Using Accelerator Directives to Adapt Science Applications for State-of-the-Art HPC Architectures

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/

WACCPD 2017: Fourth Workshop on Accelerator Programming Using Directives
9:15am-10:00am, Room 710-712, Colorado Convention Center,
Denver, CO, Monday Nov 13th, 2017

Overview

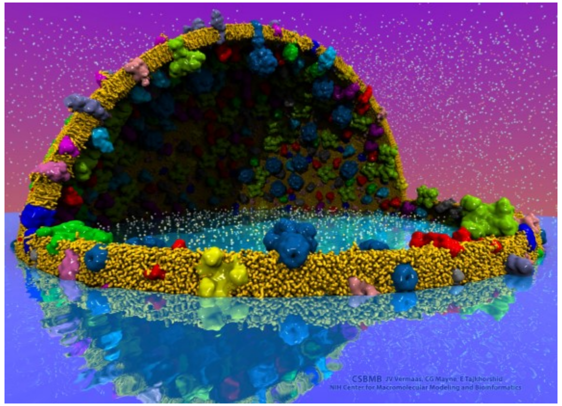
- My perspective about directive-based accelerator programming today and in the near-term ramp up to exascale computing
- Based on our ongoing work developing VMD and NAMD molecular modeling tools supported by our NIH-funded center since the mid-90's
- **What is a person like me doing using directives?** I'm the same guy that likes to give talks about CUDA and OpenCL, x86 intrinsics, and similarly lower level programming techniques. **Why am I here?**

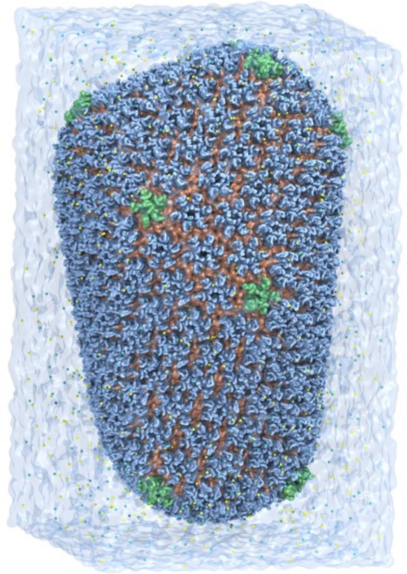
Spoilers:

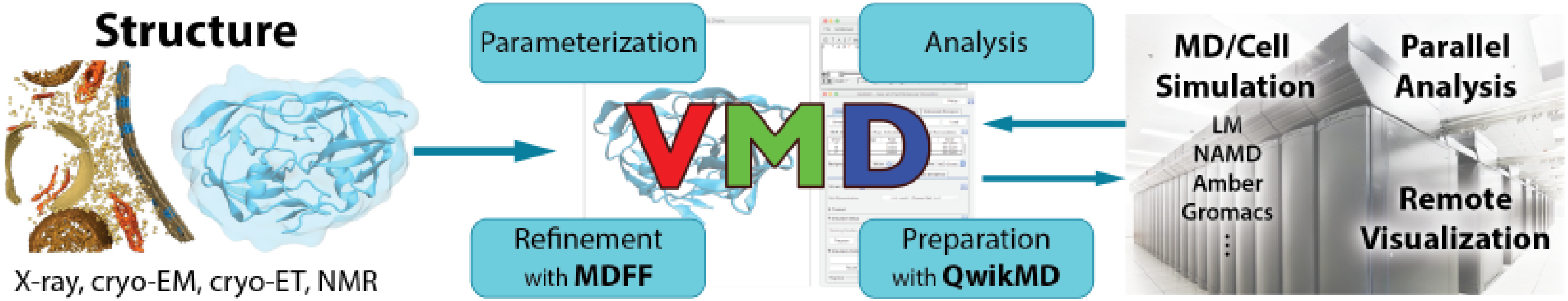
- Directives are a key solution in the **“all options on the table”** type of approach that I believe is required as we work toward **exascale** computing
- **There aren’t enough HPC developers in the world** to write everything entirely in low level APIs fast enough to keep pace
- **Science is an ever changing landscape** – significant methodological developments come every few years in active fields like biomolecular modeling…
- **Code gets (re)written for new science methodologies before you’ve finished optimizing the old code for the previous science method!?!?!?!**
- **Hardware is still changing very rapidly, and more disruptively than during the blissful heyday of “Peak Moore’s Law”**

VMD – “Visual Molecular Dynamics”

