IWOCL 2024

The 12th International Workshop on OpenCL and SYCL

Powering Amber molecular dynamics simulations on GPUs with SYCL.

Andreas W. Götz, San Diego Supercomputer Center, University of California San Diego

Andreas Goetz, Guoquan Chen, Ram Ramanujam, Kittur Ganesh.

APRIL 8-11, 2024 | CHICAGO, USA | IWOCL.ORG

Acknowledgments

- Guoquan Chen (Intel)
  - Support from many Intel software engineers
- Ram Ramanujam (Intel)
- Kittur Ganesh (Intel)
- Intel oneAPI Centers of Excellence Program (Funding)

SDSC
SAN DIEGO SUPERCOMPUTER CENTER

intel.

What is Amber?

A set of force field parameters

Currently recommended fixed charge force fields

- Proteins: ff14SB
- DNA and RNA: OL15 and OL3
- Lipids: lipid17
- Carbohydrates: GLYCAM_06j

Parameters for general organic molecules

- GAFF2 (General Amber Force Field version 2)

Experimental polarizable force fields

- E.g. ff02EP

Parameters for solvents and ions

What is Amber?

A set of force field parameters

Currently recommended fixed charge force fields

- Proteins: ff14SB
- DNA and RNA: OL15 and OL3
- Lipids: lipid17
- Carbohydrates: GLYCAM_06j

Parameters for general organic molecules

- GAFF2 (General Amber Force Field version 2)

Experimental polarizable force fields

- E.g. ff02EP

Parameters for solvents and ions

A molecular dynamics simulation package

- Around for over 30 years
- Approx. 50 principal contributors to current codes
- Independent of accompanying force fields (supports also CHARMM force field)
- Distributed in two parts

AmberTools

- Preparatory and analysis programs
- Molecular dynamics programs **`sander`**, **`mdgx`**
- Open source, mostly GPL
- Yearly release (spring)

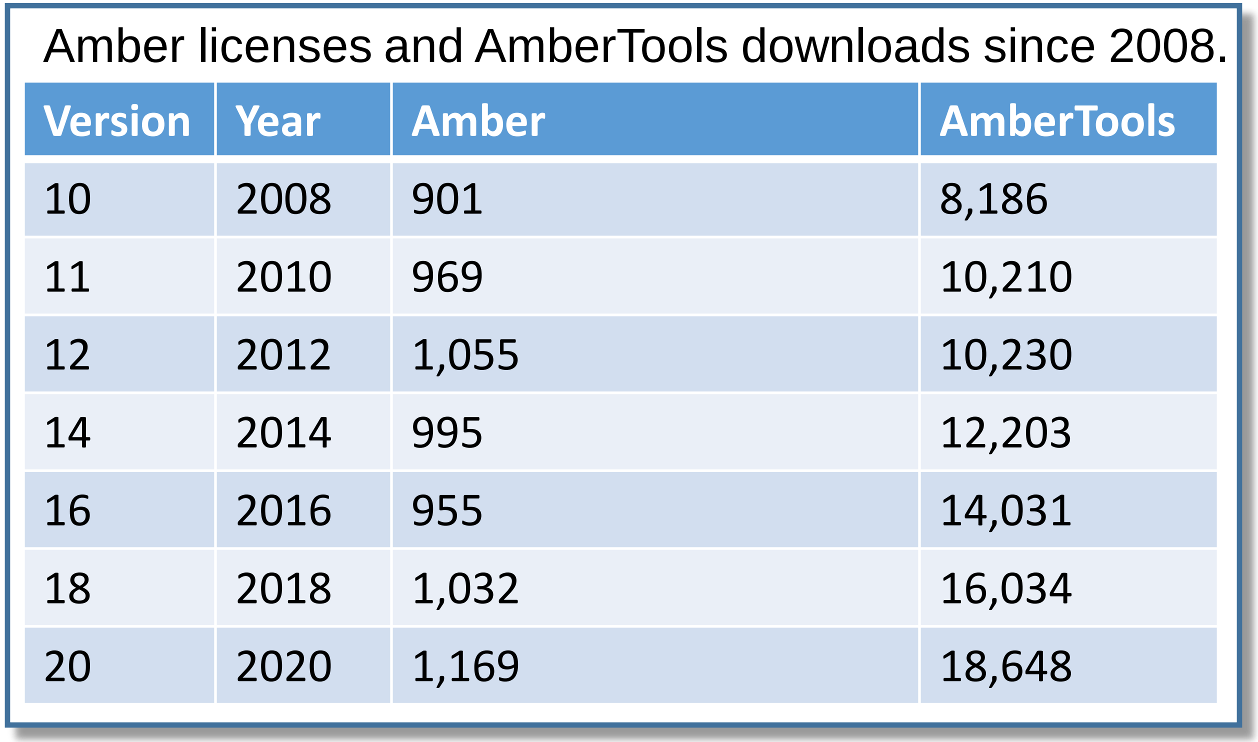
Amber

- High-performance MD program **`pmemd`**
- Academic licensing
- Release every even year

Amber is widely used

Amber user base

- Academic and industrial research
- User base steadily growing
- Installed on all major compute centers
- Popularity has been growing in particular since about 2010
  - Release of CUDA code
  - Wide availability of GPUs enabled science for large user base



Amber licenses and AmberTools downloads since 2008.

Version	Year	Amber	AmberTools
10	2008	901	8,186
11	2010	969	10,210
12	2012	1,055	10,230
14	2014	995	12,203
16	2016	955	14,031
18	2018	1,032	16,034
20	2020	1,169	18,648

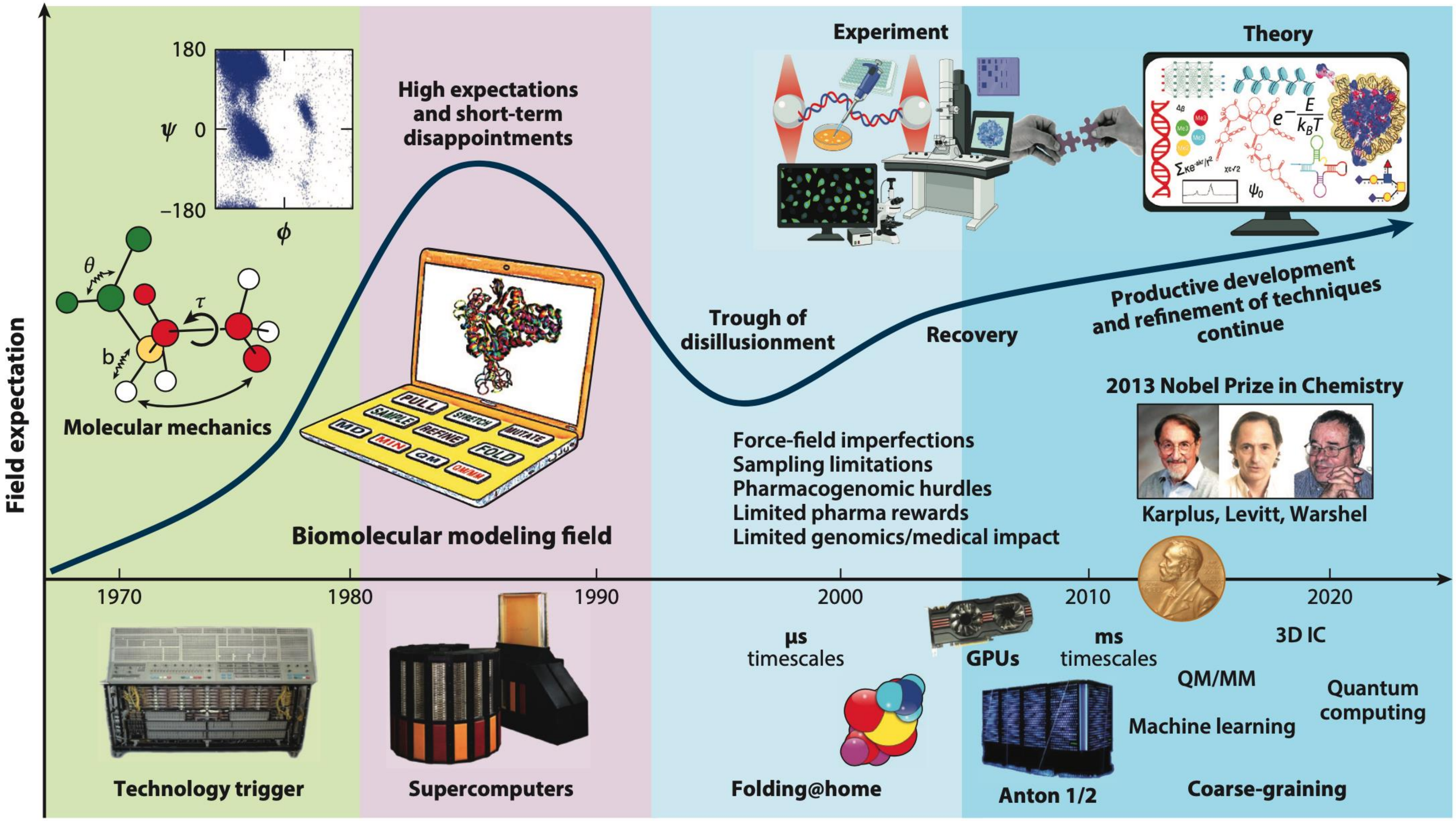
Routine microsecond molecular dynamics simulations with AMBER on GPUs. 2. Explicit solvent particle mesh Ewald 2926 2013
R Salomon-Ferrer, AW Gotz, D Poole, S Le Grand, RC Walker
Journal of chemical theory and computation 9 (9), 3878-3888

PME GPU implementation over 250 citations per year!

Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. Generalized born 1395 2012
AW Götz, MJ Williamson, D Xu, D Poole, S Le Grand, RC Walker
Journal of chemical theory and computation 8 (5), 1542

SPFP: speed without compromise—a mixed precision model for GPU accelerated molecular dynamics simulations 1020 2012
SL Grand, AW Götzx, RC Walker
Computer Physics Communications

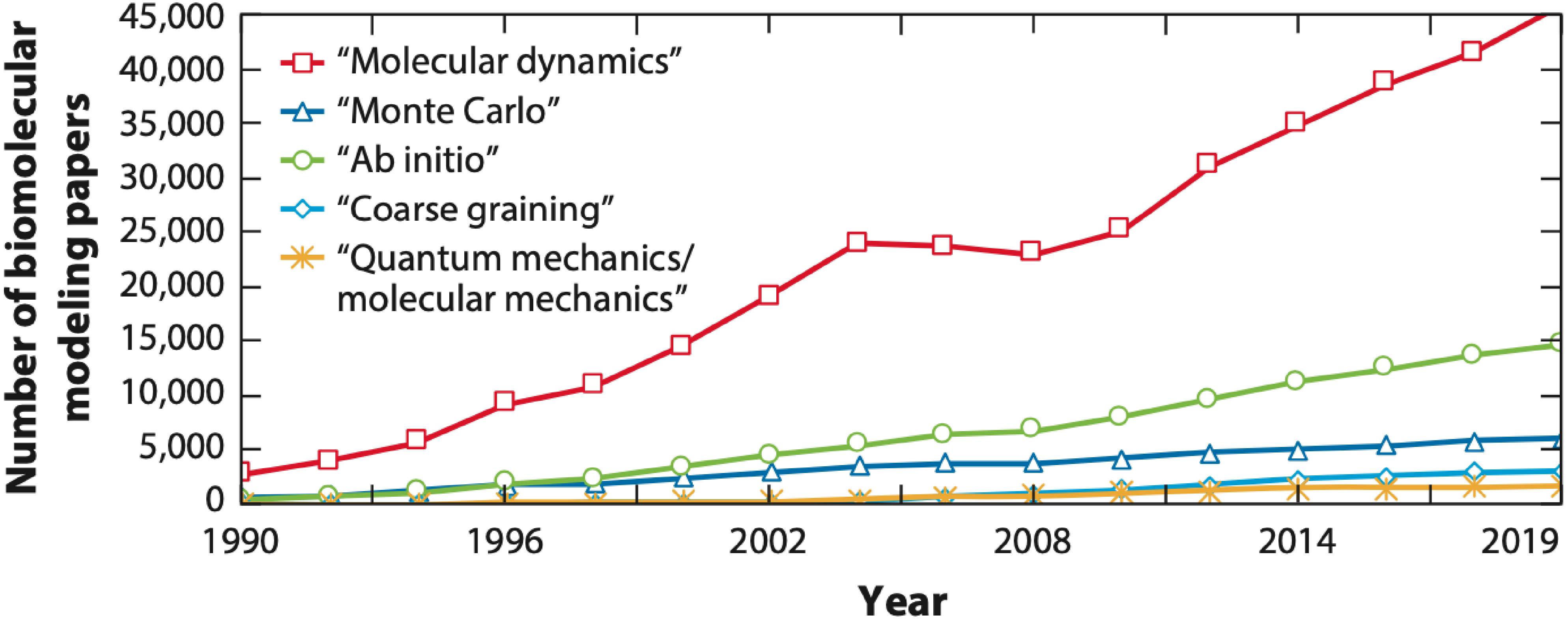
Biomolecular modeling over the years



Schlick *et al.* *Annu. Rev. Biophys.* **50** (2021) 267.

