Simulating GPGPUs
ESESC Tutorial

Speaker: Alamelu Sankaranarayanan
Outline

• Background
• GPU Emulation Setup
• GPU Simulation Setup
• Running a GPGPU application
The Landscape Today

• *Heterogeneous Computing*: an alternate Paradigm
• GPUs are being increasingly used to augment CPU cores
  • Popularity of programming languages like CUDA / OpenCL
  • Application in Computer Vision & Image Processing, Augmented reality, Big Data, Machine Learning, etc.
The Landscape Today

• More computational capability with each new GPU
  • Increasing processing elements with each new generation

• Tighter coupling of the CPU and GPU
  • AMD’s APUs, HSA

• Mobile / Embedded applications
  • Emphasis on energy efficiency

• Newer processor architectures like Knights Corner
Expectations from a simulator

- More computational capability with each new GPU
  - Increasing processing elements with each new generation
- Tighter coupling of the CPU and GPU
  - AMD’s APUs, HSA
- Mobile / Embedded
  - Emphasis on energy efficiency
- Newer processor architectures like Knights Corner
  - More PEs → More threads → Longer Simulation Times
    - FAST simulators needed!
  - Ability to easily vary the architectural specifications like number of PEs, memory subsystem configuration, Allowable threads, Divergence mechanisms etc.
Expectations from a simulator

• More computational capability with each new GPU
  • Increasing processing elements with each new generation

• Tighter coupling of the CPU and GPU
  • AMD’s APUs, HSA
  • Ability to model a heterogeneous system with both CPUs and GPUs

• Mobile / Embedded applications
  • Emphasis on energy efficiency

• Newer processor architectures like Knights Corner
Expectations from a simulator

• More computational capability with each new GPU
  • Increasing processing elements with each new generation

• Tighter coupling of the CPU and GPU
  • AMD’s APUs, HSA

• Mobile / Embedded applications
  • Emphasis on energy efficiency

• Newer processor architectures like Knights Corner

• Integrated Power Model
  • Thermal?
Expectations from a simulator

• More computational capability with each new GPU
  • Increasing processing elements with each new generation

• Tighter coupling of the CPU and GPU
  • AMD’s APUs, HSA

• Mobile / Embedded applications
  • Emphasis on energy efficiency

• Newer processor architectures like Knights Corner
  • Flexibility in architectural description
  • Ease of extension
## Available GPGPU Simulators

<table>
<thead>
<tr>
<th>GPGPU Simulators</th>
<th>Key Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPGPU Simulation</td>
<td>Most Popular, Can model Fermi like architectures.</td>
</tr>
<tr>
<td>Multi2Sim</td>
<td>Heterogenous simulator, capable of simulating both OpenMP and OpenCL threads.</td>
</tr>
<tr>
<td>GPU Wattch</td>
<td>Power model for GPGPUs. Now integrated with GPGPU Simulation.</td>
</tr>
<tr>
<td>GPU Sim Pow</td>
<td>Another Power Model, based on GPGPU Simulation.</td>
</tr>
<tr>
<td>Ocelot</td>
<td>Dynamic JIT compilation framework translating PTX to run on several backends</td>
</tr>
</tbody>
</table>
Simulating GPGPUs

Generic Simulators

- Simulator
  - IPC
  - Cache hit & miss rates
  - Timing Model

- Interface
  - Translate the trace to an IR
  - Manage feeding the trace to the simulator

- Emulator
  - Power Model
  - Power Estimates

- Application Binary
  - TRACE
Simulating GPGPUs

- **Simulator**
  - Timing Model
    - IPC
    - Cache hit & miss rates
  - Power Model
    - Power Estimates

- **Emulator**
  - Interface
    - Generate a trace and translate to IR
    - Interpret assembly and model the GPU Behavior
  - GPU Binary
  - Application assembly Code

- **SLOW!**
How can we make it faster?

