Experiences Porting NAMD to the Data Parallel C++ Programming Model

David J. Hardy
University of Illinois at Urbana–Champaign
Beckman Institute for Advanced Science and Technology
Urbana, Illinois, USA
dhardy@illinois.edu

Jaemin Choi
University of Illinois at Urbana–Champaign
Department of Computer Science
Urbana, Illinois, USA
jchoi157@illinois.edu

Wei Jiang
Argonne National Laboratory
Argonne Leadership Computing Facility
Lemont, Illinois, USA
wjiang@alcf.anl.gov

Emad Tajkhorshid
University of Illinois at Urbana–Champaign
Beckman Institute for Advanced Science and Technology
Urbana, Illinois, USA
emad@illinois.edu

ABSTRACT

HPC applications have a growing need to leverage heterogeneous computing resources with a vendor-neutral programming paradigm. Data Parallel C++ is a programming language based on open standards SYCL, providing a vendor-neutral solution. We describe our experiences porting the NAMD molecular dynamics application with its GPU-offload force kernels to SYCL/DPC++. Results are shown that demonstrate correctness of the porting effort.

CCS CONCEPTS

• **Computing methodologies** → **Parallel programming languages**; • **Software and its engineering** → *Software notations and tools.*

KEYWORDS

Molecular Dynamics, NAMD, SYCL, DPC++, oneAPI

ACM Reference Format:
David J. Hardy, Jaemin Choi, Wei Jiang, and Emad Tajkhorshid. 2022. Experiences Porting NAMD to the Data Parallel C++ Programming Model. In *International Workshop on OpenCL (IWOCL'22), May 10–12, 2022, Bristol, United Kingdom, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3529538.3529560

1 INTRODUCTION

Now more than ever before, it is imperative for HPC software applications to be able to leverage heterogeneous computing resources through vendor-agnostic languages and libraries. The Data Parallel C++ (DPC++) programming model has been developed as a part of Intel's oneAPI to create a common set of languages and libraries to support heterogeneous computing resources from all vendors. DPC++ is a superset of the SYCL open standard, providing some additional language constructs that improve hardware support, many of which have been adopted into the SYCL 2020 standard.

In this paper, we present our experiences with porting the NAMD molecular dynamics application to SYCL, accessed through the oneAPI/DPC++ compiler implementation. NAMD is an application widely used by the biomedical research community. It is capable of scaling very large biomolecular simulations across the largest supercomputers. An upcoming challenge for NAMD is supporting, not just computers utilizing NVIDIA GPUs, but now also the two upcoming exascale class supercomputers, the first at Oak Ridge National Laboratory (ORNL) named Frontier based on AMD GPUs and the second at Argonne National Laboratory (ANL) named Aurora based on Intel GPUs. The porting of NAMD to SYCL/DPC++ is critical to support its running on ANL Aurora. Moreover, the adoption of a standards-based language for heterogeneous programming will hopefully improve NAMD's hardware support while also extending the code's longevity.

2 BACKGROUND

Molecular dynamics (MD) simulation software and HPC resources together provide access to spatial and temporal scales inaccessible to experimental imaging methods alone. Tools like NAMD produce atomic detail views of structure and dynamics that can reveal the molecular basis for disease. By studying viruses and other diseases with MD and related methods, biomedical researchers can inform the development of new treatments and therapies.

NAMD is a well known molecular dynamics software application capable of scaling large biomolecular simulations across CPU- and GPU-based computer architectures [6]. For many years, NAMD has been an important simulation tool for the biomedical research community, and the pandemic has focused a great deal of research effort to help combat COVID-19, leading to several notable high-profile investigations of the SARS-CoV-2 virus using NAMD [4]. The Amaro Lab at UCSD led a large team of researchers in 2020 to do groundbreaking all-atom modeling and simulation of the spike protein assembly and also the full virion [1]. The team included the UIUC developers of NAMD to assist scaling simulations across the Frontera supercomputer at the Texas Advanced Computing Center at UT Austin and the Summit supercomputer at ORNL. The

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
IWOCL'22, May 10–12, 2022, Bristol, United Kingdom, United Kingdom
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9658-5/22/05…$15.00
https://doi.org/10.1145/3529538.3529560

investigation was covered by the New York Times and awarded at SC20 the first ACM Gordon Bell Special Prize for High Performance Computing-Based COVID-19 Research. NAMD was also an integral part of two more Gordon Bell Special Prize finalists at SC21, which included the Amaro Lab extending the previous year's results to simulate the first ever all-atom modeling of the SARS-CoV-2 virion suspended in an aerosol droplet [2] and also the Ramanathan Lab at ANL leading the study of the SARS-CoV-2 replication–transcription complex responsible for replicating and transcribing the viral mRNA inside a human cell [9]. Studies within the Tajkhorshid Lab at UIUC have examined the part of the SARS-CoV-2 spike protein called the fusion peptide, and how three fusion peptide monomers can form a complex which then interacts with the host cell membrane to infect it [3].

