並列計算機システムに関する研究ーブロードキャストメモリ結合による多重プロセッサシステムについてー

小畑，正貴

(Degree)
博士（学術）

(Date of Degree)
1985-03-31

(Date of Publication)
2014-02-28

(Resource Type)
doctoral thesis

(Report Number)
甲0523

(URL)
https://hdl.handle.net/20.500.14094/D1000523

博　士　論　文

並列計算機システムに関する研究

—ブロードキャストメモリ結合による多重プロセッサシステムについて—

昭和59年12月

神戸大学大学院自然科学研究科

小　畑　正　貴

目次

第 1 章

緒論

近年における半導体技術・集積回路技術の発達により、電子計算機の性能は飛躍的に向上している。しかしながら、計算機の応用分野の拡大と、対象となる問題の大規模化は、さらに高性能の計算機システムを要求している。問題を数値計算にしぼって考えてみても、その応用範囲は広い。各種シミュレーション技法、数値解析技法、線形計画法や動的計画法などの最適化手法、その他各種の解析・設計・制御手法にとって数値計算は不可欠であり、問題が複雑になるにつれて、その規模も大きくなっている。

大規模数値計算の高速処理に対して、現在は、パイプライン処理を用いたスーパーコンピュータ[1)]やアレイプロセッサと呼ばれる専用計算機が用いられているが、現在の数値計算専用機は次のような問題点を含んでいる。

(a) 非常に高価である。

(b) 素子の速度には限界があるため、いずれ計算速度は頭打ちになる。

(c) 規則的なベクトル計算は効率よく処理できるが、不規則な疎行列などに対しては、有効性が十分に発揮できない。

これらの問題に対して、並列計算機は次のような点で期待が持たれている。

(a) 並列計算によって、素子のもつ速度以上の高速処理が可能となる。さらに、パイプライン処理などを併用すれば超高速計算機が実現できる。

(b) LSI 技術の進歩により出現したコストパフォーマンスの高いマイクロプロセッサを構成要素とすることにより、安価で高性能なシステムが構成できる。

(c) 多重プロセッサ構成の並列計算機では、各プロセッサが独立して動作できるため、不規則的な処理に対しても、十分な並列性を引き出せる可能性がある。

複数の計算機を使うことによって計算時間を短縮しようとする並列計算機の考え方を始めて実現したのは、1963年にウェスティングハウス社より発表された SOLOMON計算機[2]である。その後、1972年にはイリノイ大学で 64台のプロセッサをアレイ状に並べた ILLIAC-IV[3] が完成しているが、これらの並列計算機は全プロセッサが一斉に同じ動作をする方式であった。これに対し、各プロセッサが独立に動作する並列計算機の考え方は、1959年に Hollandの提案した計算機[4]に見られるが、実際に開発が行われるようになるのは、1970年代になってからである。

最近では、処理の柔軟性の点から、後のタイプ（各プロセッサが独立に動くタイプ）の並列計算機が注目を集め、マイクロコンピュータを用いた多重プロセッサシステムの開発が数多く行われるようになっている[5)~11)]。しかしながら、画像処理などの特定分野における実用例[8)]を除いて、ほとんどが試作・研究の段階である。

多重プロセッサシステムの開発・実用化における課題として

(a) プロセッサ間の結合方法

(b) 並列計算アルゴリズムの開発
(c) 並列動作の記述と実行の方法
があげられる。

まず、プロセッサ間の結合に対しては次の条件を考慮する必要がある。

(1) プロセッサ間通信におけるオーバーヘッドを小さくする。
(2) ハードウェア量（配線量）を小さくする。
(3) 信頼性・故障対策

一般に、(1) (2)を同時に満足させることは困難である。たとえば、隣接プロセッサを接続する方式では、データの移動が局所的におこる問題には効率がよいが、移動距離が大きい場合には中継のオーバーヘッドが生じる。この問題に対処するため、複数の接続法を併用しようとすると、ハードウェア量が増大する。 また、(3)については、一部分の故障によって全体の動作が停止してしまうという事態を避けるようにしなければならない。

また、課題(b)は、現在ある各種計算アルゴリズムのほとんどが、直列的な処理をする計算機上で効率よく動くように設計されているという点に起因する。並列計算機においては、これらのアルゴリズムの中から並列計算が可能な部分を見つけ、一台でも多くのプロセッサを有効に使う工夫が必要となる。

最後に(c)に対しては、 並列計算アルゴリズムを記述するための並列プログラミング言語と、これを並列計算機上で効率よく実行するためのオペレーティングシステムの開発が課題となる。プログラミング言語に関しては、同期・通信などの基本的な機能の記述法や、種々の問題に対する記述のし易さ・正当性などのさまざまな問題が

ある。また、オペレーティングシステムについては、プロセッサへのプロセスのスケジューリング方法や並列プログラムのデバッグの方法など、数多くの問題が残されている。

本論文では、ブロードキャストメモリと呼ぶ特殊なメモリシステムによって結合された多重プロセッサシステムの構成と、これによる行列計算問題(特に連立方程式の求解問題)の並列解法について論じている。同一データを複数のプロセッサに同時に転送するブロードキャスト(放送)機能の考え方は以前からあるが、ほとんどが補助的な使用であった。ブロードキャスト転送を並列計算機の結合方法として本格的に使用した例としては、慶応大学の離散系シミュレータ KDSS-1[11] があげられる程度である。

本研究では、全体で３つのシステムを提案し、ブロードキャストメモリの有効性を明らかにする。まず、最も基本となるシステムについて、試作によってその有効性を確かめる。次に、超多重プロセッサシステムへの対応として、ブロードキャストを 2次元に拡張したマトリクスブロードキャストメモリシステムと、これに、さらにパイプライン処理を組み合わせたＢＣプロセッサアレイを提案し、この上での連立方程式の並列計算法を示す。

緒論に続き、第２章では、並列計算機の開発に必要なハードウエアとソフトウエアの基礎をまとめ、試作機の基本設計を行う。ハードウエアについては種々の結合方式の持つ特徴を比較し、ソフトウエアについては並列動作の制御に必要な基礎概念をまとめる。次に、本研究の中核となるブロードキャストメモリの概念を述べ、これらをもとに、数値計算を主目的とした試作機の設計思想と基本構成を固める。最後に、シミュレーションにより、試作機に用いるメモリ

システムの基本的な動作と性質を明らかにする。

第３章では、試作システムのハードウエア・ソフトウエアの詳細を述べ、試作機上での連立方程式の並列計算方法とその実行結果を示す。ハードウエアでは特に、ブロードキャストメモリを実現するためのバス構造とバスアクセスの方法を中心に詳述する。ソフトウエアに関しては、並列プログラミング言語を設計し、試作機上にその処理系を実現する。特に、言語設計においては、

(a) Ｃ言語と互換性を持たせること

(b) 数値計算問題を対象とすること

を基本方針とする。次に、このシステム上での連立一次方程式計算の実行方法と、その実行結果を示す。計算アルゴリズムとしてはガウス消去法・共役勾配法をとりあげ、ブロードキャストメモリを生かした並列計算アルゴリズムを示す。実行結果については、試作機での実行とともに計算機シミュレーションを行い、台数を増やした場合のふるまいを見る。最後に、これらの結果をもとに、システムの評価を行う。

第４章では、マトリクスブロードキャストメモリ結合システムについて論じる。このシステムはブロードキャストメモリを 2次元格子状に接続したもので、行方向および列方向へのブロードキャスト転送によって計算を進めていく。ここでは、まずシステムの構成と基本動作を述べる。次に、このシステムによって並列計算を行うことにより、ガウス消去法と修正コレスキー法の計算時間を Ｏ(n)とすることが可能なことを示す。また、さらに、これらの計算法を拡張し、消去時の軸選択を導入した並列計算法を提案する。

第５章では、ブロードキャスト機能とパイプライン機能とを合わ

せ持つ演算プロセッサ(BC プロセッサ)で構成される、一次元または二次元のプロセッサアレイ(BC プロセッサアレイ)について論じる。ここでは、まず、BCプロセッサの構造と動作を示す。次に、このプロセッサアレイによって行列とベクトルの乗算、行列積、行列の LU分解が効率よく実現できることを示す。 また、ガウス消去法や修正コレスキー法を用いた連立方程式の計算も効率よく実現できることを示す。

最後に、結論では、本研究で得られた結果をもとに、今後の課題と将来の展望をまとめる。

第２章

ブロードキャストメモリ結合形並列計算機システムの設計

2.1 緒言

並列計算機システムとは、一つの問題を複数の部分に分解し、これらを複数の計算機によって同時に処理するシステムである。この際、この複合システムの結合方法や制御方法には種々の形態があり、対象とする問題の性質や処理効率と深く関係している。

本章では、システム設計に先だって、まず並列計算機のさまざまな結合方法についてまとめる。また、並列動作の制御に必要なソフトウエアの基礎概念を整理する。次に、数値計算を主目的とする試作システムの設計方針と基本構成を決定する。最後に、計算機シミュレーションにより、試作機で採用するメモリシステムの基本動作を確かめる。

2.2 並列計算機システム

2.2.1 並列計算機の分類

並列計算機を分類する場合、命令の流れとデータの流れに着目する Flyn の分類法がよく用いられる。これによると並列処理システムは、

(1) SISD

(2) MISD

(3) SIMD

(4) MIMD

の四つのタイプに分類される。それぞれのタイプにおける命令とデータの流れを図1.1 に示す。 ここで、 SI は Single Instruction stream、MI は Multiple Instruction stream、SD は Single Data stream、MI は Multiple Data stream の意味である。

SISD システムは通常の直列実行型計算機である。

MISD の代表的なものはパイプライン処理である。 これは一連の処理を n 段のステージに分割し、 ステージ間での同時並列動作を得ようとするものである。パイプライン処理の技術はほとんど確立されており、スーパーコンピュータなどの超高速計算機に利用されている。比較的容易に実現できることが利点であるが次のような問題点を含んでいる。

(a) 規則的なデータ列に対しては十分な効果が得られるが、不規則なデータに対しては十分な並列性が得られない。

(b) 問題の分割には限度があるので、並列性も制限を受ける。

SIMD 型では、n 台の同一プロセッサを並列にならべ、n 個のデータに対して1つの命令により同時に同じ処理を行う。プロセッサ数を増すことによって並列度をいくらでも上げることができるという利点がある半面、ハードウエア量が大きくなることや、処理に柔軟性がないことなどの問題点をもつ。実現例としては ILLIAC-IV[3] があげられる。

MIMD型は複数のデータに対して複数のプロセッサがそれぞれ別の

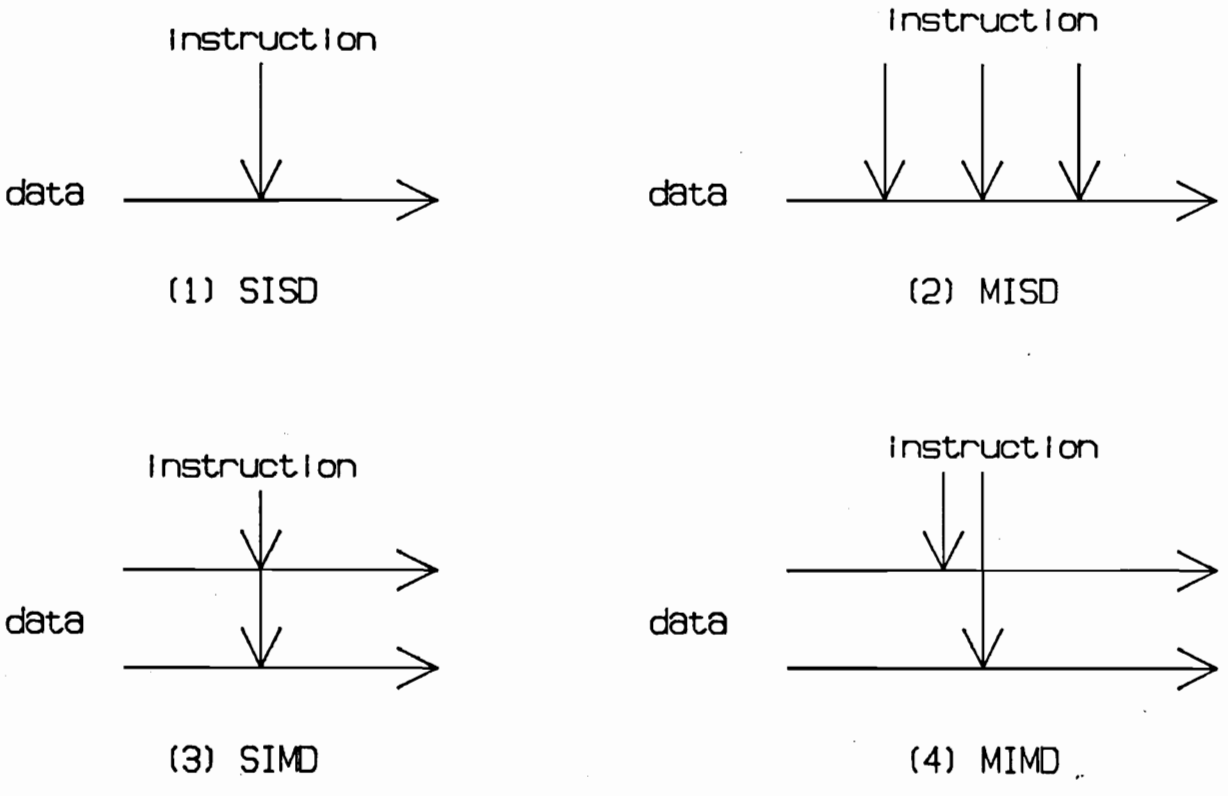


図2.1　並列計算機の分類

処理を並行して行うものである。このタイプの並列計算機は、動作に柔軟性があり、不規則な処理に対しても高並列性を引き出すことのできる可能性があるが、その半面、ハードウエア構成および制御構造が複雑になるという問題点を持っている。しかし集積回路技術の進歩によってハードウエアによる制約が緩和されてきたことから、近年、研究が盛んになっている。各所で試作されている多重マイクロプロセッサシステムやデータフローコンピュータ[27)~29)]などがこのタイプにあたる。

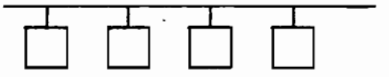
2.2.2 並列計算機の結合形態[12)~16)]

SIMD 型および MIMD 型多重プロセッサシステムで、プロセッサ - プロセッサ間あるいはプロセッサ - メモリ間の結合に用いられるネットワークの諸形態を以下に示す。

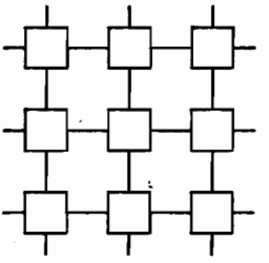
(a)共通バス結合 (図2.2-A)

複数プロセッサを共通バスによって結合する。バス上に共有メモリを置くことが多い。プロセッサ間通信は、この共有メモリを介して行うか、または、直接相手のメモリをアクセスして行う。比較的容易に実現できる結合方法であり、どのプロセッサ間でも等しい条件で通信できることが特徴である。

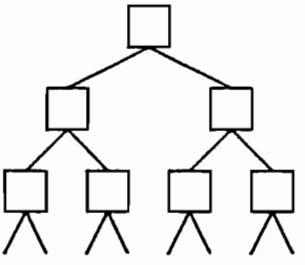
問題点は、プロセッサ数を増加させるとアクセス競合のために効率が落ちるということであり、単純なバス構成では大規模システムに対応できなくなる。複数のバスを用いてこの問題に対処した例として、C_m*[6)] や SMS201[9)] があげられる。



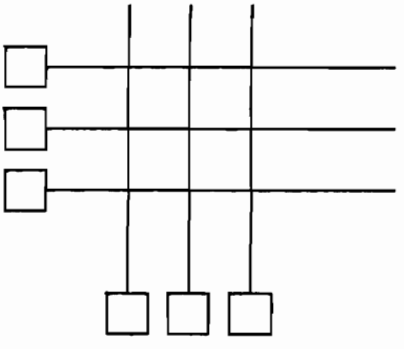
(A)



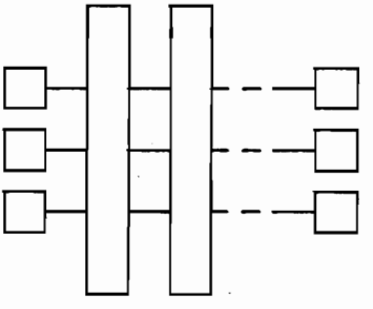
(B)



(C)



(D)



(E)

図2.2　プロセッサ間の結合方式

(b)アレイ結合（図2.2-B)

2 次元の場合、m×n 台のプロセッサをアレイ状に結合する。 各プロセッサは4個の入出力ポートを持ち、 上下左右のプロセッサと接続される。共通バスの場合と違い、プロセッサが離れるほど通信に時間を要するが、近接プロセッサ間での通信は各所で独立に、並行して行うことができる。したがって、データの移動に局所性のある問題に適している。実例として ILLIAC-N、PACS[7] 等があげられる。

(c) 2 進木（図 2.2-C)

2 進木構造では、各プロセッサは3本のポートを持ち、上の1台と下の2台のプロセッサに接続される。(b)と同様にデータの移動には局所性がある。故障によって木が分断されてしまう点が問題点としてあげられる。 CORAL[10] がこの型であるが、(a)(b)ほど実例は多くない。

(d) クロスバスイッチ（図2.2-D)

プロセッサ間の距離がすべて等しく、しかも、複数の通信を並行して行うことができる。しかし、台数 n に対してスイッチ数が n^2 となり、 配線量が多くなる点に問題がある。 実例として、Cmmp[6] があげられる。

(e) 多段ネットワーク（図2.2-E)

複数段のスイッチによるネットワークであり、代表的なものにオメガネットワークやデルタネットワークがある。 性質はクロスバ

スイッチと似ている。 クロスバスイッチに対して、 スイッチ数は $(n \log_2 n)/2$ と少なくなるが、スイッチ中段での衝突のため多重度は小さくなる。また、段数に対応したデータの遅れが生じる。
staran がこの方式をとっている。

以上を簡単にまとめると、データの移動範囲が幅広い場合はバス型がよく、局所的な場合はアレイ型や2進木がよいと言える。また、転送に時間を要してもよい場合には、多段ネットワークが並列性の高い通信方法となる。

2.2.3 ソフトウエア[17)~24)]

MIMD 型並列処理システムにおいて、 共有資源が複数プロセスから参照され更新される場合、各プロセスは互いに連絡をとり、これらが正当に行われるようにしなければならない。このために必要となるソフトウエア機能として、相互排除・同期・通信などがあげられる。

(a) 相互排除 (図2.3-A)

相互排除は、 特定の期間ある1つのプロセスのみがその共有資源のアクセス権を占有し、他のプロセスはその資源をアクセスできないという、排他的な制御である。変数の更新に複数の手順を要する場合に、更新中に他から割り込まれないために必要となる。相互排除の制御方法の1つとして lock - unlock 機構がある。 lock は相互排除領域に入る時に行う操作で、領域に入ったことを知らせるフラグを立てる。もし、他が先に相互排除領域に入っておれば、その

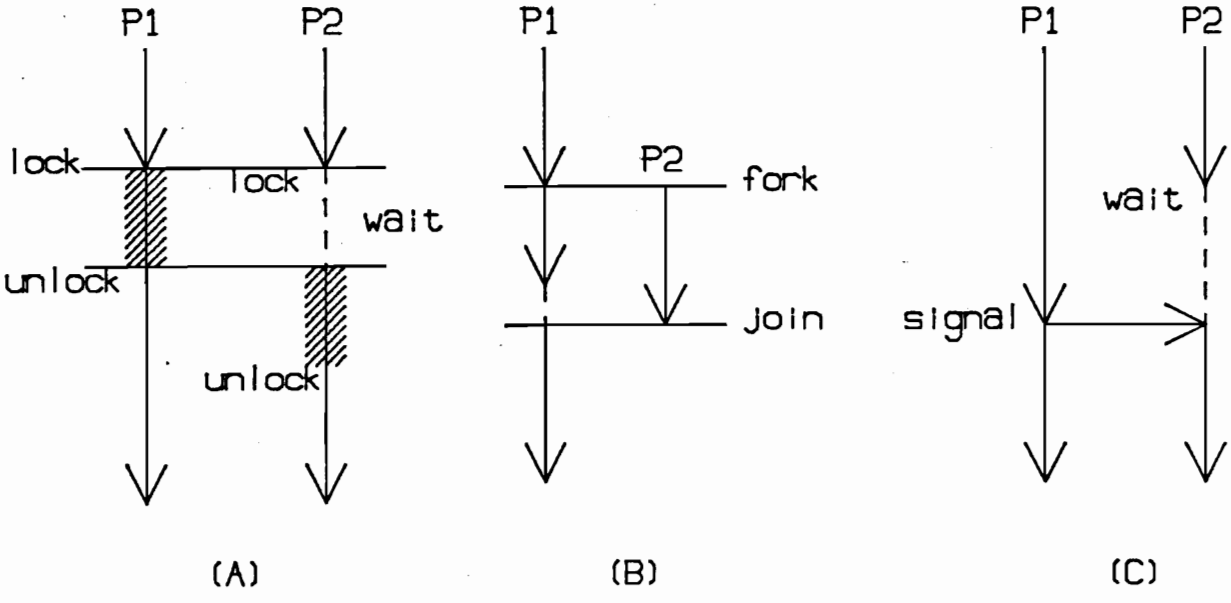


図2.3　並行プロセスの制御

終了を待つ。unlock ではフラグを落とし相互排除領域から出る。

lock - unlock は相互排除を制御する最も基本的な操作であるが、相互排除領域が長く、また、頻繁に起きる場合には、待ちによって効率が悪くなる。この点を解決したものとして Dijkstra によるセマフォと P・V 命令[19]や Hoare によるモニタプロセス[20]などがある。これらは、待ちが生じた場合に他の実行可能プロセスを実行しようとするものである。ここでは詳細をふれないが、実現のためには、マルチプロセスを取り扱うことのできるオペレーティングシステムが必要となる。

