アルゴリズム 5－2
（1989. 1. 25）

並列ストリングパターンマッチングアルゴリズムとインプリメンテーション

堀口　進　　小川則之　木村正行

東北大学　工学部　情報工学科

〒980　仙台市荒巻字青葉

あらまし　VLSI技術の進歩により、高性能なマイクロプロセッサを多数用いたマルチマイクロプロセッサシステムが盛んに研究され、並列処理による高速化が注目されている。文字列データの中から与えられた文字列を探索する操作は、文書編集、データ検索や記号処理において基本的操作としてしばしば利用される。近年、データベースやワープロ文書の扱うデータ量は益々膨大なものになり、文字列パターンマッチングの高速化が切に望まれている。我々は、マルチプロセッサ上で高速処理可能な並列ストリングパターンマッチングアルゴリズムを提案した。本論文では、これらの並列パターンマッチングアルゴリズムを32台のマイクロプロセッサからなるクラスタ型マルチプロセッサシステムMUGEN上に、インプリメントし、処理性能を測定した結果について述べる。

Parallel Pattern Matching on Multiprocessor System

Susumu HORIGUCHI　Noriyuki OGAWA　Masayuki KIMURA

Department of Information Science, Faculty of Engineering
TOHOKU University, Sendai, 980, JAPAN

Abstruct

VLSI technology has made possible construction of multiprocessor system with a number of processors. String patterns matching is a fandamental operation in the fields such as database, word processing, symbolic processing. As the total amount of data to be treated becomes huge in such fields, the fast algorithm of pattern matching has been desired. A parallel processing of pattern matching on a muntiprocessor system is capable of more improvement of matching speed for huge data set. We have proposed two paralle pattern matching algorithms on multiprocessor systems, which are based on Boyer-Moore algorithm and are suited for linearly connected multiprocesor system and common memory multiprocessor system. In this paper, we show the implimentation of these algorithms on multiprocessor system MUGEN with 32 processors. The performance evaluation of algorithms are also discussed here in ditail.

1．はじめに

文字列データの中から与えられた文字列を探索する操作は、文書編集、データ検索や記号処理を行う時にしばしば利用される。その応用範囲は広く、効率のよいパターンマッチングアルゴリズムの開発は、たいへん重要である。パターンマッチングアルゴリズムに関する研究は、古くから多くの研究者が行なっている。Kunth, Morris, そして Pratt[1][2]は、パターンマッチング問題が線形時間で解けることを示した。 Boyer と Moore[3]は、パターンをストリングの末尾から照合を行い、平均的にはパターンが長くなれば計算時間が短くなるアルゴリズムを提案した。このアルゴリズムはBoyer-Moore法とよばれ、高速なパターンマッチングアルゴリズムとして、改良が重ねられ、現在広く使用されている。 また、AhoとCorasick[4]は複数の文字列データを同時に扱えるパターンマッチングアルゴリズムを提案している。

近年、データベースやワープロ文書の扱うデータ量は益々膨大なものになり、文字列パターンマッチング処理のより一層の高速化が望まれている。高速処理の1つの方法として、演算処理を分割し同時に並行して処理を行う並列処理方式が近年注目されている[5][6][7]。 特に、VLSI技術の進歩により、高性能なマイクロプロセッサを多数用いて、マルチマイクロプロセッサシステムを構成することが可能となった。この方法は、高性能で安価なコンピュータを構築するためのアプローチとして注目され、実際にＣＭ*[8]、Ｃｅｄｅｒ[9]、ＰＡＸ[10]等の様々な試作機が製作されている。さらに、大規模な商用システムも開発されつつあり、ベクトルプロセッサの処理速度に匹敵するものが期待されるまでに至っている。このようなマルチプロセッサシステムでは数多くの処理が並列化され，高速処理が実現されている。Nielsen と Staunstrup[11]は、problem-heapアルゴリズムというマルチプロセッサシステムで一つのパターンを高速にパターンマッチングを行なう方法提案している。我々は、複数のパターンをマルチプロセッサ上で文字列データの中から探索する二つの並列パターンマッチングアルゴリズムを提案した[12]。二つの並列パターンマッチングアルゴリズムは、パイプライン法と、ローカルメモリ法のとよばれ、Boyer-Moore法を基にした並列パターンマッチングアルゴリズムである。

