Parallel Algorithm for Protein Folds Prediction

Wang Gang
Dept. of Computer Science
Nankai University
Tianjin, China
wgzwp@163.com

Liu Xiaoguang
Dept. of Computer Science
Nankai University
Tianjin, China
liuxg74@yahoo.com.cn

Liu Jing
Dept. of Computer Science
Nankai University
Tianjin, China
jingliu@nankai.edu.cn

Abstract

Protein folds prediction is one of the most important problems in computational biology. A new parallel algorithm for 3D protein structure prediction is presented in this paper. This algorithm is based on the quasi-physical algorithm that was presented in [8]. Compared with the sequential algorithm, the parallel version improves performance greatly and can obtain much lower energy states for some instances.

1. Introduction

Protein folds prediction is one of the most important problems of computational biology. It tries to determine the structures of protein molecules by their amino acid sequences.

Many simplifications were proposed because of the difficulty of this problem. A popular simplified model is the so-called HP model [1,2] where only two types of monomers: H (hydrophobic) and P (polar) ones. The polymer is modeled as a self-avoiding chain (amino acid sequences) on a regular lattice with repulsive or attractive interactions between neighboring non-bonded monomers. The energies are defined as ε_{HH}=-1, and ε_{HP}=ε_{PP}=0, and the objective is to find the lowest-energy states.

Even if this simplification is too strong, searching the lowest energy states is also a paradigm of combinatorial optimization. So many computational strategies have been tried to find low-energy states efficiently, such as Monte Carlo schemes [3], chain growth algorithms [4], genetic algorithms [5], PERM and improved PERM [6], etc. Unlike these methods, Huang devised a continuous model and a quasi-physical algorithm for this problem [7]. But the algorithm has some defects, we revised it [8], the revised version can produce good results, but was slow. In this paper, parallelization of the quasi-physical algorithm is discussed.

2. Quasi-Physical Algorithm

Huang's model considers each amino acid monomer as a rigid ball, then protein folding can be transform into finding a layout of the chain in 3D Euclidean space that has maximum tangent balls. Of course, neighboring balls in chain must be tangent and any ball can't embed in others.

Huang presented a "quasi-physical" algorithm for the continuous model [7], the algorithm get low-energy state by simulating n-balls system. We can imagine that all the balls are connected by a spring, and consider three types of forces: $\vec{F}_{ij}^{\,p}$ - the pull force of spring between any two neighboring balls, $\vec{F}_{ij}^{\,r}$ - the repulsion forces between any two embedded balls and $\vec{F}_{ij}^{\,g}$ - the gravitational forces between any two H balls (ε_{HH}=-1, ε_{HP}=ε_{PP}=0). Thus, at any time, composite force $\overrightarrow{F_i}$ decides the motion direction and velocity of ball i:

$$\vec{F}_i = \sum \vec{F}_{i,j}^{\,p} + \sum \vec{F}_{i,j}^{\,r} + \sum \vec{F}_{i,j}^{\,g} \qquad (1)$$

Apparently, under the exertion of three types of forces, the H balls tend to congregate to form a center, and the P balls tend to layout peripherally. When the system reaches an equilibrium state, we get a good approximation to 3D protein structure.

We found some defects of Huang's algorithm and revised them [8]. The new algorithm is described below:

Algorithm 1. Quasi-physical algorithm for 3D protein

```
Begin
  For t = 1 to l do
    For i = 1 to n - 1 do
      For j = i + 1 to n do
        Calculate the force ball j
extert to ball i: F_{i,j} = F^p_{i,j} + F^r_{i,j} + F^g_{i,j}
        F_{j,i} = -F_{i,j}
```

$\vec{F}_i = \vec{F}_i + \vec{F}_{i,j}$

End

Calculate new position of ball i: $\vec{r}_i^{t+1} = \vec{r}_i^t + \lambda \times \vec{F}_i$

End

Adjust step λ if necessary

End

Calculate energy, output solution

End

If we define the computation of one F_{ij} (and F_{ji}) as basic operation, and assume that it take unit time, the run time of algorithm 1 is approximately n(n-1)l/2, where l is the number of iterations. The experiments show that the solutions produced by our algorithm have lower energy than those produced by other methods. But the algorithm needs very big l (generally hundreds of millions) to get good result. For long amino acid sequences, the run time of this algorithm is not acceptable. So we consider parallelizing the algorithm.

