中央大学理工学研究所論文集　第25号　2019年
Journal of the Institute of Science and Engineering. Chuo University

タンパク質立体構造モデリングプログラム実行速度の高速化

岩舘満雄*

Speedup of the protein tertiary structure modeling program execution speed

Mitsuo IWADATE*

Abstract

Three speedup actions had performed in homology modeling method, FAMS. One of them is multi-thread to have functioned. The relationship between the degree of speedup and the number of the threads is non-liner. The reason of the non-linearity is regarded as having adopted fork function. The structure body array for many amino acid residues included in the protein was declared in the original version of FAMS software, and it requires considerable memory. Additionally, fork function makes full memory duplicate to all co-processes. The duplication affects the total spending time in multi-threading steps. Therefore, array size reduction or appropriate selection of variables is required.

1. 緒言

ＰＤＢ[1]に代表されるタンパク質の立体構造データベースでは，Ｘ線回折や核磁気共鳴（ＮＭＲ），近年では電子顕微鏡の実験技術の発展により大幅にその登録数が増えており，その増える割合もまた増えている状況にある．実験技術の進歩によって膜タンパク質や巨大分子量のタンパク質などそれまでの技術では解析不可能とされてきた構造がつぎつぎと明らかになるにしたがって構造未知のタンパク質が減少することが期待されていた．

このことはホモロジーモデリング法にも必然的に影響を与える．ホモロジーモデリングはアミノ酸配列上極めて類似性の高いタンパク質同士は立体構造が似ていることという経験的に知られている事実に基づいて，配列上類似部分を参考データにしながら立体構造解析の実験はなされていないタンパク質に対して，あたかも実験をしたかのような立体構造を提示する技術である．実験で得られた立体構造ではないために信頼性が当然のように問題となるが，大まかな目安としてはアミノ酸配列の文字が３０％一致している領域があれば，その領域は１～５Å（１Åは10のマイナス10乗メートル）の誤差範囲に収まることが，Ｘ線回折の実験構造の再現実験との比較で知られている[2]．

このホモロジーモデリング法の全過程を全自動で行うことが出来れば，ハイスループットにゲノムデータのような巨大データを処理して，データベース化することが可能である．実験結果ではないが，ホモロジーモデリング法で予測された構造を予め計算しておけば，任意の遺伝子配列をもつアミノ酸配列に対して立体構造を提供できる割合を大幅に上昇させることができるという考えに従ってモデル構造を提供するデータベースを提供維持する試みも行われてきた．継続できれば立体構造の実験結果の増加に従って，予測構造の高精度化と提供割合の増加という，いわゆる質と量の改善を同時に達成できる目的で行われていた経緯がある[3]．

しかしながら，次世代シーケンサー解析をはじめとする配列を解明する実験技術は，生物種を超えて個体

* 中央大学理工学部生命科学科　〒112-8551　東京都文京区春日1-13-27

による差異を取り扱うレベルになってきており，加速の割合は立体構造の実験技術の加速では追いつけないほど桁違いの速度となっており，予測データベースの提供維持も困難である．例えば新規の配列が一つ追加された状況において，実験構造データベースに対して類似性の検索を行う操作のみであっても，実験構造データベースが増加する状況では計算量は確実に増える．さらに配列データの登録の速度がより加速された状況においては必要な計算量は相乗効果的に増えてゆく．さらに，それらに立体構造の計算を行うことを合わせると容易に限界が訪れる．さらにホモロジーモデリングで作られた立体構造のデータベースを運用している最中のウェブ上のログをみると，大量に計算された立体構造のほとんどは全くアクセスされないことが明らかになっていった．

以上のことから，本研究テーマでは，データベースを整備するよりも，ユーザー側からリクエストされたときに迅速に処理をして求められた計算のみを高速に行う方が労力に対する価値が高いと考え，全自動のホモロジーモデリング法の高速化を行う取り組みを行った．

2.　配列数と立体構造について

立体構造情報データベース PDB の配列情報は PDB のサイトにあり，毎週更新している[4]．この情報を基にして，生物のゲノム情報におけるホモロジーモデリング法における適用可能な割合を決めることができる．例としてヒト遺伝子を取り上げて，この割合の変化を見てみた．

