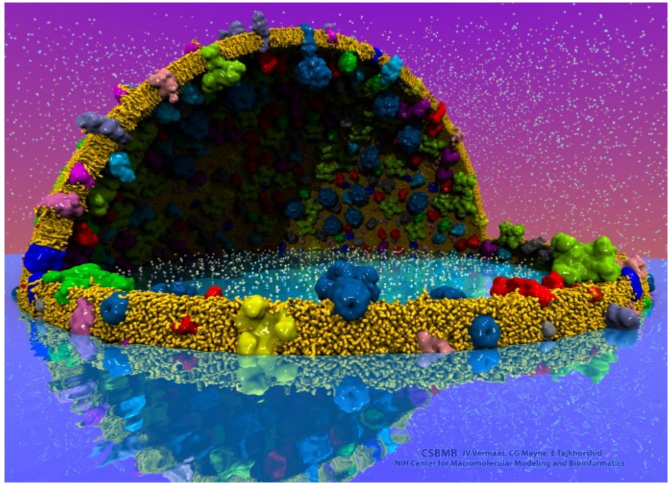
HiHAT: A New Way Forward
for Hierarchical Heterogeneous Asynchronous Tasking

John E. Stone, University of Illinois

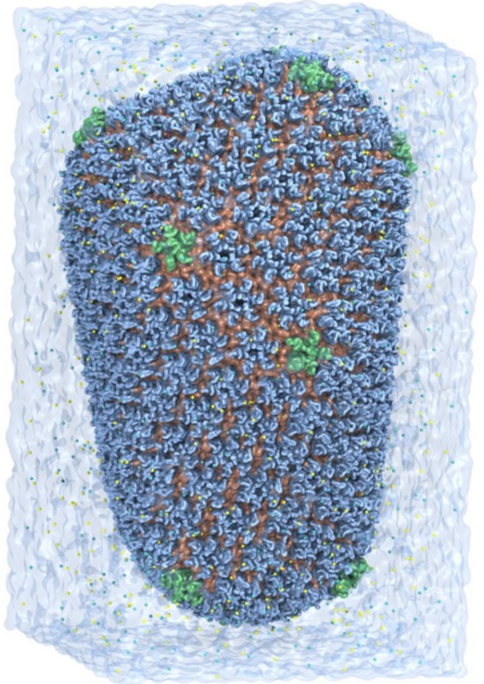
VMD adaptation for HiHAT Proof of Concept implementations

VMD – "Visual Molecular Dynamics"

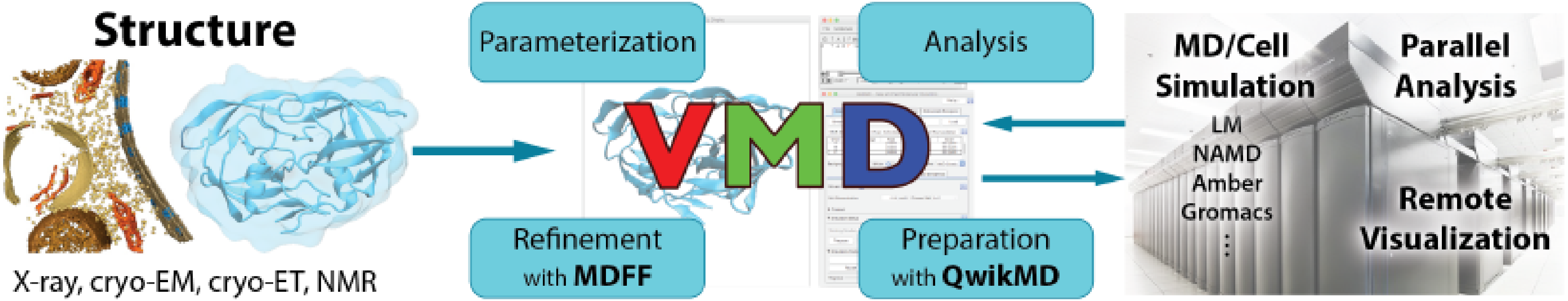
- Visualization and analysis of:
  - Molecular dynamics simulations
  - Lattice cell simulations
  - Quantum chemistry calculations
  - Sequence information
- User extensible scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/



