Charon: Basic Introduction Training

Lawrence C Musson, Mihai Negoita, Gary Hennigan, Xujiao Gao, and Andy Huang

Sandia National Laboratories

SAND2022-8987 PE

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

What does Charon do?

- Drift-Diffusion PDE solver for modeling charge carrier flow

Electric Potential

$$\begin{cases} \nabla \cdot \left(\epsilon \vec{\mathbf{E}}\right) = q\,(p - n + C) \\ \vec{\mathbf{E}} = -\nabla V \end{cases}$$

$$\left.\begin{aligned} \vec{\mathbf{J}}_n &= q\left(n\mu_n \vec{\mathbf{E}} + D_n \nabla n\right) \\ \vec{\mathbf{J}}_p &= q\left(p\mu_p \vec{\mathbf{E}} - D_p \nabla p\right) \end{aligned}\right\}$$

Carrier Transport Equations

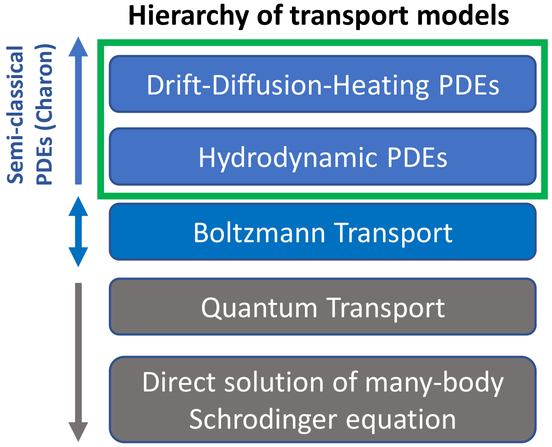
$$\left.\begin{aligned} \nabla \cdot \vec{\mathbf{J}}_n - qR &= q\frac{\partial n}{\partial t} \\ -\nabla \cdot \vec{\mathbf{J}}_p - qR &= q\frac{\partial p}{\partial t} \end{aligned}\right\}$$

Conservation

$$\nabla \cdot (\kappa \nabla T_L) + H = \rho c \frac{\partial T_L}{\partial T} \quad \Big\}$$

Lattice Heating

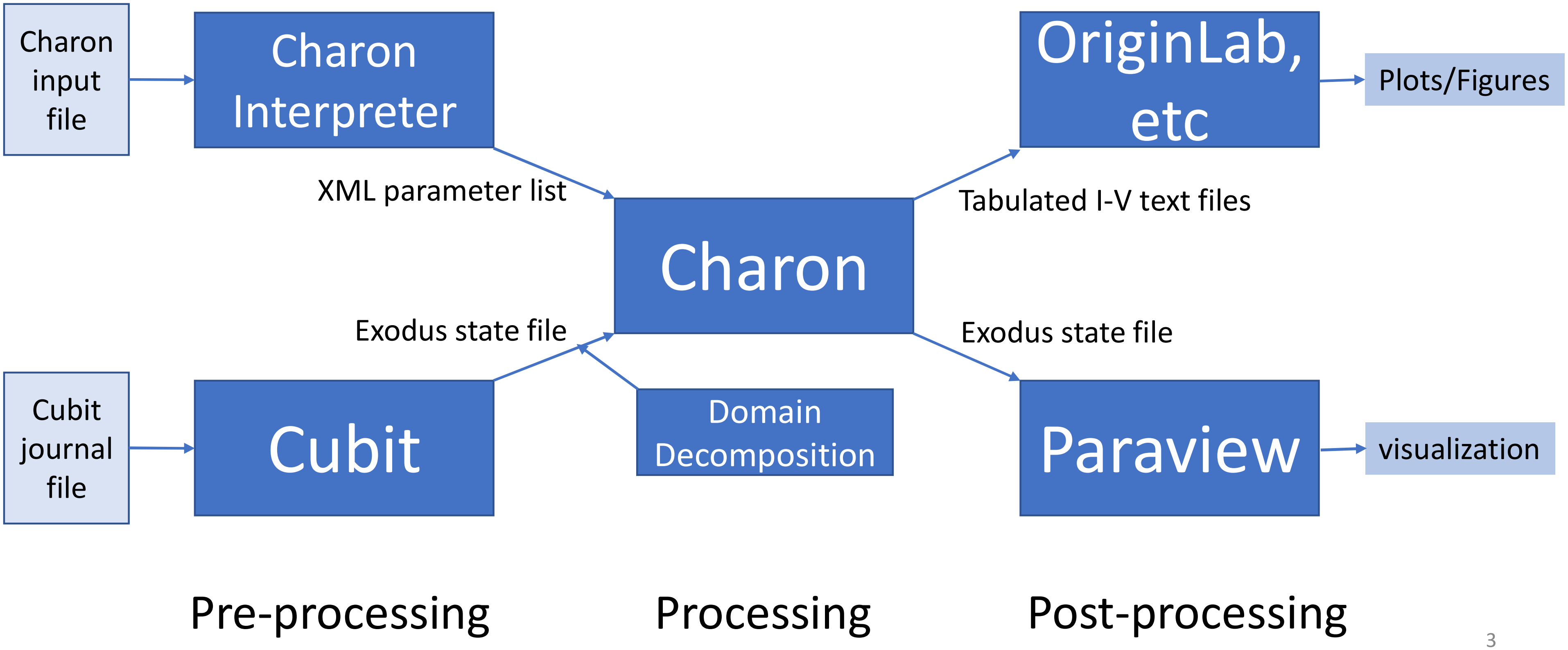
TCAD code for modeling semiconductor performance including ionizing radiation and displacement damage as a result of radiation