本論文では、32台のマイクロプロセッサからなるクラスタ型マルチプロセッサシステムＭＵＧＥＮ上に、これらの並列パターンマッチングアルゴリズムをインプリメントし、処理効率を測定した結果についてのべる。次章では、マルチプロッセッサシステムＭＵＧＥＮのハードウェアとシステムソフトウェアの概要について述べる。第３章では、パイプライン法とローカルメモリ法の並列ストリングマッチングアルゴリズムについて説明する。第４章では、マルチプロッセッサシステムへの並列アルゴリズムのインプリメント、およびその性能結果について検討する。

2．クラスタ型マルチプロセッサシステムＭＵＧＥＮ

マルチプロセッサシステムのプロセッサ結合方式として種々の形態が提案されているが、大別してネットワーク型と共有メモリ型の2方式にまとめられる。ネットワーク型は特定の問題向きには最適のシステム構成が採れるため、最大処理効率を実現できる可能性があるが、汎用性は少ない。共有メモリ型のプロセッサ結合方式で最も広く使用されているバス結合型は、ハードウェア構造が簡単でアルゴリズムに対して柔軟性があるという利点がある。

共有メモリ型の中で最も基本的なものは、単一のバスに多数のプロセッサを接続した単一バス方式である。しかし、この方式は各プロセッサの1回の処理量が少ない場合、プロセッサ数の増加に従ってバス競合によるオーバヘッドが増大し、処理効率が低下してしまう。バス競合を緩和する方式として、複数のバスを用いる複線バス方式や、プロセッサをいくつかのクラスタ単位にまとめるクラスタ方式がある。複線バス方式は、全てのプロセッサ間通信が同様に行なえる点で有効であるが、ハードウェア量は単一バス方式に比べてかなり増加する。クラスタ方式ではバスを階層化するため、異なるクラスタ内に属するプロセッサ間の通信は同一クラスタ内のプロセッサ間通信よりオーバヘッドが大きくなるが、ハードウェアの付加は複線バス方式と比較して少なくてすむ。

我々は種々の理論的検討に基づいて、32台のマイクロプロセッサを8台ずつ4クラスタにまとめたクラスタ型試作システムＭＵＧＥＮ（MUltiprocessinG system with pErfect connection Network between clusters）を設計構築した[13]。また、ＭＵＧＥＮ上で効率のよい処理を行なうためのシステムソフトウェアの開発を行なった。

2．1　ハードウェア構成

ＭＵＧＥＮシステムの構成を図１に示す。システム全体を管理するマスタプロセッサ（ＭＰ）の下に4つのクラスタがある。各クラスタは、クラスタ内のバスを管理するバスコントローラ（ＢＣ）、データの受け渡しに使用するクラスタコモンメモリ（ＣＭ）、およびローカルメモリ（ＬＭ）をもつ8つのスレーブプロセッサ（ＳＰ）で構成されている。

それぞれのＢＣ間はパラレルポートで完全結合され、異なったクラスタ間でのデータ通信を容易にしている。また隣接するＳＰ間はシリアルポートで接続され、これによって共有バスを使用しないでデータの交換を行なう

ことができる。バスはアドレスバス16ビット、データバス8ビットから成る。各ＳＰとＢＣ間および各ＢＣとＭＰ間はパラレルポートを通して結ばれる。試作システムはＳＰ32＋ＢＣ4＋ＭＰ1の計37個のマイクロプロセッサからなり、メモリは、ＲＯＭがＭＰ8＋ＢＣ8×4の40KB、ＲＡＭが、ＭＰ56＋ＢＣ8×4＋ＳＰ32×32＋ＣＭ32×4の1242KBで計1282KBから構成されている。

