Dynamic Thread Block Launch: A Lightweight Execution Mechanism to Support Irregular Applications on GPUs

Jin Wang*, Norm Rubin†, Albert Sidelnik†, Sudhakar Yalamanchili*

*Computer Architecture and System Lab, Georgia Institute of Technology
†NVIDIA Research

Sponsors: NSF, NVIDIA
Executive Summary

- Motivation: New Irregular Application Domain

- Problem: **Dynamically Formed** structured pockets of Parallelism (DFP)
  - Fine-grained parallelism
  - Large amount of dynamic workloads

- Proposed: **Dynamic Thread Block Launch** (DTBL)
  - New extension to GPU Execution Model
  - Runtime and Microarchitecture for efficient support of DFP

- Results: Average 1.21x Speedup over Flat Implementations
Irregular Applications on GPUs

- GPUs are effective for regular applications

- New Irregular Application Domain
  - Time-varying, hierarchical, fine-grained parallelism
  - Unstructured control and memory behavior
  - Low Compute Utilization

Machine Learning

Graph Processing

Relational Computing
Executive Summary

- Motivation: New Irregular Application Domain

- Problem: Dynamically Formed structured pockets of Parallelism (DFP)
  - Fine-grained parallelism
  - Large amount of dynamic workloads

- Proposed: Dynamic Thread Block Launch (DTBL)
  - New extension to GPU Programming Model
  - Runtime and Microarchitecture for efficient support of DFP

- Results: Average 1.21x Speedup over Flat Implementations
Dynamically Formed Parallelism (DFP)

- Pockets of Structured Parallelism in irregular applications
  - Locally uniform control flow
  - Locally coalesced memory accesses

Adaptive Mesh Refinement (AMR)

Graph Traversal

Pockets of Structured Parallelism

Workload Imbalance
Features of DFP in Irregular Applications[1]

- High Dynamic Workload Density: 1000~100,000 pockets of structured parallelism (# of dynamic kernels)

- Low Compute Intensity: Average degree of parallelism in dynamic workload is ~40 (# of threads in dynamic kernels)

- Workload Similarity: most dynamic workloads execute the same kernel code (with different configuration, i.e. TB size, shared memory size, etc…)

State of the Art Solution for DFP

- **Persistent Thread Model**[1]
  - TBs stay on all SMXs for lifetime
  - Work Queue
  - Consistent with current BSP model
  - Pure software solution
  - Less flexibility in TB scheduling
  - Low SMX occupancy and efficiency

- **CUDA Dynamic Parallelism (CDP)**[2]
  - Introduced in GK110 GPU
  - Launch kernels from GPU
  - Launches can be nested
  - Flexibility and productivity in processing dynamic workload
  - Increased dynamic workload throughput
  - Low SMX utilization for fine-grained kernels
  - High kernel launching latency

---

Executive Summary

- Motivation: New Irregular Application Domain

- Problem: Dynamically Formed structured pockets of Parallelism (DFP)
  - Fine-grained parallelism
  - Large amount of dynamic workloads

- Proposed: Dynamic Thread Block Launch (DTBL)
  - New extension to GPU Programming Model
  - Runtime and Microarchitecture for efficient support of DFP

- Results: Average 1.21x Speedup over Flat Implementations
Dynamic Thread Block Launch (DTBL)

**Motivation:**
- Light-weight, efficient programming model and microarchitecture for DFP

**Propose: DTBL**
- Extend the current GPU execution model
- Threads can launch fine-grained TBs on demand instead of kernels

**Goal:**
- Increase SMX execution efficiency for DFP
- Minimize hardware and runtime overhead
Semantics of DTBL

- Each thread in a kernel can launch new TBs
- Launches can be nested
- New TBs with the same configuration are coalesced with the original kernel, or
- New TBs are coalesced with another kernel
- Coalescing: new TBs belong to the same kernel and scheduled together
Example: Graph Traversal

- Launch new TBs for DFP
  - All the TBs are doing the same work
- Launch only when sufficient parallelism

