Modified PrefixSpan 法の並列化と動的負荷分散手法

高木 允[†] 田村慶一[††]
周藤俊秀[†,☆] 北上始[††]

モチーフはアミノ酸配列中に存在する特徴的なパターンであり，生物学的に意味があると考えられている．アミノ酸配列中に存在する頻出パターンからモチーフを発見することができる．アミノ酸配列中の頻出パターンを効率的に発見するために，高速な頻出パターン抽出アルゴリズムが求められている．本論文では，PC クラスタ上でアミノ酸配列から頻出パターンを並列に抽出する並列 Modified PrefixSpan 法を示し，その動的負荷分散手法を提案する．並列 Modified PrefixSpan 法は PC クラスタ間でタスクを分配する手法であり，マスタ・ワーカ型の並列処理を用いている．Modified PrefixSpan 法では，タスクの負荷に非常に大きな偏りがあり，さらにタスクの処理時間を見積もることができない．このような状況下での動的負荷分散手法として，マスタ・タスク・ステイル法を提案する．マスタ・タスク・ステイル法は，タスク粒度をできるだけ細かくし，負荷の偏りが生じた時点でのみマスタプロセスがワーカプロセスのタスクプールからタスクを集める手法である．

Parallelization and Dynamic Load Balancing for Modified PrefixSpan

Makoto Takaki,[†] Keiichi Tamura,[††] Toshihide Sutou[†,☆] and Hajime Kitakami[††]

A motif is the featured pattern which is biologically meaningful in the amino acid sequences. The motif is discovered from the frequent patterns. In order to extract the frequent patterns that can become motifs in the amino acid sequences efficiently, a high-speed frequent pattern extraction algorithm is required. In this paper, a parallel Modified PrefixSpan which extracts frequent patterns in parallel on an actual PC cluster is presented. Then the dynamic load balancing for the parallel Modified PrefixSpan is proposed. The parallel Modified PrefixSpan exploits a master-worker parallelism that distributes tasks among the computers on the PC cluster. In the Modified PrefixSpan, the bias of load of task is very large. Moreover, the processing time of the task cannot be estimated. A master-task-steal methodology is proposed for the dynamic load balancing technique under such situation. The master-task-steal methodology is the technique which gathers all tasks located in the worker processes' task pool only when the bias of the load arises.

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．モチーフを見つけ出すことでタンパク質の機能や機能部位を推定することができる．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特徴的な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，モチーフとなりう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抽出することが重要な課題となる．

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] が提案されている．Modified PrefixSpan 法は Prefix Span 法[2),3)] を拡張したものであり，複数のアミノ酸配列から固定長ワイルドカードを含む頻出パターンを抽出することができる．

本論文では PC クラスタ上の Modified PrefixSpan 法の並列処理[4),5)] を示し，その動的負荷分散手法について述べる．配列データから頻出パターンを抽出する処理は時間のかかる問題として知られており，実用的な時間で頻出パターンを抽出することが求められてい

† 広島市立大学大学院情報科学研究科
Graduate School of Information Sciences, Hiroshima City University
†† 広島市立大学情報科学部
Faculty of Information Sciences, Hiroshima City University
☆ 現在，NEC システムテクノロジー株式会社
Presently with NEC System Technologies, Ltd.

る．特にアミノ酸配列は規模が大きくなくてもパターンが長くなるため，頻出パターン抽出に非常に時間がかかる．PC クラスタでの並列化により，これまで抽出できなかったような頻出パターンが実用的な時間で抽出できるようになると期待される．

並列 Modified PrefixSpan 法では典型的なマスタ・ワーカ型[6)~8)] により並列処理を行う．典型的なマスタ・ワーカ型による並列 Modified PrefixSpan 法では，長さ k の頻出パターンから長さ $(k+1)$ 以降の頻出パターン抽出処理をタスクと定義する．本論文では，このタスクを大粒度タスクと呼ぶ．ユーザが定めた閾値 k までマスタプロセスで頻出パターンの抽出を行い，生成された大粒度タスクをマスタプロセスのタスクプールに配置する．タスクプール中の大粒度タスクを 1 つずつワーカプロセスに割り当て，大粒度タスクを早く終えたワーカプロセスに大粒度タスクを次々に割り当てていく．

マスタ・ワーカ型による並列処理では，個々の大粒度タスクの処理時間に極端な差がなければ，負荷の偏りは発生しない．しかしながら，並列 Modified PrefixSpan 法における大粒度タスクの処理時間はアミノ酸配列の特徴に依存し，極端に大きな差が発生する．この特徴により PC 間の負荷が偏り，十分なスピードアップを得ることができないという問題がある．

上記の問題点を解決するために，並列 Modified PrefixSpan 法の動的負荷分散手法として，マスタ・タスク・ステイル法を提案する．マスタ・タスク・ステイル法の特徴は，以下の 3 つである．

(1) ワーカプロセスにもタスクプールを配置
(2) 小粒度タスクを採用
(3) マスタプロセスによるワーカプロセスからのタスク奪い取り

本論文では長さ k から長さ $(k+1)$ の頻出パターンを抽出する処理を小粒度タスクと定義する．小粒度タスクを実行すると新たな小粒度タスク（長さ $(k+1)$ から $(k+2)$ の頻出パターン抽出処理）が生成される．マスタプロセス以外に，すべてのワーカプロセスにタスクプールを持たせ，ワーカプロセスは新たに生成された小粒度タスクを自身のタスクプールに挿入する．小粒度タスクがタスクプールからなくなるまで小粒度タスクをタスクプールから 1 つずつ取り出し，小粒度タスクの実行を繰り返す．

マスタプロセスのタスクプールが空になったとき，ワーカプロセスのタスクプールから小粒度タスクを集めることが可能であり，アイドル状態のワーカプロセスにタスクの再分配を行うことができる．

動的負荷分散手法としてマスタ・タスク・ステイル法を持つ並列 Modified PrefixSpan 法を実装し，実際に小規模な PC クラスタと中規模な PC クラスタで性能評価を行った．性能評価により，提案手法は負荷の偏りを解消できることを確認した．効果的な性能向上を得られることも確認できた．

本論文の構成は以下のとおりである．2 章では Modified PrefixSpan 法の処理手順を紹介する．3 章では関連研究について述べる．4 章ではマスタ・ワーカ型による並列化とその問題点について説明する．5 章ではマスタ・タスク・ステイル法の提案を行う．6 章で性能評価について述べ，最後に 7 章で本論文をまとめる．

2. Modified PrefixSpan 法

本章では，例を用いて Modified PrefixSpan 法を紹介し，その処理手順を説明する．

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の集合を 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 j 番目の文字要素は配列データとして $s_{sid}[j]$ 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 2 つのタプルから構成される．s_2 = ⟨MSPNPTNHTGKTLR⟩ であり，$s_2[1]$ =“M” である．

ここで，長さ k のパターンを $\langle pat^k\rangle = \langle A_1 - x(i_1) - A_2 - x(i_2) - \cdots - x(i_{k-1}) - A_k\rangle$ と定義する．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 $x(i_n)$ は固定長ワイルドカード領域を示し，i_n はワイルドカード領域の長さを表す．記号 “-” は，隣どうし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．

たとえば，$F{*}{*}{*}K{*}A$ 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．$x(3)$ はワイルドカード 3 文字が存在することを示している．

Modified PrefixSpan 法では，最大ワイルドカード数を指定する．最大ワイルドカード数は，パターン中

表 1 アミノ酸配列の例
Table 1 Example of amino acid sequences.

sequence id	sequence
1	MFKALRTIPVILNMNKDSKLCPN
2	MSPNPTNHTGKTLR

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 と表す．

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，最小支持数を min_sup と表す．

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

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 $\langle pat^k \rangle$ の最右端文字の右隣の位置（アミノ酸配列上のオフセット）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とおりである．$PDB(\langle pat^k \rangle) = \{(sid, pos) | s_{sid} \in S, pos$ は $\langle pat^k \rangle$ の最右端文字の右隣の位置，$1 \leq pos \leq \| s_{sid} \|\}$

