Non-bonded interactions energy computation: CPU and parallelized GPU implementation

Seyed Shahriar Arab[١], Alireza Doostmohammdi[٢], Seyed Ali Mirmostafa[٣], Maryam Rafieipour[٤], Niloufar Seyed Majidi[٥], Rayeheh Vafaiee[٦]

Abstract

The folding of proteins is one of the biggest problems in every field of science related to Biochemistry. Before entering water solutes, proteins do not have a three-dimensional structure. After entering water solutes due to emerging forces between the solute and the protein as well as interactions inside the protein gives the linear protein a three-dimensional structure which is very important and fundamental to their functions inside a living organism. Non-bonded interactions are one of the most influential factors to protein folding. Which comprise of three major interaction: Vander Waals interactions, electrostatic interactions and hydrogen bonds. Our goal in this project is to compute all of the interactions of these kinds inside a PDB file and parallelize these computations using GPU programming. For this purpose, we use a parallel computing platform called CUDA [١]. We use python Numba package to parallelize computations. In the end, we compare the computation time of the CPU program and GPU program. This comparison shows that increasing number of computation, CPU time exponentially increases and GPU time linearly increases and GPU Time is lower than CPU time.

Keywords: Non-bonded interactions, Hydrogen Bond, Electrostatic, Van der Waals, parallelize computations

Introduction

The folding of proteins is one of the biggest problems in every field of science related to Biochemistry. Before entering water solutes, proteins do not have a three-dimensional structure. After entering water solutes due to emerging forces between the solute and the protein as well as interactions inside fundamental to their functions inside a living organism. Non-bonded interactions are one of the most

[١] Assistant Professor of Tarbiat modares University, Department of Biophysics
[٢] M.Sc. student at Bioinformatics laboratory of Tarbiat modares University
[٣] M.Sc. student at Bioinformatics laboratory of Tarbiat modares University
[٤] M.Sc. student at Bioinformatics laboratory of Tarbiat modares University
[٥] M.Sc. student at Bioinformatics laboratory of Tarbiat modares University
[٦] Ph.D. student at Bioinformatics laboratory of Tarbiat modares University

influential factors to protein folding. Non-bonded interactions act between atoms which are not linked by covalent bonds. Force fields usually divide non-bonded interactions into three: Van der Waals interactions, electrostatic interactions and Hydrogen bond.

Van der Waals interactions

The interactions between electrons and nuclei, whether they are or not part of the same molecule, are electromagnetic. Van der Waals forces include attraction and repulsions between atoms, molecules, and surfaces, as well as other intermolecular forces. They differ from covalent and ionic bonding in that they are caused by correlations in the fluctuating polarizations of nearby particles. [١]There are different formulas for computing van der Waals energy between two atoms. In this project, the Lennard-Jones potential equation is used. [٢]

The Lennard-Jones is a mathematically simple model that approximates the interaction between a pair of neutral atoms or molecules by the following formula.

$$v_{LJ} = ٤\,\varepsilon\left[\left(\frac{\sigma}{r}\right)^{١٢} - \left(\frac{\sigma}{r}\right)^{٦}\right] \qquad \text{(Equation ١)}$$

Where ε is the depth of the potential well, σ is the finite distance at which the inter-particle potential is zero, r is the distance between the particles. [٣]

According to Lennard-Jones plot, in the range ٠,٣٥ to ١ nm van der Waals interaction is attraction. (fig.١)

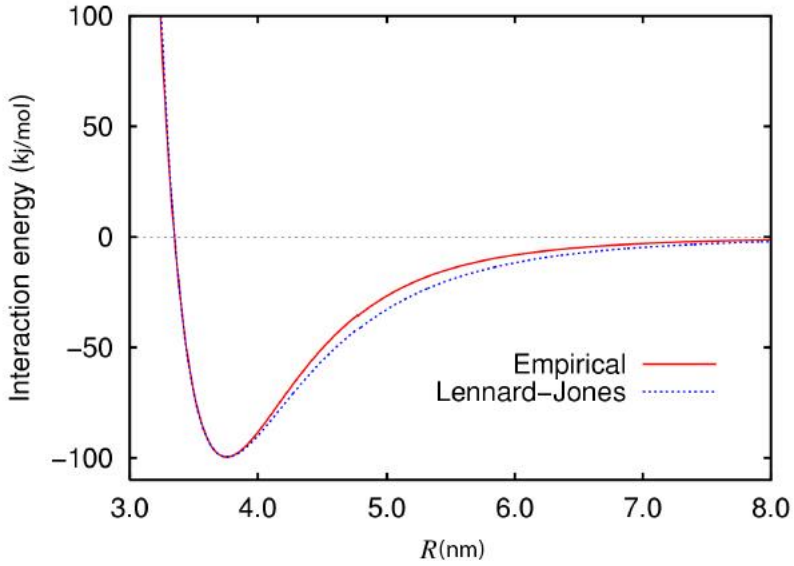


Figure ١: Van der Waals interaction base on Lennard-Jones Formula

Hydrogen Bond Interaction

A hydrogen bond is a partial intermolecular bonding interaction between a lone pair on an electron-rich donor atom, particularly the second-row elements nitrogen (N), oxygen (O), or fluorine (F), and the antibonding orbital of a bond between hydrogen (H) and a more electronegative atom or group. [٤]

Most force fields consider hydrogen bond as a part of van der Waals interaction. In this project, the following equation is used for computing hydrogen bond energy.

$$٤\ \varepsilon\left[\left(\frac{\sigma}{r}\right)^{١٢}-\left(\frac{\sigma}{r}\right)^{٦}\right]\cos(\theta) \quad \text{(Equation٢)}$$

Where ε is the depth of the potential well, σ is the finite distance at which the inter-particle potential is zero, r is the distance between the particles, θ is the angle between donor and acceptor. According to Gromacs documentation hydrogen bond is formed in distance less than ٠,٣٥ nm and angle less than ٣٠ degrees. [٥]

Electrostatic Interaction

The electrostatic interactions are the electric force between any two charged molecules and can be attractive or repulsive. The electrostatic interaction energy is computed by the following formula.

$$E = k\frac{q_١ q_٢}{r} \quad \text{(Equation٣)}$$

Where k is coulomb constant and equal to ٣٣٢.٠٦٤١ $\frac{Kcal.A}{mol.e^{٢}}$, $q_١$ is the charge of first particle, $q_٢$ is the charge of second particle and r is the distance between the particles [٦].Electrostatic interactions is formed in distance ٩ to ١٢ angstrom [٧].

