# S03: High Performance Computing with CUDA

# Heterogeneous GPU Computing for Molecular Modeling


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 S03, Supercomputing 2010,

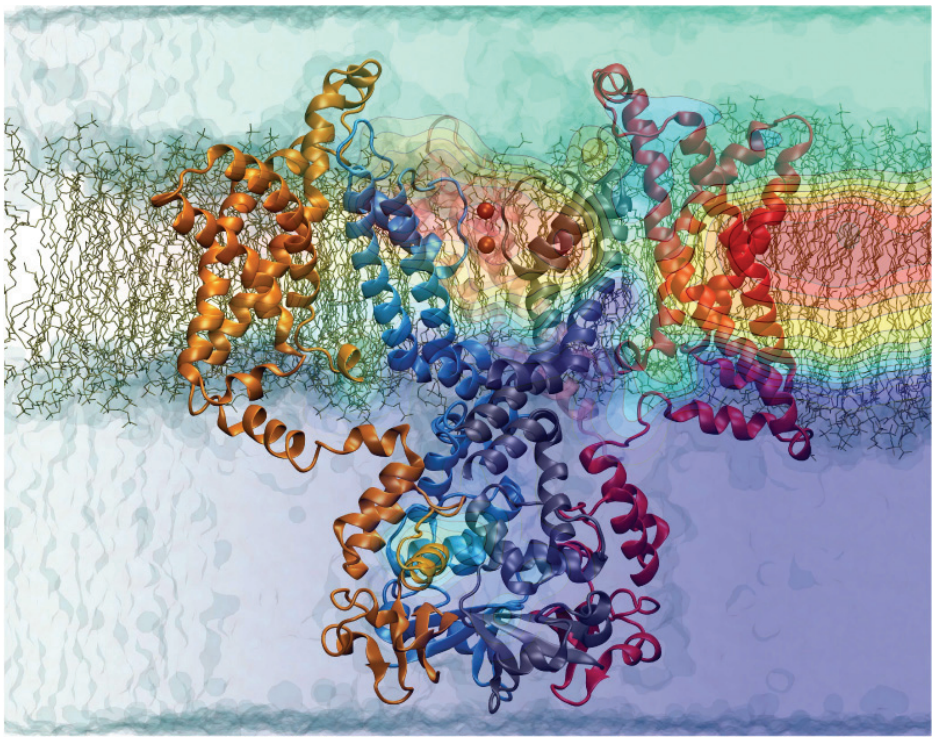
New Orleans, LA, Nov 14, 2010

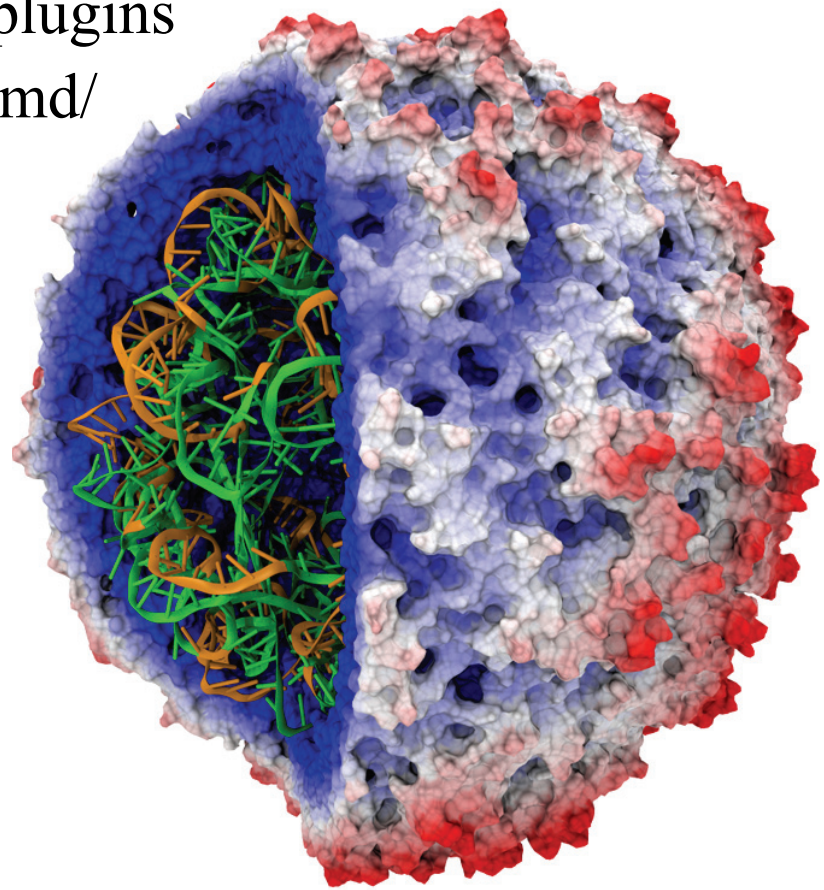
# Case Study Topics:

- Examples of CUDA kernels in the VMD molecular visualization and analysis software
- Recurring algorithm design principles
- Conversion of “scatter” to “gather”
- Overlapping CPU and GPU computations,
- Overlapping host-GPU I/O, and asynchronous stream APIs
- Zero-copy memory access strategies
- Multi-GPU work scheduling and error handling
- Low-latency processing for interactive computing

