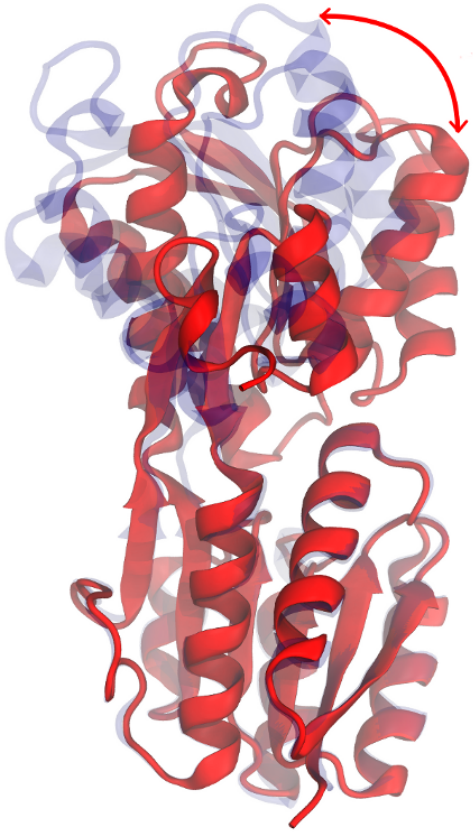
University of Illinois at Urbana-Champaign
Beckman Institute for Advanced Science and Technology
NIH Resource for Macromolecular Modeling and Bioinformatics
Theoretical and Computational Biophysics Group

GPU Accelerated Molecular Dynamics Simulation, Visualization, and Analysis



Authors:
Ivan Teo
Juan Perilla
Rezvan Shahoei
Ryan McGreevy
Chris Harrison
May 19, 2014

Please visit www.ks.uiuc.edu/Training/Tutorials/ to get the latest version of this tutorial, to obtain more tutorials like this one, or to join the `tutorial-l@ks.uiuc.edu` mailing list for additional help.

Contents

1. Introduction

This tutorial will demonstrate how to use features in NAMD and VMD to harness the computational power of graphics processing units (GPUs) to accelerate simulation, visualization and analysis. You will learn how to drastically improve the efficiency of your computational work, achieving large speedups over CPU only methods. You will explore ways of investigating large multimillion atom systems or long simulation timescales through easy to use features of NAMD and VMD on readily available hardware. Please note that completeing the tutorial examples will require a computer with a CUDA-capable NVIDIA GPU. Please see Section **2.1.** for more information.

1.1. Introduction to GPU Computing

Over the past decade, physical and engineering practicalities involved in microprocessor design have resulted in flat performance growth for traditional single-core microprocessors. Continued microprocessor performance growth is now achieved primarily through multi-core designs and through greater use of data parallelism, with vector processing machine instructions. Currently, the year-to-year growth in the number of processor cores roughly follows the growth in transistor density predicted by Moore's Law, doubling every two years. At the forefront of this parallel computation revolution are graphics processing units (GPUs), traditionally used for visualization.

Graphics workloads contain tremendous amounts of inherent parallelism. As a result, GPU hardware is designed to accelerate data-parallel calculations using hundreds of arithmetic units. The individual GPU processing units support all standard data types and arithmetic operations, including 32-bit and 64-bit IEEE floating point arithmetic. State-of-the-art GPUs can achieve peak single-precision floating point arithmetic performance of 2.0 trillion floating point operations per second (TFLOPS), with double-precision floating point rates reaching approximately half that speed. GPUs also contain large high-bandwidth memory systems that achieve bandwidths of over 200 GB/sec in recent devices. The general purpose computational capabilities of GPUs are exposed to application programs through the two leading GPU programming toolkits: CUDA [6], and OpenCL [5].

1.2. GPU Computing in NAMD and VMD

NAMD and VMD utilize GPUs to accelerate an increasing number of their most computationally demanding functions, resulting in significant speed increases. Many algorithms involved in molecular modeling and computational biology applications can be adapted to GPU acceleration, commonly increasing performance by factors ranging from 10× to 30× faster, and occasionally as much as 100× faster, relative to contemporary multi-core CPUs [10, 11, 9]. GPU-accelerated desktop workstations can now provide performance levels that used to require a cluster, but without the complexity involved in managing multiple machines or high-performance networking. Users of NAMD and VMD can now perform many computations on laptops and modest desktop machines which would have been nearly impossible without GPU acceleration. For example, NAMD users can easily perform simulations on large systems containing hundreds of thousands and even millions of atoms thanks to GPUs. The time-consuming non-bonded calculations on so many atoms can now be performed on a GPU at 20 times the speed of a single CPU core. VMD users can smoothly and interactively animate trajectories using visualization techniques such as the display of molecular orbitals or QuickSurf for surface representations. In the case of visualizing molecular orbitals, VMD's GPU algorithm obtains a 125× speedup over the CPU.

2. Introduction to Simulations using GPUs

The performance benefit NAMD's GPU acceleration feature is most clearly demonstrated by simulation of large systems, e.g. with $> 10^5$ atoms, with sufficient work to keep the GPU busy [7].

This section will guide you through the simulation of such a large system with and without a GPU, for the purpose of comparison between the two cases. For this section, please use as your working directory `gpu-tutorial/gpu-tutorial_data/1-largeSims/`.

2.1. How to run NAMD using GPUs.

To benefit from GPU acceleration you will need a CUDA build of NAMD [10, 11, 9] and a recent high-end NVIDIA video card. CUDA builds will not function without a CUDA-capable GPU. You will also need to be running the NVIDIA Linux driver version 270.41.19 or newer (released Linux binaries are built with CUDA 4.0, but can be built with newer versions as well).

Finally, the libcudart.so.4 included with the binary (the one copied from the version of CUDA it was built with) must be in a directory in your LD_LIBRARY_PATH before any other libcudart.so libraries. For example, when running a multicore binary (recommended for a single machine):

```
setenv LD_LIBRARY_PATH ".:$LD_LIBRARY_PATH"
(or LD_LIBRARY_PATH=".:$LD_LIBRARY_PATH"; export LD_LIBRARY_PATH)
./namd2 +idlepoll +p4 <configfile>
```

For more information on running NAMD on the GPU, please see the NAMD User's Guide

2.2. Looking at the System

You will now proceed to examine the example system for this section. The system is comprised of a mechanosensitive channel of small conductance (MscS) embedded in a lipid bilayer and solvated in a water box of dimensions 324Å$\times$ 324Å$\times$ 230Å. The MscS allows outflow of ions when the cell experiences osmotic shock, while maintaining charge balance across the membrane and selectively retaining crucial ions such as glutamate. The diffusive behavior of ions around and through the MscS is hence a subject of considerable scientific interest.

