MS 125
Computational Tools and Precision Medicine

Room: 206A 9:45 AM - 11:25 AM

9:45-10:05 Acceleration of Prediction of Chemical Shift Structures
Sunita Chandrasekaran and Juan Perilla, University of Delaware, U.S.
10:10-10:30 In Situ Data Analytics for Next Generation Molecular Dynamics Workflows
Michela Taufer, University of Tennessee, U.S.
10:35-10:55 Challenges for Analysis and Visualization of Atomic-detail Simulations of Minimal Cells
John E. Stone, University of Illinois at Urbana-Champaign, U.S.
11:00-11:20 Capabilities, Collaboration and Cancer: Co-design for Advanced Computing Solutions for Cancer Eric Stahlberg, National Cancer Institute, U.S.; *George Zaki*, Frederick National Lab for Cancer Research, U.S.

Acceleration of Prediction of Chemical Shift Structures

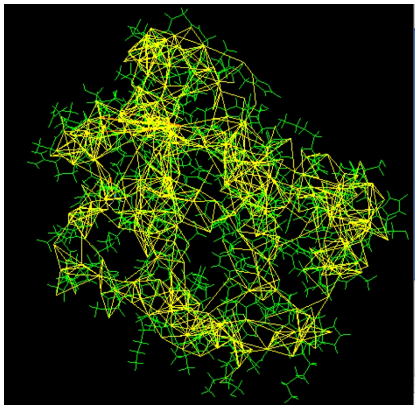
Sunita Chandrasekaran, Asst. Prof. Dept. of CIS, UDEL
Juan Perilla, Asst. Prof. Dept. of Chemistry, UDEL
schandra@udel.edu

Feb 26, SIAM CSE 2019

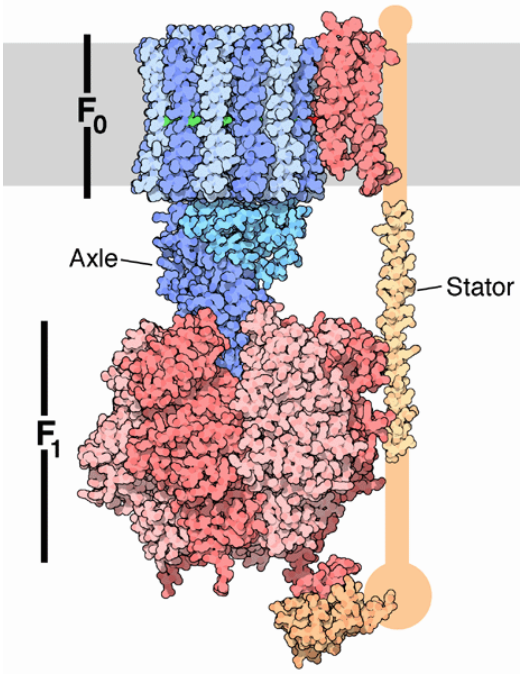
Structure is essential to function

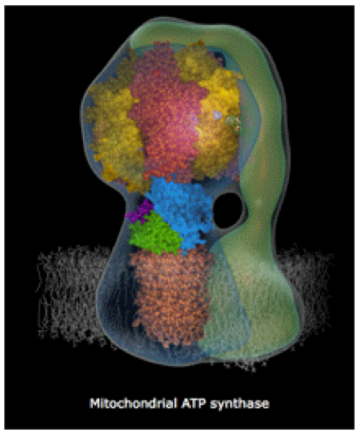
Determining a protein's native structure is the critical first step in understanding function

Tools of structure determination:
- X-Ray crystallography
- Electron microscopy
- **Nuclear Magnetic Resonance (NMR)**



PDBID 1vre





Proteins and their functions are dynamic

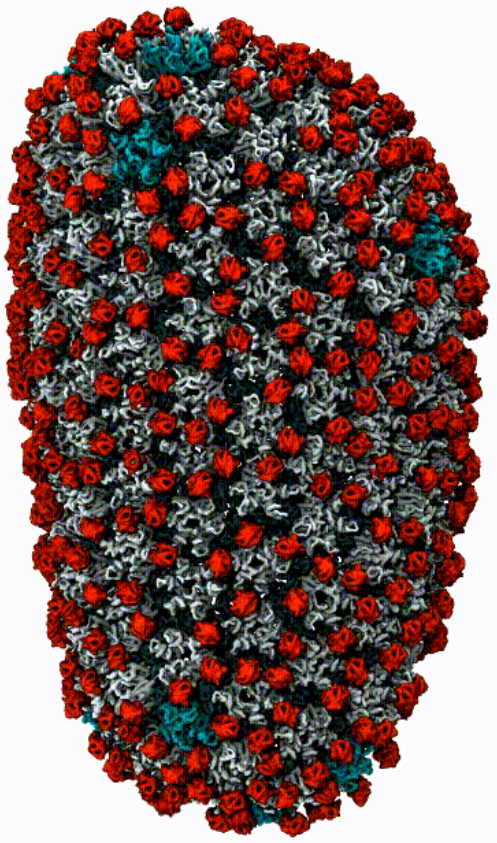
https://pdb101.rcsb org/motm/72
Medical Research Council: Mitochondrial Biology Unit

Project Motivation

- Nuclear Magnetic Resonance (NMR) is a vital tool in the biocomputational space
- Chemical shift gives insight into the physical structure of the protein
- Predicting chemical shift has important uses in scientific areas such as drug discovery

Our goal:

