KCL Verification Implementation in NGSPICE Linear and Non-Linear Contributions Separation Device Load Routines Accelerated on a NVIDIA GPU

Francesco Lannutti[1,2]

[1] University of Roma ”Sapienza”, DIET

[2] NGSPICE Team

SAPIENZA
UNIVERSITÀ DI ROMA

Sep 20th, 2013 – MOS-AK 2013, Bucharest – Romany

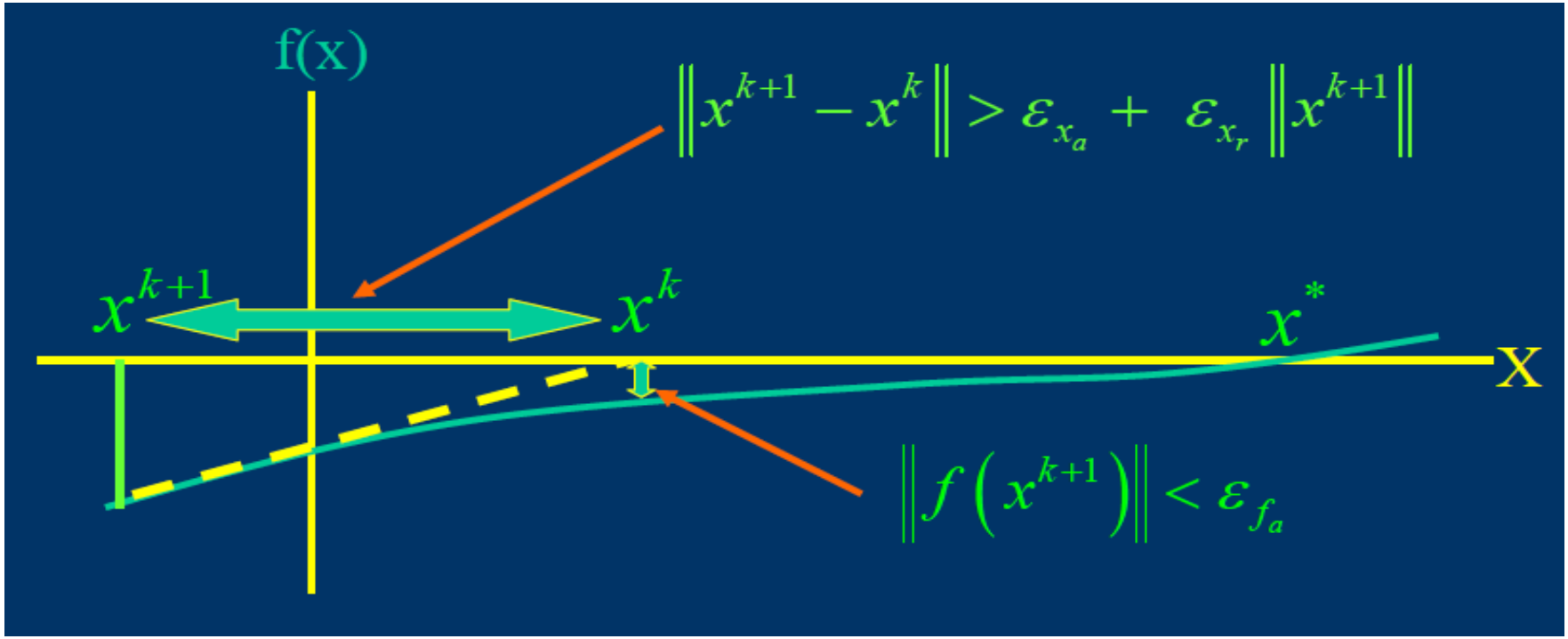
Overview

KCL Verification Implementation in NGSPICE	False Convergence Phenomenon
	KCL Verification
	Nodes Classification
	F(Vk) Extraction Flow
	Homotopy Problem
	Results
	Conclusion
Linear and Non-Linear Contributions Separation	The idea
	Separation Methodology
	Case 1: A_{11} is Full Rank
	Schur Complement
	Case 2: General Case
	Conclusion
Device Load Routines Accelerated on a NVIDIA GPU	SPICE Details
	Device Model Evaluation
	Basic Device Model Evaluation
	BSIM4v7 Device Model Evaluation
	Conclusion