1 Open VMD. Go to 'TkConsole' from 'Extensions'

2 In the TkConsole, navigate to the folder containing the files for section 2.

3 Next, open the PDB file of the system by typing in the TkConsole:

```
mol load pdb mscs.pdb
```

4 Take some time to inspect the system. Observe that some useful information about the system has been loaded in the command terminal window. In particular, there are approximately 2.3 million atoms.

5 Close VMD.

2.3. Basic benchmarking of NAMD performance.

Before starting actual runs, it is advisable to take stock of your simulation requirements and estimate how much running time it would take to finish running the simulation given the computing resources at your disposal. Benchmarking serves as a straightforward way of doing so. In the midst of any simulation run, NAMD measures the average rate of calculation over the elapsed simulation time. The rate of calculation depends on many factors, among which are the system size, configuration parameters, and the computational resources allocated to the simulation. Thus it is more sensible to empirically measure the rate over elapsed timesteps for each simulation than to perform an extremely complicated *a priori* calculation of the rate. Here, you will perform short equilibration runs of the MscS system and subsequently extract benchmark information from the generated logfiles.

1 Let us begin by taking a look at the NAMD configuration file for the benchmark run. In the folder for this section, use your favorite text editor and open `benchmark_cpu.conf`.

2 Notice the small number of timesteps near the end of the file: `run 1000`. NAMD performs benchmark measurements after 400 timesteps. However, averaging over several benchmarks gives a more reliable estimate.

3 Now close the editor. Perform the benchmark run on just CPUs by typing in the command prompt:

```
namd2 benchmark_cpu.conf > benchmark_cpu.log
```

4 Create the configuration file for the GPU benchmark by opening `benchmark_cpu.conf` with a text editor and setting `outputName` to `benchmark_gpu`. Exit and save the file as `benchmark_gpu.conf`. Note the superficial difference between the CPU and GPU configuration files; the key procedural difference between running with and without GPUs is instead in how NAMD is called on the command prompt.

5 Perform the benchmark run on CPUs together with a GPU by typing in the command prompt:

```
namd2 +idlepoll benchmark_gpu.conf > benchmark_gpu.log
```

6 Examples of `benchmark_cpu.conf` and `benchmark_gpu.conf`, as well as `benchmark_cpu.log` and `benchmark_gpu.log` have been saved in `gpu-tutorial/gpu-tutorial_data/1-largeSims/examples/`. In case of time constraints or failure in a previous step, please transfer the example files to your working directory and use them as you proceed.

7 After each run has finished, the benchmark information can be extracted from the respective logfiles. On a Linux or Mac, this can be easily done by typing into the command prompt:

```
grep Benchmark benchmark_cpu.log
```

or

```
grep Benchmark benchmark_gpu.log
```

You should see a line(s) of text that looks like:

```
Info:  Benchmark time:  12 CPUs 5.99984 s/step 34.7213 days/ns
10434.7 MB memory
```

8 Based on the `s/step` and `days/ns` numbers, approximately how long would it take, with and without the GPU, to run, say, 10^6 timesteps? What about 10 ns?

2.4. Simulating 2.3 million atoms on CPUs and GPUs.

You are now ready to perform actual equilibration runs on the MscS system. Due to time constraints, you will perform 2-hour (clock time) runs.(Feel free to perform longer runs if time allows.) Judging from the benchmarks obtained from the previous system, how many ns do you think you would be able simulate, both with and without the GPU? Record your estimate for comparison with the actual results later.

1 The benchmark configuration is virtually identical to that of the actual run. Hence, you can prepare the configuration file for the actual run simply by editing the benchmark configuration file. Use a text editor to open `benchmark_cpu.conf`.

2 Set `outputName` to `equil_cpu`, then scroll down to the bottom of the file and change the number of timesteps:

```
run 1000000
```

Of course, 10^6 should exceed the number of timesteps in your estimate. However, the simulation can be halted in 2 hours for you to view the results. In actual runs, you should set the number of timesteps according to your benchmark estimates.

3 Save the edited configuration file as `equil_cpu.conf`.

4 Create also, from `benchmark_gpu.conf`, the GPU configuration file `equil_gpu.conf` using the same procedure in the preceding steps.

5 Run the simulation with and without the GPU by typing in the command prompt:

```
namd2 equil_cpu.conf > equil_cpu.log &
namd2 +idlepoll equil_gpu.conf > equil_gpu.log &
```

After each of these commands, a process id should have been printed to the terminal. If you are running NAMD on a computer running linux or OSX, you can now use the linux "at" command to kill these two processes in 2 hours. To do this type `at "now + 2 hours"`**. This command will give you a prompt** `at>`**, at which you should enter** `at>kill _pid_`**, where** `_pid_` **is the process id printed after starting the namd run. You can now exit the prompt with "ctrl-d". You should do this process for both the cpu and gpu simulations. This will set up jobs to kill the namd simulations 2 hours from the time you entered the command..**

6 Examples of the `.conf` and `.log` files in this section have been saved in `gpu-tutorial/gpu-tutorial_data/1-largeSims/examples/`. In addition, you will also find the trajectory files `equil_cpu.dcd` and `equil_gpu.dcd` in the same location should you wish to visualize them in VMD.

2.5. Comparison of CPU and GPU performance.

1 Use a text editor to open the logfiles `equil_cpu.log` and `equil_gpu.log`. How many timesteps were run in each case?

2 Next, use `grep` to inspect the benchmarks in each logfile as you did for the benchmark runs. How do they compare to your previous benchmark results?

3 Based on your observations, how much faster did the GPU simulation run as compared to the CPU simulation? Do you think the same performance boost would be observed for a small system of, say, 5000 atoms?

3. GPU Enhanced Visualization

In addition to being computationally demanding to simulate, large biomolecular structures can be difficult to visualize as well. Not only do large systems push the abilities of the GPU to display the structures, but displaying structures such that interesting details can be easily discerned is also a challenge.

