並列 Modified PrefixSpan 法における動的負荷分散手法

高木 允　周藤 俊秀†　田村 慶一　北上 始

広島市立大学情報科学部　〒731-3194 広島県広島市安佐南区大塚東 3-4-1

†広島市立大学大学院情報科学研究科　〒731-3194 広島県広島市安佐南区大塚東 3-4-1

E-mail: {makoto, ktamura, kitakami}@db.its.hiroshima-cu.ac.jp , †tosihide@db.its.hiroshima-cu.ac.jp

あらまし モチーフ（アミノ酸配列上における特徴的な頻出パターン）発見のために，アミノ酸配列上で異なる位置にある頻出パターンを抽出する Modified PrefixSpan 法が提案されている．我々は，ひとつの PC クラスタ上において頻出パターンを並列に抽出する並列 Modified PrefixSpan 法の開発を行っている．動的負荷分散手法として，マスタ・ワーカ法とタスク・スティール法が提案されているが，本稿ではマスタ・ワーカ法にタスク・スティール法を加えたマスタ・タスク・スティール法を並列 Modified PrefixSpan 法に組み込み，評価を行った．実験の結果，効果的な負荷分散が行われていることが確認できた．

キーワード 動的負荷分散，並列 DB，データマイニング， PC クラスタ

Dynamic Load-Balancing of Parallel Modified PrefixSpan Method

Makoto TAKAKI　Toshihide SUTOU†　Keiichi TAMURA　and　Hajime KITAKAMI

Faculty of Information Sciences, Hiroshima City University, 3-4-1 Ozukahigashi,

Asaminami-ku Hiroshima-shi, Hiroshima, 731-3194 Japan

† Graduate school of Information Sciences, Hiroshima City University, 3-4-1 Ozukahigashi,

Asaminami-ku Hiroshima-shi, Hiroshima, 731-3194 Japan

E-mail: {makoto, ktamura, kitakami}@db.its.hiroshima-cu.ac.jp , †tosihide@db.its.hiroshima-cu.ac.jp

Abstract In order to discover motifs which are characteristic frequent patterns in the amino acid sequences, Modified PrefixSpan method is proposed to extract frequent patterns that exists in different position on the amino acid sequences. We are developing parallel Modified PrefixSpan method for extracting frequent patterns in parallel on a PC cluster. The master-worker paradigm and the task-steal paradigm are proposed as the dynamic load balancing technique. In this paper, we have incorporated the master-task-steal paradigm which integrated the task-steal paradigm to the master-worker paradigm, into the parallel Modified PrefixSpan method. We evaluated the master-task-steal paradigm and experimental results show good load balancing.

Keyword Dynamic Load Balancing , Parallel DB， Data Mining， PC Cluster

1. はじめに

モチーフ（アミノ酸配列上における特徴的な頻出パターン）は生物の進化の過程で保存されてきた蛋白質の機能に関係していると考えられている．モチーフ発見のために，アミノ酸配列上で異なる位置にある頻出パターンを抽出する Modified PrefixSpan 法[1]が提案されている．我々はひとつの PC クラスタ上において頻出パターンを並列に抽出するための並列 Modified PrefixSpan 法[2][3]の開発を行っている．

並列 Modified PrefixSpan 法は，典型的なマスタ・ワーカ法により動的負荷分散を行っている．並列処理を管理する 1 台の計算機が予め指定した閾値 s に対応する，s-頻出パターンを抽出する．s-頻出パターンについては 3.2 で詳しく述べる．抽出された s-頻出パターンは複数存在し，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処理を 1 つの子仕事と見なす．子仕事のことを以下，ジョブと呼ぶ．複数のジョブを，1 つ 1 つ並列処理を行う計算機に配る．ジョブを処理し終えた計算機は新たにジョブを受け取り，ジョブがなくなるまで処理を続けていく．ジョブを早く処理し終えた計算機が次々処理を続けていくことで動的な負荷分散が可能となっている．

並列 Modified PrefixSpan 法ではジョブの処理は互いに依存関係がなく独立に行えるが，ジョブの処理コストの差が大きく，処理コストを見積るのは配列の特徴に依存し予測不可能である．極端に大きなジョブを与えられた計算機の処理がなかなか終わらず効果的な台数効果が得られない．

以上の特徴により[2][3]で提案された並列 Modified PrefixSpan 法の動的負荷分散手法には次の 2 つの課題が存在した．ひとつは s を大きく設定することでジョブの粒度は細かくなるが，どこまでジョブを細かくすると各計算機にかかる負荷が均一になり，有効な台数効果が得られるかは実行してみないと分からないことである．もうひとつは，よりジョブを細かくすることで負荷の均一化を図ろうとすると，並列処理を管理する計算機に大きな負荷がかかってしまい，有効な台数効果が得られないことである．

本論文では,マスタ･ワーカ法を用いた並列 Modified PrefixSpan 法の動的負荷分散手法の問題点を解決するための負荷の再分配手法について提案する．提案する動的負荷分散手法はマスタ・ワーカ法とタスク・スティール法を組み合わせたハイブリッドな方法である（以下，マスタ・タスク・スティール法と呼ぶ）．

