Gaining Cross-Platform Parallelism for HAL's Molecular Dynamics Package using SYCL

September 16, 2023, PARS Workshop, Aachen

Viktor Skoblin[1], Felix Höfling[12], Steffen Christgau[1]

[1]Zuse Institute Berlin, [2]Freie Universität Berlin

Outline

- Motivation
- HAL's MD package
- Choice of programming model
- Code Migration from C++/CUDA to SYCL
- Performance Evaluation
- Summary

Motivation

Molecular Dynamics

- Molecular dynamics are important in classical statistical physics
- MD simulations demand resources, but can be parallelized
- Code and performance portability are important due to increasing vendor diversity in HPCs

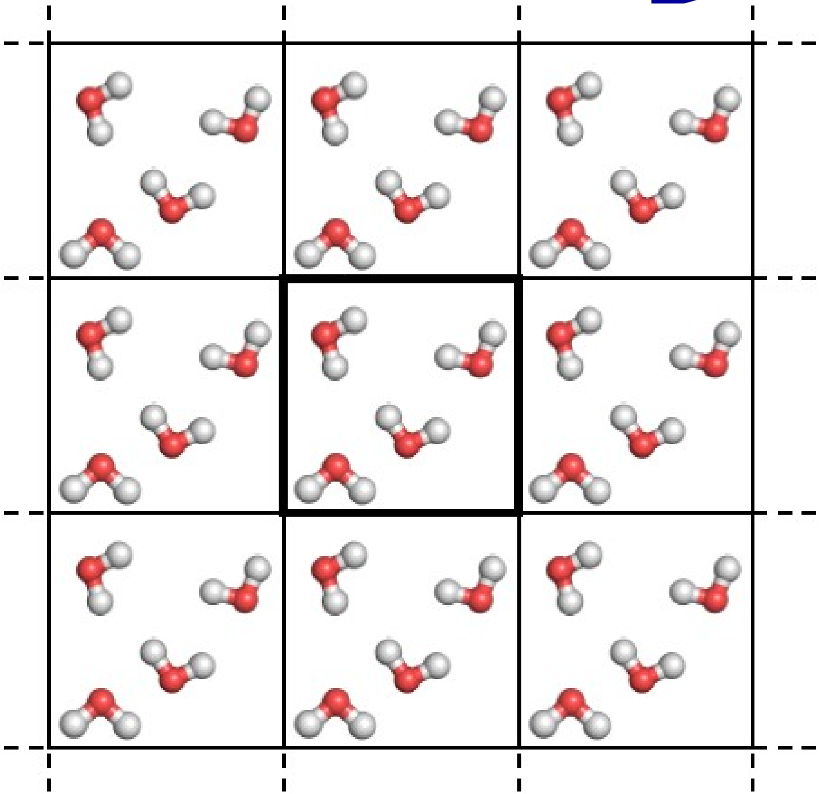
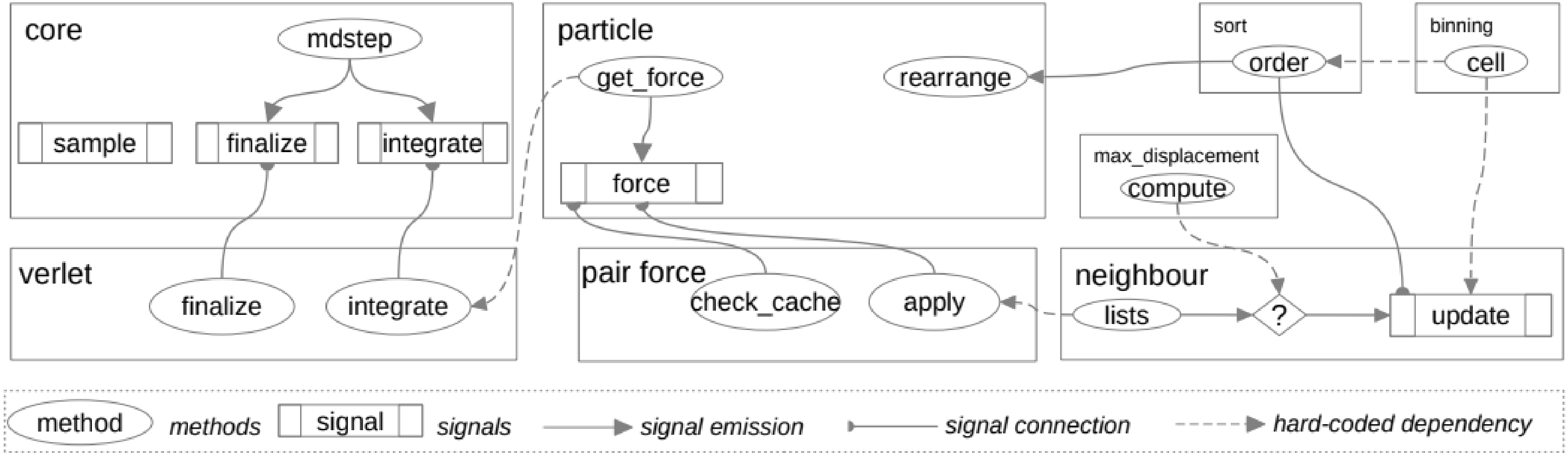