ヒトゲノム中の 29,279 個のタンパク質の遺伝子配列を国立遺伝学研究所からダウンロードし，2014 年から 2019 年までの PDB の配列情報に対して，BLAST 検索を総当たりで行った．E 値は 1e-10，配列一致率は 30%を閾値として，立体構造の実験座標が検出された場合に構築可能との基準を用いた．

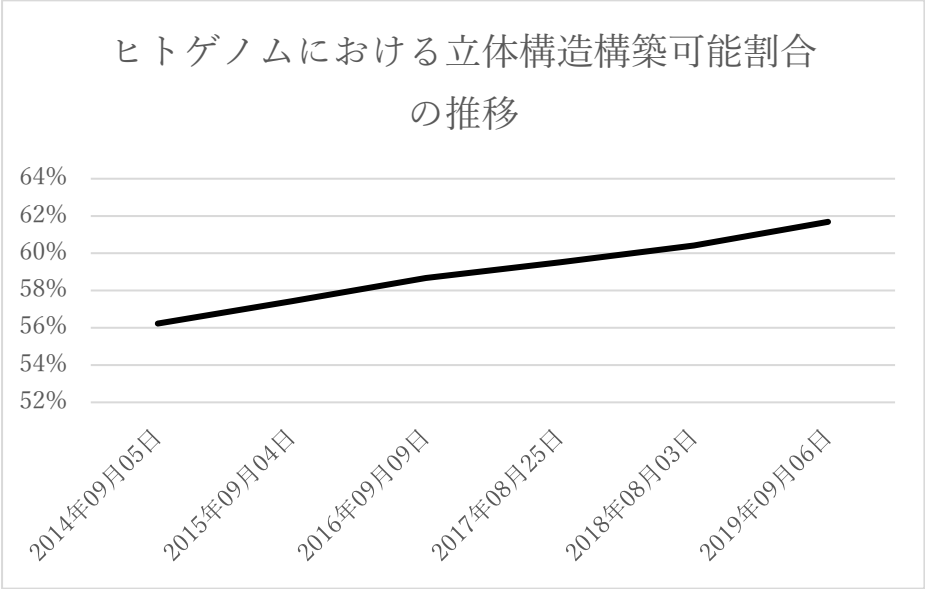

Fig.1 過去６年のヒトゲノムに対数立体構造構築可能な遺伝子の割合の推移

過去 6 年間はこの割合が直線的に増加していることが分かる．ホモロジーモデリング法の精度の向上の可能性及び技術的な重要性が高まってきていることが改めて示された結果となっている．

