CAVS REPORT

MSU.CAVS.CMD.2010-R00??

Introduction to Discrete Dislocation Dynamics Simulations using microMegas: A manual

Project: DOE-SRCLID
Project Leaders: Mark F. Horstemeyer, Paul T. Wang
Task 2: Cyberinfrastructure
Task 2 Leader: Tomasz Haupt

Prepared by: Florina Ciorba
Center for Advanced Vehicular Systems
Mississippi State University
Mississippi State, MS 39762

Web site: http://www.cavs.msstate.edu

For public release.
# Table of Contents

1. Overview of microMegas ................................................................. 4  
2. Setup .............................................................................................. 4  
3. mM ver 1.0 - Code organization ....................................................... 4  
4. Input files (mM/in) ........................................................................... 9  
5. Output files (mM/out) ...................................................................... 9  
6. Simulation source files (mM/src/simu) ............................................. 12  
7. Compiling microMegas using different Makefiles (mM/bin/README) ................................................................. 16  
   1. Compiling serial microMegas optimized with compiler options (i.e., mM ver 1.0) .................................................... 16  
   2. Compiling serial microMegas optimized with compiler options and instrumented with TAU for profiling (i.e., 'mM ver 1.0 + TAU profiling') ........................................................................................................................................... 16  
   3. Compiling serial microMegas optimized with compiler options and instrumented with TAU for tracing (i.e., 'mM ver 1.0 + TAU tracing') ........................................................................................................................................... 17  
   4. Compiling the parallelized microMegas optimized with compiler options (i.e., 'mMpar ver 1.0') ............................. 18  
   .............................................................................................................. 18  
   5. Compiling the un-instrumented parallelized microMegas optimized with compiler options (i.e., 'mMpar ver 1.0 + Intel options') ................................................................. 19  
   .............................................................................................................. 19  
   6. Compiling parallel microMegas optimized with compiler options and instrumented with TAU for profiling (i.e., 'mMpar ver 1.0 + TAU profiling') ......................... 21  
   7. Compiling parallel microMegas optimized with compiler options and instrumented with TAU for tracing (i.e., 'mMpar ver 1.0 + TAU tracing') ........................................ 22  
   8. Compiling non-optimized serial microMegas (i.e., original mM ver 3.2) ................................................................. 23  
8. Execution for production runs .............................................................. 27  
   1. Simple batch execution (command line execution) ......................... 27  
   2. PBS batch execution (command line execution) .............................. 29  
9. Sample results ................................................................................ 36  
10. Appendix ....................................................................................... 37  
    1. The detailed tree-like structure of the code is given below: ................................................................. 37  
    2. Profiling callgraphs of the serial and OpenMP-based parallel microMegas execution ........................................ 56
Introduction to Discrete Dislocation Dynamics Simulations using microMegas

1. Overview of microMegas

MicroMegas (also known as 'mM') is an open source program for DD (Dislocation Dynamics) simulations originally developed at the 'Laboratoire d'Etude des Microstructures', CNRS-ONERA, France (http://zig.onera.fr/mm_home_page/index.html). mM is a free software under the terms of the GNU General Public License as published by the Free Software Foundation (http://www.gnu.org/philosophy/free-sw.html). Discrete dislocation dynamics (DDD) is a numerical tool used to model the plastic behavior of crystalline materials using the elastic theory of dislocations. DDD is the computational counterpart to in situ TEM tests. MicroMegas is a legacy simulation code used to study the plasticity of mono-crystalline metals, based on the elasticity theory that models the dislocation interactions into an elastic continuum. In crystalline materials, plastic deformation may be explained by (i) twinning, (ii) martensic transformation or/and (iii) dislocation interactions (see Figure 1).

![Figure 1 3-D simulation box with a dislocation density of 10^{12}/m^2](image)
The elementary objects treated by the code are discrete dislocation lines embedded into an elastic continuum. To obtain a better efficiency without loss of accuracy, time and space are discretized, as well as the orientations (i.e., the characters) of the dislocation lines. The continuous shapes of these lines are decomposed into a set of straight segments lying on a lattice homothetic to that of the considered crystalline material. The parameter of this underlying lattice is adjusted to the treatment of the smallest length scale of interest for a given problem. The dislocations move by discrete jumps on the underlying lattice, which allows including the slip geometry for a range of different crystallographic structures (fcc, bcc, dc, hcp) and cutting-off very small length scales below which elasticity theory breaks down. The line orientations are discretized in such a way as to include at least the screw and edge directions in each potential slip plane, the character of the junctions formed when two attractive, non-coplanar segments cross each other and the glide direction of all the discrete dislocation characters in all potential slip planes.

The elastic properties of dislocations (line tension, dislocation-dislocation interactions, etc ...) and the computation of the effective (net) force on each dislocation segment directly follow from the classical theory of dislocations. The equation of motion of each segment is computed at its mid-point, using classical methods similar to those employed in MD simulations. Algorithmic parameters like the time step, the minimum length of the segments and their maximum travel distance during a step are optimized in such a way as to reproduce known solutions for typical elastic problems like the critical stress for a Frank-Read source.

Dislocation core properties, which are relevant of an atomic-scale treatment, cannot be accounted for in a purely elastic framework. Two of the most important ones are: stress vs. mobility relation and energetics of the cross-slip mechanism. They are incorporated through the use of "local rules" that apply to each segment at each step of the simulation. These rules are determined from the available information on core mechanisms, as obtained from continuum modeling, experiment and, increasingly in the past few years, atomistic modeling.

Periodic boundary conditions (PBC) are implemented in order to ensure that the mean-free path of dislocations in the simulation is larger than their "physical" mean-free path. The dimensions of the elementary simulated cell depends on the problem investigated and can go up to typically 10-20 cubic microns for simulations of single crystals. A simple but efficient method (i.e., the Greengard algorithm), deriving from order-N algorithms for the many-body problem, is used for the computation of long range elastic interactions of dislocations. All the critical parts of the simulation are numerically optimized and the simulation can be run on multi-core systems (using OpenMP threads) or on computer clusters (using the MPI processes).

The output of the simulation yields information about the microstructure, local quantities of interest (internal stresses, dislocation densities, slip systems activity) and on the global mechanical response. This code is perfectly adapted for performing mass simulations on single crystals. When coupled to a finite element code, thus replacing the usual constitutive relations, it can be used to investigate more complex materials and loading conditions (this coupling is not presently included in the standard distribution of ‘mM’).
2. Setup

microMegas can be freely downloaded from the original development site at the French Aerospace Lab (http://zig.onera.fr/mm_home_page/doc/Releases.html) or from the CAVS Cyberinfrastructure Wiki Page (https://ccg.hpc.msstate.edu/mediawiki/index.php/Repository_of_codes, mM ver 1.0 – serial mM [original ver. 3.2] with various Intel Compiler Optimizations, or mMpar ver. 1.0 – parallel version of mM [original ver. 3.2], where force calculations for each segment are calculated in parallel using OpenMP threads).

This document describes how to install, configure and run DDD simulations using mM ver. 1.0 (with Intel Compiler optimizations) and mMpar ver. 1.0 (with Intel Compiler optimizations and OpenMP threads).

Installation instructions for mM ver. 1.0 and mMpar ver. 1.0 can also be found in the ‘readme’ files provided in each directory and subdirectory of the code.

mM can be run in *batch mode* to get data analyzed with conventional graphical display programs (examples of Gnuplot scripts are provided) or it can be used in *interactive mode* to simply visualize dislocations activity. Herein, we describe how to run the code in batch mode. For instructions on how to run mM in interactive mode, please refer to the ‘readme’ files provided with the code.

