Accelerating Protein Coordinate Conversion using GPUs

Mahsa Bayati
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
Email: bayati.m@husky.neu.edu

Jaydeep P. Bardhan
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
Email: jbardhan@ece.neu.edu

David M. King
Department of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627
Email: dking19@u.rochester.edu

Miriam Leeser
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
Email: mel@coe.neu.edu

***Abstract*—For modeling proteins in conformational states, two methods of representation are used: internal coordinates and Cartesian coordinates. Each of these representations contain a large amount of structural and simulation information. Different processing steps require one or the other representation. Our goal is to rapidly translate between these coordinate spaces so that a scientist can choose whichever method he or she would like independent of the coordinate representation required. An algorithm to convert Cartesian to internal coordinates is implemented by taking a protein structure file and the trajectories of protein's atoms within a time frame. The implementation then computes bond distances, bond angles and torsion angles of the atoms. This is implemented on two types of hardware: CPU and a heterogeneous system combining CPU and GPU. The CPU sequential codes in MATLAB and C are compared with MATLAB Parallel Computing Toolbox, OpenMP, and GPU versions in CUDA-C and CUDA-MATLAB. The performance is evaluated on two different protein structure files and their trajectories. Our results show that this computation is well suited to the parallelism offered in modern Graphics Processing Units. We see many orders of magnitude improvement in speed over the original MATLAB code and have brought the computation time from over an hour down to tens of milliseconds.**

I. INTRODUCTION

Computer simulations of biological molecules such as proteins, DNA and therapeutic drugs offer important new insights into biology and the environment [1], and the complexity of such molecules has motivated substantial investments in parallelization and HPC for large-scale molecular dynamics (MD) simulations [2], which integrate the equations of motion over time. A main challenge in large-scale MD simulation is the fact that the resulting systems of ordinary differential equations (ODEs) are very stiff, i.e., there is a large separation in time scales between the fastest and slowest degrees of freedom. As a result, many time steps are required to evolve the system over a statistically meaningful time window, and therefore each step must be computed as quickly as possible.

The present work addresses a different computational challenge: efficient, scalable algorithms to convert between two different representations of molecular coordinates [3], [4]. Representation in *Cartesian coordinates* is intuitive: each atom is associated with a point in Cartesian space, i.e. atom i's center is located at (x_i, y_i, z_i). This representation allows easy file I/O and simple manipulations involving rigid-body motion (rotations and translations). The other representation, known as *internal coordinates*, describes a molecule's atomic positions using chemically relevant features such as the distance between two atoms that are covalently bonded, or the angle formed by a chain of three bonded atoms (Fig. 1). Physical forces between atoms are most naturally expressed in this representation; for example, the force between two bonded atoms is usually modeled as a harmonic spring. Internal coordinates therefore require more complicated data structures and management, which indicate for instance all of the chemical bonds between atoms.

Standard MD simulations convert Cartesian coordinates to internal ones (the *forward coordinate transformation*) at every time step, necessitating fast algorithms for the forward transformation. Many other kinds of calculations also require fast algorithms for the *reverse* transformation, which converts from internal to Cartesian coordinates. Examples include protein structure refinement (improving the quality of experimentally estimated protein structures using modeling) and understanding large changes in protein structure [5], [6]. In these types of applications, internal coordinates offer advantages because the relevant conformational changes involve primarily the dihedral angles (Fig. 1), which effectively reduces the number of degrees of freedom. However, the reverse coordinate transform is less easily parallelized than the forward transform, necessitating optimization or search algorithms that use Cartesian coordinates and have to impose complicated (slow) constraints. In this paper, we focus on the forward direction. Note that a protein can contain thousands to millions of atoms. In this paper we show results for two proteins, the larger of which contains 17,566 atoms. Our goal is to process much bigger molecules, such as DNA which contains over 400 million atoms. In addition, molecule shapes change over time. Thus each atom is represented in a large number of time frames. The results in this paper are for molecules for a thousand or more time frames. We investigate several forms of parallelism for the forward transform, including Mathworks Parallel Computing Toolbox (PCT), OpenMP and CUDA.

978-1-4799-6233-4/14/$31.00 ©2014 IEEE

The rest of the paper is organized as follows. The following section presents background on coordinate transformations in molecular modeling as well as related work. Sections III and IV detail our different parallel implementations and present results. Section V describes directions for further development, and Section VI summarizes the paper.