3.　ＦＡＭＳ（Full Automatic Modeling System）のアルゴリズムと高速化の３つの方法

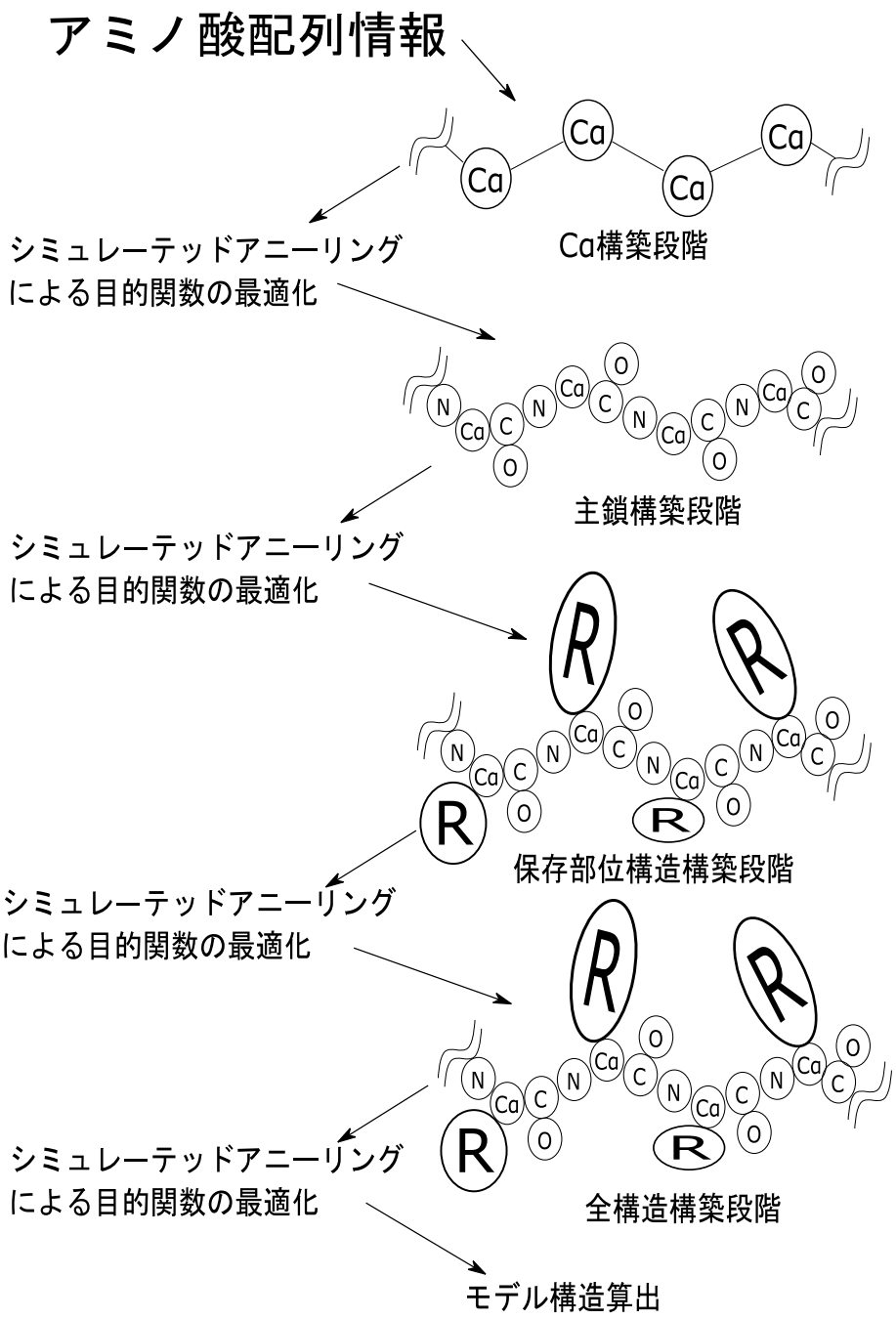

Fig.2 FAMS のアルゴリズムの概要。R は側鎖を示す。

ホモロジーモデリング法のソフトウェアとして北里大学で開発され，筆者が管理するＦＡＭＳの高速化を試みた[5]．ＦＡＭＳは大きく４段階に分かれておりＣα構造構築段階，主鎖構造構築段階，保存部位構築段階，全体構造構築段階になっている．この４段階がそれぞれにデータベースを検索するステップとシミュレーテッドアニーリングをするステップに分けられている．実行時間としてはすべてのステップに時間がかか

っており，すべてに対して有効な時間短縮がされることが必要になる．
本テーマではデータベース探索部分の時間短縮の検討を行った．データベース検索の時間短縮は，以下の３つの方法を試みた．

