FAST. FLEXIBLE. FREE.

GROMACS

Andrey Alekseenko[1,2], Szilárd Páll[1,2,3], Erik Lindahl[1,2,4]

[1] KTH Royal Institute of Technology, [2] SciLifeLab, [3] PDC Center for High Performance Computing,

[4] Stockholm University

Experiences with adding SYCL support to GROMACS

IWOCL & SYCLcon 2021

GROMACS:

- Open source molecular dynamics engine
- One of the most used HPC codes worldwide
- High-performance for a wide range of modeled systems
- … and on a wide range of platforms:
  - from supercomputers to laptops (Folding@Home)
  - X86, X86_64, ARM, POWER, SPARC
  - 14 SIMD backends
  - NVIDIA, AMD, and Intel GPUs; Intel Xeon Phi
  - Windows, MacOS, included in many Linux distros

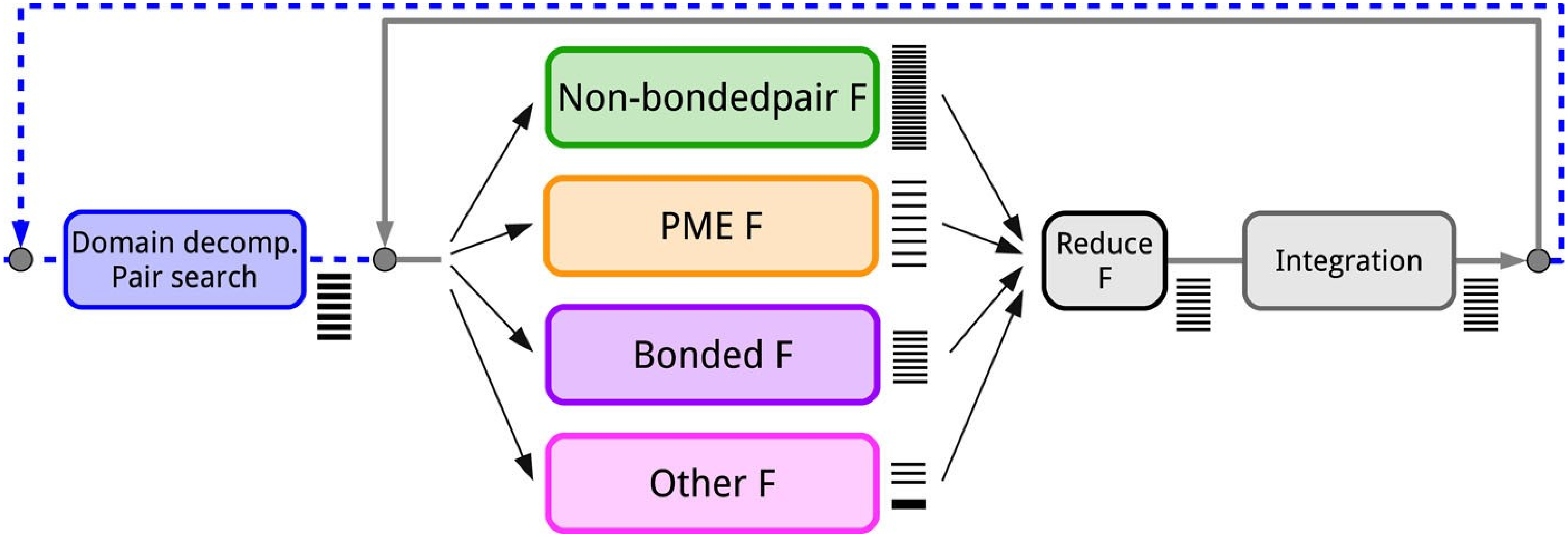
GROMACS 2021:

- (Mostly) C++17 codebase
  - With a bit of legacy
- Multi-layer parallelism for scalability:
  - SIMD for low-latency operations on CPU
  - GPU offload for high-throughput operations
  - OpenMP for SMP parallelism
  - MPI for inter-node communication
- 427k lines of C++ code

GROMACS 2021:

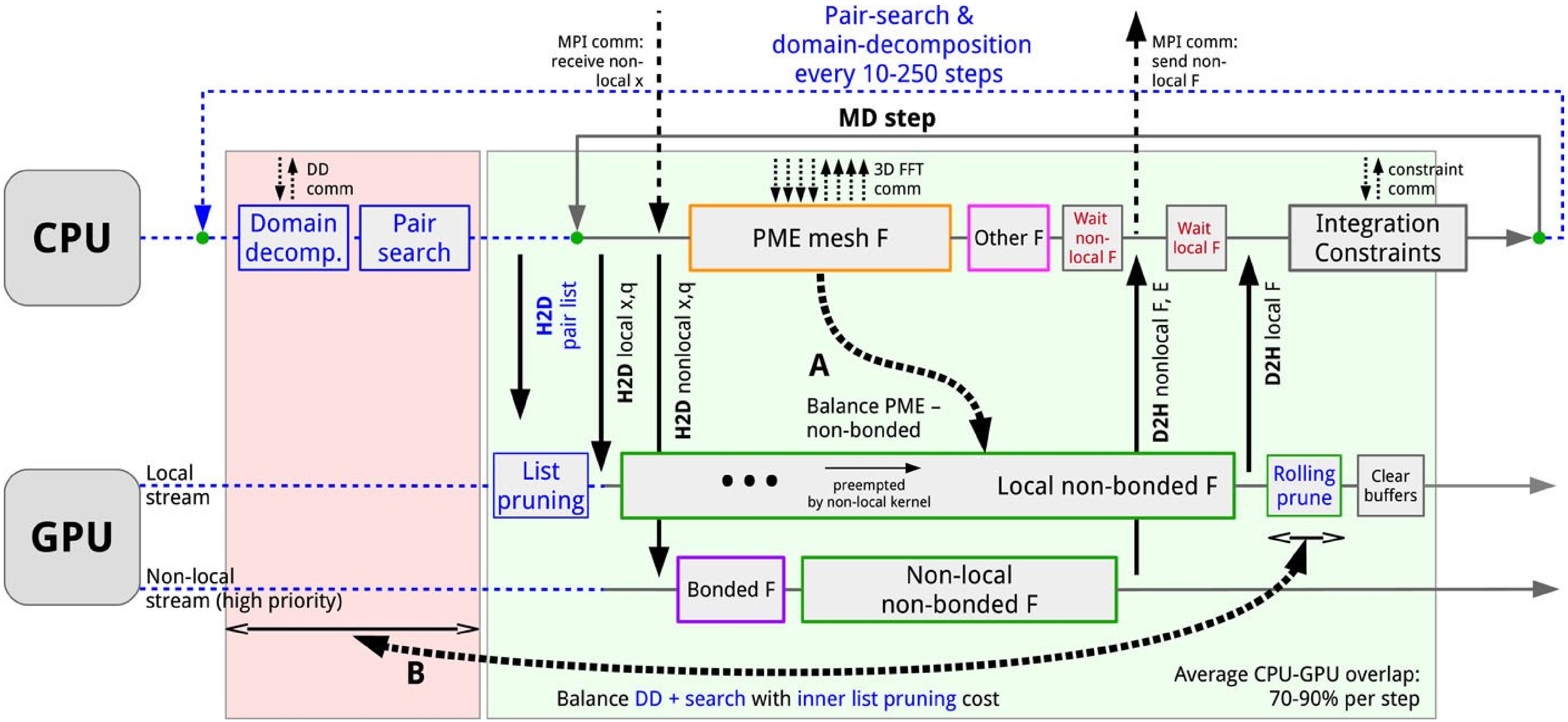
- (Mostly) C++17 codebase
  - With a bit of legacy
- Multi-layer parallelism for scalability:
  - SIMD for low-latency operations on CPU
  - GPU offload for high-throughput operations
  - OpenMP for SMP parallelism
  - MPI for inter-node communication
- 427k lines of C++ code
- 8.8k lines of CUDA code
- 5.8k lines of OpenCL code
  - Including 3.4k lines of host glue code

