

GCC FRONT-END OF COMPACT MODELING VERILOG-AMS LANGUAGE

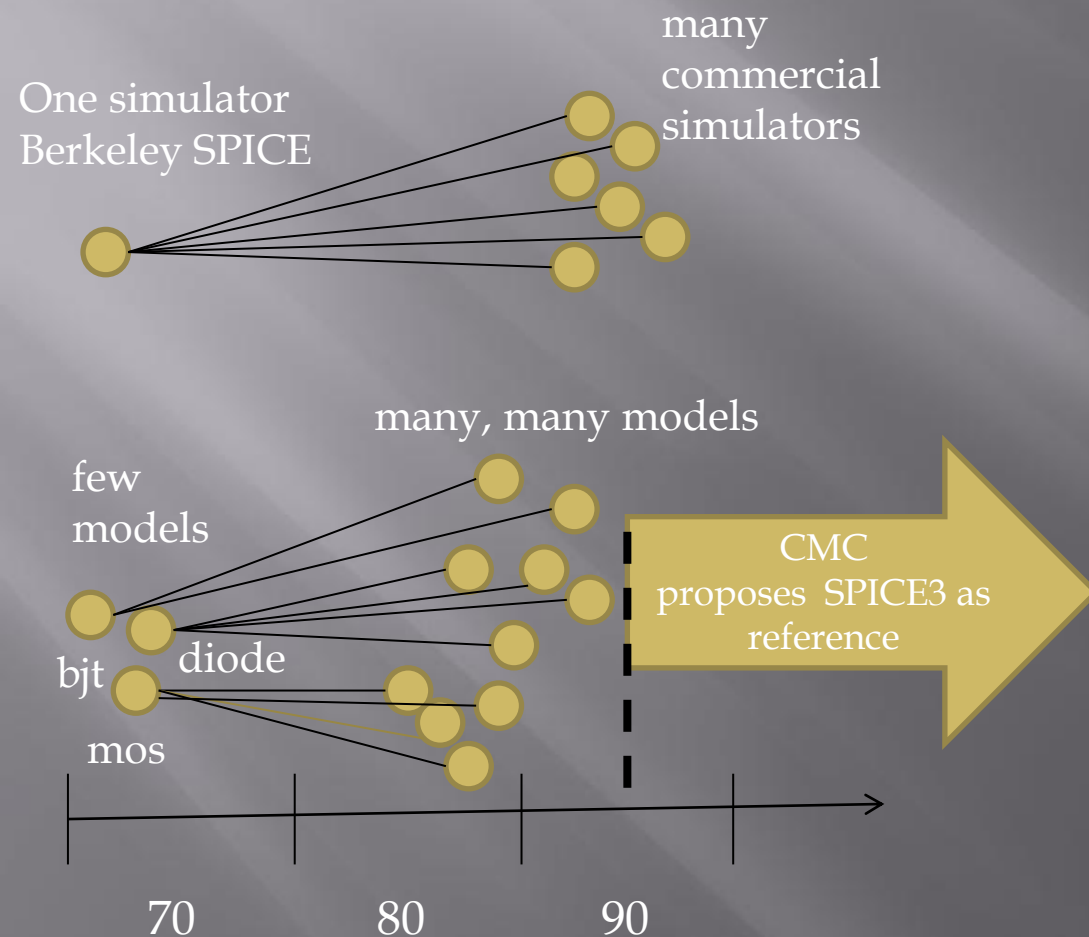
Laurent Lemaitre

www.noovela.com (F)

Contents

- ▣ General considerations about compact modeling
 - a need for standardization
- ▣ Introduce compact modeling and SPICE3 kit
 - SPICE3 kit using C language
 - SPICE3 kit using Verilog-AMS language and model compilers
- ▣ Introduce new solution using GCC front end
 - ▣ Give advantages of this solution
- ▣ Conclusion

Need for Standard to distribute Compact Model Consistently



- ▣ In the 70's:
 - One simulator and few devices
no need for standard
- ▣ In the 90's:
 - many simulators are available
 - more and more compact models
- ▣ ISSUE
 - same model BUT different simulation results between SIMULATOR A and SIMULATOR B.
- ▣ SOLUTION
 - CMC proposed to use SPICE3 as a reference
 - SPICE3 kits for compact models are delivered to EDA companies to qualify the compact model into the simulator

