FAST. FLEXIBLE. FREE.
GROMACS

SYCL™

GROMACS on AMD GPU-based HPC platforms: using SYCL for performance and portability

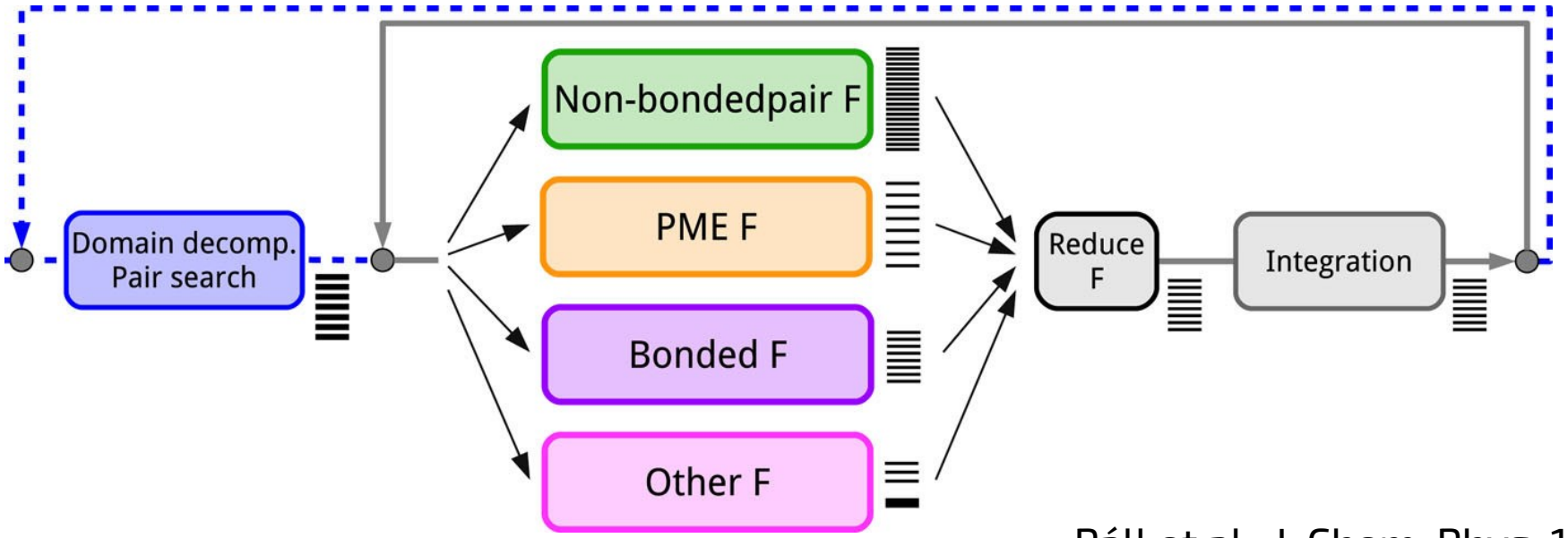
Andrey Alekseenko, Szilárd Páll, Erik Lindahl

