GPU Accelerated Visualization and Analysis in VMD
and
Recent NAMD Developments

John Stone

Theoretical and Computational Biophysics Group

Beckman Institute for Advanced Science and Technology

University of Illinois at Urbana-Champaign

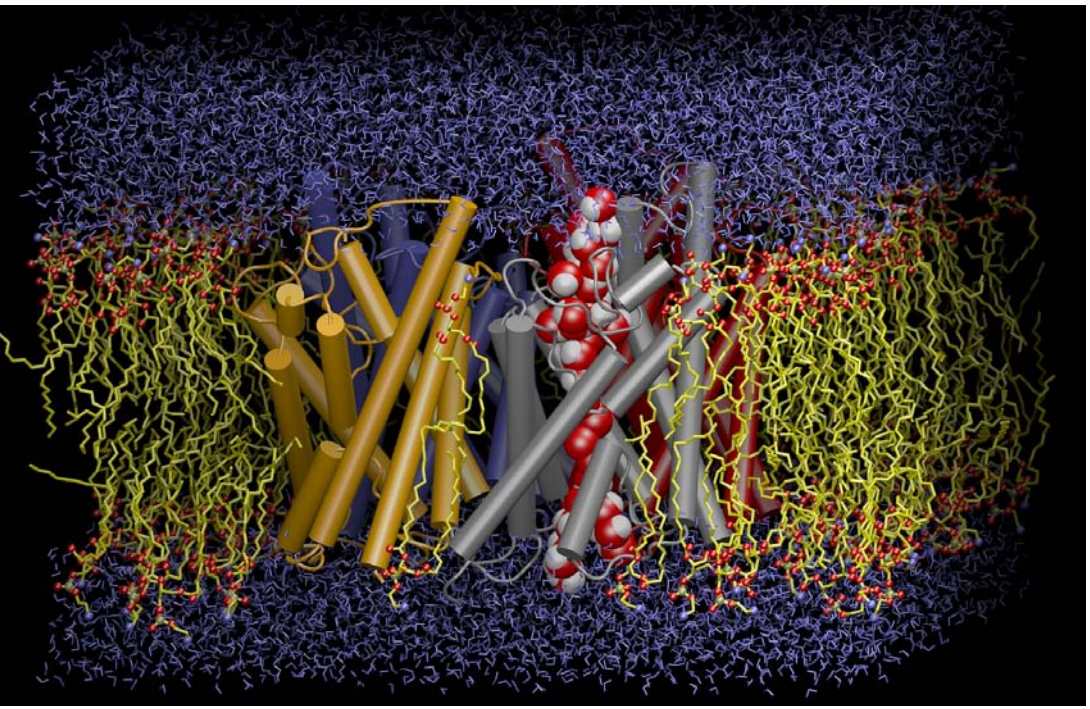
http://www.ks.uiuc.edu/Research/gpu/

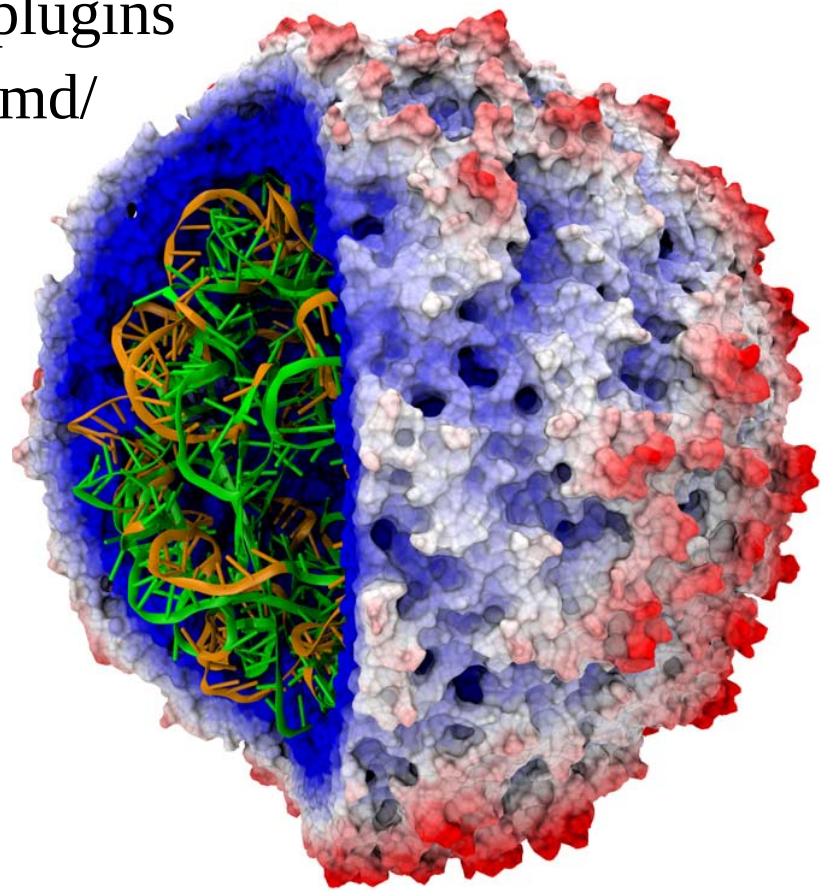
GPU Technology Conference

Fairmont Hotel, San Jose, CA, October 1, 2009

VMD – “Visual Molecular Dynamics”

- Visualization and analysis of molecular dynamics simulations, sequence data, volumetric data, quantum chemistry simulations, particle systems, …
- User extensible with scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/





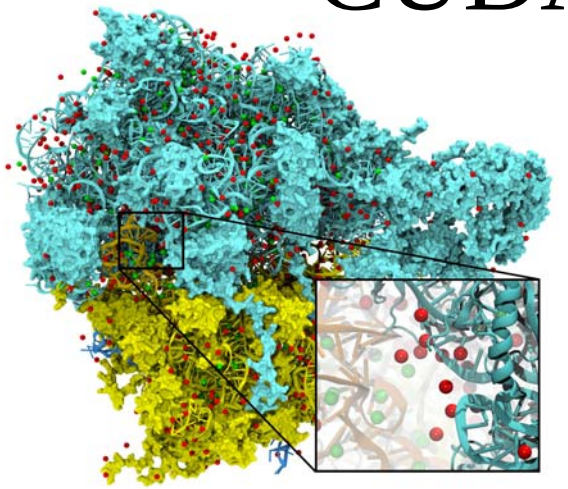
Range of VMD Usage Scenarios

- Users run VMD on a diverse range of hardware: laptops, desktops, clusters, and supercomputers
- Typically used as a desktop application, for interactive 3D molecular graphics and analysis
- Can also be run in pure text mode for numerically intensive analysis tasks, batch mode movie rendering, etc…
- GPU acceleration provides an opportunity to make some **slow, or batch** calculations capable of being run **interactively, or on-demand…**

Need for Multi-GPU Acceleration in VMD

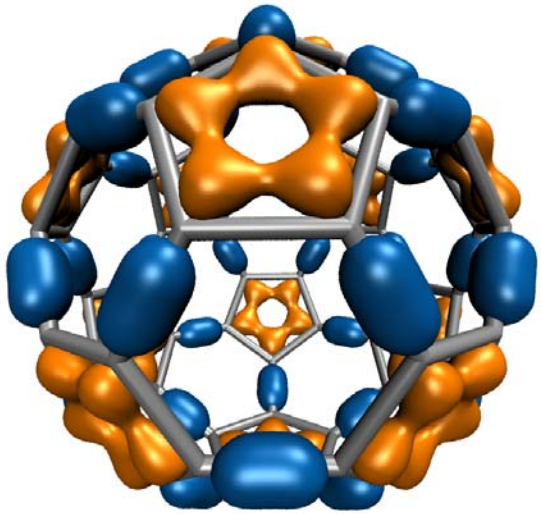
- Ongoing increases in supercomputing resources at NSF centers such as NCSA enable increased simulation complexity, fidelity, and longer time scales…
- Drives need for more visualization and analysis capability at the desktop and on clusters running batch analysis jobs
- Desktop use is the most compute-resource-limited scenario, where **GPUs can make a big impact**…

CUDA Acceleration in VMD



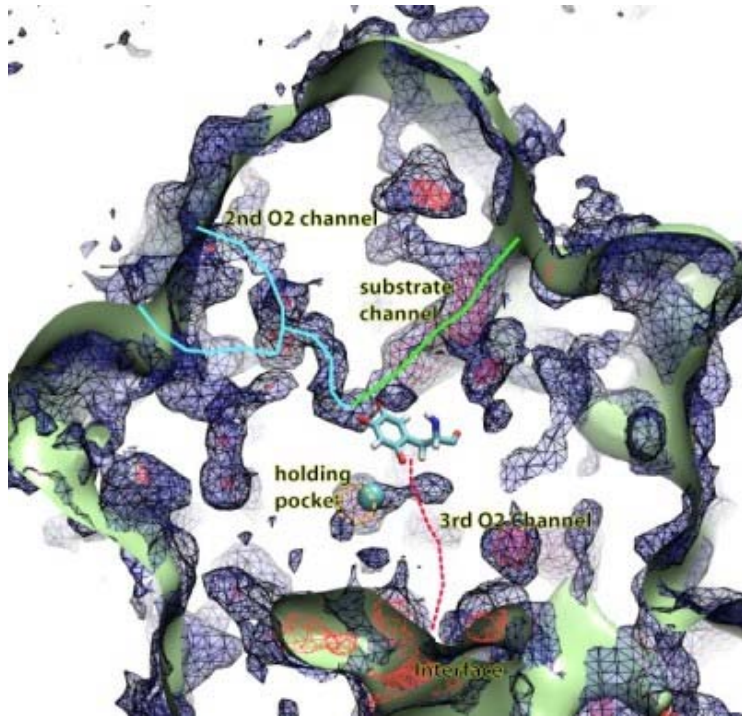
Electrostatic field calculation, ion placement

20x to 44x faster



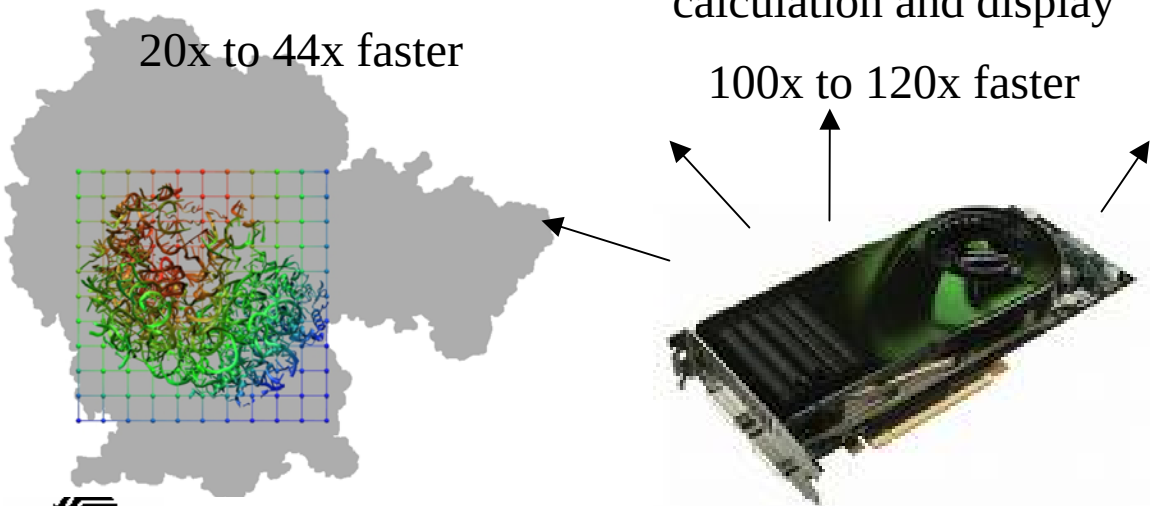
Molecular orbital calculation and display

100x to 120x faster



Imaging of gas migration pathways in proteins with implicit ligand sampling

20x to 30x faster

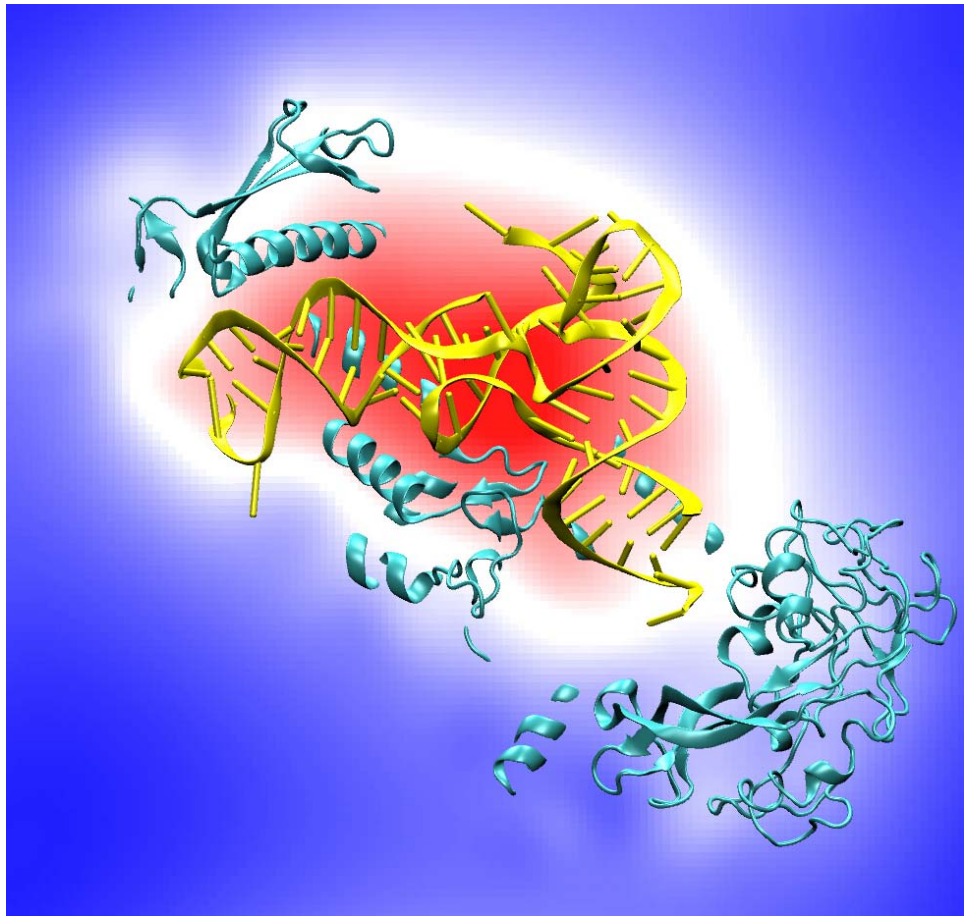


Electrostatic Potential Maps

- Electrostatic potentials evaluated on 3-D lattice:

$$V_i = \sum_j \frac{q_j}{4\pi\epsilon_0 |\mathbf{r}_j - \mathbf{r}_i|}$$

- Applications include:
  - Ion placement for structure building
  - Time-averaged potentials for simulation
  - Visualization and analysis



