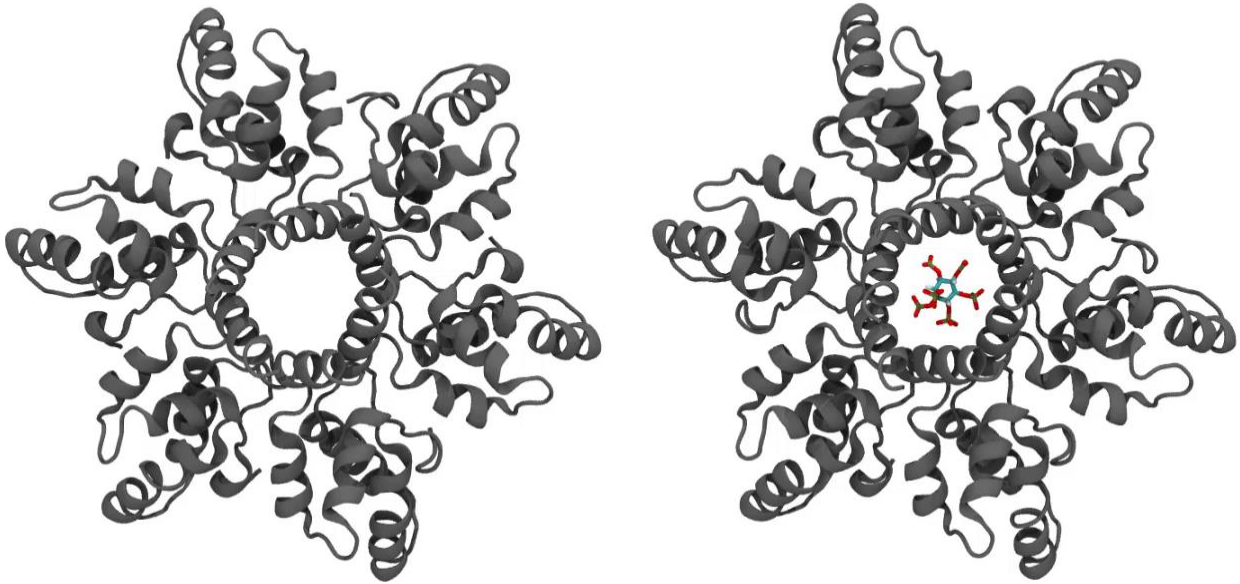
S9277 - OpenACC-Based GPU Acceleration of Chemical Shift Prediction

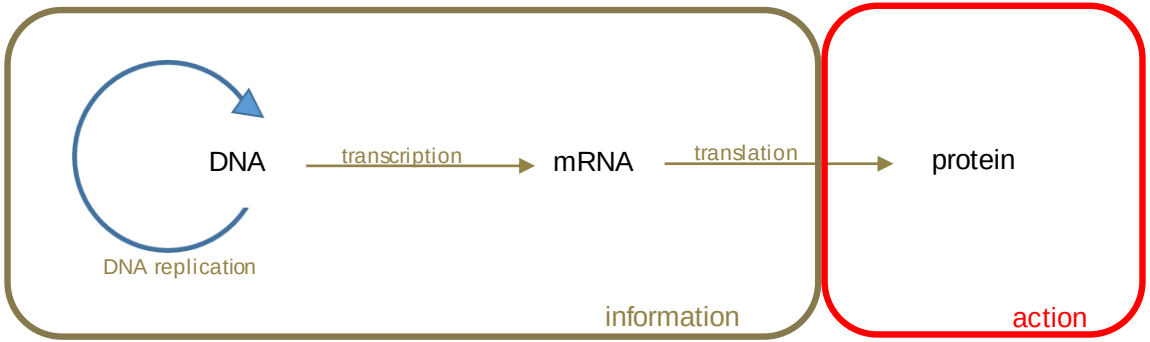
Eric Wright and Alex Bryer
Sunita Chandrasekaran and Juan Perilla
{efwright, abryer, schandra, jperilla} @udel.edu
Collaborative project from Depts of CIS and Chemistry
University of Delaware

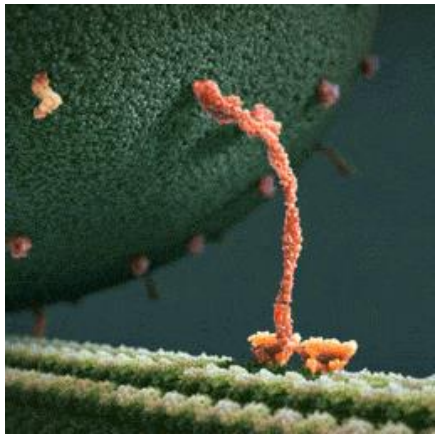
GTC March 19, 2019



Xu, et al. *Nature* (2018)

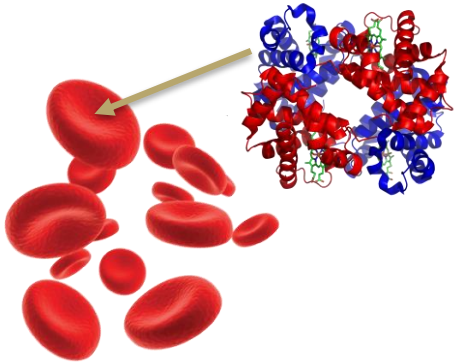
Proteins are central to biology, physiology and pathology



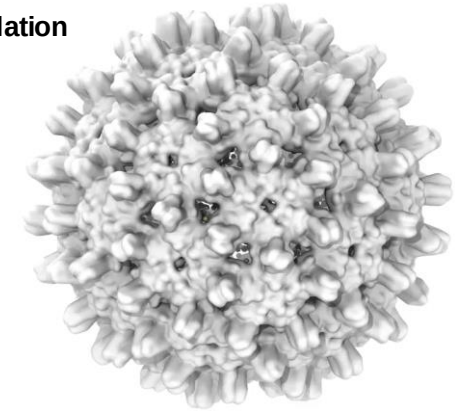


motor … and much more

transport



encapsulation



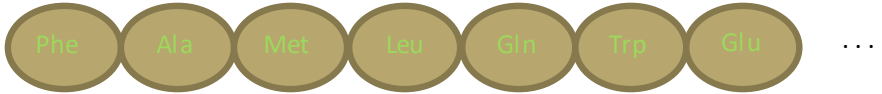
Only 20 unique amino acids...

Function arises from **structure**

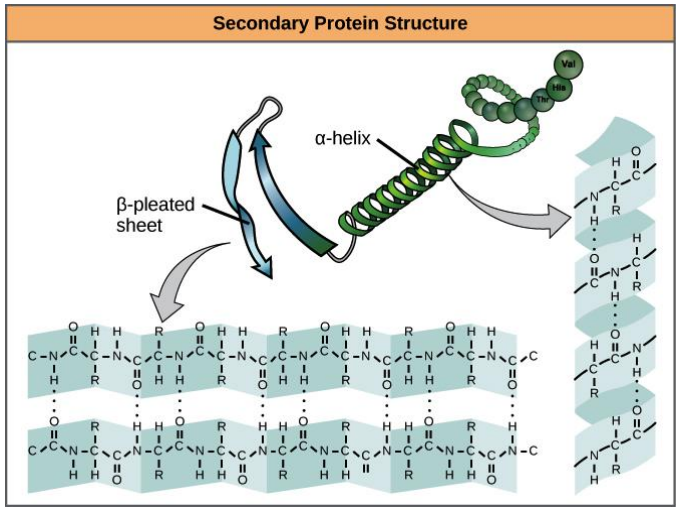
Hadden, et al. *eLife* (2018)