Ⅰ　データベース中の検索対象に対し，検出される見込みのないものを予め削減することにより時間短縮する試み．

Ⅱ　GPU を用いてシミュレ―テッドアニーリング部分を高速化する試み．

Ⅲ　複数のスレッドを搭載する CPU に対して予めデータベースを分割して検索時間を短縮する試み．

Ⅰはソフトウェア的な方法のみでデータ処理を高速化する試みであり，Ⅱ，Ⅲはハードウエアとソフトウェアの両方に関わる試みであるといえる．

4.　データベース検索のソフトウェア的な高速化

高速化部分としては最初のＣα構築段階に対してのものだった．FAMS のデータベース検索においては三次元座標の誤差の最小二乗を生み出す回転行列を求めるため，データベース中の検索対象に対し検索をするたびに特異値分解の計算を実行する．特異値分解自体が反復を多く含む計算方法であるため，データベース検索に用いると時間がかかる．この計算量を大幅に削減することを可能とする理論が開発された[6]．本テーマではこの理論のホモロジーモデリング法における実用性を試すことを行ったことになる．この試みは断片を RMSD が高い検索対象を効果的に削除することに有効だった．この方法は多くの断片から RMSD の低いものを検出する方法として極めて有効であったが，FAMS は RMSD が低いことに加えてさらに，空間的な衝突を避ける必要があった．この方法でフィルタリングされた後の断片の集団のなかで空間的に衝突が避けられた断片を探すことで適用前と結果的に再現性が見られなくなってしまった．空間衝突と低い RMSD を同時に求める関係上，そのままの適用は困難であると判断して以降この方法の適用を断念した．

5.　GPU を用いたシミュレーテッドアニーリング部分の高速化

高速化部分は主鎖構築段階の後のシミュレ―テッドアニーリング部分である．
GPGPU(General Purpose Computation on Graphics Processing Unit) で実現を試みた．GPGPU を容易に扱えるようにするため，NVIDIA 社から発表されている自由度の高い統合開発環境 CUDA を用いた．CUDA の特徴である「C 言語を拡張した言語仕様」「画像処理技術への習熟を要しない」「対応 OS に Linux が含まれる」といった点は FAMS 開発と共通しているといえる．CPU はインテル社製 Core i7 980X Extreme Edition BOX，GPU は NVIDIA 製 GeForce GTX 560 Ti を搭載したマシンを使った．ソースコードの移植は FAMS のシミュレーテッドアニーリングに相当する C 言語を対象に行い実行時間の計測を行った（表１）．
ソースコードの移植は FAMS のシミュレーテッドアニーリングに相当する C 言語部分を対象に行った．シミュレーテッドアニーリングに相当する C 言語部分内部のほぼすべての計算時間を占める部分の関数を GPU に移植することが効果的だと考えられる．グリッド・ブロック・スレッドの割り当てについては，1 要素につき，１スレッド与える方針で設計した．(sample は 168 要素あったので，168 スレッドを割り当てるようにした．)
移植するにあたって，CUDA 仕様上の大きな問題点は三つである．
１つ目は FAMS のＣ言語ソースが長大であるため C プログラムと CUDA プログラムを混在した形で組み込んでコンパイルする要件を満足すること．
２つ目はデバイス側は UNIX 系ＯＳで使われる drand48()という乱数パッケージが使えないため再現を確認することが困難であること．また，CUDA 自体にも疑似乱数生成パッケージがない．そのため，NVIDIA 社が提供している疑似乱数生成ライブラリ curand で代替した．
３つ目は GPU 上では標準入出力関数(printf,scanf など)やファイル入出力関数(fopen, fprintf など)等，C 言語で通常定義される関数の大部分が使用不可であることなどだった．
FAMS のアルゴリズムを変更せず移植した結果，実行速度は大幅に遅くなった．

Table 1　GPU を導入したあとのシミュレーテッドアニーリングの実行時間

Perturb	導入前[s]	導入後[s]
1 回目	1.940	7.670
2 回目	1.950	7.570
3 回目	1.900	7.590
平均	1.930	7.610

Perturb 関数の実行時間の増加は全体の時間に大きな影響を与える結果となっている．このような結果になってしまった原因として 2 点考えられる．一つは「ウォープダイバージェント」と呼ばれる現象で本来実行されない処理も実行されてしまうため，CPU と比べて，約 2 倍の処理時間がかかることになる．perturb 関数内ではこのような if 文が数多く記述されており，その分だけ時間を消費してしまっている．これを改善するには，処理を行う前に要素を並び替え，同一ウォープ内の全スレッドが，if 文で同じ判定になるようにするか，コード全体を書き直す必要がある．二つ目は複数のスレッドがスレッド間で共有するメモリに同時に更新する挙動である．他のスレッドが共有メモリの読み書きをさせないようにするための制御法いわゆる排他制御に不具合があることがわかり，より GPU に親和性の高いソースコードの構築の必要性が明白となった．そのため，これ以上ハードウエアに計算方法を寄せることの性能悪化を懸念し断念した．

