GPU-Accelerated Molecular Visualization and Analysis with VMD

John E. Stone

Theoretical and Computational Biophysics Group

Beckman Institute for Advanced Science and Technology

University of Illinois at Urbana-Champaign

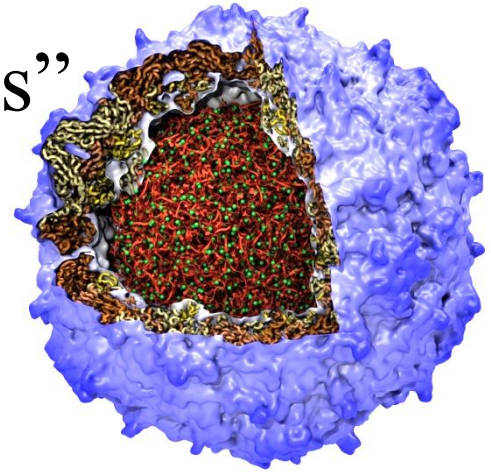
http://www.ks.uiuc.edu/Research/gpu/

Midwest Theoretical Chemistry Conference

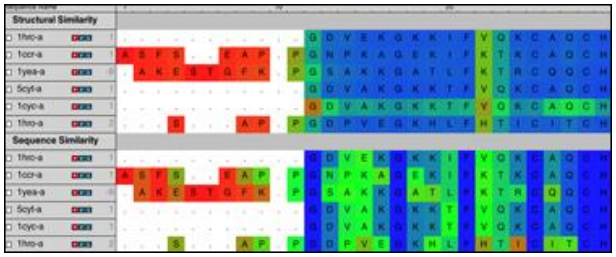
Urbana, Illinois, May 31, 2013

VMD – "Visual Molecular Dynamics"

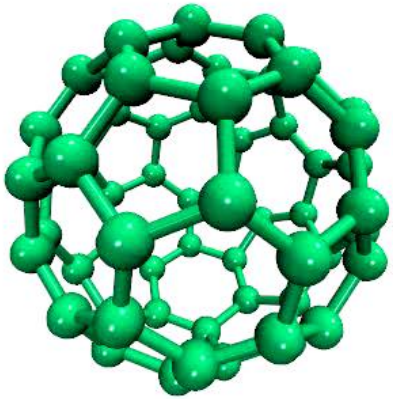
- Visualization and analysis of:
  - molecular dynamics simulations
  - quantum chemistry calculations
  - particle systems and whole cells
  - sequence data
- User extensible w/ scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/



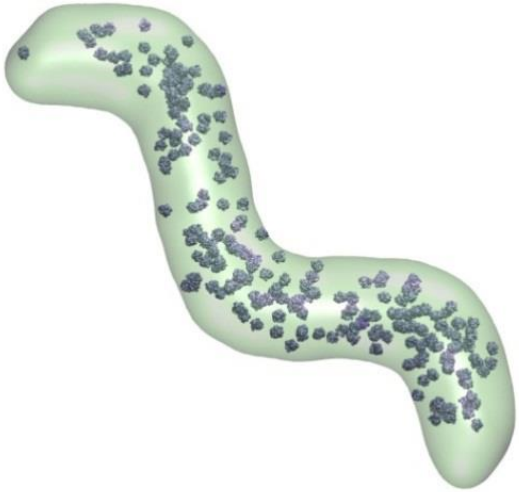
Poliovirus



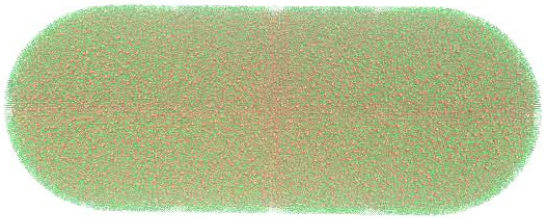
Ribosome Sequences



Electrons in
Vibrating Buckyball



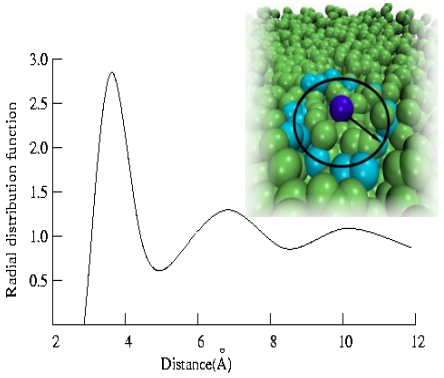
Cellular Tomography,
Cryo-electron Microscopy

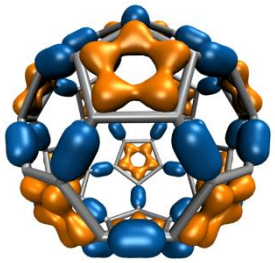


Whole Cell Simulations

GPU Accelerated Trajectory Analysis and Visualization in VMD

GPU-Accelerated Feature	Peak speedup vs. single CPU core
Molecular orbital display	120x
Radial distribution function	92x
Electrostatic field calculation	44x
Molecular surface display	40x
Ion placement	26x
MDFF density map synthesis	26x
Implicit ligand sampling	25x
Root mean squared fluctuation	25x
Radius of gyration	21x
Close contact determination	20x
Dipole moment calculation	15x



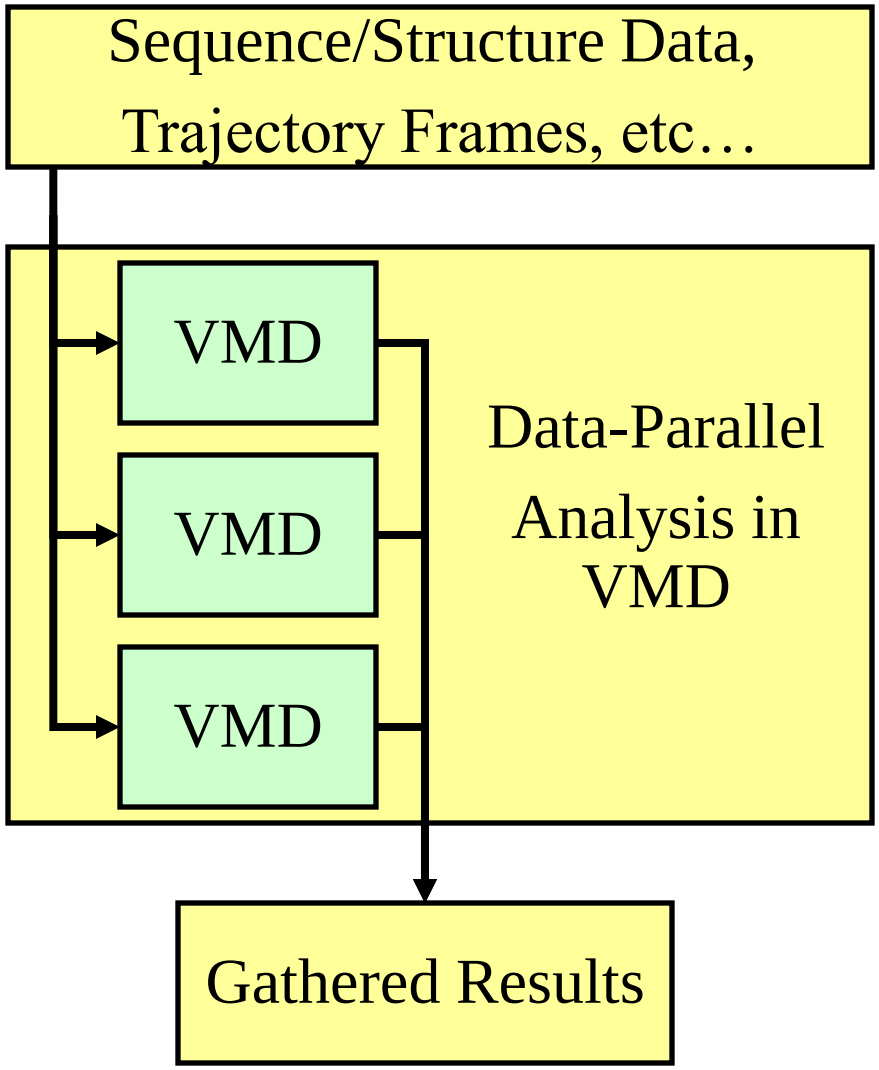


Ongoing VMD GPU Development

- New CUDA kernels for common analysis tasks
- Increase memory efficiency of CUDA kernels for large structures
- Improving CUDA performance for VMD trajectory analysis calculations w/ MPI:
  - GPU-accelerated commodity clusters
  - GPU-accelerated Cray XK7 supercomputers:
    NCSA Blue Waters, ORNL Titan