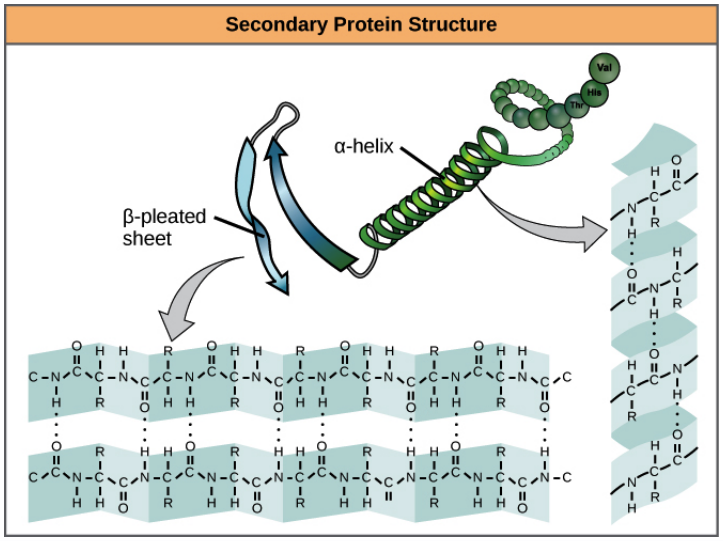
- To expedite the prediction of estimation of NMR chemical-shifts of large macromolecular complexes by manyfold
- To allow chemical shift predictions for larger scale structures



Proteins are biopolymers made of amino acid monomers

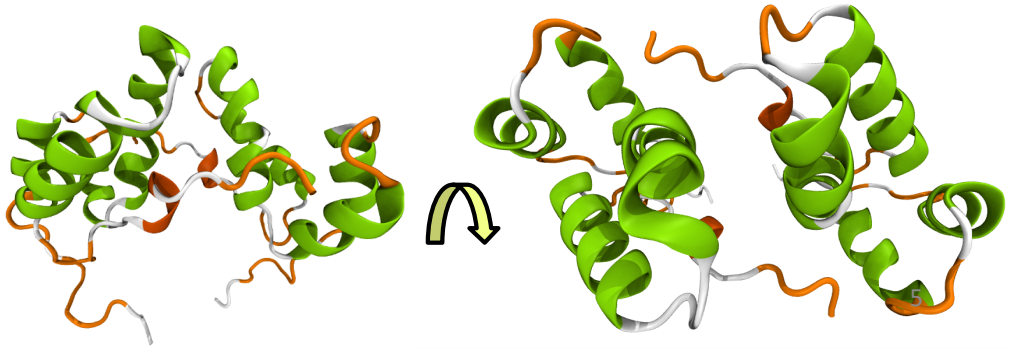
Primary structure: sequence of amino acids . . .

Sequence is organized into **secondary structure** by Hydrogen bonding



Tertiary structure is formed from grouping secondary structures:

Quaternary structure convolves organized tertiary structures:



Parts Per Million (PPM)_ONE

- Parametrize a new empirical knowledge-based chemical shift predictor of protein backbone atoms
- Accepts a single static 3D protein structure (PDB format) as input
- Emulates local protein dynamics
- Outputs chemical shift prediction with high accuracy

```
ATOM      1  N   MET A 144      16.219   4.268   2.223  1.00  0.00           N
ATOM      2  CA  MET A 144      14.894   4.097   2.883  1.00  0.00           C
ATOM      3  C   MET A 144      13.976   5.251   2.488  1.00  0.00           C
ATOM      4  O   MET A 144      13.839   6.226   3.225  1.00  0.00           O
ATOM      5  CB  MET A 144      15.082   4.077   4.402  1.00  0.00           C
ATOM      6  CG  MET A 144      15.859   2.822   4.805  1.00  0.00           C
ATOM      7  SD  MET A 144      16.042   2.778   6.605  1.00  0.00           S
ATOM      8  CE  MET A 144      16.943   1.212   6.703  1.00  0.00           C
ATOM      9  H1  MET A 144      16.141   4.979   1.468  1.00  0.00           H
ATOM     10  H2  MET A 144      16.523   3.361   1.816  1.00  0.00           H
ATOM     11  H3  MET A 144      16.917   4.586   2.924  1.00  0.00           H
ATOM     12  HA  MET A 144      14.453   3.164   2.565  1.00  0.00           H
ATOM     13  HB2 MET A 144      15.632   4.955   4.708  1.00  0.00           H
ATOM     14  HB3 MET A 144      14.116   4.070   4.885  1.00  0.00           H
ATOM     15  HG2 MET A 144      15.321   1.945   4.476  1.00  0.00           H
ATOM     16  HG3 MET A 144      16.835   2.840   4.344  1.00  0.00           H
ATOM     17  HE1 MET A 144      16.382   0.441   6.202  1.00  0.00           H
ATOM     18  HE2 MET A 144      17.078   0.941   7.741  1.00  0.00           H
ATOM     19  HE3 MET A 144      17.907   1.322   6.226  1.00  0.00           H
ATOM     20  N   TYR A 145      13.350   5.130   1.321  1.00  0.00           N
ATOM     21  CA  TYR A 145      12.448   6.172   0.838  1.00  0.00           C
ATOM     22  C   TYR A 145      11.480   6.592   1.940  1.00  0.00           C
ATOM     23  O   TYR A 145      11.464   7.751   2.353  1.00  0.00           O
ATOM     24  CB  TYR A 145      11.672   5.657  -0.383  1.00  0.00           C
```

PPM_One: a static protein structure based chemical shift predictor
Dawei Li, Rafael Brüschweiler, Journal of Biomolecular NMR. July 2015, Volume 62, Issue 3, pp 403–409

Semiempirical chemical shift prediction

Treats chemical shift as a sum of differentiable functions which depend on internal coordinates

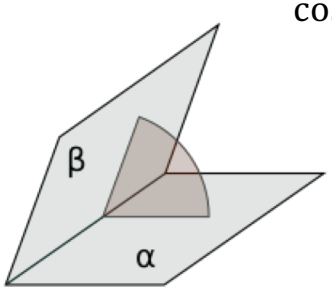
$$\delta_{CSpredicted} = \delta_{HBond} + \delta_{Dihedral} + \delta_{Ring\ Current} + \delta_{Magn\ Anisotropy} + \delta_{Electric}$$

Higher dimensional data (3D cartesian) maps to lower dimensional **internal coordinates**

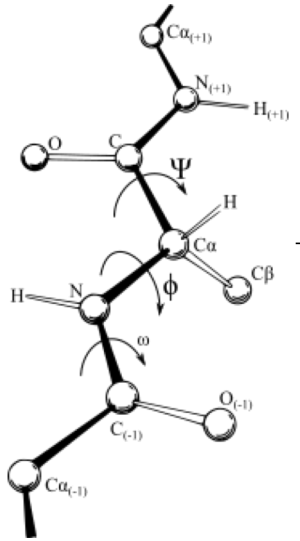
e.g., dihedral angle:

(α) $a_1x + b_1y + c_1z + d_1 = 0$

(β) $a_2x + b_2y + c_2z + d_2 = 0$



$$\cos\Psi = \frac{|\boldsymbol{n}_1 \cdot \boldsymbol{n}_2|}{|\boldsymbol{n}_1||\boldsymbol{n}_2|}$$



which is then passed to a CS function:

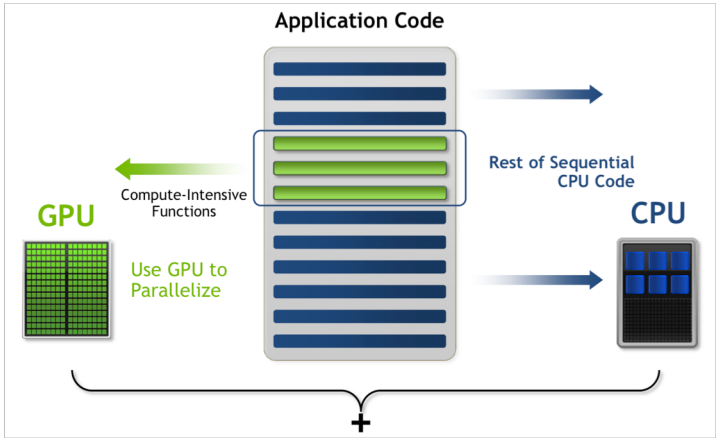
$$\delta_{\mathrm{Dihedral}} = p_1 + p_2\cos(\theta + p_3) + p_4\cos(\theta + p_5)$$

where *p* are fit parameters assigned on a per-residue basis

Using OpenACC

- OpenACC, directive based parallel programming model used to accelerate code on heterogenous systems
- Implemented by PGI, GCC, and Cray (until 2.0)
- PGI community editions are free (licensed) to use, latest version 18.10

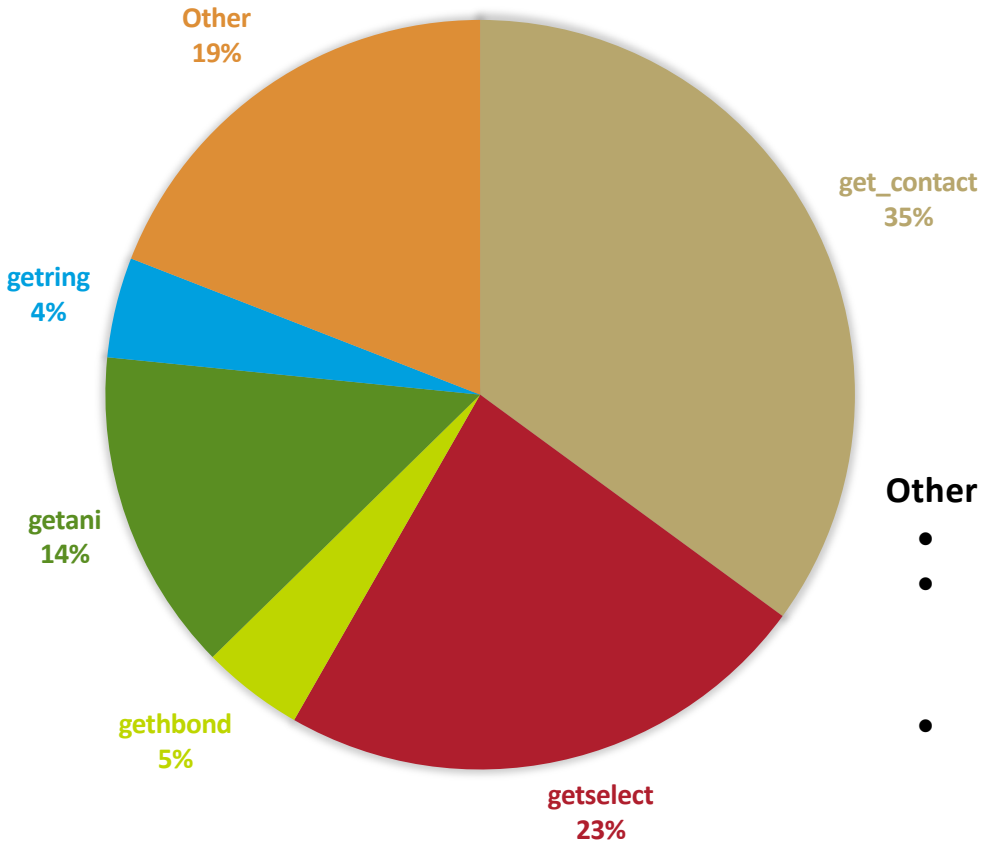
https://www.pgroup.com/products/community.htm



OpenACC
More Science, Less Programming

Serial Profile Visual

- Profiled code using PGPROF
  - Without any optimizations
- Gave a baseline snapshot of the code
  - Identified hotspots within the code
  - Identified functions that are potential bottlenecks
- Obtained large overview without needing to read thousands of lines of code