アミノ酸配列データベース S より取り出される頻出パターンの集合を P とすると，$P = \{P_1, P_2, \cdots, P_{max\_length}\}$ となる．頻出パターンに含まれる最大文字要素の数は max_length 個である．

2.2 頻出パターン抽出例

表 1 のアミノ酸配列の例を使用し，Modified PrefixSpan 法の具体的なアルゴリズムの動作を説明する．この例では，最小支持数が 2（$min\_sup = 2$）で最大ワイルドカード数を 1（$max\_wc = 1$）としている．**図 1** に Modified PrefixSpan 法におけるパターン生成手順を示す．

まず，配列データベースをスキャンし 1–頻出パターン $\langle pat^1 \rangle$ を取り出す．表 1 の例では，頻出パターンは

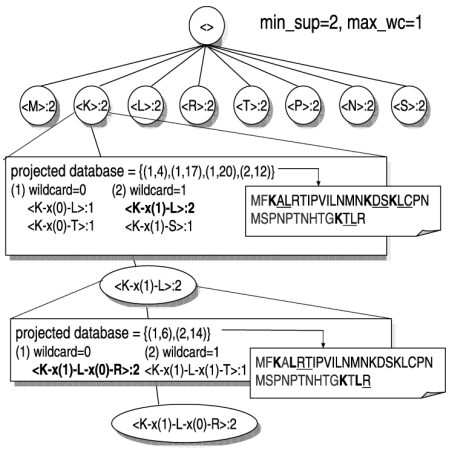

図 1 Modified PrefixSpan 法におけるパターン生成手順
Fig. 1 Example of projected tree in the MPS.

P_1=$\{\langle M \rangle : 2, \langle K \rangle : 2, \langle L \rangle : 2, \langle R \rangle : 2, \langle T \rangle : 2, \langle P \rangle : 2, \langle N \rangle : 2, \langle S \rangle : 2\}$ となる．

配列データベースをスキャンするとき，文字要素の右隣のオフセットを記録し射影データベースを作成する．1–頻出パターンである "$\langle K \rangle : 2$" の射影データベースは $PDB(\langle K \rangle)$ =$\{$(1,4), (1,17), (1,20), (2,12)$\}$ となる．すべての 1–頻出パターンの射影データベース $PDB(\langle pat^1 \rangle)$ を射影データベースの集合である $PDBLIST^1$ に集合要素として挿入する．これにより，$PDBLIST^1$=$\{PDB(\langle M \rangle), \cdots, PDB(\langle S \rangle)\}$ が成立する．$PDBLIST^1$ から $PDB(\langle pat^1 \rangle)$ を 1 つずつ取り出し，2–頻出パターン抽出処理を行う．$PDBLIST^1$ が空の場合，処理を終了する．

ここで，1–頻出パターンである "$\langle K \rangle : 2$" に着目し，2–頻出パターン以降の抽出処理内容をみていく．$PDBLIST^1$ から $PDB(\langle K \rangle)$ を取り出した場合である．

$PDBLIST^1$ からオフセットを使って長さ 2 のパターンを生成する．最大ワイルドカード数が 1 であるため，ワイルドカード数が 0 の場合から考える．それぞれのオフセットにワイルドカード数を足した値が示す文字要素は，それぞれ，$s_1[4+0] = \langle A \rangle$，$s_1[17+0] = \langle D \rangle$，$s_1[20+0] = \langle L \rangle$ と $s_2[12+0] = \langle T \rangle$ である．文字要素 A と D は 1–頻出パターンではないので除外する．文字要素 L と T をパターン $\langle K - x(0) \rangle$ に連結する．パターン $\{\langle K - x(0) - L \rangle : 1, \langle K - x(0) - T \rangle : 1\}$ の支持数はそれぞれ 1 であるために 2–頻出パターンとはならない．

次に，ワイルドカード数が 1 の場合を考える．それぞれのオフセットにワイルドカード数を足した値が示す文字要素は，$s_1[4+1] = \langle L \rangle$，$s_1[17+1] = \langle S \rangle$，

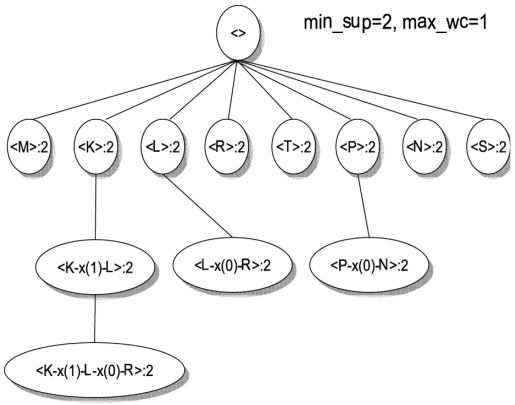

図 2 列挙木
Fig. 2 Example of projected tree.

$s_1[20+1] = \langle C \rangle$ と $s_2[12+1] = \langle L \rangle$ である．文字要素 C は 1–頻出パターンではないので除外する．文字要素 L と S をパターン $\langle K - x(1) \rangle$ に連結する．パターン $\langle K - x(1) - L \rangle$ の支持数は 2 となるので 2–頻出パターンとなる．このとき，頻出パターン $\langle K - x(1) - L \rangle$ の射影データベースは $PDB(\langle K - x(1) - L \rangle)$ ={(1,6), (2,14)} となり，集合 $PDBLIST^2$ に $PDB(\langle K - x(1) - L \rangle)$ を集合要素として挿入する．

同様に処理を続け，3–頻出パターン $\langle K - x(1) - L - x(0) - R \rangle$ が抽出される．すべての処理を終えたときに，**図 2** のような列挙木ができあがる．

2.3 処 理 手 順

Modified PrefixSpan 法の最も大事な考えは *tree projection* である．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し，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．

以下に Modified PrefixSpan 法の基本処理手順の概要を示す．Modified PrefixSpan 法の処理手順は 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 の支持数を数えあげていく．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．すべての 1–頻出パターン $\langle pat^1 \rangle$ を P_1 にそれぞれ挿入する．最後に各 1–頻出パターン $\langle pat^1 \rangle$ に対応する射影データベース $PDB(\langle pat^1 \rangle)$ を集合 $PDBLIST^1$ の集合要素として挿入する．
- **フェーズ 2：**
  もし $PDBLIST^k$ が空ならば処理を終了する．$PDBLIST^k$ が空でない場合，$PDBLIST^k$ の中から $PDB(\langle pat^k \rangle)$ $(k \geq 1)$ 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あげていく．パターン作成時に，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 $PDB(\langle pat^{k+1} \rangle)$ を構築する．$PDB(\langle pat^{k+1} \rangle)$ の構成は以下のとおりである．$(i,j) \in PDB(\langle pat^k \rangle)$，$0 \leq w \leq max\_wc$ に対して，$\langle pat^{k+1} \rangle = \langle pat^k - x(w) - S_i[j+w] \rangle (S_i[j+w] = \alpha$ とおく) とする．このとき $PDB(\langle pat^{k+1} \rangle) = PDB(\langle pat^k - x(w) - \alpha \rangle) = \{(i, j_{new}) |\ (i,j) \in PDB(\langle pat^k \rangle), \alpha = S_i[j+w], j_{new} = j + wc + 1, j_{new} \leq \| S_i \|\}$ である．作成と支持数の数えあ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．

フェーズ 2 を繰り返し，頻出パターンをすべて抽出する．

3. 関 連 研 究

3.1 配列データからの頻出パターン抽出

配列データ（系列データとも呼ばれる）から高速に頻出パターンを抽出することは，顧客の購買履歴にはじまり，ネットワークアクセス履歴と DNA 配列やアミノ酸配列などの分子生物情報などにわたり様々な分野で重要な課題となっている．特に顧客の購買履歴についての頻出パターン抽出アルゴリズムは逐次・並列処理ともに様々な研究が行われ，効果的なアルゴリズムが数多く存在する．

