Molecular Trajectory Analysis

Presented by: SUPER GROUP 3

1

**List of Contributors**

1 MIT2019002 Ankita Hariyale 2 MIT2019003 Fanishwar Satyam 3 MIT2019004 Prashant Kushwaha 4 MIT2019005 Abhinav Anand 5 MIT2019006 Deepali Jindal 6 MIT2019007 Hemant Kumar Lader 7 MIT2019009 Kavita Singh Dhruv 8 MIT2019017 Pragati Pawar 9 MIT2019020 Biswajit Majumdar 10 MIT2019023 Gaurav Rajput 11 MIT2019024 Shivam Kasat 12 MIT2019042 Komal Garg 13 MIT2019051 Mayank Kabra 14 MIT2019052 Shekhar Gupta 15 MIT2019054 Shubham Kewat 16 MIT2019055 Mani Shankar Javvaji 17 MIT2019056 Yempalli Siva Kumar 18 MIT2019065 Sonu Kumar 19 MIT2019066 Koushal Kumar Sharma 20 MIT2019071 Himanshu 21 MIT2019085 Riya Khanna 22 MIT2019086 Khushbu Jayeshkumar Mevada 23 MIT2019087 Aadil Ali 24 MIT2019089 Ajay Kumar 25 MIT2019097 Praveen Kumar Raghav 26 MIT2019112 Harish Kumar 27 MIT2019114 Rohit Kumar 28 MIT2019117 Sumit Kumar 29 MIT2019121 Sudhanshu Ranjan 30 MIT2019123 P.N. Janani 31 MBI2019011 Utkarsh Singh

2

**TABLE OF CONTENTS**

I. List of Contributors........................................................2 II. Table of Contents...........................................................3

**WHY THIS PROJECT?**

1. SoftwareSpecifications.........................................................4

2. Software Requirements............................................................4

3. Software API’s and Dependencies...........................................5

4. Software Process................................................................8

**5. DESIGN**

5.1. DFD Level 0...............................................................9

5.2. DFD Level 1...............................................................10

5.3.DFDLevel 2.................................................................11

5.2.1.1. PDB Generation......................................................11

5.2.1.2. Masking................................................................14

5.2.1.3. RMSD Calculation...................................................16

5.2.1.4. Average Structure.....................................................19

5.2.1.5. Distance and Angle..................................................20

5.2.1.6. Dihedrals..............................................................24

5.2.1.7. Hydrogen Bond........................................................28

5.2.1.8. AutoImage.............................................................29

5.2.1.9. Essential Dynamics...................................................31

5.3. Integrated DFDLevel2....................................................33

5.4.Class Diagram...............................................................35

6. Installation Instruction.........................................................36

7. SoftwareValidation.............................................................37

8. Software Outcomes.............................................................37

9. Benchmarks.....................................................................38

10. Specifications...................................................................38

11. Group Members.................................................................39

**BIBLIOGRAPHY**..................................................................41

3

**Why This Project?**

The software build in this project is to optimize the time complexity of other similar systems like cpptraj that works sequentially or uses parallelization for small number of functionalities. Such systems cannot take advantage of the growing computational power. The project aims at developing a system that can use distributed system architecture in order to provide faster results for the analysis.

**1.Software Specification:**

This Software System study the big dataset of Molecular Dynamic simulation and computes for Mask/Select, center, imaging, superposition, Average Structure, PDB generation, RMSD.

**2. Software Requirements:**

To speed up the process of simulation, we need parallel processing with distributed computing so Apache SPARK as big data processing engine and SCALA as language will be required. All the simulation will be compared with **cpptraj**. In this there will be dataset stored in files. Each file will contain frames of molecular dynamics Simulations.

**● Spark - Developed** at “UC Berkeley in 2009”, Spark is an open source big data processing engine, which has faster cluster computing platform. It works as nearby Real Time processing system as well as Batch Processing System. It was designed for fast computation. Spark is an optimized engine that provides high level expressive APIs in Java, Scala, Python and R which allows big data professionals for fast computation.

**● Scala -** Scala is a general-purpose, high-level, multi-paradigm programming language. Scala has been created by Martin Odersky and he released the first version in 2003. Scala smoothly integrates features of object-oriented and functional languages. It is a pure object-oriented programming language which also provides support to the functional programming approach. Scala programs can convert to byte codes and can run on the JVM (Java Virtual Machine). Scala stands for Scalable language. It also provides JavaScript runtimes. Scala is highly influenced by Java and some other programming languages like Lisp, Haskell, Pizza etc.