The general workflow for running discrete dislocation dynamics simulations using microMegas is illustrated in the figure below:
### 3. mM ver 1.0 - Code organization

The brief tree-like structure of the code is the following:

| mM/ (root directory) | |--- README.txt | full structure and description of the code |
|--- bin (directory) | directory containing the makefiles and the various executables of the code |
|--- changes.log | changes made to the code of the original 'mM' version 3.2 of B. Devincre |
|--- exec (directory) | directory containing executable helping scripts |
|--- in (directory) | directory containing all the simulation input files |
| | |--- ContCu | input file with parameters used for the simulation (SIDEJA - if 0 it starts a new simulation, if 1 - it restarts from a previous simulation |
| | |--- Cu | the material variables file |
| | |--- README | |
| | |--- SEG_relax | output of the simulation (after the initial relaxation or later) that can be used latter as initial configuration file |
| | |--- SEG_save | outputs of the simulation (after the initial relaxation or later) that can be used latter as initial configuration file |
| | |--- SegCu | the initial dislocation configuration file (description of the initial dislocation configuration file format is given at the end) |
| | |--- Segments | file describing the initial number, type and characteristics of dislocation segments |
| | |--- b_plan | the input files needed to run polyphase simulations |
| | |--- b_polyl | the input files needed to run polyphase simulations |
| | |--- b_spher | the input files needed to run polyphase simulations |
|--- optiz (directory) | results from running the optimized executables |
|--- out (directory) | directory where the output files of the simulations codes are written |
| | |--- BVD.CFC | the set of reference vectors used in the simulation for a given crystal |
| | |--- README | |
| | |--- bigsave.bin | binary file containing everything needed to restart a simulation if accidentally stopped |
| | |--- film.bin | binary file where the coordinates of segments are periodically saved to buildup a trajectory file |
| | |--- gamma | evolution of gamma for all existing slip systems |
| | |--- gammap | evolution of the instantaneous gamma dot for all the slip systems |
| | |--- manip.txt | |
| | |--- rau | evolution of rho, the dislocation density,
for all the slip systems
| |-- raujonc - evolution of the jonction density and number
for all slip systems
| |-- resul - GNU plotting script for plotting various
simulation data (run 'gnuplot resul' to see the
results)
| |-- sigeps - output file containing the stress, strain and
other information (a compagnon file to "stat")
| |-- stat - where most of the global statistics of the
simulation are written
| |-- test_boucle.txt
| |-- travapp - evolution of the applied mechanical work
(presently do not trust those computations)
| `-- travint - evolution of the internal mechanical work
(presently do not trust those computations)
|-- prof (directory) - various profiling screenshots from Paraprof
and Gprof
|-- src (directory) - source code directory
| |-- simu (directory) - directory with module sources for the actual
simulation
| | |-- 01constantes.f90
| | |-- 02bricamat.f90
| | |-- 03varbase.f90
| | |-- 04varglob.F90
| | |-- 05intergra.f90
| | |-- 06debug.f90
| | |-- 07init.F90
| | |-- 08connec.f90
| | |-- 09elasti.F90 - source file where interaction force is
calculated for each segment
| | |-- 10dynam.F90
| | |-- 11topolo.f90
| | |-- 12contact.f90 - source file where the interactions between
segments are updated
| | |-- 13resul.F90
| | |-- 14bigsave.F90
| | |-- CopyRight.txt
| | |-- base.f90
| | |-- carto.f90
| | |-- help.txt
| | `-- microstructure.F90

The main directory (mM) contains several subdirectories, of which the most important ones are:

mM/bin – directory containing the makefiles and the various executables of the code
mM/in – directory containing all the simulation input files
mM/out – directory where the output files of the simulations codes are written
mM/src/simu – directory with module sources for the actual simulation

4. Input files (mM/in)
ContCu

‘ContCu’ is the input file with parameters used for the simulation. For instance, one can select the type of simulation (initial or restart from previous simulation) via the parameter SIDEJA. One can also select whether cross-slip displacement of dislocations is desired by setting the GLDEV parameter accordingly (‘T’ for enabled and ‘F’ for disabled). Also, one can set the total number of simulation steps via the NSTEP parameter. Each simulation time step corresponds to $10^{-9}$ real time seconds. Therefore, for a very small simulation use NSTEP=500 while for a long running simulation set NSTEP to anything from $10^6$ and above. Finally, one can also select how often should the simulation save the current state of the code, via the KISAUVE, KISTAT, KKIM and KPREDRAW parameters. For more details, see the existing file.

Cu

‘Cu’ is the file containing the material variables. See the existing file for more details.

SegCu

‘SegCu’ is the file containing the initial dislocation configuration (e.g., the active slip systems, the number of segments, the dimensions of the simulation reference volume box, etc). See the bottom of the existing file for more details.

Segments

‘Segments’ is the file describing the initial number, type and characteristics of the dislocation segments. See the existing file for more details.

b_plan, b_poly1, b_spher

‘b_plan’, ‘b_poly’ and ‘b_spher’ are the input files needed to run polyphase simulations. See the existing files for more details.

5. Output files (mM/out)

The most important output files are briefly described below. For more details on the content and meaning of each file, please see the existing files.

BVD.CFC

‘BVD.CFC’ is the set of reference vectors used in the simulation for a given crystal

bigsave.bin

‘bigsave.bin’ is a binary file containing everything needed to re-start a simulation if it is accidentally stopped

film.bin
‘film.bin’ is a binary file where the coordinates of segments are periodically saved to build up a trajectory file

- **gamma**
  ‘gamma’ is a file containing the evolution of gamma for all existing slip systems

- **gammap**
  ‘gammap’ is a file containing the evolution of the instantaneous gamma dot for all the slip systems

- **rau**
  ‘rau’ is a file containing the evolution of rho, the dislocation density, for all the slip systems

- **raujonc**
  ‘raujonc’ is a file containing the evolution of the junction density and number for all slip systems

- **resul**
  ‘resul’ is a GNU plotting script for plotting various simulation data (run 'gnuplot resul' to see the results)

- **sigeps**
  ‘sigeps’ is an output file containing the stress, strain and other information (an accompanying file to "stat")

- **stat**
  ‘stat’ is a file where most of the global statistics of the simulation are written

- **travapp**
  ‘travapp’ is a file containing the evolution of the applied mechanical work (presently do not trust those computations)

- **travint**
  ‘travint’ is a file containing the evolution of the internal mechanical work (presently do not trust those computations)

### 6. Simulation source files (mM/src/simu)

MicroMegas is written in a mix of Fortran 90 and Fortran 95, consists of 18 source modules and contains roughly 25,000 lines of code. Figure 3 gives the pseudocode of the MAIN module in MicroMegas. A typical simulation run in MicroMegas requires somewhere between $10^6$ to $10^9$ time steps to gain more insight about the plastic deformation range. Simulations with a smaller
number of steps will very likely not capture the plastic range of deformation – the region of interest for the materials scientists studying plastic deformation in crystalline materials.

