# S04: High Performance Computing with CUDA

## Case Study:
## Molecular Modeling Applications

John E. Stone

Theoretical and Computational Biophysics Group

Beckman Institute for Advanced Science and Technology

University of Illinois at Urbana-Champaign

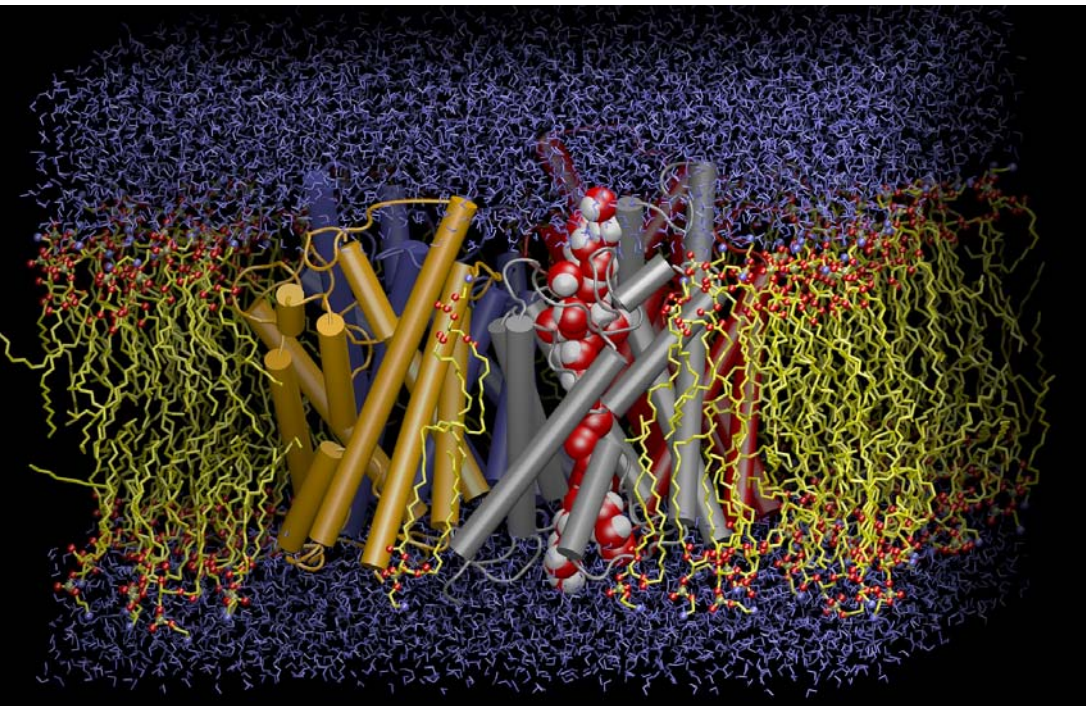
**http://www.ks.uiuc.edu/Research/gpu/**

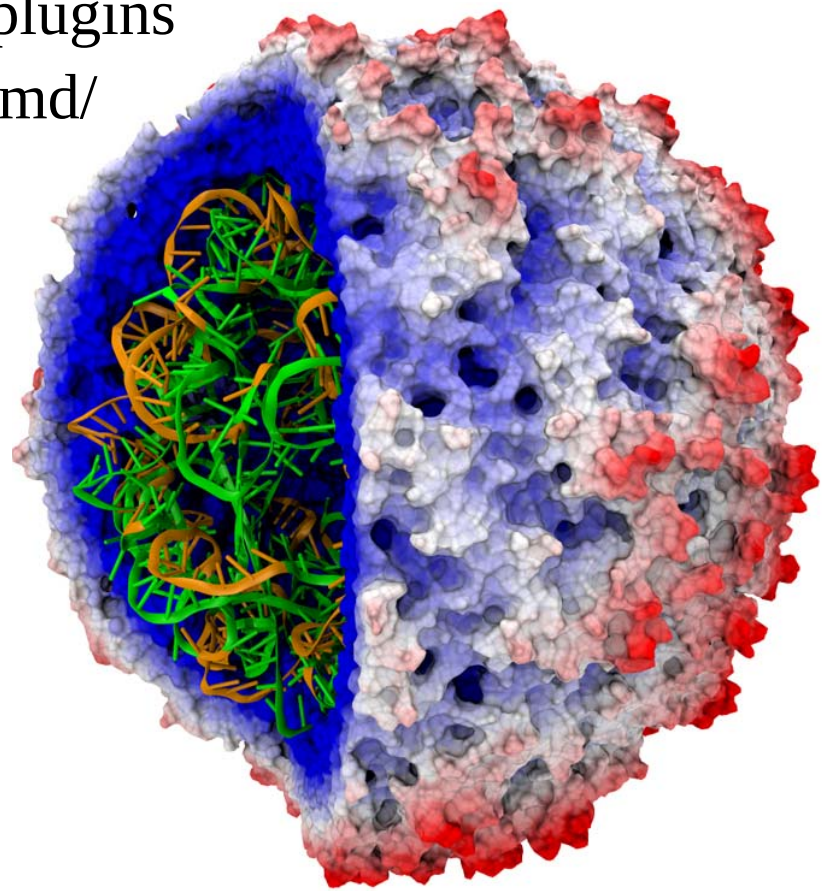
Tutorial S04, Supercomputing 2009,

Portland, OR, Nov 15, 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/





# Case Study Topics:

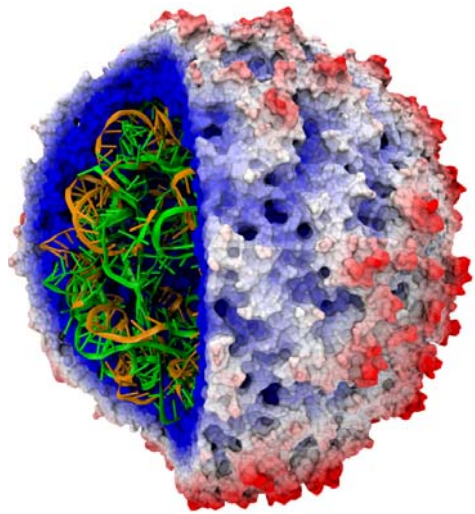
- VMD – molecular visualization + analysis
- NAMD – molecular dynamics simulation
- See our GPU article in CACM issue included in your SC2009 registration goodies bag…
- See live demos in NVIDIA booth
- Klaus Schulten Masterworks Lecture:

**"Fighting Swine Flu through Computational Medicine"**

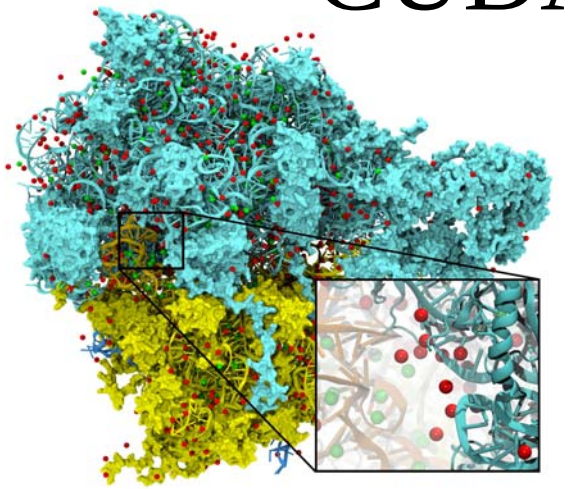
**Wednesday, 04:15PM - 05:00PM**

Room **PB253-254**

**Includes GPU perf results + scientific applications**

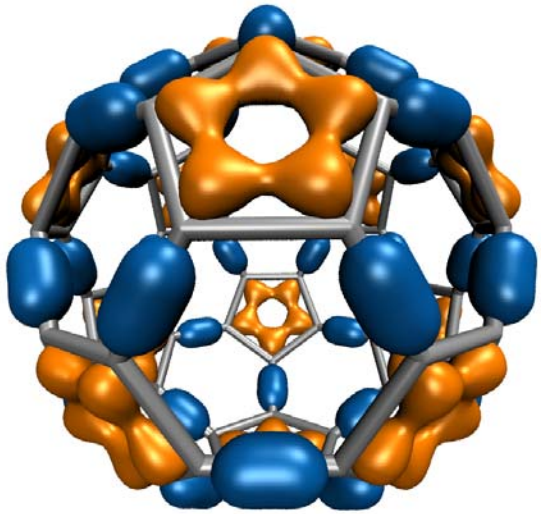


# CUDA Acceleration in VMD



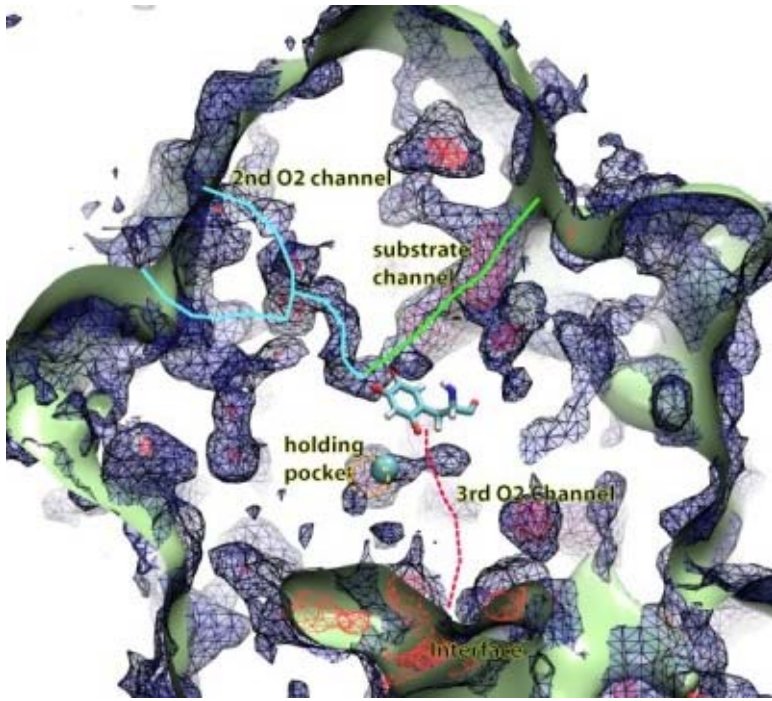
Electrostatic field calculation, ion placement

20x to 44x faster



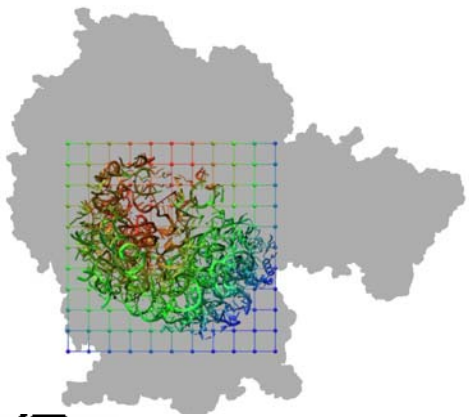
Molecular orbital calculation and display

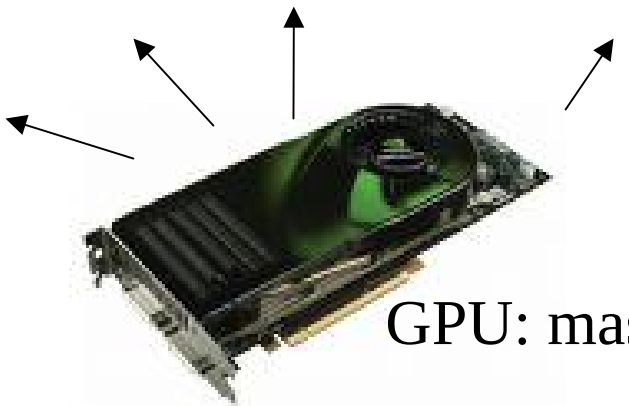
100x to 120x faster




Imaging of gas migration pathways in proteins with implicit ligand sampling

20x to 30x faster





GPU: massively parallel co-processor

# Recurring Algorithm Design Principles

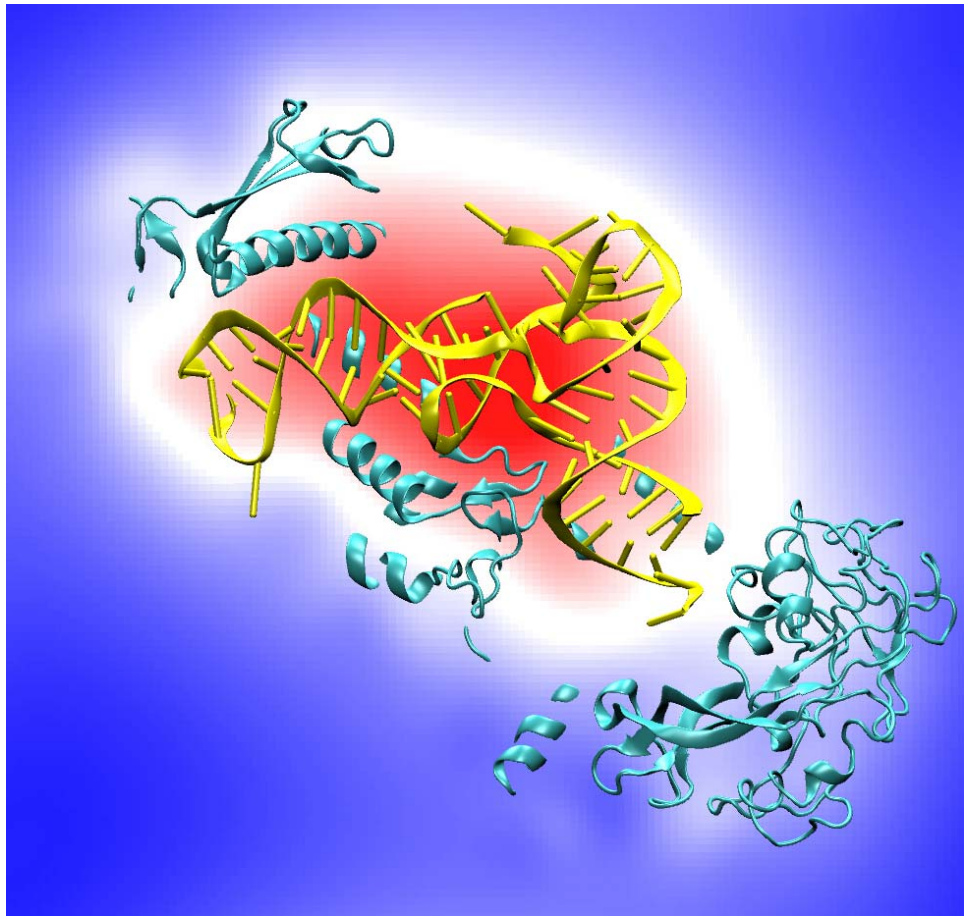
- Pre-processing and sorting of operands to organize computation for peak efficiency on the GPU
- Tiled/blocked data structures in GPU global memory for peak bandwidth utilization
- Extensive use of on-chip shared memory and constant memory to further amplify memory bandwidth
- Use of CPU to “regularize” the work done by the GPU, handle exceptions & unusual work units
- Asynchronous operation of CPU/GPU enabling overlapping of computation and I/O on both ends

# 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

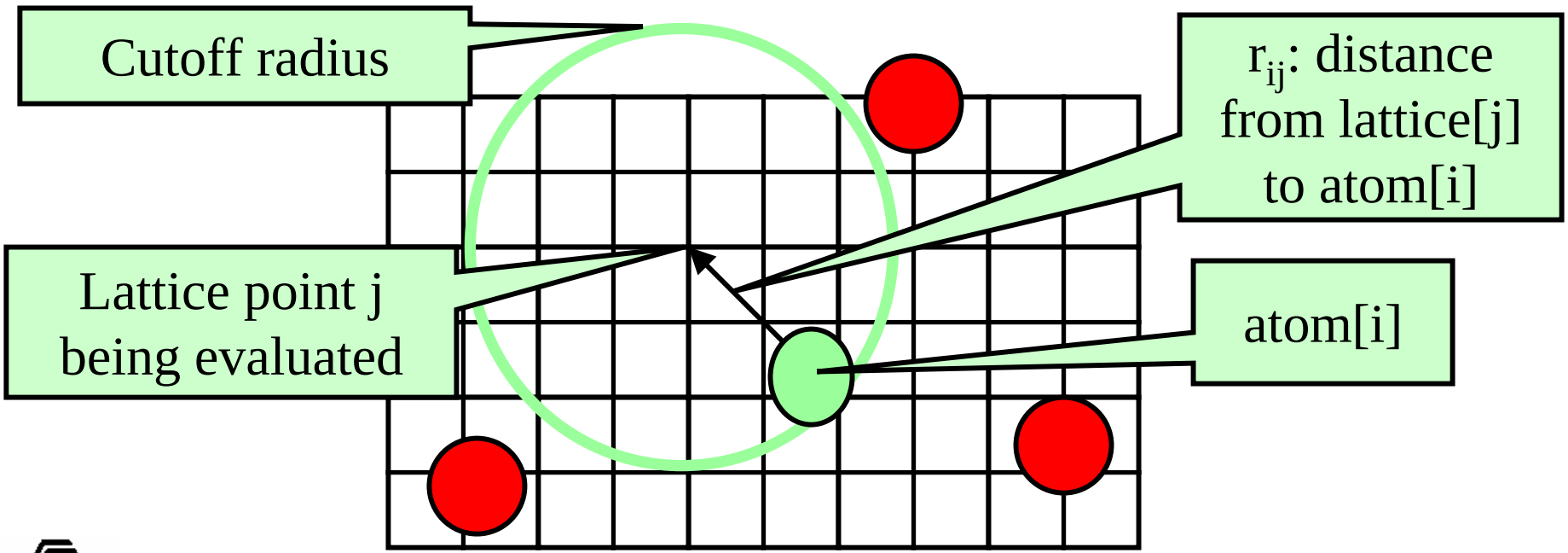
# Short-range Cutoff Summation

- 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)