KCL Verification Implementation in NGSPICE

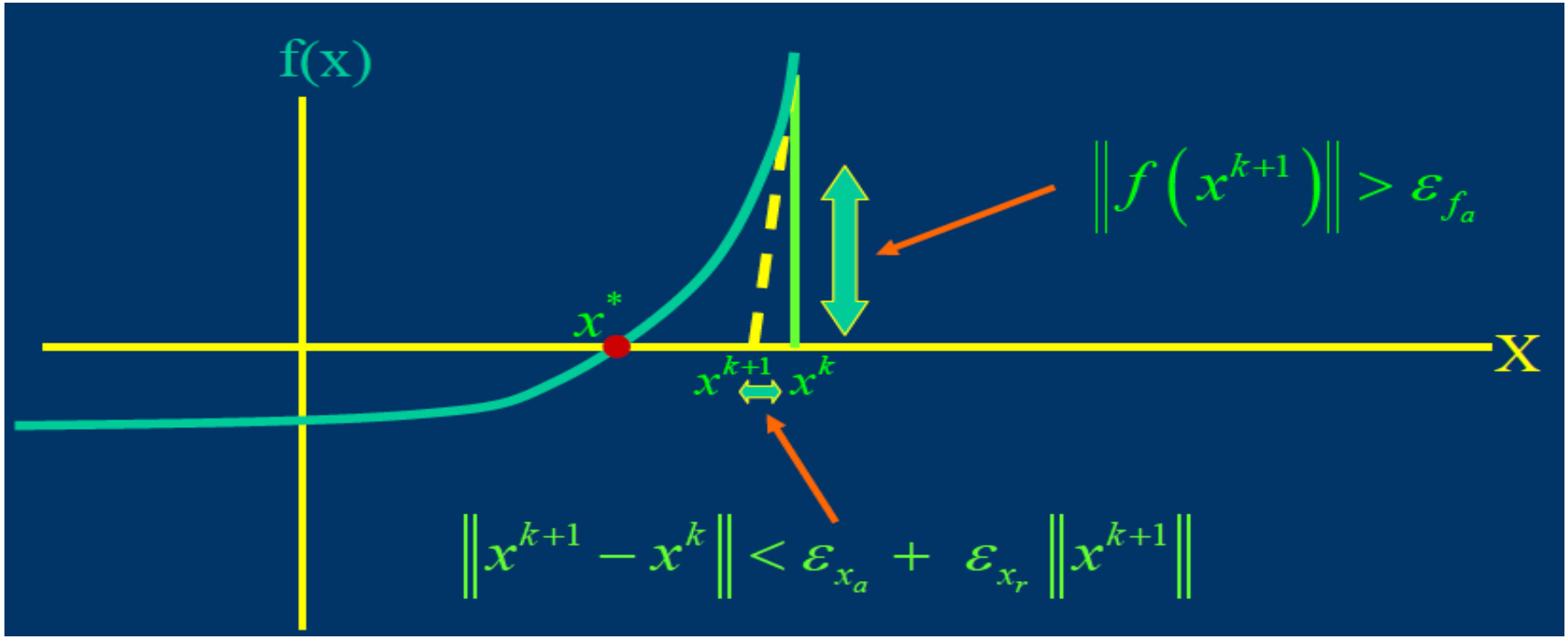
False Convergence Phenomenon

- The Newton-Raphson Method converges if Delta-V is under a certain threshold
- The example below shows a non-convergent case



False Convergence Phenomenon

- That test is not sufficient to guarantee the real convergence
- Also F(V_k) has to be limited, otherwise another non-convergent case can appear



