Task Management for Irregular Workloads on the GPU

UCDAVIS
UNIVERSITY OF CALIFORNIA

Stanley Tzeng, Anjul Patney, and John D. Owens

Introduction

Motivation:
We explore software mechanisms for managing irregular tasks on graphics processing units. Traditional GPU programming guidelines teaches us how to efficiently program the GPU for *data parallel* pipelines with regular input and output. We present a strategy for solving *task parallel* pipelines which can handle irregular workloads on the GPU.

Four Key Concepts:
- **Warp size work granularity** – 1 warp processes 1 task.
- **Persistent thread scheduler emulation** – threads return to the top of the kernel to fetch more work.
- **Uberkernel processor utilization** – Combine multiple pipeline stages into one kernel to eliminate explicit kernel barriers.
- **Task donation memory management** – each block has its own dequeue of work to be processed, when the dequeue is full, it may spill over to other blocks.

Application Example:
To demonstrate our task parallel system in action, we developed a real time Reyes rendering engine to showcase our work.

References

ARORA N. S., BLUMOFE R. D., PLAXTON C. G.: Thread scheduling for multiprogrammed multiprocessors. In Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures (June/July 1998), pp. 119–129

AILA T., LAINE S.: Understanding the efficiency of ray traversal on GPUs. In Proceedings of High Performance Graphics 2009 (Aug. 2009), pp. 145–149.

BLUMOFE R. D., LEISERSON C. E.: Scheduling multithreaded computations by work stealing. Journal of the ACM 46, 5 (Sept. 1999), 720–748.

CEDERMAN D., TSIGAS P.: On dynamic load-balancing on graphics processors. In Graphics Hardware 2008 (June 2008), pp. 57–64.

HEIRICH A., ARVO J.: A competitive analysis of load balancing strategies for parallel ray tracing. The Journal of Supercomputing 12, 1-2 (Jan. 1998), 57–68.

TATARINOV A., KHARLAMOV A.: Alternative rendering pipelines using NVIDIA CUDA. Talk at SIGGRAPH 2009, http://developer.nvidia.com/object/siggraph-2009, Aug. 2009.

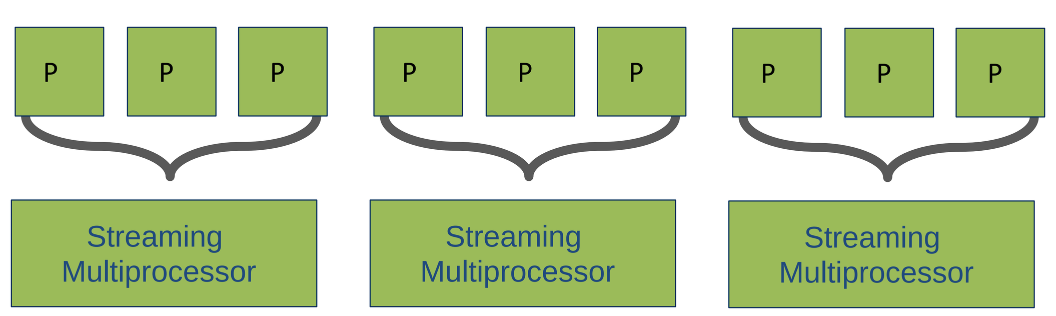
ZHOU K., HOU Q., REN Z., GONG M., SUN X., GUO B.: RenderAnts: Interactive Reyes rendering on GPUs. ACM Transactions on Graphics 28, 5 (Dec. 2009), 155:1–155:11.

Warp Size Work Granularity

Problem: We want to emulate task level parallelism on the GPU without loss in efficiency.

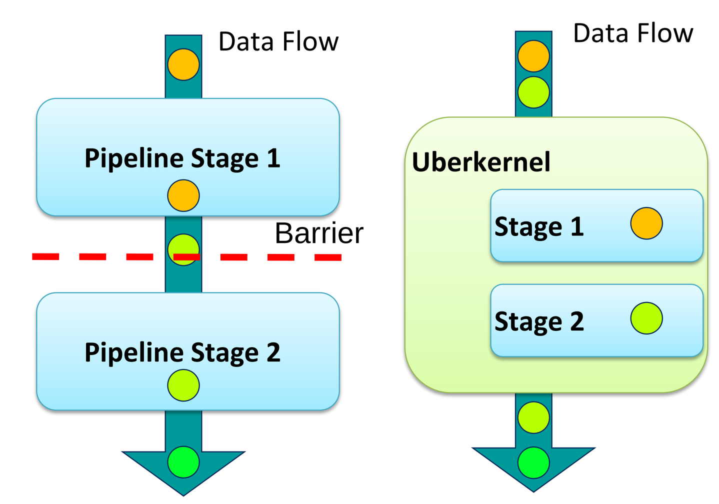
Solution: We choose block sizes of 32 threads / block. Choosing this size has two major advantages for us:

- Removes messy synchronization barriers. This is especially important when used in conjunction with Persistent Threads.
- Can view each block as a MIMD thread where the execution is independent of other blocks, maintaining the efficiency of SIMD execution while giving us MIMD style granularity. We call these blocks processors. Several processors can reside on a SM.



Uberkernels

Problem: Want to eliminate global kernel barriers for better processor utilization.
Solution: Use the Uberkernel programming model. Uberkernels pack multiple execution routes into one kernel, effectively going through multiple stages of the pipeline within one kernel. When can pack multiple kernels into one kernel, we eliminate the explicit barrier and overhead between kernel launches.



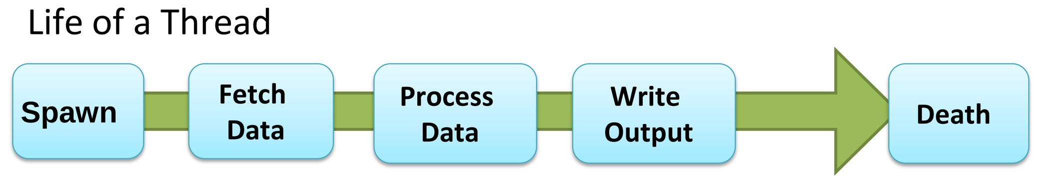

```
switch(data.inst)
{
case KERNEL1:
  runKernel1();
  break;
case KERNEL2:
  runKernel2();
  break;
}
```

A GPU Task-Based Irregular Workload Model

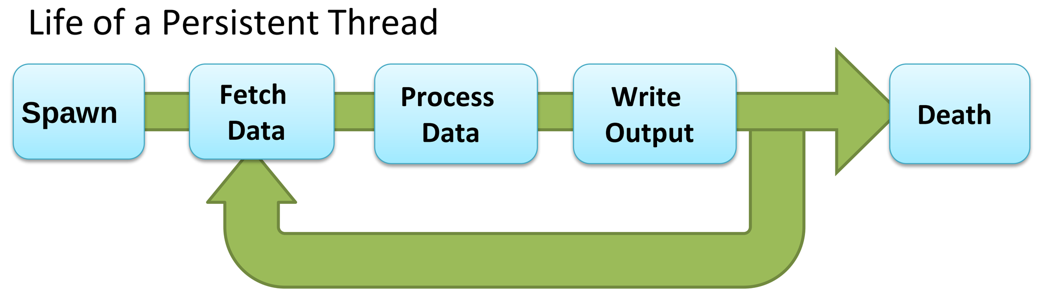
Persistent Threads

Problem: If input is irregular? How many threads do we launch?
Solution: Launch enough to keep the GPU busy, and keep them alive so they keep fetching work. This allows irregular workloads to be generated and processed all within the same kernel.

