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

**David J. Hardy**
**University Of Illinois at Urbana-Champaign**
**Beckman Institute of Advanced Science and Technology**

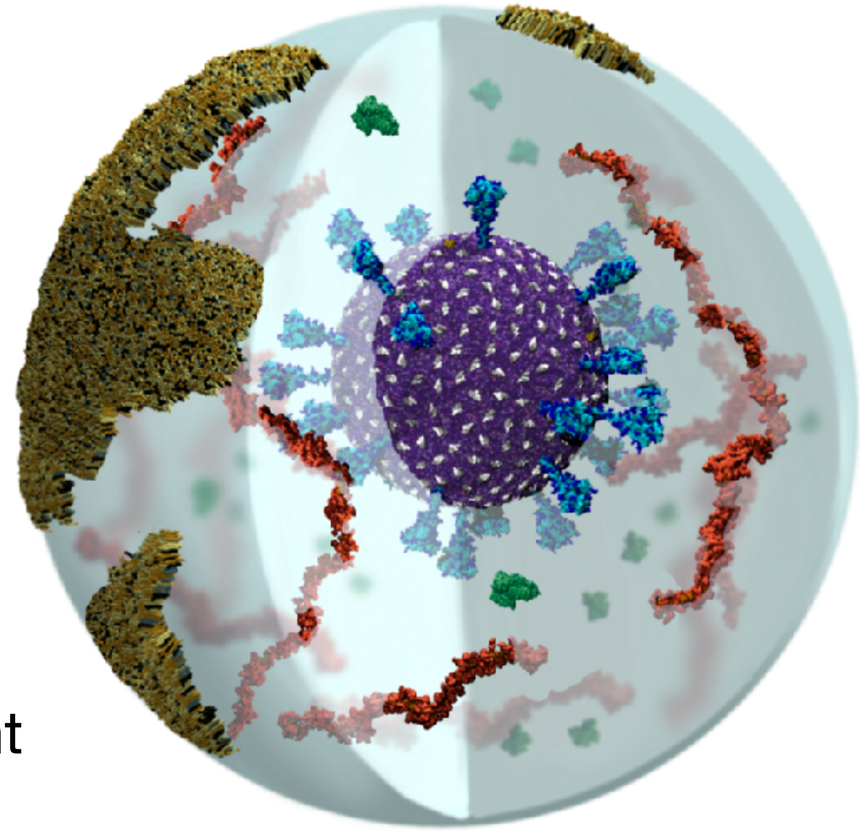
**Jaemin Choi**
**University of Illinois at Urbana-Champaign**
**Department of Computer Science**

**Wei Jiang**
**Argonne National Laboratory**
**Argonne Leadership Computing Facility**

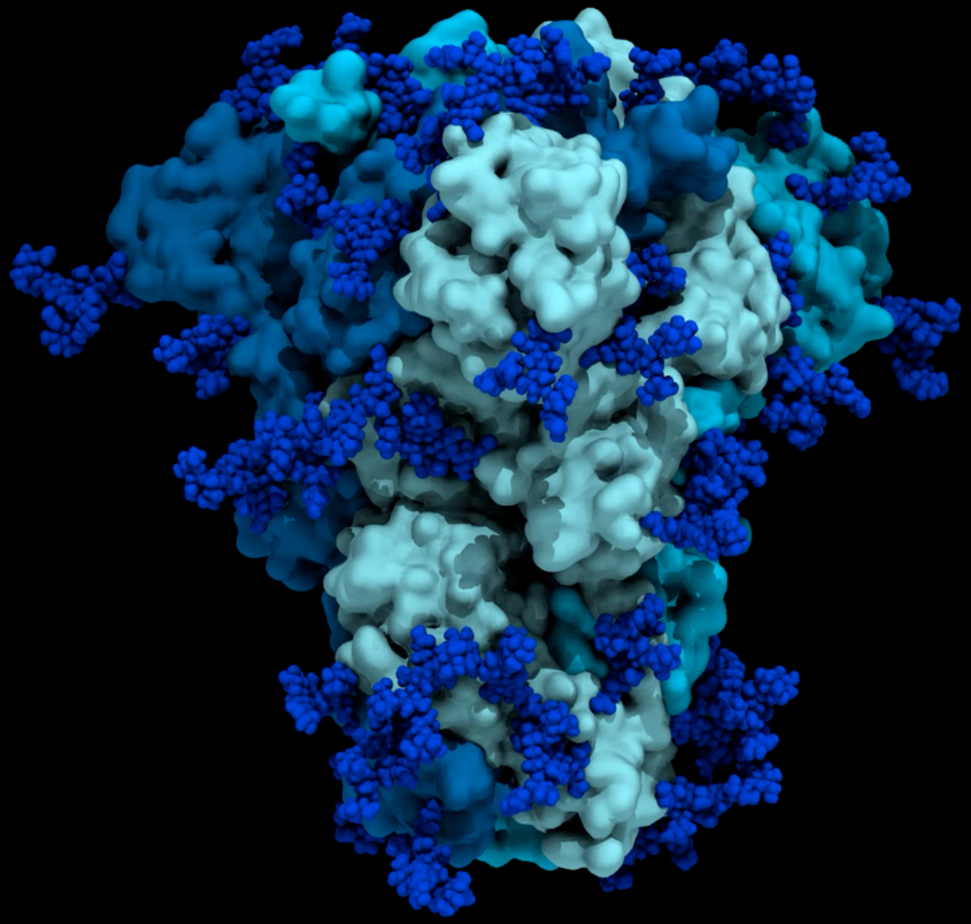
**Emad Tajkhorshid**
**University of Illinois at Urbana-Champaign**
**Beckman Institute of Advanced Science and Technology**