False Convergence Phenomenon

- **Actually $F(V_k)$ test is wrong**[1]
  - Subjected to the False Convergence Phenomenon
- This test has to be substituted by KCL Verification
- KCL Verification is a circuit level test
- There is no need to build $F(V_k)$ test **anymore** for every model
  - **ADMS benefits of this**

1: K. Kundert, I. Clifford, Achieving Accurate Results With a Circuit Simulator, Cadence Design Systems (Analog Division), San Jose, CA

KCL Verification in NGSPICE

- KCL Verification has been implemented as follows:

- $\left|F^n\left(\overrightarrow{V_k}\right) + I_k^n - I_s^n\right| \leq RELTOL * MAX\left|F^{nj}\left(\overrightarrow{V_k}\right), I_k^{nj}, I_s^{nj}\right| + ABSTOL$
  - $F^n\left(\overrightarrow{V_k}\right)$ is the total current at the n_{th} node which comes from non-linear and linear devices
  - I_k^n is the current at the n_{th} node which comes from an independent voltage source
  - I_s^n is the total current at the nth node which comes from independent current sources
  - "j" is the branch index

Nodes Classification

- Convergence tests are only needed for non-linear nodes
- So KCL is only needed for non-linear nodes
- Definition: **a node is non-linear if it connects at least a non-linear model** (not device)

$F(V_k)$ Extraction Flow based on BSIM4v7

1) Look at the RHS vector part of the device load routine at annotate every node
2) Every annotated node is non-linear or purely dynamic linear; the total equivalent current is written here
3) Going backwards, find where the total equivalent current is obtained by summing every equivalent current
4) For non-linear nodes individuate where the equivalent current is composed and extract the real current; for the purely dynamic linear nodes just consider the current stored inside the state vector

$F(V_k)$ Extraction Flow based on BSIM4v7

5) The remaining nodes are purely static liner nodes or “special” non-linear nodes
6) For these nodes it’s necessary to calculate $I = G * V$ explicitly
7) Sum every contribution to form $F(V_k)$, but also maintain it separate for every node
8) Apply the KCL Verification formula

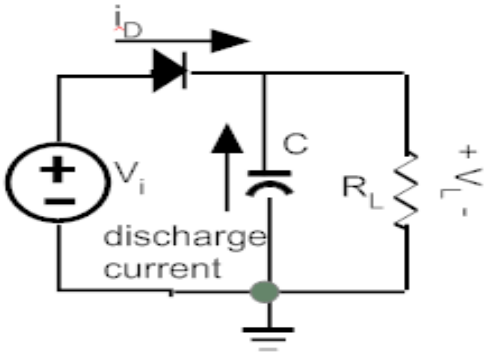
$$\left|F^n\left(\overrightarrow{V_k}\right) + I_k^n - I_s^n\right| \leq RELTOL * MAX\left|F^{nj}\left(\overrightarrow{V_k}\right), I_k^{nj}, I_s^{nj}\right| + ABSTOL$$

DiagGMIN Stepping Problem

- Actual DiagGMIN Stepping algorithm isn't topological
  - A DiagGMIN is added on every diagonal entry in the matrix **after** reordering
  - It doesn't distinguish between Voltage node and Current node
- It cannot verify the KCL
- **The same idea has been recoded to be topological and it verifies the KCL**

Homotopy problem

- Idea
  - Start the Newton-Raphson cycle from a purely linear circuit
  - Exploit GMIN which is already inside a non-linear device
  - Theoretical example: amplitude detector using a diode



  - GMIN is in parallel with the diode and bypass it for certain values
  - **The starting point is a purely resistive circuit for certain values of GMIN and it verifies for sure the KCL**

Homotopy problem

- Tests and experiments are still on going on:
  - Topological DiagGMIN Stepping and fixed device GMIN
  - Device GMIN Stepping and no DiagGMIN
  - Topological DiagGMIN Stepping and device GMIN Stepping with the same value
  - Topological DiagGMIN Stepping and device GMIN Stepping with different values