**● Cpptraj** – Cpptraj (the successor to ptraj) is the main program in Amber for processing coordinate trajectories and data files.Cpptraj has a wide range of functionality, and makes use of OpenMP/MPI to speed up many calculations, including processing ensembles of trajectories and/or conducting multiple analyses in parallel with MPI. Here, Trajectories with

4

different topologies can be processed in the same run. Several actions/analyses in cpptraj are OpenMP parallelized.

**3. Software API’s and Dependencies:**

**Dependencies:**

● **netcdfAll-5.1.0.jar** - The netCDF library implements the full Common Data Model (CDM) model, including all other jar dependencies.

● **slf4j-jdk14-1.7.9.jar** - SLF4J stands for Simple Logging Facade for Java. It provides a simple abstraction of all the logging frameworks. It enables a user to work with any of the logging frameworks such as Log4j, Logback, JUL (java.util.logging), etc. using single dependency.

● "edu.ucar" % "cdm" % "4.5.5" exclude("commons-logging", "commons-logging"),

● "edu.ucar" % "grib" % "4.5.5" exclude("commons-logging", "commons-logging"),

● "edu.ucar" % "netcdf4" % "4.5.5" exclude("commons-logging", "commons-logging"),

● "org.apache.spark" % "spark-core\_2.11" % "2.3.0",

● "org.apache.spark" % "spark-sql\_2.11" % "2.3.0",

● "org.jmockit" % "jmockit" % "1.34" % "test",

● "org.scalanlp" %% "breeze" % "1.0",

● "org.scalanlp" %% "breeze-natives" % "1.0",

● "org.scalanlp" %% "breeze-viz" % "1.0",

● "org.apache.hadoop" % "hadoop-mapreduce-client-core" % "2.7.1",

● "org.apache.hadoop" % "hadoop-common" % "2.7.1"

● "org.ejml" % "ejml-all" % "0.38"

5

**API’s :**

1. MTA.FileWrite.writePDB (pdbDF : DataFrame, directoryName: String) : Unit

2. MTA.FileWrite.writePdbFrameWise (pdbDF : DataFrame, frameNo : Int, directoryName

: String) : Unit

3. MTA.FileWrite.writeMaskedFrameWise (pdbDF : DataFrame, frameNo : Int,

directoryName : String) : Unit

4. MTA.FileWrite.writeTime(fileName : String, time : Any ): Unit

5. MTA.FileWrite.writeArraytoFile(arr : Array[String], fileName : String): Unit

6. MTA.createDF.getCoordDF (coordFileNameRef1 : String): DataFrame

7. MTA.createDF.getTopologyDF (topoList : Array[String]) : DataFrame

8. MTA.createDF.getPdbDF(coordDf: DataFrame, topologyDF : DataFrame, frameNum :

Int) : DataFrame

9. MTA.createDF.makeCoordUnionDF (coordFilameRefList : String) : DataFrame

10. MTA.createDF.getCellLengths (coordFileNameRef1 : String) : DataFrame

11. MTA.MaskData.getQuery (query : String) : String

12. MTA.MaskData.maskNETCDF (topoDF : DataFrame, coordDF : DataFrame, Query :

String): DataFrame

13. MTA.MaskData.maskPDB (PDBFileName : String, Query : String, context:

SQLContext): DataFrame

14. MTA.RMSD.rmsdSimpleFrameWise (df : DataFrame, frameCount : Int) : Array[String]

15. MTA.RMSD.rmsdForParticularAtom (df : DataFrame, atomName : String, frameCount :

Int) : Array[String]

16. MTA.RMSD.rmsdForResidueSequence (df : DataFrame, start : Int, end : Int, frameCount

: Int) : Array[String]

17. MTA.RMSD.rmsdForParticularResidue(df : DataFrame, residueNum : Int, frameCount :

Int) : Array[String]

18. MTA.AverageStructure.averageStructure(topoDF : DataFrame, coordDF : DataFrame,

directoryName: String) : DataFrame 6

19. MTA.DistanceAndAngle.distanceBetweenAtoms(topoDF : DataFrame,coordDF :

Array[DataFrame],query: String): Unit

20. MTA.DistanceAndAngle.angleBetweenAtoms (topoDF : DataFrame, coordDF :

Array[DataFrame], query: String) : Unit

21. MTA.DistanceAndAngle.distanceBetweenResidue (topoDF : DataFrame, coordDF :

Array[DataFrame], query: String): Unit

22. MTA.DistanceAndAngle.angleBetweenResidue (topoDF : DataFrame, coordDF :

Array[DataFrame], query: String): Unit

23. MTA.DistanceAndAngle.calculateDistance ( x1 : Double, y1: Double, z1 : Double, x2 :

Double, y2: Double, z2: Double): Double

24. MTA.DistanceAndAngle.calculateAngle(x1: Double, y1: Double, z1: Double, x2:

Double, y2: Double, z2: Double, x3: Double, y3: Double, z3: Double): Double

25. MTA.HBond.list(distance: Double, angle:Double,outFileName: String)

26. MTA.Dihedral.getBackboneDihedralAngle(topoDF: DataFrame, coordDF: DataFrame,

directoryName: String) : Dataset[String]