並列処理を管理する計算機だけがジョブを複数に分割するだけでなく，ジョブを受け取った計算機でもさらにジョブを細かくし，それらを動的に管理する．ジョブを処理し終えた計算機は並列処理を管理する計算機から次のジョブを受け取り，処理を続ける．並列処理を管理する計算機が全てのジョブを配り終えた後，処理を終えた計算機からジョブの請求を受けると各計算機に残っている処理中以外のジョブを全て回収し，もう一度配りなおす．こうすることで極端に大きなジョブを受け取った計算機の負荷を分散することができ，全ての計算機はジョブを全て処理し終えるまで均一に動くことが期待される．

並列 Modified PrefixSpan 法にマスタ・タスク・スティール法を組み込み，実際の PC クラスタに実装し，実験を行った．実験結果として 16 台で 1 台に比べ，約 15 倍の性能向上が得られた．各計算機の負荷が均一になっていることも確認できた．

本論文の構成は以下の通りである．第 2 章で関連研究について述べ，第 3 章では Modified PrefixSpan 法と並列 Modified PrefixSpan 法についての説明を行う．第 4 章でマスタ･タスク･スティール法の提案を行い，第 5 章でシステムの詳細設計について述べ,第 6 章で実験結果を示し，第 7 章でまとめる．

2. 関連研究

N 女王問題に代表されるような制約充足問題を並列化する場合の動的負荷分散手法に，マスタ・ワーカ法[4]とタスク・スティール法[5]とが提案されている．

マスタ・ワーカ法は，最適化問題などを分枝限定法で並列に解く場合などに用いられる．マスタで分枝操作を行い，仕事を細かくし，ワーカに配る．ワーカで解を求め，マスタに解を返す．この解を元に限定操作を行うことで有効な台数効果を得ることができる．しかし，Modified PrefixSpan 法の場合，マスタがどれだけ仕事を分割すれば効果的な台数効果が得られるかは，大雑把に近似するしかなく，極端に大きな仕事を与えられたワーカは他のワーカの仕事が終わっているにも関わらず，1 台で仕事を続けていかなければならない．マスタ・ワーカ法だけでは，各ワーカの負荷の均一化を図るという点では不十分である．

タスク・スティール法は，ワーカのみ存在し，並列計算を始める前と終わった後にあるワーカがマスタの役割を果たす．仕事を処理し終えたワーカは，仕事をかかえているワーカから仕事を得て処理を続ける．仕事の大きさが均一であるアプリケーションの場合，各ワーカが抱えている仕事の数をみれば，どのワーカから仕事をとれば負荷が均一になるか計算でき，有効な台数効果を得ることができる．しかし，Modified PrefixSpan 法の場合，仕事をかかえているワーカの中で，どのワーカが一番多く仕事をかかえているか数量的にしか判断できない．個数は少ないが，ひとつひとつの仕事量が多い仕事をかかえている場合，仕事が少ないと判断され，負荷が分散されにくい．また，ヘテロな環境を考慮した場合も同様に，各計算機にかかる負荷を分散することは難しい．

本論文で提案する手法は，マスタ・ワーカ法とタスク・スティール法の 2 つの機能を搭載し，負荷の再分配を行うことで，各ワーカの負荷の均一化を図る点が特徴である．

3. Modified PrefixSpan 法と並列 Modified PrefixSpan 法

3.1. Modified PrefixSpan 法

PrefixSpan 法[6]を改良した Modified PrefixSpan 法は支持率（頻出パターンの出現割合）だけでなく，ワイルドカード数（文字と文字の間の任意の文字数）も決める．処理の手順を以下に示す．

(1) 与えられた文字列データの集合に対して，支持率を満たす長さ 1 の頻出パターンを抽出する
(2) 求めた長さ N(N≧1)の頻出パターンをそれぞれ 1 文字目とし，長さ N+1 の頻出パターンを求める
(3) 頻出パターンがなくなるまで(2)を繰り返す

例として表 1 に示す 2 つの配列に対して，支持率を 100%，ワイルドカード数を 3 とした場合の頻出パターン抽出の例を図 1 に示す．Modified PrefixSpan 法は木構造で表すことができる．図 1 では，まず支持率を満たした長さ 1 の頻出パターンを幅優先探索で抽出する．図 1 では省略されているが 2 本の配列に共通して存在する “K”, “L”, “M”, “N”, “P”, “R”, “S”, “T” が頻出パターンとみなされる．次に各文字を接頭辞とし，2 文字目以降の頻出パターンの抽出を深さ優先探索で行う．

Modified PrefixSpan 法の特徴として複数に分けた頻出パターン抽出処理の負荷はそれぞれ異なる．図 1 中の “x” はワイルドカード（任意の 1 文字）を表している．図 1 に示されるように，例えば 1 文字目が “M” の場合，2 文字目以降に現れる頻出パターンを “N”, “S”としたときに，文字と文字の間のワイルドカード数がすべて等しくないために頻出パターンとはみなされていない．Modified PrefixSpan 法では，ワイルドカードの導入により，モチーフとなる可能性の低いパターンの抽出を以降行わないことで枝刈りを行っている．

