LaPerm: Locality Aware Scheduler for Dynamic Parallelism on GPUs

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

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

Sponsors:
Executive Summary

- **Motivation:** New memory reference locality relationship in dynamic parallelism

- **Problem:** State-of-the-art GPU schedulers are unaware of this new relationship
  - Designed for non-dynamic parallelism settings
  - Do not exploit parent-child locality in L1 and L2 cache

- **Proposed:** LaPerm
  - Locality-aware thread block scheduler
  - Three scheduling decisions
Dynamic Parallelism on GPU

- Launch workload on demand from GPU
  - CUDA Dynamic Parallelism (CDP)
  - OpenCL device-side enqueue
  - Dynamic Thread Block Launch\(^1\) (DTBL)
- Benefits
  - Apply to fine-grained parallelism in irregular applications
  - Increase execution efficiency and productivity

Memory Locality in Dynamic Parallelism

- New **data reference locality relationship** in parent-child launching

- Potential Locality:
  - Parent-child and child-sibling data sharing
  - L1/L2 locality

- Average shared footprint ratio for 8 benchmarks:
  - Parent-child: **38.4%**
  - Child-sibling: **30.5%**
Executive Summary

- **Motivation:** New memory reference locality relationship in dynamic parallelism

- **Problem:** State-of-the-art GPU schedulers are unaware of this new relationship
  - Designed for non-dynamic parallelism settings
  - Do not exploit parent-child locality in L1 and L2 cache

- **Proposed:** LaPerm
  - Locality-aware thread block scheduler
  - Three scheduling decisions
Current GPU Scheduler

- First-Come-First-Serve kernel scheduler
- Round-Robin TB Scheduler
Current GPU Scheduler (continue)

- For non-Dynamic Parallelism scenario
  - Fairness and efficiency
- For Dynamic Parallelism scenario
  - Child TBs are scheduled after parent TBs
Current GPU Scheduler: Issues

- Issue 1: Child TBs are executed far later than the parent TBs, decreasing L2 locality
Current GPU Scheduler: Issues (continue)

- **Issue 1**: Child TBs are executed far later than the parent TBs, decreasing L2 locality
- **Issue 2**: Child TBs are executed on a different SMX than its parent, decreasing L1 locality
- **Fails to exploit parent-child or child-sibling locality**
  - Exacerbated in real applications which generally have many TBs.
Executive Summary

- **Motivation**: New memory reference locality relationship in dynamic parallelism

- **Problem**: State-of-the-art GPU schedulers are unaware of this new relationship
  - Designed for non-dynamic parallelism settings
  - Do not exploit parent-child locality in L1 and L2 cache

- **Proposed**: LaPerm
  - Locality-aware thread block scheduler
  - Three scheduling decisions
LaPerm: Locality-Aware Scheduler for Dynamic Parallelism

- Leverage locality between parent and child TBs

- Three scheduling decisions
  - Accommodate different forms of locality

- Goal: improve memory efficiency and overall performance
Scheduling Decision 1: TB Prioritizing

- Prioritize child TBs to be executed immediately after direct parent TBs
- Reuse parent data and avoid L2 cache pollution

Still no L1 locality!
Scheduling Decision 2: Prioritized SMX Binding

- Bind the prioritized child TBs on the same SMX as the parent TBs
- Utilize L1 cache for data reuse

SMX load balancing issue!
Scheduling Decision 3: Adaptive Prioritized SMX Binding

- Adaptively bind child TBs on available SMXs
- Avoid SMX load balancing caused by SMX binding
Architecture Support

- Multi-level priority queues
  - Used to store new TB/Kernel information
  - Ordered using priority value
  - Divided among multiple SMXs

Off-chip overflow priority queues

On-chip priority queues
Benchmark and Experimental Environment

- Benchmark implemented with dynamic parallelism
- Simulated on GPGPU-Sim

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Input Data Set</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 1</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</td>
<td>bfs_cage15</td>
</tr>
<tr>
<td>Graph Coloring</td>
<td>Citation network</td>
<td>clr_citation</td>
</tr>
<tr>
<td></td>
<td>Cage500</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 data set</td>
<td>regx_string</td>
</tr>
<tr>
<td>Product Recommendation</td>
<td>Movie network</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>ssssp_citation</td>
</tr>
<tr>
<td></td>
<td>Fight network</td>
<td>ssssp_flight</td>
</tr>
<tr>
<td></td>
<td>Cage15 sparse matrix</td>
<td>ssssp_cage15</td>
</tr>
</tbody>
</table>

Applications

- **Physics Simulation**
- **Tree and Graph Applications**
- **Machine Learning**
- **Relational Database**
Performance of LaPerm: L2 Cache

![Graph showing L2 Cache Hit Rate for different benchmarks and strategies (RR, TB-Pri, SMX-Bind, Adaptive-Bind).]
L2 Cache hit rate benefits mainly from prioritizing child TBs
L2 Cache hit rate benefits mainly from prioritizing child TBs
Graph applications with *cage15* input have more parent-child data reuse
Performance of LaPerm: L1 Cache

![Bar chart showing L1 cache hit rate for various benchmarks with different binding strategies: RR, TB-Pri, SMX-Bind, and Adaptive-Bind. The y-axis represents the L1 cache hit rate ranging from 0 to 100, and the x-axis lists different benchmarks such as amr, bht, bfs_citation, bfs_graph500, etc.
L1 Cache hit rate benefits mainly from binding child TBs to parents’ SMXs
Performance of LaPerm: L1 Cache

- L1 Cache hit rate benefits mainly from binding child TBs to parents’ SMXs
- Product recommendation *pre* has good child-sibling data locality
Performance of LaPerm: IPC

![Graph showing normalized IPC for different workloads and thread binding policies. The graph compares TB-Pri, SMX-Bind, and Adaptive-Bind against a reference (RR) normalized to 1.0. The x-axis represents various workloads, and the y-axis shows normalized IPC values ranging from 0.5 to 1.7. The graph indicates performance improvements for TB-Pri and Adaptive-Bind compared to SMX-Bind and SMX-Guard.]
Performance of LaPerm: IPC

- **TB-Pri**: IPC (1.13x) increases due to higher L2 cache hit rate
- **SMX-Bind**: IPC decreases (1.08x) from TB-Pri due to higher L1 cache hit rate but SMX load balancing
- **Adaptive-Bind**: Overall IPC (1.27x) increases because of both memory efficiency and load balancing
Conclusion

- **LaPerm**: Locality-aware thread block scheduler for dynamic parallelism
  - Exploit new memory reference locality in the parent-child launching

- Three scheduling decisions
  - Increase **L1/L2 cache locality** while maintaining **SMX load balance**
  - Achieve overall memory system efficiency and performance improvement
THANK YOU!

QUESTIONS?