# VMD – "Visual Molecular Dynamics"

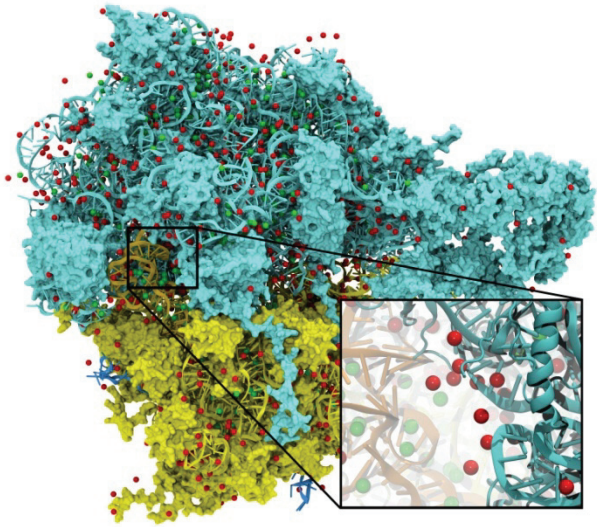
- Visualization and analysis of molecular dynamics simulations, sequence data, volumetric data, quantum chemistry simulations, particle systems, …
- User extensible with scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/





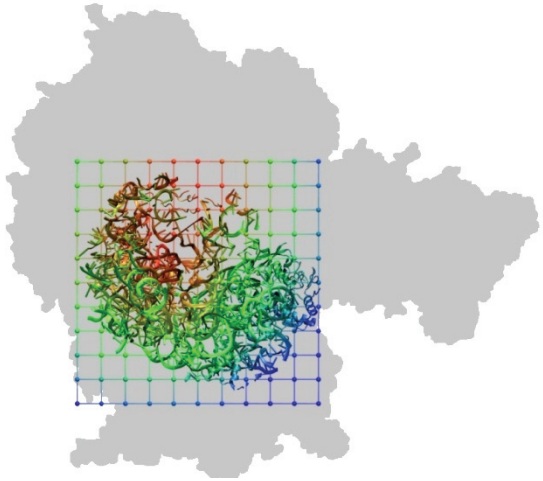
# CUDA Algorithms in VMD

Speedups vs. single CPU core (peak CPU memory bandwidth, cache perf, etc.)