MD loop overview



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

MD loop overview

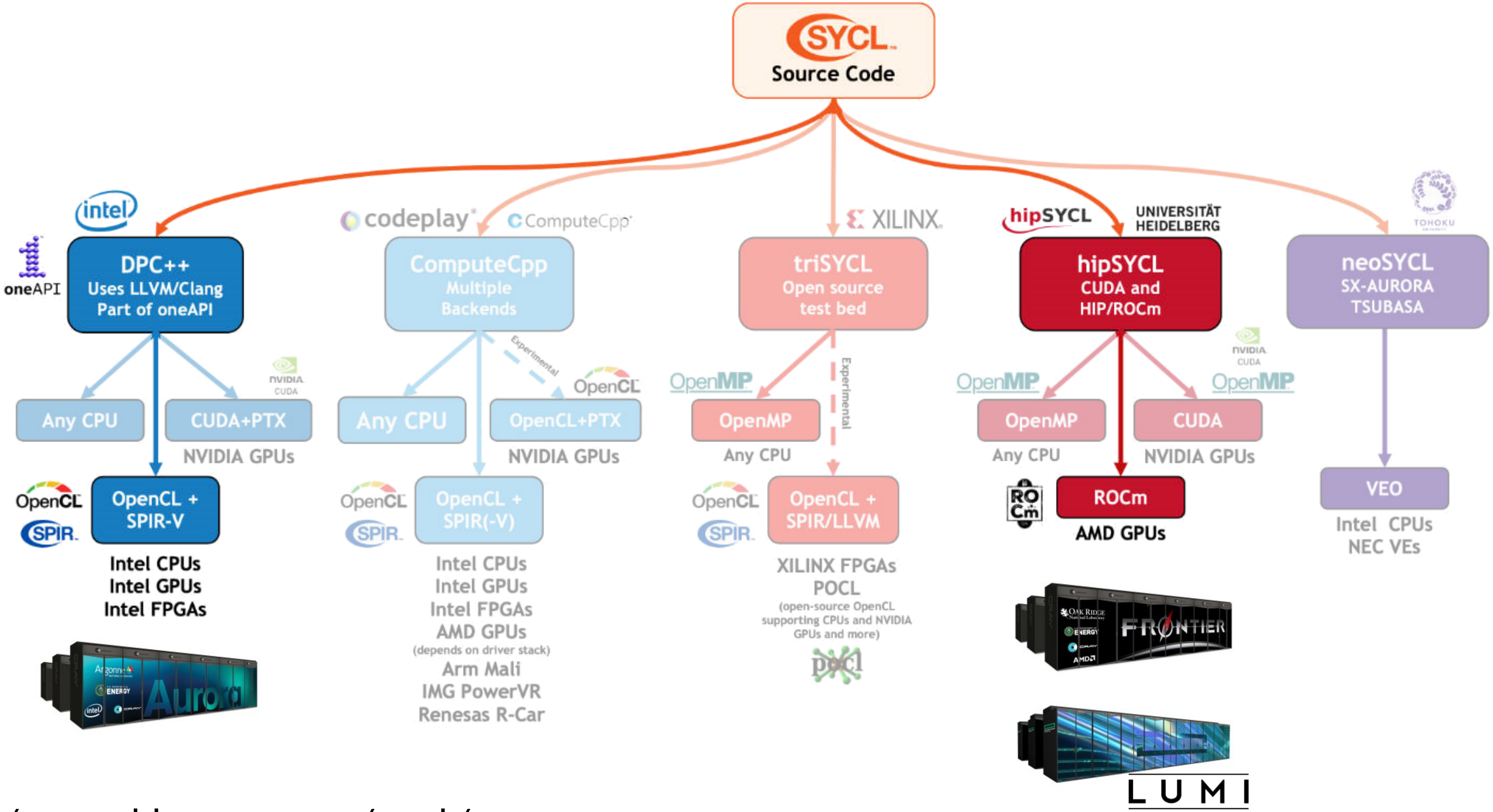


Páll et al., J. Chem. Phys. 153, 134110 (2020)

GPU APIs in GROMACS

	NVIDIA CUDA	OpenCL	SYCL
Maturity level	√	√	X
Open standard	X	√	√
Hardware support	Great NVIDIA-only	All major h/w, varying	Intel officially; NVIDIA and AMD 3rd party
Single-source model	√	X	√
Modern C++ support	√	X	√
In GROMACS	Main GPU backend for NVIDIA GPUs.	Primary support for AMD and Intel GPUs, partial support for NVIDIA. Deprecated in 2021.	In development. Early support in 2021.