- **01constantes.f90**
  - module containing the declaration of all simulation constants

- **02bricamat.f90**
  - module containing a toolbox of useful subroutines for, e.g., dot products, etc.
    - uses 01constantes module

- **03varbase.f90**
  - module containing the data structures and variables database (lattice, etc)
    - uses 01constantes module

- **04varglob.F90**
  - module containing initializations of all the constants and variables common to all the modules of the main program
    - uses 01constantes and 03varbase modules

- **05intergra.f90**
  - module that enables integration with the graphical module (for interactive modem mM simulations)
    - uses 04varglob module

- **06debug.f90**
  - module containing the subroutines required for debugging, i.e., subroutine Conf(i) and subroutine verif_reseau
    - uses 01constantes, 02bricamat, 03varbase and 04varglob modules

- **07init.F90**
  - module that reads the input files and assigns values to all other variables not initialized in 04varglob
    - uses 02bricamat, 04varglob, 06debug and carto modules

- **08connec.f90**
  - module that checks the connectivity between all segments (not CPU intensive)
    - uses 04varglob and 06debug modules

- **09elasti.F90**
• **10dynam.F90**
  - module where the moving velocity of each segment is calculated
  - uses 01constantes, 04varglob, 06debug and 08connec modules

• **11topolo.f90**
  - module containing the procedures used to generate the boundary conditions, to discretize the dislocation lines into segments and to locate the segments before they are eliminated
  - uses 04varglob, 06debug, 08connec and microstructure modules

• **12contact.f90**
  - module containing simple displacements and where the interactions between segments are updated in four steps: (1) check for every possible obstacle, (2) check for every possible contact reaction (annihilation, junction formation, etc.), (3) make the reactions, and (4) update the positions of the segments
  - uses 02bricamat, 04varglob, 06 debug, 08connec and microstructure modules

• **13resul.F90**
  - module where the results and statistics are calculated
  - uses 02bricamat, 04varglob, 06 debug and microstructure modules

• **14bigsave.F90**
  - module that saves the simulation state either when the number of selected time steps has elapsed or to be able to restart a computation
  - uses 02bricamat and 04varglob modules

• **15main.F90**
  - main module containing the simulation time loops; it calls all other modules either implicitly or explicitly; for more details see Figure 3
  - uses 02bricamat, 04varglob, 06debug, 07init, microstructure, 09elasti, 10dynam, 11topolo, 12contact, 13resul and 14bigsave modules

• **base.f90**
  - module that reads all the data of the main program, in three groups of files:
    - **materiaux** – given material physical properties
    - **control** – given simulation parameters
    - **seg3D** – regroups the characteristics of the segments given at the beginning of the simulation
  - uses 01constantes, 02bricamat and 04varglob modules
• **carto.f90**

• **microstructure.F90**

  – module containing the subroutines used to detect the obstacles, i.e., subroutine barriere_spherique and subroutine barriere_plane; it prints the segments structure

  o uses 02bricamat, 03varbase, 04varglob, 06debug and 08connec modules

```fortran
! Module MAIN: simulation time loop
TIME: do = 1, STEPS
...
call SOLLI ! Apply load
call DISCRETI ! Discretize the simulation volume into dislocation lines/segments
call FORCE ! Calculate interaction forces
!FORCE calls SIGMA_INT_CP to calculate short range interaction forces
!FORCE calls SIGMA_INT_LP to calculate long range interaction forces
call DEPPREDIC ! Predict moving segments
call UPDATE ! Search for obstacles, determine & make contact reactions, update positions of segments
call CORRIGER_CONFIG ! Check the connections between all segments
...
endo TIME
```

**Figure 3** Pseudocode of the MAIN microMegas module illustrating the most important subroutines called during each simulation time step

7. **Compiling microMegas using different Makefiles**

(mM/bin/README)

1. **Compiling serial microMegas optimized with compiler options (i.e., mM ver 1.0)**

   **Step 1.** Select the makefile from the list available in this directory (e.g. opt0-Makefile, ..., opt12-Makefile). Then type:

   ```bash
   make -f opt0-Makefile clean
   make -f opt0-Makefile mm
cp ./mm to ./mm.opt0
   ```

   **Step 2.** To run the simulation compiled with the optimization flags declared in the makefile, simply type:

   ```bash
   ./mm.opt0 > screen &
   ```
to run the simulation in batch mode, or to record the running time and save the output in a separate
directory and files, type:

```
/usr/bin/time -p -o ../optimiz/opt0/mm.opt0.time ./mm.opt0 | tee
../optimiz/opt0/mm.opt0.log
```

2. **Compiling serial microMegas optimized with compiler options and
   instrumented with TAU for profiling (i.e., ‘mM ver 1.0 + TAU
   profiling’)**

TAU (Tuning & Analysis Utilities) is an open source toolkit for application profiling and tracing. It can be

*Note*: TAU is assumed to be installed on your system (e.g. using swsetup tau). For more information on
using TAU, contact the system administrator.

To select the desired type of profiling you are interested with TAU, modify ‘bin/tau.conf’ accordingly.

**Step 1.** Select the makefile from the list available in this directory (e.g. `topt0-Makefile`, ..., `topt11-
Makefile`). For the makefile with the first compiler-based optimization flags, i.e., `topt0-Makefile`, type:

```
make -f topt0-Makefile clean
make -f topt0-Makefile mm
cp ./mm to ./mm.topt0
```

**Step 2.** To run the simulation compiled with the optimization flags declared in the makefile, simply type:

```
./mm.topt0 > screen &
```

to run the simulation in batch mode, or to record the running time and save the output in a separate
directory and files, type:

```
/usr/bin/time -p -o ../optimiz/topt0/mm.topt0.time ./mm.topt0 | tee
../optimiz/topt0/mm.topt0.log
```

3. **Compiling serial microMegas optimized with compiler options and
   instrumented with TAU for tracing (i.e., ‘mM ver 1.0 + TAU tracing’)**

TAU (Tuning & Analysis Utilities) is an open source toolkit for application profiling and tracing. It can be
Note: TAU is assumed to be installed on your system (e.g. using `swsetup tau`). For more information on using TAU, contact the system administrator.

To select the desired type of tracing you are interested with TAU, modify `bin/tau.conf` accordingly.

Step 1. Select the makefile from the list available in this directory (e.g. `topt0-Makefile`, ..., `topt11-Makefile`). For the makefile with the first compiler-based optimization flags, i.e., `topt0-Makefile`, type:

    make -f topt0-Makefile clean
    make -f topt0-Makefile mm
    cp ./mm to ./mm.topt0.tr

Step 2. To run the simulation compiled with the optimization flags declared in the makefile, simply type:

    ./mm.topt0.tr > screen &

To run the simulation in batch mode, or to record the running time and save the output in a separate directory and files, type:

    /usr/bin/time -p -o ../optimiz/topt0.tr/mm.topt0.tr.time ./mm.topt0.tr | tee ../optimiz/topt0.tr/mm.topt0.tr.log

4. Compiling the parallelized microMegas optimized with compiler options (i.e., 'mMpar ver 1.0')

Note: 'mMpar ver 1.0' is the OpenMP-based parallelized version of the 'mM ver 1.0' code.

Step 1. Configure the target system for executing OpenMP programs, by ensuring that the environment variables used by the chosen compiler and its OpenMP extension are properly set. For a quad-core Linux system running SuSE SLES 10, and the Intel Compiler, the following values are recommended.

    export OMP_THREAD_NUM=4
    export KMP_AFFINITY=verbose,respect,granularity=core,scatter
    export KMP_LIBRARY=turnaround
    export KMP_SETTINGS=1
    export KMP_STACKSIZE=512m
    export KMP_VERSION=.TRUE.

For more details on the values and meaning of these environment variables, please consult the Intel Compiler manual and its OpenMP specification. Note that these environment variables are specific to the Intel Compiler and its OpenMP specification, and that they may differ based on the compiler of your choice and the specifics of its own OpenMP extension.