6.　複数のスレッドを搭載する CPU に対するデータベースの分割

FAMS は C 言語で記述されているため CPU 中の複数スレッドの実現は fork 関数の使用によって試みられた．CPU の速度は各スレッドで実行した結果を統合して一つのプロセスで長時間かけておこなったものと同一の結果になるようにするために，各スレッドが共有メモリ部分にファイル書き込みを行い互いに参照し合うこととした．

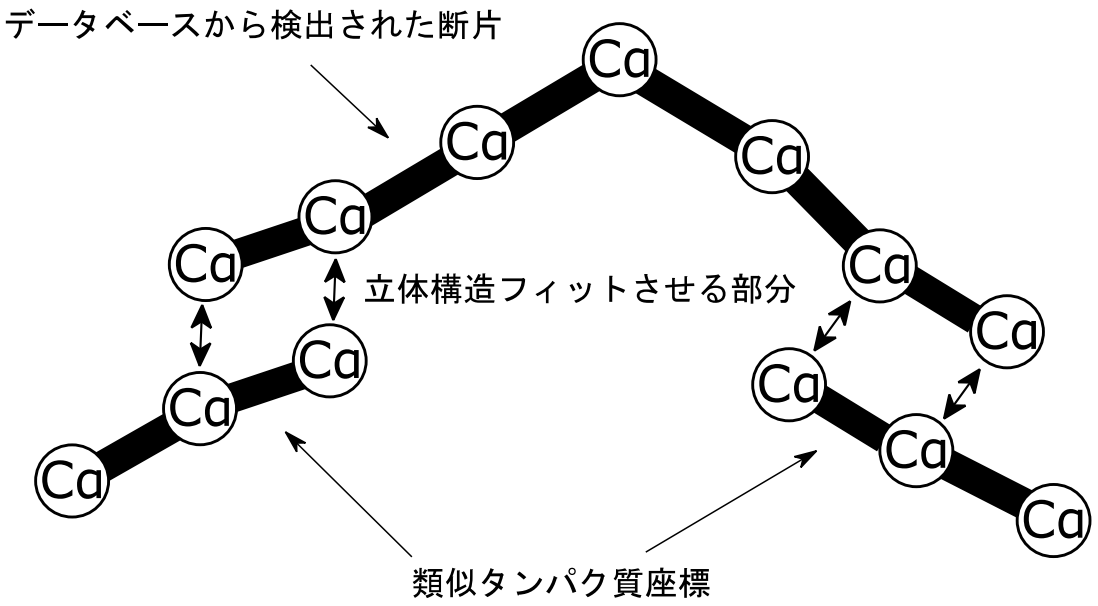

Fig.3 Cα構築段階において断片データベースからフィッティングしている。

高速化部分は FAMS の Cα構築段階（Fig.2）である．この段階の詳細は，Fig.2 のようにこの断片は PDB の立体構造データベースから取得している．フィットさせている N 末端側 2 残基分，C 末端側 2 残基分の合計 4 残基の Cα の RMSD の中である程度候補を絞り，絞られた候補の中でフィットした断片が空間的に他の原子

と衝突していないかをチェックするという 2 段階に分かれている.

この 2 段階は独立性が高いので別々に取り組む必要があった. まず, 第一段階のデータベース中の断片すべての RMSD を調べることを分散化した効果を調べた. CPU は Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz の 4 コア, 8 スレッドを用いた. なお, 8 スレッドすべてを使うのではなく 7 スレッドのみ使用した.

Table 2 データベース検索を複数スレッドに分割した実行時間

Perturb	導入前[s]	導入後[s]
1 回目	18.072	11.715
2 回目	18.060	11.720
3 回目	18.107	11.757
平均	18.080	11.731

複数スレッドになって有意に時間短縮した効果が認められる. 7 スレッド使っているので理論上は検索速度が 7 倍になるが, 実際は 1.54 倍(=18.080/11.731)であった. 高速化部分がプログラム全体に及んでいないためと考えられる.