表 1　配列データベース

配列 ID	配列
1	MFKALRTIPVILNMNKDSKLCPN
2	MSPNPTNHTGKTLR

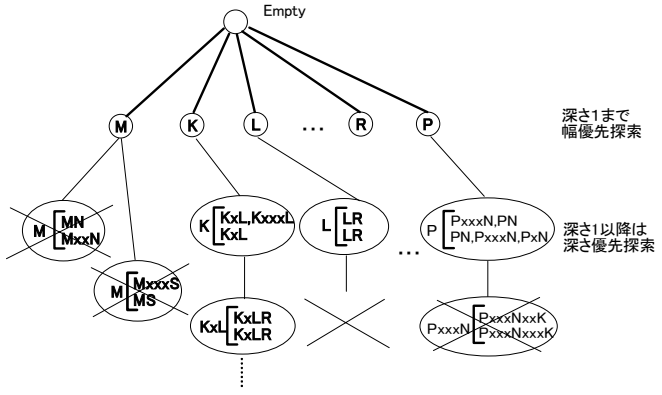

図 1　Modified PrefixSpan 法の頻出パターン抽出

1 文字目が “K” の場合，2 文字目以降に現れる頻出パターンを，“L”，“LR” としたときに，ワイルドカード数が等しくなっているので，この場合は頻出パターンとみなす．このように，Modified PrefixSpan 法は，次々に枝分かれをして頻出パターンの抽出が行われている．

3.2. 並列 Modified PrefixSpan 法

並列 Modified PrefixSpan 法は，ジョブを生成し管理するプロセスであるマスタと頻出パターンの抽出処理をおこなうプロセスであるワーカの 2 つのプロセスに分けて Modified PrefixSpan 法の並列化を行う．

Modified PrefixSpan 法はまず長さ 1 の頻出パターンを幅優先探索で求めるのに対し，並列 Modified PrefixSpan 法は，まずマスタでユーザが与えた任意の値の長さ（深さ）までの頻出パターンを幅優先探索で求める．このユーザが与える任意の値を閾値と呼び，アルファベットの文字を s 個含む頻出パターンを s-頻出パターンと呼ぶ．これによって得られた複数の s-頻出パターンを各ワーカが受け取り処理を続けていく．1 章で述べたように，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処理をジョブと呼ぶ．ジョブを処理し終えたワーカは，マスタに残っているジョブを受け取り，処理を続ける．図 2 に閾値を 2 として，5 プロセスで並列 Modified PrefixSpan 法を実行したときの頻出パターン抽出処理の例を示す．

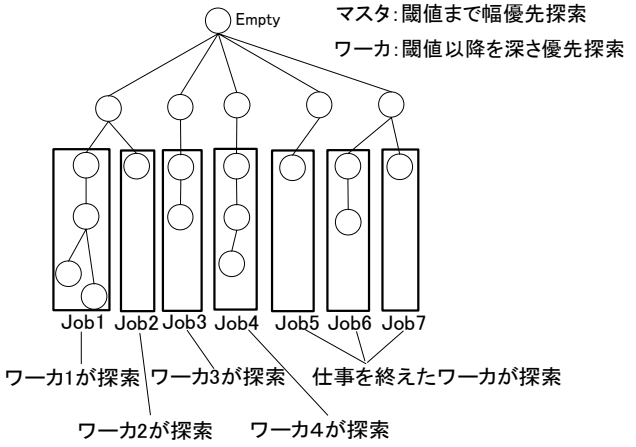

図 2　並列 Modified PrefixSpan 法の頻出パターン抽出処理

並列 Modified PrefixSpan 法は，マスタに閾値を設け，よりジョブを細かくし，処理を早く終えたワーカが残っているジョブの処理を行うことで負荷の分散を図っている．ここで問題となるのが図 3 に示すように，極端に大きなジョブを受け取ったワーカは，他のワーカが残りの全てのジョブの処理を終えていても，1 台で処理を続けなければならないということである．

図 3 の”Job1“～”Job7“は，図 2 の”Job1“～”Job7“を示している．処理を終えているワーカは，図 3 に示すように空き時間が発生してしまう．全体の処理時間は最も処理の長いワーカに依存するので，有効な並列効果を得ることができない．Modified PrefixSpan 法の特徴として頻出パターン抽出処理は互いに依存関係がなく独立に行えるが，処理のコストを見積もるのは配列の特徴に依存し予測不可能であることが挙げられる．この特徴により，予めワーカにかかる負荷の大きさを均

一に分割し，ジョブを配ることはできない．閾値を大きくして生成されるジョブをより細かくし，負荷の均一化を図ろうとすると，マスタに大きな負荷がかかってしまい，十分な並列効果を得ることができない．また，閾値をどのように設定すれば最適な並列処理を行えるか予測することもできない．

