Fast Circuit Simulation on Graphics Processing Units

Kanupriya Gulati†
John F. Croix‡
Sunil P. Khatri†
Rahm Shastry‡

† Texas A&M University, College Station, TX
‡ Nascentric, Inc. Austin, TX

Outline

- Introduction
- CUDA programming model
- Approach
- Experiments
- Conclusions

Introduction

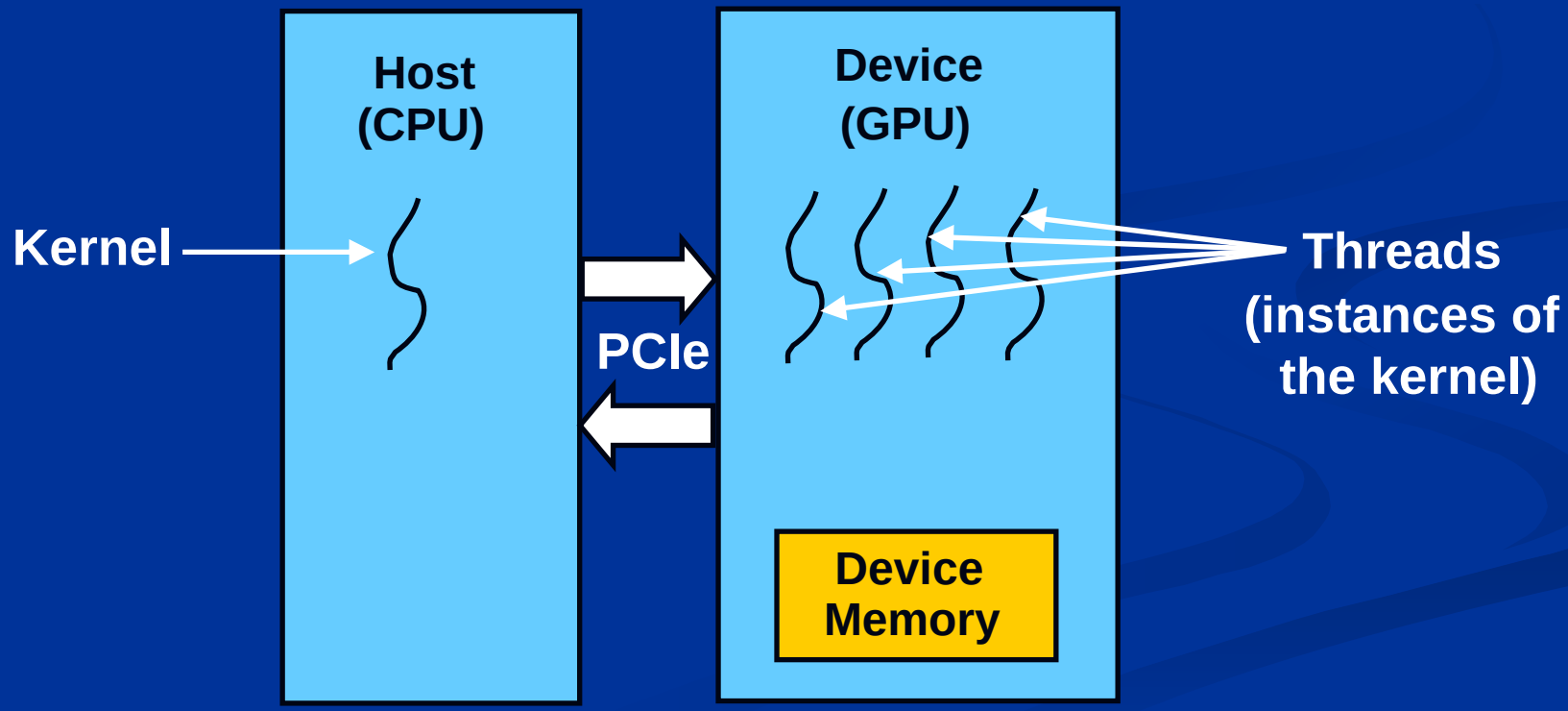
- SPICE is the de facto industry standard for VLSI circuit simulations
- Significant motivation for accelerating SPICE simulations without losing accuracy
  - Increasing complexity and size of VLSI circuits
  - Increasing impact of process variations on the electrical behavior of circuits
    - Require Monte Carlo based simulations
- We accelerate the computationally expensive portion of SPICE – **transistor model evaluation** – on Graphics Processing Units (GPUs)
- Our approach is **integrated into a commercial SPICE accelerator tool** OmegaSIM (already 10-1000x faster than traditional SPICE implementations)
- With our approach, OmegaSIM achieves a **further speedup** of **2.36X (3.07X) on average (max)**

