Distributed Time, Conservative Parallel Logic Simulation on GPUs

Bo Wang¹, Yuhao Zhu², Yangdong Deng¹

¹ Tsinghua University, ² Beihang University
Outline

- Motivation
- Background
- Parallel Logic Simulator
- Experiments
- Conclusion
Motivation

- Simulation has become a bottleneck for circuit design
  - 60~80% of design effort is now dedicated to verification[^1]
  - Example: the logic simulation of a billion-transistor design could take over one month to finish[^2]

[^1]: Chien-Nan Liu, SoC Verification Methodology, Oct 2003
[^4]: “Functional Verification on Large ASICs” by Adrian Evans, etc., 35th DAC, June 1998.
Types of simulations

- Behavioral level simulation
- Register transfer level simulation
- **Gate level simulation**
- Transistor level simulation
- ...

Outline

- Motivation
- Background
- Parallel Logic Simulator
- Experiments
- Conclusion
New platform

- General-Purpose Graphic Processing Unit (GPGPU)
GPU architecture (NVIDIA GTX280)

- 30 multi-processors, 240 streaming processors
- 1024 MB GDDR3, 141.7GB/s
- 933 GFlops
Simulation algorithms

- Oblivious algorithm
  - All gates are evaluated at each cycle
  - Simple, efficient static gate scheduling
  - Inefficient due to redundant evaluation

- Event-driven algorithm
  - A gate is simulated only if its input value changes
  - Synchronous
    - Events simulated simultaneously have the same simulation time.
  - Asynchronous
    - Events having different simulation time can be simulated simultaneously
    - Chandy-Misra-Bryant algorithm
Simulation algorithms (cnt.)

- **Chandy-Misra-Bryant algorithm**
  - Event-driven
  - Asynchronous
  - Conservative
  - Parallel and distributed

Assume the gate delay of NANDs is 1ns
Algorithm revisited

Each round
- Stimuli are fetched from primary inputs to the gate inputs
- Gate outputs are sent to the gate inputs
- Events arriving at a gate are stored in a priority queue w.r.t. timestamp
- Each gate evaluate the event with the smallest timestamp if possible
Outline

- Motivation
- Background
- Parallel Logic Simulator
- Experiments
- Conclusion
Basic simulation flow

```
while not finish
    // kernel 1: primary input update
    for each primary input(PI) do
        extract the first message in the PI queue;
        insert the message into the PI output array;
    end for each

    // kernel 2: input pin update
    for each input pin do
        insert messages from output array to input pin;
    end for each

    // kernel 3: gate evaluation
    for each gate do
        extract the earliest message from its pins;
        evaluate the message and update gate status;
        write the gate output to the output array;
    end while
```
Priority queue transformation

**Problem**: Maintenance of priority queue on GPU is inefficient

- In the original CMB algorithm, each gate stores the events arriving at all its inputs in a centralized priority queue w.r.t. the timestamp.
- Maintenance of priority queue introduces many branches, which are inefficient for SIMD-like GPU model.

**Solution**

- Divide the priority queue of a gate into multiple FIFOs w.r.t. its input pins
Dynamic memory management

Problem:
Memory demands of each pin_FIFOs are very different
  • #messages on each gate vary drastically
  • #messages on the same gate varies from time to time
  • Static memory pre-allocation is inefficient

Solution: Memory paging on GPUs
  • A memory paging mechanism is introduced for the management.
  • GPU-friendly allocate and release methods are provided.
Dynamic memory management

Maintained by CPU

- page_to_allocate: 0 5 6 4 8 13 11
- page_to_release: - - 3 - - - -

available_pages: 14 18 16 15 17 21 ...

FIFO for pin[i]

- size
- page_queue
  - head_page
  - tail_page
- head_offset
- tail_offset

release page

allocate page

main memory

- page queue: 3 20 1 - - 12
- available_pages: 14 18 16 15 17 21 ...

- main memory: 0 1 2 3 4 5 6 7 8 9 10 12 13 14 15
  - 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  - ...

- FIFO for pin[i]
### GPU/CPU co-processing

**Problem:** memory management needs CPU assistance
- Sequential execution
- Overhead of memory copy

**Solution: GPU/CPU Co-processing**
- Overlap `update_pin(GPU)` with `update page_to_release(CPU)`
- Overlap `evaluate_gate(GPU)` with `update page_to_allocate(CPU)`
- Adopt zero-copy in CUDA

<table>
<thead>
<tr>
<th>GPU execution</th>
<th>update PI</th>
<th>update pin</th>
<th>evaluate gate</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU execution</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zero copy</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| | update page_to_release | update page_to_allocate |
| | access page_to_allocate | access page_to_release |
Performance optimization

- Memory optimization
  - Coalesced access
    - AOS(Array-of-Structure) → SOA(Structure-of-Array)
  - Hierarchical memory: locality
    - Texture memory: circuit topological information
    - Constant memory: truth table