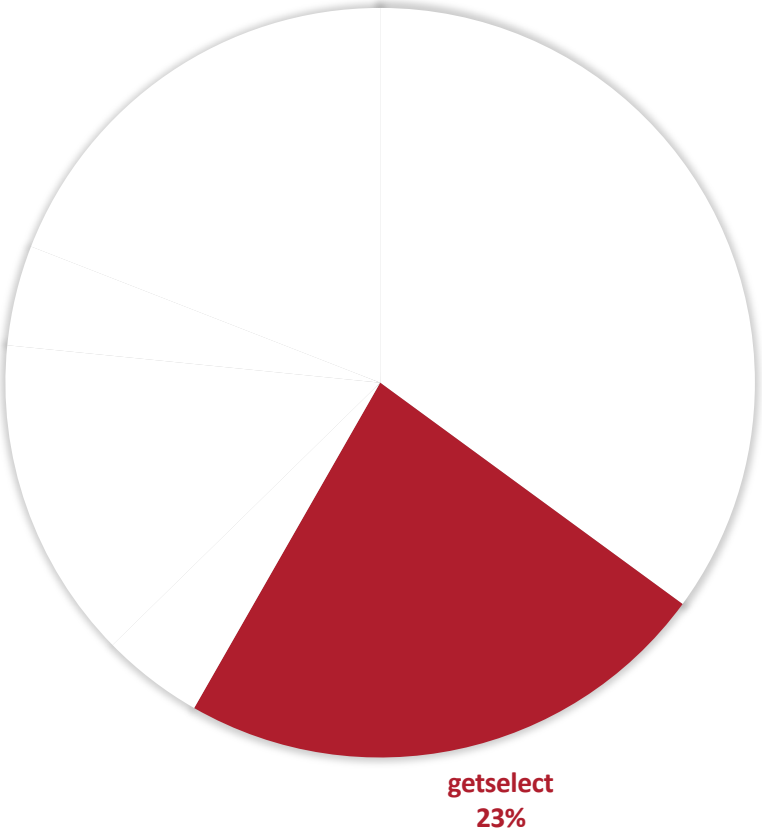


Other Contains:

- File I/O
- PDB Structure Initialization
- Data error correction

Optimization in steps

- Looking into optimizing the serial code prior to parallelizing it



Serial Optimization (getselect)

```
// Pseudocode for getselect function

for( ... )   // Large loop
{
    c2=pdb->getselect(":1-%@allheavy");
    traj->get_contact(c1,c2,&r
}
```

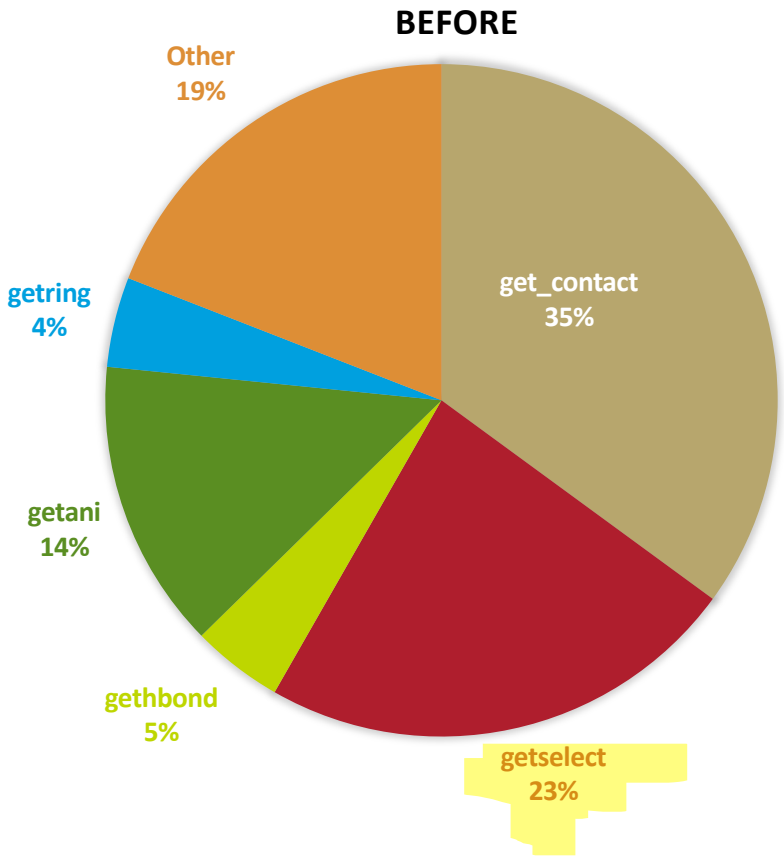
getselect originally accounted for **25%** of the codes runtime. After optimization, it takes less than **1%**.

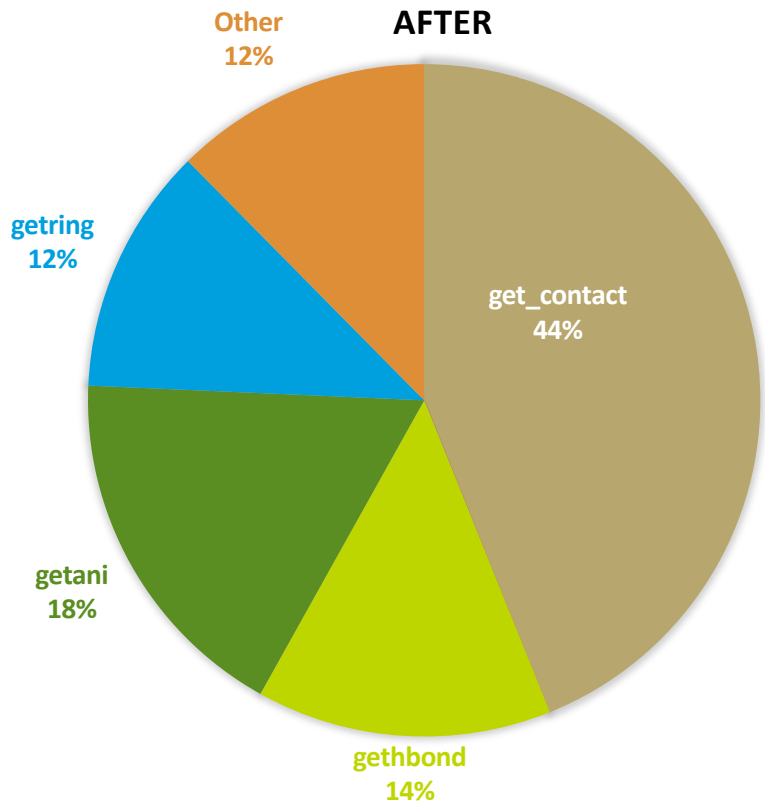
```
// Pseudocode for getselect function

c2=pdb->getselect(":1-%@allheavy");
for( ... )   // Large loop
{
    traj->get_contact(c1,c2,&result);
}
```

Serial Optimizations (other smaller optimizations)