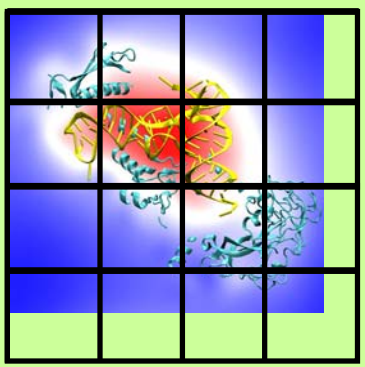
GPU thread block calculates corresponding region of potential map,

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

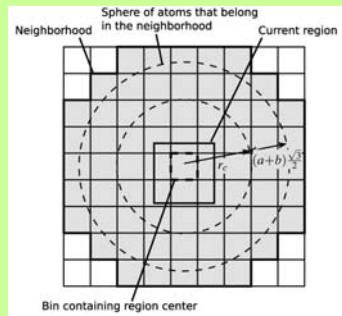
Potential map regions

Bins of atoms

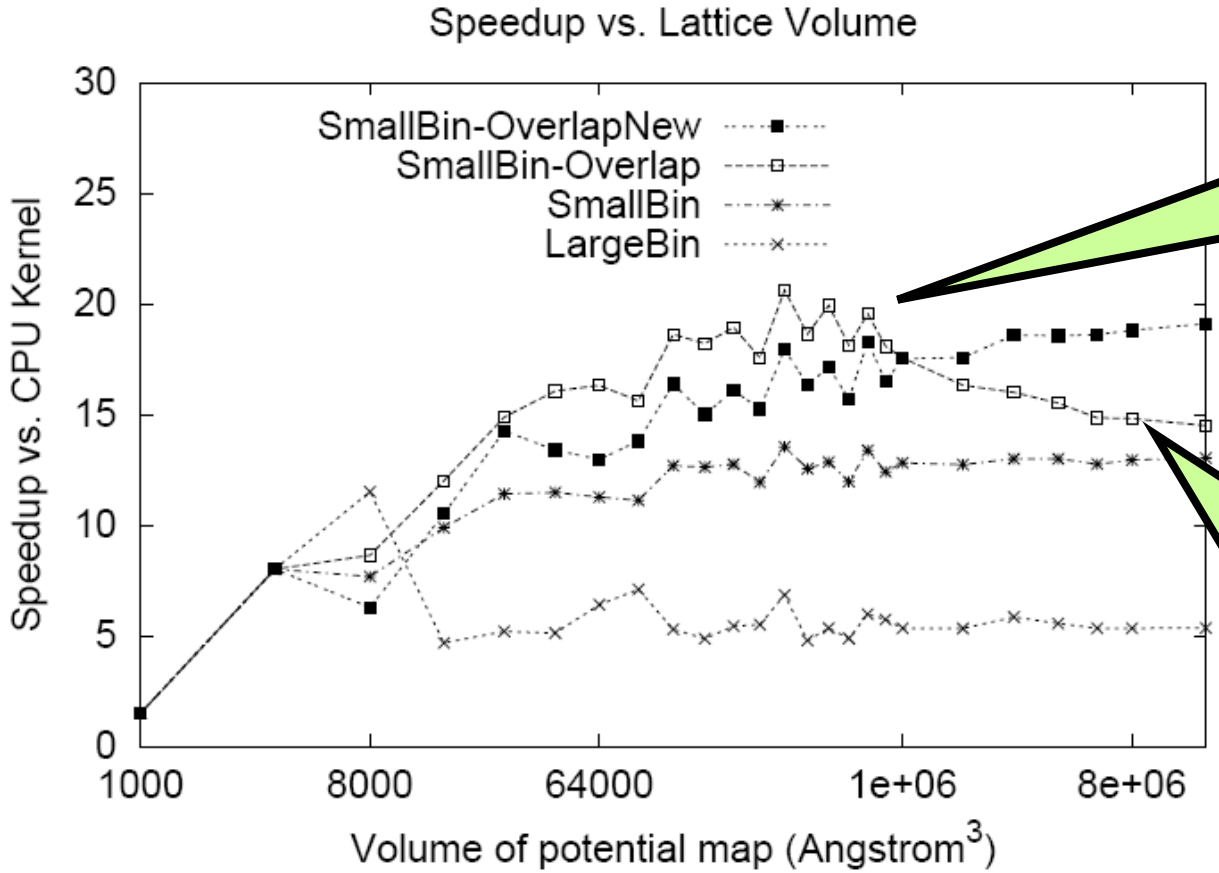
Constant memory

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…

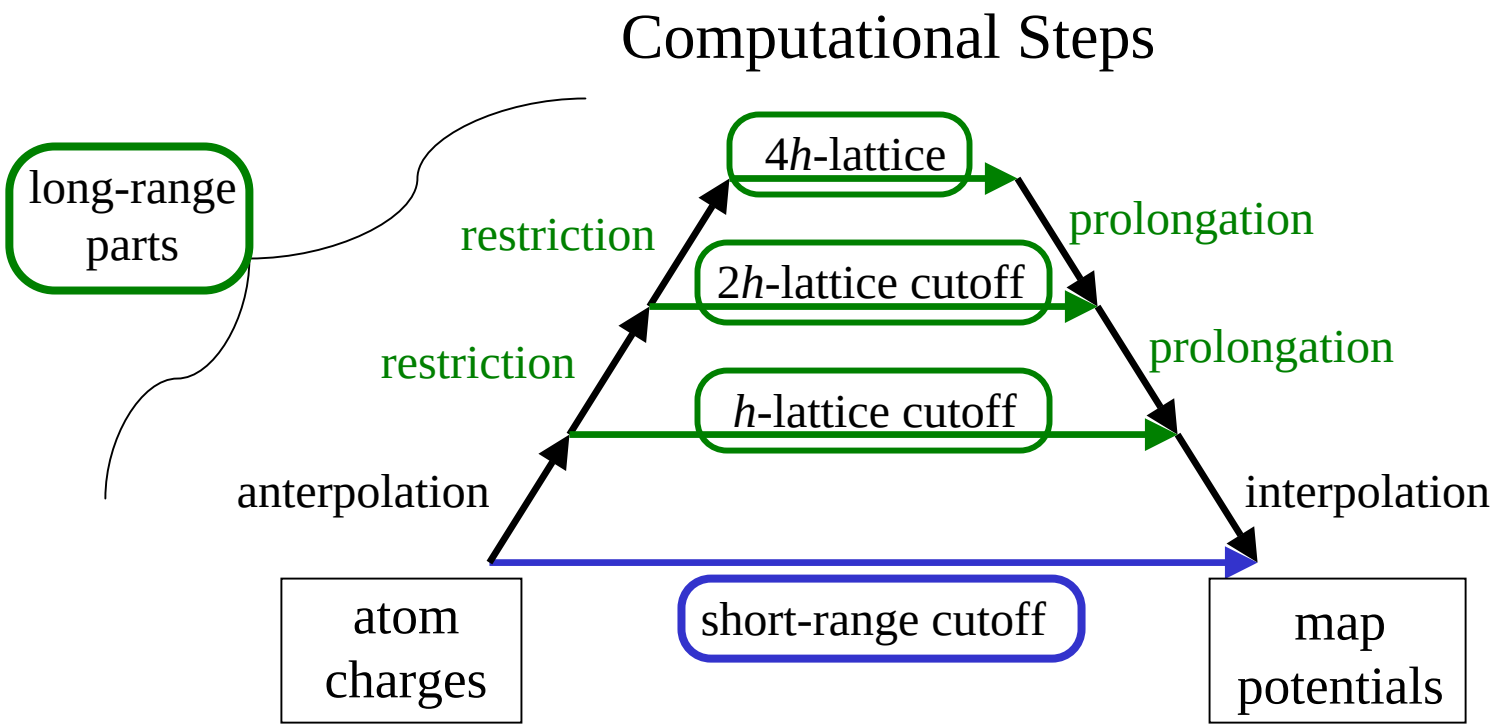
GPU acceleration of cutoff pair potentials for molecular modeling applications. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.