Amber is widely used

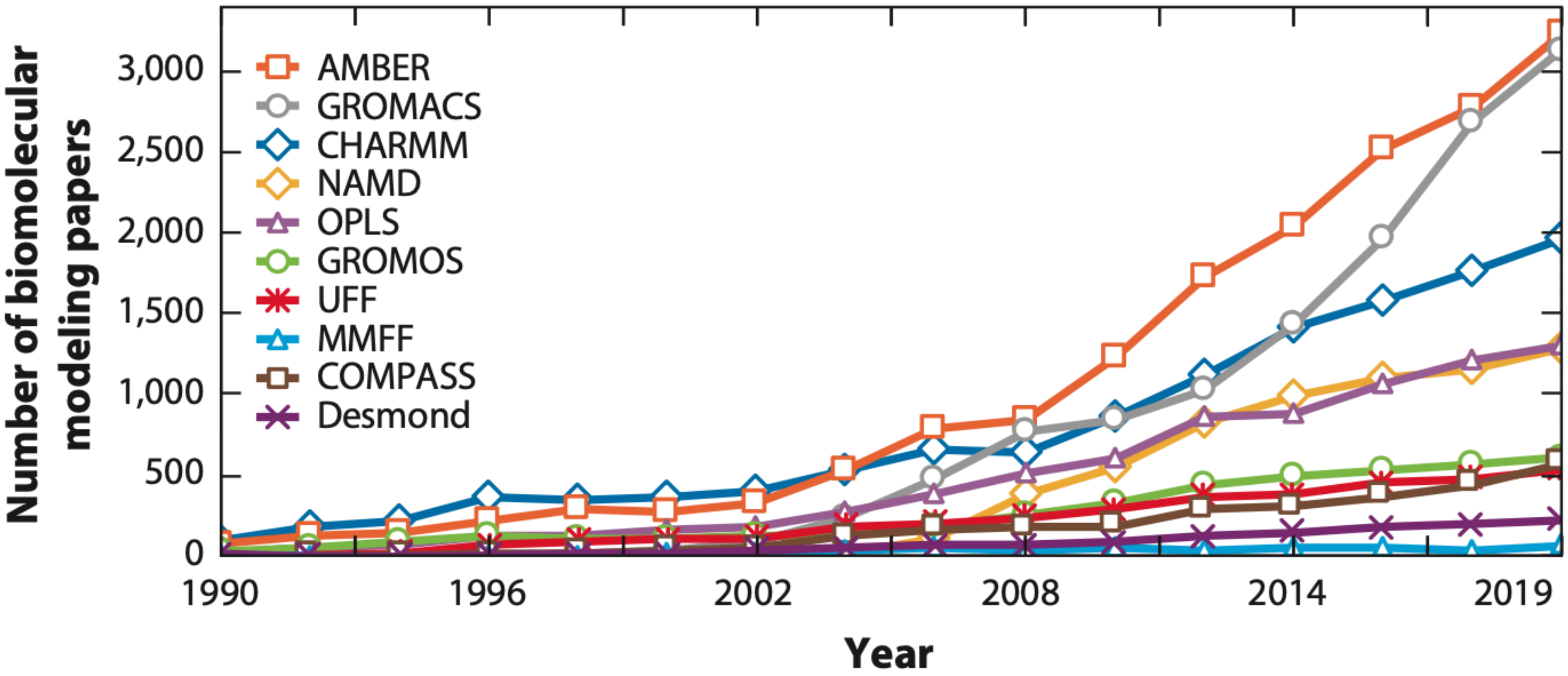
Popularity of Molecular Dynamics is Growing



Schlick *et al. Annu. Rev. Biophys.* **50** (2021) 267.

Amber is widely used

Amber is among the most popular codes for biomolecular simulations



Schlick *et al. Annu. Rev. Biophys.* **50** (2021) 267.

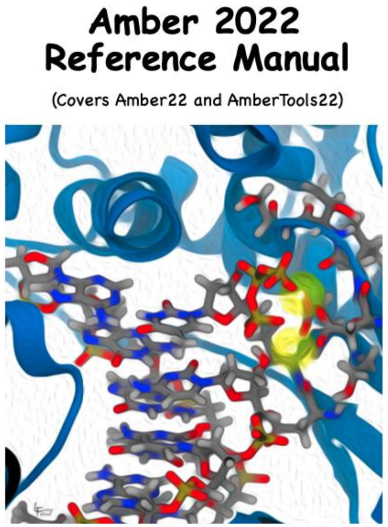
Amber typical applications

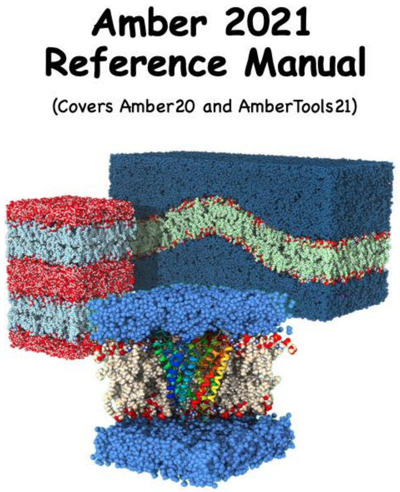
Biomolecular simulations

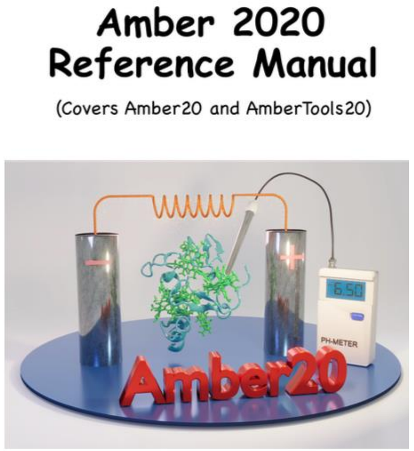
- Proteins, DNA, RNA, lipids, sugars, glycoproteins
- Building blocks of cells and tissue
- Understand how life works at the nanoscale
- Understand biomolecular function and disease
- Computational drug design

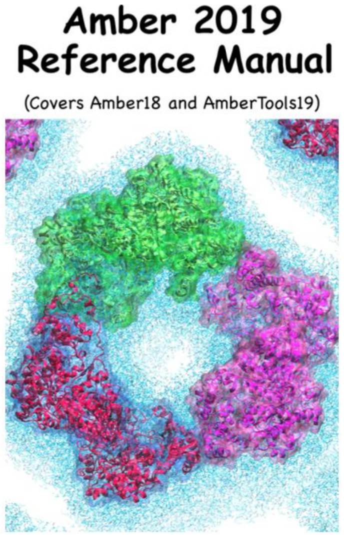
Molecular dynamics simulations

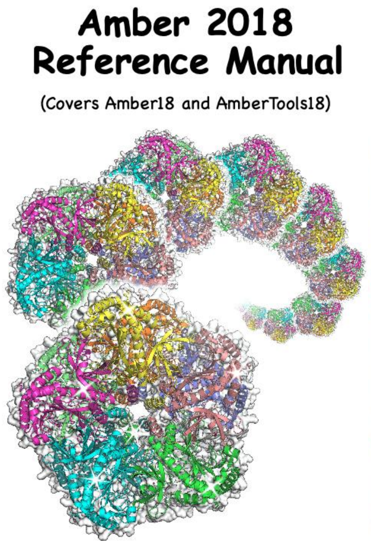
- Structure and function of biomolecular systems
- Protein folding
- Enzymatic reaction mechanisms
- Stability of protein / drug complexes
- Part of drug design pipeline (Improve drug leads via free energy simulations)
- And many more ….

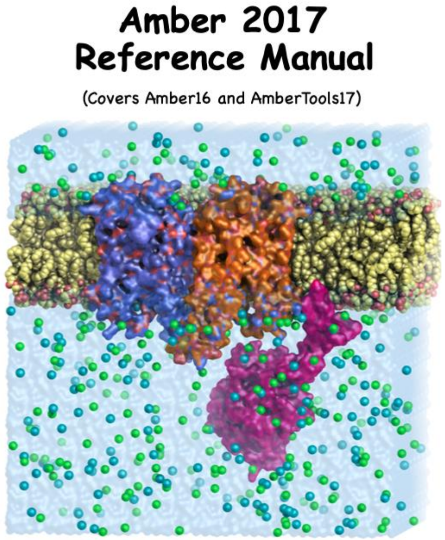

Amber 2022
Reference Manual
(Covers Amber22 and AmberTools22)


Amber 2021
Reference Manual
(Covers Amber20 and AmberTools21)


Amber 2020
Reference Manual
(Covers Amber20 and AmberTools20)


Amber 2019
Reference Manual
(Covers Amber18 and AmberTools19)


Amber 2018
Reference Manual
(Covers Amber18 and AmberTools18)


Amber 2017
Reference Manual
(Covers Amber16 and AmberTools17)

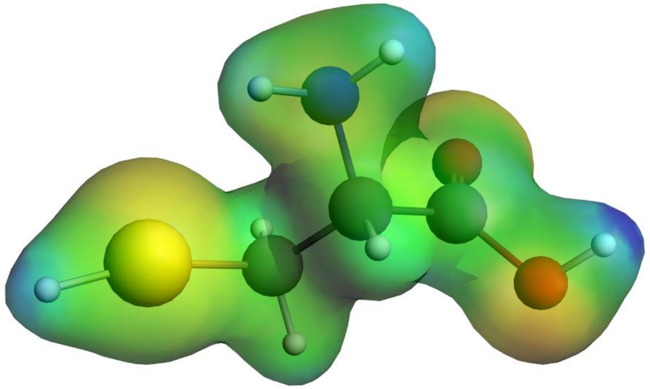
Modeling molecular interactions

Quantum Mechanics

$$\hat{H}\Psi = E\Psi$$

$$E[\rho] = T[\rho] + V_{ee}[\rho] + V_{xc}[\rho]$$

Solve Schrödinger equation for many-electron system



Electron density of amino acid cystein

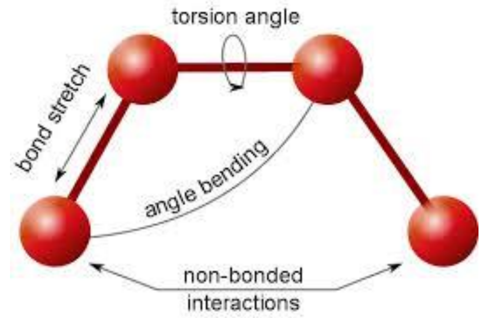
Modeling molecular interactions

Quantum Mechanics

$$\hat{H}\Psi = E\Psi$$

$$E[\rho] = T[\rho] + V_{ee}[\rho] + V_{xc}[\rho]$$

Simple “ball and stick model” for molecules



Molecular Mechanics Force Fields

$$V(\vec{R}^N) = \sum_{bonds} \frac{k_b}{2}(r_b - r_{b,o})^2 + \sum_{angles} \frac{k_a}{2}(\theta_a - \theta_{a,o})^2 + \sum_{torsion} \left[\sum_n \frac{V_n}{2}(1 + \cos(n\omega - \gamma))\right]$$

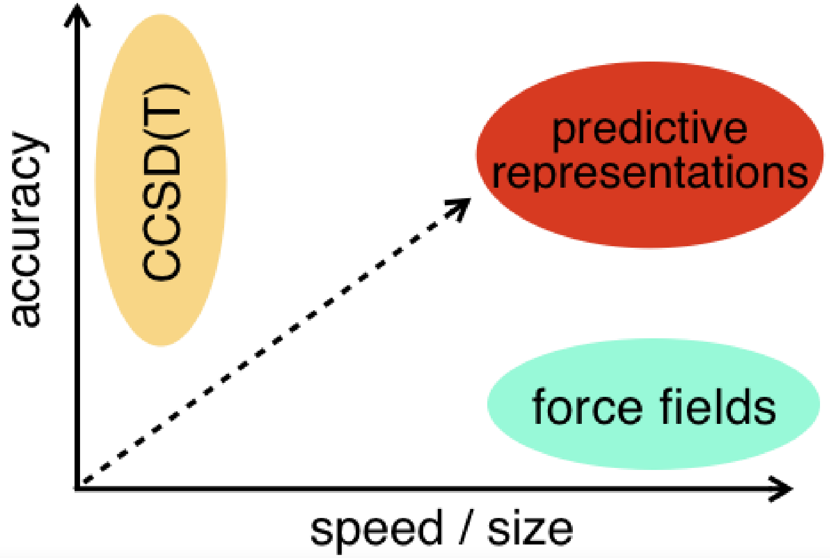
$$+ \sum_{i<j} \left\{ 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6} \right] + \frac{q_i q_j}{4\pi\epsilon_0 R_{ij}} \right\}$$

Modeling molecular interactions

Quantum Mechanics

$$\hat{H}\Psi = E\Psi$$

$$E[\rho] = T[\rho] + V_{ee}[\rho] + V_{xc}[\rho]$$

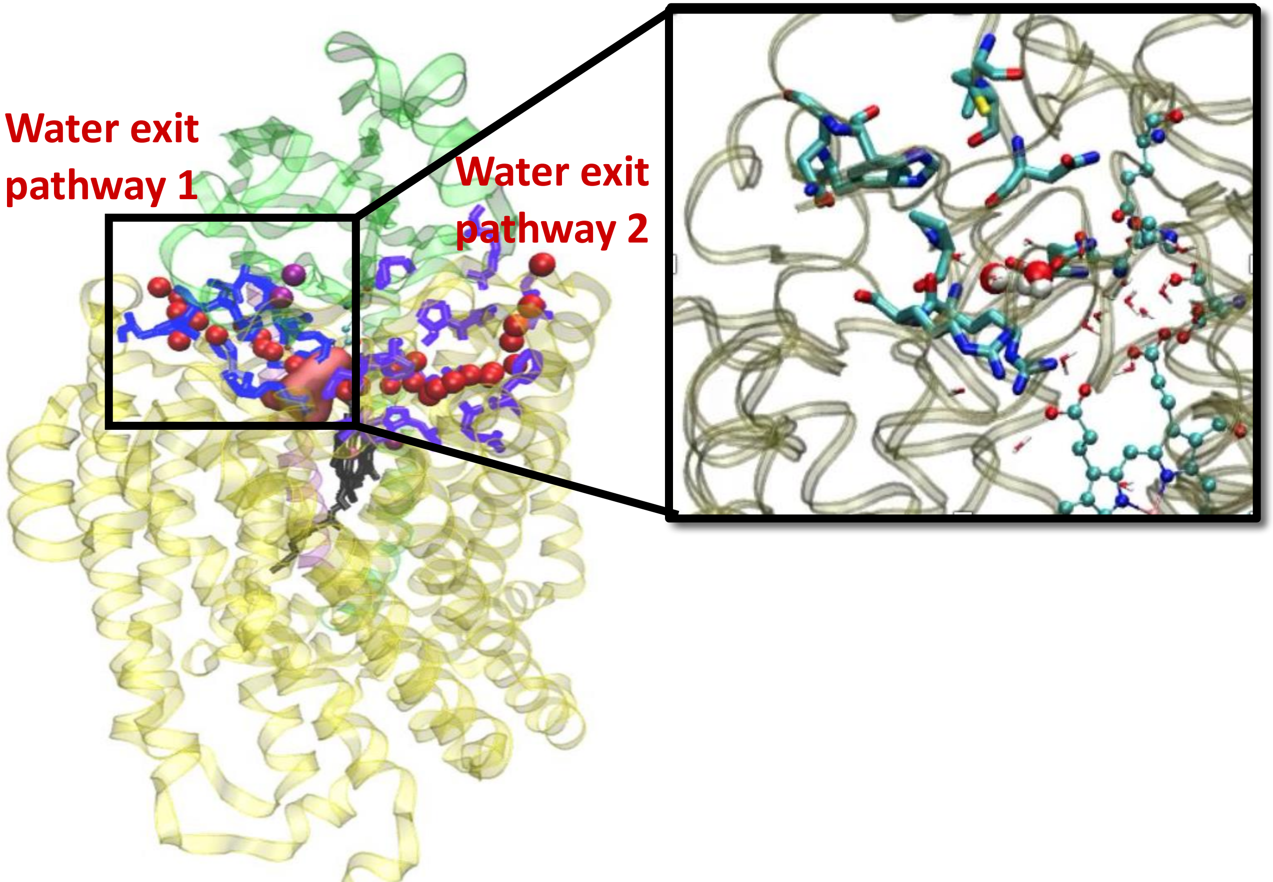


Molecular Mechanics Force Fields

$$V(\vec{R}^N) = \sum_{bonds} \frac{k_b}{2}(r_b - r_{b,o})^2 + \sum_{angles} \frac{k_a}{2}(\theta_a - \theta_{a,o})^2 + \sum_{torsion} \left[\sum_n \frac{V_n}{2}(1 + \cos(n\omega - \gamma)) \right]$$

$$+ \sum_{i<j} \left\{ 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6} \right] + \frac{q_i q_j}{4\pi\epsilon_0 R_{ij}} \right\}$$

