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

MD loop overview

## GPU APIs in GROMACS

<table>
<thead>
<tr>
<th></th>
<th>NVIDIA CUDA (CUDA)</th>
<th>OpenCL</th>
<th>SYCL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maturity level</strong></td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Open standard</strong></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Hardware support</strong></td>
<td>Great NVIDIA-only</td>
<td>All major h/w, varying</td>
<td>Intel officially; NVIDIA and AMD 3rd party</td>
</tr>
<tr>
<td><strong>Single-source model</strong></td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Modern C++ support</strong></td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th></th>
<th>__any_sync</th>
<th>__shfl_up_sync</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA (CUDA)</td>
<td>sub_group_any</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(intel_sub_group_shuffle_up)</td>
</tr>
<tr>
<td>OpenCL</td>
<td>sycl::any_of_group</td>
<td>sycl::shift_group_right</td>
</tr>
<tr>
<td>SYCL</td>
<td>cl::sycl::ONEAPI::any_of</td>
<td>cl::sycl::ONEAPI::sub_group::shuffle_up</td>
</tr>
<tr>
<td>oneAPI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hipSYCL</td>
<td>sycl::any_of</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(__shfl_up + PP magic)</td>
</tr>
</tbody>
</table>
Joys of C++

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

```cpp
#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 */

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

```c++
#include <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

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

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

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/