1990 年代後半に，*apriori* [9] ベースの頻出パターン抽出アルゴリズム[10] が提案された．*apriori* は頻出アイテムセットを抽出するアルゴリズムとして元々提案されてきたもので配列データからの頻出パターンを抽出するのにはいくつかの欠点があり，最小支持数が小さくなると性能が急激に悪化することが指摘されている[2]．性能低下の原因として，候補パターンの数が大

きくなることと何度もすべての配列データベースをスキャンしなければならないことがあげられている．

apriori ベースの頻出パターン抽出アルゴリズムの欠点を解決するために，Han らは新しい頻出パターン抽出のアルゴリズムとして，*tree projection* という考えを導入した．*tree projection* ベースのアルゴリズムでは k–頻出パターン（長さ k の頻出パターン）より $(k+1)$–頻出パターンを抽出するとき，k–頻出パターンの最右端以降のデータを 1 つ追加し，パターンを抽出していく．*tree projection* の最新アルゴリズムである PrefixSpan 法[2),3)] は *apriori* ベースのアルゴリズムよりも高速であることが性能評価により示されている．

apriori ベースと *tree projection* ベースのアルゴリズムは特に顧客の購買履歴など，ビジネス分野における頻出パターン抽出を研究の対象としていて，分子生物情報の分野を十分にサポートしていない．アミノ酸配列から分子生物学者が興味を持つような頻出パターンを抽出できるようにするため，Modified PrefixSpan 法が提案されている．Modified PrefixSpan 法は *tree projection* ベースである PrefixSpan 法を拡張したものである．複数のアミノ酸配列の様々な位置に存在する固定長ワイルドカード領域を含む頻出パターンを取り出すことができる．

一方，配列データからの頻出パターン抽出処理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行われてきた．*apriori* ベースのアルゴリズムを中心に様々な効果的な並列化手法が存在する[11)~14)]．しかしながら，我々が知る限り *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 Guralnik ら[15),16)] による研究のみである．

3.2 *tree projection* ベースアルゴリズムの並列化について

tree projection ベースのアルゴリズムの並列化手法，負荷分散手法を Guralnik らが提案している．

3.2.1 対象データの違い

文献 15)，16) では主に顧客の購買履歴などのビジネスデータを対象としている．顧客の購買履歴は 1 つの配列データサイズは小さいが，件数が数 10 万件から数 100 万件に及び，総データサイズが非常に大きい．よって，データを各計算機に分散配置させた環境下での，タスク分割による並列化手法が用いられている．

本研究が対象としているアミノ酸の配列データは，件数は多くて数千件だが 1 つの配列データのサイズが 100 以上や 1,000 に及ぶものであり，総データサイズは大きくないが組合せが爆発し非常に CPU コストを必要とする．対象とするアミノ酸の配列データは比較的小さいため，各計算機にあらかじめデータ全体を配置しておくか，処理を開始する際に全計算機に複製を転送する．

3.2.2 タスクの定義の違い

Guralnik らは，*tree projection* の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．初期タスクとしてユーザが定めた閾値 k まで頻出パターン抽出を行い，閾値以降の頻出パターン抽出処理をタスクとして各計算機に分配する．生成された初期タスクは，使用する計算機数と等しい数に分割され，各計算機に分配される．

本研究では，タスクとして小粒度タスクを扱う．小粒度タスクの特徴については，5 章で詳しく述べる．

3.2.3 負荷分散手法の違い

文献 15)，16) で提案されている負荷分散手法は，タスクの負荷の見積りを行い，各計算機の負荷が均一になるような手法を用いている．負荷の見積もり方は頻出パターンの支持数に基づいている．支持数の大きな頻出パターンからそれ以降の長さのすべての頻出パターンを抽出する場合，支持数の小さな頻出パターンよりも負荷が大きいだろうと仮定して各計算機に割当てを行う．タスクを早く処理し終えた計算機がランダムに他の計算機にタスクリクエストを送信する．タスクリクエストを受け取った計算機は，割り当てられたタスクを 2 つに分割し，分割したタスクに関連したデータベースを送信することで動的負荷分散を実現している．

本研究において，文献 15)，16) で提案されている手法を用いると，タスクの処理時間を見積もれないので，どのようにタスク分割すれば負荷が均等になるか分からない．そこで，並列 Modified PrefixSpan 法の動的負荷分散手法としてアイドル状態の計算機が現れると，一度マスタプロセスに小粒度タスクをすべて集めて小粒度タスクを再分配するマスタ・タスク・ステイル法を適用した．

4. マスタ･ワーカ型による並列化とその問題点

4.1 マスタ・ワーカ型の適用

Modified PrefixSpan 法は，完全に独立なタスク分割による並列処理が行えるため，並列化では典型的なマスタ・ワーカ型を使用する．マスタ・ワーカ型並列処理は 1 つのマスタプロセスと並列処理を行う複数のワーカプロセスから構成される．まず，マスタプロセスで 1 つのタスクを複数のタスクに分解し，分解したタスクをそれぞれワーカプロセスに割り当て，並列処

理を実行していく．

Modified PrefixSpan 法の並列処理に，マスタ・ワーカ型を使用した理由は，以下のとおりである．Modified PrefixSpan 法では，s–頻出パターンから $(s+1)$ 以降の頻出パターンを抽出するとき，s–頻出パターンを抽出するのに至った処理内容を必要としない．$(s+1)$–頻出パターンを抽出する際に $\langle pat^s \rangle$ 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．$\langle pat^s \rangle$ の最右端文字の右隣のオフセットを記憶しているのは，$PDB(\langle pat^s \rangle)$ である．よって，s–頻出パターン $\langle pat^s \rangle$ から $(s+1)$ 以降のすべての頻出パターンを抽出するためには，$\langle pat^s \rangle$ と $PDB(\langle pat^s \rangle)$ があればよい．$\langle pat^s \rangle$ と $PDB(\langle pat^s \rangle)$ を受け取ったワーカプロセスは，他のワーカプロセスからの他の情報は何も必要なく s–頻出パターン $\langle pat^s \rangle$ から $(s+1)$ 以降のすべての頻出パターンを抽出できる．

4.2 タスクプールによる動的負荷分散の問題点

マスタ・ワーカ型並列処理では通常タスクプールによる動的負荷分散手法を持つ．マスタプロセスはユーザがあらかじめ指定した閾値 s まで s–頻出パターンを抽出する．s–頻出パターンから $(s+1)$ 以降の頻出パターン抽出処理を 1 つのタスク（大粒度タスク）としてタスクプールに挿入する．大粒度タスクを早く処理し終えたワーカプロセスに次々と割り当てていく．大粒度タスクの処理時間に少しの差があったとしても早く処理を終えたワーカプロセスに次々と大粒度タスクが割り当てられるため，負荷の偏りを避けることができると期待される．

マスタ・ワーカ型並列処理で小粒度タスクを用いていない理由を示す．マスタ・ワーカ型の並列処理において，タスクプールに挿入するタスクを小粒度タスクとすると，小粒度タスクを受け取ったワーカプロセスは小粒度タスクを処理し，生成した小粒度タスクを再度マスタプロセスに送信しなければならない．これは，後に示す制限型マスタ・ワーカ法と同意であり，通信量が大幅に増加し，性能が低下してしまう．よって，マスタ・ワーカ型並列処理では大粒度タスクを用いている．

並列 Modified PrefixSpan 法における大粒度タスクの処理時間はアミノ酸配列の特徴に依存し，極端に大きな差がある．この特徴によりタスクプールを使用した動的負荷分散では，以下の 2 つの課題が存在する．

- 処理の終盤になって非常に処理時間が大きい大粒度タスクがいくつかのワーカプロセスに割り当てられた場合，十分なスピードアップが得られない．
- 初期生成した大粒度タスクがワーカプロセス数よりも小さい場合，大粒度タスクを割り当てられないワーカプロセスが発生する．

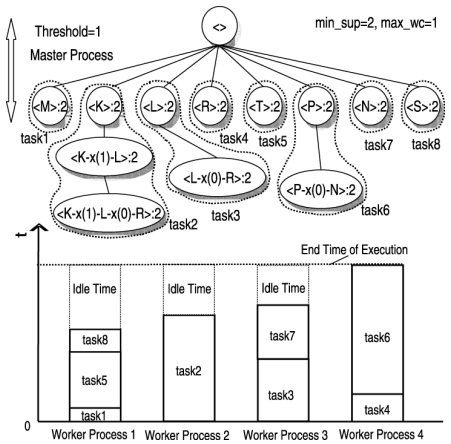

図 3 並列 Modified PrefixSpan 法における並列パターン生成手順

Fig. 3 Parallel pattern extraction steps in the Parallel Modified PrefixSpan.

