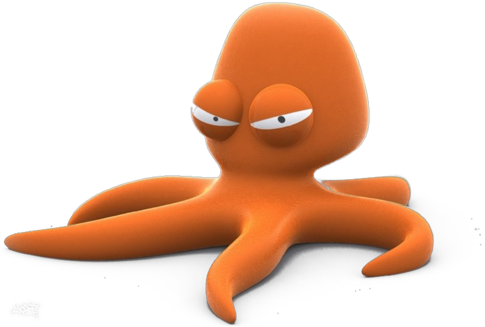
MPCDF
MAX PLANCK
COMPUTING & DATA FACILITY

Octopus on GPUs

Sebastian Ohlmann
Max Planck Computing and Data Facility, Garching

Octopus developers workshop, 09.2021



Why GPUs?

- Slower increase in CPU efficiency in last years
- Higher power efficiency of GPUs
  - Cobra CPU: 3.4 GFlops/W
  - Raven GPU: 22.9 GFlops/W
- At MPCDF:
  - Raven system
  - EOS successor

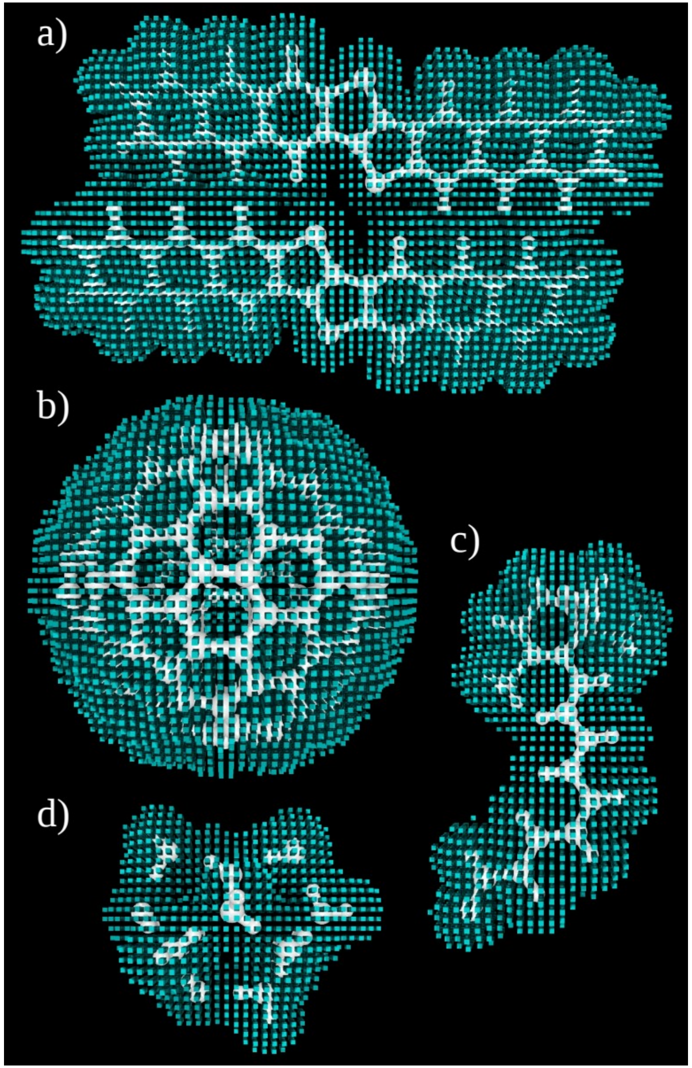
→ prepare now!

Octopus: GPU version

- Original implementation by Xavier (paper: 2013)
- Latest improvements: mainly me, Martin, Nicolas
- Written in OpenCL + wrapper for CUDA
  - OpenCL deprecated (required libraries not maintained anymore…)
- Most important features supported
  - GS: use RMMDIIS
  - TD: most efficient