SPICE3 Kit

CMC Standardization

COMPACT MODELING TEAM

SPICE3
KIT

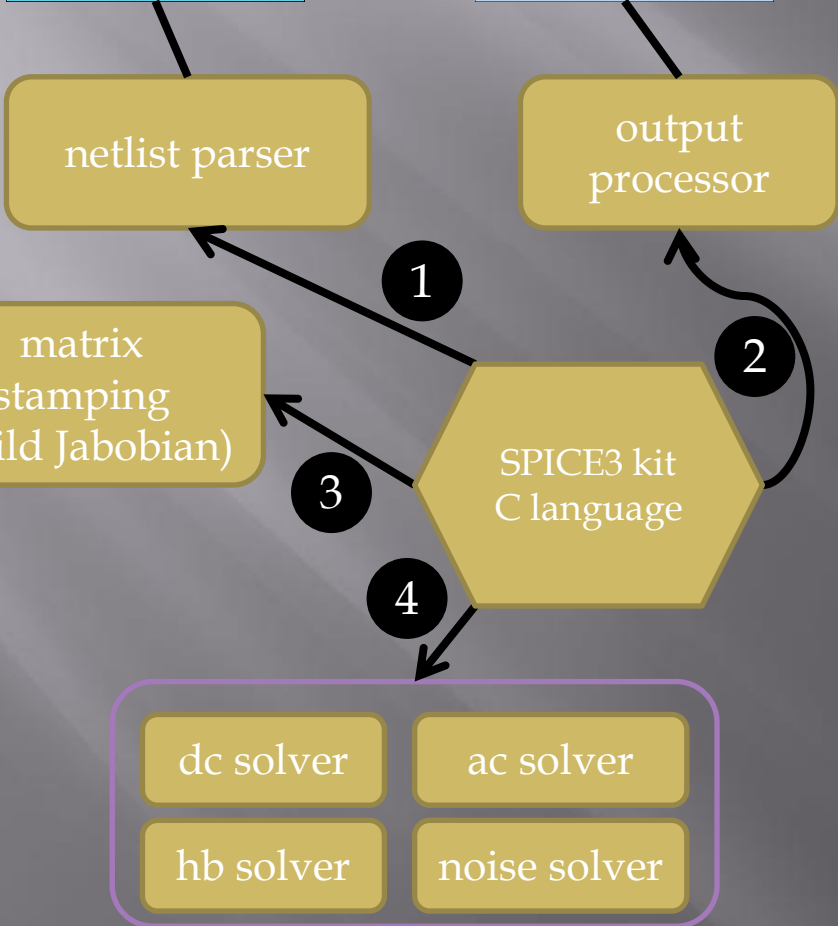
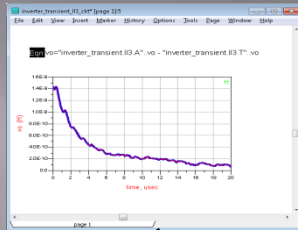
- Write code in SPICE3 format
- Provide test bench in SPICE3 language
- Provide up-to-date documentation of the model

EDA COMPANY

- Qualifies its customized implementation of the compact model by running SPICE3 test bench successfully
- Send feedbacks to compact modeling team

SPICE3 Kit Inside Simulator

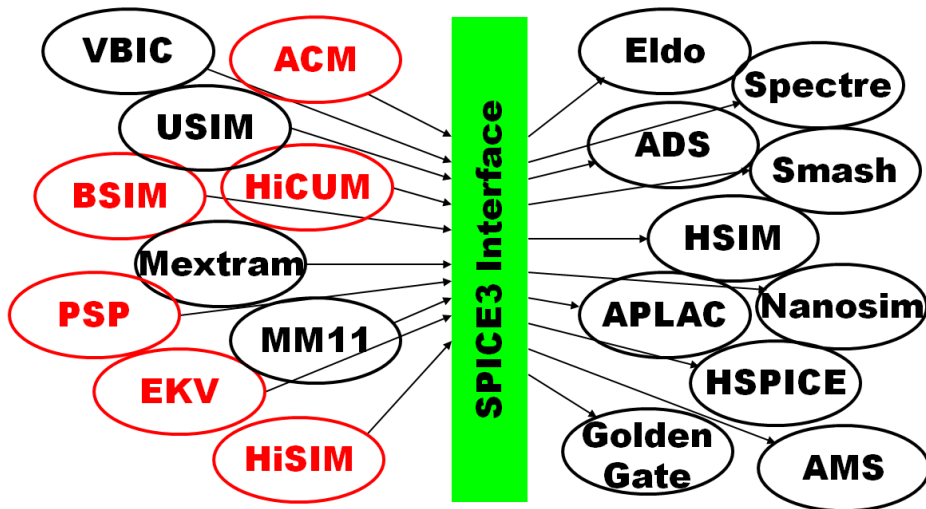
```
inverter.ckt (R_0301adms) - GVIM
File Edit Tools Syntax Buffers Window Help
* --- inverter ---
xinu1 v1 1 supply 0 ng_inu
xinu2 1 2 supply 0 ng_inu
xinu3 2 3 supply 0 ng_inu
xinu4 3 4 supply 0 ng_inu
xinu5 4 vo supply 0 ng_inu
* --- transient analysis ---
simulator lang=ads
Tran:Tran StopTime=5u MaxTimeStep=1n
22,1 95%
```



SPICE3 kit interacts with:

- 1 The netlist parser
 - specifies the syntax in spice netlist
- 2 The output processor
 - specifies how results are displayed
- 3 The Jacobian matrix
 - Simulator must solve $I - J * V == 0$
 - Kit specifies how to stamp the CM into the Jacobian J
- 4 The solvers
 - interacts with solvers to improve convergence
 - this is poor approach: simulation results may be simulator-dependant!!