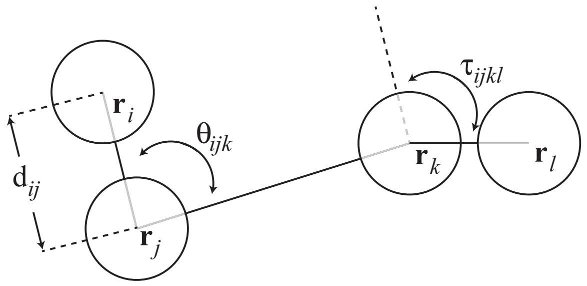

Fig. 1: Diagram of internal coordinates representation. The atom positions are, in Cartesian coordinates, represented by the vectors $\mathbf{r}_i$, $\mathbf{r}_j$, $\mathbf{r}_k$, and $\mathbf{r}_l$. The distance between two bonded atoms is written d_{ij}, the angle between three bonded atoms is written θ_{ijk}, and the dihedral angle between four bonded atoms is written τ_{ijkl}.

II. Background & Related Work

A. Background

Protein geometry changes in response to inter and intra-molecular forces. Molecules stretch, bend, and rotate. This simple description of a molecular system as a mechanical body is usually called the classical system. N atoms in 3D space with 6 degree of freedom can be described directly by a list of Cartesian coordinates. Alternatively, internal variables such as bond length, bond angles and dihedral angle may be used for addressing molecular geometry. This internal representation has been successful for the study of proteins [7]. Cartesian coordinate space is most convenient for direct differentiation of the energy in terms of independent parameters. This coordinate space is the most natural one for implementation of molecular dynamics since the general molecular trajectories of molecule X can be represented as:

$$\{X(t_0), X(t_0, \Delta t), X(t_0, ..., n\Delta t)\} \tag{1}$$

where t_0 is the initial time reference and Δt is the time step.To address the molecular geometry, scientists define analytic expression.For the molecular system of N atoms in Cartesian coordinate space let: $Atom_i = (x_i, y_i, z_i)$.

There are four different internal coordinates that need to be computed from Cartesian coordinates. Each computation is among atoms that have bonds with each other, and works on differing numbers of atoms.

(i) Calculation of the bond length between two bonded

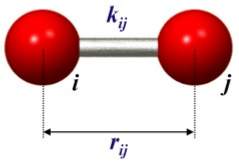

Fig. 2: Bond Length[9]

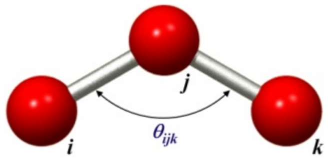

Fig. 3: Bond Angle[9]

atoms(Fig. 2):

$$BondDistance_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \tag{2}$$

where $r_{i,j}$ is a distance vector from i to j. (ii) Calculation of the angle θ_{ijk} formed by a bonded triplet of (Fig. 3):

$$\cos\theta_{ijk} = (\frac{r_{ji}.r_{jk}}{|r_{ji}|.|r_{jk}|}) \tag{3}$$

(iii) Computation of torsion (dihedral) angle (proper and improper) τ_{ijkl} defining the rotation of bond $i - j$ around $j - k$ with respect to $k - l$[7]. In other words, when four bonded atoms are in two planes the angle between these 2 planes is called the torsion angle. The torsion angle based on the position of 4 atoms can fall into two groups: (I) proper (Fig. 4) when four atoms make a linear structure and (II) improper dihedral when one of the atoms is in a central position (Fig. 5). For the forward conversion both dihedral angles use the same formulas [8]:

$$\begin{aligned}
\cos\tau_{ijkl} &= n_{ab}.n_{bc} \\
\cos\tau_{ijkl} &= \frac{a \times b}{\|a\|\|b\|sin\theta_{ab}}.\frac{b \times c}{\|b\|\|c\|sin\theta_{bc}} \\
\sin\tau_{ijkl} &= \frac{(c \times b.a).b}{\|b\|\|c\|sin\theta_{bc}.\|a\|\|b\|sin\theta_{ab}} \\
\tan\tau_{ijkl} &= \frac{\sin\tau_{ijkl}}{\cos\tau_{ijkl}}
\end{aligned} \tag{4}$$

The vectors n_{ab} and n_{bc} denote unit normals to planes spanned by vectors a, b and b, c, respectively, where the distance vectors are: $a = r_{ij}$, $b = r_{jk}$, $c = r_{kl}$.

