計算機アーキテクチャ　71－2
（1988. 7. 7）

リバモアループによるクラスタ型マルチプロセッサシステムの性能評価

Performance Evaluation of the Clustered Multiprocessor System by Livermore Loops

堀口　進　Susumu HORIGUCHI　東北大学情報工学科　Dept. Information Eng. Tohoku University
高木康志　Yasushi TAKAKI　富士通株式会社　Computers Gourp Fujitu Limited
川添良幸　Yoshiyuki KAWAZOE　東北大学情報処理教育センター　ECIP Tohoku University
重井芳治　Yoshiharu SHIGEI　東洋大学情報工学科　Dept. Information Eng. Toyo University

Abstract:

According to bus-confliction, system throughput of multiprocessor system with bus connection is significantly reduced. To improve this situation, clustering of processors is known to be effective. In this paper, we show a new system software implemented on the clustered multiprocessor system MUGEN. Thesystem software contains monitors, para-C and a para-C pre-processor (P3C) which translates standard C program into parallel program written in para-C. The system performance is experimentally evaluated by using the Livermore loops.

あらまし　バス結合型のマルチプロセッサシステムでは、バスの競合によりシステムのスループットが低下する。このバス競合を緩和する方法のひとつとして、プロセッサを複数のクラスタに分けるクラスタ方式が提案されている。本稿では、このクラスタ方式を採用した試作マルチプロセッサシステムＭＵＧＥＮ上にインプリメントしたシステムソフトウェアの構成および並列計算機の性能に関して検討している。システムソフトウェアに関しては、今回設計・開発した並列処理を陽に記述できるＣ言語ｐａｒａ－Ｃおよび通常のＣ言語プログラムをｐａｒａ－Ｃプログラムに変換するためのプリプロセッサＰ３Ｃについて述べる。また、試作システムＭＵＧＥＮのリバモアループによる性能評価について議論する。

１．はじめに

電子計算機は今日まで、構成素子の性能向上とアーキテクチャの改良に伴なって急速な進歩・発展を続けてきた。特に、近年ではパイプライン、ベクトルプロセッサ等の高速演算装置、あるいはプロセッサと主記憶の速度差を吸収するキャッシュメモリ等の、様々なアーキテクチャが研究され、スーパコンピュータに代表される高速計算機として実用化されている。しかし、これらの商用機は全て実行プログラムを主記憶中に置き、単一のプログラムカウンタにより処理のシーケンスを制御するという従来のノイマン型コンピュータであり、この形態を続ける限り、今後処理の飛躍的な高速化は望めなくなっている。一方、流体、気象、シミュレーション等、大量のデータの高速処理を必要とする分野では、さらにより一層高速な汎用計算機が望まれている。

高速計算処理の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計算機が近年注目され、種々のアーキテクチャが提案・試作されている[1][2][3]。また、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*[4]、Ｃｅｄｅｒ[5]、ＰＡＸ[6]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。

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[4][5][7]。複線バス方式は、全てのプロセッサ間通信が同様に行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[8]。

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を設計構築した[7]。更に、ＭＵＧＥＮ上で効率のよい処理を行なうための並列処理記述言語ならびにシステムソフトウェアの開発を行なった。本論文では、このクラスタ方式バス結合マルチプロセッサで動作する並列処理記述言語ならびに各システムソフトウェアについて述べる。これらを用いて、ベクトルプロセッサの性能比較用として良く知られているリバモアループによるＭＵＧＥＮの性能評価を行なった。さらに、ＭＵＧＥＮ上でＳＩＭＤ型とＭＩＭＤ型の処理方式を実行し、システムの効率を比較・検討している。

２．クラスタ型マルチプロセッサシステム

バス結合マルチプロセッサでは、プロセッサ数が増加するにつれてバス競合の生じる確率が高くなり、システムのスループットが低下する。相互結合網のコストとマルチプロセッサのスループットの比を結合の価格性能比と定義すると[8]、単一バスは非常によい価格性能比を得ることができる。それに対して、クロスバは価格性能比の点で問題がある。しかし単一バスは、プロセッサ数が大きくなるに従って、メモリのアクセス競合によりスループット自体が飽和するため、大規模なシステム構築には向かない。これに対してクロスバはメモリアクセスが多くなっても性能が比較的低下せず、スループットが増加する。従って、単一バスとクロスバの中間的な特性を持ち、メモリアクセス競合が小さいシステムを構築することが望まれる。