# Molecular Dynamics Combats Diseases Like COVID-19

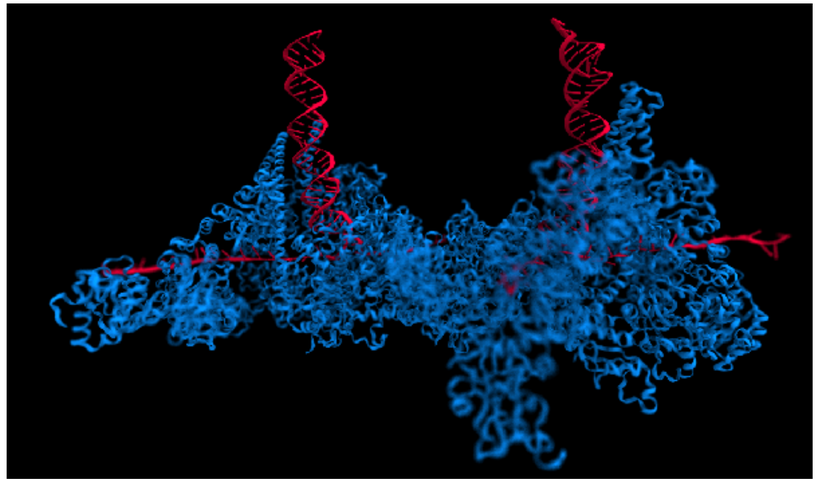
- Molecular dynamics (MD) simulation software and HPC resources provide access to spatial and temporal scales not available to physical experiments
- Atomistic dynamics can reveal the molecular basis for diseases
- By studying viruses and other diseases with MD and related methods, researchers can inform the development of new treatments and therapies



Delta variant of coronavirus (SARS-CoV-2) in aerosol droplet.
Credit: A. Dommer, L. Casalino, F. Kearns, R. Amaro (UCSD).
Simulations (1B atoms) with NAMD, renderings with VMD.



# NAMD Multi-GPU Scaling on DGX-A100

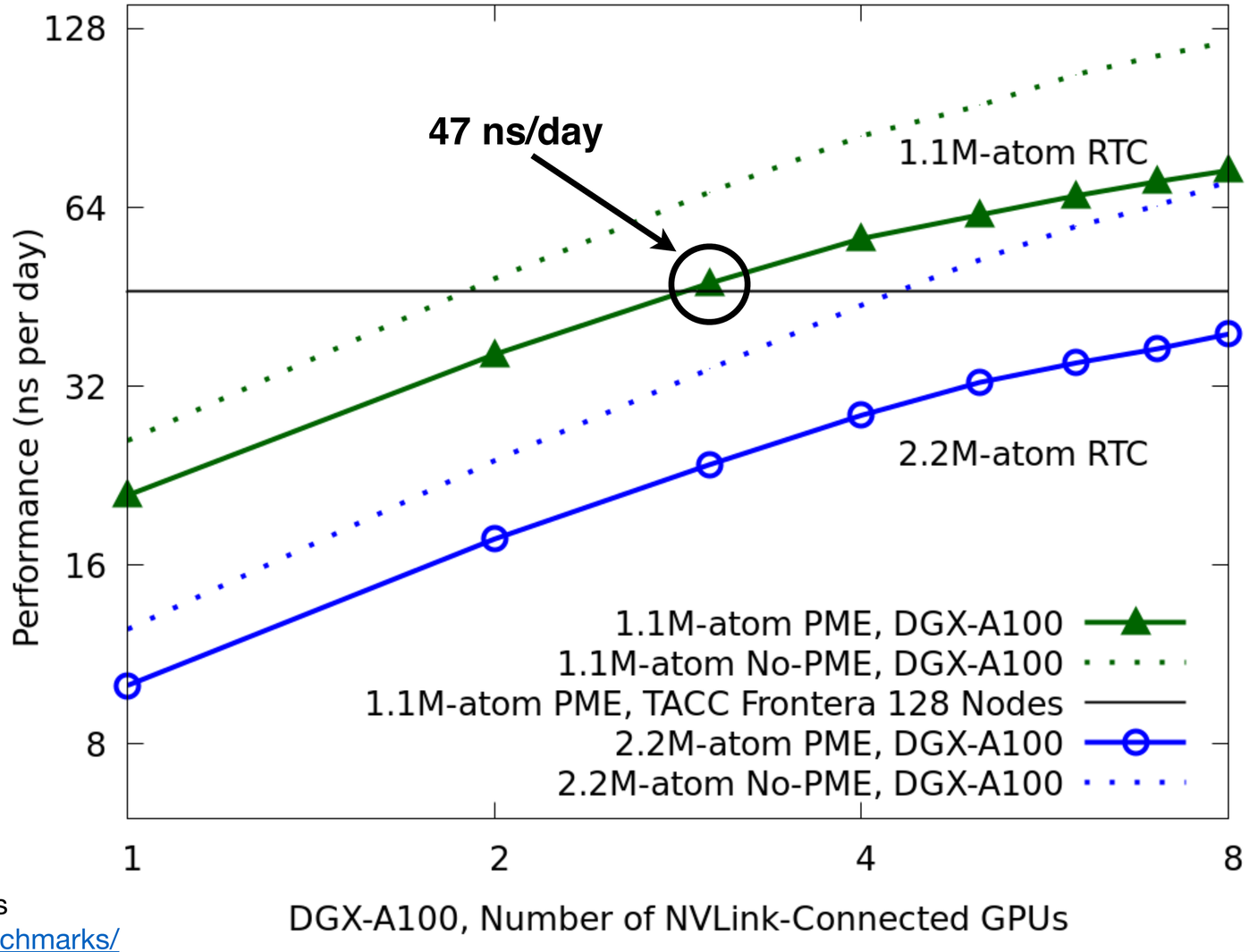


Replication Transcription Complex


ALCF ThetaGPU: DGX-A100

TACC Frontera

Simulation details:
8Å cutoff, NVT, MTS 2fs, 4fs PME, rigid bonds
https://www.ks.uiuc.edu/Research/namd/benchmarks/



