分散型ワーカモデルによるModified PrefixSpan法の並列処理とその動的負荷分散手法

高木　允† 田村　慶一†† 周藤　俊秀† 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
†† 広島市立大学情報科学部　〒 731–3194 広島市安佐南区大塚東三丁目 4 番 1 号
E-mail: †{makoto,tosihide}@db.its.hiroshima-cu.ac.jp, ††{ktamura,kitakami}@its.hiroshima-cu.ac.jp

あらまし　複数のアミノ酸配列から頻出パターンを抽出するためのアルゴリズムとして Modified PrefixSpan 法が提案されている．本論文では，分散型ワーカモデルによる Modified PrefixSpan 法の並列処理とその動的負荷分散手法を提案する．並列処理の特徴は，小粒度タスクを用いていることと，分散型ワーカモデルで用いられている動的負荷分散手法である Random Steal 法を改良したことである．本稿では，上記の特徴を説明し，100 台規模の PC クラスタでの性能評価を示す.

キーワード　配列パターン抽出，データマイニング，並列処理，動的負荷分散

Parallel Processing of Modified PrefixSpan with Distributed Worker Model and Its Dynamic Load Balancing Technique

Makoto TAKAKI†, Keiichi TAMURA††, Tosihide SUTOU†, and Hajime KITAKAMI††

† Graduate school of Information Sciences, Hiroshima City University
3–4–1, Ozuka–higashi, Asaminami–ku, Hiroshima, 731–3194 Japan
†† Facaulty of Information Sciences, Hiroshima City University
3–4–1, Ozuka–higashi, Asaminami–ku, Hiroshima 731–3194 Japan
E-mail: †{makoto,tosihide}@db.its.hiroshima-cu.ac.jp, ††{ktamura,kitakami}@its.hiroshima-cu.ac.jp

Abstract　Modified PrefixSpan which extracts frequent patterns in amino acid sequences is proposed. This paper presents a parallel processing of Modified PrefixSpan with distributed worker model and its dynamic load balancing technique. There are two characteristics of the parallel processing. The first is the small-degree-task. The second is to have improved a Random Steal method that is a dynamic load balancing technique used with distributed worker model. This paper explains the above-mentioned characteristics, and presents performance evaluations with the PC cluster of 100 scales.

Key words　Sequential Pattern Extraction, Data Mining, Parallel Processing, Dynamic Load Balancing

1. はじめに

モチーフはアミノ酸配列上における特徴的なパターンであり，生物進化の過程で保存されてきたタンパク質の機能に関係していると考えられている．近年，データマイニングの立場から，アミノ酸配列から取り出した頻出パターンよりモチーフを発見する手法が注目されている．そこで，モチーフとなり得る特徴的なパターンをアミノ酸配列上から発見するために，複数のアミノ酸配列から頻出パターンを高速に抽出することが重要な課題となっている．

複数のアミノ酸配列から頻出パターンを抽出するためのアルゴリズムとして Modified PrefixSpan 法 [1] [2] が提案されている．Modified PrefixSpan 法は PrefixSpan 法 [3] を拡張したものであり，複数のアミノ酸配列から可変長ワイルドカード領域と固定長ワイルドカード領域を含む頻出パターンを抽出することができる．Modified PrefixSpan 法は，PrefixSpan 法に最大ワイルドカード数と最大誤差数と呼ばれるパラメータを追加したもので，優れた抽出能力を持つ.

我々はこれまでに，PC クラスタを対象として Modified PrefixSpan 法の並列処理 [4] とその動的負荷分散手法 [5] に関しての研究をおこなってきた．並列化には典型的なマスタワーカモデル [6] [7] を使用してきた（図 1）．マスタワーカモデルでは，マスタプロセスが仕事を複数のタスクに分割し，タスクを複数のワーカプロセスに割り当てていく．性能評価 [5] により，64

台の PC から構成される PC クラスタにおいて十分なスピードアップを得られることを示した．

しかしながら，大規模な PC クラスタに対応するためにマスタワーカモデルでは次の点が問題となる．PC クラスタにおいて多くの PC を確保できたとしても，ワーカプロセスを統括するマスタプロセスはひとつであり，マスタプロセスが処理全体のボトルネックとなる．Modified PrefixSpan 法の並列処理においてマスタプロセスはタスクをワーカプロセスに割り当てていくだけで，マスタプロセスでの処理は比較的小さいが，PC クラスタ内の PC の台数が増えると，スケーラビリティの面で限界がある．

本論文では，マスタプロセスを配置せずにすべてのプロセスがワーカプロセスである分散型ワーカモデル [7] による Modified PrefixSpan 法の並列処理とその動的負荷分散手法を提案する．分散型ワーカモデルでは，マスタプロセスを配置せず（図 1），ワーカプロセス間で自律分散的に並列処理を進めていくため，PC の台数が多くなったとしても上記の課題は発生しない．

分散型ワーカモデルにおける動的負荷分散手法として頻繁に使用されているのが receiver-initiated ベース [8] の Random Steal（RS）法 [9] である．RS 法では，ワーカプロセスはタスクを終了すると，ランダムに他のワーカプロセスへタスク要求メッセージを出し，他のワーカプロセスからタスクの一部を奪い取る．もし，タスク要求メッセージを受け取ったワーカプロセスにタスクが存在しなければ再び，ランダムに他のワーカプロセスへタスク要求メッセージを出していく．

RS 法を Modified PrefixSpan 法の並列処理にそのまま適用すると次の点が課題となった．Modified PrefixSpan 法では一部のワーカプロセスに負荷が集中することがある（詳しくは 4. で示す）．RS 法では，一部のワーカプロセスに負荷が集中している場合，タスクを抱えたワーカプロセスに辿りつくまでタスク要求のメッセージが多数発生し，タスクが割り当てられるまでの待ち時間が大きくなる．