SPICE3 Kit in C language

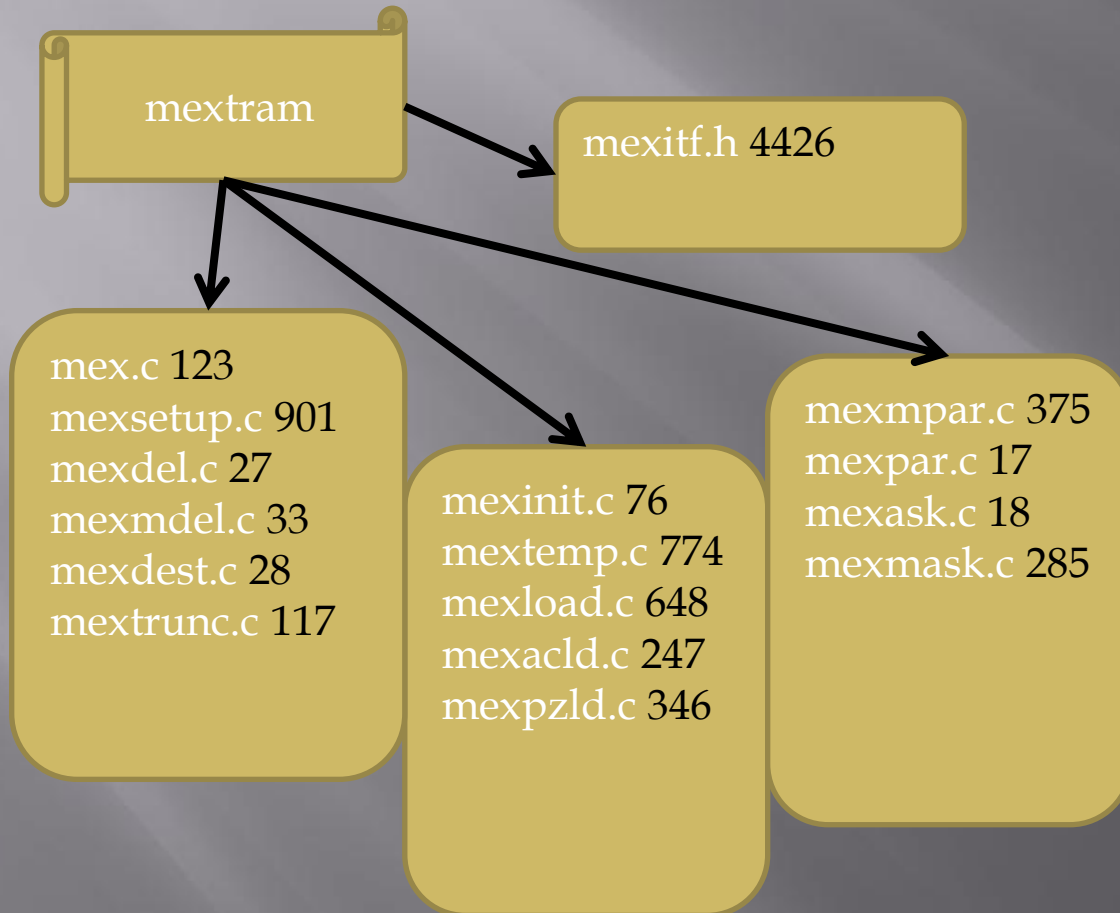


Colin McAndrew and Kevin Cao, IEEE ICCAD 2008

- More devices - More simulators available
- Each simulator support specific syntax
 - SPICE3 CM coding is customized
 - SPICE3 CM test bench are customized
- This means:
 - SPICE3 kit requires manual changes
 - SPICE3 test bench requires manual changes
 - Partial Derivatives are computed by hand
- AS A CONSEQUENCE
 - Simulators can have inconsistent implementation
 - Difficult to know which simulator is right:
 - #define CHARGE 1.6E-19
 - #define CHARGE 1.602E-19
- Next slide gives an example of SPICE3 kit and its complexity

MEXTRAM SPICE3 Kit

Illustration of Complexity



SPICE3 kit are complex.

They are made of lot of files.

Grand Total : 8K lines of code!

