Accelerating Chemical Shift Prediction for Large-scale Biomolecular Modeling

Ph.D. Students: Eric Wright (efwright@udel.edu) and Mauricio Ferrato (mferrato@udel.edu)
Mentors: Prof. Sunita Chandrasekaran, Prof. Juan Perilla, Alex Bryer, Robert Searles
Dept. of Computer and Information Sciences and Dept. of Chemistry, University of Delaware
https://crpl.cis.udel.edu/

Find the code on github!

MOTIVATION AND GOAL

MOTIVATION

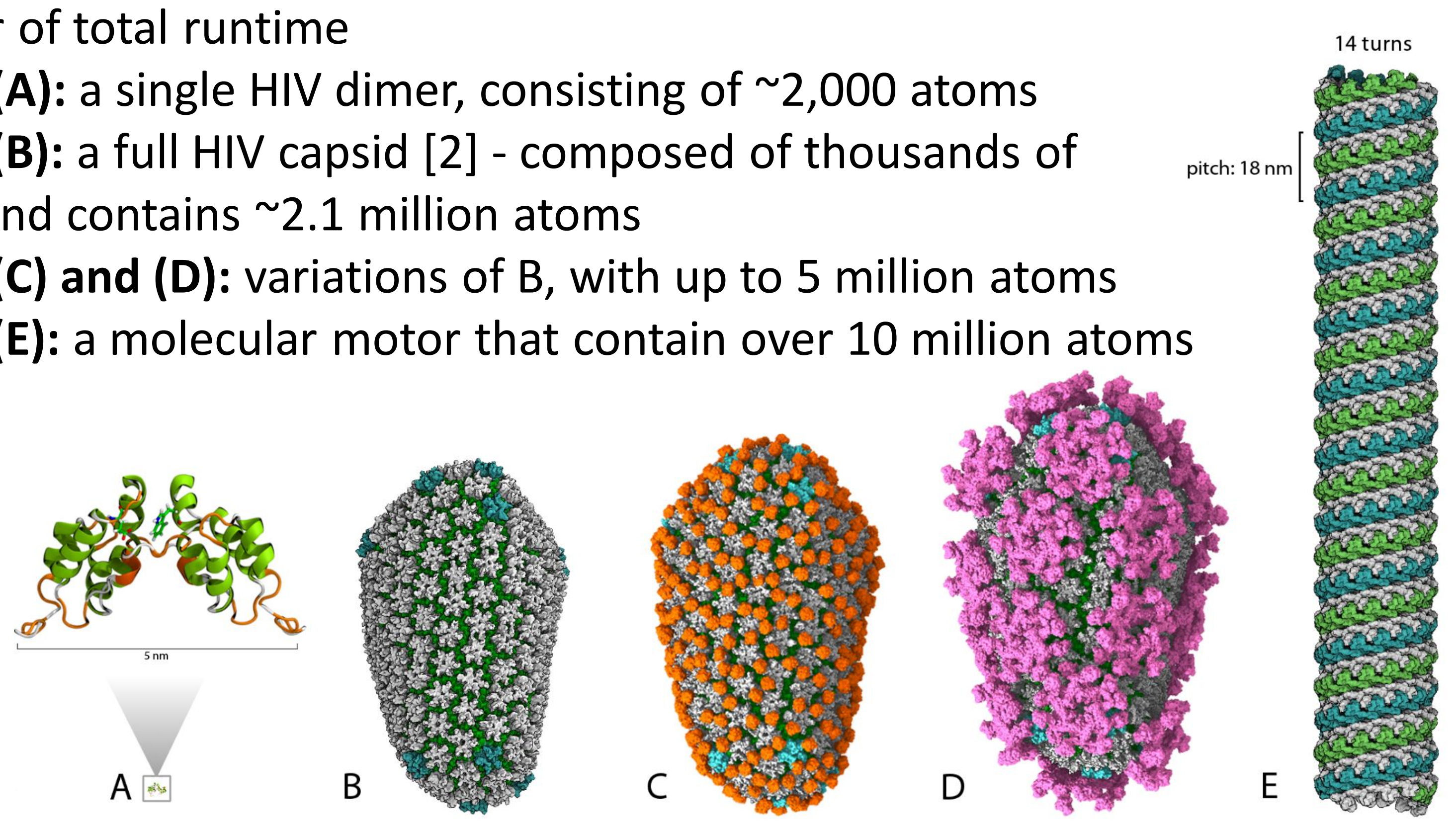
- Chemical shift, a principle observable in Nuclear Magnetic Resonance (NMR) instrumentation provides valuable insight into protein secondary structure
- Biomolecular complexes are large, with some atomic-models containing 100's of millions of atoms and structure determination of these complexes remain challenging due to computation complexity
- Recent advances in nanoscale imaging techniques (e.g., cryoEM, NMR, x-ray crystallography) make it possible for scientists to study these huge structures *in silico,* which means there's a huge need for well-optimized, parallel codes that can handle these techniques
- Chemical shift prediction algorithms have not been previously implemented for accelerated hardware, and computation of very large molecular structures was simply non-practical due to the large runtime

