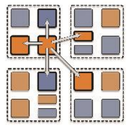
Runtime Support for Concurrent Execution of Overdecomposed Heterogeneous Tasks

Jaemin Choi, Laxmikant V. Kale (advisor), University of Illinois Urbana-Champaign

Overview

- Rise of heterogeneous systems in HPC
- Runtime support to enable concurrent execution of heterogeneous tasks in Charm++
- Performance evaluation
  - `busywait`: up to **4.79x** speedup
  - `stencil2d`: up to **2.75x** speedup

Charm++

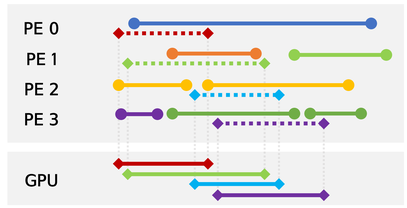


- Migratable object and task-based programming model
- **Adaptive runtime system**
- Decompose problem domain into communicating **objects**
- Method invocation = **task**
- **Overdecomposition:**
  more objects than PEs (CPU cores)
  → many CPU/GPU tasks per PE
- Intelligent scheduling decisions [1]

Issues with Overdecomposed Heterogeneous Tasks

1. **Concurrent execution of GPU tasks**
   - Probability of concurrency increases with number of PEs & duration of kernels
   - Can utilize CUDA's concurrent kernel execution feature and assign one or more streams per object
   - But degree of concurrency limited to the number of PEs

2. **Concurrent execution of both CPU/GPU tasks**
   - When a GPU task executes, any task of a different object should be able to execute
   - We want all CPU cores busy with CPU tasks, and GPU fully utilized by GPU tasks



* 2 objects per PE, each object is color coded

   - **Problem:** `cudaStreamSynchronize()` blocks the CPU
   - No other task is able to execute
   - Number of concurrent GPU tasks limited to the number of PEs

Runtime Support

- Candidate features:
  CUDA Callback, CUDA Event
- Single CUDA callback thread becomes a bottleneck when scaling
- **CUDA Event-based solution**
  - Enables polling for GPU task completion
  - Impractical for the programmer to implement directly: unclear where to integrate polling and how frequent polling should occur
  - Runtime: 1 event queue per PE
  - Scheduler polls queue before selecting a task
  - Charm++ callback invoked when polled event is marked as complete

Experimental Results

- Tested on a single compute node of OLCF Titan
- Offload percentage: percentage of GPU tasks

1. `busywait`

- Designed to validate proposed scheme
- Evaluate effectiveness with varying offload percentage and task duration
- Tasks busy-wait to simulate work
- Configurations of busy-wait time per iteration
  - CPU 10 ms, GPU 10 ms
  - CPU 10 ms, GPU 1 ms
  - CPU 1 ms, GPU 10 ms
- 8 PEs, 16 objects per PE
- 512 threads per kernel → 32 concurrent kernels with runtime support (vs. 8 without)
- 100 iterations
- Up to **4.79x** speedup compared to CUDA without runtime support
- Effectiveness of runtime support depends on application characteristics

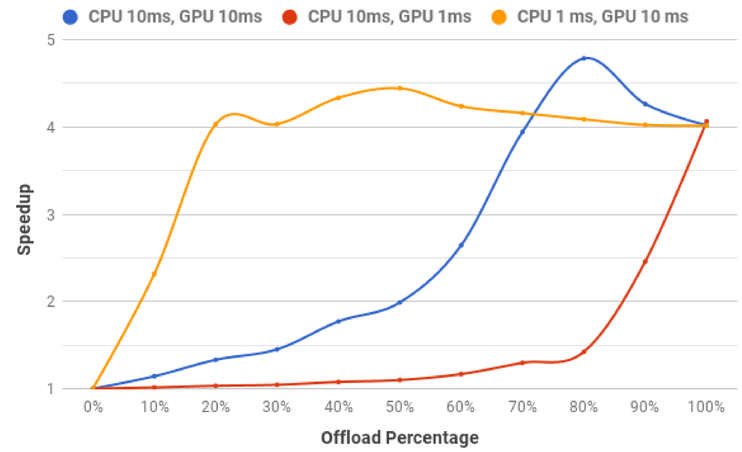


Figure 1. Speedup of `busywait` benchmark

2. `stencil2d`

- 2D 5-point iterative stencil benchmark
- Evaluate effectiveness under realistic workload
- 16384 x 16384 simulation space
- Decomposed into 512 x 512 blocks (objects)
- 8 PEs, 128 objects per PE
- Each object utilizes 64 threads
  → 32 concurrent kernels with runtime support
- 100 iterations
- Stencil operation 2x slower on CPU
  (refer to offload percentages of 0% and 100%)
- Up to **2.75x** speedup at 50% offload percentage

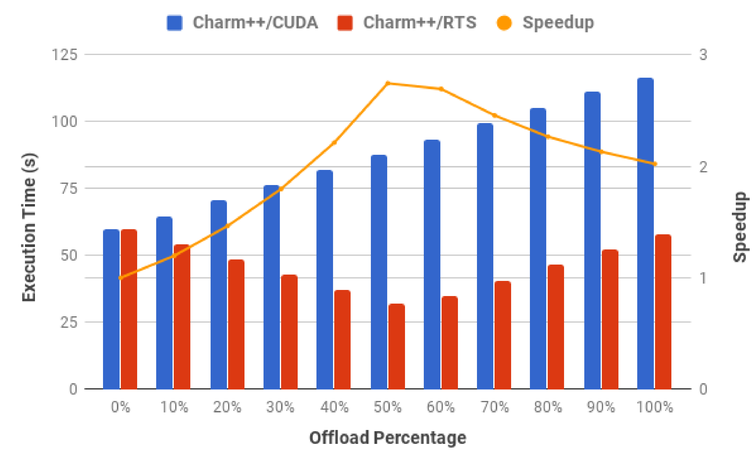


Figure 2. Execution time and speedup of `stencil2d` benchmark

[1] J. Lifflander, G. C. Evans, A. Arya, and L. V. Kale. 2012. Dynamic Scheduling for Work Agglomeration on Heterogeneous Clusters. In 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum. 2404–2413.