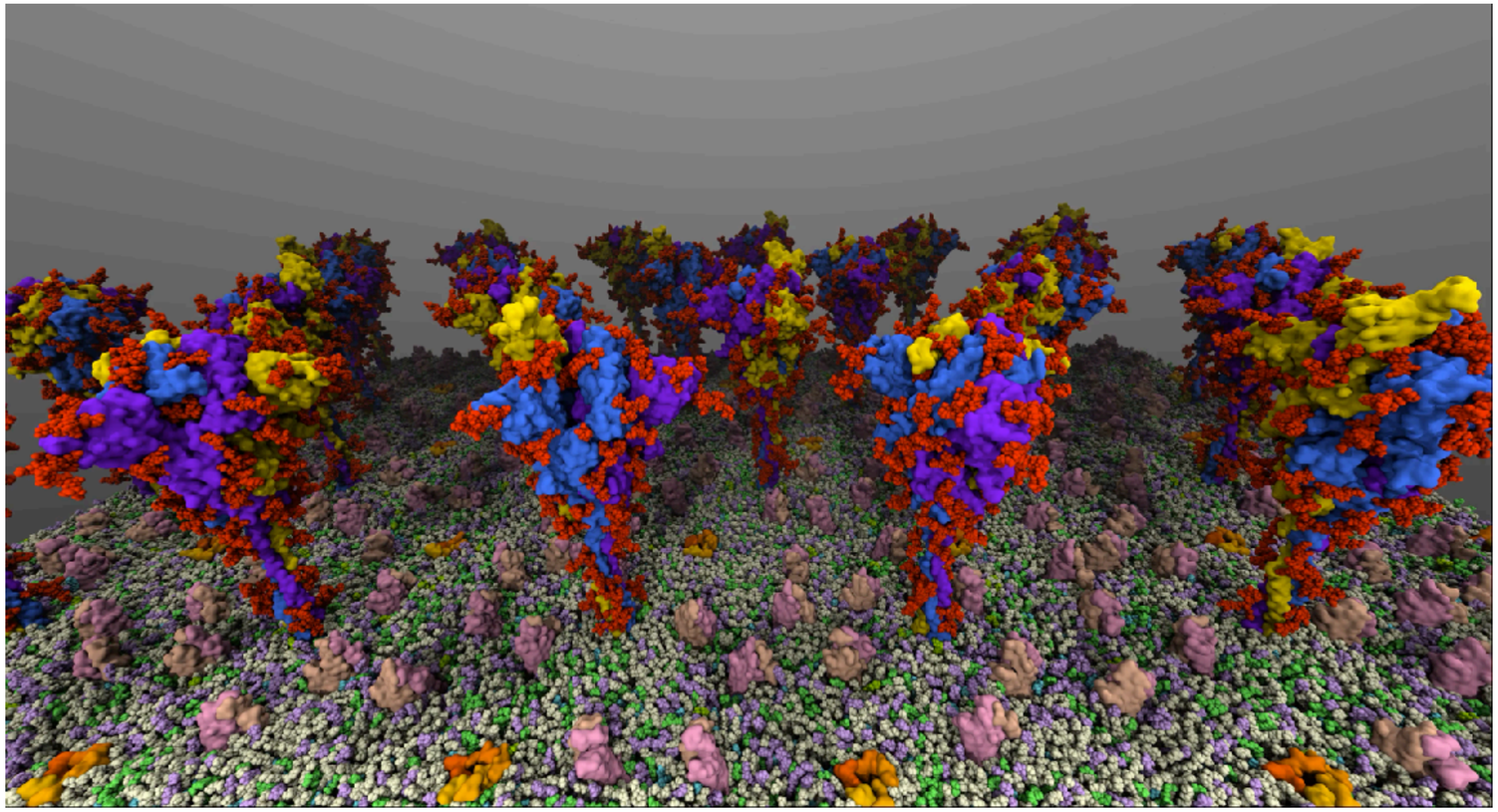
# NAMD: Scalable Molecular Dynamics

- 25-year-old molecular dynamics application written in C++ with Charm++ parallel objects
- Simulate movements of biomolecules over time
- Emphasize parallel scaling of large systems
- Over 25,000 registered users, over 16,000 citations

https://www.ks.uiuc.edu/Research/namd/

Phillips, et al. *J. Comput. Chem.* 26, 1781-1802 (2005)

Phillips, et al. *J. Chem. Phys.* 153, 044130 (2020)



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

# Molecular Dynamics Simulation

**Integrate Newton's equations of motion:**

$$m_i \frac{d^2 \vec{r}_i}{dt^2} = \vec{F}_i = -\vec{\nabla} U(\vec{R})$$

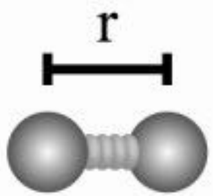
Integrate for millions of time steps

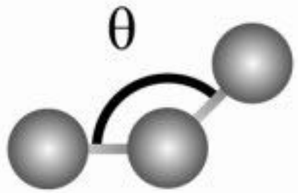
**Bonded**

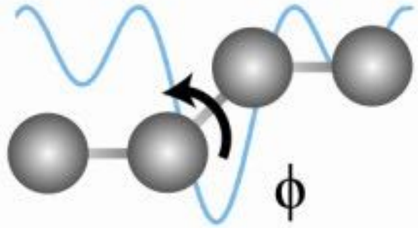
$$E_{total} = \sum_{bonds} K_r(r - r_{eq})^2 + \sum_{angles} K_\theta(\theta - \theta_{eq})^2 + \sum_{dihedrals} \frac{V_n}{2}[1 + \cos(n\phi - \gamma)]$$

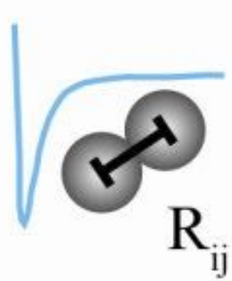
**Non-bonded**

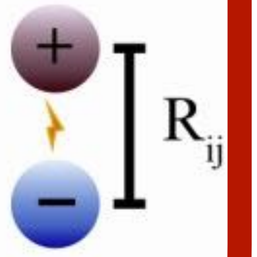
$$+ \sum_{i<j} \left[ \frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} + \frac{q_i q_j}{\epsilon R_{ij}} \right]$$



