Enabling NAMD for Intel Xe

Tareq Malas, Intel Corporation
David Hardy, University of Illinois at Urbana-Champaign

oneAPI DevSummit at SC21
Sunday, November 14, 2021

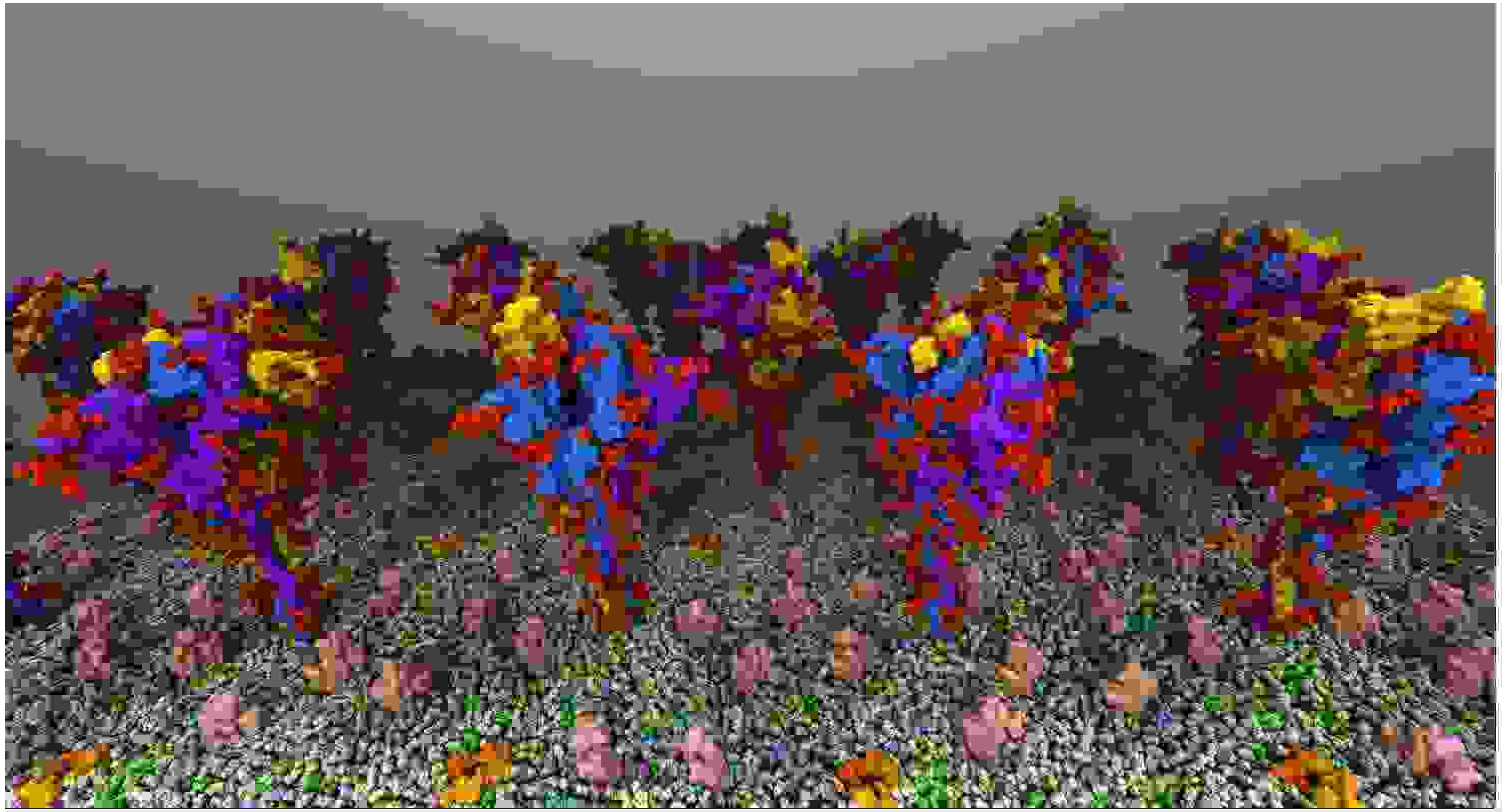
Overview of Presentation

- Introduction to NAMD molecular dynamics application
- How NAMD parallelization makes use of GPUs
- Porting NAMD from CUDA to oneAPI / DPC++
- Relative debugging techniques for porting CUDA codes

NAMD: Nanoscale Molecular Dynamics

https://www.ks.uiuc.edu/Research/namd/
Phillips, et al. *J. Chem. Phys.* 153, 044130 (2020)