Data layout

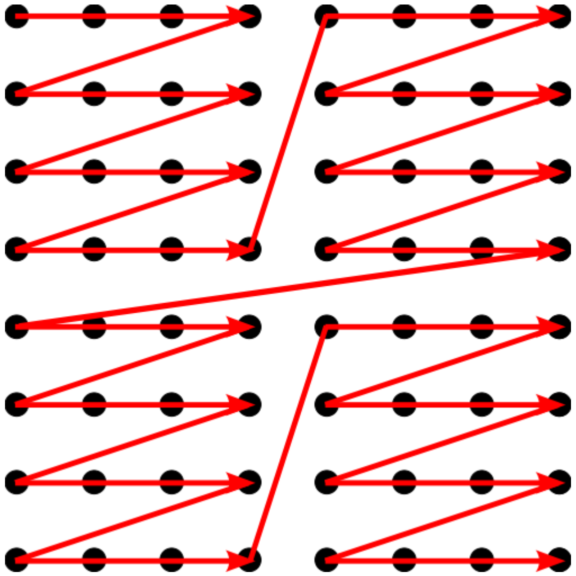
- Real-space grid for FD
- Complicated shape possible, e.g. molecules



X. Andrade & A. Aspuru-Guzik, J. Chem. Theory Comput. (2013), 9, 10, 4360-4373

Data layout

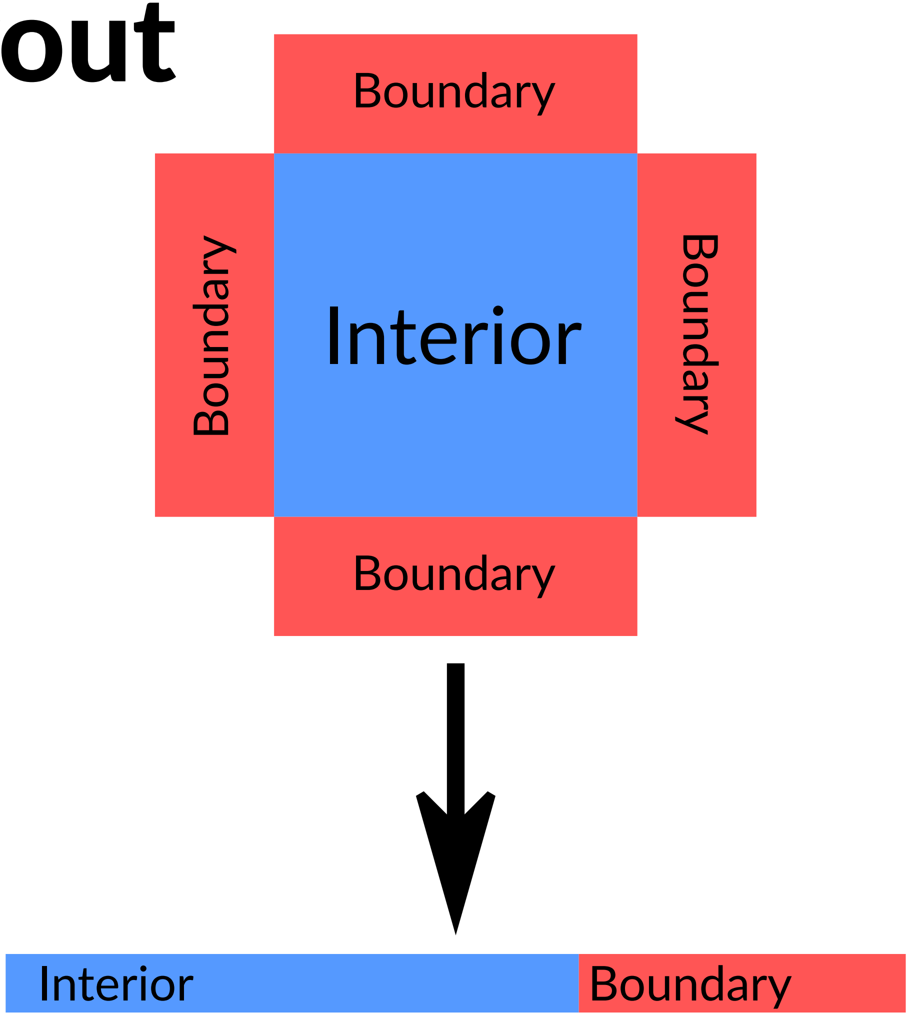
- Real-space grid for FD
- Complicated shape possible, e.g. molecules
- Cache-aware mapping to 1D array