KTH Royal Institute of Technology & SciLifeLab
Stockholm, Sweden
andreyal@kth.se

CUG2024
Diverse Universe
PERTH AUSTRALIA

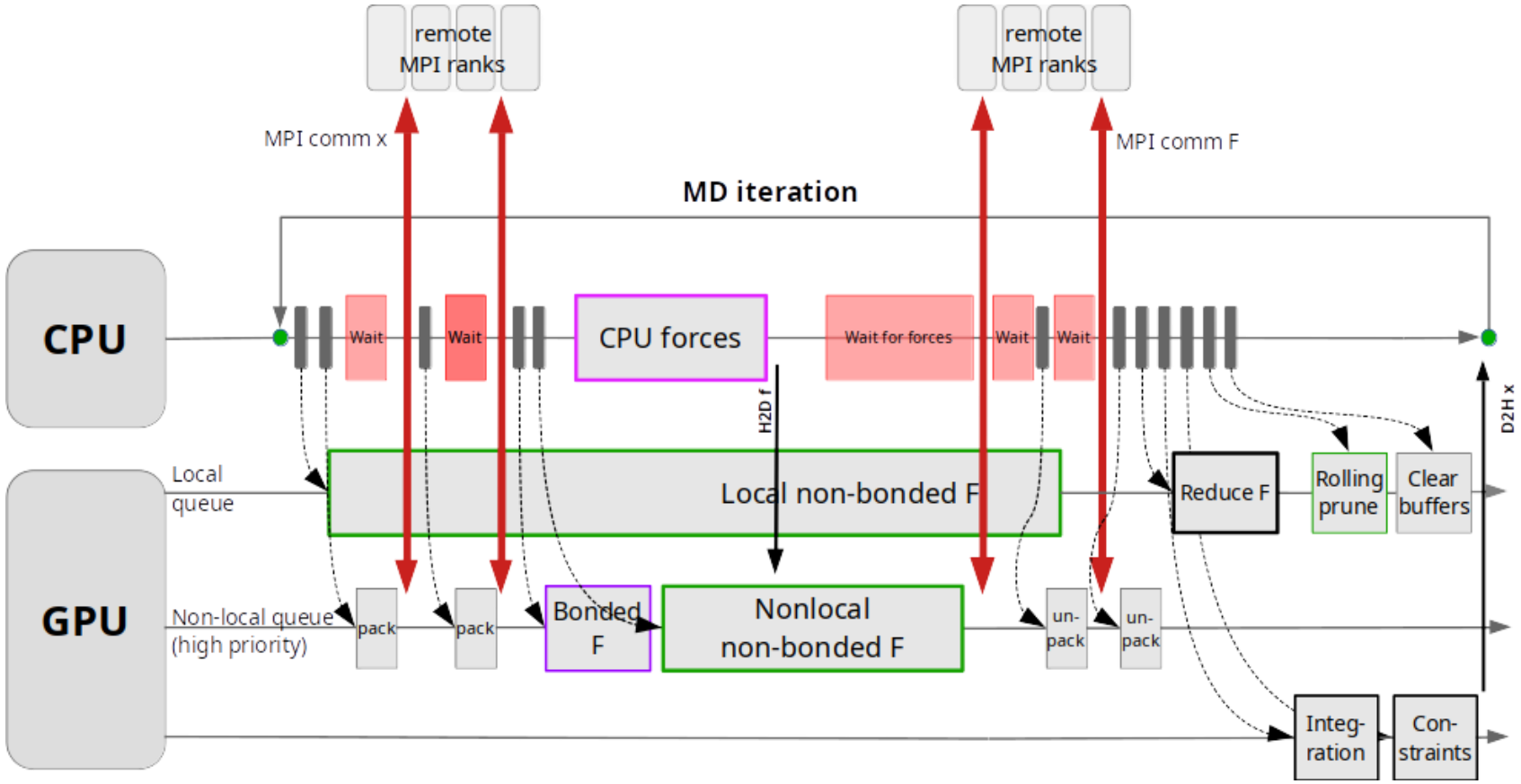
Molecular dynamics

- Newton's equations of motion
- Fixed problem size (~10^4-10^6): limited hardware parallelism
- Fast iterations: timestep ~1 fs, need to reach µs to ms



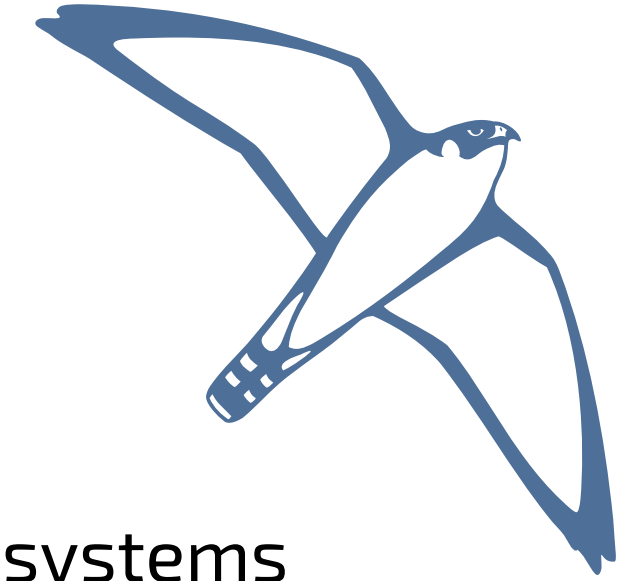
Páll et al., J. Chem. Phys. 153, 134110 (2020)

Molecular dynamics: example schedule



< 1 millisecond

GROMACS



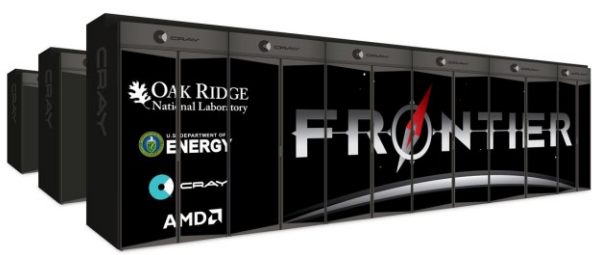
- Open-source molecular dynamics engine
  - 470k lines of C++ code
- High-performance for a wide range of modeled systems
  - From 10^4 to 10^9 particles
- … and on a wide range of platforms
  - x86-64, ARM, POWER, RISC-V
  - AMD, Apple, Intel, and NVIDIA GPUs; Intel Xeon Phi
  - Linux, Windows, MacOS, *BSD
- MPI; OpenMP; SIMD intrinsics; CUDA / OpenCL / SYCL

GROMACS

- Multi-level parallelism:
  - MPI (task- and domain decomposition)
  - OpenMP
  - CUDA / OpenCL / SYCL
  - SIMD intrinsics
- Efficient scaling:
  - Caching and locality-optimized algorithms
  - Flexible offloading scheme: GPU-resident or heterogeneous
  - Direct GPU-GPU communication
    - GPU-initiated: work in progress
  - Scalable distributed FFT

GPU support in GROMACS 2020

	NVIDIA CUDA	OpenCL™
Non-bonded offload	√	√
PME offload	√	√
Update offload	√	X
Bonded offload	√	X
Direct GPU-GPU comm	√	X
PME Decomposition	X	X
Hardware support	NVIDIA	NVIDIA, AMD, Intel



?

?

