A Fine-Grained Trace Analysis of GPU-initiated Halo Exchange in GROMACS

Presenter: Andrey Alekseenko (KCSC, KTH Royal Institute of Technology; andreyal@kth.se)
Original Paper Co-authors: Mahesh Doijade, Ania Brown, Alan Gray, Szilárd Páll

GROMACS: Open-source, high-performance molecular dynamics engine

- **Motto:** Fast, Flexible, Free
- **Domains:** Biophysics, biochemistry, materials science
- **User base:** 10k's, academic and industry
- **Performance:** Hand-tuned SIMD & SIMT kernels
- **Design:** Heterogeneous-first (CPU + GPU)
- **Technologies:** C++, OpenMP, MPI, CUDA, OpenCL, SYCL, HIP
- **Precision:** Mixed-precision optimized for high throughput; only FP32 on GPUs

Molecular Dynamics

- **MD Loop:** Calculate forces (expensive) → Update positions & velocities → Repeat
- **Workload:** Arithmetically intensive and typically compute-bound.
- **Timescales:** femtosecond steps → millisecond events
- **Scale:** 10^8 – 10^{15} steps per simulation
- **Problem size:** 10^4–10^8 particles, need strong scaling
- **Goal:** Sub-millisecond wall-time per step

GPU-initiated halo exchange [1]

Read the PAW-ATM workshop paper

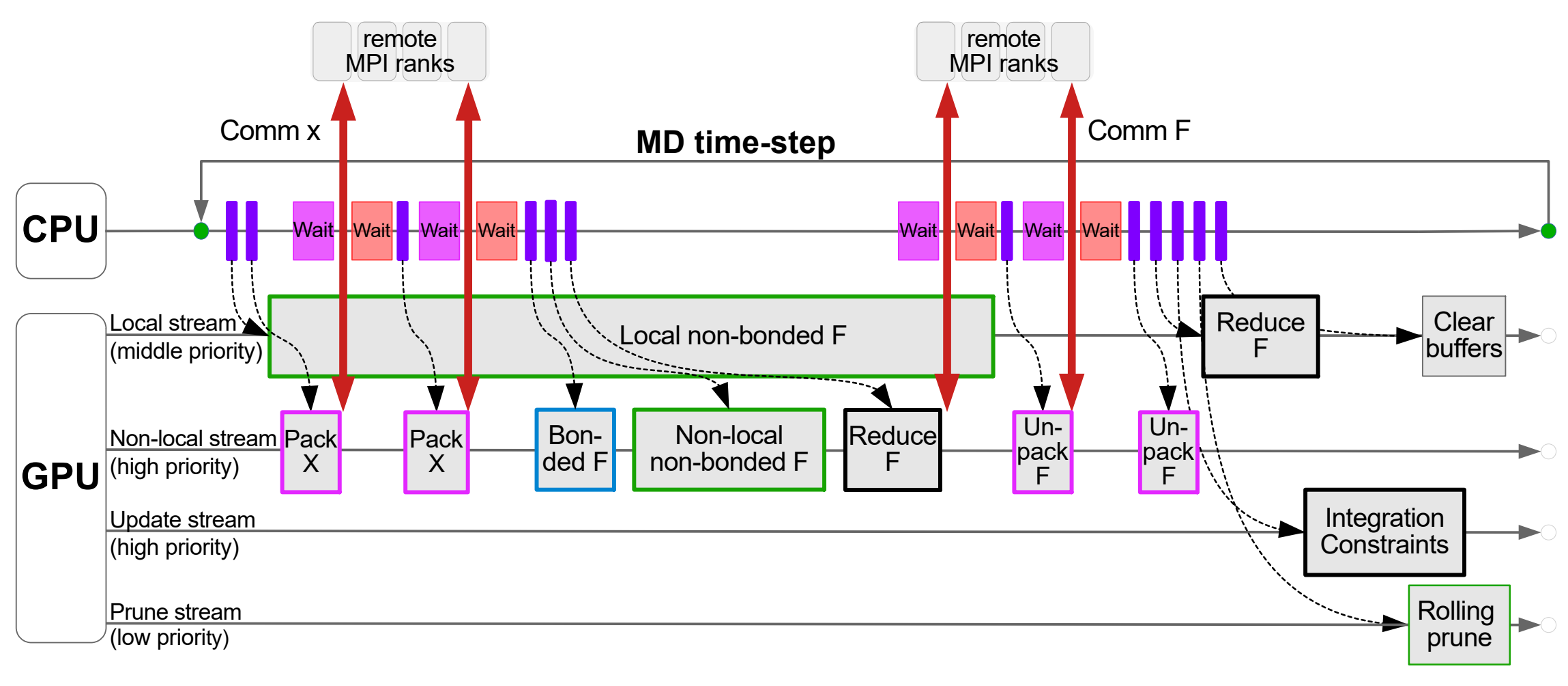
Domain decomposition: Simulation box split into spatial domains per GPU.