Our goal in this project is to compute all of the interactions of these kinds inside a PDB file. A PDB file is a piece of information about all of the atoms of a protein and their spatial coordinates. Using these coordinates and information we can compute these energies. Force fields are different simulations of these energies and their parameters. We start by implementing the GROMOS force field and using the exclusive parameters presented by this force field. After implementing this force field in a C++ program our second goal is to parallelize these computations using GPU programming. For this purpose, we use a parallel computing platform called CUDA. CUDA is a parallel computing platform and API [١] model created by NVidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit for general purpose processing – an approach termed GPGPU [٢]. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. Due to limitations of programming in C++, we used python Numba package. Numba is an open-source JIT compiler that translates a subset of Python and NumPy into fast machine code using LLVM, via the llvmlite Python package. It offers a range of options for parallelizing Python code for CPUs and GPUs, often with only minor code changes. Numba also supports CUDA programming which makes it a very suitable package for our purpose. Numba makes use of Numpy python package for many of its functionalities. NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. In the end, we compare the computation time of the CPU program and GPU program.

Material and methods

The input file of our software is GRO file. Some proteins structure determined with X-ray crystallography and hydrogens remove from molecules in this method. If the protein structure determined with X-ray crystallography, we should add hydrogen to protein atoms. To do this we can use some software such as GROMACS or MolProbity server. In this project, we use GROMCS software to add hydrogens. GROMACS add hydrogens command is:

$$gmx\ pdb2gmx\ -f\ protein.pdb\ -o\ protein.gro\ -water\ spce$$

Force fields are different simulations of Non-bonded energies and their parameters. We start by implementing the GROMOS force field and using the exclusive parameters presented by this force field. *aminoacids.rtp* file (fig.٢) consist of rename atoms and charge atoms information. This information is useful to compute electrostatic interactions, Van der Waals and hydrogen bonds parameters.

```
[ bondedtypes ]
; bonds  angles  dihedrals  impropers
    2       2          1          2

[ ALA ]
 [ atoms ]
    N      N     -0.31000     0
    H      H      0.31000     0
   CA    CH1      0.00000     1
   CB    CH3      0.00000     1
    C      C        0.450     2
    O      O       -0.450     2
```

Figure ٢: aminoacids.rtp file. First and second columns show the amino acid atom's name and name of them on the GROMOS force field. The third column shows the atom's charge.

ffnonbonded.itp is another force field file (fig.٣) that consist of Van der Waals and hydrogen bond σ *and* ε *parameters*.

```
[ atomtypes ]
;name  at.num    mass      charge  ptype       c6           c12
    O     8     0.000      0.000     A  0.0022619536       1e-06
   OM     8     0.000      0.000     A  0.0022619536  7.4149321e-07
   OA     8     0.000      0.000     A  0.0022619536  1.505529e-06
   OE     8     0.000      0.000     A  0.0022619536    1.21e-06
   OW     8     0.000      0.000     A  0.0026173456  2.634129e-06
    N     7     0.000      0.000     A  0.0024364096  2.319529e-06
   NT     7     0.000      0.000     A  0.0024364096  5.0625e-06
   NL     7     0.000      0.000     A  0.0024364096  2.319529e-06
   NR     7     0.000      0.000     A  0.0024364096  3.389281e-06
   NZ     7     0.000      0.000     A  0.0024364096  2.319529e-06
   NE     7     0.000      0.000     A  0.0024364096  2.319529e-06
    C     6     0.000      0.000     A  0.0023406244  4.937284e-06
  CH0     6     0.000      0.000     A  0.0023970816  0.0002053489
  CH1     6     0.000      0.000     A  0.00606841  9.70225e-05
  CH2     6     0.000      0.000     A  0.0074684164  3.3965584e-05
  CH3     6     0.000      0.000     A  0.0096138025  2.6646244e-05
  CH4     6     0.000      0.000     A  0.01317904  3.4363044e-05
 CH2r     6     0.000      0.000     A  0.0073342096  2.8058209e-05
  CR1     6     0.000      0.000     A  0.0055130625  1.5116544e-05
   HC     1     0.000      0.000     A   8.464e-05  1.5129e-08
    H     1     0.000      0.000     A           0           0
  DUM     0     0.000      0.000     A           0           0
    S    16     0.000      0.000     A  0.0099840064  1.3075456e-05
 CU1+    29     0.000      0.000     A  0.0004182025  5.1251281e-09
 CU2+    29     0.000      0.000     A  0.0004182025  5.1251281e-09
   FE    26     0.000      0.000     A           0           0
 ZN2+    30     0.000      0.000     A  0.0004182025  9.4400656e-09
 MG2+    12     0.000      0.000     A  6.52864e-05  3.4082244e-09
 CA2+    20     0.000      0.000     A  0.00100489  4.9801249e-07
    P    15     0.000      0.000     A  0.01473796  2.2193521e-05
   AR    18     0.000      0.000     A  0.0062647225  9.847044e-06
    F     9     0.000      0.000     A  0.0011778624  7.6073284e-07
```

```
[ nonbond_params ]
;   i    j   func      c6          c12
   OM    O    1   2.261954E-03   8.611000E-07
   OA    O    1   2.261954E-03   1.386510E-06
   OA   OM    1   2.261954E-03   2.258907E-06
   OE    O    1   2.261954E-03   1.100000E-06
   OE   OM    1   2.261954E-03   9.472100E-07
   OE   OA    1   2.261954E-03   1.505529E-06
   OW    O    1   2.433170E-03   1.833990E-06
   OW   OM    1   2.433170E-03   2.987943E-06
   OW   OA    1   2.433170E-03   1.991421E-06
   OW   OE    1   2.433170E-03   1.991421E-06
    N    O    1   2.347562e-03   1.943000e-06
    N   OM    1   2.347562E-03   3.577063E-06
    N   OA    1   2.347562E-03   2.384061E-06
    N   OE    1   2.347562E-03   2.384061E-06
    N   OW    1   2.525258E-03   3.153489E-06
   NT    O    1   2.347562E-03   2.542500E-06
   NT   OM    1   2.347562E-03   4.142250E-06
   NT   OA    1   2.347562E-03   2.760750E-06
   NT   OE    1   2.347562E-03   2.760750E-06
   NT   OW    1   2.525258E-03   3.651750E-06
   NT    N    1   2.436410E-03   4.371750E-06
   NL    O    1   2.347562E-03   3.466840E-06
   NL   OM    1   2.347562E-03   9.412624E-06
   NL   OA    1   2.347562E-03   3.764436E-06
   NL   OE    1   2.347562E-03   3.764436E-06
   NL   OW    1   2.525258E-03   4.979364E-06
   NL    N    1   2.436410E-03   2.319529E-06
```