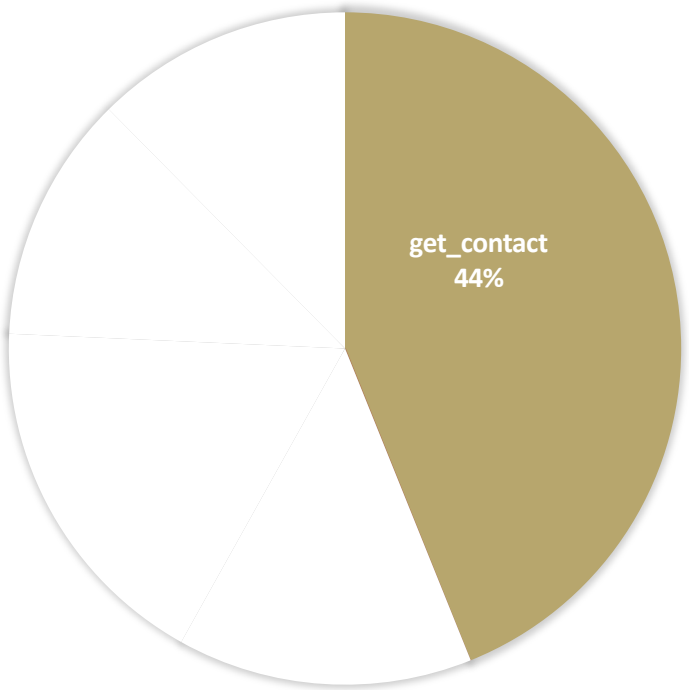
- Filtering Functions
  - Filter objects from a large list
  - Written to be C++ friendly, but was overall very inefficient
  - Runtime for filtering functions went from **5+ minutes** down to **1 second** in some cases
- Replacing C++ Vectors
  - C++ Standard Data Structure
  - Replace with basic arrays
  - No meaningful impact on performance (sequentially)

Serial Profile After Optimizations





Most compute intensive



Accelerating get_contact

- get_contact is called many times in the code
- The **“pos”** vector actually only contains 3 values; x, y, z coordinates
- The **“used”** vector contains all of the atoms in the structure
- GPU focused, we collapsed the outer loop
  - Now we compute 3 contacts simultaneously
- We also combined all calls to get_contact into one large function called **get_all_contacts**

Inside of the get_contact function

```
for(i
{
    }


}
```

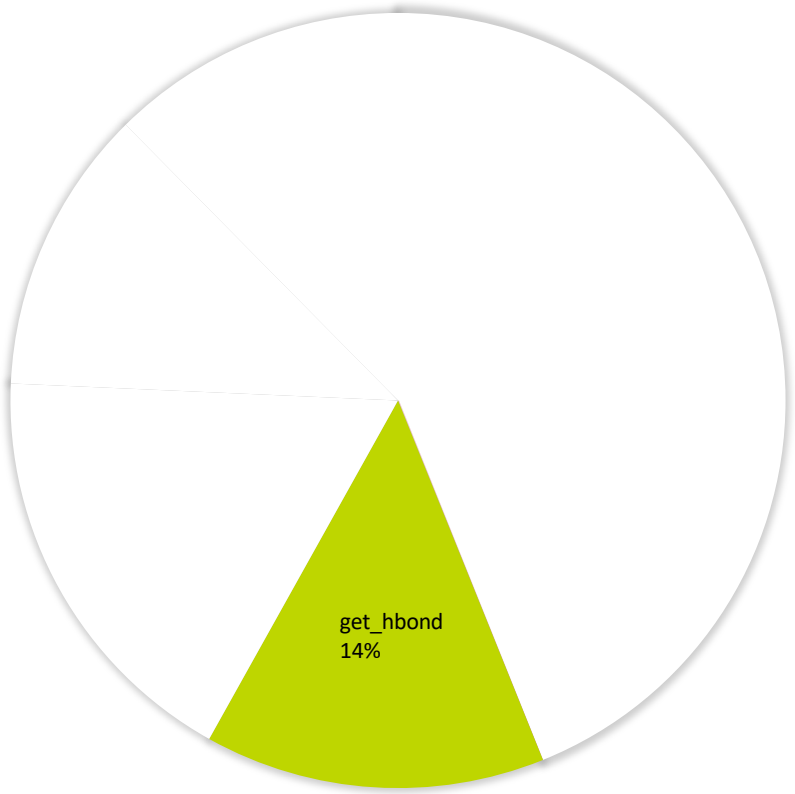
```
// For x,y,z coordinate
for(i=0;i<(int)pos.size();i++)
{
    ...
    // For every atom
    for(j=0;j<(int)used.size();j++)
    {
        // Calculate contact
        ...
    }
    result->push_back(contact);
}
```

Accelerating get_contact

```
#pragma acc parallel loop private(...) \
 present(..., results[0:results_size]) copyin(...)
for(i=1;i<index_size-1;i++)
{
    ...

    #pragma acc loop reduction(+:contact1, +:contact2, \
     +:contact3) private(...)
    for(j=0;j<c2_size;j++)
    {
        // Calculate contact1, contact2, contact3
    }
    ...
    results[((i-1)*3)+0]=contact1;
    results[((i-1)*3)+1]=contact2;
    results[((i-1)*3)+2]=contact3;
}
```

- Large outer-loop covers all individual get_contact calls
- Inner-loop still iterates over all atoms
- Now calculating 3 different contacts simultaneously
- Writing contacts to one large results array to be used later

Next most compute intensive



Acceleration of gethbond

```
#pragma acc parallel
{
#pragma acc loop gang
for(i=0;i<_hbond_size;i++)
{

    #pragma acc loop vector
    for(j=0;j<hbond_size;j++)
    {
        ...
        #pragma acc loop seq
        for(k=0;k<nframe;k++)
        {
            ...
        }
    }
}
} // end parallel region
```

Gang and vector directives allow us to implement multiple levels of loop parallelism.