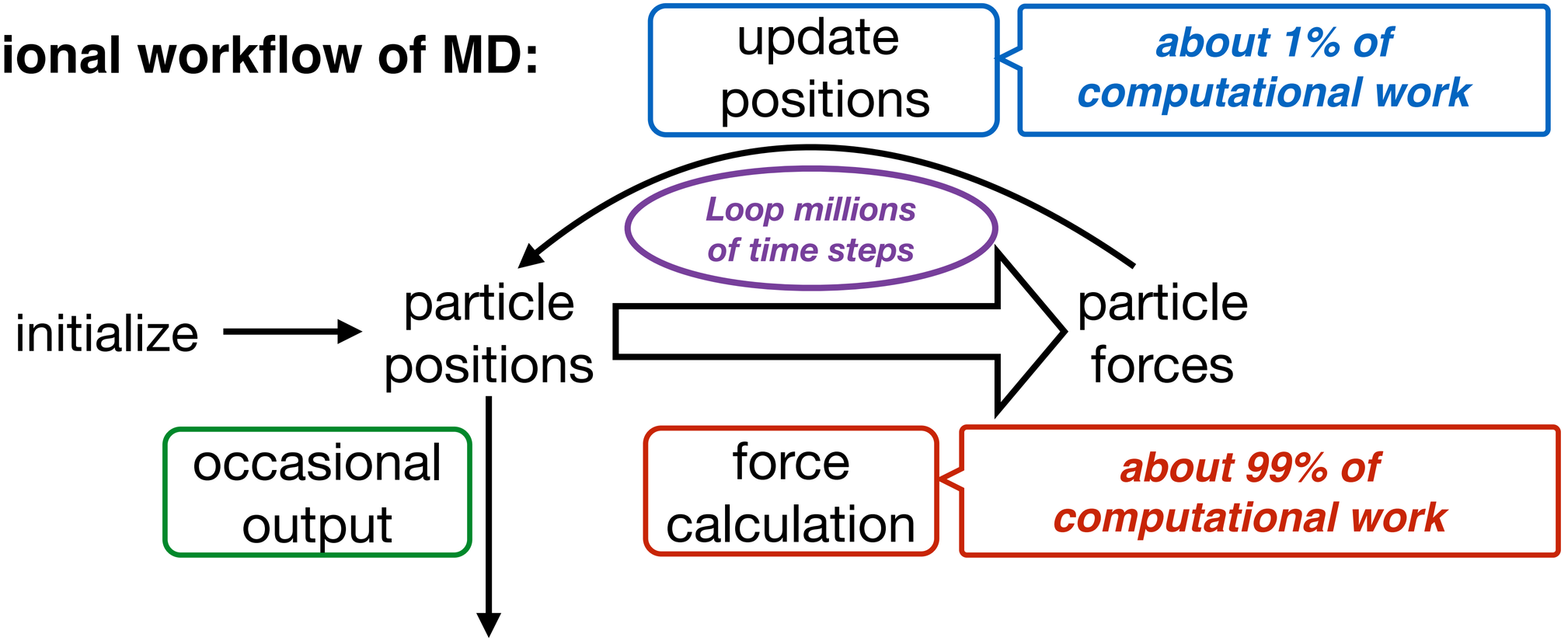




Most computationally intensive part

# Parallelism for MD Simulation Limited to Each Time Step

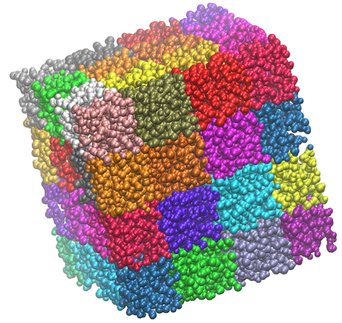
**Computational workflow of MD:**



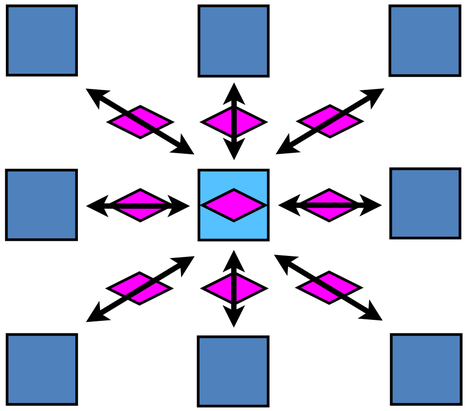
# NAMD Also Parallelizes Interaction Space

- Decompose atoms into equal volume ***patches***
- Calculate pairwise forces between atoms, treat as interactions between neighboring patches
- Decompose patch-patch interaction ***compute objects***
- Moving atoms: update spatial decomposition by ***migrating atoms*** between adjacent patches
- Load balancing: update work decomposition by ***migrating compute objects*** to keep processors consistently occupied