Step 2. Look for `openMP-Makefile` file in `mMpar/bin/`. Then type:

    make -f openMP-Makefile clean
    make -f openMP-Makefile mm_omp

Step 3. To run the optimized parallel simulation, simply type:
5. Compiling the un-instrumented parallelized microMegas optimized with compiler options (i.e., 'mMpar ver. 1.0 + Intel options')

**Step 1.** Configure the target system for executing OpenMP programs, by ensuring that the environment variables used by the chosen compiler and its OpenMP extension are properly set. For a quad-core Linux system running SuSE SLES 10, and the Intel Compiler, the following values are recommended.

```bash
export OMP_THREAD_NUM=4
export KMP_AFFINITY=verbose,respect,granularity=core,scatter
export KMP_LIBRARY=turnaround
export KMP_SETTINGS=1
export KMP_STACKSIZE=512m
export KMP_VERSION=.TRUE.
```

For more details on the values and meaning of these environment variables, please consult the Intel Compiler manual and its OpenMP specification. Note that these environment variables are specific to the Intel Compiler and its OpenMP specification, and that they may differ based on the compiler of your choice and the specifics of its own OpenMP extension.

**Step 2.** Select the makefile from the list available in the ‘mMpar/bin’ directory (e.g. paropt0-Makefile, ..., par-opt11-Makefile). For the makefile with the first compiler-based optimization flags, i.e., par-opt0-Makefile, type:

```bash
make -f par-opt0-Makefile clean
make -f par-opt0-Makefile mm_omp
cp ./mm_omp to ./mm_omp.opt0
```

**Step 3.** To run the simulation compiled with the optimization flags declared in the makefile, simply type:

```bash
./mm_omp. opt0 > screen &
```

to run the simulation in batch mode, or to record the running time and save the output in a separate directory and files, type:

```bash
/usr/bin/time -p -o ../optimiz/mm_omp.opt0/mm_omp.opt0.time ./mm_omp.opt0 | tee ../optimiz/mm_omp.opt0/mm_omp.opt0.log
```
6. Compiling parallel microMegas optimized with compiler options and instrumented with TAU for profiling (i.e., 'mMpar ver 1.0 + TAU profiling')

**Step 1.** Configure the target system for executing OpenMP programs, by ensuring that the environment variables used by the chosen compiler and its OpenMP extension are properly set. For a quad-core Linux system running SuSE SLES 10, and the Intel Compiler, the following values are recommended.

```bash
export OMP_THREAD_NUM=4
export KMP_AFFINITY=verbose,respect,granularity=core,scatter
export KMP_LIBRARY=turnaround
export KMP_SETTINGS=1
export KMP_STACKSIZE=512m
export KMP_VERSION=.TRUE.
```

For more details on the values and meaning of these environment variables, please consult the Intel Compiler manual and its OpenMP specification. Note that these environment variables are specific to the Intel Compiler and its OpenMP specification, and that they may differ based on the compiler of your choice and the specifics of its own OpenMP extension.

*Note:* TAU is assumed to be installed on your system (e.g. using `swsetup tau`). To select the desired type of profiling you are interested with TAU, modify `bin/tau.conf` accordingly.

**Step 2.** Select the makefile from the list available in the ‘mMpar/bin’ directory (e.g. `par-topt0-Makefile`, ..., `par-topt11-Makefile`). For the makefile with the first compiler-based optimization flags, i.e., `par-topt0-Makefile`, type:

```bash
make -f par-topt0-Makefile clean
make -f par-topt0-Makefile mm_omp
cp ./mm_omp to ./mm_omp.topt0
```

**Step 3.** To run the simulation compiled with the optimization flags declared in the makefile, simply type:

```
./mm_omp.topt0 > screen &
```

to run the simulation in batch mode, or to record the running time and save the output in a separate directory and files, type:

```bash
/usr/bin/time -p -o ../optimiz/mm_omp.topt0/mm_omp.topt0.time ./mm_omp.topt0 | tee ../optimiz/mm_omp.topt0/mm_omp.topt0.log
```

7. Compiling parallel microMegas optimized with compiler options and instrumented with TAU for tracing (i.e., 'mMpar ver 1.0 + TAU tracing')

*Note:* TAU is assumed to be installed on your system (e.g. using `swsetup tau`).
To select the desired type of tracing you are interested with TAU, modify `bin/tau.conf` accordingly.

**Step 1.** Select the makefile from the list available in the `mMpar/bin` directory (e.g. par-topt0-Makefile, ..., par-topt11-Makefile). For the makefile with the first compiler-based optimization flags, i.e., **par-topt0-Makefile**, type:

```
make -f par-topt0-Makefile clean
make -f par-topt0-Makefile mm_omp
cp ./mm_omp to ./mm_omp.topt0.tr
```

**Step 2.** To run the simulation compiled with the optimization flags declared in the makefile, simply type:

```
./mm_omp.topt0.tr > screen &
```

to run the simulation in batch mode, or to record the running time and save the output in a separate directory and files, type:

```
/usr/bin/time -p -o ../optimiz/mm_omp.topt0.tr/mm_omp.topt0.tr.time
./mm_omp.topt0.tr | tee ../optimiz/mm_omp.topt0.tr/mm_omp.topt0.tr.log
```

8. **Compiling non-optimized serial microMegas (i.e., original mM ver 3.2)**

**Step 1.** Compile the simulation as follows:

To compile microMegas, you need to build a makefile dedicated to the machine you want to run the simulation, in the `mM/bin` or `mMpar/bin` directory. Solutions already exist for many different platforms; you should be able to do your one without too much effort.

The "config" file is the part of "makefiles" which is the same on all the machines

To create a new machine 'makefile' you must add at the end of config the corresponding ".PHONY" definition.

Then, you need to build your one "Make_DEFS" file. The latter must contains all the headers useful for your new machine. See the following examples.

Make_DEFS.amd -> An AMD Linux platform with gcc and the Intel FORTRAN compilers
Make_DEFS.dec -> A DEC Alpha machine with the native C and FORTRAN compilers
Make_DEFS.g5   -> An Apple G5 machine with gcc and the IBM FORTRAN compilers
Make_DEFS.mac  -> An Apple G4 or G3 Machine with gcc and the ABSOFT FORTRAN compilers
Make_DEFS.mad  -> An AMD Cluster
Make_DEFS.madmax -> A cluster of Xeon machines with gcc and the Intel FORTRAN cimpilers
Make_DEFS.pc   -> A simple PC workstation
Make_DEFS.sgi  -> An SGI Itanium machine with gcc and the Intel(64) FORTRAN Compiler etc....
Once you have made your "Make_DEFS.machine_type", type:

```make
make -f config machine_type
```

For instance for my machine I simply type "make -f config mac"

At that stage you should have a "makefile" file created in the bin directory

**Step 2.** Then, according to the version of microMegas you want to execute, type:

```make
make or make all
```

to compile all the binaries (this does not include the MPI binary)

```make
make mm
```

to compile only the batch version of the simulation

```make
make gmm
```

to compile only the simulation with its graphical interface (interactive mode)

```make
make mm_omp
```

to compile only the batch version for OpenMP parallel threads

```make
make mmp
```

to compile only the batch version for MPI clusters

```make
make cam
```

to compile only the graphical interface (needed to see the simulation film)

```make
make base
```

to compile only the code needed to generate the simulation vectors base

```make
make confinit
```

to compile only the code needed to generate random intitial configurations

```make
make pavage
```

to compile only the code needed to generate the database needed for the simulation interfaces

```make
make clean
```

to sweep out all the useless pieces of codes

```make
make cleanall
```

to clean up everything

**Step 3.** To run the simulation simply type

```make
mm > screen &
```

to run the simulation in batch mode

```make
gmm
```

to run the simulation in interactive mode and with the graphic interface

```make
mm_omp
```

to run the OpenMP-based simulation in batch mode, assuming all the OpenMP-related environment variables are set as described in Subsection 7.5, Step 1.
mpirun -np "x" -machinefile ../in/hosts.dd mmp > screen &
to run the MPI batch simulation

Step 4. Some additional tools are:

**cam**
The camera code to see after and during calculations the film of the simulation

**confinit**
The code used to buildup initial configurations

**base**
The code you can use to generate alone the base of vectors used in the simulation

**pavage**
The code used to generate the interfaces files "b_poly" needed to simulate periodic polycrystals

Step 5. Where and who is who

All the inputs data are defined in the directory "mM/in". Take a look to the README file in this directory for more information.