Hierarchy of protein structure

Primary structure: sequence of amino acids

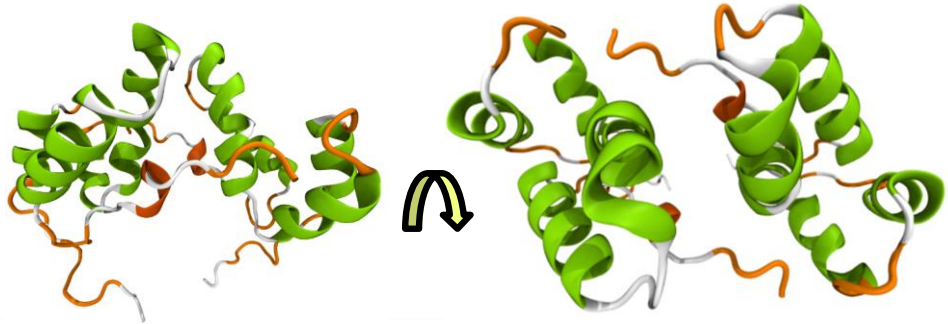


Sequence is organized into **secondary structure**



Secondary structure causes chain to fold into **tertiary structure**

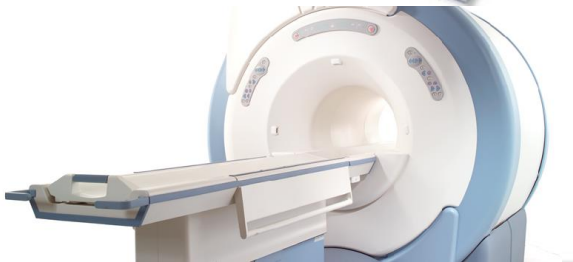
Quaternary structure complexes multiple, folded chains

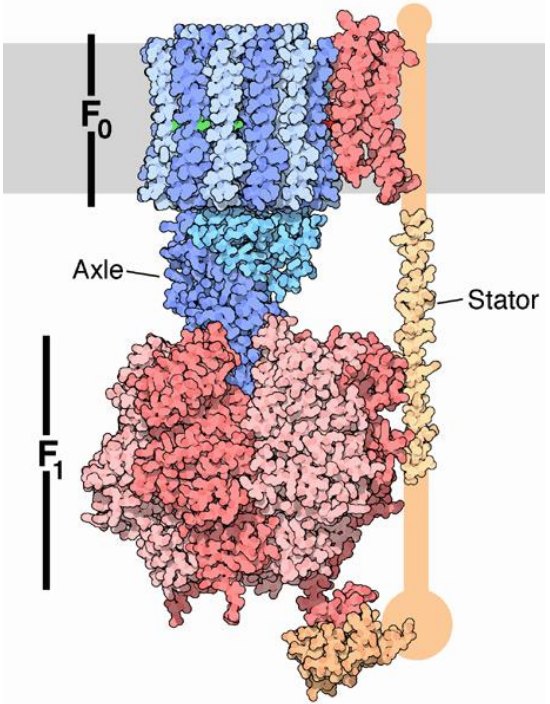


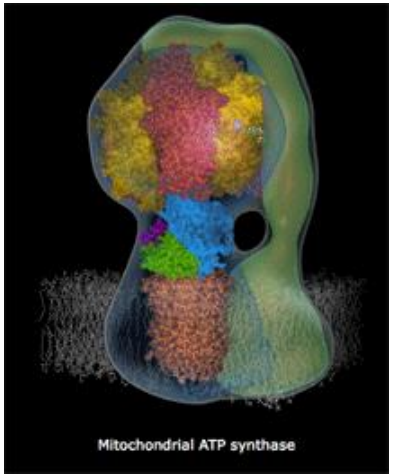
Structure is essential to function

Determining a protein's *native* structure is critical

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



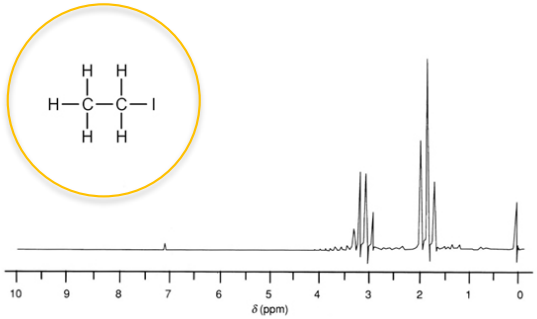




NMR studies proteins with minimal tampering (i.e., freezing or crystallization)

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

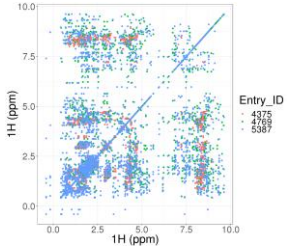
What does an NMR experiment look like?

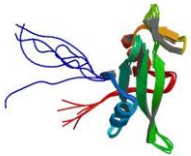


(repeat for remaining atom types) … then

Data collection (**days/weeks**)

Chemical shift assignment (**months/years**)





- ❑ Validation
- ❑ Positional restraints
- ❑ Partial occupancies
- ❑ ...
- ❑ Deposition of structure

Correlation assignment (**months/years**)

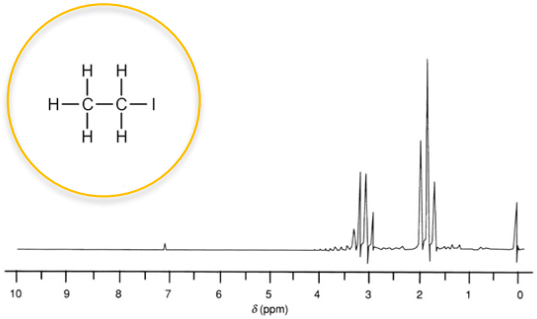
Structural ensemble

Completion

What does an NMR experiment look like?

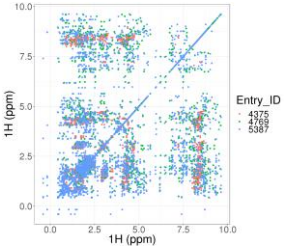
?

Data collection (**days/weeks**)

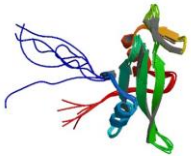


(repeat for remaining atom types) … then

Chemical shift assignment (**months/years**)



Correlation assignment (**months/years**)



Structural ensemble

- ❑ Validation
- ❑ Positional restraints
- ❑ Partial occupancies
- ❑ ...
- ❑ Deposition of structure

Completion

Semi-empirical chemical shift prediction: PPM_One

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

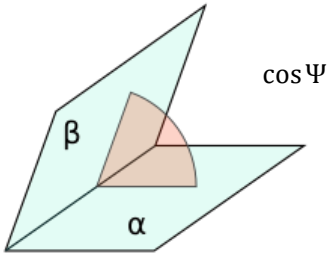
$$\delta_{CSpredicted} = \delta_{HBond} + \delta_{Dihedral} + \delta_{Ring\ Current} + \delta_{Magn\ Anisotropy} + \delta_{Electric}$$

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

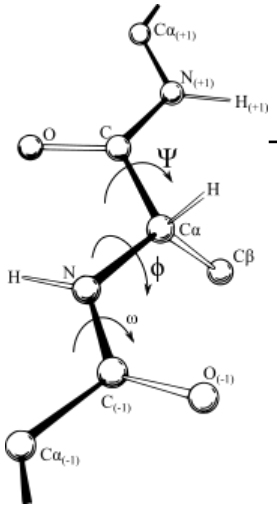
e.g., dihedral angle:

(α) $a_1x + b_1y + c_1z + d_1 = 0$

(β) $a_2x + b_2y + c_2z + d_2 = 0$



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



$$\delta_{\text{dihedral}} = k(1 + \cos(n\gamma + \phi))$$

More familiar challenges:

- NBody
- Dense linear algebra
- Unstructured grid (?)