COMPLEXITY

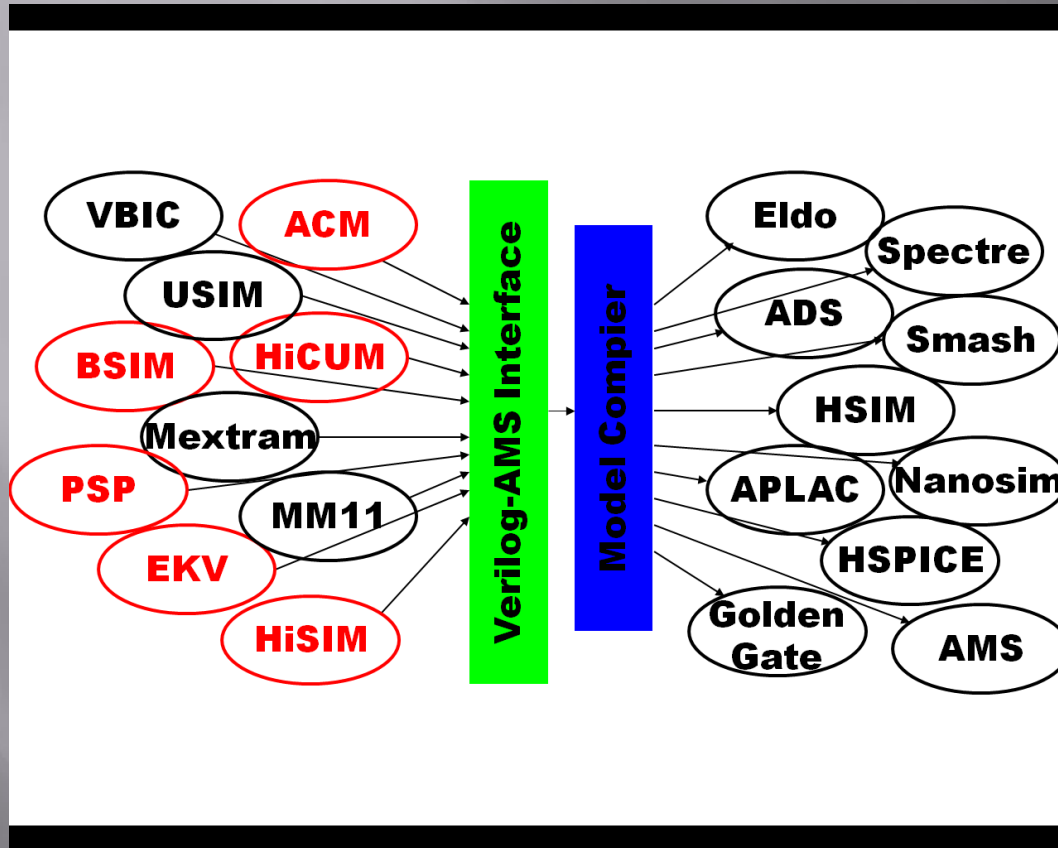
- ❑ Strong dependencies between files
- ❑ Partial derivatives of equations are done by hand
- ❑ Very easy to break consistencies between files

WORKS BUT TOO COMPLEX

- ❑ Next slide presents a new approach adopted by most CM developers

Numbers in black are number of lines in source code

SPICE3 Kit in Verilog-AMS language



SOLUTION:

- ▣ Implement compact models in the Verilog-AMS language
- ▣ Use Model Compiler to create SPICE3 Kit or Simulator Specific implementation

ADVANTAGES

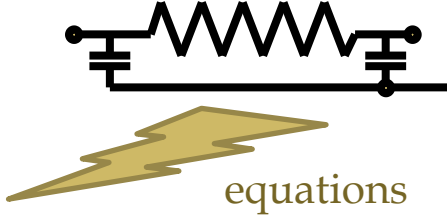
- SPICE3 kit created automatically
- SPICE3 test bench can be specified in the Verilog-AMS language
- Reduce discrepancies between simulators of the same model

DRAWBACKS

- Push all simulators to have efficient Verilog-AMS model compiler
- Loss in performances:
 - ▣ CM run slower
 - ▣ CM have bigger memory usage

Example of Verilog-AMS Coding

```
rpoly.va (R:\r29...\abb\trunk) - GVIM
File Edit Tools Syntax Buffers Window Help
[Icons]
`include "disciplines.vams"
module rpoly(p,q,sub);
// Topology
  inout      p,q,sub;
  electrical p,q,sub ;
// Parameters
  (*desc="width"*) parameter real W=1.0;
  (*desc="length"*) parameter real L=1.0;
  parameter real Cp=1.0;
  parameter real Rs=1.0;
// Variables
  real c;
  real r;
  analog begin
    c=Cp*W*L;
    r=Rs*L/W;
    I(p,q)<+ U(p,q)/r;
    I(p,sub)<+ ddt(c*U(p,sub));
    I(q,sub)<+ ddt(c*U(q,sub));
  end
endmodule
```

