CUSPICE

The revolutionary NGSPICE on CUDA Platforms

Francesco Lannutti[1,2], Francesco Menichelli[1], Mauro Olivieri[1]

[1] "Sapienza" University of Rome, DIET

[2] NGSPICE Team

SAPIENZA
UNIVERSITÀ DI ROMA

Sep 26th, 2014 – MOS-AK 2014, Venice – Italy

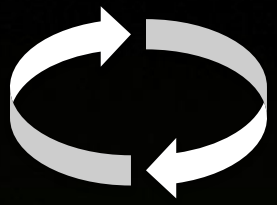
Agenda

CUSPICE

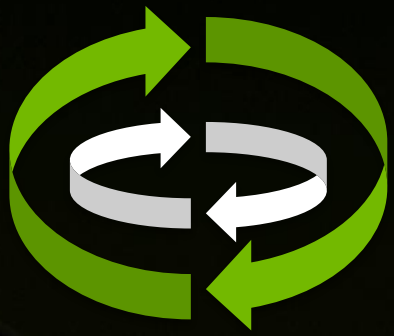
- Time Consuming Parts of SPICE
- The Stamp Method
- CUSPICE Features
- Device Model Evaluation in the GPU
- Circuit Matrix and RHS Update
- Topology Matrix Method
- Local Truncation Error in the GPU
- Simulation Speedup
- Conclusion

Time Consuming Parts of SPICE

- OP Analysis
  - Device Model Evaluation
  - Linear System Solution
- Transient Analysis
  - Device Model Evaluation
  - Linear System Solution
  - Local Truncation Error and Time Step Correction
- Device Model Evaluation
  - Takes about 40%-70% of the entire simulation time
    - It depends on the circuit size and complexity



Newton-Raphson

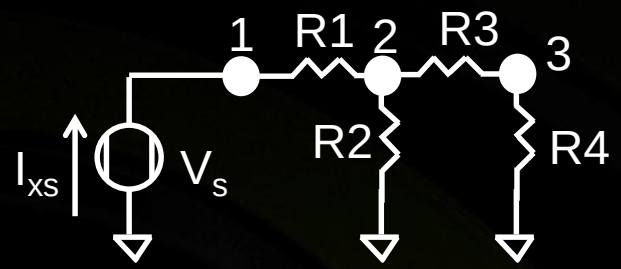


Time

Newton-Raphson

From Circuit to Netlist to Matrix

Circuit:



Netlist (text file):

	V_k	V_j	
R1	1	2	1k
R2	2	0	1k
R3	2	3	0.4k
R4	3	0	0.1k
V1	1	0	PWL (0 0 1n 0 1.1n 5 2n 5)

KCL + Device Models Laws:

$$\boldsymbol{G * V = I_S}$$

$$\boldsymbol{\sum V = V_S}$$

Circuit Matrix:

$$\begin{bmatrix} G_1 & -G_1 & 0 & 1 \\ -G_1 & G_1+G_2+G_3 & -G_3 & 0 \\ 0 & -G_3 & G_3+G_4 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ I_{x_s} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ V_s \end{bmatrix}$$

Resistor Stamp			
Sparse Matrix			RHS
	V_k	V_j	
k	G	-G	
j	-G	G	

Voltage Source Stamp				
Sparse Matrix				RHS
	V_k	V_j	I_{xs}	
k			1	
j			-1	
xs	1	-1		V_s

The Matrix Updates: Stamp Method

- In order to guarantee O(1) Circuit Matrix and RHS update
  - Every device model stores pointers to the exact locations in the Circuit Matrix and RHS where it has to write data
  - Matrix update for device model (the first instance of resistor R_1)

***(resPosPosPtr) += G_1 ;**
***(resNegPosPtr) -= G_1 ;**
***(resNegNegPtr) += G_1 ;**
***(resPosNegPtr) -= G_1 ;**

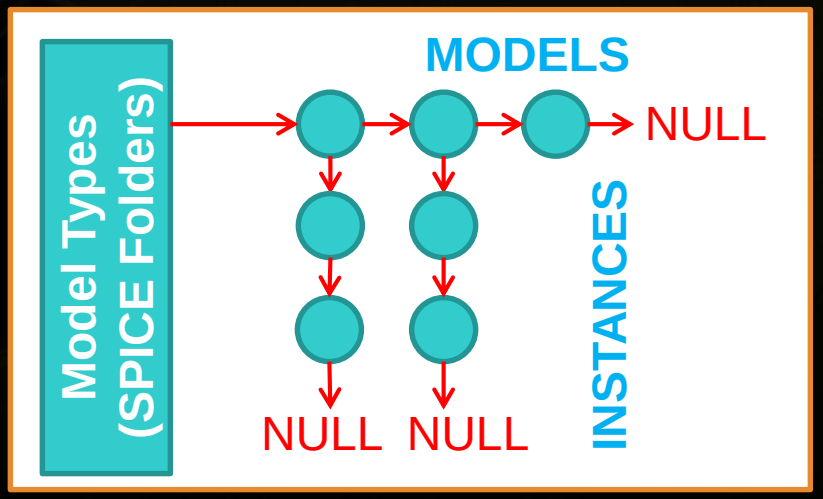
$$\begin{bmatrix} G_1 & -G_1 & 0 & 1 \\ -G_1 & G_1+G_2+G_3 & -G_3 & 0 \\ 0 & -G_3 & G_3+G_4 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ I_{x_s} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ V_s \end{bmatrix}$$