Parent Kernel

Dynamically Launched TBs

- Uniform control flow
- More memory coalescing

Similar benefit in CDP by launching child kernels
Example: Graph Traversal

- Launch new TBs for DFP
  - All the TBs are doing the same work
- Launch only when sufficient parallelism

Parent Kernel

DTBL: Coalesced with a new Kernel

Reduced workload imbalance

More memory coalescing
Microarchitecture (Baseline GPU)

Kernel Distributor

Entries

PC Dim Param ExeBL

FCFS Controller

Kernel Distributor

SMX Scheduler

Control Registers

KDEI NextBL

SMX

SMX

SMX

SMX

DRAM

Memory Controller

Thread Block Control Registers

KDEI BLKID
Microarchitecture Extension

New TB launching information

Tracking new TBs

Aggregated Group Table

Coalescing and scheduling new TBs

Microarchitecture Extension

Kernel Distributor Entries

SMX

SMX Scheduler

Control Registers

KDEI AGEI NextBL

SMX

SMX

SMX

SMX

Memory Controller

DRAM

Thread Block Control Registers

KDEI AGEI BLKID

Tracking new TBs

Kernel Distributor

Entries

PC Dim Param ExeBL

NAGEI LAGEI

SMX

FCFS Controller

Aggregated Group Information

PC Dim Param Next

KDEI AGEI NextBL

ExeBL

ExeBL

ExeBL

ExeBL
TB Coalescing and Scheduling

- Achieve high SMX occupancy and utilization

- Recall DFP are fine-grained dynamic workloads with high density.

- TB coalescing allows enough TBs to be aggregated into one big kernel and scheduled to SMXs.

- In comparison: CDP only allows 32 concurrent kernel execution. SMXs are not fully occupied for small kernels.
Examples for TB coalescing

- One TB has 64 threads or 2 warps (in CDP, that is one kernel)
- On K20c GK110 GPU: 13 SMX, max 64 warps/SMX, max 32 concurrent kernels

**CDP: All child kernels**

Kernel 1  Kernel 2  Kernel 3  ......  Kernel 100

**SMXs**

SMX0  SMX1  SMX12
Examples for TB coalescing

- One TB has 64 threads or 2 warps (in CDP, that is one kernel)
- On K20c GK110 GPU: 13 SMX, max 64 warps/SMX, max 32 concurrent kernels

**CDP: All child kernels**

```
Kernel 2  Kernel 3  ......  Kernel 100
```

```
SMX0  SMX1  SMX12
```

- SMX0
- SMX1
- SMX12

- Register File
- Cores
- L1 Cache/Shared Memory
Examples for TB coalescing

- One TB has 64 threads or 2 warps (in CDP, that is one kernel)
- On K20c GK110 GPU: 13 SMX, max 64 warps/SMX, max 32 concurrent kernels

CDP: All child kernels

SMXs

Kernel 1

Kernel 2

Kernel 3

......

Kernel 100

SMX0

SMX1

SMX12
Examples for TB coalescing

- One TB has 64 threads or 2 warps (in CDP, that is one kernel)
- On K20c GK110 GPU: 13 SMX, max 64 warps/SMX, max 32 concurrent kernels

**CDP: All child kernels**
Examples for TB coalescing

- One TB has 64 threads or 2 warps (in CDP, that is one kernel)
- On K20c GK110 GPU: 13 SMX, max 64 warps/SMX, max 32 concurrent kernels

CDP: All child kernels

Kernel 15

Kernel 100
Examples for TB coalescing

- One TB has 64 threads or 2 warps (in CDP, that is one kernel)
- On K20c GK110 GPU: 13 SMX, max 64 warps/SMX, max 32 concurrent kernels
Examples for TB coalescing

- One TB has 64 threads or 2 warps (in CDP, that is one kernel)
- On K20c GK110 GPU: 13 SMX, max 64 warps/SMX, max 32 concurrent kernels

Achieved SMX Occupancy: 32 kernels * 2 warps/kernel / (64 warps/SMX * 13 SMXs) = 0.077
Examples for TB coalescing