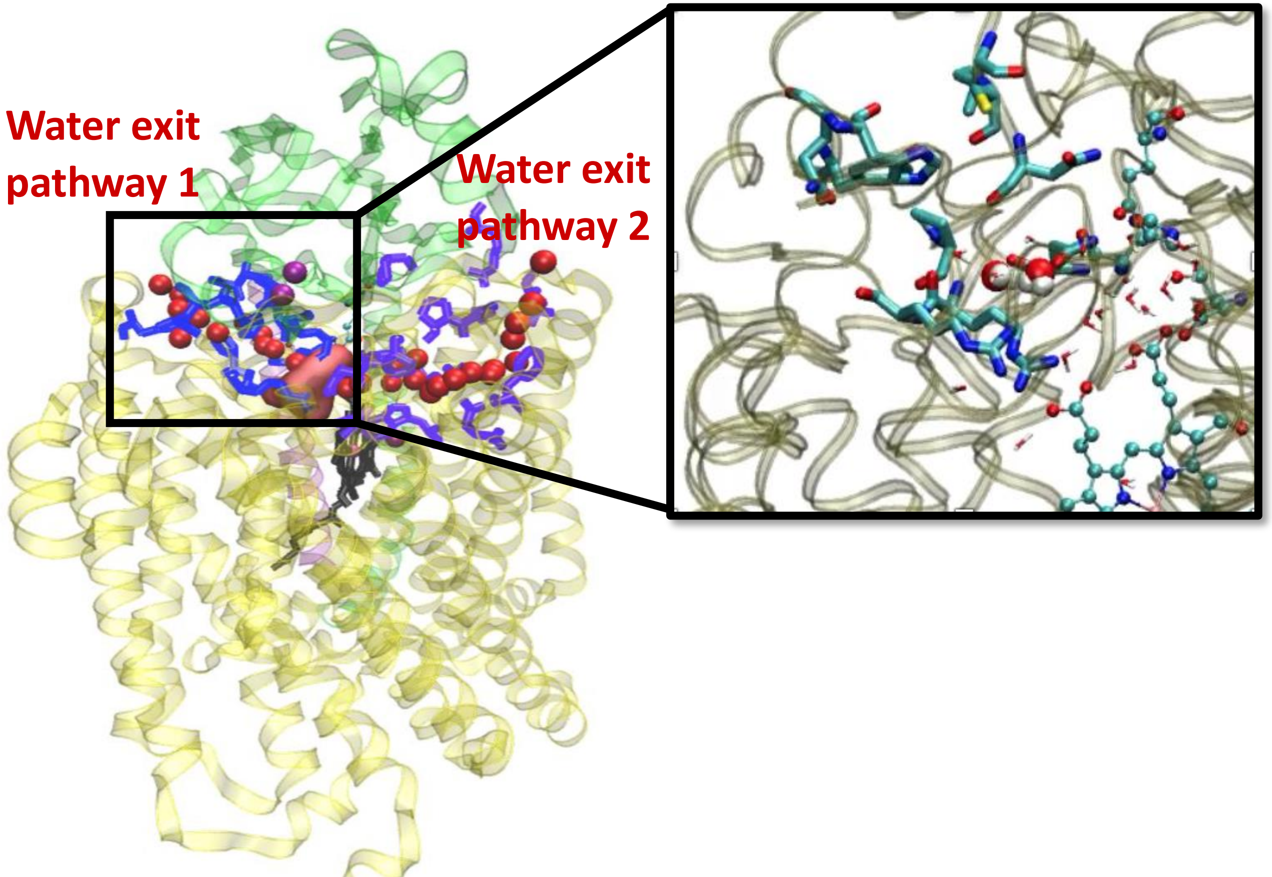
Molecular dynamics powered by GPUs



Cytochrome c oxidase enzyme

Yang, Skjevik, Han Du, Noodleman, Walker, Götz,
BBA Bioenergetics 2016 (1857) 1594.

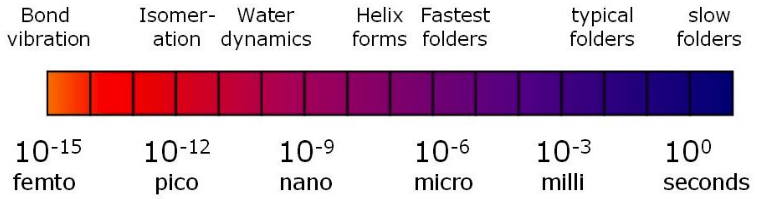
Molecular dynamics powered by GPUs



Cytochrome c oxidase enzyme

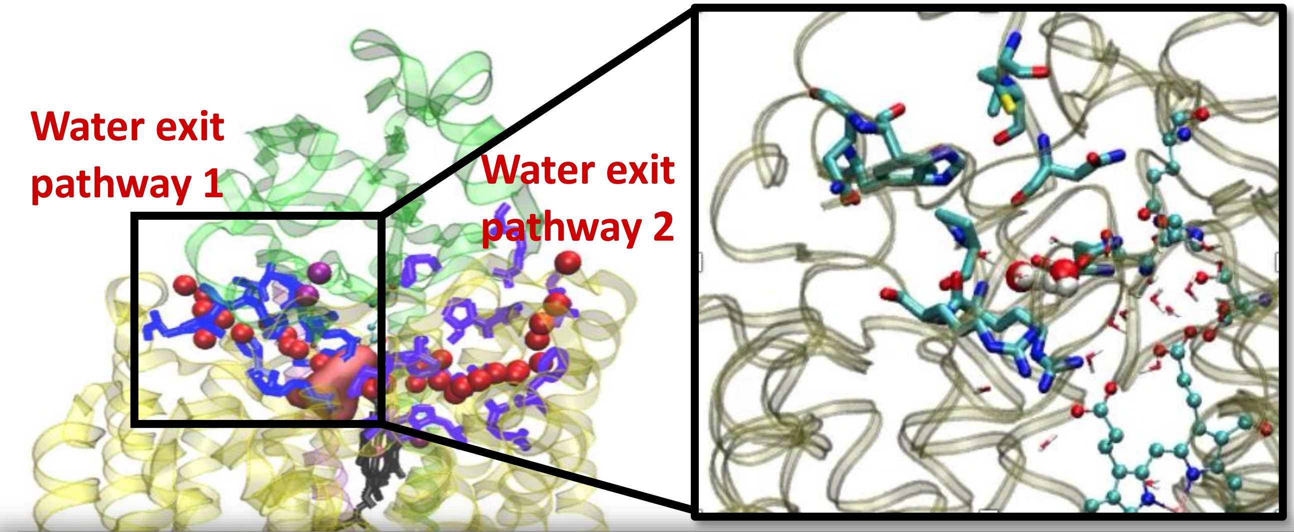
Yang, Skjevik, Han Du, Noodleman, Walker, Götz, *BBA Bioenergetics* 2016 (1857) 1594.

Relevant timescales

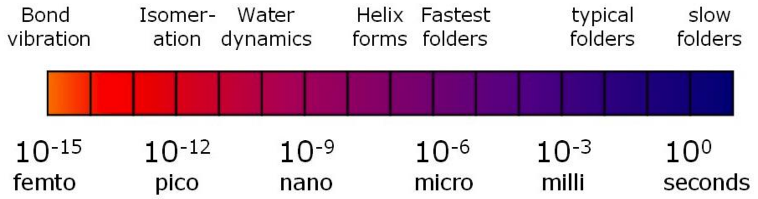


- Femtosecond timesteps
- Need to simulate micro to milliseconds
- 100s of millions of time steps required
- < 1 millisecond wall clock per time step

Molecular dynamics powered by GPUs



Relevant timescales



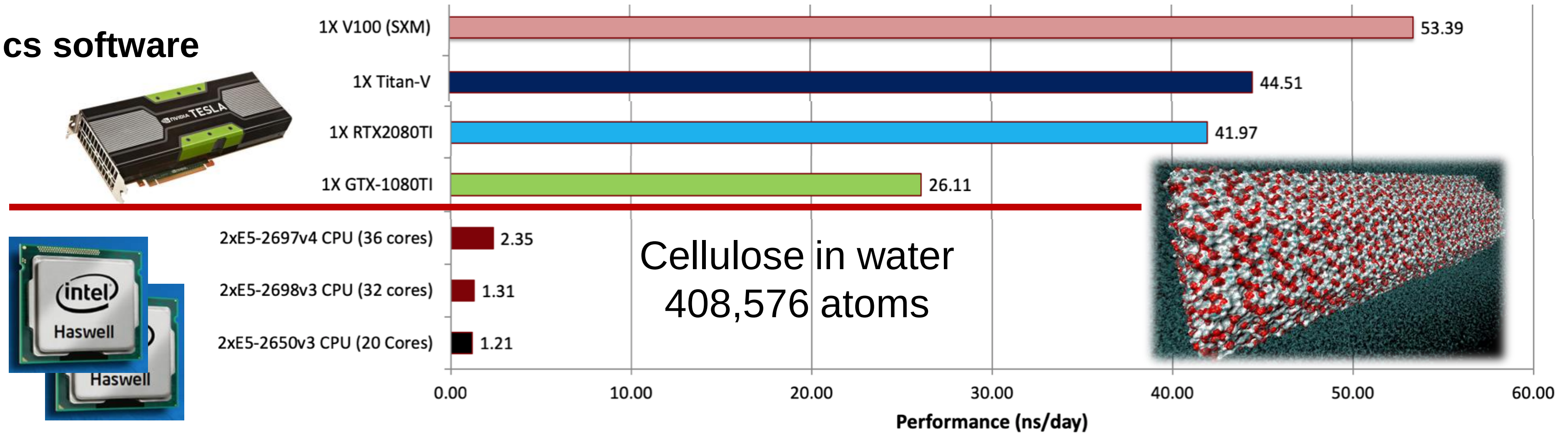
- Femtosecond timesteps
- Need to simulate micro to milliseconds
- 100s of millions of time steps required
- < 1 millisecond per time step

Amber 18 molecular dynamics software

Götz, Williamson, Xu, Poole, Le Grand, Walker, *J Chem Theory Comput* 2012 (8) 1542.

Le Grand, Götz, Walker, *Comput Phys Comm* 2013 (184) 374.

Salomon-Ferrer, Götz, Poole, Le Grand, Walker, *J Chem Theory Comput* 2012 (8) 1542.



Cellulose in water
408,576 atoms

Porting Amber to SYCL – Motivation

Amber CUDA code

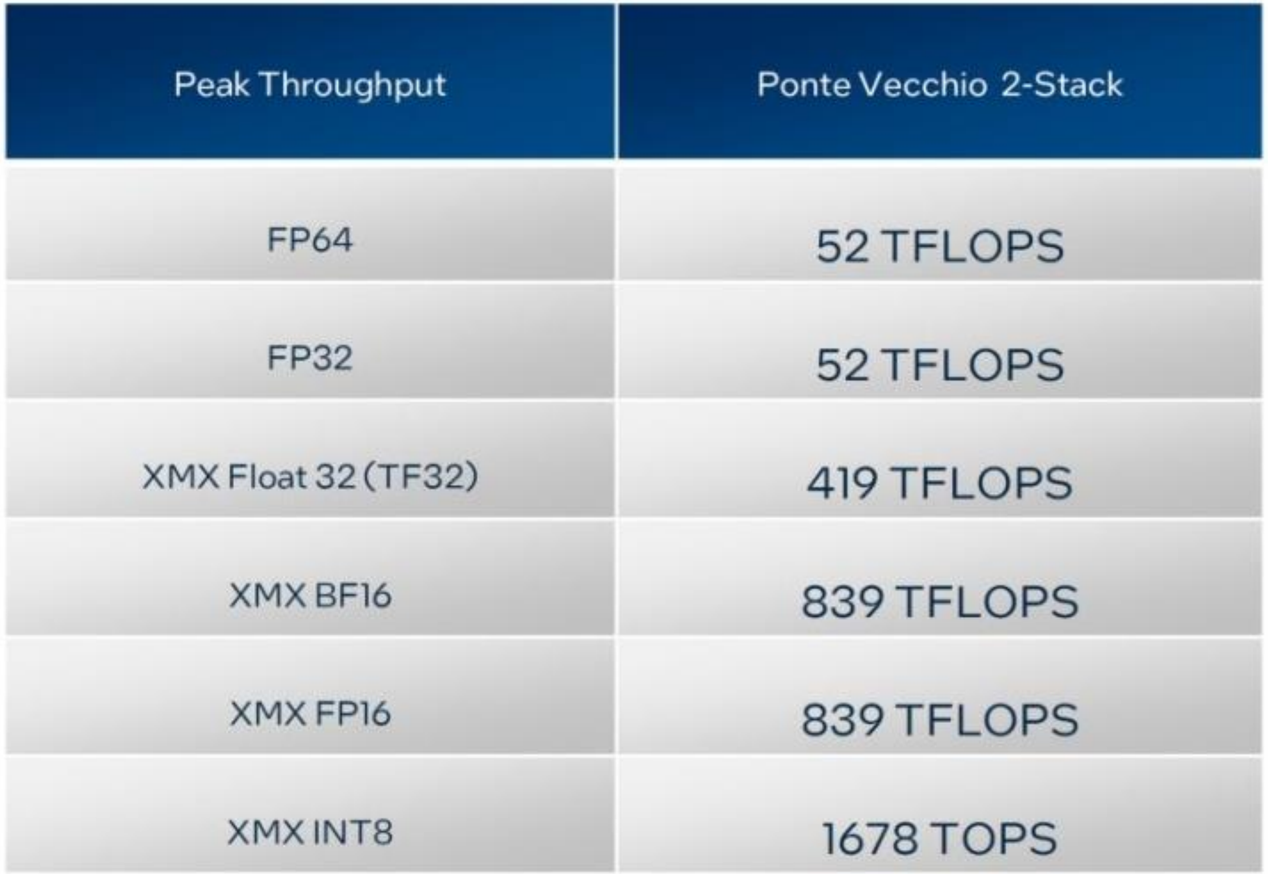
- Very successful and popular code
- Fast classical molecular dynamics code for Nvidia GPUs
- Enables microsecond MD simulations on single gaming GPUs
- Enables large scale ensemble simulations on GPU clusters and large supercomputers

Why then do we want to port Amber to SYCL?