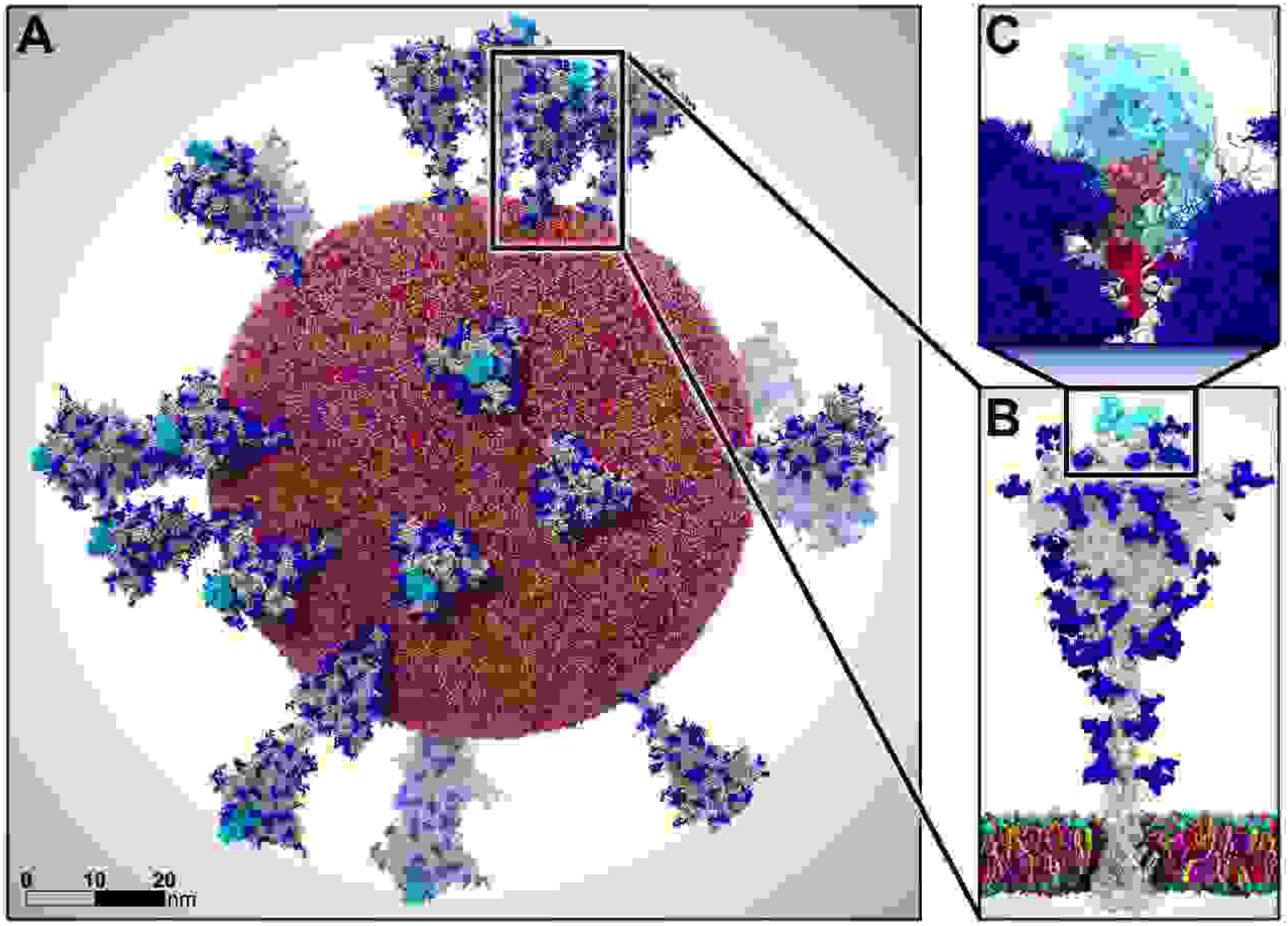
- Parallel molecular dynamics application written in C++ with Charm++ objects
- Runs on all major operating systems, on laptops up through supercomputers
- Specializes in parallel scaling of large biomolecular simulations
- Many advanced features:
  - Enhanced sampling methods
  - Alchemical free energy methods
  - Collective variables module (Colvars)
  - TCL and Python scripting
- Over 25,000 registered users
- Over 15,000 citations of our NAMD reference papers (Google Scholar)


Investigations of coronavirus (SARS-CoV-2) spike dynamics.
Credit: Tianle Chen, Karan Kapoor, Emad Tajkhorshid (UIUC).
Simulations with NAMD, movie created with VMD.