- One TB has 64 threads or 2 warps (in CDP, that is one kernel)
- On K20c GK110 GPU: 13 SMX, max 64 warps/SMX, max 32 concurrent kernels

**DTBL: All TBs coalesced with one kernel. No restrictions in scheduling.**

**SMXs**
Examples for TB coalescing

- One TB has 64 threads or 2 warps (in CDP, that is one kernel)
- On K20c GK110 GPU: 13 SMX, max 64 warps/SMX, max 32 concurrent kernels

DTBL: All TBs coalesced to one kernel. No restrictions in scheduling.
Examples for TB coalescing

- One TB has 64 threads or 2 warps (in CDP, that is one kernel)
- On K20c GK110 GPU: 13 SMX, max 64 warps/SMX, max 32 concurrent kernels

Achieved SMX Occupancy: 100 TBs * 2 warps/TB / (64 warps/SMX * 13 SMXs) = 0.24

Can be increased if more TBs!
TB Coalescing on Microarchitecture

Kernel Distributor

Aggregated Group Table

AggDim Param Next ExeBL

SMX Scheduler

Control Registers KDEI AGEI NextBL

SMX

Thread Block Control Registers KDEI AGEI BLKID

Coalescing

Newly launched TBs

Kernel Management Unit

FCFS Controller

Kernel Distributor

K

Dim Param ExeBL

NAGEI LAGEI

Microarchitecture Extension

DRAM

Memory Controller

PC Dim Param Next

TB

SMX

TB

SMX

TB

SMX

TB

SMX

TB

SMX
TB Coalescing on Microarchitecture

Scheduled together to increase SMX Occupancy
TB Launching Latency

- Exclusive launching path from SMXs
- Simple runtime and driver support with low latency

In comparison, CDP uses
  - Longer launching path
  - More complicated runtime and driver for kernel management
Launch Path for CDP and DTBL

Kernel Distributor

Aggregated Group Table

Shorter path to KDE

If launched as kernel

Manage Thousands of kernels

Kernel Management Unit

Control Registers

SMX Scheduler

SMX

Microarchitecture Extension

Thread Block Control Registers

DRAM
Launch Path for CDP and DTBL

Launch path for CDP: ~70k cycles
Launch path for DTBL: ~10k cycles

(32 launches per warp)
Executive Summary

- **Motivation:** New Irregular Application Domain

- **Problem:** Dynamically Formed structured pockets of Parallelism (DFP)
  - Fine-grained parallelism
  - Large amount of dynamic workloads

- **Proposed:** Dynamic Thread Block Launch (DTBL)
  - New extension to GPU Programming Model
  - Runtime and Microarchitecture for efficient support of DFP

- **Results:** Average 1.21x Speedup over Flat Implementations
Methodology

- **Tool: GPGPU-Sim**
  - Implemented CDP
  - Implemented DTBL
  - Modeling Tesla K20c (13 SMX, 65536 registers, 32KB shared memory, max 32 concurrent kernels)

- Compare Flat, CDP and DTBL implementations
High-level Timing Comparison

Flat Implementation

CDP Implementation

Kernel launch overhead

DTBL Implementation

Native Kernel

Coalesced TBs

DTBL overhead
### Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Input Type</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive Mesh Refinement</td>
<td>Energy</td>
<td>amr</td>
</tr>
<tr>
<td>Barnes Hut Tree</td>
<td>Random</td>
<td>bht</td>
</tr>
<tr>
<td>Breadth-First Search</td>
<td>Citation network</td>
<td>bfs_citation</td>
</tr>
<tr>
<td>USA road network</td>
<td>USA road</td>
<td>bfs_usa_road</td>
</tr>
<tr>
<td>Cage15 sparse matrix</td>
<td>Cage15</td>
<td>bfs_cage15</td>
</tr>
<tr>
<td>Graph Coloring</td>
<td>Citation network</td>
<td>clr_citation</td>
</tr>
<tr>
<td>Graph 500</td>
<td>Graph 500</td>
<td>clr_graph500</td>
</tr>
<tr>
<td>Cage15 sparse matrix</td>
<td>Cage15</td>
<td>clr_cage15</td>
</tr>
<tr>
<td>Regular Expression Match</td>
<td>Darpa</td>
<td>regx_darpa</td>
</tr>
<tr>
<td>Relational Join</td>
<td>Uniform distribution</td>
<td>join_uniform</td>
</tr>
<tr>
<td>Product Recommendation</td>
<td>Movielens</td>
<td>pre</td>
</tr>
<tr>
<td>Single-Source Shortest Path</td>
<td>Citation</td>
<td>sssp_citation</td>
</tr>
<tr>
<td></td>
<td>Fight network</td>
<td>sssp_flight</td>
</tr>
<tr>
<td></td>
<td>Cage15 sparse matrix</td>
<td>sssp_cage15</td>
</tr>
</tbody>
</table>