Halo exchange: Atoms near boundaries need neighbor coordinates to compute forces.

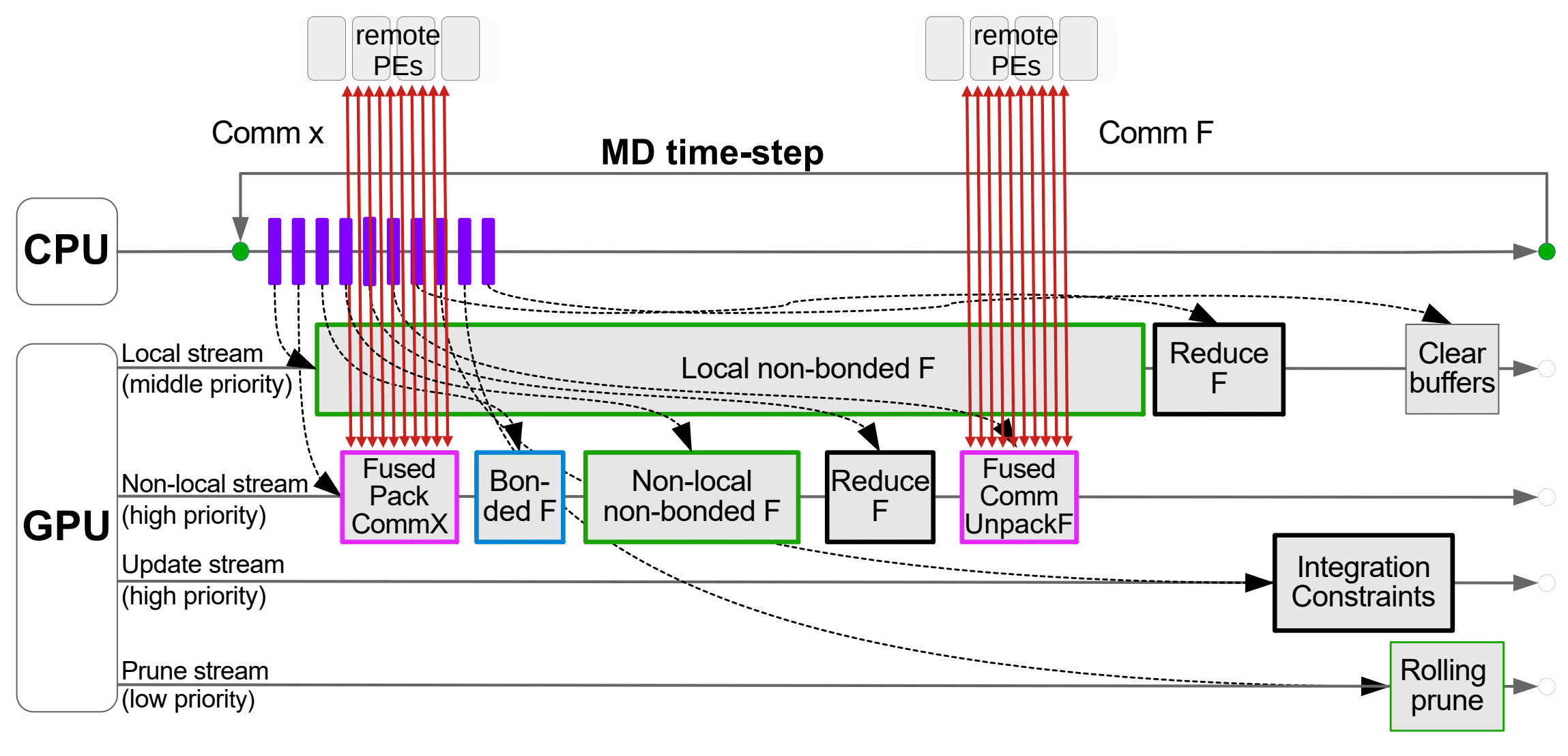
Staged forwarding: "Neutral territory" method moves data sequentially (Z→Y→X) to minimize connections.

Bottleneck: Sequential pulses and CPU synchronization expose latency on the critical path.