B. Related Work

Molecular dynamics and protein modeling are extremely computationally demanding which makes them natural candidates

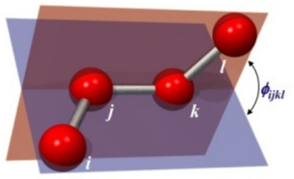

Fig. 4: Proper Dihedral[9]

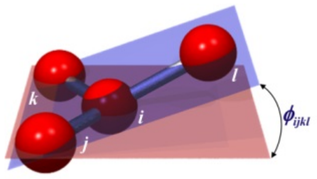

Fig. 5: Improper Dihedral[9]

for implementation on GPUs. With currently available molecular dynamics codes, we can only simulate small and fast protein folding. Some previous studies have implemented specific algorithms used in molecular dynamics and protein modeling. For example, [10] used a GPU to implement a simple implicit solvent (distant dependent dielectric) model. Several algorithms have been implemented [11], including integrator, neighbor lists, and Lenard-Jones potential. GPU implementation of the traditional force field [12] and the challenges and accuracy of it on a GPU [13] have also been presented. Molecular dynamic simulations require a realistic description of the underlying physical system and its molecular interactions [13]. The traditional force field method dates back to 1940 when F. Westhiemer formulated the molecular energy with its geometry; the spatial conformation ultimately obtained is a natural adjustment of geometry to minimize the total internal energy [7]. Since this method uses internal coordinates to calculate bonded forces, it has some similarity with our coordinate conversion. The difference is that their approach concentrates on forces introduced by internal coordinates and the accumulated results to find the energy, while the output from our approach is each set of internal coordinates in each time frame.

III. Implementation

In this paper we focus on the forward direction: Cartesian to internal coordinates. The steps for this conversion are shown in Fig. 6. The initial version of the computation was done in Matlab, and had long run times. To accelerate it, we initially tried the MATLAB Parallel Computing Toolbox (PCT), then translated the code to C, tried OpenMP and finally implemented it to run on an NVIDIA GPU by rewriting it in CUDA-C. Each of these versions is discussed below.

a) Input data: All versions of the code require two input files. The first is a protein structure file (.psf) listing each pair of atoms that has a bond together, groups of three atoms

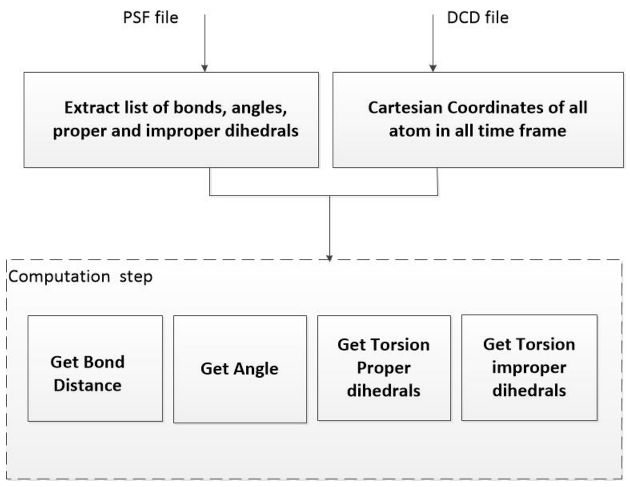
PSF file
DCD file
Extract list of bonds, angles, proper and improper dihedrals
Cartesian Coordinates of all atom in all time frame
Computation step
Get Bond Distance
Get Angle
Get Torsion Proper dihedrals
Get Torsion improper dihedrals

Fig. 6: Overview of Computation

which form an angle, and groups of four atoms forming a torsion angle. The second input is a DCD file which is a single precision binary FORTRAN file [14]; each DCD file shows the trajectory of all protein atoms in a number of time frames. To read these files, MATLAB toolboxes are used. Structure files uses MDToolbox [15], a toolbox for analysis of molecular dynamics(MD) simulation data. The DCD files are read with MatDCD – MATLAB package DCD reading/writing to read the trajectory of atoms [14]. After the data files are read, the program is ready to calculate the internal coordinates by calling the relevant functions.