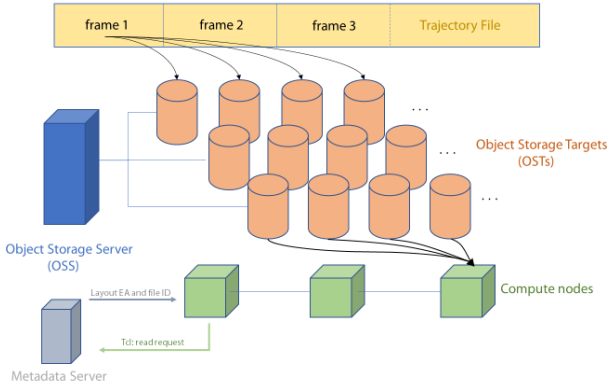
Dawei Li, Rafael Bruschweiler *J. Biomol. NMR* (2012)
Dawei Li, Rafael Bruschweiler *J. Biomol. NMR* (2015)

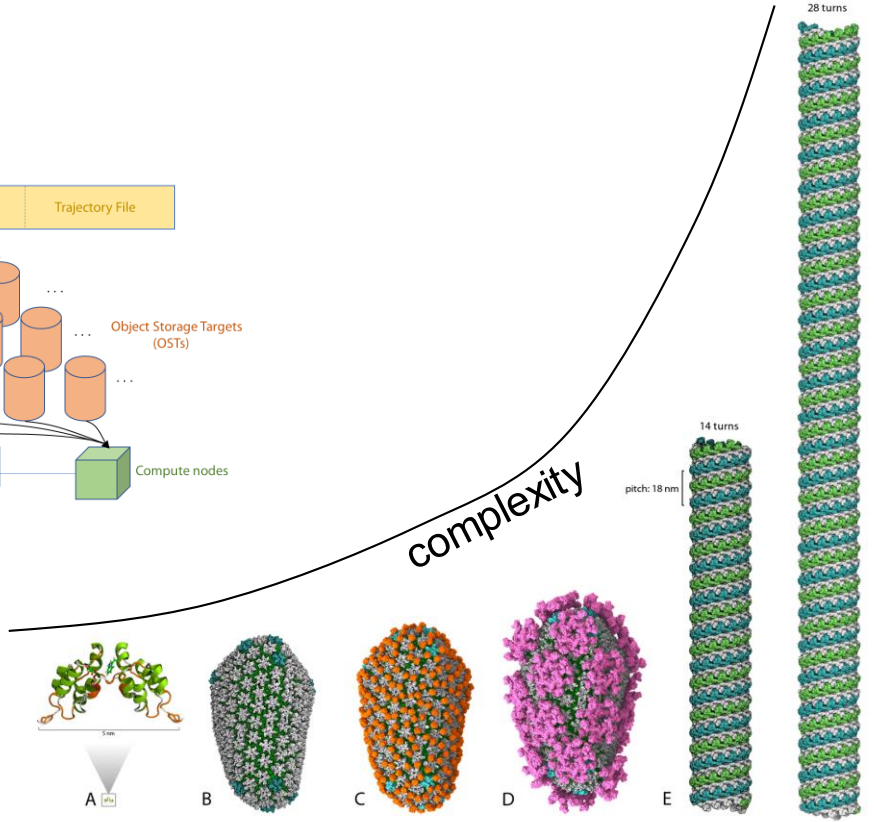
Takeaway: theoretical biophysics is compute and data intensive

Large systems necessitate high-performance codes and systems

Perilla, et al. *Nature* (2016)

64 million atomistic simulation of HIV-1 virion



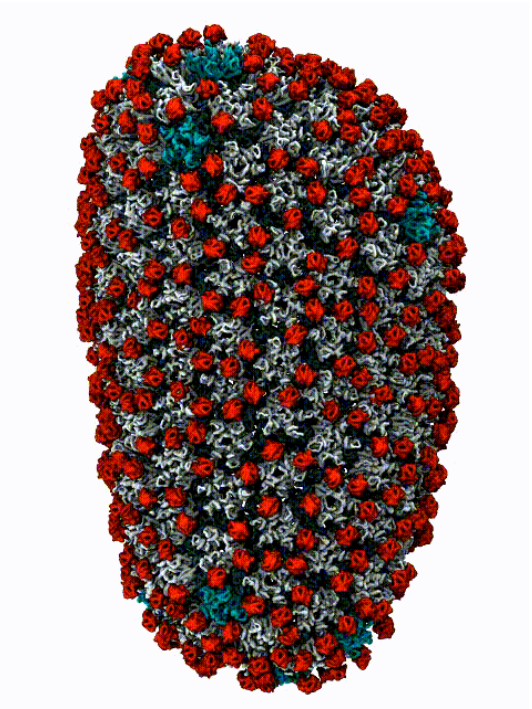


Project Motivation

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

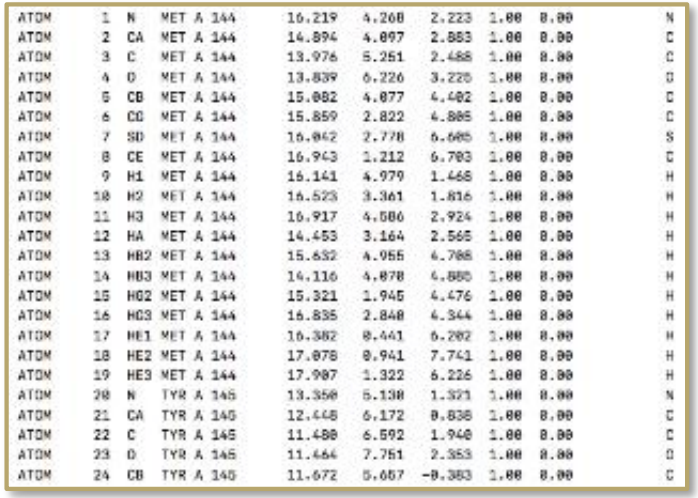
Our goal:

- To enable execution of multiple chemical shift predictions repeatedly
- To allow chemical shift predictions for larger scale structures



Introduction to the PPM_One code

- 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   3.226  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   4.886  1.00  0.00           C
ATOM      7  SD  MET A 144      16.062   2.778   6.685  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   4.078   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   2.848   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      17.987   1.322   6.226  1.00  0.00           H
ATOM     20  N   TYR A 145      13.350   5.138   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      11.488   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   5.667  -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

Profile Driven Development

Profile Driven Development

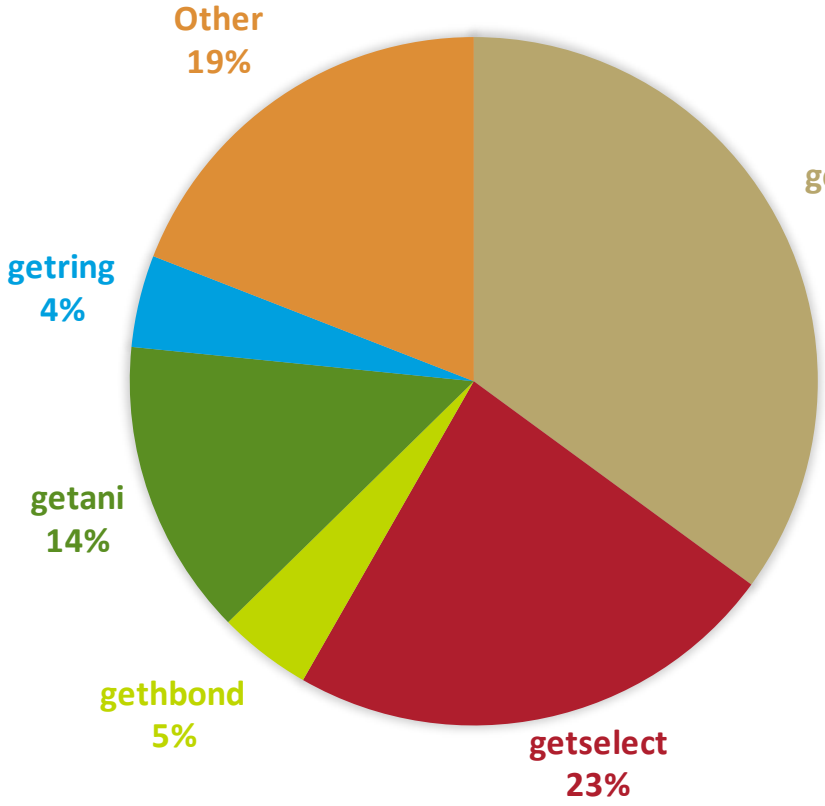
- Tackling a large and unfamiliar code is daunting
- Advantages of profiling:
  - High-level view of the code
  - Baseline performance metrics
  - Sanity check during the development process

Serial Code Profile (Main Function)

Main Function	% Runtime
main()	100%
predict_bb_static_ann(void)	81.226%
predict_proton_static_new(void)	16.276%
load(string)	1.921%