Simulator

- IPC
- Cache hit & miss rates

Timing Model

Emulator

- Generate a trace and translate to IR
- Interpreta assembly and model the GPU Behavior

Emulator

Memory

TRACE

Modified GPU Binary

Run it natively on a GPU

Pre-interpret the assembly code and generate translated IR, save more time

Interface

Power Estimates

Power Model

Pre-interpret the assembly code and generate translated IR, save more time
Simulating GPGPUs with ESESC

Simulating GPGPUs

Alamelu S

ESESC

Simulating GPGPUs with ESESC

Simulator

- IPC
- Cache hit & miss rates

Timing Model

Emulator

- Modified CUDA Binary
- Native Co-execution

Memory TRACE

Interface

- Generate the trace for the timing model
- Read the pre-translated PTX informations

Power Estimates

Power Model
Creating modified binaries

• Purpose
  • Avoid mock GPU execution of the application by the emulator (needed for memory addresses)
  • Generate a trace with the memory addresses, per thread.
  • Exploit the computational power of the GPGPU, to speed up simulation.

• Original application behavior should remain unchanged
Creating modified binaries

• Challenges
  • How can we effectively return the memory addresses per thread?
  • How can we convey the execution path of different threads? (threads can diverge)
  • How can we pass the control back and forth between the CPU and the GPU?
Creating modified binaries

"Contaminated" PTX code

1. Load the Live In data (Restore State)
2. Save the current BBID

1. Save the memory address after each Mem operation

1. Save the Live out data (Save State)
2. Save the next BBID
3. Return control back to the CPU (exit)
Creating modified binaries

“Contaminated” PTX code

CUDA Application Assembly (PTX code)

Use this “Contaminated” PTX code to create the modified application binary.
Contaminated PTX

Simulating GPGPUs

Alamelu S

ESESC
1. Save the Live out data (Save State)
2. Save the current BBID

1. Load the Live In data (Restore State)
2. Save the current BBID
3. Return control back to the CPU (exit)
Simulating GPGPUs

Kernel Name

Trace Statistics

Divergence Information

Pre-translated *.info file
Simulating a GPGPU

Simulator
- Timing Model
  - IPC
  - Cache hit & miss rates

Emulator
- Interface
  - Generate the trace for the timing model
- Power Model
  - Power Estimates
- Memory
  - TRACE

Contaminated CUDA Binary
- Native Co-execution

Read the pre-translated PTX informations

Simulating GPGPUs
Alamelu S
### Trace Generation

<table>
<thead>
<tr>
<th></th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current BBID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Next BBID</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Memory Addresses</td>
<td><img src="image" alt="Memory Addresses" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Done?</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**GPU Timing Model**

- [T0-BB1-](image)  
- [T1-BB1-](image)  
- [T2-BB1-](image)  
- [T3-BB1-](image)  
- [T4-BB1-](image)  
- [T5-BB1-](image)  
- [T6-BB1-](image)  
- [T7-BB1-](image)  

**GPU Emulator**

**GPGPU Hardware**
## Trace Generation

### GPU Timing Model

<table>
<thead>
<tr>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

### Next BBID

| 4   | 4   | 4   | 4   | 4   | 4   | 4   | 4   |

### Memory Addresses

- T0-BB2-
- T1-BB2-
- T2-BB3-
- T3-BB3-
- T4-BB2-
- T5-BB2-
- T6-BB3-
- T7-BB3-

### Done?

| 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

---

**Relaunch**

**Return**
Trace Generation

<table>
<thead>
<tr>
<th>Current BBID</th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

| Next BBID    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

<table>
<thead>
<tr>
<th>Memory Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

| Done? | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

GPU Timing Model

Application Complete

GPU Emulator

GPGPU Hardware

Relaunch

Return

Simulating GPGPUs

Alamelu S
Simulating GPGPUs

A Modern GPGPU

Thread

Per Thread Local Memory

Thread Block

Per-Block Shared Memory

Grid 0

Global Memory

Grid 1

Block (0,0)  Block (1,0)

Block (0,1)  Block (1,1)
A Modern GPGPU

Simulating GPGPUs

Alamelu S
Timing Model

• Each SM is modeled as a group of little cores (lanes)
  • Based on the in-order core modeled in ESESC
  • Each lane can be configured to have the same capabilities as a regular in-order core.