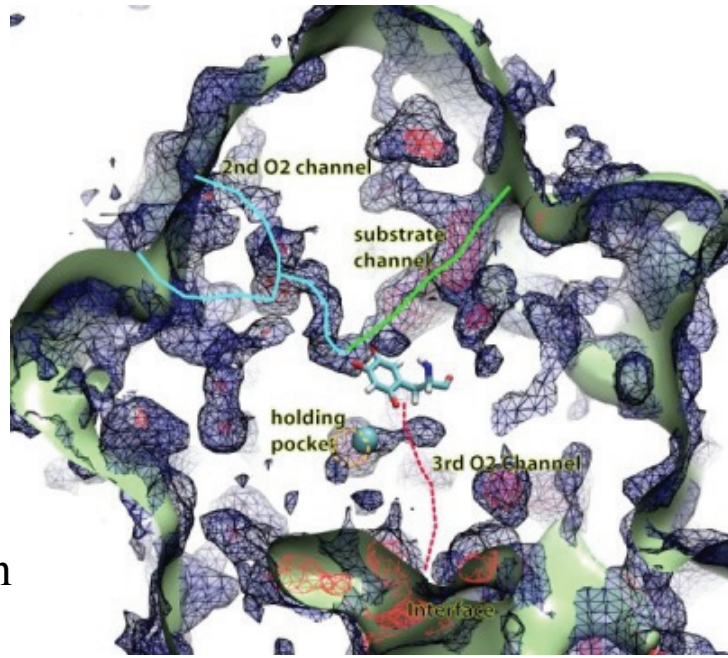
Ion placement

20x to 44x faster



Electrostatic field calculation

31x to 44x faster



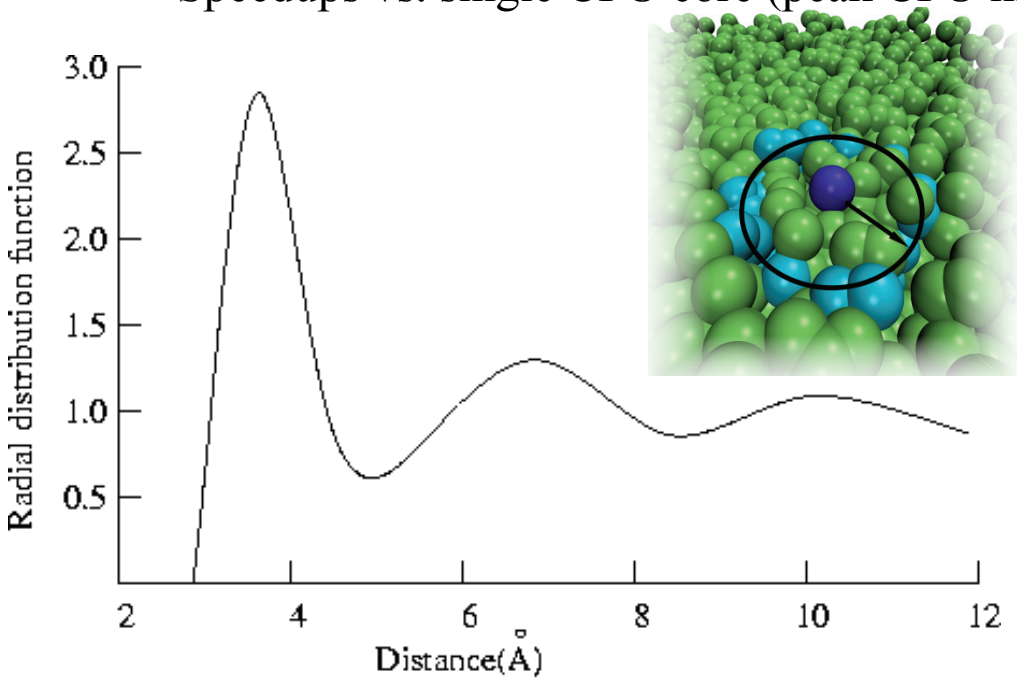

Imaging of gas migration pathways in proteins with implicit ligand sampling

20x to 30x faster

GPU: massively parallel co-processor

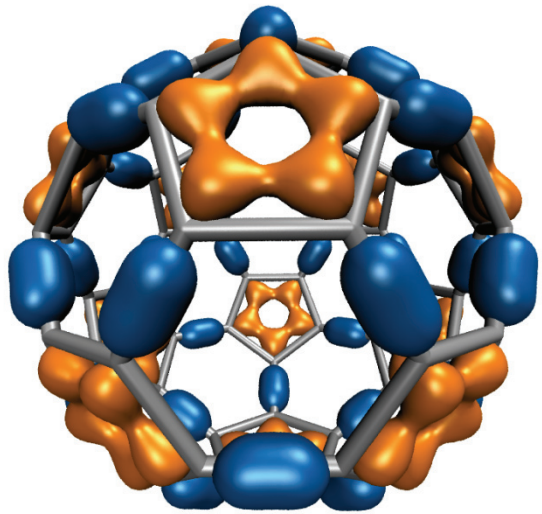
# CUDA Algorithms in VMD

Speedups vs. single CPU core (peak CPU memory bandwidth, cache perf, etc.)




Radial distribution functions

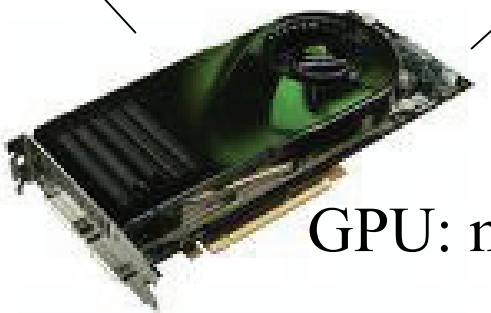
30x to 92x faster



Molecular orbital

calculation and display

100x to 120x faster



GPU: massively parallel co-processor

# Recurring Algorithm Design Principles (1)

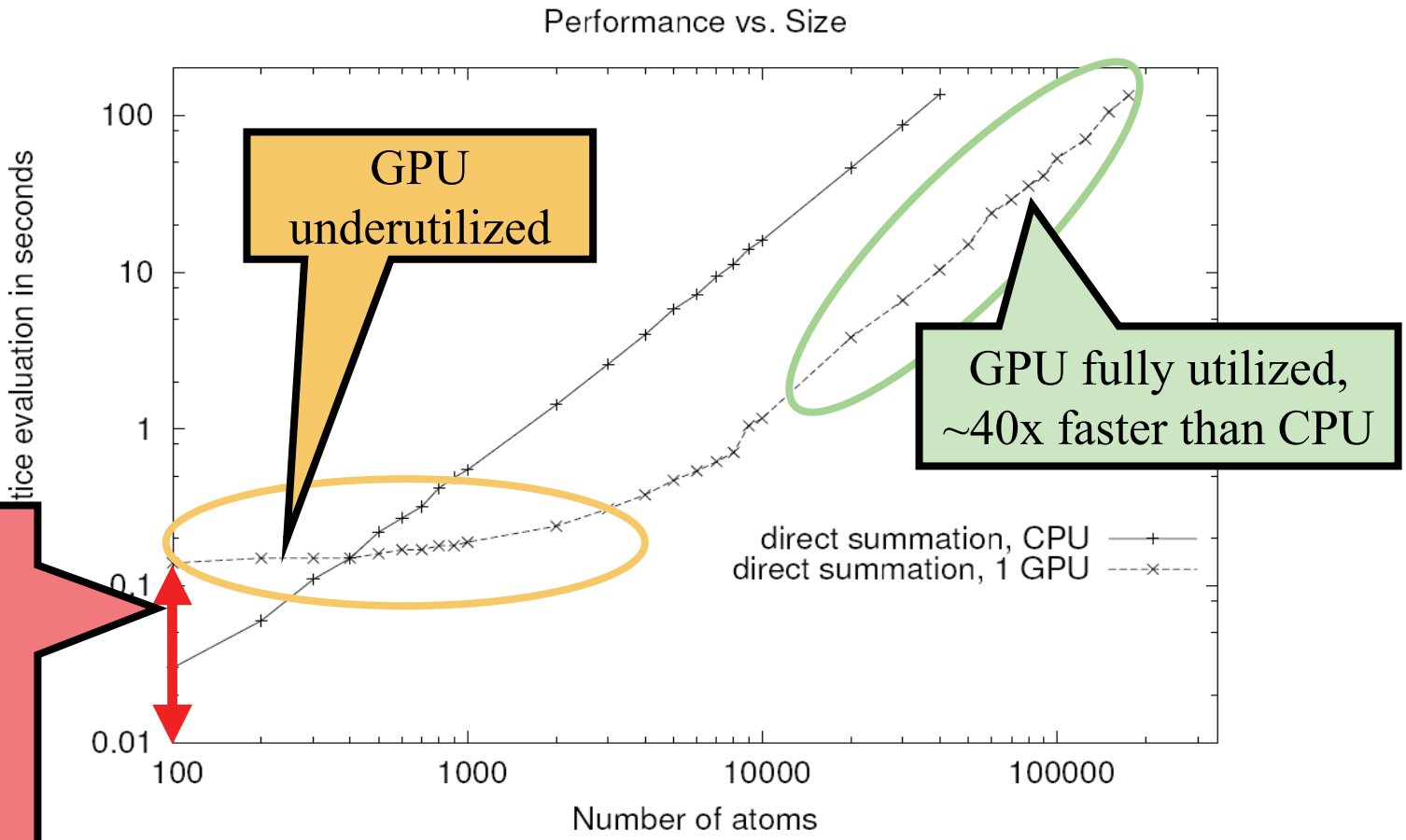
- Use registers, on-chip shared memory, L1 cache, and constant memory to amplify memory bandwidth
- Pre-process, sort, and compress operands; organize computation for peak efficiency on the GPU, particularly for best use of L1 cache and shared mem
- Use tiled/blocked data structures in GPU global memory for peak bandwidth utilization
- Use CPU to "regularize" the work done by the GPU
- Use CPU to handle exceptions & unusual work units concurrent with GPU execution
- Asynchronous operation of CPU/GPU enabling overlapping of computation and I/O on both ends