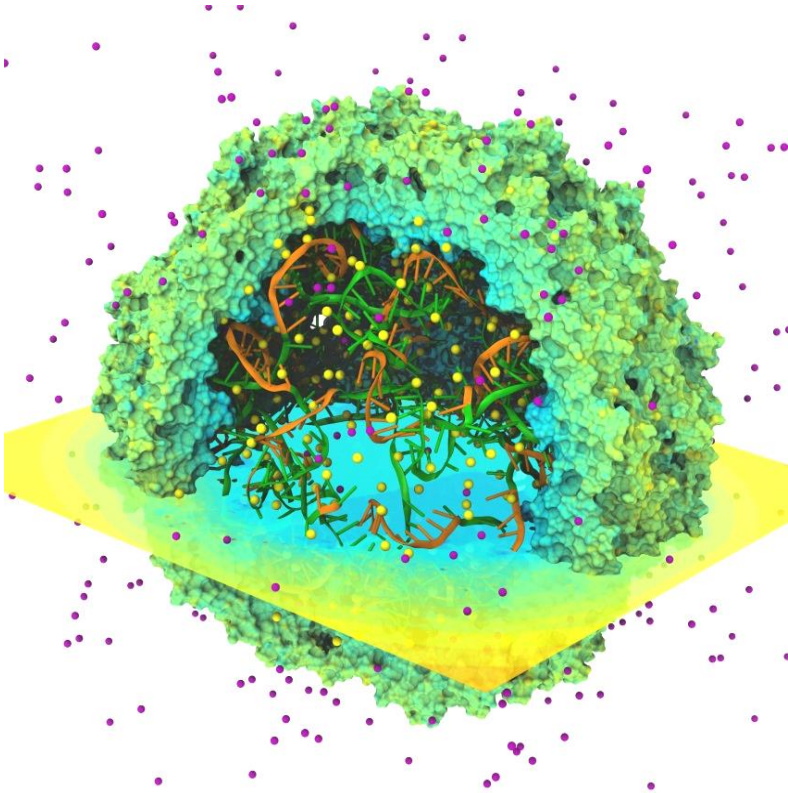
VMD for Demanding Analysis Tasks
Parallel VMD Analysis w/ MPI

- Addresses computing requirements beyond desktop
- Compute time-averaged electrostatic fields, MDFF quality-of-fit, etc.
- Parallel rendering, movie making
- User-defined parallel reduction operations, data types
- Dynamic load balancing
- **Parallel I/O: 109 GB/sec on 512 nodes of Blue Waters**
- **Supports GPU-accelerated clusters and supercomputers**



Time-Averaged Electrostatics Analysis on Energy-Efficient GPU Cluster

- **1.5 hour** job (CPUs) reduced to **3 min** (CPUs+GPU)
- Electrostatics of thousands of trajectory frames averaged
- Per-node power consumption on NCSA "AC" GPU cluster:
  - CPUs-only: 299 watts
  - CPUs+GPUs: 742 watts
- GPU Speedup: **25.5x**
- Power efficiency gain: **10.5x**



Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters. J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J. Stone, J. Phillips. *The Work in Progress in Green Computing*, pp. 317-324, 2010.

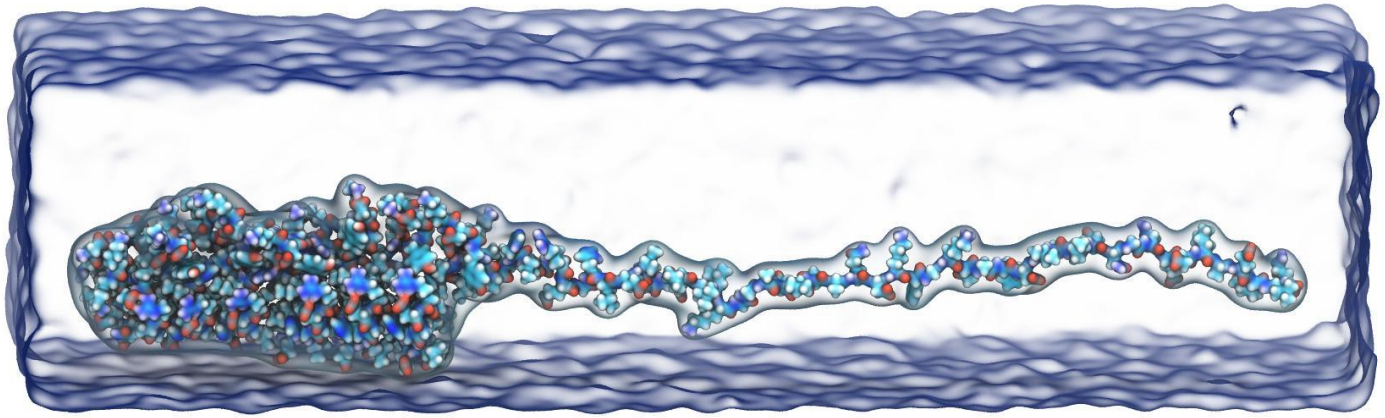
Time-Averaged Electrostatics Analysis on NCSA Blue Waters

NCSA Blue Waters Node Type	**Seconds per trajectory frame for one compute node**
Cray XE6 Compute Node: 32 CPU cores (2xAMD 6200 CPUs)	9.33
Cray XK6 GPU-accelerated Compute Node: 16 CPU cores + **NVIDIA X2090 (Fermi) GPU**	2.25
Speedup for GPU XK6 nodes vs. CPU XE6 nodes	**GPU nodes are 4.15x faster overall**
Early tests on XK7 nodes indicate MSM is becoming CPU-bound with the Kepler K20X GPU **Performance is not much faster (yet) than Fermi X2090** **May need to move spatial hashing and other algorithms onto the GPU.**	**In progress….**

Preliminary performance for VMD time-averaged electrostatics w/ Multilevel Summation Method on the NCSA Blue Waters Early Science System

VMD "QuickSurf" Representation

- Displays continuum of structural detail:
  - All-atom models
  - Coarse-grained models
  - Cellular scale models
  - Multi-scale models: All-atom + CG, Brownian + Whole Cell
  - Smoothly variable between full detail, and reduced resolution representations of very large complexes

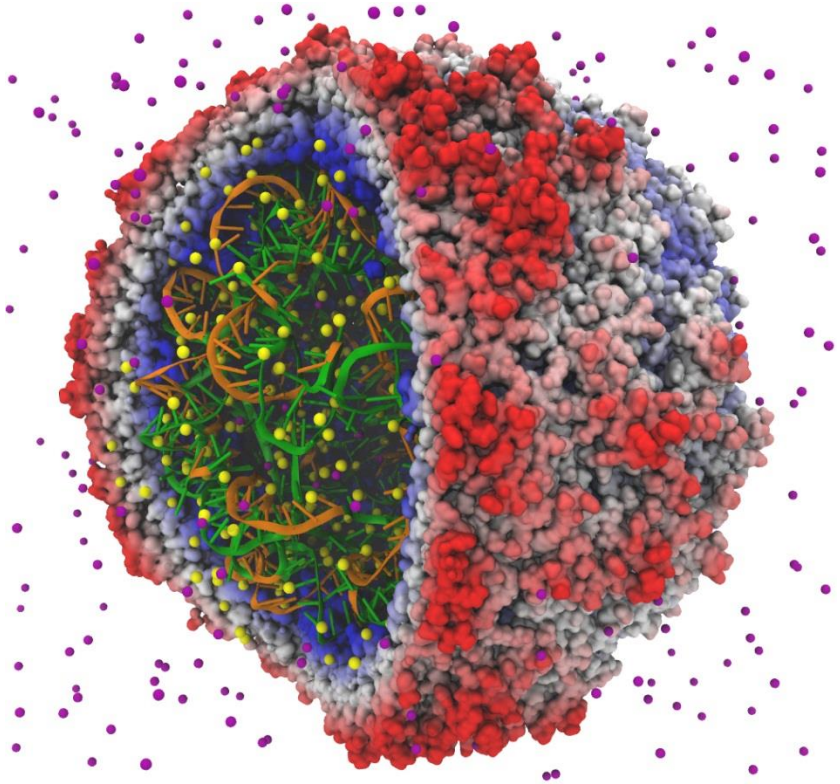


Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.