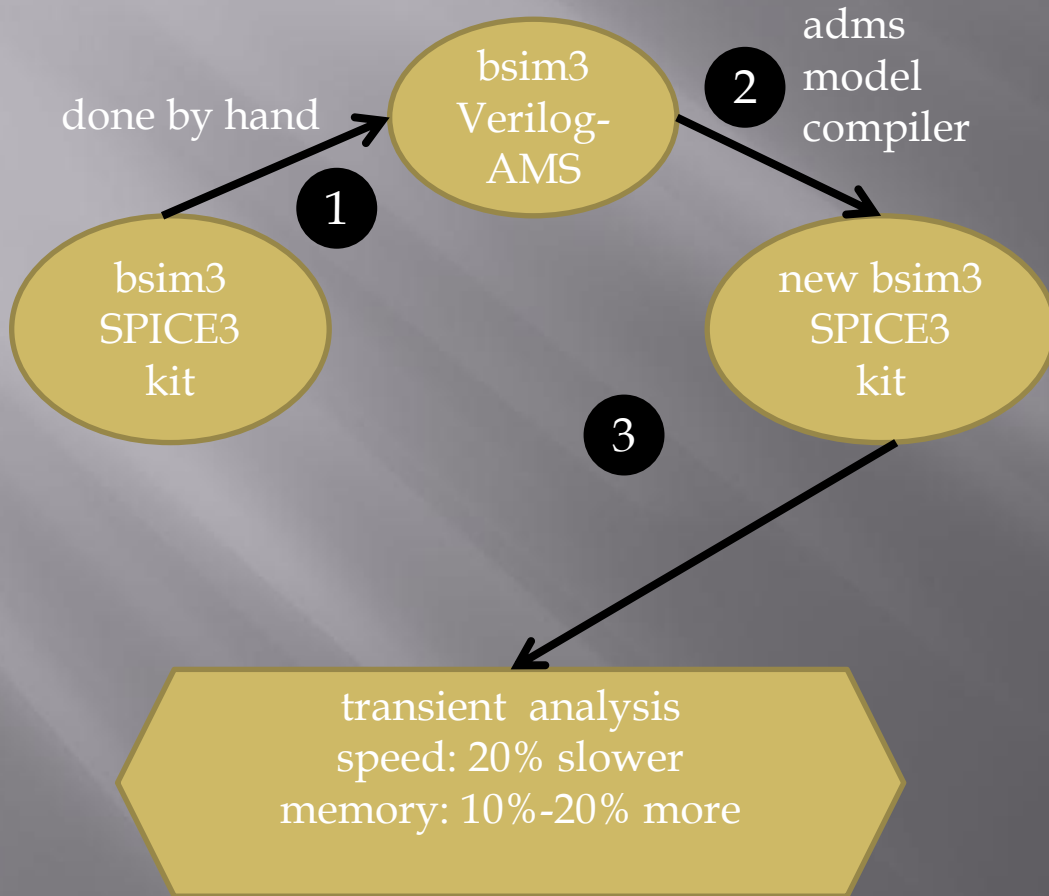


equations

11,13 Top

- Syntax closed to C language - makes it easy to learn
- The CM has about 20 lines of code
- The equivalent SPICE3 kit would contain thousand of lines

Compare Performances Model Compiler vs HandCrafted Model



Experiment:

- 1 from original handcrafted SPICE3 kit reverse engineer BSIM3 in Verilog-AMS
- 2 create new SPICE3 kit using a model compiler
- 3 run transient analysis
 - check that Newton-Raphson iterations are identical
 - check that simulation results are identical

Results and Conclusion:

- Speed is 20% slower
- Memory is 20% more

Conclusion

Performances of models compilers are OK

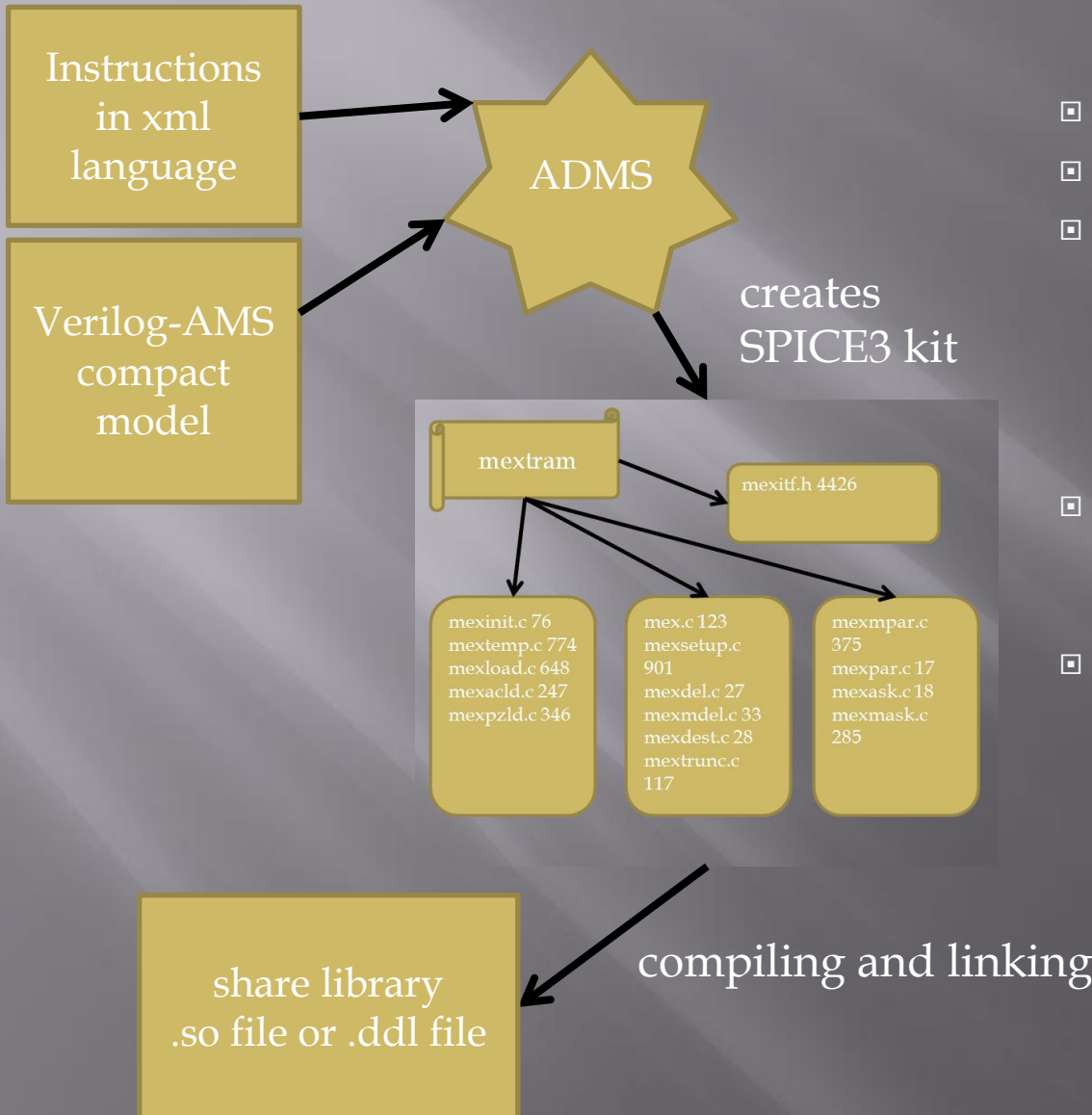
Caution: Performances of model compilers may vary