2．2　システムソフトウェア構成

ＭＵＧＥＮのために設計開発されたシステムソフトウェアは、大きく分けてモニタと並列処理言語である。開発したシステムソフトウェアの構成概念を図2に示す。

モニタプログラムはＭＰ用モニタ、ＳＰ用モニタそれにＢＣ用モニタで構成される。それぞれのモニタはシステムの各プロセッサのメモリ上に常駐し、システムの基本制御やプログラムの実行・デバッグなどの際のシステムとのインターフェースを行なう。また実際のプログラムを記述するための各種基本的ルーチンをシステムコールとして用意している。ＭＰ用のモニタプログラムはＭＵＧＥＮシステムで並列処理プログラムを実行するために必要な22種類のコマンドと33種類のシステムコールサブルーチンを用意した。

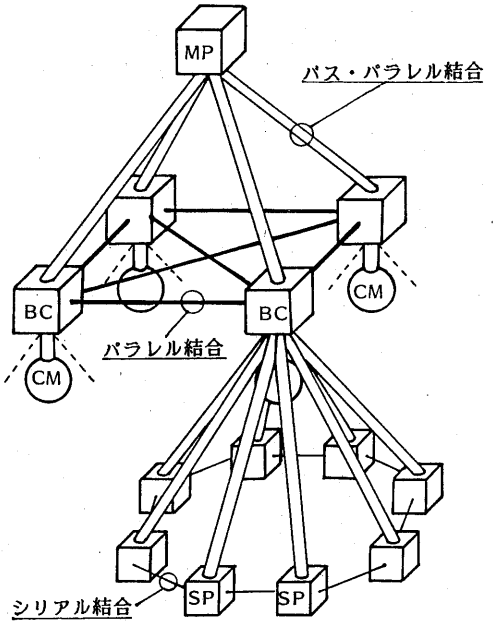

図1　クラスタ型マルチプロセッサシステム構成図

Fig.1 System configuration of the Clustered Multiprocessor System MUGEN

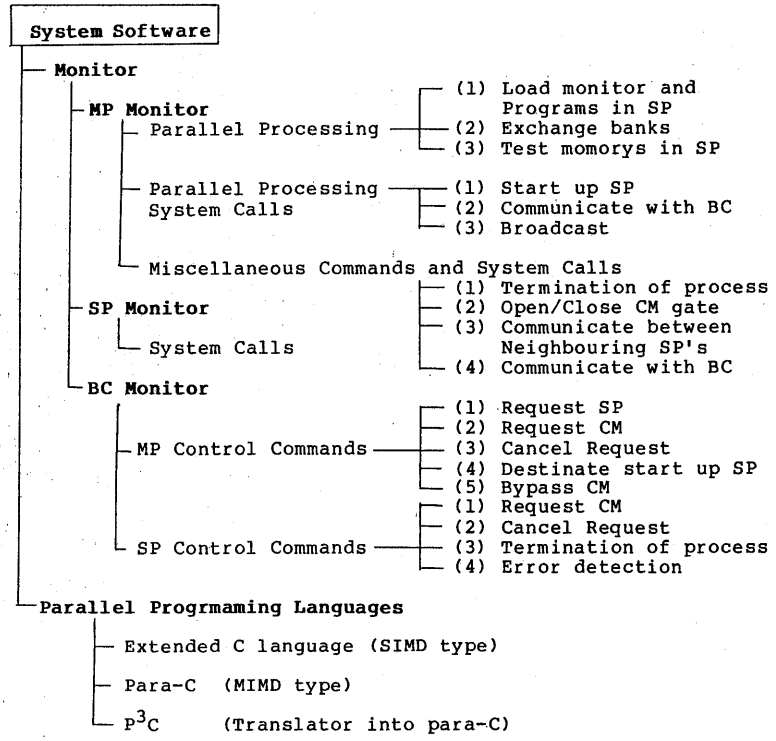

図2　ソフトウェア構成図

Fig.2 Software Configuration Developed on MUGEN

アプリケーションプログラムの開発を行う上で、アセンブラで種々の並列処理アルゴリズムを全て記述することは非常に時間がかかり、新しい並列処理アルゴリズムなどの開発を行うことは不可能に近い。そのため、プログラム開発が効率よく行え、システムの機能を十分に発揮できる高位言語は必要不可欠である。そこで、本システム上で実行可能なオブジェクトを生成できる二つの並列処理言語のコンパイラを開発した。複数のプロセッサ上で同一のプログラムによってデータを処理するＳＩＭＤ(Single Instruction stream / Multiple Data stream)型処理が可能な並列Ｃ言語『ext-C』。そして、複数のプロセッサ上で異なったプログラムの処理も可能とする、ＭＩＭＤ（Multiple Instruction stream / Multiple Data stream）型処理が可能である並列Ｃ言語『para-C』を設計・開発した。