3. Parallel Algorithm

3.1. Fine-Grained Partition

One intuitive task decomposition strategy is data decomposition. The n balls are distributed to p processes, each process is responsible for n/p balls (and computation related to these balls). This strategy is quite simple and has good load balance, but suffers from a significant disadvantage: it can't easily exploit symmetry and redundancy of computation, performs n(n-1) basic operations per iteration which is twice that of sequential algorithm. It means that the speedup can't be greater than p/2. To do the same amount of work as the sequential algorithm, the parallel algorithm must transfer every F_{ij} from the owner of ball i to the owner of ball j, or contrariwise. The communication is complex and slow.

The cause of poor performance is that the parallel algorithm considers the computation of one F_i as an atomic operation. In fact, we can break it into n tasks (computation of F_{i0}~$F_{i\,n-1}$). Then the parallel algorithm does exactly the same amount of work as the sequential algorithm. The n(n-1)/2 tasks are distributed evenly to p processes. At the beginning of each iteration, each process computes n(n-1)/2p F_{ij}, and then all the processes do an All-Reduce communication altogether to calculate all the F_is, finally every process calculates the new positions of it's own balls and the next iteration can be started. The algorithm is described below, where "id" is the rank number of the process.

Algorithm 2. Fine-grained parallel algorithm

```
Begin
  For t = 1 to l do
    For k = n(n-1)/(2p)*id to n(n-1)/(2p)*(id+1)-1 do
      Convert k to upper triangle matrix coordinate (i, j)
      Calculate the force ball j extert to ball i: F_{i,j} = F^p_{i,j} + F^r_{i,j} + F^g_{i,j}
    End
    Do All-Reduce to calculate all F_i
    Calculate new positions of all the balls
    Adjust step if necessary
  End
  Process 0 calculate energy, output solution
End
```

The parallel run time T_p, speedup S, efficiency E and isoefficiency function are given below, where t_s is the startup time of messages passing, t_w is the per-word transfer time.

$$
\begin{aligned}
T_P &= \left(\frac{n(n-1)}{2p} + (t_s + t_w n)\log p\right) l \\
S &= \frac{n(n-1)/2}{n(n-1)/2p + (t_s + t_w n)\log p} \\
E &= \frac{1}{1 + \dfrac{2(t_s + t_w n)p\log p}{n(n-1)}} \\
W &= \Theta(p^2 \log^2 p)
\end{aligned}
\qquad (2)
$$

Obviously, the algorithm is cost-optimal if plogp=O(n). But the results of the experiments are depressing. For some instances, the parallel algorithm is even slower than the sequential algorithm. The reason is that the communication overhead exceeds the effective computation time.

3.2. Coarse-Grained Parallel Algorithm

Remarkably, our objective is not to calculate movement of n balls system precisely, but just to get a low-energy state. If we use coarse gained data decomposition strategy, and exchange data between processes per r iterations (r > 1) instead of per one iteration, the processes will use "outdated" data, and don't need to communicate with each other frequently. If we choose r properly, the algorithm also can get good results. Although the amount of computation is still twice that of the sequential algorithm, the communication overhead is decreased dramatically.