b) MATLAB and MATLAB PCT: We started with a serial CPU implementation written in MATLAB. In most functions all the computation is nested in a for loop, so we use the *parfor* instruction to get a multi-threaded version of MATLAB using PCT.

c) Translation to C and OpenMP: To reduce the runtime and also move towards the GPU-based implementation using CUDA, the program was rewritten in C. Some preprocessing on both input files is done using MATLAB toolboxes to put them in a readable format for C. We used OpenMP to create a multithreaded C version by putting pragmas before for loops and collapse all nested loops.

d) CUDA: In the GPU based implementation, we designed three kernels: bond length, angle, and torsion. Each thread initiated by one of these kernels computes the internal coordinate which that specific kernel computes for one group of bonded atoms in a specific time frame. Another important step is data transfer between host and GPU device. Since proteins have a large number of atoms, the coordinate conversion has large amounts of data and thus transferring it between host and device is costly. The typical approach is to transfer the data to the device, do all the computation, and then copy the results back to save transfer time. In this approach the kernels are executed serially. However, in this coordinate conversion problem, the kernels are independent, so we can take advantage of streaming and use asynchronous memory transfer to copy data to or from the device while simultaneously doing computation on the device [16]. To achieve better performance, we do asynchronous copying and launch the kernel for each chunk of data to overlap computation.

IV. Results

The coordinate conversion is implemented in MATLAB, MATLAB PCT, C, C with openMP construct, and CUDA-C. We evaluated our implementation on two types of architecture: (i) CPU - Intel Xeon E2620 Sandy Bridge processor with 6 cores and two way hyperthreading, and (ii) GPU - NVIDIA Tesla C2075, with 448 cores and 14 streaming processors. The Tesla GPU has a maximum thread block size of $1024 \times 1024 \times 64$ and grid size 65535×65535 [17]. Our implementation is tested with two different protein structures and trajectory files. The first one is a tripeptide (3 amino acids) with 2443 atoms; its trajectories are simulated in 1000 time frames. The second file is the lysozyme protein [18] with 17566 atoms and its trajectories are given in 2210 time frames. In both cases additional atoms from water are included because the proteins are in solution. The program has 4 outputs, each representing one internal coordinate of the input protein on a time frame. The dimension of the output is the size of internal coordinates multiplied by the number of time frames. The exact number of bonds, angles and improper and proper dihedrals for each file and their output size are given in Table I. Note that all results are for end-to-end processing and include data transfer times. Because the trajectory data is going to be used by all the kernels the transferring time of that is calculated in the total time computation.

According to Table II and Table III, the MATLAB implementation is the slowest. Using MATLAB PCT as a multithreaded version with pool size 12 improved the runtime by a factor of 1.4. The single threaded C version is considerably faster (200x) than MATLAB. The multithreaded C version using OpenMP with 12 threads demonstrates around 3x to 5x speedup compared to serial C; its run time is similar to CUDA-C without streaming. The advantage of OpenMP is its ease of use. Howe ever, since the target architecture, the CPU, has a limited number of threads, as the program gets larger its performance falls behind compared to GPUs which have many more cores and are designed for large problem size and high throughput. The fastest implementation is using CUDA-C with data streaming. Because the result is calculated while data is being transferred, we see an additional 3x speedup over CUDA-C without data streaming, and a total speed-up of 13x-20x (depending on the size of the protein) compared with sequential C. Since all the toolboxes are easy to access in MATLAB we also compiled CUDA kernels and called the .ptx file from our MATLAB implementation. The CUDA-MATLAB implementation is not as efficient as the CUDA-C version but it is a good choice for those who want to take advantage of GPU processing while programming in MATLAB. Fig. 7 and Fig. 8 summarize the run time results for both files. Note that the y axis is a logarithmic scale; speedups from the original version are substantial.