X. Andrade & A. Aspuru-Guzik, J. Chem. Theory Comput. (2013), 9, 10, 4360-4373

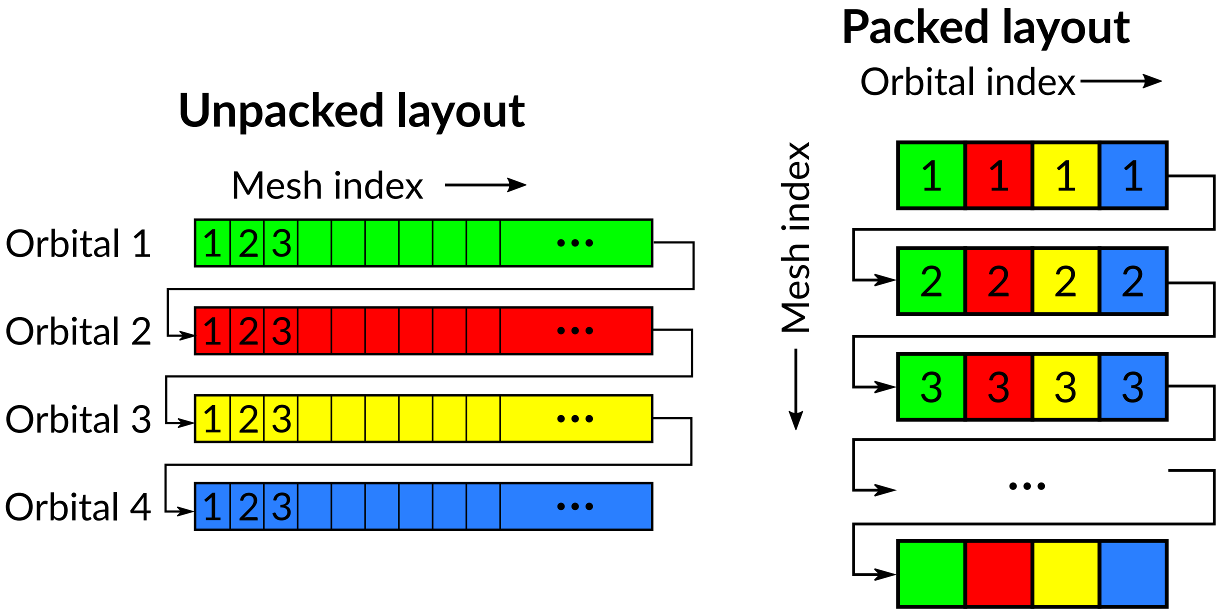
Data layout

- Real-space grid for FD
- Complicated shape possible, e.g. molecules
- Cache-aware mapping to 1D array
- 1D data layout: 2 blocks
  - Interior points
  - Boundary/ghost points



Data layout II: batches

- Aggregate several orbitals into one batch
- Operations done over batches
- 2 layouts:
  - Unpacked
  - Packed → vectorization, GPUs

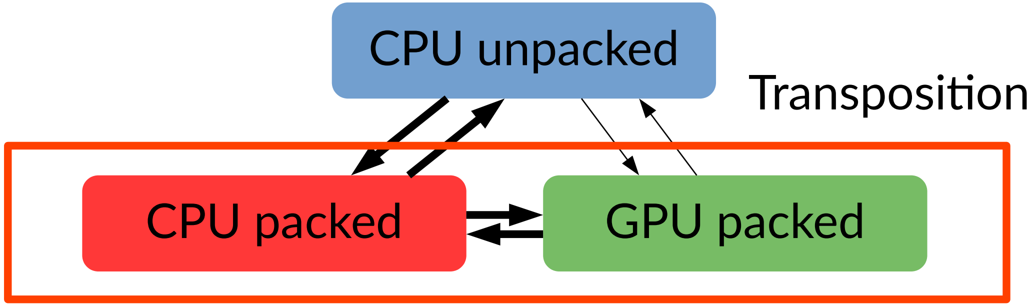


Batch handling

- Batches can have 3 states:

