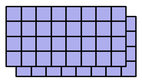
Exploring Dynamic Parallelism for Irregular Applications on GPUs

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

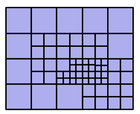
Irregular Applications and DFP

- GPUs are effective for structured applications

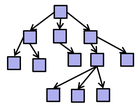


Rigid 1/2/3D data structure

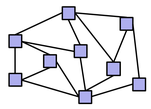
- However, for ***unstructured applications***
  - Poor workload balance -> control flow divergence
  - Un-coalesced memory access -> memory divergence
- Low Compute Utilization



Adaptive Mesh

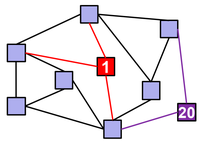


Tree



Graph

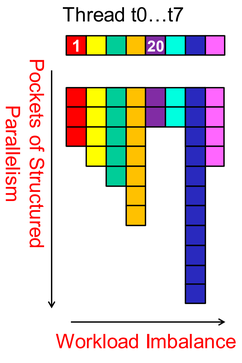
- Dynamically Formed Parallelism (***DFP***)
  - Pockets of Dynamically Formed Structured Parallelism
  - Locally uniform control flow
  - Locally coalesced memory access



Part of the graph showing vertex 1 and 20

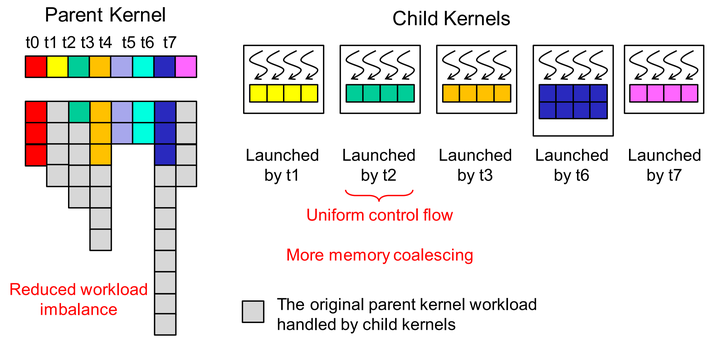
Vertex Frontier

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



Implementing Irregular Applications with CDP[1]

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



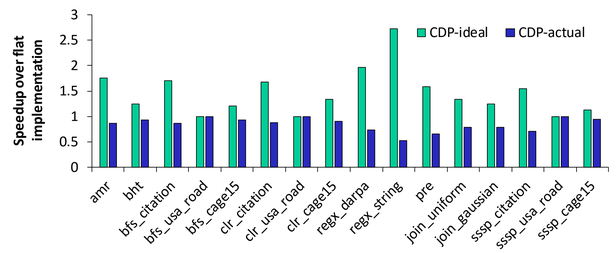
Potential Benefits

- Reduce control divergence
- Increase coalesced memory accesses

VS

Issues

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



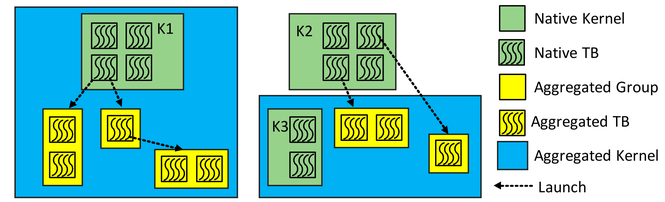
Overall Performance: Speedup over flat implementations

- CDP-ideal: excluding launch overhead, average 1.43x speedup
- CDP-actual: including launch overhead, average 1.16x slowdown

DTBL: Dynamic Thread Block Launch[2]

- Extend the current GPU execution model with DTBL
- Thread blocks can be dynamically launched from a GPU thread