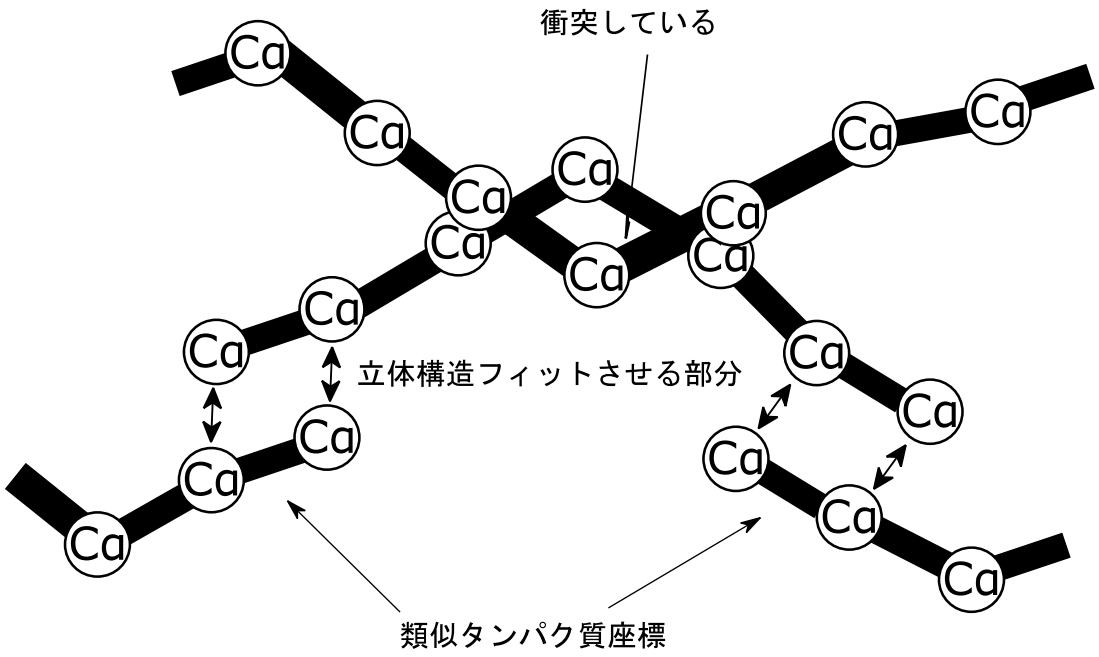

Fig.4 検出された断片が立体空間上衝突しているイメージ

次に第二段階として第一段階で各断片の RMSD が明らかになった後でその RMSD が小さい順に並べ替えて立体空間上の衝突の少ない断片を検出する段階のスレッド分散化を試みた. RMSD が小さいことは断片が自然にかつ滑らかに類似タンパク質に組み込まれることを示すが, その断片が立体構造上の衝突があると自然の状態を反映しているとは言えなくなる.

Table 3 データベース検索と衝突の少ない断片を検出する段階を複数スレッドに分割した実行時間

Perturb	導入後[s]
1回目	6.534
2回目	6.506
3回目	6.495
平均	6.512

この部分にスレッド化を試みた結果はスレッド分散化していない状態に比べて2.78倍(=18.080/6.512)の高速化が認められる．理論値7倍には届かない理由は，分散化していないソース部分が残っていること，分散化した後データを統合するのにタイムラグが生じること，負荷をスレッドに均等に分散出来ていないことなどが考えられる．

そこでCPUを8コア16スレッド搭載のIntel(R) Core(TM) i9-9900K CPU @ 3.60GHzに変更してスレッド数が増加したときの速度変化を計測してみた．1スレッドだけ使わずに残す設定にしてあるので15スレッド使用することとなる．

Table 4 スレッド数を7から15に増やした時の実行時間

Perturb	1スレッドのみでの実行[s]	データベース検索のみ複数スレッド化[s]	データベース検索と衝突回避の両方を複数スレッド化[s]
1回目	10.179	6.710	3.245
2回目	9.560	6.671	3.170
3回目	10.181	6.673	3.428
平均	9.973	6.685	3.281

データベース検索部分のみ複数スレッド化した高速化の効果は15スレッドの時に1.49倍(=9.973/6.685)と7スレッドの時の1.54倍より低下している．データベース検索と衝突回避の両方を15スレッドに分散した場合3.04倍(=9.973/3.281)となり，7スレッドの2.78倍と比べスレッド数の増加で期待できるほどの高速化は認められなかった．