この課題を解決するために RS 法の改良をおこなった．改良版 RS 法では，最初の 1 回目は，ランダムに他のワーカプロセスへタスク要求メッセージを出し，他のワーカプロセスからタスクの一部を奪い取る．タスクを獲得できたワーカプロセスの識別子を記憶し，次に，ワーカプロセスにおいてタスクがなくなった場合，ランダムではなく先ほど記憶したワーカプロセスにタスク要求メッセージを出す．

改良版 RS 法では，一部のワーカプロセスに負荷が集中した場合，タスクを抱えたワーカプロセスに辿りつくまでタスク要求のメッセージが多数発生することが回避できる．また，負荷の偏りが小さい場合であっても，記録したワーカプロセスからタスクが得られなければ再びランダムに他のワーカプロセスへタスク要求メッセージを出すため， RS 法と遜色なく負荷の偏りを解消することができる．

提案手法を実際の PC クラスタ上で実装し，100 台規模の PC クラスタで性能評価をおこなった．PC を 100 台使用した測定において十分な性能向上比が得られ，分散型ワーカモデル適用の有効性を示すことができた．また，RS 法は台数が増えるごとに徐々に性能の低下がみられたが，改良版の RS 法は性能の低下は無く， RS 法と比べて良い性能向上比が得られた．

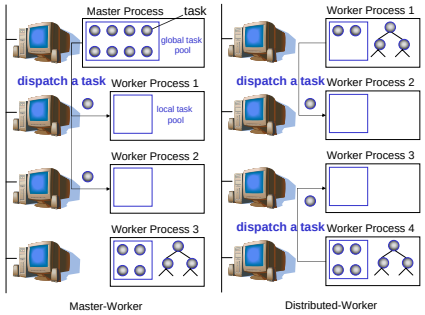

図 1　マスタワーカモデルと分散型ワーカモデル

本論文の構成は以下の通りである．2. では Modified PrefixSpan 法の説明をおこなう．3. では関連研究について述べる．4. では分散型ワーカモデルによる並列 Modified PrefixSpan 法を示す．5. では改良版 RS 法の説明をおこなう．6. で性能評価の結果を示し，7. で本論文をまとめる．

2. Modified PrefixSpan 法

本節では，並列処理の説明で必要となる Modified PrefixSpan 法の大まかな処理内容を説明する．より詳細なアルゴリズムは [2] を参照されたい．

2.1 記号定義

アミノ酸配列データベースを S と表す．1 つのアミノ酸配列を s_{sid} と表すと（sid は配列の識別子である），S はタプル $\langle sid, s_{sid}\rangle$ の集合となる．1 つのアミノ酸配列を $s_{sid} = \langle a_1 a_2 \cdots a_m\rangle$ と表現する．アミノ酸配列 s_{sid} は 20 種類のアルファベットを組み合わせた文字列で構成される．アミノ酸配列に含まれる文字要素を Σ = {A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y } と定義すると，$a_j \in \Sigma$ が成り立つ．

アミノ酸配列 s_{sid} の第 l 番目の文字要素は配列データとして $s_{sid}[l]$ で参照・更新ができる．ここで，表 1 をアミノ酸配列の例として考える．S は 5 つのタプルから構成される．$s_2 = \langle$ SFVKTA $\rangle$ であり，$s_2[1] = \langle$S$\rangle$ である．

長さ k のパターンを以下のように表す．

$$pat^k = \langle A_1 - x(i_1, j_1) - A_2 - x(i_2, j_2) - \cdots - x(i_{k-1}, j_{k-1}) - A_k\rangle.$$

記号 A_j は 20 文字のアルファベットのうち 1 文字の文字要素を示す．記号“ - ”は，隣同士がお互いに連続していることを表現するための記号である．記号 $x(i_n, j_n)$ はワイルドカード領域を示し，i_n，j_n はワイルドカード領域の長さを表す．$x(i_n, j_n)$ は，文字要素 A_{n-1} と A_n の間に長さ i_n から長さ j_n までのワイルドカード領域を含むということを意味する．$i_n = j_n$ のとき，$x(i_n, j_n)$ を $x(i_n)$ と表現する．

例えば，$F{*}{*}{*}K{*}A$ というパターンを考える．文字要素“ * ”は任意の文字要素（ワイルドカード）1 文字を示す．このパターンは $\langle F - x(3) - K - x(1) - A\rangle$ と表現される．$x(3)$ はワイル

表 1　アミノ酸配列データの例

sequence id	sequence
1	FKYAKWL
2	SFVKTA
3	ALR
4	MSKPL
5	FSKFLMAW

ドカード 3 文字が存在することを示している．また，長さ 2 のパターン $\langle F-x(0,3)-A\rangle$ は，$\langle F-x(0)-A\rangle, \langle F-x(1)-A\rangle$, $\langle F-x(2)-A\rangle$，$\langle F-x(3)-A\rangle$ の 4 つの基本パターンのいずれかを含む．

長さ k のパターン $\langle pat^k\rangle$ のアミノ酸配列データベース S における支持数は，パターン $\langle pat^k\rangle$ を含むタプルの数（配列の数）となる．本論文では，頻出パターン抽出時にユーザが指定する最小支持数を min_sup と表す．例えば，長さ 2 のパターン $\langle F-x(0,3)-A\rangle$ の支持数は，$\langle F-x(0)-A\rangle, \langle F-x(1)-A\rangle$, $\langle F-x(2)-A\rangle$，$\langle F-x(3)-A\rangle$ の 4 つの基本パターンのいずれかを含むタプルの数（配列の数）となる．

パターン $\langle pat^k\rangle$ を含むタプルの数が最小支持数以上のとき，パターン $\langle pat^k\rangle$ を k-頻出パターンと呼ぶ．支持数が cnt である k-頻出パターン $\langle pat^k\rangle$ を“ $\langle pat^k\rangle : cnt$ ”と表現する．ここで，n 個の k-頻出パターンが S より見つかったとすると，これらの k-頻出パターンを以下のように定義する．

$$P_k = \{\langle pat_1^k\rangle : cnt_1, \langle pat_2^k\rangle : cnt_2, \cdots, \langle pat_n^k\rangle : cnt_n\}$$