Image: https://www.researchgate.net/figure/Illustration-of-periodic-boundary-condition-in-MD-simulation-This-figure-was-created_fig5_344703503

HAL’s MD Package

Package overview

- High-precision molecular dynamics package
- Development started in 2007: early CUDA
- Focus on parallel execution on GPUs (NVidia)



Technical details

- Written in C++14
- Collection of modules with Lua scripting interface
- Modules have CUDA and C++ STL backends: templated code for different dimensions and precisions
- CUDA backend exists for each module (4.3k lines of CUDA code)
- Different CUDA memory types are used
- CUDA code is separated from the host code by a custom wrapper

Choice of programming model

Available open standards

- OpenMP
  - ➔ Higher level – less control
- OpenCL
  - ➔ Based on C (boilerplate)
  - ➔ Lacking vendor support
- SYCL
  - ➔ Native C++
  - ➔ Broad platform support

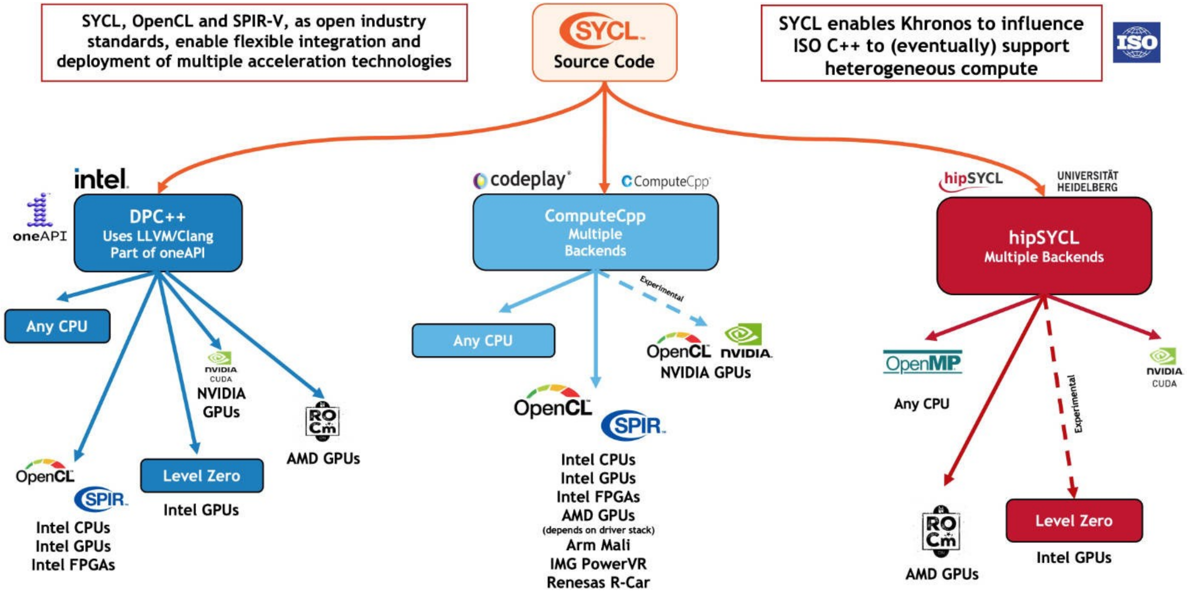


Image: https://www.khronos.org/sycl/

Code Migration from C++/CUDA to SYCL

Strategy 1: Migration with IntelDPC++ Compatibility Tool

- Intel DPC++ Compatibility Tool is designed to migrate existing CUDA codebase to SYCL
- Successful migration of the kernels _ _*global*_ _ functions
- Failed to migrate low-level API used for kernel invocations
- Problem: separation of host and device code in different compilation units