更に、通常のＣ言語で記述されたプログラムを para-C 用のプログラムに変換するプリプロセッサ P3C (Pre-Processor for Para-C) を設計し、インプリメントした。P3C は、Ｃ言語ソースプログラム中の for ループを解析し、para-C の並列処理ルーチンへの展開を行なう。これを用いることによって、通常のＣ言語で記述されたプログラムを、自動的に para-C 用プログラムに変換し、高速に並列実行することができる。

3．並列パターンマッチングアルゴリズム

文字列パターンマッチングは問題として簡単であるが、膨大なデータベースや編集文書の情報検索の基本操作としてひんぱんに利用され、その高速化が望まれている。ここで扱う並列パターンマッチングは、膨大な文字列データテキストＴから、与えられたＭ個の文字列パターン（Ｓ1, Ｓ2,..., ＳＭ）をＰ台のプロセッサを使って全テキストのどの位置に現れるかを調べることである。

先ず、アルゴリズムを説明する前に，並列パターンマッチングアルゴリズムにおける各パラメータを下記のように記述する。

文字列テキスト	Ｔ
Ｍ個の文字列パターン	Ｓ1, Ｓ2,..., ＳＭ
文字列サブテキスト	Ｔ1, Ｔ2,..., ＴＰ
文字列テキスト長	t
Ｍ個の文字列パターン長	ｓ1, ｓ2,..., ｓＭ
文字列サブテキスト長	ｔ1, ｔ2,..., ｔＰ

ここでは、t>>sj、つまり、テキスト長が探索したい文字列パターン長に比較して十分大きい場合を考える。また、文字列テキストＴは、文字列サブテキスト｛Ｔ1, Ｔ2,..., ＴＰ｝に分けて考える。

次に、プロセッサの隣接結合の機構を利用したパイプライン法と、共有メモリ型マルチプロセッサシステムに有効なローカルメモリ法の二つの並列パターンマッチングアルゴリズムを示す。

3．1　パイプライン法

トーラス結合は、単純な結合形態でありプロセッサ結合によく用いられる。パイプライン法は、トーラス状に結合されたプロセッサで高速に並列パターンマッチングを行なうアルゴリズムである。

パイプライン法による並列パターンマッチングアルゴリズムを以下に示す。

1) ｊ番目の各プロセッサにｍ個（ｍ＝Ｍ／Ｐ）の検索用パターン｛Ｓ(j-1)m+1, Ｓ(j-1)m+2,..., Ｓjm｝を与える。
2) 全テキストＴをＰ個のサブテキスト｛Ｔ1, Ｔ2,.., ＴＰ｝に分割してｊ番目のプロセッサにＴｊを配分する。
3) 各プロセッサでは、ＢＭ法に基づくてパターンマッチングを開始する。
4) プロセッサに与えられたｍ個のマッチングを終了した時、隣接する上位のプロセッサにｍ個のパターンを転送し、下位のプロセッサから新しいｍ個の文字列パターンを受け取る。この時、テキストの切れ目においては、パターンマッチングのリスト情報をＢＭアルゴリズムのテーブルと一緒に転送する。
5) パイプライン的にマッチングを行い、全パターンのマッチングが終了するまで4)を繰返す。

パイプライン法の概念を 図３に示す。パイプライン法は、各プロセッサに割り当てられたサブテキストから探索したい複数パターンのマッチングをＢＭ法で行なう。

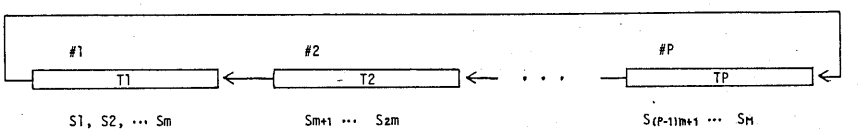

図３ パイプライン型並列パターンマッチング法
Fig.3 Parallel Pattern Maching Metod by Pipelinig Scheme

今まで、プロセッサ数Pは一般に、探索したいパターンの数Mより少ない場合（m＝M／P＞1）について述べた。プロセッサ数がパターン数より、多い場合は、同じサブパターンを異なったプロセッサに割当て、同様なアルゴリズムでパイプライン的にパターンマッチングを行なう。