#### Tree and Graph Applications

- **Physics Simulation**
- **Machine Learning**
- **Relational Database (Big Data)**
## Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Input</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive Mesh Refinement</td>
<td>Energy data set</td>
<td>amr</td>
</tr>
<tr>
<td>Barnes Hut Tree</td>
<td>Random data set</td>
<td>bht</td>
</tr>
<tr>
<td>Breadth-First Search</td>
<td>Citation network</td>
<td>bfs_citation</td>
</tr>
<tr>
<td></td>
<td>USA road network</td>
<td>bfs_usa_road</td>
</tr>
<tr>
<td></td>
<td>Cage15 sparse matrix</td>
<td>bfs_cage15</td>
</tr>
<tr>
<td>Graph Coloring</td>
<td>Citation network</td>
<td>clr_citation</td>
</tr>
<tr>
<td></td>
<td>Graph 500</td>
<td>clr_graph500</td>
</tr>
<tr>
<td></td>
<td>Cage15 sparse matrix</td>
<td>clr_cage15</td>
</tr>
<tr>
<td>Regular Expression Match</td>
<td>Darpa network</td>
<td>regx_darpa</td>
</tr>
<tr>
<td></td>
<td>Random string collection</td>
<td>regx_string</td>
</tr>
<tr>
<td>Product Recommendation</td>
<td>MovieLens</td>
<td>pre</td>
</tr>
<tr>
<td>Relational Join</td>
<td>Uniform distribution synthetic</td>
<td>join_uniform</td>
</tr>
<tr>
<td></td>
<td>Gaussian distribution synthetic</td>
<td>join_gaussian</td>
</tr>
<tr>
<td>Single-Source Shortest Path</td>
<td>Citation network</td>
<td>sssp_citation</td>
</tr>
<tr>
<td></td>
<td>Fight network</td>
<td>sssp_flight</td>
</tr>
<tr>
<td></td>
<td>Cage15 sparse matrix</td>
<td>sssp_cage15</td>
</tr>
</tbody>
</table>
Speedup of DTBL

- Ideal: excluding launch latency, 1.63x over flat implementations and 1.14x over CDP
- Actual: including launch latency, 1.21x over flat implementations and 1.40x over CDP
Speedup of DTBL

- No performance change: bfs_usa_road and sssp_flight
- Degree of parallelism is too small in DFP, no CDP kernel or DTBL TB launches.
- DTBL slowdown: regx_string and pre.
- Too many TBs are launched. Launch latency is still high, overshadowing any performance benefit.
Conclusions

- **DTBL**: New extension to GPU execution Model
- Efficient solution for **Dynamic Parallelism** in irregular applications

- Shows more benefit for dynamic, **fine-grained but sufficient** parallelism
- Dynamic **launching latency** is reduced from CDP, but can still be optimized
THANK YOU!