Hierarchy of transport models

Semi-classical PDEs (Charon)
Drift-Diffusion-Heating PDEs
Hydrodynamic PDEs
Boltzmann Transport
Quantum Transport
Direct solution of many-body Schrodinger equation

Diagram of a Simulation



Charon input file

Charon Interpreter

XML parameter list

Cubit journal file

Cubit

Exodus state file

Domain Decomposition

Charon

OriginLab, etc

Plots/Figures

Tabulated I-V text files

Exodus state file

Paraview

visualization

Pre-processing

Processing

Post-processing

Charon Files

- Input files
  - Contain state file names and a full parametric description of the simulation
- State files
  - Input
    - Meshing and initial guesses for a simulation
  - Output
    - Results of a simulation
- Various text files
  - Input
    - Assorted radiation data: pulse information, etc
  - Output
    - Tabulated I-V or I-<parameter> data from a transient or parameter sweep simulation

Charon Environment & Tutorials

- The environment needs to be set up for Charon to execute
  - Environment variables: PATH, etc
  - Modules: various TPLs
- On CEE systems
  - bash: source /projects/charon/install/BOD/setup-cee.cde.intel.env.sh
- On HPC systems (skybridge, chama, attaway, eclipse)
  - bash: source /projects/charon/install/attaway.jenkins/setup.cde.intel.env.sh
- Copy the tutorial problems to your personal account
  - cp -R /projects/charon/Training/BasicIntro .
  - Three tutorial directories
  - Charon user manual (pdf)
  - Training slides (pdf)

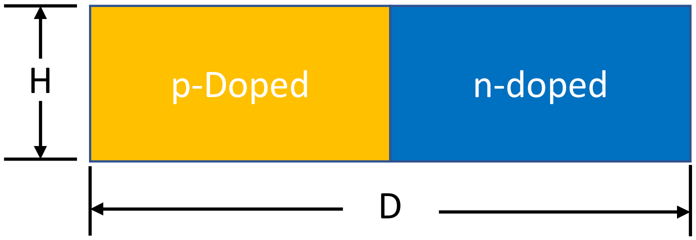
Charon Interpreter (chirp)

- Charon Interpreter
  - Python driven front end to Charon
  - Simple, readable syntax
  - Straightforward formatting
  - Available syntax reference, searchable through “less”
  - Maps input syntax to Teuchos parameter list (xml) that fully configures a simulation
  - Can execute Charon in parallel or serial
  - (in the future) Will perform domain decomposition for parallel simulations

CharonInterpreter Help (exercise)

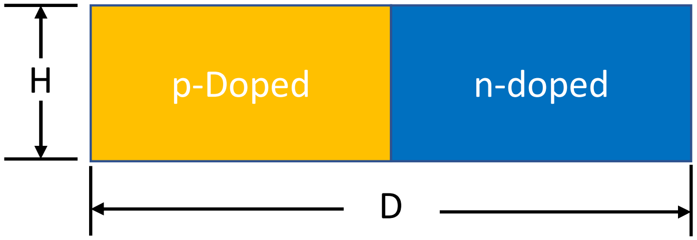
```
lcmusso@ascic0186 [~] % chirp --help
usage: chirp [-h] [-i INPUT] [-r] [--silent] [--no_current] [--np NP] [-s]
             [-S] [-d] [-R RESIZE_FROM_NPROCS] [-v VERBOSITY]
             [-p PATH_TO_CHARON] [--cleanTextData] [--mpitile] [-l LABEL]
             [--nolog] [--mpipath MPIPATH] [--mpiexec MPIEXEC]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Specify the interpreter input file
  -r, --run             Execute Charon
  --silent              Supresses screen output. Screen output will be stored
                        in log.
  --no_current          Disable text output of current
  --np NP               Specify the number of processors to use in the Charon
                        run.
  -s, --syntax          Get help for syntax of interpreter. Append "| less -i"
                        to scrolling and search.
  -S, --longsyntax      Get expanded help for syntax of interpreter. Append "|
                        less -i" to scrolling and search.
  -d, --decomp          Decompose the state file for parallel execution. Will
                        abort if decomposition of same size already exists.
  -R RESIZE_FROM_NPROCS, --resize_from_nprocs RESIZE_FROM_NPROCS
                        Resizes the decomposition from nprocs domains to np
                        domains. Will abort if decomposition of same size is
                        detected.
  -v VERBOSITY, --verbosity VERBOSITY
                        Specify verbose output of the interpreter ranging from
                        0 to 25
  -p PATH_TO_CHARON, --path_to_charon PATH_TO_CHARON
                        Specify path to Charon exectutable
  --cleanTextData       This option will remove sweep and transient text data
                        files prior to execution.
  --mpitile             uses Dakota's mpitile on large computing systems with
                        high concurrency.
  -l LABEL, --label LABEL
                        Specify an optional label for the run.
  --nolog               Opts out of logging the run.
  --mpipath MPIPATH     Specify path to an mpi run command. This path may also
                        be set through the MPIRUN_COMMAND_PATH environment
                        variable.
  --mpiexec MPIEXEC     Specify an mpi executable name. This name may also be
                        set through the MPIRUN_COMMAND environment variable.
```

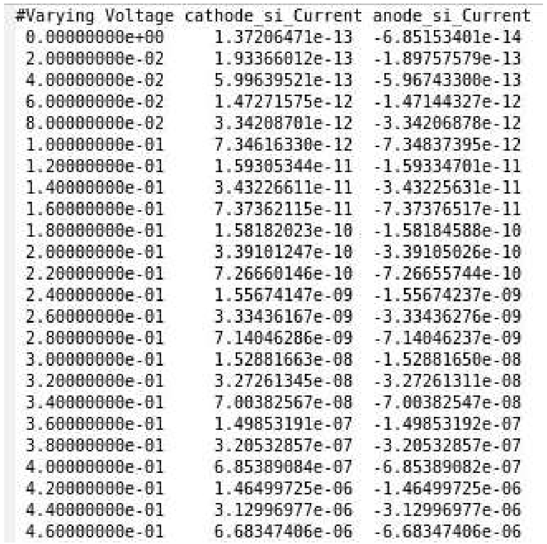
Exercise – Run a diode simulation



- In the tutorial #1 directory
  - chirp -i pndiode.iv.inp --run
  - Open paraview
    - Load pndiode.iv.exo
    - View potential, electron density, hole density
  - Plot the IV curve
    - Open the currents-loca.dat file in an editor and place a # on the first line in front of headers
    - gnuplot
      - gnuplot> set logscale y
      - gnuplot> plot ‘currents-loca.dat’

Diode exercise—What are the numbers?



