Vol.2014-MPS-98 No.21
Vol.2014-BIO-38 No.21
2014/6/27

幾何学的サフィックス木に対する並列処理性能の評価

高橋誉文[†1]　田村慶一[†1]　黒木進[†1]　北上始[†1]

ディスク上に蓄積された幾何学的サフィックス木は，蛋白質立体構造データベースに対する高速な類似検索のための索引構造として利用可能である．しかしながら，蛋白質立体構造データベースの大規模化により，その索引構造の構築に多くの時間を費やすだけではなく，索引構造を用いた類似検索に要する時間にも影響を与える．本論文では，並列化による高速処理を実現するために，幾何学的サフィックス木の構築および幾何学的サフィックス木を用いた類似構造検索の双方をマスタワーカーモデルや分散ワーカモデルを用いて並列化することにより高速処理を実現する方法を提案する．著者らは，幾何学的サフィックス木の従来の構築法が並列化に直接向いていないという点に着目し，並列化をする前に，予め，この構築方法を従来のインクリメンタルな構築方式（座標配列ごとの逐次方式）から全データをまとめて処理するトップダウン方式に変更している．また，データページを管理するバッファ管理法の変更も行っている．さらに，構築と検索のそれぞれに対して，データ分割法による並列化とタスク分割法による並列化を実施し，それらの並列性能を評価している．実験により並列性能を測定した結果，幾何学的サフィックス木の並列構築においては，データ分割法がタスク分割法よりも優れており，類似構造検索においては，タスク分割法がデータ分割法よりも優れていることが判明した．

Performance Evaluation of Parallel Processing for Geometric Suffix Tree

Yoshifumi Takahashi, Keiichi Tamura, Susumu Kuroki, and Hajime Kitakami

Geometric suffix trees stored on a disk can be used as indices to achieve a high-speed similarity search on large-scale, 3-D protein structure databases. However, due to an increasing number of data records in the protein structure database, a large amount of time must be taken to construct the index, greatly affecting the search speed. This paper proposes a parallel processing method using a master worker and distributed worker models in order to parallelize both the construction algorithm of the index and the similarity search algorithm using the index constructed in the protein structure database. The authors focus on the viewpoint that the existing construction method of the geometric suffix tree is not suitable for direct parallelization. Therefore, beforehand, the authors changed the existing incremental construction algorithm of the geometric suffix tree into a top-down algorithm to process all of the data together in the database. Moreover, the authors changed the existing buffer control method into an adaptive method for parallel processing. In order to parallelize both the construction and the similarity search, both data partition and task partition methods are applied to a given problem, such as the construction and similarity search, and sub-problems generated from the given problem are executed in parallel using each worker model. The performance evaluation experiments resulted as follows: in the parallel construction of the index, the data partition method was better than the task partition method; in the parallel similarity search using the index, the task partition method was better than the data partition method.

1. はじめに

蛋白質は生命活動の生体機能にかかわる重要な物質であり，生命科学の研究や創薬対象となっている．蛋白質は，アミノ酸配列が異なっていても，立体構造と生体機能の間には密接な関係があることが分かっている．近年，蛋白質の立体構造データが急増してきているため，蛋白質立体構造の類似部分構造を高速に見つけ出す方法の研究が盛んに行われている．その代表的な方法として，ディスク上に幾何学的サフィックス木を構築し，それを利用する類似構造検索の方法が提案されている[1][2][3][4][5].

†1 広島市立大学大学院
Graduate School of Information Science, Hiroshima City University

しかしながら，幾何学的サフィックス木の構築と検索の並列化による高速化の研究がまだおこなわれていない．与えられた問題を並列化するために，その問題を分割する方法（問題分割法）を初めとして，問題分割によって生成されるタスクの負荷分散法を明らかにすることが重要である．問題分割法には，データ分割法とタスク分割法が知られているが，どちらも問題を分割する方法のみに関心があるだけで，プロセッサの稼働率が 100%になることを保障していない．このため，問題分割法を明らかにしたら，稼働率100%を目指す負荷分散法について検討する必要がある．

しかしながら，従来の幾何学的サフィックス木では，複数の座標配列のそれぞれを幾何学的サフィックス木に逐次追加している．この方法では，並列化におけるデータ分割

法と負荷分散法を行うことができないので，並列化に適した幾何学的サフィックス木の構築ができていないという問題がある．また，どのように問題分割法と負荷分散法を組み合わせた並列化を行えばよいのかということが明らかにされていない．

このような問題を解決するために，本論文では，幾何学的サフィックス木の構築と検索の並列化を行うために，以下の２つの仕組みを提案する．

(１) 構築方法の改良

座標配列を幾何学的サフィックス木に逐次追加することによる頻繁なデータの修正回数の削減と，タスク分割を行うために，幾何学的サフィックス木の構築方法をトップダウンに変更して座標配列の全件をまとめて構築する．また，構築方法の変更に合わせ，中間ノードページのバッファ管理法を TOP-Q から LRU に変更し，隠れ配列ページのバッファ管理法を行わないように変更する．

(２)並列化の方法

幾何学的サフィックス木に対するデータ分割法やタスク分割法による問題分割法のほかに，マスタワーカ法や分散型ワーカ法による負荷分散法を組み合わせた並列化を行い，幾何学的サフィックス木の構築と検索を高速化する．

本論文の構成は以下のとおりである．２章ではサフィックス木の関連研究について述べる．３章では，従来手法であるディスクベースの幾何学的サフィックス木について述べる．４章では，提案手法について述べる．５章では提案手法の有効性を示すための実験による評価を行い，６章では本論文のまとめを行う．

