GPU-based simulation of brain neuron models
Master graduation defense

Du Nguyen Hoang Anh, Computer Engineering
August 26, 2013
Outline

1. Introduction
   - Problem statement
   - Background

2. Implementation
   - CPU
   - GPU

3. Results and Discussion
   - Configuration impact
   - Tesla C2075
   - GeForce GT640
   - Tesla vs GeForce

4. Conclusions
1 Introduction
   - Problem statement
   - Background

2 Implementation
   - CPU
   - GPU

3 Results and Discussion
   - Configuration impact
   - Tesla C2075
   - GeForce GT640
   - Tesla vs GeForce

4 Conclusions
Motivation

DIGITAL BRAIN

- **CPU**: Slow processing
- **FPGA**: Not good at floating point operations
- **GPU**: HPC, support efficiently floating point operations
- **Neuron network**: DIGITAL BRAIN PLATFORMS
  - **SUPER COMPUTER**: Costly, high power consumption
  - **GPU**: HPC, support efficiently floating point operations
  - **FPGA**: Not good at floating point operations

**FLAT FORMS**
- **CPU**: Slow processing
- **SUPER COMPUTER**: Costly, high power consumption
- **GPU**: HPC, support efficiently floating point operations
- **FPGA**: Not good at floating point operations
Problem statement
Importance of Inferior Olive Cell

Diagram of the cerebellar circuit

Central nervous system
Inferior olive model (IO)

Structure of neurons

IO model

Introduction  Problem statement  Background
IO model in a network setting
Graphics Processing Units (GPUs)

- Utilize more cores to do more computations in parallel
- Support efficiently floating point operations
- Provide high bandwidth memory GDDR DRAM
- Using SIMT (Single-Instruction Multiple-Thread)
- Exploit CUDA/OpenCL programming language
  - No concern of graphical operations
  - Unified source code for both host and devices
# Nvidia GPUs: Fermi vs Kepler

## Table of Comparisons

<table>
<thead>
<tr>
<th>Fermi architecture</th>
<th>Kepler architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>released in 2010</td>
<td>released in 2012</td>
</tr>
<tr>
<td>made to increase raw power</td>
<td>dubbed as the most power efficient GPU</td>
</tr>
<tr>
<td>bridged gap between CPU and GPU</td>
<td>more cores/SM, more threads/core, more regs/thread</td>
</tr>
<tr>
<td>use SMs</td>
<td>use SMXs</td>
</tr>
<tr>
<td>single work</td>
<td>queue hyperQ</td>
</tr>
</tbody>
</table>

Use CUDA
1. Introduction
   - Problem statement
   - Background

2. Implementation
   - CPU
   - GPU

3. Results and Discussion
   - Configuration impact
   - Tesla C2075
   - GeForce GT640
   - Tesla vs GeForce

4. Conclusions
Investigation of CPU implementation

Program structure

1st For loop: t from 0 to timesteps

2nd For loop: j from 0 to IO_NETWORK_DIM1

3rd For loop: k from 0 to IO_NETWORK_DIM2

Compute cells’ parameters: V_DEND, V_SOMA, V_AXON

Print output to file

Variables

CellStatePtr
- Data type: an array of typed cellState
- Size: 2xIO_NETWORK_DIM1xIO_NETWORK_DIM2

CellCompParamsPtr
- Data type: an array of typed cellCompParams
- Size: IO_NETWORK_DIM1xIO_NETWORK_DIM2

Intensive computational
GPU implementation

Program structure

1st For loop: t from 0 to timesteps

Transfer input (iApp)
 Transfer index t

GPU kernel invoke with number of threads: IO_NETWORK_DIM1xIO_NETWORK_DIM2

Get V_DEND of 8 neighbors
Compute cells' parameters: V_DEND, V_SOMA, V_AXON

Copy data from device to host

Print output to file

Variables

cellStatePtr
- Data type: cellState
- Size: 2xIO_NETWORK_DIM1xIO_NETWORK_DIM2

dev_cellStateTmp
- Data type: array of set of 19 double
- Size: 2xIO_NETWORK_DIM1xIO_NETWORK_DIM2

Dev_cellCompParamsTmp
- Data type: Array of set of 54 double
- Size: timestepxIO_NETWORK_DIM1xIO_NETWORK_DIM2
Optimization

Program structure

1st For loop: t from 0 to timesteps

- **GPU kernel** invoke with number of threads: \(IO_{\text{NETWORK}} \times IO_{\text{NETWORK}}\)

- **Get \(V_{\text{DEND}}\) of 8 neighbors**

- **Compute cells' parameters:** \(V_{\text{DEND}}, V_{\text{SOMA}}, V_{\text{AXON}}\)

- **Copy data from device to host**

- **Print output to file**

Variables

- **\(\text{cellStatePtr}\)**
  - Data type: array of cellState
  - Size: \(2 \times IO_{\text{NETWORK}} \times IO_{\text{NETWORK}}\)