benchmark PSP model smash other simulator	ADMS -based compiler as reference	Compilers not dedicated to compact modeling
speed	1	10 X slower
memory	1	50% more memory

Source: Verilog-A Compact Model Coding Depeyrot al.
2010 NSTI

- ▣ Verilog-AMS Compilers not dedicated to compact modeling :
 - Poor performances observed
 - Possible explanations:
 - ▣ Verilog-AMS design models have hundred of lines, few variables, event-driven
 - ▣ Compact models have thousands of lines, hundreds of variables and not event driven - UNEXPECTED
- ▣ Alternative:
 - use compilers dedicated to Compact Modeling
 - ▣ ADMS
 - ▣ Tiburon
 - ▣ ...

ADMS FLOW: create SPICE 3 Kit



- ▣ ADMS creates the SPICE3 kit at once
- ▣ ADMS is a source to source compiler
- ▣ ADMS translates Code A to Code B
 - Code A is the Verilog-AMS language
 - Code B is the C language for the SPICE3 API
- ▣ Instructions to convert from code A to code B are provided in xml language
- ▣ Note that code B could be pdf format for documentation applications

ADMS flow and ngspice

```
r29173@r2 /r/r29173/projects/abb/trunk
$ ./admsXml -e prengspice.xml rpoly.va
[info...] admsXml-2.3.0 (1500) Dec 15 2011 22:02:27
[info...] mna.va: temporary file created
[info...] elapsed time: 0 (second)
[info...] admst iterations: 28633 (28478 freed)

r29173@r2 /r/r29173/projects/abb/trunk
$ ./admsXml -e ngspice.xml mna.va
[info...] admsXml-2.3.0 (1500) Dec 15 2011 22:02:27
[info...] option --ngspice activated
[error...] rpoly: device not handled by the ngspice interface
[info...] rpolytemp.c: file created
[info...] rpolyinit.c: file created
[info...] rpolyitf.h: file created
[info...] rpolyask.c: file created
[info...] rpolymask.c: file created
[info...] rpolypar.c: file created
[info...] rpolympar.c: file created
[info...] rpolyload.c: file created
[info...] rpolyacl.d.c: file created
[info...] rpolypzld.c: file created
[info...] rpolytrunc.c: file created
[info...] rpolysetup.c: file created
[info...] rpolydel.c: file created
[info...] rpolymdel.c: file created
[info...] rpolydest.c: file created
[info...] rpoly.c: file created
[info...] rpoly.hxx: created
[info...] elapsed time: 2 (second)
[info...] admst iterations: 49565 (49272 freed)

r29173@r2 /r/r29173/projects/abb/trunk
$ gcc -g -Wall -Isrc/include --shared rpoly*.c -o librpoly.so
rpolysetup.c: In function 'rpolysetup':
rpolysetup.c:49:18: warning: unused variable 'tmp'
rpolysetup.c:48:13: warning: unused variable 'error'
rpolysetup.c:20:7: warning: unused variable 'myCKTmkVolt'

r29173@r2 /r/r29173/projects/abb/trunk
$
```