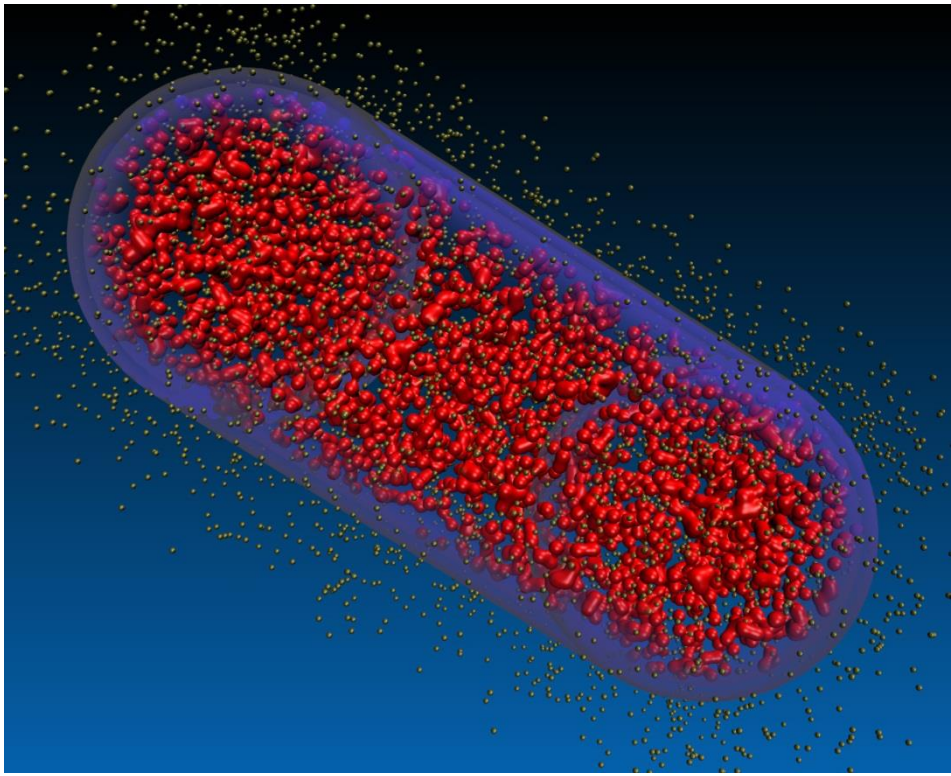
M. Krone, J. E. Stone, T. Ertl, K. Schulten. *EuroVis Short Papers*, pp. 67-71, 2012

VMD “QuickSurf” Representation

- Uses multi-core CPUs and GPU acceleration to enable **smooth real-time animation** of MD trajectories
- Linear-time algorithm, scales to millions of particles, as limited by memory capacity