2. 関連研究

文字配列に対するサフィックス木の概念は 1973 年，Weiner[6]によって初めて紹介された．その後，1976 年 McCreight[7]によって構築法を単純化され，1995 年には Esko Ukkonenn が線形時間でサフィックス木を構築するアルゴリズム[8]を紹介した．しかしながら，大規模な文字列データでサフィックス木を構築するには，その文字列データよりも多くの記憶領域を必要とするだけではなく，1970 年代のコンピュータ性能ではサフックスを構築・利用することが現実的ではなかったため，サフィックス木に関する研究が盛んではなかった．それ以後，コンピュータの性能向上とディスクの記憶容量の低価格化が劇的な速さで進んだため，実用化を意図したサフックス木に関する多くの研究が 盛んに行われてきた[9][10]．特に，サフィックス木をメモリからディスク上への展開の研究[11][12][13][14]は 2001 年頃から注目され，サフィックス木の並列化の研究[15][16][17][18][19][20][21]は，2006 年頃から注目されるようになった．ゲノム規模の文字配列に対するサフィックス木のディスクに基づく研究[15][22]は，2007 年頃 から行われている．これに対して，蛋白質立体構造に対するサフィックス木[3]は，２つの座標点集合どうしを RMSD[23][24][25][26][27][28]により空間的に重ね合わせる方法を用いた方法であり，2004 年頃から研究されている．このサフィックス木は，幾何学的サフィックス木と呼ばれている．その後，ディスクに基づく幾何学的サフィックス木の研究[1]が行われたが ，サフィックス木の並列化による高速処理の研究はまだ行われていない．

3. 従来の幾何学的サフィックス木

従来のディスクベースの幾何学的サフィックス木[1]は，幾何学的サフィックス木をディスク上に構築されたものであり，これを用いて，類似構造検索を行うことができる．

図 １にディスクベースの幾何学的サフィックス木のシステム構成を示す．点線で囲まれた範囲がバッファ管理システムである．バッファ管理は，DPM（データページ管理モジュール）と CPM（カタログページ管理モジュール）の２つのモジュールから構成される．DPM はメモリ上のデータページのアドレスの管理とディスクとの読み書き，CPM はメモリ上のカタログページのアドレスの管理とディスクとの読み書きを行う．なお，データページには．幾何学的サフィックス木のデータが格納され，カタログページには，データページを管理するためのページテーブルが格納されている．

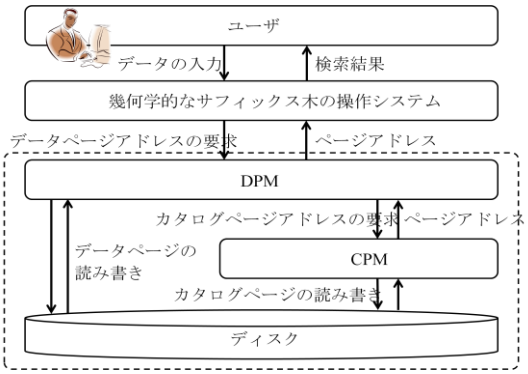

図 1．システム構成図

本章では，先ず構造間の比較に用いられる非類似度の尺度について述べ，従来の幾何学的サフィックス木の構成を初めとして，隠れ配列の仕組みやバッファ管理法について述べる．

3.1 非類似度の尺度

幾何学的サフィックス木は，蛋白質立体構造の C_α 原子の座標位置だけを抜き出して，N 末端から C 末端に並べた座標配列をもとに構築されたサフィックス木である．

幾何学的サフィックス木に対する類似構造検索では，座標配列間の非類似度の尺度として，RMSD が利用されているが，RMSD を類似構造検索に直接利用すると，サフィックス木の深さ方向に関する単調性が保障されない．このため，RMSD の代わりに，**MSSD** (minimum sum squared

distance)を利用することで，単調性を保障している．

２つの座標配列 $Q=(q_1q_2\ldots q_m)$および $T=(t_1t_2\ldots t_m)$とすると，MSSD は以下のとおりである．

$$\mathrm{MSSD}(Q,T) = m(\mathrm{RMSD}(Q,T))^2$$

幾何学的サフィックス木では, MSSD の値と分岐の閾値 b とを比較し， b 以下であれば，同じ枝とし，閾値 b を超えたとき, 中間ノードを作成し, 分岐させている. 一般に, ノード *node* に記録されるべきサフィックス座標配列は, 複数存在するが，サフィックス木構築中の分岐処理に最初に利用されたサフィックス座標配列を代表として記録されている．

3.2 幾何学的サフィックス木

幾何学的サフィックス木とは，座標配列に対するサフィックス木を構築したものである．幾何学的サフィックス木で同じ枝とする条件は，2 つの座標配列のサフィックスの間の MSSD の値が分割パラメータ b 以下となることである．分割パラメータは，MSSD の閾値である．

複数本の座標配列のそれぞれに識別子 ID が振られているとするとき，サフィックス木の各ノード *node* で[(i, j), *Length*, *Count*]のデータを記録する． (i, j) は ID が i の座標配列について，先頭から j 番目の座標点から始まるサフィックス座標配列の位置情報を意味する．以後，このサフィックス座標配列を $S_{(i, j)}$ と表記する. *Length* は, 根ノード *root* からノード *node* までの経路に対応する座標配列の長さを意味し，そのノード *node* の代表となる座標配列は $S_{(i, j)}$[1..*Length*]と表記される．*Count* は，このノード *node* 以下に接続される葉ノードの総数を意味する．