The development of NAMD dates back to the late '90s, an object-oriented C++ code using Charm++ parallel objects for message-driven execution [5]. NAMD was among the earliest adopters of CUDA in 2007 [8], and the CUDA code in NAMD has evolved over the years with the advancement of GPU capability. GPU support in NAMD began by using the GPU-offload paradigm, which at first involved just calculation of the short-range non-bonded forces that dominate the computational work required per time step. As GPUs became faster and more capable, more parts of the force calculation were offloaded to the GPU, starting with the scalable parts of the particle–mesh Ewald (PME) algorithm for long-range electrostatics and eventually including the bonded force terms and non-bonded exclusions. About four years ago in early 2018, it became evident that the GPU-offload approach was no longer able to keep a modern GPU fully engaged. Profiling showed that GPUs had become fast enough that the CPU work was no longer simply overlapping the GPU work and had instead become a performance bottleneck. A GPU-resident version of NAMD was then developed for fast single-GPU simulation, more than doubling performance by moving the remaining CPU integration and rigid bond constraint calculations to the GPU and maintaining the data on the GPU across time steps [6]. The GPU-resident version has over the past year-and-a-half been extended to scale a single simulation across tightly coupled (e.g., NVLinked) GPUs on the same node with over 70% efficiency.

3 DESIGN AND IMPLEMENTATION

3.1 Design considerations

The overall goal of porting NAMD to DPC++ has been to support the upcoming ANL Aurora supercomputer, which will be based on Intel Xe-HPC (Ponte Vecchio) GPU hardware and is to be among the first true exascale-capable machines. Implementing with DPC++ has the added advantage of enabling support for the full range of Intel's emerging GPU hardware while potentially improving support for AVX-512 enabled CPUs. From a language perspective, the choice of porting NAMD to SYCL/DPC++, rather than porting CUDA kernels to OpenCL, provides the opportunity to update NAMD's use of C++ to modern standards, while reorganizing key internal algorithms and data structures to better leverage vector processing.

Since NAMD already has years of development invested in CUDA, with good performance and scaling on NVIDIA GPUs, it has been important from a design perspective to complement and reuse these aspects without harming NAMD's existing GPU support. These considerations guided the following design decisions:

(1) Develop SYCL/DPC++ support in a way that extends NAMD without disrupting its current GPU support.
(2) Leverage the existing GPU kernels and data structures, rather than attempting to rewrite them from scratch.
(3) Add support incrementally, guided by Amdahl's Law, to accelerate the most computationally expensive parts of the code first.

The first of these considerations has been accomplished by using preprocessor switches to isolate the DPC++ extensions from the existing GPU code. The second has been accomplished by translating CUDA kernels to DPC++ and copying the supporting data structures and kernel management infrastructure into DPC++ versions. The third has been accomplished by the order in which SYCL/DPC++ support was developed, beginning with the short-range non-bonded force kernels and continuing with kernels for PME forces and bonded forces, similar to the order in which CUDA support evolved in NAMD.

3.2 Implementation details

The plan is to eventually develop SYCL/DPC++ support for all aspects of NAMD GPU acceleration, including the new GPU-resident code paths. This will be especially important for the planned science applications of NAMD on Aurora, which will make use of multiple replica simulation methodology to run a single simulation per GPU, and perhaps scaling a larger simulation across all of the tightly coupled GPUs in a single node. However, it was important to develop from a stable code base, so the decision was made to start by porting the GPU-offload force kernels from NAMD 2.14. Note that these kernels are still essential for the GPU-resident code path, so this work is a necessary prerequisite for full support. NAMD 2.14 itself has a large amount of CUDA kernel and management code, as shown in Table 1, which shows statistics on the number of files, number of kernels, and number of lines of code, listed by the different types of forces.