Molecular surface visualization allows researchers to see where structures are exposed to solvent or contact each other, and to view the overall architecture of large biomolecular complexes such as trans-membrane channels and virus capsids. VMD is capable of calculating surfaces quickly via the GPU-accelerated QuickSurf representation, which achieves performance orders of magnitude faster than the conventional Surf and MSMS representations. Hence, users can easily set up interactive displays of molecular surfaces for multi-million atom complexes, e.g. large virus capsids. Furthermore, QuickSurf enables smooth interactive animation of moderate-sized biomolecular complexes consisting of a few hundred thousand to a million atoms.

3.1. Rendering surfaces the "old" way.

In this section, you will be acquainted with surface representations using non-GPU methods.

1 Open VMD. Go to 'TkConsole' from the 'Extensions' tab on the top of VMD Main menu.

2 Ensure your working directory is the same as in Section **2.** `gpu-tutorial/gpu-tutorial_data/1-largeSims/` In the TkConsole type:

```
mol load psf mscs.psf pdb mscs.pdb
```

3 In the selected atoms field, type "segname PA PB PC PD PE PF PG".

4 For the Drawing Method, choose 'Surf' from the drop-down menu. Notice how long it takes to calculate the surface and apply it to the structure. This surface is rather slow in both generation and display for systems over several hundred atoms. The Surf calculation is quite exact and will show complete detail even when it isn't needed. The use of disk space as an interprocess communications medium takes up about half of the run time. In addition, the user's options are limited to changing the radius of the probe used in calculating the surface and the ability to render a wireframe representation of the surface.

5 If displaying one frame using Surf is slow, playing a trajectory with Surf will be impractical. For later comparison, add the GPU equilibration trajectory from Section **2.**. If you did not generate this file, one has been provided in: gpu-tutorial/gpu-tutorial_data/1-largeSims/examples/

```
mol addfile equil_gpu.dcd
```

6 Now attempt to play the trajectory or even just skip one frame forward from the VMD Main window.

7 There is another surface representation, MSMS which is faster than Surf and gives the user slightly more options. You can try using MSMS by selecting it from the Drawing Method menu. If the representation fails to load, try selecting fewer segments, e.g. 'segname PA'. MSMS may fail because while it can be faster than Surf, it is still quite limited by the size of the system it can work on.

8 Alternatively we could use space-filling models to represent our structure such as CPK or VDW, the latter also giving us an idea of the volume and surface of the protein. Try applying these Drawing Methods and subsequently rotating the structure. Notice how these representations are still slower than we would like.

3.2. Introducing GPU-accelerated QuickSurf.

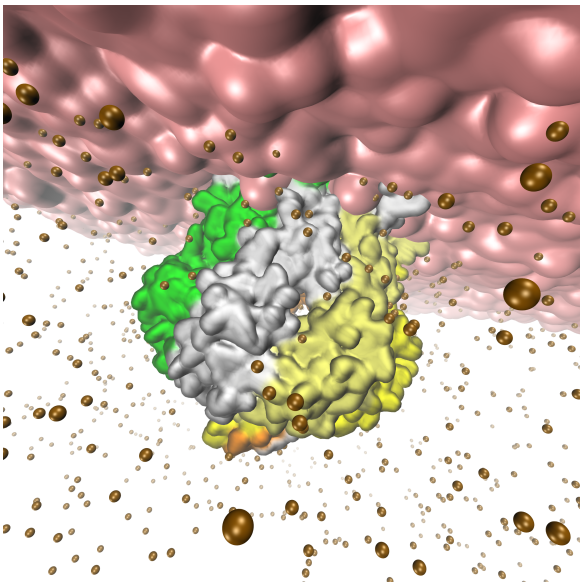


Figure 1: MscS in membrane with QuickSurf representation. Ions are represented using VdW.

1 Now select the QuickSurf representation from the Drawing Method menu. Surprised by how fast the representation loaded? As you can see, QuickSurf lives up to its name by using the computational power of GPUs to calculate quickly the surface representation.

2 In addition to being fast, QuickSurf gives the user many useful options for controlling the representation. We can change the Radius Scale, Density Isovalue or Grid Spacing individually, or use the Resolution slider which will change them in tandem to give the desired resolution. Try adjusting the resolution and see how quickly the representation responds. This can be quite useful for changing on the fly from a high resolution, when you want to see detail, to a low resolution when you want the detail obscured.

3 Aside from faster rendering of surfaces than the traditional Surf and MSMS methods, QuickSurf also allows us to view an entire trajectory with a surface representation. Try playing the trajectory as you attempted before. Note that you can also adjust the resolution even while the trajectory is playing.

3.3. Usefulness of Surface Representations

Having a fast surface representation is great, but why might we want to visualize surfaces in the first place? As an example, look at the structure of the MscS in the QuickSurf. It consists of a seven-fold