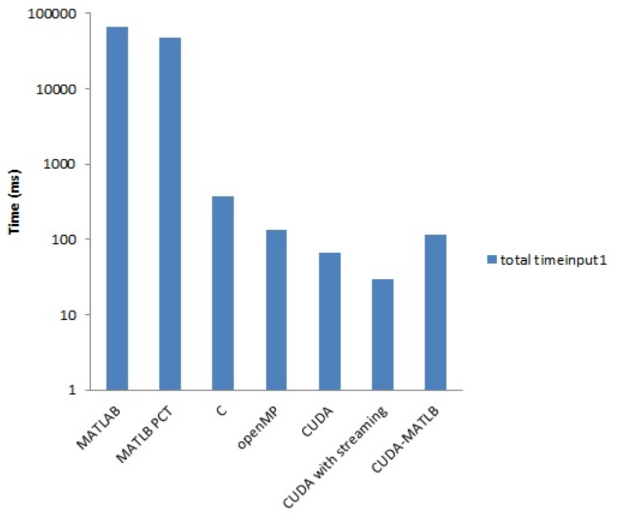

Fig. 7: Total Time (ms) for the First Protein

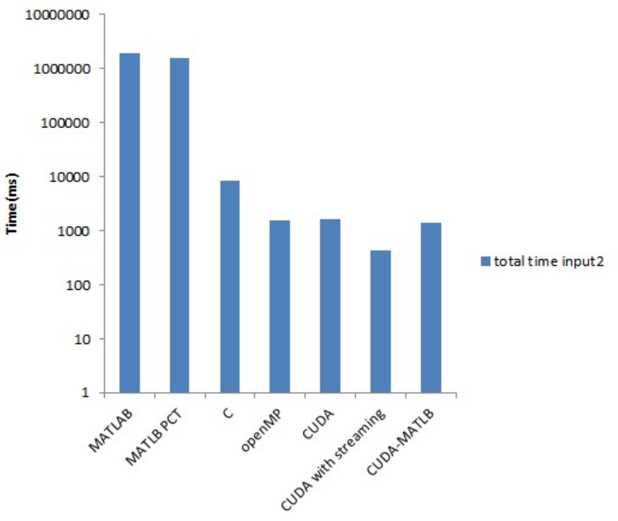

Fig. 8: Total Time (ms) for the Second Protein

TABLE I: Size of Input and Output for two Protein Datasets

	Input(I)	Input(II)
Number of Atoms	2443	17566
Number of Frames	1000	2210
Number of Bonds	1635	12185
Number of Angles	843	7702
Number of proper dihedrals	41	3293
Number of improper dihedrals	4	204
Size of Bond distance output	1635×1000	12185×2210
Size of Angle output	843×1000	7702×2210
size of Torsion output(proper dihedral)	41×1000	3293×2210
size of Torsion output(improper dihedral)	4×1000	204×2210

TABLE II: Time (ms) Result for First protein

	MATLAB	MPCT	C	OpenMP	CUDA-C without streaming	CUDA-C with streaming	CUDA-MATLAB
GetbondDistance	4344.1	3325.3	128	52	17.02		42
GetAngle	30820	22493.5	242	76	24		44.6
GetTorsion(proper dihedral)	10984	7885.8	10	3	0.23		18.8
GetTorsion(improper dihedral)	1390	1067.9	1	0.45	0.2		12.7
Total Computation	66812	48449	370	134	66.43	29.88	114.4

TABLE III: Time (ms) Result for Second Protein

	MATLAB	MPCT	C	OpenMP	CUDA-C without streaming	CUDA-C with streaming	CUDA-MATLAB
GetbondDistance	358009.5	255693.7	2097	458	450.63		437.2
GetAngle	757829.3	549829.6	4825	734	653.1		383.7
GetTorsion(proper dihedral)	728004	530758.2	1459	261	76.71		149.7
GetTorsion(improper dihedral)	43080.7	27280.3	103	22	49.75		20.4
Total Computation	1879300	1547926.8	8484	1574	1643.61	439.1	1431

V. Future Work

In this paper we have focused on the forward direction. Our current implementation produces significant acceleration. In the future we would like to be able to handle much larger proteins with millions of atoms. To handle these, we need to investigate using multiple GPUs. In addition we plan to investigate the reverse direction: internal coordinates to Cartesian. The dependency caused by relative coordinates makes this conversion more challenging to parallelize. Our ultimate goal is to accelerate conversion between these two representations so a scientist can choose either representation based on the tool they wish to use, and not be concerned with translation time.