7. 複数のスレッドに対してタンパク質の配列を分割して検索時間を短縮する試み

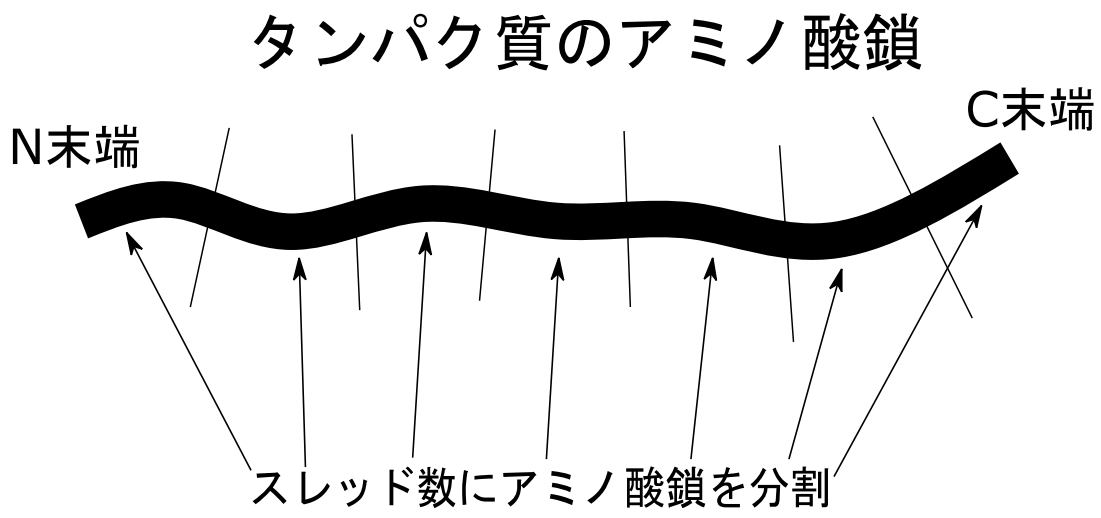

Fig.5 アミノ酸鎖をスレッドの個数分に分割のイメージ

複数スレッド化に効果的であることから第 2 段階である主鎖構築段階に関してもスレッド化分割を行った．データベース側を分割するのではなく今回はクエリであるタンパク質側の N 末端側から C 末端側までのアミノ酸残基を複数のスレッドに分割した（Fig.5）．複数のスレッドに分割する手法としては Cα構築段階と同様に fork 関数を用いた．

Table 5 主鎖構築を 7 スレッドに分割した実行時間

Get_atom2	**1 スレッドのみ[s]**	**7 スレッド[s]**
1 回目	20.793	6.296
2 回目	21.397	4.854
3 回目	21.397	5.750
平均	21.196	5.633

7 スレッドで 3.76(=21.196/5.633)倍の高速化の効果があった．主鎖データベース可動域の狭さからはCαデータベースに比べてサイズが小さいため fork 関数による複数スレッド化の際に複製するメモリ量が限定的なために効果的にスレッド化の効果が出たと考えている．引き続き 15 スレッドに分割した効果も測定した．

Table 6 主鎖構築を 15 スレッドに分割した実行時間

Get_atom2	**1 スレッドのみ[s]**	**15 スレッド[s]**
1 回目	13.451	3.388
2 回目	13.925	3.416
3 回目	14.104	2.678
平均	13.827	3.161

15 スレッドで 4.37 倍(13.827/3.161)の高速化の効果があった．Cα構築段階に比べると複数スレッド化の効果が大きいことは 7 スレッドと同じだが，スレッド数の増加とともに単位スレッドあたりの性能が十分に発揮されない傾向が同様に存在する．