STEP 1

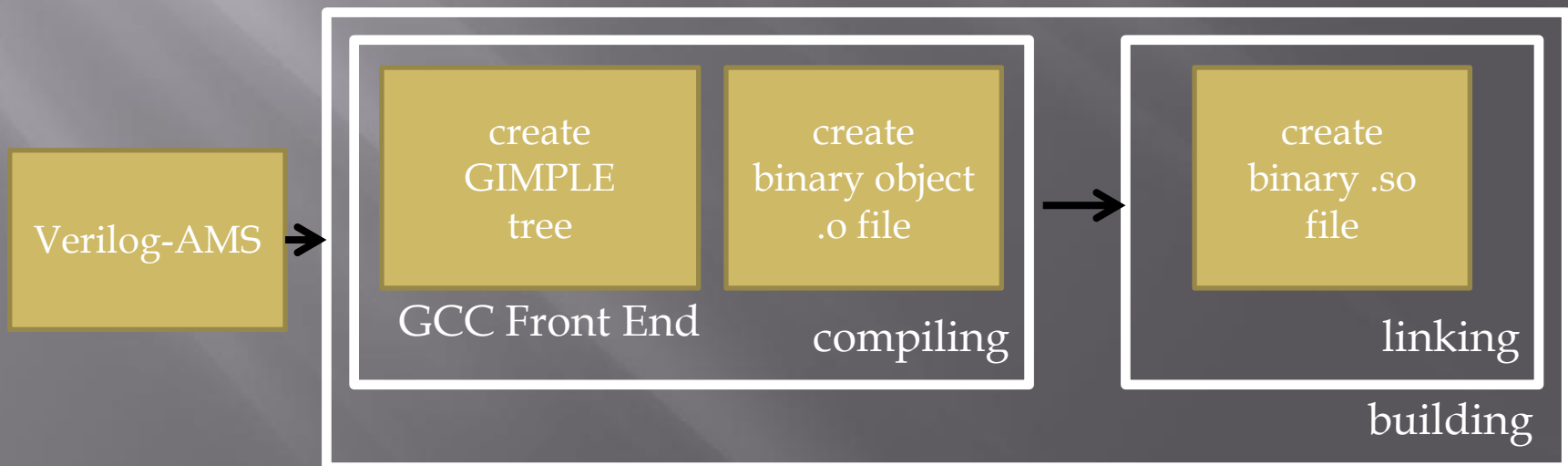
STEP 2

STEP 3

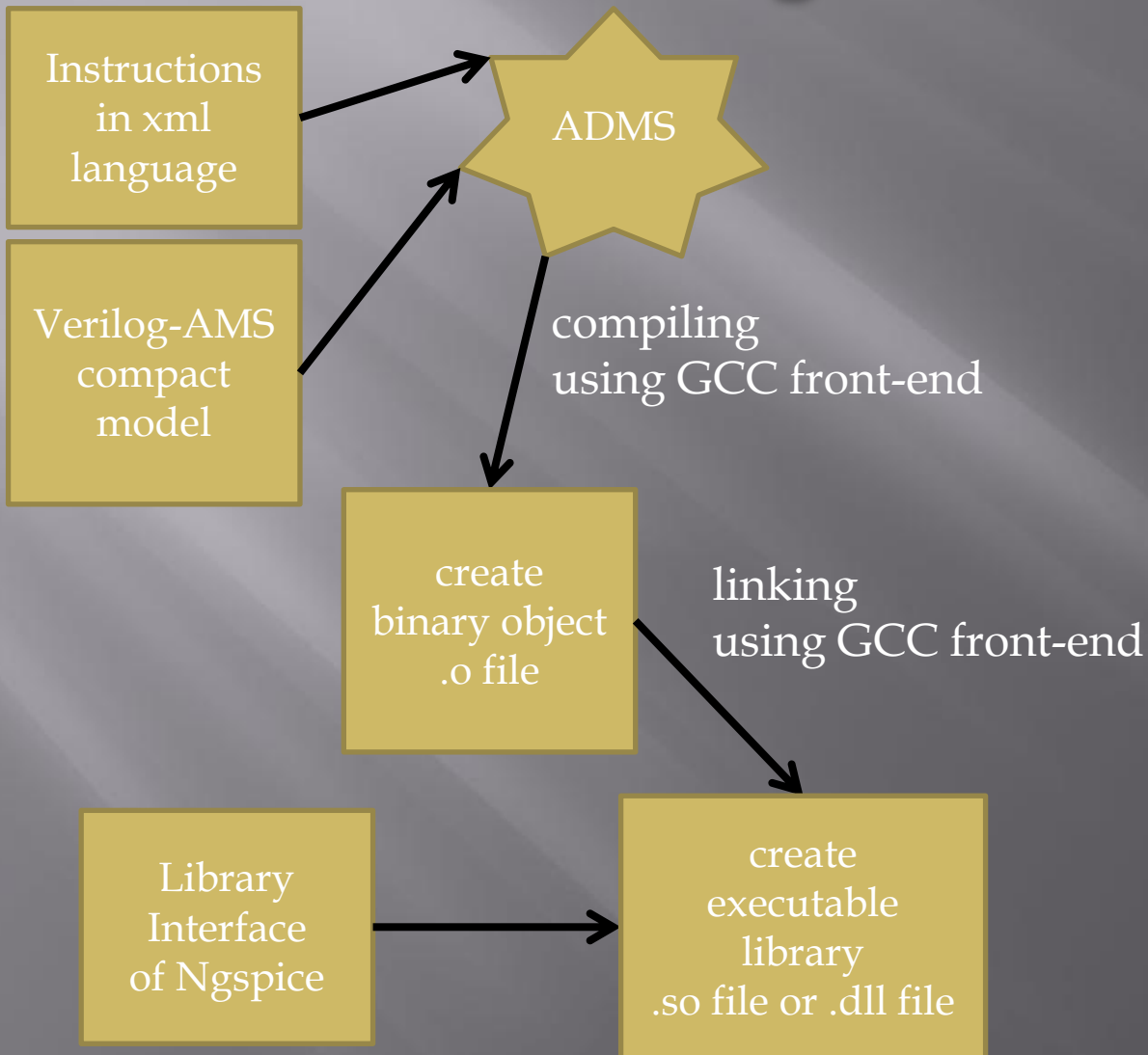
- It is a three steps process
- This command adds some back annotation information to the code to improve the performances of the final implementation.
- This command creates the SPICE3 kit of the rpoly CM.
- This command creates the NGSPICE dynamic library of the rpoly CM.
- Next slide proposes a more compact approach using a GCC front-end