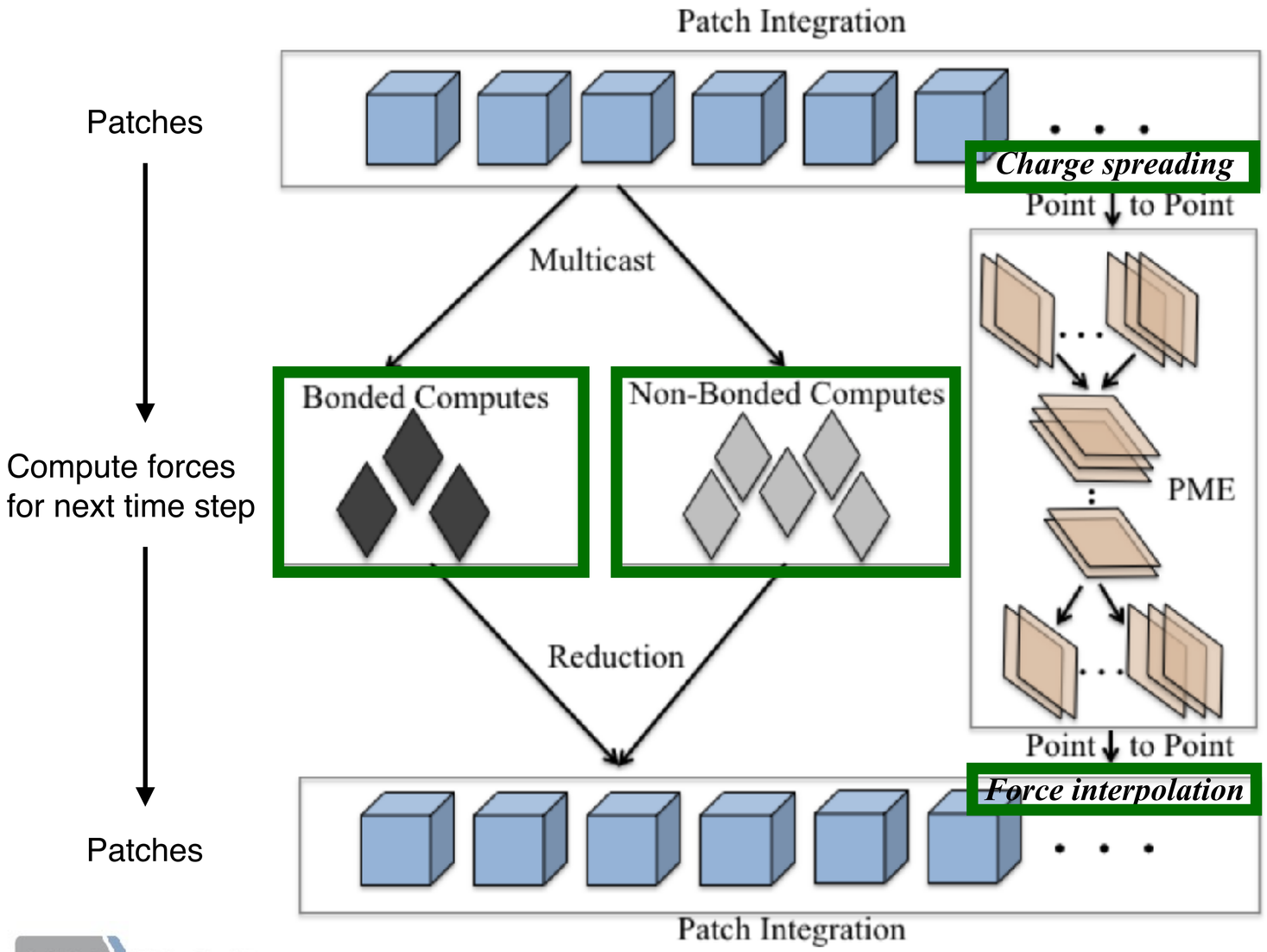
**Spatial decomposition of atoms into patches**



***Work decomposition of patch-patch interactions into migratable compute objects***



# NAMD Parallel Workflow Incorporates GPUs




**Offload force compute to GPU**


**Must aggregate positions**

# Original GPU-Offload Scheme

*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%)**

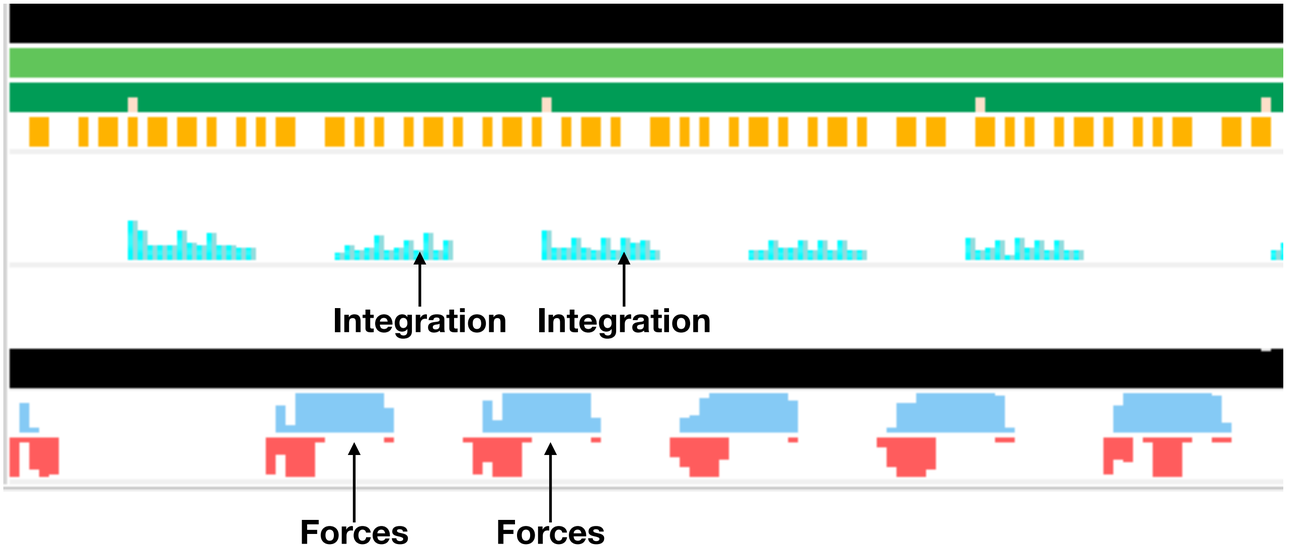
**Enhanced sampling methods: additional forces, grid potentials, collective variables**