Isoleucine tRNA synthetase

Infinite vs. Cutoff Potentials

- Infinite range potential:
  - All atoms contribute to all lattice points
  - Quadratic time complexity
- Cutoff (range-limited) potential:
  - Atoms contribute within cutoff distance to lattice points resulting in linear time complexity
  - Used for fast decaying interactions (e.g. Lennard-Jones, Buckingham)
- Fast full electrostatics:
  - Replace electrostatic potential with shifted form
  - Combine short-range part with long-range approximation
  - Multilevel summation method (MSM), linear time complexity

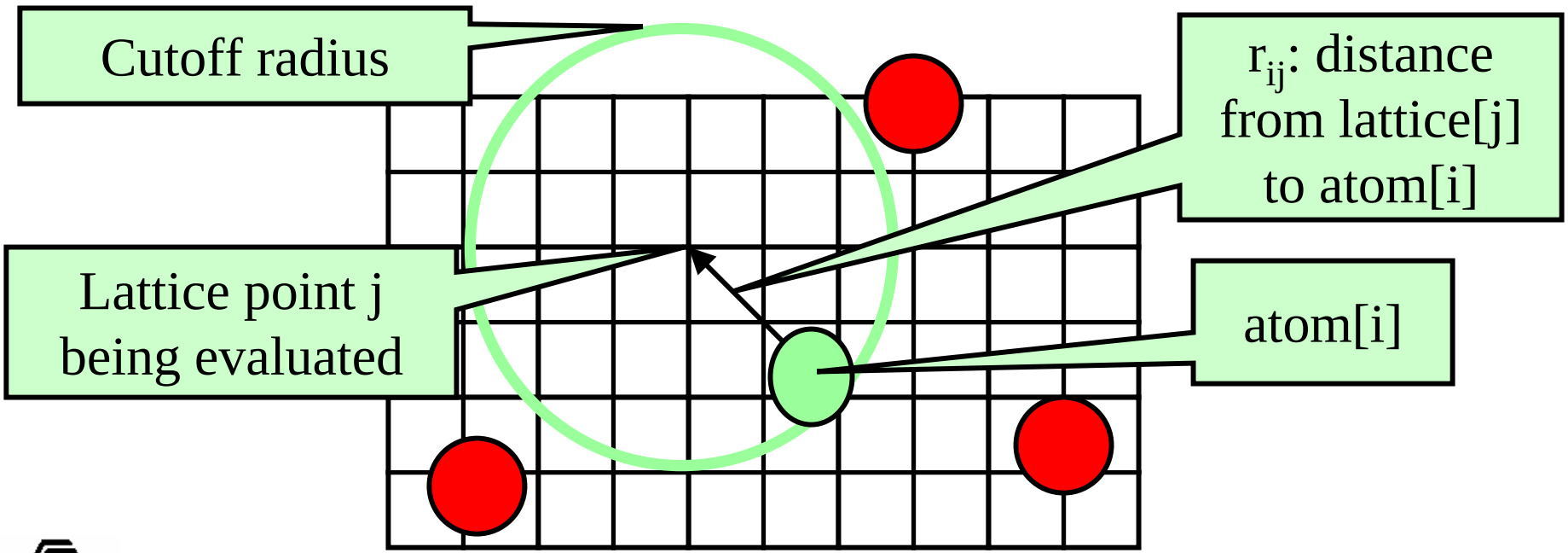
Short-range Cutoff Summation

- Each lattice point accumulates electrostatic potential contribution from atoms within cutoff distance:

  if (r_{ij} < cutoff)

    potential[j] += (charge[i] / r_{ij}) * s(r_{ij})

- Smoothing function s(r) is algorithm dependent



Cutoff Summation on the GPU

Atoms are spatially hashed into fixed-size bins

CPU handles overflowed bins (GPU kernel can be very aggressive)

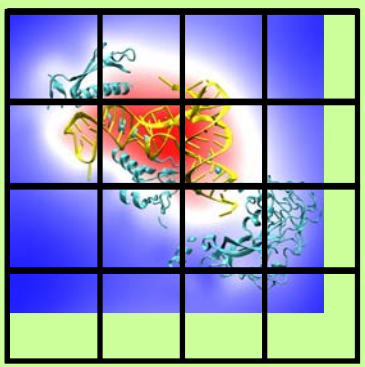
GPU thread block calculates corresponding region of potential map,

Bin/region neighbor checks costly; solved with universal table look-up

Each thread block cooperatively loads atom bins from surrounding neighborhood into shared memory for evaluation

Shared memory

atom bin



Global memory

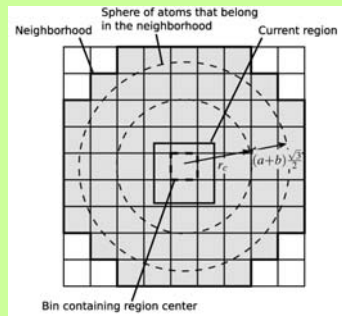
Potential map regions

Bins of atoms

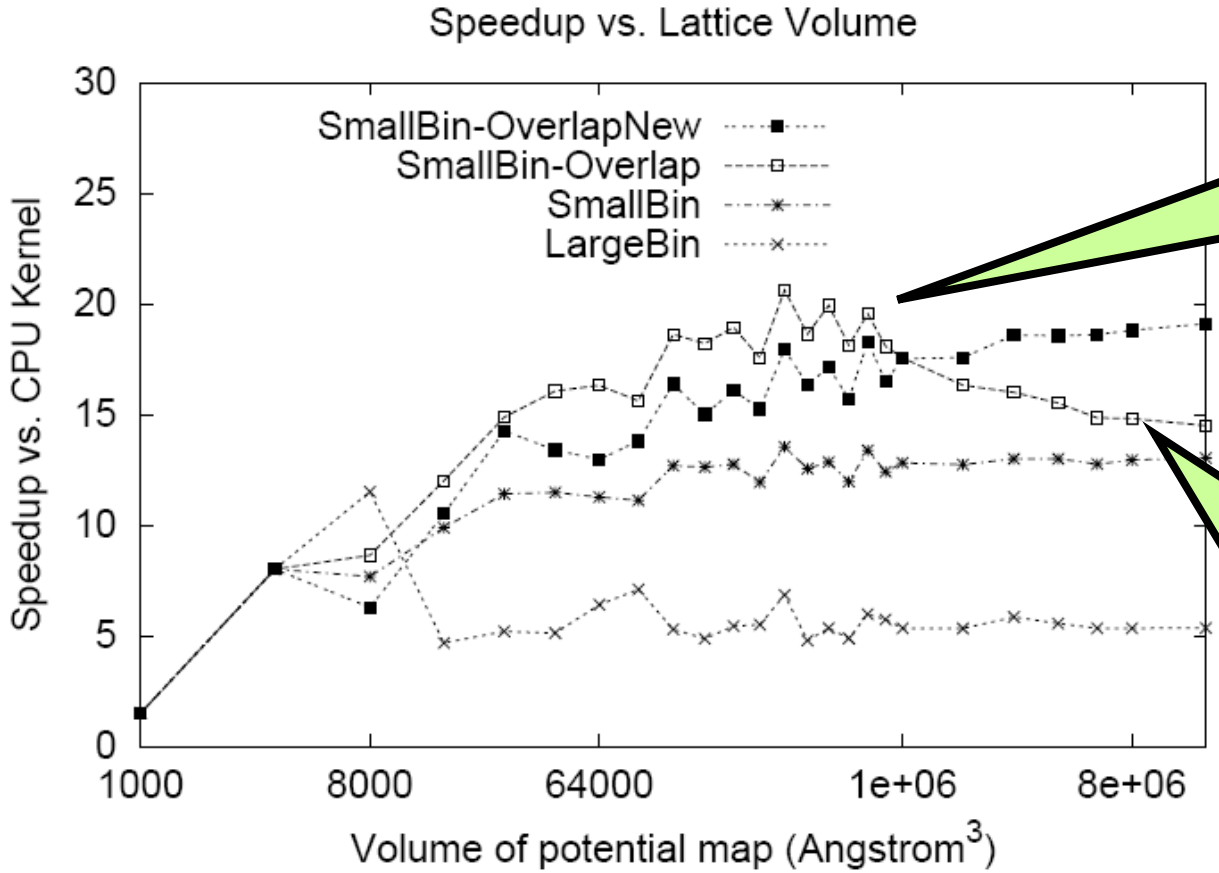
Constant memory

Offsets for bin neighborhood

Look-up table encodes "logic" of spatial geometry



Cutoff Summation Performance



GPU cutoff with CPU overlap: 17x-21x faster than CPU core

If asynchronous stream blocks due to queue filling, performance will degrade from peak…

GPU acceleration of cutoff pair potentials for molecular modeling applications. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.

Cutoff Summation Observations

- Use of CPU to handle overflowed bins is very effective, overlaps completely with GPU work
- Caveat: Overfilling stream queue can trigger blocking behavior. Recent drivers queue >100 ops before blocking.
- Higher precision:
  - Compensated summation (all GPUs) or double-precision (GT200 only) only a **~10%** performance penalty vs. single-precision arithmetic
  - Next-gen "Fermi" GPUs will have an even lower performance cost for double-precision arithmetic

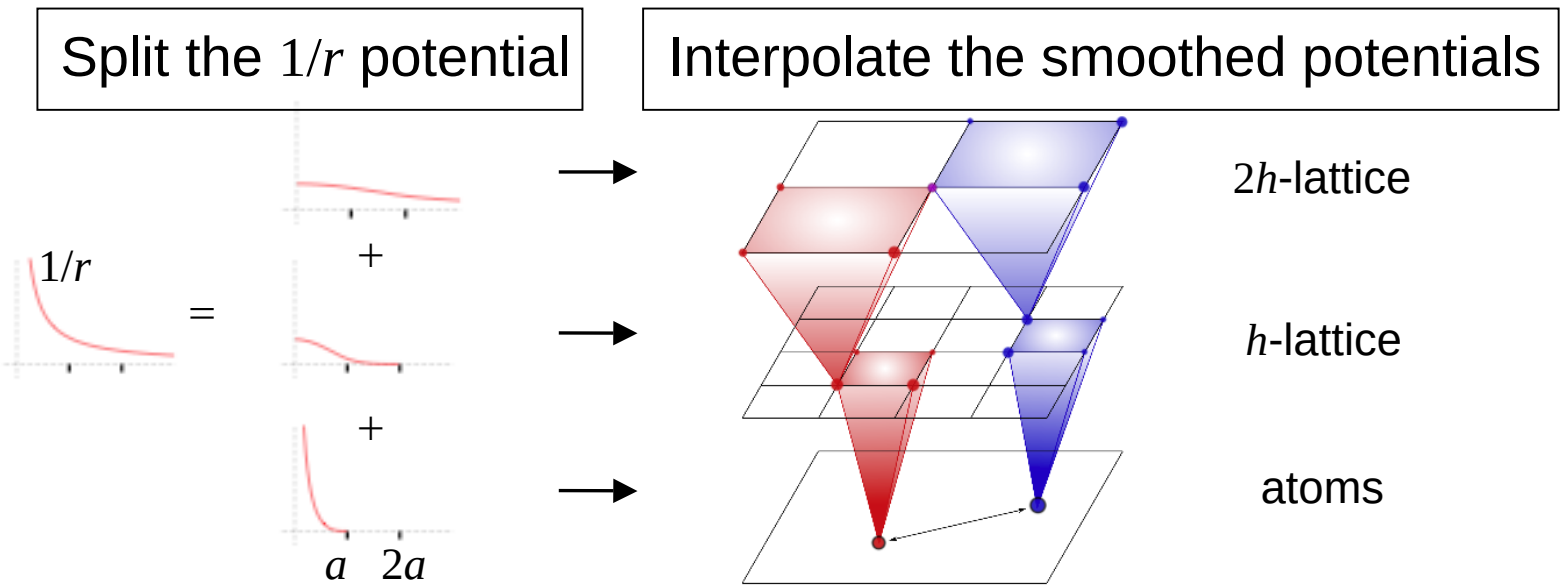
Multilevel Summation Method

- Approximates full electrostatic potential
- Calculates sum of smoothed pairwise potentials interpolated from a hierarchy of lattices
- Advantages over particle-mesh Ewald, fast multipole:
  - Algorithm has **linear time complexity**
  - Permits non-periodic and periodic boundaries
  - Produces continuous forces for dynamics (advantage over FMM)
  - Avoids 3-D FFTs for better parallel scaling (advantage over PME)
  - Spatial separation allows use of multiple time steps
  - Can be extended to other pairwise interactions

- Skeel, Tezcan, Hardy, *J Comp Chem*, 2002 — Computing forces for molecular dynamics
- Hardy, Stone, Schulten, *J Paral Comp*, 2009 — GPU-acceleration of potential map calculation

