2008－MPS－70 (10)
2008／7／14

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

澤田 祐介，田村 慶一，荒木 康太郎，高木 允，北上 始
広島市立大学大学院情報科学研究科 〒731-3194 広島市安佐南区大塚東三丁目4番1号

あらまし サフィックス木は分子生物学分野の配列データベースから頻出な配列パターンを抽出するための有用なデータ構造の一つである．しかしながら，サフィックス木は，メモリオーバーヘッドによる性能低下が原因で，大規模な配列データベースのサフィックス木を構築することは難しい．この問題を解決するために，我々は，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並列構築方式を提案する．並列モデルにはマスタ・ワーカモデルを適用する．我々は，提案方式を実装，評価した結果，大規模な配列データベースに対応できることを確認した．また，並列構築された分散サフィックス木の応用として，データマイニング処理の一機能である曖昧検索を行い，その有効性を評価した．

Construction and Query of a Disk-Based Suffix Tree on a Distributed Parallel Environment

Yusuke SAWADA, Keiichi TAMURA, Kotaro ARAKI, Makoto TAKAKI, and Hajime KITAKAMI

Graduate School of Information Sciences, Hiroshima City University, 3-4-1 Ozukahigashi Asaminami-ku, Hiroshima, 731-3194, Japan

Abstract The suffix tree is a useful data structure to extract frequent sequential patterns from the sequence database in the field of molecular biology. However, it is difficult to construct a suffix tree for a large-scale sequence database because of performance degradation due to memory overhead. To solve the problem, the authors propose a parallel construction method of a disk-based suffix tree on a distributed parallel environment. The parallel model is the master-worker model. We implemented and evaluated the proposed method could support a large-scale sequence database. In addition, we applied the distributed suffix tree which was constructed in parallel and evaluated the effectiveness of the ambiguous query that was a part of the data mining process.

1. はじめに

サフィックス木はアミノ酸配列やDNA塩基配列など生物学の配列データベースからデータマイニングを効率的に実行可能なデータ構造である．線形時間でサフィックス木を構築するアルゴリズムとして，Ukkonenのアルゴリズムが広く知られている．しかしながら，サフィックス木は構築に膨大なメモリ領域を必要とするため，メモリオーバーヘッドによる性能低下が問題となっている．ここで，メモリオーバーヘッドとは，メモリ不足から生じる頻繁なディスクアクセスに要する時間のコストを指す．

サフィックス木にはメモリベースの実装とディスクベースの実装が存在する．メモリベースの実装では，少ないメモリ領域と頻繁なディスクアクセスのため大規模な配列データベースのサフィックス木は構築不可能かまたは，構築に現実的でない時間を要する．ディスクベースの実装では，大容量のディスク領域を利用するため，メモリベースでは扱えない大規模な配列データベースを扱うことが可能である．しかし，メモリオーバーヘッドの問題は解消されない．この問題を解決するために，本研究では，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並列構築方式を提案する．分散並列環境では，ネットワークにつながれた複数台のPCから構成されるPCクラスタを用いた．

構築されたサフィックス木はデータマイニングに応用される（サフィックス木の応用に関して

はGusfield[1]を参照)．本研究では，曖昧検索に着目する．配列データベースから並列構築されたサフィックス木を用いて曖昧検索を行い，その有効性を評価する．

本論文の構成は以下の通りである．2章ではサフィックス木に並列構築に関する関連研究について述べる．3章では本研究で適用したディスクベースサフィックス木の構築手法を説明する．4章では3章のディスクベースサフィックス木の並列構築方式を説明し，評価実験の結果を示す．5章では並列構築後のディスクベースサフィックス木の応用とその評価実験の結果を示す．5章で本論分をまとめる．

2. 関連研究

近年，分散並列環境でサフィックス木を並列構築する研究が注目されている．

Chunxi Chen，Bertil Schmidtは，複数のPCをインターネットでつなげたグリッド環境下でメモリベースサフィックス木の並列構築を行っている[2]．PCクラスタ内の並列モデルとして，本研究と同様にマスタ・ワーカモデルを適用している．Chenらは，CPST (Common Prefix Suffix Tree)と呼ばれる新しいデータ構造を提案している．CPSTとはUkkonenのアルゴリズムを並列処理向けに拡張したものである．入力データはDNAの配列データベースを用いており，アルファベット$\{A,C,G,T\}$の4文字で構成された1本の配列を対象としている．