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


Cell-Scale Modeling

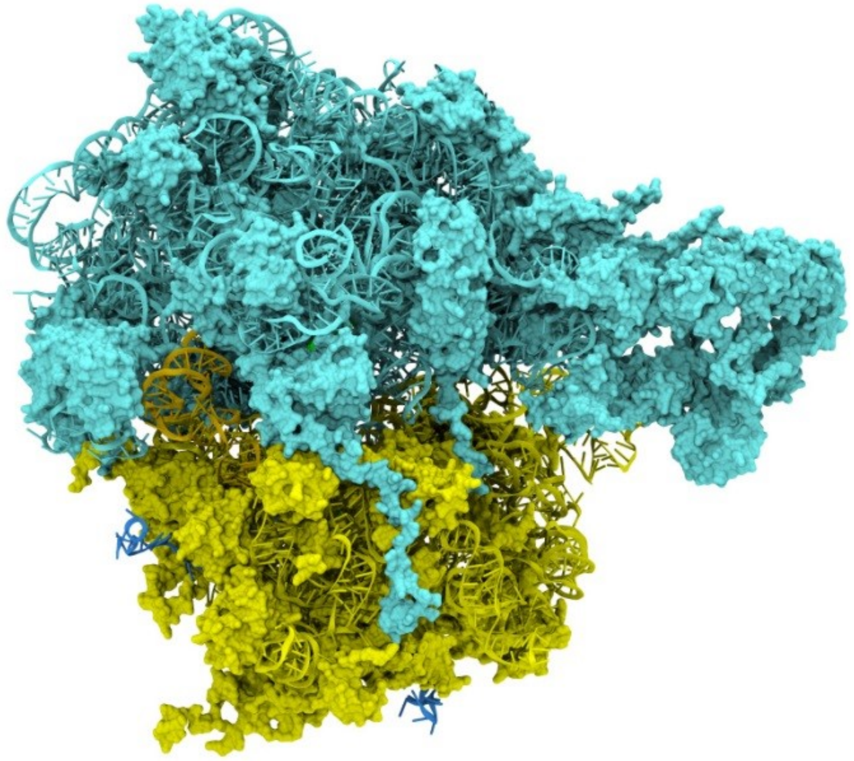

MD Simulation



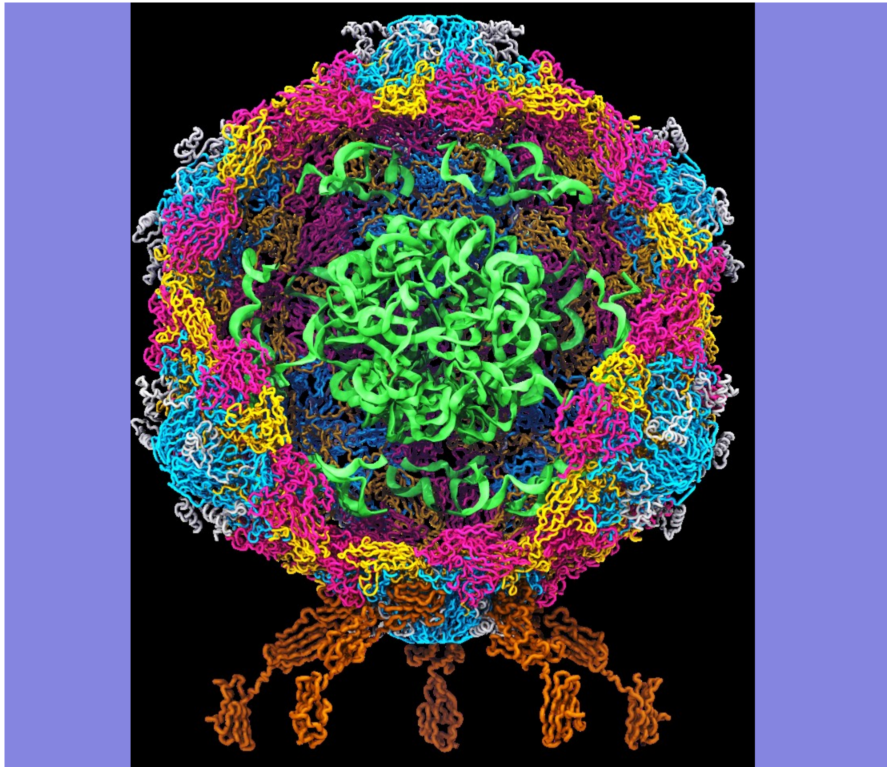
Goal: A Computational Microscope

Study the molecular machines in living cells

Ribosome: target for antibiotics



Poliovirus



Exemplary Hetereogeneous Computing Challenges

- **Tuning, adapting, or developing software for multiple processor types**
- Decomposition of problem(s) and load balancing work across heterogeneous resources for **best overall performance and work-efficiency**
- **Managing data placement** in disjoint memory systems with varying performance attributes
- **Transferring data** between processors, memory systems, interconnect, and I/O devices

Major Approaches For Programming Hybrid Architectures

- Use **drop-in libraries** in place of CPU-only libraries
  - **Little or no code development**
  - Examples: MAGMA, BLAS-variants, FFT libraries, etc.
  - **Speedups limited by Amdahl's Law** and overheads associated with data movement between CPUs and GPU accelerators
- Generate accelerator code as a variant of CPU source, e.g. using OpenMP and **OpenACC directives**, and similar
- Write **lower-level** accelerator-specific code, e.g. using **CUDA, OpenCL**, other approaches

Challenges Adapting Large Software Systems for State-of-the-Art Hardware Platforms

- Initial focus on key computational kernels eventually gives way to the need to optimize an **ocean of less critical routines**, due to observance of **Amdahl's Law**
- Even though these less critical routines might be easily ported to CUDA or similar, the **sheer number of routines often poses a challenge**
- Need a low-cost approach for **getting "some" speedup** out of these second-tier routines
- In many cases, it is completely **sufficient to achieve memory-bandwidth-bound GPU performance with an existing algorithm**

Amdahl’s Law and Role of Directives

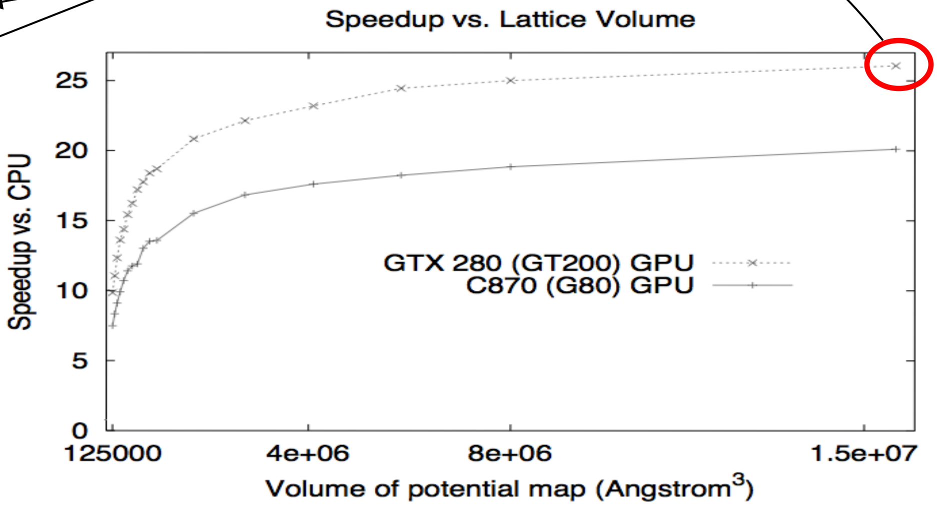
- Initial partitioning of algorithm(s) between host CPUs and accelerators is typically based on **initial performance balance point**
- **Time passes and accelerators get MUCH faster…**
- Formerly harmless CPU code ends up limiting overall performance!
- Need to address bottlenecks in increasing fraction of code
- **Directives** provide **low cost, low burden**, approach to **improve incrementally** vs. status quo
- **Directives are complementary to lower level approaches** such as CPU intrinsics, CUDA, OpenCL, and they all need to coexist and interoperate very gracefully alongside each other

Multilevel Summation on the GPU: An Amdahl's Law Example From Our Previous Work

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



Multilevel summation of electrostatic potentials using graphics processing units.
D. Hardy, J. Stone, K. Schulten. *J. Parallel Computing*, 35:164-177, 2009.

How Do Directives Fit In?

- Single code base is typically maintained
- Almost “deceptively” simple to use
- Easy route for **incremental, “gradual buy in”**
- **Rapid development cycle**, but success often follows minor refactoring and/or changes to data structure layout
- Higher abstraction level than other techniques for programming accelerators
- In many cases, **performance can be “good enough” due to memory-bandwidth limits**, or based on return on developer time or some other metric

Why Not Use Directives Exclusively?

