Project Report

Enumeration of Molecular Clusters

Karthik Mahadevan

Yuji Urano

1. Introduction

A cluster consists of particles and bonds. A bond connects two particles and it has length of one. A rigid cluster is a cluster that you cannot move except for translation and rotation. A floppy cluster is a cluster that is not a rigid cluster. An interesting problem is to enumerate all of rigid clusters for a particular N. There are known results for up to N=11. Since the number of rigid clusters grows dramatically as N gets larger, the amount of computational work you have to do in order to find out all the rigid clusters also grows exponentially. Therefore, parallelization of the algorithm helps a lot to reduce the amount of time you have to spend on the computation.

2. Manifold Parameterization

The Manifold Parameterization algorithm is used to get a rigid cluster from a floppy cluster. Floppy clusters parameterize a geometric object. And it turns out to be a differential manifold. Therefore, we can use techniques from differential geometry.

First of all, the manifold is obtained by considering the constraints formed by the existing bonds. Since we define the bond distance to be of unit length, the constraint equations are of the following form:

q = |x_i – x_j |^2 – 1 = 0.

In order to fix the cluster in the space, we find three points that form a triangle in the cluster. Then we translate and rotate the cluster so that one of those three points is at the origin, one on x axis, and one on the xy plane. This puts additional constraints on the cluster. Next, we move along the manifold slightly. We do it by moving in the direction of tangent vector and then projecting the points back onto the manifold. To find a tangent vector, we first calculate the matrix of the Jacobians of the above constrains and then, get the null space of the matrix. After this slight move, the point moves off the manifold a little, so we move it back onto the manifold by solving non-linear constraint equations. We solve this equation using Newton's method. On doing this repeatedly, the particle being moved will eventually get to a position where it is close enough to another particle that a new bond will be formed between them. This gives us a rigid cluster.

3. Enumeration Algorithm

Now, let's talk about the enumeration of all clusters from a given rigid cluster. By breaking a bond in the rigid cluster, we get a floppy cluster with particles that are free to move. We then apply the Manifold Parameterization algorithm to identify a new rigid cluster. This new cluster might be different from the ones already discovered. Starting from a rigid cluster, by breaking all the bonds in all subsequently discovered rigid clusters, we find all non-isomorphic clusters.

4. Parallelization using MPI:

Rank 0 assumes the role of the 'master' process and all other ranks play the role of 'worker'. The master dispatches tasks to workers in a first-come-first-served basis and maintains a consistent view of all the non-isomorphic clusters identified up to any given point. There are two types of messages that are involved:

i) clustertype

When sent from the master to a worker a cluster message contains:

- coordinates: the set of coordinates of an already discovered rigid cluster
- adj: the adjacency matrix of that rigid cluster
- iBond: the index of the bond which this worker has to break before applying Manifold Parameterization algorithm.

When sent from a worker to the master a cluster message contains:

- coordinates: the set of coordinates of rigid cluster newly discovered by this worker
- adj: the adjacency matrix of that new rigid cluster
- num_checked: the number of already discovered adjacency matrices that this new cluster has been checked with for isomorphism

ii) ***adjtype***

This type of message is always sent from the master to a worker. It contains the list of already discovered adjacencies.

3.1 Sequence of steps executed by the master:

1) Put the initial cluster (the cluster with which enumeration begins) in the list of discovered clusters
2) Make new tasks (whose type is equivalent to message type *clustertype*) out of the initial cluster and push them into the queue of tasks
3) Push all other ranks into the queue of available workers
4) Send the first n tasks to the first n workers, where n = min(number of workers, number of tasks)
5) Receive a *clustertype* message form a worker
6) If num_checked in the received message is -1 (it means that this cluster is isomorphic to some already discovered cluster), go to step 9
7) If num_checked in the received message is equal to the number of discovered clusters (it means that this cluster has been checked with all the already discovered clusters and is non-isomorphic), add this cluster to the list of discovered clusters and make new tasks out of this cluster and push them into the queue of tasks, go to step 9
8) If num_checked in the received message is lesser than the number of discovered clusters send a message of type *adjtype* containing all the adjacencies in the list of discovered adjacencies with index greater than num_checked, go to step 10
9) Mark the task as completed
10) If the total number of tasks is not equal to number of completed tasks go to step 4
11) Send a terminate message (a message of type *clustertype* with iBond = -1) to all workers and end.