(b) 並行プロセスの起動と同期 (図2.3-B)

並列処理といっても、すべてのプロセスがいつも並行に処理されるわけではなく、直列から並列、並列から直列へと実行状態は変化していく。一つの処理の流れを複数に分岐させる場合には並行プロセスの起動が必要であり、複数の流れを一点でまとめる場合には同期という動作が必要となる。 代表的な記述法として fork - join や cobegin - coend などがあり[24]、 fork,cobegin の後に分岐するプロセスを書くことで分岐を示し、join,coend で同期をとる。

(c) 通信 (図2.3-C)

通信機能は、プロセス間で事象の発生を知らせたり、変数の値を送ったりするのに必要となる。事象の発生を伝える手段として、たとえば signal - wait 機能がある。signal は事象の発生を知らせ、wait はこれを待つ。 また、データを送るには send - receive 機能が使われる。 send はデータを送り、receive はこれを待って受

け取る。

以上が、複数プロセスの並列実行を制御するための基本的な機能およびその代表的な記述法である。これらの機能を組み込んだ言語として、CPS(Communicating sequential process)を基にしたoccam[21]や Concurrent PASCAL[22] ,Ada などがあげられる。

また、並列性を自動的に引き出せることを目的として、単一代入言語[25),26)]やデータフロー言語[27)]などの研究も進められているが、ここではふれない。

2.3 システム設計

2.3.1 ブロードキャストメモリ

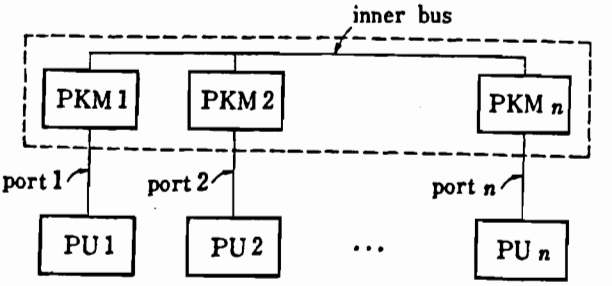


図2.4 ブロードキャストメモリ

ここでは、本論文で提案する３つのシステムの中心となるブロードキャストメモリの構成と動作について述べる。

ブロードキャストメモリは図2.4 の点線内で示されるマルチポートメモリシステムである。 容量の等しい n個のメモリバンク(バケットメモリ : PKM_1～PKM_n)には同一のアドレス付けがなされており、全バンクの同一アドレスには同一データが格納される。したがって、このメモリシステム全体の論理的な容量はバケットメモリ１つ分の容量に等しい。

このメモリシステムへのアクセスは、n個のポート($port_1$～$port_n$)を通して行う。 読み出し時には inner bus は使用されず、各PKMは各 port からそれぞれ独立にアクセスされる。 したがって、同時に最大 n台のプロセッサが n個の異なったアドレスからデータを読み出すことができる。

一方、書き込みは 1メモリサイクルに 1ポートのみから行う。書き込みを行うポートから入力されたデータは、そのポートに接続されているバケットメモリに書かれると同時に、inner bus を通して全バケットメモリの同一アドレスに転送される。この機能により、全バケットメモリ内のデータは常に等しく保たれる。

以上のような機能をもつブロードキャストメモリは、全プロセッサから大域的に参照されるデータの格納に適しており、全要素に対して共通データを作用させることの多い行列計算問題に対して効果的である。

2.3.2 基本設計

数値計算を必要とする問題の多くは大型の線形方程式の問題に帰

着させることが可能で、これを高速に解くことがきわめて重要な課題となっている。本システムは大規模連立方程式の並列計算を主目的とし、設計においては、実現の容易さ・高速性・コストパーフォーマンスの向上を目標とする。

以上の設計方針から、ハードウエアの基本構成を図2.5 のように定める。

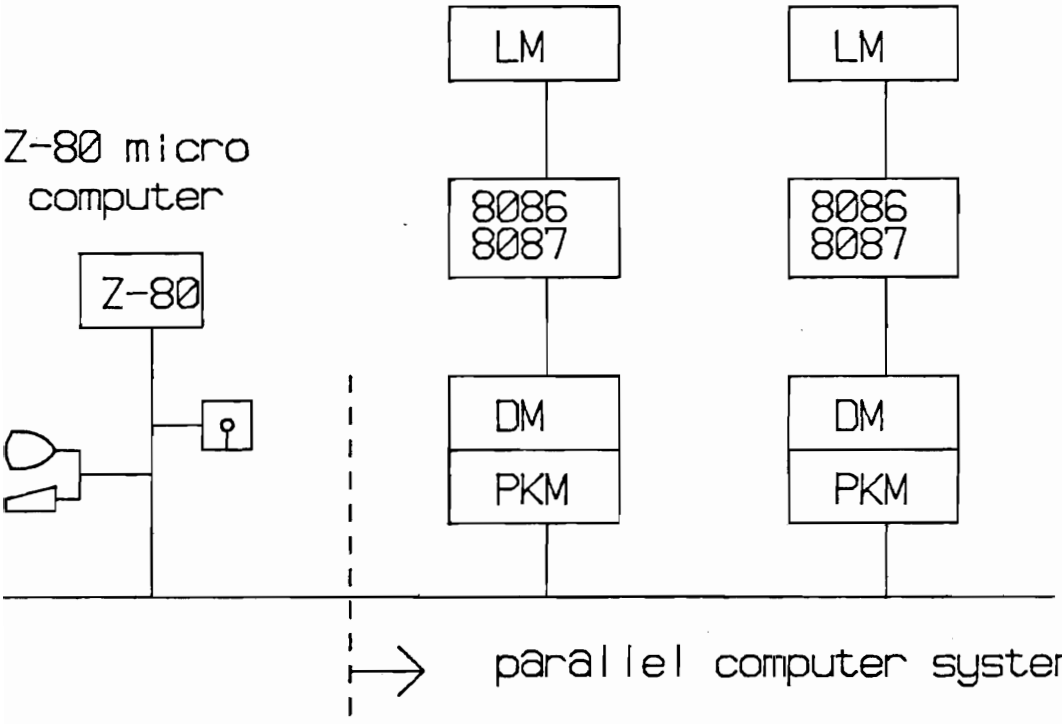


図2.5　システム構成

(1) 共通バス結合とし、実装や拡張を容易にする。
(2) プロセッサモジュールの1ボード化、1チップ化のため、モジュールの入出力信号数を少なくし、共通バスに集める。
(3) 既存のマイクロコンピュータシステムに付加する形でシステムを構成することにより、デバッグやソフトウエア開発を容易にする。
(4) ブロードキャストメモリを中心にしてメモリ構成を工夫することにより、バス結合のもつ欠点であるアクセス競合を極力少なくする。

特に(4)のメモリ構成では、ローカルメモリ(LM)、ブロードキャストメモリ(BCM)のほかにデータメモリ(DM)を設ける。データメモリは各プロセッサがそれぞれ個別に持つメモリであるが、他へのアクセスも可能であり、局所性のある共有データの格納に向いている。これらのメモリ構成については次節のシミュレーションによって、その有効性を検討する。

次に、ソフトウエアの構成について述べる。ソフトウエアにおいては、オペレーティングシステムに市販のもの(CP/M-86)を用い、プログラミング言語は新しく開発することにする。言語設計においては、数値計算問題に対する記述のし易さと実行効率の向上を目標として、次のような点を基本方針とする。

(1) 数値計算問題は潜在的に高い並列性を持っているので、あまり複雑な制御構造は考えない。
(2) メモリへのアクセス競合を低減させるため、データ(変数)の配置場所を指定できるようにする。

2.3.3 メモリシステムの評価

前節で述べたように、本システムはメモリシステムに最大の特徴がある。ここでは、このメモリシステムについて、計算機シミュレーションにより基本的性質を確かめ、次章における設計と応用の参考とする。

シミュレーションでは、ブロードキャストメモリおよびデータメモリについて、アクセス競合による効率の低下を測定する。このシミュレーションにおいては、以下の仮定を設ける。

(1) メモリアクセスは離散的に起るものとし、1回のアクセスをシミュレーションの単位時間とする。

(2) 各プロセッサは、それぞれ m回のローカルアクセスと n回のグローバルアクセス(BCM または DM へのアクセス)を行うものとし、その順序はランダムとする。

(3) 競合に対する調停は各ステップ毎に独立に、一様乱数を用いて行う。

また、アクセスの効率を表すものとして次の値を定義する。

効率 =(競合なしの時の実行時間)
/(最も遅いプロセッサの実行時間)

分子は m+n であり、競合がなければ効率は 1 となる。

本システムのメモリを評価するにあたり、比較のために通常の共有メモリ結合による並列計算機システムについても同様のシミュレーションを行う。このシステムはバス上の共有メモリを複数のプロセッサがアクセスしあうシステムであり、図2.6 にその構成を示す。

以下、ブロードキャストメモリおよびデータメモリに対するシミュレーションの結果を示す。

ブロードキャストメモリ

シミュレーションは次の 3つのパラメータを変化させて行った。

1. プロセッサ台数
2. G-L比 ：（BCM へのアクセス）/（全メモリへのアクセス）
3. Read-Write比 ：（BCM への書き込み）/（BCM へのアクセス）

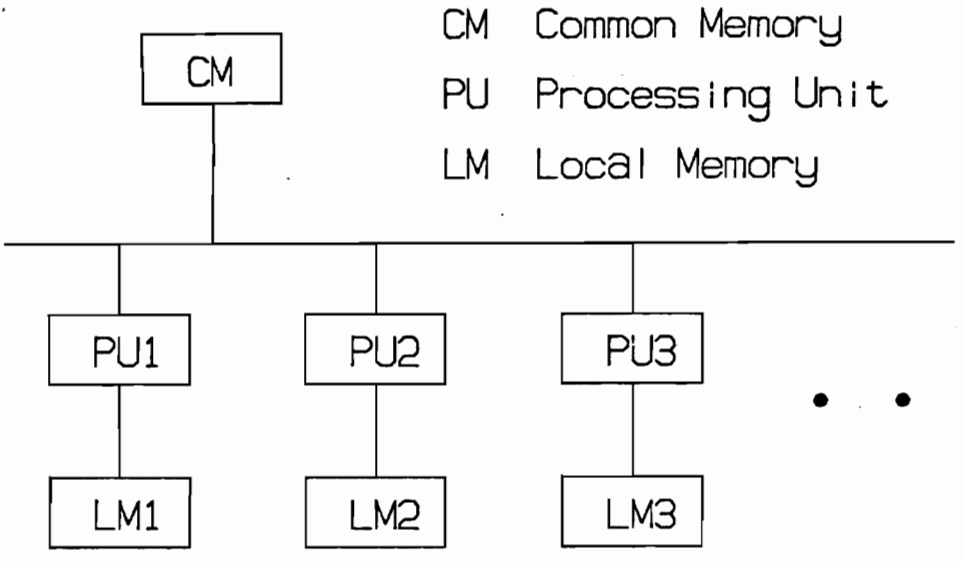


図2.6 共有メモリ結合システム

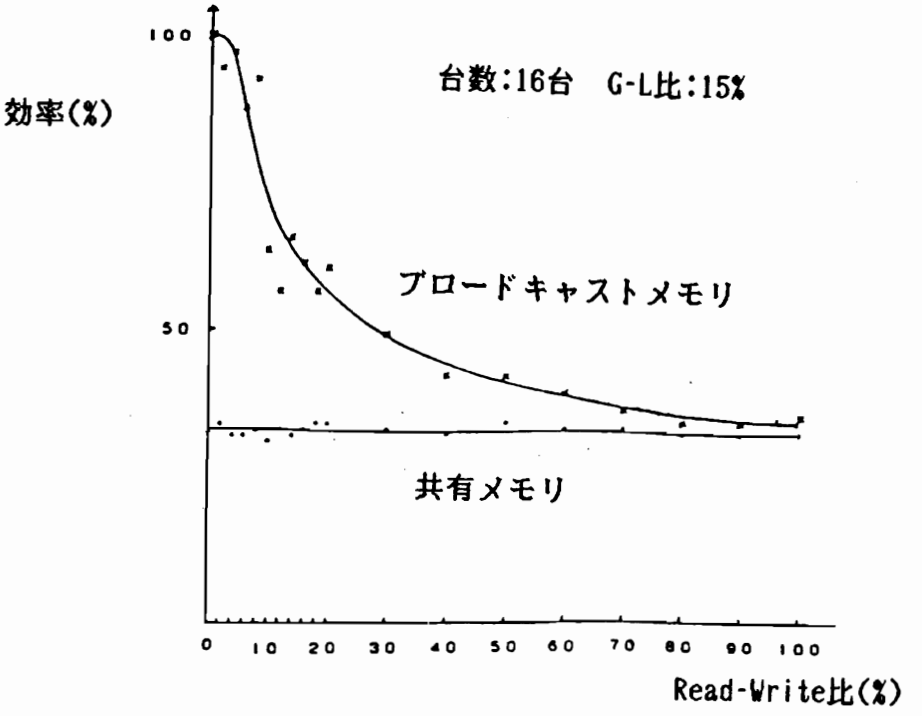


図2.7　シミュレーション結果(ブロードキャストメモリ)

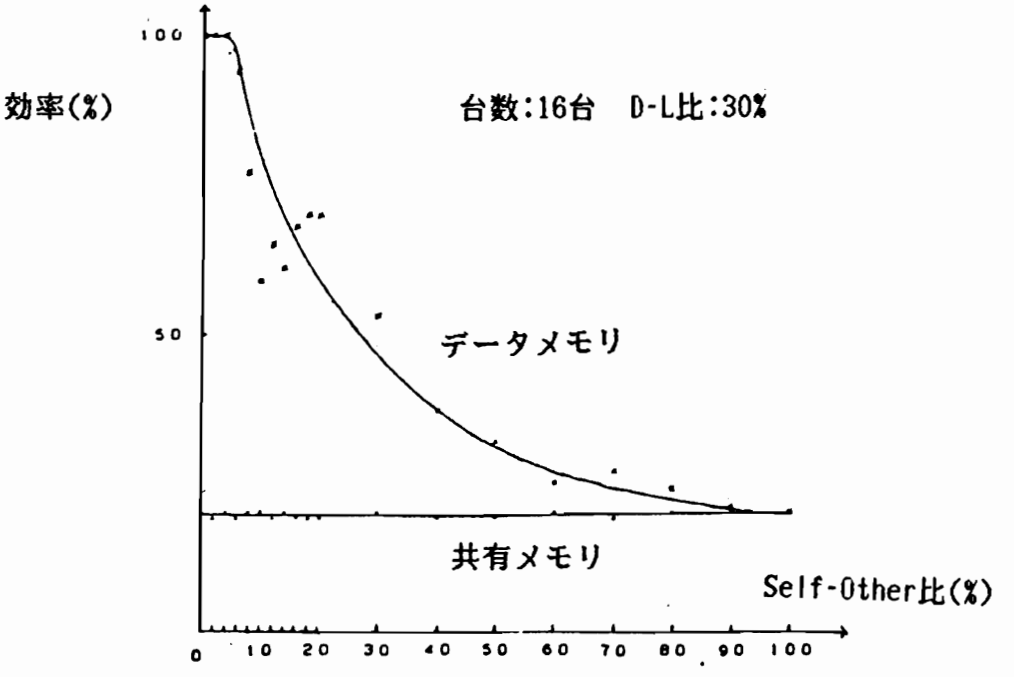


図2.8　シミュレーション結果(データメモリ)

シミュレーション結果の一例として、台数および G-L比を固定したときの、Read-Write比に対する効率の変化を図2.7 に示す。

共有メモリでは read も write も同じアクセスなので、 その比が変化しても効率は変化しないが、ブロードキャストメモリではこれが変化する。 アクセスがすべて write の時は共有メモリと同じ効率であるが、read が増すにしたがって効率は良くなり、 すべて read の時は効率は100%となる。

行列計算などの場合には、 共有データに対する Read-Write比は低く、ブロードキャストメモリは効果的である。しかし、ブロードキャストメモリは論理的な容量に対してプロセッサ台数倍のメモリを必要とする点で、大容量データの記憶は困難である。

データメモリ

データメモリに対しては、次の 3つのパラメータを変化させてシミュレーションを行った。

1. プロセッサ台数
2. D-L比 ：(DM へのアクセス) / (全メモリアクセス)
3. Self-Other比 ：(他のDM へのアクセス) / (DM へのアクセス)

ブロードキャストメモリでの場合と違い、データメモリの効率はRead-Write比には関係しない。そのかわりに、アクセスが自分のデータメモリに対してであるか(Self)、他プロセッサの持つデータメモリに対してであるか(Other)によって効率が変化する。

台数および D-L比を固定した場合の、Self-Other比に対する効率

の変化を図2.8 に示す。

並列処理されるデータには局所性を持つものがあり、その度合いは問題によって変化する。本システムのデータメモリは局所性の変化に対して柔軟に対応でき、アクセス競合を軽減できる。

2.4 結言

本章では、まず、並列計算機の結合方式と並列動作の制御機構に関して、設計に必要な基礎概念をまとめた。 次に、これをもとに、数値計算を目的としたシステムのハードウエア・ソフトウエアの基本構成を定めた。ハードウエアに関しては基本的にはバス結合を用い、メモリシステムに工夫を持たせることを述べた。また、ソフトウエアについては、数値計算問題の並列計算に焦点をおいた並列プログラミング言語を開発することを述べた。最後に、シミュレーションによって、メモリシステムの基本動作を確かめた。 次章では、これらをもとに試作システムを実現し、連立方程式の諸解法に対する有効性を確かめる。

第3章

ブロードキャストメモリ結合形並列計算機システムの実現と評価

3.1 緒言

本章では、ブロードキャストメモリ結合形並列計算機システムの詳細設計と実現について述べ、連立方程式の諸解法に対する並列計算結果を示す。ハードウエアについてはブロードキャストメモリを実現するためのバス制御機構を示し、ソフトウエアについては並列プログラミング言語の設計と並列計算の方法について述べる。 次に、連立方程式の解法であるガウス消去法と共役勾配法を取り上げ、並列計算技法と実行結果を示した後、ブロードキャストメモリの効果や並列計算の効果について考察する。

3.2 ハードウエア+

3.2.1 全体構成

ハードウエアの全体構成を図3.1 に示す。 Z-80[30]を用いたマイクロコンピュータと、並列計算を行う複数のプロセッシングユニッ

+ システムの拡張にともない、メモリ容量および Z-80 の役割が、文献39)で報告した時と少し変わっている。

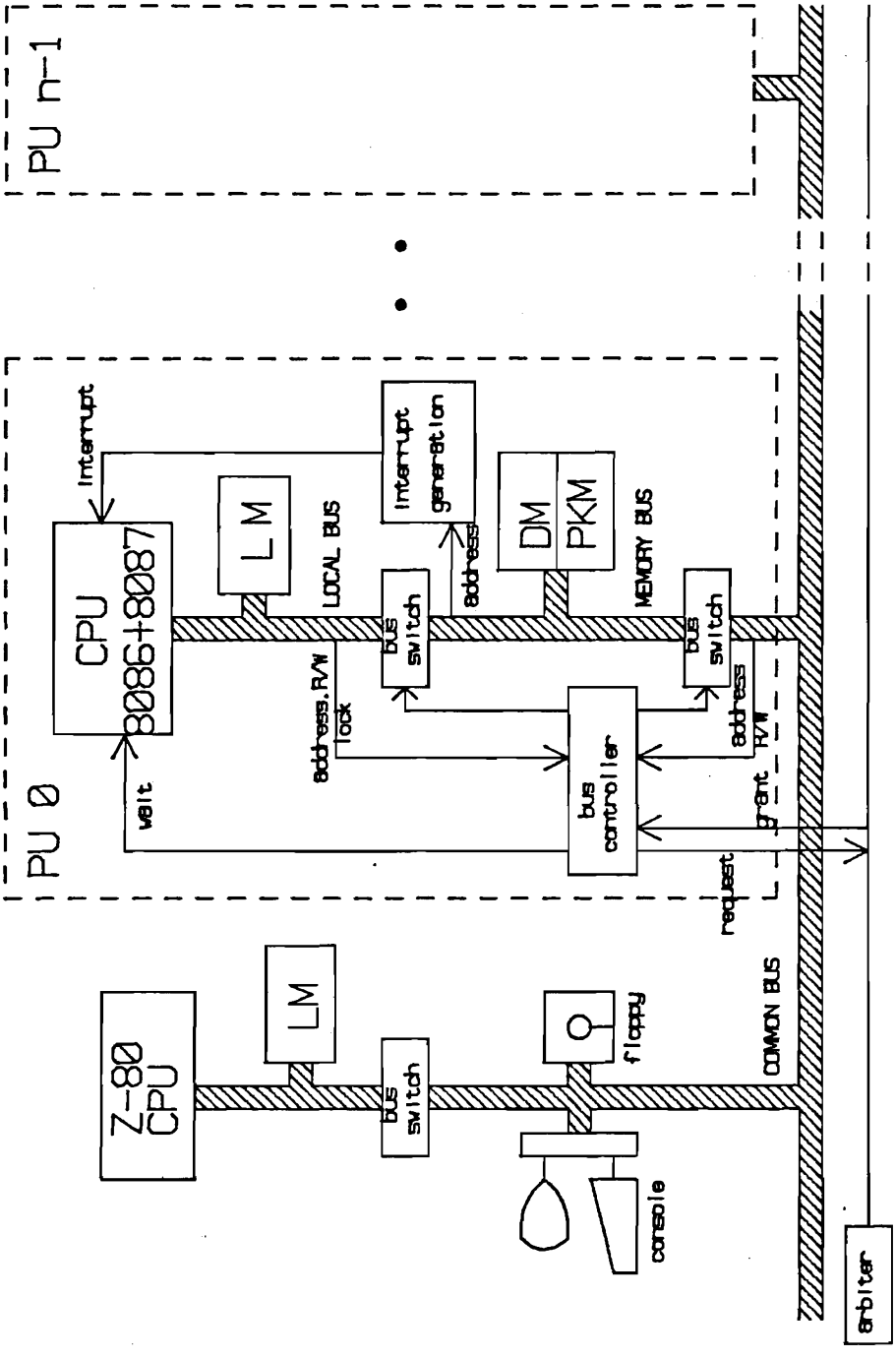


図3.1 ハードウエア構成

ト(PU)とを共通バスで接続している。 Z-80 自体はシステムの立ち上げ時およびメンテナンス時にのみ動作をし、通常は停止している。コンソールやフロッピーディスク装置などの入出力装置は、 各 PU からもアクセスが可能であり、 システム立ち上げ後は PU の内の1台がホストプロセッサとして動作し、プログラム開発やデータの入出力を行う。

