accurate soln.
piterations=375         # Uzawa iteration loops.
accuracy=1.0e-3         # Desired accuracy of Uzawa algorithm.
tole_compressibility=1e+4
adv_sub_iterations=2
finetunedt=0.6
```

The first line indicates whether or not the problem is a Stokes' flow problem. If parameter `stokes_flow_only=1`, the energy equation will not be solved. The second line indicate the solver type. The first parameter `Solver=cgrad` indicates that a conjugate gradient solver is used and the second parameter `node_assemble=1` indicates that a node assembler is used in computing $\nabla^2 u$. If `node_assemble=0`, a slower assembler (element by element) will be used. Multi-grid solver cannot be used yet in the current version. The third line specifies whether or not a preconditioner will be applied to the conjugate gradient method. In the fourth line, the first parameter `aug_lagr` indicates whether or not an augmented Lagrange multiplier method will be applied, and the other parameter `aug_number` specifies the multiplier. For many problems, the augmented Lagrange multiplier method speeds up the convergence of the pressure field [Moresi et al., 1996]. In the fifth line, parameters `vlowstep` and `piterations` give the maximum numbers of iteration for inner (for velocity) and outer (for pressure) iterations of the Uzawa algorithm. The parameters `accuracy` and `tole_compressibility` in the sixth line specify the tolerances for the inner and outer iterations, respectively. It is recommended that parameter `tole_compressibility` be one order magnitude smaller than parameter `accuracy`
for accurate pressure field. The last line is the reduction factor for the Courant time which is used for solving the energy equation. This concludes the description about the input file input.

4.2.3. The Fault Geometry File

If a model includes faults, another input file is needed to specify the fault geometry and grid information associated with faults. The file name is given in the first input file input. A sample file for the second input file fault_coord_file3d.dat is given in Appendix B. It should be pointed out that this sample file may not represent the most appropriate way to specify the information. If users would like to model other fault geometries, they should fully familiarize themselves with Faults_mesh.c and try to modify the relevant modules to accommodate their needs. The sample file is described in detail as follows.

The first line gives the number of fault segments. For each fault segment, there are six lines specifying the grid and geometry information for this segment and subducted slab connecting to this fault segment. For example, for the fourth segment, these six lines are:

4  0 56 56 13 15
1  2.0  0.0  0.75
2  2.115 0.06666667  0.75
3  2.115 0.06666667  0.8
4  2.0  0.0  0.8
60  0.077  0.52

In the first of these six lines, the first parameter indicates the segment number; the second parameter specifies whether the normal direction for this fault segment should be reverted by 180° (0:No and 1:Yes); the third and forth parameters specify the starting (nfx1) and ending (nfx2) nodal indices in the x dimension for this fault segment; and the last two parameters are the starting (nfy1) and ending (nfy2) nodal indices in y dimension (Fig. 1a for coordinates). Lines 2 to 5 specify the x, z, and y coordinates for the four points defining the fault segment. For lines 2 to 5, the first parameter is the point index for these four points, and the other three parameters are x, z, and y coordinates for this point.
The sixth line specifies the geometry of the subducted slab beneath this fault segment. The three parameters are slab dip and starting and ending depths of slab.

4.2.4. Output and Postprocessing

Output.c outputs coordinates, velocity, temperature, viscosity, and surface normal stresses. In order to conserve disk space, outputs are in binary mode. Since each processor only contains data for one computational domain, each processor generates output files on its own such that the output operations can theoretically be accomplished in a perfectly parallel manner (if the hardware allows this). For example, if six processors are used for a model run, the code will generate six velocity files: case1.velo.0, case1.velo.1, ..., case1.velo.5. The last number in each file name is the processor index which ranges from 0, 1, ..., N-1 for N processors (On the Intel Paragon, N can be as large as 512, therefore, many files may be generated in this output format. On the Intel Paragon, it is possible to use I/O function calls to store all the data in one file in either a synchronous or asynchronous mode).