27. MTA.WriteDihedralAngles.writeDihedralAngle (dihedralArray : ArrayBuffer[String]) :

Unit

28. MTA.AutoImaging.autoImaging(topoDF:DataFrame,coordDF:DataFrame,

cellLengthDF: DataFrame, directoryName : String) : Dataset[Row]

29. MTA.EssentialDynamics.getEiganValues(topoDF : DataFrame, coordDF:

Array[DataFrame]) : Unit

30. MTA.EssentialDynamics.getEigVec(Int) : Unit

7

**4. Software Process:**

For this development we’ll be using **RAD** (Rapid Application Development) software development life cycle. Because of shorter time and less pre-planning, this is right process model to use. The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable. As our project is based on iterative development and it requires rapid changes and we have the sort of time to complete the project so RAD is used in our project. As our project is based on iterative development and it requires rapid changes and we have the sort of time to complete the project so RAD is used in our project. Following are the various phases of the RAD Model:

**● Application Generation:** The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.

**● Testing and Turnover:** The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

8

**5. Design**

**5.1 DFD LEVEL 0 (Context level Diagram)**

There are 3 major modules to be included in this level. This is used for visualizing the data movement across the system. There are 2 external entities and a single ‘Process’.

The external entities here are:

1. User – Provides the Topology and Trajectory file to the Process. 2. Console – Prints the output.

Process – Read the Coordinates from trajectory file.

**DFD - LEVEL 0-**

9

**5.2. DFD Level 1**

Here the ‘Process’ in level 0 is to be decomposed to get a better view of the system.

10

**5.3. DFD Level 2**

This level describes the major modules in more detail. So this level will include :

**PDB Generation**

There are two input files in PDB generation process -

1. coordinate file(.crd file) 2. topology file(.top file)

From topology file, we stored usable data (atom number, atom name, atom type, residue number, residue name) into RAM and appended the coordinates from .crd file to that stored data and generated PDB file.

**API’s Used:**

1. MTA.createDF.getCoordDF (coordFileNameRef1 : String): DataFrame

**Input** : coordinate file **Output** : dataframe containing coordinates **Description** : takes the coordinate file as input and creates the dataframe

2. MTA.createDF.getTopologyDF (topoList : Array[String]) : DataFrame

**Input** : Topology File **Output** : Dataframe of topology file **Description** : takes the topology file as input and creates dataframe

3. MTA.createDF.getPdbDF(coordDf: DataFrame, topologyDF : DataFrame, frameNum :

Int) : DataFrame

**Input** : coordinate dataframe, topology dataframe and frame number **Output** : PDB dataframe **Description** : Merges the topology and coordinates DataFrame along with the Terminate statements to generate the PDB DataFrame

4. MTA.createDF.makeCoordUnionDF (coordFilameRefList : String) : DataFrame

**Input** : coordinate file reference list **Output** : coordinate DataFrame

11

**Description** : performs union between two coordinate DataFrames and returns the coordinate DataFrame

5. MTA.createDF.getCellLengths (coordFileNameRef1 : String) : DataFrame

**Input** : coordinate file reference list **Output** : returns cell length **Description** : calculate cell length for given coordinate file list

6. MTA.FileWrite.writePDB (pdbDF : DataFrame, directoryName: String) : Unit

**Input** : DataFrame of PDB, Directory Name **Output** : PDB File written onto Disk **Description** : Merges the topology and coordinates array along with the Terminate statements to generate the PDB file onto the disk.

7. MTA.FileWrite.writePdbFrameWise (pdbDF : DataFrame, frameNo : Int, directoryName

: String) : Unit

**Input** : PDB DataFrame, Frame Number, Directory Name **Output** : PDB File written onto Disk framewise **Description** : Merges the topology and coordinates array along with the Terminate statements to generate the PDB file onto the disk.

8. MTA.FileWrite.writeMaskedFrameWise (pdbDF : DataFrame, frameNo : Int,

directoryName : String) : Unit

**Input** : PDB DataFrame, Frame Number, Directory Name **Output** : masked PDB File written onto Disk **Description** : Merges the topology and coordinates array along with the Terminate statements to generate the PDB file onto the disk.