我々は種々の理論的検討の結果、4クラスタからなる32台のスレーブプロセッサ（ＳＰ）を持つクラスタ方式バス結合マルチプロセッサの試作システム機ＭＵＧＥＮ（MUltiprocessinG system with pErfect con-nection Network between clusters）を設計し構築した[7]。

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8つのＳＰで構成されている。ＭＰがＳＰを容易に管理するため各ＳＰに固有の番号が与えられている。クラスタ0のＳＰ0をスレーブナンバー0とし、以下、クラスタ3のＳＰ7まで各々に0から31までの番号を割り当てる。

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。システムのハードウェア仕様を表１に示す。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＋ＳＰ3

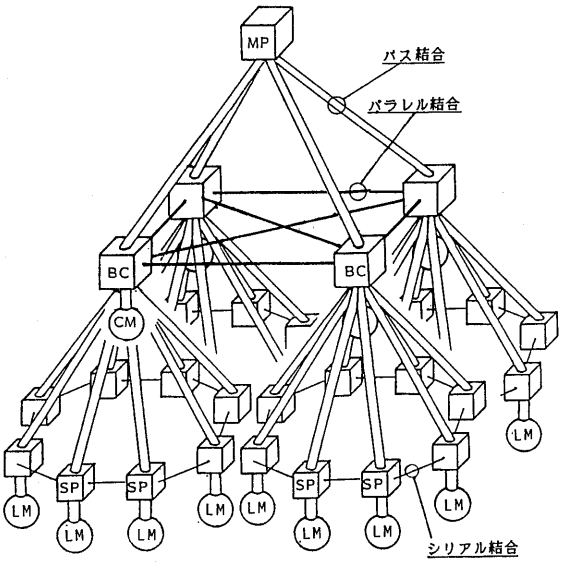

図１　システム構成概念図

MP (MASTER PROCESSOR)	Number	1 Unit
	CPU	Z80A-CPU
	ROM	8 KB
	RAM	56 KB
	PIO	Z80A-PIO×6 (12 Ports)
	SIO	Z80A-SIO (2 Ports)
	CTC	Z80A-CTC
BC (BUS CONTROLLER)	Number	4 Units
	CPU	Z80A-CPU
	ROM	8 KB
	RAM	8 KB
	PIO	Z80A-PIO×8 (16 Ports)
CM (CLUSTER COMMON MEMORY)	Number	4 Units
	RAM	32 KB
SP (SLAVE PROCESSOR)	Number	32 Units
	CPU	Z80A-CPU
	RAM	32 KB
	PIO	Z80A-PIO (2 Ports)
	SIO	Z80A-SIO (2 Ports)
	CTC	Z80A-CTC

表１　ハードウェア仕様

2×32＋ＣＭ32×4の1242KBで計1282KBから構成されている。

システムは直接の入出力装置を持たないため、ＭＰを RS232C のシリアル回線（9600bps）によりホストコンピュータと接続し、システムの操作およびプログラムの開発などはホストコンピュータ上で行う。

３．システムソフトウェア

３．１　基本構成

ＭＵＧＥＮのソフトウェア環境は、大きく分けてモニタと並列処理言語から構成される。基本システムソフトウェアの構成概念を図２に示す。計算機では、システムの立ち上げ時から動作を制御するシステムモニタが必要となる。ＭＵＧＥＮでは、モニタはシステムの各プロセッサのメモリ上に常駐し、システムの動作の制御を行ない、またプログラムの実行やデバッグなどの際にはシステムの運用・操作が効率的に行えるよう便宜を図っている。また実際のプログラムを作成する時のために、各種の基本的な操作用ルーチンをシステムコールとして用意している。

計算機システム上で実際のプログラム開発を効率よく行い、システムの機能を十分に発揮できるようにするためには、高位言語が利用できることが必要不可欠である。ＭＵＧＥＮでは、ホストコンピュータの CP/M-80 上で動作するＣ言語のコンパイラによって、本システム上で実行可能なオブジェクトを生成し、利用することができる。