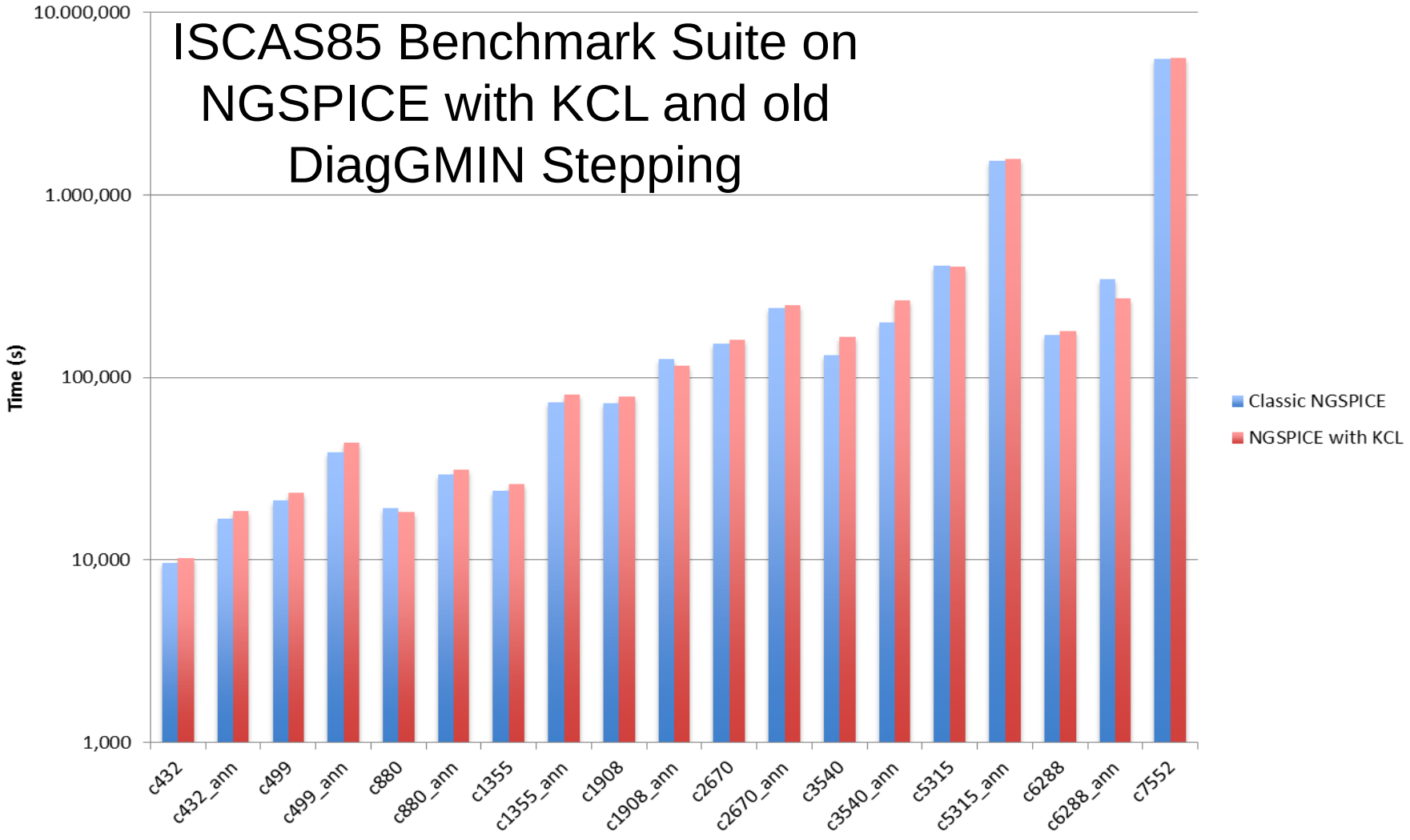
Results

- The ISCAS85 c7552_ann benchmark circuit reached convergence using a target $GMIN = 10^{-12}$ and KCL
  - This circuit has never reached convergence before

- The ISCAS85 c6288_ann benchmark circuit showed an improved speed due to the new GMIN technique