9. MTA.FileWrite.writeTime(fileName : String, time : Any ): Unit

**Input** : File name and time

**Output** : Time taken

**Description** : calculates the time taken for writing each file.

10. MTA.FileWrite.writeArraytoFile(arr : Array[String], fileName : String): Unit

**Input** : String array and file name

**Output** : writes the array into the given file

12

**Description** : takes a file as input and writes the input array into file

**DFD-**

13

**Masking**

It’s a process of filtering of atoms that takes topology file and Trajectory file as input and generates the required atoms as the output , upon which the further calculations are performed on them.

Masking can be done in 2 ways –

1. By Atoms – Atom name, Atom Number, Atom Type 2. By Residue – Residue name, Residue Number

**“ @ ”** – This is used for selection of Atoms. **“ : ”** – This is used for selection of Residues.

**API’s Used:**

1. MTA.MaskData.getQuery (query : String) : String

**Input** : User Query **Output** : Masking Query String

2. MTA.MaskData.maskNETCDF (topoDF : DataFrame, coordDF : DataFrame, Query :

String): DataFrame

**Input**: Array of Coordinate DF, Topology DF, Masking String **Output**: Masked Dataframe **Description**: This function takes a single dataframe containing all the frames and removes water from it.

3. MTA.MaskData.maskPDB (PDBFileName : String, Query : String, context:

SQLContext): DataFrame

**Input**: Pdb file name, Masking Query, SQL context **Output**: Masked Dataframe **Description**: This function takes a single dataframe containing all the frames and removes water from it.

14

**DFD-**

15

**RMSD Calculation**

It’s the **Root Mean Square Deviation Analysis** of the atomic position used in MD Simulations to study the stability of our simulated system. It’s the physical distance between structures.

The steps involved in this – In bioinformatics, the root-mean-square deviation of atomic positions is the measure of the average distance between the atoms. In the study of globular protein conformations, one customarily measures the similarity in three-dimensional structure by the RMSD of the Cα atomic coordinates after optimal rigid body superpositon.

**API’s Used:**

1. MTA.RMSD.rmsdSimpleFrameWise (df : DataFrame, frameCount : Int) : Array[String]

**Input** : Taking the pdb dataframe. **Output** : Array[String] : It will return an array of string, which contain RMSD value with frame number. **Description** : It will calculate the RMSD values for each atom framewise. FrameCount : It will contain numbers of frame in given dataframe.

2. MTA.RMSD.rmsdForParticularAtom (df : DataFrame, atomName : String, frameCount :

Int) : Array[String]

**Input** : Taking the pdb dataframe, atom name. **Output** : Array[String] : It will return an array of string, which contain RMSD value with frame number. **Description** : It will return RMSD values for particular atom framewise. atomName : It will hold the given atom for which RMSD value will be calculated. frameCount : It will store the number of frame in given dataframe.

3. MTA.RMSD.rmsdForResidueSequence (df : DataFrame, start : Int, end : Int, frameCount

: Int) : Array[String]

**Input** : Taking the pdb dataframe, start residue no., end residue no. **Output** : Array[String] : It will return an array of string, which contain RMSD value with given Residue Sequence. **Description** : It will return RMSD values for particular Residue Sequence. atomName : It will hold the given atom for which RMSD value will be calculated. frameCount : It will

16

store the number of frame in given dataframe.

4. MTA.RMSD.rmsdForParticularResidue(df : DataFrame, residueNum : Int, frameCount :

Int) : Array[String]

**Input** : Taking the pdb dataframe, residue no. **Output** : Array[String] : It will return an array of string, which contain RMSD value with frame number. **Description** : residueNum : It will store residue number for which RMSD value would be calculated. frameCount : It will store the number of frame in given dataframe.

**Sample Output** :-

**#Frame C\_RMSDValue\_000**

**03**

**CA\_RMSDValue\_0 0003**

**N\_RMSDValue\_00 002**

**N\_RMSDValue\_00 002**

**N\_RMSDValue\_00 002**

1 0.0000 0.0000 0.0000

2 0.6999 0.7098 0.6975

3 0.8023 0.8114 0.7972

4 0.8104 0.8191 0.8032

5 0.8960 0.9058 0.8927

6 0.9623 0.9736 0.9574

7 0.9782 0.9853 0.9706

8 0.9234 0.9326 0.9145

9 0.8276 0.8398 0.8228

10 0.8040 0.8144 0.7968

17

**DFD-**

18

**Average Structure**

Average structure is simply the average of co-ordinates visited during the trajectory of Atoms. We calculate the average of co-ordinates by getting the co-ordinates from the crd file.