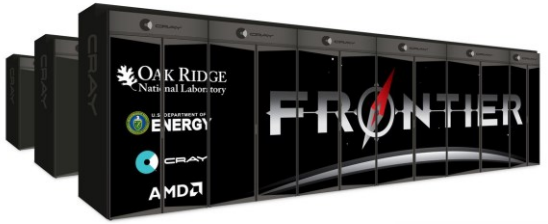
SYCL

- Open standard
- High-level
- Vendor-neutral
- Multiple implementations
- Based on C++17


```
#include <sycl/sycl.hpp>

// Boilerplate
sycl::queue queue{{sycl::property::queue::in_order()}};
// Allocate GPU memory
float* Ad = sycl::malloc_device<float>(n, queue); // ...
// Copy the data from CPU to GPU
queue.copy<float>(Ah, Ad, n); // ...
// Submit a kernel into a queue; cgh is a helper object
queue.submit([&](sycl::handler &cgh) {
  cgh.parallel_for<class Kernel>(sycl::range<1>{n},
    [=](sycl::id<1> i) {
    Cd[i] = Ad[i] + Bd[i];
  });
});
```

Supported hardware



- Primary targets:
  - **AMD CDNA2** GPUs with AdaptiveCpp
  - **Intel Xe-HPC** GPUs with oneAPI
- Secondary targets:
  - Other AMD GPUs with oneAPI and AdaptiveCpp
  - Other Intel GPUs with oneAPI
- Should work:
  - NVIDIA GPUs with oneAPI and AdaptiveCpp

Why SYCL for AMD GPUs

- Open, vendor-independent **standard**
- Intel GPU support
  - We want less backends, not more
- Two relevant implementations
  - AdaptiveCpp (previously known as hipSYCL)
  - Intel oneAPI DPC++
- Built upon the HIP toolchain:
  - Day-one hardware support
  - Can use native code inline
  - Vendor tools and native libraries just work

GROMACS 2024

- SYCL nearly on par with CUDA in feature-support
  - Forces and update offload, GPU-aware MPI
- Already used for large-scale runs on LUMI

- Two versions of AdaptiveCpp runtime compared
  - 0.9.4
  - 23.10.0
- Intel oneAPI also works, but slower for now