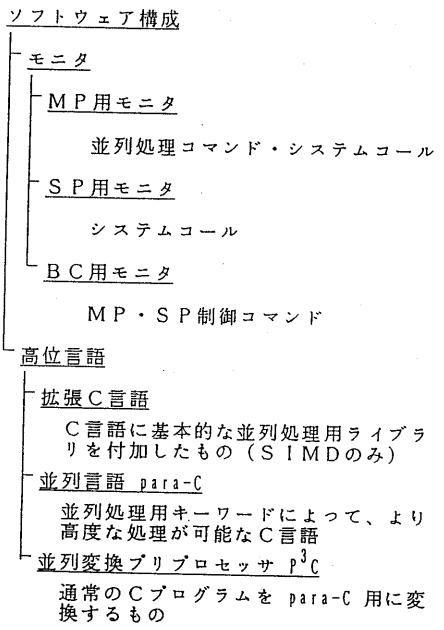

図２　ソフトウェア構成図

３．２　モニタプログラムの概要

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。図３にモニタの基本構成図を示す。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8KBのＲＯＭの中に書き込まれており、電源投入時からプログラムが走り出す。現在のバージョン（V.4.0）には、本システムで走らせる並列処理プログラムを実現するために必要な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がある。

ＳＰはメモリにＲＯＭを持たず、全てＲＡＭで構成されているため、電源投入後ＭＰでＬＭコマンドを実行してＳＰにモニタプログラムをロードする必要がある。ＭＰ用のモニタのＬＭコマンドは全てのＳＰのＬＭにモニタプログラムをロードした後、個々のＳＰに順にプロセッサ番号を埋め込む操作を行う。

３．３　並列処理言語

実際の並列処理の応用プログラム開発を行う上で、アセンブラで種々の数値計算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高位言語のコンパイラを開発した。高位言語には、以下の理由でＣ言語を採用した。

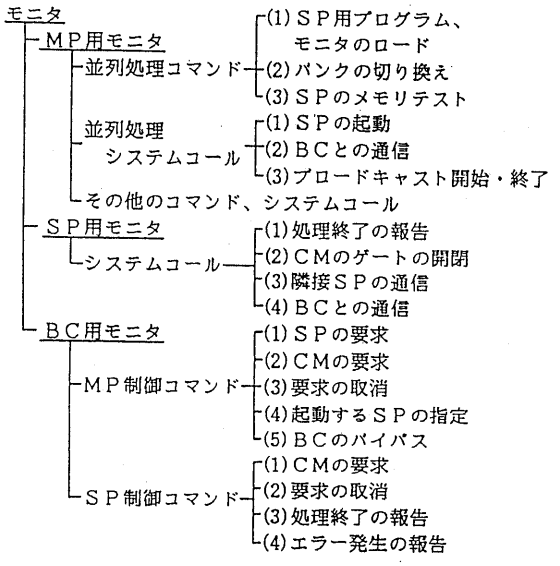

図３　モニタ基本構成図

(1) アセンブラなみの細かいシステム記述ができる.
(2) アセンブラやC自身で記述したプログラムを新しい関数や手続きとしてライブラリに加えることが容易である。
(3) ライブラリのソースが公開されているため既存の関数を本システム上で実行可能にすることが容易に行える。

具体的には、ＳＩＭＤ(Single Instruction stream / Multiple Data stream)型処理が可能な並列Ｃ言語『ext-C』と、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。前者は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だけの機能であるが、後者は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。

３．３．１　ｅｘｔ－Ｃ言語

まず、ＭＵＧＥＮシステム上でＳＩＭＤ型の並列処理を実行するために、ホストコンピュータの CP/M-80 上で動作するＣ言語のコンパイラに、ＳＩＭＤ並列処理用のライブラリを付加し、並列処理を記述できるようにした。これによって、ＭＰ用、ＳＰ用の各プログラムを記述し、システム上で実行できる。

本システムには、ディスク等の二次記憶装置を接続していないため、Ｃ言語で通常使用できる機能の内のファイルアクセス関係の機能はサポートしていない。しかし、UNIX 上で走るＣ言語の関数や手続きの中でシステムに依存したもの以外は、ほとんど使用できる。

３．３．２　ｐａｒａ－Ｃ言語

ＳＩＭＤ型処理のみに用いられるext-C言語でＭＵＧＥＮシステム用の応用プログラムを作成する場合、ＭＰや各ＳＰ用のソースプログラムを別々に記述し、各々をホストコンピュータ上で別々にコンパイルして、各プロセッサへの転送および実行を行なう必要がある。この方法は、ＭＩＭＤ型のプログラムを記述する場合は、各プロセッサ毎にプログラムを開発するなどの問題が起こる。また、アセンブラなどで記述するライブラリのみで、大規模なＭＩＭＤプログラムをサポートすることは困難である。そこで、ひとつのソースプログラムの中で並列処理を記述できる、より高度な並列Ｃ言語 para-C の設計・開発を行なった。