3．2　ローカルメモリ法

一方、ローカルメモリ法は各プロセッサに与えられたパターンはそのままで、サブテキストが次々にローカルメモリにロードされパターンマッチング処理を行う。ローカルメモリ法のパターンマッチングアルゴリズムを下記に示す。

1) j番目の各プロセッサにm個（m＝M／P）の検索用パターン{S(j-1)m+1, S(j-1)m+2,..., Sjm}を与える。
2) 各プロセッサのローカルメモリに同じサブテキストをコモンメモリから読み込む。
3) プロセッサはBM法のテーブルを作成し、パターンマッチングを開始する。
4) プロセッサに与えられたm個のマッチングを終了した時、共有メモリから別のサブテキストをロードする。テキストの切れ目においては、パターンマッチングのリスト情報をローカルメモリに保存する。
5) 全サブテキスト {T1、T2,... TQ} のパターンマッチングが終了するまで2)-4)の処理を続ける。

図4にローカルメモリ法による並列パターンマッチングを示す。ローカルメモリ法は共有メモリ型マルチプロセッサシステムに有効なアルゴリズムである。これらのアルゴリズムは並列処理言語para-Cにて記述され、クラスタ型マルチプロセッサシステムMUGENにインプリメントされている。

4．並列パターンマッチングの性能評価

並列パターンマッチングの性能測定に用いたテキストは、サンスクリットの法華教経典のデータベース[14]を使用いた。図5に、パイプライン法を用いた並列パターンマッチングの実行処理時間を示す。パターン長（s＝s1=s2=･･･=sM）が、それぞれ4文字、8文字、16文字列の48組のパターンを8台、16台、24台のプロセッサで処理した。パターン長が長く、プロセッサ数が多くなればなるほど並列パターンマッチングの実行処理時間が短くなることがわかる。図6に、パイプライン法を用いた並列パターンマッチングの単一のプロセッサとの処理速度高上比を示す。パターン長8文字の48組のパターンを24台のプロセッサで処理した場合に、15.2倍の処理速度向上比が得られることがわかる。パターン長が長い場合には、プロセッサ数を多くすると、隣接プロセッサ間のBM法のテーブルおよびサブテキストの切れ目のパターンマッチング情報のリスト転送オーバーヘッドによりプロセッサ数に比例した処理速度高上比が得られない。

図7に、ローカルメモリ法を用いた並列パターンマッチングの実行処理時間を示す。ここではサブテキストのローカルメモリへのデータ転送に要する時間を省いてある。ローカルメモリ法は、プロセッサ数が多い時、パイプライン法より速い実行処理速時間が得られる。図8に、ローカルメモリ法を用いた並列パターンマッチングの一台のプロセッサとの処理速度高上比を示す。ローカルメモリ法は、プロセッサ数にほとんど比例した処理速度向上比がえられている。パイプライン法に比較して、隣接プロセッサ間を複数のパターン、テーブルならびにリスト情報が移動するオーバヘッドを含んでいないためである。

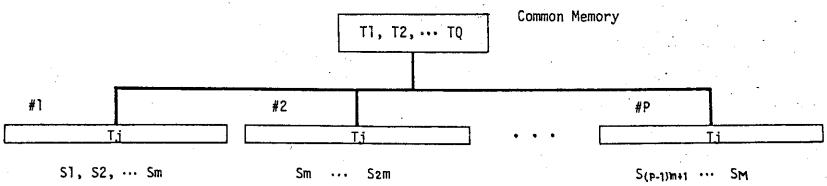

図4　ローカルメモリ型並列パターンマッチング法
Fig.4 Parallel Pattern Maching Metod
by Local Memory Scheme.