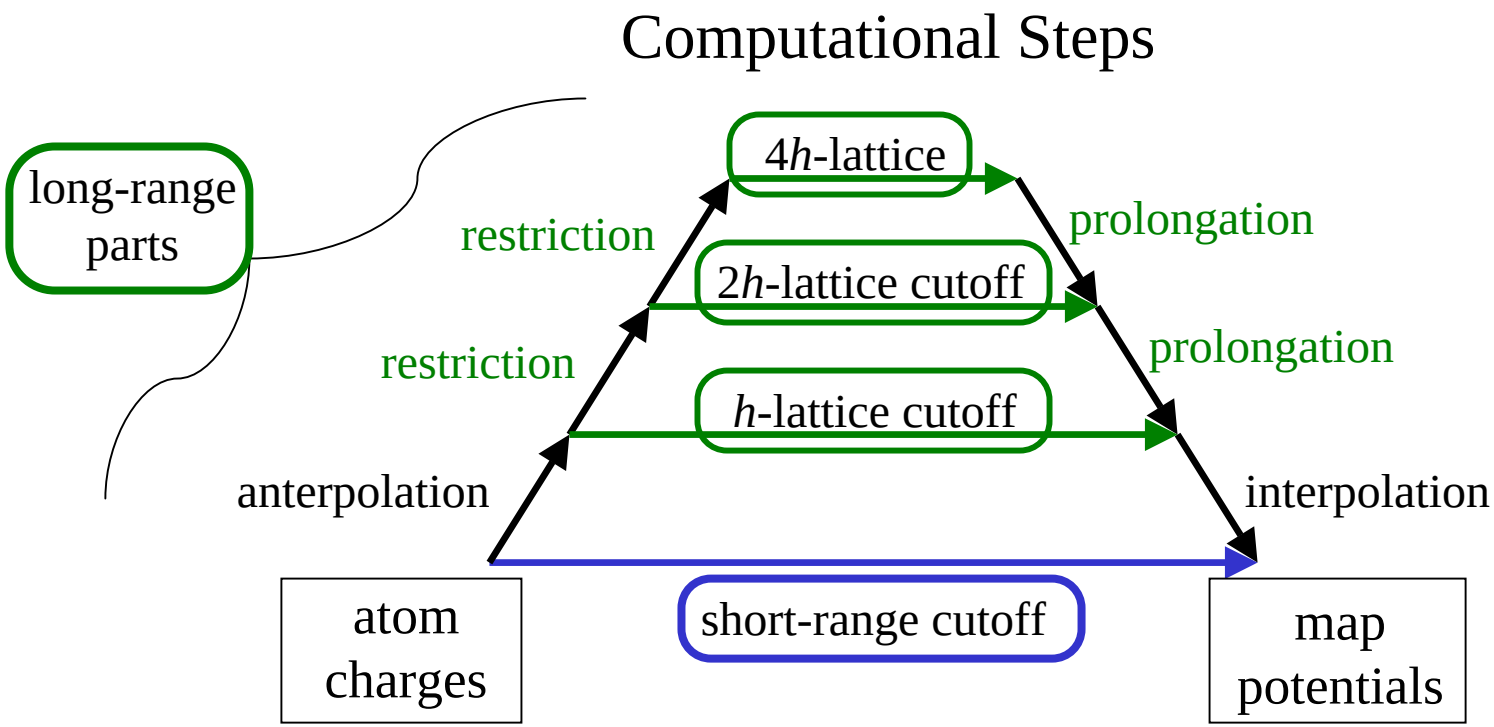
Multilevel Summation Main Ideas

- Split the 1/*r* potential into a short-range cutoff part plus smoothed parts that are successively more slowly varying. All but the top level potential are cut off.
- Smoothed potentials are interpolated from successively coarser lattices.
- Finest lattice spacing *h* and smallest cutoff distance *a* are doubled at each successive level.



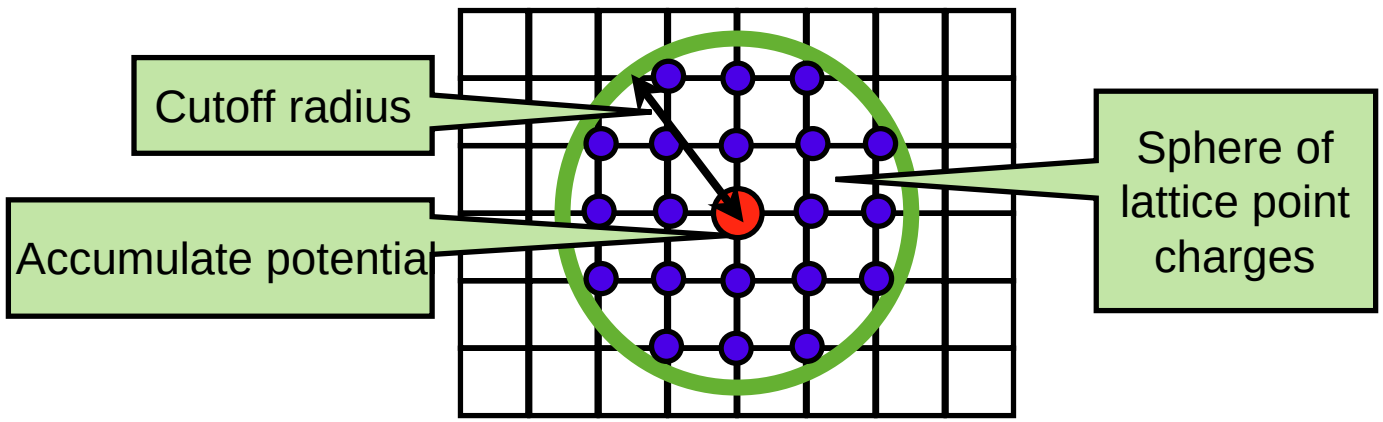
Multilevel Summation Calculation

map potential = **exact short-range interactions** + **interpolated long-range interactions**



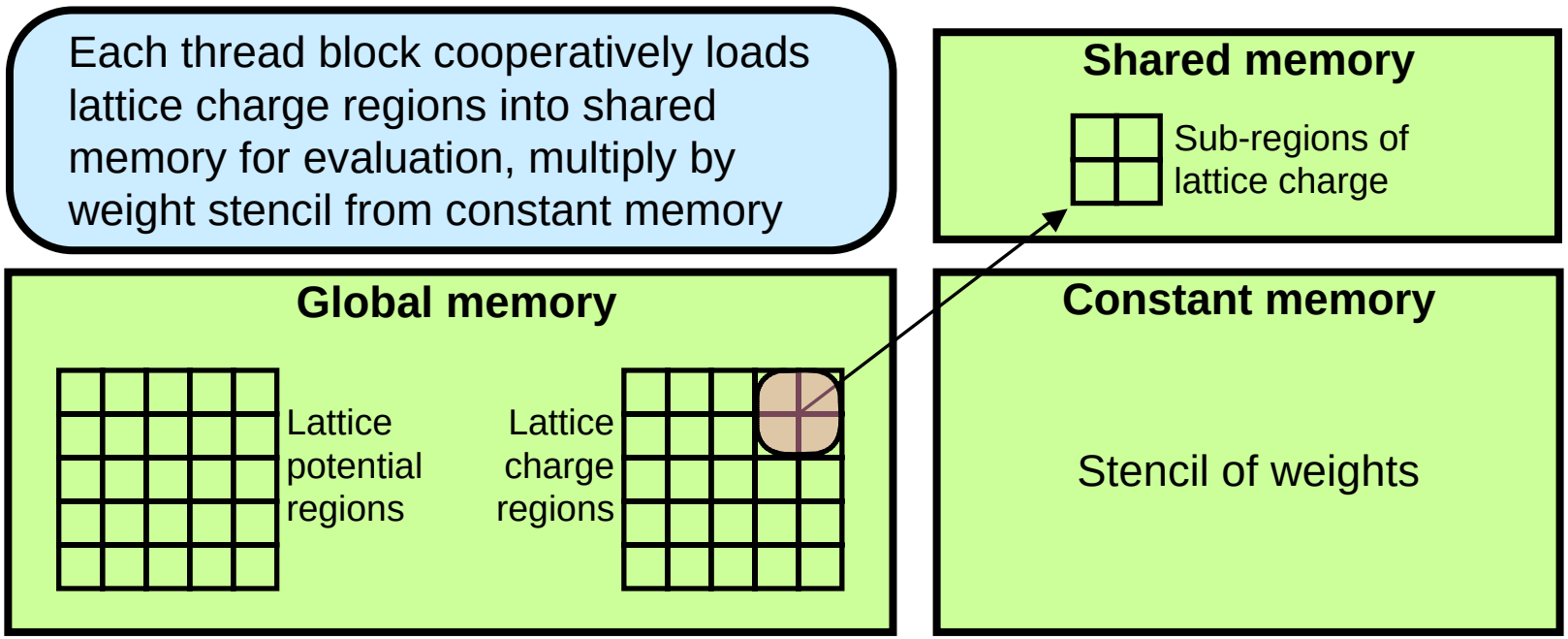
Lattice Cutoff Summation

- Each lattice point accumulates electrostatic potential contribution from all lattice point charges within cutoff distance
- Relative distances are the same between points on a uniform lattice, multiplication by a precomputed stencil of “weights”
- Weights at each level are identical up to a scaling factor (due to choice of splitting and doubling of lattice spacing and cutoff)
- Calculate as 3D convolution of sub-cube of lattice point charges with enclosing cube of weights



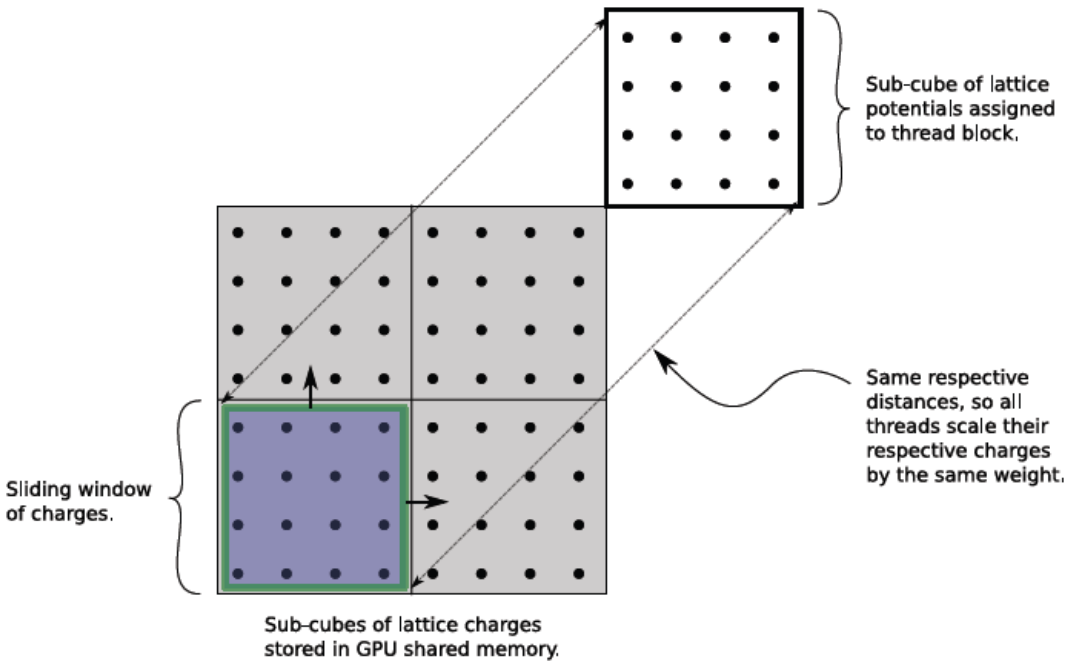
Lattice Cutoff Summation on GPU

- Store stencil of weights in constant memory
- Thread blocks calculate 4x4x4 regions of lattice potentials
- Load nearby regions of lattice charges into shared memory
- Evaluate all lattice levels concurrently, scaling by level factor (keeps GPU from running out of work at upper lattice levels)



Evaluation Using Sliding Window

- Every thread in block needs to simultaneously read and use the same weight from constant memory
- Read 8x8x8 block (8 regions) of lattice charges into shared memory
- Slide 4x4x4 window by 4 shifts along each dimension

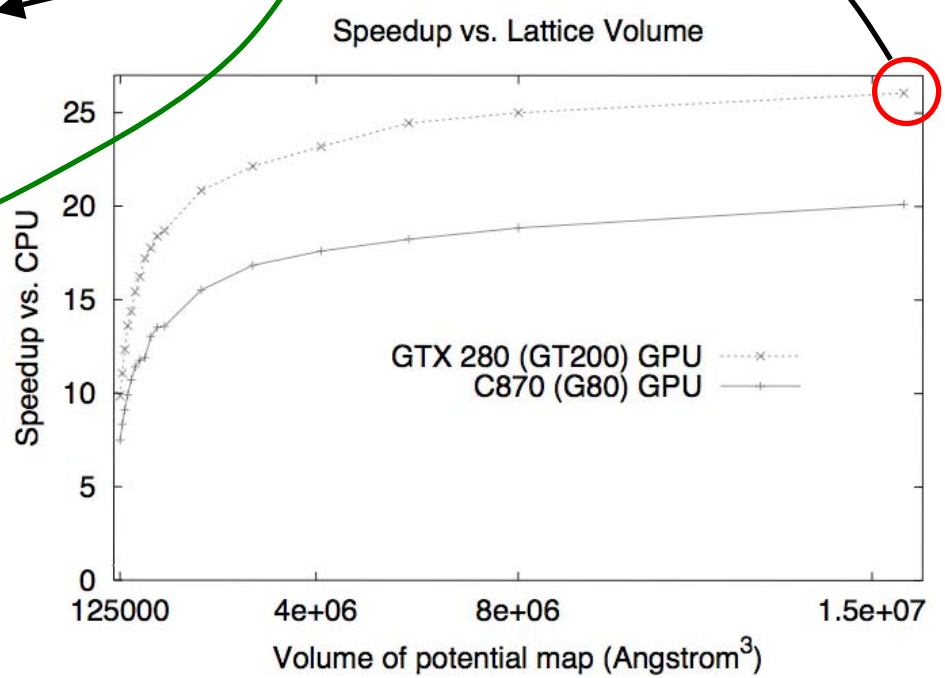


Multilevel Summation on the GPU

Accelerate **short-range cutoff** and **lattice cutoff** parts

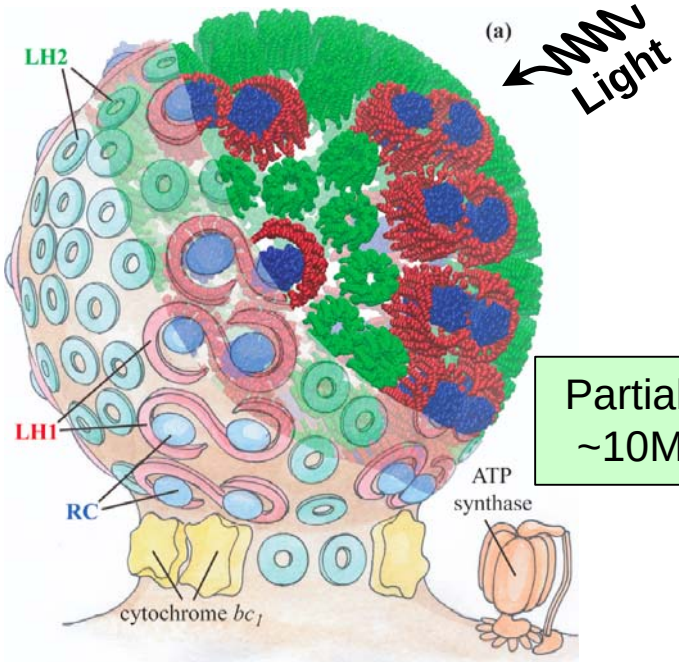
Performance profile for 0.5 Å map of potential for 1.5 M atoms.
Hardware platform is Intel QX6700 CPU and NVIDIA GTX 280.