並列計算の実行時には、 ホストプロセッサが自分を含めた全 PU に同一プログラムをロードし、全 PU に起動をかける。並列計算が終了すると各 PU は動作を止め、再びホストプロセッサのみの動作に戻る。

なお、ハードウエア上は PU_0～PU_{n-1} のすべてのプロセッシングユニットは同じものであり、基板上のスイッチの切り替えによってホストプロセッサを他のユニットに移すことができる。

3.2.2 ハードウエアの詳細設計

(1) プロセッサ

プロセッサにはインテル社の16ビットマイクロプロセッサ8086に、数値データプロセッサ 8087 を付加して用いる[31]。

8086 は 1Mバイトのアドレス空間を持ち、マルチプロセッサシステムへの対応を考慮した機能も組み込まれている。 8086 の持つマルチプロセッサ機能として次の2点がある。

(a) バスロック機能

メモリ上のデータに対して、交換やインクリメント・デクリメントなどの操作を行う場合には、2回のメモリアクセスが行われる(すなわち、旧データの読み出しと新データの書き込み)。 バスロック

はこの一連のメモリアクセスを不可分の動作とするための機能であり、これによって相互排除動作の実現が可能となる。

(b) WAIT 命令

CPU の動作を外部ハードウエアに同期させるための命令である。プログラム中の WAIT 命令を実行したCPUは、test と名付けられた入力端子の状態を調べ、これが偽ならば真になるまで次の命令の実行を遅らせる。

また、 8087 は 8086 に付加して用いる数値計算専用プロセッサである。 8087 による浮動小数点計算は 8086 と比較して数十倍高速であり、単精度加算を 17μsで、同じく乗算を 19μsで処理する。8087 は 8086 と並行して動作し、8086 との同期は前述の WAIT 命令によってとられる。

(2) メモリ構成

メモリ構成を図3.2 に示す。1Mバイトのアドレス空間を 64Kバイトずつのセグメントに分け、ローカルメモリ(LM)、ブロードキャストメモリ(BCM)、データメモリ(DM)を図のように配置する。

ローカルメモリは各プロセッサ固有のメモリであるため、全プロセッサとも同じセグメント上に置かれる。ブロードキャストメモリは前述のように論理的に1台分の容量しか持たないので、これも1セグメントに置く。データメモリは、すべてのプロセッサがシステム全体のデータメモリをアクセスできるようにする必要があるため、各データメモリにそれぞれ固有のセグメントを割り当てる。

ローカルメモリは主としてプログラムの格納に用い、ブロードキャストメモリとデータメモリは、それぞれ広域的なデータと局所的

なデータの格納に用いる。また、ブロードキャストメモリは、同期や相互排除制御に必要な制御変数の格納にも用いる。

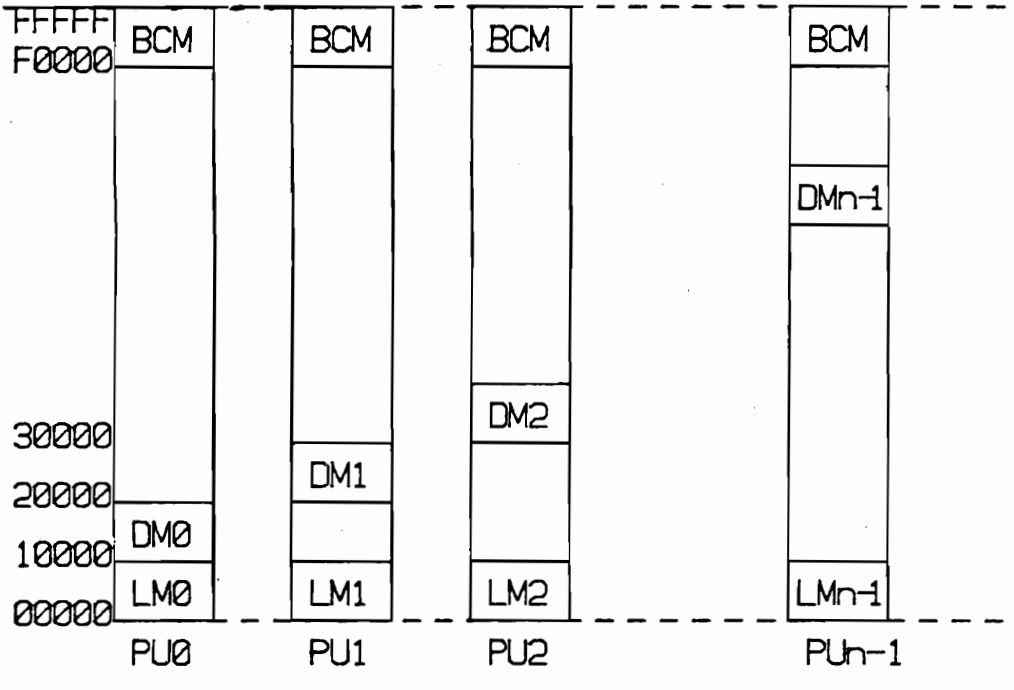


図3.2　メモリ構成

(3) バス

バス構成と各メモリの位置を図3.3 に示す。ローカルバスと共通バスとは2つのバススイッチ(SWL と SWC)を通して結ばれる。 ローカルメモリはローカルバス上にあり、バススイッチの状態に関係なく CPU からアクセスできる。

データメモリとバケットメモリ (ブロードキャストメモリの構成要素)は、2つのバススイッチの間に位置する。バススイッチの状態とこれらのメモリアクセスとの関係を表3.1 に示す。

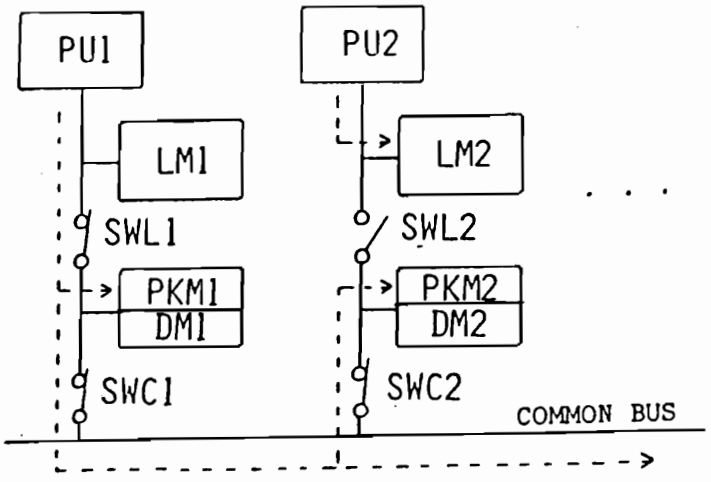


図3.3 バス構成

表 3.1 バススイッチとメモリアクセスの関係

SWL	SWC	メモリアクセス
ON	ON	他プロセッサのDM(read,write),PKM(write)
ON	OFF	自分のDM(read,write),PKM(read)
OFF	ON	他プロセッサからのDM,PKMへのアクセス
OFF	OFF	LMのアクセス

SWC は双方向性であり、アクセスの方向によってスイッチの ON/OFF と同時にその方向も制御する必要がある。

プロセッサ1 がブロードキャスト転送を行っている場合のバススイッチの状態が図3.3 である。PU_1 は両方のスイッチを閉じ、その他のすべてのプロセッサは SWC のみを閉じる。 PU_1 から出力されたデータは、自分のバケットメモリに書かれると同時に、共通バスを通してすべてのバケットメモリに転送される。 このとき、PU_2～PU_{n-1} はデータメモリに対してアクセスできないが、ローカルメモリに対するアクセスは同時に行うことができる。

バススイッチの制御機構を図3.4 に示す。CPU からのアクセスがどのメモリに対するものであるかという判断は、そのアドレスによって行う。

図3.4 の上側の要求発生部(request generation 部) はローカルバスのアドレス・リード・ライト信号からそのプロセッサがアクセスしようとしているメモリを知り、バススイッチをそのアクセスに

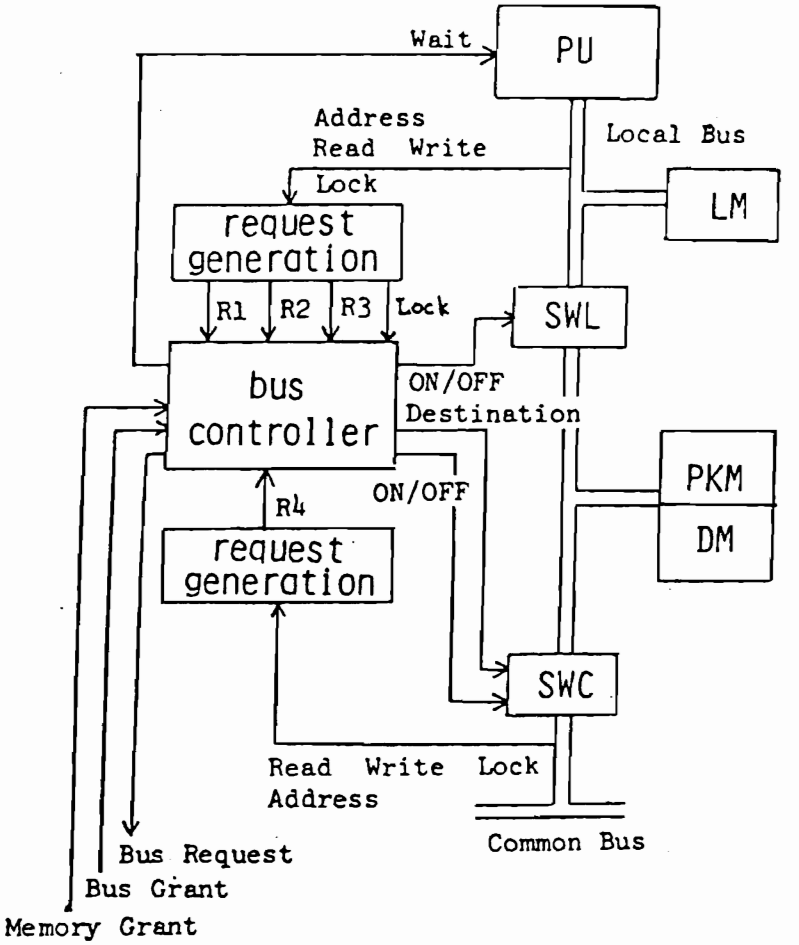

Wait
PU
Address
Read Write
Local Bus
Lock
LM
request
generation
R1
R2
R3
Lock
SWL
bus
controller
ON/OFF
Destination
PKM
DM
R4
ON/OFF
request
generation
SWC
Read Write Lock
Address
Common Bus
Bus Request
Bus Grant
Memory Grant

図3.4 バス制御機構

対応した状態(表3.1)に変更すべく、 バスコントローラに対して要求を(R1,R2,R3)出す。一方、PKM あるいは DM が共通バスを通して他のプロセッサからアクセスされようとしている場合は、下側の要求発生部が共通バス上のアドレス・リード・ライト信号からこれを判断し、要求(R4)を出す。

各メモリアクセスと、そのときに出力される要求信号を以下に示す。

R1:自分の DM の読み書き。PKM の読み出し。
R2:PKM への書き込み。
R3:他の DM の読み書き。
R4:共通バスからの DM,PKM へのアクセス。

バスコントローラはこれらの要求を受けると、それぞれの要求に合わせてバススイッチを切り換える。また、複数の要求に対して競合がおこった場合には、これを調停し、必要に応じてプロセッサに wait 信号を出して実行を待たせる。

各要求に対するバスコントローラの処理を以下に述べる。まずR1に対しては、 メモリが他プロセッサから使用中(すなわち R4 が先に出されている)ならばプロセッサに wait をかけ、 他プロセッサからの使用が終わった時点でバススイッチを切り換えて wait を解除する。

次に、R2 あるいは R3 が出された場合(すなわち、プロセッサが他のプロセッサのメモリをアクセスしようとしている場合) のコントロール手順を以下に示す。

1. 共通バス使用の調停を行うバスアービタ(図1) に対して共通バス使用要求(bus request)を出力する。 要求に対して許可(bus grant)がおりないときはプロセッサに対して wait 信号を出力する。
2. 共通バスの使用許可を得たバスコントローラは、SWL,SWC を閉じて共通バスに信号を出力し、相手のバスコントローラに対してメモリの使用を要求する。
3. R3 に対しては相手のバスコントローラからの、R2 に対しては全バスコントローラからのメモリ使用許可(memory grant)が得られた時点で wait を解除し、相手メモリをアクセスする。メモリ使用許可信号は共通バスにオープンコレクタ出力されており、ワイヤードロジックによって作成される。

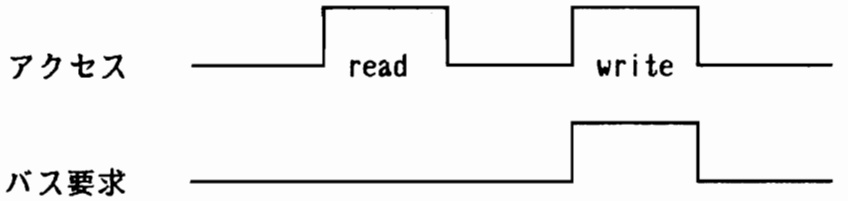
一方、他プロセッサからアクセスされる側のバスコントローラは次の動作を行う。

1. 共通バス上に自分の持つメモリのアドレスが現れたとき、要求発生部はバスコントローラに対して R4 を出力する。
2. R4 を受けたバスコントローラはバススイッチを切り換え、 メモリ使用許可信号を出力する。このとき、もし内部からアクセス中であれば、許可信号の出力を遅らせる。

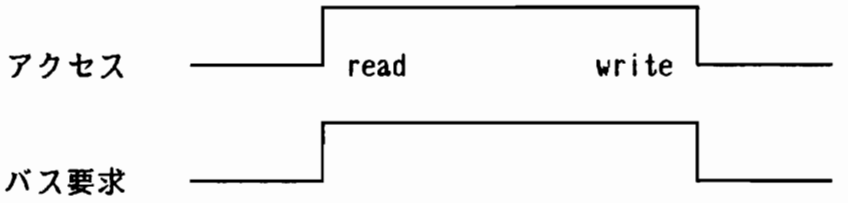
以上のように、他プロセッサのメモリをアクセスするには、バスの使用権と相手メモリの使用権とを得る必要がある。バス使用の調

停に対しては、プロセッサ台数の少ない試作システムにおいてはプライオリティを固定にしても問題ないとの判断から、ハードウエアの簡単なデイジーチェーン方式をとる。 また、 メモリ使用の調停(R1 と R4 との間での調停)では、バス側からの要求(R4)に優先度を持たせる。これは、バスの使用時間をできるだけ少なくするためである。

また、バスコントローラに対して lock 信号が出されると共通バスの使用権は固定され、lock信号が落とされるまで他のプロセッサは共通バスを使用することができなくなる。交換命令などにおける複数のメモリアクセスは、lockを用いることによって一連のメモリアクセスと見なされ、相互排除の処理が可能となる。lockの使用例



(a) 通常のアクセス



(b) バスロック・アクセス

図 3.5 バスロック・アクセスのタイミング

として、ブロードキャストメモリ上のデータとCPU 内のデータとを交換する場合のバスアクセスタイミングを図3.5 に示す。(a)は通常のアクセスであり、readとwrite は分断され、バス要求は write 時にしか出力されない。 これに対して lock を使用した実行では、(b)のように read と write は一連の動作と見なされ、バス要求は全区間で出される。 ただし、実際にバスが使われるのは write 時のみである。

以上がバス制御の詳細であるが、ブロードキャスト転送などのバスアクセスはすべてこの部分で制御されているので、PKM と DM には通常のメモリ基板を用いることができる。

最後に、ローカルメモリについて補足を加える。ローカルメモリは通常、それを持つプロセッサからしかアクセスできない。しかし、次の2つの場合には、Z-80 およびホストプロセッサが他のプロセッサのローカルメモリをアクセスできなければならない。

1. 起動時に Z-80 がホストプロセッサのローカルメモリにオペレーティングシステム(CP/M-86)をロードするとき
2. 並列実行時にホストプロセッサが全プロセッサに並列プログラムをロードするとき

このため、ローカルメモリには DMAモードと呼ぶモードを設け、このモード時のみ特別に外部からのアクセスを可能にする。 DMAモードには次の2つの場合がある。

1. Z-80 動作時のDMAモード

Z-80 から、 ホストプロセッサのローカルメモリのみがアクセスできる。

2. ホストプロセッサ動作時のDMAモード

ホストプロセッサからの出力データは自分を含めた全プロセッサのローカルメモリにブロードキャスト転送される。これにより、全プロセッサに同一プログラムが同時にロードされる。

(4) Z-80 および入出力装置

入出力装置は共通バスを使って Z-80 と 8086 の両CPU から直接アクセスできるようになっている。しかし Z-80 と 8086 とが同時に動くことはなく、常にいずれか一方のみから制御される。Z-80が動作するのはシステム立ち上げ時とデバッグ・メンテナンス時のみである。Z-80 は 8086 の持つ1Mバイトのアドレス空間全体を 4Kバイト幅のウインドウを通してアクセスできる。

電源投入からホストプロセッサの起動までの手順を以下に示す。

1. 電源投入
2. Z-80 上で CP/M を起動
3. ローカルメモリを DMAモードにして、ホストプロセッサのローカルメモリに CP/M-86 をロード
4. ホストプロセッサに起動をかけ、Z-80 は停止

Z-80 からのハードウエアデバッグは、上記の 2 の状態で行う。この際、ローカルメモリとパケットメモリはホストプロセッサの分しか見ることができないので、基板上にあるホスト切り換えスイッチによりホストプロセッサを切り換えて特定のユニットをデバッグ

する。

3.2.3 ハードウエアの製作

試作機のハードウエアサイズについて述べる。プロセッサ(8086)1台当りのハードウエア量は、メモリを除いて LSI 3個、TTL-IC 約50個となっている。メモリ容量は LM,PKM,DM ともに各 64K バイトで、市販のスタティックRAMボードを用いている。プロセッサを8台持つ現システム全体では、 ローカルメモリ各64Kバイト、ブロードキャストメモリ64Kバイト、データメモリ512Kバイトとなる。

また、プロセッサとバスとの接続信号数は

アドレス信号	21 本
データ信号	16 本
コントロール信号	18 本
計	55 本

となっている。

次に、実装上の注意点について述べる。 試作機では、8台のプロセッサは2個の標準ラックに収められており、 バスラインは全長1m程度になっている。このため、バスラインの終端処理を行わないとシステムは正常に動作しない。 さらに台数を増加するためには、1ボード化によってユニット間距離を短縮させる必要がある。

3.3 ソフトウエア

3.3.1 全体構成

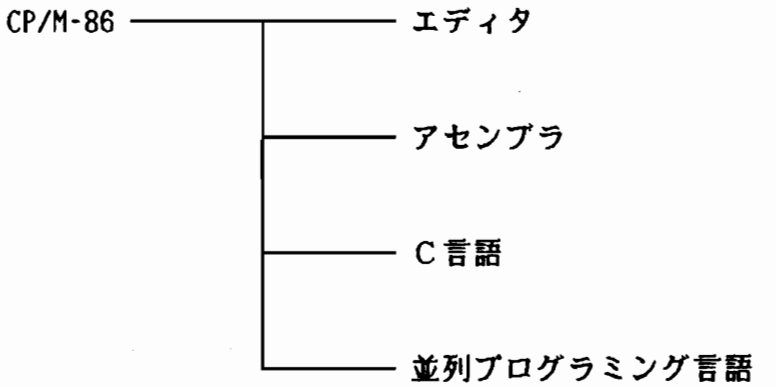


図3.6 ソフトウエア構成

ソフトウエアの全体構成を図3.6 に示す。オペレーティングシステムには CP/M-86[33]を用いる。 この図で、最後にある並列プログラミング言語については次章でその詳細を述べる。本並列計算機に対する並列アルゴリズムはこの言語で記述し実行する。

CP/M-86 自体はマルチプロセッサに対応する機能をいっさい持っていないため、並列実行に必要な制御機能はすべてこの言語とその処理系に持たせなければならない。

3.3.2 並列プログラミング言語

並列処理を記述できる高級言語としては、2.2.3 節であげたようなものが提案されている(Concurrent PASCAL や occam など)。 し

かし、これらの言語は汎用的な非同期並行プロセスの記述を目的としており、本システムの目的である共有メモリシステムによる大規模数値計算に対して、必ずしも最適な言語とは言えない。

そこで、ここでは次の2点を特徴とする並列プログラミング言語を提案する。

(1) 「ローカルメモリ」と「共有メモリ」とを持つシステムに広く対応でき、これらの性質を十分生かすことができる。

(2) 数値計算などのように、潜在的に高い並列性を持つ問題に対して、記述が容易である。

言語設計においては、C言語に並列処理機能を付加する形をとる。以下、新たに付け加えた機能について述べる。

データ構造の拡張

本システムを含めたメモリ共有形並列計算機システムのハードウエアモデルを図3.7 に示す。本システムにおいては、ブロードキャストメモリが共有メモリに対応し、ローカルメモリとデータメモリがローカルメモリに対応する。