Algorithm	Time (s)	Fill-ins	DC Iterations
Old DiagGMIN	1264,530	124242	533
New GMIN and KCL	860,160	119450	443

Speed Results of the sole KCL



Conclusion

- KCL avoids the False Convergence Phenomenon
- KCL improves the **circuital** accuracy of results
- KCL doesn't decrease the simulation speed

- **KCL lets the model coder avoid to write the $F(V_k)$ convergence test**
  - ADMS doesn't need to take care of convergence tests anymore

- New GMIN Stepping algorithm is crucial for KCL and to solve some other Homotopy Problems

Thanks to my friend Lung-Sheng Chien for the mathematical suggestions

Linear and Non-Linear Contributions Separation

The Idea

- Linear System Factorization
  - Takes between 30%-60% of the simulation time
- Linear part of a circuit matrix
  - Completely fixed during the Newton-Raphson cycle
  - Only the dynamic part changes during the transient analysis
- **Pre-process the linear part**
  - This leads to a **reduced** linear system to be factored at every Newton-Raphson iteration

Nodes Classification

- Definition 1: A node is Non-Linear if it connects to at least a Non-Linear model
- Definition 2: A node is Linear if it connects only Linear models
- It's possible to reorder the unknown vector to reflect these definitions

$$V = \begin{bmatrix} V_L \\ V_N \end{bmatrix}$$

Separation Methodology

- It's possible to find a permutation vector P (and Q) which separates a matrix A in the following form

- $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

- A_{11} is the Linear Part
- A_{12} connects Linear and Non-Linear Parts
- A_{21} connects Non-Linear and Linear Parts
- A_{22} is the Non-Linear Part

Case 1: A_{11} is full rank

- Step 1 – LU decomposition of A:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I & \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix} = LU$$

$\boldsymbol{A_{22} - A_{21}A_{11}^{-1}A_{12}}$ is the Schur Complement of A_{11}

- Step 2 – Solve $Ly = b$:

$$\begin{bmatrix} I & \\ A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad \longrightarrow \quad \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 - A_{21}A_{11}^{-1}b_1 \end{bmatrix}$$

Case 1: A_{11} is full rank

- Step 3 – Solve $Ux = y$ by LU factorization of the Schur Complement

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \longrightarrow$$

$$\longrightarrow \quad \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A_{11}^{-1}(y_1 - A_{12}x_2) \\ (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}y_2 \end{bmatrix}$$

- This approach requires that A_{11} is invertible, so it must be full rank

Schur Complement

- $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

- Definition: the Schur Complement of A is the following

$$A_{22} - A_{21}A_{11}^{-1}A_{12}$$

- Only A_{22} is variable inside a Newton-Raphson cycle

- That Schur Complement or its LU can be calculated efficiently using known methods

Case 2: General Case

- We can still find two permutation vectors P and Q such that $PA_{11}Q^T = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$ and C_{11} is full rank

- Using this idea, the linear system becomes:

$$\begin{bmatrix} P & \\ & I \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} Q^T & \\ & I \end{bmatrix} \begin{bmatrix} Q & \\ & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} P & \\ & I \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \longrightarrow$$

$$\longrightarrow \begin{bmatrix} PA_{11}Q^T & PA_{12} \\ A_{21}Q^T & A_{22} \end{bmatrix} \begin{bmatrix} Qx_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} Pb_1 \\ b_2 \end{bmatrix}$$

Case 2: General Case

- Inserting the full rank sub-matrix:

$$\begin{bmatrix} C_{11} & C_{12} & (PA_{12})_1 \\ C_{21} & C_{22} & (PA_{12})_2 \\ (A_{21}Q^T)_1 & (A_{21}Q^T)_2 & A_{22} \end{bmatrix} \begin{bmatrix} (Qx_1)_1 \\ (Qx_1)_2 \\ x_2 \end{bmatrix} = \begin{bmatrix} (Pb_1)_1 \\ (Pb_1)_2 \\ b_2 \end{bmatrix}$$