8. まとめ

本テーマで 3 つの高速化の取り組みを行った．機能したのはマルチスレッド化のみだった．

単純にマルチスレッド化してスレッド数を増やしても高速化にはつながらず，1 スレッドでかなり高速な CPU を用いた場合は分散しても効果が期待できないこととなり，近年のマルチスレッド化を強力に推し進めている半導体分野の恩恵を直ちに受けられる結果とはならなかった．大きな原因は Fork 関数を採用したことと考えられる．FAMS のプログラムはタンパク質に含まれる多くのアミノ酸残基に対して構造体を宣言しておりオリジナルのプログラムの状態でかなりのメモリを使用している．Fork 関数はその性質上親プロセスから派生したプロセスは親プロセスの変数を基本的にすべて継承しているためそのデータ複製にかかる時間が無視できないほど大きく，マルチスレッド化すればするほど複製時間が余計にかかる結果となって処理を分散化した時間短縮の効果を奪ってしまったと考えられる．プログラム中で扱う変数を派生したプロセスに必要不可欠な変数のみ取捨選択してマルチスレッド化すれば大きな効果が期待できると考えている．

ソフトウェア的な取り組みに関しては，本テーマではうまく機能しなかった．特異値分解の解の範囲を限定する理論自体は機能していたと考えられるが，空間衝突を避ける部分と組み合わせると低速化し，またデータベース中の断片全てに対して特異値分解を丁寧に行った場合と比べて再現性が取れない結果となった．

ハードウエア的な取り組みである GPU による高速化と CPU マルチスレッド化の二つを比較して，前者の適用時の難易度の困難さが顕著だった．

今回は FAMS のアルゴリズムのデータベース探索部分とシミュレーテッドアニーリング部分の二大要素のうちデータベース探索部分のみ高速化した結果となった．残るシミュレーテッドアニーリング部分に関しては，マルチスレッド化によって高速化を試みてゆく必要性があると考えている．また，FAMS のＣα構築段階，主鎖構築段階，保存部位構築段階，全体構造構築段階のうち前の 2 段階のみ高速化した結果となっており，後の 2 段階に関しても着手してゆく必要性がある．

今後はマルチスレッド化を中心にした高速化に重心を置いて進めることになると考えられる．単純にスレッド数を増やしても速度の増加が望めないことから複製する変数を限定化することがまずは優先事項と考えられる．しかしながら変数を制限することは現在機能しているアルゴリズムを変更してしまうことになる．アルゴリズムを変更してより価値のある状態であることを含めた包括的な高速化が重要であると結論付けられる結果となった．変数の簡素化が実現できれば本テーマではうまくいかなかった GPU による分散化の道もまた開ける可能性がある．

9. 参考文献

[1] Burley SK, Berman HM, Bhikadiya C, Bi C, Chen L, Di Costanzo L, Christie C, Dalenberg K, Duarte JM, Dutta S, Feng Z, Ghosh S, Goodsell DS, Green RK, Guranovic V, Guzenko D, Hudson BP, Kalro T, Liang Y, Lowe R, Namkoong H, Peisach E, Periskova I, Prlic A, Randle C, Rose A, Rose P, Sala R, Sekharan M, Shao C, Tan L, Tao YP, Valasatava Y, Voigt M, Westbrook J, Woo J, Yang H, Young J, Zhuravleva M, Zardecki C, RCSB Protein Data Bank: biological macromolecular structures enabling research and education in fundamental biology, biomedicine, biotechnology and energy, Nucleic Acids Res., 47, D464-D474 (2019)

[2] Yamaguchi A, Iwadate M, Suzuki E, Yura K, Kawakita S, Umeyama H, Go M, Enlarged FAMSBASE: protein 3D structure models of genome sequences for 41 species, Nucleic Acids Res., 31, 463-468 (2003)

[3] Kanou K, Hirata T, Iwadate M, Terashi G, Umeyama H, Takeda-Shitaka M, FAMSD: A powerful protein modeling platform that combines alignment methods, homology modeling, 3D structure quality estimation and molecular dynamics, Chem. Pharm. Bull., 58, 1-10.(2009)

[4] ftp://ftp.wwpdb.org/pub/pdb/derived_data/pdb_seqres.txt.gz

[5] Umeyama H, Iwadate M, FAMS and FAMSBASE for protein structure, Curr. Protoc. Bioinformatics, Chapter 5:Unit5.2.(2004)

[6] Shibuya T, Searching protein 3-D structures in linear time, J. Comput. Biol., 17, 203-219. (2010)