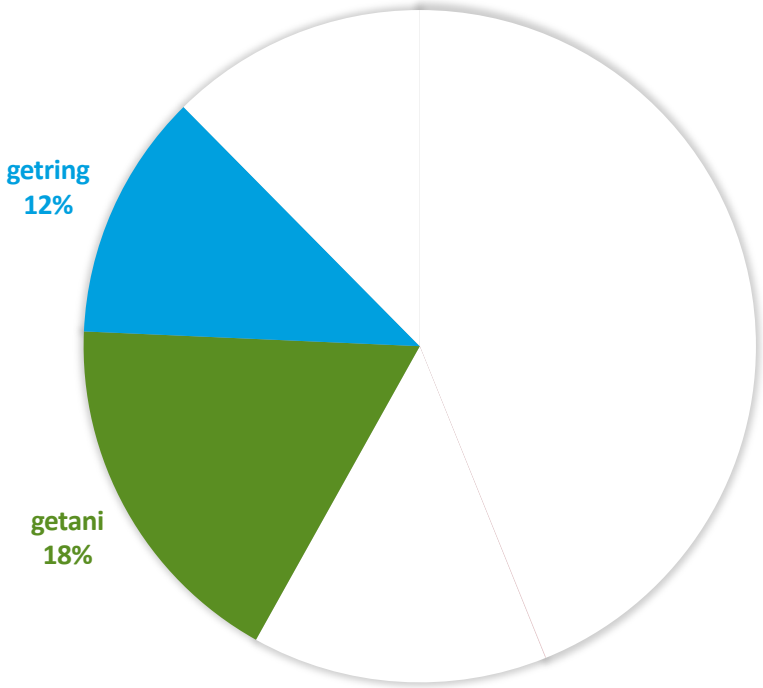
The innermost loop is typically very small, and would provide no benefit in parallelizing, so we mark it as “sequential”

Acceleration of gethbond

```
#pragma acc parallel
{
#pragma acc loop gang
for(i=0;i<_hbond_size;i++)
{

    #pragma acc loop vector
    for(j=0;j<hbond_size;j++)
    {
        ...
        #pragma acc loop seq
        for(k=0;k<nframe;k++)
        {
            ...
        }
    }
}
} // end parallel region
```

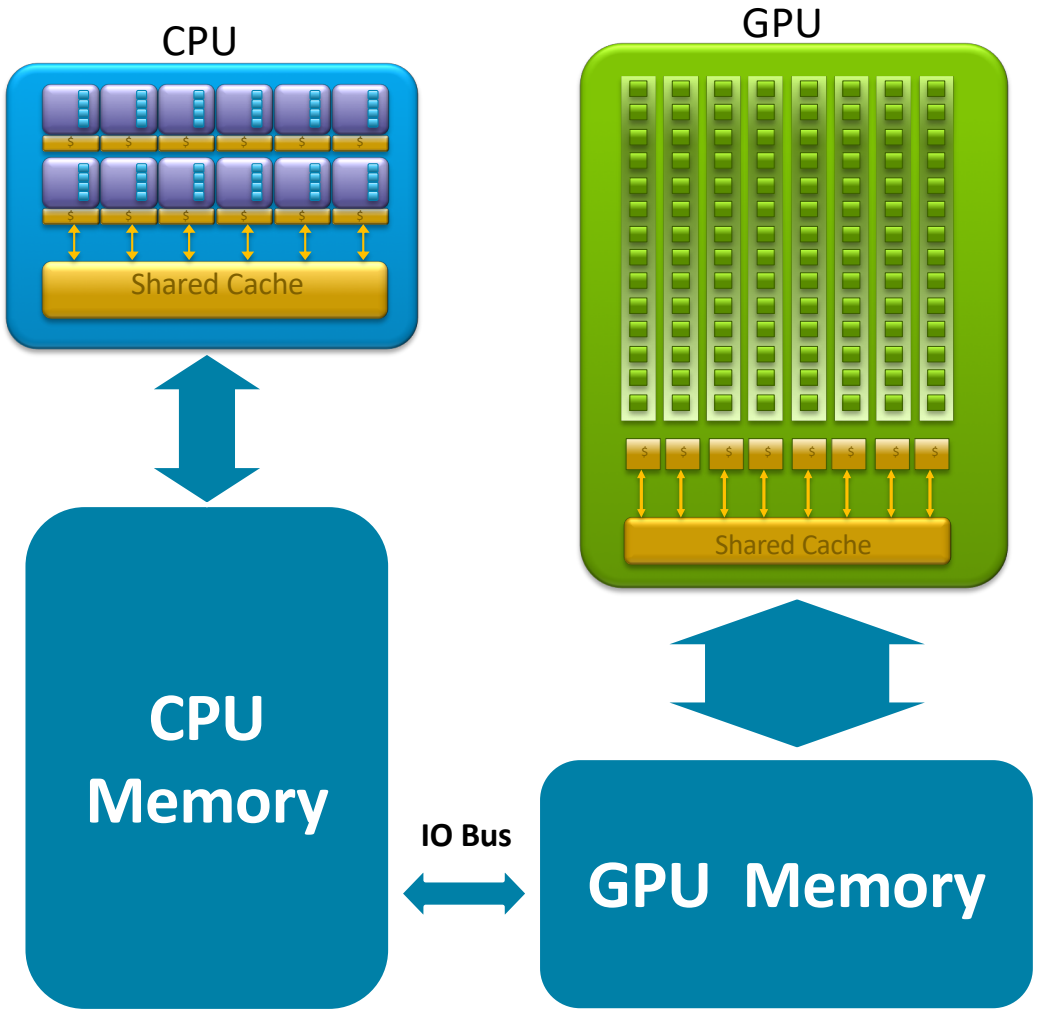
```
if(hbond[i].type==1){
    #pragma acc atomic update
    effect_arr[nid].n_length+=d;
    #pragma acc atomic update
    effect_arr[nid].n_phi+=phi;
    #pragma acc atomic update
    effect_arr[nid].n_psi+=psi
}
if(hbond[j].type==1){
    #pragma acc atomic update
    effect_arr[cid].c_lengh+=d;
    #pragma acc atomic update
    effect_arr[cid].c_phi+=phi;
    #pragma acc atomic update
    effect_arr[cid].c_psi+=psi;
}
```