CPU unpacked   CPU packed   GPU packed

- Transitions



Transposition needed

Default, most efficient

Simple copy to GPU

Newer features

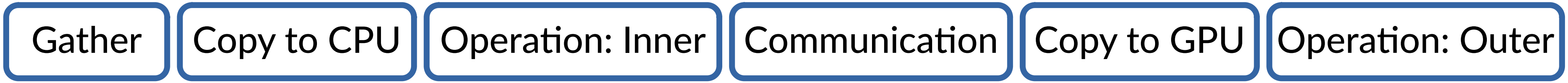
- Pinned memory → faster transfer speed
- Streams → asynchronous operations
- CUDA-aware MPI → GPU-GPU communication
- Prefetching → overlap communication & computation
- New and optimized kernels

CUDA-aware MPI

- Extension of MPI, available for some flavours (OpenMPI, MPICH, MVAPICH, …)
- Requires compatible low-level drivers
- Usage:
  - Pass GPU pointers to MPI calls
  - MPI library can directly access the GPU memory
- Advantages:
  - Peer-to-peer copies on the same node (even better with NVLink)
  - Less latency for inter-node communication
- Needed for efficient domain parallelization!

CUDA-aware MPI in octopus

- Timeline before (domain parallelization):



Gather

Copy to CPU

Operation: Inner

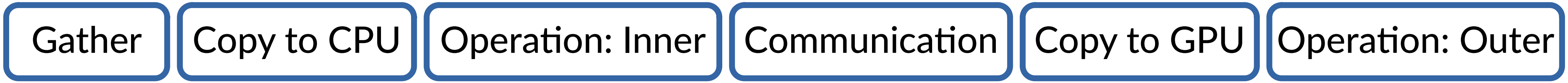
Communication

Copy to GPU

Operation: Outer

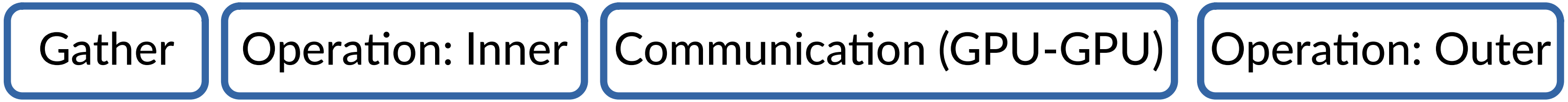
CUDA-aware MPI in octopus

- Timeline before (domain parallelization):



Gather | Copy to CPU | Operation: Inner | Communication | Copy to GPU | Operation: Outer

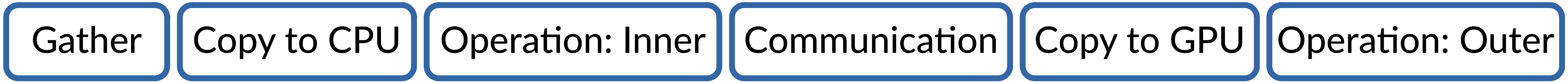
- With CUDA-aware MPI: communication between GPUs → no copies to/from GPU



Gather | Operation: Inner | Communication (GPU-GPU) | Operation: Outer

CUDA-aware MPI in octopus

- Timeline before (domain parallelization):