CUSPICE Features

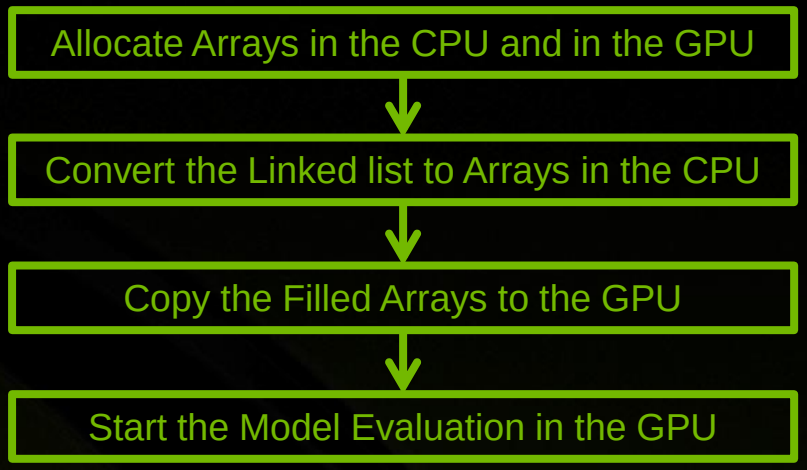
- It’s the NGSPICE version for CUDA Platforms
- It’s fully integrated in the NGSPICE framework
- It supports the following models:
  - Basic models
    - Resistor, Capacitor, Inductor, Voltage and Current Sources
  - Transistor models
    - MOSFET transistor (BSIM4v7)
- It requires at least a Fermi generation GPU
  - Kepler architecture gives better performances

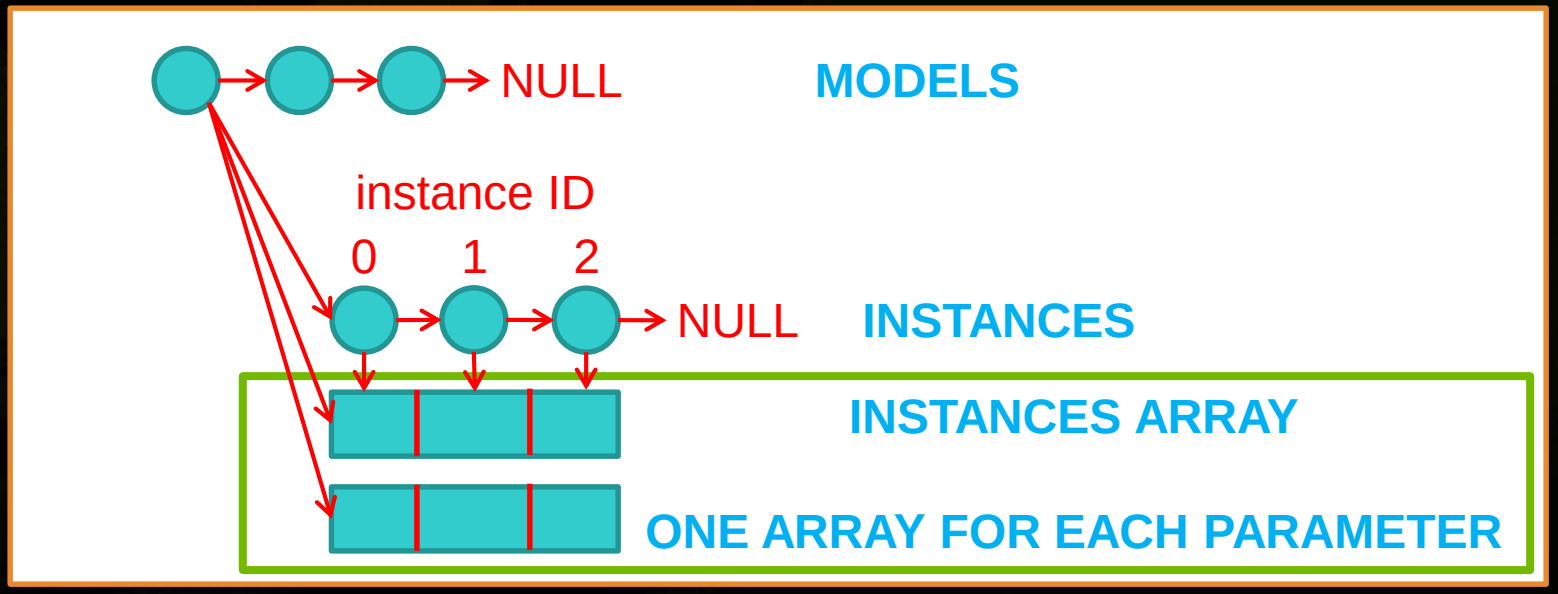
Model Evaluation in the GPU

- Linked lists are used to store models and instances
  - This is not good for the GPU
  - They have been converted into arrays
- Many branches (if-else) are related to fixed parameters
  - Temperature
  - Process
- Reorganize the code (slightly)
  - Minimize thread divergence
  - Maximize memory coalescence