# Recurring Algorithm Design Principles (2)

- Take advantage of special features of the GPU memory systems
  - Broadcasts, wide loads/stores (float4, double2), texture interpolation, write combining, etc.
- Avoid doing complex array indexing arithmetic within the GPU threads, pre-compute as much as possible outside of the GPU kernel so the GPU is doing what it's best at: **floating point arithmetic**

# GPUs Require ~20,000 Independent Threads for Full Utilization

Lower is better




Host thread GPU context initialization and device binding time: ~110ms

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.

# Avoid Output Conflicts, Conversion of Scatter to Gather

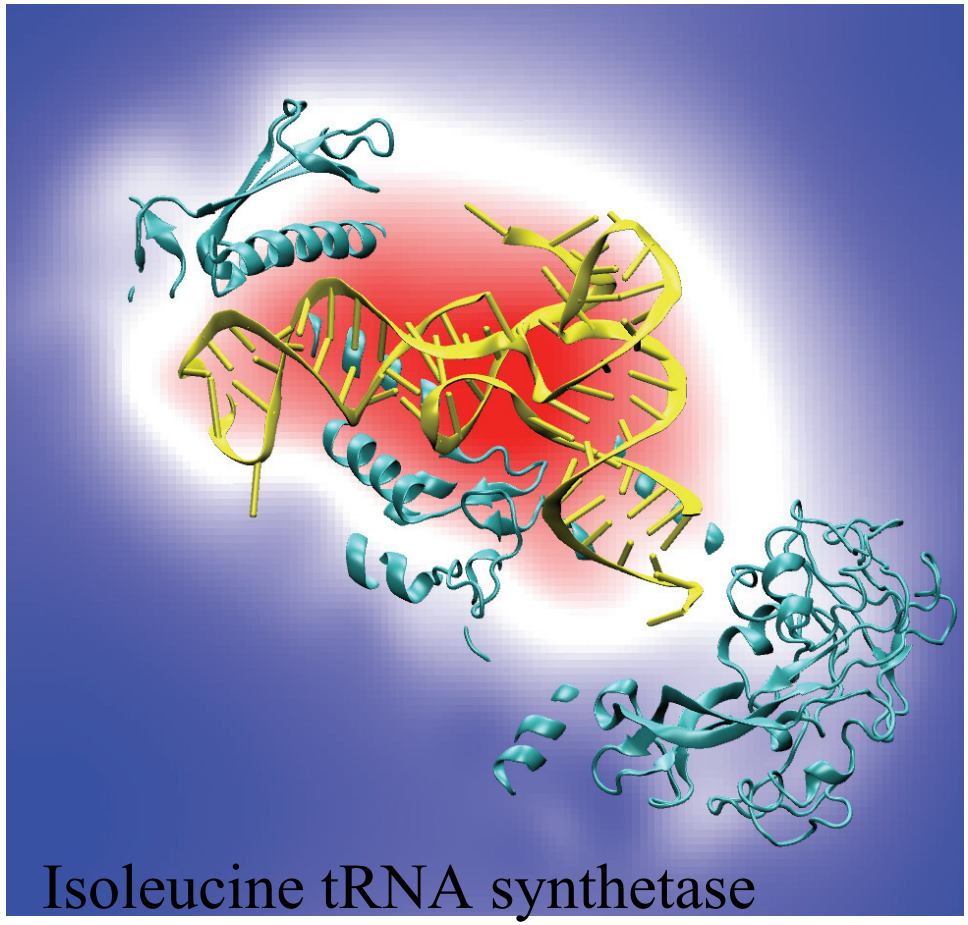
- GPUs provide tremendous memory bandwidth, but even so, **memory bandwidth often still ends up being the performance limiter**
- Many CPU codes contain algorithms that "scatter" operands to memory, to reduce arithmetic
- Scattered output can create bottlenecks for GPU performance
- On the GPU, it's often better to do **more** arithmetic, in exchange for a **regular output pattern**, or to convert "scatter" algorithms to "gather" approaches

# 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



# CUDA Cutoff Electrostatic Potential Summation

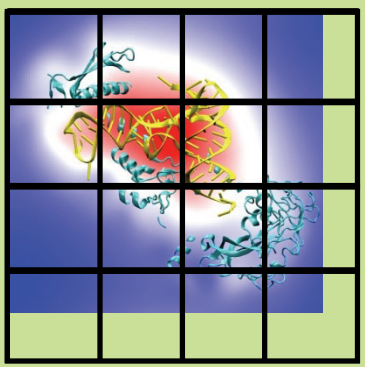
- Atoms are spatially hashed into **fixed-size bins** (guarantees coalescing)
- CPU handles overflowed bins (GPU kernel can be very aggressive)
- GPU thread block calculates corresponding region of potential map,
- GPU bin/region neighbor checks are costly; solved with universal table look-up