Component	# C/h files	# cu files	# kernels	# lines
Non-bonded forces	6	2	20	5.8 K
Bonded forces	3	1	2	3.9 K
PME – single node	6	1	5	4.1 K
PME – scalable	6	1	3	3.3 K
Utilities	8	1	1	1.7 K
Total	29	6	31	18.8 K

Table 1: Statistics on the amount of CUDA kernel and supporting code contained in NAMD 2.14 for each force type.

A divide-and-conquer strategy was employed using preprocessor switches to decouple components in the CUDA code and the emerging SYCL/DPC++ code, which significantly reduced development and debugging complexity. As Table 1 reveals, the force components were separated into non-bonded force and device utilities, bonded force, and particle–mesh Ewald (PME) long-range electrostatics. The PME component itself has two different implementations, one supporting single-node computation, with all parts of the algorithm

implemented on a single GPU, and the other supporting multi-node parallelism, with GPU support for just the scalable parts of the algorithm.

The porting of the individual CUDA kernels and surrounding `.cu` files was accelerated by utilizing the Intel DPC++ Compatibility Tool provided with oneAPI, which saved at least 80% of the code porting effort. The compatibility tool saves much time by translating many common CUDA idioms, such as converting the expression

```
threadIdx.x
```

into its comparable SYCL/DPC++ expression

```
ndItem.get_local_id(2)
```

which also provides developers new to SYCL/DPC++ with good instruction for learning its syntax.

Several syntactic differences with CUDA are due to the use of modern C++ in SYCL/DPC++. Key examples include the use of unnamed lambda expressions to implement SYCL/DPC++ kernels and the need for error handling with try–catch blocks. Other differences with CUDA are due to design decisions in SYCL and OpenCL. For example, the SYCL work queue is analogous to the CUDA stream, except that a CUDA stream processes commands in order, whereas a SYCL work queue is by default unordered and must be explicitly initialized to support ordered processing. Yet another example is the need to specify accessor functions to enable SYCL kernels to access device buffers. SYCL/DPC++ improves vectorization by permitting flexible vector width optimization to enable performance portability across different architectures. The NAMD implementation exploits this flexibility by changing the use of CUDA warp primitives to the more generalized approach offered by SYCL/DPC++ subgroups.

There is some additional GPU library support that NAMD utilizes for its CUDA implementation. Reductions performed across thread blocks, required for summing scalar potential energies and the tensor pressure virial, make use of the CUB library. For the SYCL/DPC++ port, the CUB calls are replaced with calls to oneAPI's oneDPL (data parallel library), which uses C++17 parallel STL for `reduce`, `sort`, and `scan` operations. The PME algorithm for long-range electrostatics requires two 3D FFT calculations, first a forward FFT from real space to reciprocal space, then a summation in reciprocal space, followed by a backward FFT from reciprocal space back to real space. The multi-node code path provides various strategies, decomposing the 3D FFT into 2D slabs or 1D pencils, depending on the number of nodes being used, calling the FFTW library on the CPU to accomplish these partial calculations, because the ensuing latency from attempting to perform these partial FFT calculations on GPUs would hinder performance. However, there is a single-node code path in NAMD 2.14 that benefits from performing these FFTs on a single GPU, making use of the cuFFT library. For the SYCL/DPC++ port, the cuFFT calls are replaced by calls to oneAPI's oneMKL (math kernel library) FFT routines.

The short-range non-bonded forces account for roughly 90% of the computational work per step, making it first priority to accelerate on GPU devices. The porting approach involved creating new `Dpcpp`-prefixed versions of the relevant files, where object classes were renamed from a `Cuda` prefix to a `Dpcpp` prefix. Some of the fundamental data elements needed to also be changed, for example, changing CUDA `float4` to SYCL/DPC++ `sycl::float4`. The main kernel for calculating these forces makes use of multiple sub-kernel calls for aspects of its calculation and is further assisted by auxiliary kernels that generate the necessary input data structures and kernels that finish the sum reduction of the virial and energies. The underlying algorithmic approach and its CUDA implementation have been previously described [7]. Although much of the porting of these kernels to SYCL/DPC++ is straightforward, if tedious, there were a few issues that bear mentioning. The CUDA implementation relies heavily on warp-level intrinsics, which required in the SYCL/DPC++ port using subgroups together with oneDPL calls for `reduce`, `shuffle`, and `atomic_ref` operations. The CUDA implementation also makes special use of the texture memory cache on NVIDIA GPUs for reading Lennard–Jones interaction parameters from a small square table for the van der Waals force calculation and also for interpolating the force magnitude and potential energies from a precomputed lookup table. The first case for parameter lookup is simply to obtain constants for the calculation, whereas the second makes use of special texture memory hardware to perform the linear interpolation. For the SYCL/DPC++ port, the values are read from global device memory, and the linear interpolation has to be explicitly performed from the table values. The non-bonded kernel calculations themselves are in single precision, but the later summation of the virial and potential energy values use double precision.