All the outputs data are written in the directory "mM/out". Take a look to the README file in this directory for more information.

8. Execution for production runs

To get an idea of the type of simulations that can be conducted with microMegas, we give here the parameters of a representative simulation selected in the input files, the compilation and execution commands, as well as the timings we obtained on a dedicated commodity Nehalem quad-core Xeon W3570 processor, running at 3.2GHz with 6GB DDR-3 RAM, SLES 10 OS, 2.6.16.60 Linux kernel.

The simulation parameters of a representative microMegas simulation are:

- 0.5% plastic deformation
- 10x10x10 µm³ simulation box dimensions
- 10¹² m⁻² initial density
- 10 s⁻¹ strain rate in multi-slip conditions

*Note:* Multi-slip calculations were performed to evaluate and demonstrate the efficiency of the parallel version of microMegas.

- Material: representative volume elements of Al (FCC crystal structure with Burgers vector of magnitude b = 2.86 Å) of dimensions 9x10x12 µm³
- loading along the [001] direction
- strain rate of 20 s⁻¹
- temperature of 300 K under periodic boundary conditions
- time step was considered to be $10^{-9}$ seconds

Note: Screw dislocations were not allowed to cross-slip at any time.

### 1. Simple batch execution (command line execution)

To run **serial microMegas for production simulations** (i.e., without any performance profiling), first, compile it as described in Subsections 7.1, 7.4 and 7.8. Second, add the corresponding software modules (compilers, libraries, etc.) to load in your `.bashrc` file (in your home directory, i.e. home/⟨your_username⟩). To load the compiler of your choice, e.g., my choice is Intel Fortran, type:

```
swsetup intel
```

Then in the `mM/bin` directory, to compile only the batch version of the simulation type:

```
make –f serial-Makefile clean
make –f serial-Makefile mm
```

Launch the serial version of the simulation from the same directory by typing:

```
mkdir ../production_runs
/usr/bin/time -p -o ../production_run/mm.time ./mm | tee ../production_runs/mm.log
```

To run **parallel microMegas for production simulations** (i.e., without any performance profiling), first compile it as described in Subsections 7.1, 7.4 and 7.8. Second, add the corresponding software modules (compilers, libraries, visualisers, etc.) to load in your `.bashrc` file. To load the compiler of your choice, e.g., my choice is Intel Fortran, type:

```
swsetup intel
```

For the parallel microMegas simulations, load the MPI libraries, e.g. OpenMPI ver. 1.4.2, by typing:

```
swsetup openmpi-intel-64
```

*Note: To avoid any compilation or execution errors, please make sure that during the selection of any additional libraries, such as MPI, you choose the library version that was compiled using the same compiler of your choice. For instance, if you compile the code using Intel compilers, please select the*
MPI library that was compiled using Intel compilers. Not doing so, may cause unpredictable errors during the simulation.

Then in the ‘mMpar/bin’ (for OpenMP) or ‘mM/bin’ (for MPI) directory, to compile only the batch version of the simulation type:

```
make -f openMP-Makefile clean or make -f openMPI-Makefile clean
make -f openMP-Makefile mm_omp or make -f openMPI-Makefile mmp
```

Launch the parallel version of the simulation from the same directory by typing:

```
mkdir ../production_runs

/usr/bin/time -p -o ../production_run/mm_omp.time ./mm_omp | tee ../production_runs/mm_omp.log

or

/usr/bin/time -p -o ../production_run/mmp.time mpirun -np "x"./mmp | tee ../production_runs/mmp.log
```

2. PBS batch execution (command line execution)

The serial code (mm), the OpenMP-based code (mm_omp) and the OpenMPI-based code (mmp) can be launched either locally (as described in Subsection 8.1) or remotely. For remote execution on high performance compute clusters, a PBS (Portable Batch Script) is needed to submit the execution as a job. Below are three sample PBS scripts one could use to run microMegas in any of the three versions on the talon.hpc.msstate.edu high-performance cluster at HPC2. Each of these scripts can be cut and pasted into a file, e.g., mm.pbs.talon or mm_omp.pbs.talon or mmp.pbs.talon. To submit a pbs script to the jobs queue on talon, first log on to the talon-login node, typing:

```
rlogin talon-login
```

from any HPC2 machine, and then type:

```
qsub mm.pbs.talon or
qsub mm_omp.pbs.talon or
qsub mmp.pbs.talon
```

**PBS script for serial mM execution (mm executable) on talon.hpc.msstate.edu**

```bash
#!/bin/bash
#PBS -N mm
#PBS -q special@talon
#PBS -l nodes=1:ppn=12
#PBS -l walltime=700:00:00
#PBS -m abe
```
PBS script for openMP-based mMpar execution (mm_omp executable) on talon.hpc.msstate.edu

#!/bin/bash
#PBS -N mm_omp
#PBS -q special@talon
#PBS -l nodes=1:ppn=12
#PBS -l walltime=700:00:00
#PBS -m abe
#PBS -j oe
#PBS -r n

# Set the stack size to unlimited
ulimit -s unlimited

# Set the core size to zero
ulimit -c 0

# List all resource limits
ulimit -a

echo "I ran on:"

cat $PBS_NODEFILE

# Change your execution directory to /data/lustre/<your_username> for fast I/O
    cd /data/lustre/<your_username>

# Copy all necessary files from your project directory
    cp -fr /cavs/cmd/data1/users/<your_username>/<your_project_directory>/mm .

# Go to the directory with the 'mm' executable
    cd /data/lustre/<your_username>/mm/bin

# Run the serial 'mm' code with/without cross-slip activated (set GLDEV to be T or F in /data/lustre/<your_username>/mm/in/ContCu)
    /usr/bin/time -p -o ../serial_tests/[no-]cross-slip/mm.time ./mm | tee ../serial_tests/[no-]cross-slip/mm.log

# Move all files from the execution directory (/data/lustre/<your_username>) back to your project directory
    cd /data/lustre/<your_username>/
    cp -fr mm/ /cavs/cmd/data1/users/<your_username>/<your_project_directory>/mm_test/

# "All Done!"
#PBS -V
# Set the stack size to unlimited
ulimit -s unlimited
# Set the core size to zero
ulimit -c 0
# List all resource limits
ulimit -a

echo "I ran on:"
# Print the nodes on which the project will run
cat $PBS_NODEFILE

# Change your execution directory to /data/lustre/<your_username> for fast
# I/O
cd /data/lustre/<your_username>

# Copy all necessary files from your project directory
# (/cavs/cmd/data1/users/<your_username>/<your_project_directory>) to the
# execution directory (/data/lustre/<your_username>)
cp -fr /cavs/cmd/data1/users/<your_username>/<your_project_directory>/mm_omp ./

# Go to the directory with the 'mm_omp' executable
cd /data/lustre/<your_username>/mm_omp/bin

# Run the parallel 'mm_omp' code on 12 parallel threads with/without cross-
# slip activated (set GLDEV to be T or F in
# /data/lustre/<your_username>/mm_omp/in/ContCu)
/usr/bin/time -p -o ../omp_tests/[no-]cross-slip/mm_omp.time ./mm_omp | tee ../omp_tests/[no-]cross-slip/mm_omp.log

# Move all files from the execution directory (/data/lustre/<your_username>)
# back to your project directory
# (/cavs/cmd/data1/users/<your_username>/<your_project_directory>)
cd /data/lustre/<your_username>/
cp -fr mm_omp /
cavs/cmd/data1/users/<your_username>/<your_project_directory>/mm_omp_test/1

# echo "All Done!"

PBS script for MPI-based mMpar execution (mmp executable) on talon.hpc.msstate.edu

#!/bin/bash
#PBS -N mmp
#PBS -q special@talon
#PBS -l nodes=4:ppn=12
#PBS -l walltime=700:00:00
#PBS -m abe
#PBS -j oe
#PBS -r n
#PBS -V
# Set the stack size to unlimited
ulimit -s unlimited
# Set the core size to zero
ulimit -c 0
# List all resource limits
ulimit -a

echo "I ran on:"
# Print the nodes on which the project will run
cat $PBS_NODEFILE
# Change your execution directory to /data/lustre/<your_username> for fast I/O
cd /data/lustre/<your_username>

# Copy all necessary files from your project directory
# (/cavs/cmd/data1/users/<your_username>/<your_project_directory>) to the # execution directory (/data/lustre/<your_username>/)
cp -fr /cavs/cmd/data1/users/<your_username>/<your_project_directory>/mmp .

