RUNTIME SUPPORT FOR ADAPTIVE SPATIAL PARTITIONING AND INTER-KERNEL COMMUNICATION ON GPUs

Yash Ukidave\textsuperscript{\dag}, Perhaad Mistry\textsuperscript{§}, Charu Kalra\textsuperscript{\dag}, Dana Schaa\textsuperscript{§} and David Kaeli\textsuperscript{\dag}

\textsuperscript{\dag}Department of Electrical and Computer Engineering
Northeastern University, Boston, USA

\textsuperscript{§}Advanced Micro Devices (AMD), USA

SBAC-PAD 2014
Paris, France
22\textsuperscript{nd} October, 2014
WHAT IS THIS TALK ABOUT?

- Improving concurrent kernel execution through adaptive spatial partitioning of compute units on GPUs
- Implementing a pipe based memory object for inter-kernel communication on GPUs
TOPICS

- Introduction
- Background & Motivation
- GPU workgroup scheduling mechanism
- Adaptive spatial partitioning
- Pipe based communication channel
- Evaluation methodology and benchmarks
- Performance results
- Conclusion
- Future work
INTRODUCTION

- GPUs have become the most popular accelerator device in recent years.
- Applications from scientific and high performance computing (HPC) domains have reported impressive performance gains.
- Modern GPU workloads possess multiple kernels and varying degrees of parallelism.
- Applications demand concurrent execution and flexible scheduling.
- Dynamic resource allocation is required for efficient sharing of compute resources.
- Kernels should adapt at runtime to allow other kernels to execute concurrently.
- Effective inter-kernel communication is needed between concurrent kernels.
BACKGROUND & MOTIVATION

Concurrent Kernel Execution on current GPUs

- NVIDIA Fermi GPUs support concurrent execution using a “left-over policy”
- NVIDIA Kepler GPUs use Hyper-Q technology with multiple hardware queues for scheduling multiple kernels
- AMD uses Asynchronous Compute Engines (ACE) units to manage multiple kernels
- ACE units allow for interleaved kernel execution

- Concurrency is limited by the number of available CUs and leads to a fixed partition

- We implement adaptive spatial partitioning to dynamically change CU allocation
  - Kernels adapt their hardware usage to accommodate other concurrent kernels
  - A novel workgroup scheduler and partition handler is implemented
  - Our scheme improves GPU utilization and avoids resource starvation by smaller kernels
Inter-Kernel Communication on GPUs

- Stage-based computations gaining popularity on GPUs (e.g., audio and video processing)
- Applications require inter-kernel communication between stages
- Real-time communication between executing kernels is not supported on GPUs
- The definition of a “pipe object” was introduced in the OpenCL 2.0 spec
- We have implemented a pipe channel for inter-kernel communication
**RELATED WORK**

- Gregg et al. examine kernel concurrency using concatenated kernel approach [USENIX 2012]

- Tanasic et al. introduce preemption support on GPUs with a flexible CU allocation for kernels [ISCA 2014]

- Lustig et al. propose memory design improvements for computation and communication overlap for inter-kernel communication [HPCA 2013]

- Boyer et al. demonstrate dynamic load balancing of computation shared between GPUs [Computing Frontiers 2013]
WORKGROUP SCHEDULING FOR FIXED PARTITION

- OpenCL **sub-devices API** creates sub-device with a fixed number of CUs

- Multiple command queues (CQ) are mapped to different sub-devices
  - NDRange computation is launched on different sub-device through the CQs
  - NDRange launches on one sub-device use CUs assigned to that sub-device

- Sub-device maintains the following information:
  - Number of “mapped” CQ
  - Number of NDRange launches on each CQ
  - Number of compute units allocated to each sub-device
ADAPTIVE SPATIAL PARTITIONING

- Fixed partitions lead to starvation by smaller kernels in a multi-kernel application
- Adaptive spatial partitioning is implemented as an extension to fixed partition
- Two new properties added to the OpenCL `clCreateSubdevices` API
  - **Fixed property**: Creates sub-device with a fixed number of CUs
  - **Adaptive property**: Creates a sub-device which can dynamically allocate CUs
- Partition handler allocates CUs to *adaptive sub-device* based on size of the executing NDRange
- Adaptive Partition handler is invoked when:
  - A new NDRange arrives
  - An active NDRange completes execution
- Adaptive partition handler consists of 3 modules:
  - Dispatcher
  - NDRange scheduler
  - Load balancer
ADAPTIVE SPATIAL PARTITIONING

Dispatcher:

- Invoked when an NDR arrives or leaves the GPUs
- Checks for new NDRanges
  - Dispatches them to the NDR-scheduler
- Checks for completed NDRanges
  - Invokes the Load Balancer
- Manages the pending NDRanges
ADAPTIVE SPATIAL PARTITIONING

NDR Scheduler:

- Checks sub-device property for new NDRanges
  - Calls Load balancer to manage adaptive NDRanges
  - Assigns requested CUs to fixed partition NDRanges
- Also manages pending NDRanges
ADAPTIVE SPATIAL PARTITIONING

Load Balancer:

- Handles all adaptive NDRanges
- Assigns CUs to adaptive NDRanges based on their size
  - Considers ratio of NDR sizes for CU allocation
- Maintains at least 1 CU per **active** Adaptive NDRRange
- Calls the workgroup scheduler

---

CU assignment According to NDR ratio

Maintain 1 CU per Active adaptive NDR

---

New Adaptive NDR

Add new NDR to Adaptive NDR list

\[ M = \text{Free_CUs} + \text{Adaptive_CUs} \]

Assign \( M \) CUs to \( \text{adapt}_\_\text{NDRs} \) according to ratio of the NDR size

For each \( \text{active}_\_\text{adapt}_\_\text{NDRs} \) is \( \text{CU} / \text{NDR} \geq 1 \)

- Yes
  - Move Adaptive NDR \( \rightarrow \) Active Adaptive NDRRanges
  - Ensure \( \text{CU} / \text{active}_\_\text{adapt}_\_\text{NDRs} \geq 1 \)
  - Execute Workgroup Scheduler

- No
  - Move Adaptive NDR \( \rightarrow \) PendingNDRanges
  - Exit Loop
WORKING OF ADAPTIVE PARTITIONING

- NDR#0 allocated CU #1-22
- NDR#0 starts execution on CQ#1-22
- NDR#1 not yet scheduled for execution

- NDR#1 assigned 12 CUs
- CU#23-32 execute work from NDR#1
- CU#21-22 blocked to complete execution of NDR#0 workgroups

- CU#21-22 complete work from NDR#0
- CU#21-22 removed from NDR#0 usable list
- CU#21-22 allocated to NDR#1

- Two adaptive NDRages (NDR#0, NDR#1) mapped to two command queues (CQ#0, CQ#1)
- Initial allocation of CUs done for NDR#0
- NDR#1 arrives for execution, causing reassignment of resources
  - Blocking policy implemented to allow execution of NDR#0 workgroups
  - Blocking also prevents more workgroups from NDR#0 being scheduled on blocked CUs
- De-allocation of CUs for NDR#0 and mapping of CU#21 and CU#22 to NDR#1
## Workgroup Scheduling Mechanisms

1. **Occupancy Based Scheduling:**
   - Maps workgroups to CU and moves to next if:
     - Max workgroup limit is for CU reached
     - CU expends all compute resources
   - Attempts for maximum occupancy on GPU

2. **Latency Based Scheduling:**
   - Iterates over CUs in round robin:
     - Assigns 1 workgroup to each CU in iteration
     - Continues till all workgroups are assigned
   - Minimizes compute latency by utilizing each CU

## Partitioning Policies

1. **Full-fixed**
   - Each sub-device gets fixed number of CUs
   - Completely controlled by user
   - Best when *user is knowledgeable* about device hardware

2. **Full-adaptive**
   - Each sub-device has adaptive property
   - CU assignment *controlled by runtime*
   - Best when user does not know about device

3. **Hybrid**
   - Combination of fixed and adaptive property sub-devices
   - Best for performance tuning of applications
**PIPE BASED COMMUNICATION CHANNEL**

- Pipe is a typed memory object with **FIFO functionality**
- Data stored in form of packets with scalar and vector data type (int, int4, float4, etc.)
- Size of pipe based on number of packets and size of each packet
- Transactions done using OpenCL built in functions `write_pipe` and `read_pipe`
- Can be accessed by a kernel as a read-only or write-only memory object
- Used for **producer-consumer based communication** pattern between concurrent kernels
## BENCHMARKS USED FOR EVALUATION

### Set 1. Adaptive Partition Evaluation:

1. **Matrix Equation Solver (MES):**
   - Linear solver with 3 kernels

2. **Communication Channel Analyzer (COM):**
   - Emulates 4 communication channels 4 kernels

3. **Big Data Clustering (BDC):**
   - Big Data analysis application with 3 kernels