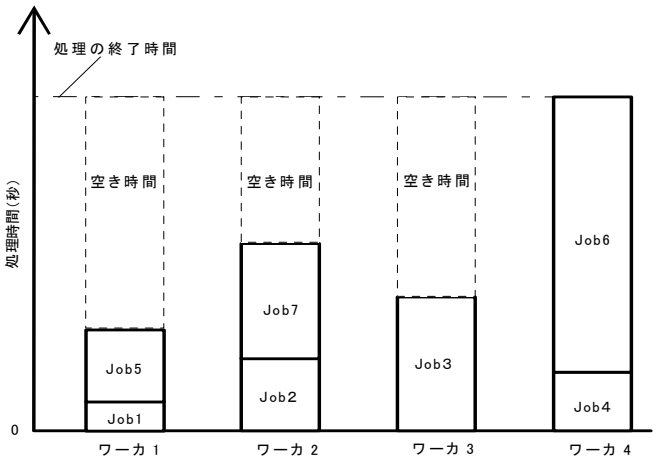

図 3　負荷の不均衡

4. マスタ・タスク・スティール法

既存の並列 Modified PrefixSpan 法ではマスタにのみ閾値を設けていたが，負荷の再分配を行うためにワーカにも閾値を設ける.図 4 に負荷の再分配の手順を示す.図 4 は，分かりやすいようにジョブの大きさを示しているが,実際はジョブの大きさを知ることはできない.ジョブを受け取ったワーカは閾値まで幅優先探索を行い，ジョブをより細かくする．分配されたジョブが大きなジョブでも，ワーカに閾値を設けるので，マスタの閾値を大きくしなくてもジョブを細かくでき，マスタに大きな負荷がかからなくなる．図 4 のジョブを点線で区切っているのは，ワーカでジョブを細かくしたことを示している．細かくされたジョブをひとつずつ深さ優先探索で処理していく．処理が全て終了したワーカはマスタにジョブの請求をし，新たなジョブを受け取り，処理を続ける．ワーカがジョブを請求した際にマスタにジョブがなければ（図 4 のワーカ 1 の処理が終了した時点),全てのワーカにある処理中以外のジョブをマスタが集め，ジョブを再分配する．ワーカ 1 の処理が終了した時点で，ジョブの回収が行われるが，”Job7-1“, ”Job3-2“, “Job6-1“は処理中なので回収されない.

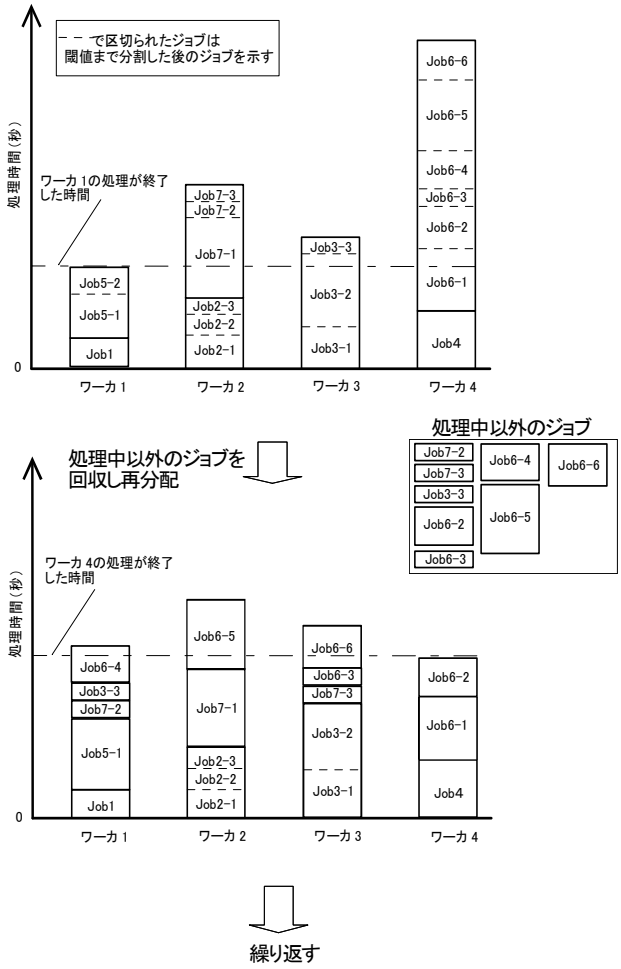

図 4　負荷再分配の手順

この方法を繰り返すことで，マスタのジョブがなくなるとすぐに終了していたワーカを，全てのジョブを処理し終えるまで働かせることができる．マスタとワーカに残っているジョブが全てなくなれば終了する.

5. 詳細設計

マスタとワーカの通信にはソケット通信と MPI ライブラリを利用する．マスタが複数台のワーカと通信を行うために，マルチスレッド方式を用いた．マスタはワーカの台数だけスレッドを生成する．以後，このスレッドをマスタスレッドと呼ぶことにする．従来の並列 Modified PrefixSpan 法は，マスタにのみジョブを格納する Global Job プールという構造体を作っていた．本手法では，ワーカにもジョブを格納する構造体を作成し，各ワーカでジョブの管理ができるようにする．この構造体を Local Job プールと呼ぶことにする.また，ワーカにもひとつスレッドを作成する．このスレッドを待機スレッドと呼ぶことにする．待機スレッドは，Local Job プールを監視しており，マスタスレッドから