- Some projects do…but:
  - Back-end runtimes for compiler directives sometimes have unexpected extra overheads that could be a showstopper in critical algorithm steps
  - High abstraction level may mean lack of access to hardware features exposed only via CUDA or other lower level APIs
  - Fortunately, **interoperability APIs** enable directive-based approaches to be used side-by-side with hand-coded kernels, libraries, etc.
  - Presently, sometimes-important capabilities like **JIT compilation of runtime-generated kernels** only exist within lower level APIs such as CUDA and OpenCL

What Do Existing Accelerated Applications Look Like?

I'll provide examples from digging into modern versions of VMD and NAMD that both have already incorporated acceleration in a deep way.

Questions:

- How much code needs to be "fast"
- What fraction runs on accelerator now?
- Using directives, how much more coverage can be achieved, and with what speedup?
- Do I lose access to any points of execution or resource control that are critical for the application's performance?

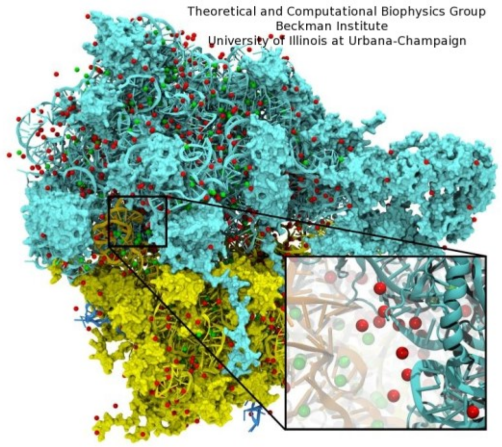
VMD: 10 Years of GPU-Accelerated Computing

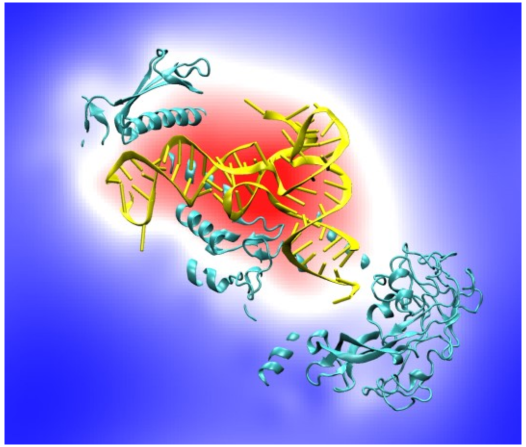
- Has stood the test of time
- Modeling, Visualization, Rendering, and Analysis

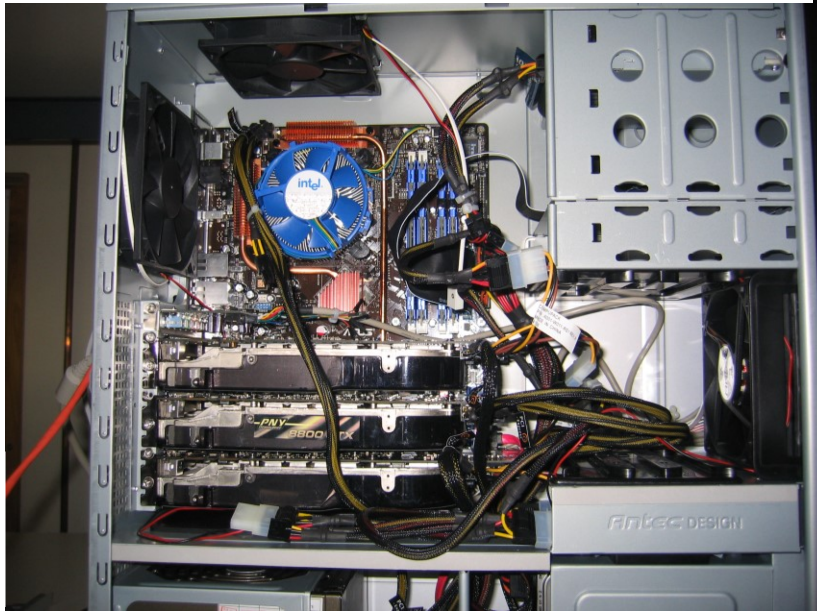
Blast from the past:

CUDA starting with version 0.7 !!!

Quad core Intel QX6700, three NVIDIA GeForce 8800GTX GPUs, RHEL4 Linux


Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champaign





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.

VMD Petascale Visualization and Analysis

- Analyze/visualize large trajectories too large to transfer off-site:
  - User-defined parallel analysis operations, data types
  - Parallel rendering, movie making
- Supports GPU-accelerated Cray XK7 nodes for both visualization and analysis:
  - **GPU accelerated trajectory analysis w/ CUDA**
  - **OpenGL and GPU ray tracing for visualization and movie rendering**
- Parallel I/O rates up to **275 GB/sec** on 8192 Cray XE6 nodes – can read in **231 TB in 15 minutes!**

Parallel VMD currently available on:

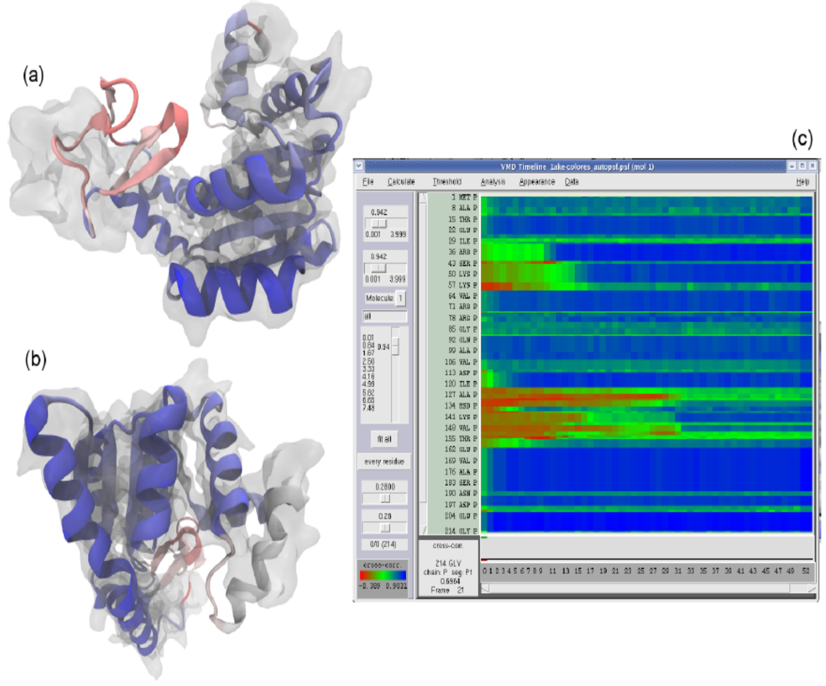
ORNL Titan, NCSA Blue Waters, Indiana Big Red II, CSCS Piz Daint, and similar systems