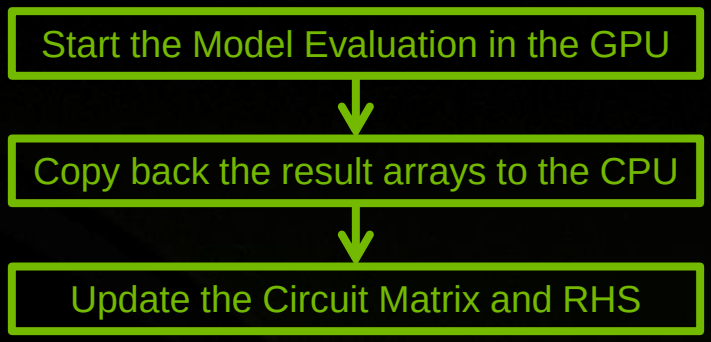


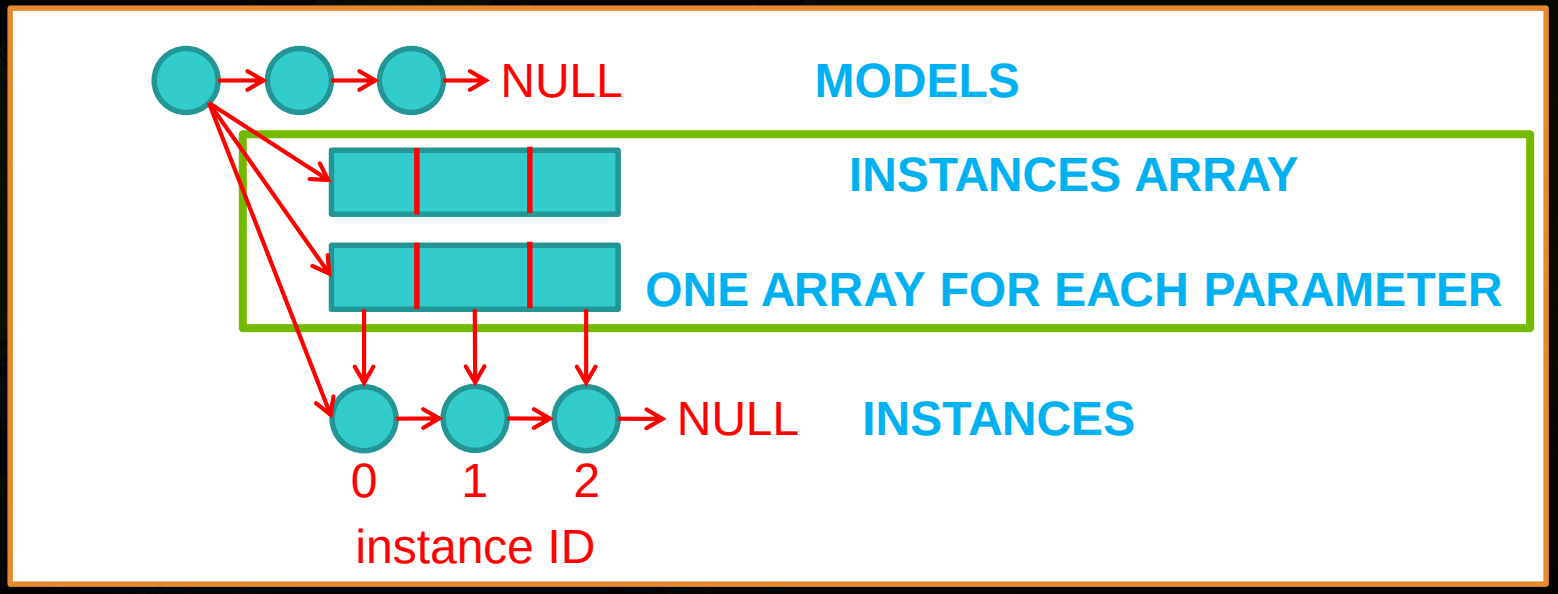
Model Evaluation in the GPU





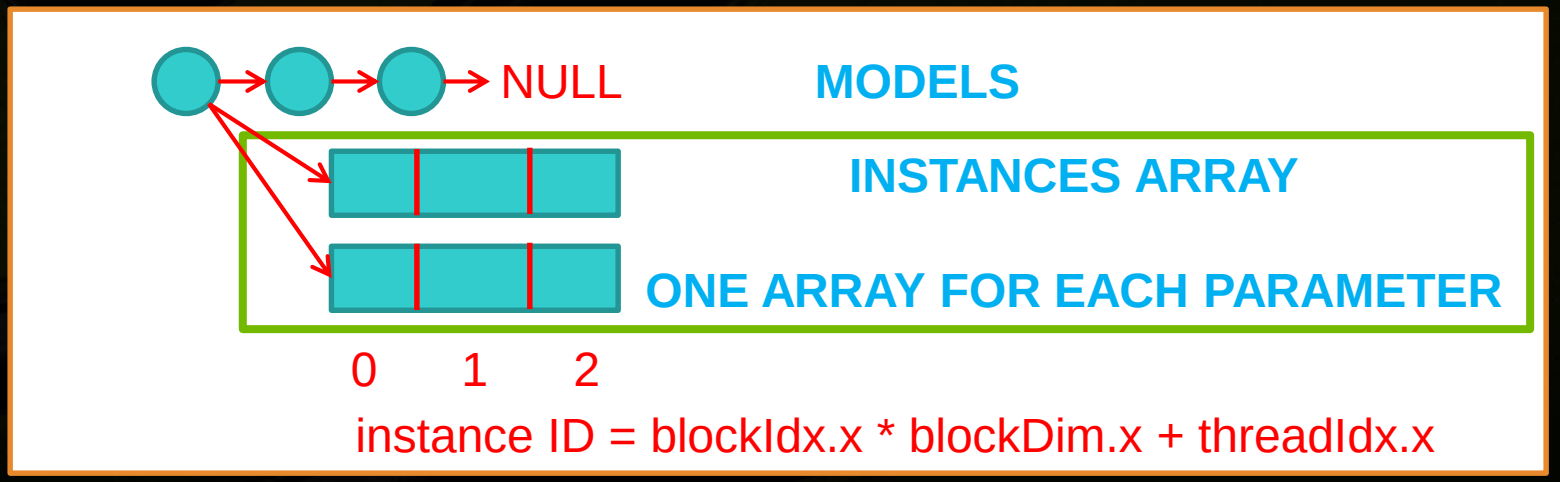
Model Evaluation in the GPU



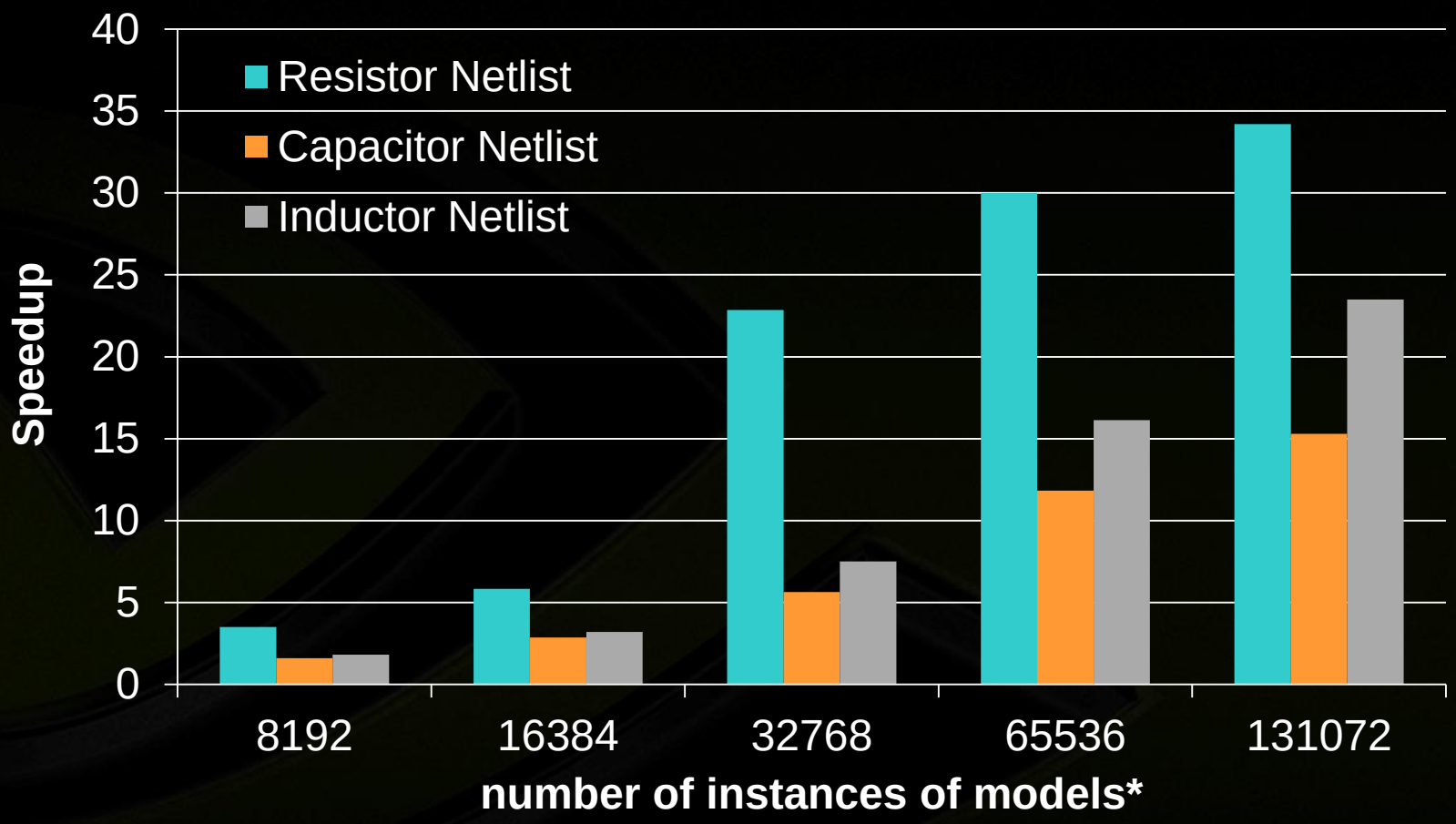


Model Evaluation in the GPU

- Every CUDA thread computes one instance
- **Coalesced access to the global memory is exploited**
- CUDA Streams have been used during the evaluation of p-type and n-type transistor (BSIM4v7)