Computational steps	CPU (s)	w/ GPU (s)	Speedup
Short-range cutoff	480.07	14.87	32.3
Long-range anterpolation	0.18		
restriction	0.16		
lattice cutoff	49.47	1.36	36.4
prolongation	0.17		
interpolation	3.47		
Total	533.52	20.21	26.4

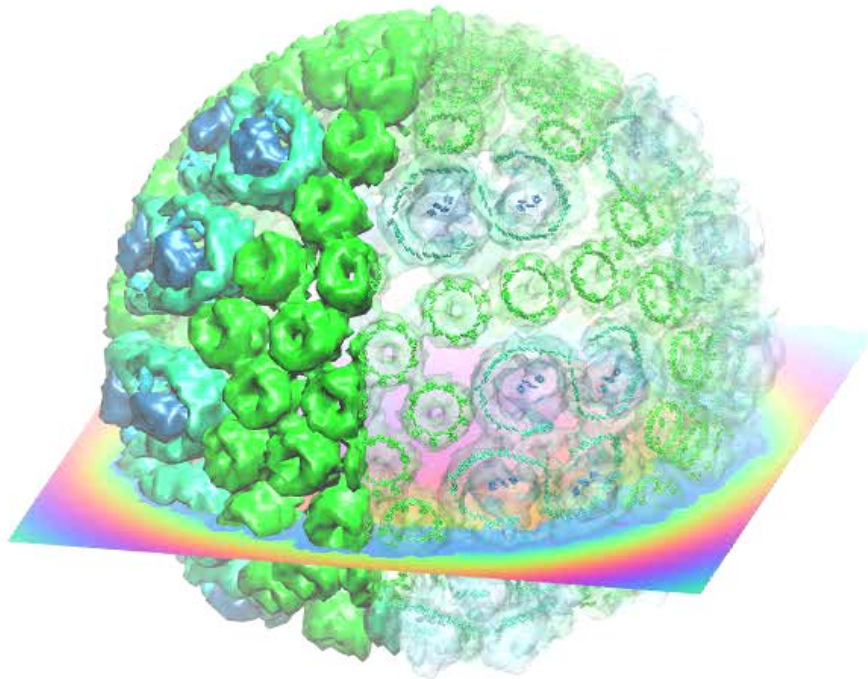


Photobiology of Vision and Photosynthesis

Investigations of the chromatophore, a photosynthetic organelle



Partial model:
~10M atoms



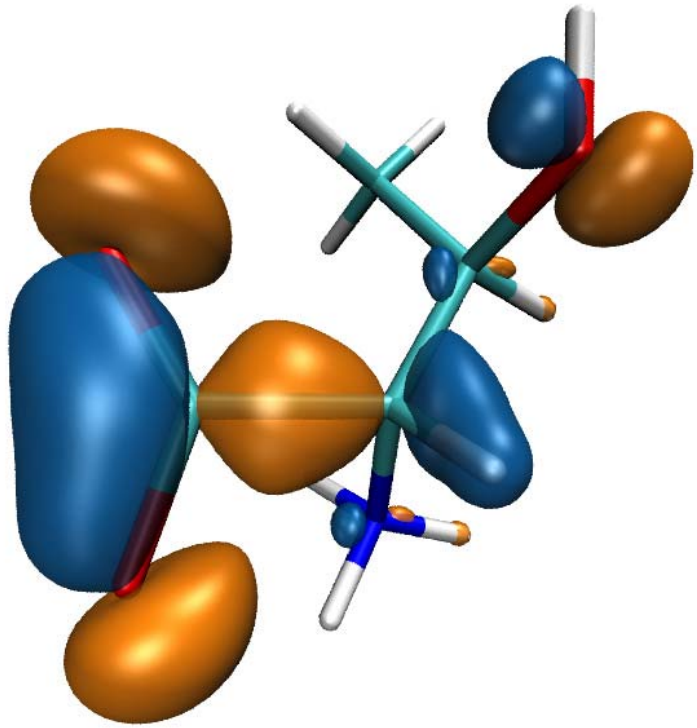
Electrostatics needed to build full structural model, place ions, study macroscopic properties

Electrostatic field of chromatophore model from multilevel summation method: computed with 3 GPUs (G80) in ~90 seconds, 46x faster than single CPU core

Full chromatophore model will permit structural, chemical and kinetic investigations at a structural systems biology level

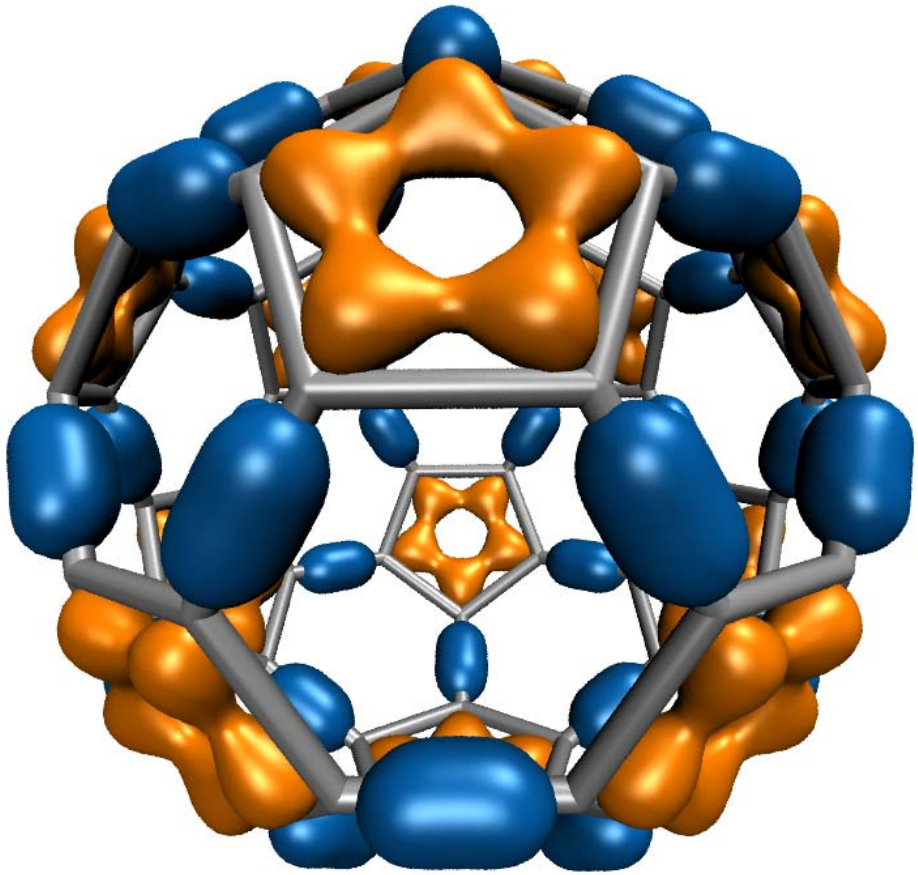
Computing Molecular Orbitals

- Visualization of MOs aids in understanding the chemistry of molecular system
- MO spatial distribution is correlated with probability density for an electron(s)
- Calculation of high resolution MO grids can require tens to hundreds of seconds on CPUs



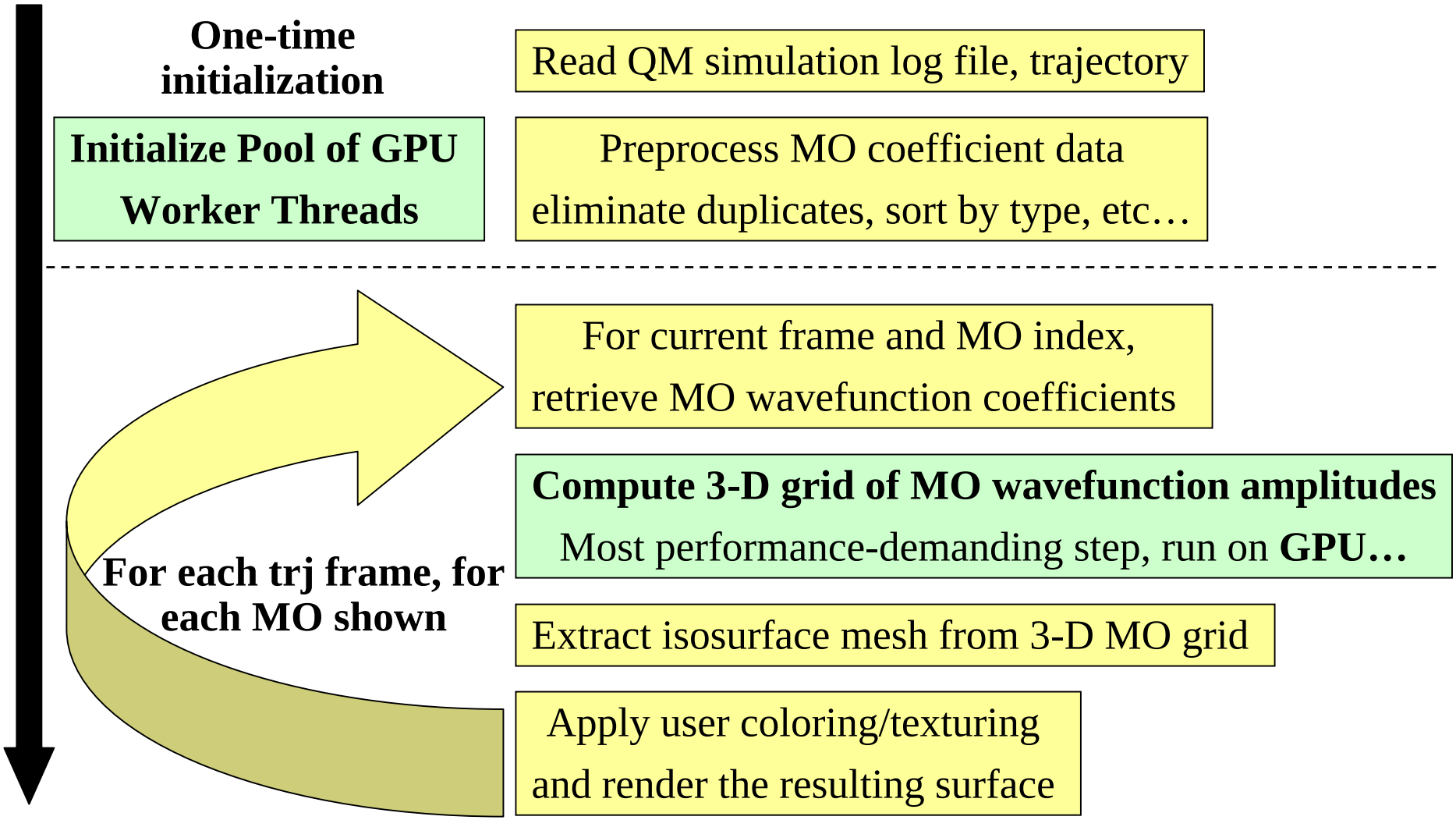
Animating Molecular Orbitals

- 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!