GCC Verilog-AMS Front-End Compilation/Linking

- **Compilation**
 - Process that converts code written in C language, C++ language, ... into machine binary code
 - In GCC this step is done in two steps:
 - create an internal tree representation called GIMPLE representation
 - dump GIMPLE representation into binary code specific to targeted CPU
- **Linking**
 - Process that put all together binary codes and makes an executable binary file
- **Building**
 - Process that combines both sequences: compiling + linking
- Our approach is to create the GIMPLE representation directly from the Verilog-AMS language: IT IS CALLED GCC Verilog-AMS Front -End



NEW ADMS FLOW: create object code using GCC front-end



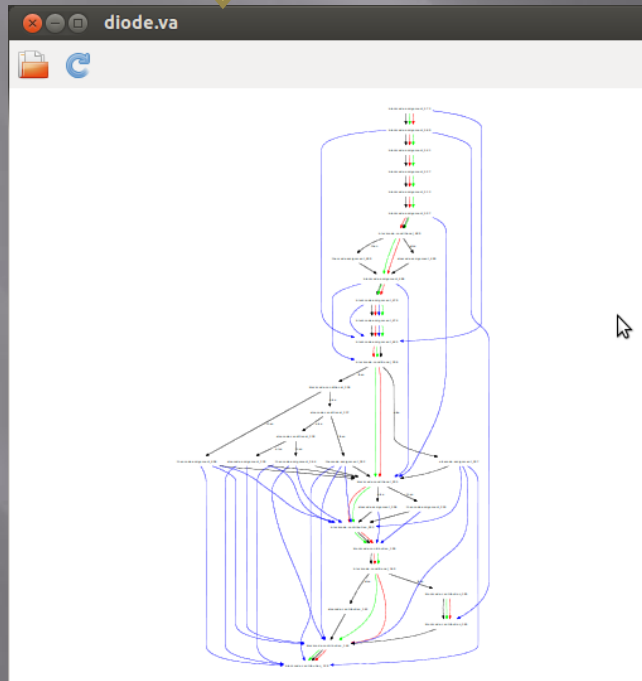
- ▣ ADMS does not produce C language code
- ▣ ADMS produces GIMPLE code
- ▣ GCC compiles GIMPLE code and produces .o object file
- ▣ GCC links .o object to ngspice interface and produces .so file
- ▣ Advantages:
 - avoid intermediate coding in C language
 - take advantage of GCC optimization techniques
 - speed-up the process of building models into simulators
 - make possible interactive applications that require fast re-compilation
 - get gdb debugger for free
- ▣ Drawback
 - Approach tied to GCC compiler
 - user forces to adopt GCC compiler

Verilog-AMS GCC Front End

Verilog-AMS

ADMS creates the GIMPLE tree

Simplified view of the GIMPLE tree of a Verilog-AMS model




- ▣ ADMS creates the GIMPLE tree representation from the Verilog-AMS code
- ▣ Edges: relationships between different pieces of code used by the GCC compiler
- ▣ GCC compiler techniques can be applied to ADMS:
 - tree reduction
 - pattern matching
 - dead code detection

Example of Optimization Code Partitioning

Number of evaluations	Partitioning in SPICE3
few times	model section (process variables, doping, temperature)
some times	instance section (geometries)
many many times	load section (bias dependant)

move invariant code
can have a very
big impact on speed performance



Optimization done
by applying tree operations
on the GIMPLE tree
Easy to implement and debug

Conclusion

- ▣ Compact modeling can easily be done using the Verilog-AMS language
- ▣ Using a dedicated model compiler speed and memory performances using Verilog-AMS is acceptable
- ▣ ADMS can create a Verilog-AMS front end for the GCC compiler in order to speed up the building of Verilog-AMS models into simulators
- ▣ Applications that require fast re-compilation of Verilog-AMS code become feasible:
 - fast re-computation of partial derivatives for sensitivity analysis
 - fast re-computation of partial derivatives used by advanced algorithms in compact model characterization

- ▣ Finally I would like to acknowledge the great work done by MOS-AK and Wlodek Grabinski for the adoption of the Verilog-AMS language as compact modeling language.

THANK YOU!