Algorithm 3. Coarse-grained parallel algorithm

```
Begin
  For t = 1 to l do
    For i = n/p*id to n/p*(id+1)-1 do
      For j = 1 to n do
```

```
        Calculate F_ji if necessary,
calculate F_ji, accumulate F_i
      End
    End
    If (t % r == 0) Do All-to-All
Broadcast to exchange new positions
    Adjust step if necessary
  End
  Process 0 calculate energy, output
solution
End
```

The parallel run time, speedup, efficiency, isoefficiency function are:

$$T_P = (\frac{n/p(n/p-1)}{2} + \frac{n}{p}(n-\frac{n}{p}))l + (t_s \log p + t_w \frac{n}{p}(p-1))\frac{l}{r}$$

$$S = \frac{n(n-1)}{n/p(2n-n/p-1) + 2(t_s \log p + t_w n(p-1)/p)/r}$$

$$E = \frac{1}{1 + \frac{n(1-1/p)}{n-1} + \frac{2t_s p \log p + t_w n(p-1)}{n(n-1)r}}$$

$$W = \Theta(p^2)$$

(3)

It's easy to see that algorithm 3 has higher scalability than algorithm 2. Although algorithm 3 has extra computation overhead, but in fact, the amount of computation is less than twice that of sequential algorithm.

4. Experimental Results

We tested algorithm 3 on Nankai Star supercomputer, which is composed of 400 nodes, two 3.0G Hz Intel Xeon CPUs per node, all the nodes are connected by Myrinet. Other parameters and inputs of our experiment are the same as those in [8].

4.1. Performance for a given problem size as p increases and r increases

Figure 1 shows the run times for instance n = 58 and precision = 1e-4. It is easy to see that the run time decreases as r increases or p increases as we expected. But the curves are fluctuant, the main reasons are:

Firstly, the instance size is not only decided by n, but also l. The algorithm 3 has to use "outdated" data sometimes, and the amount of "outdated" data increases as p increases or r increases. Thus generally, for a given instance, the algorithm 3 need to do more computation to get solutions with the same energies as p or r increases.

Secondly, for a given instance, the amount of extra computation goes up as p increases. Thus if n and r are fixed, the performance maybe decrease as p increases. Especially, if the first reason was also considered, the probability of performance degradation will become higher. For example, figure 1 shows that for n = 58 and r = 1, the parallel runtime increases as p increases.

Thirdly, we can't control CPU assignment of Nankai Star, the scheduler may assign 1 CPU or 2 CPUs from a node to our program. Moreover, the nodes of Nankai Star are partitioned into three subnets, and we can't control node assignment too. So the communication overhead can't be predicted accurately, the performance maybe decreases as p increases, especially for small r.

Lastly, compared with CPU speed, the Myrinet communication speed of our supercomputer is slow. The cost of All-to-All broadcast and All-Reduce are superlinear functions of p. Thus for small instance and small r, the communication overhead dominates the parallel run time, and increases rapidly as p increases.

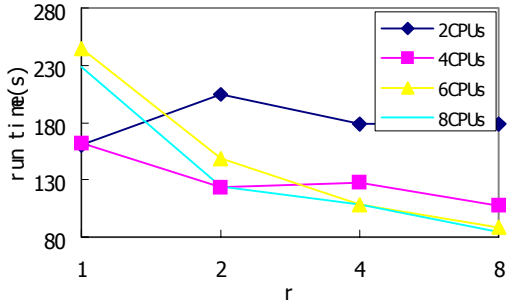

Fig. 1. Parallel run time for instance n=58

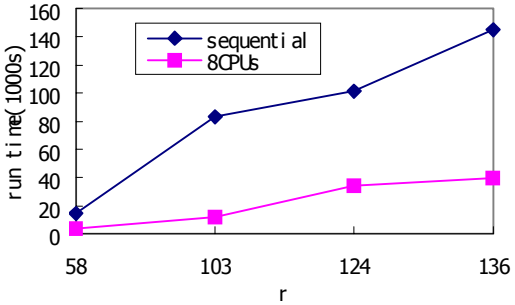

Fig. 2. Run time for a given p as n increases

4. 2. Performance for a given p and a given r as the problem size increases

Figure 2 shows the performance for p = 8 and r = 8 (we found that if r > 8, it is hard to get satisfactory solutions) as n increases, the run time of the sequential algorithm is also shown in the figure. In the experiments, the precision is set to 1e-6. It's easy to see that the speedup tend to increase as the problem size increases. Since algorithm 3 is cost-optimal, this result is expectable. As the precision of solution

increases, the speedup also tend to increase, detailed results are not listed here for paper length limit. But we can see that the speedup does not keep increasing, it varies from 3.01 to 7.26. It looks like that some values much higher than theoretic value given by equation 3, and some values much less than theoretic value. The reason was described in section 4.1, the problem size is not only decided by n, but also l which influenced by p and r. Thus it seems hard to predict the speedup.