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

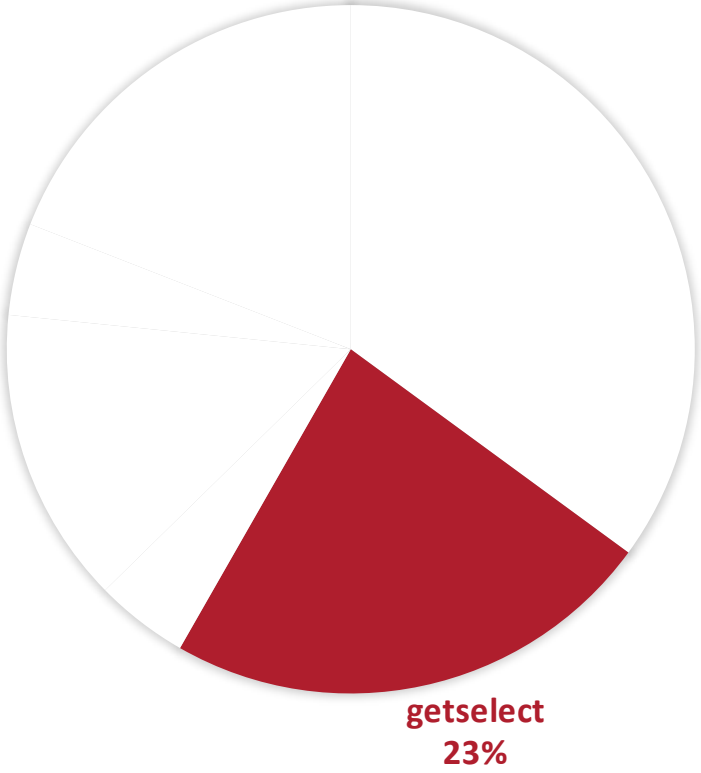


Other Contains:

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

Optimization in steps

- getselect()
- 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,&result);
}
```

Reusing the same flags results in the function returning the same set of atoms

Serial Optimization (getselect)

```
// Pseudocode for getselect function

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

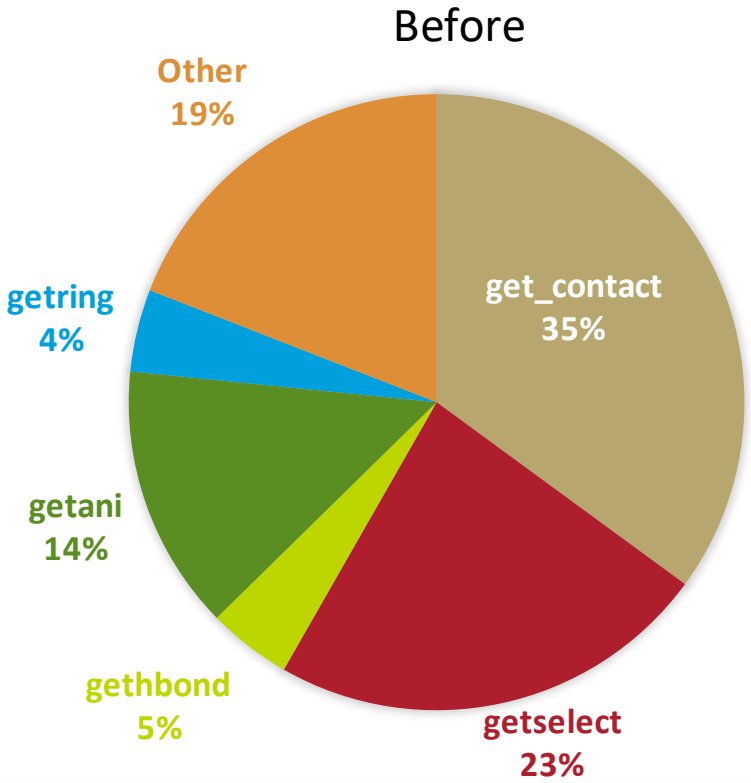
getselect originally accounted for **25%** of the codes runtime. After optimization, it takes less than **1%**.

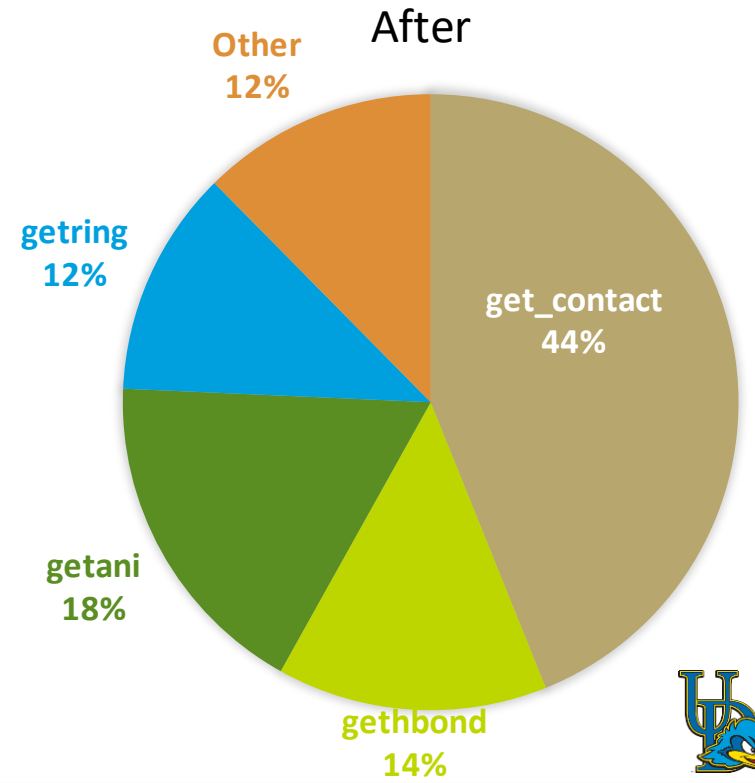
```
// Pseudocode for getselect function

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 in an inefficient C++ style way
  - Runtime for filtering functions went from **5+min to 1 second** for some datasets
- Replace C++ stl vectors:
  - All data is stored within stl vectors
  - There are a few ways to work around this for GPUs
  - We chose to just replace them with pointers when possible

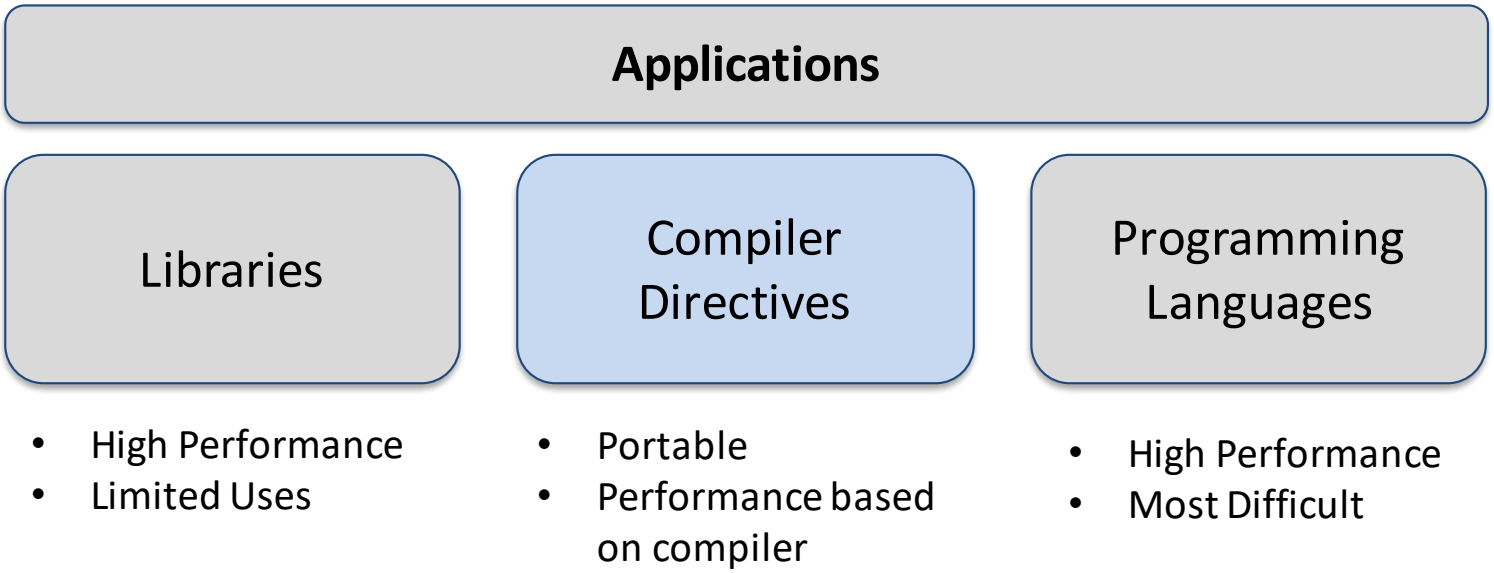
Serial Profile After Optimization





Porting PPM to GPUs

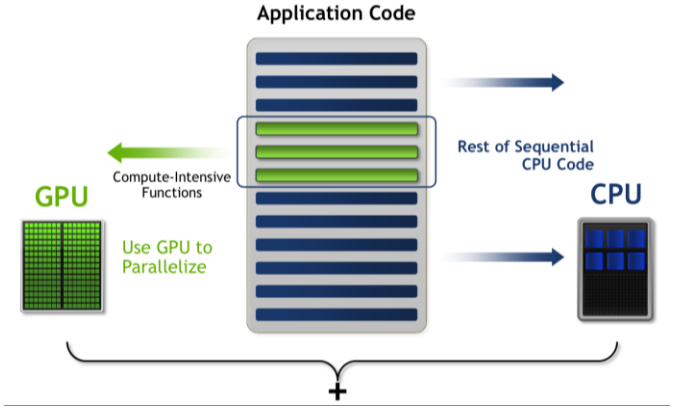
Our Weapon of Choice



Introduction to OpenACC

- OpenACC is a 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 freely available:

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

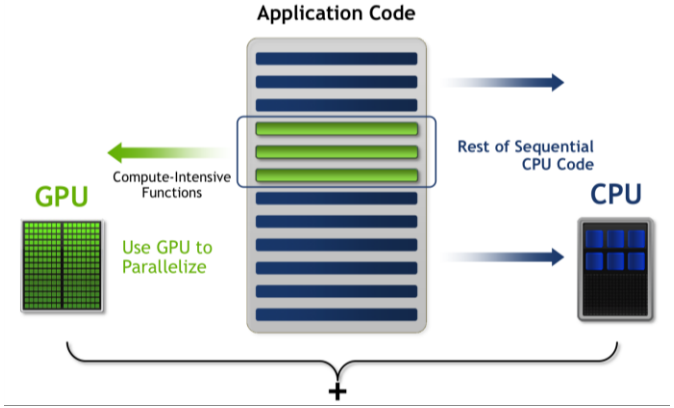


OpenACC
More Science, Less Programming

Introduction to OpenACC

Benefits:

- Portable without sacrificing performance
- Simple, based on directives
- Ease of code porting (no large code rewrites)