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

Shared memory

atom bin

Global memory



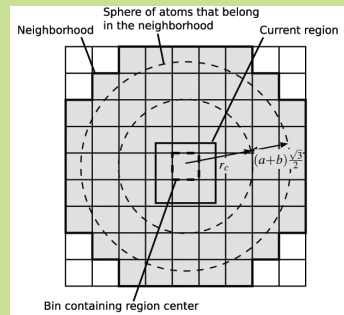
Potential map regions

Bins of atoms (fixed size)

Constant memory

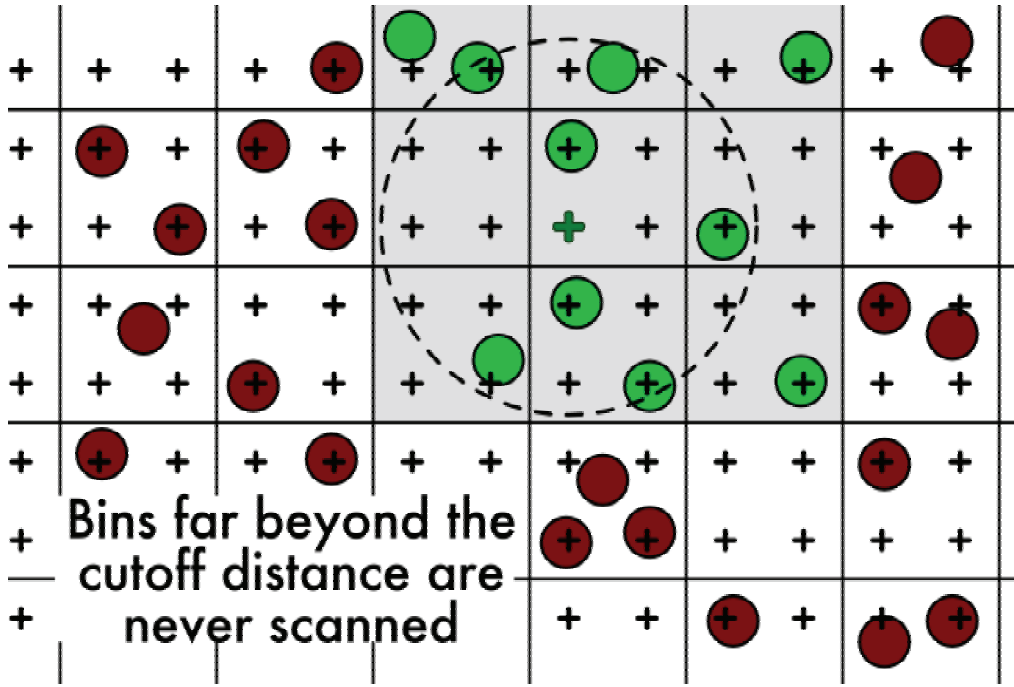
Offsets for bin neighborhood

**Look-up table encodes "logic" of spatial geometry**



# Spatial Sorting of Atoms Into "Bins"

- Sort atoms into *bins* by their coordinates
- Each bin is sized to guarantee GPU memory coalescing
- Each bin holds up to 8 atoms, containing 4 FP values (3 coords, 1 charge)
- Each lattice point **gathers** potentials from atom bins within cutoff