GOAL

- **Create a GPU accelerated chemical shift prediction application based on the PPM_One [1] code**

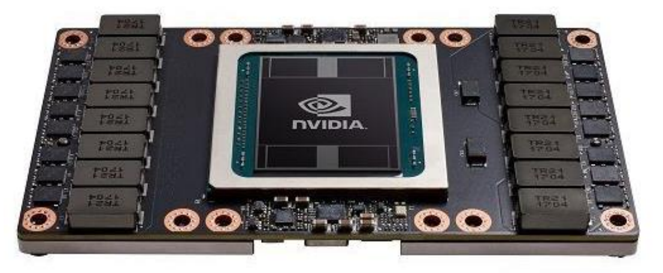
DATASETS

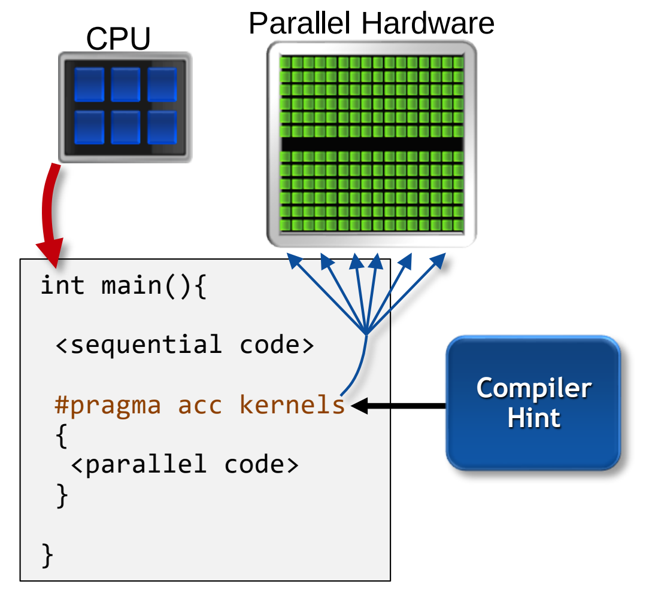
- A single molecular structure (typically a protein)
- Size of the molecule (number of total atoms) is the most reliable indicator of total runtime
- **Protein (A):** a single HIV dimer, consisting of ~2,000 atoms
- **Protein (B):** a full HIV capsid [2] - composed of thousands of dimers and contains ~2.1 million atoms
- **Protein (C) and (D):** variations of B, with up to 5 million atoms
- **Protein (E):** a molecular motor that contain over 10 million atoms



USING OpenACC DIRECTIVES

- In order to maintain portability of many different hardware architectures, and to ease the development process of working on a pre-existing code, we decided to use OpenACC to parallelize the code
- OpenACC [3], a directive-based parallel programming model for accelerators





```
int main(){

 <sequential code>

 #pragma acc kernels
 {
  <parallel code>
 }

}
```

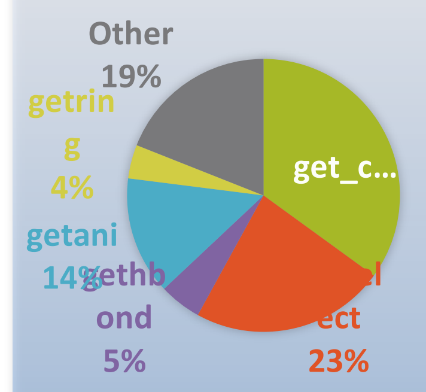
PROJECT ROADMAP

Project Start

- This is a multi-semester project with collaboration from the University of Delaware's computer science and chemistry departments
- The majority of the project was completed by a small team of undergraduate CIS students, two of which eventually moved on to be graduate students
- Two other graduate students (from CIS and CHEM) supported the team with GPU and Chemistry mentoring
- The team won the VIP Mid-Atlantic research competition.