例えば，表 1 の 2 本の座標配列を用いて，MSSD の閾値を b=20 とするとき，図 2 に示されるような幾何学的サフィックス木が得られる．

表 1　座標配列データベースの例

ID	座標配列
1	<(3,5,2),(4,6,3),(4,5,3),(7,7,4),(4,4,2)>
2	<(2,5,1),(3,5,3),(2,7,4),(6,7,3),(3,4,2)>

この例で構築された幾何学的サフィックス木は，７つの中間ノードと８つの葉ノードから成る．図中の中間ノード $node_2$ に記録されているノード情報[(1,1), 3, 4]の(1,1)は, ID が 1 の座標配列で 1 番目の座標点から始まるサフィックス配列 $S_{(1,1)}$ = < (3,5,2), (4,6,3), (4,5,3), (7,7,4), (4,4,2) >であることを意味する．

ノード情報[(1,1), 3, 4]の 3 と 4 はそれぞれ，サフィックス配列(1,1)の先頭から 3 番目までの座標配列が $node_2$ を表現する座標配列 $S_{(1,1)}$[1..3] = < (3,5,2), (4,6,3), (4,5,3) >であり, $node_2$ 以下に接続される葉ノードの総数が 4 であることを意味する. また, 葉ノード $leaf_4$ に記録されている (2,3)は, ID が 2 の座標配列< (2,5,1), (3,5,3), (2,7,4), (6,7,3), (3,4,2) > の 3 番目の座標点(2,7,4)から始まるサフィックス配列 $S_{(2,3)}$ = < (2,7,4), (6,7,3), (3,4,2) >を意味する．

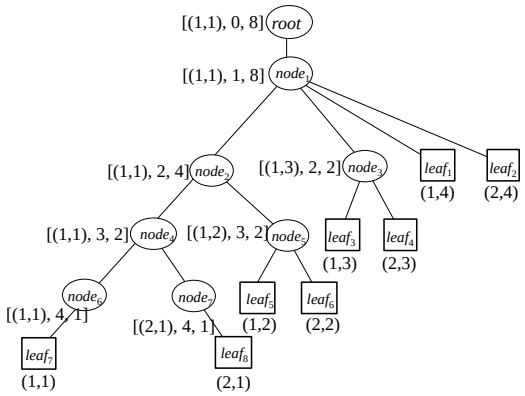

図 2：幾何学的サフィックス木の構築例

幾何学的サフィックス木が分岐の閾値 b の下で構築されているとしよう．ここでは，ノード $node_k$ に記録されているデータを[(i, j), $Length_k$, $Count_k$]とするとき，$Length_k$ を *length* ($node_k$)と表記している．また，根からノード $node_k$ までの経路を表現する座標配列の中で，$S_{(i, j)}$ [1..$Length_k$]を R_k と表記する．構築されている幾何学的サフィックス木から, 長さ m の検索キーを Q とし, 許容誤差を ε とするとき，これらの問合せ条件が満たされる類似部分構造（部分座標配列）を全て検索する手順は以下のとおりである．

（１）根から木をたどり，以下の条件を全て満たす枝 e_{kl} = ($node_k$, $node_l$)を全て見つける．

① $length(node_k) < m$ かつ $length(node_l) \geq m$

② $\mathrm{MSSD}(Q, R_l[1..m]) \leq m\times(\varepsilon+\sqrt{b/m}\ )^2$

（２）中間ノード $node_l$ 以下に存在する葉ノード *leaf* を全て調べ，$\mathrm{RMSD}(Q, T[1..m]) \leq \varepsilon$ を満たす部分構造 T[1..m]をすべて見つける．ただし，T は葉ノード *leaf* を表現するサフィックス座標配列とする．

検索キーQ = <(2, 4, 1), (5, 5, 3), (4, 7, 5)>, RMSD の許容誤差 ε = 2.0 で検索を行ってみよう．先ず，$node_1$ まで探索を行うと，$length(node_1) = 2 < 3$ かつ $length(node_2) = length(node_3) = 3 \geq 3$ により，$node_2$ および $node_3$ は，上記の（１）①の条件を満たす(k=1, l∈{2,3} , m=3)．また，R_2=<(3,5,2), (4,6,3), (4,5,3)>， R_3=<(4,5,3), (7,7,4), (4,4,2)>，$\mathrm{MSSD}(Q, R_2[1..3]) = 8.53$， $\mathrm{MSSD}(Q, R_3[1..3]) = 12.99$，$3(2.0+\sqrt{20/3}\ )^2 = 62.98$ により, $node_2$ および $node_3$ は上記（１）②の条件を満たす．

次に，上記（２）の処理に入ると，$node_2$ 以下の葉ノードは，$leaf_5$，$leaf_6$，$leaf_7$，$leaf_8$ であり，$node_3$ 以下の葉ノードは，$leaf_3$，$leaf_4$ であることがわかる．これらの合計６本のサフィックス座標配列のそれぞれについて，Q との RMSD を計算すると，$\mathrm{RMSD}(Q, S_{(1,2)}[1..3]))=1.41$, $\mathrm{RMSD}(Q, S_{(2,2)}[1..3]))=1.11$, $\mathrm{RMSD}(Q, S_{(1,1)}[1..3]))=1.68$, $\mathrm{RMSD}(Q, S_{(2,1)}[1..3]))=0.81$，$\mathrm{RMSD}(Q, S_{(1,3)}[1..3]))=2.08$，$\mathrm{RMSD}(Q, S_{(2,3)}[1..3]))=1.10$ となる．$node_3$ 以下に存在する２件の葉ノード