NAMD Simulating SARS-CoV-2 on Frontera and Summit

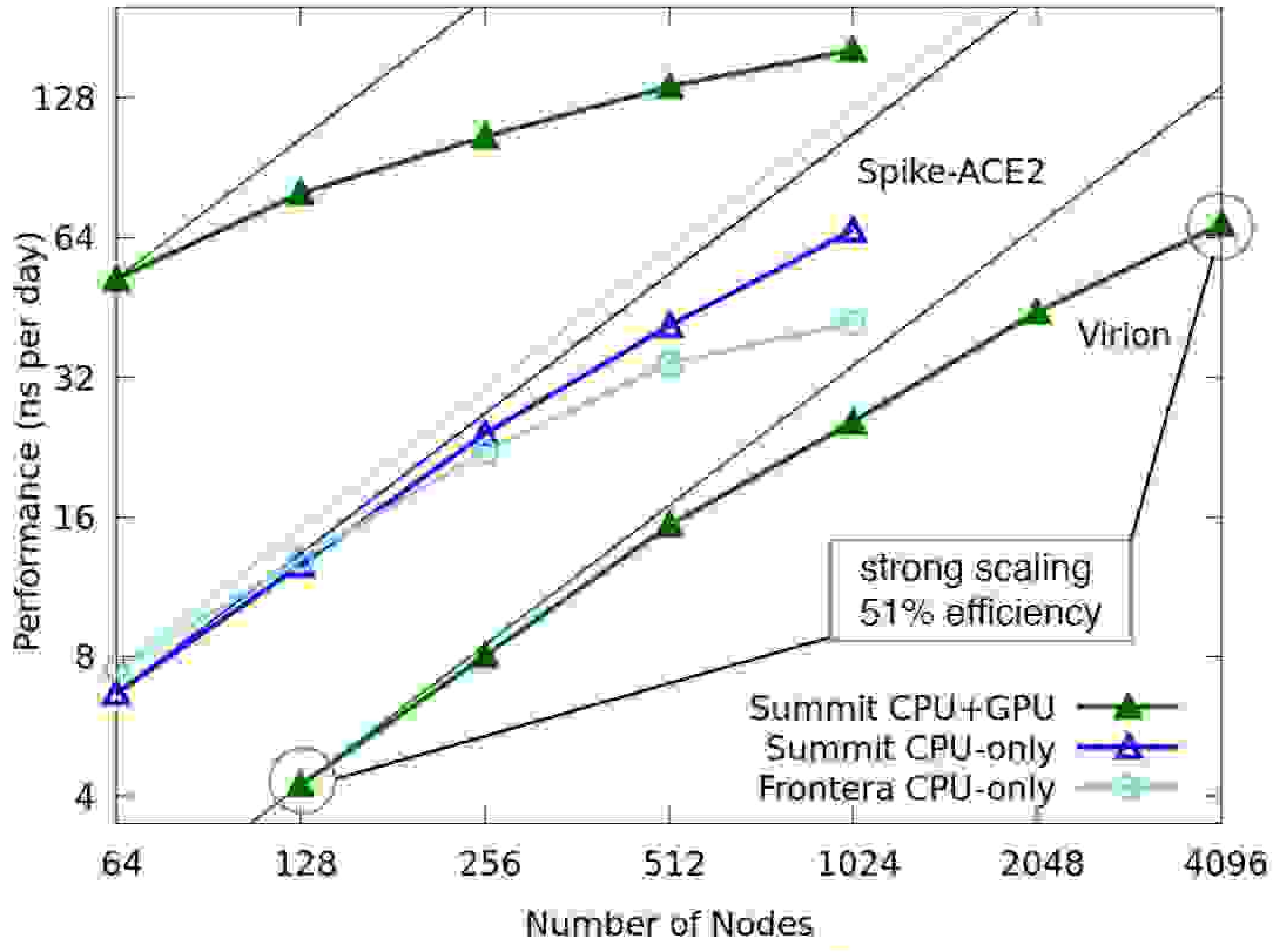
Collaboration with Amaro Lab at UCSD, images rendered by VMD
Winner of Gordon Bell Special Prize at SC20, project involved overall 1.13 Zettaflops of NAMD simulation

(A) Virion, (B) Spike, (C) Glycan shield conformations



Scaling performance:

- ~305M atom virion — @4096 nodes 12.4K atoms/GPU
- ~8.5M atom spike — @1024 nodes 1.4K atoms/GPU

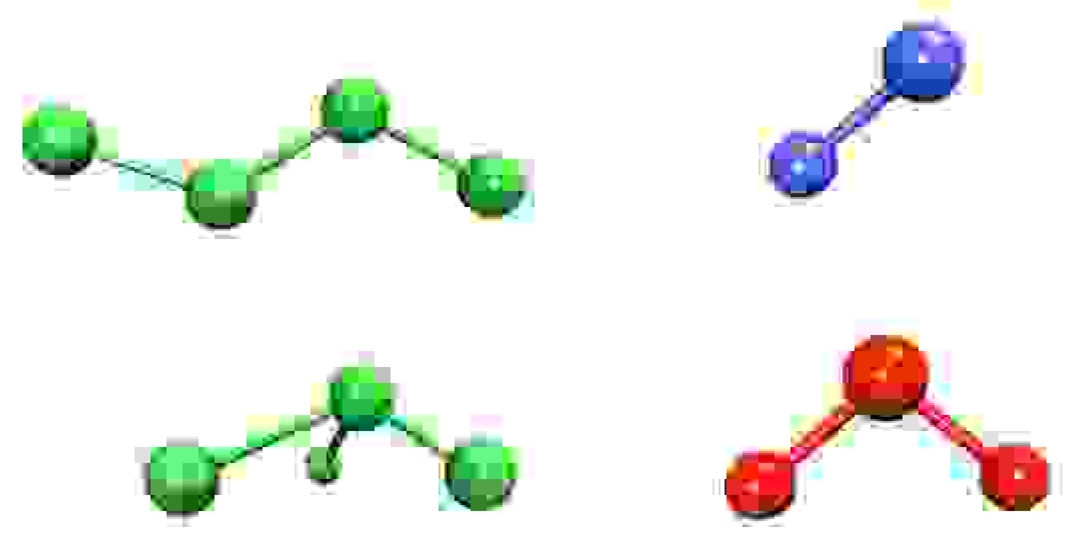


Casalino, et al. *IJHPCA*, 2021 https://doi.org/10.1177%2F10943420211006452

Molecular Dynamics Simulation

- Most fundamentally, integrate Newton's equations of motion:

$$m_i \frac{d^2\vec{r}_i}{dt^2} = \vec{F}_i = -\vec{\nabla} U(\vec{R}) \longleftarrow \text{integrate for up to billions of time steps}$$



$$U(\vec{R}) = \underbrace{\sum_{bonds} k_i^{bond}(r_i - r_0)^2}_{U_{bond}} + \underbrace{\sum_{angles} k_i^{angle}(\theta_i - \theta_0)^2}_{U_{angle}} + \underbrace{\sum_{dihedrals} k_i^{dihe}[1 + \cos(n_i\phi_i + \delta_i)]}_{U_{dihedral}} +$$