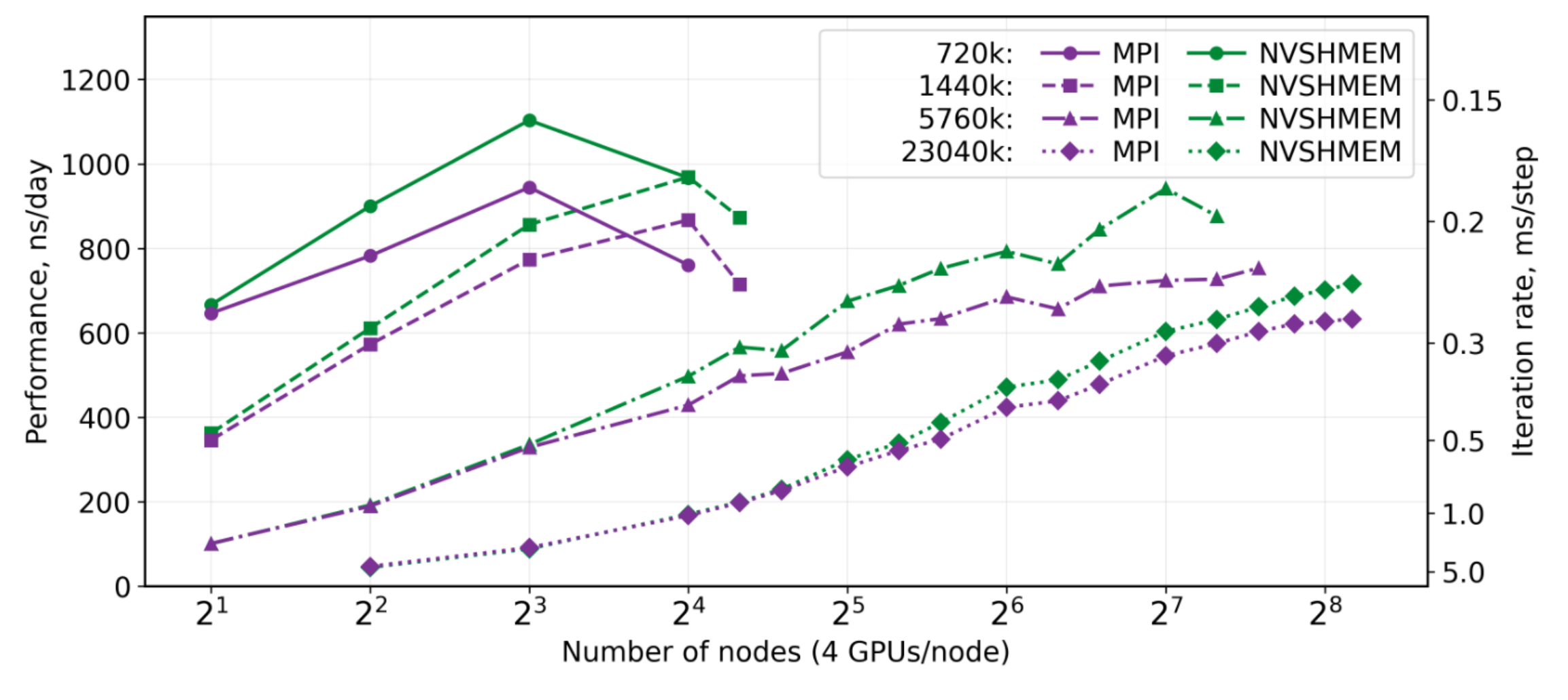
GPU-aware MPI is still CPU-initiated, requiring host-device synchronizations and multiple pack/unpack kernel launches on the critical path.



GPU-initiated communications using NVSHMEM avoid CPU overheads to maximize compute-communication overlap. Fused pack/unpack kernels minimize GPU API overheads.



Multi-node performance improvements



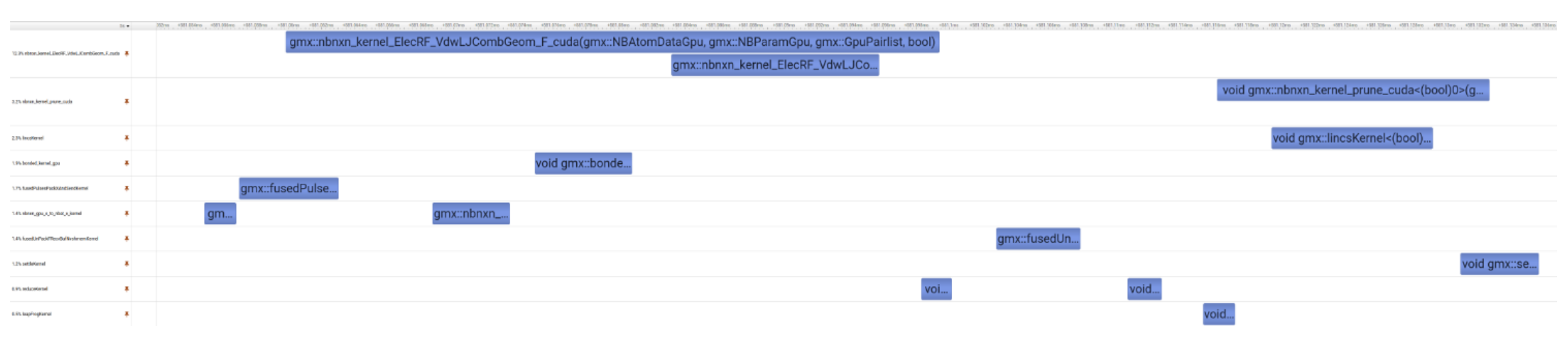
Results above obtained on NVIDIA Eos: 2× Intel Xeon 8480C and 4× H100 per node, NVLink 4.0 + NDR 400G InfiniBand