4. **Search Application (SER):**
   - Distributed search using 2 kernels mapped to 2 CQs

5. **Texture mixing (TEX):**
   - Image application to perform mixing of 3 textures

### Set 2. Pipe-Based Communication Evaluation:

1. **Audio Signal Processing (AUD):**
   - Two channel audio processing in 3 stages
   - Stages connected using pipe

2. **Search-Bin Application (SBN):**
   - Search benchmark with bin allocation
   - 2 kernels to perform distributed search
   - 1 kernel performs bin allocation
   - Search kernels and bin kernel connected by pipes
EVALUATION PLATFORM

- Multi2Sim simulation framework used for evaluation [multi2sim.org]
  - **Cycle level GPU simulator**
  - Supports x86 CPU, AMD Southern Islands (SI) GPU simulation
  - Provides OpenCL runtime and driver layer

- Simulator configured to match AMD SI Radeon 7970 GPU

- GPU scheduler updated for:
  - New Workgroup scheduling
  - Adaptive Partitioning handling

- Runtime updated for:
  - Sub-device property support
  - OpenCL pipe support

---

<table>
<thead>
<tr>
<th>Device Configuration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compute Unit Configuration</strong></td>
<td></td>
</tr>
<tr>
<td># of CUs</td>
<td>32</td>
</tr>
<tr>
<td># of Wavefront pools / CU</td>
<td>4</td>
</tr>
<tr>
<td># of SIMD/CU</td>
<td>4</td>
</tr>
<tr>
<td># of lanes/SIMD</td>
<td>16</td>
</tr>
<tr>
<td>Vector Regs/CU</td>
<td>64K</td>
</tr>
<tr>
<td>Scalar Regs/CU</td>
<td>2K</td>
</tr>
<tr>
<td>Frequency</td>
<td>1GHz</td>
</tr>
</tbody>
</table>

| **Memory Configuration**                  |                  |
| Global Memory                             | 1GB              |
| Local Memory/CU                           | 64KB             |
| L1 cache                                  | 16KB             |
| L2 cache                                  | 768KB            |
| # of Mem controller                       | 6                |
**EFFECTS OF ADAPTIVE PARTITIONING**

**Occupancy based scheduling:**
- **Full-fixed** limits applications to fixed CUs
  - Leads to poor utilization of GPU
- **Full-adaptive** shows avg. **improvement of 26%**
  - BDC shows **32% degradation**
- **Hybrid partition**
  - Effective load balancing over fixed and adaptive

**Latency based scheduling:**
- Best performance using full-fixed and hybrid
- Uses CUs effectively in fixed partition
- Degradation using full-adaptive
  - First NDRange occupies entire GPU
  - **Reassignment is slow**
  - BDC shows **96% degradation** over full-fixed
  - Others show **35% degradation** on average
LOAD BALANCING MECHANISM

- Timeline showing allocation and de-allocation of CUs for each kernel
- Full-adaptive partitioning used
- Sampling interval of 5000 GPU cycles
- Load Balancing Mechanism:
  - NDR 0 arrives first on GPU and receives large number of CUs
  - CUs reassigned as other NDRs arrive
  - Steady state reached when all kernels are executing
- Average CU reassignment latency is 9300 cycles
PERFORMANCE OF PIPE COMMUNICATION

- **Set-2 benchmarks** evaluated with pipe object
- Baseline is single-CQ execution
- **Average 2.9x speedup** observed with multiple CQ and pipe
- Pipe object overlaps computation and communication
- Shared data buffers across stages improves cache hits
  - 35% increase in cache hits
CONCLUSION

- We evaluate the **benefits of multiple command queue** mapping on GPUs
- We consider the effects of different workgroup scheduling policies using multiple command queues
- Applying tighter control over the workgroup to CU mapping produces **over 3x speedup**
- Adaptive partitioning helps in effective resource utilization and load balancing
- Developers get a choice of different partitioning policies according to the computation requirements
  - Use **adaptive partitioning** when not knowledgeable about GPU hardware
  - Use **fixed or hybrid partitioning** for greater control over hardware utilization
- We evaluate the performance and programmability of OpenCL pipes for inter-kernel communication
FUTURE WORK

- Consider spatial affinity between CU and command queues in scheduler policy
- Experiment with on-chip and off-chip memory based pipe objects
- Evaluate pipe object efficiency for HSA class APUs with shared virtual memory
- Implement priority based execution for kernels
THANK YOU!

QUESTIONS? COMMENTS?

Yash Ukidave
yukidave@ece.neu.edu