NCSA Blue Waters Hybrid Cray XE6 / XK7
22,640 XE6 dual-Opteron CPU nodes
4,224 XK7 nodes w/ Telsa K20X GPUs

GPUs Can Reduce MDFF Trajectory Analysis Runtimes from Hours to Minutes

GPUs enable laptops and desktop workstations to handle tasks that would have previously required a cluster, or a *very long wait*…

GPU-accelerated petascale supercomputers enable analyses that were previously impractical, allowing detailed study of very large structures such as viruses



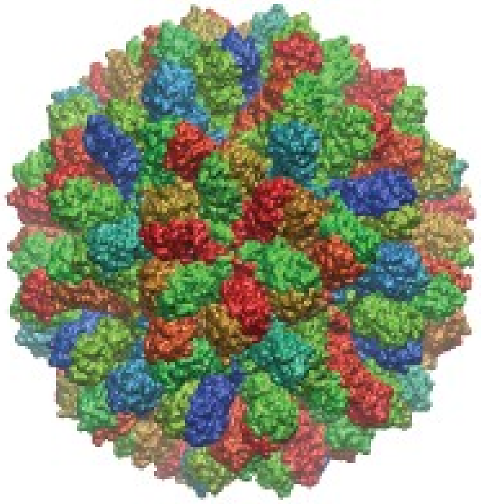
GPU-accelerated MDFF Cross Correlation Timeline

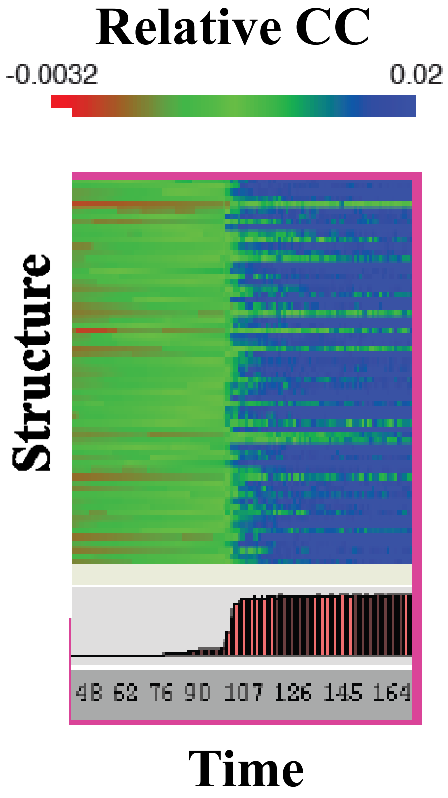
Regions with poor fit **Regions with good fit**

Parallel MDFF Cross Correlation Analysis on Cray XK7

Rabbit Hemorrhagic Disease Virus (RHDV)	
Traj. frames	10,000
Structure component selections	720
Single-node XK7 (projected)	336 hours (14 days)
128-node XK7	3.2 hours 105x speedup
2048-node XK7	19.5 minutes 1035x speedup

Calculation of 7M CCs would take **5 years** using serial CPU algorithm!

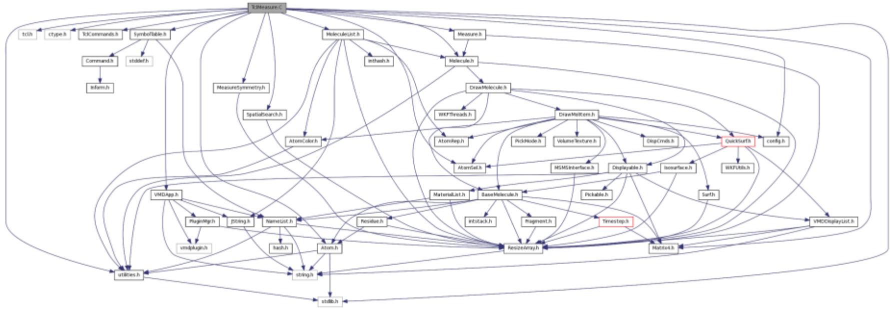

RHDV colored by relative CC

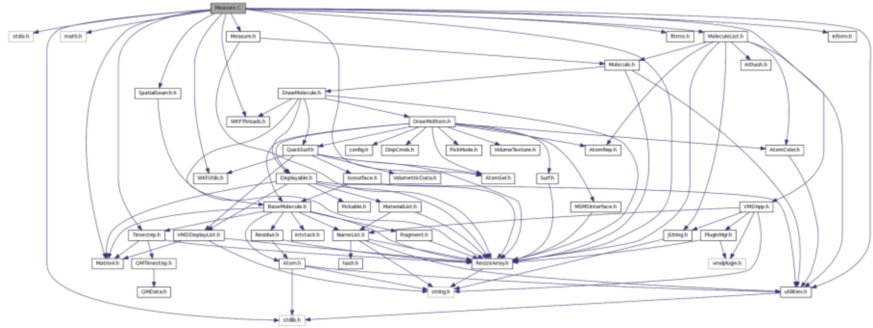


Stone et al., Faraday Discuss., 169:265-283, 2014.

Example of VMD Module Connectivity

- Early progress focused acceleration efforts on handful of high level analysis routines that were the most computationally demanding
- Future hardware requires **pervasive acceleration**
- Top image shows script interface links to top level analytical routines
- Bottom image shows links among subset of data analytics algorithms to **leaf-node functions**





VMD Software Decomposition

VMD Core	**(~230,000 LoC)**
• C++:	140,000 LoC
• Headers:	36,000 LoC
• C:	14,000 LoC
• Tcl bindings:	12,000 LoC
• Python bindings:	8,000 LoC

Hand-coded accelerator and vectorization:

• CUDA:	17,000 LoC
• Intel x86 intrinsics:	2,500 LoC
• IBM POWER intrinsics:	500 LoC
• ARM NEON intrinsics:	100 LoC

Externally developed collective variables module:

• C++:	20,000 LOC
• Headers:	11,000 LOC

Internally+externally developed scripts

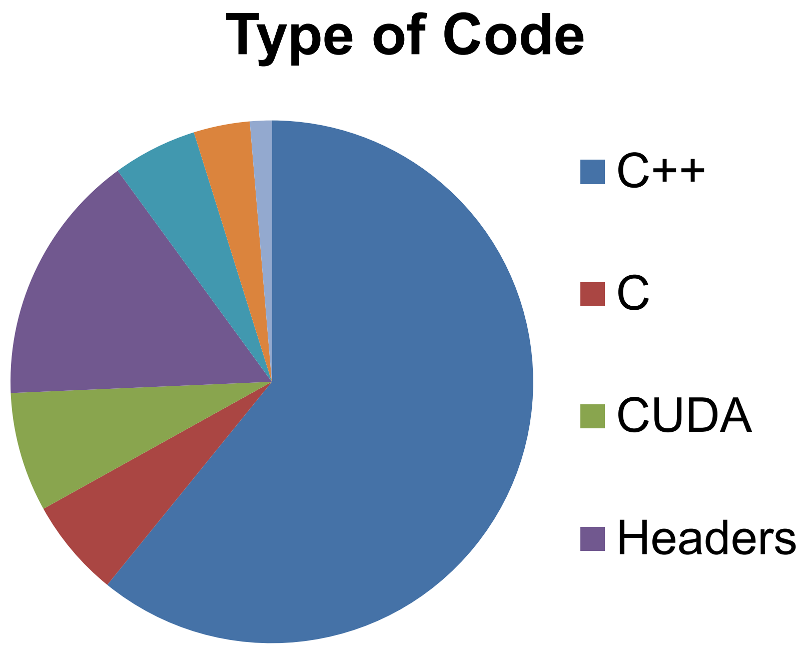
- Tcl / Python scripts: **284,000 LoC**

VMD “plugin” shared lib modules:

• C:	102,000 LoC
• C++:	36,000 LoC
• Headers:	17,000 LoC
• CUDA:	5,000 LoC

VMD Software Decomposition

- All hand-written accelerated or vectorized code (CUDA + CPU intrinsics) represents only 9% of core VMD source code
- Percent coverage of leaf-node analytical functions is lower yet
- Need to evolve VMD toward high coverage of performance-critical analysis code with fine-grained parallelism on accelerators and vectorization

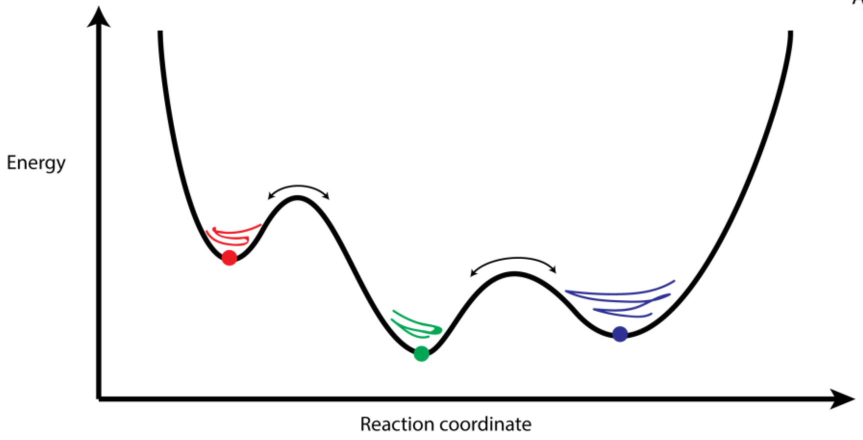


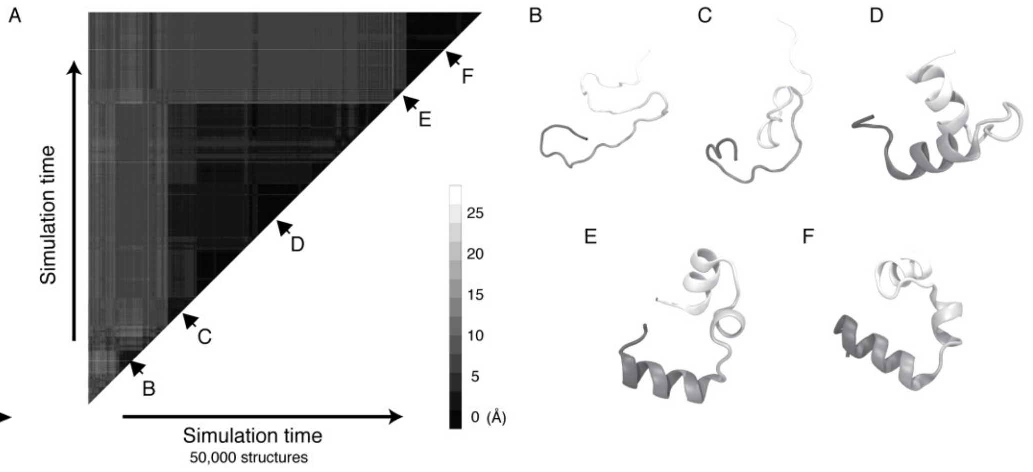
Directive-Based Parallel Programming with OpenACC

- Annotate loop nests in existing code with #pragma compiler directives:
  - Annotate opportunities for parallelism
  - Annotate points where host-GPU memory transfers are best performed, indicate propagation of data
- Evolve original code structure to improve efficacy of parallelization
  - Eliminate false dependencies between loop iterations
  - Revise algorithms or constructs that create excess data movement

- **How well does this work if we stick with "low cost, low burden" philosophy I claim to support?**

Clustering Analysis of Molecular Dynamics Trajectories





GPU-Accelerated Molecular Dynamics Clustering Analysis with OpenACC. J.E. Stone, J.R. Perilla, C. K. Cassidy, and K. Schulten. In, Robert Farber, ed., Parallel Programming with OpenACC, Morgan Kaufmann, Chapter 11, pp. 215-240, 2016.

Serial QCP RMSD Inner Product Loop

- Simple example where directive based parallelism can be applied easily and effectively
- Such a loop is inherently a memory-bandwidth-bound algorithm, so that's the goal for acceleration

```
for (int l=0; l<cnt; l++) {
  double x1, x2, y1, y2, z1, z2;
  x1 = crdx1[l];
  y1 = crdy1[l];
  z1 = crdz1[l];

  G1 += x1*x1 + y1*y1 + z1*z1;

  x2 = crdx2[l];
  y2 = crdy2[l];
  z2 = crdz2[l];

  G2 += x2*x2 + y2*y2 + z2*z2;

  a0 += x1 * x2;
  a1 += x1 * y2;
  a2 += x1 * z2;

  a3 += y1 * x2;
  a4 += y1 * y2;
  a5 += y1 * z2;

  a6 += z1 * x2;
  a7 += z1 * y2;
  a8 += z1 * z2;
}
```

OpenACC QCP RMSD Inner Product Loop

- Simple example where directive based parallelism can be applied easily and effectively
- Such a loop is inherently a memory-bandwidth-bound algorithm, so that’s the goal for acceleration

```
// excerpted code that has been abridged for brevity…
void rmsdmat_qcp_acc(int cnt, int padcnt, int framecrdsz,
                     int framecount, const float * restrict crds,
long i, j, k;
#pragma acc kernels copyin(crds[0:tsz]), copy(rmsdmat[0:msz])
  for (k=0; k<(framecount*(framecount-1))/2; k++) {
    // compute triangular matrix index ‘k’ in a helper function
    // to ensure that the compiler doesn’t think that we have
    // conflicts or dependencies between loop iterations
    acc_idx2sub_tril(long(framecount-1), k, &i, &j);
    long x1addr = j * 3L * framecrdsz;
    long x2addr = i * 3L * framecrdsz;

#pragma acc loop vector(256)
    for (long l=0; l<cnt; l++) {
    // abridged for brevity ...

    rmsdmat[k]=rmsd; // store linearized triangular matrix
  }
}
```