Introduction

- GPU – a commodity stream processor
  - Highly parallel
  - Very fast
  - Single Instruction Multiple Data (SIMD) operation
- GPUs, owing to their **massively parallel architecture**, have been used to accelerate several **scientific computations**
  - Image/stream processing
  - Data compression
  - Numerical algorithms
    - LU decomposition, FFT etc
- For our implementation we used
  - NVIDIA GeForce 8800 GTS (128 processors, 16 multiprocessors)
  - Compute Unified Device Architecture (CUDA)
    - For programming and interfacing with the GPU

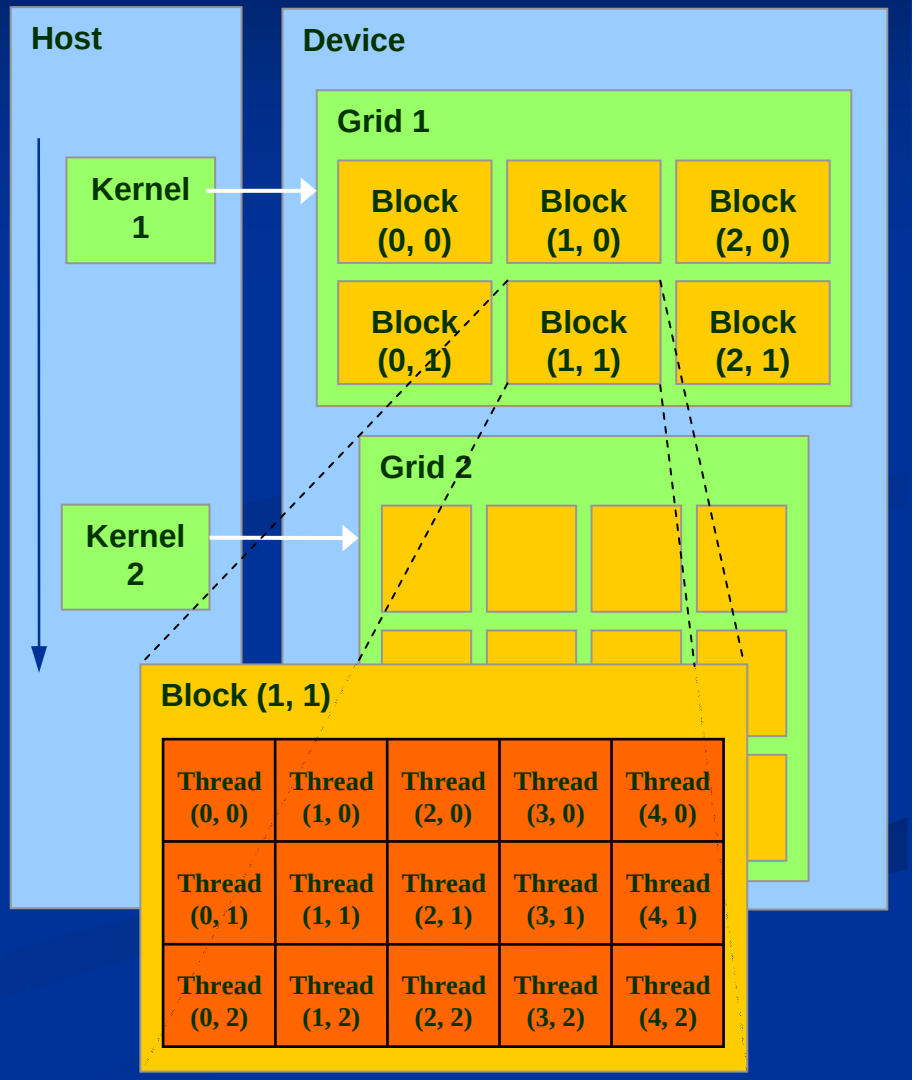
CUDA Programming Model

- The GPU is viewed as a compute **device** that:
  - Is a coprocessor to the CPU or **host**
  - Has its own DRAM (**device memory**)
  - Runs many **threads in parallel**