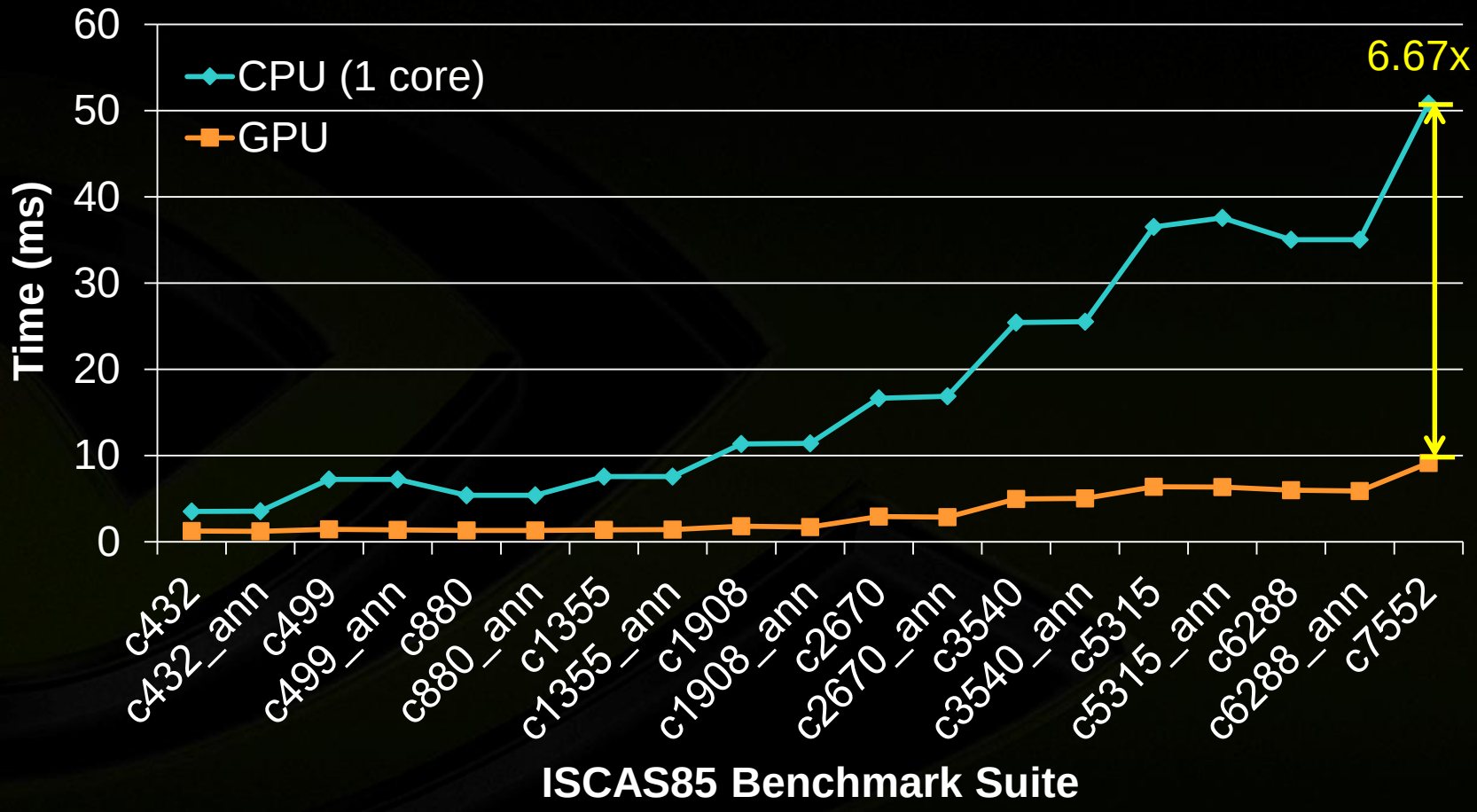
Basic Device Model Evaluation



*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

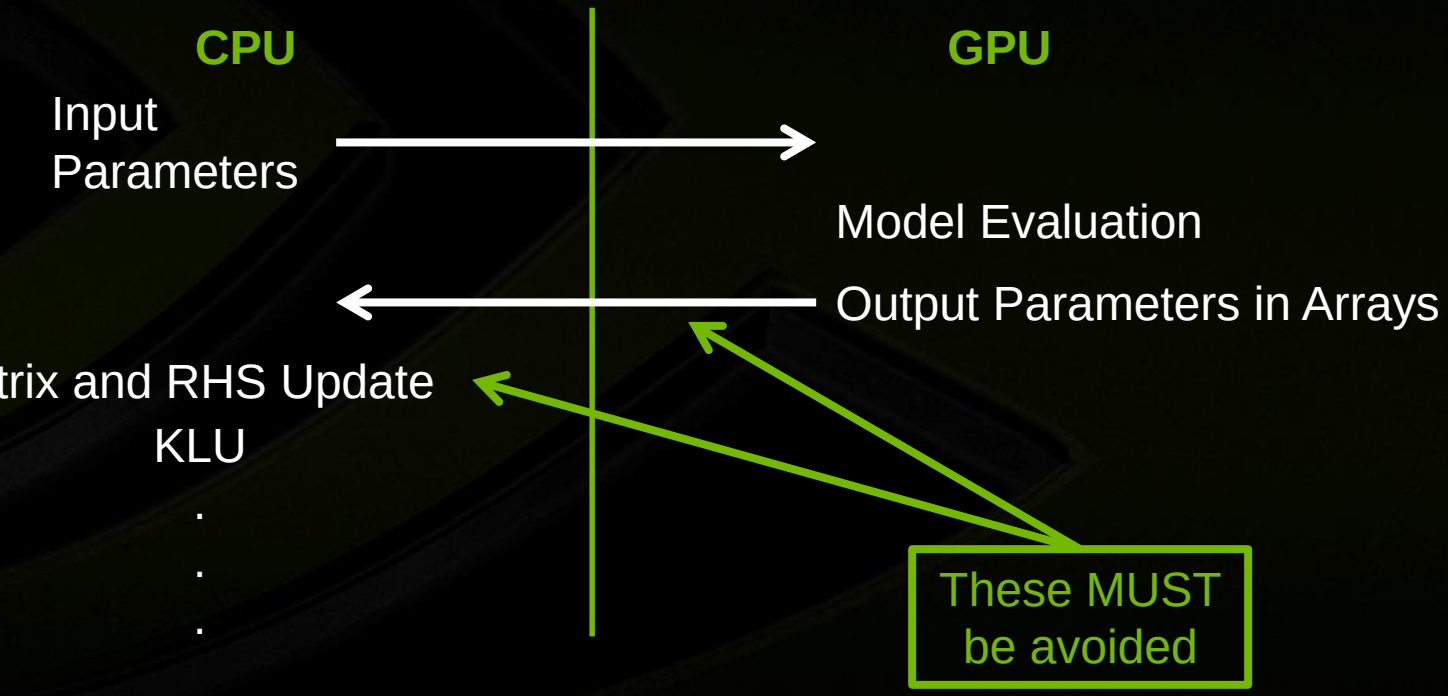


*NVIDIA C2070, ECC on
*Intel X5690 (6 Core™) @ 3.47GHz

Circuit Matrix and RHS Update

After Model Evaluation in the GPU

For each Timestep {
  For each Newton-Raphson Iteration {

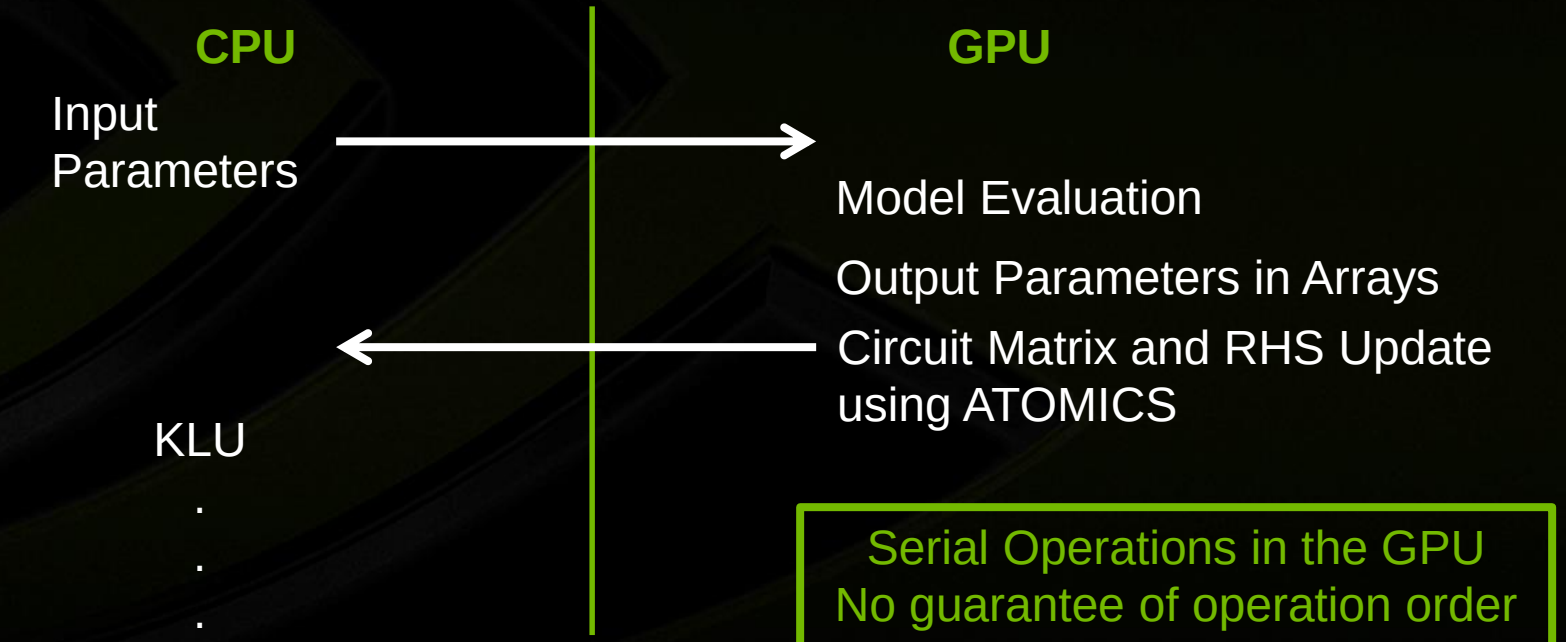


  }
}

Circuit Matrix and RHS Update

After Model Evaluation in the GPU and moving the Circuit Matrix and RHS Update on the GPU

For each Timestep {
  For each Newton-Raphson Iteration {



  }
}

Serial Operations in the GPU
No guarantee of operation order
No Coalesced Access

Topology Matrix Method

- The entire Update Process has been changed completely, using a totally new approach, that we have called **Topology Matrix Method**
- The models still write their results in an array form
- The arrays are not copied back to the CPU anymore and the Circuit Matrix and RHS are **Created** (**and not Updated**) directly on the GPU
- There are 2 Topology Matrices (Circuit Matrix and RHS)

Topology Matrix Method

- Every Topology Matrix is created upfront, during the Setup Phase, in the following way:

CSC Circuit Matrix or RHS

CSR Topology Matrix for the Circuit Matrix or the RHS