Gather

Copy to CPU

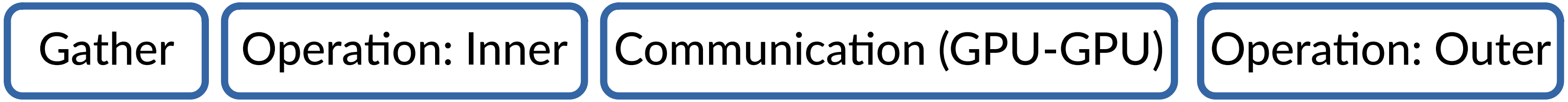
Operation: Inner

Communication

Copy to GPU

Operation: Outer

- With CUDA-aware MPI: communication between GPUs → no copies to/from GPU



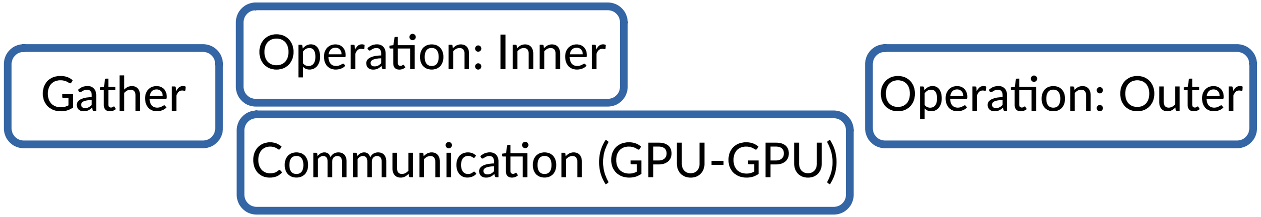
Gather

Operation: Inner

Communication (GPU-GPU)

Operation: Outer

- CUDA-aware MPI + streams: overlap communication & computation



Gather

Operation: Inner

Communication (GPU-GPU)

Operation: Outer

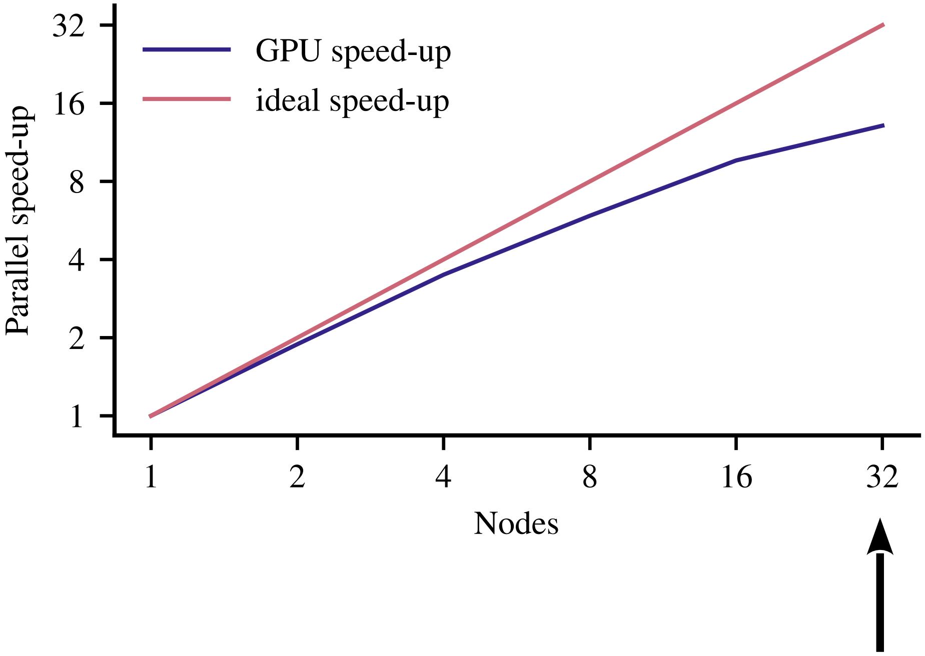
Benchmark results on Raven

- Relevant:
  - Available now to MPSD members
  - Same architecture as EOS successor (early 2022)
- Benchmark problems:
  - Twisted bilayer hBN (2800 states, 3.2M grid points)
  - Adenine with high resolution (44 states, 9.5M grid points)

