Resource Sharing for GPUs

CGO 2016

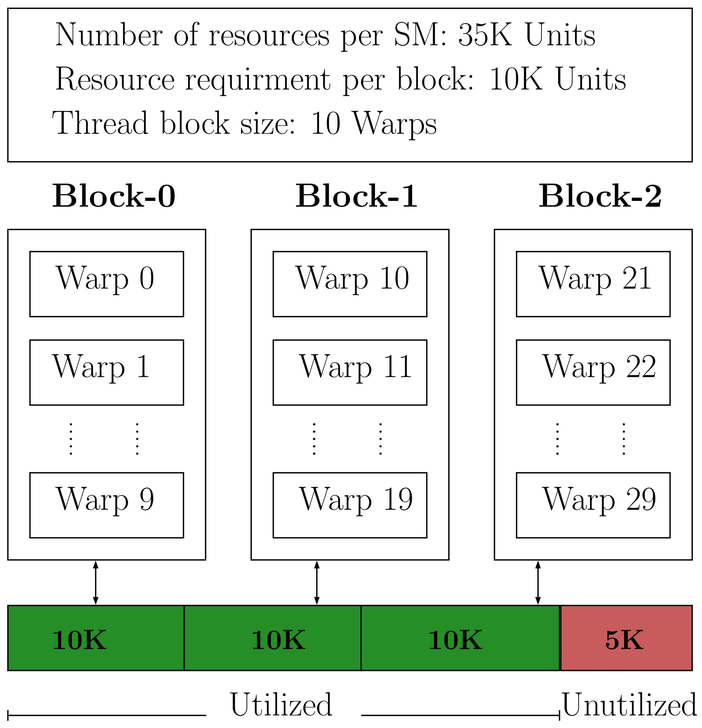
Vishwesh Jatala (IITK), Jayvant Anantpur (IISc), Amey Karkare (IITK)

Problem

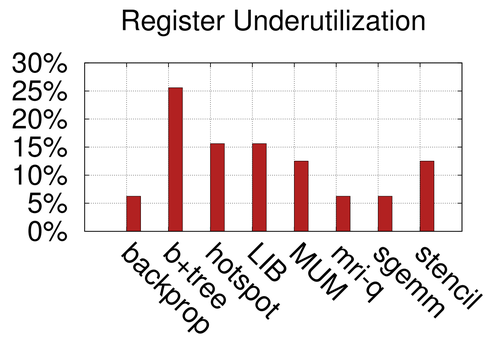
Resource allocation at thread block granularity in Graphics Processing Units (GPUs) has the following disadvantages [1]:

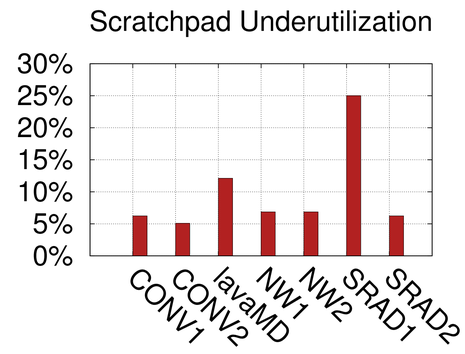
1. Resource underutilization
   - Registers and Scratchpad Memory get underutilized
2. Reduction in thread level parallelism (TLP)
   - Limited number of resident threads and blocks in streaming multiprocessors (SMs)
3. Potential reduction in throughput

Motivating Example



Resource Underutilization





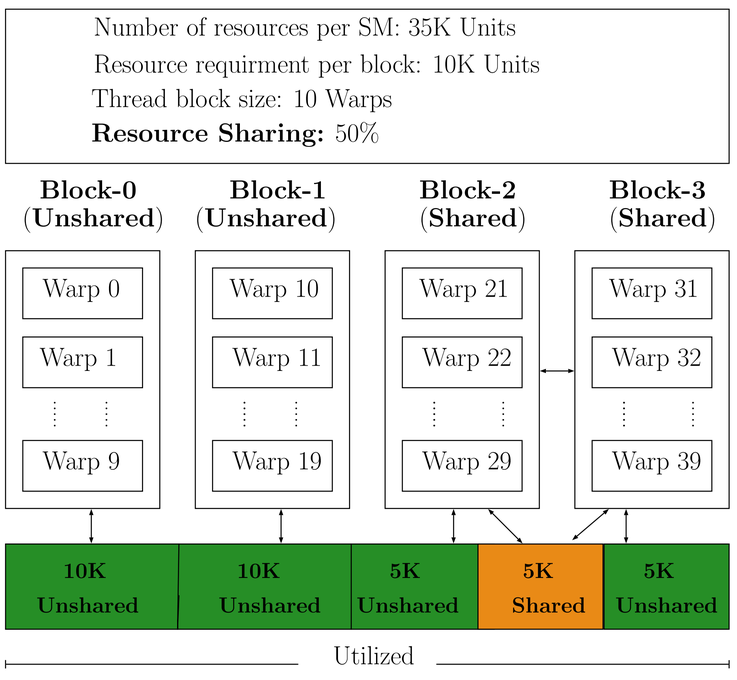
Solution: Resource Sharing

Idea: Share the resources between thread blocks

Strategy:

- Increase the TLP by launching additional thread blocks in each SM
- Minimize the resource wastage with the help of addition thread blocks that:
  1. Use wasted resource
  2. Share the resources with other resident blocks
- Access resources effectively to avoid deadlocks and to guarantee minimum number of blocks that always make progress.