ここで実行効率を左右する通信コストについて考察する。図3.7 における プロセッサ - メモリ間の通信コストを C(P,M) で現すとすると、一般に次の関係が成り立つ。

$$C(PUi,LMi) < C(PUi,CM) < C(PUi,LMj) \ , \ i \neq j$$

すなわち、あるプロセッサに着目すると、自分のローカルメモリ、

共有メモリ、他プロセッサのローカルメモリの順に通信コストは大きくなっていく。したがって、処理するデータがどこに置かれているかによって実行効率は大きく左右される。　このために本言語では、これらのメモリに対する変数の割り付けを指定できるようにする。

1．記憶クラスの拡張

C言語では変数のクラスとして、

a）自動変数 ： 関数内でのみアクセス可能

b）外部変数 ： 関数間でアクセス可能

というクラスが用意されている。本言語では、プロセッサ間でのアクセスが可能な2つの記憶クラスを導入する(universalとshared)。

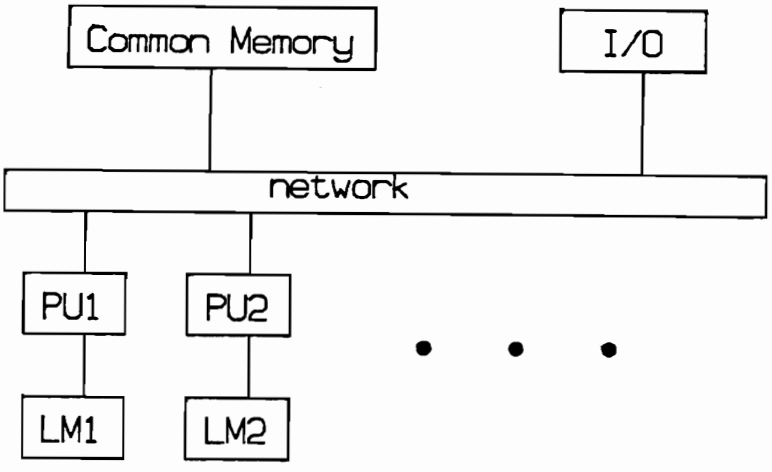


図3.7　　メモリ共有形並列計算機システム

これらはいずれも、すべての関数とすべてのプロセッサからアクセス可能であるが、 universal 変数は共有メモリに置かれ、 shared 変数はローカルメモリに置かれる点が違っている。 shared 変数は多量のデータを分配して処理する場合に向いている。

2. 配列の分配宣言

行列計算などにおける大量のデータを、前述の shared 変数として各プロセッサに分配する場合に用いられる。 たとえば shared a[100][100] とすることにより、配列 a は図3.8 のように各プロセッサに分配される。

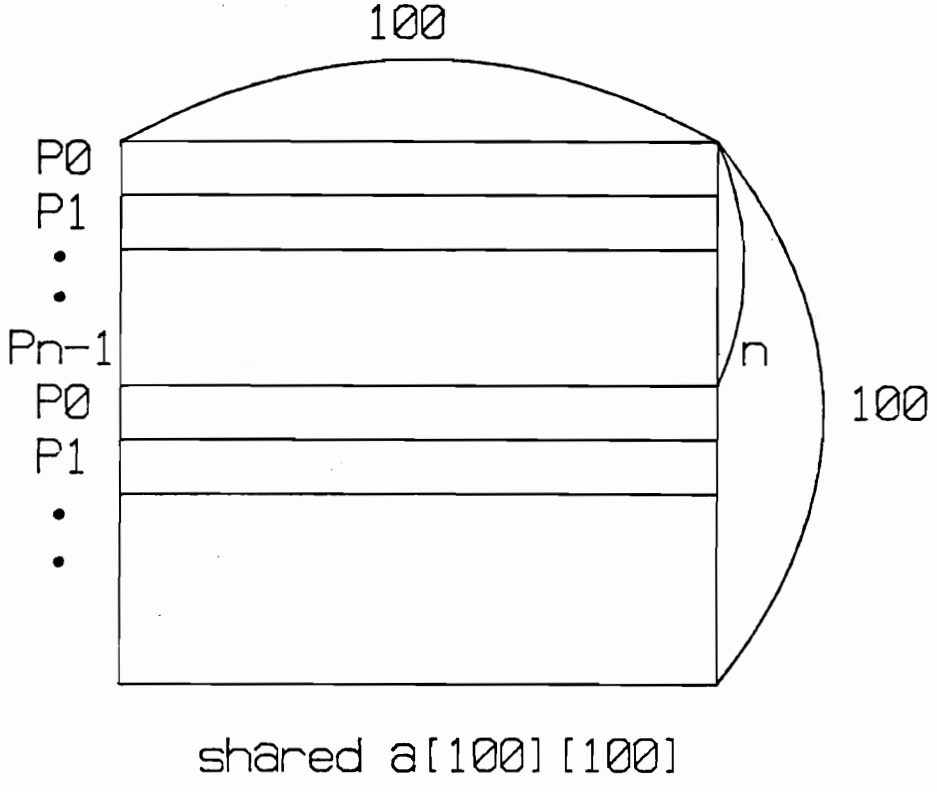


図3.8 配列要素の分配

3. PU間での参照

他のPUの shared クラスの変数を参照する場合、それが 2. によって分配されたものならばどのPUか一意的に決定できる。 しかし、すべてのPU上に同じ名前でとられている変数に対してはPU指定が必要となる。このため他のPUの変数を参照する場合には、相手のPU番号を変数名の前につけてこれを指定する。ただし何も指定しない場合は、自分自身を指定するものとする。

制御構造の拡張

1. 単一パス指定

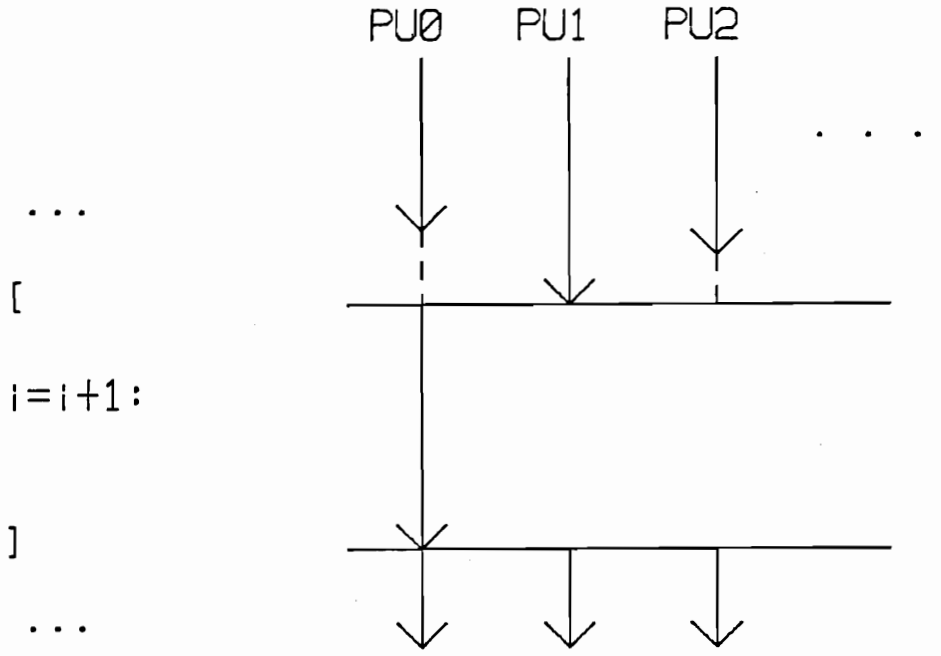


図3.9 単一パス指定

従来の並列動作の記述法には、直列部分から並列部分への移行に着目したものが多い(fork - join や cobegin - coend など)。本言語では、並列性の高い問題を取り扱うという点から、逆に並列実行中の直列部分に着目した記述を行う。すなわち、通常は並列に動いているものとし、直列動作の必要がある場合にこれを指定する。

単一パスは "[" と "]" でくくることによって指定する。図3.9にこの例を示す。"[" で全プロセッサは同期をとり、ホストプロセッサのみが中の文を実行する。その後、"]" によって再び全プロセッサに起動がかけられる。特に、内部に文を持たない "[]" は、並列実行中での一斉同期を表す。

2. 仕事の分配

並列化できる処理を各プロセッサに分担させるために、pnum と inum と名付けられた2つのシステム変数を導入する。pnum はその値として全プロセッサ台数を持ち、inum は各プロセッサ番号(0～pnum-1)を持っている。これらを用いて、たとえばループの並列処理は次のように記述できる。

```
for(i=0;i++;i<n)
    if(i%pnum==inum){
        ・・・・・・
    }
```

ここで、% は剰余を求める演算であり、if 文によって i に関する処理は PU_0 から順に割り当てられていくことになる。

なお、 pnum と inum の値は実行時にオペレーティングシステムから与えられるので、 プログラム自体は PU台数によらないものを書くことができる。

3. 相互排除

相互排除は universal 変数に対するアクセス時に必要となる。本言語では、前述の lock - unlock による制御を取り入れる。変数 v に対して、

```
lock(v);
```

を実行することによって、v に対する相互排除領域に入り、

```
unlock(v);
```

を実行することによって、相互排除領域から出る。lock - unlock 文は universal 変数に対してのみ有効であるものとする。

4. 通信

相互排除と同じく universal 変数に対してのみ有効とする。変数 v に対して値を送る場合は、v に値を代入した後、

```
send(v);
```

によってそのことを知らせる。一方、

```
receive(v);
```

では、 send(v) の実行を待って、v の使用を始める。ただし send 文の実行が先に行われておれば、 ただちに v の使用を始める。また、

```
release(v);
```

は、初期化の時に変数を未使用状態にする文である。

以上が本言語の持つ並列処理機能であるが、ここで例として、行列とベクトルの積を求める並列プログラムを示す。行列 a[100,100] とベクトル b[100] に対して、その積 c[100] を求めるプログラムは図3.10 のように記述できる。

```
main{
shared float a[100][100],c[100];
universal float b[100];
int i,j;
float t;
[ (a,bの入力) ]
for(i=0;i++;i<100)
   if(i%pnum==inum){
      t=0;
      for(j=0;j++;j<100)
         t=t+a[i][j]*b[j];
      c[i]=t;
   }
[ (cの出力) ]
}
```

図3.10　行列・ベクトル積のプログラム

変数宣言の部分では配列 a,c をローカルメモリに、 b を共有メモリに置いている。次に、単一パス指定を用いて、ホストプロセッサのみが a,b に外部からデータを入力する。配列 a は行単位で各プロセッサに分配されており、これをそれぞれのプロセッサが並行して計算する。最後に、全プロセッサの同期をとって、答を出力する。

この例からもわかるように、本言語による並列プログラミングの考え方は次のようにまとめることができる。

a. 全プロセッサに対して同一の一つのプログラムを書く。
b. ホストプロセッサがデータを各プロセッサに分配する
c. 各プロセッサは自分の分担に対して計算を行う。
d. 各プロセッサ内では通常の直列実行を行う。

3.3.3 言語処理系

ここでは、この並列プログラミング言語を試作機のハードウエア上に実装する。まず、前節で述べたデータ構造および制御構造の拡張部分(並列処理の制御部分)の実装方法を示す。

データ構造(図3.11)

C言語における自動変数と外部変数はローカルメモリ上に置き、拡張されたクラスである universal 変数と shared 変数はそれぞれブロードキャストメモリとデータメモリ上にとられる(3.2.2 メモリ構成を参照)。

ブロードキャストメモリ上のすべての変数に対して(それが配列の場合は各要素に対して)、制御用フラグを 1対1 に対応させ、こ

れによって lock - unlock および send - receive - release を実現する。 ただし、このフラグは 1変数に対して 1つであるので、lock - unlock と send - receive とは同時に使用できないことになる。

また、shared 変数として配列を分配する場合には、 全体のプロセッサ台数によって実行時に各データメモリ上で占める領域が変化する。 すなわち、大きさ ■ の配列に対して各データメモリ上には (■/pnum)の大きさの領域が必要であるが、コンパイル時には pnum の値がわがらないので領域の大きさを決定できないという問題が生じる(pnum の値は実行時に与えられる)。このため、shared 変数における配列のベースアドレスは pnum や inum と同様に、プログラムのロード時にオペレーティングシステムが与えるものとする。

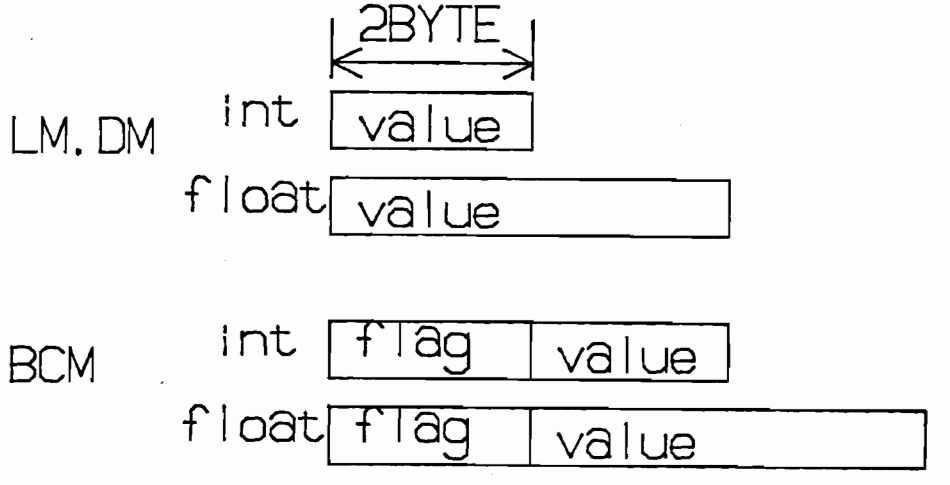


図3.11 データ構造

制御構造の実装

1. 単一パス指定([ ])

ブロードキャストメモリ上に同期制御用のカウンタを設け、初期値を pnum(プロセッサ台数)とする。"[" においては、ホストプロセッサと他のプロセッサは次の動作を行う。

ホストプロセッサ:

カウンタをデクリメントした後、カウンタが0になるのを待って中の文を実行する。

他のプロセッサ:

カウンタをデクリメントし、次に再びその値が pnum になるまで実行を停止する。

また、単一パス終了を示す "]" では次の処理を行う。

ホストプロセッサ:

カウンタの値を pnum に戻し、実行を続ける。

他のプロセッサ:

カウンタの値が pnum に戻るのを待って、実行を開始する。

すなわち、このカウンタは実行中のプロセッサ数を示しており、各プロセッサが "[" までの実行を終える毎にこの値は減少していく。値が 0 となった時点でホストプロセッサは単一パス部分を実行し、最後にカウンタを pnum に戻すことで再び全プロセッサに起動をかける。この際、同期用カウンタのデクリメントには、前述のバスロ

ック機能を使用しなければならない。

2. pnum,inum

これらの変数はローカルメモリ上に自動的にとられ、その値はプログラムのロード時に与えられる。プログラム中では、他のローカル変数と同様に参照できるが、代入はできない。

3. lock - unlock

ブロードキャストメモリ上の変数n に対応づけられたフラグを用いて実現する。変数 v に対するフラグを v.semで表すと、v.sem=1 はどのプロセッサも変数v に対する相互排除を行っていないことを示し、v.sem=0 はいずれかのプロセッサが相互排除領域にあることを示す。lock(v) では v.sem が 1 になるのを待ち、これを 0にしてから lock - unlock間の文を実行する。また、unlock(v)はv.sem を 1 に戻す。

4. send - receive - release

lock - unlock と同じく、変数 v に付加されたフラグ v.sem を用いる。この場合、v.sem=0 は値の未確定を示し、v.sem=1 は値の確定を示す。 send,receive,release では、それぞれ次の処理を行う。

send(v)　　：v.sem を 1 にする。
receive(v)：v.sem が 1 になるのを待つ。
release(v)：v.sem を 0 にする。

以上が、C言語に追加した各文の処理内容である。

試作機上では、この言語のコンパイラはC言語を用いて作成している。構文解析には、再帰的下向き構文解析法を用いており、目的コードとして 8086 と 8087 のアセンブリ言語を生成する。このあと、CP/M-86 の持つアセンブラ(ASM86)とローダ(GENCMD)を通して、実行形式のプログラムファイルが得られる。　このようすを図3.12に示す。なお現在の版では、関数・構造体および演算子の一部が実現されておらず、コンパイラの機能拡充が課題として残されている。

これらの操作の後、コンソールよりプログラム名と動作台数を入力することによって実行が行われる。この時に入力した動作台数は pnum に代入され、指定した台数分のプロセッサ(PU_0～PU_{pnum-1})に対してのみ起動がかけられる。

本システムでは、このように、プログラムの実行は並列状態から始まる。また、終了時には、全プロセッサで同期をとった後、ホストプロセッサを残して全プロセッサは停止する。

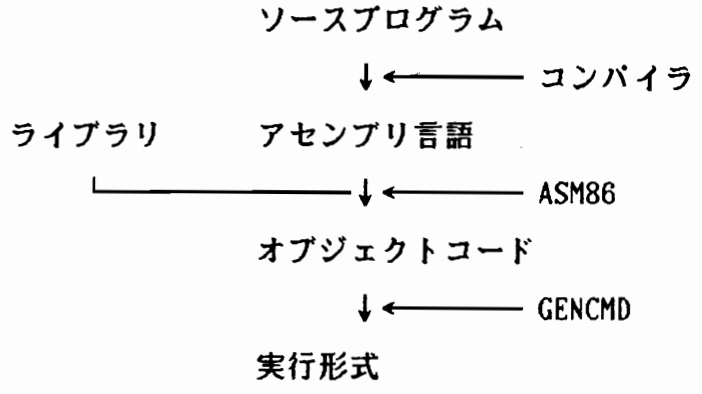


図3.12　処理手順

最後に、試作機のハードウエア・ソフトウエアの性能を示すめやすとして、行列積に対する実行時間を示す(表3.2、図3.13)。プログラムは3.3.2 節の例で示した行列ベクトル積のプログラム(図3.10)とほとんど同じもので、試作機上でコンパイル・実行する。計算は単精度浮動小数点で 8087 を用いて行っている。 表3.2 では 30×30 の行列の積を 20回行った計算時間を示している。

n×n の行列積では n^3 回ずつの乗算と加算が必要である。8台の場合、1回の計算に要する時間は

$$14.11 / (30^3 \times 2 \times 20) = 13.06(\mu s)$$

となり、MFLOPS で表すと 0.077 MFLOPS となる。8087 自体の計算速度は約 0.05 MFLOPS であり、8台で 0.4 MFLOPS となるはずであるから、この結果は 8086 がデータの準備に手間取り、これがボトルネックになっていることを示している。

表3.2　行列積の実行時間

PU台数	実行時間（sec）	speed up
1	100.87	1
2	50.88	1.98
3	34.15	2.95
4	27.45	3.67
5	20.77	4.86
6	17.46	5.78
7	17.44	5.78
8	14.11	7.15

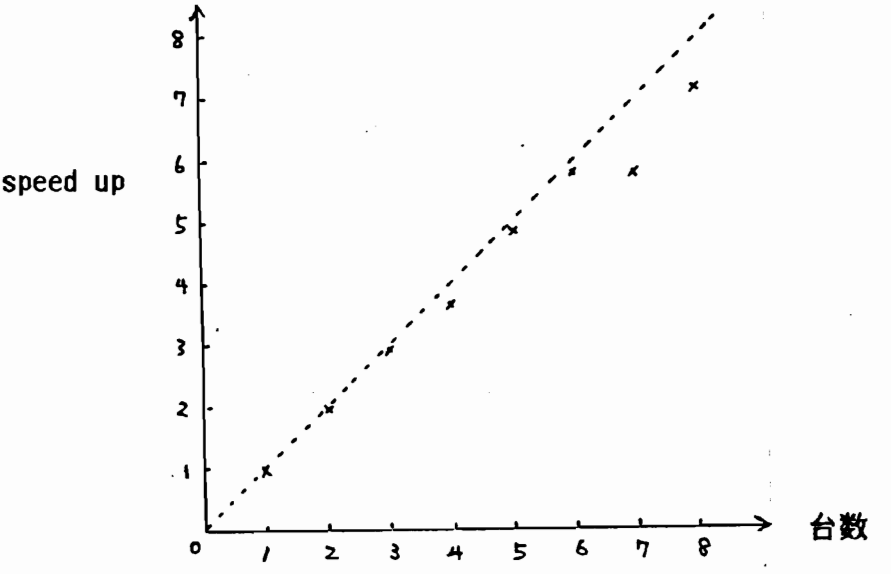


図3.13　行列積の実行結果

3.4 連立一次方程式の並列計算[34)]

ここでは、ガウス消去法および共役勾配法を用いて連立方程式を解く問題を取り上げ、ブロードキャストメモリ結合形並列計算機による並列処理方法を示す。対象の連立一次方程式は Ａｘ＝Ｂ(Ａはｎ×ｎ、Ｂとｘは ｎ×l のマトリクス）で表現されるものとする。この形は、 同じ係数行列を持つ l 組の連立方程式を同時に解く場合に現れる。 なおＡのij要素を a_{ij} 、Ｂのst要素を便宜上 $a_{8\ n+t}$ と表現する。

3.4.1 ガウス消去法

(1) 計算方法

ガウス消去には前進消去と後退代入の手順がある。

前進消去の手順は i,j,k を整数変数として、

```
for k:=1 to n-1 do
  begin
    for j:=k to n+l do
      akj:=akj/akk;
    for i:=k+1 to n do
      for j:=k to n+l do        ]*
        aij:=aij-aik*akj;
  end
```