• Graphic specific blocks (rasterizer, clipping) are not modeled
Simulating GPGPUs

Timing Model

• The trace generator / manager for ESESC models
  • Barriers
  • Execution strategies
  • Divergence mechanisms
    • Serial execution
    • Post Dominator convergence [1]
    • Simultaneous Branch Interleaving [2]


Simulating GPGPUs
Alamelu S
Timing Model

- Memory Hierarchy is defined and used just as for CPU simulations
  - Extensions to indicate if an address is a shared or global address
  - Extensions to indicate which thread or warp a memory address belongs
Simulating GPGPUs

Software architecture

Modified Binary

- InstDoctor to contaminate PTX
- Custom compilation flow using NVCC

Interface

- GPUInterface
- Modifications to QEMU

Trace Mgmt

- GPUPThreadManager
- GPUEmulInterface

Timing/Power Model

- GPUSMProcessor
- gpu.cpp
- Existing ESESC infrastructure
Software architecture

Simulating GPGPUs

Alamelu S
Simulating GPGPUs

Software architecture

GPUSMPProcessor

Modified Binary

Emulator

Interface

Trace Generation

Lane 0
Lane 1
Lane 31

Coalescing

Scratch Pad

DL1

SM3
SM2
SM1
SM0

SM3
SM2
SM1
SM0

Register File

L2

Cache

To lower levels
Software architecture

Simulating GPGPUs

Alamelu S
Simulating GPGPUs

Alamelu S

Software architecture

GPUSMProcessor

Modified Binary

GPUInterface

GPUEmulInterface

Emulator

Interface

Emulator

Trace Generation

gpu.cpp

GPUThreadManager

To lower levels

Cache

Power Model

Simulating GPGPUs
Running a GPGPU application

Simulating GPGPUs

---

1. System requirements
   - A desktop with a GPGPU
   - CUDA version 3.2 installed
   - Last tested with driver version 304.51
   - All other packages needed by ESESC
   - An ARM machine to compile your own contaminated binary (not needed at the moment, since pre-built binaries will be provided)

2. Running a GPGPU application

   ```
   > nvidia-smi
   Tue Jun 10 06:53:20 2014
   +-----------------------------------------------------------------------------+
   | NVIDIA-SMI 4.304.51  Driver Version: 304.51                               |
   +-----------------------------------------------------------------------------+
   | GPU  Name                        | Bus-Id       Disp. | Volatile Uncorr. ECC | GPU-Util | Compute M. |
   | Fan Temp Perf Pwr:Usage/Cap | Memory-Usage | GPU-Util | Compute M. |
   +-----------------------------------------------------------------------------+
   | 0  GeForce GTX 480 | 0000:01:00.0 | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | Default |
   +-----------------------------------------------------------------------------+
   | Compute processes: | GPU Memory |
   | GPU PID Process name Usage | |
   +-----------------------------------------------------------------------------+
   | 0 Not Supported |
   +-----------------------------------------------------------------------------+
   ```
Running a GPGPU application

• Step 1: Creating a contaminated binary
  • Code cleanup in progress, detailed instructions will be made available soon after.
  • A few contaminated binaries will be provided for now.
Running a GPGPU application

• Step 2: Compiling esesc.
  • Need two additional flags
    • Enable 32 bit mode
    • Enable GPU mode (link with CUDA libraries)
  • Command to build in Release Mode

```
> cmake
-DCMAKE_HOST_ARCH=i386
-DCMAKE_BUILD_TYPE=Release
-DENABLE_CUDA=1
~/projs/esesc
```
Running a GPGPU application

• Step 3: Configure esesc.conf

```bash
# Select simulated core type. Defined in simu.conf
coreType = 'tradCORE'
#coreType = 'scooreCORE'
SMcoreType = 'gpuCORE'

# Sampling mode
samplerSel = "TASS"
gpusampler = "GPUSpacialMode"

# Set the correct number of processors
cpuemul[0] = 'QEMUSectionCPU'
cpuemul[1:4] = 'QEMUSectionGPU'

cpusimu[0] = "$(coreType)"
cpusimu[1:4] = "$(SMcoreType)"

SP_PER_SM = 32
```

NOTE! New coretype for GPGPU

NOTE! Sampling?

NOTE! Section where additional GPU parameters are specified

NOTE! Number of SMs

NOTE! Number of Lanes
Running a GPGPU application

• Step 3 : Configure esesc.conf

```
benchName = "-s 8192000 kernels/bfs kernels/graph4096.txt"
infofile = "kernels/bfs.info"
reportFile = 'gpu_bfs'
MAXTHREADS = 1024

enablePower = true

[GPUSpacialMode]
type = "GPUSpacial"
nMaxThreads = $(MAXTHREADS)
nInstSkip = 0
nInstMax = 1e14
```

NOTE! Pre-translated PTX

NOTE! Special Sampler for GPU

NOTE! Selective execution of threads
Sampling, for GPGPUs?