The SYCL/DPC++ porting of the bonded force kernels and the PME kernels was largely straightforward. The CUDA implementation of the bonded force kernels maintains a bit flag for each of the five different force types, to indicate whether or not the calculation is to call the GPU kernels or performed on the CPU, which made it easier to debug and test the SYCL/DPC++ port. As mentioned previously, there are two different GPU-based code paths for the PME long-range electrostatics implementation, one set of kernels for multi-node parallelism and the other for single-node simulation. Porting of the PME CUDA kernels posed no major challenge. However, the replacement of the cuFFT library with oneMKL proved more challenging. The cuFFT library provides an interface similar to FFTW, and the CPU-based MKL FFT library provides an FFTW-compatible wrapper and is even offered by the NAMD build architecture scripts as an alternative to building with FFTW support. The oneMKL FFT library for GPU devices does not provide an FFTW-compatible wrapper. Instead, the caller must specify the layout of both input and output data, in the form of memory access strides for each dimension. For the real-to-complex forward FFTs used in NAMD, the strides for the complex domain are approximately half of those for the real domain, taking advantage of symmetry in the transformation. Besides getting the strides right, oneMKL FFT requires that the input data be padded to the SIMD vector length, with the resulting output similarly padded. This required launching extra kernels to pad the input grid before invoking the FFT and then to undo the padding from the FFT output.

The porting effort was assisted by creating an array debug utility to enable careful examination of device buffers before and after kernel executions. The utility provides a mechanism to write a device buffer to a file or read a device buffer from a file. A buffer comparison function is provided to allow in-place comparison of a buffer with a trusted result, reporting array index locations of integer differences when nonzero or floating point differences when above a given tolerance. The idea is to facilitate direct comparison

between the SYCL/DPC++ code and the existing CUDA code. The utility interface employs macro expansion, to completely hide its use in production builds.

4 RESULTS

The main goal of this project has been to prepare NAMD for the ANL Aurora supercomputer, which has been pursued in collaboration with Intel software engineering personnel. However, since good performance from NAMD requires hardware support for double-precision atomics, which will not be available until the release of the Intel Xe-HPC (Ponte Vecchio) GPUs, we will omit reporting performance results on the currently available Intel GPU hardware. Although the possibility exists to build SYCL/DPC++ code for NVIDIA GPUs using the DPC++ LLVM Compiler project from Codeplay, the project does not provide a working version of the oneDPL or the oneMKL FFT libraries, which prevents NAMD from being built without significant code workarounds. Instead we will report only on correctness of the NAMD SYCL/DPC++ implementation tested on various available hardware targets from Intel.

Assessing correctness of a molecular dynamics code is somewhat tricky due to the nature of time integration of a Hamiltonian system. A solution through phase space is chaotic, where a small perturbation to the initial conditions results in exponential divergence of one solution from another, even though both are valid solutions within the same symplectic manifold. In practice, with an approximate solution computed numerically using a symplectic integrator, even when starting two simulations with the same initial conditions, every time step generates its own perturbations due to the non-determinism of parallel computation coupled with the fact that floating point addition does not strictly obey the associative property. This means that repeated runs of NAMD will generate trajectories that diverge from each other, although the results are still statistically valid, correctly sampling the microcanonical ensemble in the case of constant energy simulation.

The divergence in solutions is gradual enough that we can run a constant energy simulation for up to 500 steps and still expect to have sufficient agreement in energies with that of a second run. We use this approach to evaluate correctness of the SYCL/DPC++ code execution. The maximum relative error in total energy over 500 steps is reported in Table 2 for two different sized systems running on various DPC++ target architectures, comparing to a CPU-only run. Having maximum relative error for a GPU run less than 1e-05 indicates correctness for that test case. Note that the CPU-only run being one to two orders of magnitude better than the GPU runs is expected, reflecting the use of double precision for the CPU-only calculation versus mixed precision for the GPU calculations. The "CPU (DPC++)" run targets the CPU as a SYCL device.

architecture	ApoA1 (92K atoms)	STMV (1M atoms)
CPU-only (2nd run)	4.35841e-08	3.95857e-07
A5000 (CUDA)	3.17476e-06	4.97212e-06
CPU (DPC++)	2.89769e-06	3.29257e-06
Gen9 (DPC++)	2.82174e-06	3.84310e-06
ATS/Xe-HP (DPC++)	2.09291e-06	3.39862e-06

Table 2: The maximum relative error in total energy for a 500-step constant energy simulation is shown for various architectures for two different sized systems, demonstrating the correctness of the SYCL/DPC++ implementation.