# 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 Calculation

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

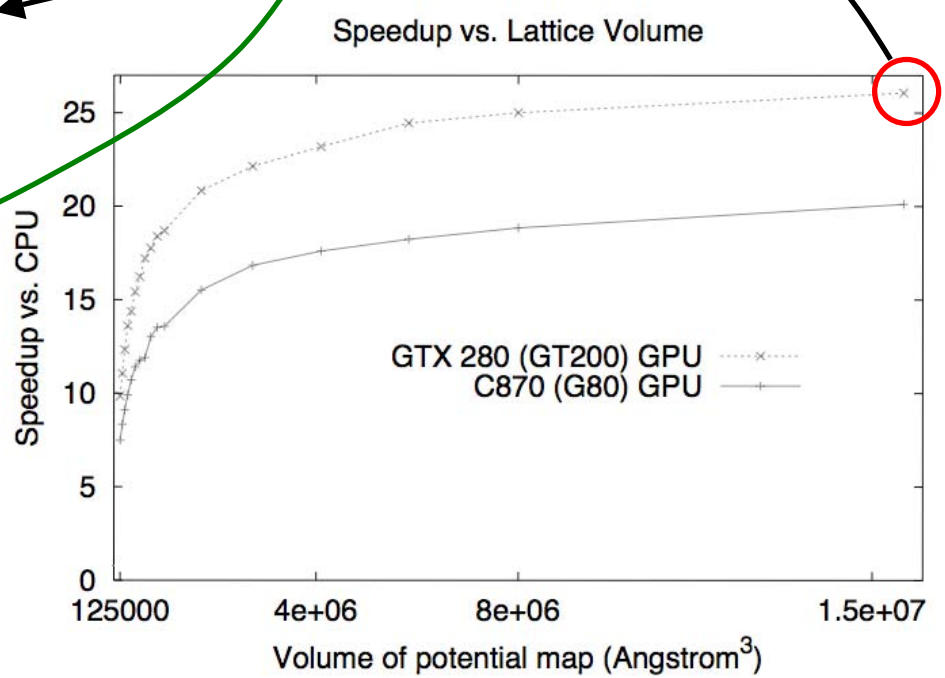


# Multilevel Summation on the GPU

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

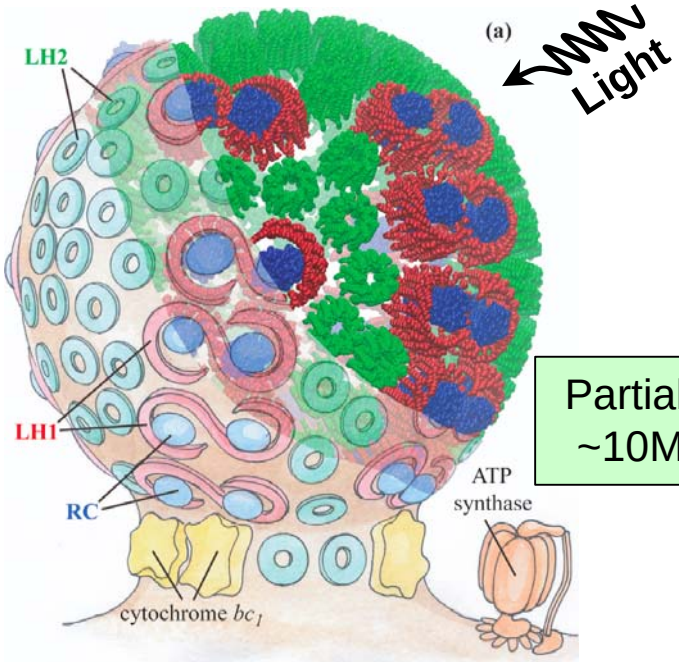
Performance profile for 0.5 Å map of potential for 1.5 M atoms.
Hardware platform is Intel QX6700 CPU and NVIDIA GTX 280.

| 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 |

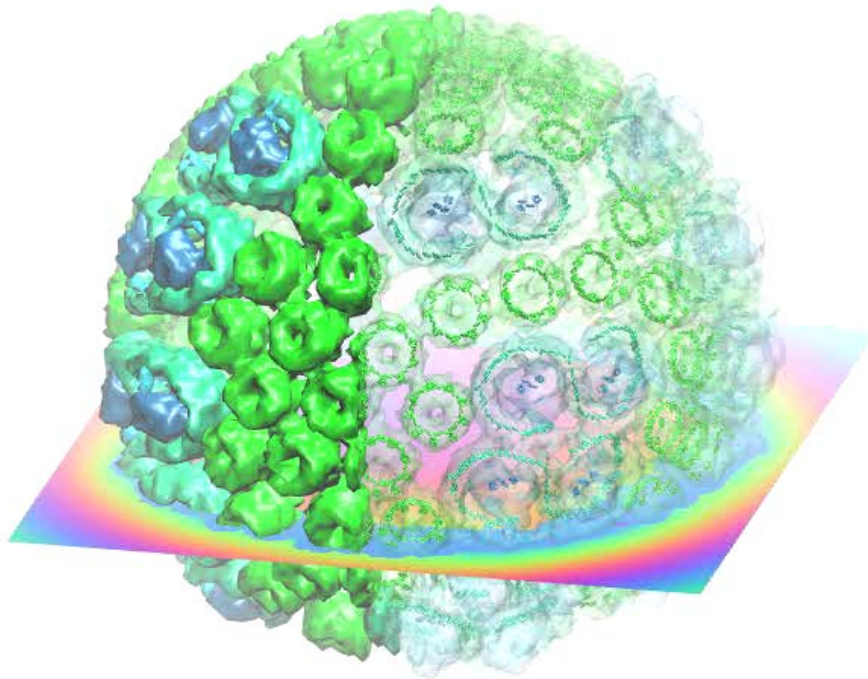


# Photobiology of Vision and Photosynthesis

## Investigations of the chromatophore, a photosynthetic organelle




Partial model:
~10M atoms



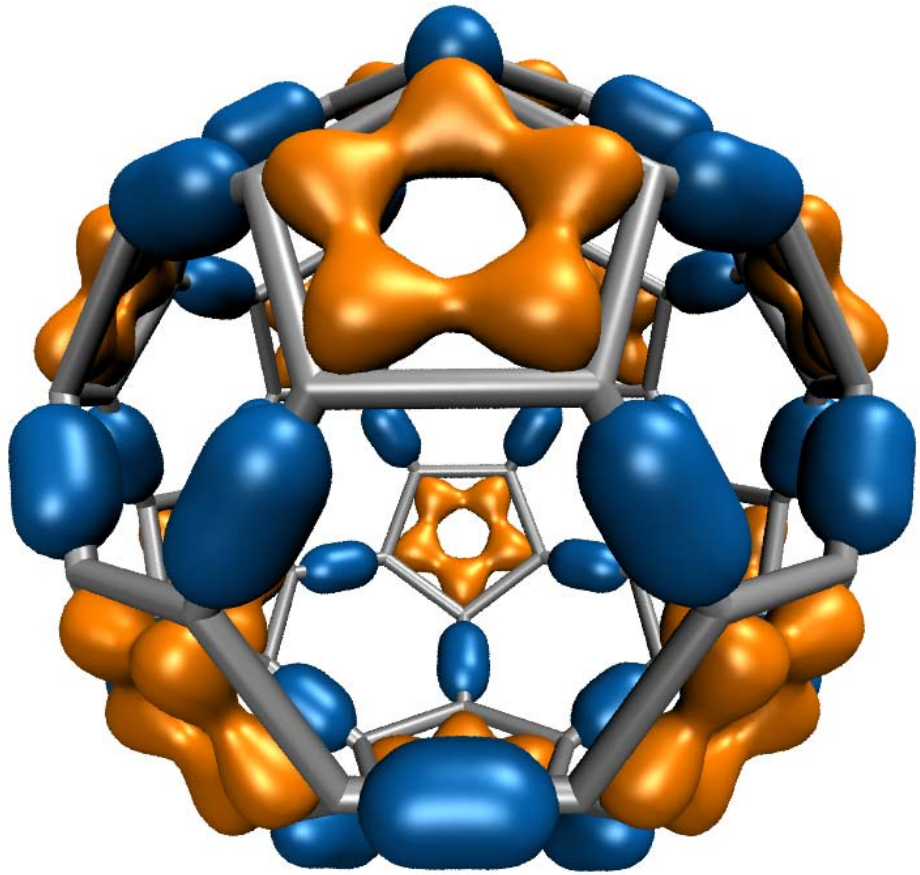
Electrostatics needed to build full structural model, place ions, study macroscopic properties