Thread Batching: Grids and Blocks

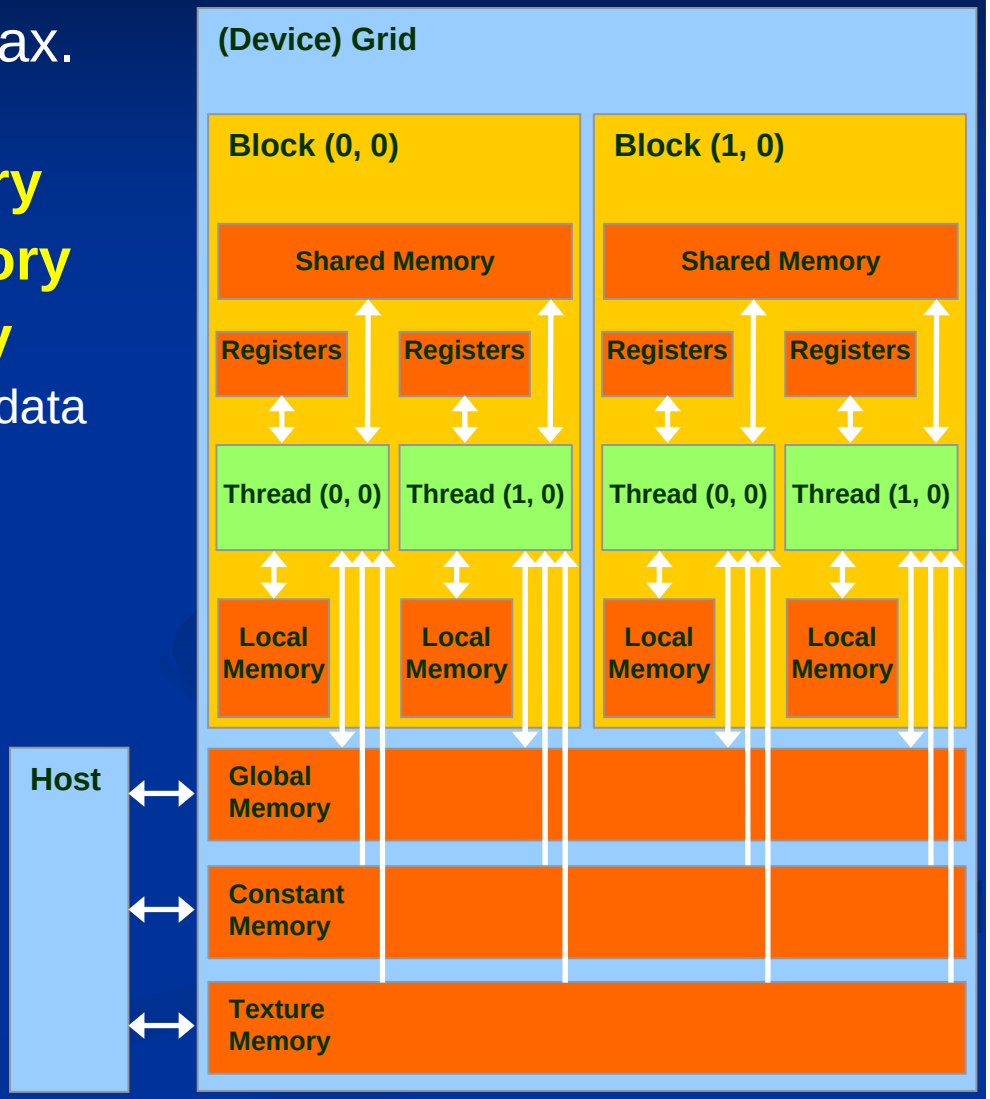
- A kernel is executed as a **grid of thread blocks (aka blocks)**
  - All threads within a block share a portion of data memory
- A **thread block** is a batch of threads that can **cooperate** with each other by:
  - Synchronizing their execution
    - For hazard-free common memory accesses
  - Efficiently sharing data through a low latency **shared memory**
- Two threads from two different blocks cannot cooperate