```
#pragma acc parallel loop
for(int i = 0; i < N; ++i)
    a[i] = a[i]*b[i] + c[i];
```

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**

```
for(i=1;i<index_size-1;i++)
{
    ...
    traj->get_contact(c1,c2,&result);
    ...
}
```

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

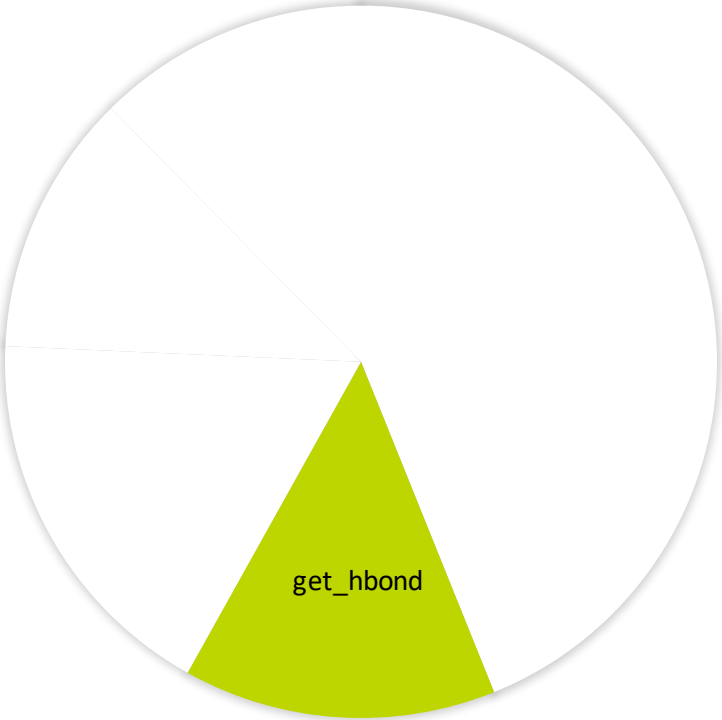
```
// For x,y,z coordinate
for(i=0;i<(int)pos.size();i++)
{
    ...
    // For every atom
    for(j=0;j<(int)used.size();j++)
    {
        // Calculate contact
        ...
    }
    result->push_back(contact);
}
```

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

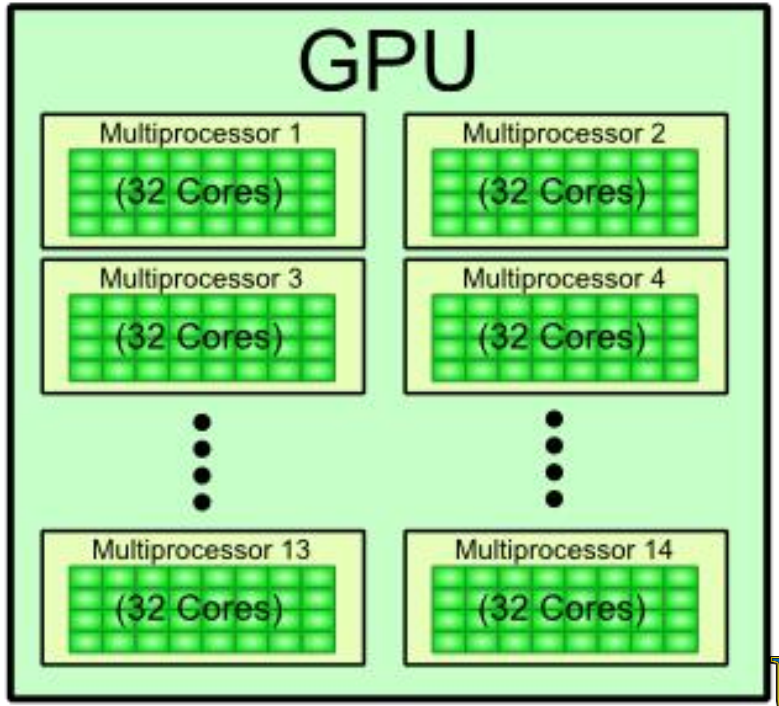
```
#pragma acc parallel loop gang
for(i=0;i<_hbond_size;i++)
{

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

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 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++)
        {
            ...
        }
    }
}
```

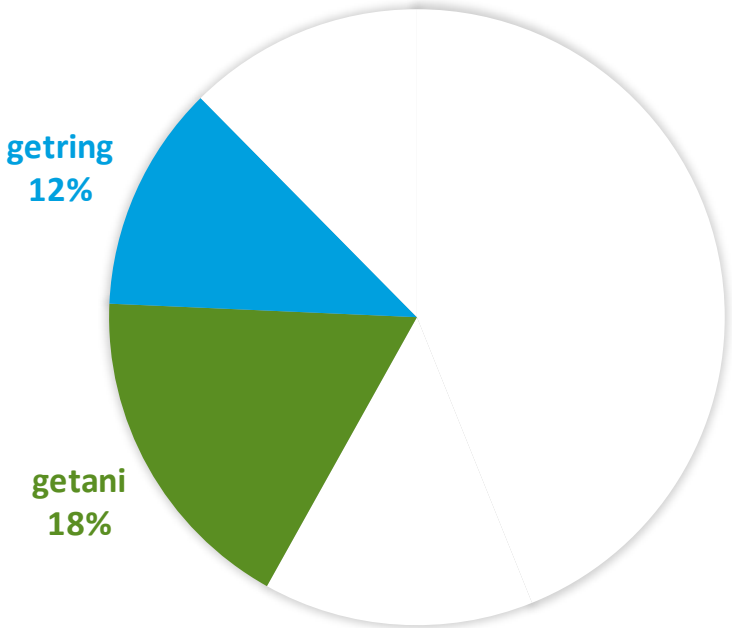


Acceleration of gethbond