- Simplifying the notation:

$$\begin{bmatrix} C_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

Case 2: General Case

- Now it's possible to perform LU:

$$\begin{bmatrix} C_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix} = \begin{bmatrix} I & \\ D_{21}C_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} C_{11} & D_{12} \\ & D_{22} - D_{21}C_{11}^{-1}D_{12} \end{bmatrix} = LU$$

- As showed before, we need to calculate LU of the Schur Complement of C_{11}, in a similar way we did in the Full Rank Case

- Only D_{22} changes inside a Newton-Raphson cycle

Conclusion

- This approach reduces the matrix order
  - Back-annotated netlists
  - Post-Layout circuits
  - Post-Synthesis circuits in topographical mode
- It can bring a lot of speedup in the simulation time, depending on the circuit
- The Schur Complement can be denser than the original matrix
  - Good news: **dense approach to calculate LU**: $\frac{N^3}{1TFlops} = 1ms$ using a K20X NVIDIA GPU card on a matrix with N=1024
  - Bad news: Storage problems on big matrices

From the GTC 2013 presentation:

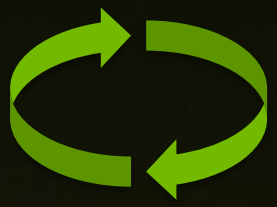
What does it take to accelerate SPICE on the GPU?

M. Naumov, F. Lannutti, S. Chetlur, L.S Chien, P. Vandermersch

Device Load Routines Accelerated on a NVIDIA GPU

SPICE Details

- Device Model Evaluation
  - Takes between 30%-60% of the simulation time
- DC Analysis
  - Device model evaluation
  - Linear system solution



Newton-Raphson

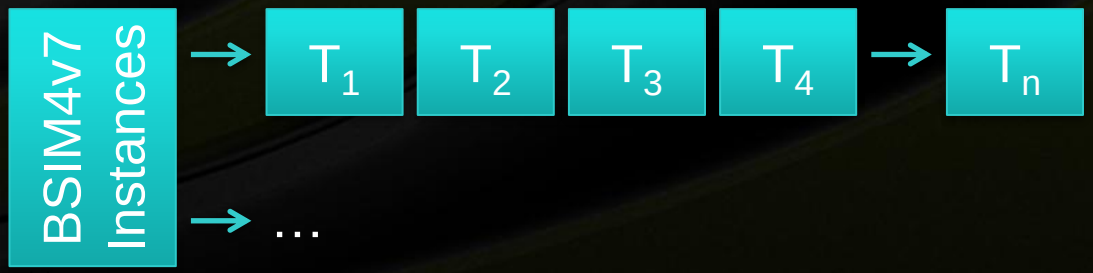
- Transient Analysis
  - Device model evaluation
  - Linear system solution
  - Truncation error + Time step correction