2.2　アルゴリズム

Modified PrefixSpan 法では，k-頻出パターンの最右端に続く 1 文字をすべてアミノ酸配列より取り出し，k-頻出パターンの最右端に結合することにより長さ（$k+1$）のパターンを作成する．もし，作成した長さ（$k+1$）のパターンが最小支持数を満たすならば長さ（$k+1$）のパターンは（$k+1$）-頻出パターンとなる．

k-頻出パターンから長さ（$k+1$）のパターンを作り出すために，アミノ酸配列データベース中に存在するすべての k-頻出パターンについて，各 k-頻出パターンの最右端文字の右隣の位置（オフセット）を記録する．このオフセットの集合を射影データベース（Projected Databases）と呼ぶ．頻出パターン $\langle pat^k\rangle$ の射影データベースの定義は次の通りである．

$$PDB(\langle pat^k\rangle) = \{(sid, pos) | s_{sid} \in S, pos \text{ は } \langle pat^k\rangle \text{ の} \\ \text{最右端文字の右隣の位置}, 1 \leqq pos \leqq \| s_{sid} \|\}$$

Modified PrefixSpan 法では，抽出処理のパラメータとして，最小支持数，最大ワイルドカード数，最大誤差数を指定する．

最大ワイルドカード数は，パターン中に連続して出現してもよいワイルドカードの最大数を示している．例として，最大ワイルドカード数を 2 とした場合を考える．$F**K*A$ は最大ワイルドカード数 2 を満たしているが，$F***K*A$ はワイルドカードが連続して 3 つ出現しているので，最大ワイルドカード数 2 を満たしていない．本論文では，最大ワイルドカード数を max_wc と表し，$i_n \leqq max\_wc$ が成り立つ．また，$\varepsilon = j_n - i_n$ を誤差数と呼び，許容する可変長ワイルドカード領域の領域長の最大値を最大誤差数 ε_{max} と呼ぶ．

以下に Modified PrefixSpan 法の基本アルゴリズムの概要を示す．アルゴリズムは 2 つのフェーズから構成される．

- **フェーズ 1:**

まず，アミノ酸配列データベース S をスキャンし，現れるアルファベット α の支持数を数え上げていく．スキャン時に最右端文字の右隣のオフセットを記憶し，射影データベース $PDB(\langle\alpha\rangle)$ を構築していく．スキャンが終わったときに最小支持数よりも支持数が多いアルファベット α が 1-頻出パターン $\langle pat^1\rangle$ となる．1-頻出パターン $\langle pat^1\rangle$ を P_1 にそれぞれ挿入する．最後に 1-頻出パターン $\langle pat^1\rangle$ に対応する射影データベース $PDB(\langle pat^1\rangle)$ を $PDBLIST^1$ に挿入する．

- **フェーズ 2:**

もし $PDBLIST^k$ が空ならば処理を終了する．$PDBLIST^k$ の中から $PDB(\langle pat^k\rangle)$ $(k \geqq 1)$ を 1 つ取り出す．射影データベース $PDB(\langle pat^k\rangle)$ を使用して k-頻出パターンより長さ（$k+1$）のパターンを作成し，それぞれ支持数を数え上げていく．作成時に最右端文字の右隣のオフセットを記憶し射影作成と支持数の数え上げが終わったときに最小支持数よりも支持数が多い長さ（$k+1$）のパターンが（$k+1$）-頻出パターン $\langle pat^{k+1}\rangle$ となる．（$k+1$）-頻出パターン $\langle pat^{k+1}\rangle$ を P_{k+1} にそれぞれ挿入する．最後に，($k+1$)-頻出パターン $\langle pat^{k+1}\rangle$ に対応する射影データベース $PDB(\langle pat^{k+1}\rangle)$ を $PDBLIST^{k+1}$ に挿入する．

3.　関 連 研 究

頻出パターン抽出の主なアプローチとして *apriori* [10] ベースと *tree projection* [3] ベースのアルゴリズムが提案されている．*tree projection* ベースである PrefixSpan 法は *apriori* ベースのアルゴリズムよりも支持数が小さくなる場合やパターン長が長くなる場合に優位であることが性能評価によって示されている．

また，頻出パターン抽出は非常に時間のかかる処理として知られているため並列化による高速化がおこなわれてきた．*apriori* ベースのアルゴリズムでは様々な並列化手法が存在する [11]〜[14]．一方，*tree projection* ベースのアルゴリズムに関する並列化手法の研究は少なく，Valerie Guralnik と George Karypis がおこなった研究 [15], [16] が唯一あるのみである．

ここでは，本研究と，まず，Valerie Guralnik と George Karypis（以下，Valerie らと略す）がおこなった研究との違いを，研究対象，並列化のモデル，タスクの定義，負荷分散という点でみていく．次に，マスタプロセスのボトルネック解消に関して，階層的マスタワーカ法との比較をおこなう．

3.1　*tree projection* ベースの頻出パターン抽出処理の並列処理に関して

3.1.1　研究対象の比較

Valerie らがおこなった研究では特に対象とするデータを特定していないが，並列化の検討において顧客データの購買履歴の

特徴を考慮している．顧客データの購買履歴は 1 件のデータサイズは高々10 ぐらいだが，件数が何十万件や何百万件に及ぶものである．パターン長はあまり大きくならない（つまり探索木の深さは小さい）．

本研究は，アミノ酸配列データを研究の対象としている．アミノ酸配列データは，件数は多くて高々数千件だが，1 つの配列データのサイズが 100 以上や 1,000 に及ぶものがある．データサイズは大きくないが抽出される最大パターン長は，顧客の購買履歴データのときと比べて非常に長くなることがある．