となる。

前進消去では1行の消去 (* の部分)を1つのプロセスと考え、行単位での並列性を考える。前進消去の過程をペトリネットで表現すると図3.14 のようになる。

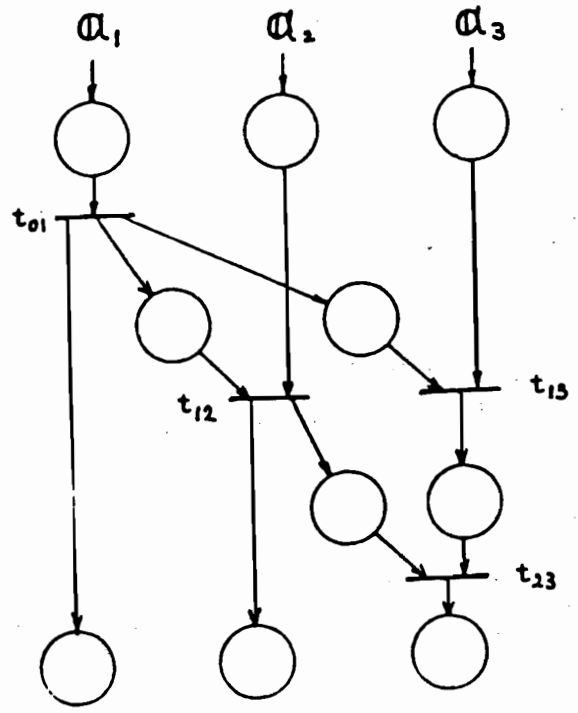


図3.14　前進消去の実行過程

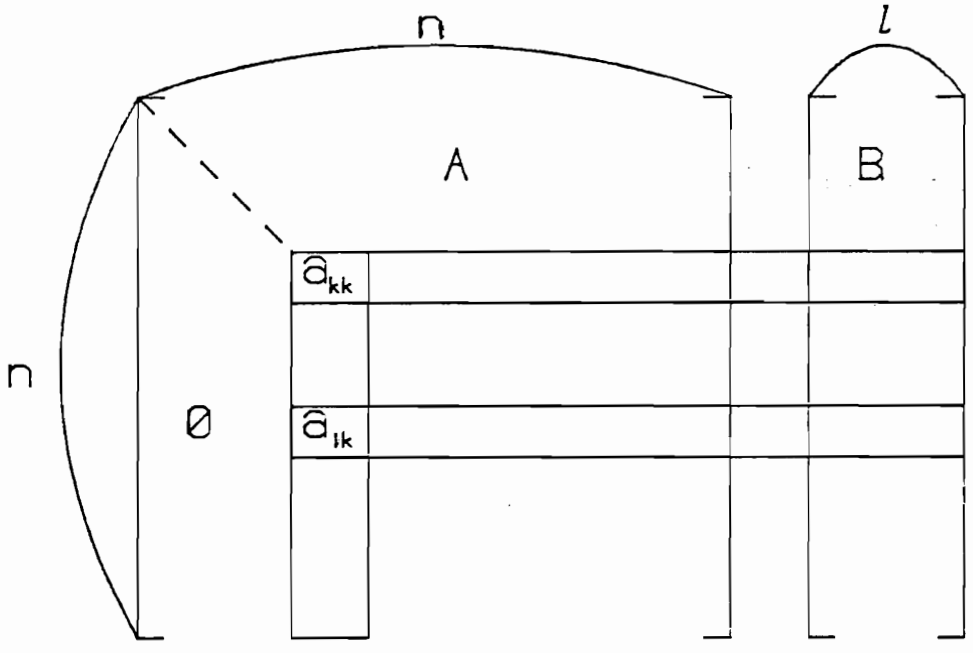


図3.15　ガウス消去の前進消去(a_k行によるa_i行の計算)

$t_{i,j}$は第i行をピボットとして第j行の消去を行う動作を表し、a_iは第i行を表す(図3.15)。図3.14 において、垂直方向のアークはその行での計算を表し、ななめ方向のアークはピボット行の転送を表す。$t_{i,j}$が発火可能となる条件は、$t_{i-1,\ i}$ および$t_{i-1,\ j}$が終わっていることである。行ごとにプロセッサを割り当て、i の昇順に計算させることにすれば、$t_{i,j}$の実行は他のプロセッサの $t_{i-1,\ i}$の終了とその結果の転送によって同期をとればよいことになる。

一方、 後退代入の手順は係数行列の対角要素がすべて1となっているから、すべての t に対して

$$x_{n,t} := a_{n,\ n+t}$$

とし、i=n-1 から 1 まで i を 1 ずつ減じながら

$$x_{i,t} := a_{i,\ n+t} - \sum_{j=i+1}^{n} a_{i,j} x_{j,t}$$

を計算することになる。プログラムの形で書くと、

```
for i:=1 to n do
   for t:=1 to l do
      x[i,t]:=a[i,n+t];
for i:=n-1 downto 1 do
   for j:=i+1 to n do
      for t:=1 to l do                ]**
         x[i,t]:=x[i,t]-a[i,j]*x[j,t];
```

となる。後退代入では ** の計算を実行の単位と考えて並列処理を行う。

前進消去と後退代入の計算時間はそれぞれ $O(n^3)$ と $O(n^2)$ であり、全計算時間の大部分は前進消去に費やされる。

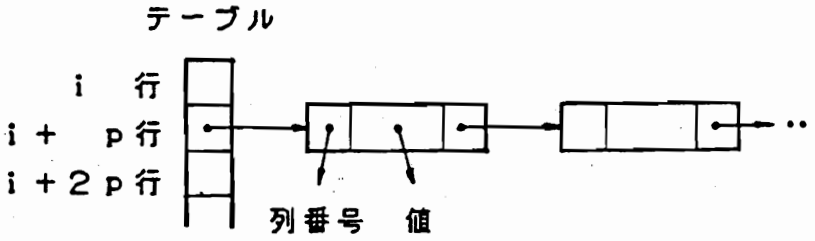
以下に、試作機上へのガウス消去法の実装方法を述べる。ここでは、係数行列Ａを対称スパース行列とし、l=1（Ｂをベクトルｂ）として話を進める。

データ構造

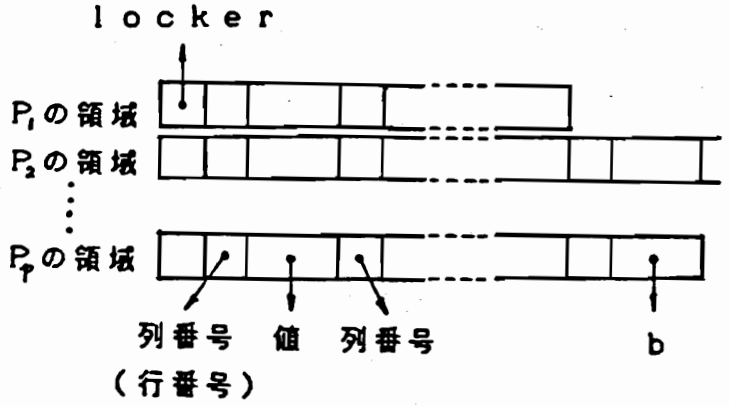
係数行列Ａとベクトルｂは行単位に分割し、 shared 変数として各データメモリに台数おきに割り当てていく。ゼロ要素は記憶せず、非ゼロ要素のみをその列番号とともにリスト構造にして格納する。i 番目のプロセッサが持つデータを図3.16-a に示す。

Ａは対称行列であれば上三角部分のみ記憶すればよく、各行の先頭は常に対角要素となる。消去によっていらなくなった要素はリストから取り去り空領域に戻す。また、新しく非ゼロ要素が生じた場合はリストの中に組み込む。

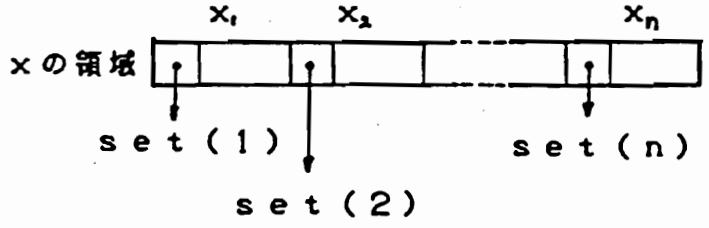
ピボット行はブロードキャストメモリに格納する(図3.16-b)。前述の前進消去アルゴリズムにおいて i に対するループ毎に全プロセッサで一斉同期をとるようにすれば、ピボット行領域は 1行分でよいが、 ここでは図3.14 で示した実行過程の持つ並列性をできるだけ引き出せるように、次のような方式を提案する。すなわち、ブロードキャストメモリ上にはプロセッサ毎にピボット領域を用意し、プロセッサ台数分のピボット行が同時に存在しうるようにしておく。各ピボット行は登録と同時に locker と呼ぶカウンタを pnum(プロセッサ台数)にセットする。 各プロセッサはあるピボット行が必要でなくなれば、その locker より1を引く。したがって locker=0はそのピボット行がいらなくなったことを意味し、その領域への新しいピボット行の書き込みが許される。



(a) 係数行列(DM)



(b) ピボット行(BCM)



(c) 解行列(BCM)

図3.16 ガウス消去法のデータ構造

ピボット行は非ゼロ要素のみを列番号と共に並べていく。ピボット行の先頭は対角要素であるため、その列番号は行番号でもあり、現在何行目がピボットになっているかを知ることができる。

また、解行列xは後退代入時に全プロセッサで必要となるため、ブロードキャストメモリ上にとる(図3.16-c)。ここで、set はsend - receive 命令のためのフラグである。

アルゴリズム

i) 前進消去

1. i=1 とする。PU_0は第1行をピボット行としてブロードキャストメモリに転送。
2. ピボット行i がブロードキャストメモリにあるかどうかを調べ、なければ送られてくるまで待つ。
3. i より大きい最初の行を j とする。
4. i 行を使って j 行の消去を行う。
5. 自分の担当している行のうちの次のピボット行候補が次の二条件を満たしておれば、これを新しいピボット行としてブロードキャストメモリに転送する。条件を満たさなければ 6 へ。
   i) 自分のピボット領域の locker が 0 である。
   ii) その行に対する消去が終了している。
6. j=j+pnum とする。j≦n ならば 4 へ。
7. ピボット行 i の locker から 1 を引く。
8. i=i+1 とする。自分の担当している最大行よりも i が小さければ 2 へ。

9. おわり。

ii) 後退代入

1. 自分の担当している最大の行番号を i とする。i=n ならば x_n $=b_n$を実行して x_nの set を1にする。
2. j=n→i+1 に対して s=s+a_{ij}*x_j を実行していく。もし、x_jがセットされていなければそこで待つ。
3. $x_i=b_i-s$ を実行し、set を1にする。
4. i=i-pnum として、i>0 ならば 2 へ。
5. おわり。

(2) 実行結果と考察

上記の計算方法による並列計算の結果を以下に示す。例題として図3.17-a の片支持ばりの曲げ問題を有限要素法で解く時に現れる

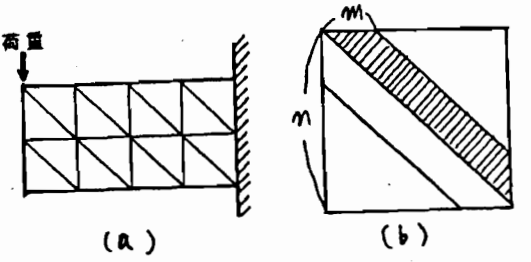


図3.17 例題

連立方程式をとりあげる。 係数行列は図3.17-b のような帯対称行列となるため、計算はこの図の斜線部のみに対して行えばよい。ここでは、次の3つの場合について実行を行う。

例1. n= 60,m=14	非ゼロ要素数	259
例2. n=462,m=24	〃	1770
例3. n=378,m=44	〃	2334

図3.18 は実行結果であり、 前進消去でのプロセッサの増加に対する速度向上のようすを示している。ここで、シミュレーションとあるのは、FORTRAN 上に作成された、試作機と同じ動作をする仮想マシンによる実行結果を示している。

図3.18 における速度向上の鈍化の原因として、次の2点を考えることができる。

i) アクセス競合(ハードウエア上の問題)

ii)同期待ち(アルゴリズム上の問題)

まず、i) については、プログラムの解析とシミュレーションから

a) 1台のプロセッサがバスを使用する時間は 0.5% 程度である。

b) アクセス競合による待ち時間は最大でも 0.2% 程度である。

という事実が得られた。 このことから、20～30 台程度のシステムではアクセス競合はほとんど起らず、速度向上の鈍化にはほとんど影響していないことがわかる。この事実は、ブロードキャストメモリがバスアクセス競合の低減に対して効果的であることを示している。

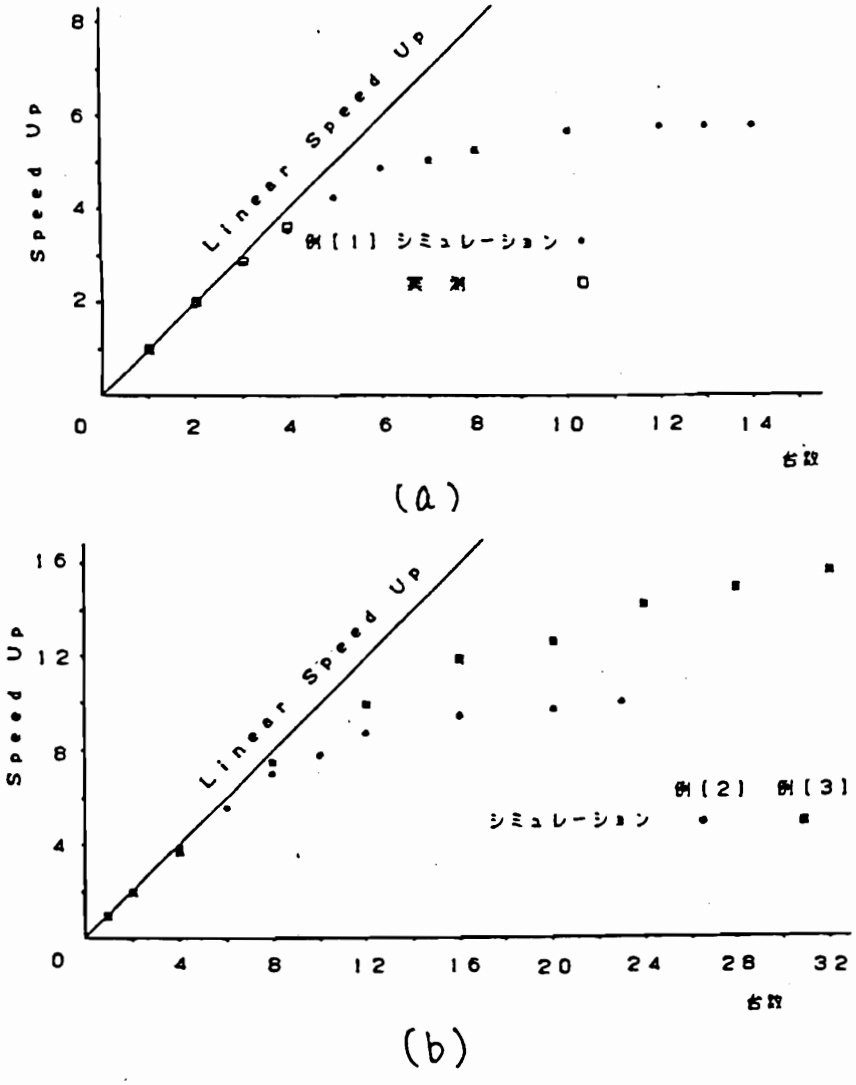


図3.18　前進消去における速度向上

一方、ii）については、シミュレーションでの動作追跡から、効率の低下はピボット待ち(前進消去におけるステップ2 の部分)によるものであることが確認できた。この原因である非ゼロ要素の不均衡は、ピボット行を複数同時に存在させることによってある程度吸収されるが、台数の増加にともなってこの不均衡が吸収しきれなくなってくる。そして、遅れているプロセッサが他を待たせることになる。

3.4.2 共役勾配法

(1) 計算方法

共役勾配法は連立方程式の反復解法の1つであり、 次の操作を行う。

1. 初期値ベクトル $x^{(1)}$を決め、$p^{(1)}=r^{(1)}=b-Ax^{(1)}$とする。
2. 次の操作を反復する。

$$y^{(k)}=Ap^{(k)}$$
$$\gamma^{(k)}=(p^{(k)},\ y^{(k)})$$
$$\alpha^{(k)}=(p^{(k)},\ r^{(k)})\ /\ \gamma^{(k)}$$
$$x^{(k+1)}=x^{(k)}+\alpha^{(k)}\cdot p^{(k)}$$
$$r^{(k+1)}=r^{(k)}-\alpha^{(k)}\cdot y^{(k)}$$
$$\beta^{(k)}=-(r^{(k+1)}\cdot y^{(k)})\ /\ \gamma^{(k)}$$
$$p^{(k+1)}=r^{(k+1)}+\beta^{(k)}\cdot p^{(k)}$$

ここで、$\alpha^{(k)}$および$\beta^{(k)}$はスカラー量であり、これらが求まるまで次のステップに進めないことから、$\alpha^{(k)}$、$\beta^{(k)}$に対して全プロ

セッサに同期がかかることになる。

計算の流れは図3.19 のようになる。ここで、t_α、t_β はそれぞれ$\alpha^{(k)}$、$\beta^{(k)}$を求める計算を表している。

図3.20 にデータ構造を示す。ここで、$\delta^{(k)}=(p^{(k)}, r^{(k)})$ $\varepsilon^{(k)}=(r^{(k+1)}, y^{(k)})$ であり、これらはα、βの計算に必要な中間結果である。

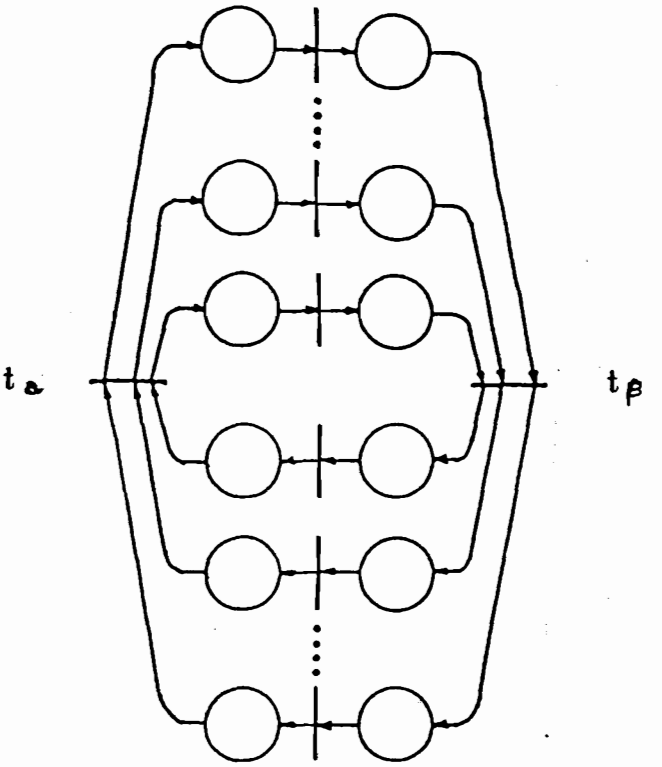
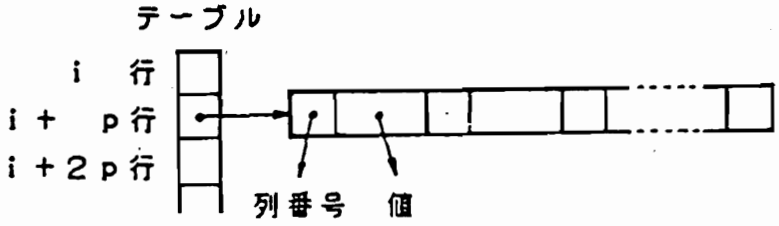
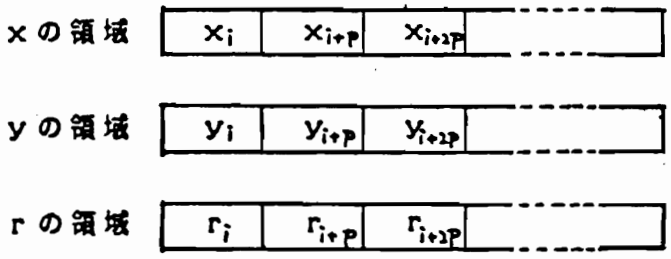


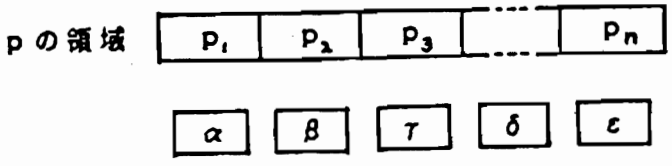
図3.19　共役勾配法の実行過程



(a) 係数行列(DM)



(b) x, y, r (DM)



(c) p および各スカラ(BCM)

図3.20 共役勾配法のデータ構造

(a) 行列Aは行単位に分割し、各データメモリにプロセッサの台数おきに分配する。ガウス消去法のように新しく非ゼロ要素ができることはないので、リスト構造にする必要はなく、非ゼロ要素のみを順に並べていく。

(b) ベクトルx、y、rは局所性を持つので、要素ごとに分割して各データメモリに割り当てる。

(c) ベクトルpはy＝Apの計算において全プロセッサから参照されるので、ブロードキャストメモリ上に置く。また各スカラー量もブロードキャストメモリ上に置く。

計算方法としては、各プロセッサ毎に前述の反復計算式に従って、行列・ベクトル積、ベクトル・スカラー積、および内積計算を行えばよい。この際、行列・ベクトル積とベクトル・スカラー積とは各プロセッサでまったく独立に計算できる。同期をとる必要があるのは次の場所である。

i) $\gamma^{(k)}$、$\delta^{(k)}$、$\varepsilon^{(k)}$の計算(ただし、全プロセッサで一斉に同期をとる必要はない)。

ii) $\alpha^{(k)}$、$\beta^{(k)}$の計算(全プロセッサでの一斉同期)。

i)はベクトルの内積計算の部分であり、各プロセッサ内で部分計算された内積の和をとるのに同期が必要となる。

(2) 結果と考察

ガウス消去法の時に用いた例題のうち例1 と例2 に対する、プロ

セッサ台数と計算速度の関係を図3.21 に示す。 ガウス消去法の場合と同じく、速度向上の鈍化の原因はソフトウエア上の同期待ちによるものであるが、共役勾配法では特に内積計算の部分での同期待ちが主な原因となっている。これは、内積計算では最終的に全プロセッサの和を取らなければならず、どうしても直列的な部分が多くなってしまうためである。