```
#pragma acc parallel 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++)
        {
            ...
        }
    }
}
```

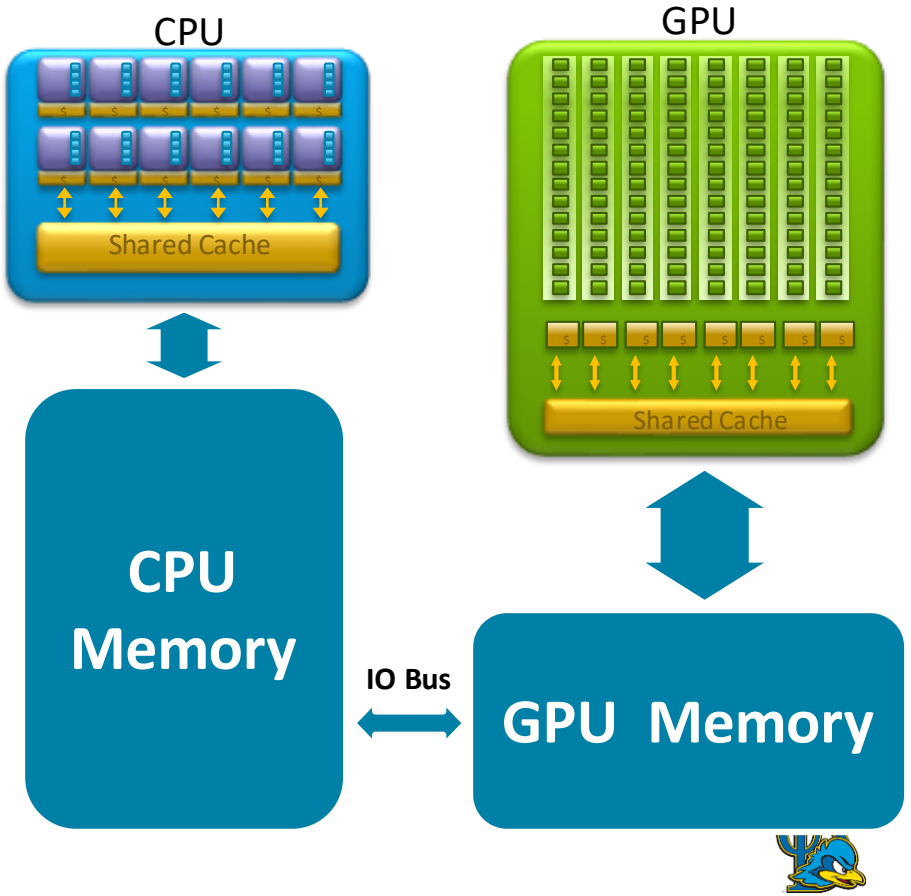
```
if(hbond[i].type==1){
    #pragma acc atomic update
    effect_arr[nid].n_length+=d;
    #pragma acc atomic update
    effect_arr[nid].n_phi+=phi;
    #pragma acc atomic update
    effect_arr[nid].n_psi+=psi
}
if(hbond[j].type==1){
    #pragma acc atomic update
    effect_arr[cid].c_lengh+=d;
  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

And the next most…and so on



Data Movement

- CPU and GPU memory is separate in a heterogenous system
- Connected via an IO Bus (PCI-E or NVLink)
- Programmer must explicitly manage two separate memory pools

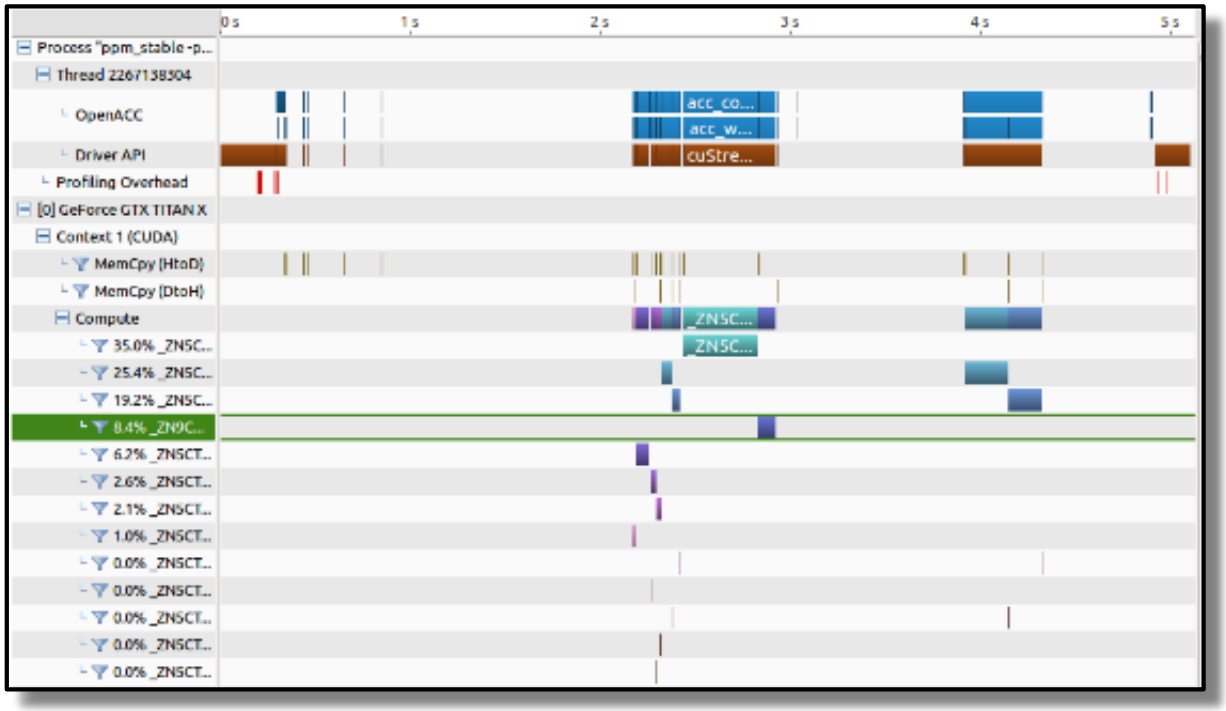


Data Movement

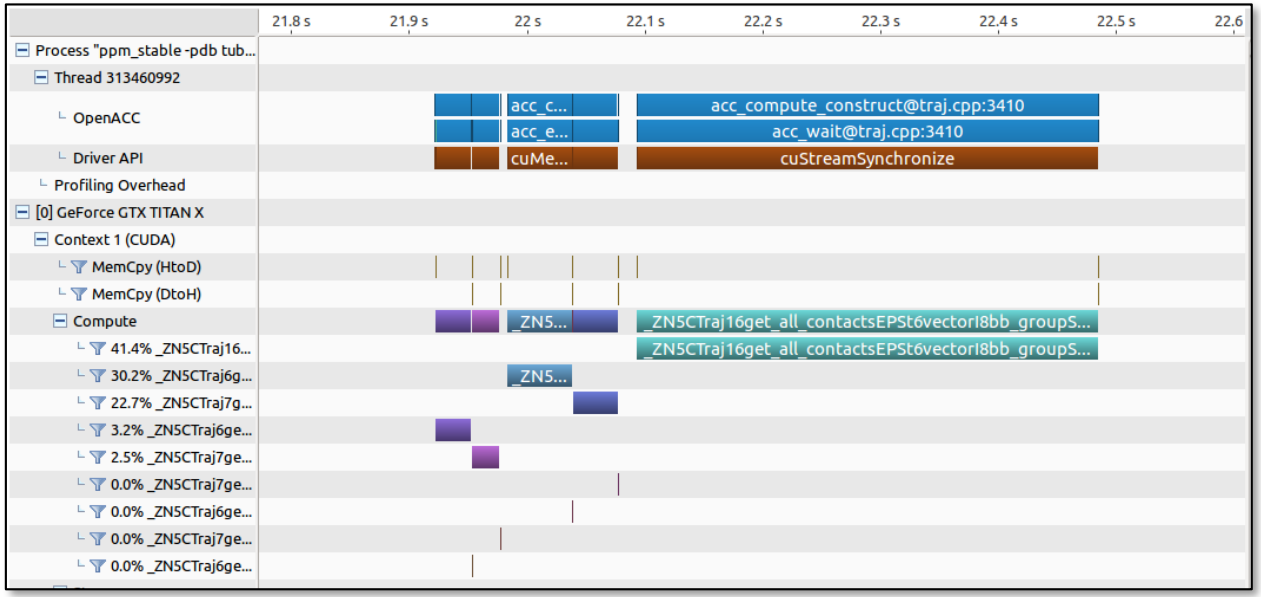
```
// Initialize X, Y, Z on host

...

#pragma acc enter data copyin(x_arr[0:x_size], \
                              y_arr[0:y_size], \
                              z_arr[0:z_size])
```

