GnuCap – Architecture, Algorithms and Applications

Felix Salfelder

CAD and Open Hardware devroom
FOSDEM 2019

UNIVERSITY OF LEEDS
Gnucap – Architecture, Algorithms and Applications

- About Gnucap & motivation
- Aims and objectives
  - discrete & continuous models
- Algorithms
- Architecture aspects
  - what makes Gnucap
- Applications
  - gnucap-python
  - QUCS & gnucsator
- Attempt to clarify license issues
About GnuCap & motivation

History

- 1983. First traces (Albert Davis)

Motivation

- replace Spice, the old approach
- better, faster algorithms
- mixed signal simulation (now verilog-AMS)
- ongoing research
Aims & Objectives, some Basics

- digital circuit recap
- discrete voltages
- discrete time, *event queue*
- evaluate event, create new ones, then
- advance time to next event etc.
Aims & Objectives, some Basics

- analogue circuit net with conductances and controlled sources
- operating point and transients: similar problems, look at the former.
From circuit to matrix

- conductances and current sources are known
- currents into a net sum to zero. let $g = 1/r$
- top right $(u_1 - u_2) \cdot g_1 - (u_2 - u_0) \cdot g_2 = 0$
- top left $(u_1 - u_2) \cdot g_1 - i = 0$ etc.

$$
\begin{pmatrix}
g_2 & -g_2 & 0 \\
-g_2 & g_2 + g_1 & -g_1 \\
0 & -g_1 & g_1
\end{pmatrix}
\begin{pmatrix}
u_0 \\
u_1 \\
u_2
\end{pmatrix}
=
\begin{pmatrix}
i \\
-i \\
0
\end{pmatrix}
$$
Analog circuit, some facts

\[
\begin{pmatrix}
g_2 & -g_2 & 0 \\
-g_2 & g_2 + g_1 & -g_1 \\
0 & -g_1 & g_1 \\
\end{pmatrix}
\begin{pmatrix}
u_0 \\
u_1 \\
u_2 \\
\end{pmatrix} -
\begin{pmatrix}
i \\
-i \\
0 \\
\end{pmatrix} = 0, \text{ easy}
\]

▶ more common: \( g \) and \( i \) depends on \( u \)

\[
F(u) = M(u) \cdot \begin{pmatrix}
u_0 \\
u_1 \\
u_2 \\
\end{pmatrix} -
\begin{pmatrix}
i_0(u) \\
i_1(u) \\
i_2(u) \\
\end{pmatrix} = 0
\]

▶ need \( u \) s.th. \( F(0) = 0 \)

▶ compute \( M(u), y(u) \) and \( M(u)^{-1}y(u) \)

for many \( u \) ("Newton iteration")

▶ quite expensive operations
Algorithms, bypassing

- event queue $\rightarrow$ evaluation queue
- Gnucap keeps track of voltage changes.
- $M(u), y(u)$, model evaluation as needed
- $M(u)$, filling in the matrix as needed
- Inversion more difficult.
- $M^{-1}$ is never computed, decompose $M = L \cdot U$ instead
- But only update $L$ & $U$ where $M$ has changed
- Tons of overhead, gain is even higher (without loss in accuracy)
Algorithms – remarks

- More bypassing is possible. Not implemented
- Swapping $n_1$ and $n_2$ affects the nonzero pattern

\[
\begin{pmatrix}
* & * & 0 \\
* & * & * \\
0 & * & * \\
\end{pmatrix}
\iff
\begin{pmatrix}
* & 0 & * \\
0 & * & * \\
* & * & * \\
\end{pmatrix}
\]

- The node ordering determines the storage
- which affects LU decomposition
- Finding a good node ordering is hard.
- the (related) *bandwidth problem* is NP-hard.
Architecture – why bother?

- the parts that will not easily change
- Conceptual integrity, reflect the vision of the architect.
- It’s hard to get all of
  - stability
  - scalability
  - maintainability
  - extensibility
- Verilog-AMS, VHDL, SystemC/AMS considered
- Only few unsettled parts
Architecture – library and plugins

- **Shared library**
  - objective approach (subset of C++)
  - base classes for the replaceable parts
  - POSIX (but portable)
  - generic relation pattern
  - application independent, minimal

- **Everything else: plugins (dlopen)**
  - components
  - commands, algorithms
  - room for contributions and WIP
  - unsupervised but controlled growth
Architecture – plugin benefits

- easy development & deployment
- Touring completeness
- (usually) no forking required
- success stories
  - >>> import module (python)
  - # insmod module (linux)
- new ideas (customisation, gnuicap-custom)
Applications: gnucap-python

- use Gnucap in a Python program\(^1\)
  do plotting or parameter optimisation etc.
- Gnucap in a Python/Jupyter notebook\(^2\)

![Code snippet and plot]

- readily available in Debian, Arch (AUR)
- the usual python wrapping
- but not just that

\(^1\)Henriks idea
\(^2\)kindly provided by Patrick
more gnucap-python

- implement components in Python
  - suited for testbenching
  - custom probes, logic analyser etc.
  - arbitrary data sources in simulations
- implement commands in Python
  - access internal data (numpy)
  - combine with other Python libraries
  - e.g. SPICE-like .pz command, calling scipy.linalg.eig()
- the implementation
  - SWIG and some tweaks
  - share a symbol space
  - maximise hackability
Application in QUCCS

- graphical schematic capture
- displays simulation results
- uses qucsator circuit simulator
  - custom netlist format
- (Qt5 port pending)
Application in QUICS

- gnucsator: a few plugins to read qucsator format and produce qucsator output.
- relevant components in a library

```
$ QUCSATOR=gnucsator.sh qucs -i rc.sch
```
QUCS/gnucssator and Verilog-A
Model license considerations

- Support for models from other projects (not all projects are GPLv3+)
- 1. Models distributed as source code
   - gnucap> load_va neuron.va
     (no issue whatsoever)
- 2. Models distributed as binary blob
   - now need model blob and wrapper
   - model distribution is legal, if it’s yours
     (and no copyleft code involved)
   - wrapper binary might not be distributable
     (just compile as needed)
   - see examples in gnucap-models
     not all those models are free
Summary

- faster algorithms by thinking mixed mode
- Gnucap provides the space for improvement
- more frontend work is on the way
- model licenses are not an issue
Thank You.