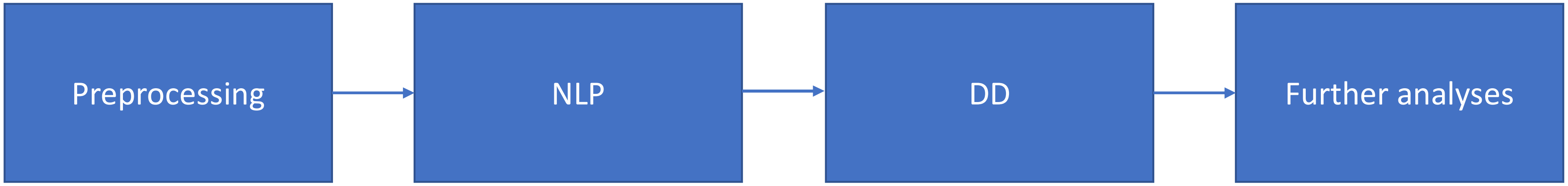


```
#Varying Voltage cathode_si_Current anode_si_Current
  0.00000000e+00      1.37206471e-13  -6.85153401e-14
  2.00000000e-02      1.93366012e-13  -1.89757579e-13
  4.00000000e-02      5.99639521e-13  -5.96743300e-13
  6.00000000e-02      1.47271575e-12  -1.47144327e-12
  8.00000000e-02      3.34208701e-12  -3.34206878e-12
  1.00000000e-01      7.34616330e-12  -7.34837395e-12
  1.20000000e-01      1.59305344e-11  -1.59334701e-11
  1.40000000e-01      3.43226611e-11  -3.43225631e-11
  1.60000000e-01      7.37362115e-11  -7.37376517e-11
  1.80000000e-01      1.58182023e-10  -1.58184588e-10
  2.00000000e-01      3.39101247e-10  -3.39105026e-10
  2.20000000e-01      7.26660146e-10  -7.26655744e-10
  2.40000000e-01      1.55674147e-09  -1.55674237e-09
  2.60000000e-01      3.33436167e-09  -3.33436276e-09
  2.80000000e-01      7.14046286e-09  -7.14046237e-09
  3.00000000e-01      1.52881663e-08  -1.52881650e-08
  3.20000000e-01      3.27261345e-08  -3.27261311e-08
  3.40000000e-01      7.00382567e-08  -7.00382547e-08
  3.60000000e-01      1.49853191e-07  -1.49853192e-07
  3.80000000e-01      3.20532857e-07  -3.20532857e-07
  4.00000000e-01      6.85389084e-07  -6.85389082e-07
  4.20000000e-01      1.46499725e-06  -1.46499725e-06
  4.40000000e-01      3.12996977e-06  -3.12996977e-06
  4.60000000e-01      6.68347406e-06  -6.68347406e-06
```

- Currents-loca.dat has I-V data
  - What are the currents?
  - Charon is 2D & 3D only
    - 2D: The current values will have units of Amps/cm width
    - 3D: The current values will have units of Amps
  - This simulation is set up as a quasi-1D problem
    - It is technically 2D, but all the action is in a single direction—H is an arbitrary length
    - Current is contactArea*current/H

Path to a Full Drift-Diffusion Simulation

- Two steps are required to produce a drift-diffusion (DD) simulation
- Usually, it isn’t possible to go from a null state to a DD state in a single pass
- A nonlinear Poisson (NLP) solution is computed first for the electrostatic potential
- This is used as the initial guess for a DD solution



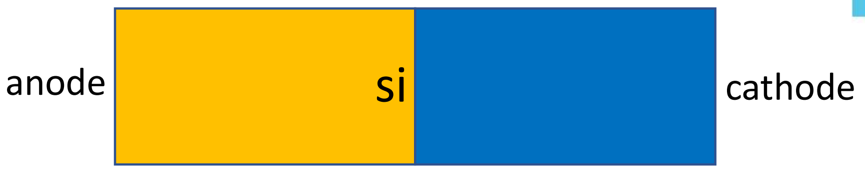
Cubit—Solid Model & Meshing

- Cubit is the Sandia tool for creating solid models and meshes
  - Not much covered in Charon quick start training
    - Introductory training taught by Cubit team should be sufficient
  - Most TCAD solid models are rectangular
  - Implicit surfaces (junctions) make meshing difficult, sometimes impossible with Cubit
  - Charon’s pyMesh tool is available, but still in an early state
    - Automatically refines meshes around junctions.
    - Can easily be configured to mesh around lines, planes, points…