**API’s Used:**

1. MTA.AverageStructure.averageStructure(topoDF : DataFrame, coordDF : DataFrame,

directoryName: String) : DataFrame

**Input:** topoDF and coordDF, Directory name. **Output:** Array of masked Data Frames **Description:** This function takes topoDF and coordDF and remove water from dataframes

generated by combining them.

**DFD-**

19

**Distance And Angle -**

In molecular geometry, bond length or bond distance is defined as the average distance between the nuclei of two bonded atoms in a molecule. Here we take coordinates of two atoms to calculate bond distance which further can be used by other modules. Bond angle is simply the angle between two bonds. And here we calculate the angle between the given bond which can be used by other modules.

**API’s Used:**

1. MTA.DistanceAndAngle.distanceBetweenAtoms(topoDF : DataFrame,coordDF :

Array[DataFrame],query: String): Unit

**INPUT**: topology dataframe, coordinate dataframe, Masking string. **OUTPUT**: Returns distance between atoms in Å. **DESCRIPTION**: Gets the filtered data by calling the masking module and gets the 3D coordinates of atoms from filtered data. Distance is calculated passing the 3D coordinates of atoms to calculateDistance function.

2. MTA.DistanceAndAngle.angleBetweenAtoms (topoDF : DataFrame, coordDF :

Array[DataFrame], query: String) : Unit

**INPUT**: topology dataframe, coordinate dataframe, Masking string.

**OUTPUT**: Returns bond angle between the bonds formed by three atoms in degree.

**DESCRIPTION**: Gets the filtered data by calling masking module and gets 3D coordinates of atoms from filtered data. Angle is calculated by passing the 3D coordinates of atoms to calculateAngle function.

3. MTA.DistanceAndAngle.distanceBetweenResidue (topoDF : DataFrame, coordDF :

Array[DataFrame], query: String): Unit

**INPUT**: topology dataframe, coordinate dataframe, Masking string.

**OUTPUT**: Returns distance between Residue in Å.

**DESCRIPTION**: calculates distance between atoms of two residues.

20

4. MTA.DistanceAndAngle.angleBetweenResidue (topoDF : DataFrame, coordDF :

Array[DataFrame], query: String): Unit

**INPUT**: topology dataframe, coordinate dataframe, Masking string.

**OUTPUT**: Returns bond angle between the bonds formed by three atoms in degree.

**DESCRIPTION**: calculates angle between atoms of two residues.

5. MTA.DistanceAndAngle.calculateDistance ( x1 : Double, y1: Double, z1 : Double, x2 :

Double, y2: Double, z2: Double): Double

**INPUT**: 3D coordinates of atoms.

**OUTPUT**: Returns distance between the bonds formed by three atoms in degree.

**DESCRIPTION**: Gets the filtered data by calling masking module and gets 2D coordinates of atoms from filtered data. Distance is calculated by passing the average of 2D coordinates of atoms to calculateAngle function.

6. MTA.DistanceAndAngle.calculateAngle(x1: Double, y1: Double, z1: Double, x2:

Double, y2: Double, z2: Double, x3: Double, y3: Double, z3: Double): Double

**INPUT**: 3D coordinates of atoms.

**OUTPUT**: Returns bond angle between the bonds formed by three atoms in degree.

**DESCRIPTION**: Gets the filtered data by calling masking module and gets 3D coordinates of atoms from filtered data. Angle is calculated by passing the average of 3D coordinates of atoms to calculateAngle function.

21

**Sample Output :-**

**#Frame Angle between Atom1, Atom2, Atom3**

**Distance**

**Angle between**

**between Atom1,**

**Res1, Res2, Atom2**

**Res3**

**Distance between Res1, Res2**

1 36.0360 1.0100 83.7311 3.8752

2 34.9455 1.0100 80.2145 3.8211

3 35.4726 1.0100 76.5408 3.7617

4 35.0164 1.0100 70.0013 4.1267

5 34.8741 1.0100 73.1486 3.9592

6 35.8628 1.0100 74.7676 3.9141

7 35.7157 1.0100 77.1429 3.9219

8 34.9283 1.0100 76.3094 3.8489

9 35.2580 1.0100 71.9832 4.0265

10 35.4929 1.0100 73.6441 3.9335

22

**DFD-**

23

**Dihedrals**

A dihedral angle is formed by three consecutive bonds in a molecule and defined by the angle created between the two outer bonds. The backbone of a protein has three different torsion angles.

These angles are called φ (phi) which involves the backbone atoms C-N-Cα-C, and ψ (psi) which involves the backbone atoms N-Cα-C-N, (ω) omega Cα-C-N-Cα.