OpenACC QCP RMSD Inner Product Loop Performance Results

- Xeon 2867W v3, w/ hand-coded AVX and FMA intrinsics: 20.7s
- Tesla K80 w/ OpenACC: **6.5s (3.2x speedup)**
- OpenACC on K80 achieved 65% of theoretical peak memory bandwidth, with 2016 compiler and just a few lines of #pragma directives. Excellent speedup for minimal changes to code.
- Future OpenACC compiler revs should provide higher performance yet

Caveat Emptor

- Compilers are not all equal…
- *…sometimes they make me want to scream…*
- …**but they all improve with time**
- If we **begin using directives now** to close the gap on impending doom arising from Amdahl's Law, the **compilers should be robust and performant when it really counts**

Directives and Hardware Evolution

- Ongoing hardware advancements are addressing **ease-of-use gaps** that remained a problem for both directives and hand-coded kernels
- **Unified memory**: eliminate many cases where programmer used to have to hand-code memory transfers explicitly, blurs CPU/GPU boundary
- What about distributing data structures across multiple NVLink-connected GPUs?

Performance Tuning, Profiling Wish List

- Some simple examples on my wish list:
  - **Make directive runtimes more composable** with external resource management, tasking frameworks, and runtime systems, interop APIs are already a start, to build more commonality there.
  - **Help profiling tools to clearly identify** functions, call chains, and resources associated with code produced by compiler **directives** and their runtime system(s), to clearly differentiate from hand-coded kernels, and resources used by other runtimes
  - Allow directive-based programming systems support things like **application-determined hardware scheduling priorities** that encompass both hand-coded and directive-generated kernels
  - Allow programmer oversight about what resources directive kernels are allowed to use, CPU affinity, etc

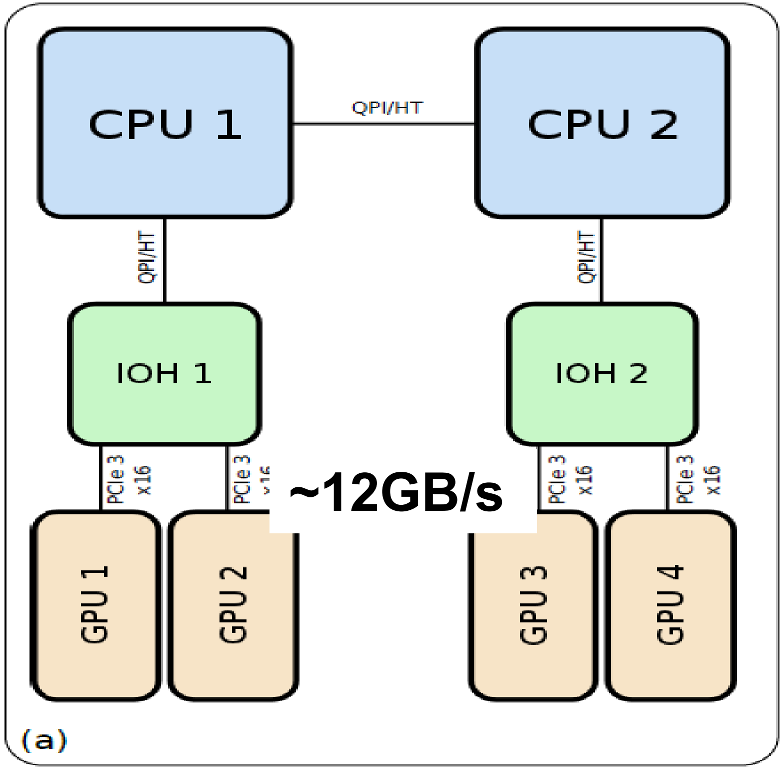
Using CPUs to Optimize Accelerator Performance

- Optimization strategy:
  - Use the CPU to ***"regularize"*** the GPU workload
  - Use optimal/fixed-size data structures, idealize layout for GPU traversal
  - Handle exceptional or irregular work units on the CPUs; GPUs processes the bulk of the work concurrently
  - On average, the GPUs are kept highly occupied, attaining a high fraction of peak performance

Heterogeneous Compute Node

- Dense **PCIe-based** multi-GPU compute node
- Application would **ideally exploit all** of the CPU, GPU, and I/O resources **concurrently**…

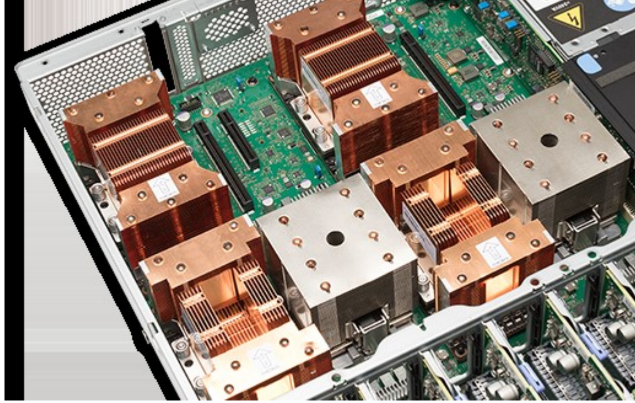
  (I/O devs not shown)