図 3 を例として考える．図 3 では，まずタスク 1 からタスク 4 までがそれぞれワーカプロセス 1 からワーカプロセス 4 に割り当てられている．最も小さなタスク 1 を割り当てられたワーカプロセス 1 が次にタスク 5 を割り当てられ，次にタスク 6 がワーカプロセス 4 に割り当てられる．そしてタスク 8 までがすべて割り当てられたとき，ワーカプロセス 1 はマスタプロセスのグローバルタスクプールが空なので他のワーカプロセスの終了を待つことになる．図 3 では，タスク 6 を実行しているワーカプロセス 4 の終了を待つことになる．

Modified PrefixSpan 法によるアミノ酸配列からの頻出パターン抽出処理の特徴を示すために Kringle データセットで予備的な解析を行った（配列データの詳細は 6.1 節で示す．使用した計算機は 6.2 節で示す PC クラスタを構成するうちの 1 台である）．パラメータとして，最小支持数を 14（最小支持率 20%），最大ワイルドカード数を 6 として解析を行った．

図 4 に各 1–頻出パターンから 2–頻出パターン以降の全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．つまり，1 つのタスクを k–頻出パターンから $(k+1)$ 以降の頻出パターン抽出処理（大粒度タスク）と見なした場合である．たとえば，“C” を接頭辞とする頻出パターンをすべて抽出するために必要となった処理時間を示している．図 4 から，特に “C”，“G”，“K” を接頭辞とする頻出パターン抽出処理に負荷が偏っていることが分かる．これらは，実際に抽出を行ってみて分かった結果であり，はじめからこれら

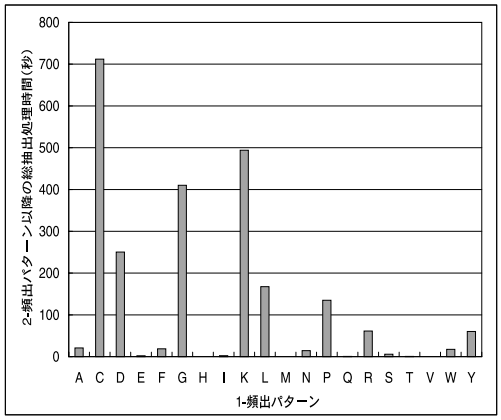

図 4 1–頻出パターンごとの 2–頻出パターン以降の総抽出処理時間（Kringle）
Fig. 4 Processing time of all frequent pattern extraction after 2-frequent pattern from each 1-frequent pattern (Kringle).

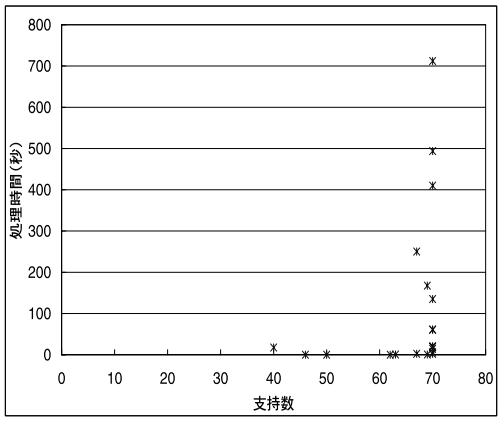

図 5 支持数と処理時間の関係（Kringle）
Fig. 5 Relation between support and processing time (Kringle).

の接頭辞に続く処理が多いとは判断できない．

図 5 は，図 4 に示す 1–頻出パターンの支持数と，1–頻出パターンから 2–頻出パターン以降のすべての頻出パターンを抽出する処理時間の関係を示している．図 5 より，支持数が最大（70）でも数秒で処理を終了するものもあることが分かる．支持数が高い大粒度タスクが支持数の低い大粒度タスクよりも処理時間を多く必要とするとは考えにくい．

図 6 は図 4 に示す 1–頻出パターンの射影データベースの要素数と，1–頻出パターンからから 2–頻出パターン以降のすべての頻出パターンを抽出する処理時間の関係を示している．図 6 より大粒度タスクの処理時間は射影データベースの要素数に比例していないことが分かる．射影データベースの要素数が 1,000

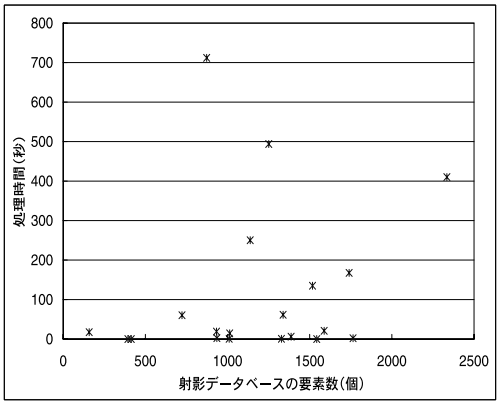

図 6 射影データベースの要素数と処理時間の関係（Kringle）
Fig. 6 Relation between the number of element in projected database and processing time (Kringle).

個程度でも約 700 秒かかっている大粒度タスクもあれば，射影データベースの要素数が 2,400 個の場合でも約 400 秒程度で処理を終了できる大粒度タスクも存在している．

5. マスタ・タスク・ステイル法

5.1 概　　要

前章で示した課題を解決するために，新しい動的負荷分散手法として以下の特徴を持った，マスタ・タスク・ステイル法を提案する．提案手法は，以下の特徴をすべて含めてマスタ・タスク・ステイル法と定義する．

（1） ワーカプロセスにもタスクプールを配置
マスタプロセス以外に，すべてのワーカプロセスにタスクプールを持たせる．マスタプロセスのタスクプールをグローバルタスクプールと呼び，ワーカプロセスのタスクプールをローカルタスクプールと呼ぶ．ワーカプロセスにローカルタスクプールを配置することで，ワーカプロセスよりタスクを奪い取ることができる．

（2） 小粒度タスクを採用
長さ k から長さ $(k+1)$ の頻出パターンを抽出する処理を小粒度タスクと呼ぶ．マスタプロセスで閾値 1 まで頻出パターンを抽出し，生成された 1–頻出パターンから 2–頻出パターンを抽出する処理を小粒度タスクとしてグローバルタスクプールに挿入する．ワーカプロセスでは，マスタプロセスから小粒度タスクを受け取ると，小粒度タスクを実行し生成された 1–頻出パターンから 2–頻出パターンを抽出する処理をそれぞれ小粒度タスクとしてワーカプロセスのローカルタスクプールに挿入する．ワーカプロセスは，ローカルタスクプールの小粒度タスクが

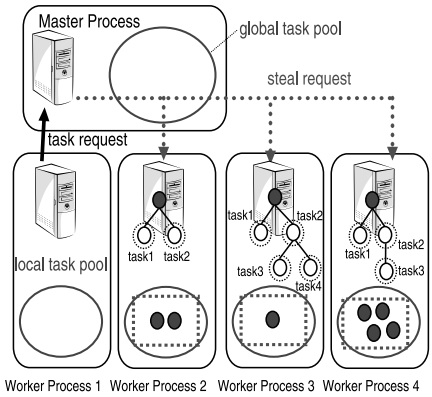

図 7 マスタ・タスク・ステイル法
Fig. 7 Master-Task-Steal methodology.