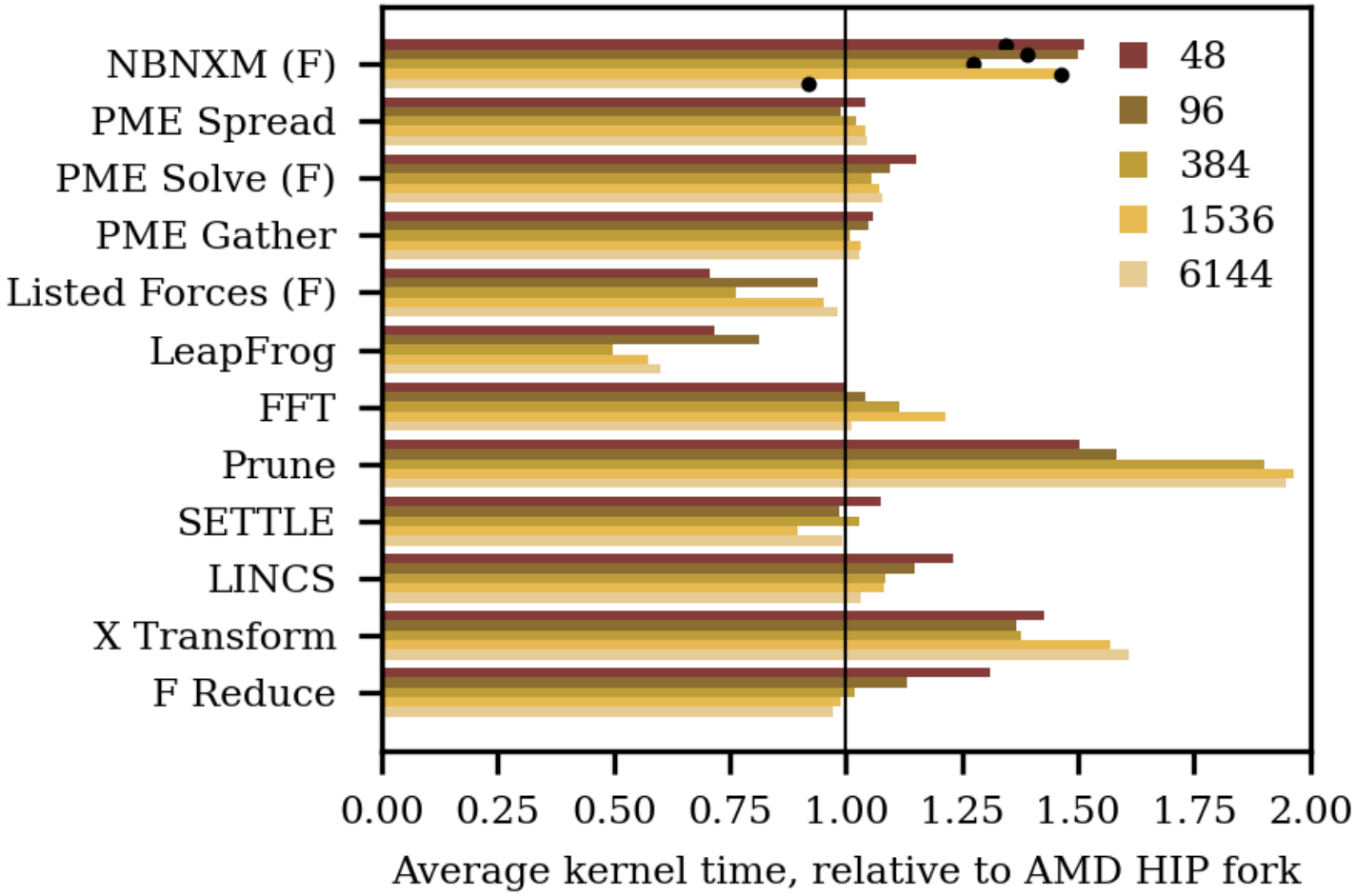
GROMACS HIP

- Independent fork by AMD and Stream HPC
- Based on 2022.beta2
  - HIPified CUDA version with many optimizations
- A lot of divergence from mainline GROMACS:
  - Conditional kernel fusion to avoid memsets
  - Pair list sorting to improve kernel scheduling
  - Hardware-specific intrinsics (unsafeAtomicAdd, warp_move_dpp)
  - Compiler workarounds (moving code around, casting pointers to different type and back, etc)
  - No pull-down of any correctness fixes or scheduling improvements

Cray EX235a

- AMD EPYC 7A53 64-core CPU
  - 8 cores per CCX, 8 CCX per CPU; some sites reserve 1 core per CCX
- 4x AMD Instinct MI250X
  - 2 GCD per board, 8 GCDs per node
- 4x Cray Slingshot-11 NICs
  - connected to MI250X

Kernel performance (single GCD)



- Many SYCL kernels are close to HIP
  - Some are faster!
- NBNXM is the longest-running
  - Pair list sorting (work-in-progress)
  - Compiler codegen optimizations
  - Better parameter tuning
- Seen 20% perf. difference between ROCm 5.3-5.6 for the same kernel

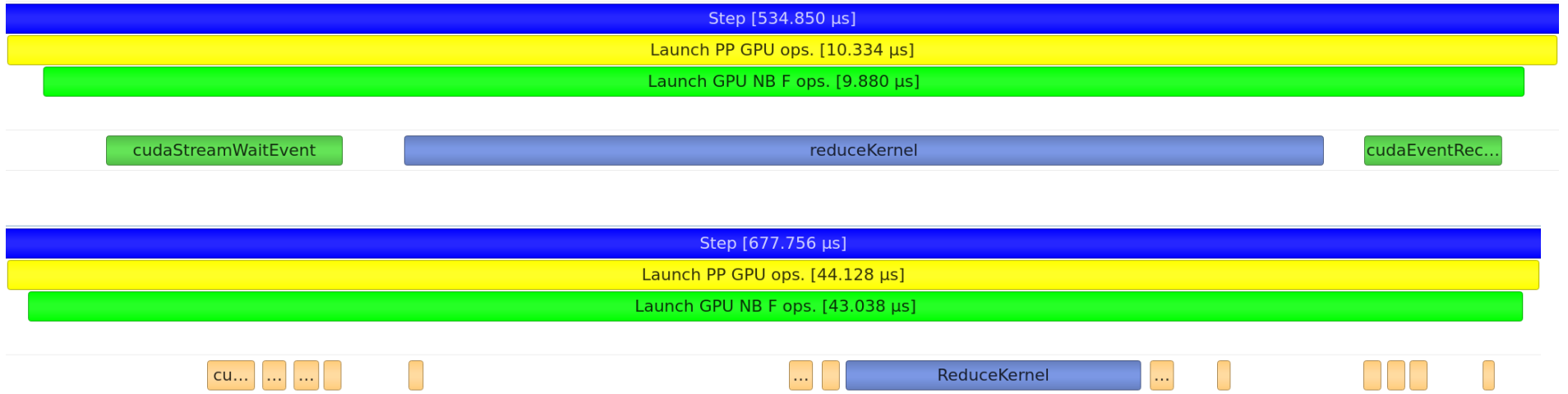
Performance/maintainability balance!

GROMACS 2024.0, ROCm 5.3.3, Dardel GPU, Grappa PME

Runtime performance: events

API spec: sycl::event sycl::queue::submit(…)
=> (cuda|hip)Event(Create|Record|Destroy)