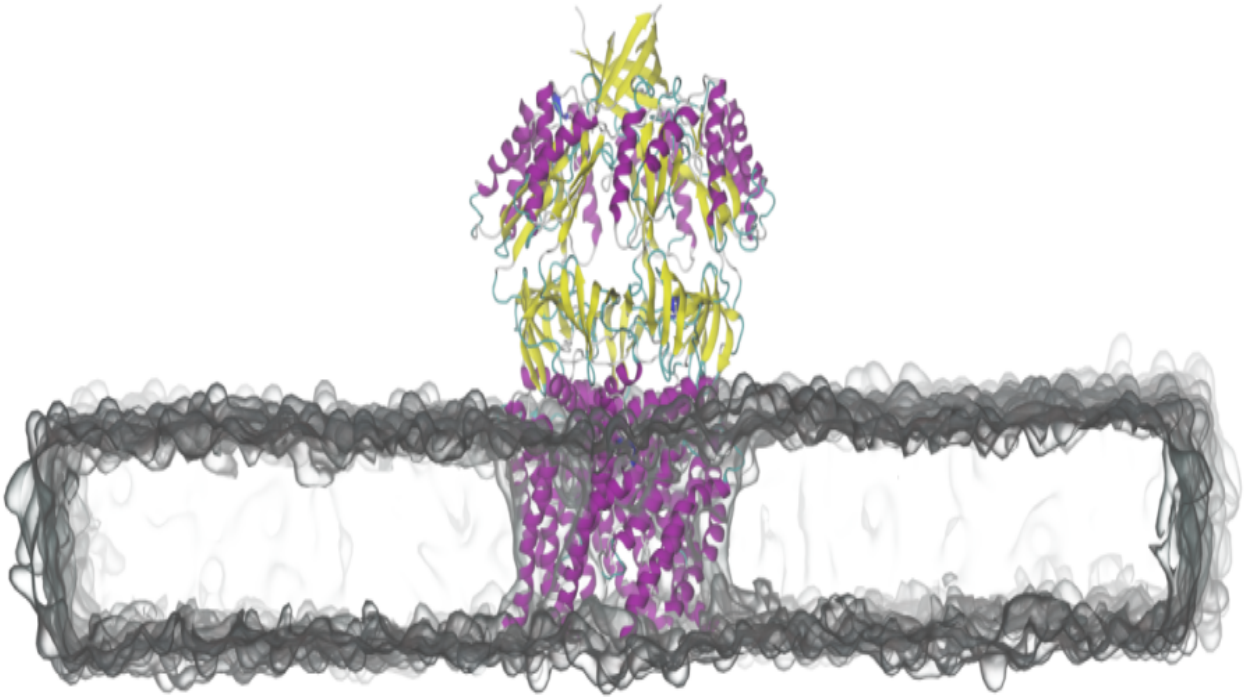


Figure 2: MscS with transparent QuickSurf membrane.

symmetric heptamer forming a balloon-like cytoplasmic domain attached to the transmembrane domain. There are several openings into the protein interior - the transmembrane channel, seven identical windows lining the balloon structure, and one at the C-terminus on the bottom of the balloon structure. Using a QuickSurf representation, you can quickly get a rough impression of how the sizes of these openings compare with one another. In particular, you can immediately tell that the C-terminus window is much smaller than the others. In fact, it is the only window which is impermeable to ions. The capability of running trajectories in QuickSurf makes it easy to see if windows are becoming wider or narrower, making it more apparent if, for example, a channel is opening or closing.
Next, we will modify the representation of different parts of the system to investigate how we can reduce the detail of certain components without removing them entirely.

1 Go to the Graphical Representation menu and create a selection “segname PA PB PC PD PE PF PG” in NewCartoon Drawing Method.

2 Create another representation of only “lipids”. by clicking the ‘Create Rep’ button and typing ‘lipids’ into the Selected Atoms field.

3 Select QuickSurf for the lipids representation.

4 Under the “Draw Style” tab you can find an option named “Material”. Select “Transparent” for the lipids representation.

5 Tune the Resolution of QuickSurf to somewhere between 1.5 and 2. Notice how the lipid detail is reduced while still giving you a clear representation of the volume and overall shape of the membrane. Zoom into the lipid bilayer and notice how you can clearly see the transmembrane portion of the protein without losing the visualization of the lipids. An example of this view can be seen in Fig. 2

While studying ion dynamics in the MscS system, you probably would not want to be encumbered by the fine details of lipid conformations. Using QuickSurf, you can reduce the detail level of the lipid bilayer. In general, the QuickSurf representation helps you to reduce the complexity of parts of the system without removing them entirely.

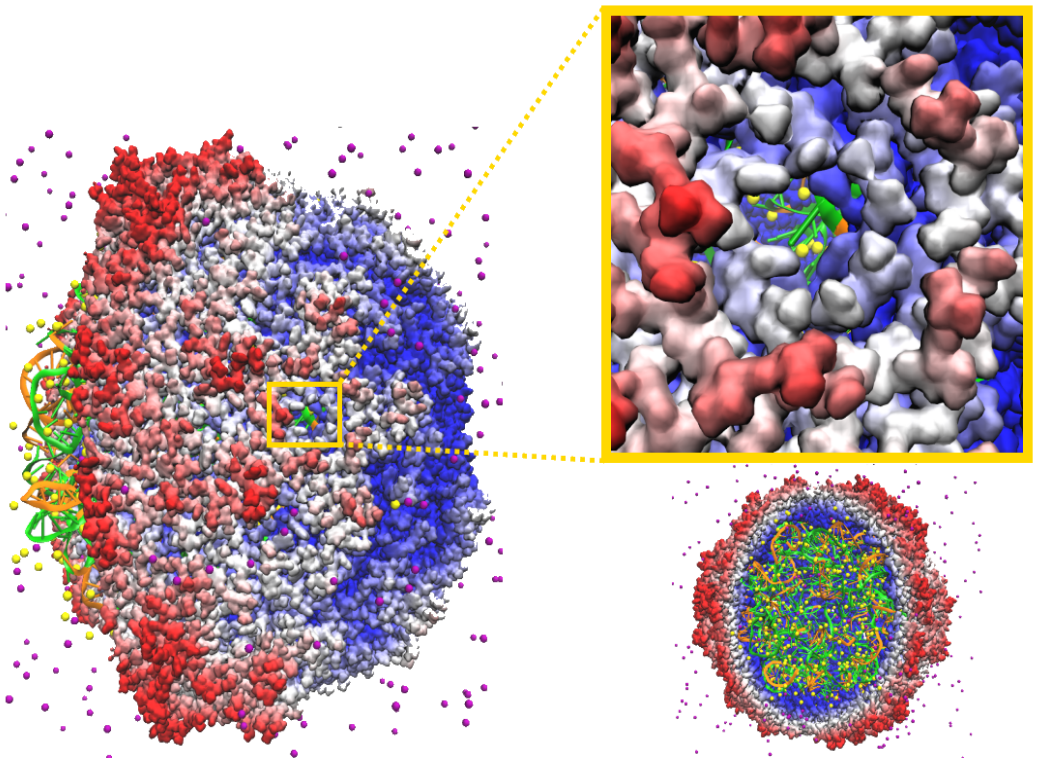


Figure 3: Front and side views of the Satellite Tobacco Mosaic Virus (STMV) using QuickSurf representation for the viral capsid. Note the hole in the capsid now visible due to surface representation.

Another important aspect of visualization is discovery of features and phenomena which would otherwise have been hidden. The right representation can drastically change how you perceive the structure being modeled. For example, the capsid of the Satellite Tobacco Mosaic Virus (STMV) can be represented using QuickSurf to visualize the volume occupied by the proteins without unecessary detail.