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**

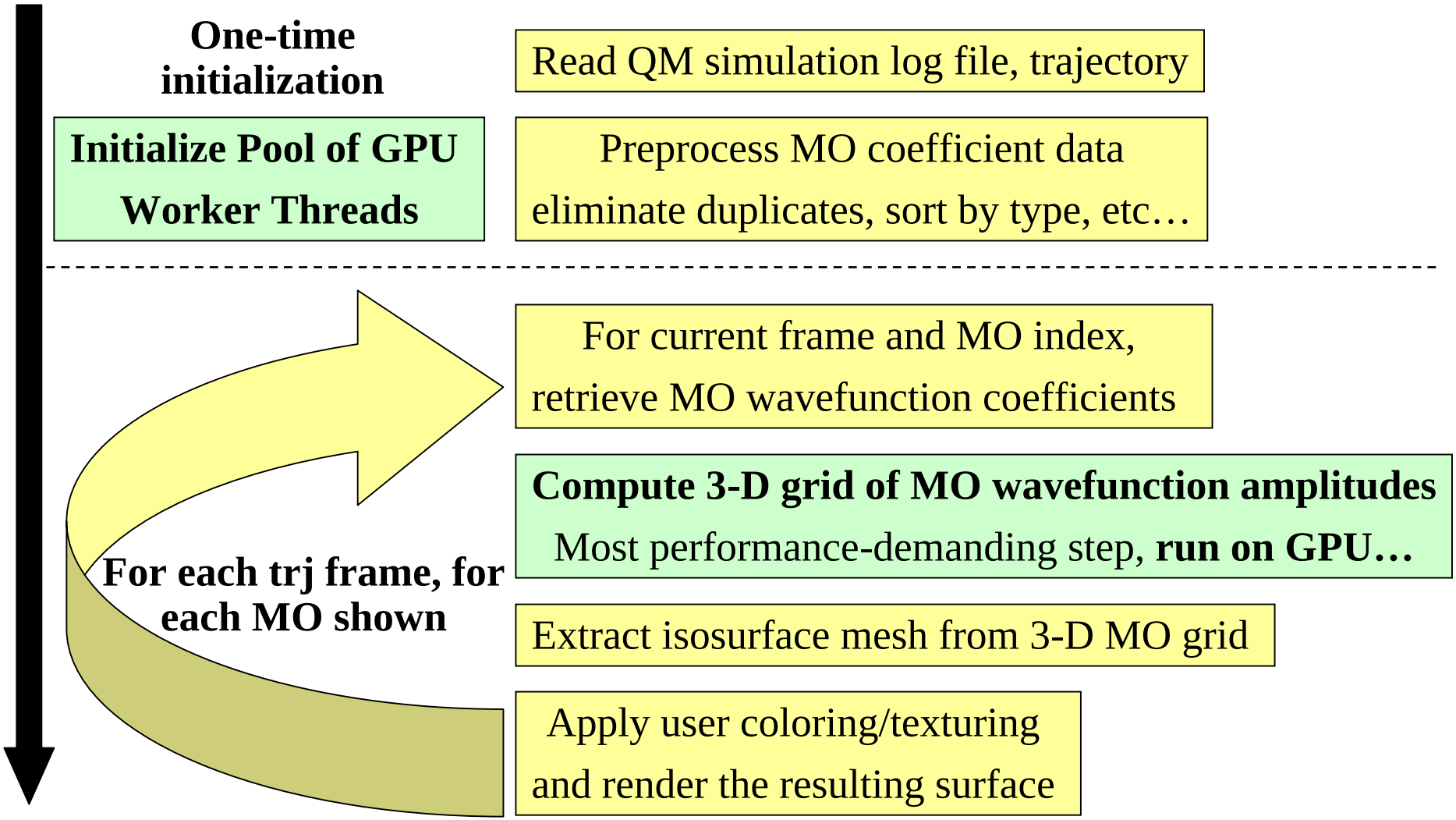
# Computing Molecular Orbitals

- Visualization of MOs aids in understanding the chemistry of molecular system
- MO spatial distribution is correlated with electron probability density
- Calculation of high resolution MO grids can require tens to hundreds of seconds on CPUs
- >100x speedup allows interactive animation of MOs @ 10 FPS



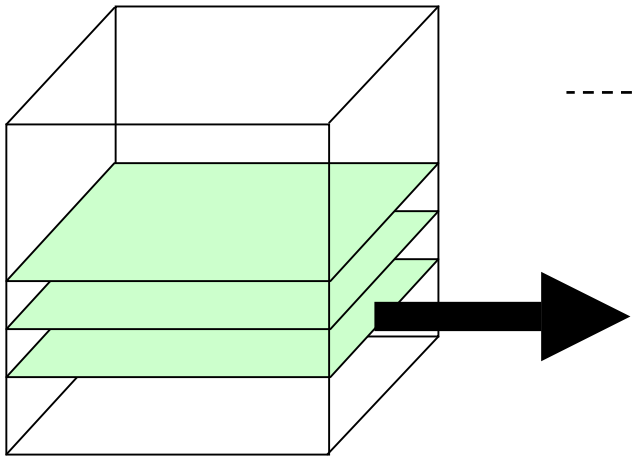
$C_{60}$

# Molecular Orbital Computation and Display Process

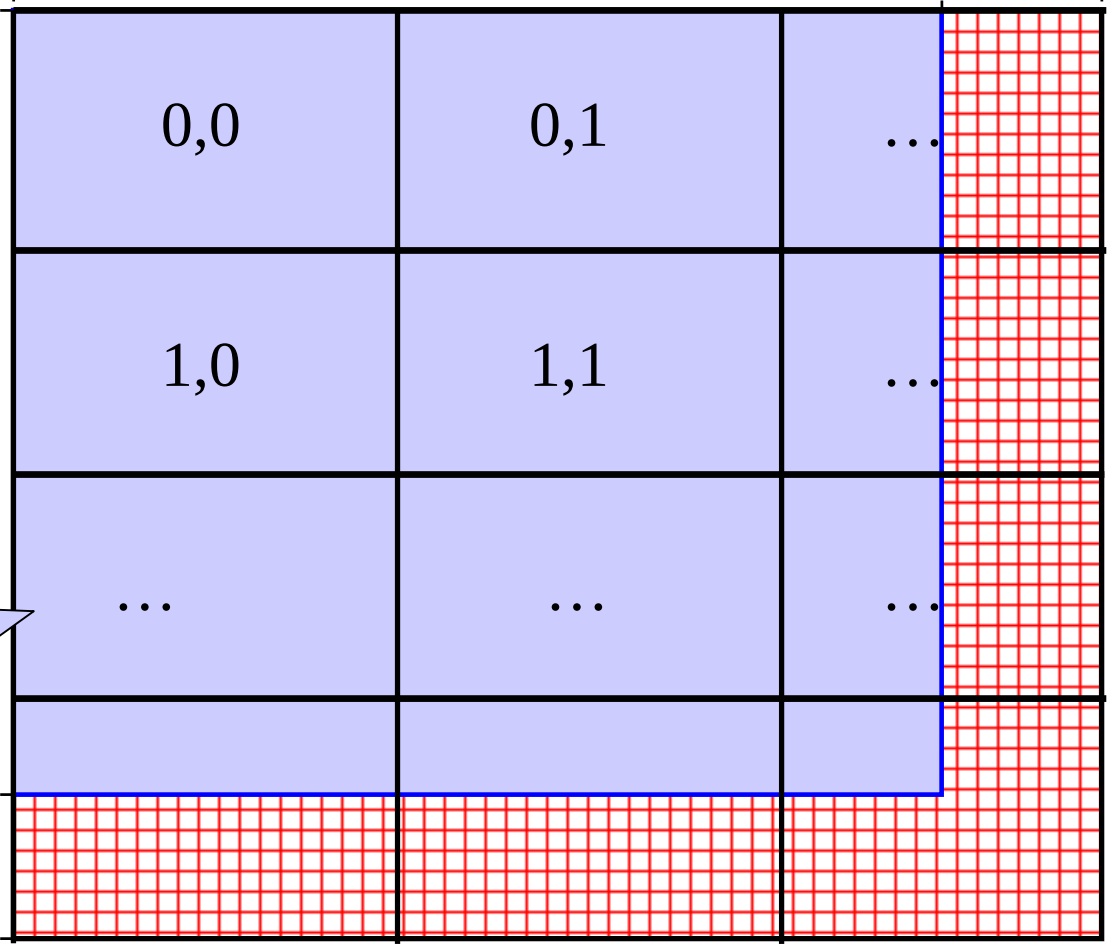


# CUDA Block/Grid Decomposition

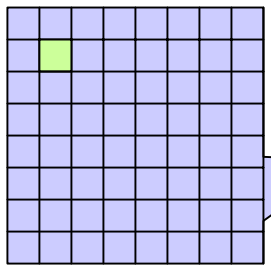
MO 3-D lattice decomposes into 2-D slices (CUDA grids)