- 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

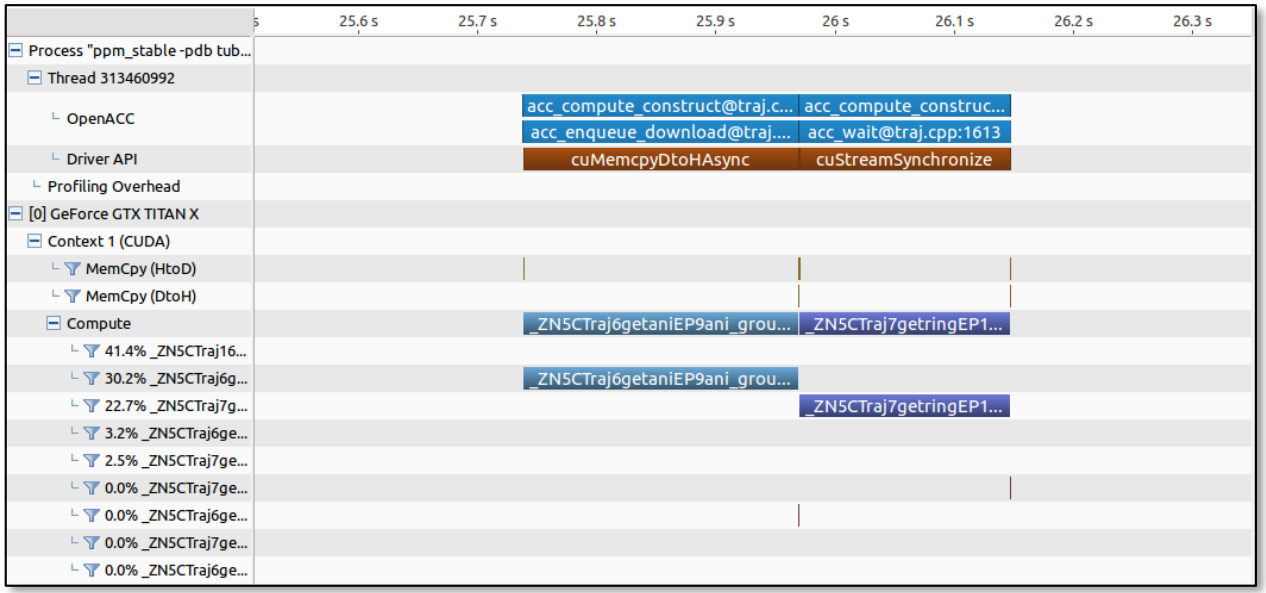
Parallel Profile



Parallel Profile



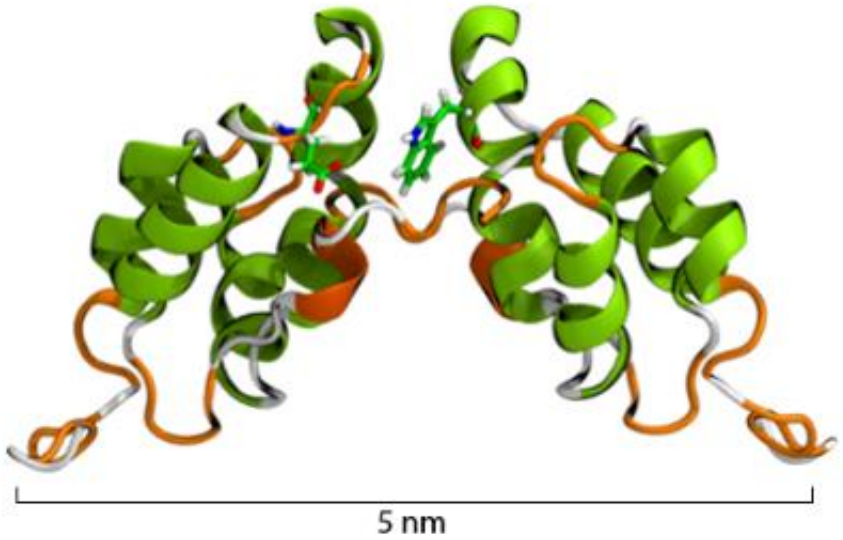
Parallel Profile



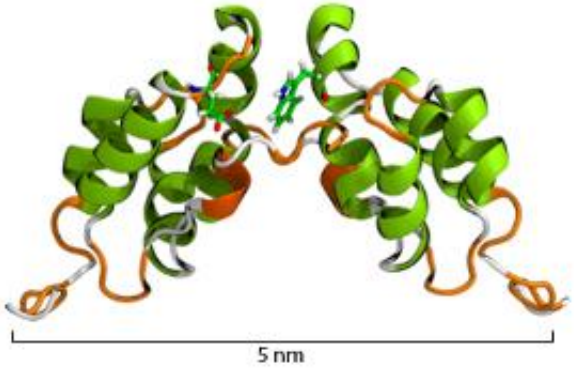
Results

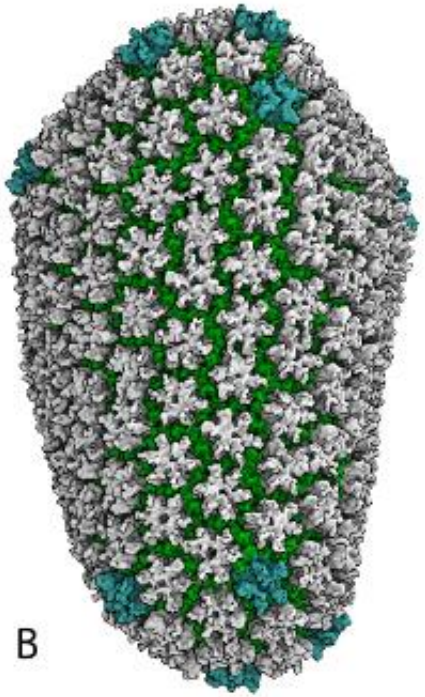
Was it worth it?

Experimental Datasets

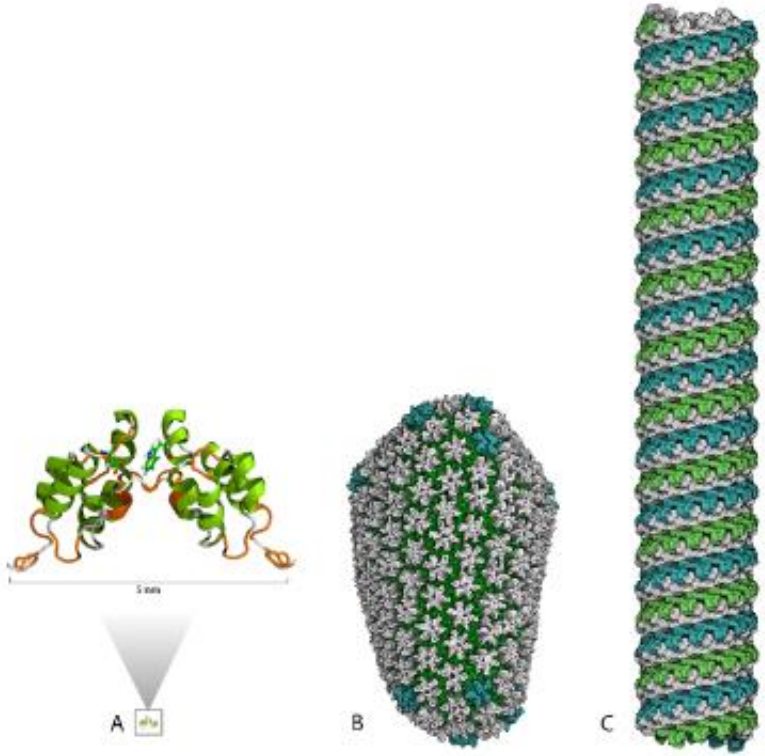


Experimental Datasets





Experimental Datasets



Experimental Setup

Machine	CPU	GPU	Machine	CPU
NVIDIA PSG (V100)	Intel Xeon E5-2698 (16 cores)	NVIDIA Tesla V100 (16GB HBM2)	NVIDIA PSG (V100)	Intel Xeon E5-2698 (16 cores)
NVIDIA PSG (P100)	Intel Xeon E5-2698 (16 cores)	NVIDIA Tesla P100 (16GB HBM2)	NVIDIA PSG (P100)	Intel Xeon E5-2698 (16 cores)
University of Delaware Vader	Intel i7 990x (12 cores)	NVIDIA Volta Titan V (12GB HBM2)	University of Delaware Vader	Intel i7 990x (12 cores)
University of Delaware Savina	Intel Xeon E5-2603 (8 cores)	NVIDIA Maxwell Titan X (12GB GDDR5)	University of Delaware Savina	Intel Xeon E5-2603 (8 cores)

Performance Results

	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (13.3M) Atoms
Serial (Unoptimized)	167.11s	3547.07 (1 hour)	7 hours approx.	14 hours approx.