Since each processor outputs data only for its own computational domain, postprocessing is needed to combine files from different processors into a single file which stores data for the entire model domain. This can be accomplished with program convert.c which is included in the subdirectory POST_Processing along with an input file runfile_case1 (Appendix C). Program convert.c reads in all the files from Citcom, including information about the layout of processors from the input file runfile_case1, and then generates files storing data for the entire model domain.

The file runfile_case1 is generated from the Citcom run. In runfile_case1, the first and second lines specify the number of nodes and the number of processors in x, z, and y dimensions for the Citcom run, respectively. For a time dependent calculation, output files may contain more than one frame. The third line indicates the frame index for which the data are extracted. This parameter should be equal 1 if the Citcom run is for an
instantaneous Stokes flow calculation. The fourth line gives the prefix of file names for
files generated from Citcom (i.e., the same prefix as specified in line 1 of output section in
the input file input). The last line gives the prefix of file names for output of the combined
files.

To compile and execute this program, type

$cc convert.c -o convert.x -lm
$conver hasta case1

As a result, files including combined_case1.xyzt, combined_case1.v,
combined_case1.tpg will be produced. These files can further be used for computing
other physical quantities such as gravity anomalies, dynamic topography, and ratio of
toroidal to poloidal components and for graphics. Programs that perform these
postprocessing calculations are also included in subdirectory POST_Processing.

4.2.5. Limitations and Precautions

There are two fundamental limitations of this version of Citcom with faults: fault
geometry and domain decomposition. Since fault planes must be on elemental boundaries,
it would be difficult to write a grid generation routine which can generate a mesh for
arbitrary fault geometry. Both the Citcom and postprocessing codes work for models with
the fault geometries presented in Zhong, Gurnis and Moresi [1997]. Users may run into
problems in using the codes for other fault geometries. Under such circumstances, users
are recommended to read Faults_mesh.c and modify it for their own use. For the domain
decomposition, the Citcom code requires that the number of elements in either x, y, or z
dimension be a multiple of the number of processors in that dimension (i.e., the number of
elements for each processor is identical).

Other limitations and precautions are listed as follows:

(1) Faults are fixed in time dependent convection models.

(2) Only 3D Cartesian geometry and the conjugate gradient solver can be used.
(3) When multiple processors are used in the vertical dimension for models with faults, the number of elements in the vertical dimension for each computational domain must be greater than the number of elements for the lithosphere, nfz (the fifth line in Grid section of Appendix A).

(4) Velocity boundary conditions with specified velocity are not fully tested.

(5) Frictional forces on faults are applied in a fixed direction with a constant magnitude. The frictional forces are within the fault planes and are opposite to the surface motion of plates.

(6) To achieve an accurate pressure field, use one order of magnitude smaller tolerance for incompressibility than for velocity.

(7) In the input file for specifying fault geometry, fault segments should be arranged one after another and they should be geometrically connected. The order of four points specifying the geometry for each fault segment is important. Users are recommended to follow the sample file.
References


Appendix A. Sample input file for citcom.mpi

# Output:
  datafile="case1"
  logfile="case1.log"
  maxstep=1  # total timesteps
  storage_spacing=1  # write data every ...