Raven architecture

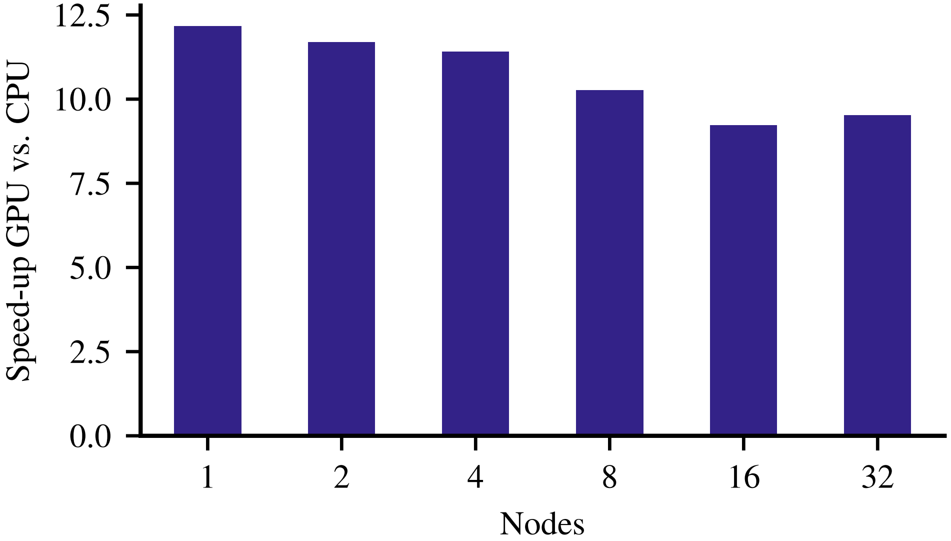
- 1592 nodes with Intel Icelake processors, 72 cores per node
- 192 nodes with 4 Nvidia A100 GPUs
  - Fast NVLink 3 interconnect between GPUs
- Network: Infiniband HDR
  - CPU nodes 100 Gbit/s
  - GPU nodes 200 Gbit/s

hBN i10 on Raven



128 A100 GPUs

GPU run: ~10x faster also when scaling



Adenine on Raven

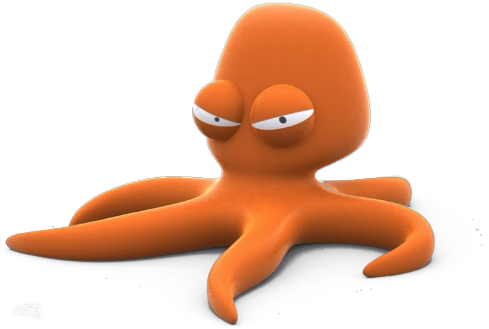
- Domain parallelization
- GPU vs. CPU speed-up on 1 node:
  - With CUDA-aware MPI: **9.5**
  - Without CUDA-aware MPI: **3.5**
- CUDA-aware MPI + communication overlap crucial for domain parallelization!

Outlook: EOS successor

- Similar architecture as Raven
- Better A100 GPU:
  - More RAM (80 GB)
  - Higher memory bandwidth
- Separate system
- Shared with MPI PKS Dresden

Summary

- Octopus now more efficient on GPUs
- Ready to be used in production
  → large resources upcoming
- In case of errors & inefficiencies: tell us!
- Goal: port more parts of the code



Backup slides

Pinned memory

- Normal allocations: pageable memory
- Transfers to GPU: pinned memory needed → faster transfer
- Solution:
  - Allocate pinned memory in C (CUDA call)
  - Use c_f_pointer in Fortran to use this memory
- Transfer speed on PCIe 3: ~12 GB/s vs. ~5 GB/s

Streams

- Default: CUDA operations are blocking
- Streams needed to overlap operations
- Also needed for CUDA-aware MPI
- 32 Streams are initialized in the C layer
- Selection from Fortran layer
- Usage example: asynchronously launch norm kernels with strides

CUDA-aware MPI in octopus

- Implementation:
  - Get pointers to GPU memory from C
  - Use c_f_pointer in Fortran to get a Fortran pointer to this memory
  - Use this Fortran pointer in the MPI calls
- On 8 GPUs with NVLink (machine @ MPSD)
  - Peer-to-peer transfer speed: ~24 GB/s
  - Speed-up of ~ 2.4x vs. normal MPI

Overlap communication & computation

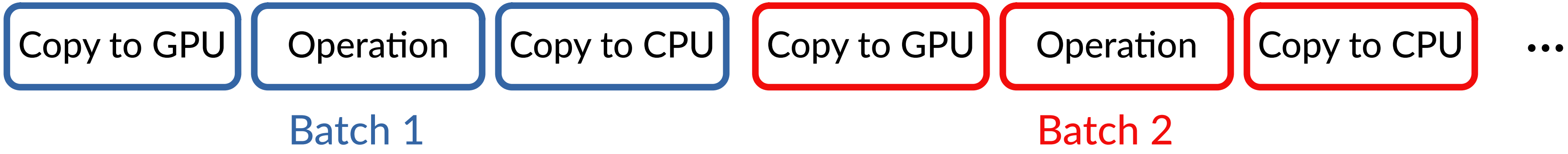
- 2 ways of running octopus on GPUs:
  - If enough GPU memory → store all batches on GPU
  - Otherwise → copy batch to GPU, operate, copy back
- For second way:
  - Overlap of communication & computation possible
  - Use asynchronous prefetching on different stream