ジョブ回収の合図を受けると，Local Job プールの全てのジョブを回収して，マスタに送る．図 5 にシステムの構成図を示す．

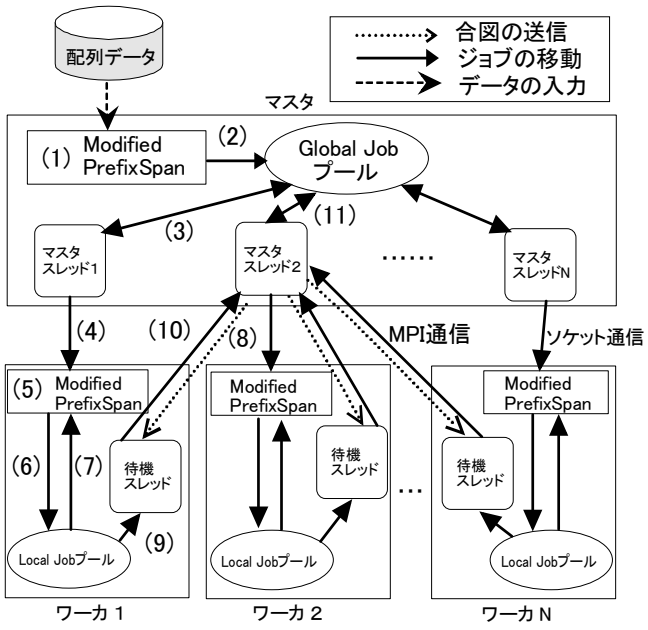

図 5　システム構成図

以下，図 5 の流れについて説明する．

(1) マスタでユーザが与えた閾値まで頻出パターンの抽出を行う
(2) 得られたジョブを Global Job プールに入れる
(3) マスタスレッドが Global Job プールからジョブをひとつ取り出す
(4) マスタスレッドはワーカからジョブの請求を受けるとジョブを送る
(5) ワーカでユーザが与えた閾値まで頻出パターンの抽出を行う
(6) ワーカで得られたジョブを Local Job プールに入れる
(7) ワーカは Local Job プールのジョブがなくなるまで Local Job プールからジョブをひとつずつ取り出し，深さ優先探索で頻出パターンの抽出を行う
(8) Global Job プールにジョブがないときにワーカからジョブの請求を受けると，マスタスレッドは全ての待機スレッドに合図を送る
(9) 合図を受け取った待機スレッドは Local Job プールから全てのジョブを取り出す
(10) 取り出したジョブを，合図を送ったマスタスレッドに送る
(11) 取り出したジョブを受け取ったマスタスレッドは Global Job プールにジョブを入れる
(12) 回収したジョブが 0 個ならば(13)へ，そうでなければ(3)へ戻る
(13) マスタが待機スレッド，ワーカに終了の合図を送り，終了

6. 評価実験

詳細設計をもとに，マスタ・タスク・スティール法を組み込んだ並列 Modified PrefixSpan 法を PC クラスタ上に実装し，実験を行った．マスタの働きをする計算機に，ワーカの動作も行うように設定を行った．すなわち，1 台だけマスタとワーカの 2 つのプロセスが動いていることになる．16 台程度の場合，マスタにあまり負荷がかからないので，1 台で 2 プロセス動作させても問題はない．表 2 に実験に用いた計算機の性能を示す．

表 2　実験に用いた計算機の性能

PC　16 台	・CPU Pentium4 2.53GHz (L1 cache 16KB, L2 cache 512KB) ・Memory PC 2700 1.5GB ・CHIPSET Intel 845GE ・DISK 80GB (Seagate ATA 7200rpm) ・NIC Intel PRO/1000MT
Network	100Mbit Ethernet Switching HUB
OS	Red Hat Linux 9.0
コンパイラ	GNU g++　ver. 3.2.2
MPI	mpich-1.2.5

この実験に使用したアルファベットの文字列は，アミノ酸配列のモチーフを含む配列データをデータベース化しているサイト，PROSITE[7] の Kringle データセットと Zinc Finger データセットを使用した．これらの詳細データを表 3 と 表 4 に示す．

表 3　使用データの詳細 1

	データ件数(件)	総長(byte)
Kringle	70	23385
Zinc Finger	467	245595

表 4　使用データの詳細 2

	平均長 (byte)	最大長 (byte)	最小長 (byte)
Kringle	334	3176	53
Zinc Finger	525	4036	34

6.1. Kringle データセットによる性能評価

負荷の再分配の性能評価を行うために，まず，Kringle データセットを用い，支持率，ワイルドカード数を固定し，閾値を変化させ性能評価を行った．図 6 と図 7 は支持率を 20%，ワイルドカード数を 5 と固定して，計算機の台数を 1 台から 16 台まで変化させたときの性能向上比を示したものである．() 内の数字は，（マスタの閾値，ワーカの閾値）を示している．

図 6 は既存の並列 Modified PrefixSpan 法と同じ手法（マスタ・ワーカ法）で実験を行った場合である．ワーカには閾値を設けていないのでワーカでの閾値を 0 としている．図 7 は本手法（マスタ・タスク・スティール法）を用いたときの性能向上比である．