**API’s Used:**

1. MTA.Dihedral.getBackboneDihedralAngle(topoDF: DataFrame, coordDF: DataFrame,

directoryName: String) : Dataset[String]

**Input**: Dataframes of PDB files, dataframes of coordinate files, directory name. **Output**: Dataset containing Dihedral angle between the selected atoms. **Description**: Using Vector of Vector of X,Y,Z co-ordinates of selected atoms, this will compute dihedral angle(PHI angle, PSI angle, OMEGA angle).

2. MTA.WriteDihedralAngles.writeDihedralAngle (dihedralArray : ArrayBuffer[String]) :

Unit

**Input**:Array containing information about all dihedral angles. **Output**: Dataset containing Dihedral angle between the selected atoms. **Description**: Using Vector of Vector of X,Y,Z co-ordinates of selected atoms, this will compute dihedral angle(PHI angle, PSI angle, OMEGA angle).

24

**Sample Output -**

**#Fram e**

**2**

**psi:1 phi:2 psi:2 omega:**

1 126.081

7

-145.994 8

-30.189 4

-30.189 4

-30.189 4

176.123 6

176.123 6

176.123 6

176.123 6

2

96.2694

-147.297

8 -9.0153

-171.60 73

-171.60 73

-171.60 73

-171.60 73

3 110.130

3

-142.362 8

-16.605 3

-16.605 3

-16.605 3

179.832 3

179.832 3

179.832 3

179.832 3

4 177.698

8

-141.922 1

-13.216 9

-13.216 9

-13.216 9

169.845 3

169.845 3

169.845 3

169.845 3

5 168.154

1

-135.648 2

-11.732 6

-11.732 6

-11.732 6

172.984 4

172.984 4

172.984 4

172.984 4

6 177.758

6

-137.165 0

-3.5000 179.461

-3.5000 179.461

0

0

7 -175.63

93

-145.711 2

-22.856 9

-22.856 9

-22.856 9

179.077 6

179.077 6

179.077 6

179.077 6

8 177.618

3

-142.635 1

-21.792 4

-21.792 4

-21.792 4

179.741 1

179.741 1

179.741 1

179.741 1

9 -178.09

15

-143.744 1

-11.215

-11.215

1 168.489

1 168.489

5

10

174.672 7

-138.044 3

-7.2591 176.289

-7.2591 176.289

2

2

25

**DFD-**

26

**Hydrogen bond**

Determine hydrogen bonds in each frame using geometric criteria: angle and distance. A hydrogen bond is defined as being between an acceptor A, hydrogen atom H and donor atom D. If the A to D distance is less than the distance cut off 3.0 A - 3.5 A and the A-H-D angle is equal to angle cut off 120o a hydrogen bond is considered formed.

Determination of hydrogen bond donors/acceptors uses the simplistic criterion that hydrogen bonds are ON, i.e., hydrogens bonded to O, and N atoms are considered donors, and O and N atoms are considered acceptors.

The number of hydrogen bonds present at each frame will be determined and written to the file specified by user. The time series for each hydrogen bond (1 for present, 0 for not present) will also be saved for subsequent analysis. Solute-solute hydrogen bonds will be saved to file name specified by the user. The atoms are written with the residue name, residue number and atoms involved in hydrogen bonds.

**API’s Used :**

1. MTA.HBond.list(distance: Double, angle:Double,outFileName: String)

**Input :** distance, angle, output file name

**Output:** File with name <outfileName> with hydrogen bond **Description:** Write the number of hydrogen bonds and its information such as occurrence , maximum resistance time and list of occurrence for all frame to <outFileName>.

**Sample Output -**

**#HBond #OCC #MRT #LIST**

GLN\_588@O-SER\_566@N 30 3 0100010010

ASP\_733@OD2-THR\_12@N 20 2

2110000000 0

VAL\_135@O-ARG\_138@N 30 3 3000001010

1

27

ASP\_693@O-THR\_697@OG1 100 10 1111111111

**DFD-**

28

**Auto-Imaging**

The first step is to find the centre of the box which can be done in different ways for example origin,reference etc.

Secondly it’s Imaging which is moving all the atoms’ coordinates in such a way that our anchor atom is at the centre of the box.

**API’s Used:**

1. MTA.AutoImaging.autoImaging(topoDF:DataFrame,coordDF:DataFrame,

cellLengthDF: DataFrame, directoryName : String) : Dataset[Row]

**Input:** coordinate dataframes,topology dataframe, cellLength dataframe and Directory name.

**Output**: Projection based on the dataset applied PDB. **Description:** The function calculates the average of non water coordinates and center is calculated.