Code Migration from C++/CUDA to SYCL

Strategy 2: Sequential Code to SYCL

- Simple algorithms are easier to port adapting the sequential code
- Data management:
  - Shared allocations of Unified Shared Memory (USM)
  - Double-buffering mechanism (copying the data if updated <u>and</u> needed)
- Migration module-by-module: non-migrated functions work, data flow is optimized

Code Migration from C++/CUDA to SYCL

Strategy 3: Mixed

- More advanced algorithms can not be simply parallelized over the particles
- Utilize the migrated kernels and write the invocations manually
  - ➔ Use Intel Compatibility Tool
- Make changes to functional calls if needed

Code Migration from C++/CUDA to SYCL

Utilized approaches

- Using manual and mixed strategies (2 and 3)
- Determine the kernels launch configurations in non-trivial cases as well as kernel dependencies
- Textures → global memory
  - ➔ Texture memory is better suited for uncoalesced memory accesses

Code Migration from C++/CUDA to SYCL

Code Portability

- Code runs on CPU in parallel
- Code runs on different GPUs (AMD, Intel, NVIDIA)

Performance Evaluation

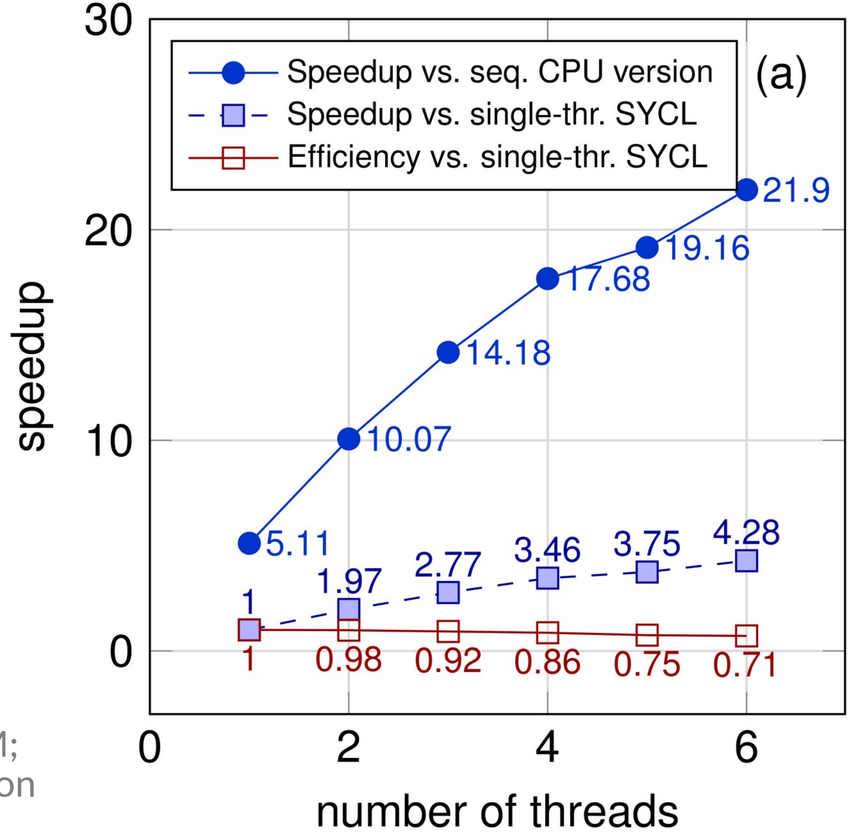
Studied Cases

1) All-to-all interactions
   - Small system (N ~ 3K particles)
   - Evaluated on CPU and GPU
2) Truncated interaction
   - System of a realistic size (N ~ 100K particles)

Performance Evaluation

CPU performance: all-to-all interactions

- Vectorized AVX-512 code
- Can be executed in parallel
- Parallel execution shows good scaling speedups
- Average efficiency 0.84