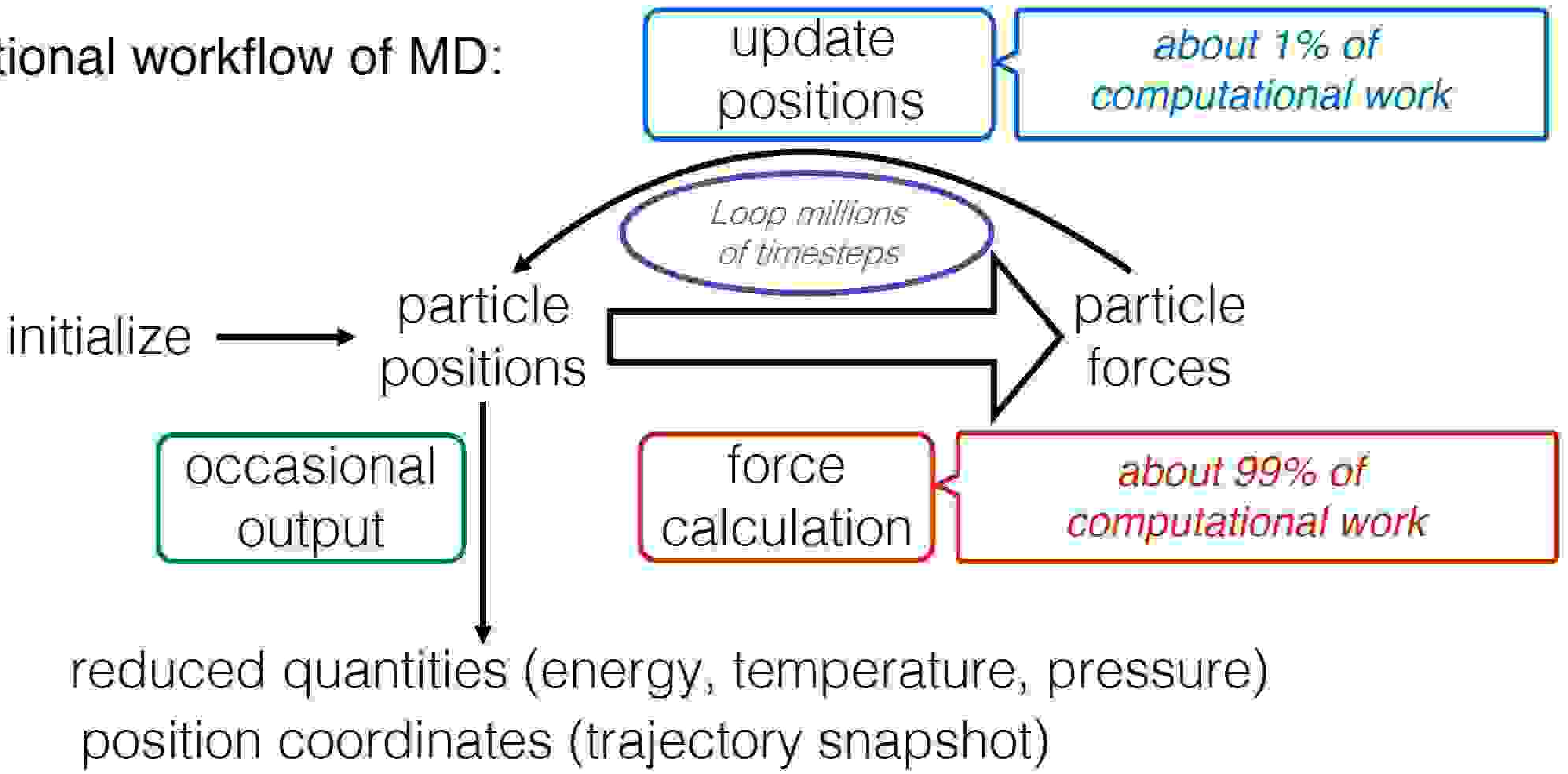
most of the computational work $\longrightarrow$

$$\underbrace{\sum_i \sum_{j\neq i} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] + \sum_i \sum_{j\neq i} \frac{q_i q_j}{\epsilon r_{ij}}}_{U_{nonbond}}$$

(Lennard-Jones) (electrostatics)

Parallelism for MD Simulation Limited to Each Timestep

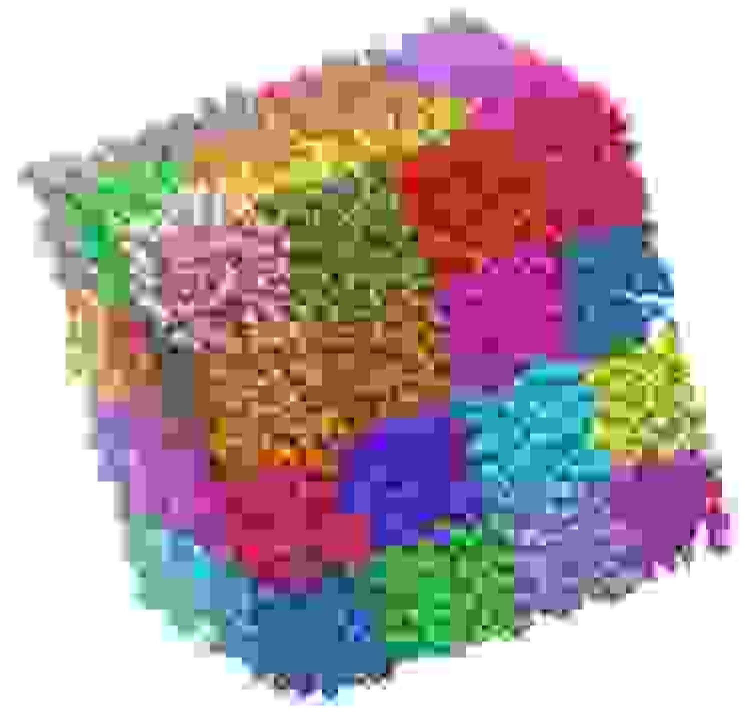
Computational workflow of MD:

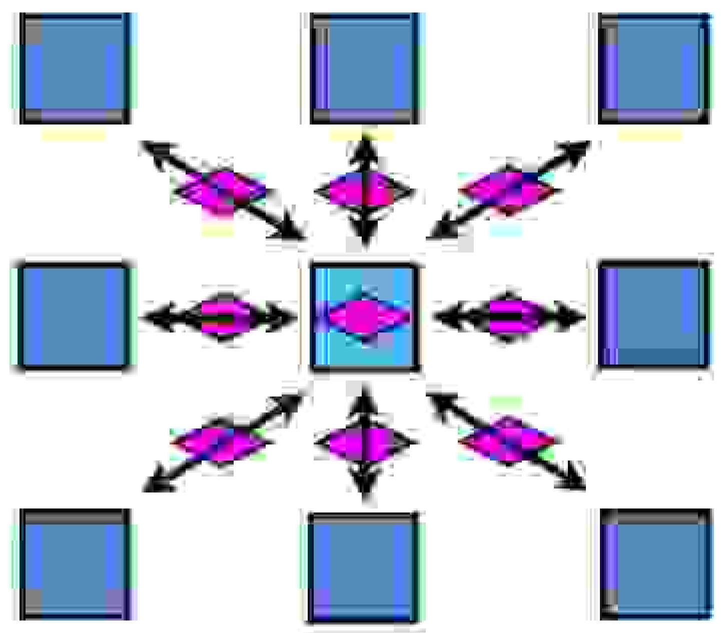

update positions
about 1% of computational work
Loop millions of timesteps
initialize
particle positions
particle forces
occasional output
force calculation
about 99% of computational work
reduced quantities (energy, temperature, pressure)
position coordinates (trajectory snapshot)

Improve Parallelism: Decompose Data **and** Work

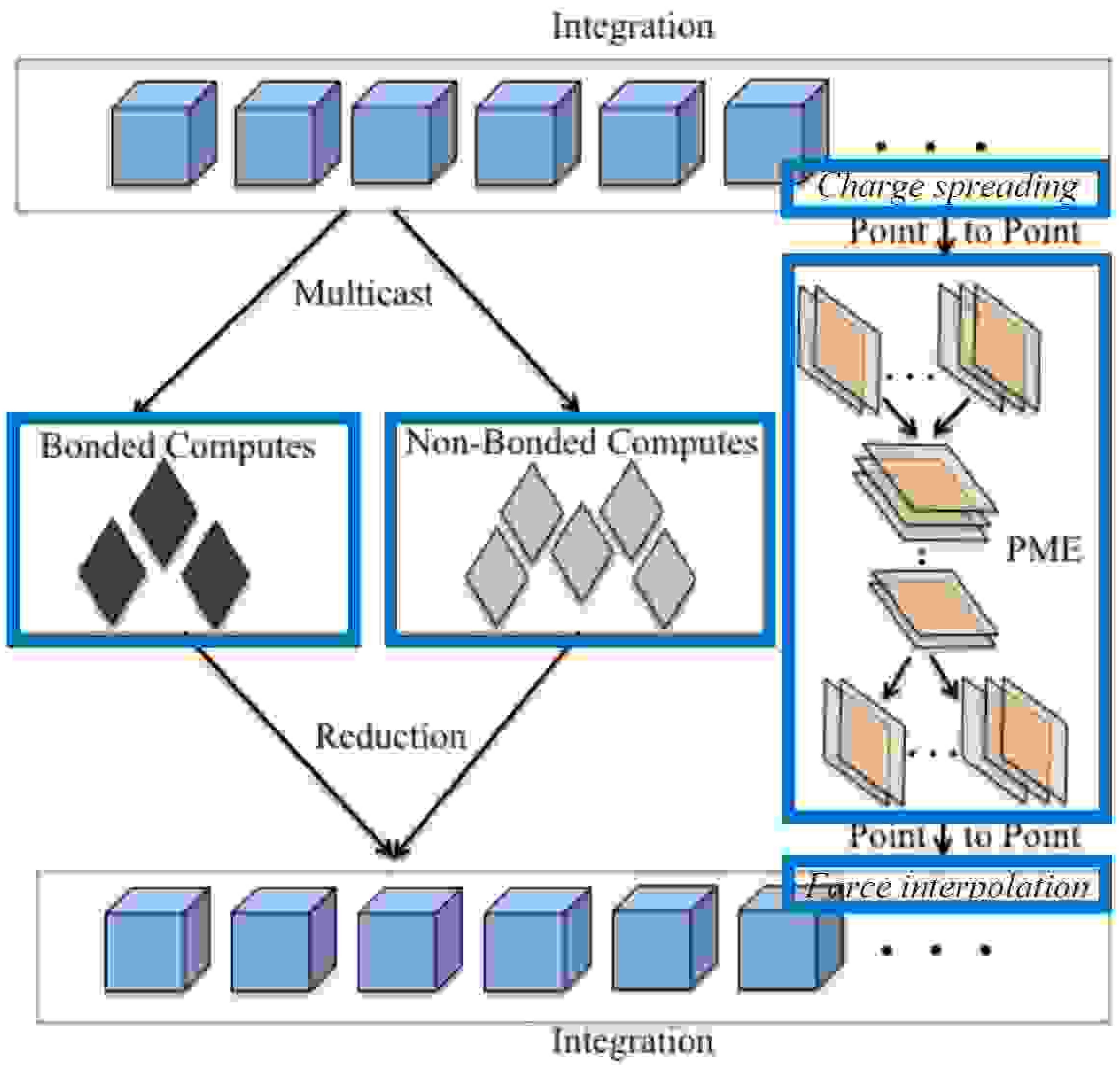
Kale et al., *J. Comp. Phys.* 151:283-312, 1999