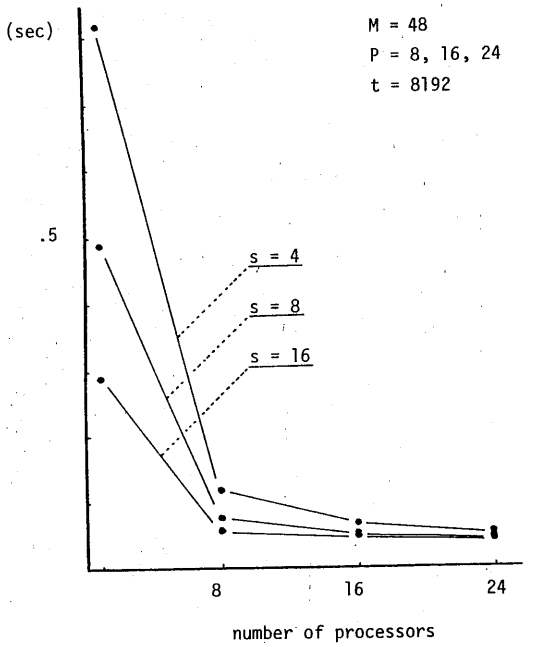

図5　パイプライン型並列パターンマッチングの処理実行時間

Fig. 5　Execution time of Parallel Pattern Maching by Pipelining Scheme

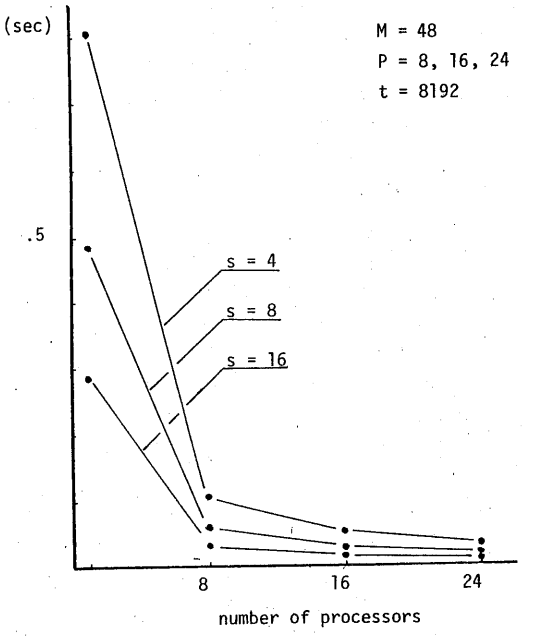

図7　ローカルメモリ型並列パターンマッチングの処理実行時間

Fig. 7　Execution time of Parallel Pattern Matching by Local Memory Scheme

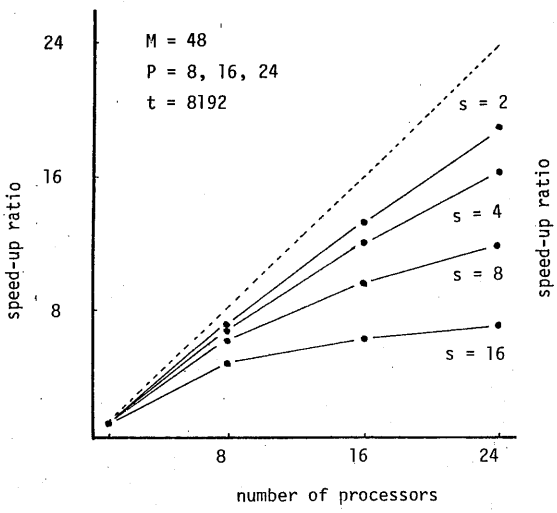

図6　パイプライン型並列パターンマッチングの処理速度向上比

Fig. 6　Speed-up Ratio of Parallel Pattern Maching by Pipelining Scheme

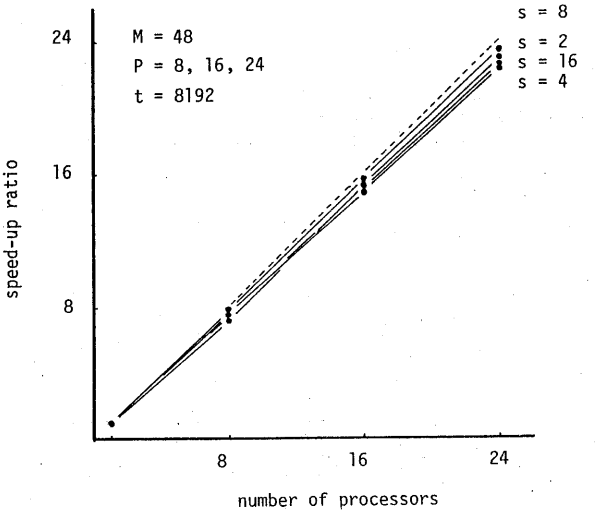

図8　ローカルメモリ型並列パターンマッチングの処理速度向上比

Fig. 8　Speed-up Ratio of Parallel Pattern Matching by Local Memory Scheme

一般に、パイプラインアーキテクチャでは、各ステージの処理時間が等しいとき最大のスループットが得られる。パイプライン型の並列ストリングマッチングアルゴリズムでは、各プロセッサで処理時間が異なる場合がある。この影響を調べるために、異なったパターン長のストリングマッチング処理の実行時間について測定した。図９に、プロセッサ数８台で、２種類のパターン長（ｓａ、ｓｂ）を有する異なる８個（Ｍ＝ｍａ＋ｍｂ）のストリングパターンマッチングを行った処理時間を示す。図の横軸の数字は、パターン長ｓｂを持ったパターンの数ｍｂを示している。全体の処理時間はパターン長ｓｂのマッチングを行うプロセッサの処理時間に支配されていることがわかる。図１０に、一台のプロセッサで実行した場合に対する処理速度比を示す。処理時間は、ｍｂ=0　以外ではほとんど変化しないが、処理速度向上比に対しては、ｍｂの影響を受けることがわかる。この様に、パイプライン法では、各プロセッサでの処理時間がほぼ等しい時に高い性能が得られるため、各プロセッサにマッチング処理に対する負荷を一様に配分しなければならない。