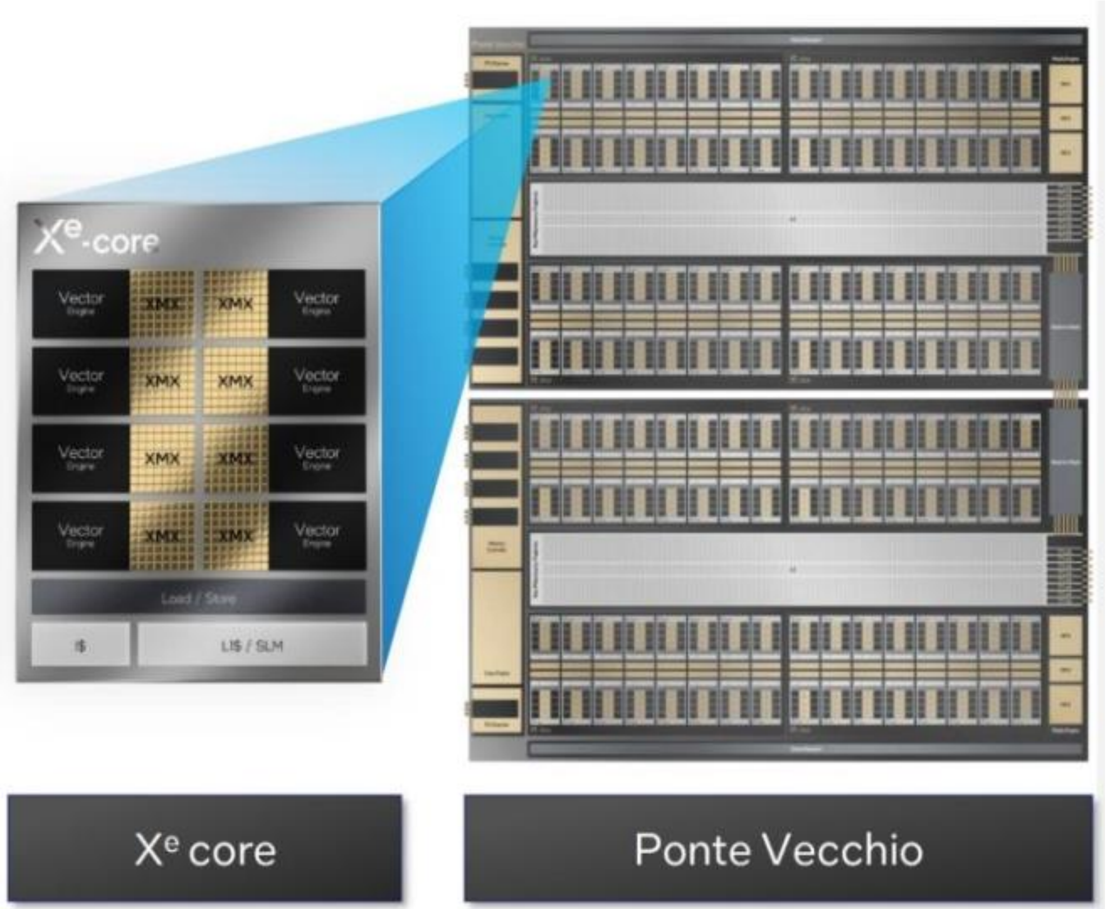
- We want to enable Amber to run on Intel GPUs
- SYCL is an open standard
- Performance portability
  - SYCL works on multi-core CPUs and accelerators from all vendors (Nvidia, AMD, Intel)
- Single code base should keep code maintainable
  - (currently Amber uses Fortran for CPU, CUDA for Nvidia GPUs, HIP/ROCm for AMD GPUs)

Intel Data Center GPU Max powers the Aurora Supercomputer at ANL.

Intel Data Center GPU Max (PVC): Performance



Peak Throughput	Ponte Vecchio 2-Stack
FP64	52 TFLOPS
FP32	52 TFLOPS
XMX Float 32 (TF32)	419 TFLOPS
XMX BF16	839 TFLOPS
XMX FP16	839 TFLOPS
XMX INT8	1678 TOPS



XMX: X^e Matrix Extensions

Source: https://wccftech.com/intel-details-ponte-vecchio-gpu-sapphire-rapids-hbm-performance-up-to-2-5x-faster-than-nvidia-a100/

Porting Amber to SYCL: Challenges

Amber GPU accelerated MD code: pmemd.cuda

- Large set of highly optimized kernels for particle force calculations,
  time stepping, temperature and pressure control and enhanced sampling algorithms

Many different methods with different code paths

- Implicit (GB) and explicit solvent (PME) molecular dynamics
- Different thermostats and barostats (for NpT and NVT simulations)
- Many enhanced sampling and free energy algorithms
  - Restraints (e.g. for umbrella sampling)
  - Steered molecular dynamics
  - Thermodynamic integration
  - Etc.
- Ensemble simulation methods
  - Temperature- and Hamiltonian- replica exchange
  - Constant pH and constant electrochemical potential methods

Porting Amber to SYCL: Challenges

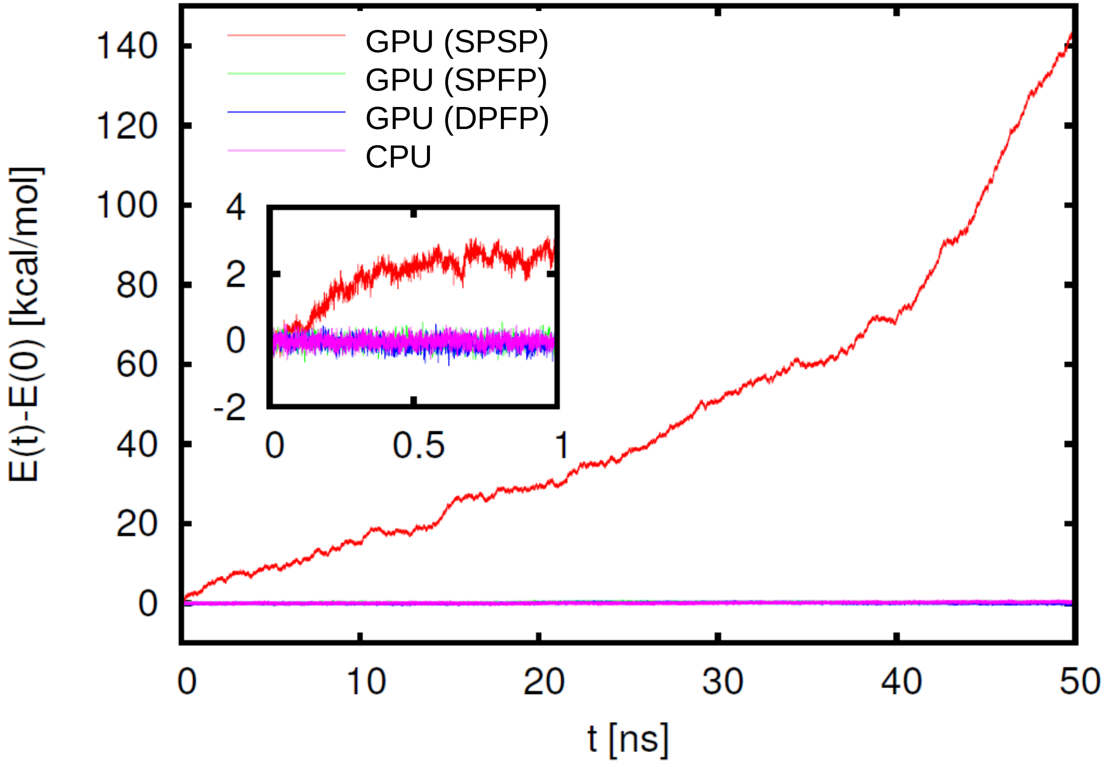
Technical details

- Multiple precision models
  - SPFP (default)
  - DPFP (slow, accurate, reference)
- Single- and multiple GPU support
  - CUDA
  - MPI + CUDA
- CUDA intrinsics
- PTX inline assembly code

CUDA code history

- Many years of initial development
- Over a decade of contributions from dozens of developers
- Not easy to re-write from scratch

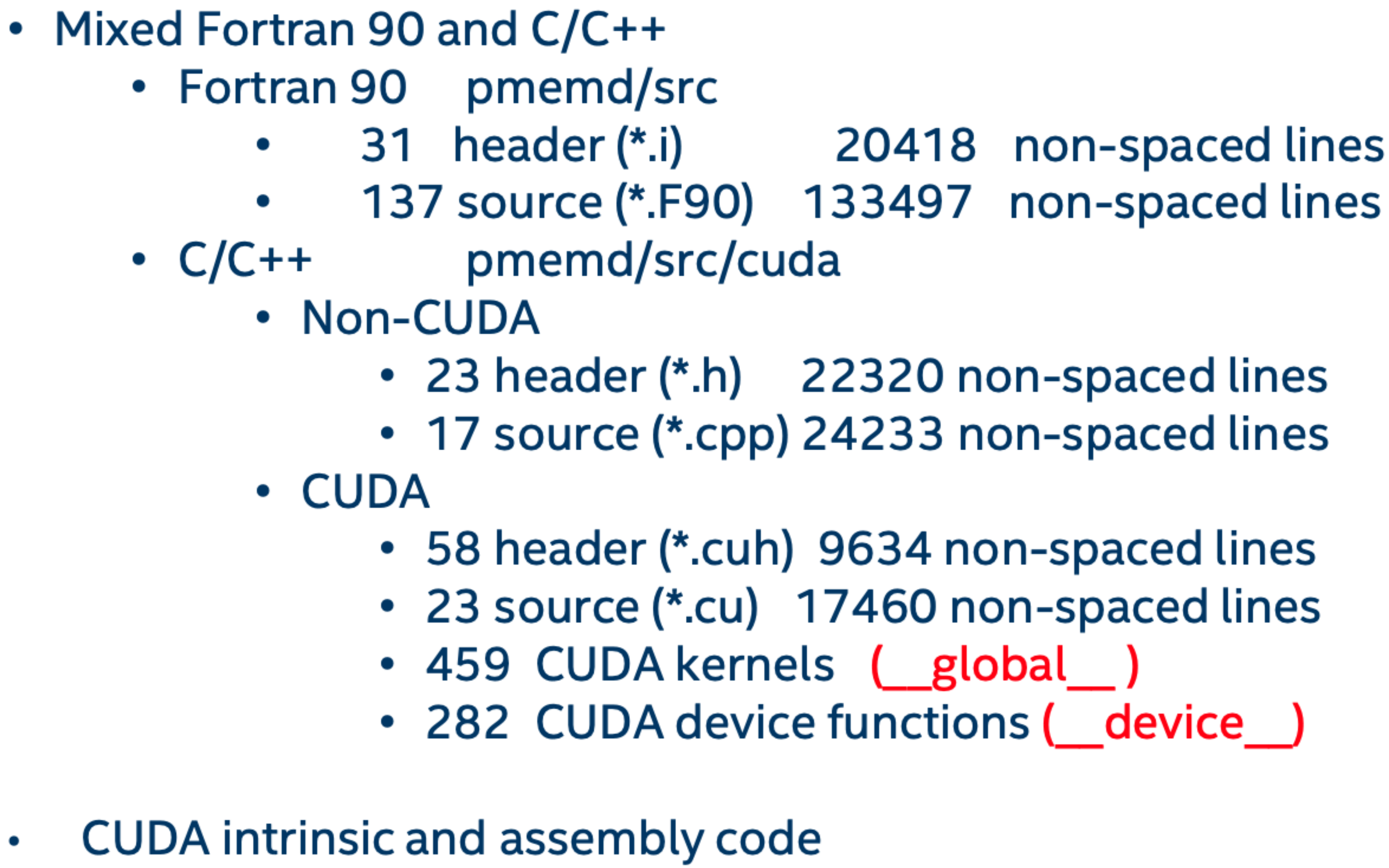
MD Energy conservation with various precision models



SP = FP32 (single precision floating point)
FP = 64-bit fixed precision
DP = FP64 (double precision floating point)
CPU

Porting Amber to SYCL: Challenges

Amber pmemd source code statistics



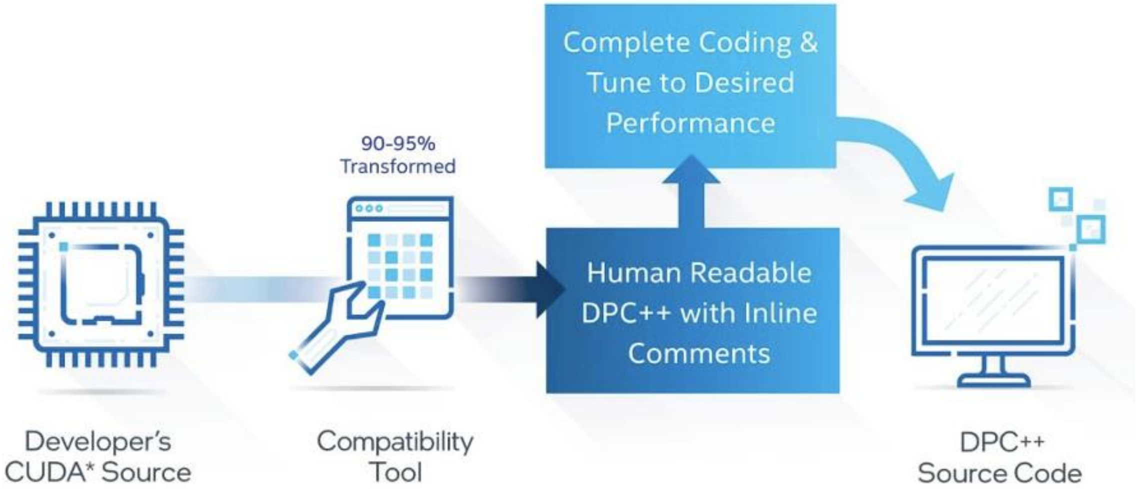
- Mixed Fortran 90 and C/C++
  - Fortran 90 pmemd/src
    - 31 header (*.i) 20418 non-spaced lines
    - 137 source (*.F90) 133497 non-spaced lines
  - C/C++ pmemd/src/cuda
    - Non-CUDA
      - 23 header (*.h) 22320 non-spaced lines
      - 17 source (*.cpp) 24233 non-spaced lines
    - CUDA
      - 58 header (*.cuh) 9634 non-spaced lines
      - 23 source (*.cu) 17460 non-spaced lines
      - 459 CUDA kernels (__global__)
      - 282 CUDA device functions (__device__)
- CUDA intrinsic and assembly code

Porting Amber from CUDA to SYCL

Steps from CUDA to a working SYCL version of Amber

- Migrate CUDA code to SYCL via Intel DPC++ Compatibility Tool (equivalent open-source version: SYCLomatic)
- Convert CUDA intrinsic and assembly code
- Compile using Intel oneAPI DPC++ compiler; fix compile time errors & warnings
- Debug and verify functional correctness
- Profile and optimize on Intel discrete GPU

Intel® DPC++ Compatibility Tool Usage Flow


Complete Coding & Tune to Desired Performance
90-95% Transformed
Human Readable DPC++ with Inline Comments
Developer's CUDA* Source
Compatibility Tool
DPC++ Source Code