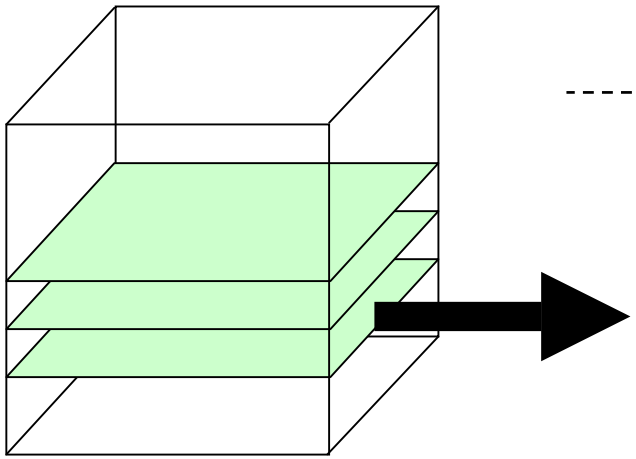
C_{60}

Molecular Orbital Computation and Display Process

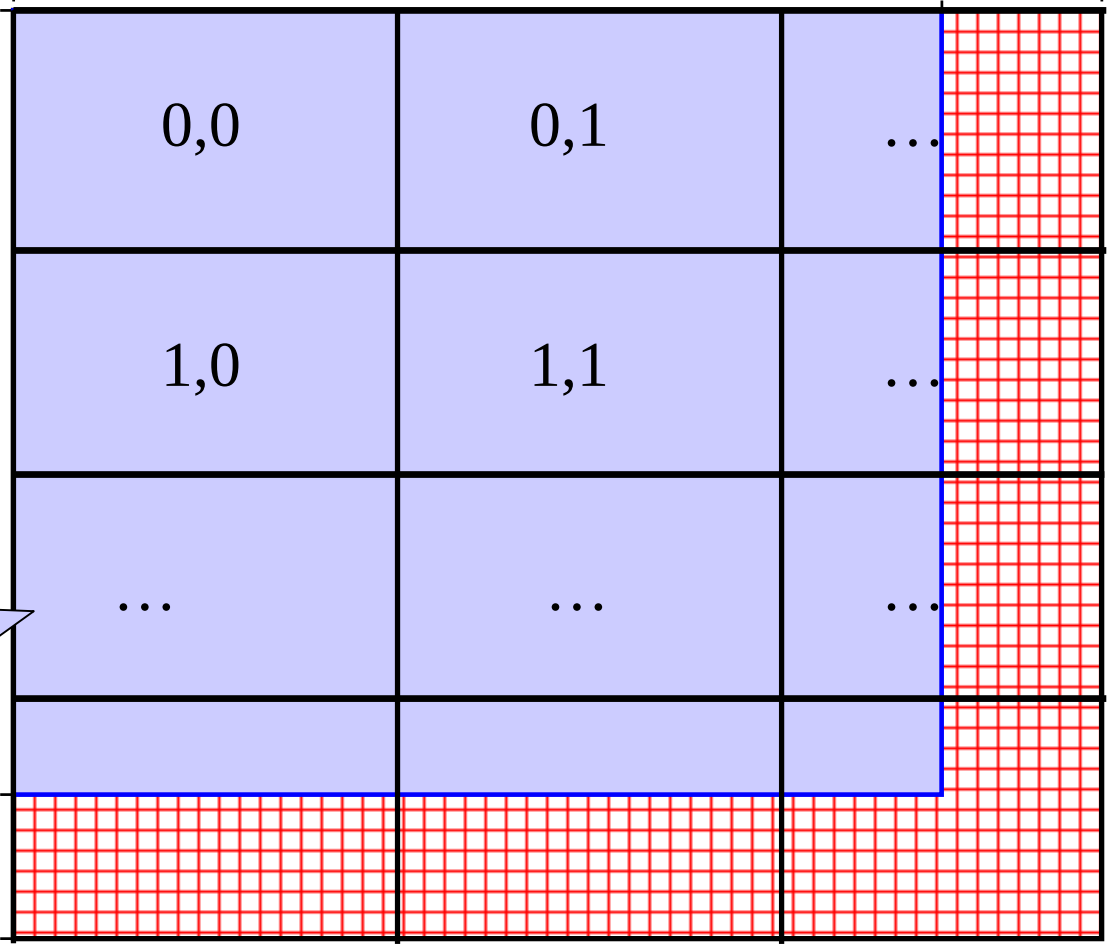


CUDA Block/Grid Decomposition

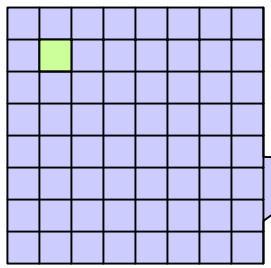
MO 3-D lattice decomposes into 2-D slices (CUDA grids)



Grid of thread blocks:



Small 8x8 thread blocks afford large per-thread register count, shared mem. Threads compute one MO lattice point each.



Padding optimizes glob. mem perf, guaranteeing coalescing

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)

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 peer 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
```

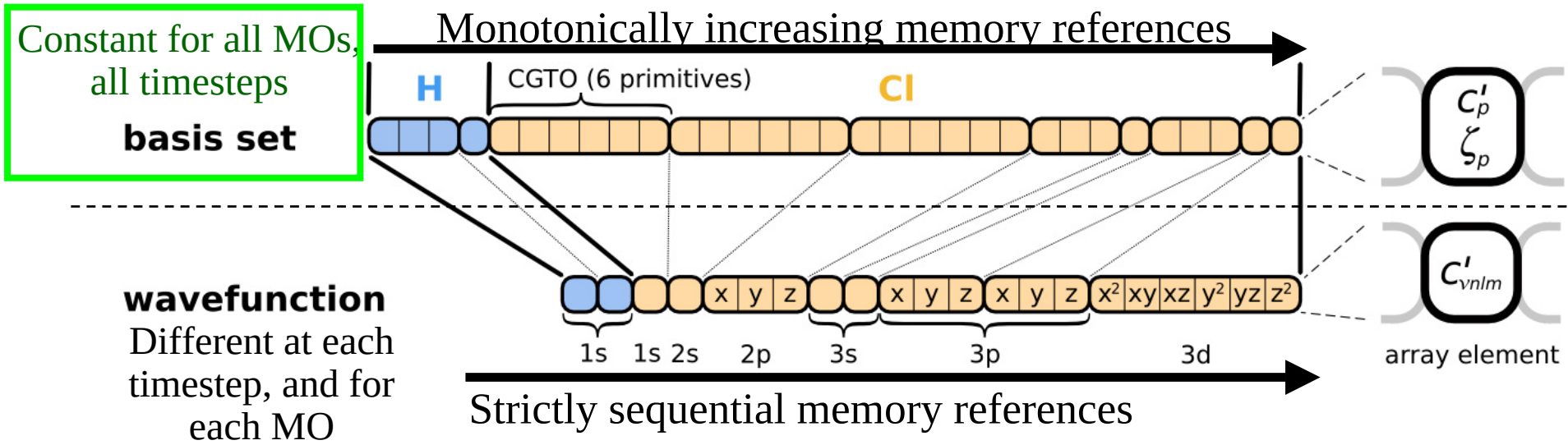
Loop unrolling:

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

Preprocessing of Atoms, Basis Set, and Wavefunction Coefficients

- Must make effective use of high bandwidth, low-latency GPU on-chip memory, or CPU cache:
  - Overall storage requirement reduced by eliminating duplicate basis set coefficients
  - Sorting atoms by element type allows re-use of basis set coefficients for subsequent atoms of identical type
- Padding, alignment of arrays guarantees coalesced GPU global memory accesses, CPU SSE loads

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

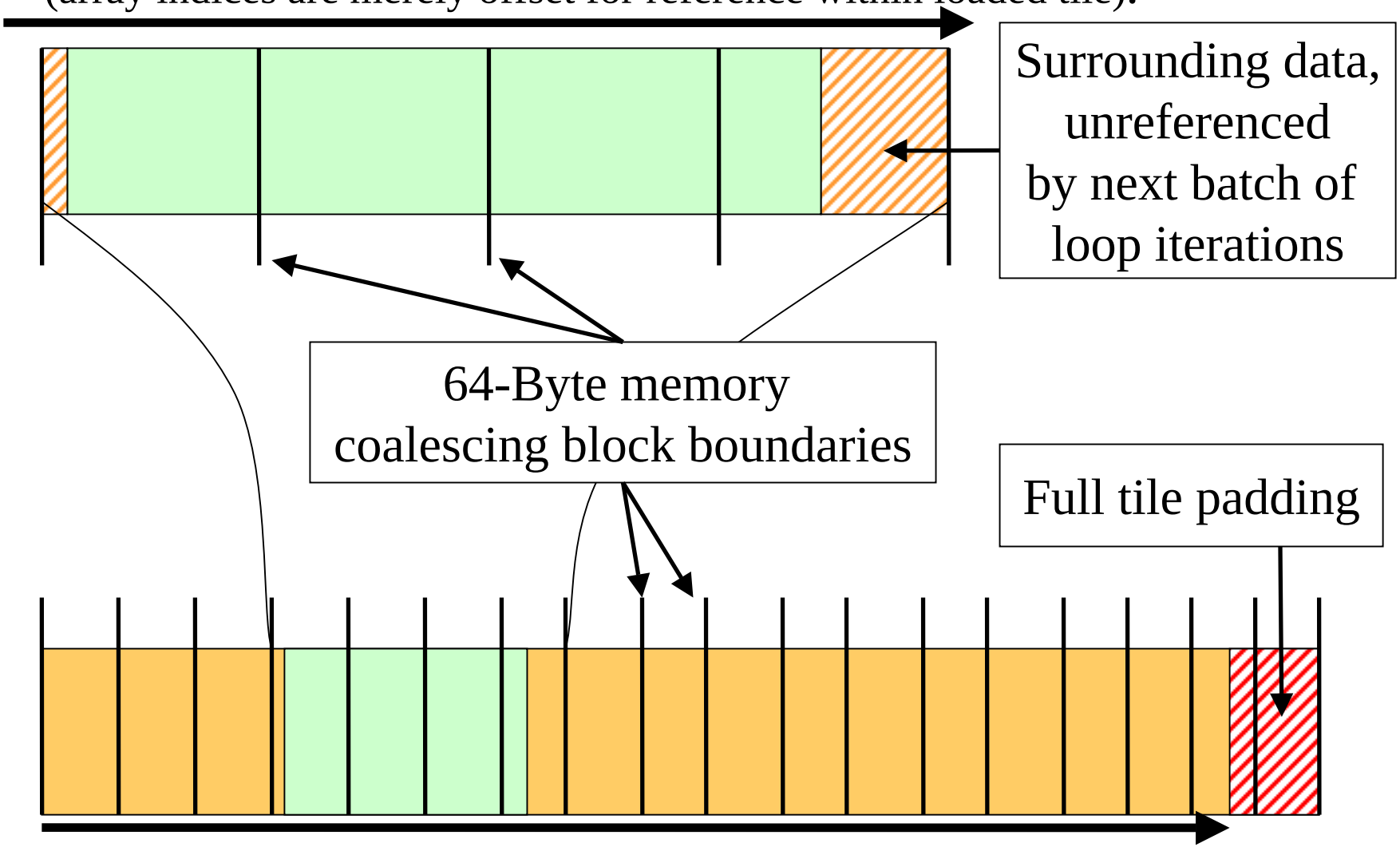


- Loop iterations always access same or consecutive array elements for all threads in a thread block:
  - Yields good constant memory cache performance
  - Increases shared memory tile reuse

Use of GPU On-chip Memory

- If total data less than 64 kB, use only const mem:
  - Broadcasts data to all threads, no global memory accesses!
- For large data, shared memory used as a program-managed cache, coefficients loaded on-demand:
  - Tiles sized large enough to service entire inner loop runs, broadcast to all 64 threads in a block
  - Complications: nested loops, multiple arrays, varying length
  - Key to performance is to locate tile loading checks outside of the two performance-critical inner loops
  - Only 27% slower than hardware caching provided by constant memory (GT200)
  - Next-gen “Fermi” GPUs will provide larger on-chip shared memory, L1/L2 caches, reduced control overhead

Array tile loaded in GPU shared memory. Tile size is a power-of-two, multiple of coalescing size, and allows simple indexing in inner loops (array indices are merely offset for reference within loaded tile).



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 …]
```

VMD MO Performance Results for C_{60}

Sun Ultra 24: Intel Q6600, NVIDIA GTX 280

Kernel	Cores/GPUs	Runtime (s)	Speedup
CPU ICC-SSE	1	46.58	1.00
CPU ICC-SSE	4	11.74	3.97
CPU ICC-SSE-approx**	4	3.76	12.4
CUDA-tiled-shared	1	0.46	100.
CUDA-const-cache	1	0.37	126.
CUDA-const-cache-JIT*	**1**	**0.27**	**173.** **(JIT 40% faster)**

C_{60} basis set 6-31Gd. We used an unusually-high resolution MO grid for accurate timings. A more typical calculation has $1/8^{th}$ the grid points.

* Runtime-generated JIT kernel compiled using batch mode CUDA tools

**Reduced-accuracy approximation of expf(), cannot be used for zero-valued MO isosurfaces

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)**

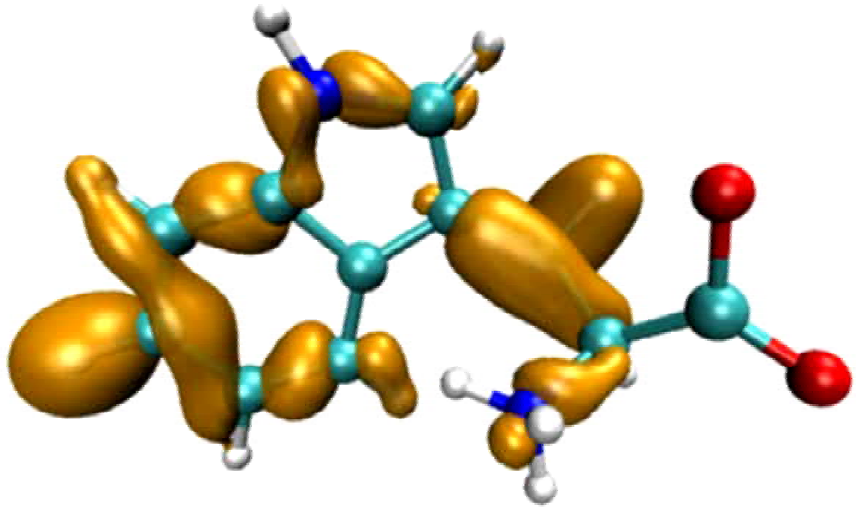
Kernel	**Cores GPUs**	Speedup vs. Molekel on 1 CPU core					
Molekel	1*	1.0	1.0	1.0	1.0	1.0	1.0
MacMolPlt	4	2.4	2.6	2.1	2.4	4.3	4.5
VMD GCC-cephes	4	3.2	4.0	3.0	3.5	4.3	6.5
VMD ICC-SSE-cephes	4	16.8	17.2	13.9	12.6	17.3	21.5
VMD ICC-SSE-approx**	4	59.3	53.4	50.4	49.2	54.8	69.8
VMD CUDA-const-cache	1	552.3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

VMD Orbital Dynamics Proof of Concept

One GPU can compute and animate this movie on-the-fly!