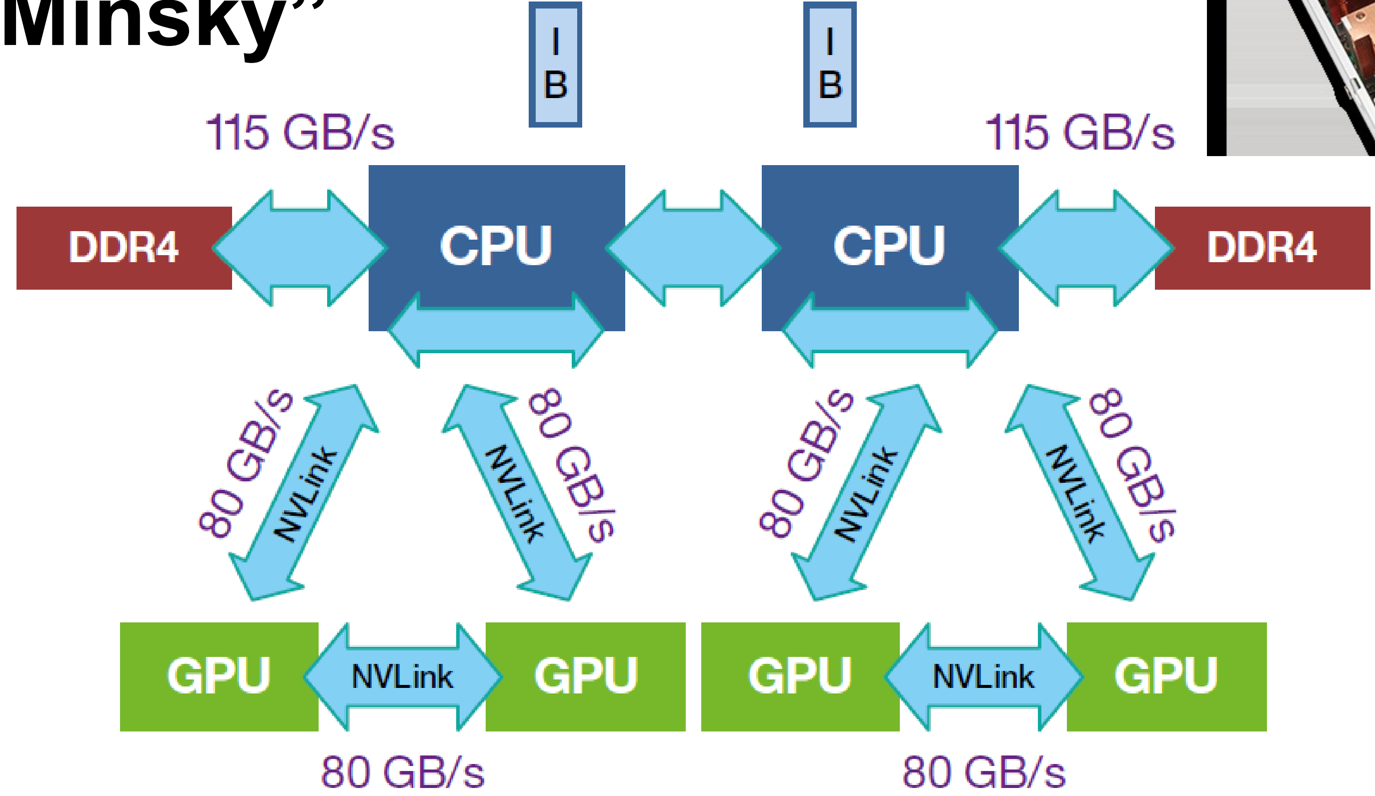


Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations
Michael J. Hallock, John E. Stone, Elijah Roberts, Corey Fry, and Zaida Luthey-Schulten.
Journal of Parallel Computing, 40:86-99, 2014.
http://dx.doi.org/10.1016/j.parco.2014.03.009

IBM S822LC w/ NVLink

“Minsky”





Ongoing VMD Work on POWER

- Early observations about P8+CUDA+NVLink:
  - P8 single-thread perf more of an issue than on x86 for small untuned parts of existing code – **greater need for GPU offload of formerly insignificant host code**
  - **P8+CUDA NUMA-correctness w/ NVLink much more important** than PCIe (e.g. x86) due to **larger benefits/penalties** when NVLink is used effectively vs. not
  - **P8 "Minsky"** systems get extra benefits for algorithms that have lots of host-GPU DMA transfers, where the NVLink interconnect speeds greatly outpeform PCIe

Benefits of P8+NVLink for VMD

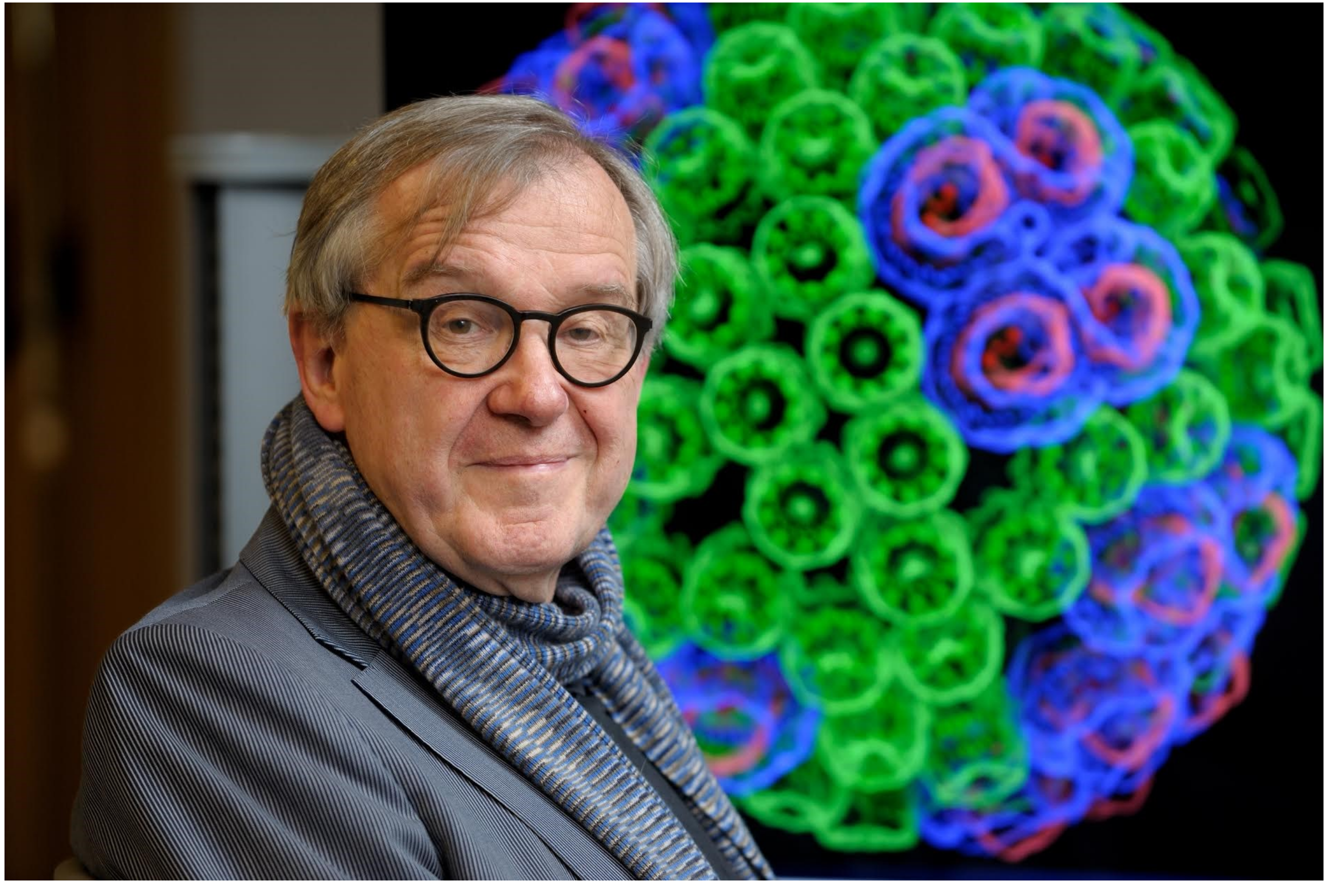
- Rapid access to host-side data too large to fit entirely in P100 GPU memory
  - Many existing VMD CUDA kernels already used this strategy w/ PCIe, performance gains from NVLink are large and immediate
- Rapid peer-to-peer GPU data transfers:
  - **Bypass host** whenever possible, perform nearest-neighbor exchanges for pairwise calculations, e.g. those that arise in algorithms for simulation trajectory clustering
  - **Use aggregate GPU memory** to collectively store/cache large data:
    - Distribute time-varying trajectory timesteps among memories of multiple GPUs
    - High-fidelity ray tracing of scenes containing massive amounts of geometry