- **\(\text{dev\_cellStatePtr}\)**
  - Data type: array of set of 27 double
  - Size of array: \(IO_{\text{NETWORK}} \times IO_{\text{NETWORK}}\)

- **\(\text{dev\_iApp}\)**
  - Data type: array of double
  - Size: timesteps

- **\(\text{dev\_cellVEndPtr}\)**
  - Data type: array of double
  - Size: \(IO_{\text{NETWORK}} \times IO_{\text{NETWORK}}\)

- **\(\text{dev\_cellCompParamsPtr}\)**
  - Data type: Array of set of 54 double
  - Size of array: \(IO_{\text{NETWORK}} \times IO_{\text{NETWORK}}\)
Optimization

Program structure

- Initialize cells
  - Represented by cellStatePtr
  - Data type: array of cellState
  - Size: 2xIO_NETWORK_DIMxIO_NETWORK_DIM
- Copy data from host to device
- 1st For loop: t from 0 to timesteps
  - GPU kernel invoke with number of threads: IO_NETWORK_DIMxIO_NETWORK_DIM
  - Compute cells' parameters:
    - V_DEND, V_SOMA, V_AXON
- dev_cellStatePtr
  - Data type: array of set of 27 double
  - Size of array: IO_NETWORK_DIMxIO_NETWORK_DIM
- dev_iApp
  - Data type: array of double
  - Size: timesteps
  - Using global index and transfer iApp as whole
  - Using efficient organizing of global variables
- dev_cellVDendPtr
  - Data type: array of double
  - Size: IO_NETWORK_DIMxIO_NETWORK_DIM
  - Using separate organizing of neighbor V_DEND
  - Using texture memory
- dev_cellCompParamsPtr
  - Data type: Array of set of 54 double
  - Size of array: IO_NETWORK_DIMxIO_NETWORK_DIM
  - Using local variables
- Copy data from device to host
- The final values of DEND, SOMA, AXON
- Print output to file

Variables

- cellStatePtr
  - Data type: array of cellState
  - Size: 2xIO_NETWORK_DIMxIO_NETWORK_DIM
- dev_cellStatePtr
  - Data type: array of set of 27 double
  - Size of array: IO_NETWORK_DIMxIO_NETWORK_DIM
- dev_iApp
  - Data type: array of double
  - Size: timesteps
- dev_cellVDendPtr
  - Data type: array of double
  - Size: IO_NETWORK_DIMxIO_NETWORK_DIM
- dev_cellCompParamsPtr
  - Data type: Array of set of 54 double
  - Size of array: IO_NETWORK_DIMxIO_NETWORK_DIM
Reorganize global variables

CellState

Prev state

New state

Reworked CellState

19 variables

8 neighbor voltages

Before computing

After computing

Prev state

New state

Implementation CPU GPU Optimization
Optimization

Program structure

1. Initialize cells
2. Copy data from host to device
3. 1st For loop: t from 0 to timesteps
   - GPU kernel invoke with number of threads: IO_NETWORK_DIM1xIO_NETWORK_DIM2
   - Compute cells’ parameters:
     - V_DEND, V_SOMA, V_AXON
4. Get V_DEND of 8 neighbors
5. Copy data from device to host
6. The final values of DEND, SOMA, AXON
7. Print output to file

Variables

- **cellStatePtr**
  - Data type: array of cellState
  - Size: 2xIO_NETWORK_DIM1xIO_NETWORK_DIM2
  - Represented by

- **dev_cellStatePtr**
  - Data type: array of set of 27 double
  - Size of array: IO_NETWORK_DIM1xIO_NETWORK_DIM2

- **dev_iApp**
  - Data type: array of double
  - Size: timesteps

- **dev cellVDendPtr**
  - Data type: array of double
  - Size: IO_NETWORK_DIM1xIO_NETWORK_DIM2

- **dev cellCompParamsPtr**
  - Data type: Array of set of 54 double
  - Size of array: IO_NETWORK_DIM1xIO_NETWORK_DIM2

Implementation

Efficient thread block size
Using efficient organizing of global variables
Using separate organizing of neighbor V_DEND
Using local variables
Using texture memory
Using global index and transfer iApp as whole
Separate array for neighbor voltages

Uncoalesced global memory

DATA 0
DATA 1
DATA n

Coalesced global memory

Thread 1 2 3 4 5

n

Figure: Coalesced global memory
## Texture memory

- Reduce access time
- Eliminate branching divergence
1 Introduction
   - Problem statement
   - Background

2 Implementation
   - CPU
   - GPU

3 Results and Discussion
   - Configuration impact
   - Tesla C2075
   - GeForce GT640
   - Tesla vs GeForce

4 Conclusions
Simulation environment

- **Baseline CPU platform**
  - Intel Core i5-2450M (2.5 GHz)
  - 4 GBs of RAM

- **GPU platforms**

<table>
<thead>
<tr>
<th></th>
<th>Tesla C2075</th>
<th>GeForce GT640</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Clock rate</td>
<td>1147 MHz</td>
<td>928 MHz</td>
</tr>
<tr>
<td>CUDA Cores</td>
<td>448</td>
<td>384</td>
</tr>
<tr>
<td>Memory Clock rate</td>
<td>1566 Mhz</td>
<td>891 Mhz</td>
</tr>
<tr>
<td>Global memory</td>
<td>4096 MBytes</td>
<td>2047 MBytes</td>
</tr>
<tr>
<td>Regs/block</td>
<td>32768</td>
<td>65536</td>
</tr>
</tbody>
</table>
Impact of thread block size