CUDA const-cache kernel,
Sun Ultra 24, GeForce GTX 285

GPU MO grid calc.	**0.016 s**
CPU surface gen, volume gradient, and GPU rendering	**0.033 s**
Total runtime	**0.049 s**
Frame rate	**20 FPS**

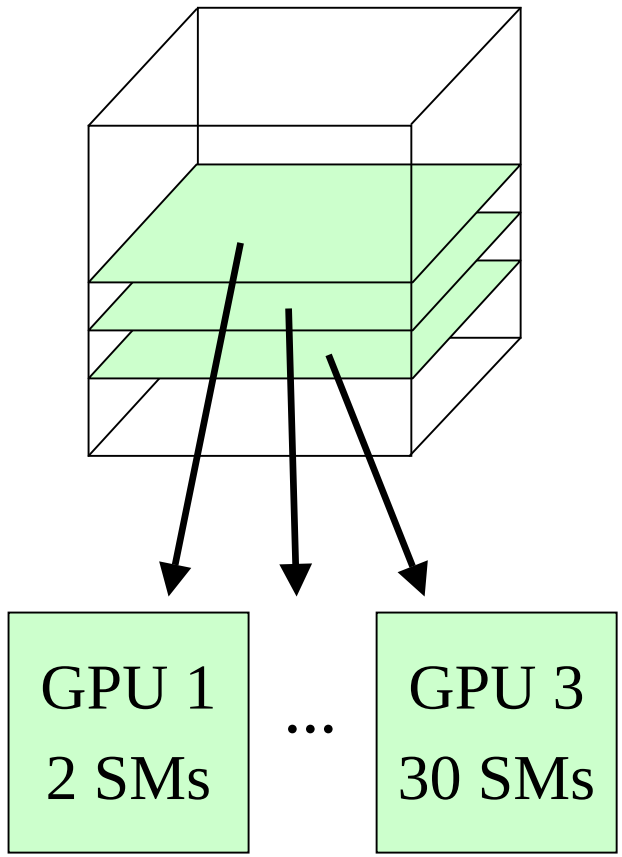


threonine

With GPU speedups over **100x**, previously insignificant CPU surface gen, gradient calc, and rendering are now **66%** of runtime. Need GPU-accelerated surface gen next…

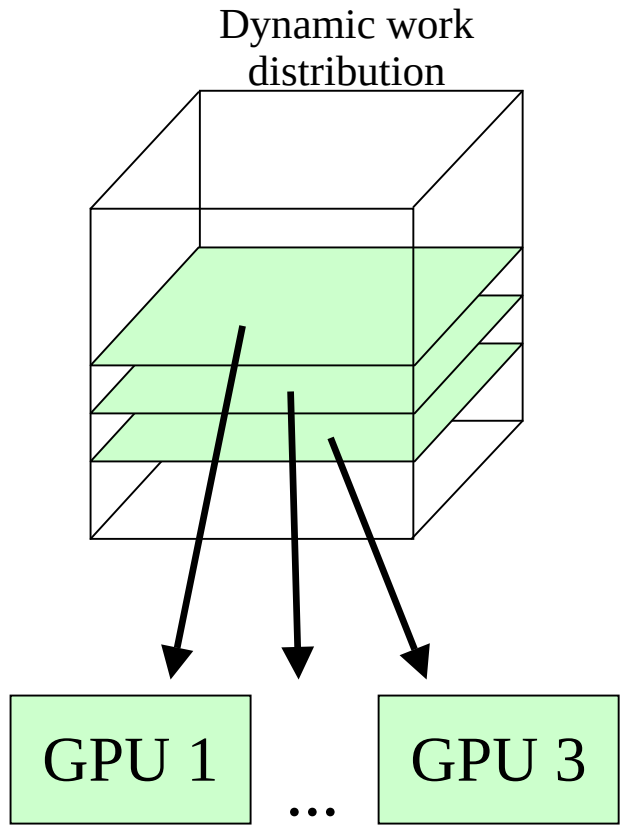
Multi-GPU Load Balance

- Many early CUDA codes assumed all GPUs were identical
- All new NVIDIA cards support CUDA, so a typical machine may have a diversity of GPUs of varying capability
- Static decomposition works poorly for non-uniform workload, or diverse GPUs, e.g. w/ 2 SM, 16 SM, 30 SM



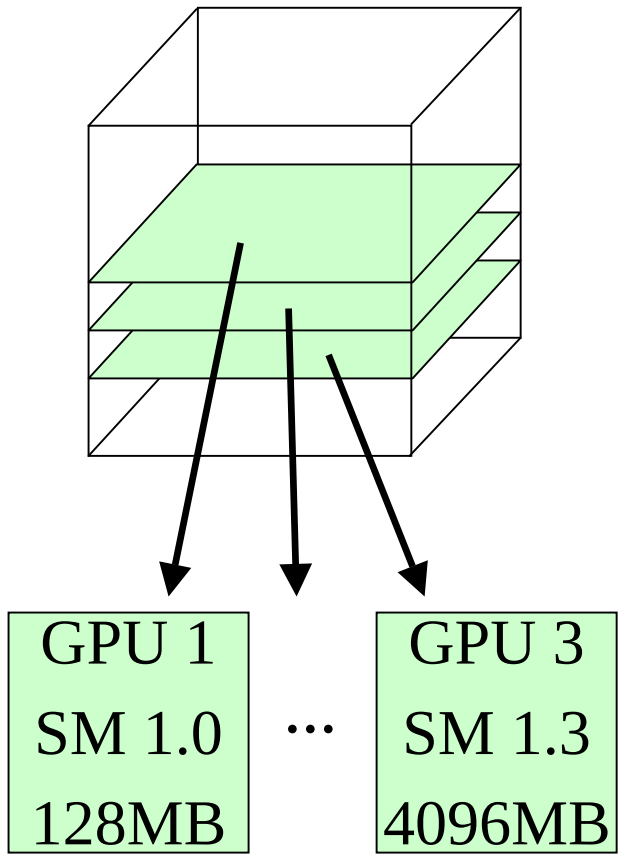
Multi-GPU Dynamic Work Distribution

```
// Each GPU worker thread loops over
// subset 2-D planes in a 3-D cube…
while (!threadpool_next_tile(&parms,
    tilesize, &tile)){
  // Process one plane of work…
  // Launch one CUDA kernel for each
  //   loop iteration taken…
  // Shared iterator automatically
  //   balances load on GPUs
}
```



Multi-GPU Runtime Error/Exception Handling

- Competition for resources from other applications or the windowing system can cause runtime failures (e.g. GPU out of memory half way through an algorithm)
- Handling of algorithm exceptions (e.g. convergence failure, NaN result, etc)
- Need to handle and/or reschedule failed tiles of work



Some Example Multi-GPU Latencies Relevant to Interactive Sci-Viz Apps

8.4us	CUDA empty kernel (immediate return)
10.0us	Sleeping barrier primitive (non-spinning barrier that uses POSIX condition variables to prevent idle CPU consumption while workers wait at the barrier)
20.3us	pool wake / exec / sleep cycle (no CUDA)
21.4us	pool wake / 1 x (tile fetch) / sleep cycle (no CUDA)
30.0us	pool wake / 1 x (tile fetch / CUDA nop kernel) / sleep cycle, test CUDA kernel computes an output address from its thread index, but does no output
1441.0us	pool wake / 100 x (tile fetch / CUDA nop kernel) / sleep cycle test CUDA kernel computes an output address from its thread index, but does no output

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

Kernel	Cores/GPUs	Runtime (s)	Speedup	Parallel Efficiency
CPU-ICC-SSE	1	46.580	1.00	100%
CPU-ICC-SSE	4	11.740	3.97	99%
CUDA-const-cache	1	0.417	112	100%
CUDA-const-cache	2	0.220	212	94%
CUDA-const-cache	3	0.151	308	92%
CUDA-const-cache	4	0.113	412	92%

Intel Q6600 CPU, 4x Tesla C1060 GPUs,

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

VMD Multi-GPU Molecular Orbital Performance Results for C_{60} Using Mapped Host Memory

Kernel	Cores/GPUs	Runtime (s)	Speedup
CPU-ICC-SSE	1	46.580	1.00
CPU-ICC-SSE	4	11.740	3.97
CUDA-const-cache	3	0.151	308.
CUDA-const-cache w/ mapped host memory	**3**	**0.137**	**340.**

Intel Q6600 CPU, 3x Tesla C1060 GPUs,

GPU kernel writes output directly to host memory, no extra cudaMemcpy() calls to fetch results!

See cudaHostAlloc() + cudaGetDevicePointer()

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**

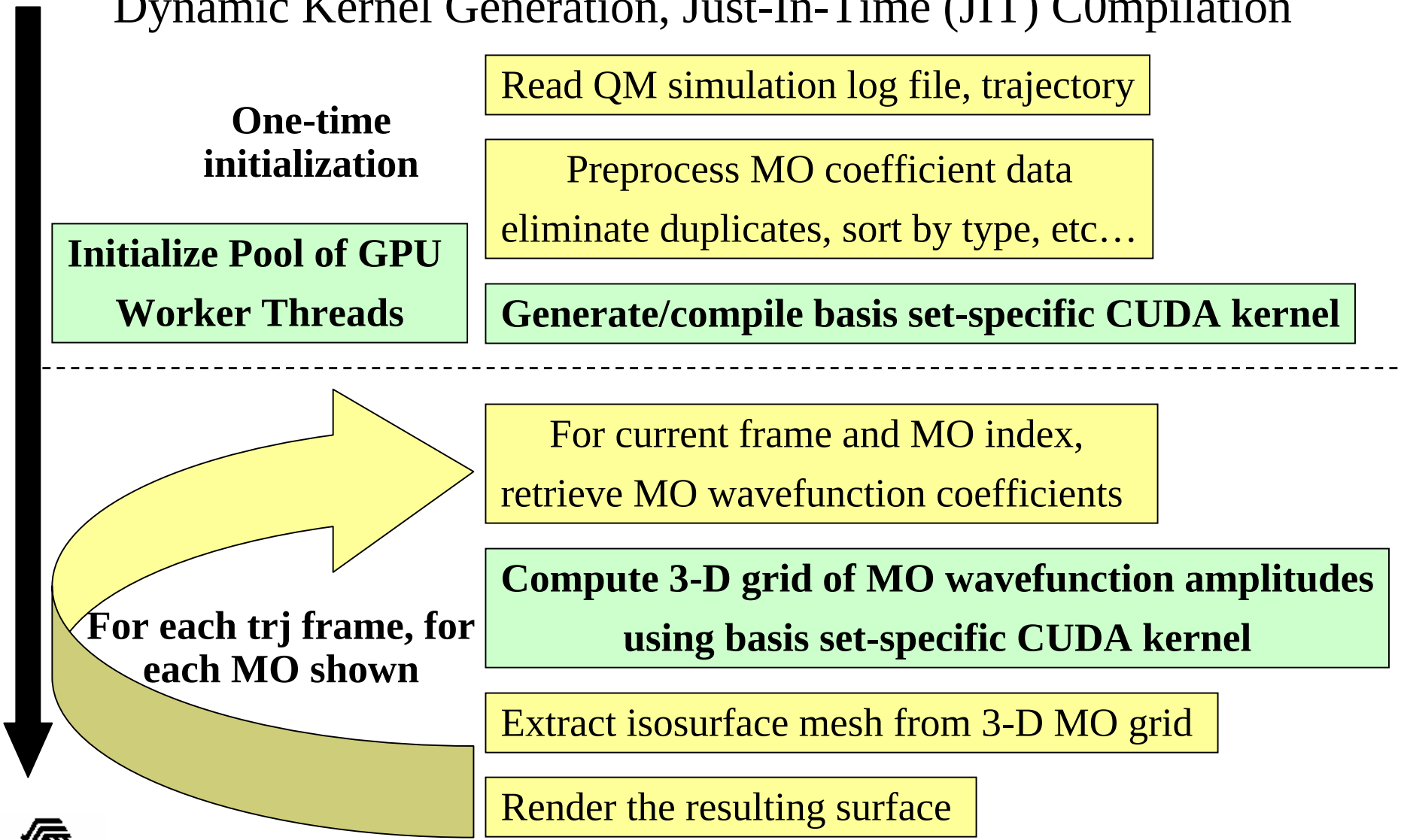
}

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



```
…..
// 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 shell_type = const_shell_symmetry[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 * exp2f(-exponent*dist2);
   prim_counter += 2;
  }

  /* multiply with the appropriate wavefunction coefficient */
  float tmpshell=0;
  switch (shell_type) {
   case S_SHELL:
    value += const_wave_f[ifunc++] * contracted_gto;
    break;
[…..]
   case D_SHELL:
    tmpshell += const_wave_f[ifunc++] * xdist2;
    tmpshell += const_wave_f[ifunc++] * ydist2;
    tmpshell += const_wave_f[ifunc++] * zdist2;
    tmpshell += const_wave_f[ifunc++] * xdist * ydist;
    tmpshell += const_wave_f[ifunc++] * xdist * zdist;
    tmpshell += const_wave_f[ifunc++] * ydist * zdist;
    value += tmpshell * contracted_gto;
    break;
```

General loop-based CUDA kernel

Dynamically-generated CUDA kernel (JIT)