のうち，$S_{(1,3)}$ [1..3]は許容誤差 $\varepsilon = 2.0$ を超えるため，この１件だけは類似部分構造ではない．

したがって，この許容誤差以内の部分座標配列は，$S_{(1,2)}$ [1..3]，$S_{(2,2)}$ [1..3])，$S_{(1,1)}$ [1..3])，$S_{(2,1)}$ [1..3]，$S_{(2,3)}$ [1..3]の５件であり，これらが問合せ Q に対して許容誤差 ε=2 以内にある類似部分構造となる．

3.3 隠れ配列

幾何学的サフィックス木の検索では検索キーの長さに対応する深さの中間ノードまで木の探索を行っても，葉ノードを使って類似部分構造を検索するためには，その中間ノード以下に存在するすべての葉ノードを取り出すために探索を行わなければならない．図 2 の $node_2$ 以下の葉ノードを調べると，$node_2$ には { (1, 1), (1, 2), (2, 1), (2, 2) } が含まれることがわかるが，$node_2$ 自体は (1, 1) しか存在しない．すべての部分構造を調べるためには 4 つの中間ノードと 4 つの葉ノードの探索が必要となり，子の探索には時間がかかる．しかし，その中間ノード以下に存在するすべての葉ノードの情報を，隠れ配列として持たせておくことで検索処理を削減できる．$node_2$ では，{ (1, 2), (2, 1), (2, 2) } を隠れ配列として保持しており，これを調べることで中間ノードの探索が削減できる．隠れ配列は(ID，参照開始位置)として表し，その集合を H とする．構築時に中間ノードを通過するとき，その中間ノードの隠れ配列の集合 H に追加し，枝を分割するときには子ノードの隠れ配列を新しいノードにコピーする．

3.4 バッファ管理

幾何学的サフィックス木をディスクベースで扱うために必要なバッファ管理法について説明する．

図 1 のデータページとは中間ノード，葉ノード，隠れ配列を格納したページである．幾何学的サフィックス木のデータである中間ノード，葉ノード，隠れ配列のすべてをディスク上に格納しながら幾何学的サフィックス木の構築，検索を行う． LRU とは，バッファ領域のページの優先度を参照された時間で管理する方法である．この方法はバッファ領域に空きがないとき，バッファ領域のページの中で参照された時間が最も古いページを削除する. TOP-Q とは，バッファ領域のページの管理を二分ヒープで行う TOP と，TOP から削除されたページを持つキューを組み合わせた方法である．この方法は，幾何学的サフィックス木の深さを優先度として持たせることで，幾何学的サフィックス木の浅いデータをバッファ領域に優先的に確保できる．

(1) 葉ノードページ

構築では，書き込み途中のページを管理するために LRU を使用する. 検索では, 提案手法では葉ノードを用いない.

(2) 隠れ配列ページ

構築では，隠れ配列は中間ノードごとに書き込み済みのページと書き込み途中のページが存在するため，書き込み途中のページを管理するために LRU を使用する. 検索では，中間ノードごとにそれぞれの隠れ配列を持つため，一度読み込んで検索に使用した隠れ配列ページを再度検索に使用することはないので，使用後は削除する．このため，バッファ管理は行わない．

(3) カタログページ

カタログページは，一度使用したページをメモリ上に残すために LRU を使用する．

(4) 中間ノードページ

このバッファ管理方式は構築と検索で同じ方式を使用している．中間ノードは幾何学的サフィックス木の構築と検索で，ルートノードに近いノードの参照が多くなるので，幾何学的サフィックス木のノードの深さが浅いノードをバッファに残す必要がある．ここでは，TOP-Q と呼ばれるバッファリング戦略を使用する．

3.5 並列化する際の問題点

従来の幾何学的サフィックス木を並列化するためには２つの問題点がある．1 つ目の問題はノード情報の修正回数の増大である．従来の幾何学的サフィックス木で図 2 の幾何学的サフィックス木を構築すると，$node_2$ はサフィックス(1,1)と(1, 2)を幾何学的サフィックス木に加えた時に作成される．その後すべてのサフィックスが追加されるまでの間に，サフィックス(2, 1)と(2, 2)を追加することでノード情報の修正が２回行われる．さらに，大規模なデータで幾何学的サフィックス木を構築することで，ノード情報の修正回数が増大する．２つ目の問題は，どのように問題分割法と負荷分散法を組み合わせた並列化を行えばよいのかということが明らかにされていないことである．

4. 提案手法

本章では，従来の幾何学的サフィックス木の構築と検索の性能を向上するために，構築方法の改良と並列化の方法を提案する．

4.1 構築方法の改良

構築方法の改良を行うために，トップダウン構築とバッファ管理法の変更について述べる．

4.1.1 トップダウン構築

ノード情報の修正回数を削減するために，従来の構築方法を改良する方法について説明する. 従来の構築方法では，それぞれの座標配列を幾何学的サフィックス木に逐次追加しているが，本研究では座標配列全件をまとめてトップダウンで構築を行う．親ノードに存在するすべての子ノードを作成することを，１つの処理単位としてタスクと呼ぶ．タスクは，タスクプールと呼ばれるタスクを管理する領域におく．タスクの処理は，隠れ配列を見ることで親ノードの深さより１つ深い子ノードの作成を行う. この方法では，作成された中間ノードに対する修正を行うことがない．

また，従来の幾何学的サフィックス木では葉ノードと隠れ配列の両方を用いている．しかし，あるノードでの隠れ配

列とそのノード以下の葉ノードでは，同じサフィックスの情報を含んでいる．そのため，幾何学的サフィックス木の情報から葉ノードを除外することで，構築処理を減らすことができる．