Acknowledgments

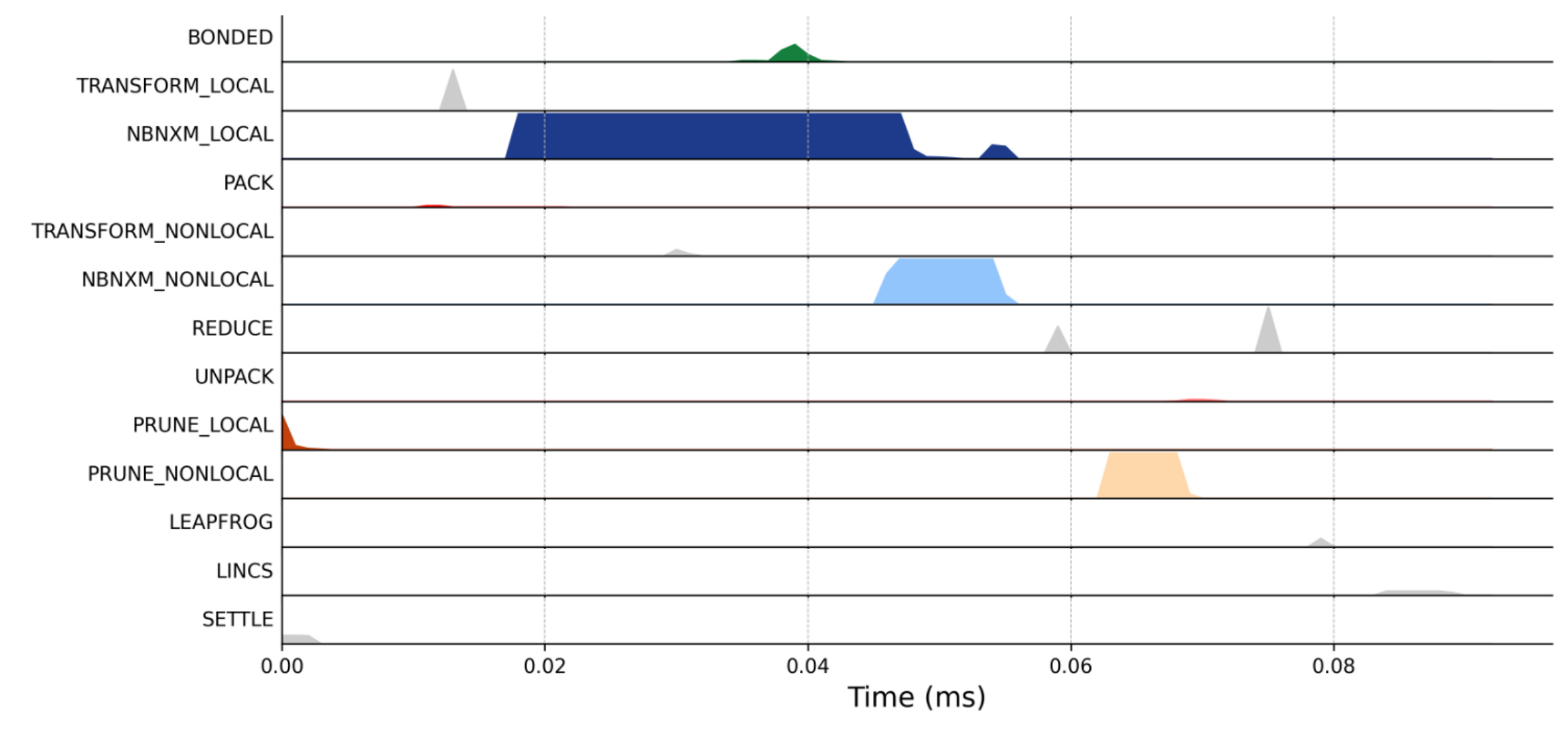
Kernel timing discussion: Timothée David–Cléris
This work was funded by the European Union under the GANANA project (Grant Agreement No. Grant #101196247) and supported by the Swedish e-Science Research Center.