Directives and Potential Hardware Evolution

Think of ORNL Summit node as an "entry point" to potential future possibilities…

Questions:

- Would the need for ongoing growth in memory bandwidth among tightly connected accelerators w/ HBM **predict even denser nodes?**
  - Leadership systems use 6-GPU nodes now, how many in 2022 or thereafter?
- As accelerated systems advance, will directives encompass peer-to-peer accelerator operations better?
- What if future accelerators can directly RDMA to remote accelerators (over a communication fabric) via memory accesses?
- In the future, will directives make it easier to program potentially complex collective operations, reductions, fine-grained distributed-shared-memory data structures among multiple accelerators?



“When I was a young man, my goal was to look with mathematical and computational means at the inside of cells, one atom at a time, to decipher how living systems work. That is what I strived for and I never deflected from this goal.” – Klaus Schulten

Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- NVIDIA, PGI, and DOE's ORNL OLCF OpenACC teams/members
- Funding:
  - NIH support: P41GM104601
  - NSF Blue Waters:
    NSF OCI 07-25070, PRAC "The Computational Microscope",
    ACI-1238993, ACI-1440026
  - DOE INCITE, ORNL Titan: DE-AC05-00OR22725

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

- **Challenges of Integrating Stochastic Dynamics and Cryo-electron Tomograms in Whole-cell Simulations.** T. M. Earnest, R. Watanabe, J. E. Stone, J. Mahamid, W. Baumeister, E. Villa, and Z. Luthey-Schulten. J. Physical Chemistry B, 121(15): 3871-3881, 2017.
- **Early Experiences Porting the NAMD and VMD Molecular Simulation and Analysis Software to GPU-Accelerated OpenPOWER Platforms.** J. E. Stone, A.-P. Hynninen, J. C. Phillips, and K. Schulten. International Workshop on OpenPOWER for HPC (IWOPH'16), LNCS 9945, pp. 188-206, 2016.
- **Immersive Molecular Visualization with Omnidirectional Stereoscopic Ray Tracing and Remote Rendering.** J. E. Stone, W. R. Sherman, and K. Schulten. High Performance Data Analysis and Visualization Workshop, IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW), pp. 1048-1057, 2016.
- **High Performance Molecular Visualization: In-Situ and Parallel Rendering with EGL.** J. E. Stone, P. Messmer, R. Sisneros, and K. Schulten. High Performance Data Analysis and Visualization Workshop, IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW), pp. 1014-1023, 2016.
- **Evaluation of Emerging Energy-Efficient Heterogeneous Computing Platforms for Biomolecular and Cellular Simulation Workloads.** J. E. Stone, M. J. Hallock, J. C. Phillips, J. R. Peterson, Z. Luthey-Schulten, and K. Schulten.25th International Heterogeneity in Computing Workshop, IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW), pp. 89-100, 2016.
- **Atomic Detail Visualization of Photosynthetic Membranes with GPU-Accelerated Ray Tracing.** J. E. Stone, M. Sener, K. L. Vandivort, A. Barragan, A. Singharoy, I. Teo, J. V. Ribeiro, B. Isralewitz, B. Liu, B.-C. Goh, J. C. Phillips, C. MacGregor-Chatwin, M. P. Johnson, L. F. Kourkoutis, C. Neil Hunter, and K. Schulten. J. Parallel Computing, 55:17-27, 2016.

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

- **Chemical Visualization of Human Pathogens: the Retroviral Capsids.** Juan R. Perilla, Boon Chong Goh, John E. Stone, and Klaus Schulten. SC'15 Visualization and Data Analytics Showcase, 2015.
- **Visualization of Energy Conversion Processes in a Light Harvesting Organelle at Atomic Detail.** M. Sener, J. E. Stone, A. Barragan, A. Singharoy, I. Teo, K. L. Vandivort, B. Isralewitz, B. Liu, B. Goh, J. C. Phillips, L. F. Kourkoutis, C. N. Hunter, and K. Schulten. SC'14 Visualization and Data Analytics Showcase, 2014.
  ***Winner of the SC'14 Visualization and Data Analytics Showcase**
- **Runtime and Architecture Support for Efficient Data Exchange in Multi-Accelerator Applications.** J. Cabezas, I. Gelado, J. E. Stone, N. Navarro, D. B. Kirk, and W. Hwu. IEEE Transactions on Parallel and Distributed Systems, 26(5):1405-1418, 2015.
- **Unlocking the Full Potential of the Cray XK7 Accelerator.** M. D. Klein and J. E. Stone. Cray Users Group, Lugano Switzerland, May 2014.
- **GPU-Accelerated Analysis and Visualization of Large Structures Solved by Molecular Dynamics Flexible Fitting.** J. E. Stone, R. McGreevy, B. Isralewitz, and K. Schulten. Faraday Discussions, 169:265-283, 2014.
- **Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations**. M. J. Hallock, J. E. Stone, E. Roberts, C. Fry, and Z. Luthey-Schulten. Journal of Parallel Computing, 40:86-99, 2014.

Related Publications

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

- **GPU-Accelerated Molecular Visualization on Petascale Supercomputing Platforms.** J. Stone, K. L. Vandivort, and K. Schulten. UltraVis'13: Proceedings of the 8th International Workshop on Ultrascale Visualization, pp. 6:1-6:8, 2013.
- **Early Experiences Scaling VMD Molecular Visualization and Analysis Jobs on Blue Waters.** J. Stone, B. Isralewitz, and K. Schulten. In proceedings, Extreme Scaling Workshop, 2013.
- **Lattice Microbes: High-performance stochastic simulation method for the reaction-diffusion master equation.** E. Roberts, J. 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. 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. L. 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.

Related 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.

Related 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. E. 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.

Related 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.