Raphael CliffordのDST (Distributed Suffix Tree)[3]も同様に，複数のPCを用いてメモリベースサフィックス木を構築している．DSTとは，SST(Sparse Suffix Tree)と呼ばれるサフィックス木の部分木の集合である．SSTもUkkonenのアルゴリズムを拡張したデータ構造である．入力データはランダムなバイナリアルファベットを用いており，1本の配列を対象としている．

本研究では，複数本の配列からなる配列データベースからディスクベースサフィックス木を並列構築するため，文献[2]，[3]が対象としているデータとは異なる複数本の配列に対応するサフィックス木の構築手法を提案する．

3. ディクスベースサフィックス木

サフィックス木は，入力データとなる文字列の全てのサフィックス木をトライ構造で表現したデータ構造である．サフィックスとは，文字列tの任意の位置からtの末尾までの範囲の文字列である．本研究では，ディスクベースサフィックス木を扱う．

ディスクベースサフィックス木の構築手法は様々存在するが，本研究では，DynaClusterアルゴリズム[4]を用いる．DynaClusterは動的クラスタリング技術を用いている．この技術は，近いノードは同ページ，もしくは隣接したページに格納され，ノード挿入時に起こる*edge-splitting*と呼ばれる枝の分割によるディスクページアクセス数を抑える工夫がされている．また，DynaClusterは配列1本を対象としているので，n本の配列を処理するには不向きである．本研究では，配列データベースに含まれるn本の配列データDB=$\{t_1,t_2,\ldots,t_n\}$を1本につなげた統合配列Sを作成し，Sに対するサフィックス木を構築する．構築後のサフィックス木は，複数本の配列を用いたサフィックス木と同じ索引構造になる．

以下では，サフィックス木のあるノードとそのノードの子ノードの集合をクラスタと呼ぶ．サフィックス木を深さ優先に構築するために必要な情報は，プレフィックスデータベース（以下，PDB）と呼ばれるテーブルに格納される．PDBはクラスタごとに生成される．ルートノードから深さkのノードまでに出現するk-部分文字列を$<string^k>$とすると，k-部分文字列$<string^k>$に対するPDB($<string^k>$)には，その次に現れる文字$\alpha \in \Sigma$の集合，各文字に対応する数値符号，各文字αが存在するすべての位置情報PListの3種類が格納されている．また，PListには，各$(k+1)$-部分文字列$<string^k\text{-}\alpha>$が存在する統合配列Sのサフィックス番号とSの配列識別子の組が格納されている．

サフィックス木の構築過程において，クラスタごとに作成されるPListの要素数は，サフィックス木のリーフノードに近くなると急激に減少する．このことから，DynaClusterでは，予め最適なPListの要素数を閾値τとして実測し，その値を用いて終端クラスタを作成している．終端クラスタでは，PDBを用いずに，未処理の部分文字列どうしを比較しながら部分木を構築する．閾値τを大きくすると終端クラスタ中で発生するノードの分割の頻度が高まり，小さくすると生成される終端クラスタの数が増加する．DynaClusterの文献[4]によれば，閾値τとして1024が最適であると報告されているので，本研究でもそれを採用する．

4. DynaClusterの並列構築方式

本節では，DynaClusterの並列構築方式を説明する．DynaClusterは，サフィックス木を構築する際，プロセスが完全に独立して構築可能なため，並列処理に適したアルゴリズムといえる．

4.1. 並列構築方式

DynaClusterの並列構築にマスタ・ワーカモデルを適用した．マスタプロセスは，並列処理を管理し，ジョブを作成する．ジョブは，文字列に存在するアルファベット文字を符号化したもので