並列処理プログラムの記述を行なう場合、並列性を陽に示すかどうかという問題がある。並列性を陽に示さない場合には、コンパイラによって並列性を検出する必要がある。また、マルチプロセッサにおいては、一般に相互結合のバンド幅が小さく、並列処理の単位が小さいと通信によるオーバヘッドによって実効的なスループットが上らないため、処理単位をある程度大きくする必要がある。

これらの点から、 para-C ではプログラマが並列処理を陽に記述する方法をとる。これを実現するために、通常の UNIX のＣ言語から構造体、共用体等を除いたサブセット版のＣ言語に、キーワードcobegin、 process、body 及び forall を追加している。para-C の文法仕様を表２に、構文ストラクチャを図４に示す。また実際のプログラム例を図５に示す。para-C において、並列処理の起動は、Ｃのメインプログラムにあたる関数 main によってのみ行なうことができる。cobegin 文と、それに続く process 文によって並列処理が起動される。process 文では、起動する並列プロセスの id、ＳＰの id、プロセス名および引数を記述する。並列処理される関数は各ＳＰにプロセスとしてあらかじめ静的に割り付けられている。並列処理は各ＳＰ上で逐次的に実行される。また、body 文はその関数が並列に処理されるプロセスであることを示す。 forall 文は、process 文を繰り返し作用させる制御構造である。

追加されたキーワード	
cobegin	並列処理の起動を宣言する
process	スレーブプロセッサに関数を割り付け、並列プロセスを起動する
body	関数がスレーブプロセッサ上の関数であることを宣言する
forall	並列プロセスを繰り返し起動する
制限事項	
シンボルは最大50個まで ローカル変数は最大128バイトまで 構造体、共用体は使用できない 倍精度変数は単精度に変換される case, break, continue 文は使用できない static 変数は使用できない	

表２　para-Cの文法仕様

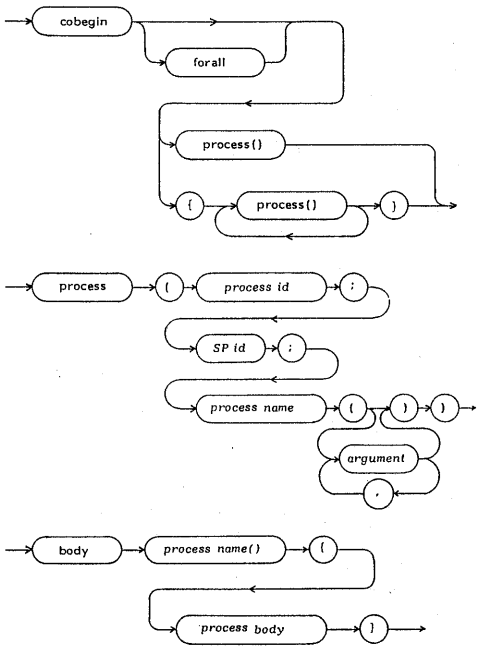

図４　para-C の構文ストラクチャ

3．3．3　プロセス間通信

para-C において、ＳＰへの引数データの転送は、 process 文によって、Ｃの関数呼び出しと同様なスタックを用いる形式で行なわれる。また、ＭＰへの結果の転送も関数によって行なわれる。通常Ｃ言語では配列の値を参照する場合、ポインタのみを引数として渡す。しかしＭＵＧＥＮの場合、この方法を可能にするためには、配列のアドレスをＣＭ上に取る必要がある。これでは配列の大きさも制限され、また処理のオーバヘッドも大きくなる。このため、配列の値そのものを転送する必要がある。これを可能にするため、特別の関数と、通常のＣの制御文字列を拡張した書式指定を用いる。また、隣接するＳＰ間を接続するシリアルポートの利用も、関数によっては行なわれる。表３に以上の配列転送ならびに隣接ＳＰ間の通信のための関数を示す。これらの関数を用いることによって、配列データの転送を簡単に記述でき、バス競合を避けて隣接するＳＰと通信することができる。

3．4　並列変換プリプロセッサ P3C

para-C では、並列処理をプログラム上に陽に記述する方法を取る。この方法によって、並列処理の単位をプログラマが任意に決定できるので、実効的にスループットを上げることができる。反面、並列処理を行なうためには、プログラマが並列処理を直接記述する必要があり、通常のＣ言語で記述されている既存のプログラムを para-C によって並列に実行することはできい。