図 2 で用いた例を使い構築方法の説明を行う．図 3(a)は，幾何学的サフィックス木構築の初期段階である．$node_1$ の(1,1)の長さ 5 は length($node_1$)=1 よりも長いので $node_2$ が作成される．次に，$node_1$ の隠れ配列(1,2)は MSSD($node_2$[1..3], H_1,(1,2)[1..3])>20 となるので $node_3$ が作成される．同様に残りの隠れ配列も処理して$node_2$と$node_3$の隠れ配列に含まれる．その結果が図 3(b)となる．(1,4)と(2,4)は長さが 1 で length($node_1$)と同じになり，葉ノードの作成を行わないため，$node_1$ より深い位置には存在しない．

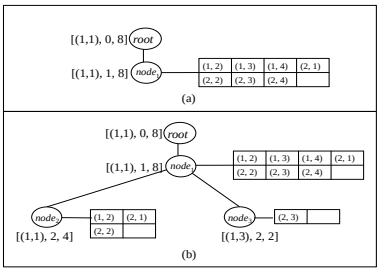

図 3. トップダウンによる構築方法

以上の改良方法による幾何学的サフィックス木のトップダウン構築の処理手順は，以下のとおりである．

(1) タスクプールからタスクを受け取る．

(2) 親ノード$node_k$のサフィックスの長さが m=length($node_k$)より長い場合，$node_k$の深さを 1 増やした子ノードを作成．

(3) $node_k$ の隠れ配列の集合 H_k のすべてに対して(4)を行う．

(4)すべての子ノード $node_l$ に対して MSSD($node_l$[1..m+1], H_k,(i,j)[1..m+1]) が最小となる $node_l$ をみつける．MSSD($node_l$[1..m+1], H_k,(i,j)[1..m+1])<b である場合 H_l に追加，そうでなければ新しい子ノードを作成する．

(5) 子ノードをすべてタスク化し，タスクプールに格納．

4.1.2 バッファ管理法の変更

中間ノードページでは，中間ノードの作成は木の深さが浅いものから行われ，兄弟ノードはページ上に固まって配置される．よって，検索処理で深い中間ノードを含むページが読み込まれず，ディスクからの読み込み時間の短縮が可能となる．

構築方法の改良により，バッファ管理方法も以下のように変更する．データページは中間ノードページだけを扱う．

(1)葉ノードページ

構築方法の改良により，葉ノードの情報を扱わないため，葉ノードページを用いない．

(2)隠れ配列ページ

親ノードの隠れ配列を子ノードに振り分けた時点で，その子ノードの隠れ配列が完成される．よって，隠れ配列ページはディスクとメモリ間のページの読み書きのための処理を行うが，メモリに保持するためのバッファ領域を持たない．

(3)カタログページ

従来手法から変更なし．

(4)中間ノードページ

作成時期が早い中間ノードから処理されるため，中間ノードの深さを考慮した TOP-Q を用いる必要がないので LRU を用いる．

4.2 幾何学的サフィックス木の並列化

幾何学的サフィックス木の並列化を行うために問題分割法と負荷分散法を使用する．問題分割法にはタスク分割とデータ分割がある．タスク分割では，1 つの幾何学的サフィックス木を複数のワーカで処理を分割し，データ分割では，入力データを小さく分割してそれぞれのデータをワーカが処理する．

負荷分散法にはマスタワーカ法と分散型ワーカ法がある．マスタワーカ法は，データを処理するワーカとデータやタスクの管理を行うマスタで構成される．分散型ワーカ法は，データを処理するワーカのみで構成され，タスクの管理は各ワーカが行っている．

本研究では，問題分割法と負荷分散法を組み合わせて以下の 3 種類の並列化を行う．また，幾何学的サフィックス木をデータ分割なしで 1 つ構築する場合と，データ分割ありで複数構築する場合の比較を行うために，分散型ワーカ法+タスク分割は対象としていない．3 種類の並列化による構築と検索のアルゴリズムは以下のとおりである．

4.2.1 マスタワーカ法+タスク分割

構築アルゴリズム

入力：蛋白質立体構造データベース DB

出力：１個のディスクベースの幾何学的サフィックス木 T

（１）幾何学的サフィックス木の根を親ノードとし，親ノードから生成される子ノード$node_i$以降の部分木構築をタスク J と定義する；タスク J のすべてをマスタ側のタスクプール P に格納し，P={J_1, J_2, …, J_N}を得る； ただし，子ノードの総数を N とする．

（２）if $P\neq\varphi$ then マスタ側は空いているワーカにタスク J を割り付け，タスクプールを P=P－{J}とする；

else if 構築処理中のワーカが存在 then タスク J をもらい空いているワーカにタスク J を割り付ける；

（３）各ワーカは，割り付けられたタスク J_i を処理し，部分木構築を行い，その部分木をディスク上に構築する．タスクの処理の終了をマス

タに通知する；

（4）マスタ側はワーカからの通知を受け取る；
if $P\neq\varphi$ then 上記（2）にもどる;
else マスタ側は処理を終了する;

検索アルゴリズム

入力：誤差 ε を許す問合せ Q，1 個のディスクベースの幾何学的サフィックス木 T

出力：Q に類似する部分構造

（1） マスタ側で，T に対応する問合せ Q をタスクと定義する；タスク J のすべてをマスタ側のタスクプール P に格納し，$P=\{J_1, J_2, \ldots, J_N\}$を得る； ただし，子ノードの総数を N とする．

（2）マスタ側が空いているワーカにタスク J_i を割り付ける；タスクプールを $P=P-\{J_i\}$とする；

（3）各ワーカは，割り付けられたタスク J_i を処理し，幾何学的サフィックス木 T_i に対する問合せ Q に類似する部分構造を検索する．タスクの処理の終了をマスタに通知する；