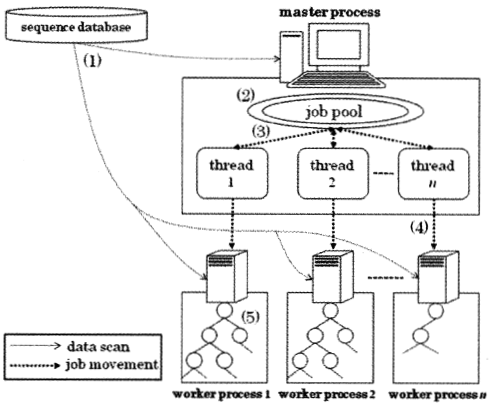

図 1　システム構成図

ある．つまり，20 種類のアルファベット文字で構成されるアミノ酸配列からは，20 個のジョブが生成される．ワーカプロセスは，マスタプロセスから受け取ったジョブを基にサフィックス木を構築する．並列構築されたサフィックス木は，各ワーカプロセスのディスク上に保存される．つまり，並列構築後のサフィックス木は，各ワーカプロセスのディスク上にサフィックス木の部分木として分散している状態である．分散された索引構造は，完全なサフィックス木と同様に利用可能である．

マスタプロセスと各ワーカプロセスのデータの送受信にはソケット通信を利用し，ジョブを動的に管理するためにマルチスレッド方式を適用した．本方式のシステム構成を図 1 に示す．以下，システムの流れを説明する．

(1) 配列データベースを全てのプロセスがスキャンする．
(2) マスタプロセスでジョブを生成し，ジョブをジョブプールに挿入する．
(3) 各スレッドは，ジョブプールからジョブをひとつ取り出す．
(4) 各スレッドは，ワーカプロセスからジョブの要求を受信すると，ワーカプロセスへジョブを送信する．
(5) 各ワーカプロセスは，受信したジョブを基にサフィックス木を構築する．構築が終了したら，再びジョブを要求する．ジョブプールが空になるまで，(3)，(4)，(5)の処理を繰り返し実行する．

4.2. 評価実験

本方式を 4 台の 64bitPC から構成される PC クラスタ上に C 言語で実装し，実験を行った．実験内容は，PC クラスタ上の 4 台の PC を用いてサフィックス木の逐次構築と並列構築の実行時間の性能評価を行った．各 PC の性能を表 1 に示す．本実験ではアミノ酸配列の ZincFinger データセットを用いた．ZincFinger データセットは配列 744 本で構成され，データ総長は 426011byte である．また，21 種類のアルファベット文字で構成されている．約 426KB から約 38GB のサフィックス木が得られる． PC4 台のうち，1 台をマスタプロセス，残りの 3 台をワーカプロセスに割り当てた．

表 1　PC の性能

CPU	Intel Core 2 Duo E6600
Memory	1GB×2
HDD	250GB
OS	Fedora Core 7.0
Network	100Mbit/sec Ethernet

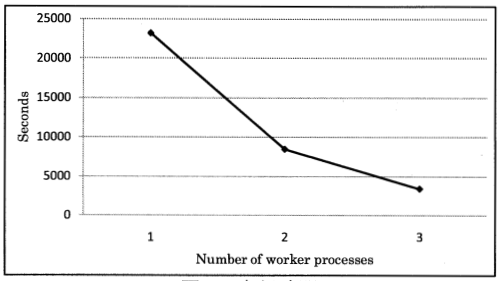

図 2　実行時間

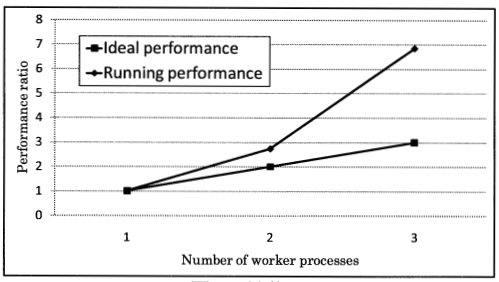

図 3　性能比

実験結果を図 2, 3 に示す．図 2 の実行時間は，すべてのプロセスの処理が終了するまでの時間を示す．図 3 は性能比を示す．図 3 から，ワーカプロセス数 2 台のとき約 2.7 倍，3 台のとき約 6.9 倍の性能が得られた．ワーカプロセス数以上の性能が得られた理由として，大幅な I/O コストの削減が挙げられる．1 台のワーカプロセスでサフィックス木を構築した場合，実行時間の 95%が I/O コストであり，CPU はほとんど稼働していない．ここで，I/O コストとは，I/O による CPU の待機時間のことを指す．本実験から，メモリ領域を増幅することで，大規模な配列データベースを扱えることを確証した．

5. 分散サフィックス木の応用

以下, 並列構築で得られたサフィックス木を分散サフィックス木と呼ぶ. 分散サフィックス木を併合したものは, 完全なサフィックス木と同じ索引構造を持つ. 分散サフィックス木は各々独立して走査することができる.