In addition to being visually appealing, when using QuickSurf we now notice that the capsid contains small holes in Fig. 3 which would have been much more difficult or even impossible to see with other representations.

4. GPU Accelerated Molecular Dynamics (aMD)

Accelerated molecular dynamics (aMD) is an all-atom enhanced sampling technique that enables the study of large conformational changes in proteins which normally occur on millisecond or longer time scales. For example, using the aMD method on GPUs the bovine pancreatic trypsin inhibitor was simulated for 500 ns. **The 500 ns aMD simulation sampled the same amount of conformational space as an unbiased 1-millisecond MD simulation.** [8].

Our model system for this section is the D-Glucose/D-Galactose binding protein (GGBP) which exhibits a "Venus Flytrap" architecture present in several other proteins like the Lac repressor, G-protein-coupled receptors, ion channels, and GABA receptors. GGBP, as several other Perisplasmic binding proteins (PBP) can exist in open (ligand-free) and closed (ligand-bound) states. Using the aMD and Generalized Born implicit solvent (GBIS) methods with GPU acceleration, you will perform in this section long-timescale simulations of D-Glucose/D-Galactose binding protein in order to sample the conformational change of its "Venus Flytrap" hinge bending motion.

4.1. "Accelerated" Molecular Dynamics:

Accelerated molecular dynamics (aMD) [4] is an enhanced-sampling method that *accelerates* the sampling of conformational space by effectively reducing the height of energy barriers that separate different states of a system. The method modifies the potential energy landscape by applying a boost potential to raise energy wells below a certain threshold level, while leaving those above this level unaffected. As a result, barriers separating adjacent energy basins are effectively reduced, allowing the system to sample conformational space that would otherwise be inaccessible in a classical MD simulation, or require extensively long simulations at least an order of magnitude longer than the corresponding aMD simulations. The threshold level energy, referred to as E, will be an important input parameter for your aMD simulations.

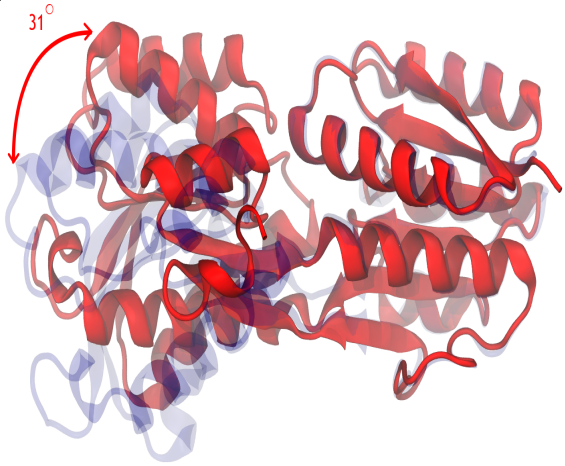


Figure 4: D-Glucose/D-Galactose binding protein (GGBP) mediates chemotaxis toward and active transport of glucose and galactose in some bacterial species [1]. The apo state (blue) shows a 31° angle difference between the α and β domains of GGBP when compared with the holo state (red).

4.1.1. Theoretical background

In Hamelberg and McCammon's aMD method [4], a system's potential energy is monitored until it falls below a threshold energy, E. A boost potential is then added, such that the modified potential, $V'(\mathbf{r})$, is related to the original potential, $V(\mathbf{r})$, via

$$V'(\mathbf{r}) = V(\mathbf{r}) + \Delta V(\mathbf{r}), \tag{1}$$

where $\Delta V(\mathbf{r})$ is the boost potential,

$$\Delta V(\mathbf{r}) = \begin{cases} 0 & V(\mathbf{r}) \geq E \\ \frac{(E-V(\mathbf{r}))^2}{\alpha+E-V(\mathbf{r})} & V(\mathbf{r}) < E. \end{cases} \tag{2}$$

As shown in the following figure, the portion of the potential surface below the threshold energy E is elevated by the boost potential. The acceleration factor, α, modulates the shape of the resulting elevated potential, permitting smooth continuous splining-together of the new elevated potential and the old potential. This approach prevents discontinuities at points where the potentials are joined. Such discontinuities lead to improper, even imposssible, energies and dynamics. Additionally, the α parameter modulates the *flexibility* of the boosted potential. By splining the potentials and modulating the curvature of the boosted potential, discontinuities are effectively eliminated and a smooth differentiable potential energy surface containing the boosted potential is generated. Note that α cannot be set to zero, otherwise the derivative of the modified potential is discontinuous.

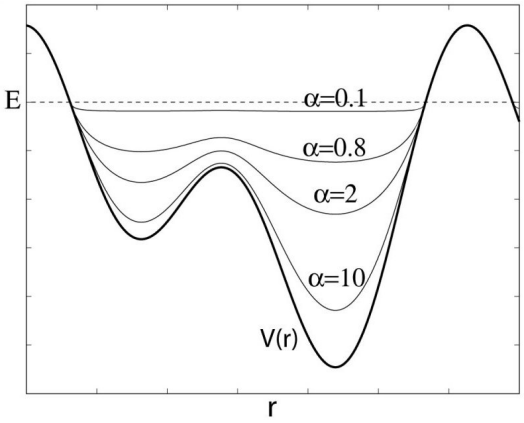


Figure 5: Schematics of the aMD method. When the original potential (thick line) falls below a threshold energy E (dashed line), a boost potential is added. The modified energy profiles (thin lines) have smaller barriers separating adjacent energy basins. Two parameters, E and α, control the portion of the affected potential landscape and the shape of the modified potential, respectively.

From an aMD simulation, the ensemble average, $\langle A \rangle$, of an observable, $A(\mathbf{r})$, can be calculated using the following reweighting procedure:

$$\langle A \rangle = \frac{\langle A(\mathbf{r}) \exp(\beta \Delta V(\mathbf{r})) \rangle^*}{\langle \exp(\beta \Delta V(\mathbf{r})) \rangle^*}, \tag{3}$$

in which β=$1/k_B T$, and $\langle \cdots \rangle$ and $\langle ... \rangle^*$ represent the ensemble average in the original and the aMD ensembles, respectively.