Status of Amber SYCL port

SYCL code is based on Amber 20

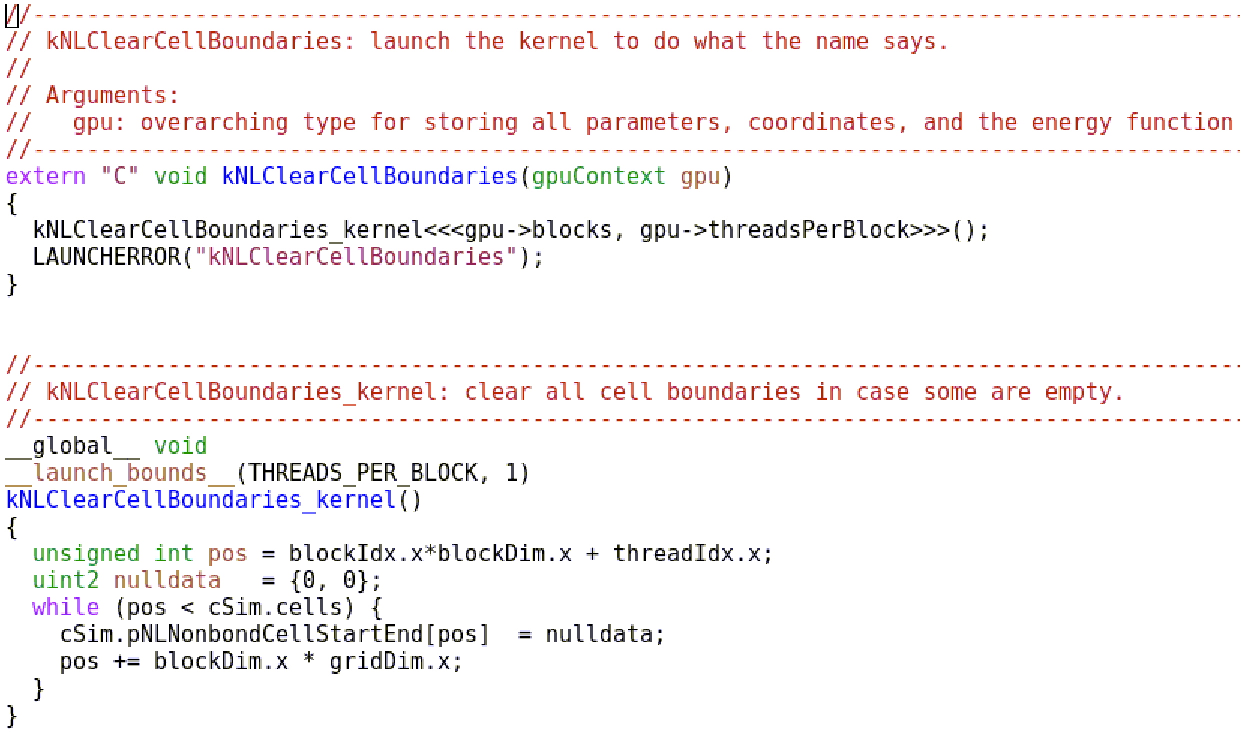
- Project started in 2021
- Work with stable code base
- Initial goal: Get basic MD working with SYCL on Intel Data Center GPUs
- Later: Feature completeness with future Amber releases

Amber 20 port and validation

- Ported CUDA header & source files (~60K CUDA source lines) using Intel DPC++ Compatibility tool
- Added kernel by kernel CUDA to SYCL verification framework
- Supports regular PME MD simulations (thermostats, barostats, restraints)
- Verified PME NVE correctness with JAC, FactorIX, Cellulose, STMV on Intel discrete GPU Ponte Vecchio
- Tests from Amber test suite
- Works on Intel Max Series Datacenter GPUs

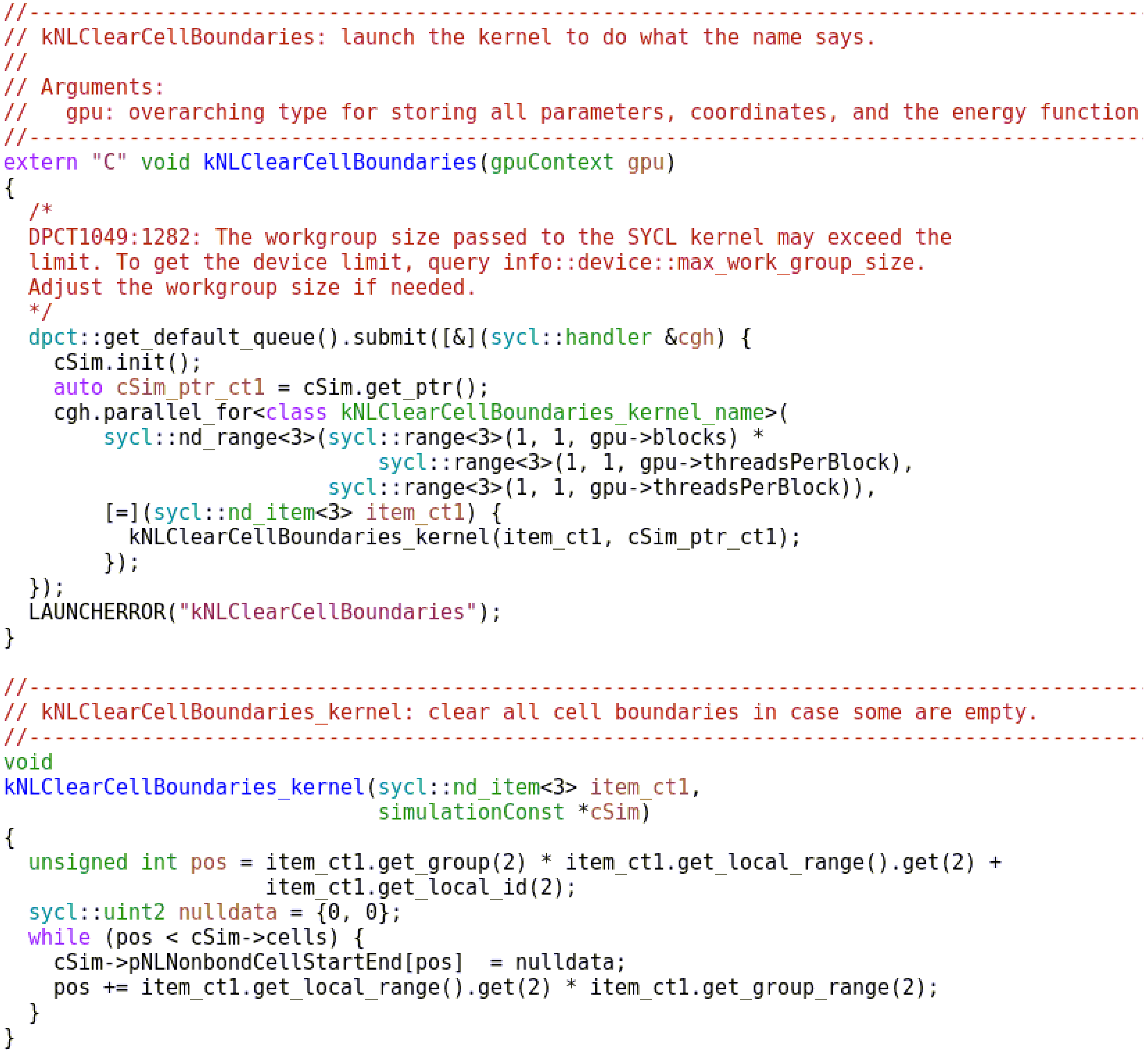
Amber CUDA vs SYCL code

CUDA and SYCL kernels are quite similar



```
//---------------------------------------------------------------------------------------------
// kNLClearCellBoundaries: launch the kernel to do what the name says.
//
// Arguments:
//   gpu: overarching type for storing all parameters, coordinates, and the energy function
//---------------------------------------------------------------------------------------------
extern "C" void kNLClearCellBoundaries(gpuContext gpu)
{
  kNLClearCellBoundaries_kernel<<<gpu->blocks, gpu->threadsPerBlock>>>();
  LAUNCHERROR("kNLClearCellBoundaries");
}


//---------------------------------------------------------------------------------------------
// kNLClearCellBoundaries_kernel: clear all cell boundaries in case some are empty.
//---------------------------------------------------------------------------------------------
__global__ void
__launch_bounds__(THREADS_PER_BLOCK, 1)
kNLClearCellBoundaries_kernel()
{
  unsigned int pos = blockIdx.x*blockDim.x + threadIdx.x;
  uint2 nulldata   = {0, 0};
  while (pos < cSim.cells) {
    cSim.pNLNonbondCellStartEnd[pos]  = nulldata;
    pos += blockDim.x * gridDim.x;
  }
}
```

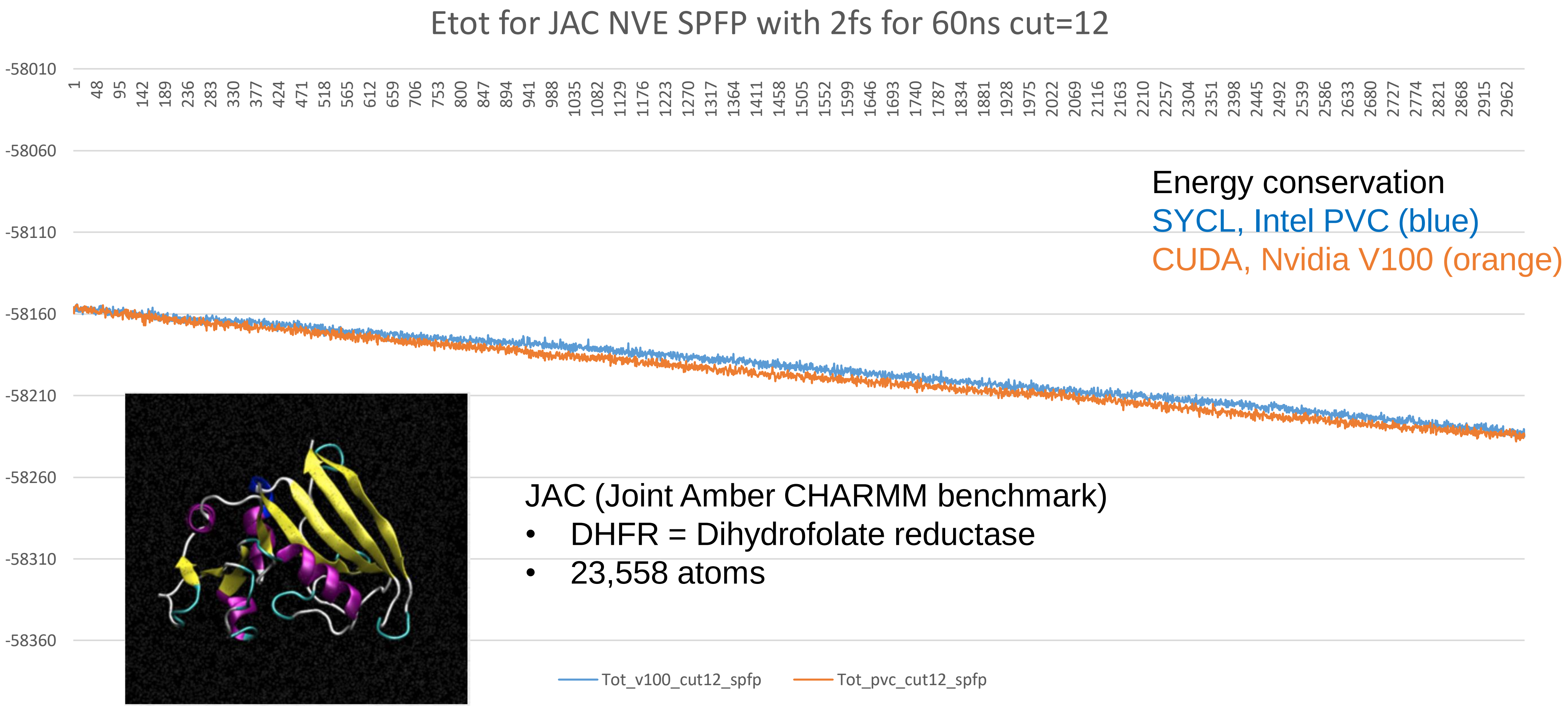


```
//---------------------------------------------------------------------------------------------
// kNLClearCellBoundaries: launch the kernel to do what the name says.
//
// Arguments:
//   gpu: overarching type for storing all parameters, coordinates, and the energy function
//---------------------------------------------------------------------------------------------
extern "C" void kNLClearCellBoundaries(gpuContext gpu)
{
  /*
  DPCT1049:1282: The workgroup size passed to the SYCL kernel may exceed the
  limit. To get the device limit, query info::device::max_work_group_size.
  Adjust the workgroup size if needed.
  */
  dpct::get_default_queue().submit([&](sycl::handler &cgh) {
    cSim.init();
    auto cSim_ptr_ct1 = cSim.get_ptr();
    cgh.parallel_for<class kNLClearCellBoundaries_kernel_name>(
        sycl::nd_range<3>(sycl::range<3>(1, 1, gpu->blocks) *
                              sycl::range<3>(1, 1, gpu->threadsPerBlock),
                          sycl::range<3>(1, 1, gpu->threadsPerBlock)),
        [=](sycl::nd_item<3> item_ct1) {
          kNLClearCellBoundaries_kernel(item_ct1, cSim_ptr_ct1);
        });
  });
  LAUNCHERROR("kNLClearCellBoundaries");
}

//---------------------------------------------------------------------------------------------
// kNLClearCellBoundaries_kernel: clear all cell boundaries in case some are empty.
//---------------------------------------------------------------------------------------------
void
kNLClearCellBoundaries_kernel(sycl::nd_item<3> item_ct1,
                              simulationConst *cSim)
{
  unsigned int pos = item_ct1.get_group(2) * item_ct1.get_local_range().get(2) +
                     item_ct1.get_local_id(2);
  sycl::uint2 nulldata = {0, 0};
  while (pos < cSim->cells) {
    cSim->pNLNonbondCellStartEnd[pos]  = nulldata;
    pos += item_ct1.get_local_range().get(2) * item_ct1.get_group_range(2);
  }
}
```

Amber SYCL code is numerically accurate

Numerically correct SYCL implementation of most important regular PME MD, verified on Intel PVC



Energy conservation
SYCL, Intel PVC (blue)
CUDA, Nvidia V100 (orange)

JAC (Joint Amber CHARMM benchmark)