Figure: Execution time of double precision on Tesla C2075
Impact of thread block size

Figure: Execution time of single precision on Tesla C2075
Impact of thread block size

Figure: Execution time of double precision on GeForce GT640
Impact of L1 cache usage

Figure: Execution time of double precision on Tesla C2075
Impact of L1 cache usage

Figure: Execution time of double precision on GeForce GT640
Performance of single precision

Results and Discussion

Configuration impact

Tesla C2075  GeForce GT640  Tesla vs GeForce
Performance of double precision

![Graph showing execution time and speed-up for different input sizes and thread counts.]

**Execution time**
- Log scale (us)
- Input size (cells)
- Execution time
- Threads/block: 32
- CPU - double

**Speed up**
- With cache - Block size 32

**Results and Discussion**
- Configuration impact
- Tesla C2075 vs GeForce GT640
- Tesla vs GeForce
Real time execution

![Graph showing execution time per time step in log scale]

<table>
<thead>
<tr>
<th>Input size (cells)</th>
<th>CPU (us)</th>
<th>GPU - Block size 32 (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>137</td>
<td>57</td>
</tr>
<tr>
<td>256</td>
<td>549</td>
<td>65</td>
</tr>
<tr>
<td>1,024</td>
<td>2,197</td>
<td>80</td>
</tr>
<tr>
<td>4,096</td>
<td>8,787</td>
<td>229</td>
</tr>
</tbody>
</table>
Performance of single precision

**Execution time**

- CPU - float
- With cache - Block size 64

**Speed up**

- With cache - Block size 64

Results and Discussion

Configuration impact

Tesla C2075  
GeForce GT640  
Tesla vs GeForce
Performance of double precision

**Execution time**

- **CPU - double**
- **With cache - Block size 64**

**Speed up**

- **With cache - Block size 64**

Results and Discussion
Configuration impact
Tesla C2075
GeForce GT640
Tesla vs GeForce
Real time execution

Execution time per time step in log scale

<table>
<thead>
<tr>
<th>Input size (cells)</th>
<th>CPU (us)</th>
<th>GPU - Block size 64 (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>128</td>
<td>136</td>
</tr>
<tr>
<td>256</td>
<td>511</td>
<td>136</td>
</tr>
<tr>
<td>1,024</td>
<td>2,025</td>
<td>181</td>
</tr>
</tbody>
</table>
Performance comparison

![Graph showing speed-up comparison between different GPU configurations](image)

- **Tesla - Single precision - Block size 64**
- **Tesla - Double precision - Block size 32**
- **GeForce - Single precision - Block size 64**
- **GeForce - Double precision - Block size 64**
## Cost efficiency

<table>
<thead>
<tr>
<th></th>
<th>Tesla C2075</th>
<th>GeForce GT640</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Double precision</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price (US dollars)</td>
<td>1789</td>
<td>79</td>
</tr>
<tr>
<td>Best achieved speed-up</td>
<td>68.1</td>
<td>21.1</td>
</tr>
<tr>
<td>Cost efficiency</td>
<td>26.3</td>
<td>3.7</td>
</tr>
<tr>
<td><strong>Single precision</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price (US dollars)</td>
<td>1789</td>
<td>79</td>
</tr>
<tr>
<td>Best achieved speed-up</td>
<td>99.4</td>
<td>34.1</td>
</tr>
<tr>
<td>Cost efficiency</td>
<td>18</td>
<td>2.3</td>
</tr>
</tbody>
</table>
1 Introduction
   • Problem statement
   • Background

2 Implementation
   • CPU
   • GPU

3 Results and Discussion
   • Configuration impact
   • Tesla C2075
   • GeForce GT640
   • Tesla vs GeForce

4 Conclusions
Conclusions

- Successful implementation of the model on the GPU platforms
- Thread block size and L1 cache support affect performance
- Performance:
  - Tesla C2075 is 67% higher than GeForce GT640
- Cost efficiency:
  - GeForce GT640 is 7.1 times higher than Tesla C2075
- Real time simulation: up to 256 cells
- Upper bound of the number of cells: 18,939,904 cells (double precision on Tesla C2075)
Limitations and Recommendations

- **Limitations:**
  - The number of registers per SM
  - Synchronization requirement

- **Recommendations:**
  - Resolve the lack of registers by:
    - Redistribute local memory to shared and global memory
    - Rework on the number of variables in the model
    - Use compiling option to limit the number of registers used
  - "Smarter" scheme of communication among cells