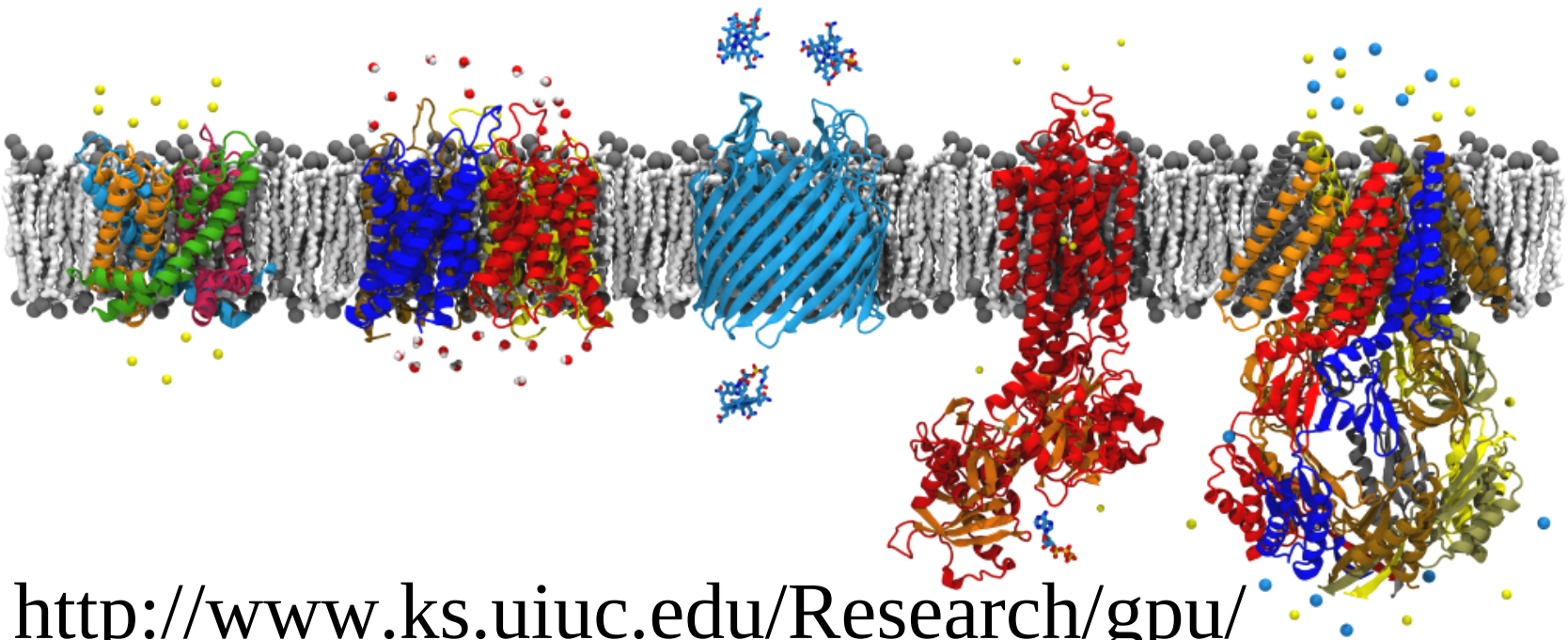
```
…..
contracted_gto = 1.832937 * expf(-7.868272*dist2);
contracted_gto += 1.405380 * expf(-1.881289*dist2);
contracted_gto += 0.701383 * expf(-0.544249*dist2);
// P_SHELL
tmpshell = const_wave_f[ifunc++] * xdist;
tmpshell += const_wave_f[ifunc++] * ydist;
tmpshell += const_wave_f[ifunc++] * zdist;
value += tmpshell * contracted_gto;

contracted_gto = 0.187618 * expf(-0.168714*dist2);
// S_SHELL
value += const_wave_f[ifunc++] * contracted_gto;

contracted_gto = 0.217969 * expf(-0.168714*dist2);
// P_SHELL
tmpshell = const_wave_f[ifunc++] * xdist;
tmpshell += const_wave_f[ifunc++] * ydist;
tmpshell += const_wave_f[ifunc++] * zdist;
value += tmpshell * contracted_gto;

contracted_gto = 3.858403 * expf(-0.800000*dist2);
// D_SHELL
tmpshell = const_wave_f[ifunc++] * xdist2;
tmpshell += const_wave_f[ifunc++] * ydist2;
tmpshell += const_wave_f[ifunc++] * zdist2;
tmpshell += const_wave_f[ifunc++] * xdist * ydist;
tmpshell += const_wave_f[ifunc++] * xdist * zdist;
tmpshell += const_wave_f[ifunc++] * ydist * zdist;
value += tmpshell * contracted_gto;
```

NAMD: Molecular Dynamics on GPUs

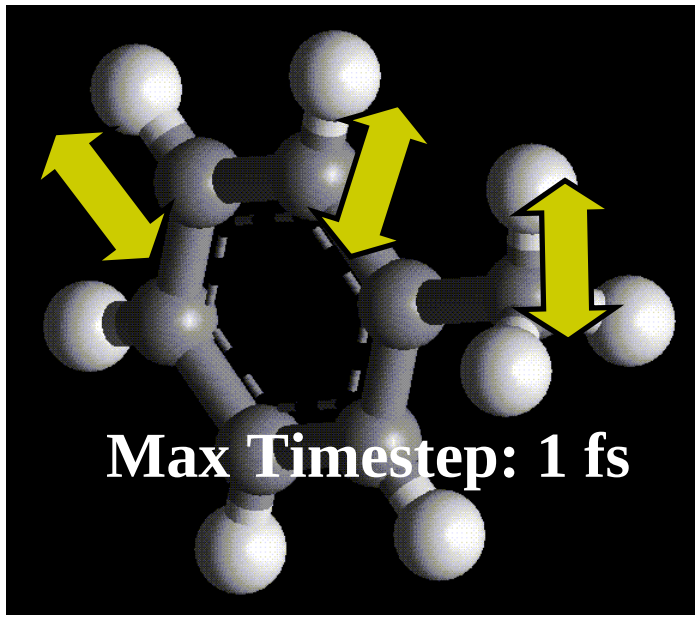


http://www.ks.uiuc.edu/Research/gpu/

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

Biomolecular Time Scales

Motion	Time Scale (sec)
Bond stretching	10^{-14} to 10^{-13}
Elastic vibrations	10^{-12} to 10^{-11}
Rotations of surface sidechains	10^{-11} to 10^{-10}
Hinge bending	10^{-11} to 10^{-7}
Rotation of buried side chains	10^{-4} to 1 sec
Allosteric transistions	10^{-5} to 1 sec
Local denaturations	10^{-5} to 10 sec

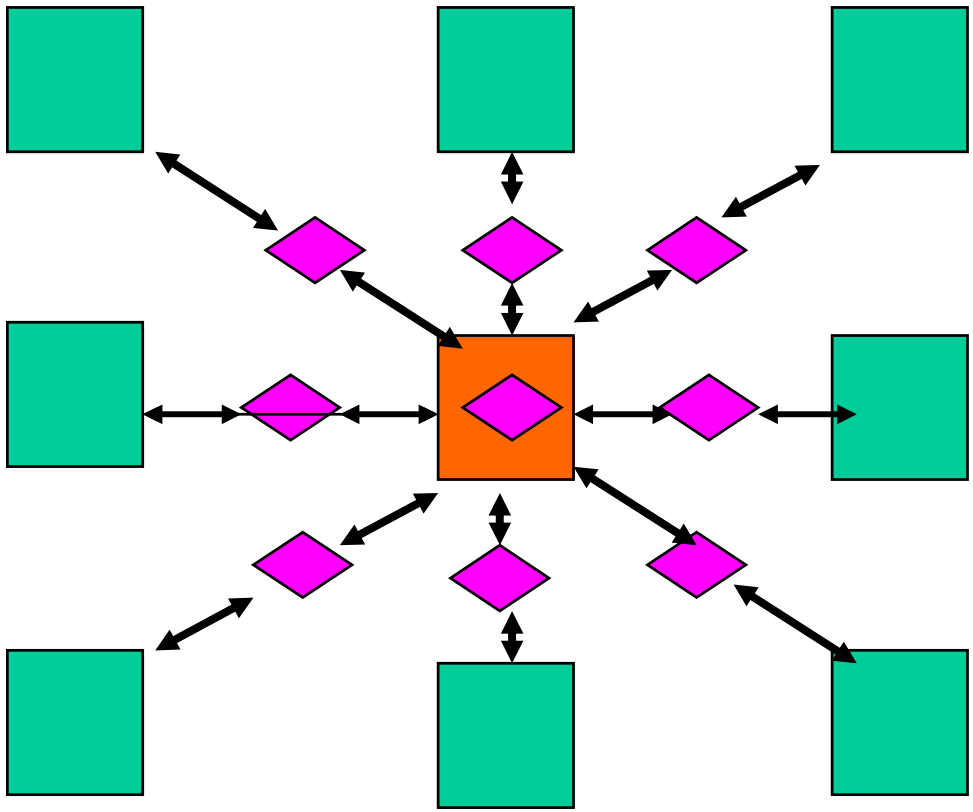


Typical Simulation Statistics

- 100,000 atoms (including water, lipid)
- 10-20 MB of data for entire system
- 100 Å per side periodic cell
- 12 Å cutoff of short-range nonbonded terms
- 10,000,000 timesteps (10 ns)
- 3 s/step on one processor (1 year total!)

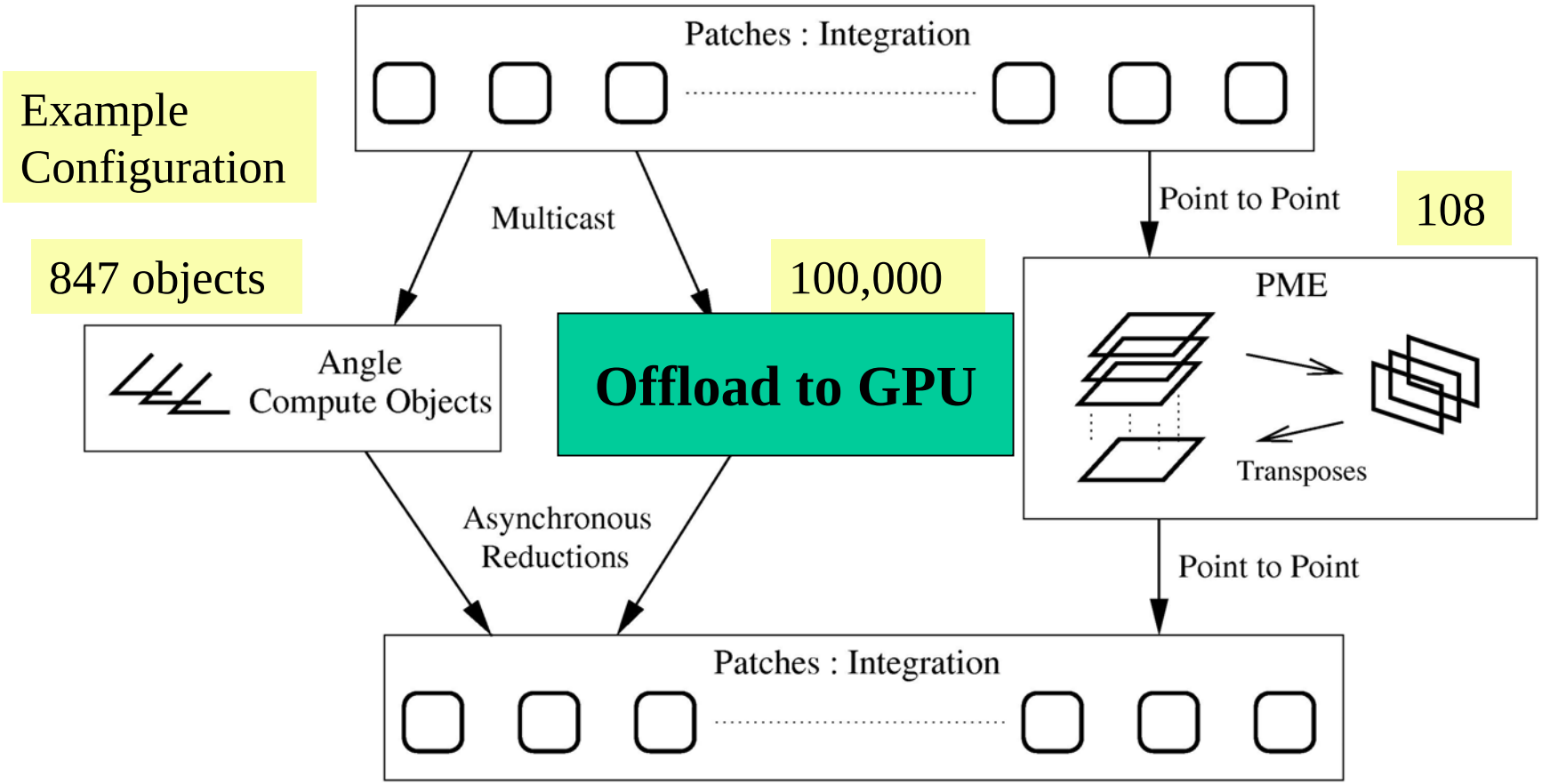
NAMD Hybrid Parallel Decomposition

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



- Spatially decompose data and communication.
- Separate but related work decomposition.
- "Compute objects" facilitate iterative, measurement-based load balancing system.

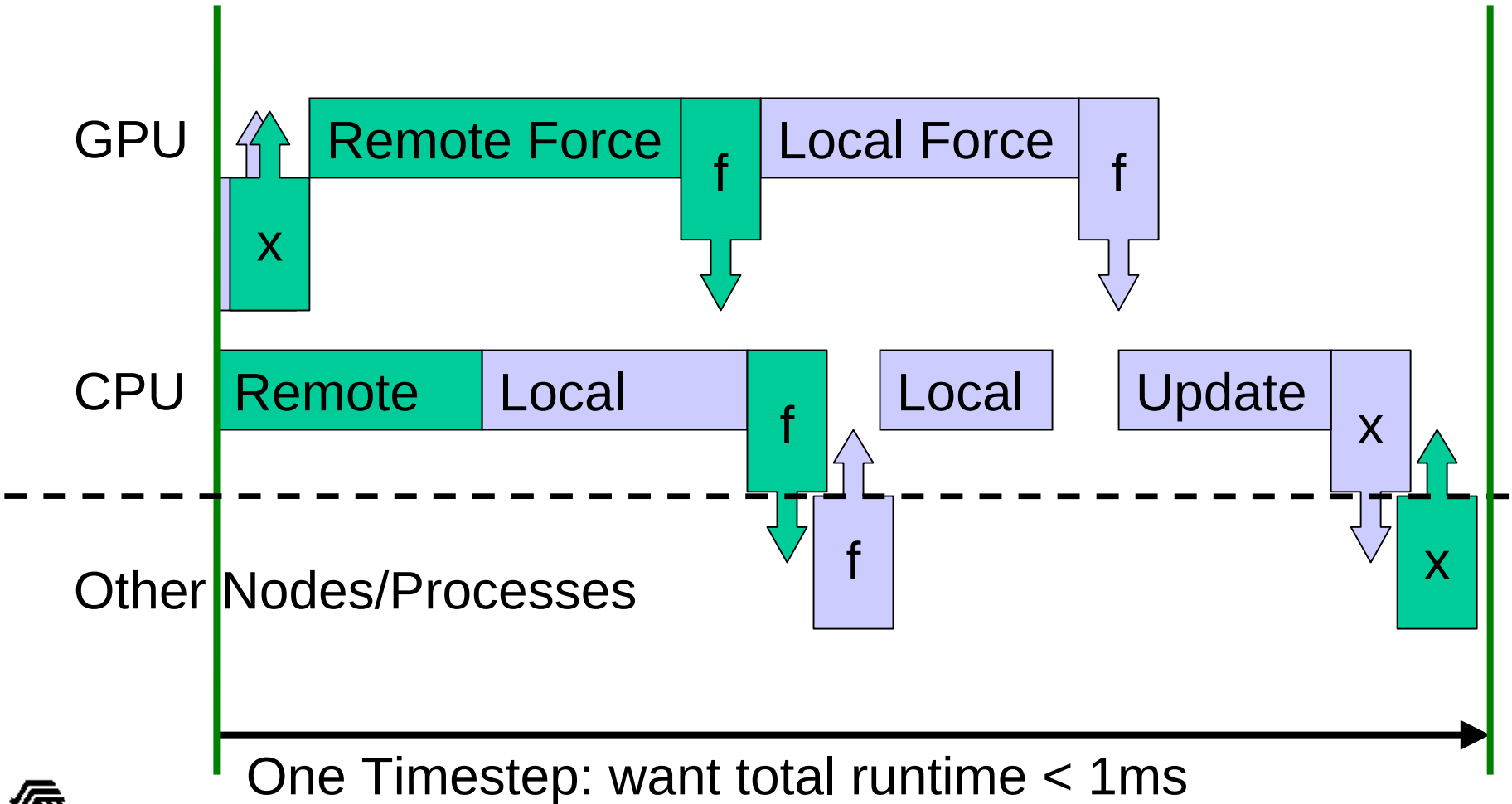
NAMD Parallel Molecular Dynamics: Overlapping CPU/GPU Execution



Objects are assigned to processors and queued as data arrives.