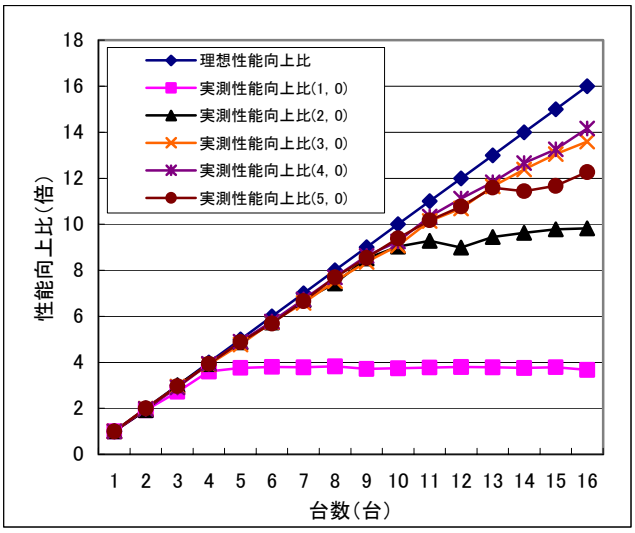

図 6　マスタ・ワーカ法を用いた場合の性能向上比

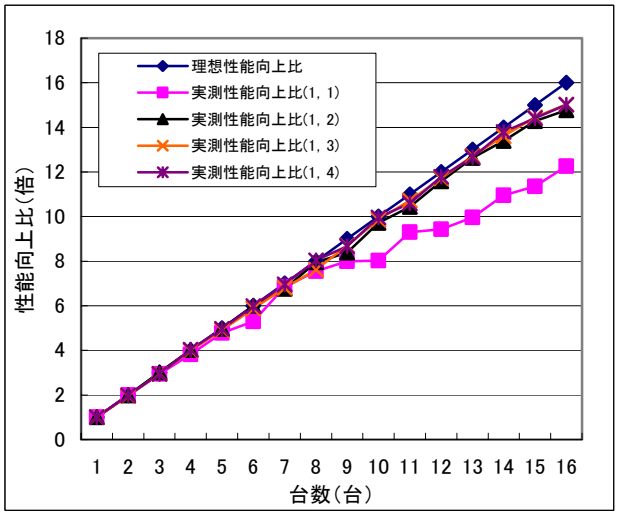

図 7　本手法を用いた場合の性能向上比

図 6 より，マスタでの閾値を上げていくと理想性能向上比に近づいていることが分かる．これは，マスタの閾値が大きいほうがジョブの粒度を細かくすることができ，ワーカにかかる負荷のばらつきが小さくなったからだと考えられる．しかし，マスタでの閾値を 5 にすると 4 の場合よりも性能向上比が下がっている．この原因として，マスタに大きな負荷がかかってしまい，台数を増やし，処理時間が短縮されるにつれ，マスタでの処理時間を無視できなくなるためだと考えられる．

図 7 をみると，閾値が(1,1)のときを除き 16 台で約 15 倍の性能向上を示している．各ワーカの負荷が均一になり，有効な台数効果が得られたと考えられる．

負荷が均一になっていることを示すために，16 台で

- 負荷の再分配を行わない
- 閾値 (1,0)

の場合と

- 負荷の再分配を行う
- 閾値 (1,3)

としたときの各ワーカの処理時間を調べた．前者を A，後者を B とする．図 8 に A の場合の各ワーカの処理時間を示し，図 9 に B の場合の各ワーカの処理時間を示す．図中の ”w1“ ～ ”w16“ はワーカを示している．

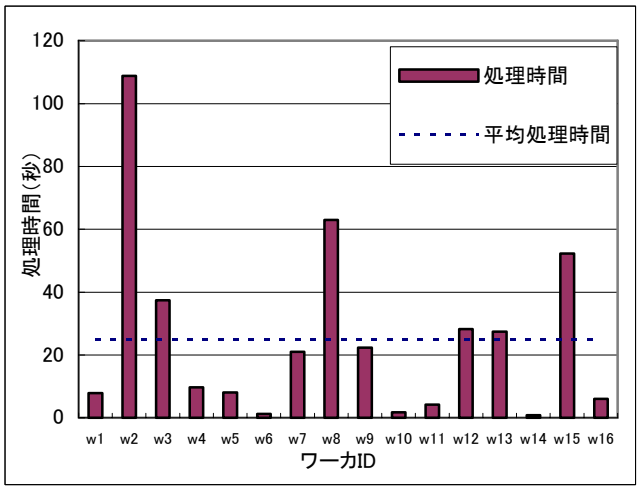

図 8　マスタ・ワーカ法(A)の各ワーカの処理時間

図 8 では各ワーカの処理時間にかなりばらつきがでている．全体の処理時間は処理が一番遅い ”w2“ の処理時間に依存している． ”w2“ の処理が終わるまで他のワーカは待ち時間が発生している．平均処理時間は約 25 秒であった．図 9 では，各ワーカの処理時間がほぼ等しくなっており，負荷が均一化されていることが分かる．このときの平均処理時間は約 26 秒であり，図 8 とほぼ等しくなっている．