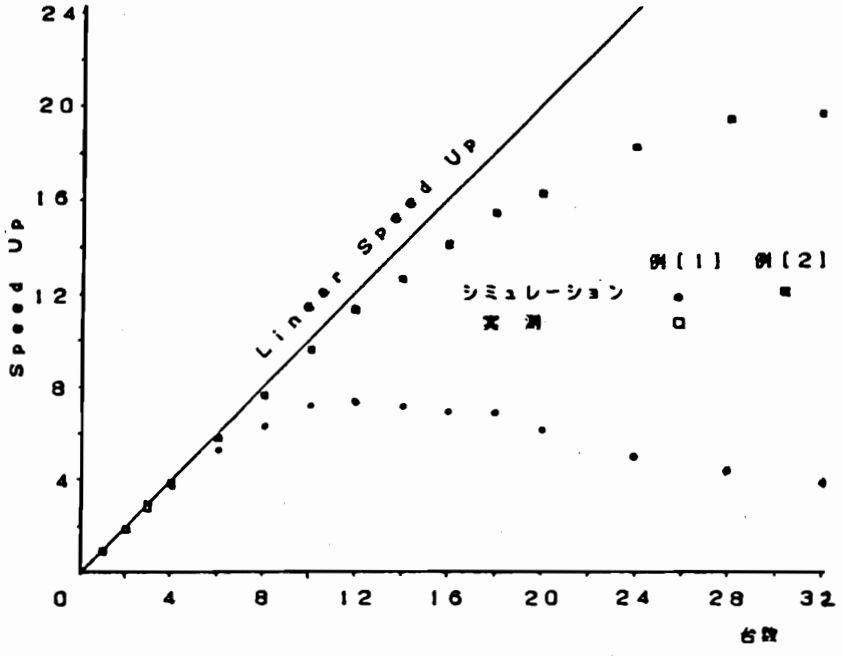


図3.21　共役勾配法における速度向上

また、このことと関連するが、例1 では12台あたりから計算速度が逆に低下している。これは、問題の規模に対して台数が増加すると、並列計算のためのオーバーヘッド（同期などの操作に要する時間)の割合が本来の計算時間に対して大きくなるからである。 このことは、MIMD型システムにおいて処理を細かく分割しすぎるとオーバーヘッドのためかえって効率が悪くなる危険性があることを示している。

ガウス消去法についても言えることであるが、ここで示した計算法は問題の規模が大きい場合に効果がある。問題の規模に対してプロセッサ数が十分に大きい場合には、システム全体で自動的に同期のとられる SIMD指向の制御方式をとるほうがよい。

3.5 結言

本章ではブロードキャストメモリ結合形並列計算機システムの試作によって、緒論であげた並列計算機システムの持つ諸問題に対する1つの解答を与えた。 本章で得た結果をまとめると次のようになる。

(1) 連立方程式の並列計算結果で示したように、ブロードキャストメモリはバス結合方式の問題点であるアクセス競合に対して効果的である。 シミュレーションなどから判断して、1本のバスから成る単純なシステムでも、数十台程度までのシステムには対応可能である。

(2) 試作機で提案したバス制御方法は、市販のメモリボードを用いてブロードキャストメモリを容易に実現できるため、実現性が高く、量産による価格の低下も期待できる。

(3) 数値計算プロセッサの採用によって、マイクロコンピュータレベルにおいても、かなりの高速数値計算が可能であることが確認できた。

(4) 試作機上に開発した並列プログラミング言語に関しては、約 1 年間の使用経験と実行結果から総合して、数値計算分野での並列プログラムの記述性と実行効率に対する要求をほぼ満足するものと判断する。

(5) 連立方程式計算に対して、ここで提案した並列計算アルゴリズムは、

i) 各プロセッサにデータをあらかじめ分配しておき、

ii) 同期をとりながら、自分の分担に対して直列的に計算を進める。

ことを基本にしたものであり、既存のアルゴリズムから比較的容易に並列計算の効果を得ることができた。この手法は他の数値計算に対しても適用できる。

(6) (5)と関連するが、 問題の規模に対してプロセッサ数が十分多い場合には、並列計算の方法(特に、同期の方法)を変える必要がある。

第４章

マトリクスブロードキャストメモリ結合形多重プロセッサシステム

4.1 緒言

前章で述べた試作機は、ブロードキャストメモリを用いた1次元構成の並列計算機であった。ここでは、ブロードキャストメモリを用いて2次元マトリクス状に結合された計算機群を提案し、これによって並列計算を行うことにより、ガウス消去法および修正コレスキー法のいずれのアルゴリズムを用いても計算時間を O(n)とすることが可能なことを示す。

次に、これらの並列計算方式に若干の拡張を加えることにより軸選択操作(ピボッティング)を消去計算過程に組み込むことが可能なことを示す。この方法により、ガウス消去法に軸列上での軸選択を導入した場合でも、その計算時間を$O(n \log_2 n)$とすることが可能なことを示す。

4.2 マトリクスブロードキャストメモリ結合形並列計算機

提案している並列計算機システムは、多数台のプロセッサをブロードキャストメモリによって 2次元マトリクス上に結合したシステムで、 p×q 台のプロセッサから成る多重プロセッサシステムである(図4.1)。

各演算プロセッサにはローカルメモリが付加されており、それぞ

れにローカルなデータとプログラムが格納されている。ブロードキャストメモリは図のように、行方向のものが p組と列方向のものがq組用意される。 以後、i行j列の演算プロセッサ(以後 PE と呼ぶ)をP_{ij}で示し、 i行の PEアレイ(P_{i1}〜P_{iq})を行PEアレイ P_iと呼び、j列のPEアレイ(P_{1j}〜P_{pj}) を列PEアレイ P_jと呼ぶことにする。 図4.1 において、 P_{ij}が上側のポートに出力したデータは行PEアレイP_iにブロードキャスティングされ、右側のポートに出力したデータは列PEアレイにブロードキャスティングされる。

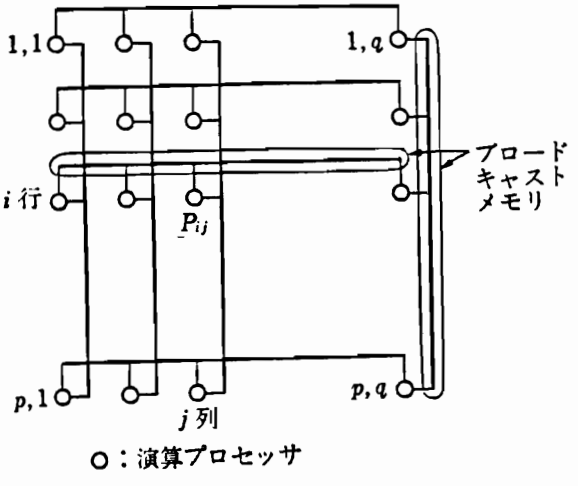


図4.1 マトリクスブロードキャストメモリ結合形並列計算機

4.3 連立一次方程式の並列計算

計算アルゴリズムとして、直接法のアルゴリズムであるガウス消去法と修正コレスキー法を取り上げる。両者とも図4.1 のシステムで O(n)時間で計算可能である。対象の n元連立一次方程式は 3.4節で用いたものである。

4.3.1 ガウス消去法

ガウス消去法の計算手順は 3.4.1節で述べたものである。 以下、PEマトリクスの大きさと係数行列の規模との関係による2つの場合について考察する。

PEマトリクスが係数行列よりも大きい場合

まず、前進消去の並列計算方法を示す。

前進消去により k-1 列まで消去が進んだ時点の係数行列の状態を図4.2 に示す。 $\{a_{ij}\}(1 \leqq i \leqq k-1, i \leqq j \leqq n+l)$ は消去操作によって確定した係数行列と定数項行列の部分行列である。$\{a_{ij}\}(k \leqq i \leqq n, k \leqq j \leqq n+l)$ は消去操作の完了していない部分行列である。$n \leqq p$, $n+l \leqq q$ とし、a_{ij}を p_{ij}に割り当てることにすると、 第k行をピボット行とする消去を進める場合の消去作業の並列化は以下のようになる。a_{kk}は p_{kk} のローカルメモリに存在するので消去手順は以下のようになる。

1. a_{kk}～a_{nk}の値を p_{kk}～p_{nk} が平行して行方向にブロードキャスティングする。
2. $p_{k\ k+1}$～$p_{k\ n+l}$ が並行して

$a_{kj}:=a_{kj}/a_{kk}$ $(k \leqq j \leqq n+l)$

を計算する。

3. a_{kk}～$a_{k\ k+l}$の値を p_{kk}～$p_{k\ n+l}$ が並行して列方向にブロードキャスティングする。

4. 各 $p_{ij}(k+1 \leqq i \leqq n, k \leqq j \leqq n+l)$ が並行して

$a_{ij}:=a_{ij}-a_{ik} \cdot a_{kj}$

の計算を行う。

k=1 の時の、各ステップの実行のようすを図4.3(1)～(4) に示す。

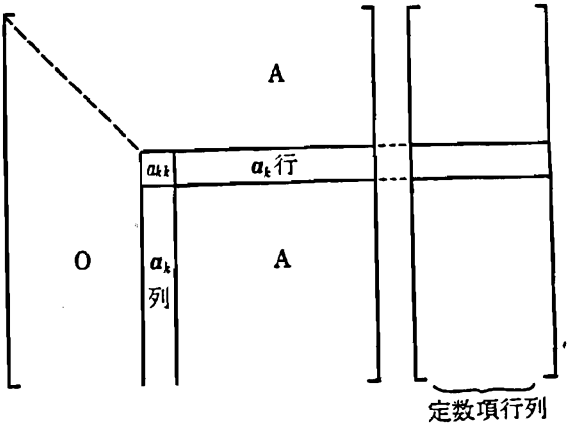


図4.2 ガウス消去の前進消去過程

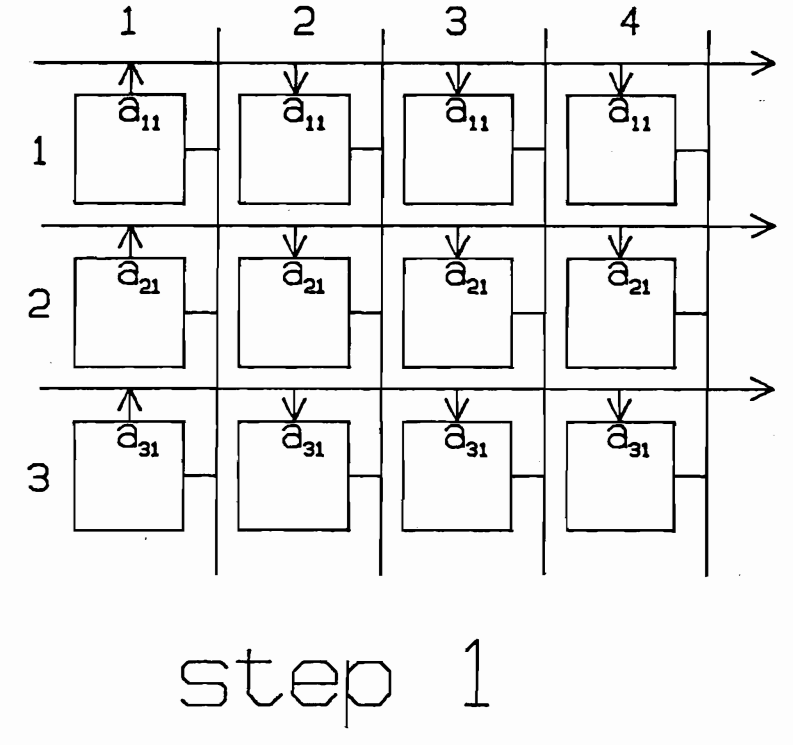
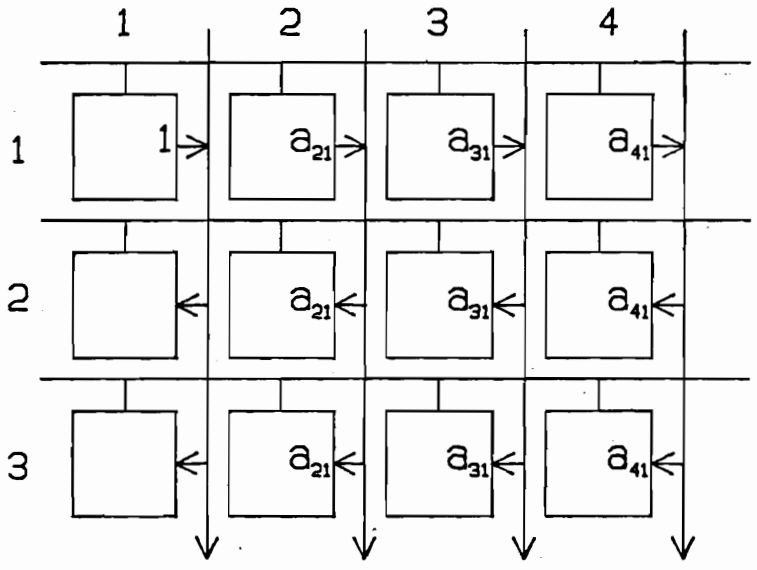
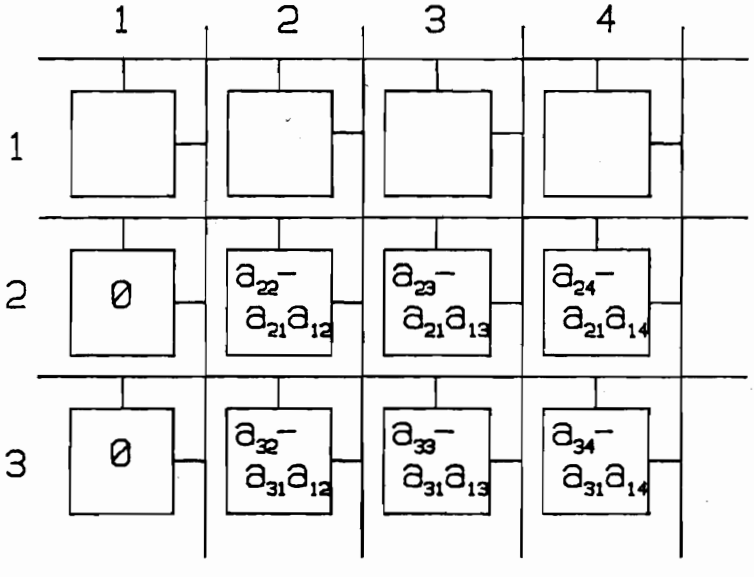


図4.3　前進消去の実行ステップ


1
2
3
4
1
2
3
1
a_{21}
a_{31}
a_{41}

step 3


1
2
3
4
1
2
3
0
$a_{22}-a_{21}a_{12}$
$a_{23}-a_{21}a_{13}$
$a_{24}-a_{21}a_{14}$
0
$a_{32}-a_{31}a_{12}$
$a_{33}-a_{31}a_{13}$
$a_{34}-a_{31}a_{14}$

step 4

以上の四つのステップを k=1～n-1 に対して順次行っていけば前進消去が完了する。 各ステップに要する時間は n と独立に一定であり、 かつステップ1、3 におけるブロードキャストメモリの書き込みにおいて競合が発生していないから計算時間はO(n)となる。

次に、 後退代入では $x_{n\ t}, x_{n-1\ t}, \cdots, x_{1\ t}$ と順次値を計算していくことになるが、$x_{i\ t}(1 \leq t \leq l)$ の値は t に関して互いに独立なので t を固定して考える。解ベクトル x_t の計算は以下のステップを k=n～1 に対して k を 1 ずつ減じながら実行していくことにより計算される(図4.4)。

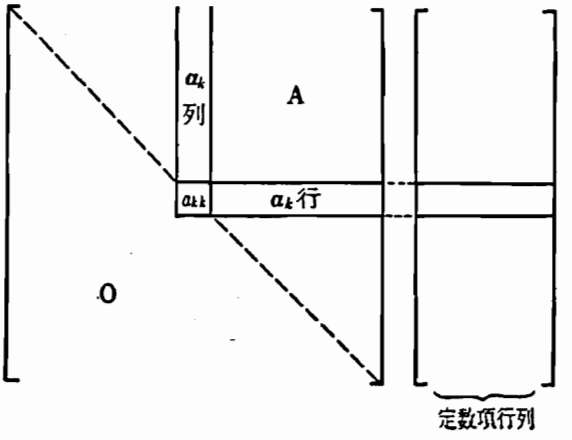


図4.4　後退代入の手順

1. 列PEアレイ p_kが平行して a_k列要素中の $a_{i,k}(1 \leqq i \leqq k-1)$ を行方向にブロードキャスティングする。
2. $p_{k\ n+t}$が

   $x_{kt} := a_{kn+t}$

   としその値を列方向にブロードキャスティングする。
3. 列PEアレイ p_{n+t}が並行して

   $a_{i\ n+t} := a_{i\ n+t} - a_{ik} \cdot x_{kt}$ $(1 \leqq i \leqq k-1)$

   の計算を行う。

l=1、t=1、k=3 での各ステップの動作を図4.5(1)～(3) に示す。

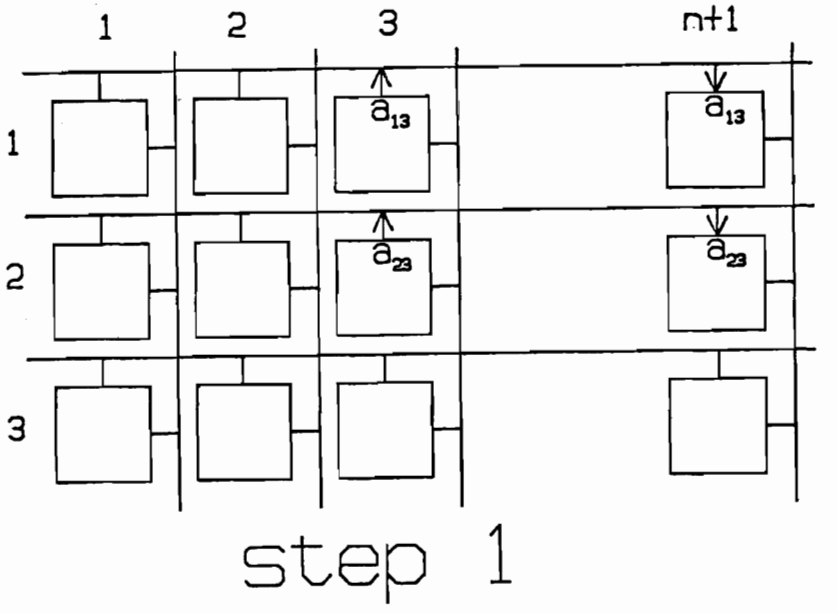
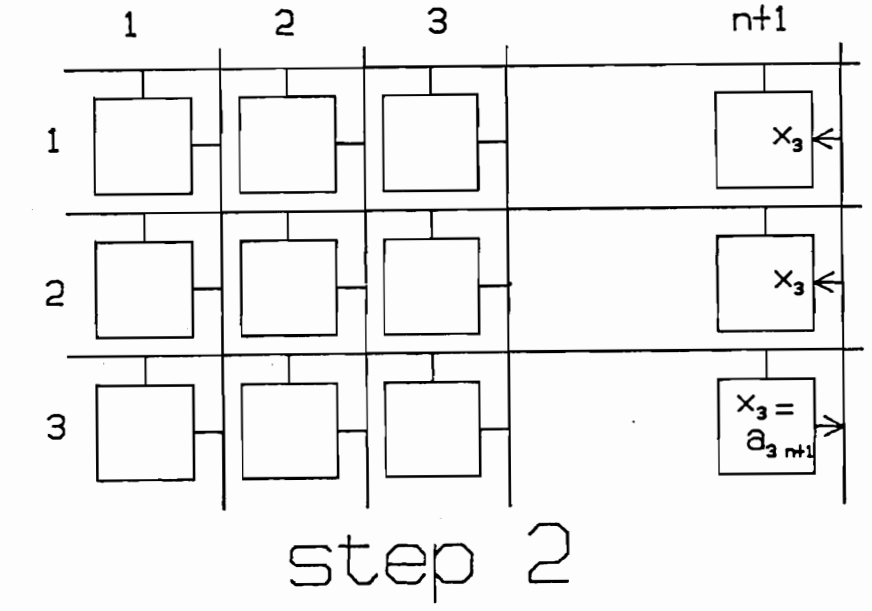
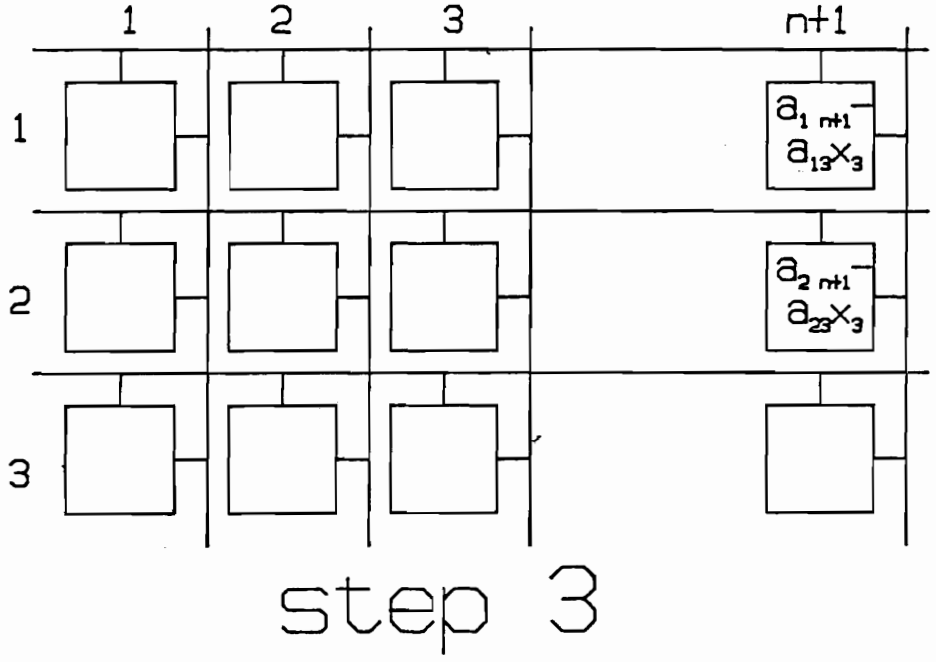