5 FUTURE WORK

Now that the SYCL/DPC++ port of the NAMD GPU-offload kernels is demonstrated to be correct, work is commencing on porting the GPU-resident code path. Providing GPU-resident NAMD for the ANL Aurora supercomputer will be crucial to the scientific studies that are planned for this exciting exascale resource.

It will also be important to optimize these new SYCL/DPC++ implementations. Although the most important optimization work will require access to Ponte Vecchio, there are still opportunities for performance tuning on the currently available Intel architectures. One consideration for supporting the mainstream commodity Intel GPUs is modifying portions of the code that rely on hardware supported double precision atomics. Reductions of double precision values could be transformed into summing with two single precision values, using Kahan summation or compensated summation technique to preserve the low order bits of precision. Providing this alternative approach in the code could improve NAMD performance across other commodity GPUs, since lower end GPUs generally offer limited, if any, support for double precision.

Finally, the SYCL/DPC++ port has generated much duplicated code. Code duplication poses problems with future software maintenance, since a bug fix to one algorithm will likely need to be applied in more than one place. Performing "diffs" between the respective SYCL/DPC++ and CUDA files shows that most of the higher level kernel management differences are due simply to a renaming of objects and functions, that appear to be easily unified by creating, say, generic NAMD "vectorization" objects. Even looking at the kernel level differences, most of these are changes in syntax rather than function and could also be unified with suitably defined macro expressions, for example, providing a consistent means of access to hide the differences between accessing `f.x` of a CUDA `float3` and `f.x()` of a SYCL/DPC++ `sycl::float3`.

6 CONCLUSIONS

In this paper, we have presented our experiences porting the NAMD molecular dynamics application to Data Parallel C++ (DPC++) and demonstrated correctness of the present SYCL/DPC++ port of the GPU-offload force kernels of NAMD. In the near term, the SYCL/DPC++ porting effort will enable NAMD to support the upcoming Aurora supercomputer at Argonne National Laboratory, which will be among the first true exascale computers. This porting effort will also provide NAMD support across all of the Intel GPU devices. Moreover, the goal of Intel's oneAPI is to provide a common set of languages and libraries to support heterogeneous computing solutions from all vendors, and we note that there is already third-party compiler support targeting NVIDIA and AMD GPUs. We see the adoption of open standards like SYCL/DPC++ to be ultimately beneficial to NAMD, giving the code a better opportunity to outlive its creators.

ACKNOWLEDGMENTS

The authors wish to acknowledge the programming contributions of Tareq Malas (formerly of Intel, now of Meta) and Mike Brown (Intel). This work was supported by NIH grant P41-GM104601, Intel funding for the oneAPI academic Center of Excellence at UIUC, and the Aurora Early Science Program of the Argonne Leadership Computing Facility, which is a DOE Office of Science user facility supported under Contract DE-AC02-06CH11357.

REFERENCES

[1] Lorenzo Casalino, Abigail C Dommer, Zied Gaieb, Emilia P Barros, Terra Sztain, Surl-Hee Ahn, Anda Trifan, Alexander Brace, Anthony T Bogetti, Austin Clyde, Heng Ma, Hyungro Lee, Matteo Turilli, Syma Khalid, Lillian T Chong, Carlos Simmerling, David J Hardy, Julio DC Maia, James C Phillips, Thorsten Kurth, Abraham C Stern, Lei Huang, John D McCalpin, Mahidhar Tatineni, Tom Gibbs, John E Stone, Shantenu Jha, Arvind Ramanathan, and Rommie E Amaro. 2021. AI-Driven Multiscale Simulations Illuminate Mechanisms of SARS-CoV-2 Spike Dynamics. *Int. J. High Perform. C.* (2021), 10943420211006452. https://doi.org/10.1177/10943420211006452