なくなるまで「ローカルタスクプールからの小粒度タスクの取り出し，小粒度タスクの実行，生成された小粒度タスクのローカルタスクプールへの挿入」を繰り返す．

(3) マスタプロセスによるワーカプロセスからのタスク奪い取り

ワーカプロセスよりタスクリクエストを受け取ったときにグローバルタスクプールが空であれば，マスタプロセスは全ワーカプロセスにタスク奪い取りのメッセージを送信する．タスク奪い取りのメッセージを受け取ったワーカプロセスはローカルタスクプールの小粒度タスクをマスタプロセスに送信する．マスタプロセスは受け取った小粒度タスクをグローバルタスクプールに挿入する．

本論文で提案しているマスタ・タスク・ステイル法はマスタ・ワーカ型の並列処理をモデルとして適応している．タスク・ステイル法は，マスタプロセスを持たない完全分散型の並列処理のモデルであり，マスタプロセスを持つマスタ・タスク・ステイル法とはまったく異なる．よって，マスタ・タスク・ステイル法は，マスタ・ワーカ型と完全分散型の相反するモデルどうしを単に組み合わせるという考え方で得られるものではなく，マスタ・ワーカ型の並列処理のモデルに新しい動的負荷分散機能を追加するという考え方で得られた．

5.2 処 理 手 順

以下，マスタプロセスとワーカプロセスの詳細な処理ステップを示す（**図 7** にマスタ・タスク・ステイル法の概念図を示す）．

マスタプロセス：

(1) マスタプロセスは 1–頻出パターンを抽出する．

(2) 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 つのタスク（小粒度タスク）としてグローバルタスクプールに挿入する．グローバルタスクプールの 1 つの小粒度タスクの表現は $\langle pat^1 \rangle$ と $PDB(\langle pat^1 \rangle)$ のペアとしている．

(3) マスタプロセスはワーカプロセスからタスクリクエストを受け取る．グローバルタスクプールに小粒度タスクがあれば小粒度タスクを 1 つ取り出し，その小粒度タスクをワーカプロセスに送信する．(3) へ戻る．もしグローバルタスクプールが空であれば (4) へ．

(4) マスタプロセスはすべてのワーカプロセスにタスク奪い取りのメッセージを送信し，すべてのワーカプロセスから小粒度タスクを受け取る．受け取った小粒度タスクが 1 つ以上存在するかどうかにより次の処理を行う．

(a) 受け取った小粒度タスクが 1 つ以上存在すれば，その小粒度タスクをグローバルタスクプールに挿入する．タスクリクエストを送信したワーカプロセスに 1 つ小粒度タスクを送信し，(3) へ．

(b) 1 つも小粒度タスクを受け取らなかった場合，すべてのワーカプロセスのタスクリクエストに対して終了のメッセージを返信し，処理を終了する．

ワーカプロセス：

(1) ワーカプロセスはマスタプロセスにタスクリクエストを送信する．

(2) タスクリクエストの返信により次の処理を行う．

(a) マスタプロセスよりタスクリクエストの返信として小粒度タスクを受け取ると，ワーカプロセスは小粒度タスクをローカルタスクプールに挿入し，(3) へ．

(b) マスタプロセスよりタスクリクエストの返信として終了のメッセージを受け取ると，処理を終了する．

(3) もしローカルタスクプールが空であれば (1) へ．ローカルタスクプールより小粒度タスクを取り出す．小粒度タスクを実行し，生成された小粒度タスクをローカルタスクプールに挿入し，(3) へ．上述の処理中にマスタプロセスからタスク奪い取りメッセージを受け取った場合，ワーカプロセスは，小粒度タスクの処理を終えた後（処理中の小粒度タスクをすべてローカルタスクプールに挿入した後），ローカルタスクプールにあるすべての小粒度タスクをマスタプロセスに送信する．つまり，ローカルタスクプールにタスクが存在しないとい

表 2 配列データの詳細
Table 2 Details of sequence data.

データセット＼パラメータ	データ件数（件）	総長（バイト）	平均長（バイト）	最大長（バイト）	最小長（バイト）
Kringle	70	23,385	334	3,176	53
Zinc Finger	467	245,595	525	4,036	34
Leucine	370	139,422	376	3,224	3

表 3 パラメータ
Table 3 Parameters.

データセット＼パラメータ	評価	最小支持数（最小支持率%）	最大ワイルドカード数
Kringle	評価 1	14（20%）	5
	評価 2	14（20%）	7
Zinc Finger	評価 1	140（30%）	5
	評価 2	117（25%）	7
Leucine	評価 1	37（10%）	6
	評価 2	37（10%）	9

うことは，そのワーカプロセスには処理中の小粒度タスクも存在しないことになる．この処理の後にマスタプロセスが小粒度タスクをまったく受け取っていなければ，すべての小粒度タスクを処理し終えたことになる．個々の小粒度タスクの実行は瞬時に終了するので小粒度タスクの処理を待つアイドル時間は考慮しなくてもよい（6 章「性能評価」で示す）．

5.3 既存の動的負荷分散手法との比較

マスタ・ワーカ型の動的負荷分散手法として，制限型マスタ・ワーカ法[17] が存在する．

制限型マスタ・ワーカ法は並列分枝限定法に適している．ワーカプロセスでユーザが指定した閾値までタスクを生成し，生成されたタスクをマスタプロセスに戻すことで，限定操作ができる．マスタプロセスに戻したタスクをワーカプロセスに再分配することを繰り返すので，タスク粒度が細かくなり，きめ細かな負荷の偏りの解消が可能となる．しかしながら，解を毎回マスタプロセスに戻すので，通信のオーバヘッドが大きくなる．

マスタ・タスク・ステイル法は，負荷が偏った時点でのみワーカプロセスのすべてのタスクをマスタプロセスに集めるので，制限型マスタ・ワーカ法程通信量は増えず，ある程度きめ細かな負荷分散を行うことができる．

6. 性 能 評 価

PC クラスタに並列 Modified PrefixSpan 法を実装し，並列化とその動的負荷分散手法であるマスタ・タスク・ステイル法の性能評価を行った．

6.1 配列データ

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）と，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），Leucine というモチーフを含む配列データ（以下，Leucine データセット）を使用した．各配列データの詳細を**表 2** に示す．

6.2 評価 1：小規模クラスタでの性能評価

16 台で構成された PC クラスタで性能評価を行った．各計算機の性能は次のとおりである．Intel Pentium4 プロセッサ 2.53 GHz，1.5 GB メモリを搭載し，各計算機は 100 Mbit/sec イーサネットスイッチで接続されている．OS には RedHat9.0 を使用し，通信にはソケット通信と MPICH[19] の version 1.2.5 を MPI ライブラリとして使用した．評価 1，評価 2 で使用した各配列のパラメータを**表 3** に示す．

図 8，**図 9**，**図 10** に並列 Modified PrefixSpan 法の動的負荷分散手法としてタスクプールによる動的負荷分散手法のみ（制限型マスタ・ワーカ法ではない）を適用した場合の性能向上比を示す．マスタプロセスでの閾値は “Threshold n” と表現する．

マスタプロセスでの閾値を大きくすると性能が向上していることが分かる．図 8 より，Kringle データセットでの性能評価で，16 台で最大 14 倍の性能向上が得られている．マスタプロセスでの閾値を大きく設定することで生成される大粒度タスクの粒度が細かくなるからである．大粒度タスクの粒度を細かくすることで，ある程度大粒度タスクの負荷を均一にすることができ，タスクプールによる動的負荷分散手法が効果的に機能

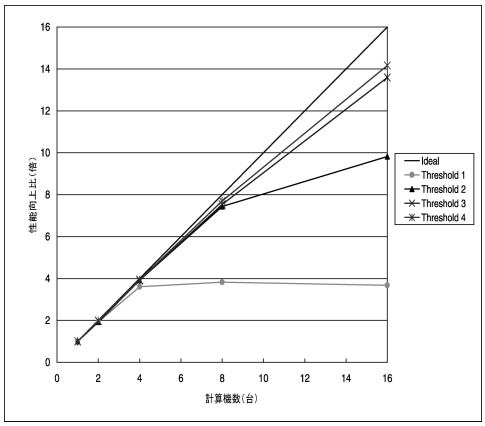

図 8 性能向上比（タスクプール，Kringle）
Fig. 8 Speed up (Task Pool, Kringle).