# Original GPU-Offload Scheme

*CPU-bound on Volta and beyond*

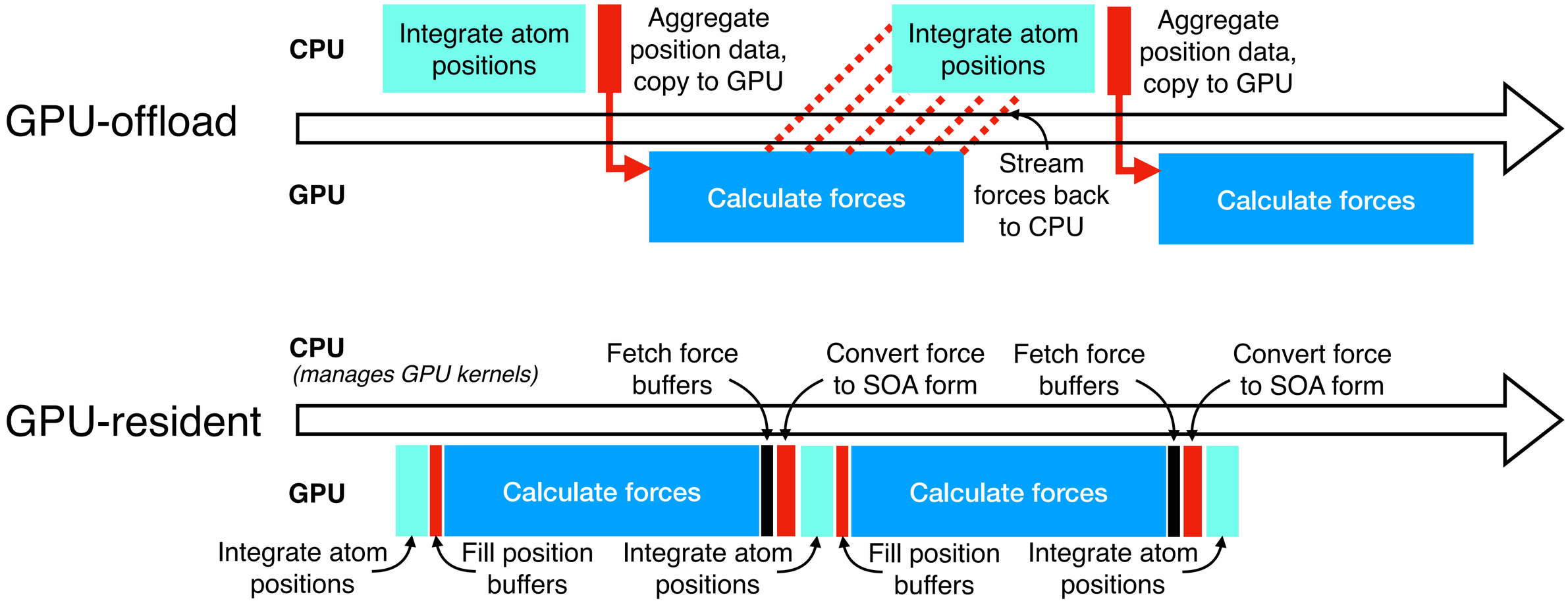
- GPUs had become **much** faster!
- Attempt to overlap CPU and GPU causes performance bottleneck
- Unable to fully utilize GPU with offload approach

Profile using **Nsight Systems** with NVTX tags to trace execution of CPU kernels:



# New GPU-Resident Scheme

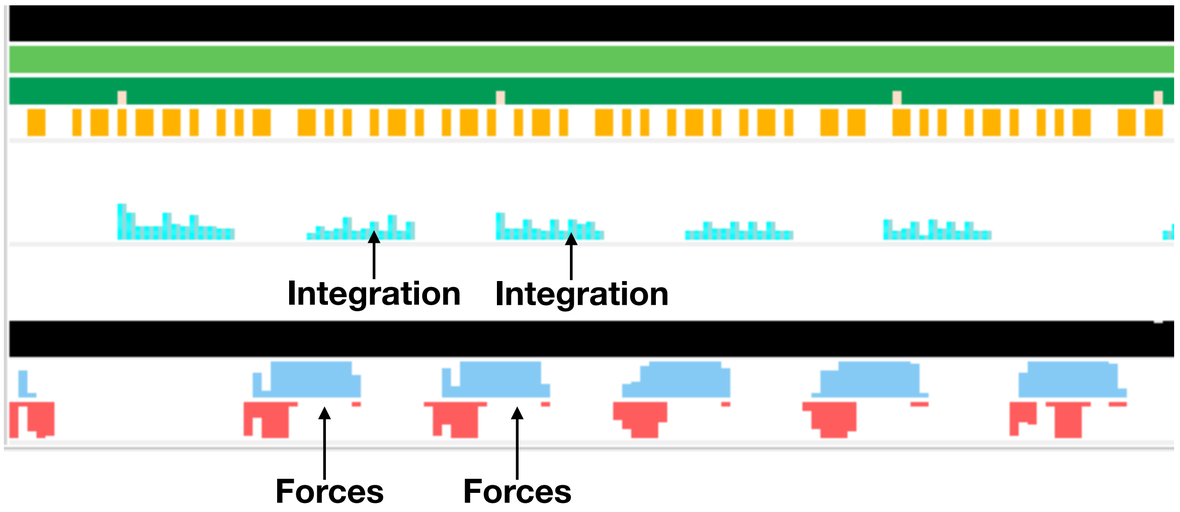
*Move integrator to GPU and maintain data between time steps*