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
```

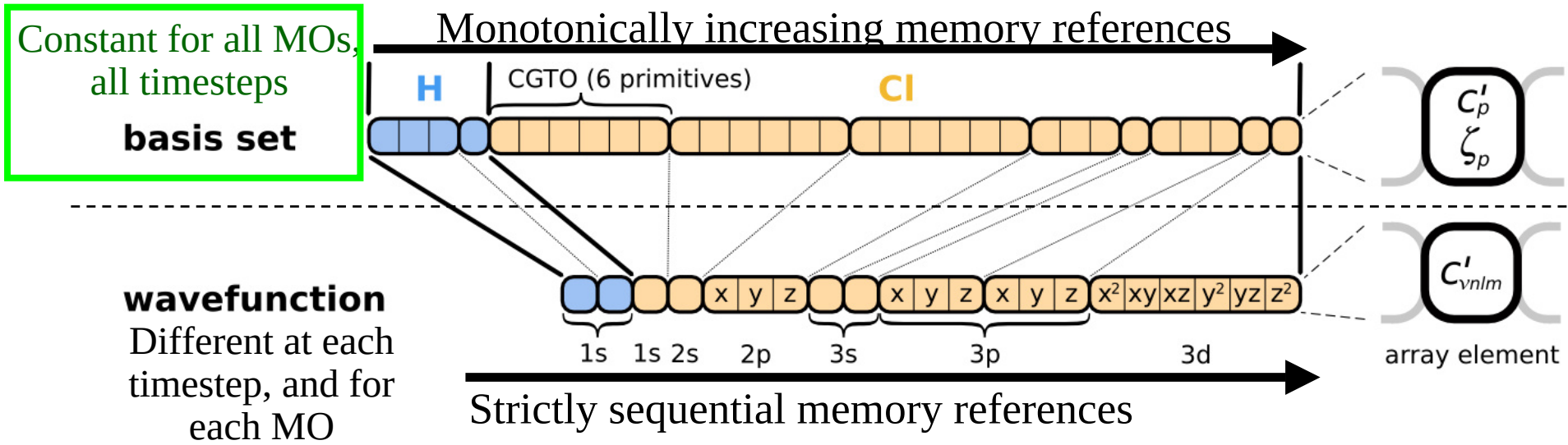
Loop unrolling:

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

# Preprocessing of Atoms, Basis Set, and Wavefunction Coefficients

- Make more effective use of high bandwidth, low-latency GPU on-chip memory:
  - 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

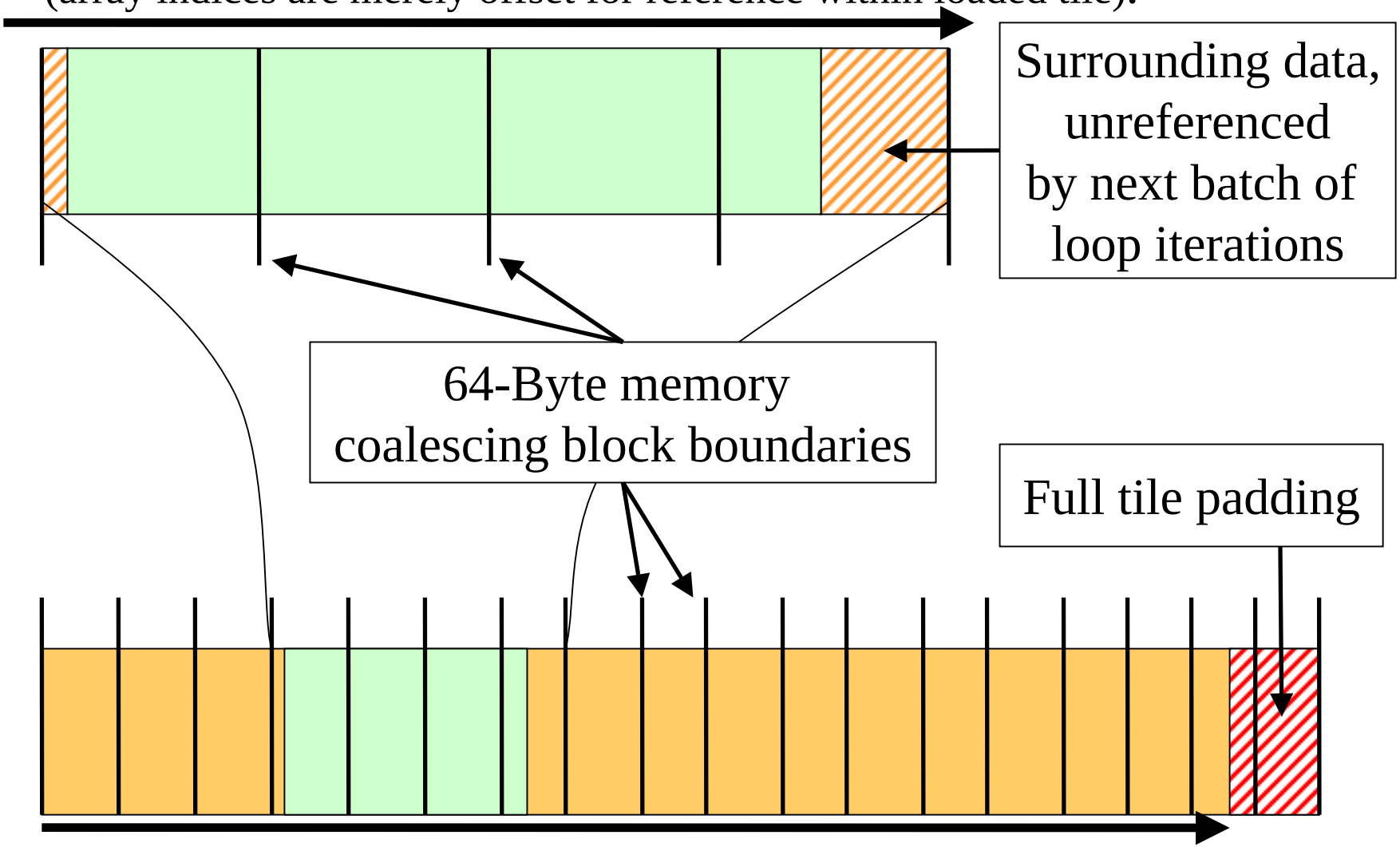



- Loop iterations always access same or consecutive array elements for all threads in a thread block:
  - Yields good constant memory cache performance
  - Increases shared memory tile reuse

# 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)** |

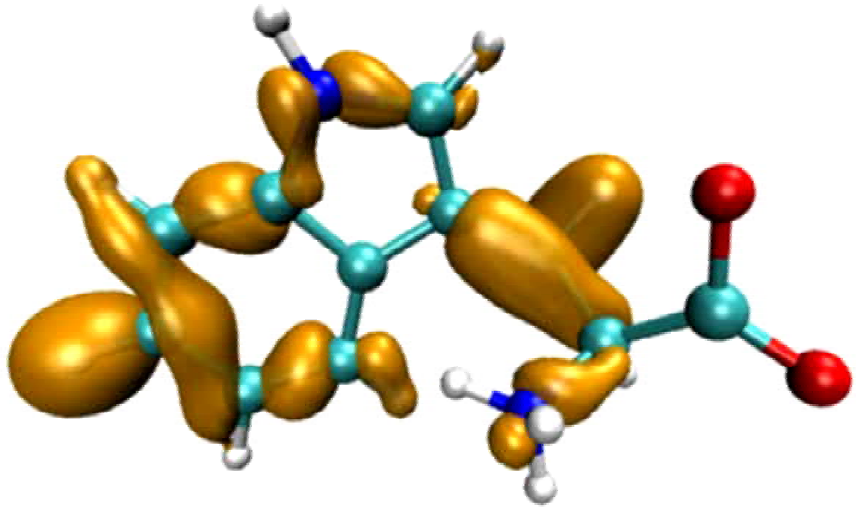
| Kernel | **Cores GPUs** | Speedup vs. Molekel on 1 CPU core | | | | | |
|---|---|---|---|---|---|---|---|
| Molekel | 1* | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| MacMolPlt | 4 | 2.4 | 2.6 | 2.1 | 2.4 | 4.3 | 4.5 |
| VMD GCC-cephes | 4 | 3.2 | 4.0 | 3.0 | 3.5 | 4.3 | 6.5 |
| VMD ICC-SSE-cephes | 4 | 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** |

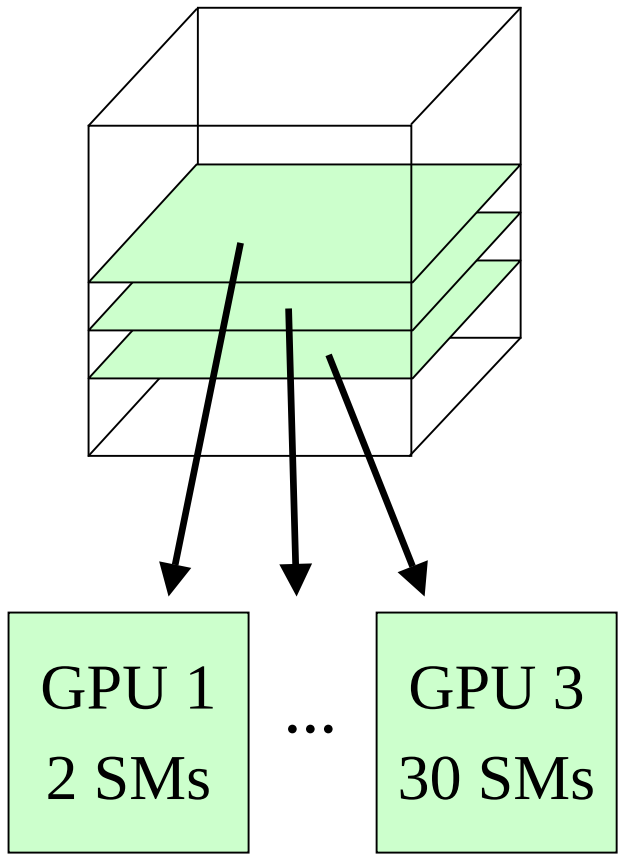


tryptophane

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…

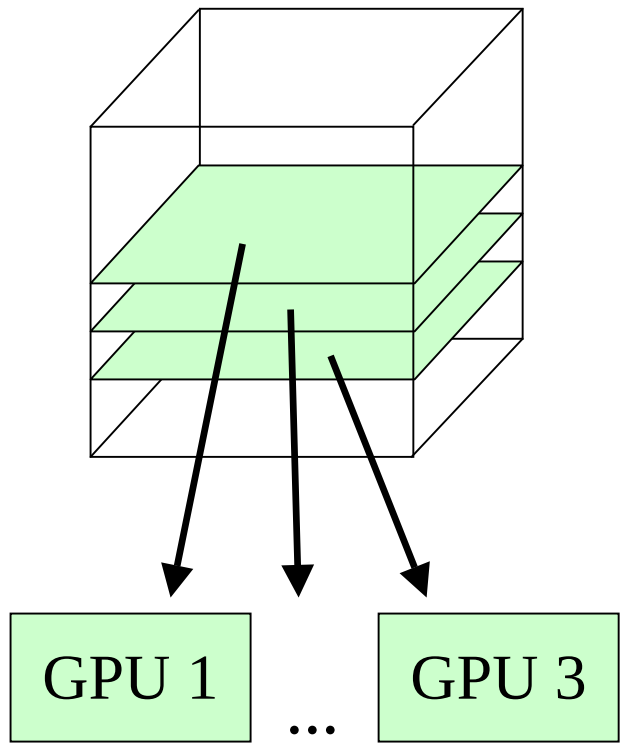
# Multi-GPU Load Balance

- Many early CUDA codes assumed all GPUs were identical
- All new NVIDIA GPUs 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



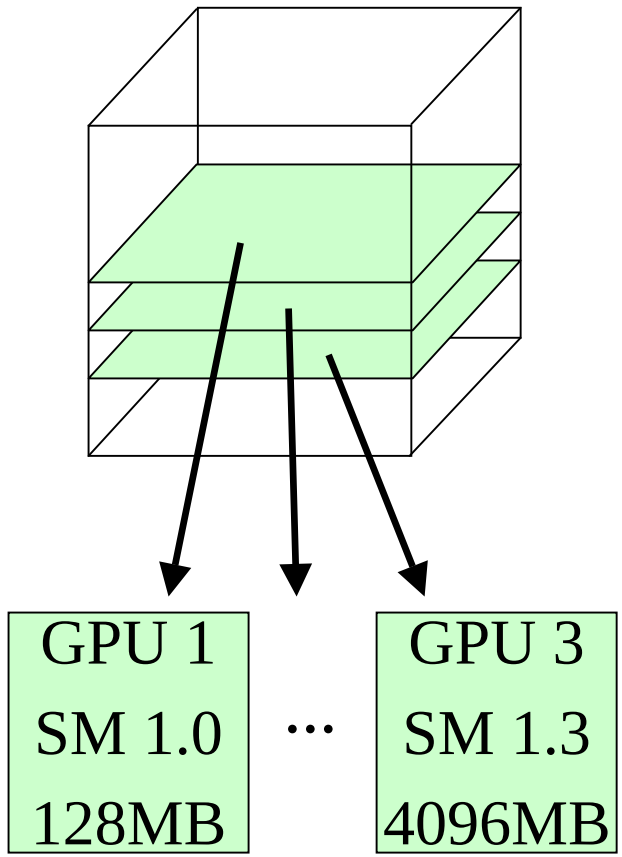
# Multi-GPU Dynamic Work Distribution