Unique Values from Device Models

$$\begin{bmatrix} A_{11} \\ A_{21} \\ \vdots \\ A_{n1} \\ \vdots \\ A_{nn} \end{bmatrix} = \begin{bmatrix} -1 & 0 & \dots & 0 & \dots & -1 \\ 0 & 0 & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & \dots & -1 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & -1 & \dots & 0 & \dots & 0 \end{bmatrix} \cdot \begin{bmatrix} Value1 \\ Value2 \\ \vdots \\ Value3 \\ \vdots \\ Value4 \end{bmatrix}$$

Rows depend on the circuit
Columns are arbitrary

Rows order has to follow the Topology Matrix columns order

Topology Matrix Method

- The Topology Matrices are extremely sparse (just 7-8 elements per row); the ground node is not included

KLU takes care of the Circuit Matrix

Just 1 bit per value is needed to code the Topology Matrix

For example, G is the only value that comes out from the Resistor Model

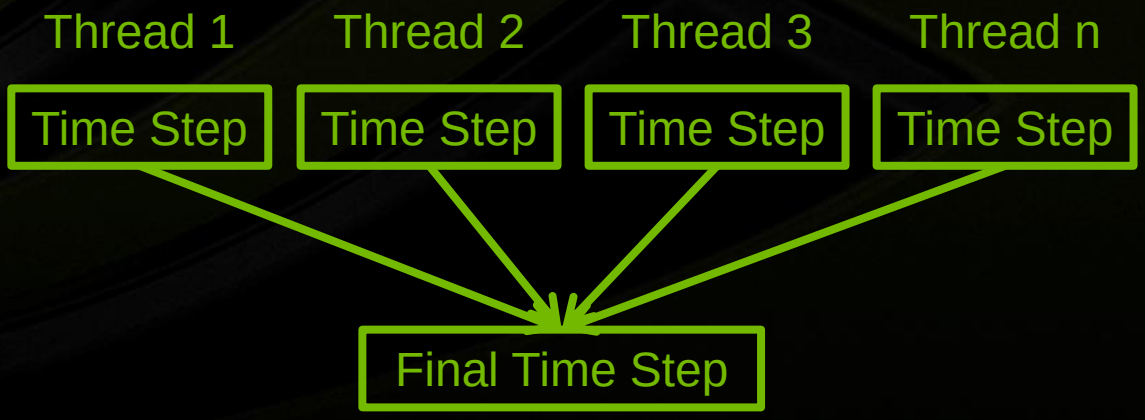
$$\begin{bmatrix} A_{11} \\ A_{21} \\ \vdots \\ A_{n1} \\ \vdots \\ A_{nn} \end{bmatrix} = \begin{bmatrix} -1 & 0 & \dots & 0 & \dots & -1 \\ 0 & 0 & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & \dots & -1 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & -1 & \dots & 0 & \dots & 0 \end{bmatrix} \cdot \begin{bmatrix} Value1 \\ Value2 \\ \vdots \\ Value3 \\ \vdots \\ Value4 \end{bmatrix}$$