Figure٣: ffnonbonded.itp file contains three parts: atomtypes, nonbond_params and pairtypes. When we compute the interactions of the same atoms, atomtypes part is useful and we use nonbonded_params part in other states. This file reports $c^{٦}$ and $c^{١٢}$. These parameters use to compute Van der Waals and hydrogen bonds interactions.

The software flowchart (fig.٤) to compute Non-bonded interactions is:

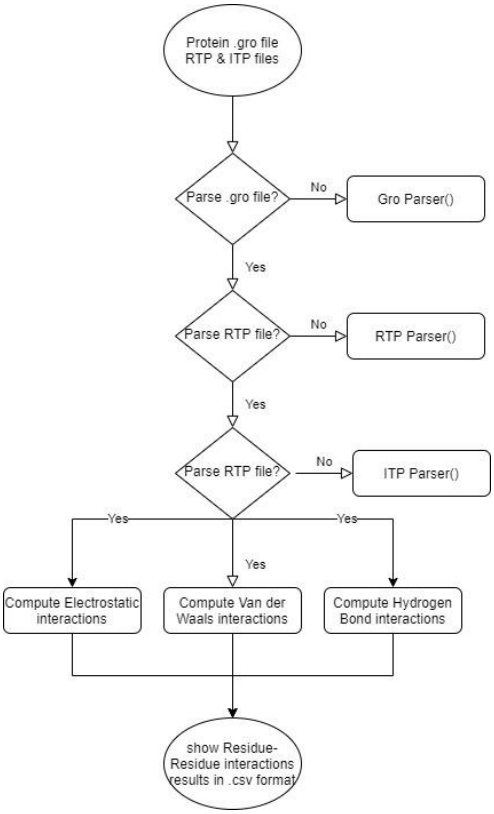


Figure٤: The software flowchart. The input files are Gro, ITP and RTP files. We parse input files and compute Non-bonded interactions and show residue-residue interactions in CSV format.

The input files are GRO, RTP and ITP files. First of all, we should parse the input files. To validate our software, we use the protein with ١٢as PDB ID[٧]. This protein has two chains, ٣٣٠ residues and ٦٠٥٢ atoms.

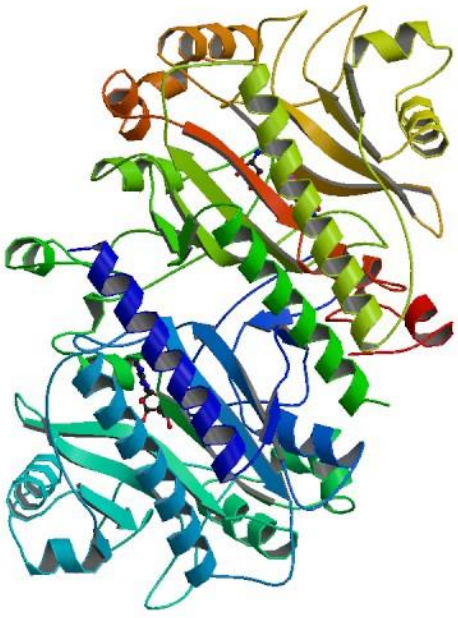
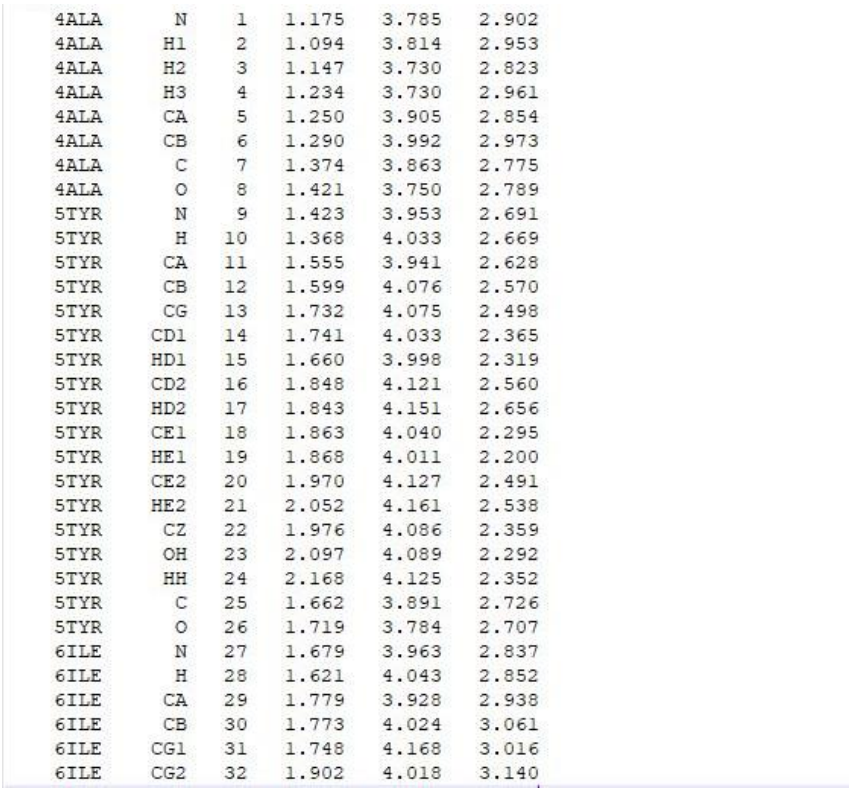