Semantics



(a) Generated by K1 and coalesced with K1

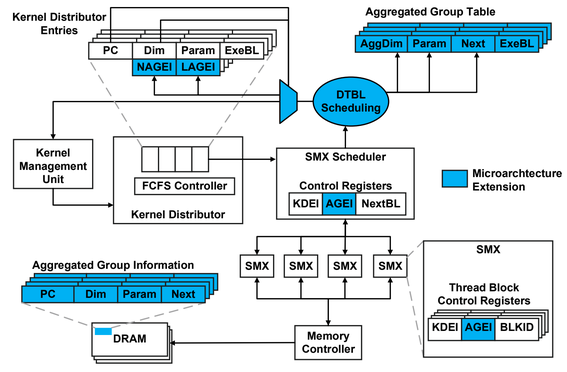
(b) Generated by K2 and coalesced with K3

Programming Interface

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

Architecture Extensions and SMX Scheduling

- New data structures for TB management
- New SMX scheduler for efficient TB scheduling

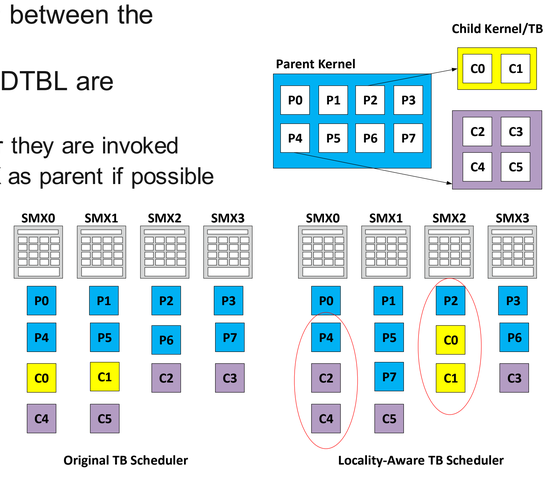


Locality-Aware Kernel/Thread Block Scheduler

- Utilize the spatial locality and temporal locality between the parent and the child
- Child kernels in CDP or child thread blocks in DTBL are scheduled closer to parents
  - Temporal: children are scheduled immediate after they are invoked
  - Spatial: children are scheduled on the same SMX as parent if possible
  - Load balancing: allow SMXs to pick children of other parents if necessary

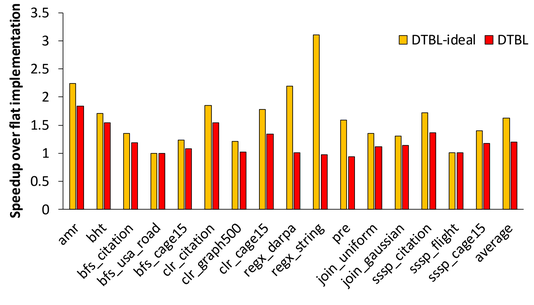
Microarchitecture Implementation

- Assign priority information to TBs
- Assign SMX binding information to TBs
- Extended SMX scheduler to manage prioritized TBs



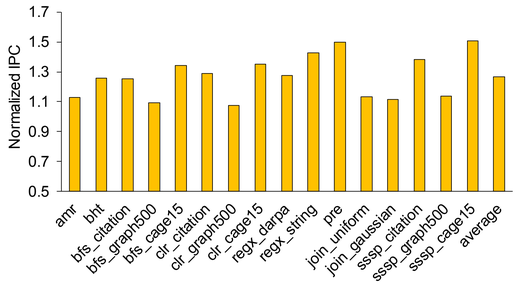
Performance of DTBL

- Increase SMX execution efficiency
- Reduce launching latency
- 1.21x speedup over flat implementation



Performance of Locality-Aware Scheduler

- Increase cache hit rate and memory efficiency
- Another 1.27x speedup over DTBL



References
[1] J. Wang, and S. Yalamanchili. "Characterization and Analysis of Dynamic Parallelism in Unstructured GPU Applications." IISWC-2014, Best Paper Finalist.
[2] J. Wang, N. Rubin, A. Sidelnik and S. Yalamanchili. "Dynamic Thread Block Launch: A Lightweight Execution Mechanism to Support Irregular Applications on GPUs." ISCA-2015.