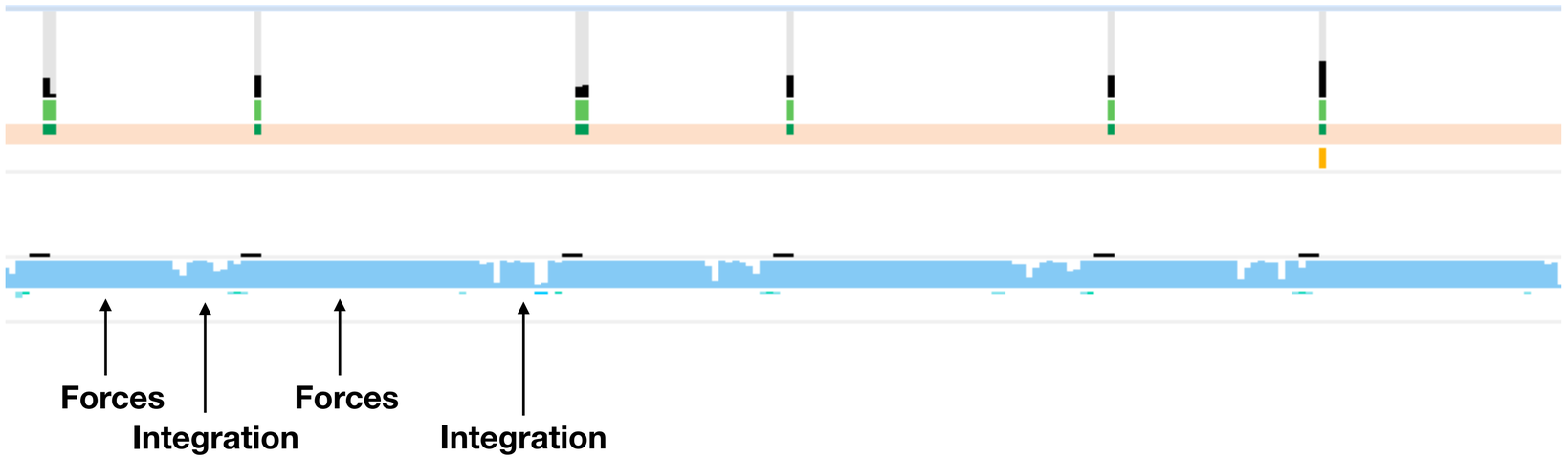
# New GPU-Resident Scheme

*Profiling shows new scheme fully utilizes GPU, no more CPU bottleneck*

Before (GPU-offload):



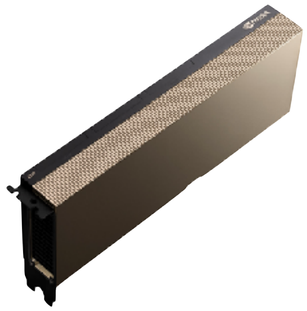

After (GPU-resident):



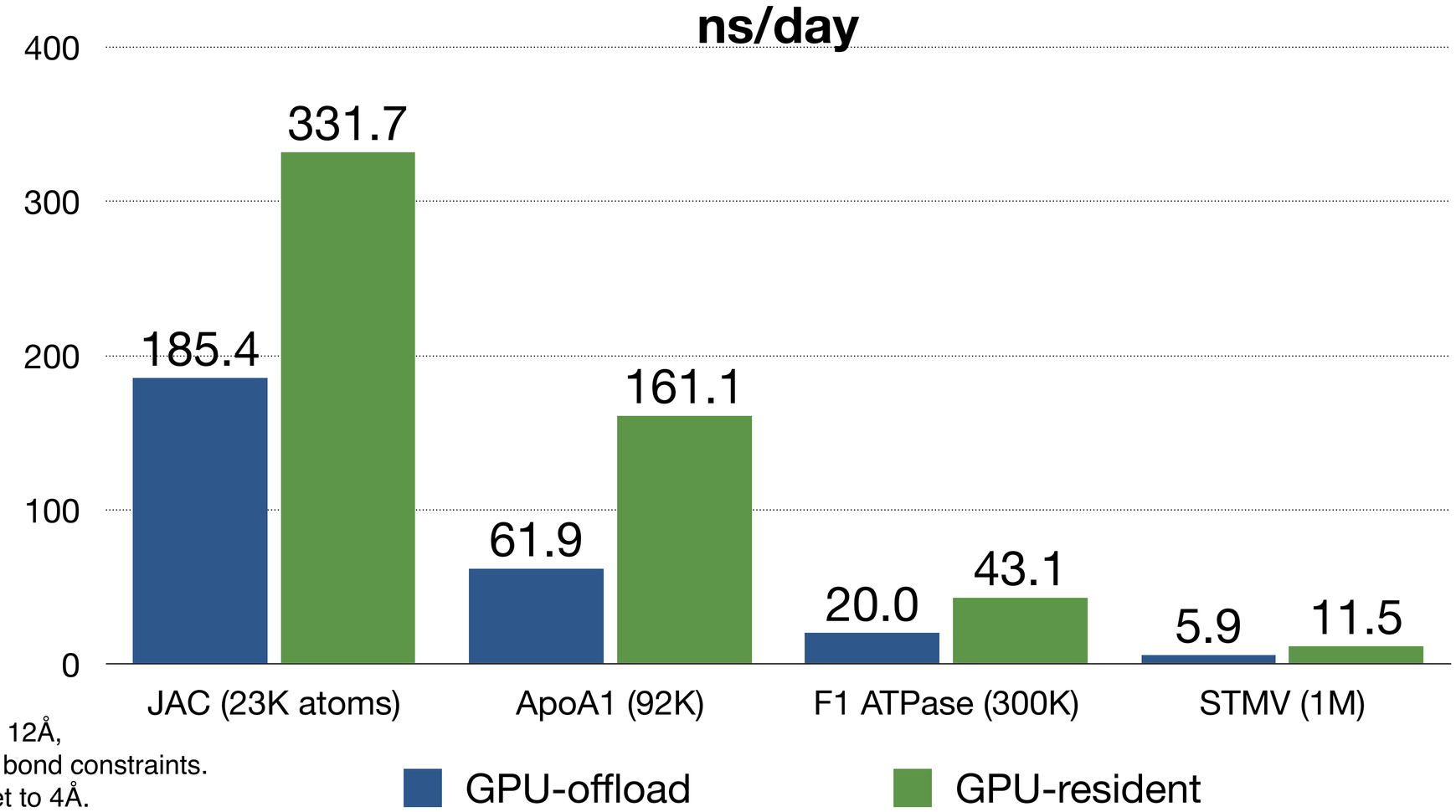
# New GPU-Resident Scheme

*Performance for constant energy (NVE) simulation on single GPU (Aug 2020)*


**Intel Xeon Gold 6134 @ 3.2 GHz**



**NVIDIA A100-PCIe**




**Simulation details:**
NVE, CHARMM force field, cutoff distance 12Å,
MTS with 2fs time step and 4fs PME, rigid bond constraints.
Performance tuning parameter “margin” set to 4Å.
https://www.ks.uiuc.edu/Research/namd/benchmarks/