3.1.2 並列化モデルの比較

Valerie らも分散型ワーカモデルを採用している．対象とする顧客データの購買履歴は大規模になるため，データを，並列処理をおこなうサイトに分割配置した環境下でのタスク分割による並列処理方式を提案している．

アミノ酸配列は顧客データの購買履歴と比べるとデータ量は小さい．本研究では，すべてのワーカプロセスがデータ全体を持つか，もしくは処理の開始時にデータがワーカプロセスに転送されるような環境下でのタスク分割による並列処理をおこなう．

3.1.3 タスク定義の比較

Valerie らの研究では，探索木を部分木に分け，部分木探索をタスクと定義している．タスクはワーカプロセス数のみ作成する．複数のタスクに分割する方法としてタスクの処理量を見積もる方法と各サイトに転送されるデータ量を最小化する方法との 2 種類を提案している．

本研究では，タスクを k-頻出パターンから $(k+1)$-頻出パターンを抽出する処理とする．我々はこのタスクのことを小粒度タスクと呼ぶことにする．k-頻出パターンから $(k+1)$-頻出パターンを抽出する処理を実行する（つまり小粒度タスクを実行する）と，新たに $(k+1)$-頻出パターンから $(k+2)$-頻出パターンを抽出する複数の小粒度タスクが生成される．部分木探索は小粒度タスクの集まりともいえる．

本研究において部分木探索をタスクと定義しなかった理由は以下の通りである．

- Modified PrefixSpan 法は部分木探索の処理量と支持数やアドレステーブルの数に依存しない．よって何らかの見積もりをおこなうことができない．
- ある接頭辞（prefix）に続く頻出パターンが極端に多く抽出されることがあり，タスクを部分木探索とすると負荷の偏りをうまく解消できない．

3.1.4 負荷分散手法の比較

Valerie らの研究では負荷分散の方法として RS 法を用いている．評価で使用している計算機の台数が 16 台（32CPU）で台数が比較的少ないことと，対象とするデータの特徴によりあまり負荷の差が出てくることがないため RS 法でもよい結果が得られている．本研究では，100 台までの測定をおこなっており，台数がそれほど多くない時は RS 法で十分であるが，台数が増加すると性能が低下してしまうことを確認している．そこで，RS 法の改良をおこなう必要があった．

3.2 階層的マスタワーカモデルの検討

マスタワーカモデルにおけるマスタプロセスのボトルネックに関する課題を解決する方法として，階層的マスタワーカモデル [17] がある．階層的マスタワーカモデルでは，複数のグループを作成し，グループごとにマスタワーカ構造を持つ．グループに 1 つのマスタプロセスがあり，マスタプロセスを管理するプロセスとしてスーパバイザプロセスを配置する．この手法を並列 Modified PrefixSpan 法に適用するには次の 2 つの課題がある．

- 各グループ間の負荷の偏りを解消するために階層的マスタワーカモデルでは，マスタプロセス間の負荷の偏りをスーパバイザプロセスが定期的に解消していく．この負荷の偏りの解消はタスクの処理時間を見積もれることが前提となり，並列 Modified PrefixSpan 法にそのまま適用できない．
- ワーカプロセス数が増加していった場合，グループ数を増やさなければ再び各グループ内のマスタプロセスがボトルネックとなる．また，グループ数を増やすとマスタプロセスを管理するスーパバイザプロセスがボトルネックとなる．

4. 分散型ワーカモデルによる並列化

4.1 予備実験による解析

Modified PrefixSpan 法による頻出パターン抽出処理の特徴を示すために予備実験による解析をおこなった．Kringle データセット（データセットの説明は 6. を参照のこと，使用した計算機は 6.1 で示す PC クラスタを構成する PC）を使用し，最小支持数 59（%に換算すると 85%），最大ワイルドカード数 9，最大誤差数 3 で抽出時間を解析した．

図 2 に 1-頻出パターン毎に，その 1-頻出パターンより 2-頻出パターン以降の頻出パターンを抽出するのに必要となった処理時間を示す．例えば C の部分は接頭辞として“ C ”を持つ頻出パターンをすべて取り出すために必要となった抽出処理時間を示す．このグラフから分かるように，“ C ”，“ G ”，“ Y ”に抽出処理時間が偏っている．ただし，実際に抽出をおこなってみて分かった結果であり，はじめからこれらの接頭辞に続く処理が多いとは判断できない．パラメータを変えることでこの特徴も様々変化することを確認している．

また，図 3 に“ C ”から取り出された 2-頻出パターンから 3-頻出パターン以降の頻出パターンをすべて取り出すための総抽出処理時間を示す．グラフでは便宜上，横軸は 2-頻出パターンを番号で示している．図 3 は，図 2 の“ C ”の部分の内訳を示していることになる．図のグラフから分かるように処理時間が均一に分かれるのではなく 1-頻出パターンのときと同様に抽出時間が偏っていることが分かる．図 3 以降に“ G ”の内訳を同様に示すが，“ C ”と同様に偏っていることが分かる．

4.2 タスク分割とタスクの定義

負荷の偏りが大きいことと抽出処理時間を予め見積もることができないため，Modified PrefixSpan 法の並列化では小粒度タスクをタスクとして採用した（詳しくは 3.1.3 を参照のこと）．小粒度タスクを用いることで，ワーカプロセス間で極端に負荷の偏りがあったとしてもタスクを RS 法により均等にできる．