```
// 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 / queue 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

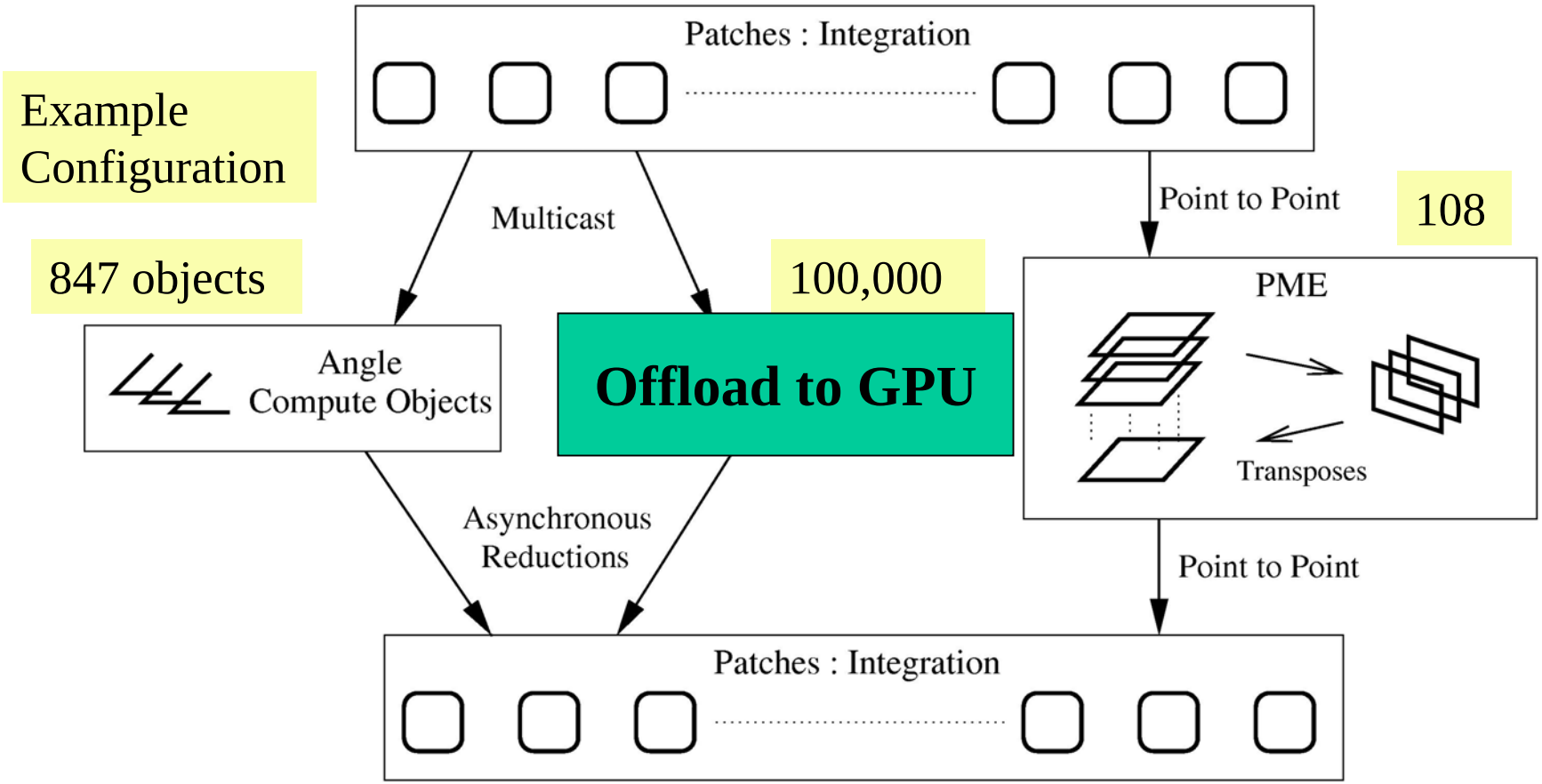
| Kernel | Cores/GPUs | Runtime (s) | Speedup |
|---|---|---|---|
| CPU-ICC-SSE | 1 | 46.580 | 1.00 |
| CPU-ICC-SSE | 4 | 11.740 | 3.97 |
| CUDA-const-cache | 3 | 0.151 | 308. |
| **CUDA-const-cache w/ mapped host memory** | **3** | **0.137** | **340.** |

