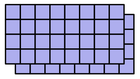
Dynamic Thread Block Launch: A Lightweight Execution Mechanism to Support Irregular Applications on GPUs

Jin Wang†, Norm Rubin*, Albert Sidelnik* and Sudhakar Yalamanchili†
†Georgia Institute of Technology, *NVIDIA Research

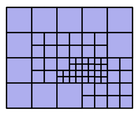
Motivation

- GPUs are effective for structured applications

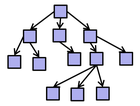


Rigid 1/2/3D data structure

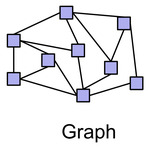
- However, for ***unstructured applications***
  - Poor workload balance -> control flow divergence
  - Un-coalesced memory access -> memory divergence
- Low Compute Utilization



Adaptive Mesh

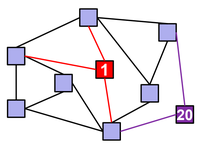


Tree



Graph

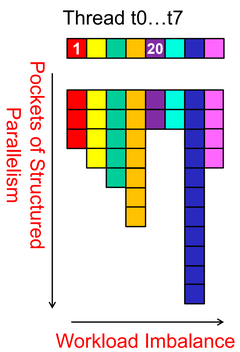
- Dynamically Formed Parallelism (***DFP***)
  - Pockets of Dynamically Formed Structured Parallelism
  - Locally uniform control flow
  - Locally coalesced memory access



Part of the graph showing vertex 1 and 20

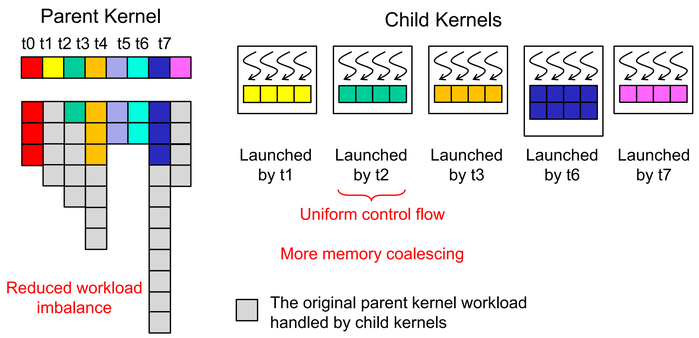
Vertex Frontier

Vertex #		Degree of Vertex
1		3
		4
		5
		7
20		2
		2
		11
		4



Implementing DFP with CUDA Dynamic Parallelism (CDP) [1]

- Launch a child kernel for detected DFP
- Launch only when sufficient parallelism is available



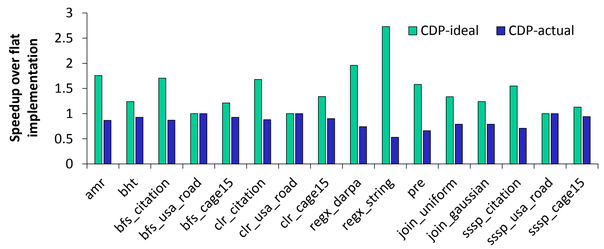
Potential Benefits

- Reduce control divergence
- Increase coalesced memory accesses

VS

Issues

- Large number of kernels
- Kernel launch ***overhead***
- Memory footprint



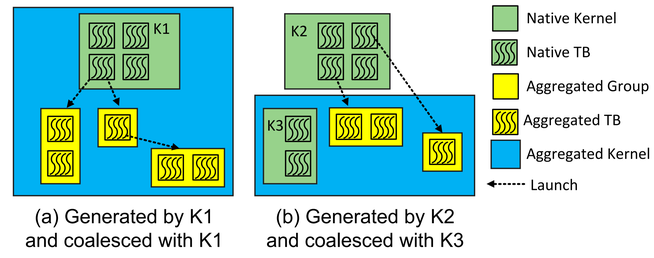
Overall Performance: Speedup over flat implementations