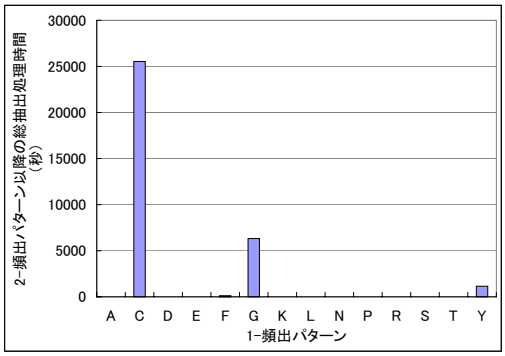

図 2　1-頻出パターン毎の 2-頻出パターン以降の総抽出処理時間（Kringle データセット）

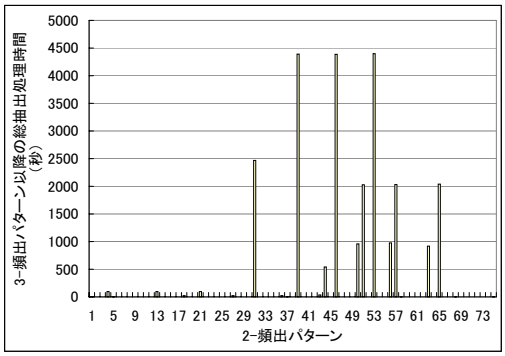

図 3　2-頻出パターン毎の 3-頻出パターン以降の総抽出処理時間（接頭辞が C）

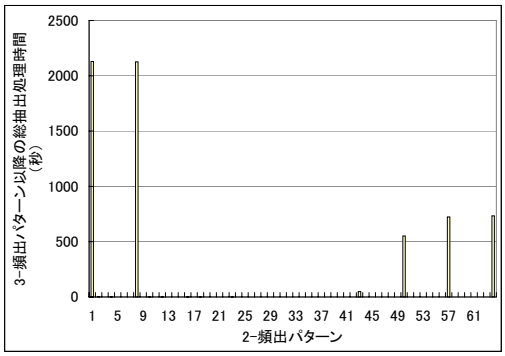

図 4　2-頻出パターン毎の 3-頻出パターン以降の総抽出処理時間（接頭辞が G）

図 5 に例を示す．まず，1-頻出パターンを抽出すると，5 つの頻出パターンが取り出されるのでそれぞれを 1 つのタスクとみなし，5 つのタスクが生成される．タスク 1 を実行すると，頻出パターンは取り出されないので，新たにタスクは生成されない．タスク 2 を実行すると，5 つの 2-頻出パターンが取り出されるため，それぞれをタスク 6 からタスク 10 とする．

小粒度タスクは完全に独立実行することができる．k-頻出パターンから $(k+1)$-頻出パターンを抽出するとき，k-頻出パターンを抽出するのに至った処理内容や他の k-頻出パターンに関する情報を必要としない．これは，$(k+1)$-頻出パターンを抽出する際に $\langle pat^k \rangle$ の最右端文字以降の文字を調べるだけでよいからである．$\langle pat^k \rangle$ の最右端の文字の右隣のオフセットを記憶しているのは，$PDB(\langle pat^k \rangle)$ である．よって，k-頻出パターン $\langle pat^k \rangle$ から $(k+1)$-頻出パターンを抽出するためには，$\langle pat^k \rangle$ と $PDB(\langle pat^k \rangle)$ があればよい．

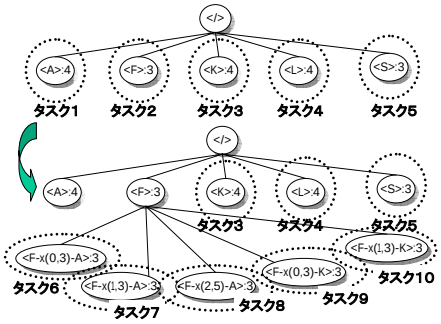

図 5　タスク分割とタスク定義の例

4.3 処理手順

各ワーカプロセスの処理手順は以下通りである．問題を解き始めるワーカプロセスが 1 つ必要であり，そのワーカプロセスをリーダと呼ぶことにする．リーダ以外は処理手順（3）よりスタートする．

（1） リーダとなったワーカプロセスは 1-頻出パターンを抽出する．1-頻出パターンを解の集合 P_{local} に挿入する．

（2） リーダとなったワーカプロセスは，抽出した 1-頻出パターンから 2-頻出パターンを抽出する処理を 1 つの小粒度タスクとしてローカルタスクプールにその小粒度タスクを挿入する．小粒度タスクのデータ表現は $\langle pat^1 \rangle$ と $PDB(\langle pat^1 \rangle)$ とのペアとなる．

（3） ローカルタスクプールより小粒度タスクをひとつ取り出す．もし，ローカルタスクプールが空であれば，手順（5）に進む．取り出した小粒度タスクのデータ表現が $\langle pat^k \rangle$ と $PDB(\langle pat^k \rangle)$ とのペアであったとする．k-頻出パターンより $(k+1)$-頻出パターンを抽出し，$(k+1)$-頻出パターンから $(k+2)$-頻出パターンを抽出する処理を 1 つの小粒度タスクとしてローカルタスクプールに挿入する．$(k+1)$-頻出パターンを解の集合 P_{local} に挿入する．

（4） 手順（3）に戻る．

（5） ランダムにワーカプロセスを選択する．

（6） 選択したワーカプロセスにタスク要求メッセージを送信する．

（7）タスク要求メッセージの返事により次の処理をおこなう．

（a） タスク要求メッセージの返事として小粒度タスクが返ってきた場合，小粒度タスクをローカルタスクプールに挿入する．手順（3）に戻る．

（b） タスク要求メッセージの返事として小粒度タスクが返ってこなかった場合，ランダムにワーカプロセスを選択する（ただし，すでにタスク要求メッセージを送ったワーカプロセスは選択しない）．手順（6）に戻る．もし，すべてのワーカプロセスにタスク要求メッセージを送信済みで返事として小粒度タス

クが返ってこない場合は，手順（8）へ進む．

（8）ワーカプロセスは終了処理に入る．P_{local} をリーダであるワーカプロセスに送信する．リーダであるワーカプロセスはすべてのワーカプロセスより P_{local} を受け取り，自身の P_{local} に挿入する．