Intel Xeon E5-2698 (32 cores)

Performance Results

	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (13.3M) 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)

Intel Xeon E5-2698 (32 cores)

Performance Results

	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (13.3M) 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 cores)	2.93s	109s	172s	427s

Intel Xeon E5-2698 (32 cores)

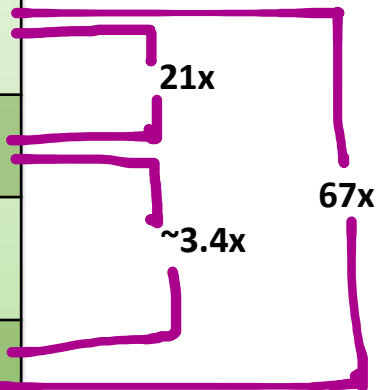
Performance Results

	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (13.3M) 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 cores)	2.93s	109s	172s	427s
NVIDIA PASCAL P100 GPU	1.72s	36s	69s	170s

Intel Xeon E5-2698 (32 cores)

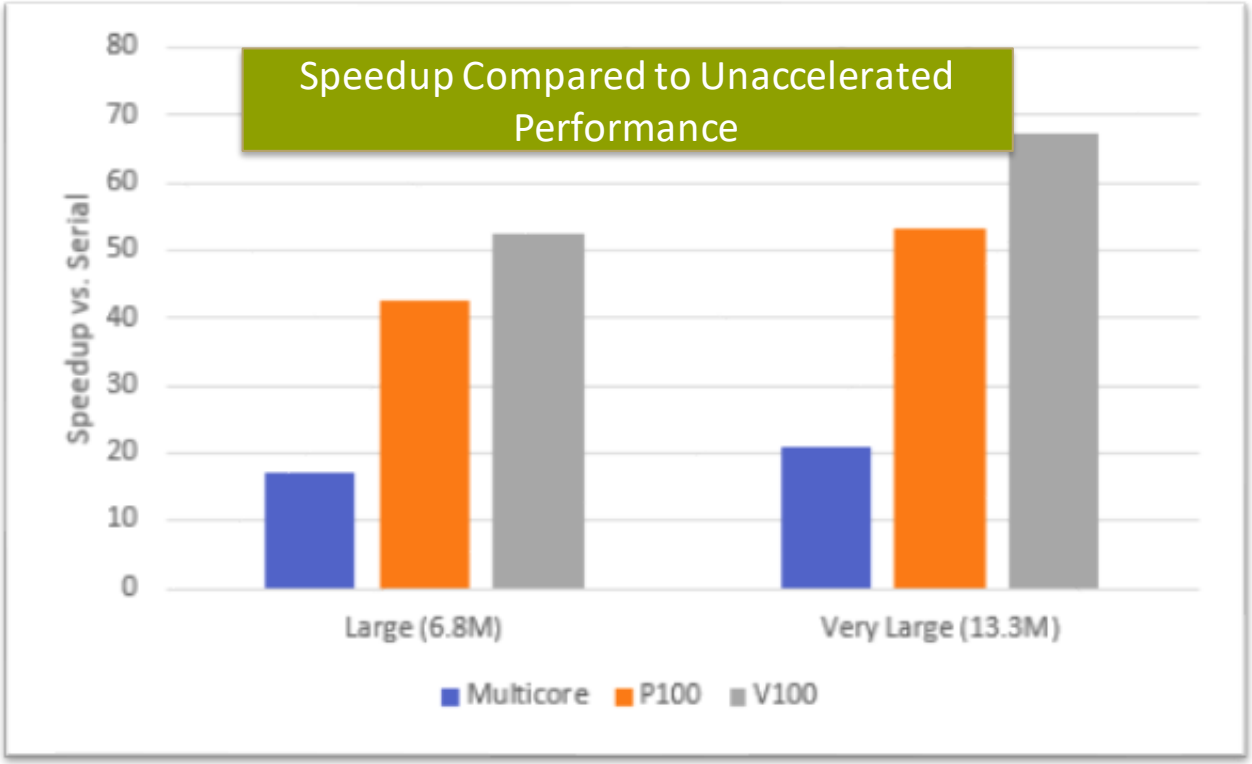
Performance Results

	Very Small (100K) Atoms	Medium (2.1M) Atoms	Large (6.8M) Atoms	Very Large (13.3M) 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 cores)	2.93s	109s	172s	427s
NVIDIA PASCAL P100 GPU	1.72s	36s	69s	170s
NVIDIA VOLTA V100 GPU	1.68s	29s	56s	134s



Intel Xeon E5-2698 (32 cores)

Performance Results



Performance Results (per function)

Function Name	Serial
get_contact	2505s
gethbond	337s
getani	29s
getring	19s

Performance Results (per function)

Function Name	Serial	Multicore	Speedup (Multicore vs Serial)
get_contact	2505s	100s	25x
gethbond	337s	19s	17x
getani	29s	1.5s	19x
getring	19s	0.84s	22x

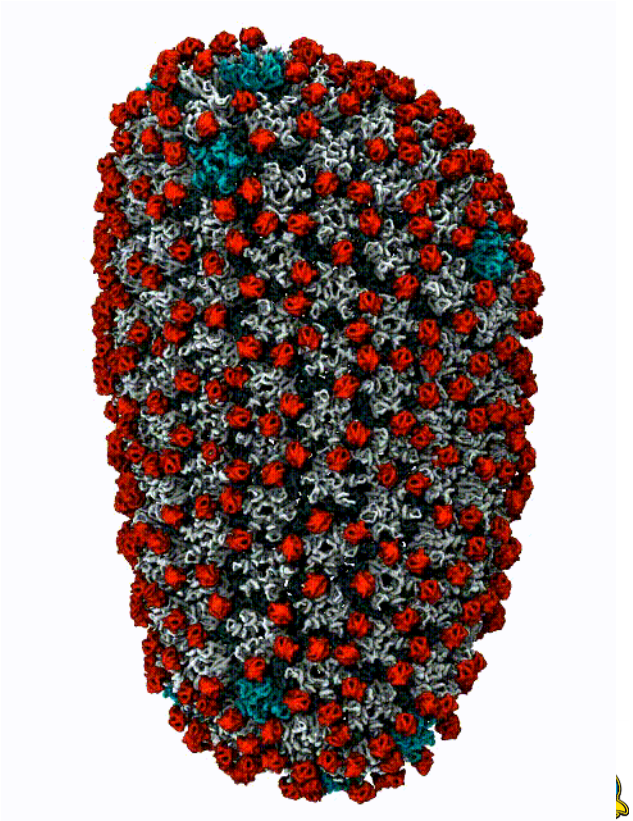
Performance Results (per function)

Function Name	Serial	Multicore	Speedup (Multicore vs Serial)	V100 GPU	Speedup (V100 vs Serial)
get_contact	2505s	100s	25x	15s	167x
gethbond	337s	19s	17x	1.24s	271x
getani	29s	1.5s	19x	0.09s	322x
getring	19s	0.84s	22x	0.09s	211x

Performance Results (per function)

Function Name	Serial	Multicore	Speedup (Multicore vs Serial)	V100 GPU	Speedup (V100 vs Serial)	Speedup (V100 vs Multicore)
get_contact	2505s	100s	25x	15s	167x	7x
gethbond	337s	19s	17x	1.24s	271x	15x
getani	29s	1.5s	19x	0.09s	322x	17x
getring	19s	0.84s	22x	0.09s	211x	9x

3D printed



Conclusions

- Achieved ~67x performance (in our best case) using a directive based programming model on GPUs
- Created a portable code that can run on single core, multicore, and GPU
- Allowed chemical shift to be estimated for large structures in a much more realistic amount of time
- Maintain the same accuracy (10e-3) as the base code