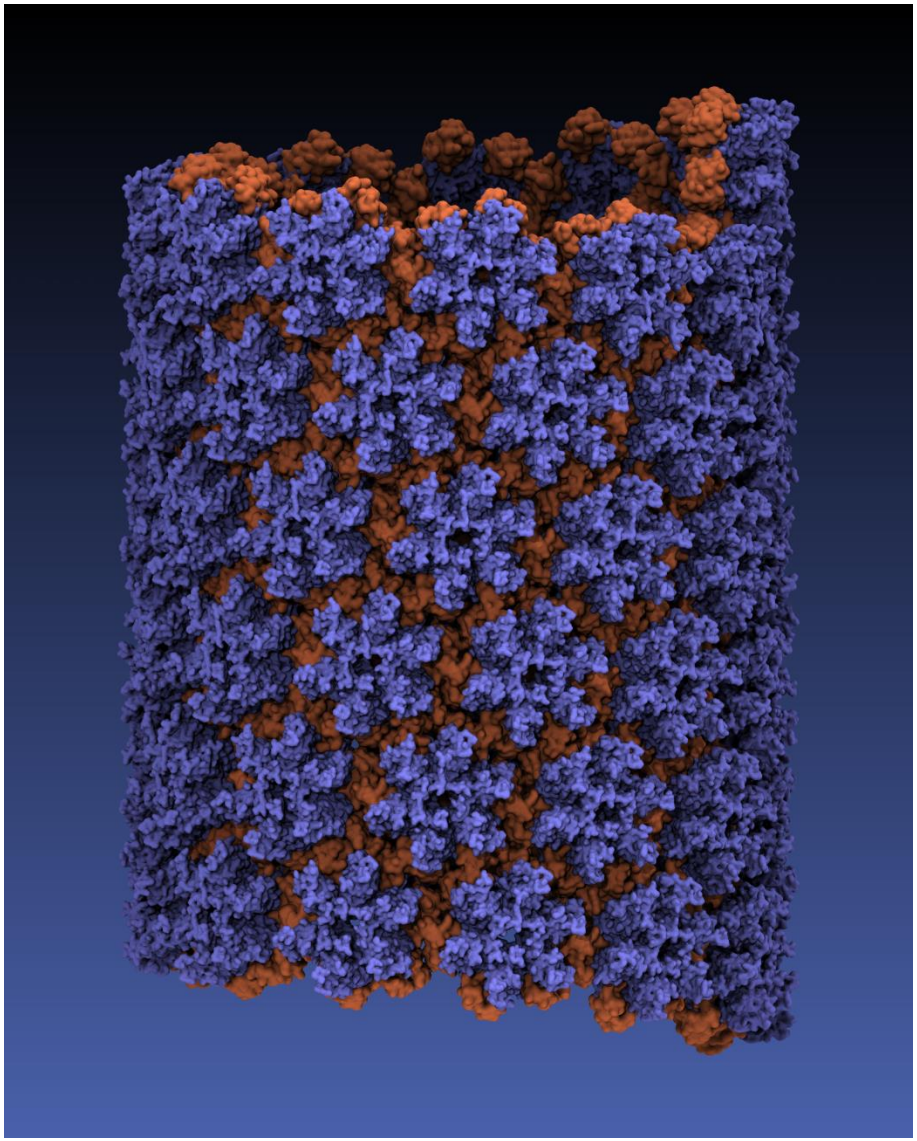


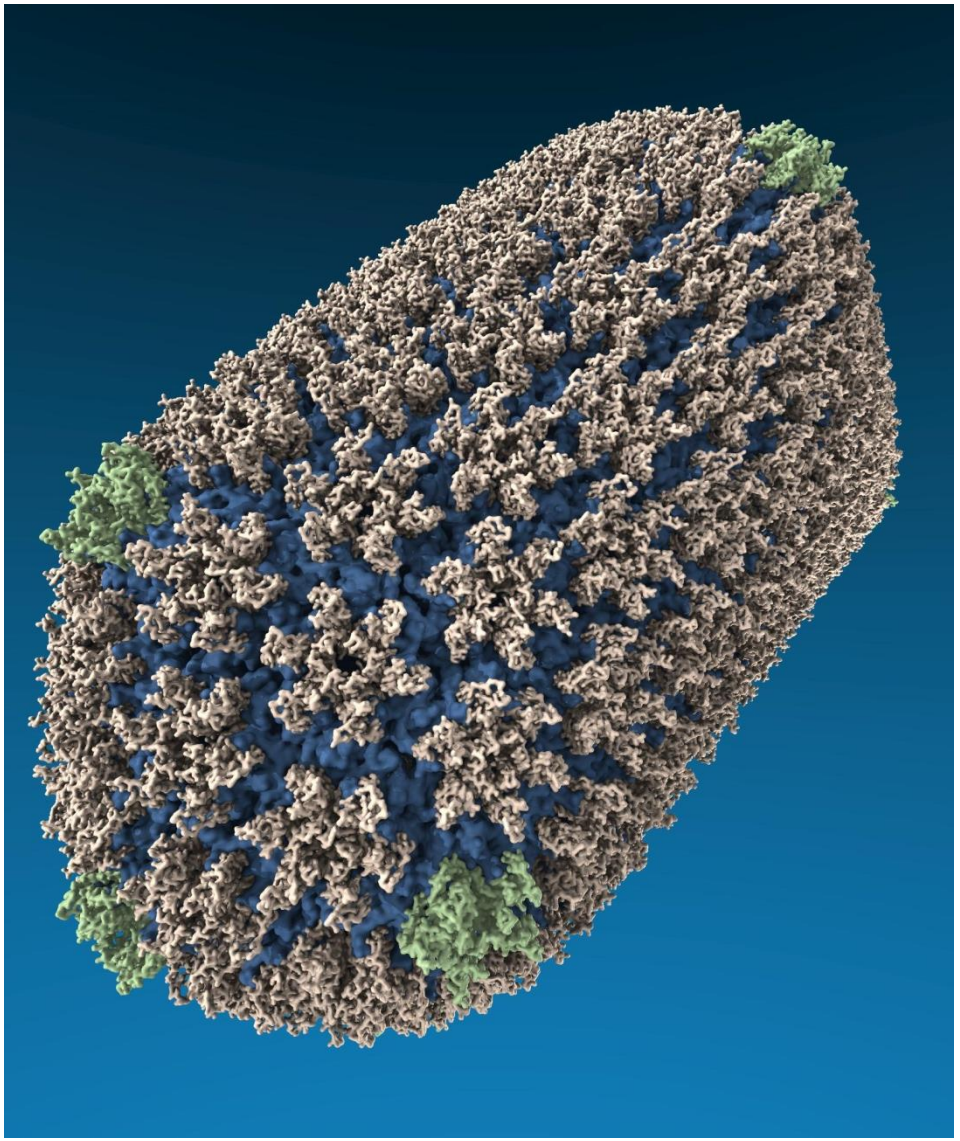
Satellite Tobacco Mosaic Virus



Lattice Cell Simulations

VMD “QuickSurf” Representation





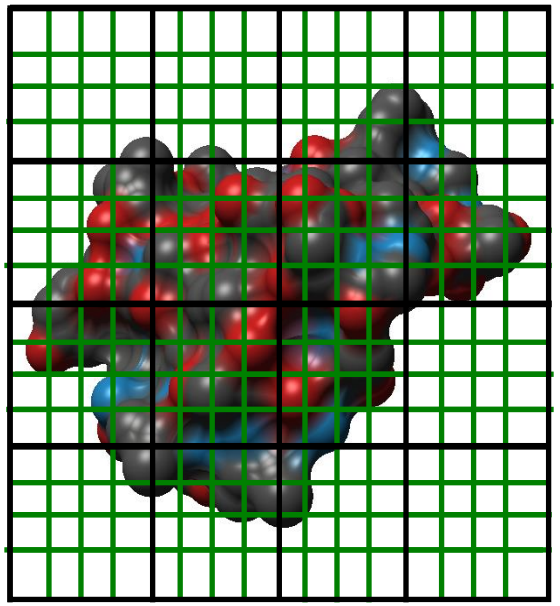
All-atom HIV capsid simulations

QuickSurf Algorithm Overview

- Build spatial acceleration data structures, optimize data for GPU
- Compute 3-D density map, 3-D volumetric texture map:

$$\rho(\vec{r}; \vec{r}_1, \vec{r}_2, \ldots, \vec{r}_N) = \sum_{i=1}^{N} e^{\frac{-|\vec{r}-\vec{r}_i|^2}{2\alpha^2}}$$

- Extract isosurface for a user-defined density value



3-D density map lattice, spatial acceleration grid, and extracted surface

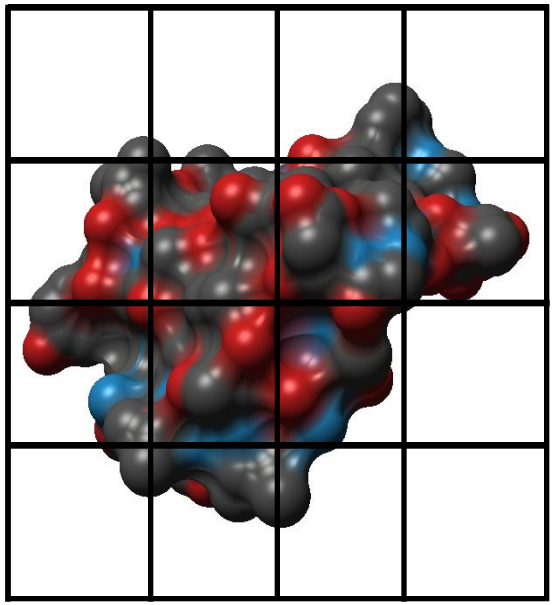
Spatial Hashing Algorithm Steps/Kernels

1) Compute bin index for each atom, store to memory w/ atom index

2) **Sort** list of bin and atom index tuples (1) by bin index **(thrust kernel)**

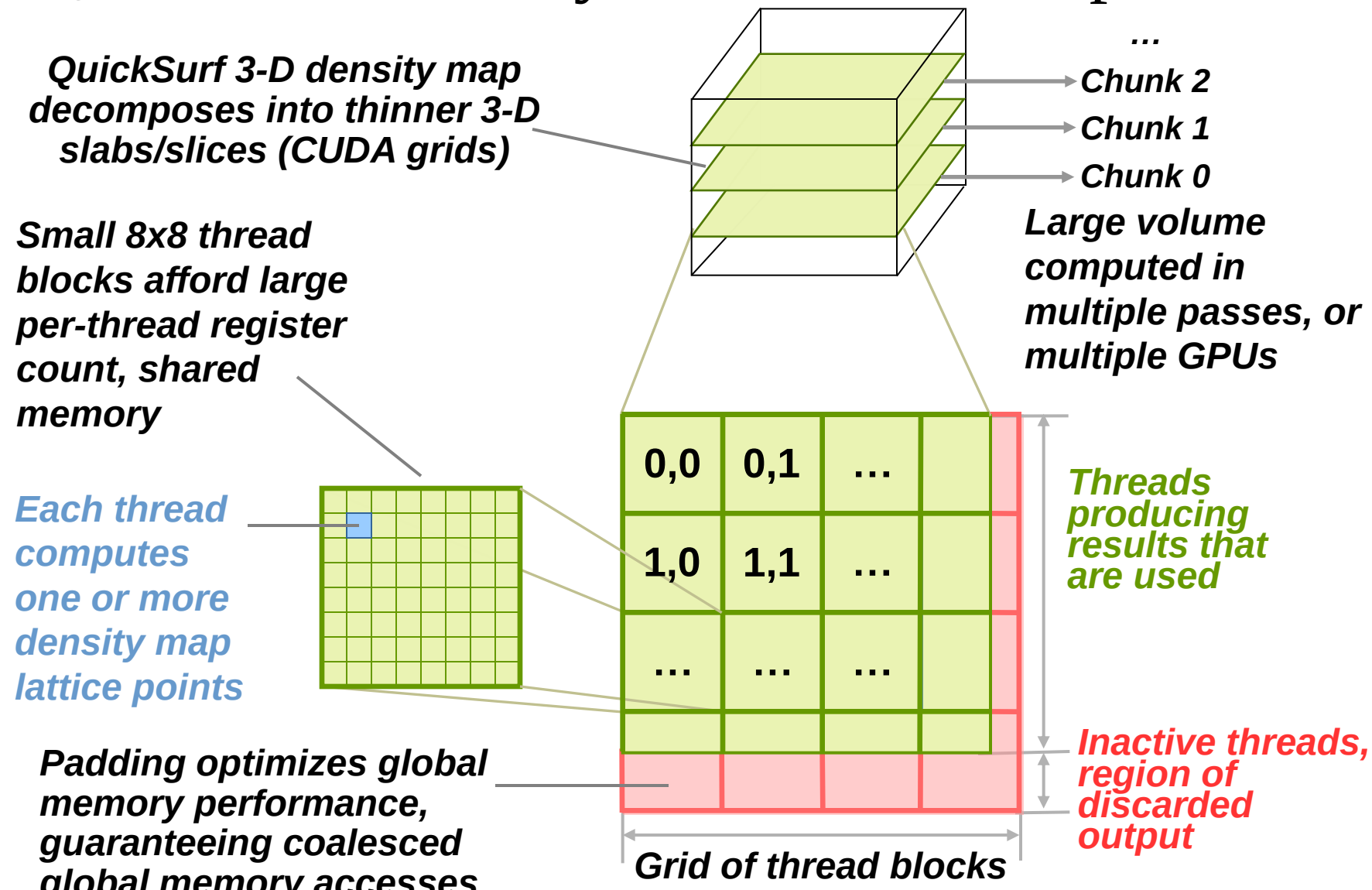
3) Count atoms in each bin (2) using a **parallel prefix sum, aka *scan***, compute the destination index for each atom, store per-bin starting index and atom count **(thrust kernel)**