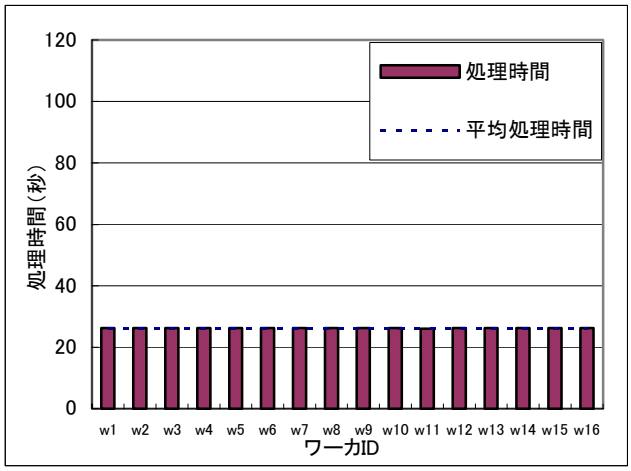

図 9　本手法(B)の各ワーカの処理時間

閾値を(1,1)と設定した場合，他の閾値に設定した場合と比べると 16 台で約 12 倍と性能向上比が低い．この原因として，ジョブの粒度が十分に細かくならないことが考えられる.そのことを示すために図 10 に閾値を(1,1)と設定したときの各ワーカの処理時間を示す.

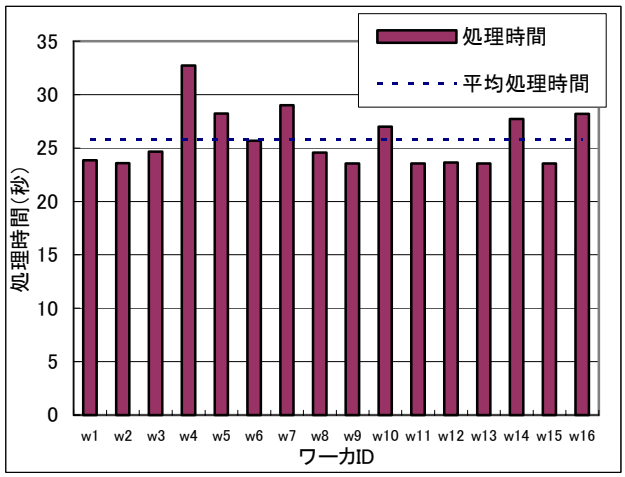

図 10　閾値(1,1)の場合の各ワーカの処理時間

粒度が粗いと，再分配の対象となるジョブが少なくなり, 各ワーカの処理時間に差ができている. マスタ・ワーカ法と同様の問題が生じてしまう．この各ワーカの処理時間の差が影響して性能向上比の低下に繋がったと考えられる.

次に，図 8 と図 9 に示すワーカの総処理時間と総 CPU 使用時間，CPU 使用率を表 5 に示す．総処理時間とは，全てのワーカが処理を終えるまでの時間を合計したもので，総 CPU 時間とは全てのワーカの CPU 時間を合計したものである．CPU 使用率は，（総 CPU 時間／総処理時間）*100 である.

表 5 ワーカの総処理時間，総 CPU 時間，CPU 使用率

測定項目 / 処理方法	総処理時間（秒）	総 CPU 時間（秒）	CPU 使用率（%）
A	1741.77	392.21	22.52
B	419.24	398.78	95.12

A の CPU 使用率が低いのは，処理を終えたワーカの待ち時間が多いからである．B は 95%と高いが，5%程度 CPU に空きの時間が発生している. この原因として，ジョブを請求して受け取るまでの待ち時間，排他制御による待ち時間が挙げられる．A の場合もこれらの時間は発生しているが，B は何度もジョブを回収し分配するので A に比べそれらの待ち時間は長い．また，A よりも B の総 CPU 時間が多い．ジョブを回収し，負荷の再分配を行う処理により B の仕事量が増えたからだと考えられる.

次に，どの程度通信が行われていたか調べたものを図 11 に示す.

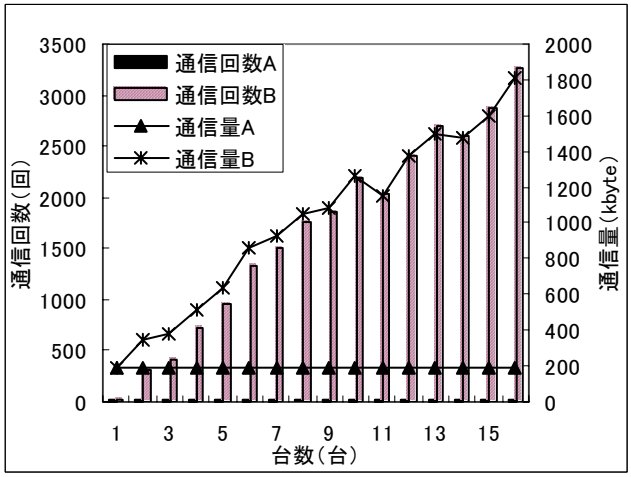

図 11　通信回数と通信量

A の場合，台数を増やしても通信量，通信回数は常に一定である．B の場合，台数を増やしていくと通信量, 通信回数ともに増加している. 16 台使用した場合，全体で約 1.8Mbyte の通信が起きているが, この程度の通信は処理時間にほとんど影響しない.

6.2. Zinc finger データセットによる性能評価

6.1 と同様に, 支持率とワイルドカード数を固定し，閾値を変化させ性能評価を行った．図 12 と図 13 は支持率を 30%，ワイルドカード数を 5 として 16 台まで変化させたときの性能向上比を示したものである.

図 12 はマスタ・ワーカ法を用いて実験を行ったときの性能向上比である. Kringle データセットと同様に有効な性能向上比は得られていない.