ワーカプロセスは上記の通常処理に加えて，タスク要求メッセージを受け取ると以下の処理をおこなう．タスク要求メッセージを受け取るタイミングはいろいろ考えられるが，実装依存であるためここでは手順のみを示す．詳しい実装方法は 6. 性能評価の部分で示す．

（1） ローカルタスクプールが空であるかどうか調べる．空であれば手順（2）へ進む．空でなければ手順（3）に進む．

（2） ワーカプロセスで現在，最後の小粒度タスクを実行中であれば，抽出処理の終わるのを待ち，(1）に戻る．そうでなければ，タスク要求メッセージの送信元にタスクなしを示す返事を送信する．

（3） ローカルタスクプールより小粒度タスクを取り出し，タスク要求メッセージの送信元に返事として取り出した小粒度タスクを送信する．

5. Random Steal（RS）法の改良

並列 Modified PrefixSpan 法では一部のワーカプロセスに負荷が集中することがある（4.1 で示した）．RS 法は，一部のワーカプロセスに負荷が集中している場合，タスクを抱えたワーカプロセスに辿りつくまでタスク要求のメッセージが多数発生し，タスクが割り当てられるまでの待ち時間が大きくなる．

改良版 RS 法では，RS 法同様に最初の 1 回目は，ランダムにワーカプロセスにタスク要求メッセージを出し，他のワーカプロセスのローカルタスクプールより小粒度タスクを奪い取る．ここで，小粒度タスクを獲得できたワーカプロセスの識別子を記憶し，次に，ワーカプロセスにおいてタスクがなくなった場合，ランダムではなく先ほど記憶したワーカプロセスにタスク要求メッセージを出す．一部のワーカプロセスに負荷が集中した場合，上記の変更により，タスクを抱えたワーカプロセスに辿りつくまでのタスク要求メッセージ発生数を少なくすることができる．

以下，改良版 RS 法により各ワーカプロセスの処理手順で変更をおこなった部分を示す．タスク要求メッセージを受け取ったときの処理内容は全く同じであるため省略する．

（5） タスクを獲得できたワーカプロセスの識別子を記憶する変数として lasted を用意する．最初 lasted の値は-1 に設定される．変数 lasted が-1 であれば，ランダムにワーカプロセスを選択する．もし変数 lasted が 0 以上であれば，変数 lasted が示す識別子を持つワーカプロセスを選択する．

（6） 選択したワーカプロセスにタスク要求メッセージを送信する．

（7）タスク要求メッセージの返事により次の処理をおこなう．

（a）タスク要求メッセージの返事として小粒度タスクが返ってきた場合，タスクをローカルタスクプールに挿入する．タスク要求メッセージを送信したワーカプロセスの識別子を変数 lasted に保存する．手順（3）に戻る．

（b）タスク要求メッセージの返事として小粒度タスクが返ってこなかった場合，ランダムにワーカプロセスを選択する（ただし，すでにタスク要求メッセージを送ったワーカプロセスは選択しない）．手順（6）に戻る．もし，すべてのワーカプロセスにタスク要求メッセージを送信済みで返事として小粒度タスクが返ってこない場合は，手順（8）へ進む．

小粒度タスクを採用せず，部分木探索を一つのタスクとみなす場合を考える．タスク処理中であるワーカプロセス A のタスクプールにタスクが存在していない場合，ワーカプロセス B がワーカプロセス A にタスクを要求してもタスクを得ることができず，ワーカプロセス A のタスク処理が終了するまで待たされる．ワーカプロセス A の負荷が大きな場合でも負荷が分散されず，効果的な台数効果が期待できない．また，ワーカプロセス B がワーカプロセス A の識別子を覚えておいても，タスクを奪い取れないので改良版 RS 法の動的負荷分散がうまく機能しない．

1 つのタスクを小粒度タスクとみなした場合，小粒度タスクを実行し生成された小粒度タスクをタスクプールに挿入することを繰り返すので，ワーカプロセス B はワーカプロセス A からすぐにタスクを奪い取ることができ，負荷を分散することができる．小粒度タスク，改良版 RS 法を用いることで負荷の偏りが大きなプロセスからタスクを奪い取ることができ，各ワーカプロセス間の負荷を均一にすることができる．

6. 性 能 評 価

6.1 実 験 環 境

実験では，100 台の PC からなる PC クラスタを用いた．PC クラスタの構成は次の通りである．Intel Pentium4 プロセッサ 2.8GHz，1.0GB メモリを搭載し，各計算機は 1,000 Mbit/sec イーサネットスイッチで接続されている．OS には Fedora 2.0 を使用し，通信にはソケット通信と MPICH の version 1.2.5 を MPI ライブラリとして使用した．

この実験で使用した配列データは PROSITE [18] が提供している，Kringle というモチーフを含む配列データ（以下，Kringle データセットと呼ぶ），Zinc Finger というモチーフを含む配列データ（以下，Zinc Finger データセットと呼ぶ），Liucine というモチーフを含む配列データ（以下，Liucine データセットと呼ぶ）を使用した．

Kringle データセットの詳細は，データ件数：70 件，総長：23385 バイト，平均長：334 バイト，最大長：3176 バイト，最小長：53 バイトである．Zinc Finger データセットの詳細は，データ件数：467 件，総長：245595 バイト，平均長：525 バイト，最大長：4036 バイト，最小長：34 バイトである．Leucine データセットの詳細は，データ件数：370 件，総長：139422 バイト，平均長：376 バイト，最大長：3224 バイト，最小長：3 バイトである．