**Sample Output -**

**#CRYST 1**

**Atom Numb er**

**Atom Name**

**Atom Name**

**Residu e Name**

**Residu e Name**

**Residu e Name**

**Residu e Numb er**

**Residu e Numb er**

**Residu e Numb er**

**Residu e Numb er**

**x-coor d**

**x-coor d**

**x-coor d**

**x-coor d**

**x-coor d**

**y-coor d**

**y-coor d**

**y-coor d**

**y-coor d**

**y-coor d**

**y-coor d**

**z-coor d**

**z-coor d**

**z-coor d**

**z-coor d**

**z-coor d**

**z-coor d**

**z-coor d**

ATOM 1 N ALA 1 82.419 56.509 76.041

ATOM 2 H1 ALA 1 82.465 56.214 75.077

ATOM 3 H2 ALA 1 82.654 55.717. 76.621

ATOM 4 H3 ALA 1 83.106 57.229 76.214

29

**DFD-**

30

**Essential Dynamics**

Essential Dynamics is all about analysing molecular structure and their interactions. So, we need to study about the internal motions of the molecule which shows the functional role of the molecule.

We analyse only the CA atoms from the given molecule. So firstly we need to mask CA atoms.

Initially, we calculate the covariance matrix of N atoms of atomic displacements which shows that most positional fluctuations are in subspace of very few degrees of freedom.

Secondly the matrix is diagonalized, in this process we get to know the eigen values. Next for each eigen value , eigen vectors are calculated. Later the projection is being shown in the form of graph.

**API’s Used:**

1. MTA.EssentialDynamics.getEiganValues(topoDF : DataFrame, coordDF:

Array[DataFrame]) : Unit

**Input:** Topology DataFrame, coordinate DataFrame Array **Output:** Covariance matrix, Eigen values, Eigen Vectors **Description:** This function generates the covariance matrix for all the C-alpha atoms, finds the eigen values accordingly. For each eigen value generates the eigen vectors which is used for projection.

2. MTA.EssentialDynamics.getEigVec(index : Int) : Unit

**Input:** index of eiganvalue corresponding **Output:** eigan vector corresponding to the eigan value of which we are passing index **Description:** This function generates the eigen vectors which is used for projection.

31

**DFD-**

32

**Integrated DFD**

contd..

33

34

**Class Diagram**

35

**6. Installation Instruction:**

● Source code is placed in **MTA** folder on local.

● run **sbt package** command and it will create the executable jar file.

● The created jar file will be in- **MTA/target/scala-2.11.12/mta\_2.11-0.1.jar**

● Now we can run the modules using **spark-submit** script. This script takes care of setting up the classpath with Spark and its dependencies, and can support different cluster managers and deploy modes that Spark supports:

○ --class: The entry point for your application

○ --master : The master URL for the cluster

○ --deploy-mode: Where to deploy your driver on the worker nodes

○ --driver-memory: used driver memory

○ --executor-memory: used memory for each executor

○ --executor-cores: number of executor-cores used

○ --num-executors: number of executors used

○ --queue: configuring queue for yarn capacity scheduler for running separate spark jobs

○ --jars: if there are any external jar files to use, we can mention them with this option

○ <executable jar>: here mention the executable jar file of application

○ <application arguments>: Arguments passed to the main method of main class, if any

**● Command:**

spark-submit --class class\_name --master cluster\_scheduler --deploy-mode cluster --driver-memory 100g --executor-memory 44g --executor-cores 6 --num-executors 20 --queue default executable\_jar\_name argument[1] argument[2] argument[3]

**here** -

class\_name = name of object which contains main class

Cluster\_scheduler = we are using yarn

deploy-mode = cluster

driver-memory = 100GB

executor-memory = 44GB

executor-cores = 6

executors = 20

executable\_jar\_name = name of executable jar file of application

36

argument[1] = path to topology file

argument[2] = path to text file containing paths of all coordinate input files

argument[3] = strings for calling different modules to be executed , like-

● AutoImaging - “autoimaging”

● Dihedral - “dihedral”

● Average Structure - “averagestructure”

● Distance - “distance dist\_between”

● Angle - “angle angle\_between”

● RMSD - “rmsd rmsd\_type”

rmsd\_type -

➔ framewise ➔ atom atom\_name ➔ residuerange range(seperated by ‘-’) ➔ residuenum num

● Essential Dynamics - “”

● Masking - “mask mask\_command”

● PDB generation - “writepdb”

**6. Software Validation:**

Testing shows the presence of errors. With the help of testing we can see the difference between our outcome and desired outcome which can be seen with the help of cpptraj. With the help of validation, we can be sure if we are on the correct path.

We can do various types of testing like beta testing, component testing, system testing, acceptance testing.

**7. Software Outcomes:**

Various measurements of Molecular trajectory analysis like Select, RMSD, AutoImaging, superImposition, Average Structure, Hydrogen Bond, Dihedrals, Distance and Angles, Essential Dynamics will be computed with the help of Spark and Scala.