Device Model Evaluation

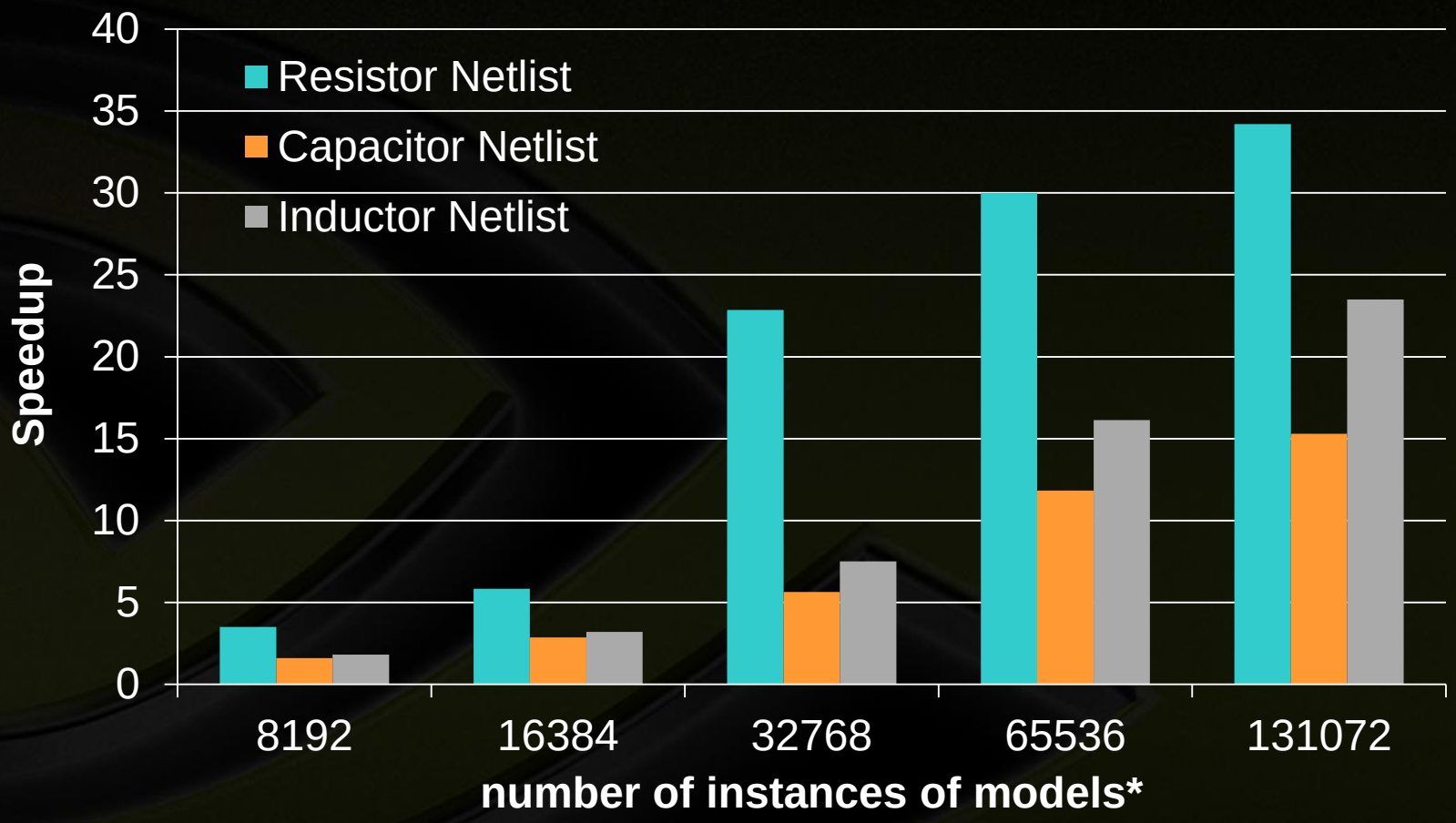
- Basic models
  - Resistor, Capacitor, Inductor, Voltage and Current Sources
- Transistor models
  - MOSFET transistor (BSIM4v7, PSP, etc.)
  - Bipolar transistor (Ebers-Moll, Gummel-Poon, etc.)
- Other models
  - Diodes, etc.

Device Model Evaluation

- Key Idea (Transistor – BSIM4v7)
  - Many branches (if-else) are related to fixed parameters
    - Temperature
    - Process
  - Reorganize the code (slightly)
    - Minimize thread divergence
    - Maximize memory coalescing