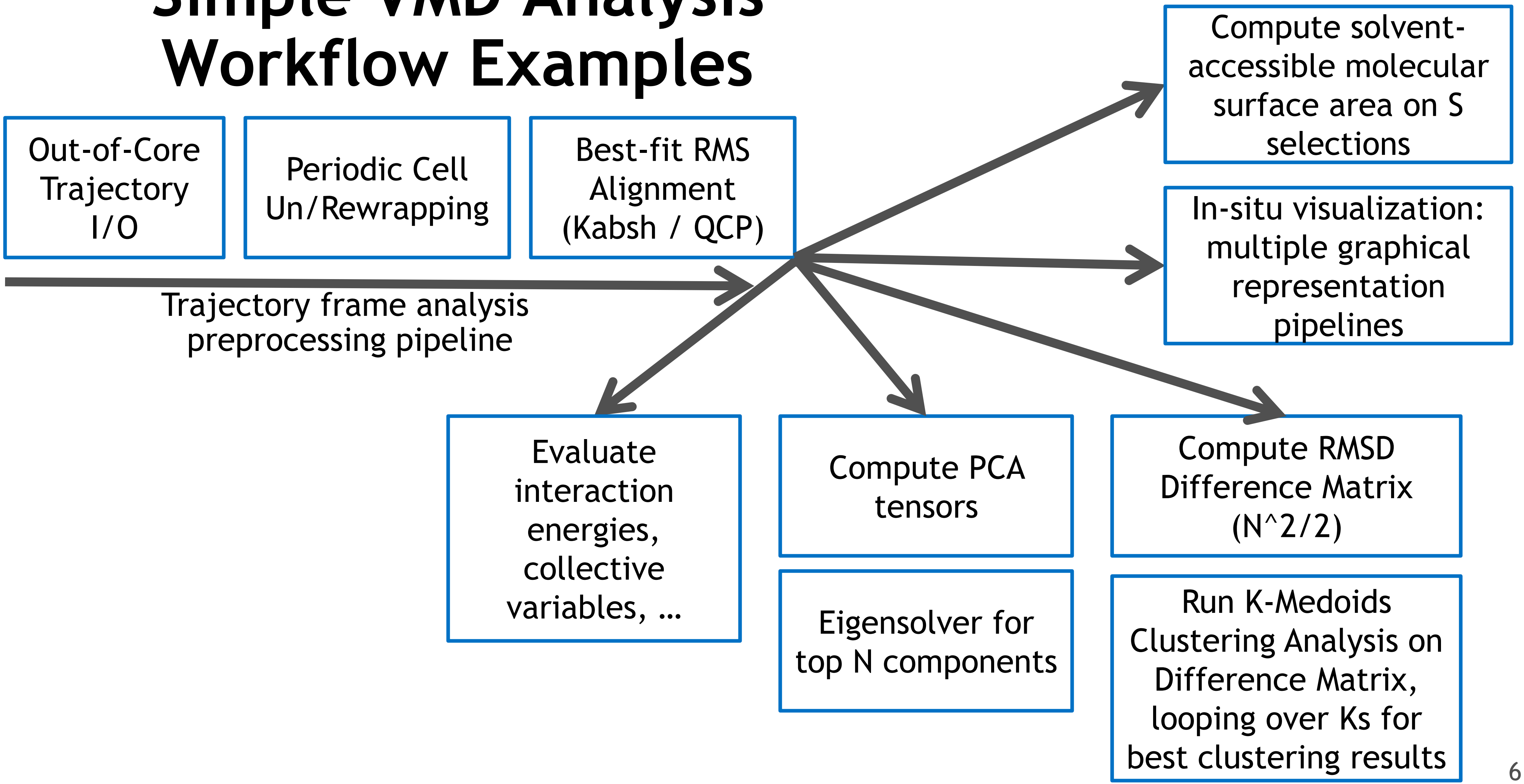
Cell-Scale Modeling



MD Simulation

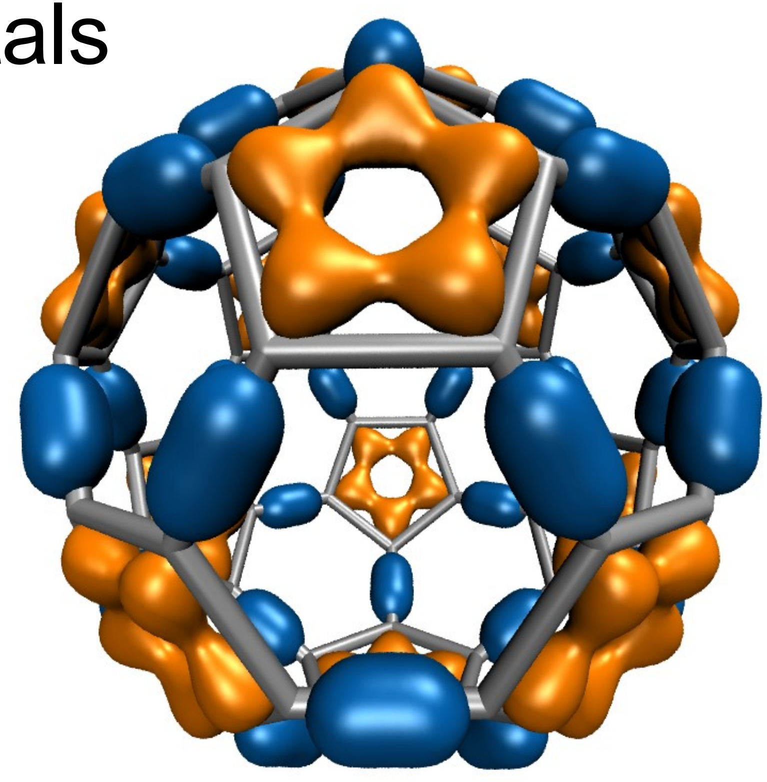


Simple VMD Analysis Workflow Examples



VMD: Visualization of Molecular Orbitals

- Visualization of MOs aids in understanding the chemistry of molecular system
- MO spatial distribution is correlated with probability density for electron(s)
- **Animation** of (classical mechanics) molecular dynamics trajectories provides insight into simulation results
  - To do the same for QM or QM/MM simulations MOs must be computed at **10 FPS** or more
  - **Large GPU speedups** over existing tools makes this possible!



C_{60}

High Performance Computation and Interactive Display of Molecular Orbitals on GPUs and Multi-core CPUs. J. E. Stone, J. Saam, D. Hardy, K. Vandivort, W. Hwu, K. Schulten, *2nd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU-2), ACM International Conference Proceeding Series*, volume 383, pp. 9-18, 2009.

Adapting VMD for HiHat PoC Implementations

- VMD QM molecular orbital (MO) viz. algorithms
  - Existing code targets both many-core CPUs and GPUs
  - Incrementally adapt for **HiHat PoC data movement**, and **tasking** APIs as implementations progress
  - Algorithm variants map different data to different memory systems
  - Proxy for other algorithms in VMD, Lattice Microbes, that have more complex data movement needs
  - Adaptation of GPU code to HiHat PoC APIs should be largely non-invasive
  - Standalone code variant already exists from past work, easy to share with others as an example