4) Write atoms to the output indices computed in (3), and we have completed the data structure



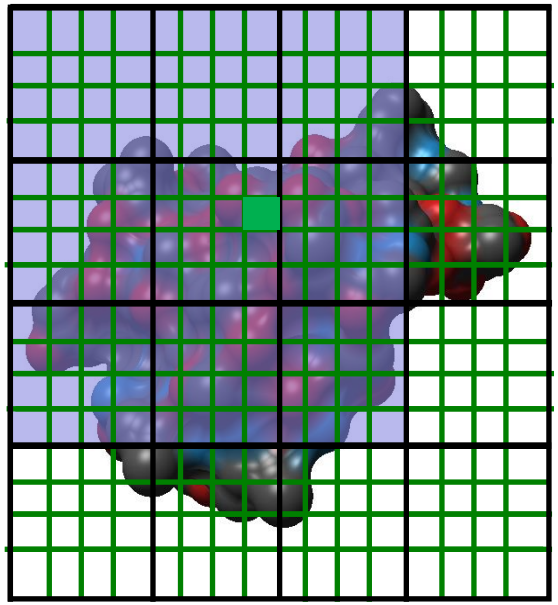
QuickSurf uniform grid spatial subdivision data structure

QuickSurf Density Parallel Decomposition



QuickSurf Density Map Algorithm

- Spatial acceleration grid cells are sized to match the cutoff radius for the exponential, beyond which density contributions are negligible
- Density map lattice points computed by summing density contributions from particles in 3x3x3 grid of neighboring spatial acceleration cells
- Volumetric texture map is computed by summing particle colors normalized by their individual density contribution



3-D density map lattice point and the neighboring spatial acceleration cells it references

QuickSurf Density Map Kernel Snippet…

```
for (zab=zabmin; zab<=zabmax; zab++) {
  for (yab=yabmin; yab<=yabmax; yab++) {
    for (xab=xabmin; xab<=xabmax; xab++) {
      int abcellidx = zab * acplanesz + yab * acncells.x + xab;
      uint2 atomstartend = cellStartEnd[abcellidx];
      if (atomstartend.x != GRID_CELL_EMPTY) {
        for (unsigned int atomid=atomstartend.x; atomid<atomstartend.y; atomid++) {
          float4 atom = sorted_xyzr[atomid];
          float dx = coorx - atom.x;          float dy = coory - atom.y;        float dz = coorz - atom.z;
          float dxy2 = dx*dx + dy*dy;
          float r21 = (dxy2 + dz*dz) * atom.w;
          densityval1 += exp2f(r21);
          /// Loop unrolling and register tiling benefits begin here……
          float dz2 = dz + gridspacing;
          float r22 = (dxy2 + dz2*dz2) * atom.w;
          densityval2 += exp2f(r22);
          /// More loop unrolling ….
```

QuickSurf Performance
GeForce GTX 580

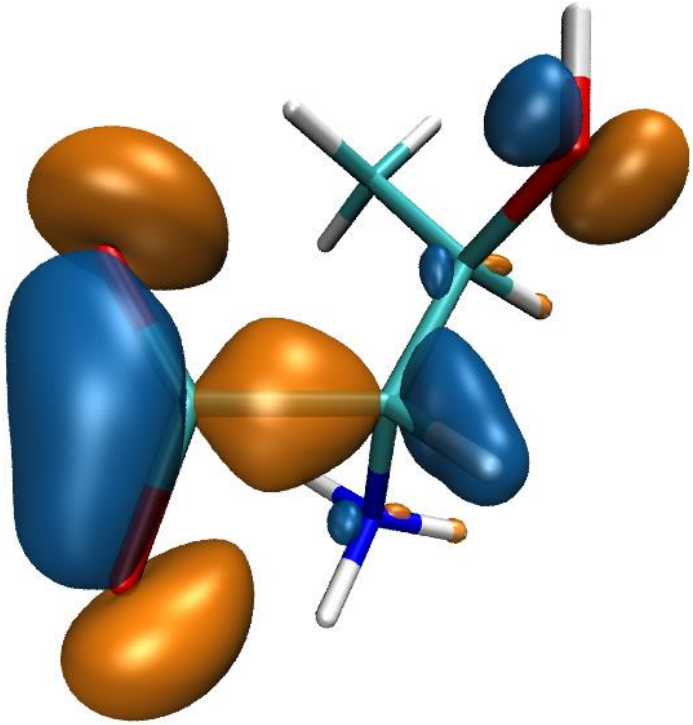
Molecular system	Atoms	Resolution	T_{sort}	$T_{density}$	T_{MC}	# vertices	FPS
MscL	111,016	1.0Å	0.005	0.023	0.003	0.7 M	28
STMV capsid	147,976	1.0Å	0.007	0.048	0.009	2.4 M	13.2
Poliovirus capsid	754,200	1.0Å	0.01	0.18	0.05	9.2 M	3.5
STMV w/ water	955,225	1.0Å	0.008	0.189	0.012	2.3 M	4.2
Membrane	2.37 M	2.0Å	0.03	0.17	0.016	5.9 M	3.9
Chromatophore	9.62 M	2.0Å	0.16	0.023	0.06	11.5 M	3.4
Membrane w/ water	22.77 M	4.0Å	4.4	0.68	0.01	1.9 M	0.18

Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.

M. Krone, J. E. Stone, T. Ertl, K. Schulten. *EuroVis Short Papers*, pp. 67-71, 2012

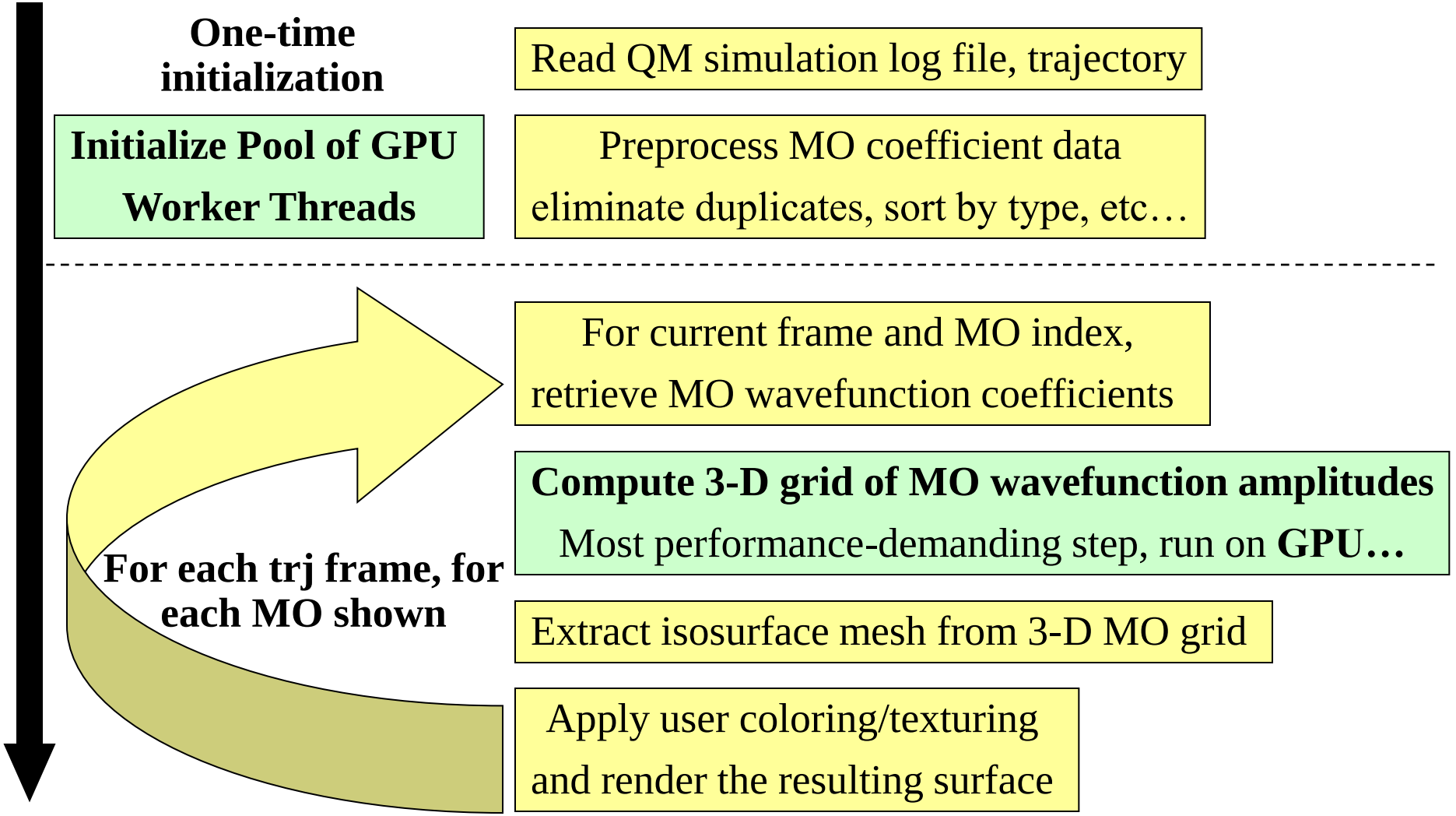
Animating Molecular Orbitals

- Animation of (classical mechanics) molecular dynamics trajectories provides insight into simulation results
- To do the same for QM or QM/MM simulations one must compute MOs at ~**10 FPS** or more
- **>100x** speedup (GPU) over existing tools now makes this possible!