- DHFR = Dihydrofolate reductase
- 23,558 atoms

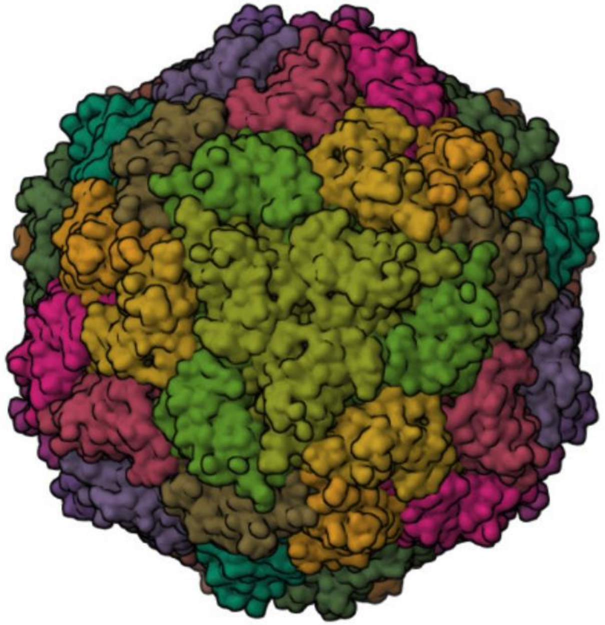
Performance for STMV NVE 4fs benchmark

Benchmark setup

- Intel GPU Max 1550 with 2 tiles and 1024 Eus
- Each tile exposed as device
- Amber benchmark STMV NVE 4fs
  (Satellite Tobacco Moasic Virus in water, about 1M atoms)

Performance optimizations

- Guided by Intel oneProf, oneTrace, and Vtune

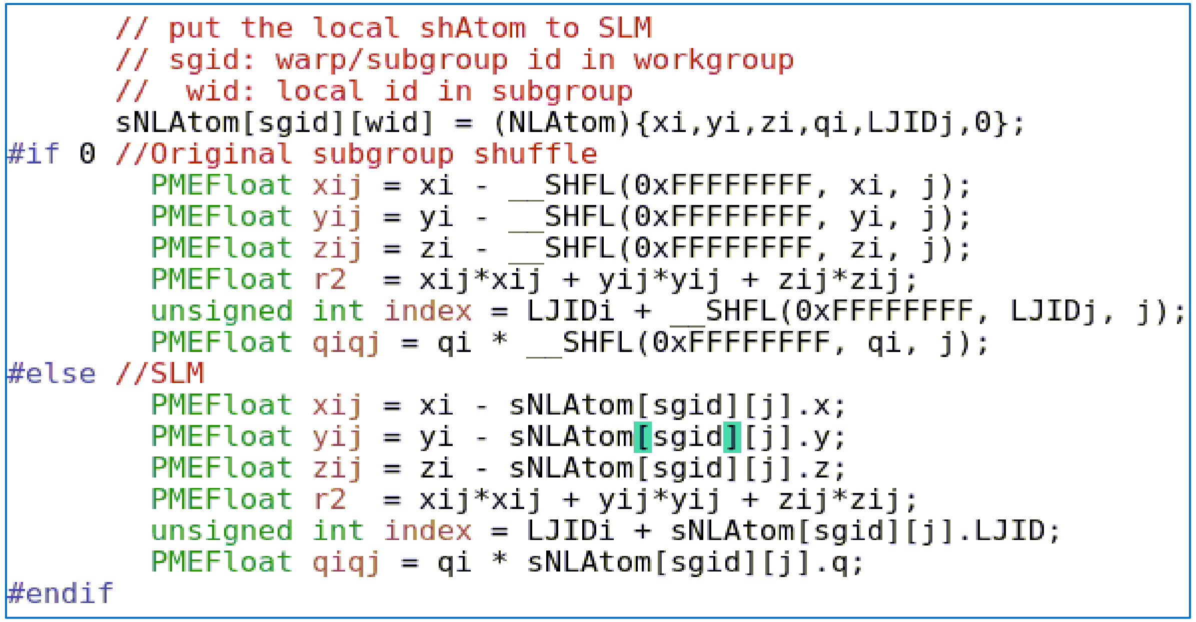


STMV image
(RCSB PDB)

Key optimizations of nonbonded force kernels

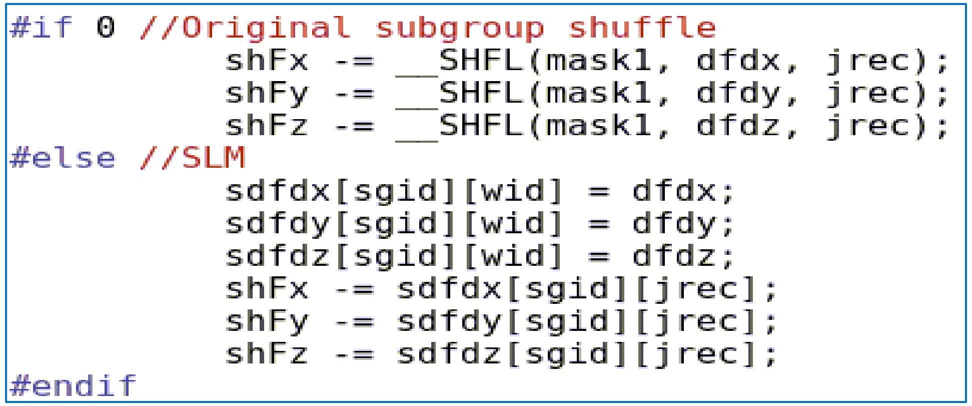
Replace subgroup shuffle with shared local memory (SLM)

- Subgroup shuffle with variable lane is (currently) inefficient
- Introduce SLM to replace all relevant shuffles with nonuniform lanes
- Significant stalls are reduced, especially pipeline stalls



```
      // put the local shAtom to SLM
      // sgid: warp/subgroup id in workgroup
      //  wid: local id in subgroup
      sNLAtom[sgid][wid] = (NLAtom){xi,yi,zi,qi,LJIDj,0};
#if 0 //Original subgroup shuffle
        PMEFloat xij = xi - __SHFL(0xFFFFFFFF, xi, j);
        PMEFloat yij = yi - __SHFL(0xFFFFFFFF, yi, j);
        PMEFloat zij = zi - __SHFL(0xFFFFFFFF, zi, j);
        PMEFloat r2  = xij*xij + yij*yij + zij*zij;
        unsigned int index = LJIDi + __SHFL(0xFFFFFFFF, LJIDj, j);
        PMEFloat qiqj = qi * __SHFL(0xFFFFFFFF, qi, j);
#else //SLM
        PMEFloat xij = xi - sNLAtom[sgid][j].x;
        PMEFloat yij = yi - sNLAtom[sgid][j].y;
        PMEFloat zij = zi - sNLAtom[sgid][j].z;
        PMEFloat r2  = xij*xij + yij*yij + zij*zij;
        unsigned int index = LJIDi + sNLAtom[sgid][j].LJID;
        PMEFloat qiqj = qi * sNLAtom[sgid][j].q;
#endif
```

Force computation (atom properties like coordinates x, y, z, charge q in registers / private memory of work items)



```
#if 0 //Original subgroup shuffle
            shFx -= __SHFL(mask1, dfdx, jrec);
            shFy -= __SHFL(mask1, dfdy, jrec);
            shFz -= __SHFL(mask1, dfdz, jrec);
#else //SLM
            sdfdx[sgid][wid] = dfdx;
            sdfdy[sgid][wid] = dfdy;
            sdfdz[sgid][wid] = dfdz;
            shFx -= sdfdx[sgid][jrec];
            shFy -= sdfdy[sgid][jrec];
            shFz -= sdfdz[sgid][jrec];
#endif
```

Pairwise force accumulation

Key optimizations of nonbonded force kernels

Fix ND range

- Was fixed to 800, increase to (10 * max_eu_number)
- Increases XVE occupancy from 77% to 98%

Memory space casting in global atomic operations

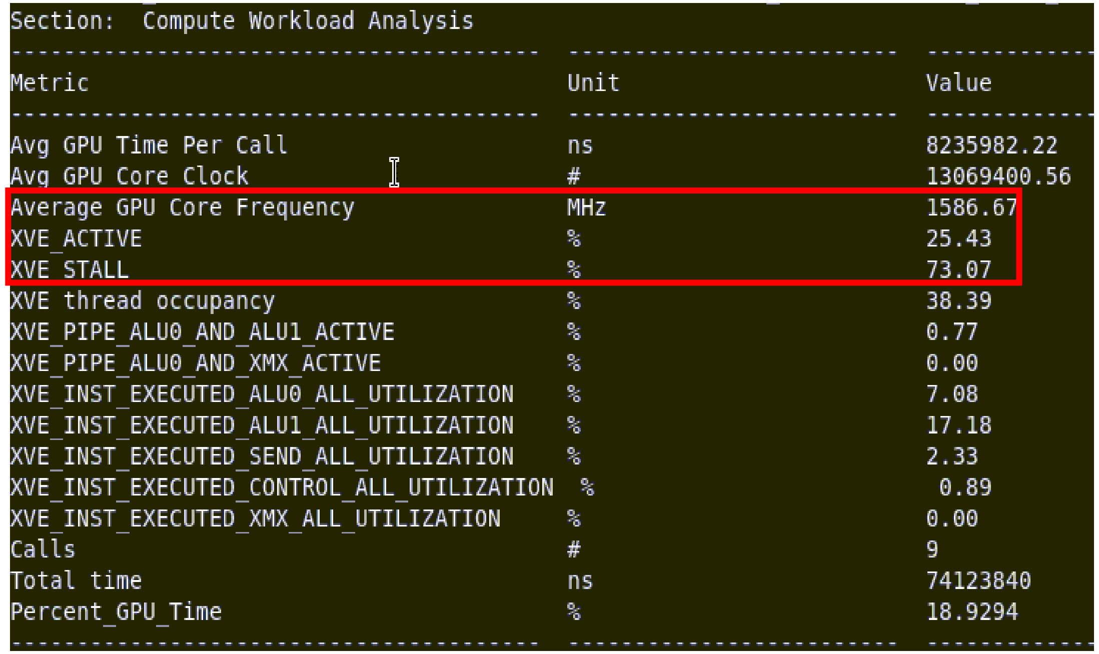
- Pointer alias outside the kernel to simplify the pointer address in the atomic operator
- Original member pointer in generic address space resulted in dynamic address space checking and casting

Use faster esimd based radix sort implementation

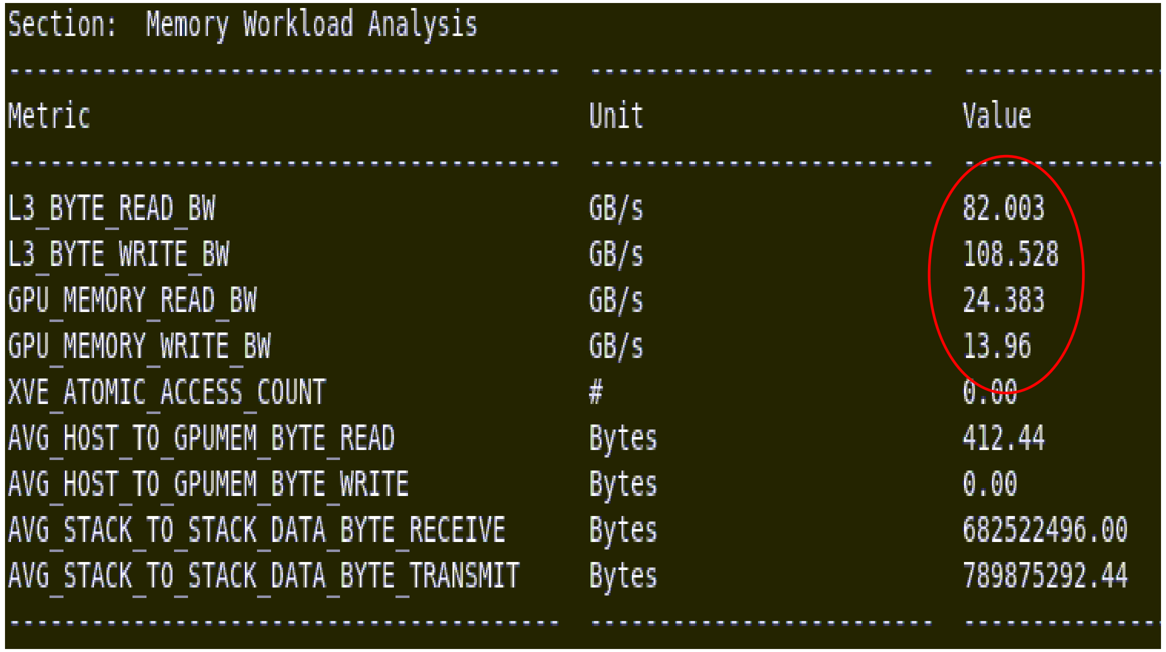
- oneDPL stable_sort => oneDPL Radix_sort_by_key (esimd based)

Nonbonded force kernel profiling data

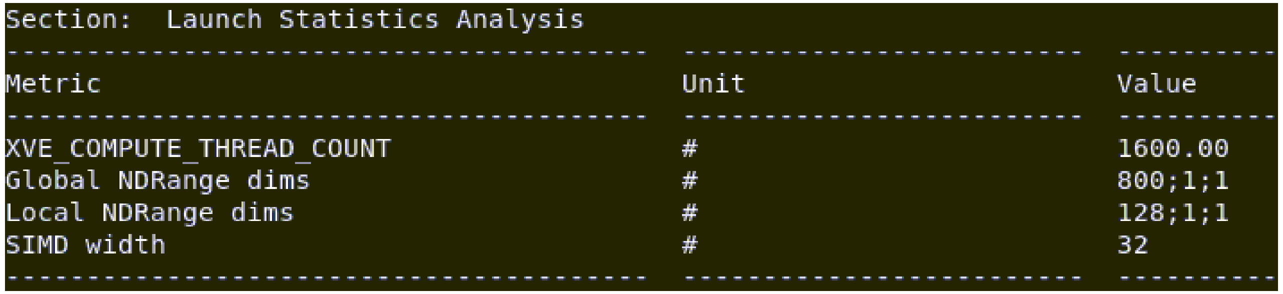
Oneprof: Wrong ND range, low XVE activity / thread occupancy



```
Section:  Compute Workload Analysis
----------------------------------------  -------------------------  -----------
Metric                                    Unit                       Value
----------------------------------------  -------------------------  -----------
Avg GPU Time Per Call                     ns                         8235982.22
Avg GPU Core Clock                        #                          13069400.56
Average GPU Core Frequency                MHz                        1586.67
XVE_ACTIVE                                %                          25.43
XVE_STALL                                 %                          73.07
XVE thread occupancy                      %                          38.39
XVE_PIPE_ALU0_AND_ALU1_ACTIVE             %                          0.77
XVE_PIPE_ALU0_AND_XMX_ACTIVE              %                          0.00
XVE_INST_EXECUTED_ALU0_ALL_UTILIZATION    %                          7.08
XVE_INST_EXECUTED_ALU1_ALL_UTILIZATION    %                          17.18
XVE_INST_EXECUTED_SEND_ALL_UTILIZATION    %                          2.33
XVE_INST_EXECUTED_CONTROL_ALL_UTILIZATION  %                          0.89
XVE_INST_EXECUTED_XMX_ALL_UTILIZATION     %                          0.00
Calls                                     #                          9
Total time                                ns                         74123840
Percent_GPU_Time                          %                          18.9294
----------------------------------------  -------------------------  -----------
```