- Atoms are decomposed into fixed volume **patches** within the system
- Forces that move atoms are calculated in parallel at each step between adjacent patches
- Work decomposition into **compute objects** creates much greater amount of parallelization, facilitates measurement-based load balancing with Charm++
- Migrate atoms to adjacent patches, updating domain decomposition after every **cycle** (e.g., 20 steps)


Spatial decomposition of atoms into patches

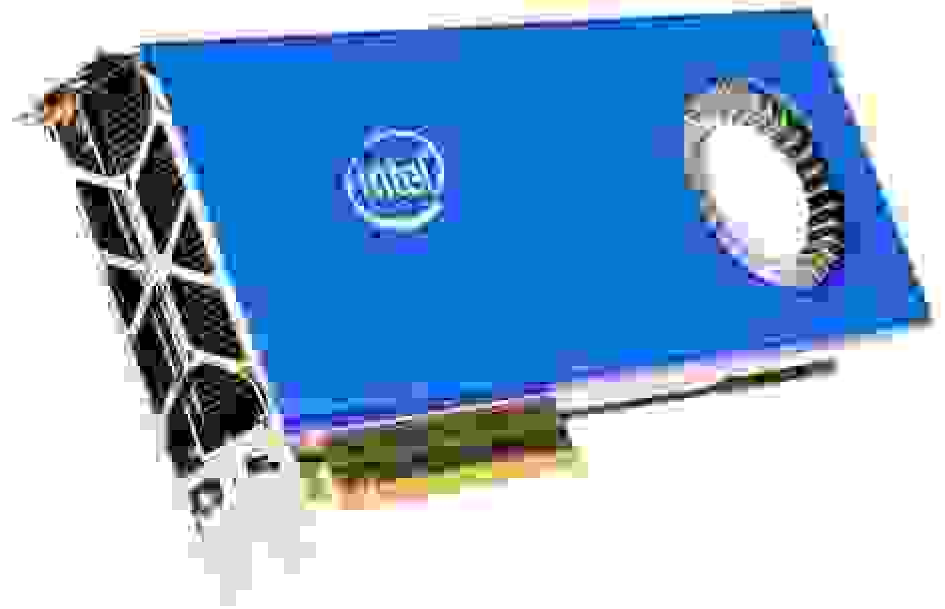

Work decomposition of patch interactions

NAMD Decomposes Force Terms into Fine-Grained Objects for Scalability



Offload forces to GPU

Requires aggregating compute objects



NAMD Offloads Force Calculation to GPU

Partition work between CPU and GPU

Showing approximate percentage of total work per step:

force calculation on GPUs

- **Short-range non-bonded forces (90%)**
- Long-range PME electrostatics (5%)
- Bonded forces (2%)
- Corrections for excluded interactions (2%)

update coordinates on CPUs

- Integrator, rigid bond constraints (1%)

Enhanced sampling methods: additional forces, grid potentials, collective variables

Why is NAMD adopting oneAPI?

- Support upcoming exascale computers: ANL Aurora (Intel)
- oneAPI / DPC++ provides advantages
  - Modern C++ interface to GPU devices
  - Host-side code is **much** simpler than OpenCL
  - Same data structure definitions for both host and device
  - DPC++ incorporates open-standard SYCL with community extensions
  - Code portability across various hardware targets: CPU, GPU, FPGA

How does DPC++ differ from CUDA?