Blog about the competition

Serial Profile



- We were originally introduced to the code by our CHEM collaborators
- Profiling with the PGPROF tool allowed us to get a high-level overview without reading through thousands of lines of code
- Profiling also gave us a clear idea of which parts of the code we would be tackling
- We then looked at these functions to see why they were taking this amount of time
- We also broke the "Other" category down to see what parts of it we should focus on

Serial Optimizations

- Our first major roadblock was with C++ STL Vectors
- This data structure obscures the data from us and makes it difficult to manage CPU-GPU data
- We replaced vectors with C-style arrays when possible, and in some cases we could simply use data() to get the underlying pointer for our OpenACC data constructs

```
vector<proton> protons;
vector<double> results;
traj->getani(protons.data(),
             results.data());
```

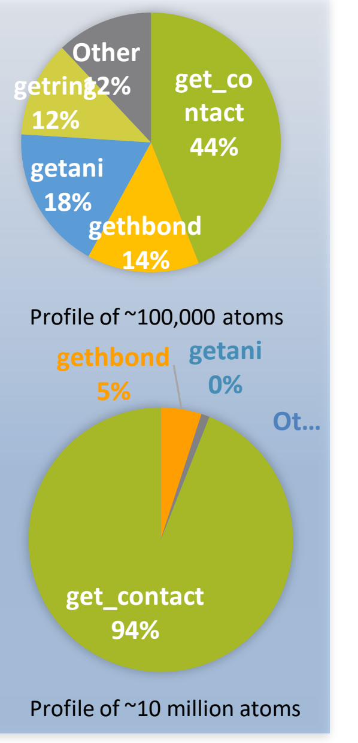
```
c2=getselect(":1-%@allheavy");
for( ... ) {  // Large main loop
// c2=getselect(":1-%@allheavy");
  get_contact(..., c2, ...);
}
```

Some simple code reordering is enough to reduce total time by over 20%!

- Some parts of the code showed up in the profile as time-consuming
- Some parts of the code were not very parallelizable because were originally written "algorithm focused" instead of "performance focused"
- We rewrote and reorganized some of these parts to make more sense for the parallel application
- One example is the "getselect" function which was profiled as the second-most time consuming function
- Optimizing "getselect" was very simple, but was not something we would have seen without a profile
- The original filter algorithm was not an ideal implementation (running in $O(n^2)$ time)
- We replaced it with an $O(n)$ implementation which reduced the execution time by minutes

Re-Profile

- Re-profiling after our optimizations shows dramatic reduction of the runtime of "getselect" and "Other"
- "get_contact" was still by far the most intensive function, followed somewhat by "gethbond"
- Re-profiling with larger data sizes also revealed that "get_contact" and "gethbond" were the strongest scaling functions as well, meaning that we would parallelize them first
- We also continued to use the profiler tool to check ourselves and ensure that the changes we were making were meaningful



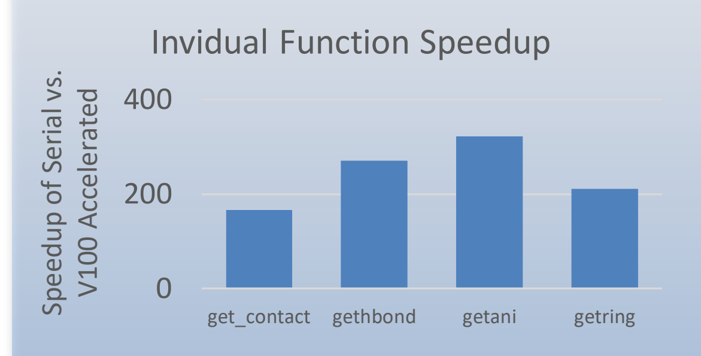
Data Management

- PPM_One is an object-orientated C++ code with most of our data existing within a class
- Because of this, we implemented our data management as an "OpenACC GPU Aware Class"
- This means tying GPU data alloc/dealloc to the class alloc/dealloc, and handling data movement with update directives in class member functions
- Most of our data is also read-only, allowing us to allocate them on the device and not have to worry about movement

```
CTraj::CTraj() {
    double *x = new double[size];
    #pragma acc enter data create(x[:size])
}

void CTraj::updateHost() {
    #pragma acc update self(x[:size])
}
```

Function Parallelization



	Serial	Multicore	V100 GPU
get_contact	2505s	100s	15s
gethbond	337s	19s	1.24s
getani	29s	1.5s	0.09s
getring	19s	0.84s	0.09s

Using Large 5.8M Atom Dataset on 32 core CPU and Volta V100 GPU (same setup as Results section)