Currently, aMD can be applied in three modes in NAMD: dihedral boost (`accelMDdihe`), total energy boost, and dual boost (`accelMDdual`) modes [12]. In *dihedral boost mode* (`accelMDdihe`) the boost energy is applied only to the dihedral potentials. In *total energy boost mode*, the default mode in NAMD, the boost potential is applied to the total potential of the simulation. In the *dual boost mode* (`accelMDdual`) [4] two independent boost energies are applied, one to the dihedral potential and a second to the potential resulting from the difference of the total potential and the dihedral potential, ie ($V_T - V_{dihe}$).

Most long-timescale motions of proteins and large biomolecules correspond to large motions, and these in-turn are often thought linked to the slower vibrational modes of the system. Of the bonded potentials in MD forcefields that correspond to internal degrees-of-freedom of the system (bonds, angles, dihedrals and impropers), dihedrals are typically the lowest energy and slowest moving of the internal degrees-of-freedom. By accelerating, through aMD, the dihedral potentials of a system, the key internal degrees-of-freedom responsible for long-timescale motions are accelerated to produce these large, slow motions in a shorter simulation timescale. With such an understanding in mind, the *dihedral boost mode* is the most frequently used mode of aMD in NAMD.

However, boosting only the dihedrals in protein systems that also exhibit large diffusive motions of a domain may not be sufficient to accelerate that protein's motions. To overcome this challenge, the *dual boost mode* may be used in which the internal dihedrals are boosted as well as the diffusive degrees of freedom of the system, but with two seperate boost potentials, to accelerate large, diffusive, long-timescale motions of protein domains.

Citing accelerated MD in NAMD. Please include the following reference in publications that include usage of NAMD's GPU and aMD implementations:

- Accelerating Molecular Modeling Applications with Graphics Processors, John E. Stone, James C. Phillips, Peter L. Freddolino, David J. Hardy, Leonardo Trabuco and Klaus Schulten. *J. Comp. Chem.*, 28:2618-2640, 2007.,
- Accelerated Molecular Dynamics: A Promising and Efficient Simulation Method for Biomolecules, D. Hamelberg, J. Mongan, and J. A. McCammon. *J. Chem. Phys.*, 120:11919-11929, 2004.
- Implementation of Accelerated Molecular Dynamics in NAMD, Y. Wang, C. Harrison, K. Schulten, and J. A. McCammon, *Comp. Sci. Discov.*, 4:015002, 2011.

4.1.2. Compute the aMD parameters from cMD simulations.

Here you will perform a dihedral boost aMD simulation and need to obtain the following input parameters: the dihedral threshold energy (`accelMDE`) and dihedral alpha value (`accelMDalpha`). To determine these values for your aMD simulation, you must first perform a short classical MD simulation of your system. Typically 500 ps to a few ns are sufficient.

After completion of the classical MD simulation, calculate the mean of the dihedral energies. This mean energy value is needed to calculate the aMD input parameters. Determining the mean dihedral energy can be done by hand, for example by extracting the values into a plain text file and evaluating in xmgrace or your favorite spreadsheet software, or by using the `NAMD Plot` plugin in VMD in conjunction with xmgrace. In our example, the average dihedral energy from a classical MD simulation for GGBP was found to be 871 kcal mol^{-1}. Now calculate the mean dihedral energy for your system.

1 Navigate to `gpu-tutorial/gpu-tutorial_data/3-aMD/`, where the necessary files for the simulation are stored.

2 Start the simulation by entering.

```
namd2 +p8 +idlepoll 2fw0.namd > 2fw0_autopsf.log
```

3 At the end of the simulation, start a VMD session and open the TkConsole. Ensure that you are in the directory stated above.

4 Calculate the average energies. Open Extensions -> Analysis -> NAMD Plot. Click on File -> Select NAMD Log File, then select the logfile from your simulation. Click on DIHED. Click on File -> Plot Selected Data. On the plot window, pull down the 'File' menu and export the file to a format that you can work with, such as Xmgrace. You can find example datasets in the `.txt` files in the `examples` directory. Open the exported format using its corresponding software and calculate the mean of the dataset.

5 Calculate the dihedral E and α parameter input values by plugging the mean dihedral energy values into the below equations [3, 2]. These equations provide the values to use for `accelMDE` and `accelMDalpha`:

$$V_{dihe} = 871\,\text{kcal mol}^{-1} + (4\,\text{kcal mol}^{-1}\,\text{residue}^{-1} \times 306\,\text{solute residues}) = 2095\,\text{kcal mol}^{-1}, \tag{4}$$

$$\alpha_{dih} = \frac{1}{5} \times \left(4\,\text{kcal mol}^{-1}\,\text{residue}^{-1} \times 306\,\text{solute residues}\right) = 244.8\,\text{kcal mol}^{-1}. \tag{5}$$

4.2. Using aMD & GPUs for long-timescale molecular dynamics.

1 In this exercise you will perform an aMD simulation with GPUs. Using the parameters you previously calculated, add a new section to your NAMD configuration containing the below aMD keywords with your corresponding values. Our example values are shown below for comparison.

You can find an example of the modified configuration file at `examples/2fw0_aMD.namd`.

```
accelMD on
accelMDdihe on
accelMDE 2095
accelMDalpha 244.8
accelMDOutFreq 1000
```

2 Generalized Born implicit solvent (GBIS) is a fast but approximate method for calculating molecular electrostatics in solvent as described by the Poisson Boltzmann equation which models water as a dielectric continuum. GBIS enables the simulation of atomic structures without including explicit solvent water. In aMD simulations involving large motions, eliminating the water can sometimes play a critical role in accelerating those large motions as the resistance and viscous drag of explicit water is absent when using the GBIS model.

In addition, the elimination of explicit solvent actually accelerates simulations by eliminating the cost of calculating bulk water; however this speedup is somewhat hampered by the

increased computational complexity of the implicit solvent electrostatic calculation and a longer interaction cutoff.

We will set the ion concentration to 300mM using the `ionConcentration` keyword.