Manual instrumentation for GPU tracing

NVIDIA Nsight Systems (nsys) captures kernel concurrency but lacks visibility into resource division. The timeline of a single MD step below shows kernels overlapping, but cannot quantify how many Streaming Multiprocessors (SMs) each kernel actually holds or how they fight for space:



By reading %globaltimer (timestamps) and %smid (SM index) per thread block, we can reconstruct exact SM occupancy over time. The plot below, showing the fraction of total device SMs that each kernel is executed on, reveals dynamic resource contention: notice how the high-priority NBNXM_NONLOCAL (light blue) gradually preempts the running NBNXM_LOCAL (dark blue), visibly taking time to "push out" the lower-priority work from the GPU:



The default timer updated frequency of 1 µs is sufficient for this work. The update frequency can be increased to 32 ns using an undocumented mechanism invoked by NVIDIA profiling tools, e.g., by initializing CUPTI at the start of the application with cuptiSubscribe + cuptiEnableDomain.

Alternatives: Similar fine-grained timings can be obtained using automated and semi-automated approaches, such as CUPTI or Neutrino. Here, manual instrumentation was used since the end goal is to use it not only for development but also in production for dynamic load balancing.

Used hardware & software

Tracing restricted to single-node execution to isolate ensure reproducible communication patterns by only using NVLink interconnect.

Dardel-GH (HPE Cray EX254n)

- 4× 72-core Grace CPU (Neoverse V2)
- 4× Hopper GH200 GPU
- Fully-connected NVLink 4.0, 6 lanes/link

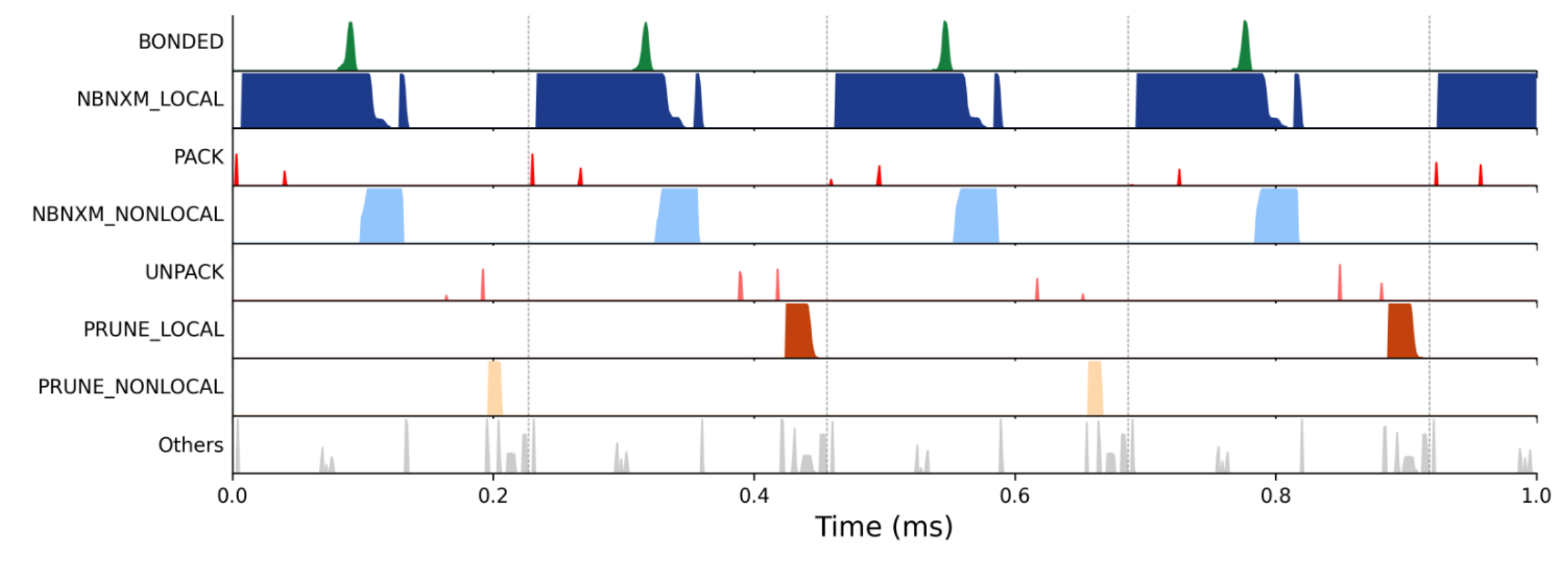
CUDA 12.6, NVSHMEM 3.3.9, GCC 12.3

GROMACS 2026.rc, with device-side timing and asynchronous pruning patches.