And the next most…

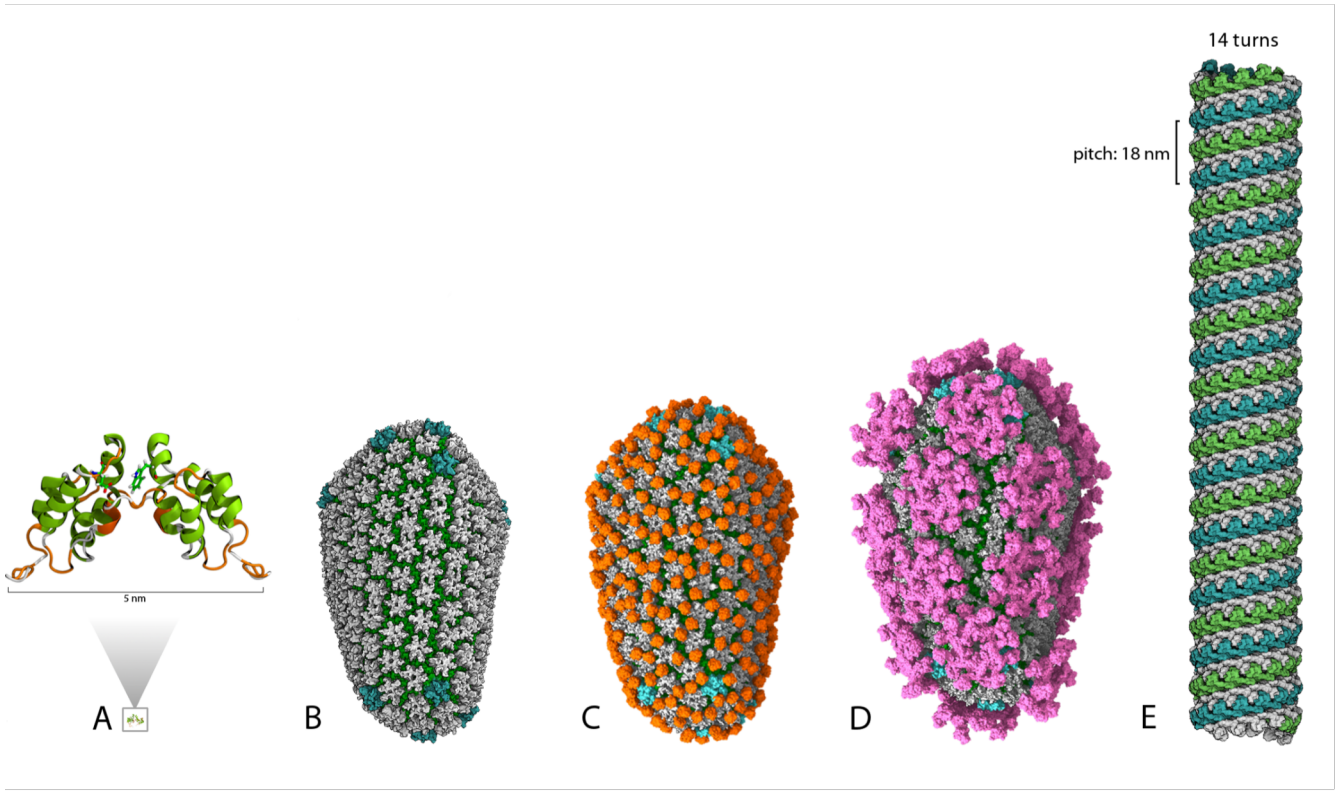


Data Movement

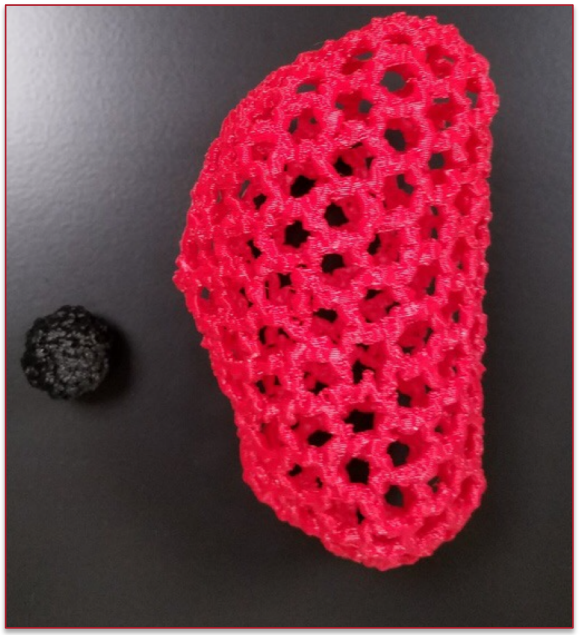
- Allocate memory on host first (main memory)
- Create copy of our data on the device (GPU memory)
- Ensure that the correct data is on the GPU when we need it
  - And vice versa