Figure ٥: ١٢ as protein structure

[٧] Asparagine synthetase mutant C١٥A, C٣١٥A complexed with L-Asparagine an AMP

The GRO file contains (fig.٦) residue index, residue name, atom name, atom index and atom coordinates. We store these information in seven vectors. GROMACS report atoms coordinates in nanometers unit.



4ALA	N	1	1.175	3.785	2.902
4ALA	H1	2	1.094	3.814	2.953
4ALA	H2	3	1.147	3.730	2.823
4ALA	H3	4	1.234	3.730	2.961
4ALA	CA	5	1.250	3.905	2.854
4ALA	CB	6	1.290	3.992	2.973
4ALA	C	7	1.374	3.863	2.775
4ALA	O	8	1.421	3.750	2.789
5TYR	N	9	1.423	3.953	2.691
5TYR	H	10	1.368	4.033	2.669
5TYR	CA	11	1.555	3.941	2.628
5TYR	CB	12	1.599	4.076	2.570
5TYR	CG	13	1.732	4.075	2.498
5TYR	CD1	14	1.741	4.033	2.365
5TYR	HD1	15	1.660	3.998	2.319
5TYR	CD2	16	1.848	4.121	2.560
5TYR	HD2	17	1.843	4.151	2.656
5TYR	CE1	18	1.863	4.040	2.295
5TYR	HE1	19	1.868	4.011	2.200
5TYR	CE2	20	1.970	4.127	2.491
5TYR	HE2	21	2.052	4.161	2.538
5TYR	CZ	22	1.976	4.086	2.359
5TYR	OH	23	2.097	4.089	2.292
5TYR	HH	24	2.168	4.125	2.352
5TYR	C	25	1.662	3.891	2.726
5TYR	O	26	1.719	3.784	2.707
6ILE	N	27	1.679	3.963	2.837
6ILE	H	28	1.621	4.043	2.852
6ILE	CA	29	1.779	3.928	2.938
6ILE	CB	30	1.773	4.024	3.061
6ILE	CG1	31	1.748	4.168	3.016
6ILE	CG2	32	1.902	4.018	3.140

Figure٦: ١٢as.*gro* file. It contains residue index, residue name, atom name, atom index and atom coordinate.

After GRO file is parsed, ITP and RTP are needed to be parsed. We use the map data structure to store residue, rename atoms and atom charge of RTP file, $C6$ and $C12$ parameters of ITP file.

```
ITP Parser(itp file):
    define map_atomtype class.class members are at_num,mass,charge and ptype
    map<string,map<string,int>> mapitp;
    map<string,int> map_atomtype_name;
    double atomtype_val[58][2];
    double nonbond_val[1485][2];
   Line = Read lines.
   If (line = 'atomtypes'){
        While(ldata != "["){
            map_atomtype_name.insert({name, indx + ١});
        atomtypes.at_num = at.num;
        atomtypes.mass = mass;
        atomtypes.charge = charge;
            atomtypes.ptype = ptype;
        atomtype_val[indx][٠] = c6;
        atomtype_val[indx][١] = c12;
            indx+= ١;
      }
   }
   indx = ٠;
   If (line = 'nonbond_params'){
        While(data != "["){
```

```
                mapitp.insert(make_pair(nonbond.i, map < string, int > ()));
            mapitp[nonbond.i].insert(make_pair(nonbond.j, indx + ١));
            nonbond_name[indx][٠] = nonbond.i;
            nonbond_name[indx][١] = nonbond.j;
                nonbond_val[indx][٠] = nonbond.c6;
            nonbond_val[indx][١] = nonbond.c12;
             indx+= ١;
        }
    }
}
```

```
RTP Parser(Rtp file):
    map<string,map<string,string>> mapreplace;
    map<string,map<string,double>> mapreplace_es;
   Line = Read lines.
   If (line = 'atomtypes'){
        While(ldata ! =  "["){
                Read lines;
        }else{
          if(data is residue name or HISA or HISB or HISH or HIS1 or HIS2{
        while(data ! =  "["){
            mapreplace.insert(make_pair(amino_acide,map<string,string>()));
            mapreplace[amino_acide].insert(make_pair
                                    (rtp_rename.i,rtp_rename.j));
          mapreplace_es.insert(make_pair(amino_acide,map<string,double>()));
          mapreplace_es[amino_acide].insert(make_pair
                                    (rtp_rename.i,rtp_rename.charge+٢));
        }
      }
    }
  }
}
```

We store $charge + ٢$ in map because electrostatic atom charge detector function condition is based on identify zero value.