SYCL ecosystem



https://www.khronos.org/sycl/

SYCL version requirements

- Kernels already highly optimized:
  - Lots of subgroup-level functionality
  - Floating-point atomics
- SYCL 1.2.1 is not enough!
- SYCL 2020 published on Feb 9, 2021
- DPC++ and hipSYCL implement some features differently
- A thin compatibility layer required

Subgroup operations:

NVIDIA CUDA	__any_sync	__shfl_up_sync
OpenCL	sub_group_any	N/A (intel_sub_group_shuffle_up)
SYCL	sycl::any_of_group	sycl::shift_group_right
oneAPI	cl::sycl::ONEAPI::any_of	cl::sycl::ONEAPI::sub_group::shuffle_up
hipSYCL	sycl::any_of	N/A (__shfl_up + PP magic)

Joys of C++

- No more duplicating structure definitions
- No more duplicating helper functions
- Templates instead of preprocessor:

OpenCL

```
#ifdef LJ_FORCE_SWITCH
#    ifdef CALC_ENERGIES
            calculate_force_switch_F_E(nbparam, c6, c12, inv_r, r2, &F_invr, &E_lj_p);
#    else
            calculate_force_switch_F(nbparam, c6, c12, inv_r, r2, &F_invr);
#    endif /* CALC_ENERGIES */
#endif     /* LJ_FORCE_SWITCH */
```

SYCL

```
if constexpr (props.vdwFSwitch)
{
    ljForceSwitch<doCalcEnergies>(
            nbparam, c6, c12, rInv, r2, &fInvR, &energyLJPair);
}
```

Joys of C++

OpenCL

```
#    ifndef LJ_COMB
     __local int* atib = (__local int*)(LOCAL_OFFSET); //NOLINT(google-readability-casting)
#        undef LOCAL_OFFSET
#        define LOCAL_OFFSET (atib + c_nbnxnGpuNumClusterPerSupercluster * CL_SIZE)
#    else
     __local float2* ljcpib      = (__local float2*)(LOCAL_OFFSET);
#        undef LOCAL_OFFSET
#        define LOCAL_OFFSET (ljcpib + c_nbnxnGpuNumClusterPerSupercluster * CL_SIZE)
#    endif
```

SYCL

```
auto sm_atomTypeI = [&]() {
     if constexpr (!props.vdwComb)
     {
         return cl::sycl::accessor<int, 2, mode::read_write, target::local>(
                 cl::sycl::range<2>(c_nbnxnGpuNumClusterPerSupercluster, c_clSize), cgh);
     }
     else { return nullptr; }
}();

auto sm_ljCombI = [&]() {
     if constexpr (props.vdwComb)
     {
         return cl::sycl::accessor<Float2, 2, mode::read_write, target::local>(
                 cl::sycl::range<2>(c_nbnxnGpuNumClusterPerSupercluster, c_clSize), cgh);
     }
     else { return nullptr; }
}();
```

GROMACS GPU support

- Originally designed for CUDA
- OpenCL added later

GROMACS GPU support

- Originally designed for CUDA
- OpenCL added later

- But most GPU frameworks are similar, right?
  1. Initialize device
  2. Allocate memory on device
  3. Copy initial data
  4. Launch a kernel spanning 1000s of threads
  5. Copy data back