VI. Conclusions

We have presented the translation of Cartesian coordinates to internal coordinates to represent large proteins. Our CUDA-C implementation, using data streaming and overlapping computation outperforms other parallel versions. The results shows that the CUDA code takes approximately 30 milliseconds for a protein model with 2,443 atoms, and 440 milliseconds for a protein with 17,566 atoms, which is approximately 20 times faster than the single threaded C implementation and around 5 times faster than the multithreaded C with OpenMP version.

Acknowledgments

The authors would like to thank Mathworks for their support. The work of JPB and DMK has been supported by the National Institute of General Medical Sciences (NIGMS) of the National Institutes of Health (NIH) under award number R21GM102642. The content is solely the responsibility of the author and does not necessarily represent the official views of the National Institutes of Health.

References

[1] M. Karplus, "Molecular dynamics of biological macromolecules: a brief history and perspective," *Biopolymers*, vol. 68, pp. 350–358, 2003.

[2] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, "Scalable molecular dynamics with NAMD," *J. Comput. Chem.*, vol. 26, pp. 1781–1802, 2005.

[3] M. Totrov and R. Abagyan, "Efficient parallelization of the energy, surface, and derivative calculations for internal coordinate mechanics," *J. Comput. Chem.*, vol. 15, pp. 1105–1112, 1994.

[4] C. D. Schwieters and G. M. Clore, "Internal coordinates for molecular dynamics and minimization in structure determination and refinement," *J. Magnetic Resonance*, vol. 152, pp. 288–302, 2001.

[5] J. R. López-Blanco, R. Reyes, J. I. Aliaga, R. M. Badia, P. Chacón, and E. S. Quintana-Ortí, "Exploring large macromolecular functional motions on clusters of multicore processors," *J. Comput. Phys.*, vol. 246, pp. 275–288, 2013.

[6] J. R. Wagner, G. S. Balaraman, M. J. M. Niesen, A. B. Larsen, A. Jain, and N. Vaidehi, "Advanced techniques for constrained internal coordinate molecular dynamics," *J. Comput. Chem.*, vol. 34, pp. 904–914, 2013.

[7] T. Schlick, *Molecular Modeling and Simulation: an Interdisciplinary Guide*. Springer-Verlag New York, 2002.

[8] A. Neumaier, "Molecular modeling of proteins and mathematical prediction of protein structure," *SIAM review*, vol. 39, no. 3, pp. 407–460, 1997.

[9] Eindhoven University biomedical group, "Internal coordinates theory figures," http://cbio.bmt.tue.nl/pumma/index.php/Theory/Potentials.

[10] E. Elsen, V. Vishal, M. Houston, V. S. Pande, P. Hanrahan, and E. Darve, "N-body simulations on gpus," *CoRR*, vol. abs/0706.3060, 2007.

[11] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten, "Accelerating molecular modeling applications with graphics processors," *Journal of Computational Chemistry*, vol. 28, no. 16, pp. 2618–2640, 2007. [Online]. Available: http://dx.doi.org/10.1002/jcc.20829

[12] M. S. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. Legrand, A. L. Beberg, D. L. Ensign, C. M. Bruns, and V. S. Pande, "Accelerating molecular dynamic simulation on graphics processing units," *Journal of Computational Chemistry*, vol. 30, no. 6, pp. 864–872, 2009. [Online]. Available: http://dx.doi.org/10.1002/jcc.21209

[13] M. Taufer, N. Ganesan, and S. Patel, "Gpu-enabled macromolecular simulation: Challenges and opportunities," *Computing in Science & Engineering*, vol. 15, no. 1, pp. 56–65, Jan 2013.

[14] J. Gullingsrud, "MatDCD – Matlab package DCD reading/writing," http://www.ks.uiuc.edu/Development/-MDTools/matdcd/.

[15] Mathworks, "MD Toolbox," http://mdtoolbox.readthedocs.org/en/-latest/introduction.html,.

[16] NVIDIA, “Optimizing CUDA,” http://www.sdsc.edu/us/training/assets/-docs/NVIDIA-04-OptimizingCUDA.pdf, Last Accessed May 2014.

[17] ——, “Tesla C2075 Guide,” http://www.nvidia.com/docs/IO/43395/BD-05880-001_v02.pdf.

[18] Protein Data Bank, “Lysozyme(1HEL),” http://www.rcsb.org/-pdb/explore/explore.do?structureId=1HEL.