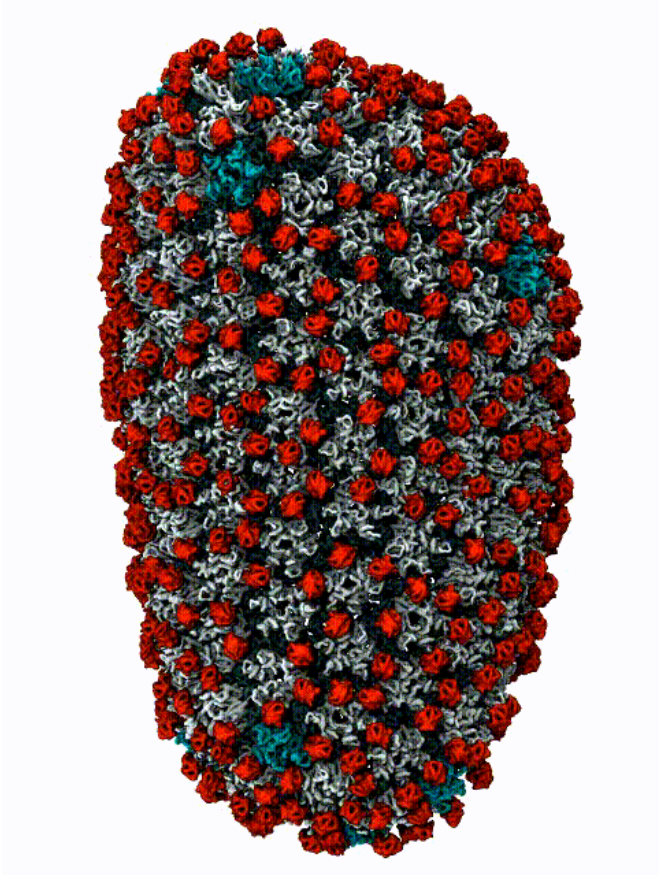


Experimental Datasets

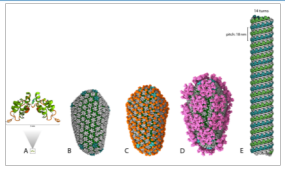


3D printed



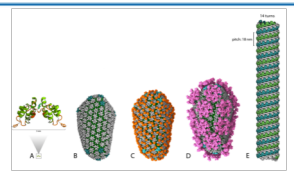


Results



Serial (Unoptimized)				
Serial (Optimized)				
Multicore (32 Xeon cores)				
NVIDIA PASCAL P100 GPU				
NVIDIA VOLTA V100 GPU				

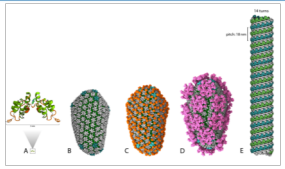
Experimental Setup



	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (11M) Atoms
Serial (Unoptimized)				
Serial (Optimized)				
Multicore (32 Xeon cores)				
NVIDIA PASCAL P100 GPU				
NVIDIA VOLTA V100 GPU				

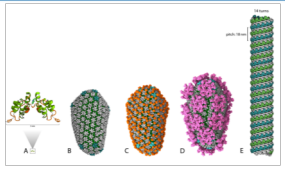
PGI 18.4, Community Edition

Results



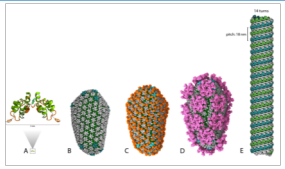
	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (11M) Atoms
Serial (Unoptimized)	167.11s	3547.07 (1 hour)	7 hours *approx.*	14 hours *approx.*
Serial (Optimized)				
Multicore (32 Xeon cores)				
NVIDIA PASCAL P100 GPU				
NVIDIA VOLTA V100 GPU				

Results



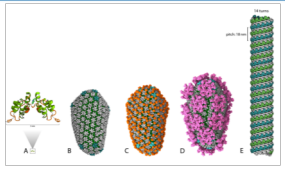
	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (11M) Atoms
Serial (Unoptimized)	167.11s	3547.07 (1 hour)	7 hours *approx.*	14 hours *approx.*
Serial (Optimized)	32s	2209.64s (37 min)	2939s (48 min)	9035s (2.5 hours)
Multicore (32 Xeon cores)				
NVIDIA PASCAL P100 GPU				
NVIDIA VOLTA V100 GPU				

Results



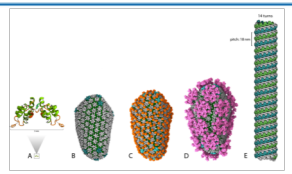
	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (11M) Atoms
Serial (Unoptimized)	167.11s	3547.07 (1 hour)	7 hours *approx.*	14 hours *approx.*
Serial (Optimized)	32s	2209.64s (37 min)	2939s (48 min)	9035s (2.5 hours)
Multicore (32 Xeon cores)	2.93s	109s	172s	427s
NVIDIA PASCAL P100 GPU				
NVIDIA VOLTA V100 GPU				

Results

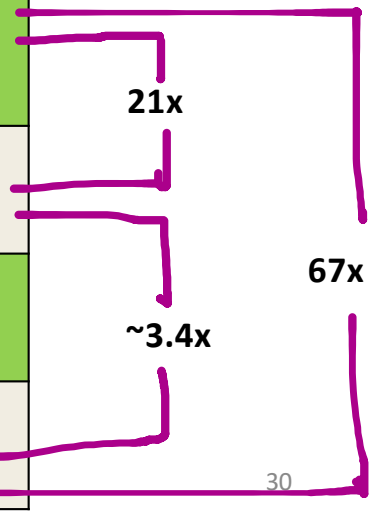


	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (11M) Atoms
Serial (Unoptimized)	167.11s	3547.07 (1 hour)	7 hours *approx.*	14 hours *approx.*
Serial (Optimized)	32s	2209.64s (37 min)	2939s (48 min)	9035s (2.5 hours)
Multicore (32 Xeon cores)	2.93s	109s	172s	427s
NVIDIA PASCAL P100 GPU	1.72s	36s	69s	170s
NVIDIA VOLTA V100 GPU	1.68s	29s	56s	

Results



	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (11M) Atoms
Serial (Unoptimized)	167.11s	3547.07 (1 hour)	7 hours *approx.*	14 hours *approx.*
Serial (Optimized)	32s	2209.64s (37 min)	2939s (48 min)	9035s (2.5 hours)
Multicore (32 Xeon cores)	2.93s	109s	172s	427s
NVIDIA PASCAL P100 GPU	1.72s	36s	69s	170s
NVIDIA VOLTA V100 GPU	1.68s	29s	56s	134s



Results Takeaway

- Of 134s on V100, 110s spent on data preprocessing; rewriting code could bring it ~13x on V100 GPU over 32-core CPU
- On V100 GPU, 67x compared to the optimized serial code
- On 32 E5-2698 dual socket Xeon cores, ~21x compared to the optimized serial code
- Compared to a fully-utilized 32 E5-2698 dual socket Xeon cores, V100 achieves ~3.4X
- Single source code maintained using OpenACC on both multicore as well as GPU

Scientific Impact

- TTBOOK first work on accelerated prediction of chemical shift
- Accelerated PPM_One is being used during an MD simulation to predict shifts at every timestep and validate the structure
- Following advances in imaging techniques such as cryo-electron microscopy (cryo-EM), empirical data for very large biological complexes/structures enables in silico study thereof, giving weight/significance to such studies and creating a need for software and tools that can handle the size and complexity of these structures.