Meshing the diode

```
# Create a three-dimensional volume. The two-dimensional diode will be
# created on a surface of this three-dimensional volume. By default
# Charon assumes the dimensions are in microns
create brick x 1.0 y 0.5 z 0.1

# This makes the coordinates of the resulting mesh all positive. This
# isn't required but can be useful for post-processing
move vertex 4 location 0 0 0

# These will be the contacts, anode and cathode. The names are used in
# the input file to distinguish them
sideset 1 curve 3
sideset 1 name "anode"

sideset 2 curve 1
sideset 2 name "cathode"

# "blocks" are typically regions of different materials or distinct
# regions of the device. For this simple problem we only have one
# region
block 1 surface 1
block 1 name "si"

# Quads (bricks) or triangles (tetrahedra) are the
# preferred element type for Charon simulations.
block 1 element type quad4


# The interval specifications set how dense or coarse the
# discretization is

## Long side
curve 2 4 interval 100

## Short side (contacts)
curve 3 1 interval 1

# Generate, or "mesh", the problem geometry
mesh surface 1

# Create the output "exodus" file, or overwrite it if it already
# exists.
export mesh "pndiode.exo" dimension 2 overwrite
```

Solid model (use microns)



Always use lower case for names

Naming Contacts

Naming geometric regions

Meshing

Exporting mesh

Charon Input File

```
import state file pndiode.exo

start output parameters
  output state file pndiode.dd.iv.exo
end output parameters

start physics block Semiconductor
   geometry block is si
   standard discretization type is drift diffusion gfem
   material model is siliconParameter
end physics block

start material block siliconParameter
   material name is Silicon
   relative permittivity = 11.9
   start step junction doping
      acceptor concentration =1.0e16
      donor concentration =1.0e16
      junction location = 0.5
      dopant order is PN
      direction is x
      end step junction doping
end material block siliconParameter

initial conditions for ELECTRIC_POTENTIAL in si is exodus file
initial conditions for ELECTRON_DENSITY in si is equilibrium density
initial conditions for HOLE_DENSITY in si is equilibrium density

bc is ohmic for cathode on si fixed at 0.0
bc is ohmic for anode on si swept from 0.0 to 1.0

start sweep options
   initial step size = 0.02
   minimum step size = 0.02
   maximum step size = 0.02
end sweep options

Tpetra is on
start solver block
  start tpetra block
    problem type is steady state
    verbosity level is high
  end
end solver block
```

- The highlights indicate where information supplied in the geometry/meshing phase of preprocessing ties into the Charon input
- The imported state file contains the mesh and a state—if there is one.
- The remainders are names tied to regions or boundaries of the device as named during the meshing phase.

charonInterpreter (chirp) Syntax Reference

- The interpreter can supply a syntax reference on command
  - charonInterpreter.py --syntax, or charonInterpreter -s
    - Provides an abbreviated help
  - charonInterpreter.py --longsyntax, or charonInterpreter -S
    - Provides a longer help
- Syntax help can be piped through less to make it scrollable and searchable
  - charonInterpreter.py -S | less -i
  - To search: /<search term>
- Exercise: Search the syntax help for “state file”

charonInterpreter Syntax Reference

{} indicates user-supplied entries—these are ALWAYS case sensitive

Import State File {filename} [at Index {index}]

[] indicates optional entries to the command

charonInterpreter Syntax Reference

BC is ohmic for {sidesetID} on {geometryBlock} [fixed at {potential} [swept from {potential1} to {potential2}]]

- Sometimes multiple options are available
  - Nested in the reference by [option1 [option2]]
  - Voltage on a contact can be fixed at a value
  - Voltage can be swept from one value to another
    - Such as an IV sweep
  - In this instance, at least one of the options must be selected

Input File Structure—Essential Elements

- Import/Export
  - Specifies state files to import for geometry, states
  - Specifies state files and other variables for export in the state file
- Physics Blocks
  - Sets the equations to be solved
  - Toggles various physics on & off
  - Ties physics to geometry and Material parameters in the Material Block
- Material Blocks
  - Defines material properties
- Initial Conditions
  - Specify what to use for initial values for variables (electrons, holes, potential, temperature)
- Boundary Conditions
  - Contact potentials, boundary temperatures
- Solver Specifications
  - Tolerances, preconditioner and solver methods

Input File Structure—State file import & Output

- Charon state files contain geometric information and solution
  - Exodus formatted
  - Geometric information includes the mesh
  - State can be null
    - File contains only a mesh
  - There can be multiple states
    - Output from transient or parameter sweep
- File names are specified in the input file for input and output state files

Input File Structure—State file import & Output

```
import state file pndiode.exo
```

- Imports the state file
  - Will contain at least the mesh and geometry
  - Might contain a state to use as an initial guess

```
start output parameters
  output state file pndiode.nlp.exo
end output parameters
```

- Specifies the output state file name
  - Will contain the mesh, geometry and the state just calculated
  - Can contain other directives for output of specific variables or tabulated data from sweeps (Exercise on this later)

Input File Structure—Physics Block

```
start physics block Semiconductor
   geometry block is si
   standard discretization type is drift diffusion gfem
   material model is siliconParameter
end physics block
```

- Physics block contains information about a region
  - The block must have its own unique name (semiconductor)
  - The corresponding region name from geometry
  - The equations to be solved
  - The name of the associated material block

Input File Structure—Material Block

```
start material block siliconParameter
   material name is Silicon
   relative permittivity = 11.9
   start step junction doping
      acceptor concentration =1.0e16
      donor concentration =1.0e16
      junction location = 0.5
      dopant order is PN
      direction is x
   end step junction doping
end material block siliconParameter
```

- Material block contains material property information
  - Must have its own unique name
  - Specifies a material name for parameters
  - Specifies the doping
  - Potentially numerous other material properties (mobility) if not defined internally or a different value from default is desired

Input File Structure—Boundary Conditions

```
bc is ohmic for anode on si fixed at 0
bc is ohmic for cathode on si fixed at 0
```

- Boundary conditions
  - Specifies the voltage on the contacts or heat flux/temperature at a boundary

Input File Structure—Initial Conditions

```
Initial conditions for ELECTRIC_POTENTIAL in si equilibrium potential
```