3.2 Sequence of steps executed by workers

1) Receive a task (of type equivalent to *clustertype*) from the master
2) If iBond in the received message is equal to -1, go to step10.
3) Run the Manifold Parameterization algorithm to identify a rigid cluster
4) Send the identified cluster to the master in a *clustertype* message with num_checked set to zero
5) Receive the already discovered clusters against which this newly identified cluster has not been checked for isomorphism from the master
6) If no clusters were received in step5, go to step1
7) Do isomorphism check on the received clusters

8) If the identified cluster is isomorphic send a *clustertype* message with num_checked = -1, go to step1
9) Send a *clustertype* message with num_checked updated to the total number of already discovered clusters that have been checked against for isomorphism, go to step5
10) End

5. Scale:

The scalability of the problem is limited by the fact that the steps specified in the mathematical description of the algorithm strictly have to be executed in serial order. However that part of the program is not computationally intensive. Since the algorithm is repeated for each bond in each non-isomorphic cluster for a given N, this algorithm will be executed 24*259 for N=10, 27*1617 times for N=11 and so on. This is what makes permuting all non-isomorphic clusters computationally intensive. By exploiting the independence of each run of the algorithm from other runs, we may execute as many of them in parallel as possible. The scalability of this project is therefore limited by the number of processes that can be executed in parallel which translates to the availability of computers.

6. Existing work/software:

Professor Holmes-Cerfon provided us with a MATLAB implementation of the Manifold Parameterization algorithm. We followed her code to reproduce the algorithm in C++.

7. Code from other sources:

- In order to do the isomorphism check for adjacency matrices, we used Boost library.
- We used Eigen library functions for computing Pseudo-inverse and to get the vector in null space.

8. Accuracy

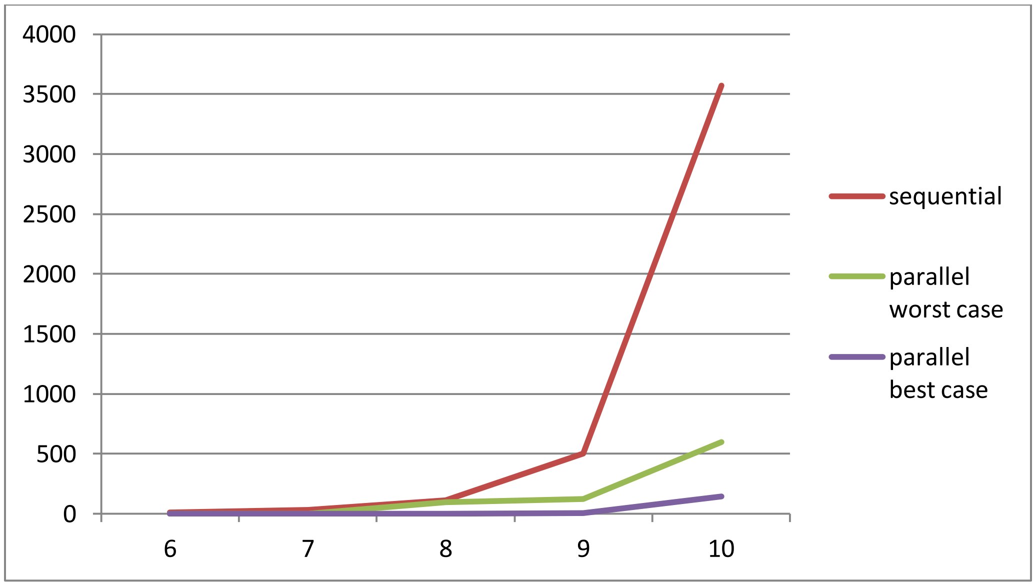
Our implementation of the Manifold Parameterization algorithm produces the correct result up to N=9. However, N=10 includes some singular clusters, and the structure of those clusters is much subtler. As a result, our algorithm identifies 268 non-isomorphic clusters when it should only identify 259.