After parsing input files, we computing non-bonded interactions same as below algorithms:

Computing Electrostatic Interactions:

The electrostatic interaction energy is computed by the following formula.

$$E = k\frac{q_١ q_٢}{r} \quad (٣)$$

Where k is coulomb constant and equal to $٣٣٢.٠٦٤١\ \frac{Kcal.A}{mol.e^٢}$, $q_١$ is the charge of the first particle, $q_٢$ is the charge of the second particle and r is the distance between the particles [٦]. Electrostatic interaction is formed in distance ٩ to ١٢ angstrom [٧]. We compute atoms distance and if the distance is in the range of ٩ to ١٢ angstrom, find atoms charge in RTP file and compute Electrostatic interaction. To report the electrostatic energy between two atoms in kj/mol unit we divide k to ٤٨١,. Also, the distance should be divided to ١٠ to change the distance's unit from nm to A. The charge's unit in RTP file is

electron. Thus the formula is used for computing the electrostatic energy is $٧.٣٦٩\frac{q_١ q_٢}{r}$, where $٧.٣٦٩ = \frac{٣٣٢.٠٦٤١}{٤.٨١\times١٠}$. We implement the below algorithm to compute electrostatic interactions:

```
atomes_charge(residuei,string residuej,string ati, string atj){

        double atom_i_charge=٠,atom_j_charge=٠;

        while(atom_i_charge == ٠){
                atom_i_charge=mapreplace_ES[residuei][ati];
                        ati.pop_back();
        }
        while(atom_j_charge ==٠){
                atom_j_charge=mapreplace_ES[residuej][atj];
                        atj.pop_back();
        }
                charge[٠]=atom_i_charge-٢,٠;
                charge[١]=atom_j_charge-٢,٠;

                return charge;
}

Electrostatic(residue indx,vector, residue name vector, atom,coordinates, mapreplace_es){

    for (i in range(٠:atom number)){
                for(j in range(i+١:atom numbers)){
                        dis=compute distance between atom i and atom j;
                                if(dis > ٩ and dis <١٢){
                                        atomes_charge(ith atom's residue, ith atom ,jth atom's
                                                residue, jth atom  );
                                        val=٧,٩٣٦* charge[٠]*charge[١]/dis;
                                        save val,residues index and name and atoms name in
                                    vectors;
                                        save uniq residue index pair in pairs map;
                                    }
                                }
                    }
}
```

Computing Van der Waals interactions:

There are different formulas for computing van der Waals energy between two atoms. In this project, the Lennard-Jones potential equation is used. [٢]

The Lennard-Jones is a mathematically simple model that approximates the interaction between a pair of neutral atoms or molecules by the following formula.

$$v_{LJ} = ٤\,\varepsilon\left[\left(\frac{\sigma}{r}\right)^{١٢} - \left(\frac{\sigma}{r}\right)^{٦}\right] \qquad (١)$$

Where ε is the depth of the potential well, σ is the finite distance at which the inter-particle potential is zero, r is the distance between the particles. [٣]

According to Lennard-Jones plot, in the range ٠,٣٥ to ١ nm van der Waals interaction is attraction. (fig.١)

We compute atoms distance and if the distance is in ٠٫٣٥ to ١ nm range, find the name of atoms in RTP file and atom pairs $C6$ and $C12$ parameters in ITP file and compute Van der Waals interaction. We implement the below algorithm to compute Van der Waals interactions:

```
atome_pairs_parameters(residuei,string residuej,string ati, string atj){

        int indx=٠;
        string atom_i="",atom_j="";

         while(atom_i == ""){
                        atom_i=mapreplace_V[residuei][ati];
                        ati.pop_back();
         }
         while(atom_j ==""){
                        atom_j=mapreplace_V[residuej][atj];
                        atj.pop_back();
         }

        if(atom_i != atom_j){
                indx=mapitp_V[atom_i][atom_j];
                if(indx==٠){
                        indx=mapitp_V[atom_j][atom_i];
                }
                                coef_V[٠]=nonbond_val_V[indx-١][٠];
                                coef_V[١]=nonbond_val_V[indx-١][١];
        }else{
                indx= map_atomtype_name_V[atom_i];
                coef_V[٠]=atomtype_val_V[indx-١][٠];
                coef_V[١]=atomtype_val_V[indx-١][١];
        }
        return coef_V;
}

VanderWaals(residue indx vector,residue name vector,atom name vector,atom coordinates,
                        nonbond_val[][٢],atomtype_val[][٢],mapreplace,mapitp) {

      for (i in range(٠:atom number)){
                for(j in range(i+١:atom numbers)){
                        dis=compute distance between atom i and atom j;
                        if(dis > ٠٫٣٥ and dis <١){
                                atome_pairs_parameters(ith atom's residue, ith atom ,jth atom's
                                                residue, jth atom  );
                                val=((coef_V[١]/pow(dis,١٢))-(coef_V[٠]/pow(dis,٦)));
                                save val,residues index and name and atoms name in vectors;
                                save uniq residue index pair in pairs map;
                        }
                }
         }
}
```

Computing Hydrogen Bonds interactions :

Most force fields consider hydrogen bond as a part of van der Waals interaction. In this project, the following equation is used for computing hydrogen bond energy.

$$٤\ \varepsilon\left[\left(\frac{\sigma}{r}\right)^{١٢}-\left(\frac{\sigma}{r}\right)^{٦}\right]\cos(\theta) \qquad (٢)$$

