Achieving Computation-Communication Overlap with Overdecomposition on GPU Systems

Jaemin Choi

PhD Candidate in Computer Science
University of Illinois Urbana-Champaign

Nov 11, 2020
ESPM2 Workshop at SC’20
Overview

• Increasing gap between single-node computational power and inter-node communication performance on leadership-class systems e.g. OLCF Titan → Summit: 1.4 → 40 TFLOPS, 6.4 → 23 GB/s

• Can be tackled from at least 2 directions
  1. Improve communication performance itself with software optimizations and better utilization of hardware support (e.g. GPUDirect, SHARP, hardware tag-matching)
  2. Reduce impact of communication on overall performance (e.g. communication-avoiding algorithms, computation-communication overlap)

• Focus on achieving computation-communication overlap on GPU systems with overdecomposition using Charm++ ([https://github.com/UIUC-PPL/charm](https://github.com/UIUC-PPL/charm))
Overdecomposition

Per-process decomposition (MPI)

Overdecomposition (Charm++)

4 CPU cores
Asynchronous Message-Driven Execution

Chares

Processing Element (PE)  Message queue
GPU Execution in Charm++

1. Asynchronously offload work to GPU

2-1. Scheduler progresses communication & executes next chare
2-2. GPU work completes, runtime enqueues new message

3. Work that depends on the completed GPU work can continue (e.g. another method of the original chare)
Achieving Computation–Communication Overlap

1. **Support asynchronous progress in the runtime**
   - Avoid synchronization CUDA APIs (e.g. cudaStreamSynchronize)
     - Charm++ scheduler blocked from performing other chares’ work
     - Cannot make forward progress on communication (without comm. threads)
   - Directly using CUDA async APIs to determine completion is infeasible
     - Scheduler-driven execution in Charm++
     - CUDA-generated thread disassociated from the Charm++ runtime
   - `hapiAddCallback(cudaStream_t stream, CkCallback* callback)`
     - Allows user to schedule a Charm++ callback to be invoked when GPU operations complete in the specified CUDA stream
     - Two compile-time configurable mechanisms based on CUDA Callback and CUDA Events (default)
2. **Prioritize communication-related GPU operations in the application**
   - Single CUDA stream per chare: delays in communication-related operations (host-device data transfers, packing/unpacking kernels) due to computational kernels offloaded from other chares to the same GPU
   - Need separate streams for compute and communication (with higher priority for communication)
   - More complex design may be necessary, as for MiniMD (described in paper)
Achieving Computation-Communication Overlap

(a) Single CUDA stream per chare. Communication is delayed by a computational kernel enqueued from another chare, causing idle time between iterations.

(b) Separate compute/communication CUDA streams per chare, with the communication stream given higher priority. Iterations continue without idle times in between.

Fig. 3. Execution timelines of Jacobi2D with four chares mapped to a single GPU.
Evaluation Platforms

- OLCF Summit
  - 6 NVIDIA Tesla V100s per node

- LLNL Lassen
  - 4 NVIDIA Tesla V100s per node

- 1 process with 1 PE/core per GPU
  - e.g. 6 PEs and 6 GPUs per compute node on Summit
Iterative proxy apps

**Jacobi3D**
- Jacobi iteration performed on 3D grid, overdecomposed into chares
- Near-neighbor exchange of halo data (up to 6 neighbors)

**MiniMD**
- Proxy app for LAMMPS molecular dynamics code
- Converted MPI + Kokkos to Charm++ + Kokkos
- CUDA-aware MPI converted to explicit host-device transfers and host messages
- Kokkos responsible for computational kernels and host-device data movement
- Neighbor exchange of atoms, Lennard-Jones force calculation
Performance Results – Jacobi3D

Fig. 5. Performance of Jacobi3D with varying overdecomposition factors on a single node of OLCF Summit.
Performance Results – Jacobi3D

Fig. 6. Weak & strong scaling performance of Jacobi3D.
Performance Results – MiniMD

Fig. 7. Weak & strong scaling performance of MiniMD.
Conclusion

• Up to 50% and 47% improvement in overall performance with Jacobi3D and MiniMD, respectively

• With careful design of the application to prioritize communication and support for asynchronous progress of GPU work in the runtime system, overdecomposition can significantly improve performance, especially in weak scaling

• Combining Charm++ (for overdecomposition and communication) with performance portability models (e.g. Kokkos) can be a good path forward for upcoming vendor-heterogeneous GPU systems
Ongoing & Future Work

• Support for direct GPU-GPU transfer in Charm++
  • Release 6.11: CUDA backend for within-node messaging
  • Exploring UCX, NCCL backends to support inter-node messaging

• Mitigating fine-grained overheads
  • Improve effect of overdecomposition with strong scaling
  • Kernel aggregation

• Dynamic load balancing with GPU loads
Thank you! Questions?

Please feel free to reach out to jchoi157@illinois.edu.