[2] Abigail Dommer, Lorenzo Casalino, Fiona Kearns, Mia Rosenfeld, Nicholas Wauer, Surl-Hee Ahn, John Russo, Sofia Oliveira, Clare Morris, Anthony Bogetti, Anda Trifan, Alexander Brace, Terra Sztain, Austin Clyde, Heng Ma, Chakra Chennubhotla, Hyungro Lee, Matteo Turilli, Syma Khalid, Teresa Tamayo-Mendoza, Matthew Welborn, Anders Christensen, Daniel G. A. Smith, Zhuoran Qiao, Sai Krishna Sirumalla, Michael O'Connor, Frederick Manby, Anima Anandkumar, David Hardy, James Phillips, Abraham Stern, Josh Romero, David Clark, Mitchell Dorrell, Tom Maiden, Lei Huang, John McCalpin, Christopher Woods, Alan Gray, Matt Williams, Bryan Barker, Harinda Rajapaksha, Richard Pitts, Tom Gibbs, John Stone, Daniel Zuckerman, Adrian Mulholland, Thomas Miller III, Shantenu Jha, Arvind Ramanathan, Lillian Chong, and Rommie Amaro. 2022. #COVIDisAirborne: AI-Enabled Multiscale Computational Microscopy of Delta SARS-CoV-2 in a Respiratory Aerosol. *Int. J. High Perform. C.* (2022). To appear.

[3] Defne Gorgun, Muyun Lihan, Karan Kapoor, and Emad Tajkhorshid. 2021. Binding Mode of SARS-CoV2 Fusion Peptide to Human Cellular Membrane. *Biophys. J.* 120, 3 (2021), 191a.

[4] David J. Hardy, John E. Stone, Barry Isralewitz, and Emad Tajkhorshid. 2021. Lessons Learned from Responsive Molecular Dynamics Studies of the COVID-19 Virus. In *2021 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*. 1–10. https://doi.org/10.1109/UrgentHPC54802.2021.00006

[5] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, and Klaus Schulten. 2005. Scalable Molecular Dynamics with NAMD. *J. Comput. Chem.* 26 (2005), 1781–1802. https://doi.org/10.1002/jcc.20289

[6] James C. Phillips, David J. Hardy, Julio D. C. Maia, John E. Stone, João V. Ribeiro, Rafael C. Bernardi, Ronak Buch, Giacomo Fiorin, Jérôme Hénin, Wei Jiang, Ryan McGreevy, Marcelo C. R. Melo, Brian Radak, Robert D. Skeel, Abhishek Singharoy, Yi Wang, Benoît Roux, Aleksei Aksimentiev, Zaida Luthey-Schulten, Laxmikant V. Kalé, Klaus Schulten, Christophe Chipot, and Emad Tajkhorshid. 2020. Scalable Molecular Dynamics on CPU and GPU Architectures with NAMD. *J. Chem. Phys.* 153 (2020), 044130. https://doi.org/10.1063/5.0014475

[7] John E. Stone, Antti-Pekka Hynninen, James C. Phillips, and Klaus Schulten. 2016. Early Experiences Porting the NAMD and VMD Molecular Simulation and Analysis Software to GPU-Accelerated OpenPOWER Platforms. *International Workshop on OpenPOWER for HPC (IWOPH'16)* (2016), 188–206.

[8] John E. Stone, James C. Phillips, Peter L. Freddolino, David J. Hardy, Leonardo G. Trabuco, and Klaus Schulten. 2007. Accelerating Molecular Modeling Applications with Graphics Processors. *J. Comput. Chem.* 28 (2007), 2618–2640. https://doi.org/10.1002/jcc.20829

[9] Anda Trifan, Defne Gorgun, Michael Salim, Zongyi Li, Alexander Brace, Maxim Zvyagin, Heng Ma, Austin Clyde, David Clark, David J. Hardy, Tom Burnley, Lei Huang, John McCalpin, Murali Emani, Hyenseung Yoo, Junqi Yin, Aristeidis Tsaris, Vishal Subbiah, Tanveer Raza, Jessica Liu, Noah Trebesch, Geoffrey Wells, Venkatesh Mysore, Thomas Gibbs, James Phillips, S. Chakra Chennubhotla, Ian Foster, Rick Stevens, Anima Anandkumar, Venkatram Vishwanath, John E. Stone, Emad Tajkhorshid, Sarah A. Harris, and Arvind Ramanathan. 2022. Intelligent Resolution: Integrating Cryo-EM with AI-Driven Multi-Resolution Simulations to Observe the SARS-CoV-2 Replication–Transcription Machinery in Action. *Int. J. High Perform. C.* (2022). To appear.