# Go to the directory with the `mmp` executable
cd /data/lustre/<your_username>/mmp/bin

# Run the parallel `mmp` code on 4x12=48 parallel processes with/without # cross-slip activated (set GLDEV to be T or F in # /data/lustre/<your_username>/mmp/in/ContCu)
/usr/bin/time -p -o ../mmp_tests/[no-]cross-slip/mmp.time ./mmp | tee ../mmp_tests/[no-]cross-slip/mmp.log

# Move all files from the execution directory (/data/lustre/<your_username>) # back to your project directory # (/cavs/cmd/data1/users/<your_username>/<your_project_directory>)
cd /data/lustre/<your_username>/
cp -fr mmp /
cavs/cmd/data1/users/<your_username>/<your_project_directory>/mmp_test/1

#
echo "All Done!"

9. **Sample results**

We also verified the correctness of the parallel OpenMP-based code compared to the serial code. To this end, we conducted two sets of experiments:

**Experiments set #1:** Verify the total number of segments, \(N\), and the number of moving segments with length greater than zero, \(ND\), produced by the serial and parallel application codes.
Experiments set #2: Verify that the mechanical properties produced by the serial and the parallel version of microMegas are the same.

These sets of experiments verify that, indeed, the evolution of $N$ and $ND$ over 100,000 time steps is the same for both serial and parallel application (see Figure 4), and that the stress-strain response, together with the evolution of the total dislocation density as a function of the plastic strain obtained using the parallel code are the same with the ones produced by the serial code (see Figures 5 and 6).

![Figure 4](image)

**Figure 4** Evolution of the total number of segments, $N$, and the number of moving segments, $ND$, over 100,000 time steps. $N$ and $ND$ from the parallel simulation run are equal to those from the serial simulation run.
Figure 5  Validation of stress-strain response obtained from the parallel simulation against the serial simulation over 100,000 time steps.

Figure 6 Validation of the evolution of the total dislocation density as a function of the plastic strain obtained from the parallel simulation against the serial simulation over 100,000 time steps.
Figure 7 Distribution of the serial and parallel execution times across code subroutines. For every simulation run of 10,000 time steps, the time spent in SIGMA_INT_CP is significantly decreased when running on 4 CPUs, yielding a speedup of a factor of 2 for the corresponding parallel time of a simulation run. The total time over 100,000 time steps, calculated as the sum of the running time of the 10 consecutive simulation runs each of 10,000 time steps, is, therefore, decreased from 28 days (serial) to 13.8 days (parallel).

The simulations presented in this manual are for a number of $10^5$ time steps, each of which corresponds to $10^9$ seconds. A single simulation run over as little as 10,000 steps using serial version of microMegas requires as much as 68 hours, on average, and reaches 0.2% of the plastic deformation (or strain %, see Figures 5 and 6) on a Nehalem quad-core Xeon W3570 processor, with 6GB of triple channel 133MHz DDR-3 RAM. As explained in the beginning of this manual, simulations of as many as $10^9$ time steps are needed to reach the desired percentage of deformation, that is, a strain rate as high over 1% as possible.

10. Appendix

1. The detailed tree-like structure of the code is given below:

```
/mM/
|-- README.txt
|-- bin
| |-- Make_DEFS.Cobra
| |-- Make_DEFS.dec
| |-- Make_DEFS.fuj
| |-- Make_DEFS.g5
| |-- Make_DEFS.mac
| |-- Make_DEFS.madmax
| |-- Make_DEFS.madmax2
| |-- Make_DEFS.pc
| |-- Make_DEFS.pgi
| |-- Make_DEFS.sgi
| |-- Make_DEFS.sun
| | |-- Make_DEFS.xeon
| | |-- Makefile
| | |-- Opt2
| | |-- README
| | | |-- config
| | | |-- confinit
| | | |-- make_test
| | | |-- makefile-raptor
| | | | |-- mm.opt0 - optimized executable using the compiler options described in bin/test_runs and it's corresponding Makefile, e.g. bin/opt0-Makefile
| | | | |-- mm.opt1 - optimized executable using the compiler
```
options described in bin/test_runs and it's corresponding Makefile, e.g. bin/opt1-Makefile
|-- mm.opt2 - optimized executable using the compiler options described in bin/test_runs and it's corresponding Makefile, e.g. bin/opt2-Makefile
|-- mm.opt12 - optimized executable using the compiler options described in bin/test_runs and it's corresponding Makefile, e.g. bin/opt12-Makefile
|-- mm.topt0 - optimized executable using the compiler options described in bin/test_runs and it's corresponding Makefile
|-- mm.topt1 - optimized executable using the compiler options described in bin/test_runs and it's corresponding Makefile
|-- mm.topt11 - optimized executable using the compiler options described in bin/test_runs and it's corresponding Makefile
|-- mm_ori - original non-optimized executable (=mM 3.2)
|-- opt0-Makefile - makefile with compiler optimizations used for producing the optimized executables
|-- opt1-Makefile - makefile with compiler optimizations used for producing the optimized executables
|-- opt1.make - makefile with compiler optimizations used for producing the optimized executables
|-- opt12-Makefile - makefile with compiler optimizations used for producing the optimized executables
|-- output.png
|-- output_basic.png
|-- par_run_comp_test - script to run comparative tests in simultaneously
|-- par_single_run - script to run a single test
|-- par_single_run_basic - script to run a single test
|-- par_single_run_stable - script to run a single test
|-- pbs_script - script to run tests on a cluster with PBS batch scheduling
|-- profile1
|-- run_comp_test - script to run comparative tests
|-- run_ptest
|-- run_stest - script to run serial test
|-- run_test
|-- run_tstest
|-- run_tstest_prof - script to run serial tests of the code instrumented with TAU for profiling
|-- run_tstest_trc - script to run serial tests of the code instrumented with TAU for tracing
|-- screen
|-- select.tau - file declaring the routines selected for TAU instrumentation
|-- single_run_basic - script to run a single test
|-- tau.conf - configuration file for TAU instrumentation
|-- test_runs - file describing the various incremental compiler optimizations
|-- topt0-Makefile - makefile with compiler optimizations used for
producing the optimized executables instrumented with TAU

|-- topt1-Makefile - makefile with compiler optimizations used for producing the optimized executables instrumented with TAU

|-- changes.log - changes made to the code of the original 'mM' version 3.2 of B. Devincre

|-- exec
  |-- README
  |-- comp
  |-- compcam
  |-- conf
  |-- configcfc
  |-- configcoin
  |-- confighc
  |-- configvis
  |-- courbes
  |-- gcomp
  |-- getmanip
  `-- save