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

Molecular Orbital Computation and Display Process



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

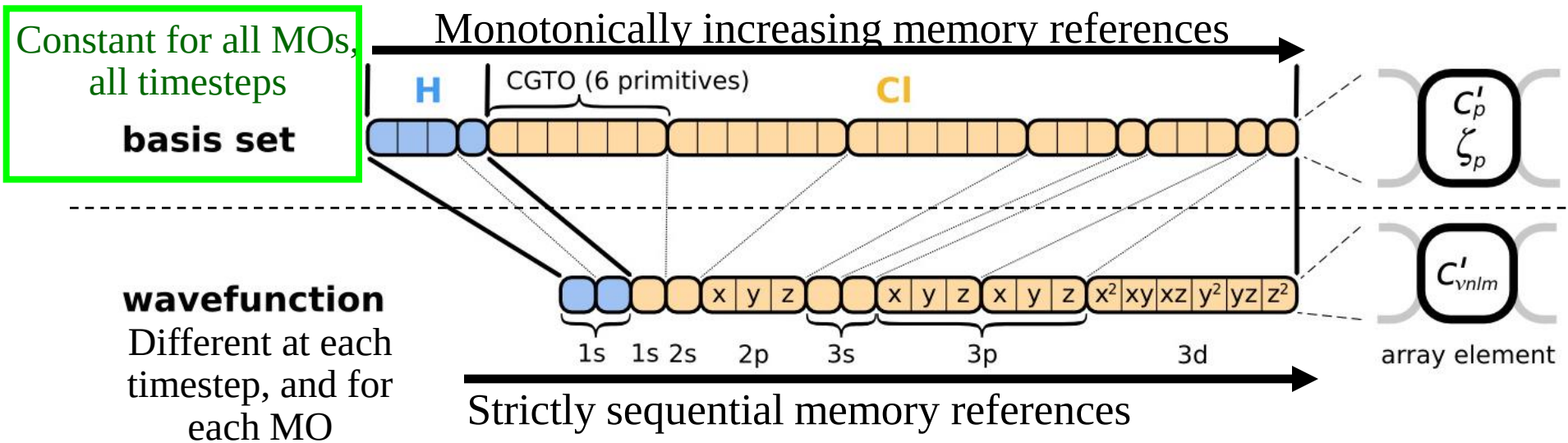
Loop over shells

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

Loop over angular momenta

(unrolled in real code)

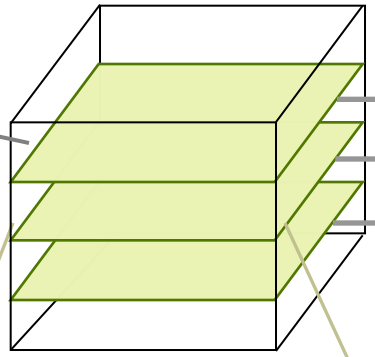
GPU Traversal of Atom Type, Basis Set, Shell Type, and Wavefunction Coefficients



- Loop iterations always access same or consecutive array elements for all threads in a thread block:
  - Yields good constant memory and L1 cache performance
  - Increases shared memory tile reuse

MO GPU Parallel Decomposition

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



...

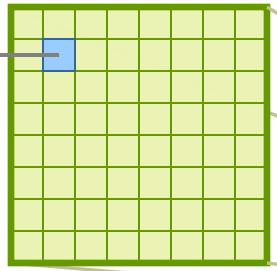
GPU 2

GPU 1

GPU 0

Lattice can be computed using multiple GPUs

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



Each thread computes one MO lattice point.

Threads producing results that are used

Threads producing results that are discarded

Grid of thread blocks

Padding optimizes global memory performance, guaranteeing coalesced global memory accesses

MO GPU Kernel Snippet:
Contracted GTO Loop, Use of Constant Memory

```
[… outer loop over atoms …]
  float dist2 = xdist2 + ydist2 + zdist2;
  // Loop over the shells belonging to this atom (or basis function)
  for (shell=0; shell < maxshell; shell++) {
    float contracted_gto = 0.0f;
    // Loop over the Gaussian primitives of this contracted basis function to build the atomic
     orbital
    int maxprim = const_num_prim_per_shell[shell_counter];
    int shelltype = const_shell_types[shell_counter];
    for (prim=0; prim < maxprim;  prim++) {
      float exponent       = const_basis_array[prim_counter    ];
      float contract_coeff = const_basis_array[prim_counter + 1];
      contracted_gto += contract_coeff * __expf(-exponent*dist2);
      prim_counter += 2;
    }
[… continue on to angular momenta loop …]
```

Constant memory: nearly register-speed when array elements accessed in unison by all threads….

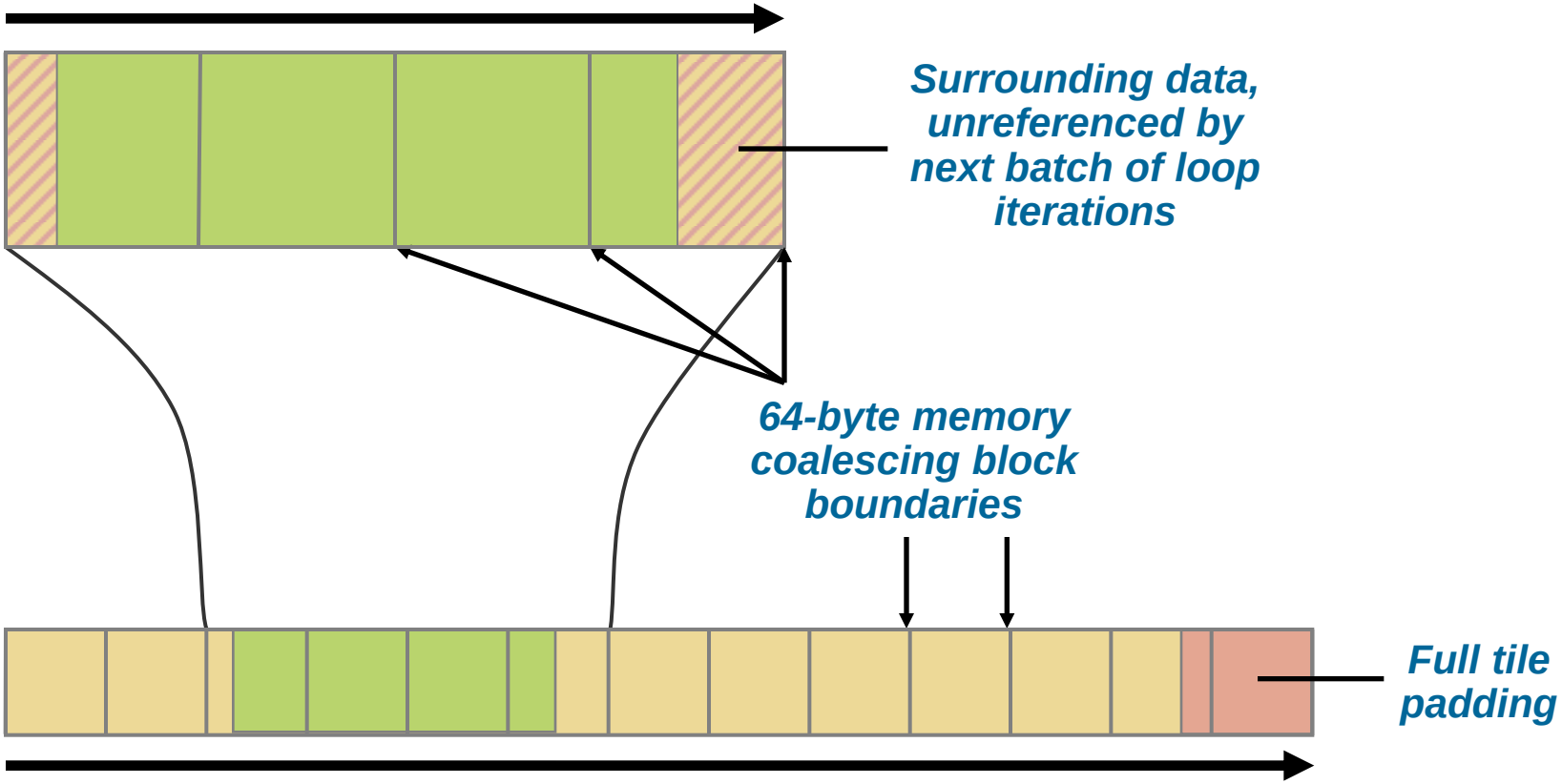
MO GPU Kernel Snippet:
Unrolled Angular Momenta Loop

```
/* multiply with the appropriate wavefunction coefficient */
float tmpshell=0;
switch (shelltype) {
  case S_SHELL:
    value += const_wave_f[ifunc++] * contracted_gto;
    break;
[… P_SHELL case …]
  case D_SHELL:
    tmpshell += const_wave_f[ifunc++] * xdist2;
    tmpshell += const_wave_f[ifunc++] * xdist * ydist;
    tmpshell += const_wave_f[ifunc++] * ydist2;
    tmpshell += const_wave_f[ifunc++] * xdist * zdist;
    tmpshell += const_wave_f[ifunc++] * ydist * zdist;
    tmpshell += const_wave_f[ifunc++] * zdist2;
    value += tmpshell * contracted_gto;
    break;
[... Other cases: F_SHELL, G_SHELL, etc …]
} // end switch
```