CUDA: 10.3 µs



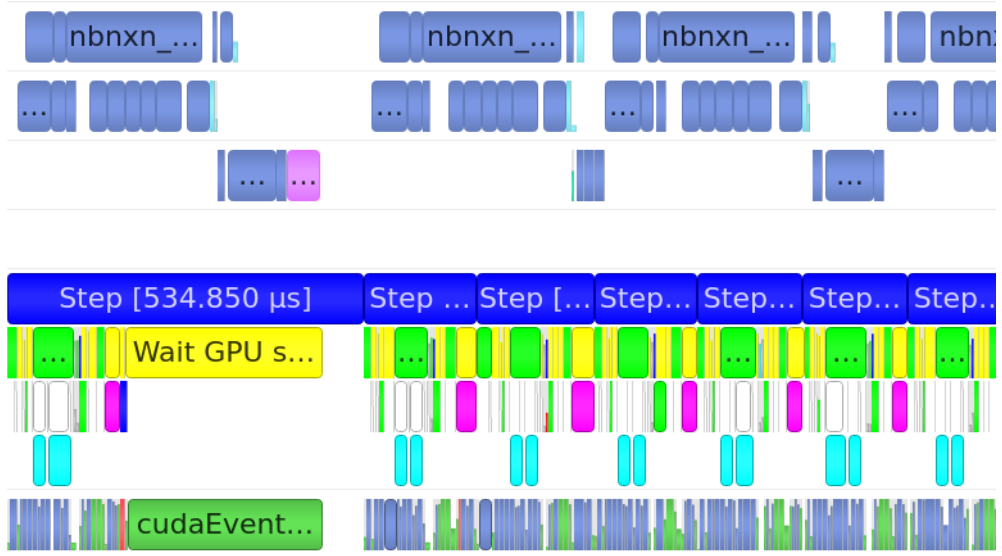
SYCL: 44.1 µs

Solution: **HIPSYCL_EXT_COARSE_GRAINED_EVENTS**

Note: the figure is with CUDA+oneAPI, but HIP+AdaptiveCpp behaves similarly

Runtime performance: deferred launch

CUDA: 3.9 ms per 10 steps



SYCL: 6.1 ms per 10 steps

Solution: **HIPSYCL_MAX_RT_CACHED_NODES**

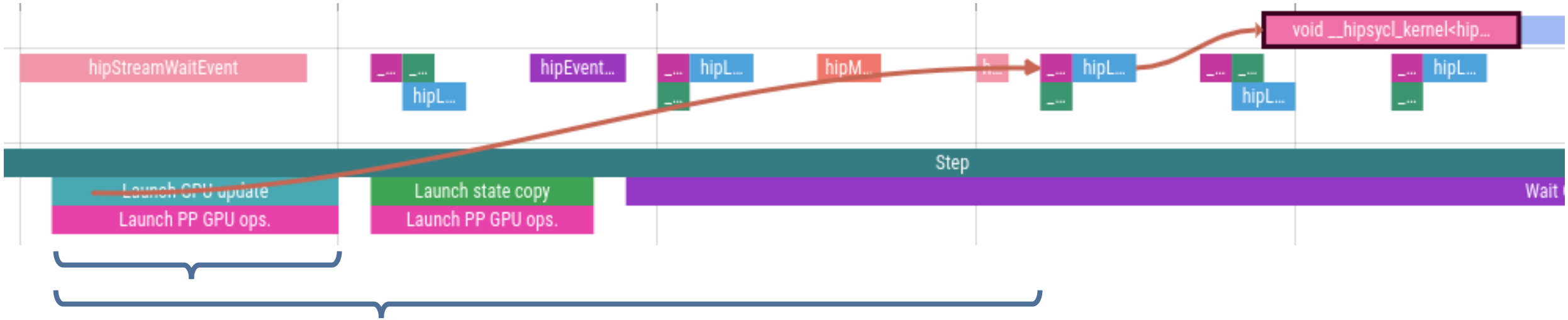
Note: the figure is with CUDA+AdaptiveCpp, but HIP+AdaptiveCpp behaves similarly

Runtime performance: worker threads

- OpenMP parallelism in GROMACS
- AMD HSA worker thread (1 core)
- Two AdaptiveCpp worker threads to call HIP API
  - Optimized in AdaptiveCpp 23.10.0, but still need 1 core

- Why even have worker threads?
  - Instant submission mode!