（4）マスタ側はワーカからの通知を受け取る；
if P≠ ϕ then 上記（2）にもどる; else 処理を終了する；

4.2.2 マスタワーカ法+データ分割

構築アルゴリズム

入力：蛋白質立体構造データベース DB，データベースの分割数 N

出力：N 個のディスクベースの幾何学的サフィックス木の集合$\{T_1, T_2, \ldots T_N\}$

（1）マスタ側で，入力データベース DB を N 個のデータセットに分割する；各データセットに対する幾何学的サフィックス木の構築をタスクと定義し，定義されたすべてのタスク J_i をマスタ側のタスクプール $P=\{J_1, J_2, \ldots, J_N\}$に保存する$(1\leqq i\leqq N)$；

（2）マスタ側が空いているワーカにタスク J_i を割り付ける；タスクプールを $P=P-\{J_i\}$とする；

（3）各ワーカは，割り付けられたタスク J_i を処理し，幾何学的サフィックス木 T_i をディスク上に構築する．タスクの処理の終了をマスタに通知する；

（4）マスタ側はワーカからの通知を受け取る；
if $P\neq\varphi$ then 上記（2）にもどる; else 処理を終了する；

検索アルゴリズム

入力：誤差 ε を許す問合せ Q，N 個のディスクベースの幾何学的サフィックス木の集合$\{T_1, T_2, \cdots, T_N\}$

出力：Q に類似する部分構造

（1）マスタ側で，$\{T_1, T_2, \cdots, T_N\}$の各要素に対応する問合せ Q をタスクと定義する；それらのタスク集合をタスクプール P に保存し，$P=\{J_1, J_2, \cdots, J_N\}$を得る；

（2）マスタ側が空いているワーカにタスク J_i を割り付ける；タスクプールを $P=P-\{J_i\}$とする；

（3）各ワーカは，割り付けられたタスク J_i を処理し，幾何学的サフィックス木 T_i に対する問合せ Q に類似する部分構造を検索する．タスクの処理の終了をマスタに通知する；

（4）マスタ側はワーカからの通知を受け取る；
if P≠ ϕ then 上記（2）にもどる; else 処理を終了する；

4.2.3 分散型ワーカ法+データ分割

構築アルゴリズム

入力：蛋白質立体構造データベース DB，データベースの分割数 N

出力：N 個のディスクベースの幾何学的サフィックス木の集合$\{T_1, T_2, \ldots T_N\}$

（1）あるワーカ W で，入力データベース DB を N 個のデータセットに分割する；各データセットに対する幾何学的サフィックス木の構築をタスクと定義し，定義されたすべてのタスク J_i をワーカ W のタスクプール $P=\{J_1, J_2, \ldots, J_N\}$に保存する$(1\leqq i\leqq N)$；

（2）ワーカ W で，あるタスクの処理が終了；
if $P\neq\varphi$ then P からタスク J を取り出し，$P=P-\{J\}$として，タスク J により幾何学的サフィックス木を構築し，（2）の先頭にもどる；
else $P\neq\varphi$ となってる他ワーカにタスクを要求する；
if タスクを取得 then そのタスクをもとに，ディスクベースの幾何学的サフィックス木を構築し，（2）の先頭にもどる；
else ワーカ W の処理を終了する；

検索アルゴリズム

入力：誤差 ε を許す問合せ Q，N 個のディスクベースの幾何学的サフィックス木の集合$\{T_1, T_2, \cdots, T_N\}$

出力：Q に類似する部分構造

（1）あるワーカ W で，$\{T_1, T_2, \cdots, T_N\}$の各要素に対応する問合せ Q をタスクと定義する；定義されたすべてのタスク J_i をワーカ W のタスクプール P に保存し，$P=\{J_1, J_2, \cdots, J_N\}$を得る；

（2）ワーカ W で，あるタスクの処理が終了；
if $P\neq\varphi$ then P からタスク J を取り出し，$P=P-\{J\}$として，タスク J により幾何学的サフィッ

Vol.2014-MPS-98 No.21
Vol.2014-BIO-38 No.21
2014/6/27

クス木を構築し，（2）の先頭にもどる；
else $P \neq \varphi$ となってる他ワーカにタスクを要求する；
if タスクを取得 then そのタスクをもとに，ディスクベースの幾何学的サフィックス木を構築し，（2）の先頭にもどる；
else ワーカ W の処理を終了する；

5. 評価実験

本章では新しく前章で提案した幾何学的サフィックス木の並列化について評価を行う．

5.1 実験方法

提案手法の評価を行うために，wwPDB に 2012 年 1 月 11 日時点で登録されていた PDBML 形式の座標配列データファイルを，PDB の ID に含まれる鎖ごとにアミノ酸配列の Cα原子の座標データを取り出し，評価実験データに使用した．PDB から取得した実験データは 366,146 件，データサイズは 1400MB である．検索キーに使用したデータについては，アミノ酸モチーフは kringle，PDB ID は 1A0H である．分岐パラメータは b = 400，検索パラメータは ϵ = 2.5，ページサイズは 8KB，データページのバッファサイズは 70GB，カタログページのバッファサイズは 70MB とする．実験環境は CPU：2.60GHz× 18 コア，メモリ：192GB，ディスク容量：12TB である．提案手法の優位性を確認するために従来手法に対し比較を行う．

5.2 評価

ここでは，まず，従来手法と提案手法を比較し，提案手法の優位性を確かめる．つぎに，幾何学的サフィックス木の構築と検索の並列化による高速化の効果を調べる．構築と検索の実行時間を調べ，それぞれに適した並列化方法を確かめる．