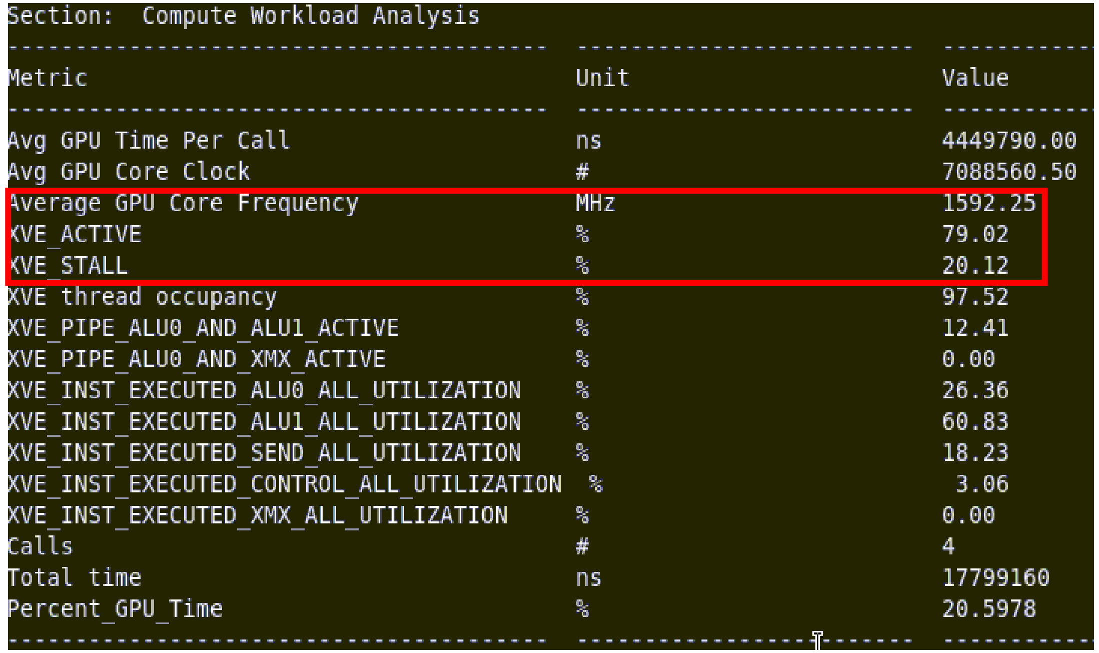
```
Section:  Memory Workload Analysis
----------------------------------------  -------------------------  -----------
Metric                                    Unit                       Value
----------------------------------------  -------------------------  -----------
L3_BYTE_READ_BW                           GB/s                       82.003
L3_BYTE_WRITE_BW                          GB/s                       108.528
GPU_MEMORY_READ_BW                        GB/s                       24.383
GPU_MEMORY_WRITE_BW                       GB/s                       13.96
XVE_ATOMIC_ACCESS_COUNT                   #                          0.00
AVG_HOST_TO_GPUMEM_BYTE_READ              Bytes                      412.44
AVG_HOST_TO_GPUMEM_BYTE_WRITE             Bytes                      0.00
AVG_STACK_TO_STACK_DATA_BYTE_RECEIVE      Bytes                      682522496.00
AVG_STACK_TO_STACK_DATA_BYTE_TRANSMIT     Bytes                      789875292.44
----------------------------------------  -------------------------  -----------
```



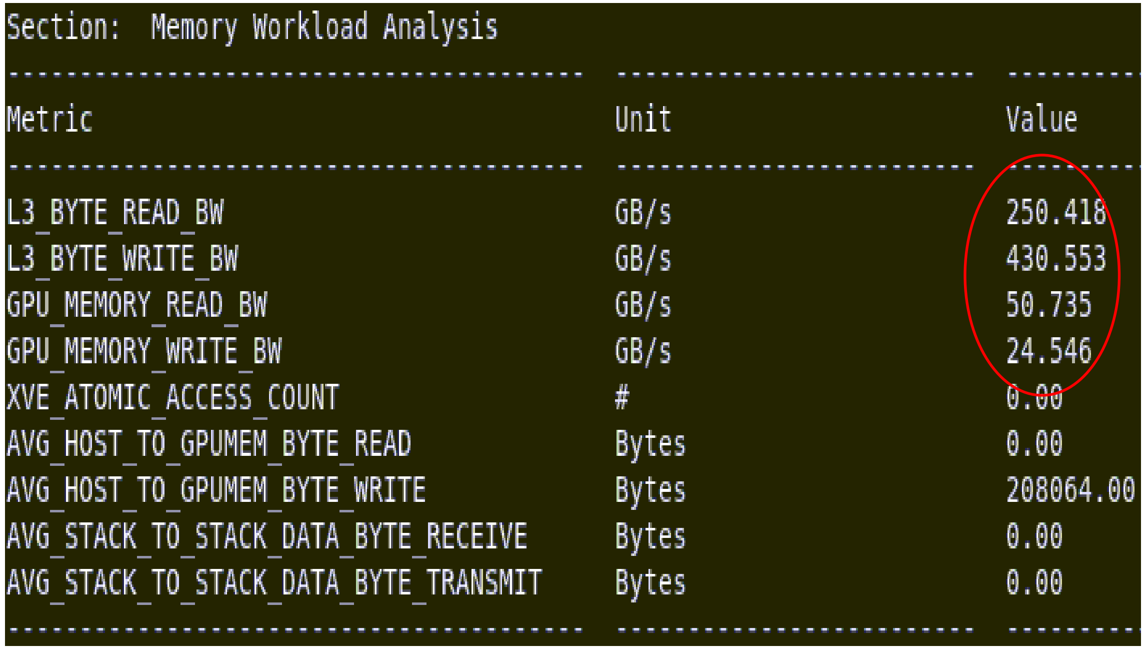
```
Section:  Launch Statistics Analysis
----------------------------------------  -------------------------  ----------
Metric                                    Unit                       Value
----------------------------------------  -------------------------  ----------
XVE_COMPUTE_THREAD_COUNT                  #                          1600.00
Global NDRange dims                       #                          800;1;1
Local NDRange dims                        #                          128;1;1
SIMD width                                #                          32
----------------------------------------  -------------------------  ----------
```

Nonbonded force kernel profiling data

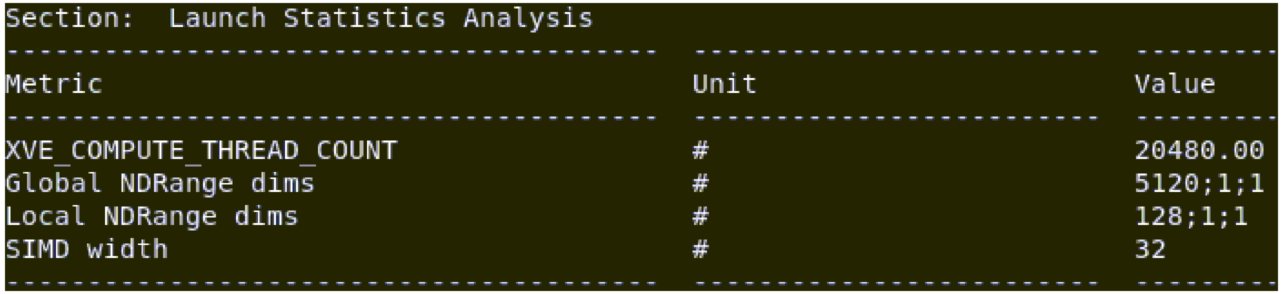
Oneprof: Optimized ND range + other optimizations



```
Section:  Compute Workload Analysis
----------------------------------------  -------------------------  ----------
Metric                                    Unit                       Value
----------------------------------------  -------------------------  ----------
Avg GPU Time Per Call                     ns                         4449790.00
Avg GPU Core Clock                        #                          7088560.50
Average GPU Core Frequency                MHz                        1592.25
XVE_ACTIVE                                %                          79.02
XVE_STALL                                 %                          20.12
XVE thread occupancy                      %                          97.52
XVE_PIPE_ALU0_AND_ALU1_ACTIVE             %                          12.41
XVE_PIPE_ALU0_AND_XMX_ACTIVE              %                          0.00
XVE_INST_EXECUTED_ALU0_ALL_UTILIZATION    %                          26.36
XVE_INST_EXECUTED_ALU1_ALL_UTILIZATION    %                          60.83
XVE_INST_EXECUTED_SEND_ALL_UTILIZATION    %                          18.23
XVE_INST_EXECUTED_CONTROL_ALL_UTILIZATION  %                          3.06
XVE_INST_EXECUTED_XMX_ALL_UTILIZATION     %                          0.00
Calls                                     #                          4
Total time                                ns                         17799160
Percent_GPU_Time                          %                          20.5978
----------------------------------------  -------------------------  ----------
```

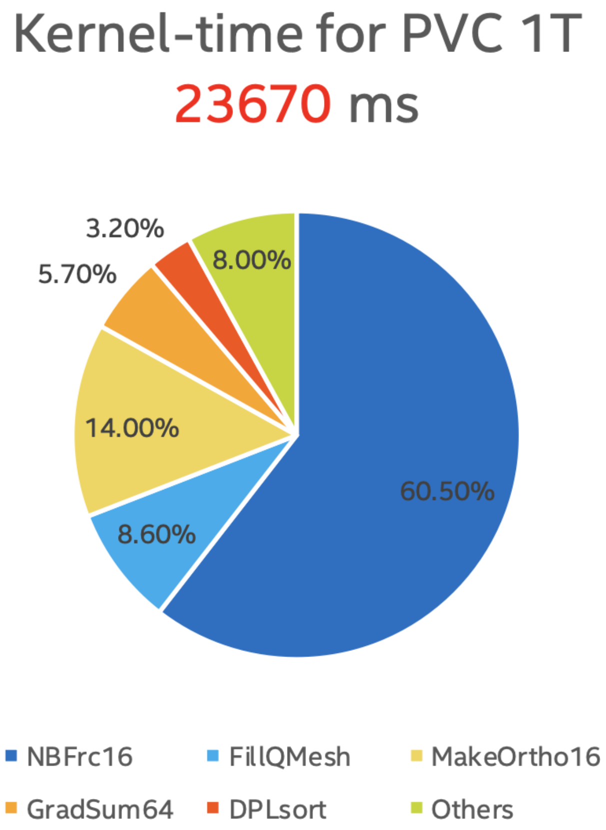


```
Section:  Memory Workload Analysis
----------------------------------------  -------------------------  ---------
Metric                                    Unit                       Value
----------------------------------------  -------------------------  ---------
L3_BYTE_READ_BW                           GB/s                       250.418
L3_BYTE_WRITE_BW                          GB/s                       430.553
GPU_MEMORY_READ_BW                        GB/s                       50.735
GPU_MEMORY_WRITE_BW                       GB/s                       24.546
XVE_ATOMIC_ACCESS_COUNT                   #                          0.00
AVG_HOST_TO_GPUMEM_BYTE_READ              Bytes                      0.00
AVG_HOST_TO_GPUMEM_BYTE_WRITE             Bytes                      208064.00
AVG_STACK_TO_STACK_DATA_BYTE_RECEIVE      Bytes                      0.00
AVG_STACK_TO_STACK_DATA_BYTE_TRANSMIT     Bytes                      0.00
----------------------------------------  -------------------------  ---------
```



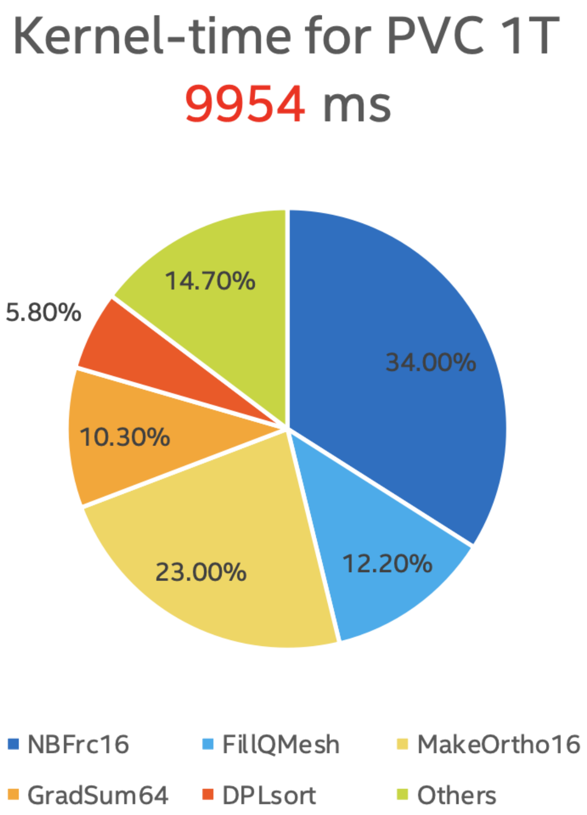
```
Section:  Launch Statistics Analysis
----------------------------------------  -------------------------  ---------
Metric                                    Unit                       Value
----------------------------------------  -------------------------  ---------
XVE_COMPUTE_THREAD_COUNT                  #                          20480.00
Global NDRange dims                       #                          5120;1;1
Local NDRange dims                        #                          128;1;1
SIMD width                                #                          32
----------------------------------------  -------------------------  ---------
```