Loop unrolling:

- Saves registers (important for GPUs!)
- Reduces loop control overhead
- Increases arithmetic intensity

Array tile loaded in GPU shared memory. Tile size is a power-of-two, a multiple of coalescing size, and allows simple indexing in inner loops. Global memory array indices are merely offset to reference an MO coefficient within a tile loaded in fast on-chip shared memory.



MO coefficient array in GPU global memory. Tiles are referenced in consecutive order.

VMD MO GPU Kernel Snippet:
Loading Tiles Into Shared Memory On-Demand

```
[… outer loop over atoms …]
  if ((prim_counter + (maxprim<<1)) >= SHAREDSIZE) {
    prim_counter += sblock_prim_counter;
    sblock_prim_counter = prim_counter & MEMCOAMASK;
    s_basis_array[sidx      ] = basis_array[sblock_prim_counter + sidx      ];
    s_basis_array[sidx +  64] = basis_array[sblock_prim_counter + sidx +  64];
    s_basis_array[sidx + 128] = basis_array[sblock_prim_counter + sidx + 128];
    s_basis_array[sidx + 192] = basis_array[sblock_prim_counter + sidx + 192];
    prim_counter -= sblock_prim_counter;
    __syncthreads();
  }
  for (prim=0; prim < maxprim;  prim++) {
    float exponent       = s_basis_array[prim_counter    ];
    float contract_coeff = s_basis_array[prim_counter + 1];
    contracted_gto += contract_coeff * __expf(-exponent*dist2);
    prim_counter += 2;
  }
[… continue on to angular momenta loop …]
```

Shared memory tiles:

- Tiles are checked and loaded, if necessary, immediately prior to entering key arithmetic loops
- Adds additional control overhead to loops, even with optimized implementation

VMD MO GPU Kernel Snippet:
Fermi/Kepler kernel based on L1 cache

```
[… outer loop over atoms …]
  // loop over the shells/basis funcs belonging to this atom
  for (shell=0; shell < maxshell; shell++) {
    float contracted_gto = 0.0f;
    int maxprim    = shellinfo[(shell_counter<<4)     ];
    int shell_type = shellinfo[(shell_counter<<4) + 1];
    for (prim=0; prim < maxprim; 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;
    }
    [… continue on to angular momenta loop …]
```

L1 cache:

- Simplifies code!
- Reduces control overhead
- Gracefully handles arbitrary-sized problems
- Matches performance of constant memory on Fermi

Performance Evaluation:

Molekel, MacMolPlt, and VMD

Sun Ultra 24: Intel Q6600, NVIDIA GTX 280

	C_{60}-A	C_{60}-B	Thr-A	Thr-B	Kr-A	Kr-B
Atoms	60	60	17	17	1	1
Basis funcs (unique)	**300 (5)**	**900 (15)**	**49 (16)**	**170 (59)**	**19 (19)**	**84 (84)**

Kernel	Cores GPUs	Speedup vs. Molekel on 1 CPU core					
Molekel	1*	1.0	1.0	1.0	1.0	1.0	1.0
MacMolPlt	4	2.4	2.6	2.1	2.4	4.3	4.5
VMD GCC-cephes	4	3.2	4.0	3.0	3.5	4.3	6.5
VMD ICC-SSE-cephes	4	16.8	17.2	13.9	12.6	17.3	21.5
VMD ICC-SSE-approx**	4	59.3	53.4	50.4	49.2	54.8	69.8
VMD CUDA-const-cache	1	552.3	533.5	355.9	421.3	193.1	571.6

VMD Single-GPU Molecular Orbital Performance Results for C_{60} on Fermi

Intel X5550 CPU, GeForce GTX 480 GPU

Kernel	Cores/GPUs	Runtime (s)	Speedup
Xeon 5550 ICC-SSE	1	30.64	1.0
Xeon 5550 ICC-SSE	8	4.13	7.4
CUDA shared mem	1	0.37	83
CUDA L1-cache (16KB)	**1**	**0.27**	**113**
CUDA const-cache	1	0.26	117
CUDA const-cache, zero-copy	1	0.25	122

Fermi GPUs have caches: match perf. of hand-coded shared memory kernels. Zero-copy memory transfers improve overlap of computation and host-GPU I/Os.

Preliminary Single-GPU Molecular Orbital Performance Results for C_{60} on Kepler

Intel X5550 CPU, GeForce GTX 680 GPU

Kernel	Cores/GPUs	Runtime (s)	Speedup
Xeon 5550 ICC-SSE	1	30.64	1.0
Xeon 5550 ICC-SSE	8	4.13	7.4
CUDA shared mem	1	0.264	116
CUDA L1-cache (16KB)	**1**	**0.228**	**134**
CUDA const-cache	1	0.104	292
CUDA const-cache, zero-copy	1	0.0938	326

Kepler GK104 (GeForce 680) seems to strongly prefer the constant cache kernels vs. the others.

VMD Multi-GPU Molecular Orbital Performance Results for C_{60}

Intel X5550 CPU, 4x GeForce GTX 480 GPUs,

Kernel	Cores/GPUs	Runtime (s)	Speedup
Intel X5550-SSE	1	30.64	1.0
Intel X5550-SSE	8	4.13	7.4
GeForce GTX 480	1	0.255	120
GeForce GTX 480	2	0.136	225
GeForce GTX 480	3	0.098	312
GeForce GTX 480	4	0.081	378

Uses persistent thread pool to avoid GPU init overhead, dynamic scheduler distributes work to GPUs

Molecular Orbital Dynamic Scheduling Performance with Heterogeneous GPUs

Kernel	Cores/GPUs	Runtime (s)	Speedup
Intel X5550-SSE	1	30.64	1.0
Quadro 5800	1	0.384	79
Tesla C2050	1	0.325	94
GeForce GTX 480	1	0.255	120
GeForce GTX 480 + Tesla C2050 + Quadro 5800	3	0.114	268 (91% of ideal perf)

Dynamic load balancing enables mixture of GPU generations, SM counts, and clock rates to perform well.