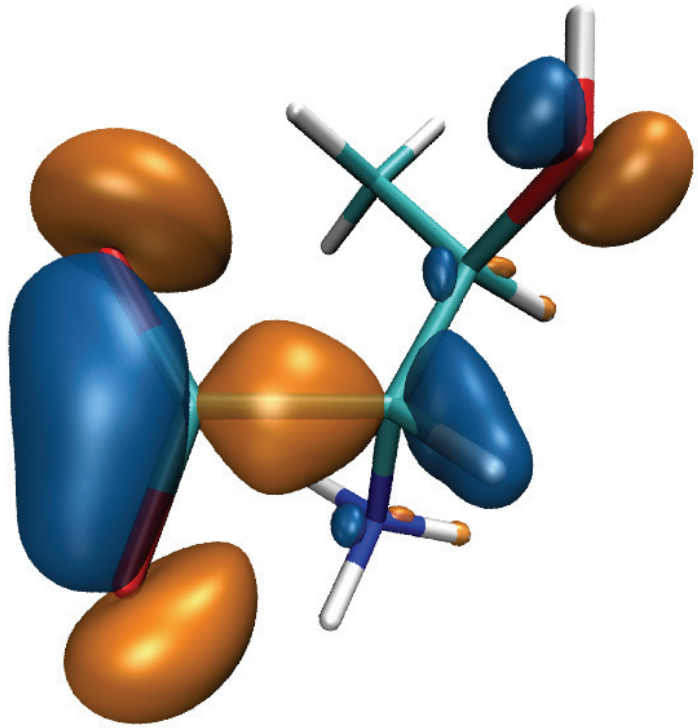


# Using the CPU to Optimize GPU Performance

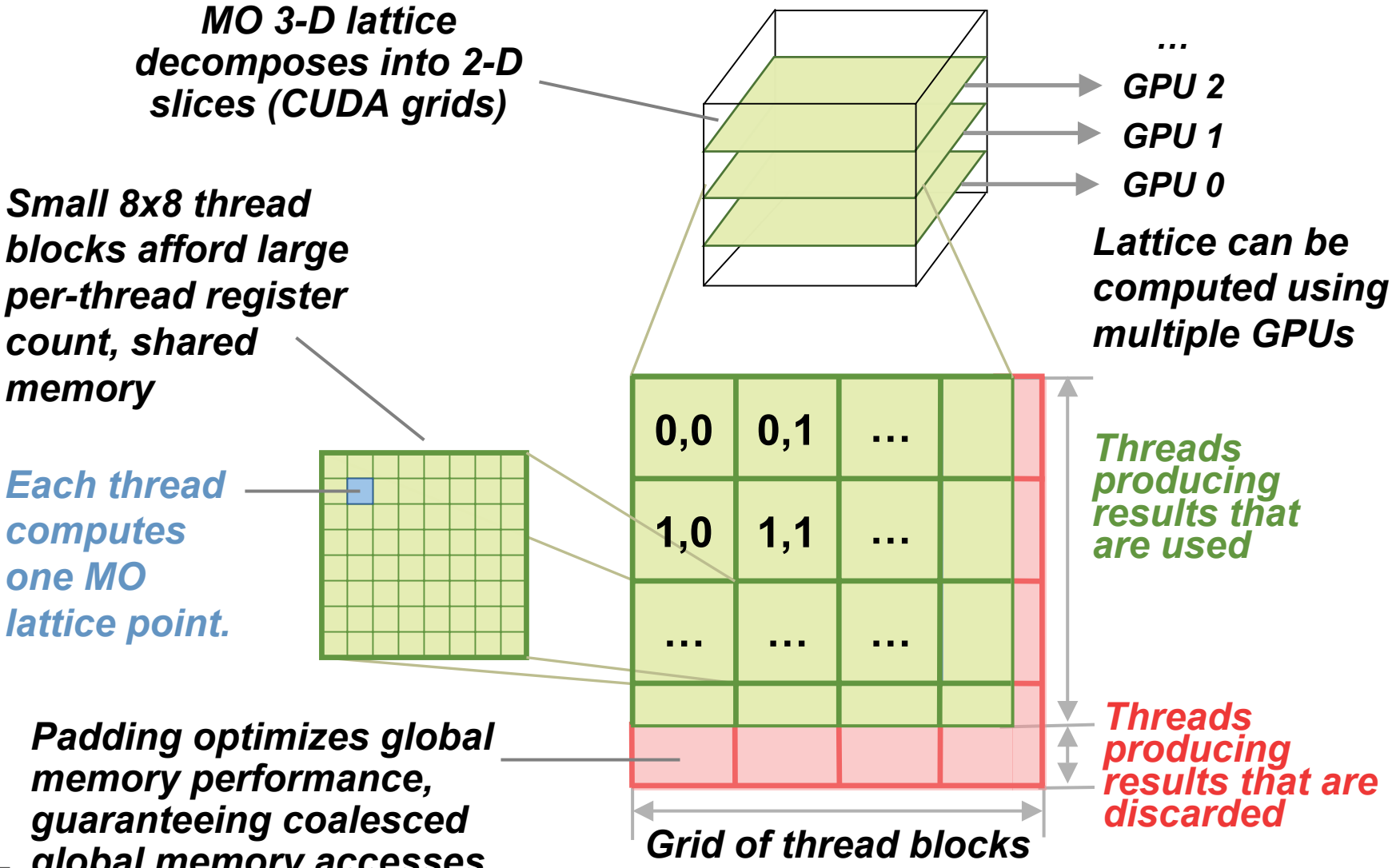
- GPU performs best when the work evenly divides into the number of threads/processing units
- Optimization strategy:
  - Use the CPU to ***"regularize"*** the GPU workload
  - Use fixed size bin data structures, with "empty" slots skipped or producing zeroed out results
  - Handle exceptional or irregular work units on the CPU; GPU processes the bulk of the work concurrently
  - On average, the GPU is kept highly occupied, attaining a high fraction of peak performance

# Computing Molecular Orbitals

- Visualization of MOs aids in understanding the chemistry of molecular system
- Calculation of high resolution MO grids for display can require tens to hundreds of seconds on multi-core CPUs, even with the use of hand-coded SSE