Runtime performance: instant mode



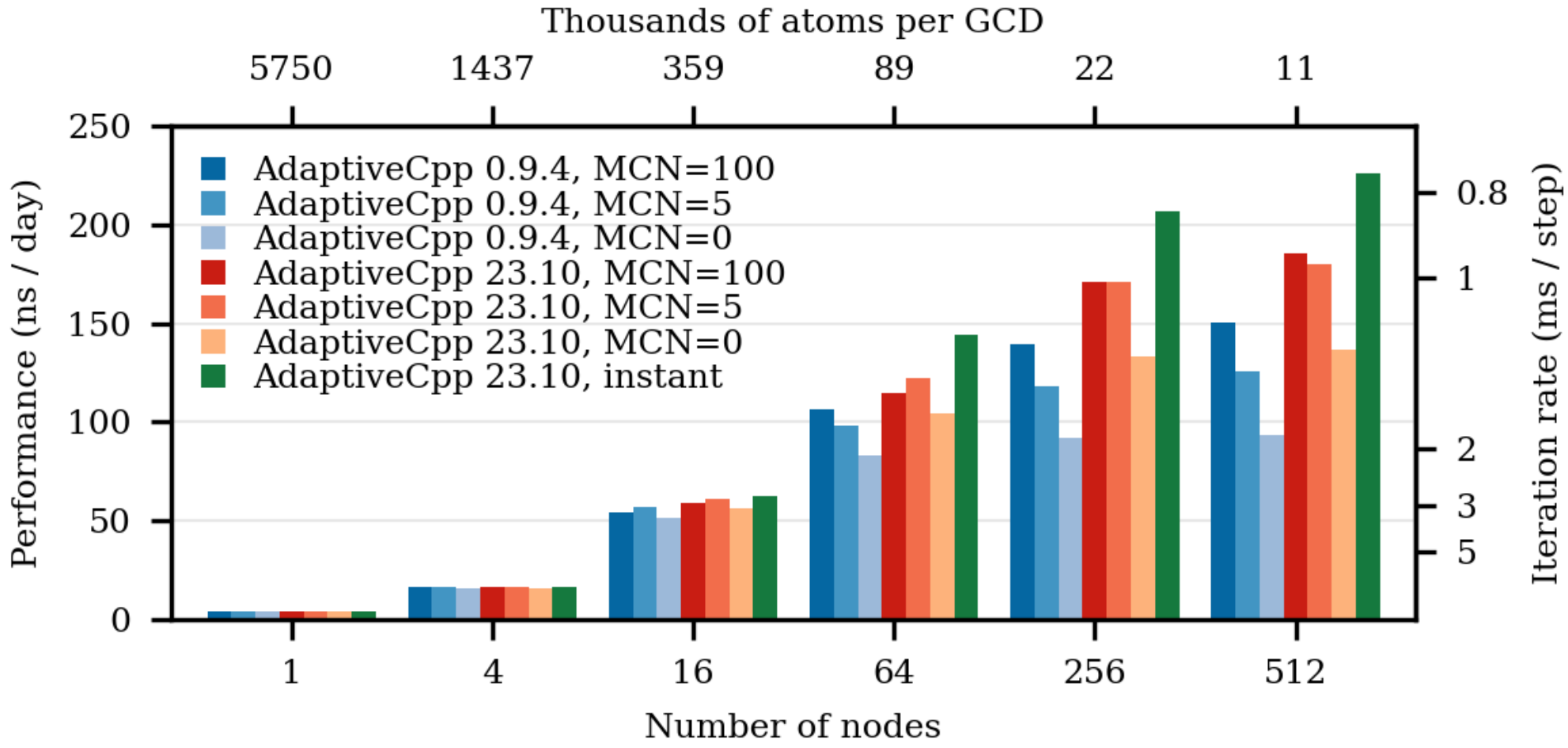
8µs	72µs	MCN=100
15µs	30µs	MCN=0
40µs	14µs	Instant

Instant submission mode was the best in all our tests.

Might change with GPU-initiated communications!

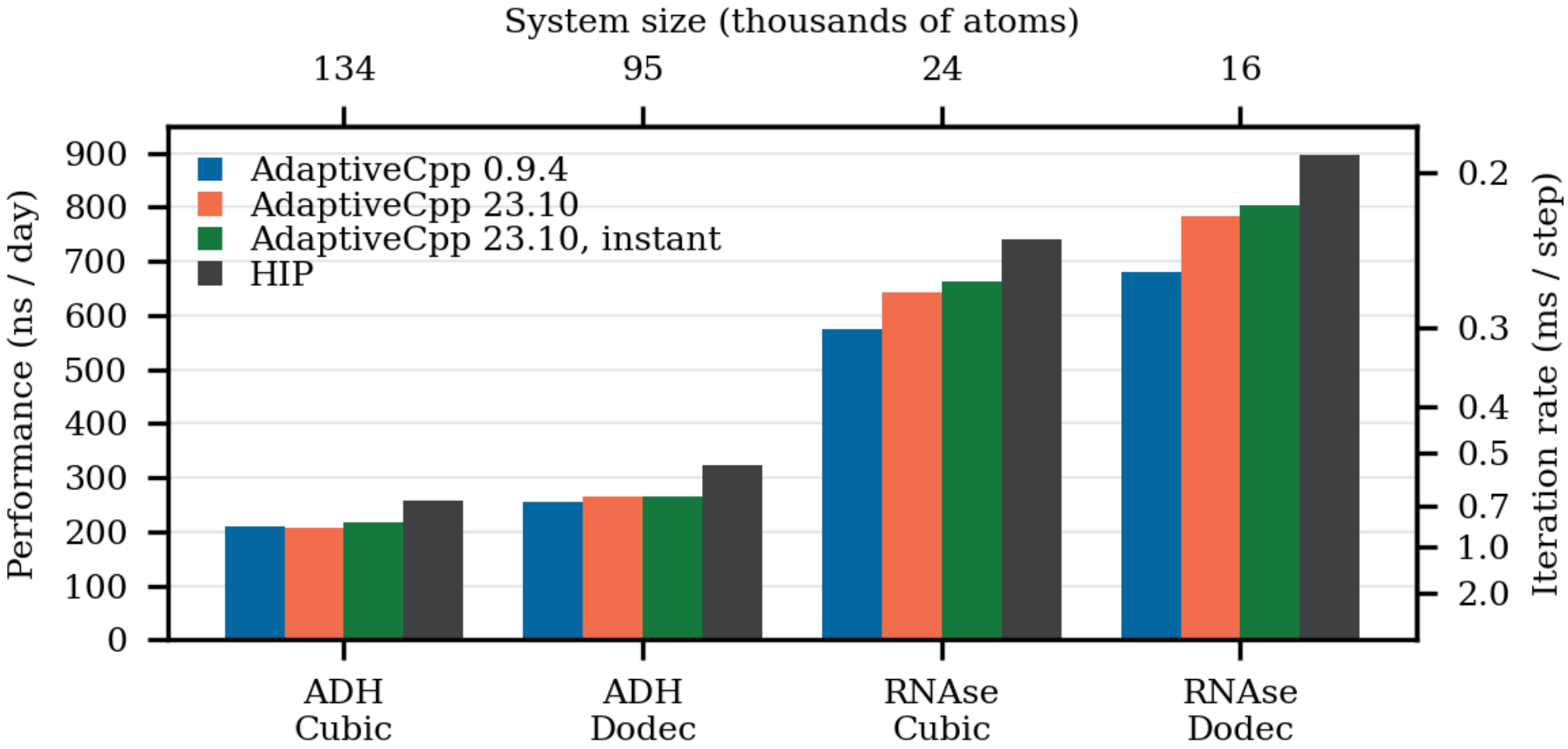
GROMACS 2024, ROCm 5.3.3, AdaptiveCpp 23.10.0, Dardel GPU. **Qualitative** data for illustration only.