Prefetching batches

- Advantage:
  - Hide copy latency, except for first & last copy
- Disadvantages:
  - Needs memory for 3 batches
  - Does not overlap completely if operation involves copies to/from the GPU
- For TD runs: speed-up of 1.8x

Prefetching batches

- Timeline without prefetching:



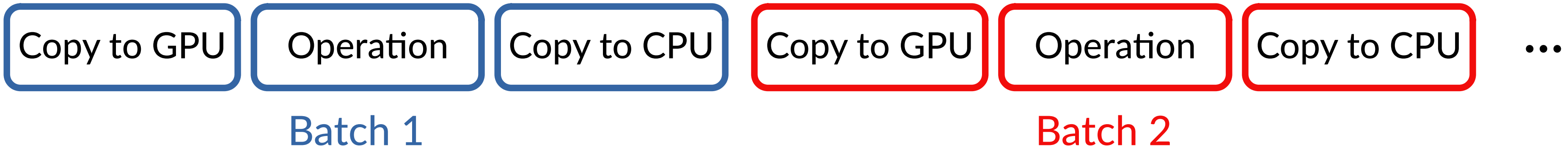
Copy to GPU | Operation | Copy to CPU | Copy to GPU | Operation | Copy to CPU …

Batch 1

Batch 2

Prefetching batches

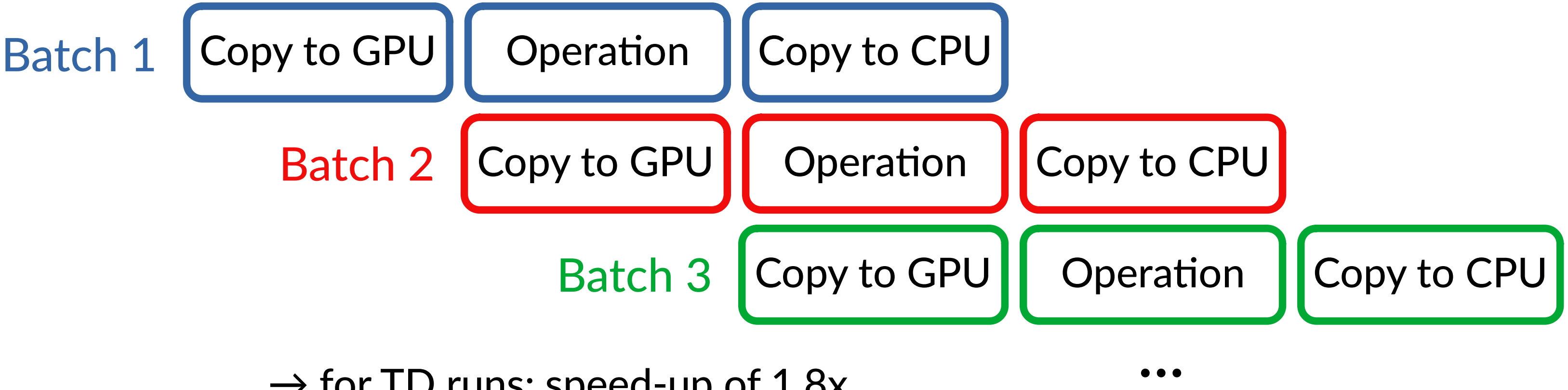
- Timeline without prefetching:



Copy to GPU | Operation | Copy to CPU | Copy to GPU | Operation | Copy to CPU …

Batch 1 | Batch 2

- Timeline with prefetching:



Batch 1 Copy to GPU | Operation | Copy to CPU

Batch 2 Copy to GPU | Operation | Copy to CPU

Batch 3 Copy to GPU | Operation | Copy to CPU

…

→ for TD runs: speed-up of 1.8x
(only used if states do not fit in GPU memory)