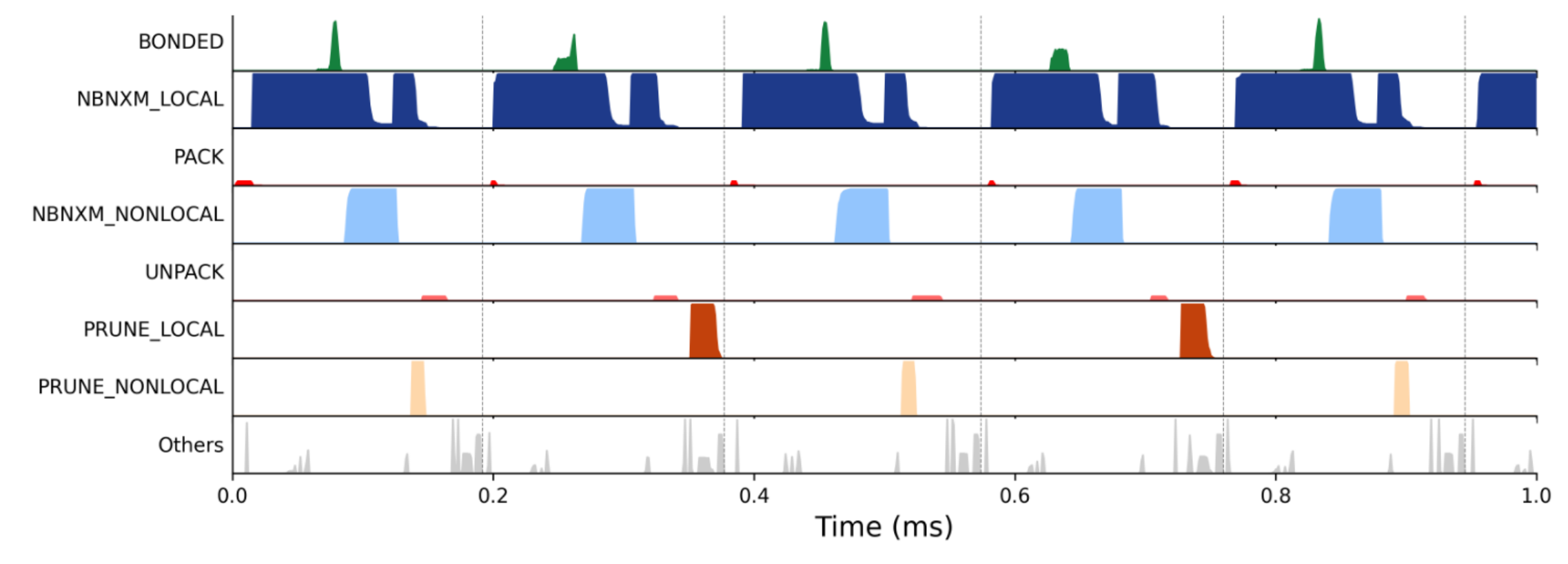
MPI vs. NVSHMEM: Trace analysis

The plots below each show a 1 ms snapshot of a 360k atoms (90k atoms/GPU) simulation. Dashed vertical lines mark MD step boundaries.

With a 2D decomposition, the necessary two pulses (Y→X) must be serialized. With MPI, this requires two pairs of PACK/UNPACK kernels and associates host-device synchronizations and MPI calls each step, leading to low GPU utilization:



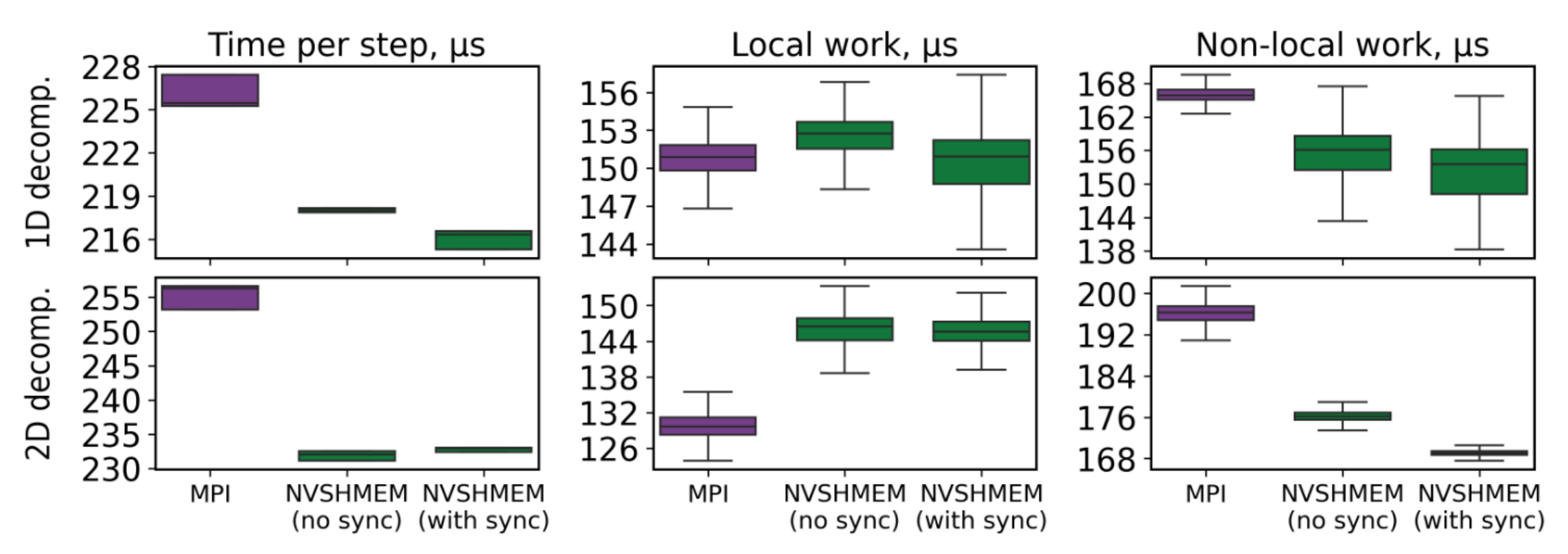
With NVSHMEM, we fuse computation, communication, and signalling for all pulses into single PACK and UNPACK kernels. This increases the iteration rate and improves GPU utilization:



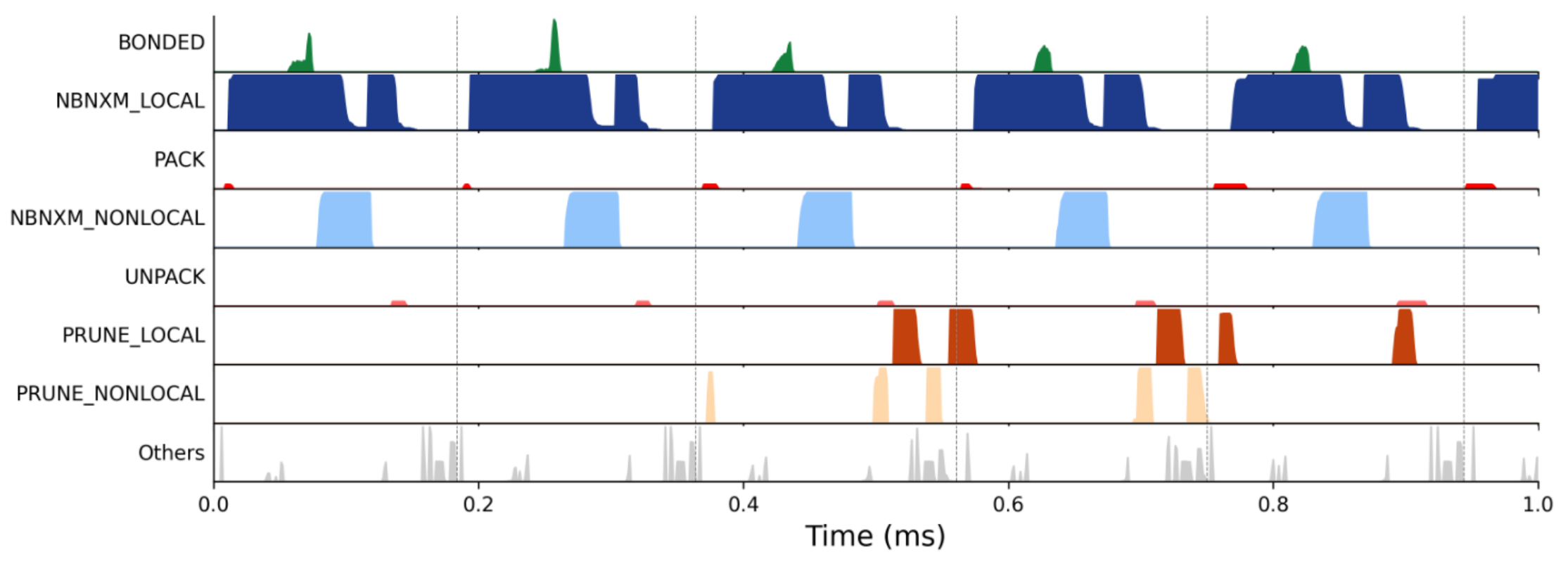
Synchronization paradox

GROMACS periodically prunes the lists of nearby atoms to keep force calculations efficient. To prevent this operation from affecting the critical path, the **"asynchronous pruning" optimization** used in [1] moves pruning to a low-priority stream, intending it to run in gaps between other kernels. These pruning kernels can drift freely, synchronizing with the main compute loop only once every 8 steps.