MO GPU Parallel Decomposition

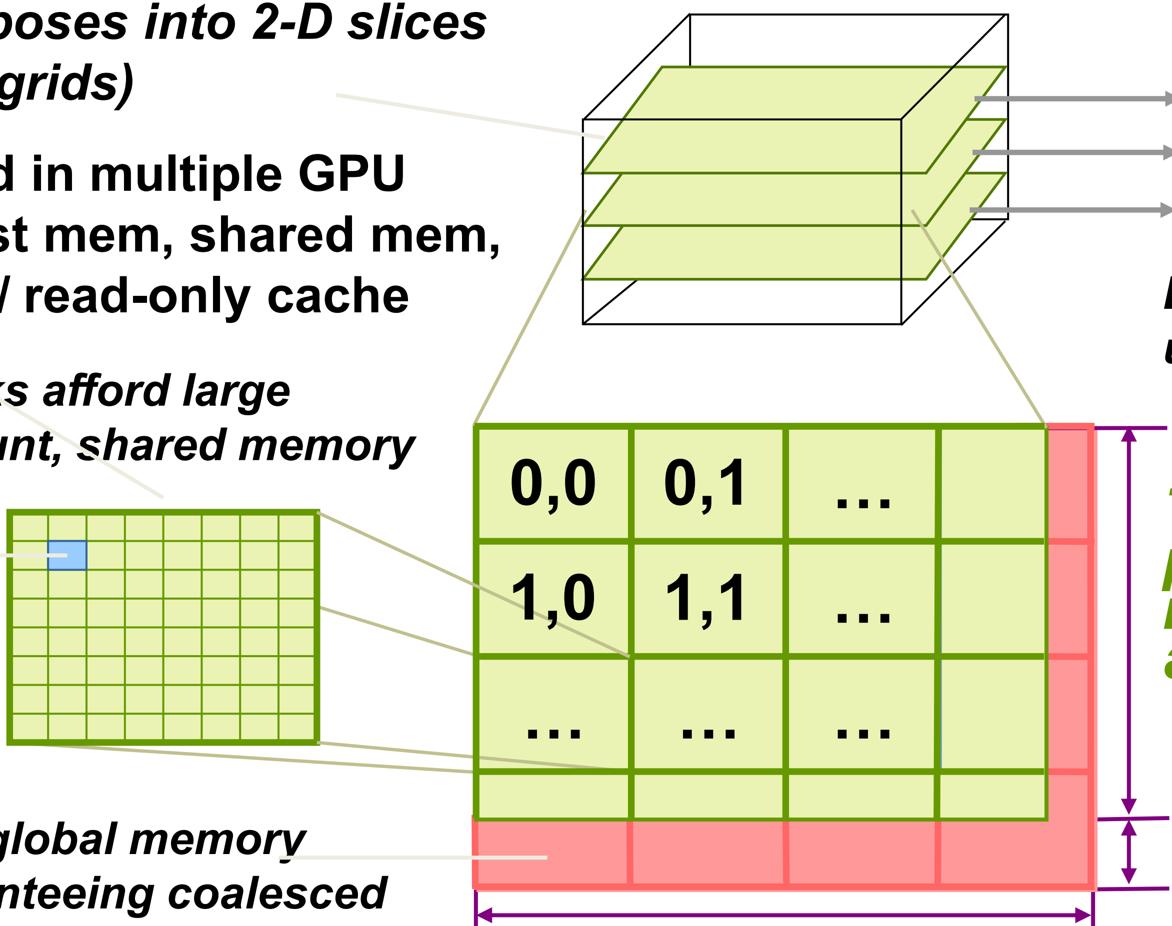
MO 3-D lattice decomposes into 2-D slices (CUDA grids)

Key data are stored in multiple GPU memory systems, const mem, shared mem, global mem, and w/ read-only cache

Small 8x8 thread blocks afford large per-thread register count, shared memory

Each thread computes one MO lattice point.

Padding optimizes global memory performance, guaranteeing coalesced global memory accesses



…

GPU 2

GPU 1

GPU 0

Lattice computed using multiple GPUs

Threads producing results that are used

Threads producing results that are discarded

Grid of thread blocks

MO Kernel for One Grid Point (Naive C)

```
…
for (at=0; at<numatoms; at++) {
    int prim_counter = atom_basis[at];
    calc_distances_to_atom(&atompos[at], &xdist, &ydist, &zdist, &dist2, &xdiv);
    for (contracted_gto=0.0f, shell=0; shell < num_shells_per_atom[at]; shell++) {
        int shell_type = shell_symmetry[shell_counter];
        for (prim=0; prim < num_prim_per_shell[shell_counter];  prim++) {
            float exponent          = basis_array[prim_counter       ];
            float contract_coeff = basis_array[prim_counter + 1];
            contracted_gto += contract_coeff * expf(-exponent*dist2);
            prim_counter += 2;
        }
        for (tmpshell=0.0f, j=0, zdp=1.0f; j<=shell_type; j++, zdp*=zdist) {
            int imax = shell_type - j;
            for (i=0, ydp=1.0f, xdp=pow(xdist, imax); i<=imax; i++, ydp*=ydist, xdp*=xdiv)
                tmpshell += wave_f[ifunc++] * xdp * ydp * zdp;
        }
        value += tmpshell * contracted_gto;
        shell_counter++;
    }
} …..
```

Loop over atoms

Loop over shells

Loop over primitives: largest component of runtime, due to **expf()**

Loop over angular momenta

(unrolled in real code)

Adapting VMD MO Algorithms for **HiHat Data Movement PoC**

- Three different CUDA kernel variants, different approaches to use of GPU memory systems
- First HiHat PoC port will be “L1 Cache” algorithm variant favored by “Fermi” and “Volta” GPUs
  - Simplest use of data movement APIs, minimal changes to original code
- Ports of algorithms that use GPU constant memory and shared memory tiling next
- Try managed memory variants, NVLink performance, ...