Source : "NVIDIA CUDA Programming Guide" version 1.1

Device Memory Space Overview

- Each thread has:
  - R/W per-thread **registers** (max. 8192 registers/MP)
  - R/W per-thread **local memory**
  - R/W per-block **shared memory**
  - R/W per-grid **global memory**
    - Main means of communicating data between host and device
    - Contents visible to all threads
    - Not cached, coalescing needed
  - Read only per-grid **constant memory**
    - Cached, visible to all threads
  - Read only per-grid **texture memory**
    - Cached, visible to all threads
- The host can R/W **global**, **constant** and **texture** memories



Source : "NVIDIA CUDA Programming Guide" version 1.1

Approach

- We first profiled SPICE simulations over several benchmarks
  - **75% of time spent in BSIM3 device model evaluations**
  - Billions of calls to device model evaluation routines
    - Every device in the circuit is evaluated for every time step
    - Possibly repeatedly until the Newton Raphson loop for solving non-linear equations converges
  - **Asymptotic speedup of 4X** considering Amdahl's law.
- These **calls are parallelizable**
  - Since they are independent of each other
  - Each call performs identical computations on different data
    - **Conform to the GPU's SIMD operating paradigm**

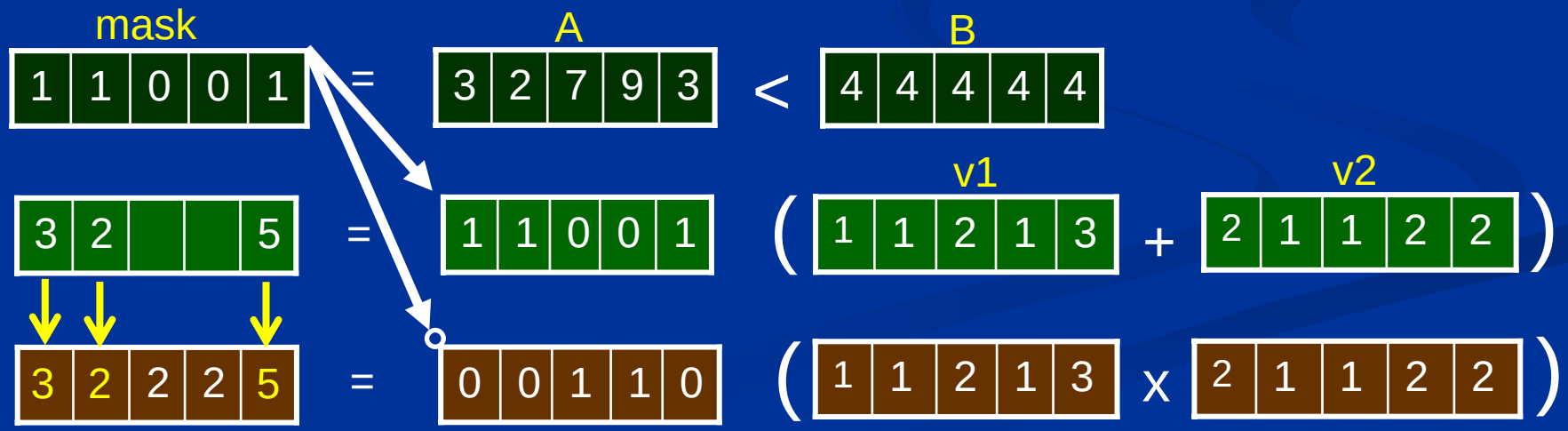
Approach

- **CDFG-guided manual partitioning** of BSIM3 evaluation code
  - **Limitation on the available hardware resources**
    - Registers (8192/per multiprocessor)
    - Shared Memory (16KB/per multiprocessor)
    - Bandwidth to global memory (max. sustainable is ~80 GB/s)
  - **If entire BSIM3 model is implemented as a single kernel**
    - Number of threads that can be issued in parallel are not enough
      - To hide global memory access latency
  - **If BSIM3 code is partitioned into many (small) kernels**
    - Requires large amounts of data transfer across kernels
      - Done using global memory (not cached)
      - Negatively impacts performance
  - So, in our approach, we:
    - **Create CDFG** of the BSIM3 equations
    - **Use maximally disconnected components** of this graph as different kernels, considering the above hardware limitations

