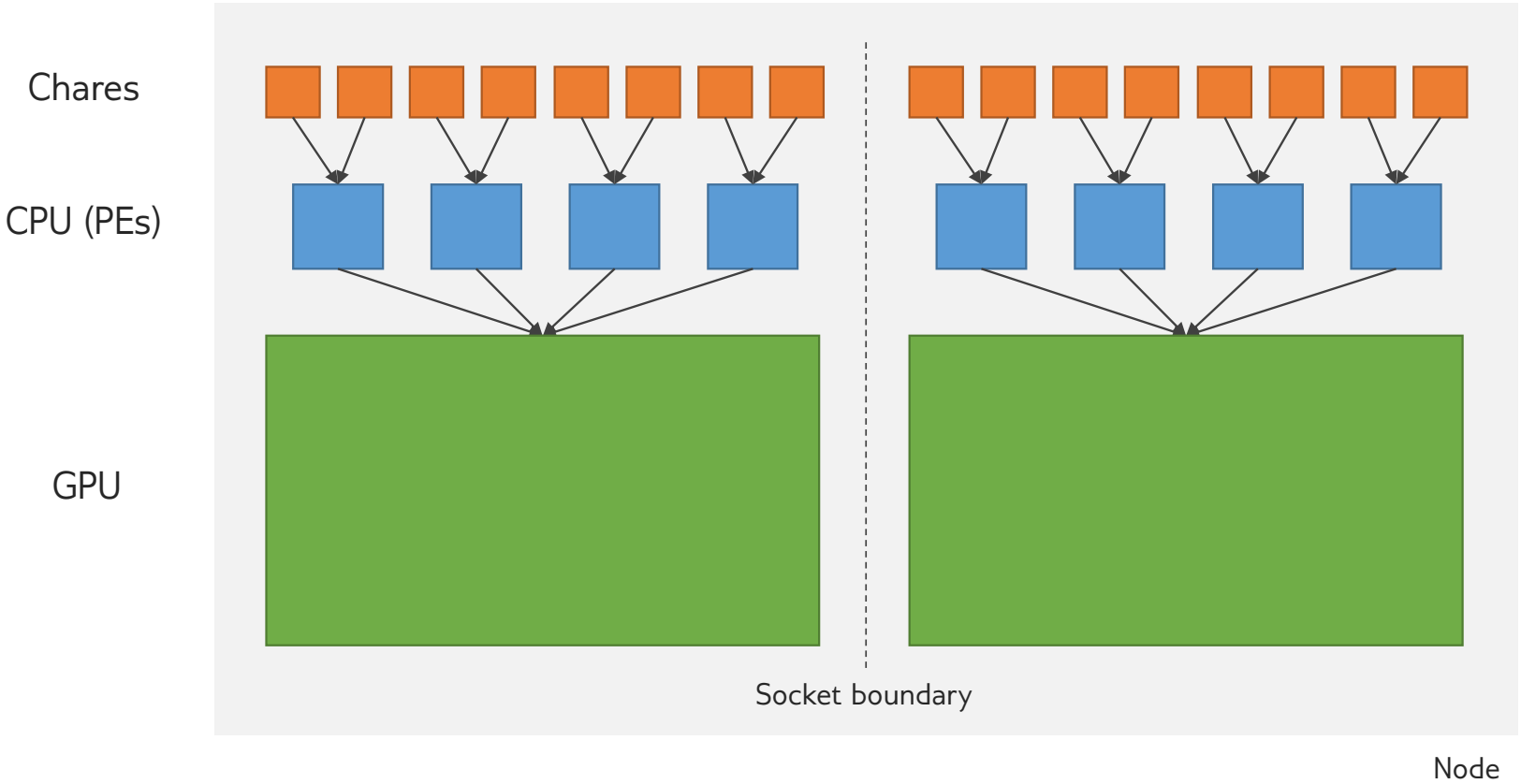
Messaging with GPU-resident Data

Jæmin Choi

Parallel Programming Laboratory
University of Illinois Urbana-Champaign

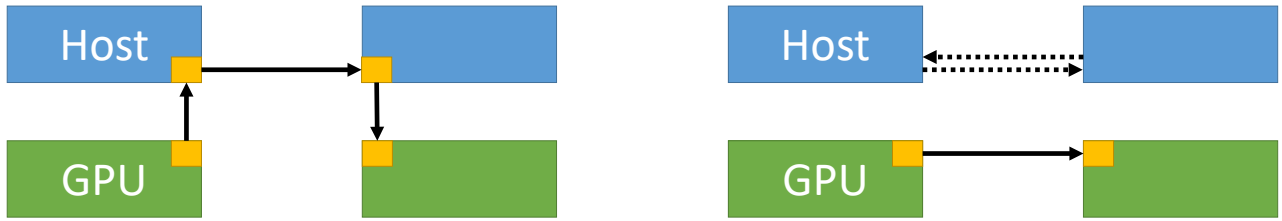
Charm++ Birds of a Feather, SC'19

Charm++ on a Multi-GPU System



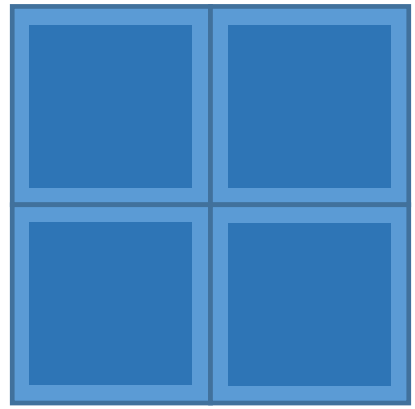
- What if a chare wants to send data to another chare?

GPU Messaging



- Sending data from one chare to another
  - With *GPU-resident* data
- Current: D2H transfer $\rightarrow$ Charm++ message $\rightarrow$ H2D transfer
- **Want to bypass host memory!**
  - Need RDMA-like behavior
  - Assume single-host, multi-GPU
- Sender sends device pointer $\rightarrow$ Receiver initiates D2D transfer $\rightarrow$ ACK to sender

GPU Messaging: 2D Stencil



- Configurations
  - Grid: 16,384 x 16,384
  - Block: 8,192 x 8,192 (64KB halo data)
  - 4 chares, 1 block per chare
  - LLNL Lassen: 4 PEs, 4 GPUs (1 PE per GPU)
  - No overdecomposition
- Comparison
  1. (Current) Host-device transfers + message containing data
  2. (New) Peer device transfer with metadata/ACK messages

GPU Messaging: 2D Stencil

- Preliminary results
  - Aggregate time for 100 iterations[1]
  1. Current method
     - D2H: 127 ms
     - H2D: 19 ms
     - + Messages containing halo data
  2. New method
     - D2D: 15 ms
     - + Metadata/ACK messages
- Need analysis of Charm++ message times, but significant reduction in GPU transfer times

[1] 800 halo transfers: 2 neighbors x 4 chares x 100 iterations

Updates in Charm++ 6.10

- Dynamic allocation of pinned host memory pool
- Restore asynchronous data transfers for WorkRequest API

Upcoming Features

- Direct GPU messaging
- Support for load balancing chares with GPU-resident data
- Measurement of GPU load $\rightarrow$ CPU + GPU load balancing
- Other improvements to asynchronous execution

Conclusion

- Many features and fixes in the pipeline
- Request features & report bugs to **Github Issues** (https://github.com/UIUC-PPL/charm/issues)

GPU support