Where ε is the depth of the potential well, σ is the finite distance at which the inter-particle potential is zero, r is the distance between the particles, θ is the angle between donor and acceptor. According to Gromacs documentation hydrogen bond is formed in distance less than ٠,٣٥ nm and angle less than ٣٠ degrees. [٥] We find all Nitrogen and Oxygen atoms in the protein and save these atoms index in vector. In another vector, we store Oxygen or Nitrogen atoms bond with Hydrogen and compute the distance between donor and acceptor and if the distance is lower than ٠,٣٥ nm, compute the angle between donor and acceptor and if degree is lower than ٣٠ degrees (٠,٥٢ radian), find the name of atoms in RTP file and atom pairs $C6$ and $C12$ parameters in ITP file and compute Hydrogen Bond interaction energy. We implement the below algorithm to compute Hydrogen Bond interactions:

```
atome_pairs_parameters(residuei,string residuej,string ati, string atj){

{
        int indx=٠;
        string atom_i="",atom_j="";

         while(atom_i == ""){
                        atom_i=mapreplace_V[residuei][ati];
                        ati.pop_back();
         }
         while(atom_j ==""){
                        atom_j=mapreplace_V[residuej][atj];
                        atj.pop_back();
         }

        if(atom_i != atom_j){
                indx=mapitp_V[atom_i][atom_j];
                if(indx==٠){
                        indx=mapitp_V[atom_j][atom_i];
                }
                                coef_V[٠]=nonbond_val_V[indx-١][٠];
                                coef_V[١]=nonbond_val_V[indx-١][١];
        }else{
                indx= map_atomtype_name_V[atom_i];
                coef_V[٠]=atomtype_val_V[indx-١][٠];
                coef_V[١]=atomtype_val_V[indx-١][١];
        }
        return coef_V;
}

Hydrogenbond(residue indx vector,residue name vector,atom name vector,atom coordinates,

        nonbond_val[][٢],atomtype_val[][٢],mapreplace,mapitp) {

                 int indx_flag=٠;
                 int residue_flag=ith atoms'residue index;
                double degree,p١٢,p٢٣,p١٣;

                for(i in  range(٠ to atom num)){
                        if(residue_flag != ith atom's residue index){
                                indx_flag=i;
                                residue_flag=ith atom's residue index;
                        }
                        if(atom.at(i)[٠]=='O' || atom.at(i)[٠]=='N'){
                                save 'O' and 'N' index and residue index in indx_all_ON and
                                        resNum_all vector;
```

```
                                int j=٠;
                                while(kth atoms residue index !=residue_flag){
                                        if(atom k[٠]=='H'){
                                                dis=compute distance between atom i and atom k;

                                                if(atom i[٠]=='O' && dis<٠,١٤٥٥){
                                                        save 'O' and 'H' index and residue index in
                                                         indx_ON,indx_hydrogen and
                                                                resNum vector ;
                                                }else if(atom i[٠]=='N' && dis<٠,١٥٣){
                                                        save 'N' and 'H' index and residue index in
                                                          indx_ON,indx_hydrogen and
                                                          resNum vector ;
                                                }
                                        }
                                         if(k == atom number-١) break;
                                        k++;
                                }

        for (i in range(٠:indx_ON)){
            for(j in range(٠:indx_all_ON numbers)){
                dis=compute distance between atom i and atom j;
               if(dis <٠,٣٥){
                    p١٢=compute ith atom in indx_ON vector and ith hydrogen;
                     p٢٣=compute jth atom in indx_all__ON vector and ith hydrogen;
                     p١٣=compute jth atom in indx_all__ON vector and ith atom in indx_ON vector;

                     degree=arccos((P١٢^٢ + P١٣^٢ - P٢٣^٢) / (٢ * P١٢ * P١٣));
                     if(degree<٠,٥٢){
                             atome_pairs_parameters(ith atom's residue, ith atom ,jth atom's residue,
                                                                                     jth atom  );
                               val=((coef_V[١]/pow(dis,١٢))-(coef_V[٠]/pow(dis,٦)))*cos(degree);
                               save val,residues index and name and atoms name in vectors;
                               save uniq residue index pair in pairs map;
                                                }
                                        }
                                }
                        }
                }
}
```

These algorithms compute atom-atom interactions. To show these results in residue-residue format, we implement the below algorithm:

```
residue-residue_results(){

    vector<string> final_residue_i,final_residue_j;
    vector<int> final_residuei_indx,final_residuej_indx;
    vector<double>sum_val;

    for(int i in range (٠ to interactions value numbers){
            sum_value[ith indx_ON atom's residue index]
                    [ith indx_Oall_N atoms residue index]+= ith interacion value;
            final_residue_i[ith indx_ON atom's residue index]
                    [ith indx_Oall_N atoms residue index]=ith indx_ON atom's residue;
            final_residue_j[ith indx_ON atom's residue index]
                    [ith indx_Oall_N atoms residue index]=ith indx_Oall_N atoms residue;
            final_residuei_indx[ith indx_ON atom's residue index]
                    [ith indx_Oall_N atoms residue index]=ith indx_ON atom's residue index;
             final_residuej_indx[ith indx_ON atom's residue index]
```

```
                    [ith indx_Oall_N atoms residue index]=ith indx_Oall_N atoms residue index;
        }

        save vectors in .csv file;
}
```

This session is CPU algorithm bottleneck. To solve this problem, we parallelize this session on GPU.