|-- in
  |-- CRISTALLOGRAPHIE
    |-- BVD.BCC
    |-- BVD.CFC
    |-- BVD.CS
    |-- BVD.HCP
    |-- BVD.ORT
    |-- ROTATIONS_BCC
    |-- ROTATIONS_CFC
    |-- ROTATIONS_CS
    |-- ROTATIONS_CUBIC_TOTALES
    |-- ROTATIONS_HC
    |-- ROTATIONS_HC1
    |-- ROTATIONS_HC2
    |-- ROTATIONS_HC3
    |-- ROTATIONS_HC_ori
    |-- ROTATIONS_ORT
    |-- '-- ROTATIONS_TOTALES
  |-- ContCu - input file with parameters used for the simulation (SIDEJA - if 0 it starts a new simulation, if 1 - it restarts from a previous simulation

  |-- ContCu.orig
  |-- ContCu_Ori
  |-- ContCu~
  |-- ContFe
  |-- ContMg
  |-- Cs

|-- Cu - the material variables file

  |-- Fe_HT
  |-- INPUTCONFIG.CFC
  |-- INPUTCONFIG.HC
  |-- INPUTCONFIG.ORT
--- Mg
|   |-- Mg_base
|   |-- README
|   |   |-- SEG_relax - output of the simulation (after the initial relaxation or later) that can be used latter as initial configuration file
|   |-- SEG_save - outputs of the simulation (after the initial relaxation or later) that can be used latter as initial configuration file
|   |-- SegCu - the initial dislocation configuration file (description of the initial dislocation configuration file format is given at the end)
|   |   |-- SegFe
|   |   |-- SegMg
|   |   |-- Segments - file describing the initial number, type and characteristics of dislocation segments
|   |   |   |-- Segments_48
|   |   |   |-- Segments_Ori
|   |   |-- Segments_old
|   |   |-- Zr
|   |   |   |-- b_plan - the input files needed to run polyphase simulations
|   |   |-- b_poly1 - the input files needed to run polyphase simulations
|   |   |-- b_spher - the input files needed to run polyphase simulations
|   |-- biphase
|   |   |-- carto
|   |   |   |-- README
|   |   |-- couleur.micmeg
|   |   |-- couleur.micmeg_12
|   |   |-- couleur.micmeg_g1
|   |   |-- couleur.micmeg_g2
|   |   |-- couleur.micmeg_old
|   |   |-- couleur.micmeg_ori
|   |   |-- disdi - the file containing tabulated values of anisotropic line tension as a function of dislocation character (generated with disdi, J. Douin's computation code)
|   |-- eulerangle - the input files needed to run polyphase simulations
|   |-- hosts.dd
|   |-- input.carto
|   |-- input.dd
|   |   |-- input.dd_ori - the input file used by the application "confinit" to generate an initial configuration
|   |-- inputconfinit
|   |-- inputconfinit_0
|   |-- inputconfinit_ori
|   |-- jonction
|   |-- matrice_rotation_BVD
|   |-- olivine
|   |-- particules - the input file needed to run DD simulation in a matrix including small (point) elastic inclusions
|   |-- random - a seed file used by the Fibonacci random generator
|-- shift_rotation - the input file used to impose a shift and/or a rotation at the boundaries of the simulation periodic cell
|-- tensapp - file containing a reference applied tensor used for complex loading
`-- tensapp_ori

|-- optiz - results from running the optimized executables
  |-- README
  |-- opt0 - results from running the compiler optimized test
    |-- opt0.output
    |-- opt0.run
  |-- opt1 - results from running the compiler optimized test
    |-- opt1.output
    |-- opt1.run
  ... 
  |-- opt12 - results from running the compiler optimized test
    |-- opt12.output
    |-- opt12.run
  |-- topt0 - results from running the compiler optimized test instrumented with TAU (using PAPI) for profiling
    |-- MULTI__CPU_TIME
    |   |-- profile.0.0.0
    |   |-- MULTI__GET_TIME_OF_DAY
    |   |   |-- profile.0.0.0
    |   |-- MULTI__LINUX_TIMERS
    |   |   |-- profile.0.0.0
    |   |-- topt0.output
    |   |-- topt0.ppk
    |   |-- topt0.run
    |-- topt1 - results from running the compiler optimized test instrumented with TAU (using PAPI) for profiling
      |-- MULTI__CPU_TIME
      |   |-- profile.0.0.0
      |   |-- MULTI__GET_TIME_OF_DAY
      |   |   |-- profile.0.0.0
      |   |-- MULTI__LINUX_TIMERS
      |   |   |-- profile.0.0.0
      |   |-- topt1.output
      |   |-- topt1.ppk
      |   |-- topt1.run
      ... 
  |-- topt11 - results from running the compiler optimized test instrumented with TAU (using PAPI) for profiling
    |-- MULTI__CPU_TIME
    |   |-- profile.0.0.0
    |   |-- MULTI__GET_TIME_OF_DAY
    |   |   |-- profile.0.0.0
    |   |-- MULTI__LINUX_TIMERS
    |   |   |-- profile.0.0.0
    |   |-- topt11.output
|-- out - directory where the output files of the simulations codes are written
|-- BVD.CFC - the set of reference vectors used in the simulation for a given crystal
|-- README
|-- bigsave.bin - binary file containing everything needed to restart a simulation if accidentally stopped
|-- film.bin - binary file where the coordinates of segments are periodically saved to buildup a trajectory file
|-- gamma - evolution of gamma for all existing slip systems
|-- gammap - evolution of the instantaneous gamma dot for all the slip systems
|-- manip.txt
|-- rau - evolution of rho, the dislocation density, for all the slip systems
|-- raujonc - evolution of the jonction density and number for all slip systems
|-- resul - GNU plotting script for plotting various simulation data (run 'gnuplot resul' to see the results)
|-- sigeps - output file containing the stress, strain and other information (a compagnon file to "stat")
|-- stat - where most of the global statistics of the simulation are written
|-- test_boucle.txt
|-- travapp - evolution of the applied mechanical work (presently do not trust those computations)
|-- travint - evolution of the internal mechanical work (presently do not trust those computations)
|-- prof - various profiling screenshots from Paraprof and Gprof
    |-- README
    |-- callgraph.eps
    |-- callgraph.png
    |-- callgraph.svg
    |-- callgraph_screenshot1.png
    |-- callgraph_screenshot2.png
    |-- callgraph_screenshot3.png
    |-- callgraph_statistics.doc
    |-- callgraphs.ppt
    |-- cp_relations.png
    |-- dd-graph-01-005.png
    |-- dd-graph-full.png
    |-- dd-graph.png
    |-- dd_prof
    |-- dd_prof_call_graph
    |-- gmon.out
    |-- gprof2dot.py - script to convert profiling output to a dot Graph (http://code.google.com/p/jrfonseca/wiki/Gprof2Dot); see prof/profi for examples on using gprof2dot
    |-- mm
    |-- mm.opt6.gprof.out
```
|    |-- profi
|    |   -- routines_barchart.png
|    |   -- script with examples on using gprof2dot to visualize the output of codes profiled with GPROF as a dot graph
|    |   `-- src - source code directory
|    |       |   `-- README
|    |       |       |   `-- CrystalRotation.pdf
|    |       |   `-- doc
|    |       |       |   |   |-- README
|    |       |       |   |   `-- Wait.pdf
|    |       |   `-- graph - source files for the interactive graphical version of the code (not used/tested in mM 1.0)
|    |       |       |   |   |   `-- Graphics - this is the directory where all the sources related to the graphical X interface are.
|    |       |       |   |   |       |   |-- 01Cmain.c
|    |       |       |   |   |       |   |-- 01Cmain_ori.c
|    |       |       |   |   |       |   |-- 02Ccristallo.c
|    |       |       |   |   |       |   |-- 03Cdraw.c
|    |       |       |   |   |       |   |-- 03Cdraw_ori.c
|    |       |       |   |   |       |   |-- 04Cfame.c
|    |       |       |   |   |       |   |-- 05Clook_seg.c
|    |       |       |   |   |       |   |-- 06Ccaide.c
|    |       |       |   |   |       |   |-- 07Cperspective.c
|    |       |       |   |   |       |   |-- 08Cpredraw.c
|    |       |       |   |   |       |   |-- 09Csegments.c
|    |       |       |   |   |       |   |-- 10Csegout.c
|    |       |       |   |   |       |   |-- 11Ctrace.c
|    |       |       |   |   |       |   |-- 12Cwindow_init.c
|    |       |       |   |   |       |   |-- 13C_post_image_ps.c
|    |       |       |   |   |       |   |-- 14_bricamat.c
|    |       |       |   |   |       |   |-- BlackBG
|    |       |       |   |   |       |   |-- BlackBG_dec
|    |       |       |   |   |       |   |-- BlackBG_mac
|    |       |       |   |   |       |   |-- BlackBG_madmax
|    |       |       |   |   |       |   |-- BlackBG_sgi
|    |       |       |   |   |       |   |-- WhiteBG
|    |       |       |   |   |       |   |-- cooleur.h
|    |       |       |   |   |       |   |-- couleur.fuj
|    |       |       |   |   |       |   |-- couleur.mad
|    |       |       |   |   |       |   |-- couleur.madison
|    |       |       |   |   |       |   |-- extercool.h
|    |       |       |   |   |       |   |-- graph.c
|    |       |       |   |   |       |   |-- graph_ori.c
|    |       |       |   |   |       |   `-- varcool.h
|    |       |   `-- README
|    |   |-- lib
|    |       |   |-- CRISTALLOGRAPHIE - empty directory
|    |       |   `-- outils - directory as a kind of toolbox where all the Fortran sources connected (but not part of) to the simulation are stored
|    |       |       |   |-- AlphaAfact.ctl11
|    |       |       |   |   |-- Comp_Dconf
|    |       |       |   |   |-- ConstrucPoly2.f90
|    |       |       |   |   |-- DConf.f90
|    |       |       |   |   |-- Diophan - Program ecrit par Ronan Madec pour resoudre les equations diophantines solution du probleme des dimensions du plus petit libre parcourt
```
d'une