Approach

- **Vectorizing ‘if-else’ statements**
  - BSIM3 model evaluation code has nested if-else statements
  - For a SIMD computation - they are restructured using masks
  - **CUDA compiler has inbuilt ability to restructure** these statements

```
if( A < B )
    x = v1 + v2;
else
    x = v1 * v2;
```



Approach

- Take GPU memory constraints into account
  - **Global Memory**
    - Used to **store intermediate data** – which is generated by one kernel and needed by another
      - Instead of transferring this data to host
  - **Texture Memory**
    - Used for **storing 'runtime parameters'**
      - Device parameters which remain unchanged throughout the simulation
    - Advantages
      - It is **cached,** unlike global memory
      - **No coalescing** requirements, unlike global memory
      - **No bank conflicts,** such as possible in shared memory
      - CUDA's efficient **built in texture fetching routines** are used
      - Small texture memory loading overhead is easily amortized

Experiments

- Our device model evaluation is implemented and integrated into a commercial SPICE accelerator tool – **OmegaSIM**
  - Modified version of OmegaSIM **referred to as AuSIM**
- Hardware used:
  - CPU: Intel Core 2 Quad, 2.4 GHz, 4GB RAM
  - GPU: NVIDIA GeForce 8800 GTS, 128 Processors, 675 MHz, 512 MB RAM
- **Comparing BSIM3 model evaluation alone**

# Eval.	GPU runtimes (ms)			CPU runtimes (ms)	Speedup
	Proc.	Tran.	Tot.		
1M	81.17	196.48	277.65	8975.63	**32.33X**
2M	184.91	258.79	443.7	18086.29	**40.76X**

Experiments - Complete SPICE Sim.

Ckt. Name	# Trans.	Total # Evals.	OmegaSIM (s)	AuSIM (s)	Speedup
			CPU-alone	GPU+CPU	
Industrial_1	324	1.86×10^{7}	49.96	34.06	1.47X
Industrial_2	1098	2.62×10^{9}	118.69	38.65	3.07X
Industrial_3	1098	4.30×10^{8}	725.35	281.5	2.58X
Buf_1	500	1.62×10^{7}	27.45	20.26	1.35X
Buf_2	1000	5.22×10^{7}	111.5	48.19	2.31X
Buf_3	2000	2.13×10^{8}	486.6	164.96	2.95X
ClockTree_1	1922	1.86×10^{8}	345.69	132.59	2.61X
ClockTree_2	7682	1.92×10^{8}	458.98	182.88	2.51X
Avg.					**2.36X**

- **With increase in number of transistors, speedup obtained is higher**
  - More device evaluation calls made in parallel, latencies are better hidden
- **High accuracy with single precision floating point** implementation
  - Over 1M device evals. avg. (max.) error of 2.88×10^{-26} (9.0×10^{-22}) Amp.
  - Newer devices with double precision capability already in market

Conclusions

- Significant interest in accelerating SPICE
- **75% of the SPICE runtime spent in BSIM3 model evaluation** – allows asymptotic speedup of 4X
- Our approach of accelerating model evaluation using GPUs has been **implemented and integrated with a commercial fast SPICE tool**
  - Obtained **speedup of 2.36 X** on average.
- BSIM3 model evaluation can be sped up by 30-40X over 1M-2M calls
- With a more complicated model like BSIM4
  - Model evaluation would possibly take a yet larger fraction of SPICE runtime
  - Our approach would likely provide a higher speedup
- With increase in number of transistors, a higher speedup is obtained