5.1. 曖昧検索

曖昧検索は, 配列データベースから類似する部分文字列を検索する処理である. 入力パラメータとして, 配列データベース, 検索パターン長 k, 許容誤差 r が与えられる.

5.2. 分散サフィックス木を用いた曖昧検索

サフィックス木を用いた曖昧検索は, サフィックス木を深さ優先に走査することで達成される.

分散サフィックス木を用いた場合は, 図 1 のマスタプロセスが, 入力した検索パターンとその長さ k, 許容誤差 r をワーカプロセスにスレッドを介して送る. ワーカプロセスはそのパラメータを基に曖昧検索を独立して実行し, 抽出したパターンを出力する.

5.3. 評価実験

本実験では, ZincFinger の実験で得られた分散サフィックス木を用いた曖昧検索を行う. 4.2 節の実験環境で, ワーカプロセス 1 台で逐次構築したサフィックス木を基準に, 2 台, 3 台で並列構築した分散サフィックス木を用いた曖昧検索の実行時間の性能を評価した. 検索パターンは ZingFinger モチーフの一部分を与え, 許容誤差は 1 に設定した. 検索パターン長 k=23 である. 本実験では, ワイルドカード領域の変化は考慮せず, 単純に分散サフィックス木を用いた曖昧検索の実行時間に着目している. また, この入力パラメータから抽出されるパターン数は 316 である.

実験結果を表 2, 3, 4 に示す. 表 2 は, 曖昧検索の実行時間と性能比を示す. 表 3, 4 は, 保存された分散サフィックス木のサイズ, 抽出パターン数, 各ワーカプロセスにおける曖昧検索の実行時間を示す. 表 2 から, 分散サフィックス木を用いた曖昧検索の有効性が確認できた. ここで, 各ワーカプロセスの実行時間に着目すると, 表 3, 4 から, 実行時間は, 抽出パターン数よりサフィックス木のサイズに依存していることがわかる. これより, 分散サフィックス木を用いた曖昧検索をより効率的に行うためには, 均等なサイズの分散サフィックス木を構築する必要がある.

表 2　実行時間と性能比

Number of worker processes	Running time (sec)	Performance ratio
1	8248.87	1
2	3314.10	2.5
3	2531.77	3.3

表 3　ワーカプロセス数 2 の実行時間と抽出パターン数

ID	Size (GB)	Number of Extracted patterns	Running time (sec)
1	19.448	298	3212.21
2	18.794	18	3314.07

表 4　ワーカプロセス数 3 の実行時間と抽出パターン数

ID	Size (GB)	Number of Extracted patterns	Running time (sec)
1	12.908	30	2190.32
2	13.889	13	2531.73
3	12.254	273	1905.33

6. おわりに

本研究では, ディスクベースサフィックス木の並列構築方式を提案した. 本方式を PC クラスタ上で実験することでその特性を評価した. そして, 本方式が大規模な配列データベースから効率的にディスクベースサフィックス木を構築できることを確証した. また, 分散サフィックス木の応用として, 分散サフィックス木上で曖昧検索を行い, その並列処理の有効性を確認した.

今後の課題として, 負荷分散手法の実装が挙げられる. 他に, 分散サフィックス木上で, 曖昧な頻出配列パターンの抽出を考慮した曖昧検索を行うことなどが挙げられる.

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)(一般), 課題番号:20500137), および, 文部科学省・科学研究費補助金 (課題番号:18700094) の支援により行われた.

参考文献

[1] D. Gusfield. Algorithm on strings, trees and sequences. *Computer Science and Computational Biology. Cambridge University Press*, 1997.

[2] C. Chen, B. Schmidt. Construction large suffix trees on a computational grid. *Journal of Parallel and Distributed Computing*, Vol.66, No.12, pp.1512-1523, 2006.

[3] R. Clifford. Distributed suffix trees. *Journal of Discrete Algorithms*, Vol.3, pp.176-197, 2005.

[4] C. F. Cheung, J. X. Yu, and H. Lu. Constructing suffix tree for gigabyte sequences with megabyte memory. *IEEE Trans. Knowl. Data Eng*, Vol.17, No.1, pp.90-105, 2005.