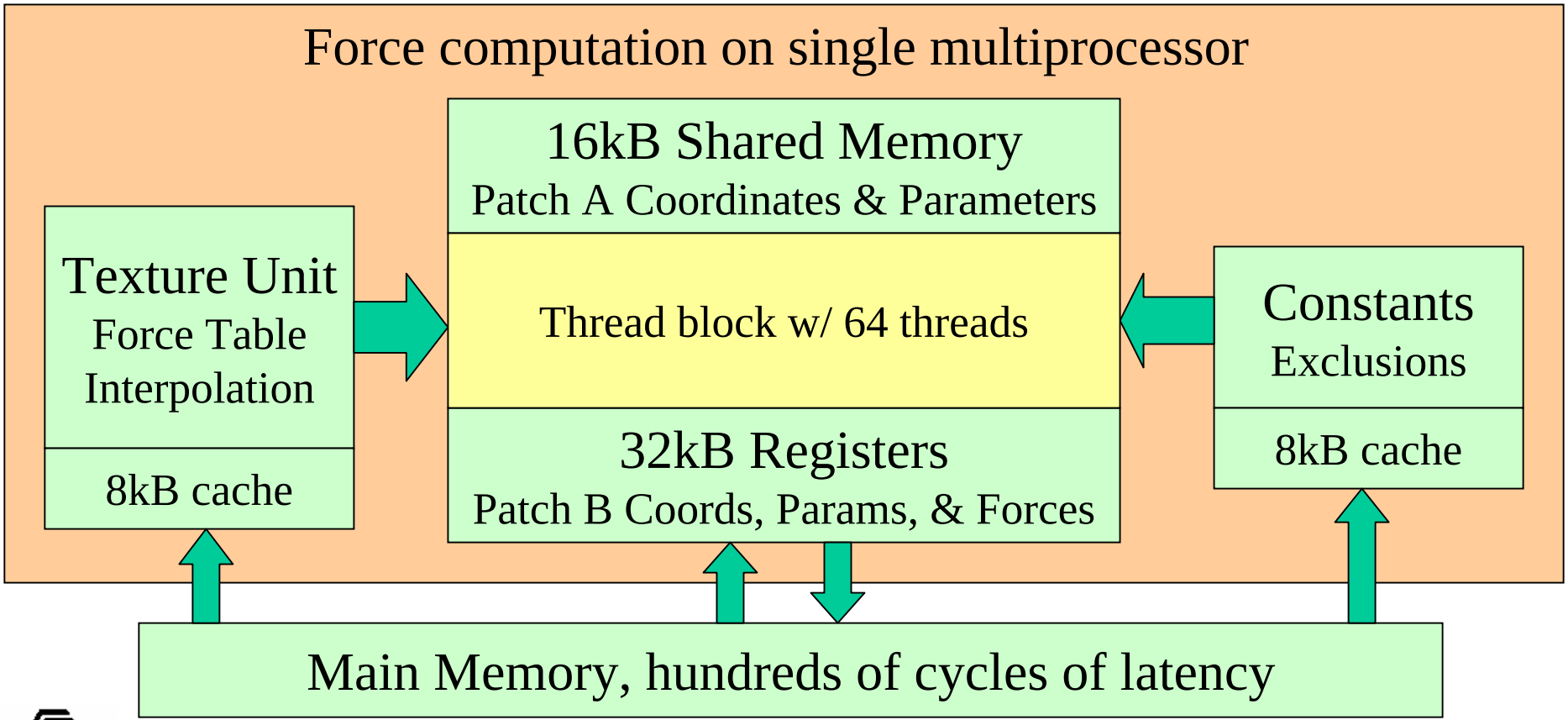
Phillips *et al.*, *SC2002*. Phillips *et al.*, *SC2008*.

Overlapping GPU/CPU Computation with Communication



Nonbonded Forces on CUDA GPU

- Most expensive calculation (~95% runtime in CPU versions)
- Work decomposed into patch pairs, as in regular NAMD



Stone *et al.*, *J. Comp. Chem.* **28**:2618-2640, 2007.

Each Block Gets a Pair of Patches

- Block-level constants in shared memory to save registers
- patch_pair array is 16-byte aligned
- To coalesce reads each thread loads one int from global memory and writes it into a union in shared memory

```
#define myPatchPair pp.pp
 __shared__ union { patch_pair pp; unsigned int i[8]; } pp;
 __shared__ bool same_patch;
 __shared__ bool self_force;

 if ( threadIdx.x < (sizeof(patch_pair)>>2) ) {
   unsigned int tmp = ((unsigned int*)patch_pairs)[
                                        (sizeof(patch_pair)>>2)*blockIdx.x+threadIdx.x];
   pp.i[threadIdx.x] = tmp;
 }
 __syncthreads();
 // now all threads can access myPatchPair safely
```

Coalesced Loading of Atom Data

- Want to copy two 16-byte structs per thread from global to shared memory
- Global memory access should be aligned on 16-byte boundaries to be coalesced
- 16-byte structs in shared memory cause bank conflicts, 36-byte structs do not

Right-Sized Atom Data Structures

```
struct __align__(16) atom {  // must be multiple of 16!
  float3 position;
  float charge;
};

struct __align__(16) atom_param {  // must be multiple of 16!
  float sqrt_epsilon;
  float half_sigma;
  unsigned int index;
  unsigned short excl_index;
  unsigned short excl_maxdiff;
};

struct shared_atom {  // do not align, size 36 to avoid bank conflicts
  float3 position;
  float charge;
  float sqrt_epsilon;
  float half_sigma;
  unsigned int index;
  unsigned int excl_index;
  unsigned int excl_maxdiff;
};
```

Texture Unit Force Interpolation

- rsqrt() is implemented in hardware
- $\mathbf{F(r^{-1})/r = \varepsilon(\sigma^{12}A(r^{-1}) + \sigma^{6}B(r^{-1})) + qqC(r^{-1})}$
- $F = r\ \mathbf{F(r^{-1})/r}$
- Piecewise linear interpolation of A,B,C
  - F(r) is linear since $\mathbf{r\ (a\ r^{-1} + b) = a + r\ b}$
- Texture unit hardware is a perfect match

Const Memory Exclusion Tables

- Need to exclude bonded pairs of atoms
  - Also apply correction for PME electrostatics
- Exclusions determined by using atom indices to bit flags in exclusion arrays
- Repetitive molecular structures limit unique exclusion arrays
- All exclusion data fits in constant cache

Overview of Inner Loop

- Calculate forces on atoms in registers due to atoms in shared memory
  - Ignore Newton's 3rd law (reciprocal forces)
  - Do not sum forces for atoms in shared memory
- All threads access the same shared memory atom, allowing shared memory broadcast
- Only calculate forces for atoms within cutoff distance (roughly 10% of pairs)

Nonbonded Forces CUDA Code

```
texture<float4> force_table;
__constant__ unsigned int exclusions[];
__shared__ atom jatom[];
atom iatom;      // per-thread atom, stored in registers
float4 iforce;   // per-thread force, stored in registers
for ( int j = 0; j < jatom_count; ++j ) {
  float dx = jatom[j].x - iatom.x;   float dy = jatom[j].y - iatom.y;  float dz = jatom[j].z - iatom.z;
  float r2 = dx*dx + dy*dy + dz*dz;
  if ( r2 < cutoff2 ) {
    float4 ft = texfetch(force_table, 1.f/sqrt(r2));
    bool excluded = false;
    int indexdiff = iatom.index - jatom[j].index;
    if ( abs(indexdiff) <= (int) jatom[j].excl_maxdiff ) {
      indexdiff += jatom[j].excl_index;
      excluded = ((exclusions[indexdiff>>5] & (1<<(indexdiff&31))) != 0);
    }
    float f = iatom.half_sigma + jatom[j].half_sigma;  // sigma
    f *= f*f;  // sigma^3
    f *= f;  // sigma^6
    f *= ( f * ft.x + ft.y );  // sigma^12 * fi.x - sigma^6 * fi.y
    f *= iatom.sqrt_epsilon * jatom[j].sqrt_epsilon;
    float qq = iatom.charge * jatom[j].charge;
    if ( excluded ) { f = qq * ft.w; }  // PME correction
    else { f += qq * ft.z; }  // Coulomb
    iforce.x += dx * f;   iforce.y += dy * f;    iforce.z += dz * f;
    iforce.w += 1.f;  // interaction count or energy
  }
}
```

Force Interpolation

Exclusions

Parameters

Accumulation

Stone *et al., J. Comp. Chem.* **28**:2618-2640, 2007.

What About Warp Divergence?

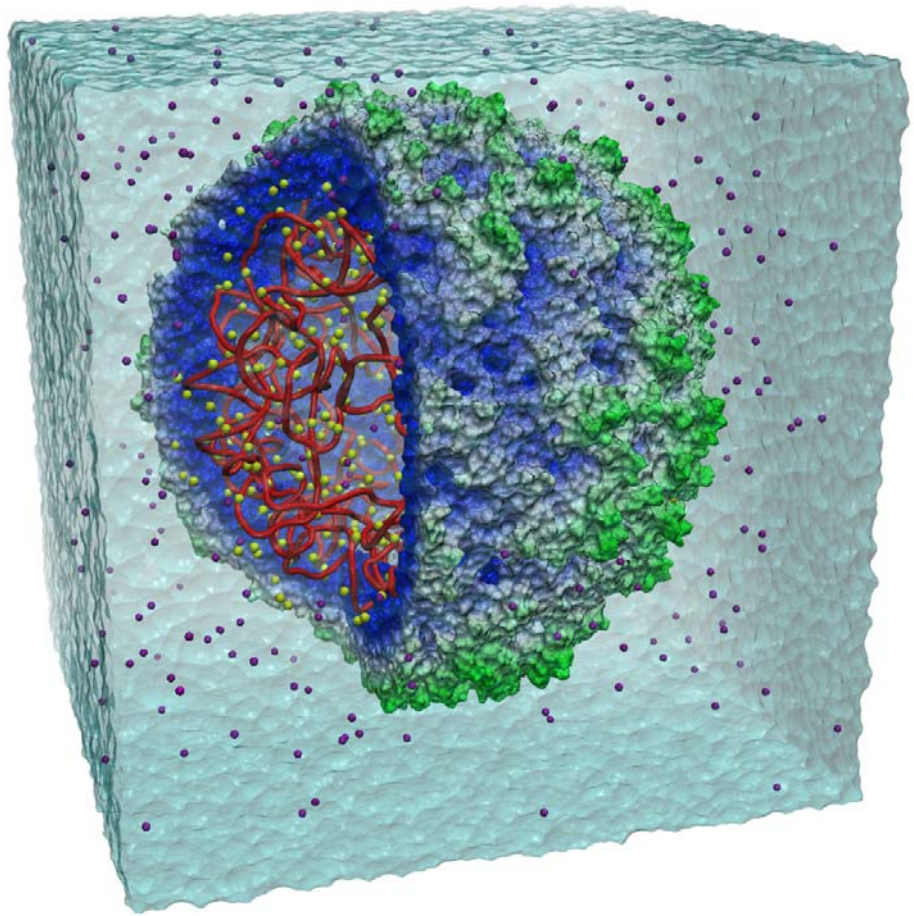
- Almost all exclusion checks fail, and the extra work for an excluded pair is minimal
- Cutoff test isn't completely random
  - Hydrogens follow their heavy atoms
  - Atoms in far corners of patches have few neighbors within cutoff distance

Recent NAMD GPU Developments

- Features:
  - Full electrostatics with PME
  - Multiple timestepping
  - 1-4 Exclusions
  - Constant-pressure simulation
- Improved force accuracy:
  - Patch-centered atom coordinates
  - Increased precision of force interpolation
- GPU sharing with coordination via message passing
- Next-gen “Fermi” GPUs:
  - Double precision force computations will be almost “free”
  - Larger shared memory, increased effective memory bandwidth
  - Potential for improved overlap of “local” and “remote” work units

NAMD Beta 2 Coming Soon

- Nightly builds of CUDA binaries for 64-bit Linux are available on the NAMD web site now…



National Center for Research Resources

Beckman Institute, UIUC

Acknowledgements

- Additional Information and References:
  - http://www.ks.uiuc.edu/Research/gpu/
- Questions, source code requests:
  - John Stone: johns@ks.uiuc.edu
- Acknowledgements:
  - J. Phillips, D. Hardy, J. Saam,
    UIUC Theoretical and Computational Biophysics Group,
    NIH Resource for Macromolecular Modeling and Bioinformatics
  - Prof. Wen-mei Hwu, Christopher Rodrigues, UIUC IMPACT Group
  - CUDA team at NVIDIA
  - UIUC NVIDIA CUDA Center of Excellence
  - NIH support: P41-RR05969

Publications

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

- Probing Biomolecular Machines with Graphics Processors. J. Phillips, J. Stone. *Communications of the ACM*, 52(10):34-41, 2009.
- 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)*, IEEE Cluster 2009. In press.
- 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.
- Multilevel summation of electrostatic potentials using graphics processing units. D. Hardy, J. Stone, K. Schulten. *J. Parallel Computing*, 35:164-177, 2009.

Publications (cont)

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.