Parallelizing Residue by Residue energies on GPU

By using GPU we obtain the ability to parallelize our algorithm on the condition that our algorithm could be parallelized. So the first step is to determine if our algorithm could be parallelized and if so how should we parallelize it so that we harness the most computational power from our GPU. In spite of providing us with parallel computation, GPU programming has its constraints. Like CPU, GPU has its own memory to store the information and using CPU memory while computing in GPU is not advised due to the access to CPU memory from GPU being very slow and time consuming. So one of the most important stages to compute on GPU is to transfer all of the information and data that the GPU might need during the computation to the GPU memory. (Figure ٦)

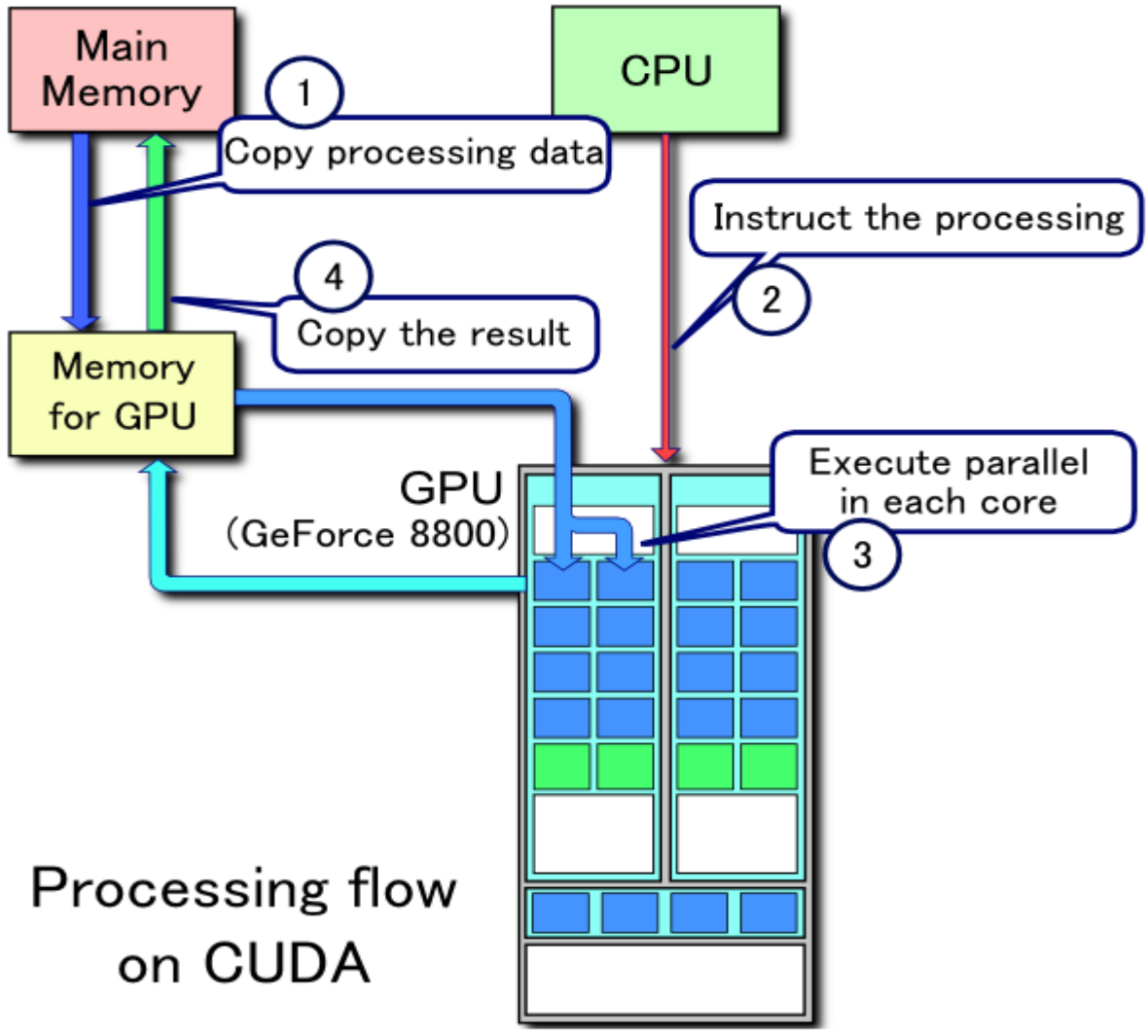


Figure ٧: Architecture of CUDA program

This stage is very time consuming but with enough arrangements we can accomplish this stage so that the data would only be transferred once from CPU to GPU for computation and once from GPU to CPU for saving and documenting. GPU is not good at reading from files or writing to files so because of the hurdles mentioned above we did many of the part of preparing the data inside our CPU program and we transferred the data that was needed for the final computation to our GPU program. For the purpose of GPU programming we used python Numba package which is a just in time compiler for python. This package enables us to transfer data to GPU and from GPU as well as writing GPU kernels that would run on GPU threads. The architecture of GPU is different from CPU. CPU has few cores with high speed and complex calculations and a memory unit for storing the data. GPU has huge number of processing cores which are called CUDA cores. Different GPUs have different number of CUDA cores. Our machine GPU model is Nvidia GTX ٩٨٠m which is the laptop version of Nvidia GTX ٩٨٠. The number of CUDA cores on our GPU is the same as the number of CUDA cores on the desktop version of this graphic card which is ٢٠٤٨ CUDA cores.