図4.5　後退代入の実行ステップ



step 2



step 3

いずれのステップでもブロードキャストメモリへの書き込みの競合は発生していない。ステップ2、3 に着目すると列PEアレイ p_{n+t} のみ計算を担当している。

したがって $1 \leqq t \leqq l$ に関して列PEアレイ群 $p_{n+1} \sim p_{n+l}$ が並行して計算をすすめることができその計算時間はO(n)となる。

<u>$p<n$、$q<n+l$ の場合</u>

一般に大型連立一次方程式の場合には $p<n$、$q<n+l$ となるのが普通である。この場合には各PEは複数要素の計算を引き受けて行うことになる。この場合、各PEの負荷のバランスを考え要素計算の割り当てを工夫し $a_{i,j}$ は $p_{(i-1) \bmod p+1,\ (j-1) \bmod q+1}$ に担当させるようにする。データの割り当てのようすを図4.6 に示す。

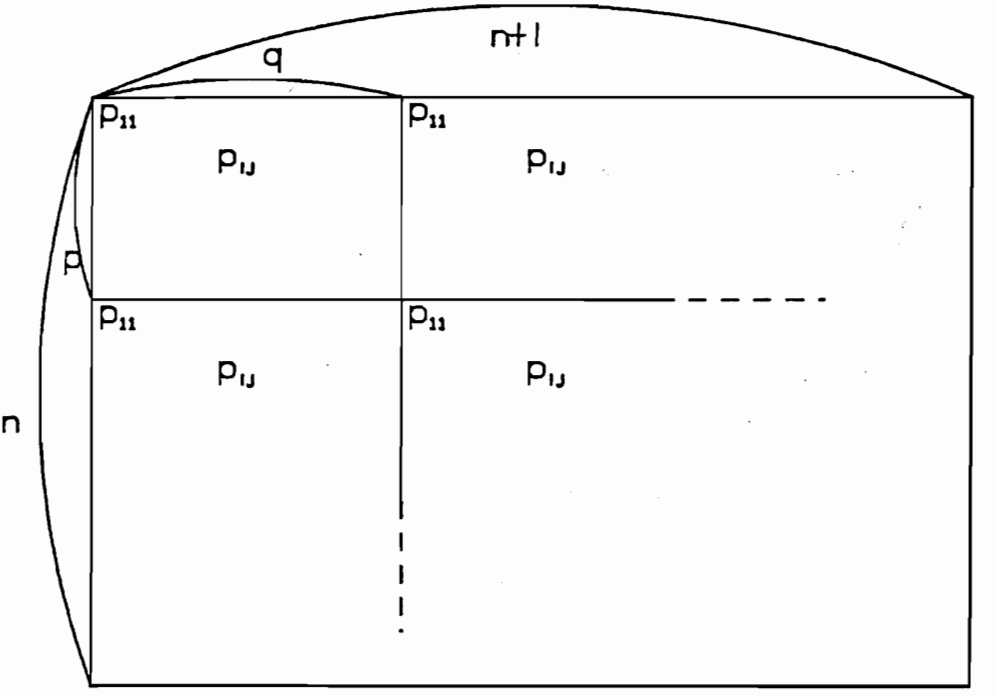


図4.6　行列要素の割り当て

i) 前進消去のステップ

第 k-1 行まで消去が進み、第 k 行をピボット行として消去を進める場合を考える。a_k列要素は列PEアレイ $p_{(k-1) \bmod q+1}$に存在するので消去手順は以下のようになる。

1. a_k列の a_{kk}～a_{nk}の値を PEアレイ $p_{(k-1) \bmod q+1}$が並行して行方向にブロードキャスティングする。
2. 行PEアレイ $p_{(k-1) \bmod p+1}$ が並行して $k \leq j \leq n+l$ なるすべての j について

   $a_{kj} := a_{kj} / a_{kk}$

   を計算する。
3. 新たに計算した $a_{kj}(k \leq j \leq n+l)$ の値を行PEアレイ $p_{(k-1) \bmod p+1}$が並行して列方向にブロードキャスティングする。
4. 全PEは並行して担当する全要素 $a_{ij}(k+1 \leq i \leq n, k \leq j \leq n+l)$ の更新を

   $a_{ij} := a_{ij} - a_{ik} \cdot a_{kj}$

   の計算によって行う。

以上の四つのステップを k=1～n-1 に対して順次行えば前進消去が完了する。計算時間は1台のプロセッサが約 $n(n+l)/pq$ 要素担当するので $O(n^2(n+l)/pq)$ となる。

ii) 後退代入のステップ

後退代入の手順についても、基本的には前述のものと同じになる。解ベクトル x_t の計算は k=n～1 に関して k の値を 1 ずつ減じな

がらの以下の計算となる。

1. 列PEアレイ $p_{(k-1) \bmod q+1}$ が並行して $a_{ik}(1 \leqq i \leqq k-1)$を行方向にブロードキャスティングする。
2. $p_{(k-1) \bmod p+1\ (n+t-1) \bmod q+1}$が

   $x_{kt} := a_{k\ n+t}$

   を求め値を列方向にブロードキャスティングする。
3. 列PEアレイ $p_{(n+t-1) \bmod q+1}$が並行して

   $a_{i\ n+t} := a_{i\ n+t} - a_{ik} \cdot x_{kt}$ $(1 \leqq i \leqq k-1)$

   の計算を行う。

いずれのステップもブロードキャストメモリへの書き込みの競合は発生しない。以上のステップは各 $t(1 \leqq t \leqq l)$ について並行に進めることができるから全体の計算時間は $l \leqq q$ の場合には $O(n^2/p)$時間、$l > q$ の場合には $O(n^2 l/pq)$時間となる。

4.3.2 修正コレスキー法

係数行列が正定値対称の場合に用いられる手法で、主要な計算は $Ax = B$ を変形しAを右上三角行列に変換する前進消去で、後退代入はガウス消去法の場合とほぼ同一の手順となる。

前進消去の手順（図4.7）は a_1行の値は変更せずに、$i=2 \sim n$ についてiを1ずつ増しながら $i \leqq j \leqq n+l$ なる全jについて、

$$a_{ij} := a_{ij} - \sum_{k=1}^{i-1} a_{ki} \cdot a_{kj} / a_{kk}$$

の値を計算することになる。

実際の計算はAと同一サイズの行列A'を導入し、

$a'_{1i} := a_{1i}/a_{11}$ $(1 \leqq i \leqq n+l)$

とし以下の計算を行う。

iを2からnまで1ずつ増しながら $i \leqq j \leqq n$ かつ $j \leqq k \leqq n+l$ なるすべての j、k に関して、

$a_{jk} := a_{jk} - a'_{i-1\ j} * a_{i-1\ k}$

$a'_{ij} := a_{ij}/a_{ii}$

を計算していくことになる。

プログラムの形で記述すると、

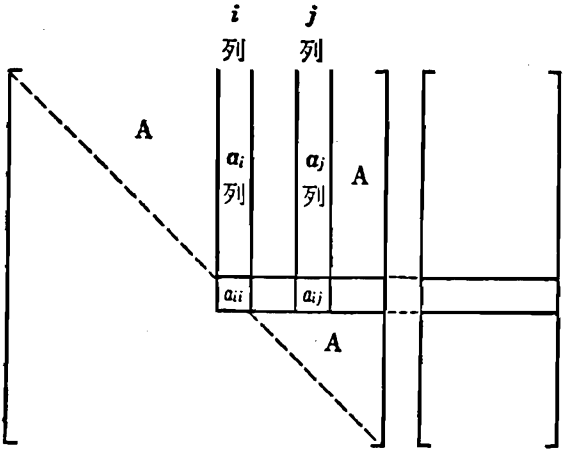


図4.7　修正コレスキー法における前進消去

```
for i:=1 to n+l do a'1,i:=a1,i/a1,1;
for i:=2 to n do
   for j:=i to n do begin
      for k:=j to n+l do
         aj,k :=aj,k-a'i-1,j*ai-1,k;
         a'i,j:=ai,j/ai,i
      end
```

となる。

p≧n、q≧n+l の場合

$a_{i,j}$、$a_{i,j}$を $p_{i,j}$のローカルメモリに割り当てておくと、以下のように並列計算可能となる。

i-1 列まで消去が進みi列の消去を行う計算は、

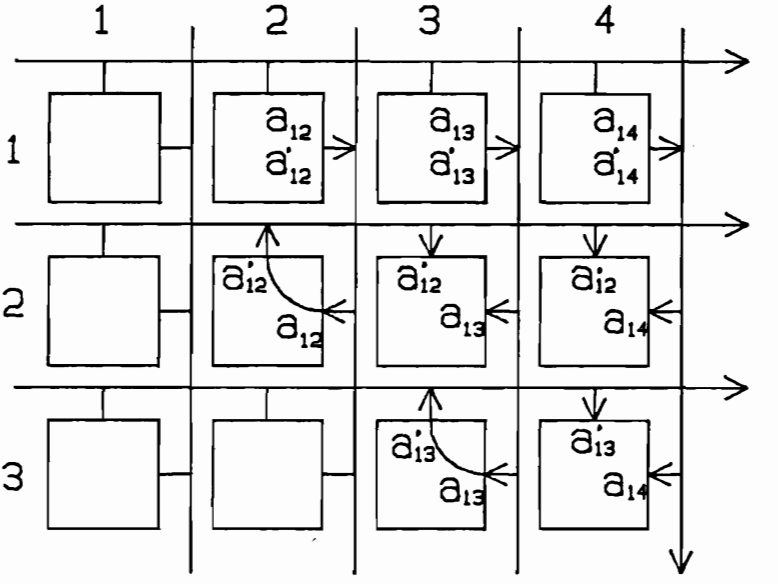
1. i-1 列の消去の段階で求められた a_{i-1} 行、 a'_{i-1} 行の値を $p_{i-1,i}$～$p_{i-1,n+l}$が並行して列方向にブロードキャスティングする。引き続きそれらデータを受けた $p_{kk}(i \leqq k \leqq n)$が$a'_{i-1,k}$を行方向にブロードキャスティングする。
2. ステップ1の操作により各 $p_{j,k}(i \leqq j \leqq n, j \leqq k \leqq n+l)$は $a'_{i-1,j}$と $a_{i-1,k}$がアクセス可能となるから、並行して

   $a_{j,k} := a_{j,k} - a'_{i-1,j} \cdot a_{i-1,k}$

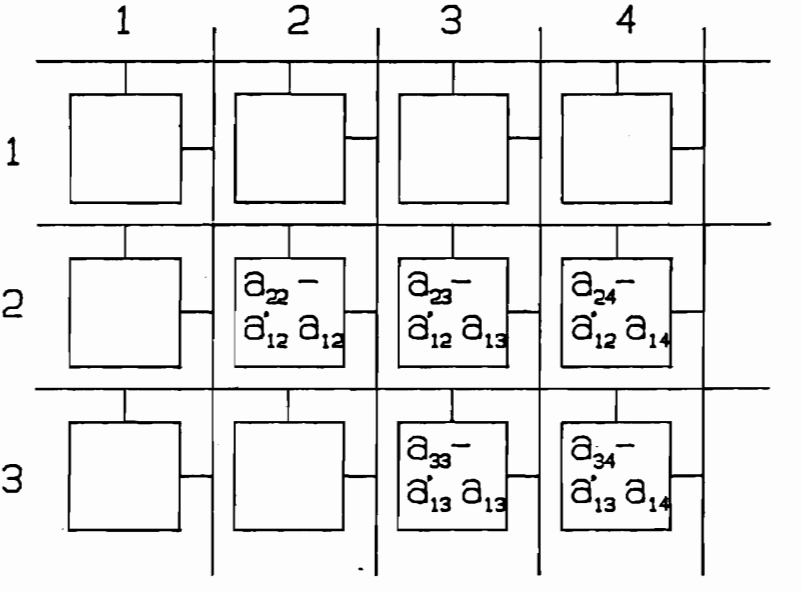
   の計算を行う。
3. $p_{i,i}$が $a_{i,i}$を行方向にブロードキャスティングする。
4. 行PEアレイ p_iが並行して、$i \leqq j \leqq n$ について

   $a'_{i,j} := a_{i,j}/a_{i,i}$

   を計算する。

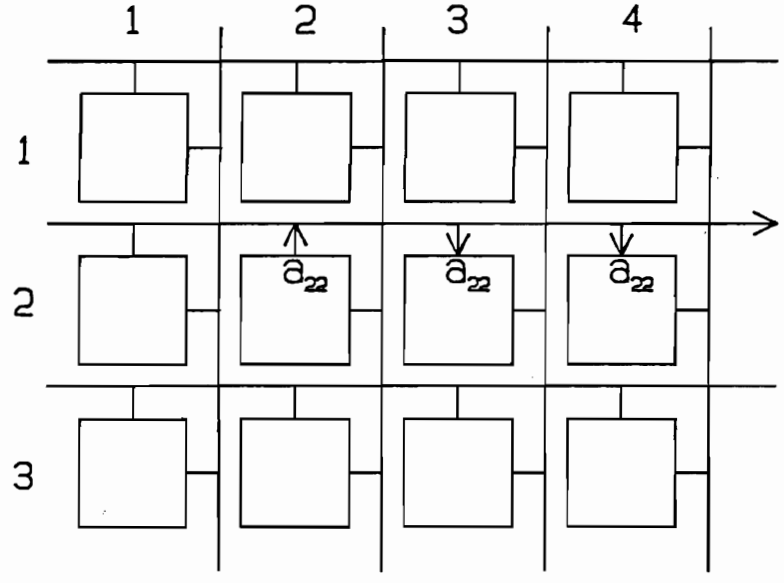


step 1

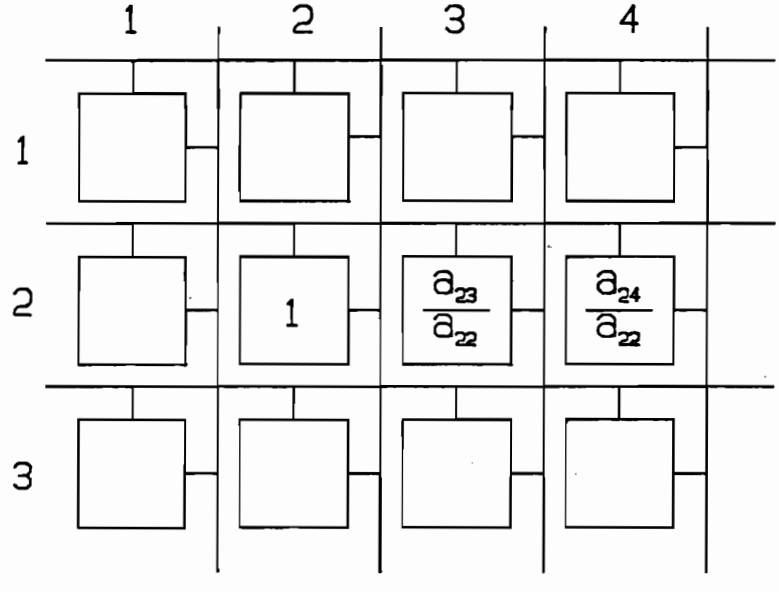


step 2

図4.8　修正コレスキー法の実行ステップ



step 3



step 4

k=1 の時の各ステップにおける実行のようすを図4.8(1)～(4) に示す。

ステップ1～4 のいずれの計算時間も n と独立で一定であり、ブロードキャストメモリへの書き込みの競合も発生しないから、全計算時間は O(n) となる。

<u>p<n、q<n+l の場合</u>

修正コレスキー法の場合もガウス消去法の場合と同様に並列計算が可能である。

いま i行まで前進消去が進んだとすると次のステップは以下のようになる。

1. i-1 列の消去の終了で求まった a_{i-1}行、a'_{i-1}行の値を行プロセッサアレイ $p_{(i-1) \bmod q+1}$が並行して列方向にブロードキャスティングし、引き続きそのデータを受けた対角プロセッサ群が a'_{i-1}行の値を行方向にブロードキャスティングする。この操作により a_{jk} の計算を行うプロセッサが直接 $a'_{i-1\ j}$ と $a_{i-1\ k}$とをアクセスできるようになる。
2. 各PEが担当する

   $$a_{jk} := a_{jk} - a'_{i-1\ j} \cdot a_{i-1\ k}$$

   の計算を $i \leqq j \leqq n, j \leqq k \leqq n+l$ なるすべての j,k について並列計算する。
3. $p_{(i-1) \bmod p+1\ (i-1) \bmod q+1}$により求められた a_{ii}を行方向にブロードキャスティングし、行PEアレイ $p_{(i-1) \bmod p+1}$の働きにより並行して $1 \leqq j \leqq n$ なるjについて

$a_{i,j}:=a_{i,j}/a_{i,i}$

を計算する。

計算時間は $O(n^2(n+l)/pq)$ となる。後退代入はガウス消去法の場合と同一手順となる。

4.3.3 帯行列と疎行列

帯行列

係数行列が帯形で半帯幅が対角要素を含んで m+1 であるとする。n>p,n+l>q とし、各PEへの割り当ては図4.6 に示した方式と同一と

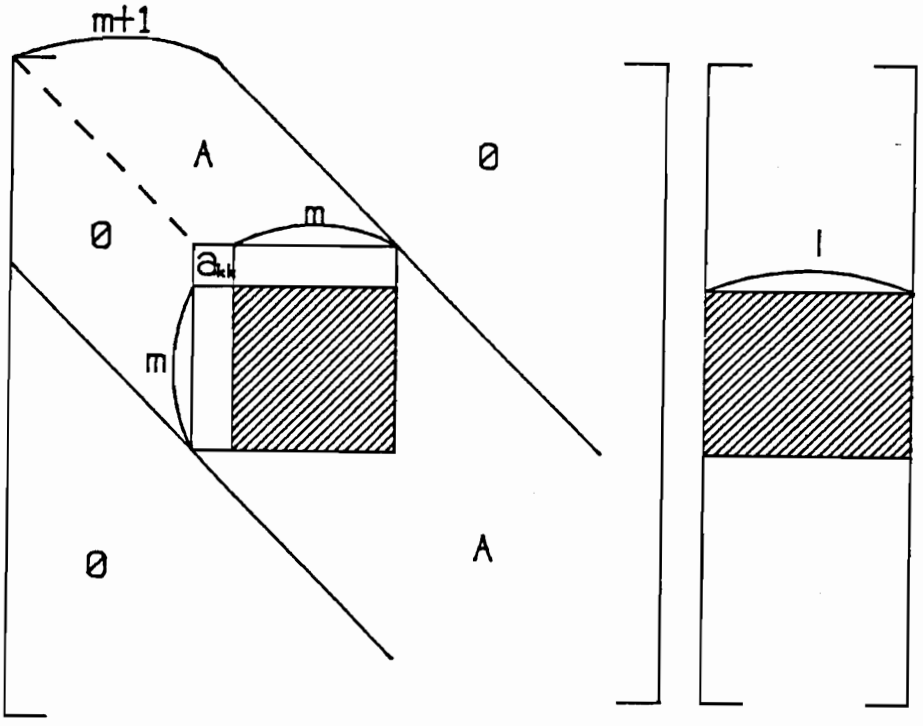


図4.9 帯行列に対する消去計算

する。ガウス消去法および修正コレスキー法の計算手順は、それぞれ 4.3.1節の後半と 4.3.2節の後半で示したステップと同一となる。しかしながら、計算過程において帯の外の要素が非ゼロになる可能性がないので帯外の部分の計算はすべて省略することができ、計算量は大幅に減少する。

半帯幅 m+1 の帯係数行列Aに対して、ガウス消去法の前進消去により k-1 列まで消去が進んだ時点のAの状態を図4.9 に示す。帯外の要素はすべてゼロであることから、第k行をピボット行とする消去計算は図4.9 の斜線部のみに対して行えばよい。そして、この計算によって帯外の要素が新しく非ゼロになることはない。

計算時間のオーダは $p \geq m$、$q \geq m+l$ のときには、前進消去、後退代入とも O(n) となる。また、$p \ll m$、$q \ll m+l$ のときには、前進消去は $O(nm(m+l)/pq)$ であり、後退代入は $O(nml/pq)$ となる。

疎行列の場合

大型の連立一次方程式がスパース係数行列をもっている場合、$n>p$、$n+l>q$ とし、各PEへの割り当ては図4.6 に示した方式と同一とする。

この場合、1次元のシステムに対して 3.4.1節で用いたリンクリスト表現を 2次元型システムに拡張すればよい。マトリクスブロードキャストメモリ結合形システムでは、各PEには非ゼロ要素のみ格納するようにし、割り当てられた非ゼロ要素は二つのポインタ要素と値自身を持ち、行方向および列方向のリンクリストの形で格納されているとする。消去過程で新しく生じた非ゼロ要素はリンクリストに挿入され、消去された要素の領域は自由リスト領域にもどされ

再利用される。この方式を採用することにより必要なメモリ容量は圧縮され、実際の計算も非ゼロ要素のみに行われることになる。

4.4 軸選択形ガウス消去法の並列計算

4.3 節で示したガウス消去法の並列計算法では消去の過程での軸選択(ピボッティング)を行っておらず、軸要素が 0 となったり、0 にきわめて近い値をもったときの対策が講じられていない問題点がある。

