GPU Task-Parallelism: Primitives and Applications

Stanley Tzeng, Anjul Patney, John D. Owens
University of California at Davis
This talk

• Will introduce task-parallelism on GPUs
  – What is it?
  – Why is it important?
  – How do we do it?

• We will discuss
  – Primitives
  – Applications
What is Task Parallelism

• **Task:** A logically related set of instructions executed in a single context.

• **Task-parallelism:** Tasks processed concurrently
  – A scheduling component figures out how to distribute tasks to available computing resources

• **Examples:** Cilk, Intel TBB, OpenMP

Task graph of Frostbite Engine
GPU task parallelism: Why NOT?

- GPUs are data-parallel
  - GPU hardware built around data-parallel processing
  - CUDA is a data-parallel abstraction

- Task-based workloads are ignored (so far)
Task Parallelism on GPUs
GPU task parallelism: Why?

- Extends the scope of GPU programming
- Many task-parallel problems still exhibit ample amount of parallelism
- This lecture: programming the GPU as a task-parallel device
- Split into two parts: **Primitives** and **Applications**

You after this lecture
Primitives: Goals
Build a task-parallel system that can:

- Handle divergent workflows
- Handle irregular parallelism
- Respect dependencies between tasks
- Load balance all of this
Primitives: Goals

Build a task-parallel system that can:

• Handle divergent workflows
• Handle irregular parallelism
• Respect dependencies between tasks
• Load balance all of this
Primitives: Outline

CUDA Task-Parallel Model

- Task Granularity
- Task Communication
- Task Management (launch, retire)
- Task Dependency
Task Granularity

What is the right parallel granularity to handle tasks?
• One task per thread? Okay
• One task per warp? Better

Become SIMD-aware! Think of warps as MIMD threads with a 32-wide vector lane.
Task Management

How do we keep processing tasks until there are none left?

• Persistent thread programming model.
• while(stillWorkToDo){ // run task }
• Decouples launch bounds from amount of work.

Beware of deadlocks!
Task Communication

How do we distribute tasks evenly between SMs?

• Distributed queues with a work donation routine (see GTC 2010)
• A single block queue: atomics are now fast enough and this is simple enough
Task Dependency

• What if tasks had dependencies?
• How do we augment our current system to respect dependencies?
Dependency Resolution

• All tasks put on our queue are executed without notion of dependencies.

• Dependencies affect which tasks can be placed in the work queue.

• Maintain a task dependency map that each warp must check before queuing additional work.
while (Q is not empty)
{
    task t = Q.pop()
    Process (t)
    Neighbors tnset = dependencyMap(t)
    For each tn in tnset
        tn.dependencyCount--; 
        if (tn.dependencyCount == 0) Q.push(tn);
}
Applications

• A variety of scenarios demand task-parallelism
• We will discuss three
  – Reyes Rendering
  – Deferred Lighting
  – Video Encoding
• We only use primitives that are necessary
Application: Reyes
Application: Reyes

• Why we need task-parallelism
  – Irregular parallelism
  – Dynamic communication

• What primitives do we need
  – Persistent Threads
  – Dynamic Task-Queues
Application: Deferred Lighting

• Different lights affect different parts of the screen
• So we subdivide tiles with too many lights
• Original idea by Lauritzen and at DICE
Application: Deferred Lighting

• Why we need task-parallelism
  – Irregular parallelism
  – Dynamic communication

• What primitives do we need
  – Persistent Threads
  – Dynamic Task-Queues
Application: Video Encoding

• H.264 Encoding using GPUs
  – Past work on inter-prediction
  – We consider intra-prediction

• We show both using task-parallelism

• Consider 16x16 macroblocks
  – Each forms a task
Application: Video Encoding

• Why we need task-parallelism
  – Task dependencies
  – Dynamic communication

• What primitives do we need
  – Dependency resolution
Intra Prediction

- Intra prediction has dependencies between tasks
Summary

• Task parallelism is important
  – Many application scenarios

• Several fundamental primitives
  – Braided task-granularity
  – Persistent threads
  – Dynamic queuing
  – Dependency resolution
Thanks!

For papers on these works, please visit
http://csiflabs.cs.ucdavis.edu/~stzeng/
Backup slides
Optimization?

• What if:
  – The number of tasks is fixed, but cannot be executed because of dependencies.

• Introducing Static Dependency Resolution!