How to Write a Parallel GPU Application Using CUDA and Charm++

Presented by Lukasz Wesolowski

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

PARALLEL PROGRAMMING LAB
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS
PPL UIUC

Outline

- GPGPUs and CUDA
- Requirements for a GPGPU API (from a Charm++ standpoint)
- CUDA stream approach
- Charm++ GPU Manager

General Purpose GPUs

- Graphics chips adapted for general purpose programming
- Impressive floating point performance
  - 4.6 Tflop/s single precision (AMD Radeon HD 5970)
  - Compared to about 100 Gflop/s for a 3 GHz quad-core quad-issue CPU
- Throughput oriented
- Good for large scale data parallelism

CUDA

- A popular hardware/software architecture for GPGPUs
- Supported on NVIDIA GPUs
- Programmed using C with extensions for large-scale data parallelism
- CPU is used to offload and manage units of GPU work

API Requirements

- GPU operations should not block the CPU
  - blocking wastes CPU cycles and reduces response time for messages
- Chares should be able to share the GPU without synchronizing with each other

Direct Approach

- User makes CUDA calls directly in Charm++
- CUDA Streams
  - allow specifying an order of execution for a set of asynchronous GPU operations
  - Operations in different streams can overlap in execution
- User assigns a unique CUDA stream for each chare and makes polling or synchronization calls to determine completion of operations

Problems with Direct Approach

- Each chare must poll for completion of GPU operations
  - Tedious
  - Inefficient
- Streams need to be carefully managed to allow overlap of GPU operations

Stream Management

- Common stream usage

  CPU → GPU data transfer

  kernel_call

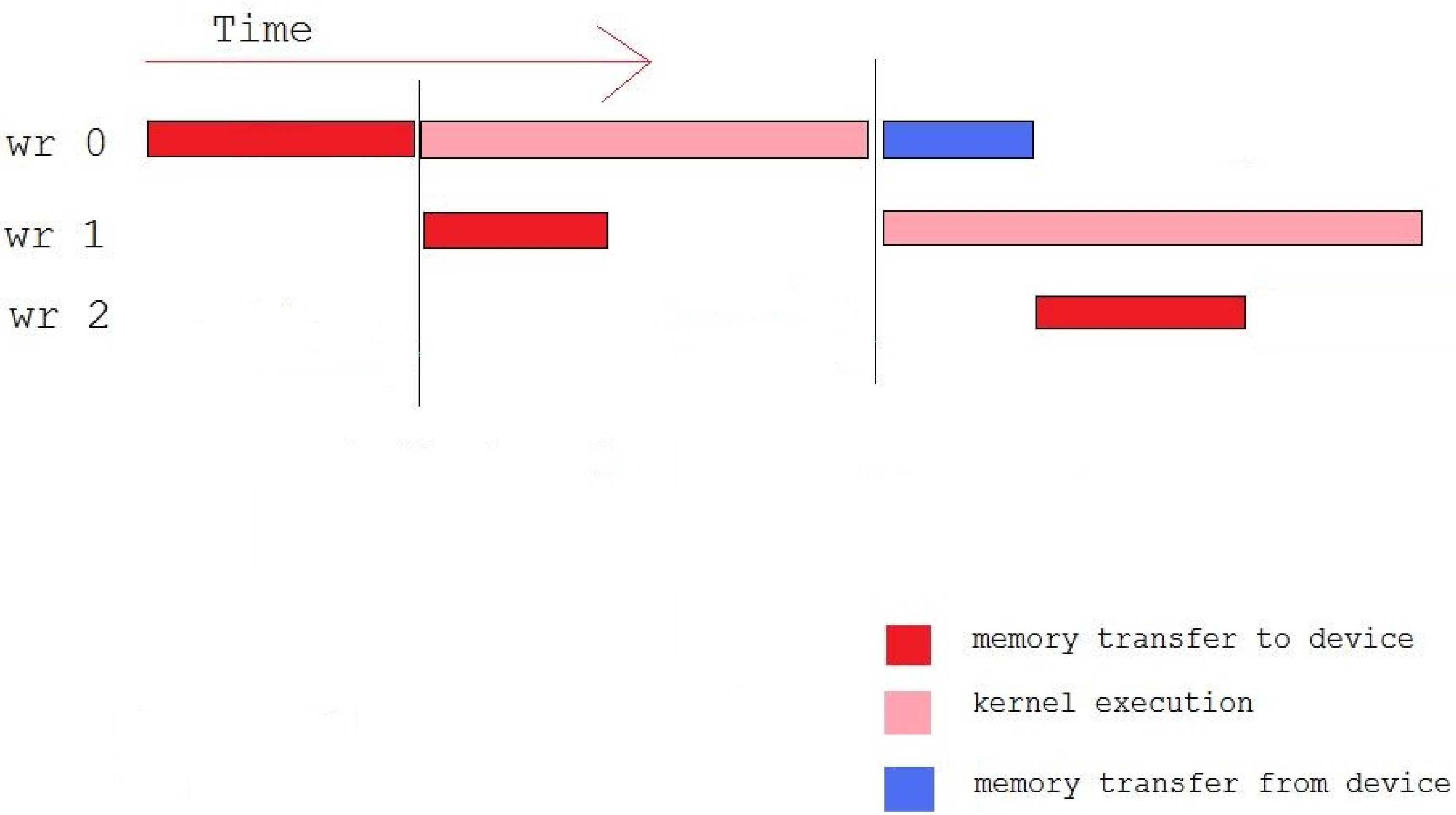
  GPU → CPU data transfer

- Third operation blocks DMA engine until kernel is finished
- Can be avoided by delaying GPU → CPU data transfer until kernel is finished
  - Requires an additional polling call

Overview of GPU Manager

- User submits requests specifying work to be executed on the GPU, associated buffers, and callback
- System transfers memory between CPU and GPU, executes request, and returns through a callback
- GPU operations performed asynchronously
- Pipelined execution

Execution of Work Requests



GPU Manager Advantages

- No polling calls in user code
  - Simpler code
  - More efficient
- System ensures overlap of GPU operations
  - Scheduling of pinned memory allocations
- GPU profiling in Projections