この問題を解決するために、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。for ループを並列化する処理のフローチャートを図６に示す。具体的には、以下の手法を用いて並列処理への展開を行なう[9]。

(1) インデックス変数についての並列化

各 for ループの内部における変数の依存関係を調べ、各インデックス変数の値に対してループが独立に実行できる場合は、ループをインデックス値に対して分割し、各ＳＰ上で並列に実行するように変換する。変換されたルーチンはＳＩＭＤ形式で実行できる。

(2) ループの分割・統合

ループ内部の変数に依存関係が存在し、(1) の処理ができない場合、ループの内部を分割して複数のループに分け、(1) の処理が可能なループを抽出する。また、並列処理の単位を大きくするため、統合可能なループを結びつける。

(3) ループ間の独立性による並列化

最終的にインデックス値による並列化が不可能で、かつループ内の変数が他のループと独立なループは、ループ全体を1つのＳＰに割り当て、並列に実行する。変換した部分はＭＩＭＤ形式で実行される。ループの独立性を高めるために、変数名の変更も行なう。

Ｃ言語で記述された応用プログラムは、図７の手順で処理される。通常のＣ言語で記述されたソースプログラムは、ホストコンピュータ上でプリプロセッサ P3C によって para-C 用プログラムに変換され、次に para-C コンパイラによって、ＭＵＧＥＮシステム中の構成要素プロセッサである Z80のアセンブリ言語に変換される。この時、１つのソースプログラムから、ＭＰ用とＳＰ用の２つのアセンブリ

```
main()
{
        int i;
        long n, result, answer, start[32], end[32];

        scanf("%ld", &n);
        for(i = 0; i < 32; ++i)
                start[i] = i*n/32 + 1;
                end[i] = (i+1)*n/32;
        cobegin                          データの転送、ＳＰの起動
                forall(i = 0; i < 32; ++i);
                        process(1; i; sum(start[i], end[i]));
        for(answer = 0, i = 0; i < 32; ++i) {
                hrecieve(i, "%ld", &result);
                answer += result;
        }                                        データの受け取り
}
body sum(lower, upper)
long lower, upper;
{
        int i;
        long total;

        for(total = 0, i = lower; i <= upper; ++i)
                total += i;
        hsend("%ld", total);                          結果の転送
}
```

図５　para-C のプログラム例

brdcst("string", addr1, addr2, ...); ＭＰの各配列のアドレスから、制御文字列に従った要素数を全ＳＰに転送する。
getarry("string", addr1, addr2, ...); ＭＰからブロードキャストされた値を、ＳＰの各配列にロードする。
hsend("string", var1, var2, ...); ＳＰの変数および配列の値を、制御文字列に従ってＣＭ上に転送する。
hreceive(SPid,"string", addr1, addr2,...); id で指定したＳＰについて、 hsend で転送されたＣＭ上の値を、ＭＰの変数および配列にロードする。
sendh("string", var1, var2, ...), sendl("string", var1, var2, ...) 隣接するＳＰにデータを転送する。sendh はＳＰの id が（自分の id + 1）のＳＰへ、sendl は（自分の id - 1）のＳＰへ転送する。
rcvh("string", addr1, addr2, ...), rcvl("string", addr1, addr2, ...), 隣接するＳＰからデータを受け取る。 rcvh はＳＰのid が（自分の id + 1）のＳＰから、rcvl は（自分のid - 1）のＳＰから受け取る。

表３　para-Cの通信用関数

プログラムが出力される。このそれぞれをアセンブラによって機械語プログラムに変換し、ＭＵＧＥＮ上に転送して実行する。para-C の文法上の制限により、プリプロセッサで処理できるＣ言語のプログラムには、幾つかの制限が設けられる。すなわち、構造体や共用体は使用できず、浮動小数点演算では、double による倍精度の演算は、単精度の float(32bits) に変換される。