# Why is NAMD adopting SYCL / DPC++?

- Support upcoming exascale computers: ANL Aurora (Intel)
- SYCL / 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
  - Single source and single compiler (DPC++) for host and device code
  - Vendor-neutral language and library solution

# How does SYCL differ from CUDA?

- Use of modern C++
  - Kernels defined as unnamed lambda expressions
  - Error-handling with try–catch block
- Design decisions in SYCL and OpenCL
  - SYCL work queue is analogous to CUDA stream, but defaults to out-of-order execution
  - Must specify accessor functions to enable SYCL kernels to access device buffers
  - Permit flexible vector width for performance portability across different hardware

# Design decisions for porting NAMD

- **Extend NAMD without disrupting current GPU support**
  - Use preprocessor switches to isolate DPC++ extensions from existing code
- **Leverage existing GPU kernels and data structures**
  - Translate CUDA kernels to DPC++ and copy supporting data structures and kernel management infrastructure into DPC++ versions
- **Add support incrementally, guided by Amdahl's Law, to accelerate most computationally expensive parts first**
  - Begin by porting short-range non-bonded force kernels
  - Continue with PME and bonded force kernels

*Mirrors order of original CUDA development*

# NAMD has a **LOT** of CUDA code

- Eventually develop SYCL support for everything, including GPU-resident version
- Start porting from **stable code base** with GPU-offload (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 |

# Overall porting strategy

- We employed a divide-and-conquer strategy, using preprocessor switches to decouple components in the CUDA code
  - Significantly reduces development and debugging complexity
- Separated components
  - Short-range non-bonded force & device utilities
  - Bonded force
  - PME (Particle-Mesh Ewald) — requires FFT
- Utilized supplemental libraries from oneAPI
  - oneDPL for C++17 parallel STL *reduce, shuffle, atomic_ref, sort* and *scan* replaces CUB library
  - oneMKL FFT replaces cuFFT library

# Faster porting with code conversion tool

- Utilized Intel's DPC++ Compatibility Tool for faster development
- Started with converting the CUDA implementation
  - Saves > 80% of code porting effort
  - For example, threadIdx.x → ndItem.get_local_id(2)
- Provides a good guide to practice DPC++ syntax

# Considerations for short-range non-bonded force kernels

- Create new "Dpcpp"-prefixed versions of relevant files
- Rename object classes from "Cuda" to "Dpcpp"
- Change certain fundamental data types, e.g., CUDA *float4* to SYCL *sycl::float4*
- CUDA warp-level intrinsics require SYCL subgroups using *reduce, shuffle, atomic_ref*
- CUDA texture memory for force parameters and interaction lookup table
  - Read values from global memory
  - Linear interpolation explicitly performed from lookup table
- Non-bonded kernel calculates in single precision, later summation of virial and potential energy uses double precision

# Considerations for bonded force and PME kernels

- Create new "Dpcpp"-prefixed versions of relevant files
- Rename object classes from "Cuda" to "Dpcpp"
- CUDA implementation maintains bit flag for the five different bonded force types, indicating GPU or CPU kernels — makes debugging easier
- PME has two GPU-based code paths — single-node and multi-node
- Single-node required extra work to replace cuFFT with oneMKL FFT
  - oneMKL FFT does not provide FFTW-compatible interface
  - Must account for data buffer strides for real-to-complex forward FFT and pad input grid to SIMD vector length, output similarly padded for backward FFT
  - Avoid modifying surrounding code by introducing additional data buffering kernels

# Debugging challenges

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

# Create an array debug utility

- Code porting mostly involves changing the syntax and library calls
  - Almost all of the algorithms and results remain the same
  - Most host-side code remains intact
- Add utility to capture device kernel buffers across languages (CUDA, SYCL)
  - Buffers from reference language are written to a file before and after each kernel call
  - Read reference data files to compare kernel buffers for development language
- Results for **array debug utility**
  - Provide easy-to-add macros around kernel calls
  - Determine earliest code location and element index for difference between device kernel buffers

# Validating SYCL port of GPU-offload kernels

- Presently lacking performance results
  - Intel GPU performance is not meaningful until we can access Xe-HPC (Ponte Vecchio)
  - Codeplay DPC++ compiler does not provide oneDPL or oneMKL FFT libraries
- Determine correctness through short trajectory runs
  - Run MD simulation at constant energy
  - Long trajectories will diverge due to non-determinism in the order of operation of parallel calculation and non-associativity of floating point addition
  - Compare total energy values with a reference run up to 500 steps

# Validating SYCL port of GPU-offload kernels

- Maximum relative error in total energy for 500-step constant energy simulation
- CPU-only run shows lower error due to double precision compared to mixed-precision GPU runs

| 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 |

# Future work

- Continue porting efforts with GPU-resident version of NAMD
- Optimize for Intel Xe-HPC (Ponte Vecchio) GPU
  - Multi-node runs on Cray Slingshot using GPU-offload version
  - Single-node runs using GPU-resident version
- Improve support for mainstream commodity Intel GPUs
  - Transform double precision reduction into two single precision values
  - Using Kahan summation or compensated summation algorithm to preserve low order bits
- Merge SYCL and CUDA versions into unified GPU code path

# Acknowledgments


- NAMD GPU development: David Clark (NVIDIA), Julio Maia (AMD), John Stone (UIUC); past work by Jim Phillips (UIUC), Antti-Pekka Hynninen (ORNL), Peng Wang (NVIDIA)
- SYCL/DPC++ porting: Tareq Malas (Intel), Jaemin Choi (UIUC), Mike Brown (Intel)
- ANL Aurora Early Science Program of the Argonne Leadership Computing Facility
- Intel funding through UIUC oneAPI Center of Excellence
- NIH Grant P41-GM104601


NIH Center for Macromolecular Modeling and Bioinformatics
Beckman Institute, University of Illinois at Urbana-Champaign