• GPGPU applications are largely homogeneous.

• Do we need to execute and simulate all the threads?

• Use “MAXTHREADS” to simulate the first “$(MAXTHREADS)$” threads.
  • The others are executed natively on hardware (for correct execution).

• Extract significant speedup!
  • Need to profile applications to see how much we can skip simulating.
Running a GPGPU application

- Step 4: Configure simu.conf (if needed)

```
[gpuCORE]
sp_per_sm = $(SP_PER_SM)  # needed to instantiate the GPU SM
areaFactor = 2  # Area in relation with alpha264 EV6
issueWrongPath = false
fetchWidth = $(SP_PER_SM)
instQueueSize = $(SP_PER_SM)*2
inorder = true
throttlingRatio = 2.0
issueWidth = $(SP_PER_SM)
retireWidth = $(SP_PER_SM)
decodeDelay = 3*2
renameDelay = 2*2
```
Running a GPGPU application

- Step 4: Configure simu.conf (if needed)
Running a GPGPU application
Simulating GPGPUs

Alamelu S

---

Sample Report

---

```
Sampler 1 (Proc 1 2 3 4) (24.441 million Rabit, 0.825 million Timsion, 25.266 million Timsion Instructions)

Rabite: Wramup Detail Timing Total KIPS

KIPS 158031 N/A 158494 158321 0.5 0.5 1.5 92.5 Sim Time (s) 0.113 Exec 0.035 ms Sim (1700MHz)

Inst 96.7 0.0 0.0 3.3 Approx Total Time 0.000 ms Sim (1700MHz)

1 : 066572.756 : nottakenenhanced : 58.18 : 0.06 0.60 : 58.40 : (0.06% of 0.60) : 58.71 : (0.06% of 0.53) : 6.00
2 : 527051.282 : nottakenenhanced : 58.78 : 0.06 0.60 : 58.84 : (0.06% of 0.53) : 6.00
3 : 537277.718 : nottakenenhanced : 58.21 : 0.06 0.60 : 58.47 : (0.06% of 0.53) : 6.00
4 : 572807.883 : nottakenenhanced : 58.39 : 0.06 0.60 : 58.65 : (0.06% of 0.53) : 6.00

1 : 205292 : 205117 : 205118 : 58.39 : 13.14 : 6.44 : 9.84 : 7.19 : 0.06 : N/A : GUNIT ALU 0.01
2 : 207631 : 207655 : 207657 : 58.26 : 13.07 : 6.45 : 9.95 : 7.26 : 0.06 : N/A : GUNIT ALU 0.01
3 : 204697 : 204792 : 204793 : 58.43 : 13.15 : 6.48 : 9.82 : 7.15 : 0.06 : N/A : GUNIT ALU 0.00
4 : 206928 : 206753 : 206754 : 58.54 : 13.14 : 6.40 : 9.91 : 7.10 : 0.06 : N/A : GUNIT ALU 0.01

1 : 11.49 : 3.55 : 0.00 : 57725 : 11.0 : 0.0 : 0.0 : 7.8 : 0.0 : 0.0 : 0.0 : 361322.9 : 0.0 : 0.0
2 : 08.60 : 3.52 : 0.00 : 59824 : 11.0 : 0.0 : 0.0 : 8.6 : 0.0 : 0.0 : 0.0 : 361325.0 : 0.0 : 0.0
3 : 08.50 : 3.57 : 0.00 : 57392 : 11.0 : 0.0 : 0.0 : 8.3 : 0.0 : 0.0 : 0.0 : 339551.7 : 0.0 : 0.0
4 : 08.50 : 3.61 : 0.00 : 57000 : 11.0 : 0.0 : 0.0 : 6.6 : 0.0 : 0.0 : 0.0 : 355672.1 : 0.0 : 0.0

Cache : Occ : AvgMemLat : MemAccesses : MissRate : HIU : WR : BUS : Dyn_Pow (mW) : Lng_Pow (mW)
IL1(1) : 0.0 4.2 43305 : 0.64 : (100.0% of 0.64) : 0.0
IL1(2) : 0.0 4.2 43304 : 0.64 : (100.0% of 0.64) : 0.0
IL1(3) : 0.0 4.2 43304 : 0.64 : (100.0% of 0.64) : 0.0
IL1(4) : 0.0 4.2 43304 : 0.64 : (100.0% of 0.64) : 0.0

DL1(1) : 0.0 47.4 24683 : 20.48 : (70.2% of 34.8) : 0.0
DL1(2) : 0.0 46.3 23549 : 20.95 : (70.2% of 33.5) : 0.0
DL1(3) : 0.0 46.7 24321 : 20.04 : (70.2% of 34.8) : 0.0
DL1(4) : 0.0 51.2 25023 : 20.33 : (70.2% of 34.8) : 0.0

L2(0) : 0.0 54.3 22681 : 18.30 : (51.9% of 43.9) : 0.0

niceCache(0) : 0.0 0.0 0.0 : (100.0% of 0.0) : 0.0

**GPU Power Metrics:** (Dynamic Power, Leakage Power)

<table>
<thead>
<tr>
<th>Proc</th>
<th>RF (mW)</th>
<th>RCR (mW)</th>
<th>Fetch (mW)</th>
<th>FEK (mW)</th>
<th>RNU (mW)</th>
<th>LSU (mW)</th>
<th>Total (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**GPU Power Metrics:** (Dynamic Power, Leakage Power)

<table>
<thead>
<tr>
<th>SMID</th>
<th>RF (mW)</th>
<th>ExeUs (mW)</th>
<th>IL1G (mW)</th>
<th>DL1G (mW)</th>
<th>DL1BG (mW)</th>
<th>ScratchP (mW)</th>
<th>Total (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>316</td>
<td>452</td>
<td>53</td>
<td>0</td>
<td>3579</td>
<td>0</td>
<td>4.76</td>
</tr>
<tr>
<td>2</td>
<td>646</td>
<td>488</td>
<td>53</td>
<td>0</td>
<td>3618</td>
<td>0</td>
<td>4.86</td>
</tr>
<tr>
<td>3</td>
<td>316</td>
<td>453</td>
<td>53</td>
<td>0</td>
<td>3518</td>
<td>0</td>
<td>4.78</td>
</tr>
<tr>
<td>4</td>
<td>639</td>
<td>456</td>
<td>54</td>
<td>0</td>
<td>3608</td>
<td>0</td>
<td>4.80</td>
</tr>
</tbody>
</table>

**L2 Power:** 3 (Dyn: 0 Lkg)

**L3 Power:** 6 (Dyn: 0 Lkg)

Total GPU Power = 10.13 (Dyn) 0.00 (Lkg)
```
Roadmap