5.2.1 従来手法との比較

ここでは幾何学的サフィックス木の構築・検索の並列化による高速化の効果を従来手法と提案手法それぞれで調べる．提案手法と条件を合わせるために，並列化の方法として分散型ワーカ法+データ分割を用いた．その結果が図 4 と図 5 となり，提案手法が従来手法に比べ，速度向上比が大きくなっている．このことから，トップダウン構築に変更することで，並列化による構築・検索時間の高速化を確認した．また，検索では，提案手法で従来手法と同じ類似部分構造検索の結果が得られた．

幾何学的サフィックス木ではデータ分割を行っても構築された木のサイズと同じように分割されない．従来手法で速度向上比が減少しているのは，幾何学的サフィックスの中間ノードの修正のときに行われるバッファ管理でデータページの読み書きが集中して起きているからである．

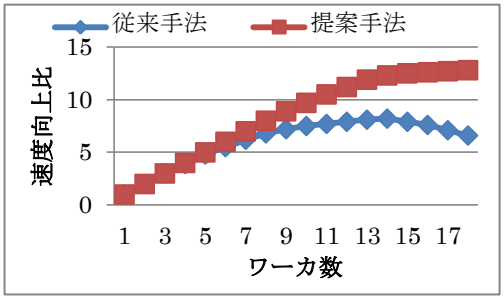

図 4. 従来手法との構築の速度向上比の比較

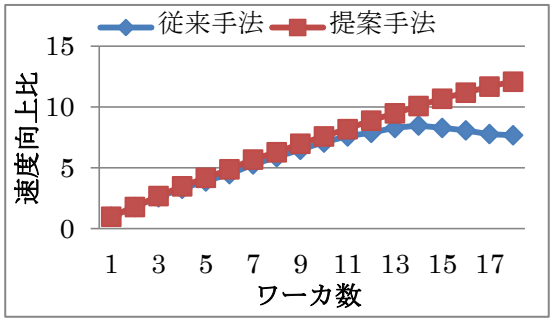

図 5. 従来手法との検索の速度向上比の比較

5.2.2 幾何学的サフィックス木の並列化

ここでは幾何学的サフィックス木の構築・検索の並列化による高速化の効果を提案手法の 3 種類の方法それぞれで調べる．その結果が図 6 と図 7 となり，構築では分散型ワーカ法+データ分割，検索ではマスタワーカ法+タスク分割でそれぞれ速度向上比が大きくなっている．このことから，幾何学的サフィックスの構築では分散型ワーカ法+データ分割，検索ではマスタワーカ法+タスク分割が並列化による高速化の効果が高いと確認できた．また，検索では，並列化手法の 3 種類すべてで従来手法と同じ類似部分構造検索の結果が得られた．

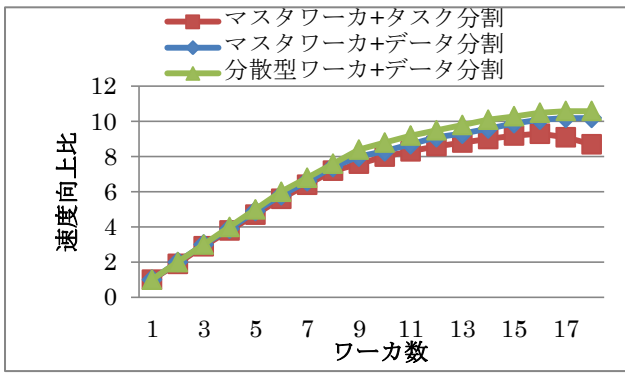

図 6. 構築の速度向上比の比較

ことで処理するデータが小さくなるため，構築時間が減少する．しかし，検索では，データ分割では分割した木のサイズが反比例せず，すべての木のサイズの合計は分割しない場合より大きくなる．よって，木の探索処理が高速化できないため，タスク分割が高速になる．

6. まとめ

本研究では，幾何学的サフィックス木の並列化を行うために２つの問題点を解決する方法を提案した．提案手法は，タスク分割を行うための構築方法の改良，幾何学的サフィックス木の構築と検索の高速化のための並列化の方法より構成されている．提案手法の評価を行うために，wwPDB からアミノ酸で構成された蛋白質の全座標配列を取得し，幾何学的サフィックス木の構築および検索実験を行った．評価実験の結果，高精度性を維持しつつ，大規模なデータに対して高速な類似部分構造検索ができることを確認した．