9. Performance

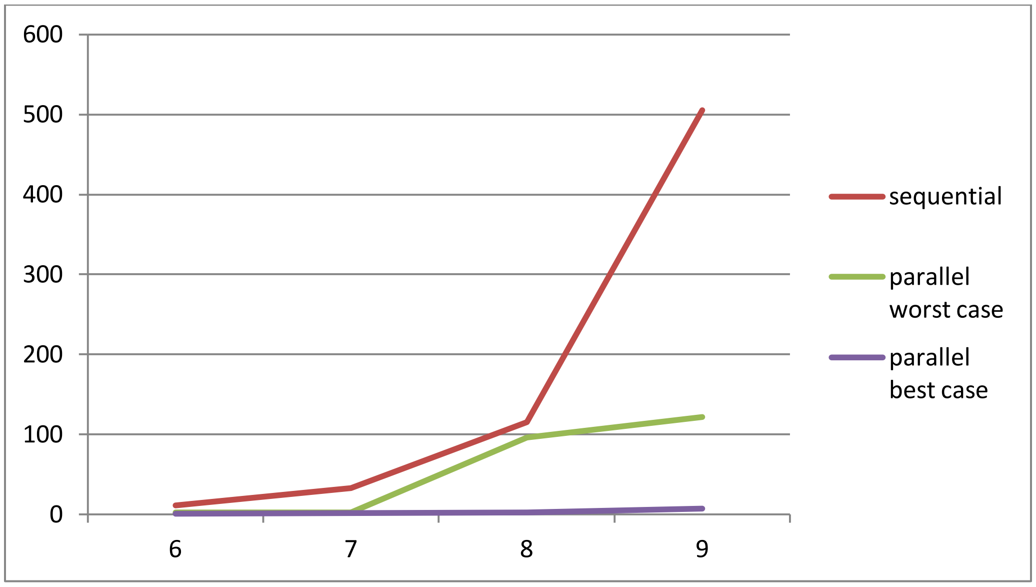
The enumeration program was repeated several times for n=6,7,8,9,10 with a range of ranks between 16 and 104. The performance of the MPI program was a bit erratic but the results were always much better than the results obtained from the sequential MATLAB code. In order to provide the complete perspective of our results we have compared the results from the sequential program with the best results that were observed from the MPI program for each N as well as with the worst results. On an average the performance of the MPI program was a lot better than the worst case performances mentioned below.

All observations are in seconds

N	sequential	parallel worst case (no. ranks)	parallel best case (no. ranks)
6	11	2.70 (72)	1.02 (24)
7	33	2.81 (64)	1.31 (56)
8	115	96.24 (64)	2.16 (88)
9	505	121.54 (72)	7.49 (80)
10	3570	598.44 (64)	146.94 (104)



The following graph shows the same observations but without N = 10 so as to reduce the scale of the graph.



10. How to run our code

Since we used MPI in the C++ environment, you need to have mpiCC to compile our code. Also, you need to download Eigen and Boost. We made our code available at the following git repository on forge: http://forge.tiker.net/p/hpc12-fp-km2549-yu266/

i) Compilation:

```
make sphere_packing
```

This generates an executable by name sphere_packing.

ii) Running the code:

The PBS scripts made available in the git repository can be used to submit jobs to the cluster. All these scripts enumerate the clusters for N=9. Output will be generated in the following format:

```
output9_[Number of ranks]
```

The file “coordinates” contains one set of coordinates each for N=6,7,8,9,10. If you wish to run the program for other N, please replace the coordinates in sphere_packing.cpp with a set of coordinates from this file.

11. References

1) Miranda Holmes-Cerfon, Steven J. Gortler, Michael P. Brenner *A geometrical approach to computing free energy landscapes from short-ranged potentials* http://arxiv.org/abs/1210.5451
2) OpenMP 3.1 Specification http://www.openmp.org/mp-documents/OpenMP3.1.pdf
3) MPI 3.0 Tutorial https://computing.llnl.gov/tutorials/mpi/
4) Boost Isomorphism http://www.boost.org/doc/libs/1_52_0/libs/graph/doc/isomorphism.html
5) Eigen Tutorial http://eigen.tuxfamily.org/dox/index.html
6) Derived Types in MPI http://static.msi.umn.edu/tutorial/scicomp/general/MPI/content6.html