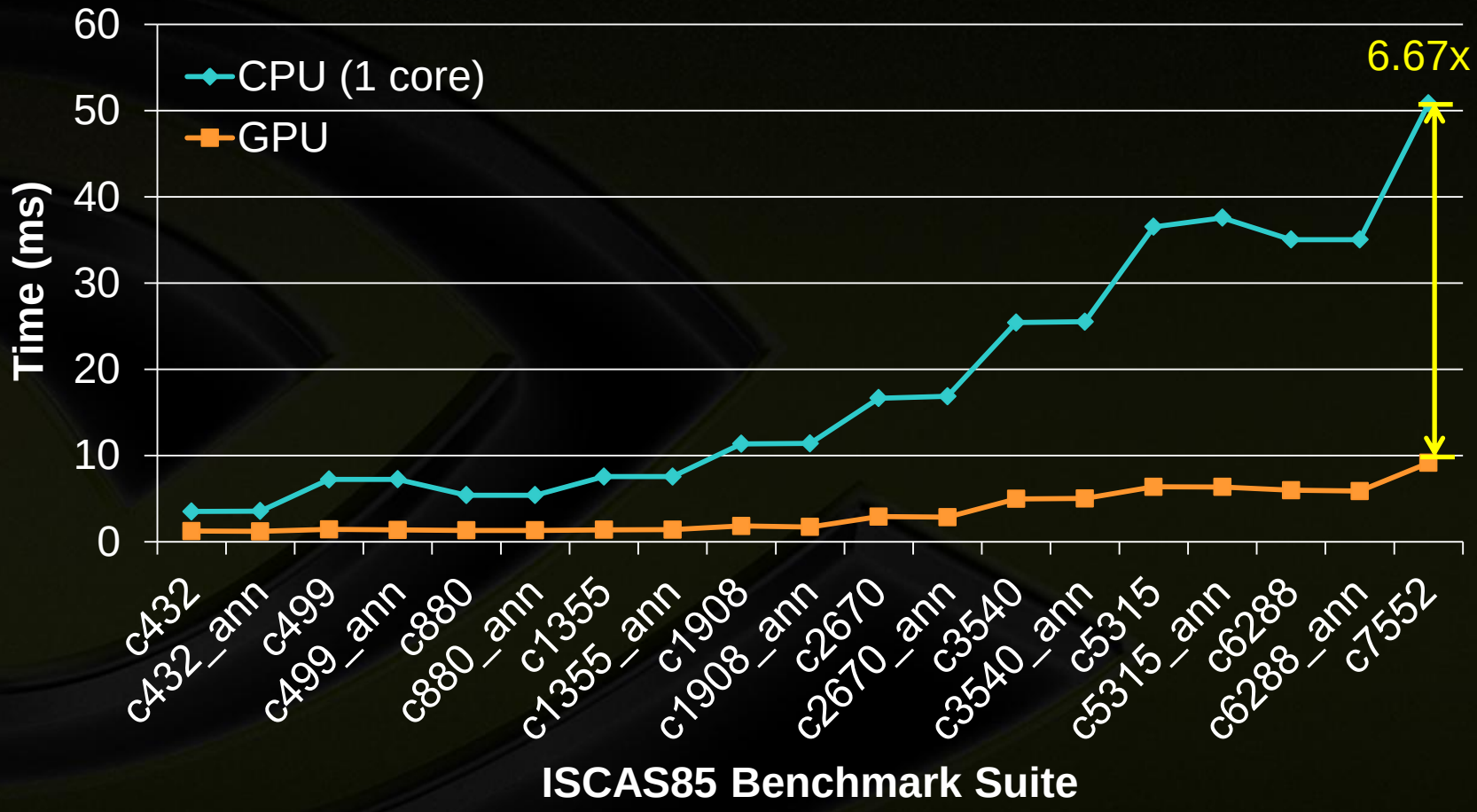
Basic Device Model Evaluation



*Resistor Netlist: all resistors;
Capacitor Netlist: half capacitors and half resistors;
Inductor Netlist: half resistors, quarter capacitors and quarter inductors

*NVIDIA C2070, ECC on
*Intel X5690 (6 Core™) @ 3.47GHz

BSIM4v7 Device Model Evaluation



*NVIDIA C2070, ECC on
*Intel X5690 (6 Core™) @ 3.47GHz

Conclusion

- SPICE simulation two most time consuming parts
  - Device Model Evaluation
  - Solution of linear systems
- Device Model Evaluation
  - Speedup* of up to 6x
- Solution of linear systems
  - Speedup* (average) of 2x
- GPU (overall) acceleration
  - SPICE (overall expected) speedup of 2-3x
  - No slowdown

*speedup is dependent on input parameters

Thanks for your attention

QUESTIONS ?