  verbose=off  #

# Grid:
  Geometry=cart3d  # cart2d,cart3d or axi, at the moment
  nodex=97 nodez=33 nodey=49
  dimenx=4.0 dimenz=1.0 dimeny=3.0
  nprocx=3 nprocz=1 nprocy=2
  nzf=6
  nz_410=17
  nz_lmantle=23
  z_lith=0.0666666667;
  z_410=0.2733333333;
  z_lmantle=0.4466666667;

# Faults:
  faults=1 faults_comparison=0
  fault_coord_file=fault_coord_file3d.dat

# Viscosity:
  TDEPV=on rheol=1
  num_mat=6
  viscE=0.0,0.0,0.0,0.0,0.0,0.0
  viscT=0.0,0.0,0.0,0.0,0.0,0.0
  visc0=2.0e7,2.0e1,2.0e1,1.e0,2.0e3,2.0e7
  VMIN=on visc_min=1.0e-3
  VMAX=on visc_max=1.0e03
  VISC_UPDATE=off
  SDEPV=off sdepv_misfit=0.010
  sdepv_expt=3,3,3,1,3,3

# BOUNDARY CONDITIONS
  topvbc=0  # velocity boundary conditions top and bottom
    topvbxval=0.0  topvbyval=0.0
  botvbc=0
    botvbxval=0.0  botvbyval=0.0
  toptbc=1  # temperature bc's top and bottom
    toptbcval=0.0
  bottbc=1
    bottbcval=1.0

# Miscellaneous parameters
  rayleigh=1.782e5  # Rayleigh number
  Ra_670=0.0  #0.0% density jump
  clapeyron670=-0.1131313  #-3.5MPa/K
  transT670=1.0  # 1600 K
  width670=0.025  # 37.5 km
  Q0=0.0  # Dimensionless internal heating rate
  frictional=0
  fric_force=0.0  # for 5 MPa
plate_velocity=0.03    # unit : m/year

# Dimensional parameters
layerd=1500.0 # km
ReferenceT=1600.0
density=3300.0
thermdiff=1.0e-6
gravacc=10.0
thermexp=2.0e-5
cp=1250
wdensity=0.0

# Solver
stokes_flow_only=1
Solver=cgrad node_assemble=1    # conjugate gradient
precond=on
aug_lagr=on
aug_number=5.0e3
vlowstep=1000           # Enough to get very accurate soln.
piterations=375         # Uzawa iteration loops.
accuracy=1.0e-3         # Desired accuracy of Uzawa algorithm.
tole_compressibility=1e+4
adv_sub_iterations=2
finetunedt=0.6
Appendix B. Sample file `fault_coord_file3d.dat` for fault geometry

```
9
1 0 3 4 13 13
1 0.0444444444 0.0 0.75
2 0.0444444444 0.0666666667 0.75
3 0.0666666667 0.0666666667 0.75
4 0.0666666667 0.0 0.75
90 0.0666666667 0.0666666667
2 0 48 13 13
1 0.0666666667 0.0 0.75
2 0.0666666667 0.0666666667 0.75
3 2.00 0.0666666667 0.75
4 1.90 0.0 0.75
90 0.0666666667 0.0666666667
3 0 56 13 13
1 1.90 0.0 0.75
2 2.00 0.0666666667 0.75
3 2.115 0.0666666667 0.75
4 2.00 0.0 0.75
90 0.0666666667 0.0666666667
4 0 56 13 15
1 2.00 0.0 0.75
2 2.115 0.0666666667 0.75
3 2.115 0.0666666667 0.80
4 2.00 0.0 0.80
60 0.077 0.52
5 0 56 15 35
1 2.00 0.0 0.80
2 2.115 0.0666666667 0.80
3 2.115 0.0666666667 2.20
4 2.00 0.0 2.20
60 0.077 0.52
6 0 56 35 37
1 2.00 0.0 2.20
2 2.115 0.0666666667 2.20
3 2.115 0.0666666667 2.25
4 2.00 0.0 2.25
60 0.077 0.52
7 1 56 48 37 37
1 2.00 0.0 2.25
2 2.115 0.0666666667 2.25
3 2.00 0.0666666667 2.25
4 1.90 0.0 2.25
90 0.0666666667 0.0666666667
8 1 48 4 37 37
1 1.90 0.0 2.25
2 2.00 0.0666666667 2.25
3 0.0666666667 0.0666666667 2.25
4 0.0666666667 0.0 2.25
90 0.0666666667 0.0666666667
9 1 4 3 37 37
1 0.0666666667 0.0 2.25
2 0.0666666667 0.0666666667 2.25
3 0.0444444444 0.0666666667 2.25
4 0.0444444444 0.0 2.25
90 0.0666666667 0.0666666667
```
Appendix C. Sample input file runfile_case1 for code convert.x

97 33 49
3 1 2
1
case1
combined_case1