Top hot spots before – after optimizations



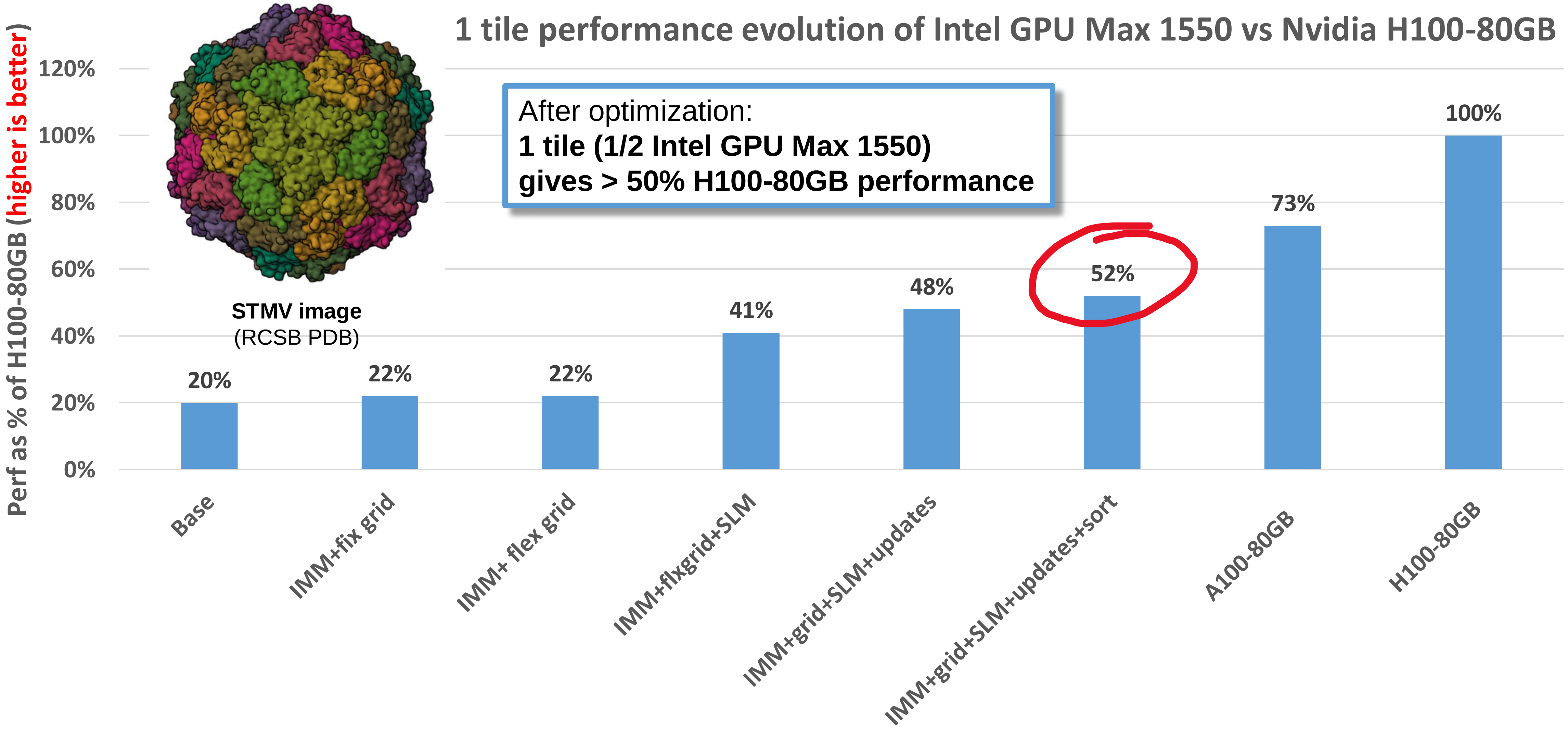
Kernel-time for PVC 1T
23670 ms

	H100 (ms) CUDA	PVC 1T (ms) Base	PVC 1T (ms) Opt
NBFrc16	1.95	14.33	4.45
MakeOrtho16	1.90	6.64	3.58
FillQMesh	0.32	2.05	0.70
ShakeHMR	0.26	0.19	0.22
DPL Sort		0.08	0.08
GradSum64	0.12	1.34	0.45



Kernel-time for PVC 1T
9954 ms

Performance evolution for STMV NVE 4fs benchmark

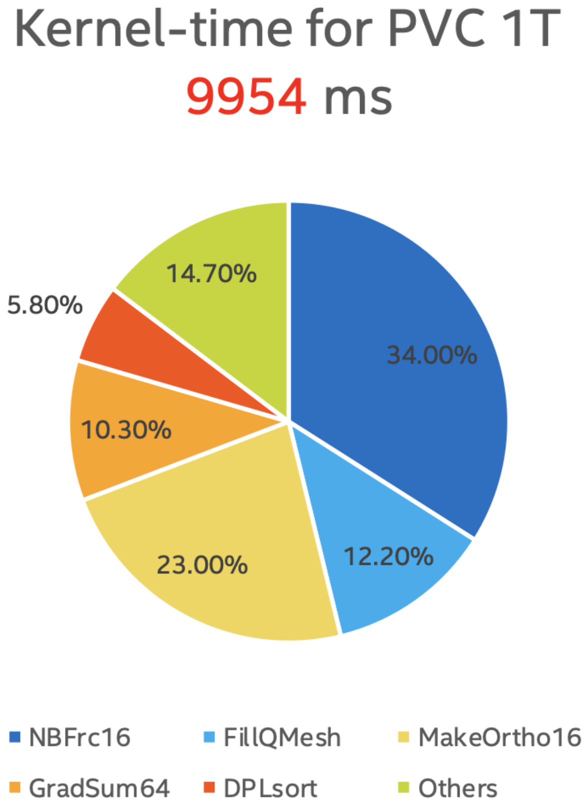


1 tile performance evolution of Intel GPU Max 1550 vs Nvidia H100-80GB

After optimization:
1 tile (1/2 Intel GPU Max 1550)
gives > 50% H100-80GB performance

STMV image
(RCSB PDB)

Hot spots – scaling to two PVC tiles



	H100 (ms) CUDA	PVC 1T (ms) Base	PVC 1T (ms) Opt	PVC 2T (ms)
NBFrc16	1.95	14.33	4.45	3.59
MakeOrtho16	1.90	6.64	3.58	2.30
FillQMesh	0.32	2.05	0.70	2.10
ShakeHMR	0.26	0.19	0.22	1.07
DPL Sort		0.08	0.08	0.08
GradSum64	0.12	1.34	0.45	0.23

How can we scale to two Intel PVC tiles?

Observations for FillQMesh kernel

- Very bad scaling 1T –> 2T
- Stalls appear to be related to inter-tile traffic
- Too many global atomics

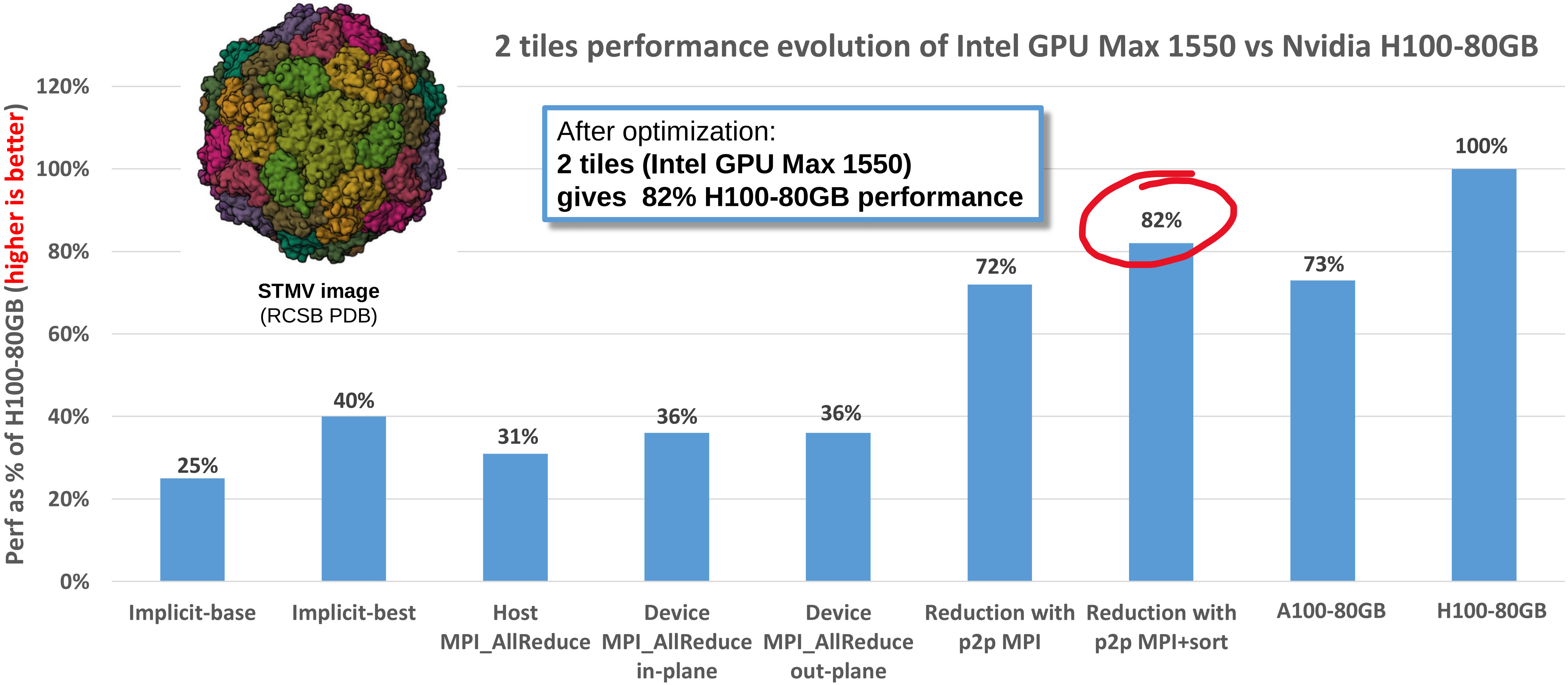
How can we use two tiles?

- Implicit scaling (treat PVC card as one device)
- Explicit scaling (treat two tiles as separate devices and parallelize with MPI)

Final optimizations for two PVC tiles

- Use MPI based explicit scaling (treat each tile as separate device)
- Optimize force reduction across 2 tiles
- Use point to point communication, reduce data on 1 tile

Performance evolution for STMV NVE 4fs benchmark

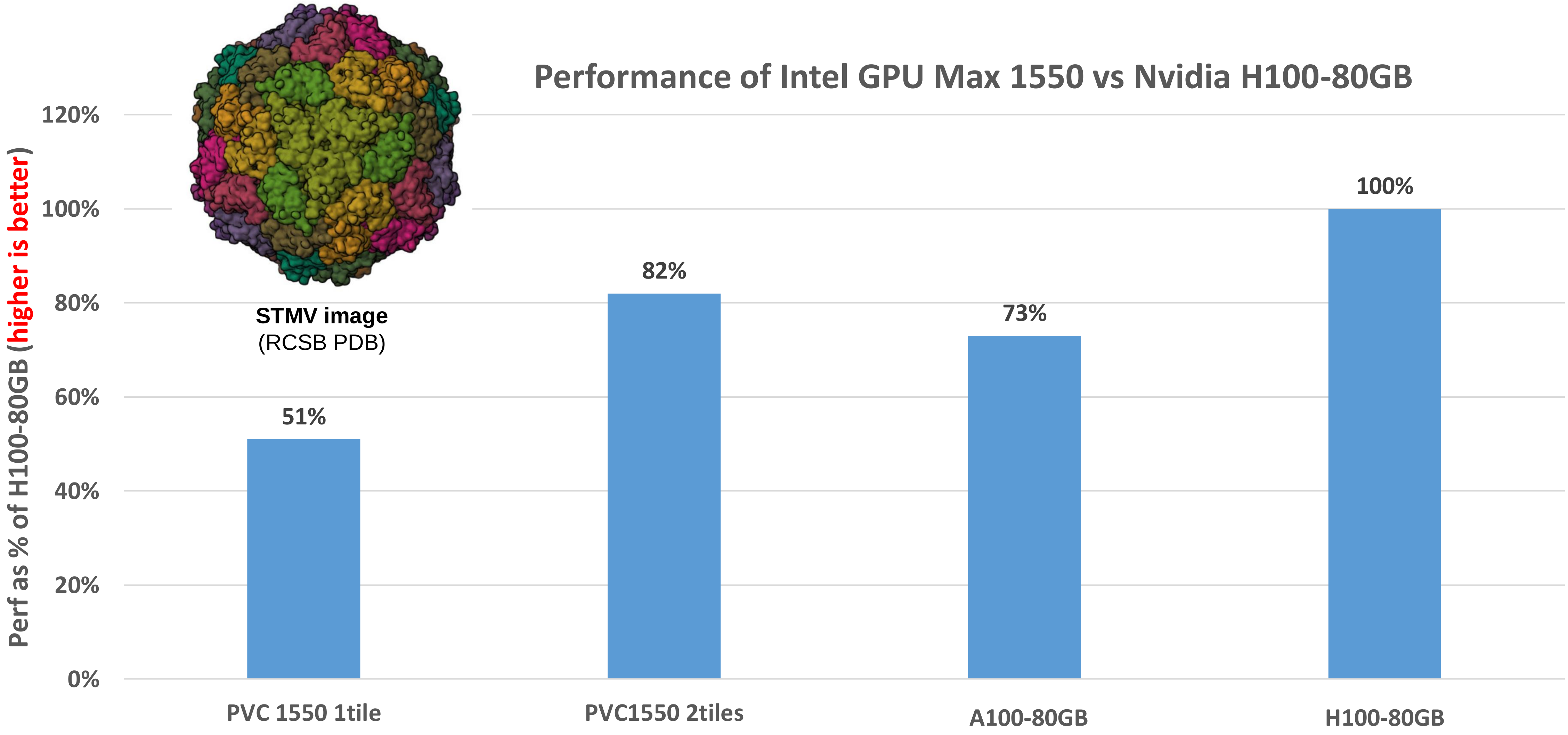


2 tiles performance evolution of Intel GPU Max 1550 vs Nvidia H100-80GB

STMV image
(RCSB PDB)

After optimization:
2 tiles (Intel GPU Max 1550)
gives 82% H100-80GB performance

Performance summary for STMV NVE 4fs benchmark



Performance of Intel GPU Max 1550 vs Nvidia H100-80GB

STMV image
(RCSB PDB)

Amber SYCL Future

Catch up to Amber 24 (almost complete)

- Initial SYCL port was based on Amber 20
- Many changes to CUDA code between Amber 20 and Amber 24
- Started from scratch with Amber 24 – this was easier than porting changes

Feature completeness

- Enable support for missing features / code paths (GB, REMD, TI, etc)
- Most code is ported but needs testing, profiling, optimization

Portability and performance portability

- Intel gaming GPUs (does not work out of the box)
- Nvida GPUs via CUDA backend
- AMD GPUs via HIP/ROCm backend
- Performance optimizations

Amber SYCL Future

Address SYCL code sustainability

- Cannot afford two code bases (CUDA/HIP and SYCL)
- Need to convince developers to switch to SYCL

Who are the (future) developers

- Domain scientists, not software engineers
- These developers want to solve scientific problems, with as little effort and as quickly as possible
- Current generation is familiar with CUDA
- It is hard to convince developers to face the learning curve to adopt new/different technologies
- Need good arguments to convince switching from CUDA

Requirements for adoption of Amber SYCL port

- Feature completeness
- Performance on Nvidia (AMD) GPUs must be similar or better than CUDA (HIP) code

Summary – Powering Amber on GPUs with SYCL

Amber is a powerful and widely used biomolecular simulations software.

Intel oneAPI was used to port the Amber high-performance molecular dynamics engine from CUDA to SYCL and optimize the SYCL code.

The Intel oneAPI powered SYCL port is numerically stable and shows strong performance on Intel Datacenter GPU Max hardware.

Future SYCL work will focus on

- **Porting of advanced simulation techniques**
- **Performance and portability on Intel consumer hardware and Nvidia and AMD GPUs**

Thank you for your attention.