6.2 実　　装

各プロセスはマルチスレッドで実装をおこなった．通信をおこなうスレッドと，タスクを実行するスレッドに分ける．タス

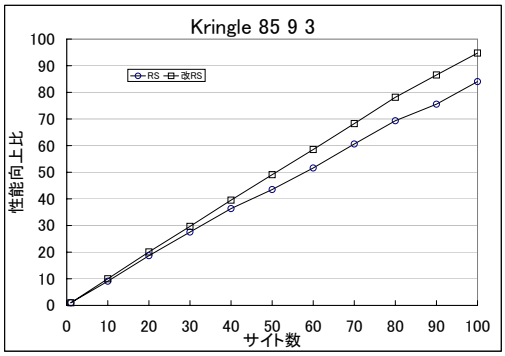

図 6　性能向上比（Kringle データセット）

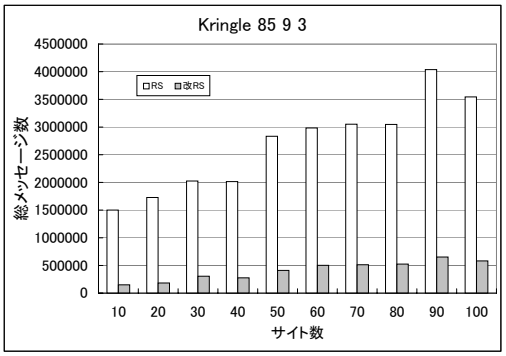

図 9　タスク要求メッセージ総数（Kringle データセット）

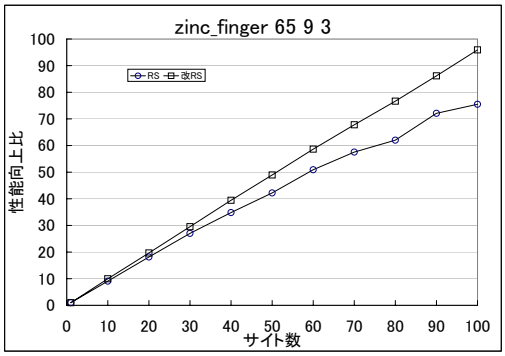

図 7　性能向上比（Zinc Finger データセット）

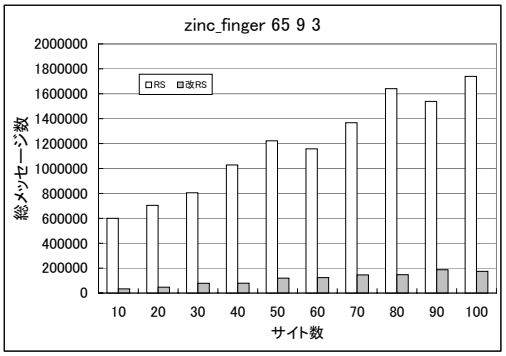

図 10　タスク要求メッセージ総数（Zinc Finger データセット）

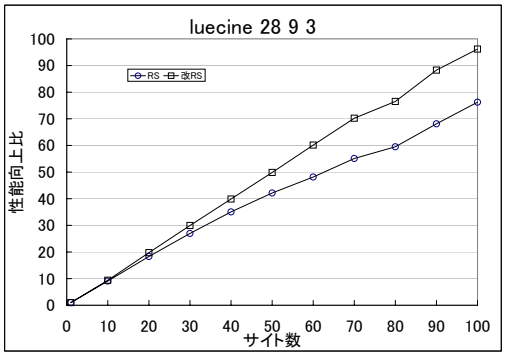

図 8　性能向上比（Leucine データセット）

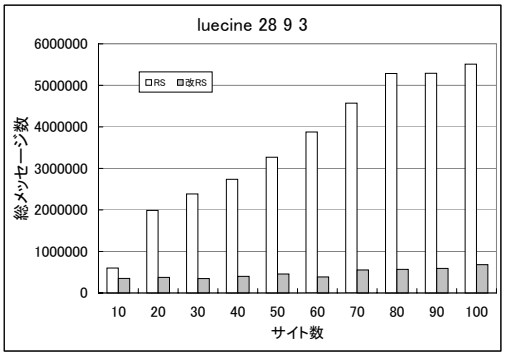

図 11　タスク要求メッセージ総数（Leucine データセット）

ク要求のメッセージは，タスクの実行とは独立に通信をおこなうスレッドにより受け取られ，タスクの奪い取りがおこなわれる．また，最後のタスクを展開中にタスク要求のメッセージを受け取った場合に，タスクなしと判断されないようなメカニズムを組み込んだ．

6.3　性能向上比

Kringle データセットに対して，最小支持数 59（%に換算すると 85%），最大ワイルドカード数 9，最大誤差数 3 で測定をおこなったときの性能向上比を図 6 に示す．グラフは横軸が使用したサイト数（PC の台数）で，縦軸が（逐次での処理時間）/（使用した台数で並列処理したときの処理時間）で性能向上比を示している．グラフの凡例で RS が通常の RS 法で測定したもので，改 RS が改良版の RS 法で測定をしたものである．

いずれの結果も，RS 法と改良版の RS 法ともに良い性能向上比が得られているが，RS 法に比べて改良版の RS 法はさらに良い性能向上比が得られている．改良版の RS 法でサイト数が 100 のときに，約 95 倍近くの性能向上比が得られた．

同様に，Zinc Finger データセットに対して，最小支持数 303（%に換算すると 65%），最大ワイルドカード数 9，最大誤差数 3 で測定をおこなったときの性能向上比を図 7，Leucine データセットに対して，最小支持数 103（%に換算すると 28%），最大ワイルドカード数 9，最大誤差数 3 で測定をおこなったときの性能向上比を図 8 に示す．いずれの結果も，Kringle データセットと同様の結果が得られた．

各パラメータ（最小支持数，最大ワイルドカード数，最大誤差数）は，100 台規模の PC クラスタで評価をおこなっても処理量が少なくなったことが原因で性能向上比がサチュレートしないような値に設定した．上記の Kringle データセットでの評