Effects of AdaptiveCpp runtime



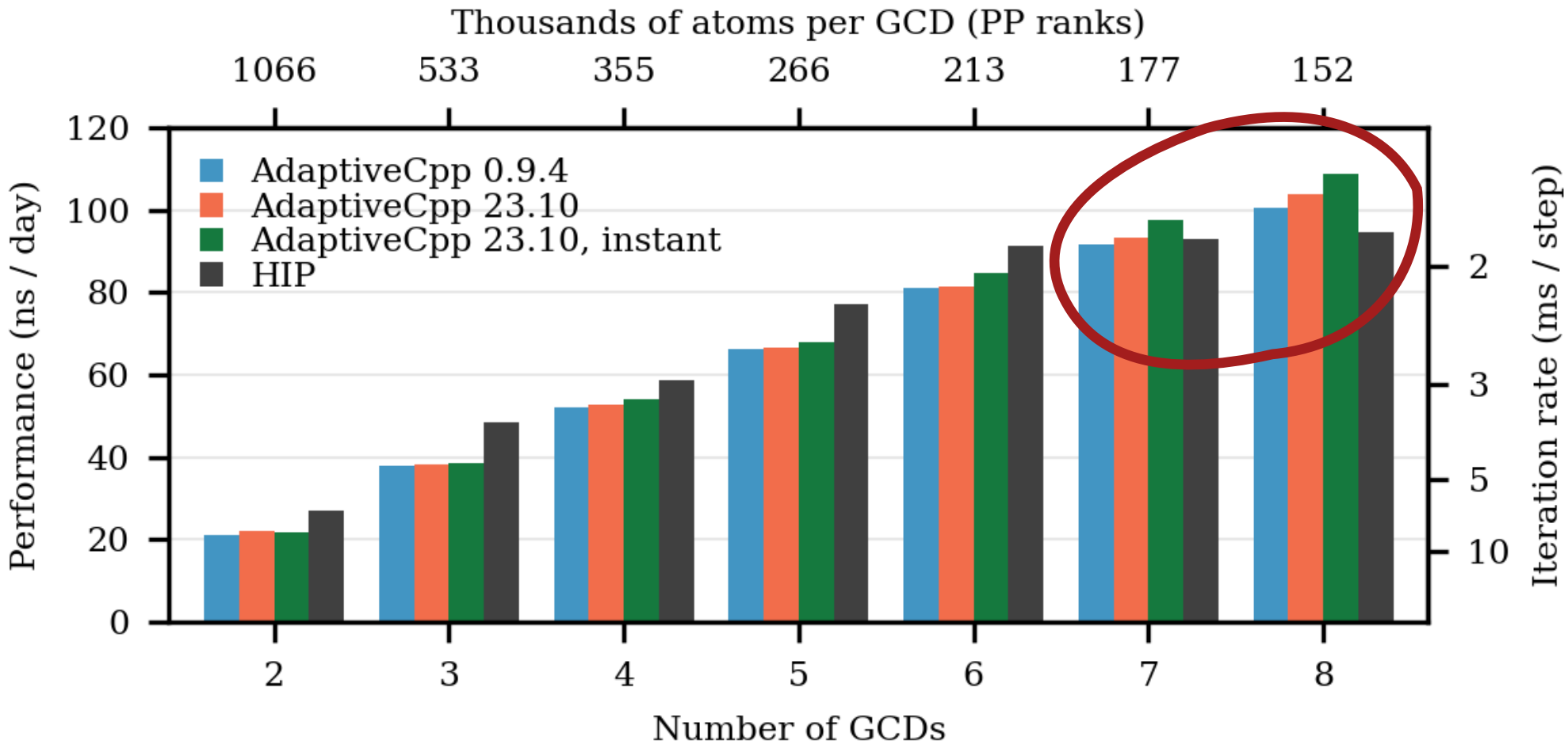
GROMACS 2024.1, ROCm 5.4.6, LUMI-G, Grappa RF (46M atoms)