図 13 は本手法を用いた場合の性能向上比である．図中に台数以上の性能向上比を示している箇所がある．1 台の計算機で処理した際にメモリが足りなくなり，処理が遅くなったことが原因だと考えられる．閾値を(1,1)，(1,2)とした場合，有効な性能向上比が得られていない．Kringle データセットを用いて実験を行った際の閾値を(1,1)と設定したときと同様に，ジョブの粒度が細かくならず，各ワーカの処理時間にばらつきがでてしまったためと考えられる．

Kringle データセットの場合，閾値を(1,2)と設定すると負荷が均一化され，効果的な性能向上比が得られたが，Zinc Finger データセットの場合，閾値を(1,2)としても効果的な性能向上比が得られなかった．Zinc Finger データセットの特徴により，閾値を上げないとジョブが十分に細かくならなかったからだと考えられる．閾値を(1,4)に上げると 16 台で約 15 倍と有効な性能向上比が得られた．処理時間の絶対値も他の閾値に比べ早くなっていた．

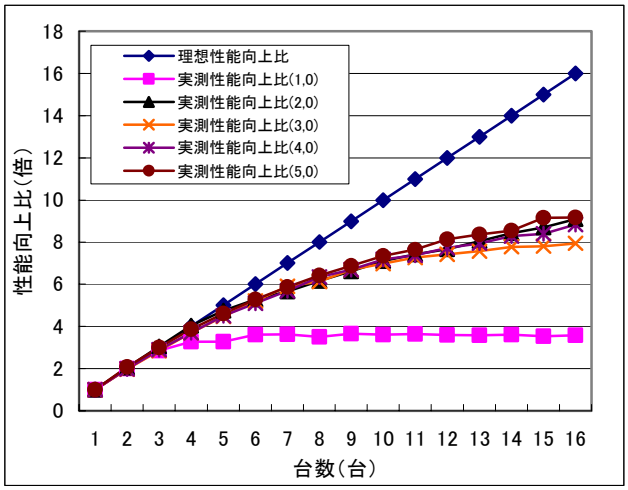

図 12　マスタ・ワーカ法を用いた場合の性能向上比

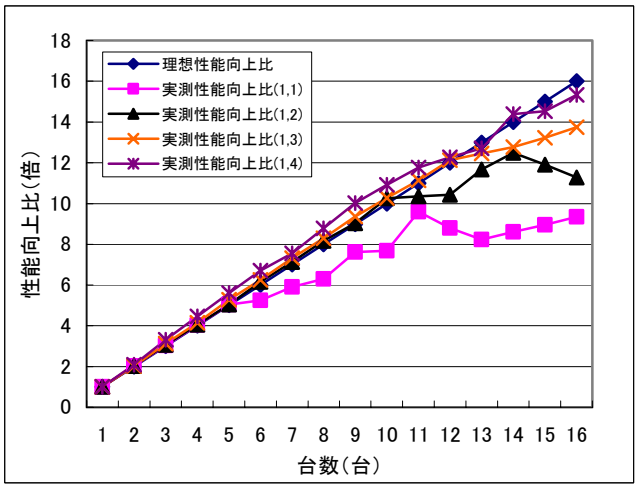

図 13　本手法を用いた場合の性能向上比

7. おわりに

本論文では，並列 Modified PrefixSpan 法にマスタ・ワーカ法とタスク・スティール法を組み合わせた，マスタ・タスク・スティール法を実装し，実験を行った．その結果，ワーカの負荷が均一化され，16 台で約 15 倍の性能向上比を得ることが確認できた．

しかし，閾値の決め方によっては十分にジョブが細かくならず，うまく負荷が分散されなかった．今後の課題として，ユーザが閾値を決めなくてもジョブを分割し，負荷分散を行うことができるようにする必要がある．また，大規模な配列を用いて実験を行い，通信による遅延を考慮しながら動的な負荷の分散を行って行く予定である．

8. 謝辞

本研究について有益なコメントや議論をしていただいた，広島市立大学情報科学部 黒木進 助教授，森康真 助手に感謝いたします．なお，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））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, Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence, pp.482-491, Springer-Verlag, August 2002.

[2] Toshihide Sutou, Keiichi Tamura, Yasuma Mori and Hajime Kitakami, Design and Implementation of Parallel Modified PrefixSpan Method, The Fifth International Symposium on High Performance Computing, Springer-Verlag, pp.412-422, October 2003, Tokyo.

[3] 周藤俊秀, 田村慶一, 森康真, 北上始：並列 Modified PrefixSpan 法の設計と実装，日本データベース学会 Letters, Vol.2. No.3, pp.25-28, 2003 年 12 月．

[4] Nicholas Carriero and David Gelernter, How to Write Parallel Programs, The MIT Press, London, 1990.

[5] Matthias Korch and Thomas Rauber, Evaluation of Task Pools for the Implementation of Parallel IrregularAlgorithms, ICPP Workshops, pp.597-606, 2002.

[6] J. Pei, J. Han, B. Mortazavi-Asl and H. Pinto: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, Proceedings of the Seventh International Conference on Data Engineering, IEEE Computer Society Press, pp.215-224, 2001.

[7] PROSITE http://www.expasy.ch/prosite