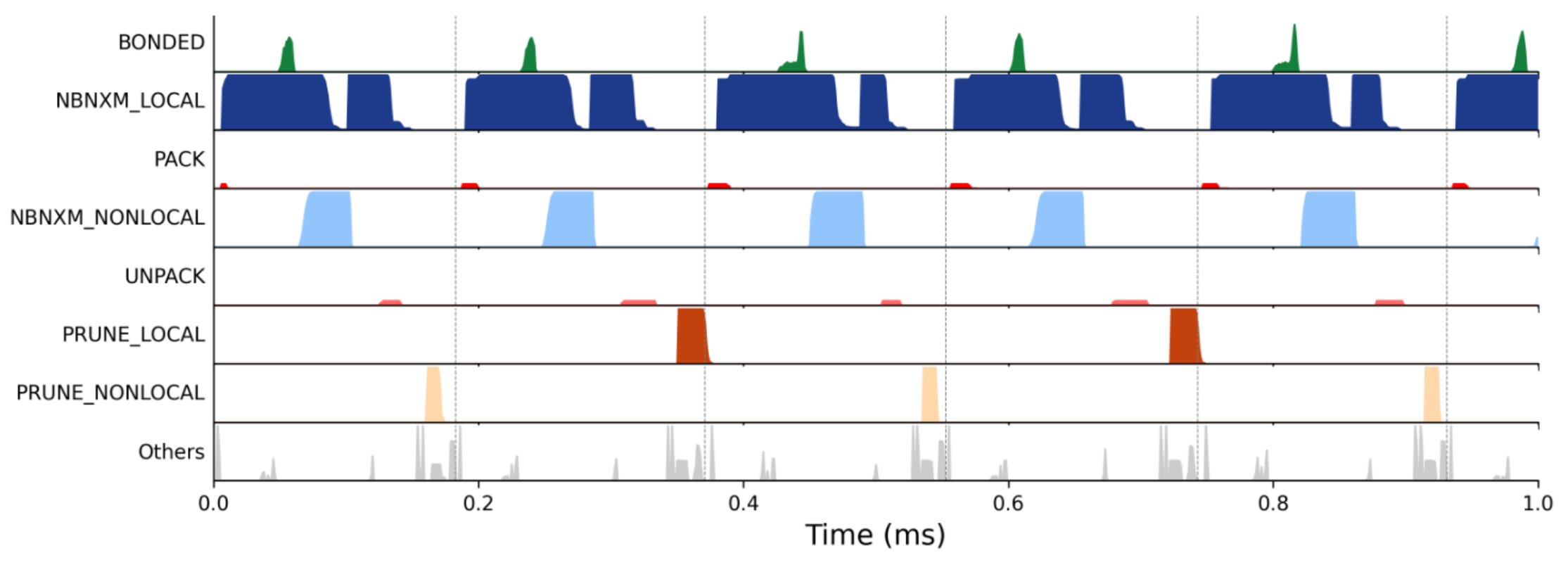
In [1], we observed that adding an **explicit device-host synchronization** after the fused UNPACK kernel paradoxically improves NVSHMEM performance, surpassing the asynchronous pruning baseline. For example, in the 90k atoms/GPU case, it speeds up Non-local work (from the beginning of PACK to the end of UNPACK) by up to 5%:



On the tested GH200 GPUs, the GPU scheduler aggressively defers these low-priority PRUNE_LOCAL and PRUNE_NONLOCAL kernels. Instead of running incrementally in the background, they accumulate until the mandatory synchronization point (every 8th step), where they launch in a concentrated burst. This burst competes for resources with other kernels, potentially slowing down the critical path. Additionally, delayed pruning can increase the amount of work for the most expensive compute kernels: NBNXM_LOCAL and NBNXM_NONLOCAL.



Explicitly synchronizing a different ("halo") stream prompts the scheduler to launch the pending low-priority kernels in the "prune" stream incrementally. This side effect smooths out resource usage, preventing the contention and slightly im:



Conclusions

Here, we analyse in depth some of the performance observations of fused GPU-initiated halo exchange in GROMACS, originally made in [1], and show the limitations of priority-based scheduling.

The in-kernel timing approach has low overheds and provides better insights into complex GPU schedules than standard profiling tools.

GPU-initiated halo exchange using NVSHMEM is available as an experimental feature in GROMACS 2026.

Future work

- **Precise dynamic load balancing (DLB):** CUDA events are unreliable for timing heavily overlapped schedules. We aim to drive DLB using in-kernel global timers (as used in our tracing) to accurately measure local work. This approach is portable to AMD GPUs (used via SYCL and HIP backends) but remains challenging on Intel GPUs due to lack of a global steady timer.
- **Better resource partitioning:** CUDA Green Contexts and Cluster Launch Control mechanisms allow limiting the resource use of a kernel and offer a more controllable mechanism to isolate the critical path from the interference from low-priority tasks.

Download this poster

References

[1] Doijade et al. Supercomputing, PAW-ATM (2025), doi:10.1145/3731599.3767508
[2] Páll et. al. J. Chem. Phys. 153 (2020), doi:10.1063/5.0018516