GPU framework comparison

	NVIDIA CUDA	OpenCL	SYCL
Scheduling	in-order queue or explicit DAG	in-order and out-of-order queues	implicit DAG
Synchronization event	separate pseudo-task	associated with a task or a pseudo-task	associated with a task
Timing measurement	regions	of a single event	of a single event
Timing enablement	at event creation	at queue creation	at queue creation
Device selection	by special function	explicit in each call	explicit in each call
Resource management	manual	manual	RAII
Native float3 size	12 bytes	16 bytes	16 bytes but might also be 12
Sampling mode selection	at texture creation	in kernel	in kernel

DAG-based scheduling

- Good: Prevent bugs and improve performance
- Bad: GROMACS is built around for in-order queues, with explicit barrier synchronizations:
  - Performance: synchronizing twice
  - Correctness: device-to-host copies
    - D2H Copy A
    - D2H Copy B
    - Enqueue event
    - ...
    - Wait for the event // both A and B completed
- Bad: without priorities, DAG can miss the critical path
- Ugly: Additional divergence between backends

DAG-based scheduling

DAG

Explicit

7% speed-up

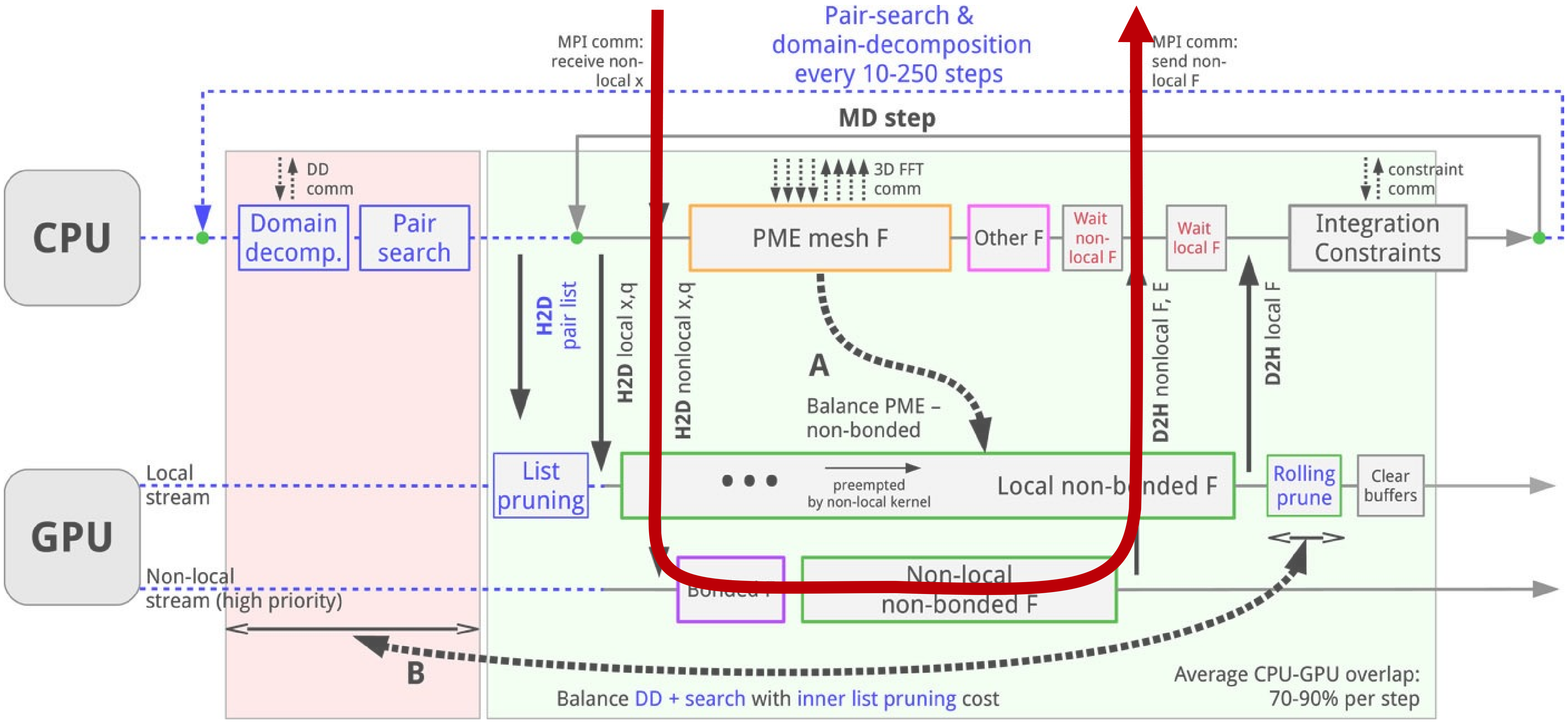
DAG-based scheduling

- Good: Prevent bugs and improve performance
- Bad: GROMACS is built around for in-order queues, with explicit barrier synchronizations:
  - Performance: synchronizing twice
  - Correctness: device-to-host copies
    - D2H Copy A
    - D2H Copy B
    - Enqueue event
    - …
    - Wait for the event // both A and B completed
- Bad: without priorities, DAG can miss the critical path
- Ugly: Additional divergence between backends

DAG-based scheduling

- Good: Prevent bugs and improve performance
- Bad: GROMACS is built around for in-order queues, with explicit barrier synchronizations:
  - Performance: synchronizing twice
  - Correctness: device-to-host copies
    - D2H Copy A
    - D2H Copy B
    - Enqueue event
    - …
    - Wait for the event // both A and B completed
- Bad: without priorities, DAG can miss the critical path
- Ugly: Additional divergence between backends

DAG-based scheduling



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

DAG-based scheduling

- Good: Prevent bugs and improve performance
- Bad: GROMACS is built around for in-order queues, with explicit barrier synchronizations:
  - Performance: synchronizing twice
  - Correctness: device-to-host copies
    - D2H Copy A
    - D2H Copy B
    - Enqueue event
    - ...
    - Wait for the event // both A and B completed
- Bad: without priorities, DAG can miss the critical path
- Ugly: Additional divergence between GPU backends

DAG-based scheduling

- Short-term: Event-based barrier synchronization
  - CUDA-like pseudo-task waiting for all previously submitted tasks
  - DPC++: SYCL_INTEL_enqueue_barrier extension
  - hipSYCL: hipEventSynchronize
  - Not optimal, but works
  - Same logic for all GPU backends
- Long-term: Refactoring
  - Queue-based scheduling unlikely to go away
  - DAG-based scheduling is nice, but has limitations
  - ???

Portability in practice: hipSYCL

- At start, only Intel DPC++ supported
    - hipSYCL added a bit later
- Effort:
    - Optimized kernels ported from OpenCL
    - Minor workarounds due to backend / compiler issues
- Performance:
    - Complex kernels much slower than HIP/OpenCL
        - Being investigated
    - Less complex kernels: on par with HIP/OpenCL

Conclusions

- “Write once, run anywhere” mostly works
  - Only trivial changes to support both DPC++ and hipSYCL
- But running fast is neither easy
  - Still need vendor-specific code branches to get high performance
- … nor guaranteed
  - On par with OpenCL with DPC++, even faster when using LevelZero
  - Occasional large regressions with hipSYCL
- Code is similar to OpenCL in spirit, but usually nicer
- Having same schedule code for both CUDA and SYCL is hard
  - CUDA Graphs + SYCL’s DAG?
  - Task priorities?

Other notes

- Using existing profiling tools is great (sometimes)
- Compilation is slow
  - Especially for multiple architectures
  - Especially with 168 templated kernel flavors in a single file
- The whole ecosystem is evolving rapidly

Acknowledgements

- Intel Corporation: postdoc scholarship to Andrey Alekseenko
- Heinrich Bockhorst and Roland Schulz
- GROMACS dev team, in particular Mark Abraham, Paul Bauer, and Artem Zhmurov

Learn more:

- https://gromacs.org/
- https://www.gromacs.org/Support/GMX-Developers_List
- https://gitlab.com/gromacs/gromacs/

- https://manual.gromacs.org/documentation/2021-sycl/download.html

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