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

DTBL: Dynamic Thread Block Launch

- Extend the current GPU execution model with DTBL
- Thread blocks can be dynamically launched from a GPU thread
- Use light-weight thread blocks rather than heavy-weight device kernel for DFP

Semantics



(a) Generated by K1 and coalesced with K1

(b) Generated by K2 and coalesced with K3

- Native Kernel: original launched by host or device
- Native TB: thread blocks in a native kernel
- Aggregated TB: dynamically launched thread blocks
- Aggregated Group: a group of aggregated TBs organized in three dimensions that are coalesced with a native kernel
- Aggregated Kernel: kernel with aggregated TBs
- New aggregated TBs can be coalesced with the same kernel (a) or to a different kernel (b)

Thread Hierarchy, Memory Model and Synchronizations

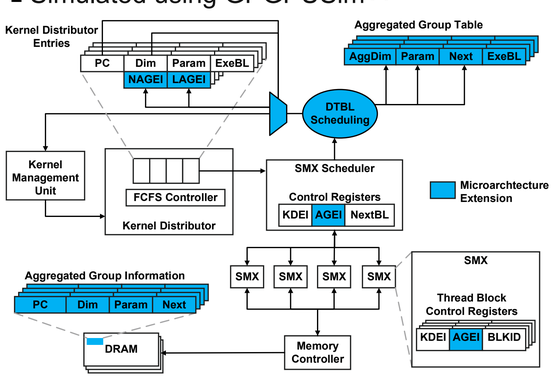
- Same as the original CUDA programming model
- Three-dimension hierarchy
- Local memory is private to thread, shared memory is private to TB
- No synchronizations between TBs

Programming Interface

- Utilize current CDP device runtime APIs
- Add a new device runtime API call: `cudaLaunchAggGroup`

Architecture Extensions and SMX Scheduling

- Introduce new microarchitecture data structures to manage the aggregated groups and TBs
- The SMX Scheduler is extended for efficient scheduling of aggregated TBs
- Simulated using GPGPUSim [2]



Overhead Analysis

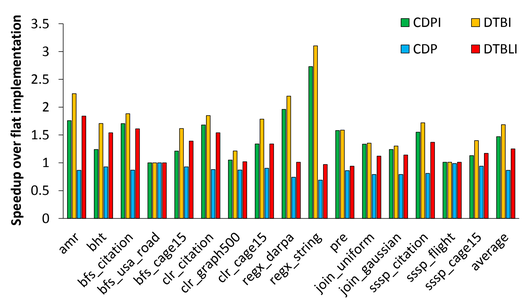
- Hardware overhead (On-chip SRAM)
  - 1024-entry Aggregated Group Table: 20KB
  - All other data structures: 1096 Bytes
- Timing
  - Allocating parameter buffers (~10,000 cycles)
  - Search kernels in Kernel Distributor Entries (max 32 cycles)
  - Search free entry in Aggregated Group Table (1 cycle)

DTBL Benefits compared with CDP

- Preserve the same benefits of CDP
  - Reduce control divergence
  - Increase coalesced memory accesses
- Reduce launch latency
  - Avoid long path to Kernel Management Unit when launching new workload
- Increase SMX execution efficiency
  - Increase TB-level concurrency
  - Higher SMX occupancy
- Reduce global memory footprint
  - Consuming dynamic workload faster so the global memory reserved to hold launching information can be released

Performance

- Ideal: excluding launch latency, 1.67x over flat implementations and 1.15x over CDP
- Actual: including launch latency, 1.25x over flat implementations and 1.44x over CDP



References
[1] J. Wang, and S. Yalamanchili. "Characterization and analysis of dynamic parallelism in unstructured gpu applications." IISWC-2014, Best Paper Finalist. November 2014.
[2] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," ISPASS-2009, April 2009.