4.3. Energy of the results

It was stated above that the using of "outdated" data will make the algorithm to do more iterations, and much more seriously, it's possible to lose solution with lower energy. But interestingly, for some instances, algorithm 3 can get better solution than the sequential algorithm. The reason needs further study. The lower energy structure for the instance n=58 is shown in figure 3.

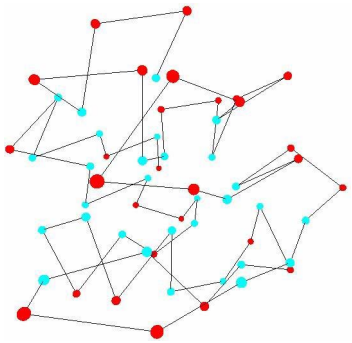

Fig. 3. Lower energy solution for instance n=58

5. Discussion

In this paper we present a new parallel algorithm for 3D protein structure prediction. It's the parallel version of the quasi-physical algorithm presented in [8]. The main idea of the parallel algorithm is that using coarse-grained data composition strategy to partition tasks, and exchanging data between processes periodically to minimum communication overhead. Theoretic analysis indicates that this algorithm is cost-optimal and scalable. The experiment results prove the theoretic analysis, the algorithm can indeed utilize increasing processes effectively as the problem size increases. Compared with sequential algorithm, the parallel algorithm decreases the run time greatly. For some instances, the parallel algorithm can get better solution than sequential algorithm.

Our work is based on Huang's continuous model, but our method can be used for a much wider range of application. In the future work, we will do more experiments to validate our algorithm, and will try to apply our method to more realistic protein models. We also plan to add more information about the proteins into our algorithm, such as molectronics, experiential data, and examine the improvement on performance of the enhanced algorithm.

ACKNOWLEDGEMENTS

The authors thank Nankai Institute of Scientific Computing for the help. This work was supported by NSF of China (No 90612001), Science and Technology Progress Plan Foundation of Tianjin (No 043800311), Science and Technology Progress Plan Foundation of Tianjin (No 043185111-14) and Innovation Foundation of Nankai Univercity.

REFERENCES

[1] K.F.Lau, K.A.Dill, A lattice statistical mechanics model of the cobformation and sequence space of proteins, *Macromolecules,* 22, 2002

[2] Shortle, D., H.S. Chan, and K.A. Dill, Modeling the Effects of Mutations on the Denatured States of Proteins, *Protein Science*, 1（1992）: 201-215.

[3] J. M. Deutsch, Long range moves for high density polymer simulations, *J. Chem. Phys*. 1997, 106, 8849-8856

[4] E.M. O'Toole,A.Z. Panagiotopoulos, Effect of sequence and intermolecular interactions on the number and nature of low-energy states for simple model proteins,*J. Chem. Phys*.1993,98(4),3185-3190.

[5] R. König and T. Dandekar Solvent entropy driven searching for protein modeling examined and tested in simplified models. *Protein Eng*.,2001, 14, 329-335.

[6] Hsiao-Ping Hsu, Vishal Mehra, Walter Nadler and Peter Grassberger, Growth Algorithms for Lattice Heteropolymers at Low Temperatures, *J. Chem. Phys*.2003(118): 444-448 .

[7] Huang Wen-Qi, Huang Qin-bo, Shi He, An Quasiphysical Algorithm for 3D Protein Structure Prediction, *J. of Wuhan University*,2004,50(5)：586-590.

[8] X. Liu, G. Wang and J. Liu, "A Global Optimization Algorithm for Protein Folds Prediction in 3D Space", *ICNC-FSKD 2005*, Changsha, China, LNAI 3614, pp. 1031-1036.

[9] Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, "Introduction to Parallel Computing(Second Edition)", *Pearson Education*, 2003.