５．まとめ

近年のＶＬＳＩ技術の進歩により、高性能なマイクロプロセッサを多数用いたマルチマイクロプロセッサシステムが盛んに研究され、並列処理による高速化が注目されている。データ量が益々膨大なものになっている文書編集、データベースや記号処理においてパターンマッチング処理の高速化が望まれている。我々は、マルチプロセッサ上で膨大の文字列データの中から与えられた複数の文字列を探索する並列パターンマッチングアルゴリズムを提案した。本論文では、32台のマイクロプロセッサからなるクラスタ型マルチプロセッサシステムＭＵＧＥＮ上に、並列パターンマッチングアルゴリズムをインプリメントし、処理効率を測定した。その結果，提案したローカルメモリ法は、共有メモリ型マルチプロセッサシステムに有効な並列パターンマッチングアルゴリズムで、プロセッサ台数にほぼ比例した処理速度向上比が得られることがわかった。また、para-Cでインプリメントされている隣接間通信機能を用いて、複数のパターンをマチッングするパイプライン法も各プロセッサに均等に処理を割り当てれば、比較的高い処理速度高上比効率が得られていることが実験で確かめられた。現在、複数のパターンを同時にマッチングを行なうAhoとCorasickのパターンマッチングアルゴリズムのインプリメントを行っている。

謝辞　有益な御討論を頂いた九州大学工学部情報工学科の上林弥彦教授に深謝致します。

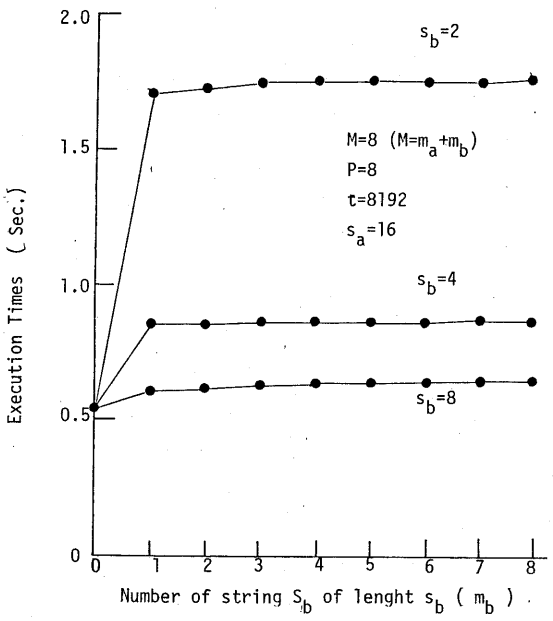

図９　パイプライン型並列パターンマッチングの処理実行時間（異なるパターン長）

Fig.9　Execution time of Parallel Pattern Maching by Pipelining Scheme in tha case of differnt perrern lengths.