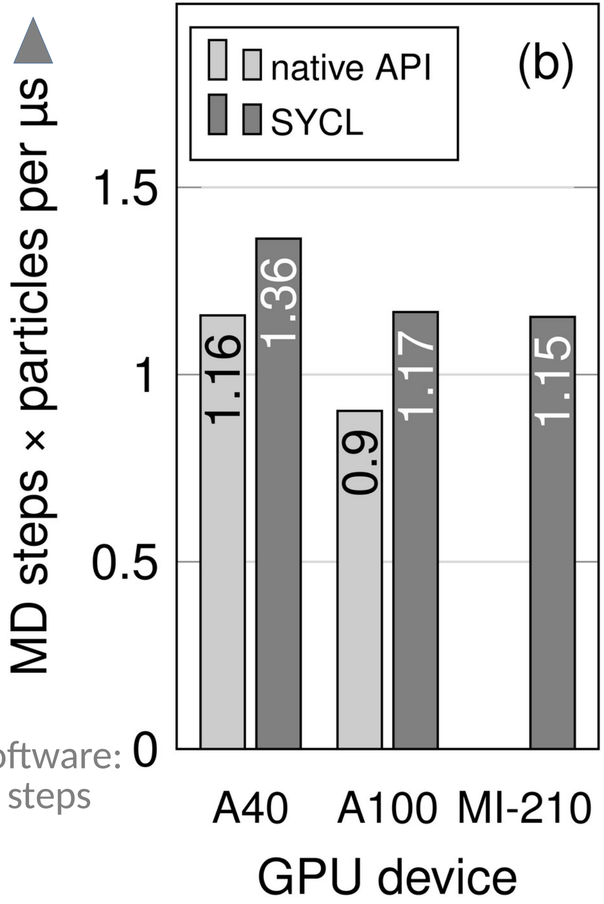


Hardware: 6-core Intel Xeon W-2133 (Cascade Lake), 32 GB RAM; Software: Debian 11, icpx 2023.1, *O3*, *mtune*, enable vectorization and math optimization; MD: 2K particles, 5K steps

Performance Evaluation

GPU performance: all-to-all interactions

- Runs on NVIDIA GPU faster than original CUDA version
- Runs on a comparable AMD GPU with comparable performance

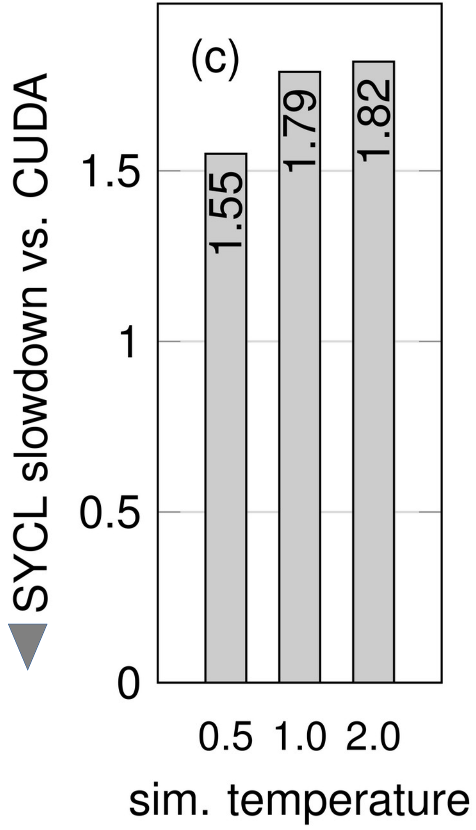


Hadrware: Nvidia A40 RTX 48GB, Nvidia A100 SXM4 80GB and AMD MI-210; Software: Original: CUDA 11.0.2; SYCL: LLVM 2022.09 + CUDA 11.7; MD: 2K particles, 20K steps

Performance Evaluation

GPU performance: truncated interactions

- Beneficial use of oneDPL for sorting on all GPUs
- Different code generation: lower utilization of FMA units
- Texture memory copes with unordered memory accesses => very limited support of SYCL images in Intel LLVM runtime



Hardware: Nvidia A40 RTX 48GB; Software: Original: CUDA 11.0.2; SYCL: Open-source Intel LLVM 2022.09 + CUDA 11.7; MD: 100K particles, 50K steps

Summary

- Achieved code portability for HAL’s MD Package
- Different migration strategies need to be considered
- Performance varies from case to case due to evolving compiler and runtime

Summary

ZIB
ZUSE INSTITUTE BERLIN

- Achieved code portability for HAL’s MD Package
- Different migration strategies need to be considered
- Performance varies from case to case due to evolving compiler and runtime

Thanks for your attention!