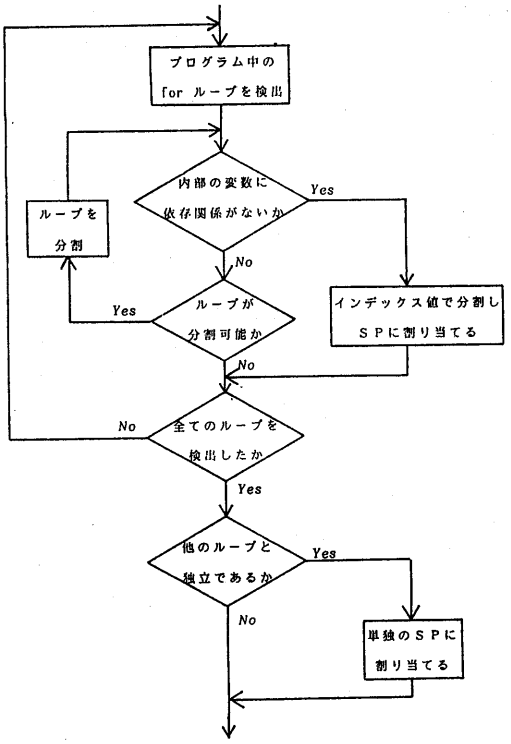

図６　P3C による for ループの並列化フローチャート

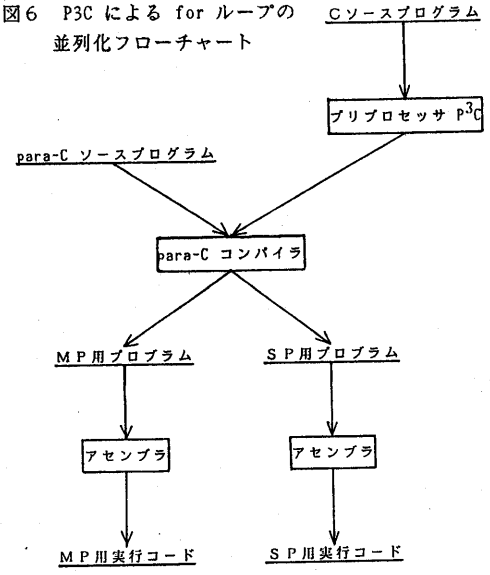

図７　アプリケーションプログラムの処理手順

４．システムの性能評価

４．１　リバモアループによる性能評価

Ｃ言語で記述したプログラムを実行する際のシステムのスループットを測定するために、リバモアループによるベンチマークテストを行なった[10]。リバモアループは、約20年前に科学計算用コンピュータのFORTRAN 処理能力を測定するためにアメリカ合衆国のリバモア研究所で作られたベンチマーク用プログラムである。基本プログラムは、流体、内積などの一般的な大規模計算の中核コードを集めた14個の DO ループで構成され、今日では計算機の処理能力を測定する一般的な検査プログラムとして、広く利用されている。

このリバモアループをＣ言語で記述し、ＳＩＭＤ型の処理が可能である拡張Ｃ言語のコンパイラを用いてコンパイルした場合のＭＵＧＥＮシステム上での処理時間を測定した。測定結果を図８に示す。縦軸の値は、ＭＰ1台で計算した場合の実行時間に対する速度比を表している。また、処理時間にはデータ転送時間も含んでいる。図９に、ループ1の場合のデータ転送時間の処理時間に占める割合を示す。5，6，11の各ループは、アルゴリズム上並列化が難しく、並列処理による速度向上は見られなかった。

並列化による速度向上比は、2から17倍と大きなばらつきが見られる。これは、演算のアルゴリズムによって並列化できる度合が違うことと、ＳＰに転送するデータ量が大きく違う事による。また同一のループではＳＰの数を、8，16，32台と増加させるにつれて、速度比は増加していくが、増分は少なくなって行く。問題の大きさに依存するので一般論を述べるのは難しいが、ここで扱った規模のループに対しては、数十台のプロセッサからなるＭＵＧＥＮ程度の並列処理システムは処理速度向上に有効であると言える。

また、各ループをext-C言語を用いてコンパイルした場合の処理速度比（ＳＰの台数は32台）とスーパコンピュータ SX-1 のベクトルプロセッサを用いた場合の速度向上比を図10に比較した。ループのアルゴリズムの違いによって、