Below is the program for transferring the data.

```
#vanderwaals

stream = cuda.stream()
energies_device = cuda.to_device(energies, stream=stream)
residue_num_i_device = cuda.to_device(residue_num_i,stream=stream)
residue_num_j_device = cuda.to_device(residue_num_j,stream=stream)
out_device = cuda.to_device(np.zeros((res_num, res_num), dtype= np.float٣٢),stream=stream)
sizes_device = cuda.to_device(sizes, stream=stream)

#electrostatic

stream = cuda.stream()
energies_device = cuda.to_device(energies, stream=stream)
residue_num_i_device = cuda.to_device(residue_num_i,stream=stream)
residue_num_j_device = cuda.to_device(residue_num_j,stream=stream)
out_device = cuda.to_device(np.zeros((res_num, res_num), dtype= np.float٣٢),stream=stream)
sizes_device = cuda.to_device(sizes, stream=stream)

#hydrogen_bond

stream = cuda.stream()
energies_device = cuda.to_device(energies, stream=stream)
residue_num_i_device = cuda.to_device(residue_num_i,stream=stream)
residue_num_j_device = cuda.to_device(residue_num_j,stream=stream)
out_device = cuda.to_device(np.zeros((res_num,res_num), dtype= np.float٣٢),stream=stream)
sizes_device = cuda.to_device(sizes, stream=stream)
```

As seen in the code, first we define a stream in which the data would be transferred to the GPU, the reason behind using a stream is that there would be no interference in transferring the data to the GPU.

We designed the algorithm so that there would be a scoring matrix. The scoring matrix dimensions is the same as the number of residues in our input protein. The (i_{th} , j_{th}) cell of our matrix is the sum of the energies of all the interactions between our i_{th} and j_{th} residues. GPU cores architecture is divided into two groups. Each CUDA core is called a thread which can run a function called CUDA kernel. The kernel that the threads run is the same through all of the threads with only the position of thread being exclusive. This position could be derived with pre-defined functions. Threads are grouped in one dimensional or two dimensional arrays called blocks. There could be more than one block for our array of threads. These blocks then are arranged into one or two dimensional arrays called grid. We write the CUDA kernel so that each thread would be computing the sum of energies of one of the matrix cells. Atom by atom energies of the protein are provided to the kernel. The kernel will loop through the atom by atom energies and would sum the energies that their residue numbers are the same as the residue number of the cells hence the threads.

```
#CUDA_kernel_code

@cuda.jit
def Vanderwaals(energy, res_i, res_j, out, sizes):
   x, y = cuda.grid(٢)

   if x < sizes[٠] and y < sizes[٠]:
      for i in range(٠, sizes[١]):
         if res_i[i] == x+١ and res_j[i] == y+١:
            out[x, y] += energy[i]
```

The CUDA kernel is similar in the codes for different energies, here the matrix "out" is our scoring matrix, "res_i" and "res_j" vectors are the number of residues that have interactions and their energies are stored in "energy" vector. "Sizes" is a ٢ cell matrix with the sizes of "out" and "res_i","res_j" stored in it, due to constraints it's better not to use size functions for our elements.

After computing these sums and storing them in the scoring matrix, this matrix would be transferred to the CPU memory for further documenting and saving to file.

```
#saving_the_data_to_file

cuda.synchronize()
out = out_device.copy_to_host()
cuda.synchronize()

wb = Workbook()
sheet١ = wb.add_sheet("electroestatic")

sheet١.write(٠, ٠, "residue name")
sheet١.write(٠, ١, "residue number")
sheet١.write(٠, ٢, "residue name")
sheet١.write(٠, ٣, "residue number")
sheet١.write(٠, ٤, "electroestatic")

index = ١
for i in range(٠, res_num):
   for j in range(٠, res_num):
      if out[i,j] != ٠:
         sheet١.write(index, ٠,  unique_residue[i])
         sheet١.write(index, ١, i+١)
         sheet١.write(index, ٢,unique_residue[j] )
         sheet١.write(index, ٣, j+١)
         sheet١.write(index, ٤,str(out[i][j]))
         index +=١

wb.save("xlwt electroestatic.xls")
```

All the data are saved inside a .csv file. Van der Waals, Hydrogen bond and Electrostatic energy unit is $\frac{kj}{mol}$.

Conclusion:

GPU implementation improved our running time from ٢٤,٤٨٣ sec on CPU to ٦,٥٥ sec which is the sum of ٢,٩٤ sec for vanderwaals energy, ٠,٧٢ sec for hydrogen bond energy and ٢,٨٦ sec for electrostatic energy which is a fourfold improvement on the run time.

Suggestion

GROMOS force filed is used in this project. The program can be developed and other force filed can be added. Also, it is possible to change the input of program to PDB files.

References

[١] "Van der Waals Forces," Springer, ١٦ ٤ ٢٠١٥. [Online]. Available: https://link.springer.com/referenceworkentry/١٠,١٠٠٧٪٢F٩٧٨-٣-٦٤٢-٢٧٨٣٣-٤_١٦٤٧-٤.

[٢] Alexei Alexeyevich Abrikosov, I. E. Dzyaloshinski, and Lev Gor'kov, "Electromagnetic Radiation in an Absorbing Medium," in *Methods of Quantum Field Theory in Statistical Physics*, Dover Publications, ١٩٧٥, p. chapter٦.

[٣] Kensal E van Holde, Curtis Johnson, Pui Shing Ho, Principles of Physical Biochemistry, Prentice Hall, ٢٠٠٥.

[٤] Frank Weinhold, Roger A. Klein, "What is a hydrogen bond? Resonance covalency in the supramolecular domain," *Chemistry Education Research and Practice,* no. ١٥, pp. ٢٧٦-٢٨٥, ٢٠١٤.

[٥] "Hydrogen bonds," GROMACS development team, [Online]. Available: http://manual.gromacs.org/current/reference-manual/analysis/hydrogen-bonds.html.

[٦] Mina Maleki, Gokul Vasudev, Luis Rueda, "The role of electrostatic energy in prediction of obligate protein-protein interactions," *Proteome Sci,* no. ١١, ٢٠١٣.

[٧] "Parameter files," GROMACS development team, [Online]. Available: http://manual.gromacs.org/current/reference-manual/topologies/parameter-files.html.