- Still in an early stage.
  - Code cleanup
  - Update the compilation flow to more recent versions of CUDA
    - Add support for newer features released with newer CUDA versions.

- Validation
  - Performance
  - Power
Summary

• ESESC provides a fully customizable platform to model GPGPUs

• One of the key differentiators is the enormous speedups we achieve with techniques like native co-execution and selective thread execution

• Integrated timing and power model

• Very early stages, but expect to release a stable version in the coming months.
ESESC Mailing List
esesc@googlegroups.com
GPU Specific questions
alamelu <at> soe <dot> ucsc <dot> edu

Questions?
Acknowledgements

• Dr José Luis Briz Velasco

  Profesor Titular
  Associate Professor Computer Architecture and Technology Depto. de Informática e Ingeniería de Sistemas (DIIS) Escuela de Ingeniería y Arquitectura - University of Zaragoza (UZ)
  briz@unizar.es

• Dr Ehsan K. Ardestani
  ehsanardestani@gmail.com
Backup Slides
<table>
<thead>
<tr>
<th>GPGPU Simulators</th>
<th>Slowdown compared to Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPGPUSim [2013]</td>
<td>90000 (1350s)[1]</td>
</tr>
<tr>
<td>Multi2Sim</td>
<td>8700 (functional)</td>
</tr>
<tr>
<td></td>
<td>44000 (arch simulation)[1]</td>
</tr>
</tbody>
</table>
## Backup 2: List of available contaminated benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Benchmark Suite</th>
<th>#Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKPROP</td>
<td></td>
<td>1048576</td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td>1000000</td>
</tr>
<tr>
<td>CFD</td>
<td></td>
<td>97152</td>
</tr>
<tr>
<td>HOTSPOT</td>
<td></td>
<td>1893376</td>
</tr>
<tr>
<td>KMEANS</td>
<td></td>
<td>495616</td>
</tr>
<tr>
<td>LEUKOCYTE</td>
<td></td>
<td>104296</td>
</tr>
</tbody>
</table>