MO Kernel Structure, Opportunity for JIT…

Data-driven, but representative loop trip counts in (…)

Loop over atoms (1 to ~200) {

Loop over electron shells for this atom type (1 to ~6) {

Loop over primitive functions for this shell type (1 to ~6) {

Unpredictable (at compile-time, since data-driven) but small loop trip counts result in significant loop overhead. **Dynamic kernel generation and JIT compilation can unroll entirely, resulting in 40% speed boost**

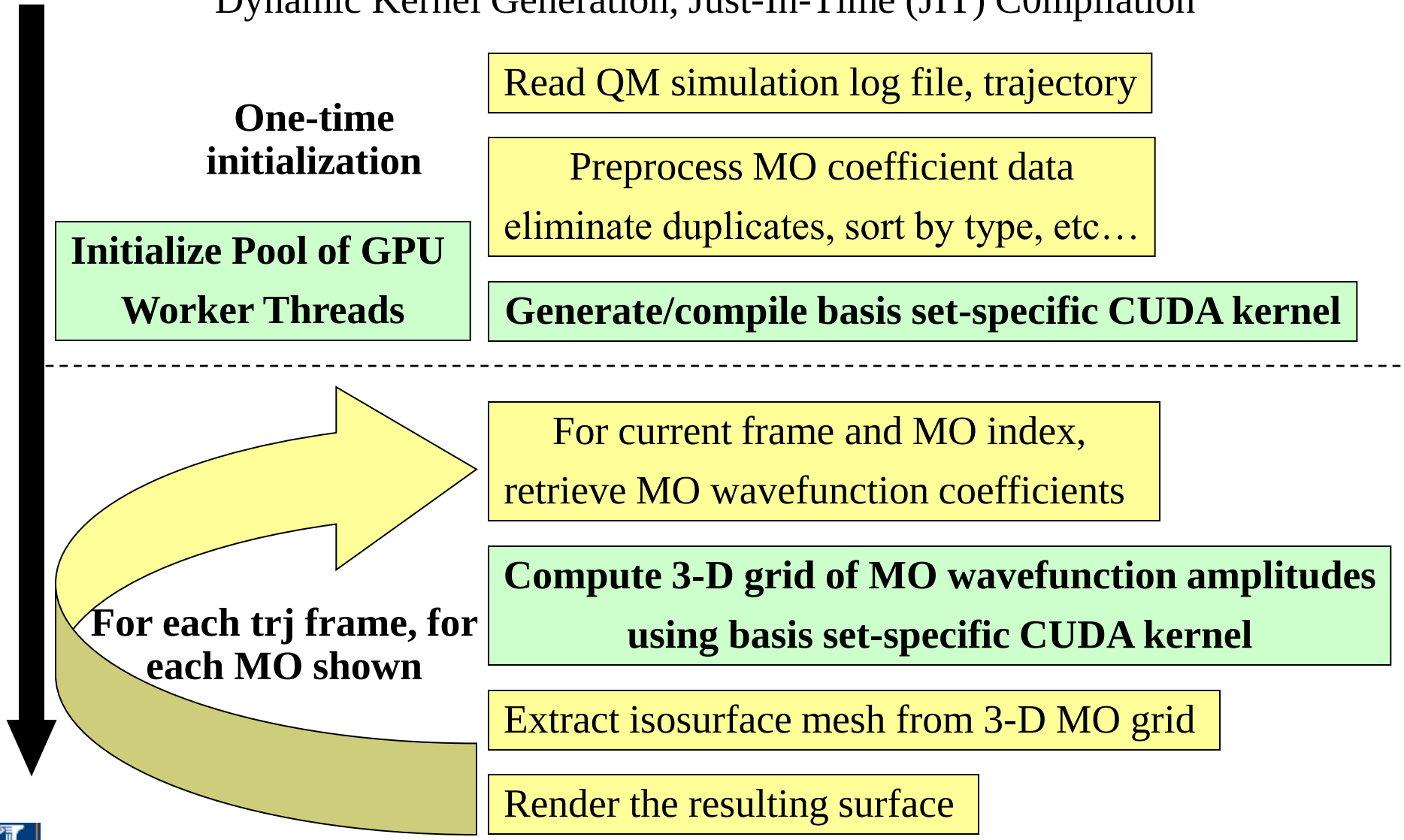
Loop over angular momenta for this shell type (1 to ~15) {}

}

}

Molecular Orbital Computation and Display Process

Dynamic Kernel Generation, Just-In-Time (JIT) C0mpilation



Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- NVIDIA CUDA Center of Excellence, University of Illinois at Urbana-Champaign
- NCSA Blue Waters Team
- NCSA Innovative Systems Lab
- The CUDA team at NVIDIA
- NIH support: 9P41GM104601, P41RR005969

GPU Computing Publications

http://www.ks.uiuc.edu/Research/gpu/

- **Lattice Microbes: High-performance stochastic simulation method for the reaction-diffusion master equation.**
  E. Roberts, J. E. Stone, and Z. Luthey-Schulten.
  J. Computational Chemistry 34 (3), 245-255, 2013**.**
- **Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.** M. Krone, J. E. Stone, T. Ertl, and K. Schulten. *EuroVis Short Papers*, pp. 67-71, 2012.
- **Immersive Out-of-Core Visualization of Large-Size and Long-Timescale Molecular Dynamics Trajectories.** J. Stone, K. Vandivort, and K. Schulten. G. Bebis et al. (Eds.): *7th International Symposium on Visual Computing (ISVC 2011)*, LNCS 6939, pp. 1-12, 2011.
- **Fast Analysis of Molecular Dynamics Trajectories with Graphics Processing Units – Radial Distribution Functions.** B. Levine, J. Stone, and A. Kohlmeyer. *J. Comp. Physics*, 230(9):3556-3569, 2011.

GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/

- **Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters.** J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J. Stone, J Phillips. *International Conference on Green Computing*, pp. 317-324, 2010.
- **GPU-accelerated molecular modeling coming of age. J. Stone, D. Hardy, I. Ufimtsev, K. Schulten.** *J. Molecular Graphics and Modeling*, 29:116-125, 2010.
- **OpenCL: A Parallel Programming Standard for Heterogeneous Computing. J. Stone, D. Gohara, G. Shi.** *Computing in Science and Engineering*, 12(3):66-73, 2010.
- **An Asymmetric Distributed Shared Memory Model for Heterogeneous Computing Systems**. I. Gelado, J. Stone, J. Cabezas, S. Patel, N. Navarro, W. Hwu. *ASPLOS '10: Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 347-358, 2010.

GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/

- **GPU Clusters for High Performance Computing**. V. Kindratenko, J. Enos, G. Shi, M. Showerman, G. Arnold, J. Stone, J. Phillips, W. Hwu. *Workshop on Parallel Programming on Accelerator Clusters (PPAC)*, In Proceedings IEEE Cluster 2009, pp. 1-8, Aug. 2009.
- **Long time-scale simulations of in vivo diffusion using GPU hardware**. E. Roberts, J. Stone, L. Sepulveda, W. Hwu, Z. Luthey-Schulten. In *IPDPS'09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Computing*, pp. 1-8, 2009.
- **High Performance Computation and Interactive Display of Molecular Orbitals on GPUs and Multi-core CPUs**. J. Stone, J. Saam, D. Hardy, K. Vandivort, W. Hwu, K. Schulten, *2nd Workshop on General-Purpose Computation on Graphics Pricessing Units (GPGPU-2), ACM International Conference Proceeding Series*, volume 383, pp. 9-18, 2009.
- **Probing Biomolecular Machines with Graphics Processors**. J. Phillips, J. Stone. *Communications of the ACM*, 52(10):34-41, 2009.
- **Multilevel summation of electrostatic potentials using graphics processing units**. D. Hardy, J. Stone, K. Schulten. *J. Parallel Computing*, 35:164-177, 2009.

GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/

- **Adapting a message-driven parallel application to GPU-accelerated clusters**. J. Phillips, J. Stone, K. Schulten. *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, IEEE Press, 2008.
- **GPU acceleration of cutoff pair potentials for molecular modeling applications**. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, and W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.
- **GPU computing**. J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips. *Proceedings of the IEEE*, 96:879-899, 2008.
- **Accelerating molecular modeling applications with graphics processors**. J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, K. Schulten. *J. Comp. Chem.*, 28:2618-2640, 2007.
- **Continuous fluorescence microphotolysis and correlation spectroscopy**. A. Arkhipov, J. Hüve, M. Kahms, R. Peters, K. Schulten. *Biophysical Journal*, 93:4006-4017, 2007.