今後の課題として，タスク分割と分散型ワーカ法を組み合わせた並列化や，より幾何学的サフィックス木の並列化に適した方法の検討などがあげられる．

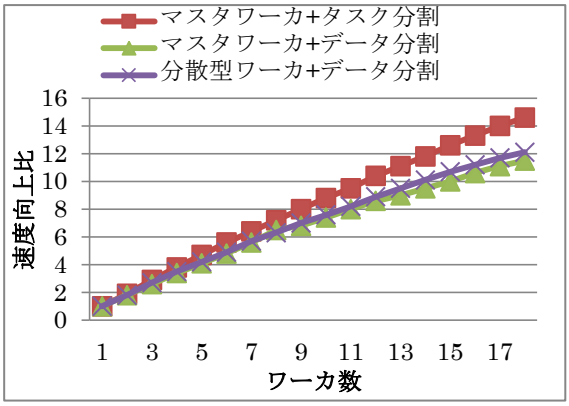

図 7．検索の速度向上比の比較

謝辞

本研究の一部は日本学術振興会・科学研究費補助金(基盤研究(C)，課題番号：20500137 および 26330139)の助成を受けたものです．

参考文献

1) 高橋 誉文，田村 慶一，黒木 進，北上 始：幾何学的なサフィックス木による高速類似構造検索手法，情報処理学会論文誌データベース，Vol.6，No.5，pp.62-70 (2013).
2) Shibuya, T.: Geometric Suffix Tree: A New Index Structure for Protein 3-D Structures, Combinatorial Pattern Matching 2006, LNCS 4009, pp.84-93 (2006).
3) Shibuya, T.: Geometric suffix tree: Indexing protein 3-D structures, J. ACM, Vol.57, No.3, pp.15:1-15:17 (2010).
4) 高橋 誉文，黒木 進，田村 慶一，北上 始: 複数の蛋白質立体構造データに対するディスク版索引構造の構築方式，データ工学と情報マネジメントに関するフォーラム DEIM2009，電子情報通信学会データ工学研究専門委員会, 8 pages (2009).
5) Takahashi, Y., Kuroki, S. and Kitakami, H.: Efficient Query Processing in Protein Structure Databases, Proceedings of the 2nd International Workshop with Mentors on Databases , Web and Information Management for Young Researchers (iDB Workshop 2010), pp.47-55 (2010).
6) Weiner, P., Linear pattern matching algorithms, Proceedings of the 14th Annual Symposium on Switching and Automata Theory (swat 1973), SWAT '73, pp.1-11 (1973).
7) McCreight and Edward, M.: A Space-Economical Suffix Tree Construction Algorithm, J. ACM, Vol.23, No.2, pp.262-272 (1976).
8) Esko Ukkonenn: On-Line Construction of Suffix Trees, Algorithmica, pp.249-260 (1995)
9) R Hariharan: Optimal parallel suffix tree construction,Proceedings of the twenty-sixth annual ACM, pp.290-299 (1994).
10) P Bieganski, J Riedl, JV Cartis: Generalized suffix trees for biological sequence data: Applications and implementation, pp.35-44 (1994).
11) Esko Ukkonenn: On-Line Construction of Suffix Trees, Algorithmica, pp.249-260 (1995).
12) S. Bedathur and J. Haritsa: Engineering a fast online persistent suffix tree construction, In 20th Int'l Conference on Data Engineering, pp.720-731, (2004).
13) Ching-Fung, C., Jeffrey, X. Y. and Hongjun L.: Constructing Suffix Tree for Gigabyte Sequences with Megabyte Memory, IEEE Transactions on Knowledge and Data Engineering, Vol.17, No.1, pp.90-105 (2005).
14) Tian, Y., Tata, S., Hankins, R. A. and Patel, J. M.: Practical methods for constructing suffix trees, VLDB Journal, Vol.14, No.3, pp.281-299 (2005).
15) Benjarath Phoophakdee, Mohammed J. Zaki: Genome-scale Disk-based Suffix Tree Indexing, SIGMOD'07, pp.833-844 (2007).
16) Amol Ghoting, Konstantin Makarychev: Serial and Parallel Methods for I/O Efficient Suffix Tree Construction, SIGMOD'09, pp.827-840, (2009).
17) Yousuke Watanuki, Keiichi Tamura, Hajime Kitakami, and Yoshifumi Takahashi: Parallel Processing of Approximate Sequence Matching using Disk-based Suffix Tree on Multi-core CPU, 2013 IEEE 6th International Workshop on Computational Intelligence and Applications, pp. 137-142 (2013).
18) Yousuke Watanuki, Keiichi Tamura, Hajime Kitakami and Yoshifumi Takahashi: Multiple Buffering for Parallel Approximate Sequence Matching using Disk-based Suffix Tree on Multi-core CPU, GSTF Journal on Computing, JoC Vol.3 No.3, pp.51-57 (2014)
19) 澤田 祐介，田村 慶一，荒木康太郎，高木 允，北上 始: PC クラスタ上でのディスクベースサフィックス木の並列構築方式，DEWS2008, Online proceedings, D2-3, 2008
20) Yusuke Sawada, Keiichi Tamura, Kotaro Araki, Makoto Takaki and Hajime Kitakami, Parallel Construction Method of a Disk-Based Suffix Tree on a PC Cluster, PDPTA'08, 797-803(2008)
21) 澤田 祐介，田村 慶一，荒木康太郎，高木 允，北上 始: 分散並列環境におけるディスクベースサフィックス木の構築と検索，MPS70, pp.39-42 (2008)
22) Benjarath, P and Mohammed J. Z.: TRELLIS+: An Effective Approach for Indexing Genome-Scale Sequences Using Suffix Trees, Pacific Symposium on Biocomputing, pp.90-101 (2008).
23) Arun, K. S., Huang, T. S. and Blostein, S. D.: Least-Squares Fitting of Two 3-D Point Sets, IEEE Trans. Pattern Anal. Mach. Intell., Vol.9, No.5, pp.698-700(1987).
24) Eggert, D. W., Lorusso, A. and Fisher, R. B.: Estimating 3-D rigid body transformations: a comparison of four major algorithms, Mach. Vision Appl., Vol.9, No.5-6, pp.272-290 (1997).
25) Kabsch, W.: A solution for the best rotation to relate two sets of vectors, Acta Crystallographica Section A, Vol.32, No.5, pp.922-923 (1976).
26) Kabsch, W.: A discussion of the solution for the best rotation to relate two sets of vectors, Acta Crystallographica Section A, Vol.34, No.5, pp.827-828 (1987).
27) Schwartz, J. T. and Sharir, M.: Identification of partially obscured objects in two and three dimensions by matching noisy characteristic, Int. J. Rob. Res, Vol.6, No.2, pp.29-44 (1987).
28) Umeyama, S.: Least-Squares Estimation of Transformation Parameters Between Two Point Patterns, IEEE Trans. Pattern Anal. Mach. Intell., Vol.13, No.4, pp.376-380 (1991) .