Life of a Thread



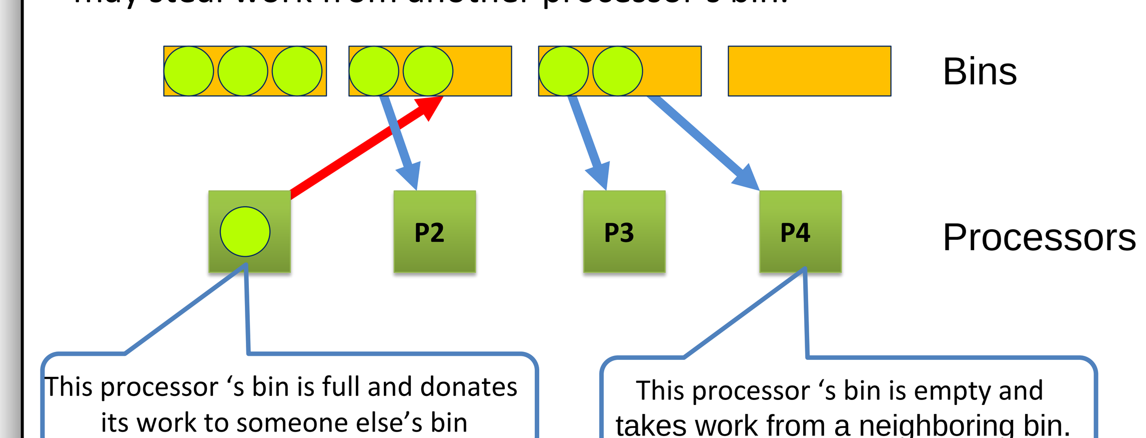
Life of a Persistent Thread



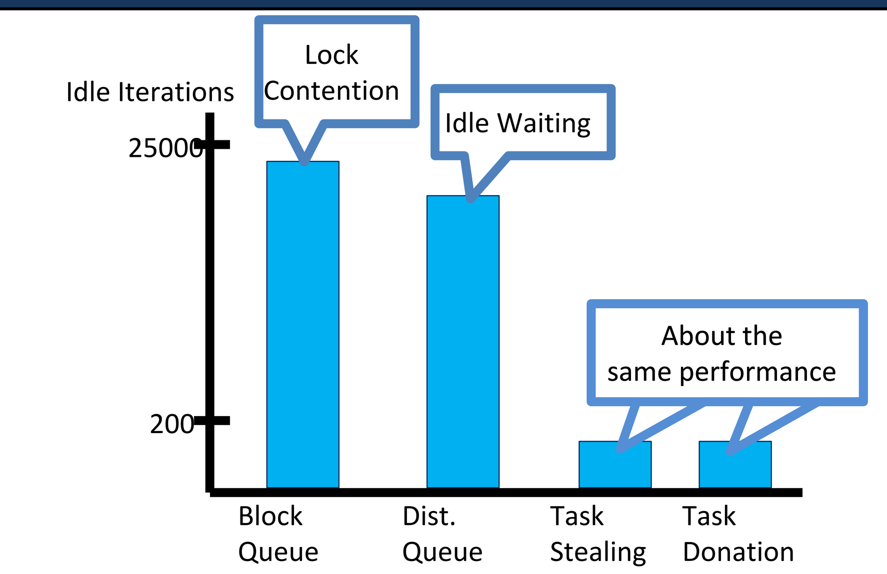
Go back and keep fetching work until there is no more work left!

Task Donation Memory Management

Problem: We need to ensure that our processors are constantly working and not idle.
Solution: Design a software memory management system. The goal of our memory management system is to ensure that all processors can get work quickly.
Task Donation:
We develop a task donation memory management system. Each processor is given a dequeue (called a bin)in which it takes and stores work units, when that bin is full, the processor may spill generated work to another processor's bin. When a processor's bin is empty, it may steal work from another processor's bin.



Results



This graph shows our task donation memory scheme in terms of idle iterations. We define an idle iteration as any one processor either waiting for a lock or waiting for other processors to finish before it can terminate. Our scheme is roughly the same as a previous task stealing scheme, but it uses less memory. For more experiments and results, please see our paper.

REYES:
As an application of our work, we demonstrate how our system can be used to implement an alternative rendering pipeline on the GPU: the Reyes Pipeline. Its highly irregular workload in several stages of its pipeline forms an ideal testing ground for our work. We are able to achieve real time frame rates of ~20fps on the majority of our models using a single GeForce GTX 280. Please see our talk, "Real-time Reyes: Programmable Rendering on Graphics Processors" on Wednesday 5pm.

Conclusion

We have demonstrated how to build a system on the GPU that can deal with irregular workloads on a task size granularity. With our system we built a Reyes rendering pipeline which can achieve real time framerates.

Novelty:
Our work is the first to combine a work-donation approach for work queue management with Uberkernel and persistent thread programming styles to exploit task parallelism and handle irregular workloads.

Future Work:
In the future we hope to explore how newer GPU hardware can help our hardware, asFermis are known to have much faster atomics than its predecessor. We hope that these advancements in hardware will allow us to explore further abstractions and models for designing different pipelines on the GPU.