# MO GPU Parallel Decomposition



# 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 kernel based on L1 cache

```
[… outer loop over atoms …]
  // loop over the shells belonging to this atom (or basis function)
  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

# VMD Single-GPU Molecular Orbital Performance Results for $C_{60}$

## 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: may outperform hand-coded shared memory kernels. Zero-copy memory transfers improve overlap of computation and host-GPU I/Os.

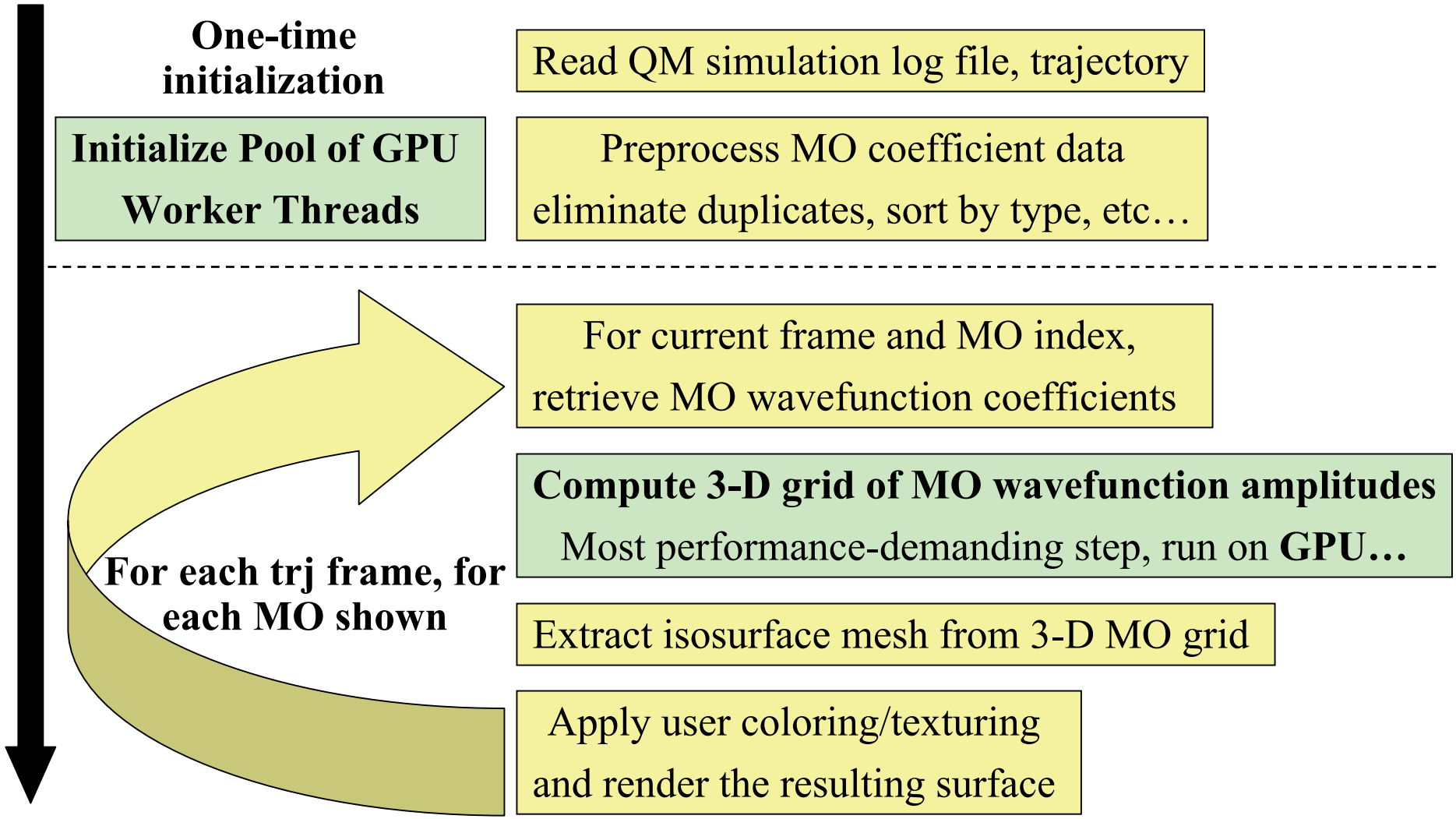
# VMD Multi-GPU Molecular Orbital Performance Results for $C_{60}$

## 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 |

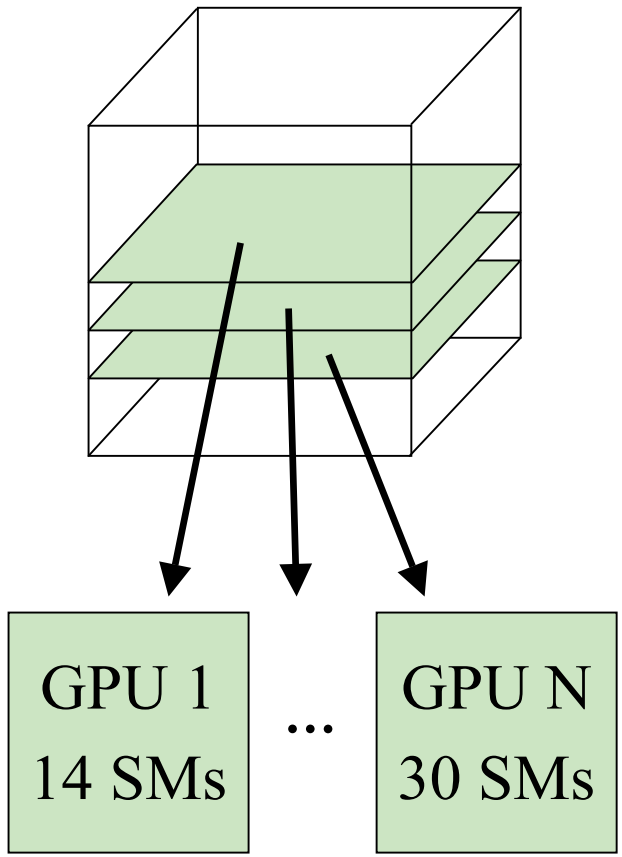
Uses persistent thread pool to avoid GPU init overhead, dynamic scheduler distributes work to GPUs

# Molecular Orbital Computation and Display Process



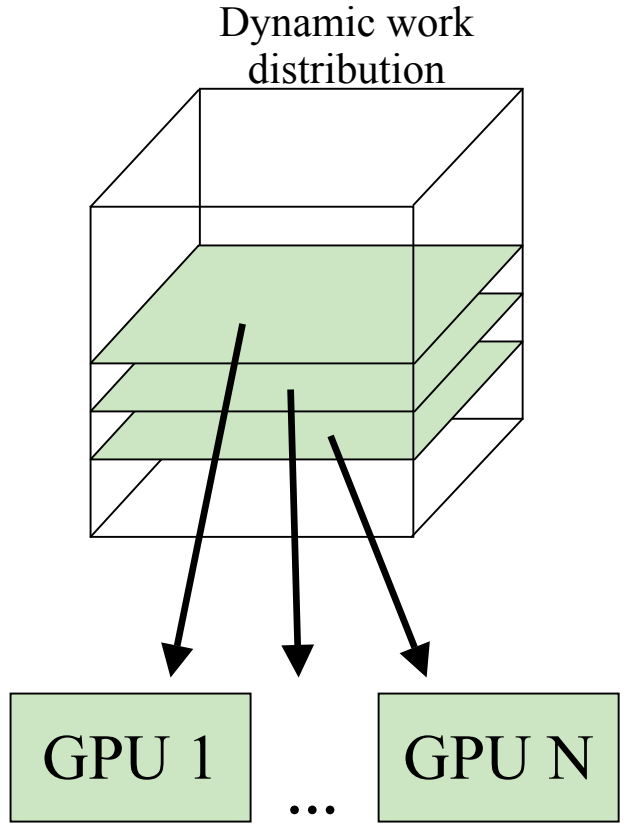
# Multi-GPU Load Balance

- Many early CUDA codes assumed all GPUs were identical
- Host machines may contain a diversity of GPUs of varying capability (discrete, IGP, etc)
- Different GPU on-chip and global memory capacities may need different problem “tile” sizes
- Static decomposition works poorly for non-uniform workload, or diverse GPUs



# 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
}
```