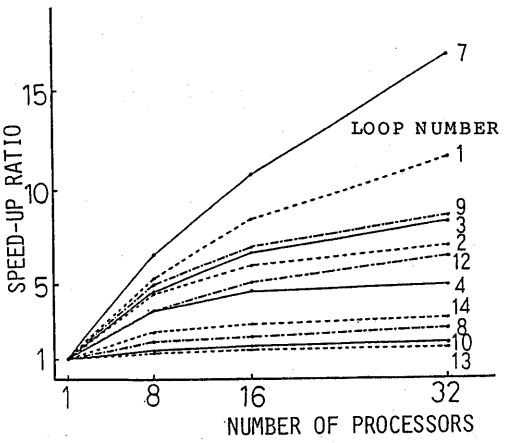

図８　リバモアループに対する処理速度比

速度向上の度合がベクトルプロセッサと並列計算機とで異なるが、処理速度が全く向上しないループは共通である。ループ１３および１４に対してＳＸ－１より本システムが相対的に良い結果を出しているが、両者とも非常に速度向上比が悪い。ＳＰの台数を増やした場合、ループ２および７で本システムは秀れた処理速度向上比を実現できる。このように、並列計算機とベクトル計算機では、それぞれ処理に　適する問題が異なっている。一般に、異なる型の演算処理が混在している場合には、並列計算機が優位であり、この意味で汎用性が高いと言える。

次に、通常のＣ言語で記述したプログラムから P3C によってループのインデックス値による並列性を検出し、 para-C を用いてコンパイルした場合の処理速度比と、ＭＰ，ＳＰ用の各プログラムをext-C言語を用いてコンパイルした場合の処理速度比を図11に示した。ループ１の場合についてみると、P3C と para-C を用いた場合の処理速度比は、ext-C言語を用いた場合の約1/2の値を示す。これは、実際の処理がＳＩＭＤで記述できるにもかかわらず、 para-C でコンパイルされたオブジェクトは、ＭＩＭＤ対応の汎用性の高い処理を行ない、このためオーバヘッドが大きくなるからである。ループ1の場合の処理とオーバヘッドの時間を図12に示す。

４．２　ＳＩＭＤ型とＭＩＭＤ型処理の比較

次に、para-C のＭＩＭＤ処理における性能を測定するために、リバモアループ14個全てを単一のプログラムにまとめて記述し、para-C によって各ループをそれぞれ別のＳＰに割り付けてＭＩＭＤ形式で処理した場合と、４．１．１と同様にext-C言語を用いてＳＩＭＤ形式で処理した場合とを比較した。各ループの長さを変化させた場合の両者の処理時間を図13に、またＭＰ1台で実行した場合に対する両者の処理速度比を図14に示す（ＳＰの台数は16台）。para-C を用いて処理を行なった場合、最大でおよそ7の処理速度比となり、ext-C言語を用いた場合の約2.5倍の速度向上が得られる。このように、プログラムがある程度の長さを持ち、独立したループをＭＩＭＤ形式で処理する場合、para-C によってかなり良い速度向上比が得られることが分った。