- Specifies initial conditions to the solve
  - Could be uninitialized
  - Could be a model—an estimation of the solution at equilibrium
  - Could be “exodus file” if read from a previous solution

Input File Structure—Solver Block

```
Tpetra is on
start solver block
  start tpetra block
    problem type is steady state
    verbosity level is high
  end
end solver block
```

- Solver block specifies solver parameters
  - Specifies which solvers and preconditioners to use
  - Specifies tolerances for nonlinear solves

Exercise—Run a Diode Start to Finish

- From the Tutorial #2 directory, run the full sequence of tools to produce the same diode data as the first exercise
  - Generate the mesh
  - Decompose the mesh for a parallel run
  - Run the nonlinear Poisson solve
  - Run the drift-diffusion solve with anode sweep

Exercise—Generate and Decompose the Mesh

- cubit pndiode.jou
    - Generates the mesh
    - Use cubit to examine (it’s not exciting)

- decomp --processors 4 pndiode.exo
    - Decompose the mesh with the decomp tool

Exercise—Run the nonlinear Poisson Solve

- chirp -i pndiode.nlp.inp --np 4 --run
  - Use paraview to examine the potential field
  - Can add --decomp to this if the decomp step was skipped in the previous setp

Exercise—Run the Drift-Diffusion Solve

- chirp -i pndiode.iv.inp --np 4 --run
    - Note the initial conditions in the input file
    - Examine the potential and carrier densities over the sweep in paraview
    - Compare the iv data to the first diode exercise
    - Use paraview to examine the potential / depletion region

Exercise—Extra Credit Problem

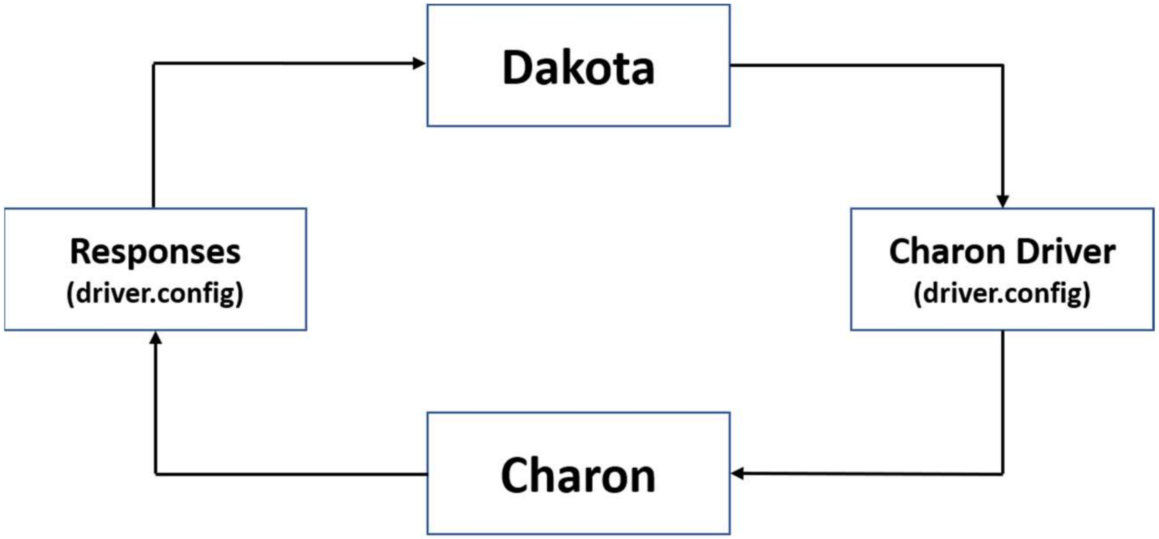
- Use the interpreter reference to:
  - Name the file for iv output srh-currents.dat
    - Add to the output parameters block
  - Add srh recombination
    - Toggle on srh recombination in the physics block
    - Add srh lifetimes to the material block (fixed 1e-9 for both electrons and holes)
- Need only run the iv sweep
  - All else including nonlinear Poisson solve remains unchanged
- Compare the iv sweep with and without recombination
    - Open the currents-loca.dat file in an editor and place a # on the first line in front of headers
    - gnuplot
      - gnuplot> set logscale y
      - gnuplot> plot ‘currents-loca.dat’,’srh-currents.dat’

Exercise—
Extra Credit Problem

```
import state file pndiode.exo

start output parameters
  output state file pndiode.dd.iv.exo
  output tabulated parameter currents to srh-currents.dat
end output parameters

start physics block Semiconductor
   geometry block is si
   standard discretization type is drift diffusion gfem
   material model is siliconParameter
   srh recombination is on
end physics block

start material block siliconParameter
   material name is Silicon
   relative permittivity = 11.9
   start step junction doping
      acceptor concentration =1.0e16
      donor concentration =1.0e16
      junction location = 0.5
      dopant order is PN
      direction is x
    end step junction doping

    start Carrier Lifetime Block
      electron lifetime is constant = 1e-9
      hole lifetime is constant = 1e-9
    end Carrier Lifetime Block

end material block siliconParameter

initial conditions for ELECTRIC_POTENTIAL in si is exodus file
initial conditions for ELECTRON_DENSITY in si is equilibrium density
initial conditions for HOLE_DENSITY in si is equilibrium density

bc is ohmic for cathode on si fixed at 0.0
bc is ohmic for anode on si swept from 0.0 to 1.0

start sweep options
   initial step size = 0.02
   minimum step size = 0.02
   maximum step size = 0.02
end sweep options

Tpetra is on
start solver block
  start tpetra block
    problem type is steady state
    verbosity level is high
  end
end solver block
```