QUESTIONS?
BACKUP
GPU and CUDA

GPU: Many-Core Architecture

CUDA: NVIDIA’s GPU Programming Model

- CUDA Kernel (1D-3D grid of TBs)
- Thread Block (TB)
- Warp (32 threads)
- Register File
- Cores
- L1 Cache/Shared Memory
- SMXs on GPU
- Device Memory
- Thread
- Branch divergence
- Reconverge
- Thread Block

42
Semantics of DTBL Cont.

- Thread hierarchy: 1D-3D same as original CUDA
- Synchronization: only within TB, not across different TBs, same as the CUDA model
- Memory model: same as the CUDA model
- Dependency: no dependency between TBs

- Program Interface: One additional CUDA Device Runtime API
  
  \texttt{cudaLaunchAggGroup}

- Minimal extension to CUDA programming model and interface
  - Simplify compiler and runtime
Major Microarchitecture Overhead

- Aggregated Group Table (AGT): ~20KB on-chip SRAM
- Control registers: 1096 Bytes

- TB coalescing through AGT and control registers: 1 ~ 32 cycles (excluding parameter buffer setup)
Speedup of CDP implementation

- CDP-ideal: excluding launch overhead, average **1.47x speedup**
- CDP-actual: including launch overhead, average **1.21x slowdown**
Baseline Microarchitecture

Kernel Distributor Entry
- PC
- Dim
- Param
- ExeBL

Concurrent Kernel Execution
- Warp Schedulers
  - Warps
  - Warps
  - Warps
  - Warps

- Warp Context
- Registers
- Core
- Core
- Core
- Core

L1 Cache / Shard Memory

Pending Kernels

Kernel Management Unit
- SMX
- Core
- Core
- Core
- Core

Interconnection Bus

Baseline Microarchitecture
- Host
- CPU
- Interconnection Bus
- GPU

Kernel Distributor
- SMX Scheduler
- Control Registers
- SMX
- SMX
- SMX
- SMX

Registers

Memory Controller
- DRAM

L2 Cache
Why Slowdown?

Limit: 32 for concurrent kernels

SMX underutilized

Interconnection Bus

Pending Kernels

Host CPU

Interconnection Bus

Kernel Management Unit

Kernel Distributor

Warp Schedulers

Warp Context

Registers

Core Core Core Core

L1 Cache / Shard Memory

Memory Controller

DRAM

L2 Cache

Interconnection Bus

Control Registers

TB TB SMX SMX SMX SMX

Interconnection Bus

try m ExeBL

Interconnection Bus

Limit: 32 for concurrent kernels

Interconnection Bus

Kernel Distributor Entry

Control Registers

TB TB SMX SMX SMX SMX

Limit: 32 for concurrent kernels

Interconnection Bus

Kernel Distributor Entry

Control Registers

TB TB SMX SMX SMX SMX

Limit: 32 for concurrent kernels

Interconnection Bus

Kernel Distributor Entry

Control Registers

TB TB SMX SMX SMX SMX

Limit: 32 for concurrent kernels

Interconnection Bus

Kernel Distributor Entry

Control Registers

TB TB SMX SMX SMX SMX

Limit: 32 for concurrent kernels

Interconnection Bus

Kernel Distributor Entry

Control Registers

TB TB SMX SMX SMX SMX

Limit: 32 for concurrent kernels

Interconnection Bus

Kernel Distributor Entry

Control Registers

TB TB SMX SMX SMX SMX

Limit: 32 for concurrent kernels

Interconnection Bus

Kernel Distributor Entry

Control Registers

TB TB SMX SMX SMX SMX

Limit: 32 for concurrent kernels
Why Slowdown?

Management of Thousands of Kernels

- Interconnection Bus
- Kernel Distributor Entry
  - PC
  - Dim
  - Param
  - ExeBL
- SMX
- SMX Scheduler
- Control Registers
- SMX
- SMX
- SMX
- SMX
- Warp Schedulers
- Warp Context
- Registers
- Core
- Core
- Core
- Core
- L1 Cache / Shard Memory

Device Kernel Launch

- Host CPU
- L2 Cache
- Memory Controller
- DRAM
Benefit of DTBL and proposed Microarchitecture

- Kernel Distributor
- Aggregated Group Table
- SMX Scheduler
- Control Registers
- Memory Controller
- SMX
- Thread Block Control Registers
- Microarchitecture Extension

Shorter path than to KMU

If launched as kernel