|-- Diophan.IN  - dislocation (dans un plan de glissement
donne) avec des conditions aux limites
periodiques en DD

|-- Diophan.f90

|-- README

|-- Epstime.ctl1

|-- GamaSys.ctl1

|-- GamaSysP.ctl1

|-- Hlj

|-- ImpIt.ctl1

|-- ImpIt.tcl

|-- MergeManip

|-- PlotItArch

|-- PolyFront  - Programme fait par Ronan Madec pour
construire géométriquement un pavage de
polygones définissant les dimensions du plan de
glissement des

|-- PolyFront.ctl1  - dislocations dans une boîte de DD avec des
conditions aux limites périodiques

|-- PolyFront.f90

|-- README

|-- PrepIncF90.c

|-- PrepProcF90.c

|-- README

|-- RoEps.ctl1

|-- RoSys.ctl1

|-- SigEps.ctl1

|-- SigRo.ctl1

|-- Sigtime.ctl1

|-- Teps.ctl1

|-- TsigInt.ctl1

|-- airecoin.ctl1

|-- airevcm.ctl1

|-- airevis.ctl1

|-- camera.f90

|-- camera.f90_ori

|-- compose_aij

|-- Mh.f90

|-- madhocSAR.in

|-- out

|-- CVS

|-- Entries

|-- Repository

|-- Root

|-- convers

|-- curves.ctl2

|-- depou_aij

|-- coeff.f90

|-- in_depou

|-- out

|-- diaphantine.f90

|-- epspoit.ctl1

|-- fin.ctl1

|-- histo.f90

|-- hlj.f90

|-- load_manip
2. Profiling callgraphs of the serial and OpenMP-based parallel microMegas execution

We measured the performance of microMegas using TAU, a state-of-the-art portable profiling and tracing package. TAU allowed easy and customizable instrumentation of microMegas. We ran microMegas instrumented with TAU over 100,000 time steps. The instrumented version of the original code was executed on a single core of the machine. TAU enabled us to collect profiling information at various levels: outer loops, routine level and memory leaks. Using TAU, we were also able to collect
traces of the serial execution of the instrumented code. Being able to obtain these measurements was crucial to understanding the behavior of the simulation code and to identifying the performance bottlenecks of the application.

We collected profiling information from a serial run of the application over 100,000 time steps. Profiling yields timing information summed over all invocations of a function. The execution callgraph of the original application code is illustrated in Figure 8. The callgraph was illustrated with Paraprof, TAU’s 3-D profiling data visualization tool. The red box indicates the most time consuming part of the application, i.e., the ‘hot spot’, namely subroutine SIGMA_INT_CP of module ELASTI. The green box indicates the second most time consuming part of the application, i.e., the ‘warm spot’, namely subroutine UPDATE in module CONTACT. Blue color indicates subroutines that take less time than those in the green or red color boxes, and are called ‘cold spots’. Based on the profiling information obtained with TAU, we were able to sort the modules of MicroMegas in decreasing order of importance, with respect to the percentage of the total serial time attributed to each module.

The profiling data for the simulations over 100,000 time steps indicated that the most important module was ELASTI, and the most important subroutine of this module was SIGMA_INT_CP, called by subroutine FORCE. Similarly, the second most important module was CONTACT with the most important subroutine of this module being UPDATE. Subroutine ELASTI::SIGMA_INT_CP accounts for 72.675% of the serial run time, while CONTACT::UPDATE is responsible for 26.885% of the serial run time, respectively.

Subroutine SIGMA_INT_CP contains a mix of nested DO loops and a series of consecutive DO loops that iterate over the number of boxes, the number of segments in each 3-D box and over each segment in a box. These nested DO loops calculate the short-range interaction force between the segments, and, therefore, the iterations of these loops are independent of each other. We parallelized the outermost loop, which iterates over the number of boxes of the simulation domain, by distributing its iterations among the processing cores in a load balanced fashion.

Analogous to Figure 8, the red box in the right side of Figure 9 indicates the most time consuming subroutine of the application. In this case, however, the red box indicates the subroutine UPDATE, which in the serial code was indicated in a green box. Also, subroutine SIGMA_INT_CP is no longer indicated in a red box but in a blue box, which signifies that it is no longer the most time consuming part of the application.

Regard the work distribution methods, we chose static scheduling for distributing the iterations of the outermost loop in SIGMA_INT_CP, which iterates over the number of boxes. The chunk size was equal to the number of boxes divided by the number of threads. Then, we used selected another chunk size equal to half the original chunk size, that is, the number of boxes divides by twice the number of threads. The results presented in this paper were obtained with static scheduling and the second choice of chunk size. Due to the fact that all cores are homogeneous and due to the dedicated test system, the master thread executes SIGMA_INT_CP in a perfectly load balanced fashion simultaneously with the worker threads.
Figure 8  Serial microMegas: Visualization of serial execution callgraph for 100,000 time steps. Important subroutines are highlighted in colors, where red denotes a 'hot spot', green a 'warm spot' and blue a 'cold spot'.

Figure 9 Parallel microMegas: callgraphs visualization for parallel execution on 4 cores: master thread callgraph (right) and the worker threads callgraph (left) for 100,000 time steps.