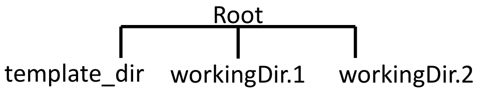
Dakota usage with Charon

- Scripts that ease the use of Dakota with Charon have been created and are presently in a beta stage of release.
- The basic flow structure is:



Dakota usage with Charon

- A Dakota input file must be supplied. The driver in that input file must always be set to charonDriver.py and a template directory specified. Individual executions are optionally stored in the work_directory.



```
interface
    id_interface = 'I1'
  analysis_drivers = 'charonDriver.py'
    fork
    asynchronous
    evaluation_concurrency 2
    work_directory named 'workingDir'
    directory_save
    link_files = 'template_dir/*'
```

Dakota usage with Charon

- The charonDriver.py script can be invoked directly to get limited help:

```
charonDriver help:
for a list of available responses (QOIs), charonDriver.py --listresponses
For a description of how to use a particular response: charonDriver.py --help <response>
```

Dakota usage with Charon

- charonDriver.py --listresponses

```
Response  0  is  thresholdVoltage
Response  1  is  currentAtVoltage
Response  2  is  betaGain
Response  3  is  IVCurve
Response  4  is  compositeFunction
```

Dakota usage with Charon

- charonDriver.py --help currentAtVoltage

```
reponse currentAtVoltage argument1=value1 argument2=value2...

This response returns the current at a specified voltage from a voltage sweep simulation.

filename -- Name of the file where the currents and voltages are expect to be found.


voltage -- Set a numerical value of the voltage at which the current is calculated.


 target -- The target is used for calibration.  It is the expected value of the current at the specified voltage.  When the target is
specified, a residual of the expected and calculated values is returned.


weighted -- When the residual is calculated with a target value, setting this to yes or no will opt into weighting the residual by the
expected value.  The default is NOT to weight the residual.


responsename -- The default name of the response is currentAtVoltage.  However, if different currents are to be calculated such as
weighted and unweighted or at multiple voltages values, a unique name must be given to each reponse.


voltColumn -- Specify a custom column index for the volts in the I-V data file.  Note that the index starts from 0.  I SAY AGAIN!!  THE
FIRST COLUMN OF THE DATA IS INDEX 0. THE SECOND COLUMN IS 1.  And so on.


currentColumn -- Specify a custom column index for the current in the I-V data file.  Note that the index starts from 0.  I SAY AGAIN!!
THE FIRST COLUMN OF THE DATA IS INDEX 0. THE SECOND COLUMN IS 1.  And so on.
```

Dakota usage with Charon

- An example of diode doping calibration is supplied in the Tutorial4 directory
- The template directory must contain a file named driver.config

executeMethods—can be any in number, but the apps are limited to charon, cubit, pyMesh

```
executeMethods   app=charon   template=pndiode.nlp.template
executeMethods   app=charon   template=pndiode.iv.template

executeProcs     4

response    IVCurve  filename=currents-loca.dat voltColumn=0 \
   currentColumn=2 coordinates=IVCurveA.cords \
   targetFile=IVCurveA.dat  responseName=IVCurveA
```

executeProcs—number of processors used in a single evaluation

response—can be any in number, but the responses are limited to those available in the scripts: IVCurve, compositeFunction, currentAtVoltage, thresholdVoltage.
If multiple responses are used of identical type, they must each be given a unique name, e.g. IVCurveA, IVCurveB…

Dakota usage with Charon

- The template files MUST contain parameters that Dakota will modify:

```
start step junction doping
      acceptor concentration = {dopingCoefficient*1e16}
      donor concentration = {dopingCoefficient*1e16}
      junction location = 0.5
      dopant order is PN
      direction is x
   end step junction doping
```

From the template file

The parameter names must match!!

```
variables
     id_variables = 'V1'
  continuous_design 1
    upper_bounds   =  10
    lower_bounds   =  0.01
    initial_point  =  0.1
    descriptors       'dopingCoefficient'
```

From the Dakota input file

Dakota usage with Charon

- Responses in the Dakota input file MUST match those in the driver.config!!

```
response  IVCurve filename=currents-loca.dat voltColumn=0 currentColumn=2 \
    coordinates=IVCurveA.coords targetFile=IVCurveA.dat  \
    responseName=IVCurveA
```

From the driver.config file

The response names must match!!

From the Dakota input file

```
responses,
     id_responses = 'R2'
  calibration_terms = 1
    response_descriptors      = 'IVCurveA'
  numerical_gradients
  method_source dakota
  interval_type forward
  fd_step_size =        0.0001
```