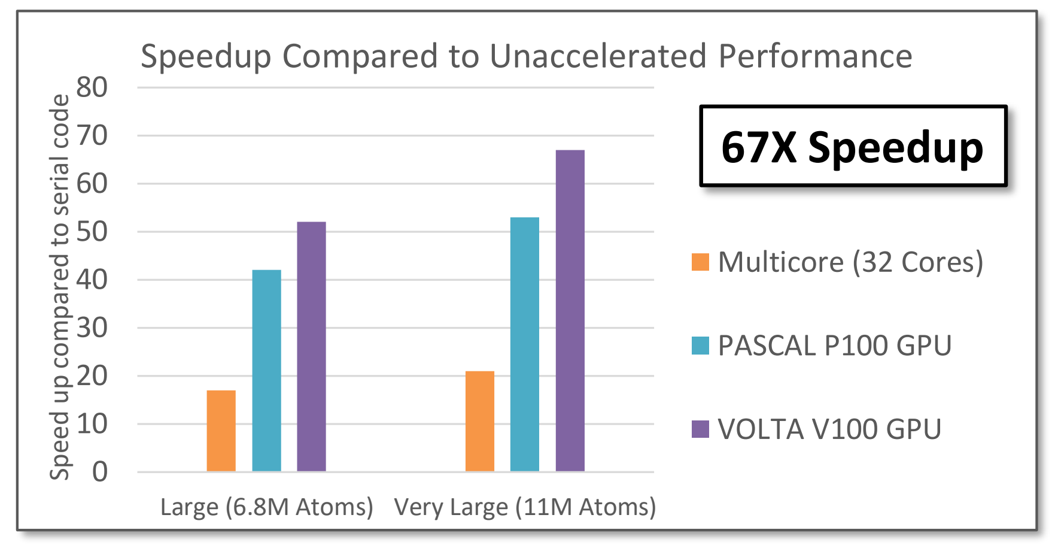
- There are several individual Molecular Dynamics functions that are used to estimate chemical shift (mostly rooted in linear algebra problems)
- These functions are **get_contact, gethbond, getani,** and **getring**
- Since they are all standalone, they present a good opportunity in porting them to other GPU accelerated molecular dynamics packages (see future work!)
- For this reason, it is important to note by how much each function sped up when using multicore CPU and GPUs

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 cores)	2.93s	109s	172s	427s
NVIDIA PASCAL P100 GPU	1.72s	36s	69s	170s
NVIDIA VOLTA V100 GPU	1.68s	29s	56s	134s

- Here we show the absolute runtime of the code with various datasets and GPUs
- All results using the Intel Xeon E5-2698 (32 cores) CPU and single GPUs
- PGI 18.4 compiler and CentOs 7.5 OS
- Accelerated speedup is measured with respect to the optimized serial performance

- With largest available dataset (11M atoms) and an NVIDIA V100 GPU, we see up a 67xspeedup respectively when compared to single core optimized performance
- OpenACC multicore with a dual socket 32 core CPU is at 21x speedup
- Compared to a fully utilized CPU node (32 cores), the V100 GPU is seeing ~3.4x speedup
- The main limitation we are facing is the extensive data pre-processing step



- While we brought down the time taken from ~14 hours to 134 seconds, of the 134 seconds of our best runtime, pre-processing takes about 110 seconds of it
- If we completely re-wrote the entire pre-processing portion of the code, we estimate that we would achieve ~13x speedup of V100 GPU vs. 32-core CPU

Future Work

- Scale PPM_ONE across nodes and multiple GPUs using MPI + OpenACC
- Incorporate core functions from PPM_One into other GPU accelerated packages, such as:
    - NAMD (Nanoscale Molecular Dynamics) enabling protein structure refinement combined with other experimental techniques
    - VMD (Visual Molecular Dynamics) enabling scientists to perform structure validation

References & Acknowledgements

1. Li, D., and R. Brüschweiler, 2015. PPM_One: a static protein structure based chemical shift predictor. Journal of Biomolecular NMR 62:403–409
2. Perilla, J.R., Zhao, G., Lu, M., Ning, J., Hou, G., Byeon, I.J.L., Gronenborn, A.M., Polenova, T. and Zhang, P., 2017. CryoEM structure refinement by integrating NMR chemical shifts with molecular dynamics simulations. The Journal of Physical Chemistry B, 121(15), pp.3853-3863.
3. https://www.openacc.org/

We gratefully acknowledge the support of NVIDIA PSG Cluster for access to their P100 and V100 GPUs used for this research. We also gratefully acknowledge Prof. Andy Novocin at the University of Delaware as the work was partially done under Vertically Integrated Project (VIP).