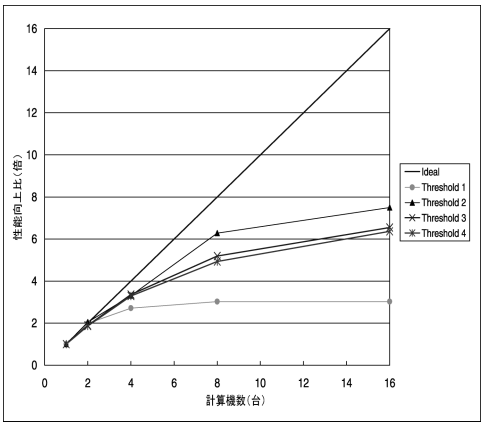

図 10 性能向上比（タスクプール，Leucine）
Fig. 10 Speed up (Task Pool, Leucine).

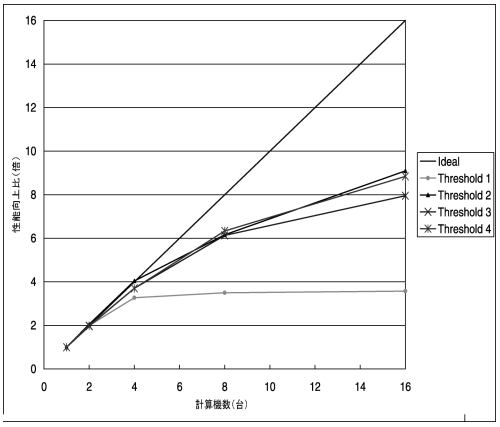

図 9 性能向上比（タスクプール，Zinc Finger）
Fig. 9 Speed up (Task Pool, Zinc Finger).

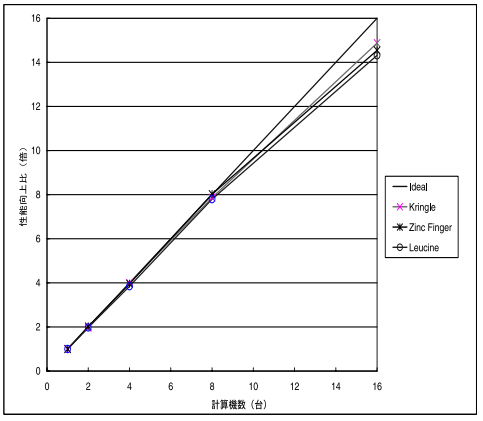

図 11 性能向上比（マスタ・タスク・ステイル法）
Fig. 11 Speed up (Master-Task-Steal methodology).

する．しかしながら，Zinc Finger データセットで性能評価を行った図 9 では 16 台で最大 9 倍程度，Leucine データセットで性能評価を行った図 10 では 16 台で最大 8 倍程度の性能向上しか得られていない．大粒度タスクの粒度は細かくなったが，大粒度タスクの負荷に大きな差があり十分な性能向上が得られなかったと考えられる．

閾値を大きく設定すると，大粒度タスクの粒度が細かくなり性能向上が期待できるが，マスタプロセスでの閾値までの処理（初期タスク生成）がボトルネックになってしまう可能性がある．そこで，Leucine データセットで評価を行ったときのマスタプロセスでの閾値までの処理時間が全体の処理時間の何%程度を占めているのか測定した．Threshold 1 の場合，全体の 0.04%，Threshold 2 の場合，全体の 1.4%，Threshold 3 の場合，全体の 11%，Threshold 4 の場合，全体の 37%，であった．閾値を大きくするとマスタプロセスでの処理量が増加しており，閾値を 4 に設定した場合は明らかにマスタプロセスでの初期タスク生成がボトルネックとなっている．

閾値の設定はユーザの経験的な判断に基づくしかなく，どのように閾値を設定すれば最適な性能向上が得られるか判断することはできない．

図 11 に並列 Modified PrefixSpan 法の動的負荷分散手法としてマスタ・タスク・ステイル法を適用した場合の性能向上比を示す．図 11 より，各データセットで評価を行った場合，16 台で約 15 倍の性能向上が得られた．この理由として，マスタ・タスク・ステイル法により各計算機の処理時間が均一になり，有効な性能向上比が得られたと考えられる．

図 12 は，16 台の計算機を使用し，タスクプールによる動的負荷分散手法のみ（制限型マスタ・ワーカ

表 4 処理時間（秒）
Table 4 Processing time (sec).

データセット＼台数（台）	1	2	4	8	16	32	64
Kringle	5,098.982	2,385.872	1,201.855	597.019	297.027	149.664	79.775
Zinc Finger	14,111.351	6,980.963	3,484.779	1,738.217	872.557	439.467	224.565
Leucine	2,071.000	1,755.243	653.764	288.179	128.780	68.684	38.529

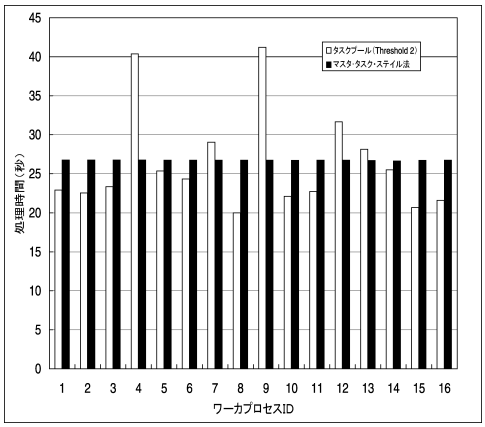

図 12 各ワーカプロセスの処理時間（Kringle）
Fig. 12 Processing time of each worker process (Kringle).

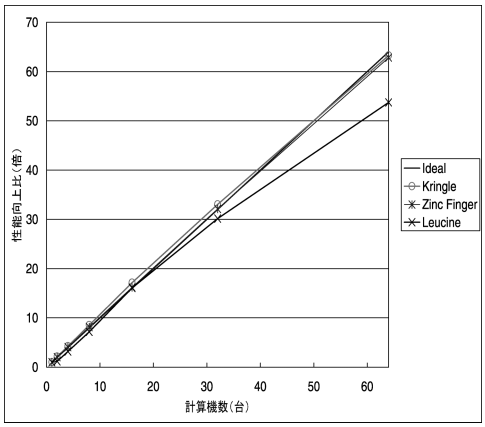

図 13 性能向上比（マスタ・タスク・ステイル法）
Fig. 13 Speed up (Master Task Steal methodology).

法ではない）を用いた場合と，マスタ・タスク・ステイル法を用いた場合の各ワーカプロセスの処理時間を調べたものである．タスクプールにおけるマスタプロセスの閾値は 2 と設定している．使用した配列データは Kringle データセットである．タスクプールによる動的負荷分散手法を用いた場合，ワーカプロセス間の負荷が偏っていることが分かる．マスタ・タスク・ステイル法では，各ワーカプロセスの処理時間が均一になっており，図 11 に示すような効果的な性能向上比が得られた．

並列 Modified PrefixSpan 法の動的負荷分散手法としてマスタ・タスク・ステイル法を適用した場合，16 台規模の PC クラスタでは 16 台で約 15 倍と，効果的な性能向上比が得られた．各計算機の負荷が均一になっていることも確認できた．

6.3 評価 2：中規模クラスタでの性能評価

64 台で構成された PC クラスタを使用して性能評価を行った．本評価ではマスタ・タスク・ステイル法についての評価を行う．各計算機の性能は次のとおりである．Intel Pentium4 プロセッサ 2.8 GHz，1.0 GB 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 を使用した．評価で使用した各配列のパラメータは表 3 に示すとおりである．

表 5 初期タスク生成時間と初期タスク数
Table 5 Initial task generation time and the number of initial tasks.