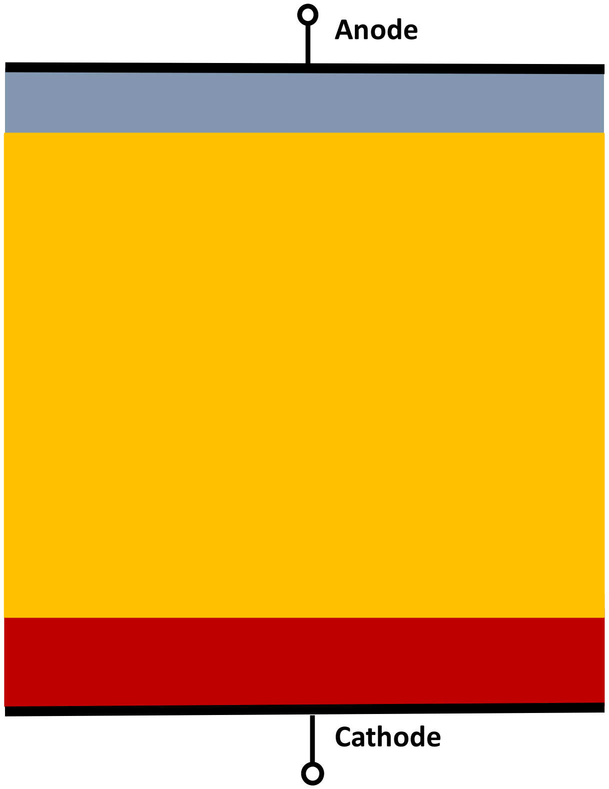
Dakota usage with Charon

- A target IV curve was generated with **`#{dopingCoefficient=1.0}`**
- The test is to see if this coefficient can be recovered in calibration
- Try it now! dakota -i dakota-hybrid.in

```
%eval_id  interface dopingCoefficient IVCurveA_1
1         I1        5.005          3.237884086
2         I1        8.335          7.360924143
3         I1        1.675          0.1775380634
4         I1        2.785          0.9413572675
5         I1        0.565          0.1139817785
6         I1        6.115          4.563392539
7         I1        3.895          2.007893428
8         I1        0.935          0.002133588337
9         I1        0.195          0.4966495347
10        I1        9.445          8.79297729
11        I1        7.225          5.945763053
12        I1        2.045          0.3839108667
13        I1        1.305          0.040805769
14        I1        1.058333333    0.001634491354
15        I1        0.8116666667   0.01890317923
16        I1        3.155          1.272426658
17        I1        2.415          0.6424202468
18        I1        1.428333333    0.07720425106
19        I1        1.181666667    0.01512785733
20        I1        1.099444444    0.004675086903
21        I1        1.017222222    0.000144811942
%eval_id  interface dopingCoefficient IVCurveA_1
22        I1        1.017222222    0.000144811942
23        I1        1.000463732    1.057037657e-07
24        I1        0.9999951517   1.155613614e-11
```

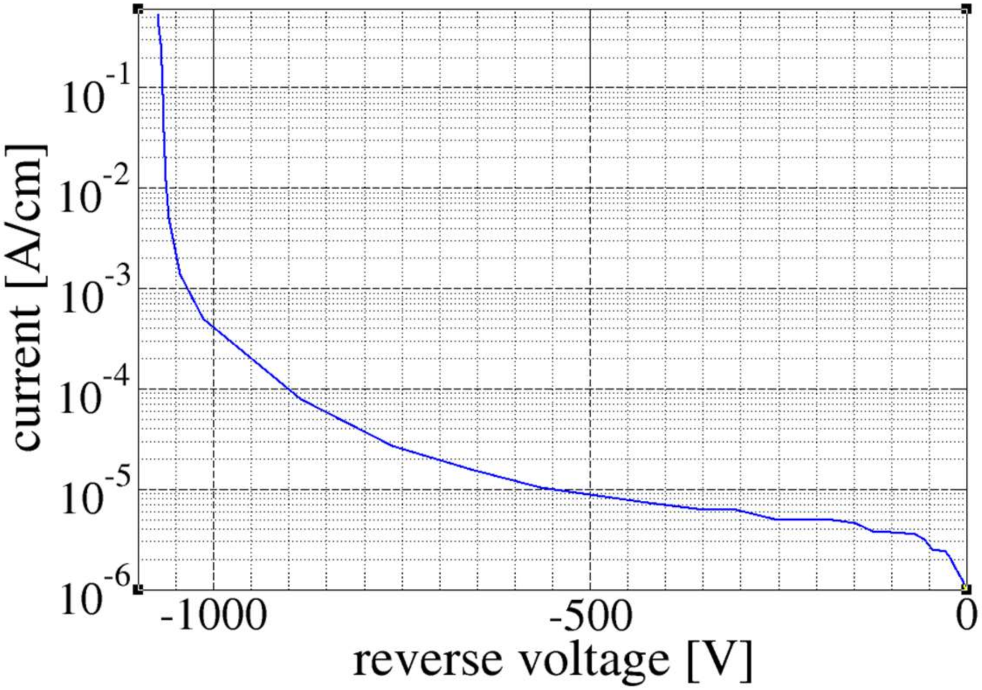
```
********************************************************
             OPT++ TERMINATION CRITERION
             Return Code                3
           SUCCESS - optpp_q_newton converged to a solution
Algorithm converged - Norm of gradient is less than gradient tolerance
********************************************************
<<<<< Function evaluation summary (I1): 8 total (5 new, 3 duplicate)
<<<<< Best parameters          =
                      9.9999515175e-01 dopingCoefficient
<<<<< Best residual terms =
                      1.5520000000e-14
                      8.7800000002e-14
                      5.8000000000e-13
                      3.7769999999e-12
                      2.4620000000e-11
                      1.5910000000e-10
                      1.0040000000e-09
                      5.7399999998e-09
                      2.2020000000e-08
                      2.6200000003e-08
                     -5.7200000000e-08
                     -2.7400000002e-07
                     -6.3700000000e-07
                     -1.1430000000e-06
                     -1.7890000000e-06
                     -2.5619999999e-06
<<<<< Best residual norm =  3.3994317380e-06; 0.5 * norm^2 =  5.7780680707e-12
<<<<< Best data captured at function evaluation 27


<<<<< Iterator optpp_q_newton completed.

<<<<< Iterator hybrid completed.
<<<<< Environment execution completed.
DAKOTA execution time in seconds:
  Total CPU        =        0.2 [parent =   0.203672, child =  -0.003672]
  Total wall clock =    113.539
```