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()

# NAMD Parallel Molecular Dynamics: Overlapping CPU/GPU Execution

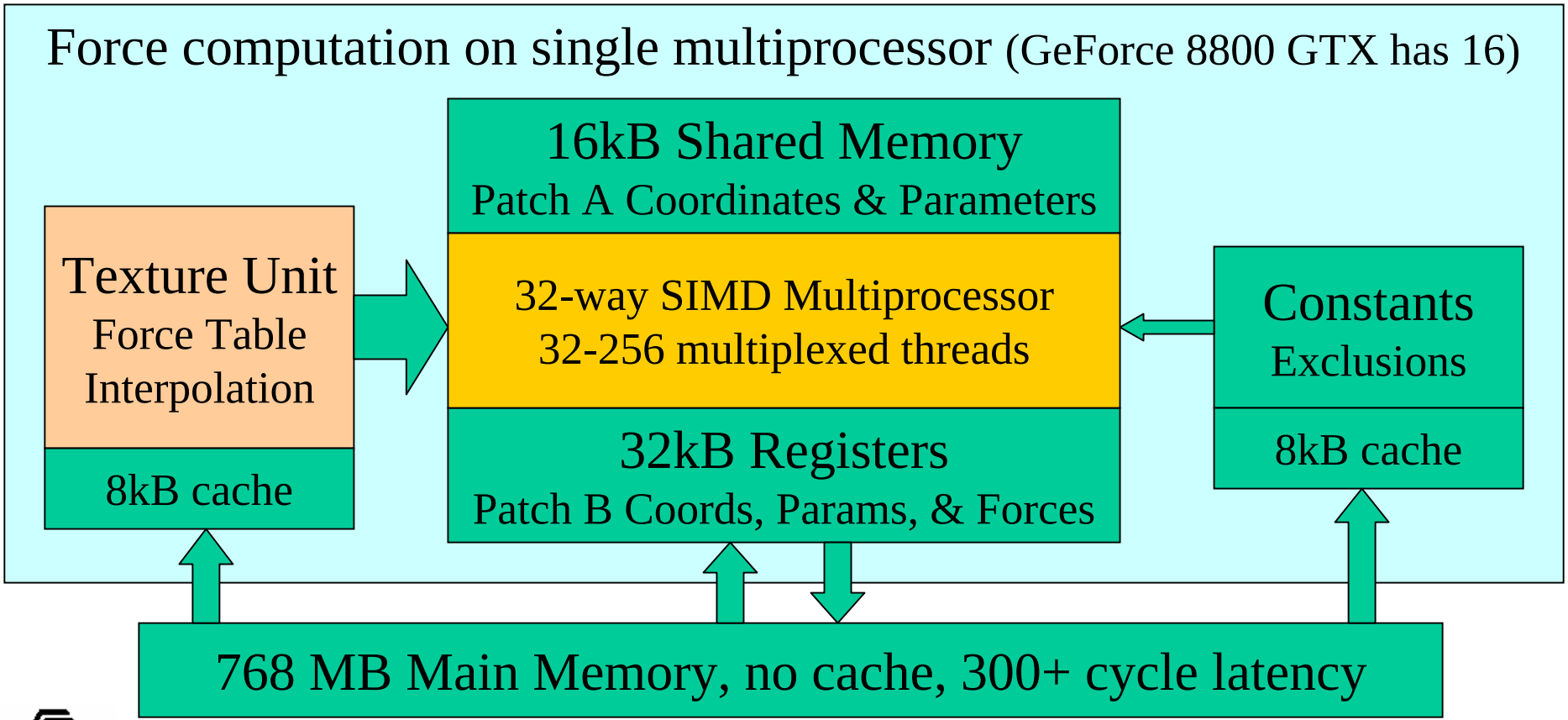



Objects are assigned to processors and queued as data arrives.

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

# Nonbonded Forces on CUDA GPU

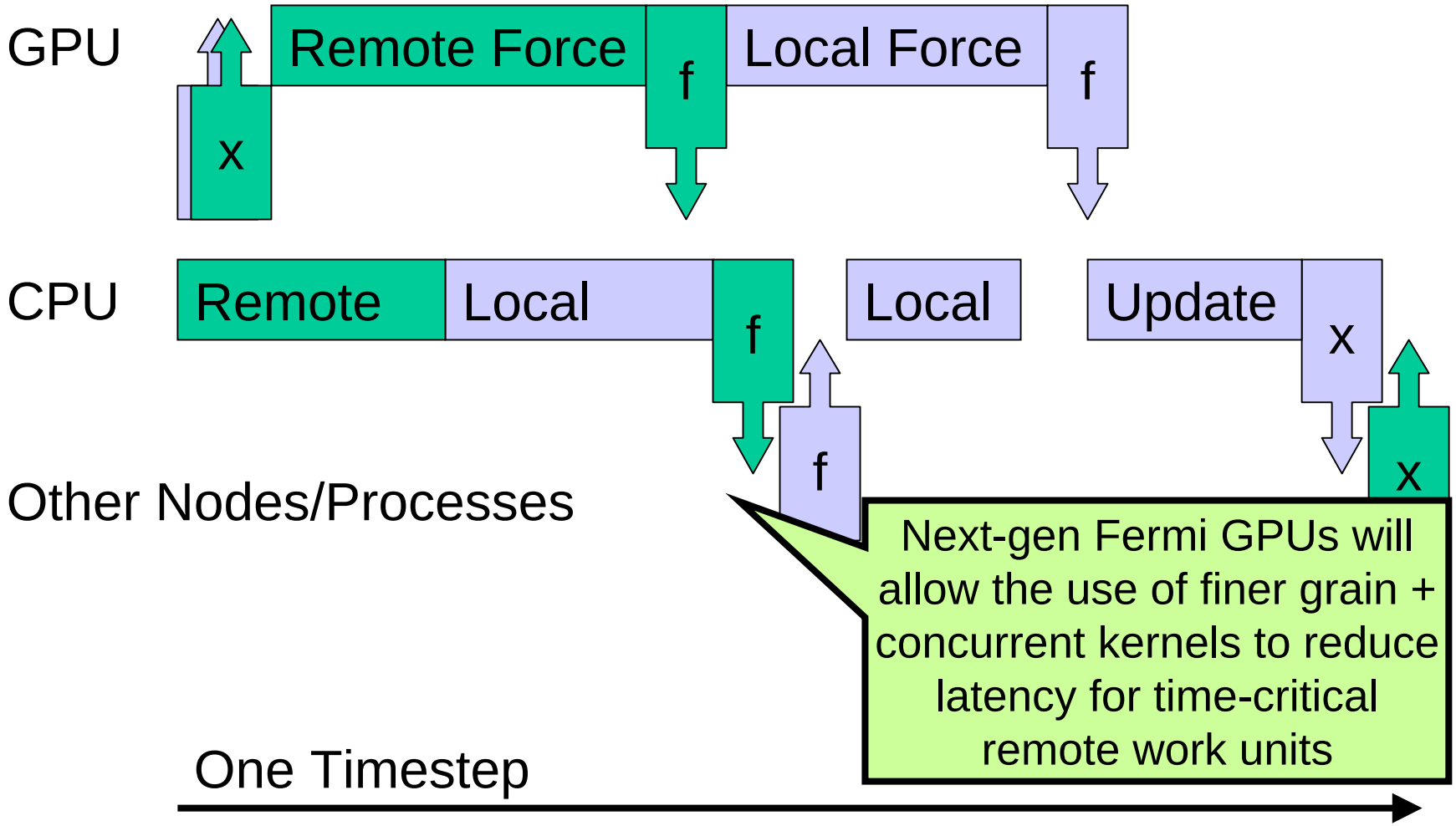
- Start with most expensive calculation: direct nonbonded interactions.
- Decompose work into pairs of patches, identical to NAMD structure.
- GPU hardware assigns patch-pairs to multiprocessors dynamically.






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

# NAMD: Overlapping GPU and CPU with Communication



# 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.