データセット	初期タスク生成時間（秒）	タスク数（個）
Kringle	0.0045	20
Zinc Finger	0.0576	20
Leucine	0.0314	20

各データセットでの処理時間を**表 4** に，性能向上比を**図 13** に示す．図 13 より，16 台以上の計算機を使用しても Kringle データセット，Zinc Finger データセットでの評価において，64 台で約 64 倍と理想性能向上比と同様の性能向上比が得られている．Leucine データセットでの評価においては，64 台で約 53 倍の性能向上比が得られている．Leucine データセットにおいては，表 4 に示すように，32 台の計算機を使用した場合，すでに 70 秒程度で処理が終了しているので 64 台使用した場合でも 53 倍程度の性能向上しか得られなかった．

初期タスクが使用する計算機数よりも少ない場合，タスクを割り当てられない計算機があるという問題を回避できているか調べた．**表 5** に示すように，初期タスクはすべてのデータセットにおいて 20 個であり，20 台よりも多い計算機を使用した場合，初期タスクが割り当てられない計算機が出現する．しかしながら，

表 6 マスタプロセスの送受信量と通信回数
Table 6 Amount of sending and receiving and frequency in master process.

パラメータ / データセット	台数	通信回数（回）	通信量（Kbytes）
Kringle	32 台	4,837	3,664
	64 台	7,382	4,600
Zinc Finger	32 台	4,499	47,202
	64 台	8,016	75,335
Leucine	32 台	8,164	25,968
	64 台	9,761	26,882

各データセットでの性能向上比より，タスクが割り当てられないワーカプロセスが存在していないと考えられる．マスタ・タスク・ステイル法の機能が効果的に働いている．

使用する計算機数が増加すると，マスタプロセスでの処理がボトルネックになると予想できる．そこで，マスタプロセスでの処理がボトルネックになっていないか調べた．

まず，マスタプロセスでの初期タスク生成の処理時間を調べた．表 5 にマスタプロセスでの初期タスク生成の処理時間を示す．初期タスク生成の処理時間は 1 秒もかかっておらず，大規模な抽出処理において初期タスク生成処理がボトルネックになることはない．

個々の小粒度タスクの実行時間は，0.001 秒程度で終了する．Zinc Finger データセットによる性能評価の際に個々の小粒度タスクの実行時間を計測し，平均値を算出したところ，0.001156 秒であった．よって，5.2 節のワーカプロセスの処理手順に示したように，タスク奪い取りの際に小粒度タスクの実行待ちによるアイドル時間は発生しないと考えてよい．

次に，各データセットでの性能評価におけるマスタプロセスの送受信量と通信回数を**表 6** に示す．表 6 より，各データセットにおいて台数が増加するごとにマスタプロセスでの通信回数，送受信量ともに増加していることが分かる．Zinc Finger データセットによる評価で最高約 75 M バイトの送受信量が発生している．しかしながら，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おり，全体で約 75 M バイトの通信はほぼオーバヘッドが発生しないと考えてよい．Kringle データセット，Leucine データセットにおいても同様である．

図 14，**図 15** は，64 台用いた場合の Zinc Finger データセットでの性能評価における通信量の詳細を示す．通信回数は表 6 より，8,016 回発生している．図 14，図 15 より，マスタプロセスでの送信回数は受信回数よりも少ないが，個々の通信の通信量をみると送信量よりも受信量のほうが多い．これは，マスタプロセスはタスク奪い取りが発生している時点でのみ各ワーカプロセスから小粒度タスクをすべて受信するので，個々の通信をみると送信量よりも受信量のほうが多い．最高で 1.4 M バイトの通信量が発生しているのみであるので，通信の遅延による問題は発生しないと考えられる．しかしながら，通信量が小さな通信が数多く発生していることから，グリッド環境のように通

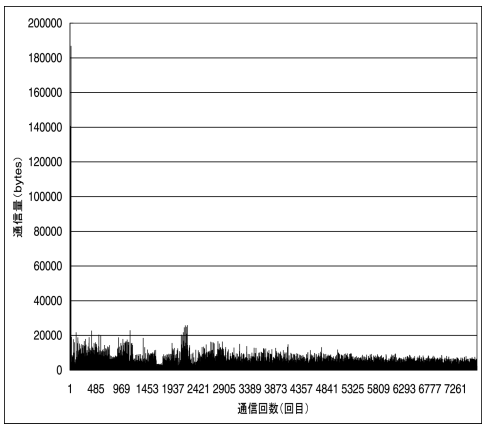

図 14 送信量の詳細（Zinc Finger，64 台）
Fig. 14 Details of sending in master process.

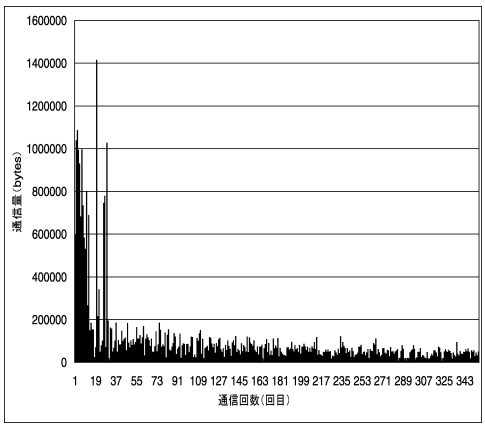

図 15 受信量の詳細（Zinc Finger，64 台）
Fig. 15 Details of receiving in master process.

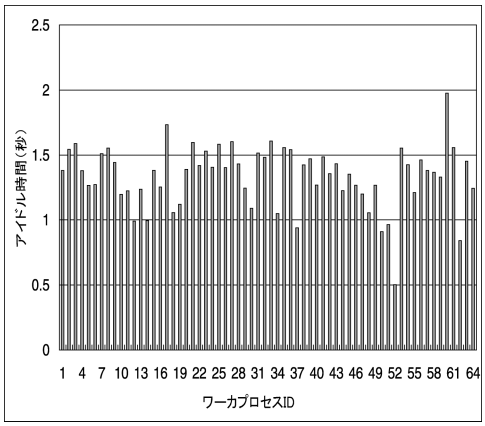

図 16 ワーカプロセスのアイドル時間（Zinc Finger）
Fig. 16 Idle time in worker processes.

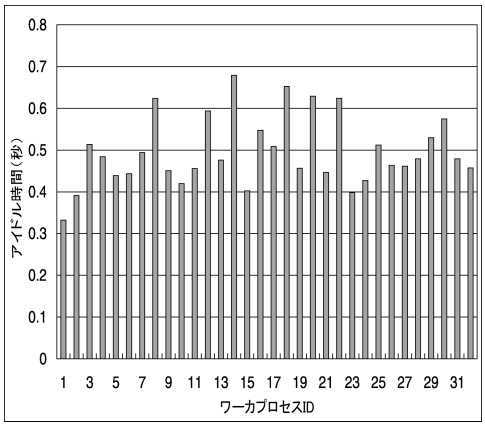

図 17 ワーカプロセスのアイドル時間（Leucine，32 台）
Fig. 17 Idle time in worker processes.