- Uses advanced C++ features, such as lambda expressions to define per-thread work and exceptions (try-catch block) for error handling
- Memory management & transfers
  - Asynchronous by default
  - Associated with a SYCL queue (including allocations/frees)
- Shared memory allocations → local memory accessors (created before kernel invocation)
- Does not assume a certain SIMD width (warp and sub-group)
  - Should generalize warp-based mechanisms
  - Can enforce a sub-group size

NAMD has a **LOT** of CUDA code

- Start oneAPI / DPC++ porting with **stable code base** (version 2.14)

Component	# of C/h files	# of cu files	# of kernels	src line count
Non-bonded force	6	2	20	5.8k
Bonded force	3	1	2	3.9k
PME - single node	6	1	5	4.1k
PME - scalable	6	1	3	3.3k
Utilities	8	1	1	1.7k
Total	29	6	31	18.8k

Porting Strategy

- We used a divide-and-conquer strategy, by using preprocessor switches to decouple the components in the CUDA code
  - Significantly reduces development and debugging complexity
- Separated components include
  - Non-bonded force & device utilities
  - Bonded force
  - PME (Particle-Mesh Ewald) – uses FFT
- Utilizing oneAPI libraries:
  - oneDPL uses C++17 parallel STL *sort* and *scan* operations to replace CUB library
  - oneMKL FFT replaces cuFFT library

Accelerated Development with DPCT

- Utilized Intel® DPC++ Compatibility Tool (DPCT) to accelerate code development
- Started with migrating the CUDA implementation
  - Saves > 80% of code porting effort
  - For example, threadIdx.x → ndItem.get_local_id(2)
- Provides a good source to practice DPC++ syntax

DPC++ Offloading of the Force Computations

- Successful DPC++ offloading of NAMD to:
  - Intel® Xeon® CPU
  - Intel Gen9 integrated graphics
  - Intel® Iris® Xe Max and Xe – HP Software Development Platform discrete graphics
- Enabled multi-GPU and multi-node scalability with DPC++
- Includes implementation of DPC++ offload code management in NAMD
  - Interface with Charm++
  - Perform data management (transfer and storage)
  - Multiple CPU threads offloading to multiple DPC++ devices
- Validated benchmarks: Tiny (512 atoms), ApoA1 (90K), F1-ATPase (328K), STMV (1M)

DPC++ Improves Vectorization

- Using flexible vector width optimization towards performance portability to various architectures

- Changed use of CUDA warp primitives to generalized code supporting DPC++ sub-groups for efficient vector computation on different target architectures

Future Plans

- Make the DPC++ implementation available to NAMD community
  - Merge into the main public repository – targeting end-of-year
- Port NAMD GPU-resident code path (NAMD 3.0) to DPC++
- Use Intel® Vtune Profiler and Intel® Advisor tools to continuously optimize NAMD DPC++ for performance on Aurora supercomputer
- Experiment with NAMD DPC++ on NVIDIA and AMD GPUs

Debugging Challenges

- Chasing bugs in ported large applications from CUDA to DPC++ can be involved
  - Especially when dealing with large irregular arrays of structures
  - Large arrays may be pipelined to multiple kernels and code crashes at later stage when numbers become far from the expected value (e.g. NaNs)
  - Sometimes we are porting a complex application outside of our domain of expertise

Proposed Solution

- Code porting mostly involves changing the syntax and library calls
  - All/most of the algorithm and result remain the same
  - Most/all non kernels code remains intact
- Add utilities to capture kernels' input/output across languages (DPC++, CUDA)
  - Write to a file the input/output data of the kernels in reference language
  - Read the data files in the development code and compare the arrays
- Results
  - Developed easy to add macros around the kernel call
  - Allows the developer to capture the first difference location in code and data

Acknowledgments

- NAMD development is funded by NIH P41-GM104601
- NAMD collaboration with Intel:
  - Grant from Intel for COVID research software tool development
  - UIUC with Intel has established oneAPI academic Center of Excellence
  - David Hardy is an Intel Software Innovator
  - DevMesh project URL: https://devmesh.intel.com/projects/namd-scalable-molecular-dynamics
  - Special thanks to Intel engineers: Mike Brown, Alex Wells, and Robert Cohn
- NAMD oneAPI team includes: Jaemin Choi (UIUC) and Wei Jiang (ANL)