In our example, we turn on GBIS and set the salt concentration to 300mM with the following commands:

```
# GBIS Parameters
GBIS on
ionConcentration 0.3
```

3 When using GBIS, you should not use `PME` because it is not compatible with GBIS. Periodic boundary conditions are supported but are optional. You will also need to increase the `cutoff`; 16-18 Å is a good starting value but some systems may vary.

```
# Nonbonded Parameters
switching on
switchdist 15
cutoff 16
pairlistdist 17.5
Pme off
```

4 Select an `exclude` value appropriate for the forcefield you will be using; in this case, CHARMM. When using GBIS, interactions are not excluded regardless of the type of force field used, thus any value for `exclude` will work without affecting GBIS. You should choose `exclude` values appropriate for the force field you will be using. If you are unsure of the value to use, examine the input config for your earlier classical MD simulation. Notice that the `exclude` values will therefore be the same both with and without explicit solvent.

5 Select multiple timestepping values compatible with GBIS. When using GBIS, multiple timestepping behaves as follows: the dEdr force is calculated every `nonbondedFreq` steps (as with explicit solvent, 2 is a reasonable frequency) and the dEda force is calculated every `fullElectFrequency` steps (because dEda varies more slowly than dEdr, 4 is a reasonable frequency).

```
# Multistep Parameters
timestep 1
nonbondedFreq 2
fullElectFrequency 4
```

6 There are many ways to tune NAMD to obtain the best possible speeds for a particular system running on specific hardware. One of the easiest methods is making sure the number of patches (sections of your system split by NAMD which can be determined by the line ”PATCH GRID IS nx BY ny BY nz” in NAMD output) are equal to or greater than the number of processors. If this is not the case, you can force NAMD to almost double the number of patches in each dimension by putting the parameters shown below in your configuration file. This has doubled performance for this system on several test machines. You can explore the effects of turning this on or off in different combinations of dimensions to see if you can obtain improved performance on your own hardware. For further information about performance tuning, please see the NAMD wiki.

```
twoAwayX yes
twoAwayY yes
twoAwayz yes
```

7 You are now ready to run your simulation using the aMD method on GPUs. Please proceed and begin your simulation. Your simulation will run between 12 and 24 hours depending on your GPU hardware. You can find an example trajectory of this run at `examples/ggbp-aMD.dcd`.

8 While your aMD simulation is running, prepare a second, exact copy of the simulation in a new directory. This simulation will be identical to the aMD simulation in every respect except that you will remove from the config file all aMD commands (i.e., any command prefixed with `accel`). Once the aMD simulation that you started in the previous step is complete, run this new classical version of the system. Possessing all the same settings as the aMD simulation, this classical simulation will be used to evaluate differences in sampling between simulations that do and do not use aMD. Examples of the configuration file and output trajectory have been included in the `examples` directory, with filename `2fw0_cMD`.

5. GPU Augmented Analysis

5.0.1. Analysis of GGBP trajectories

After the GPU and aMD simulations are finished something you will notice is that the protein diffuses in the simulation box. Hence, we first need to remove the rotational and translational degrees of freedom of the protein by aligning the entire trajectories to the first frame.

1 Load both cMD and aMD simulations in VMD.

2 In VMD go to Extensions->Analysis->RMSD Trajectory Tool, a new windw opens up. For the alignment we are going to use residues from 1 to 109. Thus, in the text box where it says `protein`, add `resid 1 to 109` then click on *ALIGN*.

3 Use the *QuickSurf* representation to look at the trajectories. Color the protein by residue type. Can you identify the binding site?

4 Now that your trajectory is aligned, make sure the text box contains the entire `protein` as selection. Check the *Plot* box and click on *RMSD*. You will immediatly notice a difference between the RMSD curves for the cMD simulation and aMD simulation.

Analysis of protein dynamics in VMD. VMD contains a powerful interface to study the modes of a protein using ProDy (http://www.csb.pitt.edu/ProDy/plugins/index.html). You can follow the instructions on the website to install ProDy. Once you have it installed and running, in VMD go to Extensions->Analysis->Normal Mode Wizard. Then select the ProDy interface. Try different calculations, ANM, ENM and PCA. Are the modes obtained from the cMD and aMD different? Look at different modes and animate them to get a better insight of the motions of the protein.

5.1. Calculating g(r) using GPUs.

In VMD, GPUs have extended not only visualization capabilities, but also boosted the efficiency of calculations for analysis of trajectories, such as that of the pair correlation function, $g(r)$. In this section, you will perform $g(r)$ calculations using a pre-generated MscS trajectory.

1 In VMD, open the TkConsole and navigate to `gpu-tutorial/gpu-tutorial_data/4-analysis/`.

2 Due to time constraints, you are not expected to have generated a long enough trajectory to yield meaningful data for analysis. Hence, you have been provided with a pre-generated long trajectory for this section. The following optional procedure describes how this pre-generated trajectory was built. If you have had enough time to run a long simulation, then you are encouraged to go through the following optional steps.

Downsizing your DCDs (Optional). In some cases, you may have generated very large trajectory files. Large trajectories slow down loading times, depending on your hardware, and inconvenience transfer of files to collaborators. To get around this problem, you can extract from your original file the trajectories of just the atoms you wish to study, using CatDCD. CatDCD is a procedure in VMD that manipulates `.dcd` files, appending them to one another or extracting from them trajectories of selected atoms into a separate file. In the current exercise, we will be calculating $g(r)$ for certain MscS residues and nearby ions in solution, hence you will extract the trajectories of just these atoms into a separate file. The following steps, labeled **Optional**, will guide you through the process.

3 (Optional)

Copy the MscS PSF, PDB and trajectory files from section 1:

```
cp ../1-largeSims/mscs.psf .
cp ../1-largeSims/mscs.pdb .
cp ../1-largeSims/equil_gpu.dcd .
```

4 (Optional) Load the system onto VMD by typing in the TkConsole:

```
mol load psf mscs.psf pdb mscs.pdb
```

5 (Optional) You will now create a file listing the indices of atoms whose trajectories you will be extracting. Source the provided script `Make_ind.tcl` or enter the following lines into the TkConsole:

```
set seltext1 "resname GLU and not segname PA PB PC PD PE PF PG and name CA"
set seltext2 "name POT"
set seltext3 "segname PA PB PC PD PE PF PG and resid 158"
set sel [atomselect top "($seltext1) or ($seltext2) or ($seltext3)"]
set dat [$sel get index]
set filename "atoms.ind"
set fileID [open $filename "w"]
puts -nonewline $fileID $dat
close $fileID
$sel writepsf mscs_cat.psf
mol delete top
```

6 **(Optional)** Call CatDCD, using as input the file you have just created:
`catdcd -o equil_cat.dcd -i atoms.ind equil_gpu.dcd`

7 **(Optional)** Compare the filesize of the new trajectory file you have created with the original one. How much of a difference did CatDCD make?