- Gate reordering
  - Reducing branches
    - Gates sharing the same inputs are closer to each other
    - Gates of the same type are closer to each other
Outline

- Motivation
- Background
- Parallel Logic Simulator
- Experiments
- Conclusion
Experiments

- Platform
  - Intel Core 2 Duo E6750 2.66 GHz
  - Memory: DDR2 4GB
  - NVIDIA GTX 280 (DDR3 1GB)

- Baseline
  - A *synchronous* event-driven simulator on single core

- Test cases
  - ITC99
  - OpenCores

<table>
<thead>
<tr>
<th>DESIGN</th>
<th>#GATES</th>
<th>#PINS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>14511</td>
<td>35184</td>
<td>AES encryption core</td>
</tr>
<tr>
<td>DES</td>
<td>61203</td>
<td>138419</td>
<td>DES3 ENCRYPTION CORE</td>
</tr>
<tr>
<td>M1</td>
<td>17700</td>
<td>42139</td>
<td>3-stage pipelined ARM core</td>
</tr>
<tr>
<td>SHA1</td>
<td>6212</td>
<td>13913</td>
<td>Secure Hashing algorithm core</td>
</tr>
<tr>
<td>R2000</td>
<td>10451</td>
<td>27927</td>
<td>MIPS 2000 CPU core</td>
</tr>
<tr>
<td>JPEG</td>
<td>117701</td>
<td>299663</td>
<td>JPEG image encoder</td>
</tr>
<tr>
<td>B18</td>
<td>78051</td>
<td>158127</td>
<td>2 Viper processors and 6 80386 processors</td>
</tr>
<tr>
<td>NOC</td>
<td>71333</td>
<td>181793</td>
<td>Network-on-Chip simulator</td>
</tr>
</tbody>
</table>
## Performance

<table>
<thead>
<tr>
<th>Design</th>
<th>Simulated cycles</th>
<th>CPU simulation time (s)</th>
<th>GPU simulation time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>42,935,000</td>
<td>109.90</td>
<td>4.45</td>
<td>24.7</td>
</tr>
<tr>
<td>DES3</td>
<td>30,730,000</td>
<td>183.11</td>
<td>4.50</td>
<td>40.7</td>
</tr>
<tr>
<td>SHA1</td>
<td>2,275,000</td>
<td>56.66</td>
<td>0.41</td>
<td>138.2</td>
</tr>
<tr>
<td>R2000</td>
<td>28,678,308</td>
<td>9.20</td>
<td>3.15</td>
<td>2.9</td>
</tr>
<tr>
<td>JPEG</td>
<td>26,132,000</td>
<td>136.33</td>
<td>43.09</td>
<td>3.2</td>
</tr>
<tr>
<td>NOC</td>
<td>1,000,000</td>
<td>5389.42</td>
<td>347.95</td>
<td>15.5</td>
</tr>
<tr>
<td>M1</td>
<td>99,998,019</td>
<td>118.48</td>
<td>22.43</td>
<td>5.3</td>
</tr>
<tr>
<td>b18</td>
<td>19,125,000</td>
<td>37.30</td>
<td>11.49</td>
<td>3.3</td>
</tr>
</tbody>
</table>

*Speedup is closed related to the stimuli density!*
Irregular distribution of events

- Irregularity
  - some pins are very hot, some are very cold

- Testcase
  - 50,000 simulation cycles with random stimuli

<table>
<thead>
<tr>
<th>Peak number of messages</th>
<th>DES3</th>
<th>R2000</th>
<th>M1</th>
<th>JPEG</th>
<th>NOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-9</td>
<td>68170</td>
<td>15747</td>
<td>24788</td>
<td>178728</td>
<td>157891</td>
</tr>
<tr>
<td>10-99</td>
<td>63895</td>
<td>11567</td>
<td>16506</td>
<td>117820</td>
<td>23297</td>
</tr>
<tr>
<td>100-999</td>
<td>3960</td>
<td>53</td>
<td>663</td>
<td>2913</td>
<td>590</td>
</tr>
<tr>
<td>1000-9999</td>
<td>2253</td>
<td>2</td>
<td>3</td>
<td>202</td>
<td>0</td>
</tr>
<tr>
<td>10000-50000</td>
<td>85</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>15</td>
</tr>
</tbody>
</table>
Outline

- Motivation
- Background
- Parallel Logic Simulator
- Experiments
- Conclusion
Conclusion

Parallel Logic Simulator on GPUs

- Developed a GPU-friendly CMB algorithm
- Designed efficient dynamic memory management
- Utilized GPU/CPU co-processing to hide overhead
- Achieved high performance

Future works

- Study the scalability on industry-strength circuits
- Apply the techniques to system and RTL simulations
THANK YOU!