DEVtopology is called for every Device Model

A Position Vector is used to easily change the values position

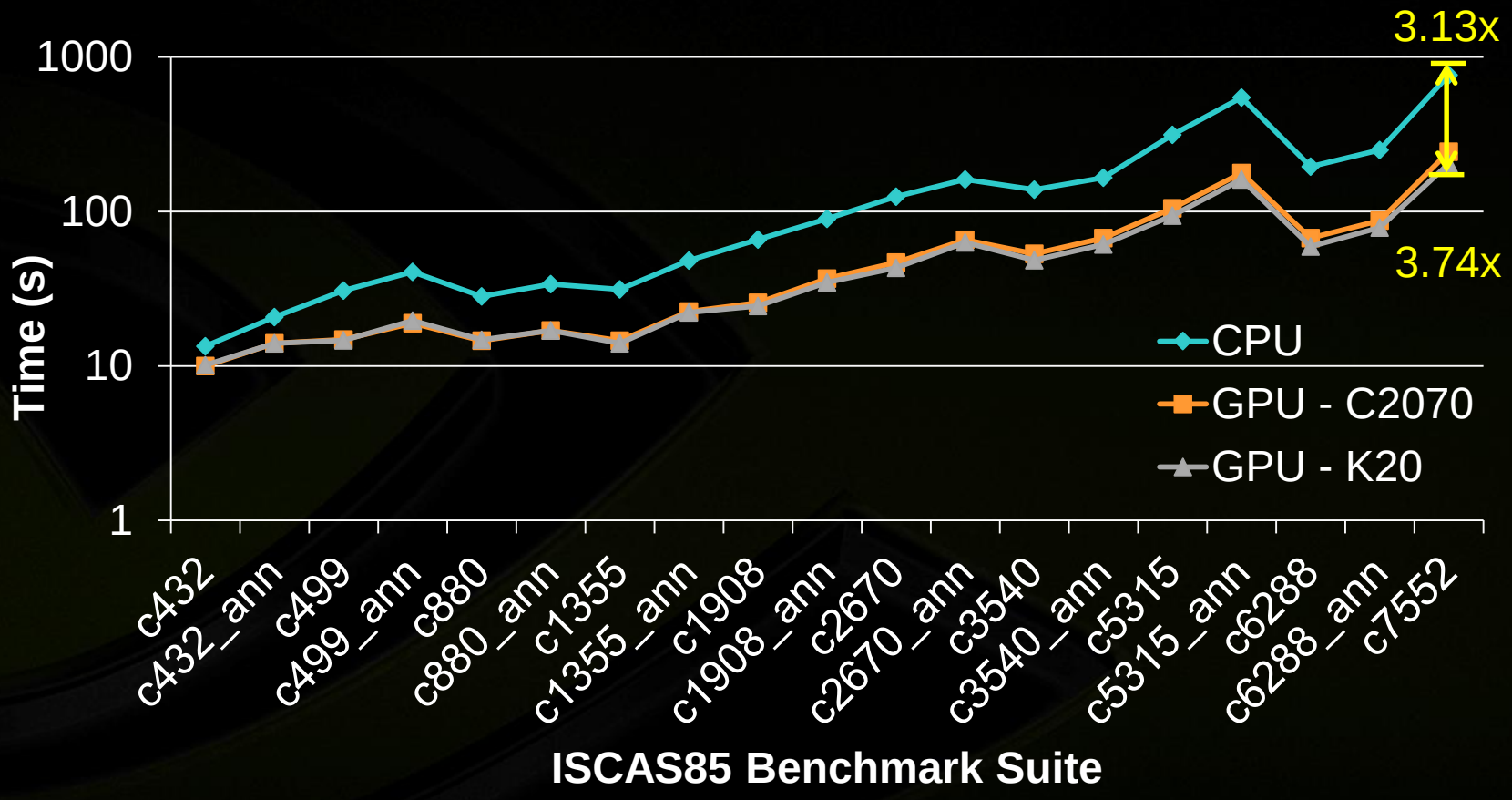
Local Truncation Error in the GPU

- In some circuits, LTE is still about the 10% of the total simulation time
- LTE is:
  - A sequence of checks to determine a new time step
  - The final new time step is the minimum of the previous ones
- LTE needs to be parallelized as well