# Example Multi-GPU Latencies Relevant to Interactive Sci-Viz, Script-Driven Analyses

(4 Tesla C2050 GPUs, Intel Xeon 5550)

| | |
|---|---|
| 6.3us | CUDA empty kernel (immediate return) |
| 9.0us | Sleeping barrier primitive (non-spinning barrier that uses POSIX condition variables to prevent idle CPU consumption while workers wait at the barrier) |
| 14.8us | pool wake, host fctn exec, sleep cycle (no CUDA) |
| 30.6us | pool wake, 1x(tile fetch, simple CUDA kernel launch), sleep |
| 1817.0us | pool wake, 100x(tile fetch, simple CUDA kernel launch), sleep |

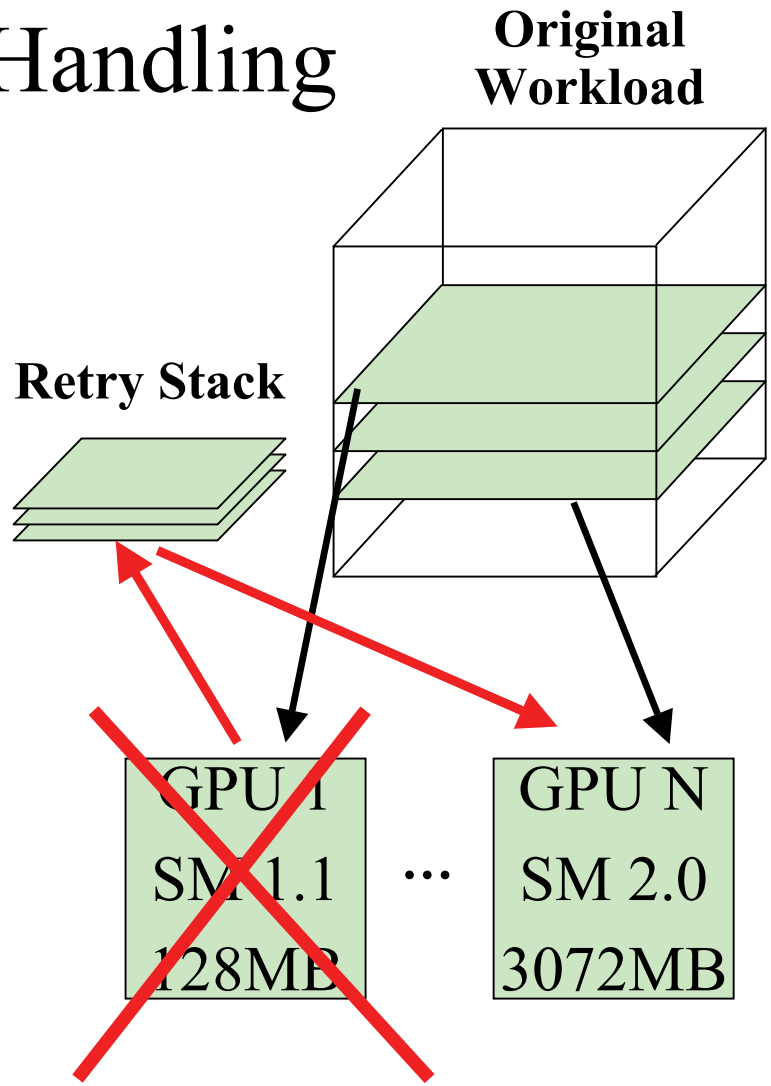
# Multi-GPU Dynamic Scheduling Performance with Heterogeneous GPUs

| 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.

# Multi-GPU Runtime Error/Exception Handling

- Competition for resources from other applications can cause runtime failures, e.g. GPU out of memory half way through an algorithm
- Handle exceptions, e.g. convergence failure, NaN result, insufficient compute capability/features
- Handle and/or reschedule failed tiles of work



# 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
  - Ben Levine, Axel Kohlmeyer, Temple University
  - CUDA team at NVIDIA
  - UIUC NVIDIA CUDA Center of Excellence
  - NIH support: P41-RR05969

# 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. *The Work in Progress in Green Computing,* 2010. In press.
- **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/

- **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),* 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.
- **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.