信性能がクラスタより劣る場合，本手法を用いると通信回数が多く，通信の遅延によりワーカプロセスでのタスク待ちのアイドル時間が増加してしまう可能性がある．

最後に，**図 16** に各ワーカプロセスがマスタプロセスにタスクリクエストを送信してから小粒度タスクを受け取るまでのアイドル時間を合計したものを示す．図 16 は 64 台の計算機を使用し，Zinc Finger データセットを用いて評価を行ったときの各ワーカプロセスのアイドル時間を示している．

各ワーカプロセスとも 1 秒程度のアイドル時間が発生しているのみで，マスタプロセスにタスクリクエストを送信して小粒度タスクを受け取るまでほぼアイドル時間が発生していない．最大でも全体の処理時間の 2%程度のアイドル時間が発生しているだけである．

Leucine データセットで性能向上が 64 台で 53 倍程度しか得られていない理由がマスタプロセスのボトルネックでないことを示す．**図 17**，**図 18** は Leucine データセットを使用して 32 台，64 台で評価を行った際の各ワーカプロセスのアイドル時間を示している．32 台の場合，アイドル時間は全体の処理時間の 1%程度であり，64 台でも全体の処理時間の 4%程度のアイドル時間しか発生していない．このことより，Leucine データセットでの評価において 64 台で 53 倍の性能向上しか得られなかった理由がマスタのボトルネックではないことが分かる．

64 台で構成された中規模 PC クラスタでの性能評価においては，64 台で最大 64 倍の性能向上比が得られた．マスタプロセスがボトルネックになっていないことも確認できた．

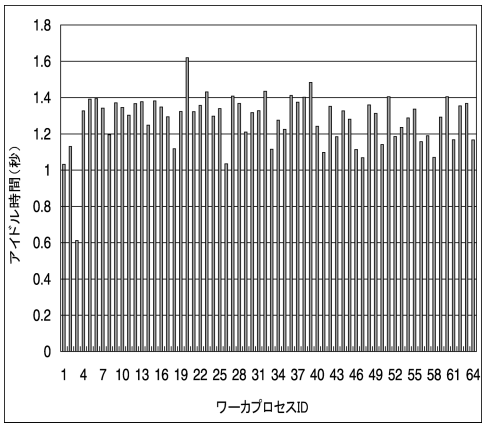

図 18 ワーカプロセスのアイドル時間（Leucine，64 台）
Fig. 18 Idle time in worker processes.

7. ま と め

本論文では，Modified PrefixSpan 法を並列化した並列 Modified PrefixSpan 法を示し，実際の PC クラスタに実装した．Modified PrefixSpan 法の並列化には，タスク分割によるマスタ・ワーカ型の並列化手法を用いた．また，その動的負荷分散手法であるマスタ・タスク・ステイル法を提案し，性能評価を行った．マスタ・タスク・ステイル法では，小粒度タスクを採用した．ワーカプロセスにもタスクプールを持たせ，ワーカプロセスが小粒度タスクを処理するごとに生成した小粒度タスクをタスクプールに挿入する．アイドル状態のワーカプロセスが出現した時点で，マスタプロセスがすべてのワーカプロセスのタスクプールから小粒度タスクを集め，再分配することで負荷の均一化を図った．性能評価では，64 台で最大約 64 倍の性能

向上比を得ることができた．各計算機の負荷が均一になっていること，台数を増やしてもマスタプロセスがボトルネックになっていないことも確認した．

今後の課題として，台数が数百台規模や数千台規模になった場合や，グリッド環境における並列処理方式の検討があげられる．

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），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：17500097）の支援により行われた．

参 考 文 献

1) Kitakami, H., Kanbara, T., Mori, Y., Kuroki, S. and Yamazaki, Y.: Modified prefixspan method for motif discovery in sequence databases, *Proc. Trends in Artificial Intelligence, 7th Pacific Rim International Conference on Artificial Intelligence* (*PRICAI2002*), Lecture Notes in Computer Science, Vol.2417, pp.482–491, Springer (2002).
2) Han, J. and Pei, J.: Mining frequent patterns by pattern-growth: Methodology and implications, *SIGKDD Explorations*, Vol.2, No.1, pp.14–20 (2000).
3) Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U. and Hsu, M.: Prefixspan: Mining sequential patterns by prefix-projected growth, *Proc. 17th International Conference on Data Engineering*, pp.215–224, IEEE Computer Society (2001).
4) Sutou, T., Tamura, K., Mori, Y. and Kitakami, H.: Design and implementation of parallel modified prefixspan method, *High Performance Computing, Proc. 5th International Symposium, ISHPC*, Lecture Notes in Computer Science, Vol.2858, pp.412–422, Springer (2003).
5) 高木 允，田村慶一，周藤俊秀，北上 始：並列 modified prefixspan 法における動的負荷分散手法，情報処理学会研究報告，Vol.2004, No.67, pp.9–15 (2004).
6) Carriero, N. and Gelernter, D.: How to write Parallel Programs, The M.I.T. Press (1990).
7) Quinn, M.J.: *PARALLEL PROGRAMMING in C with MPI and OpenMP*, McGraw-Hill College (2003).
8) Wilkinson, B. and Allen, M.: *Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall (1999).
9) Agrawal, R., Imielinski, T. and Swami, A.N.: Mining association rules between sets of items in large databases, *Proc. 1993 ACM SIGMOD International Conference on Management of Data*, pp.207–216, ACM Press (1993).
10) Agrawal, R. and Srikant, R.: Mining sequential patterns, *Proc. 11th International Conference on Data Engineering*, pp.3–14, IEEE Computer Society (1995).
11) Agrawal, R. and Shafer, J.C.: Parallel mining of association rules, *IEEE Trans. Knowl. Data Eng.*, Vol.8, No.6, pp.962–969 (1996).
12) Shintani, T. and Kitsuregawa, M.: Parallel mining algorithms for generalized association rules with classification hierarchy, *Proc. ACM SIGMOD International Conference on Management of Data* (*SIGMOD 1998*), pp.25–36, ACM Press (1998).
13) Tamura, M. and Kitsuregawa, M.: Dynamic load balancing for parallel association rule mining on heterogenous pc cluster systems, *Proc. 25th International Conference on Very Large Data Bases*, pp.162–173, Morgan Kaufmann (1999).
14) Park, J.S., Chen, M.-S. and Yu, P.S.: Efficient parallel and data mining for association rules, *Proc. 1995 International Conference on Information and Knowledge Management* (*CIKM '95*), November 28 – December 2, 1995, Baltimore, Maryland, USA, pp.31–36, ACM (1995).
15) Guralnik, V. and Karypis, G.: Parallel tree-projection-based sequence mining algorithms, *Parallel Computing*, Vol.30, No.4, pp.443–472 (2004).
16) Guralnik, V. and Karypis, G.: Dynamic load balancing algorithms for sequence mining, *Computer Science and Engineering Technical Report*, No.01-020 (2001).
17) Aida, K., Futakata, Y. and Hara, S.: High-performance parallel and distributed computing for the bmi eigenvalue problem, *IPDPS* (2002).
18) http://kr.expasy.org/prosite/
19) http://www-unix.mcs.anl.gov/mpi/mpich/

(平成 16 年 7 月 26 日受付)
(平成 16 年 11 月 25 日再受付)
(平成 17 年 1 月 11 日再々受付)
(平成 17 年 1 月 31 日採録)

高木　　允（学生会員）

2003 年広島市立大学情報科学部知能情報システム工学科卒業．同年同大学大学院情報科学研究科知能情報システム工学専攻修士課程入学，現在に至る．データマイニングに興味を持つ．

田村　慶一（正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科知能システム学専攻修士課程修了．2003 年同大学院システム情報科学府知能システム学専攻博士後期課程単位取得のうえ退学．博士（情報科学）．2002 年より広島市立大学情報科学部助手，現在に至る．データベース，グリッドコンピューティングの研究に従事．日本データベース学会，IEEE CS 各会員．

周藤　俊秀（正会員）

2002 年広島市立大学情報科学部知能情報システム工学科卒業．2004 年同大学大学院情報科学研究科知能情報システム工学専攻修士課程修了．同年 NEC システムテクノロジー株式会社入社．グリッドに関する研究・開発に従事．

北上　　始（正会員）

1976 年東北大学大学院工学研究科博士前期課程修了．同年富士通株式会社入社．以後，富士通研究所，新世代コンピュータ技術開発機構，国立遺伝学研究所客員助教授を経て，1994 年広島市立大学情報科学部教授，現在に至る．データマイニング，生命情報学，知識ベース等の教育研究に従事．博士（工学）．情報処理学会 25 周年記念論文．日本工学教育協会論文論説賞，情報処理学会一般情報処理教育小委員会委員，人工知能学会評議員，電子情報通信学会，IEEE，ACM 各会員．