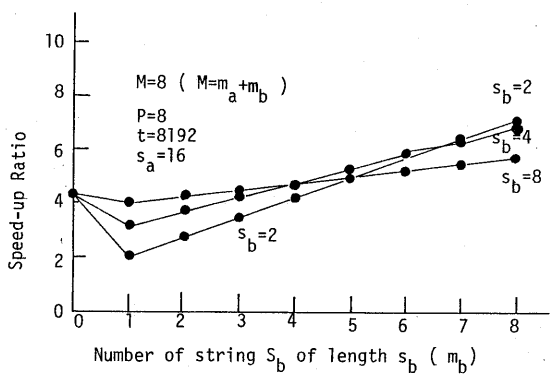

図１０　パイプライン型並列パターンマッチングの処理速度向上比（異なるパターン長）

Fig.10　Speed-up Ratio of Parallel Pattern Maching by Pipelining Scheme in case of different pattern lengths.

参考文献

[1] D.E.Knuth, J.H.Morris and V.R.Pratt,"Fast Pattern Matching Algorithm for Strings," Tech. Rep. CS 440, Comput. Sci. Dept. Stanford Univ. (1974).

[2] D.E.Knuth, J.H.Morris and V.R.Pratt,"Fast Pattern Matching Algorithm for Strings," SIAMJ. Comput. Vol.6, No.2, pp.323-350 (1977).

[3] R.S.Boyer and J.S. Moore,"A Fast String Searching Algorithm,", Comm. ACM, Vol.20, No.10, pp.762-772 (1977).

[4] A.V. Aho and M.J.Crasick,"Efficient String Matching: An Aid to Bibliographic Search", Comm. ACM, Vol.18, No.6, pp.333-340 (1975).

[5] 有川節夫, 篠原武：文字列パターン照合アルゴリズム, コンピュータソフトウェア, Vol.4,No.2,pp.98-119 (Apr. 1988).

[6] G.H.Barnes, R.M.Brown, M.Kato, D.J.Kuck, D.J.Slotnick and R.A.Stockes: "The ILLIAC-IV computer", IEEE Trans. Comput., C-17, 8, pp.746-757 (1968).

[7] R.W.Hockney and C.R.Jesshohe: "Parallel Computers", Adam Hilger Ltd., Bristol(1981)

[8] A.K.Jones and E.F.Gehringer: "The Cm* Multiprocessor Project: Research Review", Dept. of Computer Science, Carnegie-Mellon Univ., CMU-CS-80-131 (1980).

[9] 小原："階層構造のＭＩＭＤ型スーパコンピュータ", 情報処理, 25, 5, pp.480-490 (昭59-05).

[10] 白河, 影山, 阿部, 星野："並列計算機ＰＡＸ-128", 信学論(D), J67-D, 8, pp.853-860 (昭59-08).

[11] P.M.Nealsen nad J. Staunstrup,"Experiments with A Fast String Searching Algorithm," Information Processing Letter, Vol.18, pp.129-135 (1984).

[13] 堀口,小川,木村:"マルチプロセッサにおける並列パターンマッチング",電情通学会, CPSY88-42, pp.25-30, (Aug. 1988).

[14] 中田,堀口,高木,川添,重井:"クラスタ方式マルチプロセッサシステム",電子通信学会論文誌, vol.J70-D, No.8, pp.1469-1477 (Aug. 1987)

[15] K.Tsukamoto and Y. Kawazoe,"Buddhist Sanskrit Studies by Computer; Tohoku University Project," Reports of Faculty of Arts and Letters, Tohoku Univ. vol 37,pp.1-36 (Mar. 1988)