価において，最小支持数を 63（%に換算すると 90%）に変更すると，60 台使用したあたりから性能向上比がサチュレートし，100 台で最高 80 倍程度の性能向上比を得る結果となった．これは，抽出される頻出パターン数（仕事量）が少ないために最小支持数 59 の場合に比べ性能向上比が低下したと考えられる．Zinc Finger データセットでも同様で，最小支持数を 327（%に換算すると 70%）に設定すると 100 台で 60 倍程度の性能向上比しか得られていない．

6.4 タスク要求メッセージ総数の比較

RS 法と改良版の RS 法を比較するために，発生したタスク要求メッセージの総数を測定した．図 9 に Kringle データセット，図 10 に Zinc Finger データセット，図 11 に Leucine データセットでのタスク要求メッセージ総数を示す．

グラフから明らかに分かるように RS 法の方がタスク要求メッセージの総数が，改良版の RS 法と比べ多い．RS 法ではランダムにワーカプロセスを指定するために，負荷の偏りが大きい場合，タスクを持っているワーカプロセスにたどり着くまでの試行回数が増加し，タスク要求メッセージの総数が大きくなったと考えられる．試行回数が大きくなると，次のタスクが割り当てられるまでの待ち時間が大きくなり結果として性能の低下を招いてしまう．

7. ま と め

本論文では，分散型ワーカモデルによる Modified PrefixSpan 法の並列処理方式とその負荷分散手法について提案した．Modified PrefixSpan 法が対象とするアミノ酸配列からの頻出パターン抽出処理は極端な負荷の偏りがあり，どのように負荷が偏るかは実際に実行してみないと分からない．そこで，従来の研究よりもタスク定義を変え，柔軟に負荷分散がおこなえるようにした．

また，分散型ワーカモデルの負荷分散手法として提案されている RS 法は Modified PrefixSpan 法の並列化に適用すると，タスクにたどり着くまでの待ち時間が大きくなり性能の低下をまねくという課題があった．そこで，この課題を解決するために改良版の RS 法を提案し，改良版の RS 法により RS 法の課題を解決できることを性能評価により確認した．

これからの課題として種々のパラメータを用いての評価と，さらに時間のかかる抽出処理に関してグリッドコンピューティングへの展開をおこなっていく予定である．

謝　　辞

本研究の一部は広島市立大学・特定研究費（一般研究費（コード番号：3106））， 文部科学省科学研究費補助金（課題番号：16700114）の支援により行われた．

文　　献

[1] Hajime Kitakami, Tomoki Kanbara, Yasuma Mori, Susumu Kuroki, and Yukiko Yamazaki. Modified prefixspan method for motif discovery in sequence databases. In *PRICAI2002, Proceedings of Trends in Artificial Intelligence, 7th Pacific Rim International Conference on Artificial Intelligence*, Vol. 2417 of *Lecture Notes in Computer Science*, pp. 482–491. Springer, 2002.

[2] 塔野薫隆, 北上始, 田村慶一, 森康真, 黒木進. Modifiled prefixspan 法を用いた頻出正規パターンの抽出をめざして. 日本データベース学会 Letters, Vol. 3, No. 1, pp. 61–64, 2004.

[3] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proceedings of the 17th International Conference on Data Engineering*, pp. 215–224. IEEE Computer Society, 2001.

[4] Toshihide Sutou, Keiichi Tamura, Yasuma Mori, and Hajime Kitakami. Design and implementation of parallel modified prefixspan method. In *High Performance Computing, Proceedings of 5th International Symposium,ISHPC*, Vol. 2858 of *Lecture Notes in Computer Science*, pp. 412–422. Springer, 2003.

[5] Makoto TAKAKI, Keiichi TAMURA, Tosihide SUTOU, and Hajime KITAKAMI. Dynamic load balancing for parallel modified prefixspan. In *Proceedings of The 2004 International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA 2004)*, pp. 352–358. CSREA Press, 2004.

[6] Nicholas Carriero and David Gelernter. How to write parallel programs: a guide to the perplexed. *ACM Computing Surveys*, Vol. 21, No. 3, pp. 323–357, 1989.

[7] Barry Wilkinson and Michael Allen. *Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hall, 1999.

[8] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. A comparison of receiver-initiated and sender-initiated adaptive load sharing (extended abstract). In *Proceedings of the 1985 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pp. 1–3. ACM Press, 1985.

[9] R. Blumofe and C. Leiserson. Scheduling multithreaded computations by work stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico.*, pp. 356–368, November 1994.

[10] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp. 207–216. ACM Press, 1993.

[11] Rakesh Agrawal and John C. Shafer. Parallel mining of association rules. *IEEE Trans. Knowl. Data Eng.*, Vol. 8, No. 6, pp. 962–969, 1996.

[12] Takahiko Shintani and Masaru Kitsuregawa. Parallel mining algorithms for generalized association rules with classification hierarchy. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, pp. 25–36. ACM Press, 1998.

[13] Masahisa Tamura and Masaru Kitsuregawa. Dynamic load balancing for parallel association rule mining on heterogenous pc cluster systems. In *Proceedings of 25th International Conference on Very Large Data Bases*, pp. 162–173. Morgan Kaufmann, 1999.

[14] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. Efficient parallel and data mining for association rules. In *CIKM '95, Proceedings of the 1995 International Conference on Information and Knowledge Management, November 28 - December 2, 1995, Baltimore, Maryland, USA*, pp. 31–36. ACM, 1995.

[15] V. Guralnik and G. Karypis. Dynamic load balancing algorithms for sequence mining, technical report 00-056, department of computer science, university of minnesota (2001), 2001.

[16] Valerie Guralnik and George Karypis. Parallel tree-projection-based sequence mining algorithms. *Parallel Comput.*, Vol. 30, No. 4, pp. 443–472, 2004.

[17] Kento Aida, Wataru Natsume, and Yoshiaki Futakata. Distributed computing with hierarchical master-worker paradigm for parallel branch and bound algorithm. In *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pp. 156–163. IEEE Computer Society, 2003.

[18] http://kr.expasy.org/prosite/.