8 Load the MscS trajectory using the command:

```
mol load psf mscs_cat.psf dcd equil_cat.dcd
```

9 On the menubar, click on Extensions, then under Analysis, select Radial Pair Distribution Function g(r). A user interface should pop up.

10 Make sure that the field 'Use Molecule:' is set to `mscs_cat.psf`.

11 Notice that you can set the beginning and end frames of the time period that you wish to calculate g(r) over. The defaults are respectively the first and final frames of the trajectory you have loaded.

12 You can also alter the histogram parameters to suit your own purposes. For this exercise, just leave them as the defaults.

13 Check 'Use PBC', 'Display g(r)' and 'Use GPU code'.

14 Now, we will calculate $g(r)$ for the positively-charged arginine residue 158, or more specifically, the alpha carbon atom of the residue, and positively-charged potassium ions. In 'Selection 1', enter `name CA and segname PA PB PC PD PE PF PG and resid 158` and in 'Selection 2', enter `name POT`.

15 Click on 'Compute g(r)'. The resulting $g(r)$ plot will be loaded.

16 Alternatively, if you want to run this analysis from the command line, type the following into TkConsole:

```
set sel1 [atomselect top "name CA and segname PA PB PC PD PE PF PG and resid 158"]
set sel2 [atomselect top "name POT"]
measure rdf $sel1 $sel2 usepbc true
```

17 Perform the same calculation for negatively-charged glutamate ions instead of potassium by entering in 'Selection 2', `name CA and protein and not segname PA PB PC PD PE PF PG`, before clicking on 'Compute g(r)'.

18 Alternatively, if you want to run this analysis from the command line, type the following into TkConsole:

```
set sel1 [atomselect top "name CA and segname PA PB PC PD PE PF PG and resid 158"]
set sel2 [atomselect top "name CA and protein and not segname PA PB PC PD PE PF PG"]
measure rdf $sel1 $sel2 usepbc true
```

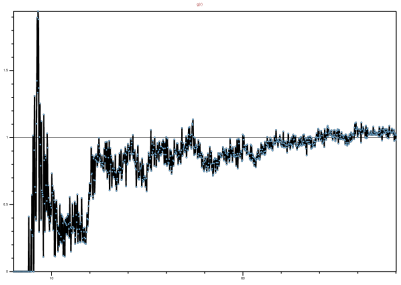


Figure 6: Pair distribution function of residue 158 and potassium ions. Note the almost uniform distribution across all lengths

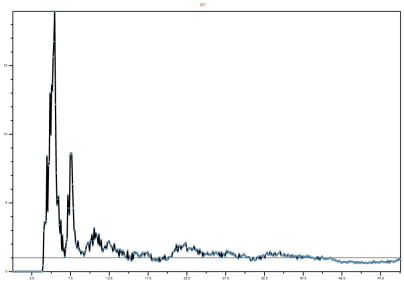


Figure 7: Pair distribution function of residue 158 and the alpha carbon of glutamate ions. In contrast to the case of potassium ions, notice the spikes at 5 Å and 7.5Å. Do you think this result makes sense?

References

[1] M. Jack Borrok, Laura L. Kiessling, and Katrina T. Forest. Conformational changes of glucose/galactose-binding protein illuminated by open, unliganded, and ultra-high-resolution ligand-bound structures. *Protein Science*, 16(6):1032–1041, 2007.

[2] Csar Augusto F. de Oliveira, Barry J. Grant, Michelle Zhou, and J. Andrew McCammon. Large-scale conformational changes of ¡italic¿trypanosoma cruzi¡/italic¿ proline racemase predicted by accelerated molecular dynamics simulation. *PLoS Comput Biol*, 7(10):e1002178, 10 2011.

[3] Barry J. Grant, Alemayehu A. Gorfe, and J. Andrew McCammon. Ras conformational switching: Simulating nucleotide-dependent conformational transitions with accelerated molecular dynamics. *PLoS Comput Biol*, 5(3):e1000325, 03 2009.

[4] Donald Hamelberg, César Augusto F. de Oliveira, and J. Andrew McCammon. Sampling of slow diffusive conformational transitions with accelerated molecular dynamics. *The Journal of Chemical Physics*, 127(15):155102, 2007.

[5] Aaftab Munshi. OpenCL Specification Version 1.0, December 2008. http://www.khronos.org/registry/cl/.

[6] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *ACM Queue*, 6(2):40–53, 2008.

[7] James C. Phillips, John E. Stone, and Klaus Schulten. Adapting a message-driven parallel application to GPU-accelerated clusters. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Piscataway, NJ, USA, 2008. IEEE Press.

[8] Levi C.T. Pierce, Romelia Salomon-Ferrer, Cesar Augusto F. de Oliveira, J. Andrew McCammon, and Ross C. Walker. Routine access to millisecond time scale events with accelerated molecular dynamics. *Journal of Chemical Theory and Computation*, 8(9):2997–3002, 2012.

[9] John E. Stone, David J. Hardy, Ivan S. Ufimtsev, and Klaus Schulten. GPU-accelerated molecular modeling coming of age. *J. Mol. Graph. Model.*, 29:116–125, 2010.

[10] John E. Stone, James C. Phillips, Peter L. Freddolino, David J. Hardy, Leonardo G. Trabuco, and Klaus Schulten. Accelerating molecular modeling applications with graphics processors. *J. Comp. Chem.*, 28:2618–2640, 2007.

[11] John E. Stone, Jan Saam, David J. Hardy, Kirby L. Vandivort, Wen-mei W. Hwu, and Klaus Schulten. High performance computation and interactive display of molecular orbitals on GPUs and multi-core CPUs. In *Proceedings of the 2nd Workshop on General-Purpose Processing on Graphics Processing Units, ACM International Conference Proceeding Series*, volume 383, pages 9–18, New York, NY, USA, 2009. ACM.

[12] Yi Wang, Chris B. Harrison, Klaus Schulten, and J. Andrew McCammon. Implementation of accelerated molecular dynamics in NAMD. *Comput. Sci. Discov.*, 4:015002, 2011. (11 pages).