Example (Resource Sharing)

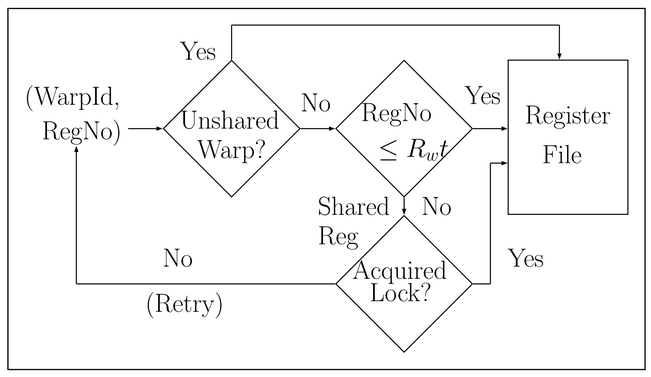


Note: In register sharing, pair of warps, one from each shared block, share their registers appropriately.

Resource Access Mechanism

A shared warp can access unshared register directly, but it can access shared register only after acquiring an exclusive lock.

R_w: Number of registers required for a warp
t: Threshold, used for computing %ge of resource sharing



Note: The scratchpad access mechanism is implemented in the similar way The details of our approach are discussed in [3].

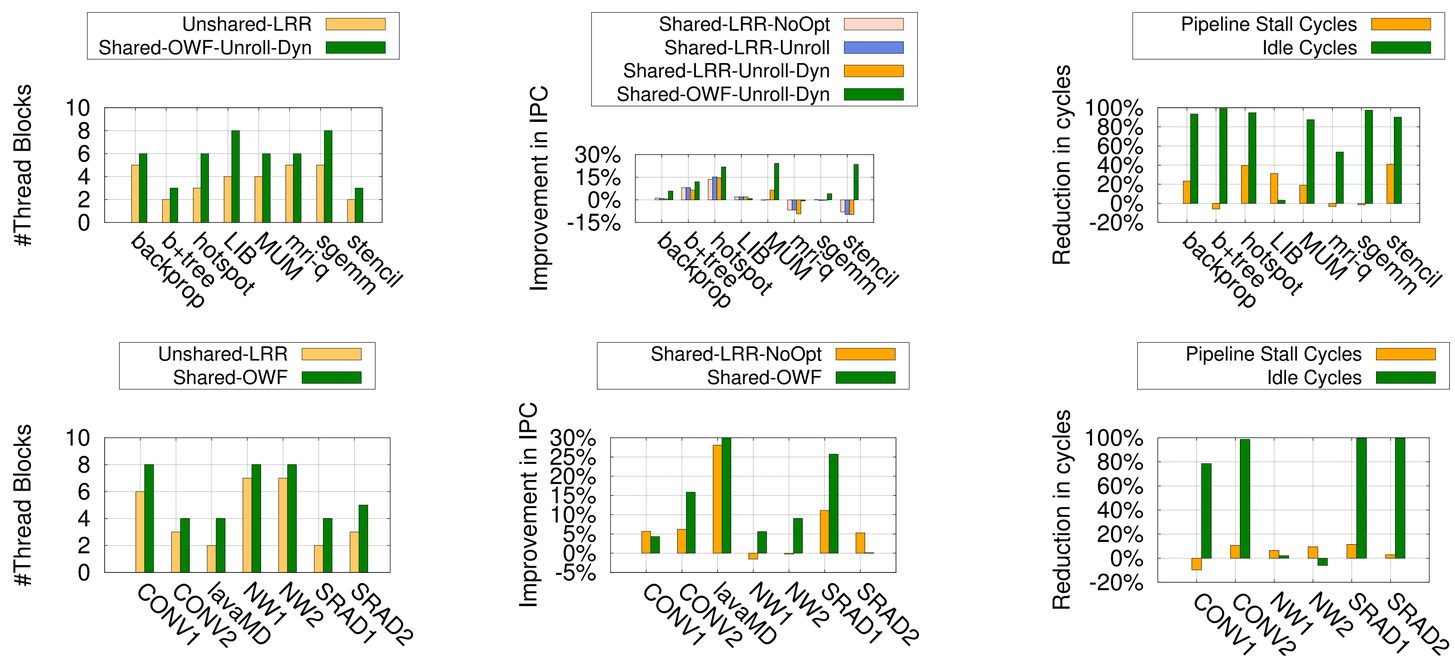
Optimizations

Type of warps in the SM:

- Unshared warps (warps from unshared thread block)
- Owner warps (warps that have exclusive lock)
- Non-owner warps (warps without lock)

1. **Owner Warp First (OWF):**
   - Schedule the warps according to the priority: owner warp, unshared warp, and non-owner warp
2. **Unroll Register Declarations:**
   - Unroll and re-order register declarations to delay access to shared registers
3. **Dynamic Warp Execution (Dyn):**
   - Control the execution of long latency instructions from non-owner warps to reduce cache misses.

Results



Increase in TLP **Improvement in IPC** **Reduction in idle and stall cycles**

Note: First row of the results corresponds to register sharing, second row corresponds to scratchpad sharing

Experimental Setup

Resource	GPU Configuration [2]
No of SMs	14
Max Num of TBs	8
Max Num of Threads	1536
Number of Registers	32768
Scratchpad Memory	16KB
Warp Scheduling	LRR

References

[1] CUDA C Programming Guide. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/`.

[2] GPGPU-Sim. `http://www.gpgpu-sim.org`.

[3] V. Jatala, J. Anantpur, and A. Karkare. Improving GPU Performance Through Resource Sharing. CoRR, http://arxiv.org/abs/1503.05694, 2015.