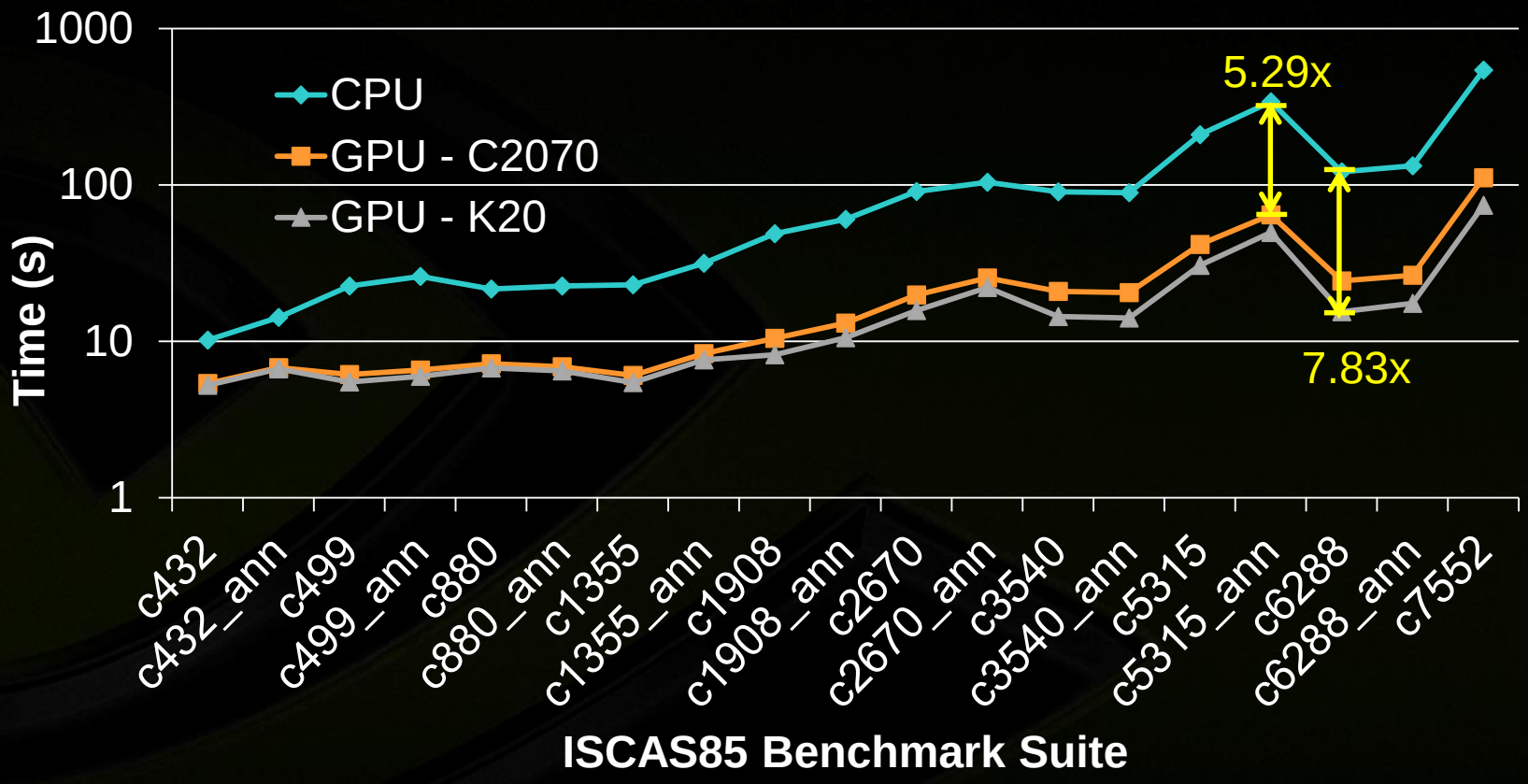
Parallel Reduction to find the minimum

Total Simulation Time without Parsing and Setup Time



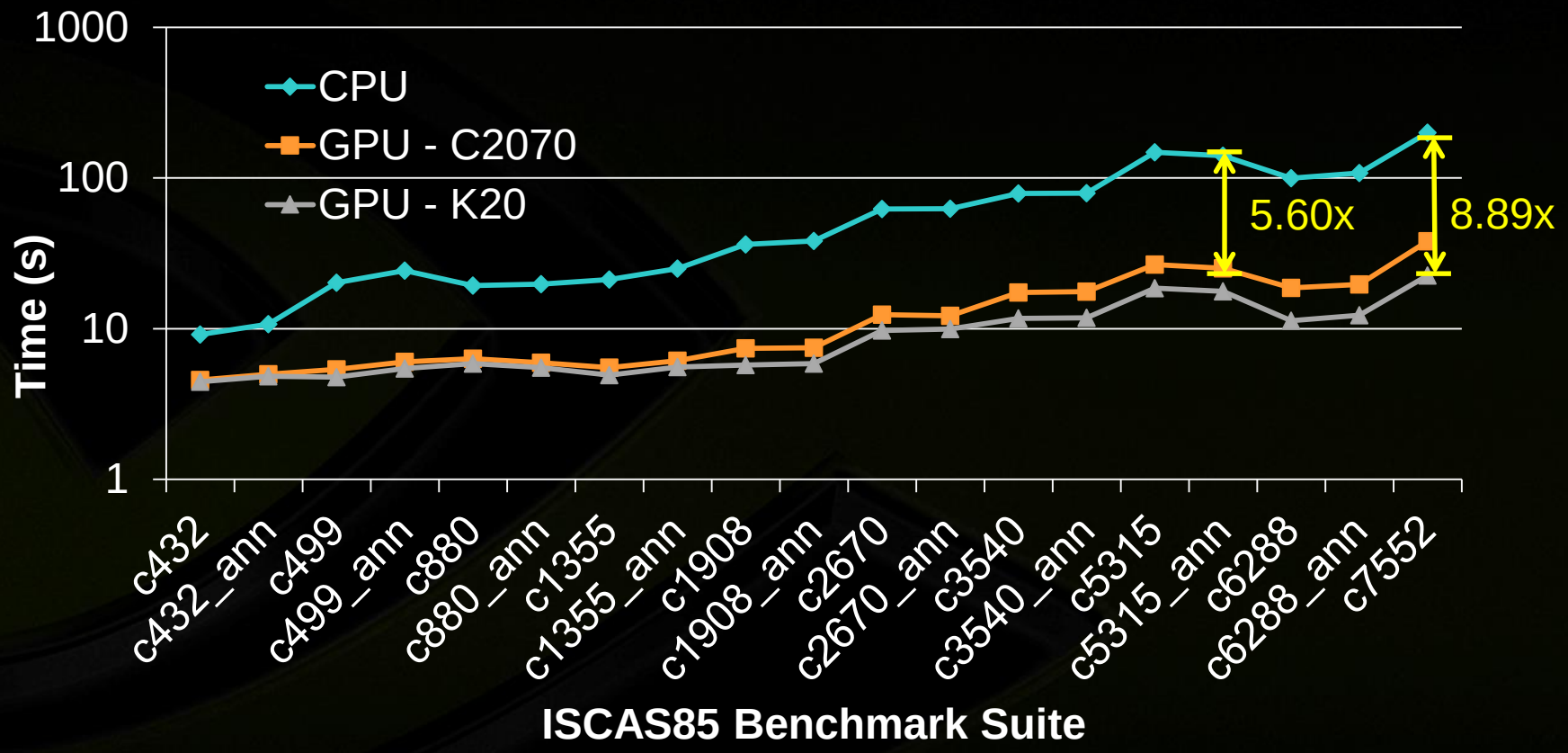
*NVIDIA C2070 and K20@600Mhz
*Intel X5690 (6 Core™) @ 3.47GHz

Total Model Evaluation Time (OP + Transient)



*NVIDIA C2070 and K20@600Mhz
*Intel X5690 (6 Core™) @ 3.47GHz

Total Model Evaluation Time in Transient Analysis



*NVIDIA C2070 and K20@600Mhz
*Intel X5690 (6 Core™) @ 3.47GHz

How to get CUSPICE

- In order to get CUSPICE, you need to:
  - 1) Clone NGSPICE GIT repository
  - 2) Execute ‘git checkout CUSPICE’
  - 3) Execute ‘./autogen.sh’
  - 4) Execute ‘./configure -enable-cuspice …’
  - 5) Execute ‘make’
  - 6) Execute ‘make install’
- You need to have NVCC installed (pick up the latest version to be sure it works)
- In addition you need to have Autotools (and maybe TCL) installed

Conclusion

- SPICE simulation has two most time consuming parts
  - Device Model Evaluation
  - Linear System Solution
- Device Model Evaluation
  - Speedup* up to 6.67x for BSIM4v7 MOSFET Model
  - Speedup* up to 35x for Resistor Model
- Entire Simulation
  - Speedup* up to 3.74x OVERALL (without Parsing and Setup Time)
  - Speedup* up to 8.89x in pure Transient Analysis

*speedup depends on the circuit

Thanks for your attention

QUESTIONS ?