37

**8. Benchmarks: MODULE Time(in Seconds/Frame)**

Angle 0.102 (1000 Frames)

AutoImaging 0.580 (1000 Frames)

Average PDB 0.287 (1000 Frames)

Dihedral 0.095 (1000 Frames)

Distance 0.980 (1000 Frames)

Hydrogen Bond -

Masking 0.120 (1000 Frames)

PDB Generation 1.237(1000 Frames)

Reading 0.190 (1000 Frames)

RMSD 0.152 (1000 Frames)

Essential Dynamics

**9. Specifications:**

Driver Memory 200 Gb

Executor Memory 50 Gb

Executor Core 20

Number of executors 6

38

**10. Group Members:**

**Angle** Ankita Hariyale(MIT2019002)

Fanishwar Satyam(MIT2019003) Gaurav Rajput(MIT2019023) Himanshu (MIT2019071)

**AutoImaging** Prashant kushwaha(MIT2019004)

Abhinav Anand(MIT2019005) Shubham kewat(MIT2019054) Rohit Kumar(MIT2019114)

**Average PDB** Sonu Kumar(MIT2019065)

Harish kumar(MIT2019112) Sudhanshu Ranjan(MIT2019121)

**Design** Kavita Singh Dhruv(MIT2019009)

Komal Garg(MIT2019042) Mayank Kabra(MIT2019051) P.N.Janani(MIT2019123)

**Dihedral** Bishwajit Majumdar(MIT2019020)

Shivam Kasat(MIT2019024) Koushal Kumar Sharma(MIT2019066)

**Distance** Ankita Hariyale(MIT2019002)

Fanishwar Satyam(MIT2019003) Gaurav Rajput(MIT2019023) Himanshu (MIT2019071)

**H-Bond** Hemant Kumar Lader(MIT2019007)

Gaurav Rajput(MIT2019023) Shekhar Gupta(MIT2019052) Himanshu(MIT2019071)

**Integration** Hemant Kumar Lader(MIT2019007)

Mayank Kabra(MIT2019051) Yempalli Siva Kumar(MIT2019056)

**Manual** Kavita Singh Dhruv(MIT2019009)

Komal Garg(MIT2019042) P.N.Janani(MIT2019123)

**Masking** Hemant Kumar Lader(MIT2019007)

Pragati Pawar(MIT2019017) Mani Shankar Javvaji(MIT2019055)

39

Yempalli Siva Kumar(MIT2019056) Khushbu Mevada(MIT2019086)

**PDB Generation** Hemant Kumar Lader(MIT2019007)

Mayank Kabra(MIT2019051) Mani Shankar Javvaji(MIT2019055) Yempalli Siva Kumar(MIT2019056)

**Reading** Hemant Kumar Lader(MIT2019007)

Mayank Kabra(MIT2019051) Shekhar Gupta(MIT2019052) Mani Shankar Javvaji(MIT2019055) Yempalli Siva Kumar(MIT2019056)

**RMSD** Riya Khanna(MIT20190085)

Aadil Ali(MIT2019087) Ajay Kumar(MIT2019089) Praveen Kumar Raghav(MIT2019097) Sumit Kumar(MIT2019117)

**Testing** Hemant Kumar Lader(MIT2019007)

Mayank Kabra(MIT2019051) Yempalli Siva Kumar(MIT2019056)

**Essential Dynamics** Deepali Jindal(MIT2019006) Pragati Pawar(MIT2019017) Komal Garg(MIT2019042) Mani Shankar Javvaji(MIT2019055) Khushbu Mevada(MIT2019086)

40

**BIBLIOGRAPHY**

1. Amber 18 Manual: http://gohom.win/ManualHom/Amber/amber8.pdf

2. CPPTRAJ: https://amber-md.github.io/cpptraj/CPPTRAJ.xhtml

3. Spark Documentation: https://spark.apache.org/docs/latest/

4. Learning Spark Lightning-Fast Big Data Analysis .pdf : Holden Karau, Andy Konwinski,

Patrick Wendell & Matei Zaharia

5. Scala Documentation: https://docs.scala-lang.org

6. scala-impatient.pdf : Cay Hortsmann

7. Kabsch Algorithm: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2828896/

8. Hydrogen Bonds in Proteins: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4005716/

9. Essential Dynamics of Proteins:

https://homepage.univie.ac.at/mario.barbatti/papers/method/essential\_dynamics\_1993.pdf\

10. Dihedral Angles in Proteins:

https://proteinstructures.com/Structure/Structure/Ramachandran-plot.html

41