GaN PIN diode structure and models



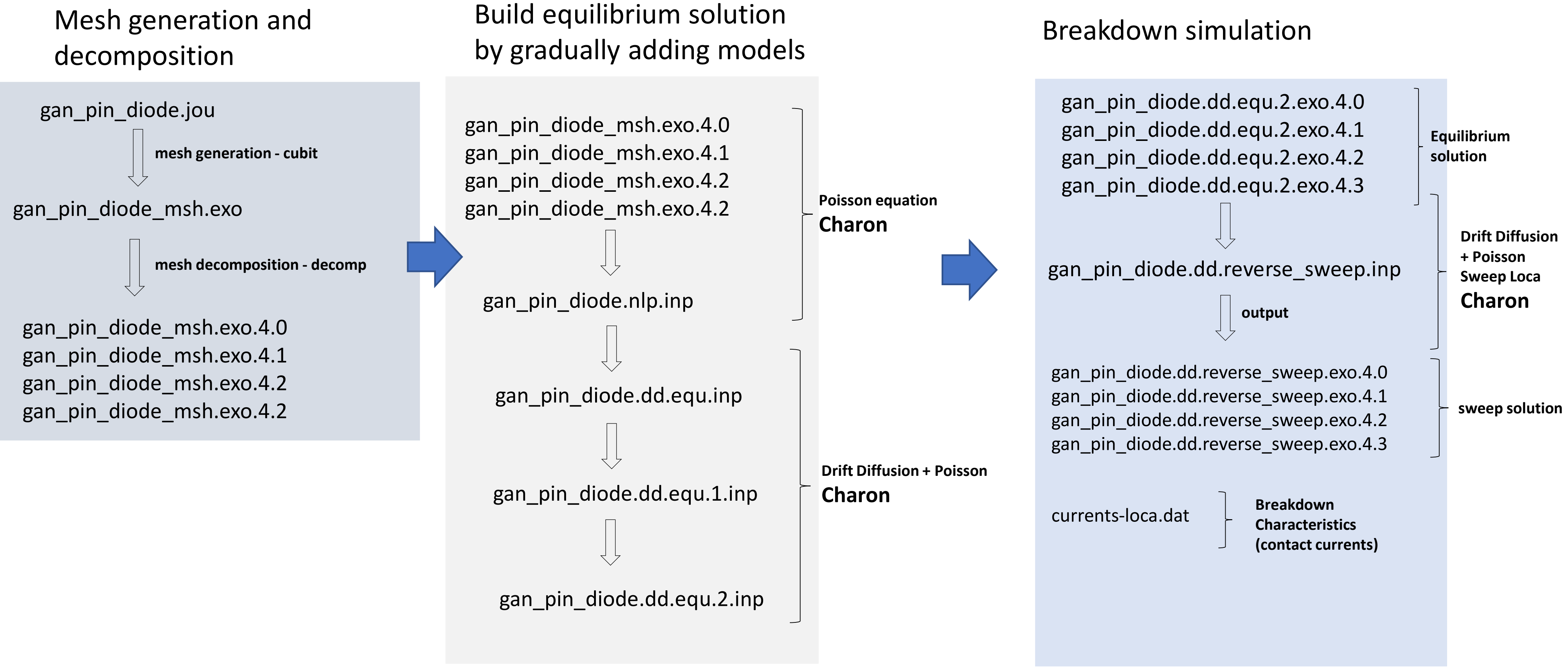
- See Tutorial5
- $G_{opt} = 1.0 \times 10^{20}\ cm^{-3}\ s^{-1}$
- SRH with $\tau_p = \tau_n = 5 \times 10^{-10}$ s
- radiative recombination with $B = 2.0 \times 10^{-11}\ cm^3 s^{-1}$
- Auger Recombination with $C = 1.5 \times 10^{-30}\ cm^6 s^{-1}$
- Farahmand mobility with high field saturation for electrons and constant mobility for holes $\mu_p = 11.0\ cm^2/Vs$
- Fermi Dirac Statistics enabled
- Incomplete ionization (acceptor ionization energy 0.18 eV, donor ionization energy 0.012 eV)
- avalanche generation with effective field as driving force for reverse bias (E0_e = E0_h = 3.5×10^7 V/cm, a0_e = a0_h = $3.1 \times 10^7\ cm^{-1}$)
- Numerical scheme: SGCVFEM

GaN PIN diode reverse characteristics



coarse mesh
no of mesh nodes: 4257
no of edges: 8032
no of cells: 3776
no of cores uses: 4

GaN PIN diode breakdown Simulation Flow



Mesh generation and decomposition

gan_pin_diode.jou

mesh generation - cubit

gan_pin_diode_msh.exo

mesh decomposition - decomp

gan_pin_diode_msh.exo.4.0
gan_pin_diode_msh.exo.4.1
gan_pin_diode_msh.exo.4.2
gan_pin_diode_msh.exo.4.2

Build equilibrium solution by gradually adding models

gan_pin_diode_msh.exo.4.0
gan_pin_diode_msh.exo.4.1
gan_pin_diode_msh.exo.4.2
gan_pin_diode_msh.exo.4.2

gan_pin_diode.nlp.inp

Poisson equation
Charon

gan_pin_diode.dd.equ.inp

gan_pin_diode.dd.equ.1.inp

gan_pin_diode.dd.equ.2.inp

Drift Diffusion + Poisson
Charon

Breakdown simulation

gan_pin_diode.dd.equ.2.exo.4.0
gan_pin_diode.dd.equ.2.exo.4.1
gan_pin_diode.dd.equ.2.exo.4.2
gan_pin_diode.dd.equ.2.exo.4.3

Equilibrium solution

gan_pin_diode.dd.reverse_sweep.inp

output

Drift Diffusion + Poisson Sweep Loca
Charon

gan_pin_diode.dd.reverse_sweep.exo.4.0
gan_pin_diode.dd.reverse_sweep.exo.4.1
gan_pin_diode.dd.reverse_sweep.exo.4.2
gan_pin_diode.dd.reverse_sweep.exo.4.3

sweep solution

currents-loca.dat

Breakdown Characteristics (contact currents)