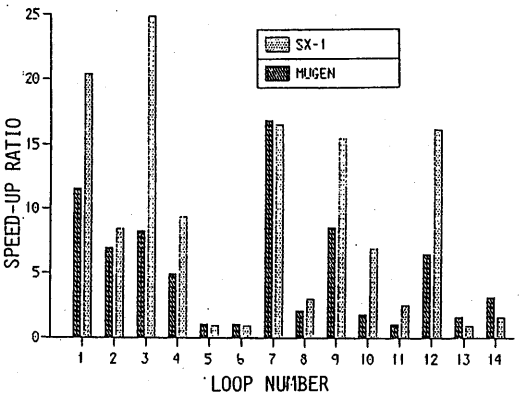

図10　ベクトルプロセッサとの比較

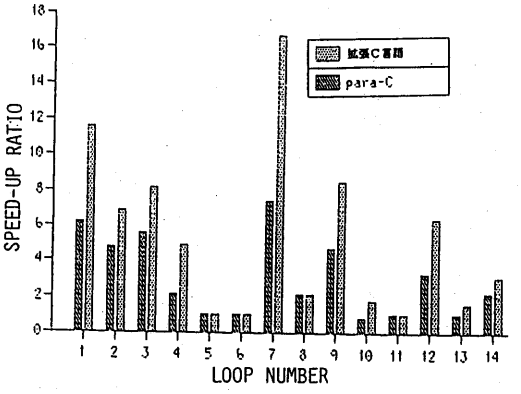

図11　拡張Ｃ言語と para-C との比較

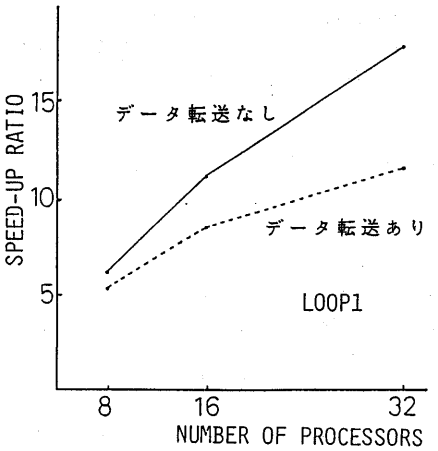

図９　データ転送の影響

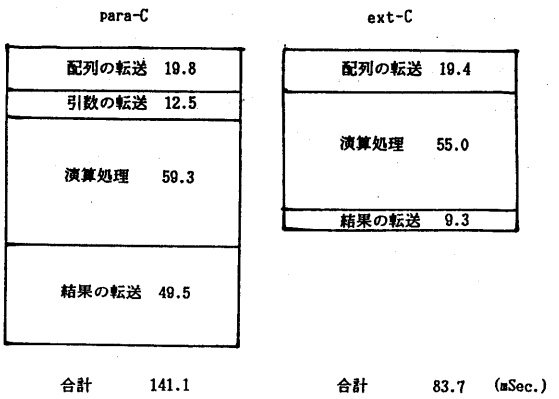

図12　para-C のオーバヘッド

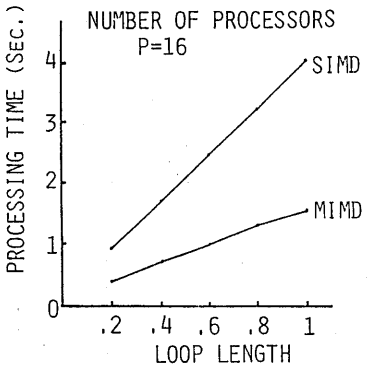

図13　ＭＩＭＤおよびＳＩＭＤ形式での処理時間

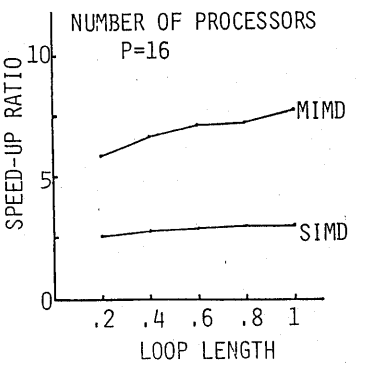

図14　ＭＩＭＤおよびＳＩＭＤ形式での処理速度向上比

５．結論

本論文では、クラスタ方式マルチプロセッサＭＵＧＥＮ上に設計・構築したシステムモニタ、ext-C言語、並列処理言語 para-C 、ならびに従来のＣ言語で記述されたプログラムを para-C に変換するプリプロセッサ P3C などのシステムソフトウェアについて述べた。さらに、ベンチマークテストとして、スーパーコンピュータの性能評価用として良く用いられているリバモアループをインプリメントし、設計構築したソフトウェアによるシステムの性能を評価した。その結果、ループごとに相当のばらつきはあるものの、一般的に相当程度処理速度の向上が得られた。また、para-C を用いてＭＩＭＤ型の処理を行なうことにより、より高速に実行できることが明らかになった。これは多数のタスクが同時実行される汎用計算機の環境に近い状態で、ＭＵＧＥＮシステムがＭＩＭＤ型の処理を実現することにより、高いスループットを達成できることを意味している。

現在、para-C でインプリメントされている通信機能を用いて、並列ＦＦＴなどの応用プログラムをシステム上で実行し、その速度比を測定している。プロセッサ間のリンクによる通信機能はバス競合の回避に有効であり、比較的高い処理効率が得られることが、実験的に確かめられている。メッセージ通信とバス結合共有メモリ型を組合せた高速並列処理システムの研究・開発が、今後に残された問題である。

参考文献

[1] 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).

[2] R.W.Hockney and C.R.Jesshohe: "Parallel Computers", Adam Hilger Ltd., Bristol(1981); 奥川, 黒住訳: "並列計算機", 共立出版 (昭59).

[3] 高橋: "並列処理のためのプロセッサ結合方式", 情報処理, 23, 3, pp.201-209 (昭57-03).

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

[5] 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).

[6] 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).

[7] 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)

[8] C.R.Das and L.N.Bhuyan: "Bandwidth Availability of Multiple-Bus Multiprocessors", IEEE Trans. Comput., C-34, 10, pp.918-926 (1985).

[9] 村岡: "並列処理", 昭晃堂 (昭61).

[10] 唐木: "Formula Translator の動向", Computer Today, 7, 2, pp35-42 (1984).