Single-GCD performance



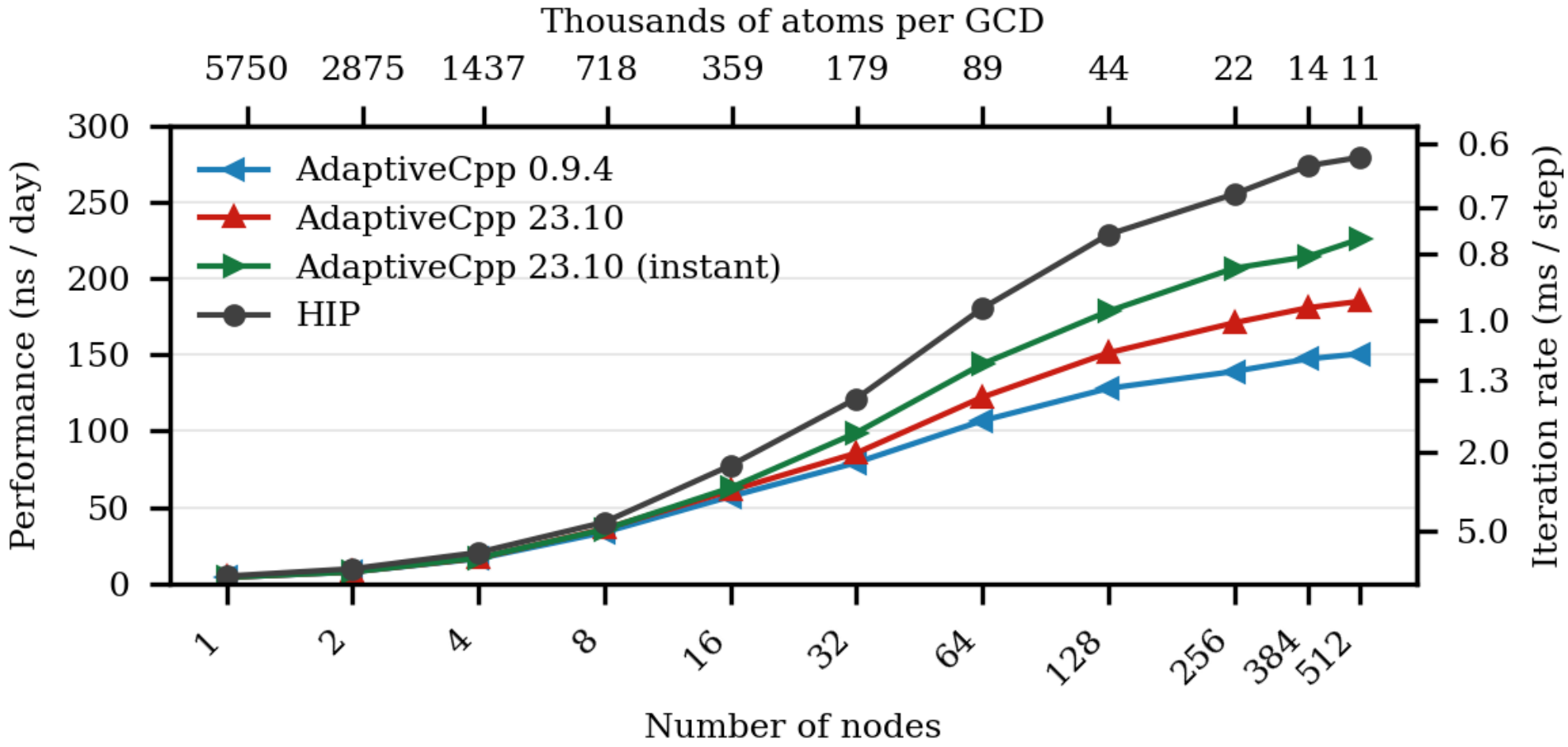
GROMACS 2024.0 / GROMACS HIP, ROCm 5.3.3, Dardel GPU

Single-node performance



GROMACS 2024.0 / GROMACS HIP, ROCm 5.3.3, Dardel GPU, STMV (1M atoms)

Multi-node scaling



GROMACS 2024.1 / GROMACS HIP, ROCm 5.4.6, LUMI-G, Grappa RF (46M atoms)

Conclusions: SYCL in GROMACS

- Portability:
  - Same code running on AMD, Intel, and NVIDIA GPUs
    - Minimal device-specific optimizations
- Performance:
  - Kernel performance is lower, but does not have to be
    - Complete native kernels can be dropped in
    - We know where the performance is lost
      - Need to find the balance between maintainability and performance
      - But easier maintenance also leads to performance improvements
  - SYCL runtime required initial effort, but now works well
    - Other projects can benefit

Conclusions: SYCL in GROMACS

- Can SYCL be used on exascale-class AMD GPU-based systems?
  - **Yes!** Even for challenging cases like MD!
- Power of collaboration:
  - A lot of work on GPU scaling initially done with NVIDIA
  - Working closely with AdaptiveCpp developers to improve the runtime
    - Ongoing work with Codeplay to help improve oneAPI DPC++ performance
  - AMD's HIP port was an important driver for optimizations
    - Some adopted by other GROMACS backends
- Next challenges:
  - GPU-initiated communications (SHMEM, MPI RMA, Stream-aware MPI)
  - 3D FFT strong scaling

Acknowledgements

- Aksel Alpay
- Maciej Szpindler
- Ragnar Sundblad
- Julio Maia
- Bálint Soproni
- And the whole GROMACS community!