本節では、前節の計算方式に若干の拡張を加えることにより、軸選択操作を消去計算に組み込むことが可能なことを示す。

4.4.1 基本操作

軸列上での軸選択を行う場合に必要となる 2つの基本操作

i) i行と j列の入れ替え

ii) i列要素(a_{1i}～a_{pi})中の絶対値最大のものの探索

は以下の手順で並列計算可能である。

i) i行と j列の入れ替え

p_{i1}～p_{iq}が並行して列方向のバスを介して a_{i1}～a_{iq}を p_{j1}～p_{jq}に転送する。引き続き p_{j1}～p_{jq}が a_{j1}～a_{jq}を列方向のバスを介して p_{i1}～p_{iq}に転送する。以上の2回の転送により i行と j行の入れ替えが可能となる。このようすを図4.10 に示す。

ii) i列要素(a_{1i}～a_{pi})中の絶対値最大のものの探索

(a) i列要素 a_{1i}～a_{pi}を行方向バスを介して行方向に転送し、

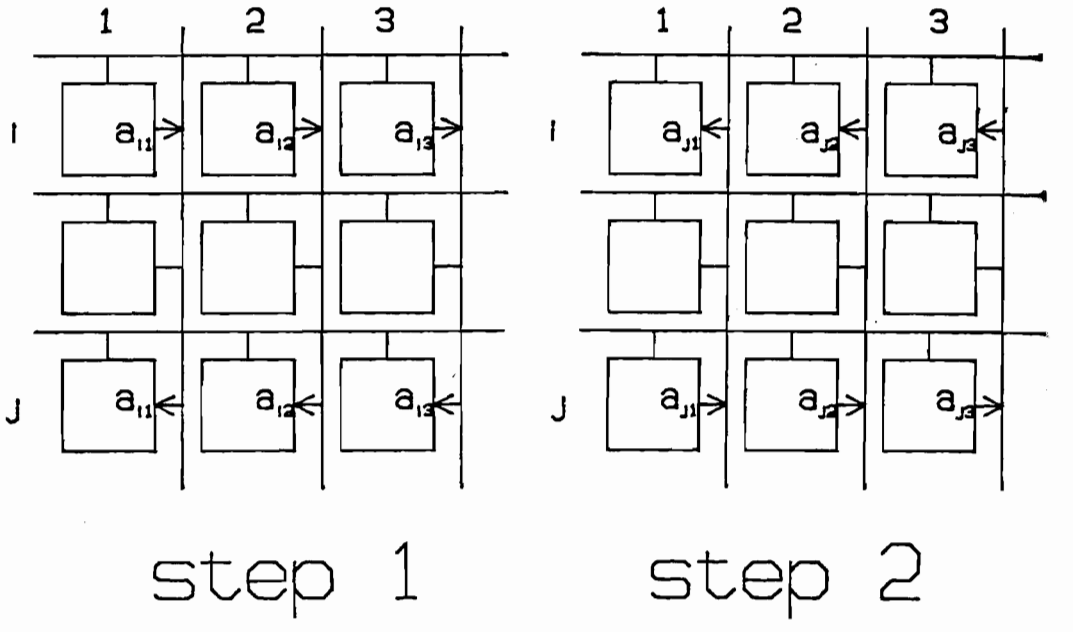


図4.10　行の入れ替え

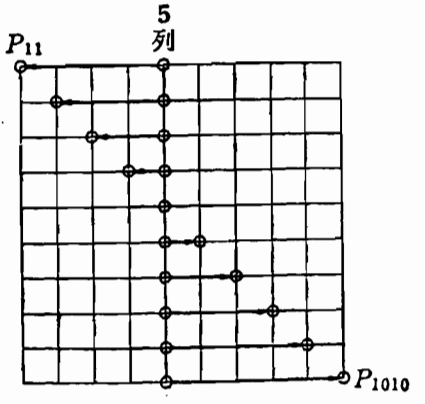


図4.11　対角プロセッサへのデータ転送

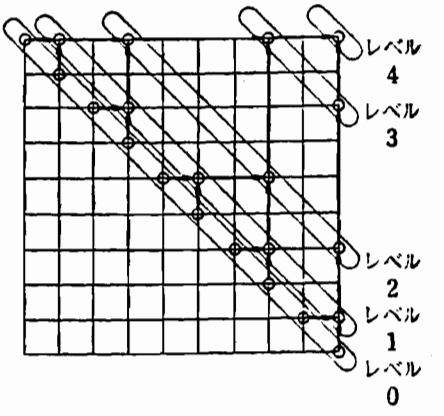


図4.12　2分サーチの手順

対角上のプロセッサ $p_{11}, p_{22}, \cdots, p_{pp}$ へ並行転送する(図4.11)。

(b)　2分探索法を用いて絶対値最大のものを探す。 対角プロセッサを葉ノードとする2分木を図4.12 のように構成し、各枝ノードと根ノードに対するプロセッサを定めそれぞれレベル0 のプロセッサ群、レベル1のプロセッサ群、…、レベルs (s=⌈ $\log_2 p$ ⌉)のプロセッサ群と呼ぶことにする。2分探索の手順は以下の手順となる。

1. レベル0 のプロセッサ群はそれぞれのもつ要素をレベル1 のプロセッサ群に、 行方向または列方向のバスを介して転送する。二つのデータが転送されてきたレベル1 のプロセッサは両データの比較を行い絶対値の大きいほうをレベル2 のプロセッサにバスを介して転送する。また一つのデータのみが転送されてきたプロセッサはその値自身をレベル2 のプロセッサにバスを介して転送する。

・・・・・・

i. レベルiのプロセッサ群はそれぞれレベル i-1 のプロセッサ群から送られてきたデータを受け取り、 2入力データを受け取ったプロセッサは絶対値の大きい方のデータを、 1入力データを受け取ったプロセッサはそのデータ自身を　i+1 レベルのプロセッサ群にバスを介して転送する。

・・・・・・

s. レベルsのプロセッサはレベル s-1 のプロセッサ群から送られてきた二つのデータを受け取り、そのうち絶対値の大きいほうのデータを選択する。

上述の計算手順を実行することにより、絶対値最大の要素を探索することが可能で、必要とする計算時間は $O(\log_2 p)$ となる。

4.4.2 軸選択形ガウス消去法の並列計算

ここでは軸選択に着目して、前進消去のアルゴリズムの並列化について論ずる。

4.3.1節の時と同様に、第 k-1 列までの前進消去が終わり第 k列上の軸選択を行う場合を考える(図4.2)。

軸選択は次の二つのステップで実現される。

i) k列要素 a_{kk}～a_{nk}のうちで絶対値最大の要素a_{jk}を見つける。
ii) k行と j行を入れ替える。

i) に示したステップの演算は n+1-k 個の要素から絶対値最大の要素を探索する演算となり、 前節で示した 2分探索法を用いると、$\lceil \log_2(n+1-k) \rceil$ 時間で実行できる。

ii)に示したステップは行の入れ替えの演算で、 前章 i) で示した手順によりバスを用いた 2回のデータ転送で実現をすることができる。

したがって、前進消去は

```
for k:=1 to n-1 do begin
    {(i)    k列要素 akk～ankのうち絶対値最大の要素 ajkを
            探索する};
    {(ii)   k行と j行の入れ替えを行う};
    {(iii)  k行を軸として k列の消去計算を進める};
```

end

となる。

(i)の計算は各 k の値について、 $\lceil \log_2(n+1-k) \rceil$ 時間必要とし、(ii)および(iii)は k の値にかかわらず一定時間で実行できる。したがって全体としての計算時間は $O(n \log_2 n)$ 時間で抑えられることになる。

4.5 結言

本章では、マトリクスブロードキャストメモリ結合形並列計算機システムを提案し、n元連立一次方程式を $O(n)$ 時間で解く可能性を示した。また、このシステムの構造上の特徴を有効に活用することにより、行選択と行入れ替えを伴うガウス消去計算を $O(n \log_2 n)$ 時間で計算できることを示した。

このシステムは、 第2章で示したブロードキャストメモリ結合形システムの特徴、

(1) MIMD 型システムで PEとして汎用マイクロプロセッサを無理なく使用できるので設計・製作が容易である。また MIMD 型なので問題に柔軟に適応できる能力を持っている。

(2) (1)と関連するが、 サイズの大幅に異なる帯行列・疎行列などの係数行列をもつ連立一次方程式に対しても無理なく対応でき、効率的に解法計算が行える。特に、実用上重要な大型疎係数行列をもつ連立方程式の計算に柔軟に適応し、効率よく疎行列の特徴をいかした計算を進めることができる。

をそこなうことなく、大規模化への対応、

(3) 汎用マイクロプロセッサ数台から数十台を用いたシステムから、

単純な演算機能をもつPE数千台のシステムまで想定することが
可能であり、実現性が高い。

を可能にしている。

第５章

ＢＣプロセッサアレイ

5.1 緒言

Kung らによって提案されたシストリックアレイ[35]は 1次元または2次元のプロセッサアレイで、 1次元では両隣接プロセッサ、2次元では6方向の隣接プロセッサが相互に接続された形をしている。各PEにはプログラムの実行能力はなく、単純で規則的なデータの流れを受け入れ、それにパイプライン制御されたシストリックアルゴリズムを適用していくことにより目的の計算を高並列に実行することができる。

本章では、 2方向の隣接プロセッサとの結合端子に加えて、新たにBC(ブロードキャスト)端子と呼ぶブロードキャスト用端子を設けた演算プロセッサ(以下 BCプロセッサと呼ぶ)を提案する。そして、このBCプロセッサから構成される1次元または2次元のプロセッサアレイがシストリックアレイよりもすぐれた高並列計算能力を有していることを示す。

5.2 BCプロセッサ

BCプロセッサはシストリックアレイにおける演算プロセッサと同様に単純な構造をもったセルであるが、異なる点は隣接プロセッサ

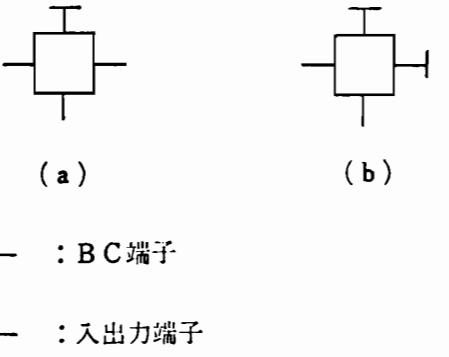


図5.1　BCプロセッサの例

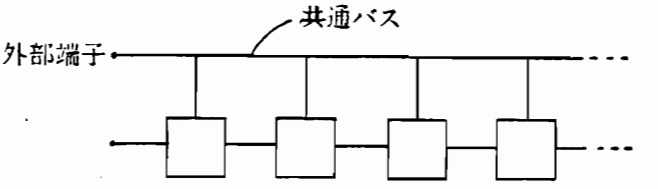


図5.2　1次元BCプロセッサアレイの例

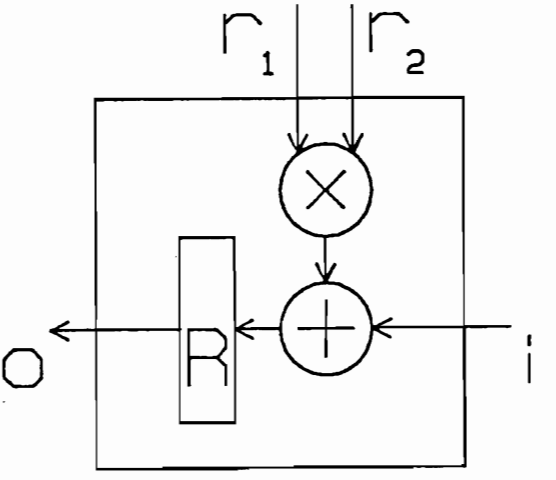


図5.3　BCプロセッサの内部構造

への接続端子のほかに一つまたは二つの BC 端子をもつことである(図5.1)。BC 端子は特別な端子で共通バスに接続されるように設計されており、 1本の共通バスに複数個のBCプロセッサがBC端子を介して接続されることになる(図5.2)。BC端子に出力されたデータは共通バスを介して他のBCプロセッサに伝送され、各BCプロセッサはそのデータを入力することができる。したがってl台のBCプロセッサ(p_1～p_l)が一本の共通バスにBC端子を介して接続されている場合、任意のp_i($1 \leq i \leq l$) または外部端子から共通バスにデータが出力されると全p_jは同時にそのデータを入力することができる。 したがって全 p_jに対するデータの放送が行われることになる。この場合、二つ以上のBCプロセッサや、外部端子と一つ以上のBCプロセッサが同時に共通バス上にデータを出力すると競合が発生することになるので、これは禁止されているとする。BCプロセッサに二つのBC端子がある場合には2本の共通バスにそれぞれ接続することができる。

BCプロセッサのBC端子には放送されてきたデータを記憶する機能はなく、この点が前述のブロードキャストメモリとの相違点となる。BCプロセッサ内部は、 演算回路とただ一つのレジスタ(パイプラインレジスタ)とから成っている。 そして、外部から与えられる共通クロックに同期して以下の動作を行う。

(1) BC端子および入力端子からの入力値に対して定められた演算を行う。

(2) その結果を内部レジスタに記憶するとともに、出力端子およびBC端子(必要な場合のみ)に出力する。

BCプロセッサの例を図5.3 に示す。これは、行列の乗算で用いる

BCプロセッサである。r_1, r_2 はBC端子、i は入力端子、o は出力端子で、R はレジスタである。この場合BCプロセッサは、$i+r_1*r_2$ を計算してRに記憶し、それをoから出力する。

5.3 行列計算

5.3.1 行列・ベクトル積

行列$A=(a_{ij})$とベクトル$x=(x_1, x_2, \cdots, x_n)^T$ の乗算問題($y=Ax$)を取り上げる。Aをn×nの帯行列とし、その帯幅をw=p+q-1 とする。

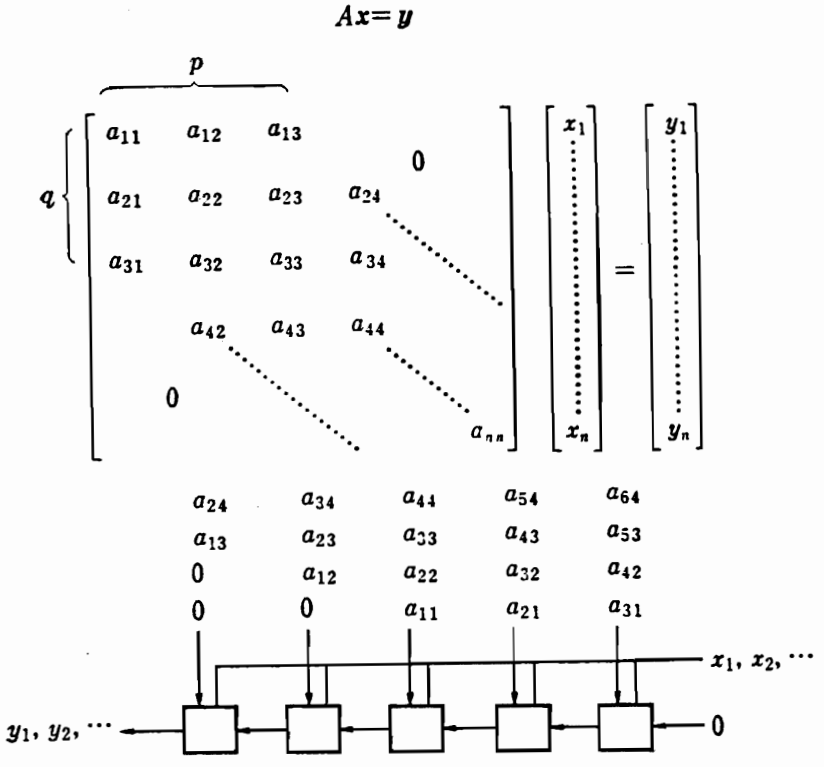


図5.4 行列・ベクトル積の場合

積 $y=(y_1,\cdots,y_n)$ の要素は

$$y_i=\sum_{k=max(1,\,i-q-1)}^{min(n,\,i+p-1)} a_{ik}x_k$$

となる。 この場合 y_iの値は図5.4 に示す構成の線状BCプロセッサアレイ（w個の BCプロセッサにより構成）を用いて並列計算を行うことができる。 ここで用いる BCプロセッサは 1BC端子、 2入力端子、 1出力端子、 1内部レジスタを持ち、右入力(r_i)、BC端子入力(r_b)、 上入力(r_u)を用いて $R=r_i+r_b \cdot r_u$ の計算を行う能力を持っている。

右端の端子には常に 0 が入力される。 計算は k=1～n に関して以下の手順で進められる。

図5.4 に示すように第1ステップからBC端子には $x_1,x_2,\cdots,x_n$ の値を、 上入力端子群には帯内の列ベクトル $a_1,a_2,\cdots,a_n$ を列単位に入力していく。 各BCプロセッサは1ステップごとに以下のサブステップの演算を行う。

1. BC端子から x_k、2入力端子から $a_{i,j}$と右BCプロセッサの演算結果を読み込み r_b,r_u,r_i とする。
2. $R=r_i+r_b \cdot r_u$
   の計算を行う。

pステップ目から順に左端の端子に $y_1,y_2,\cdots,y_n$ が出力されてきて n+p-1 ステップ目に y_nが出力される。 最初から3クロック分の動作のようすを図5.5 に示す。

上述の1ステップの動作時間を基本単位時間とすると、 シストリックアレイにおいては3入力端子、 2出力端子をもつプロセッサw台

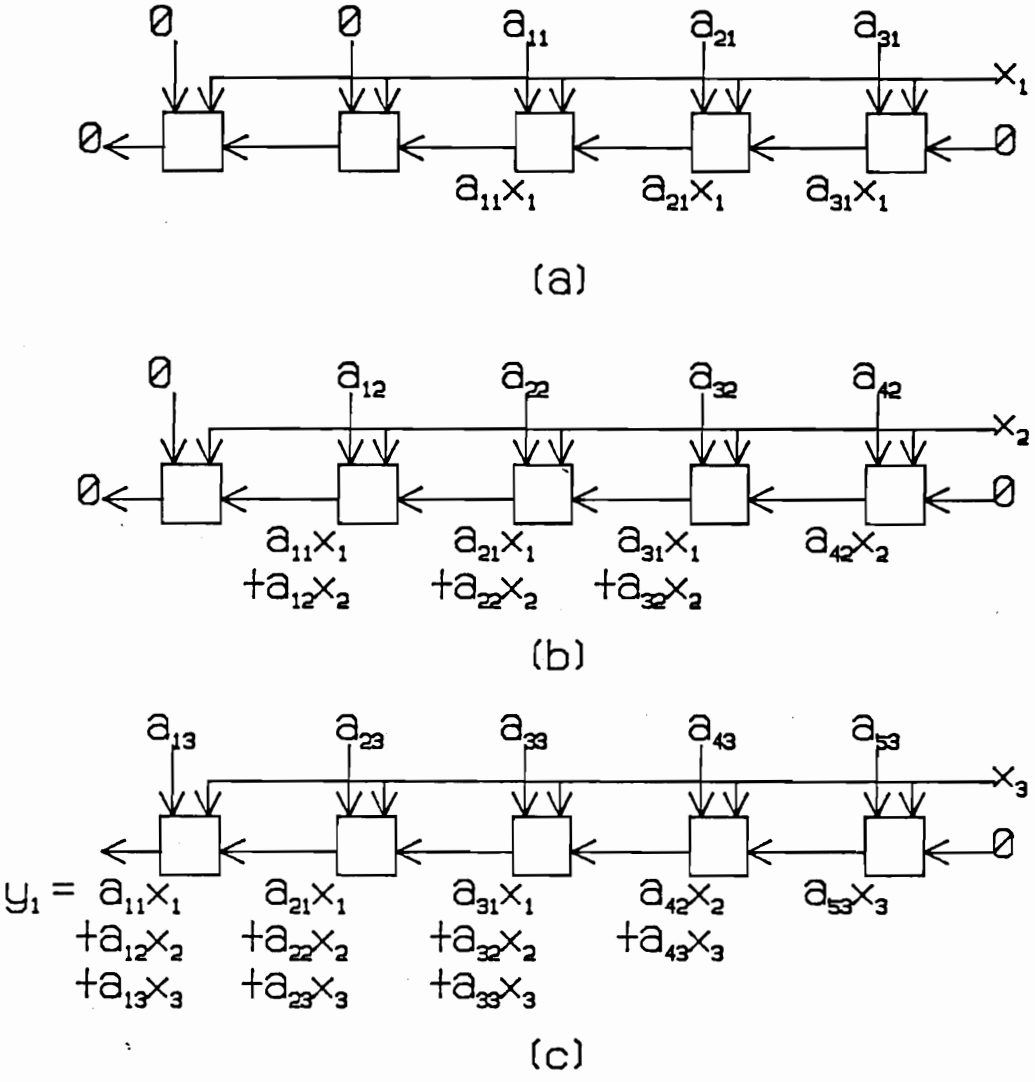


図5.5　行列・ベクトル積の実行ステップ

で 2n+w の計算時間を必要とするが、本BCプロセッサアレイではほぼ半分の n+p-1 の計算時間で実現できる。

5.3.2 行列乗算

二つの n×n 行列について考える。 $A=(a_{ij})$ と $B=(b_{ij})$ の積 $C=(c_{ij})$ は次式で計算される。

$$c_{ij}=\sum_{k=1}^{n} a_{ik}b_{kj}$$

AとBがそれぞれ帯幅を $w_1=p_1+q_1-1$、$w_2=p_2+q_2-1$ とする帯行列とすると $w_1 \cdot w_2$ のマトリクス形に接続したBCプロセッサアレイで並列計算が可能となる。用いられるBCプロセッサアレイは一つの内部レジスタRと2BC端子、1入力端子、1出力端子をもつもので(図5.6-a)、 各BCプロセッサは図5.6-c に示すように、行方向と列方向の共通バスと左上方および右下方の隣接プロセッサに接続されている。アレイの左端のBC端子群から a_i 列を1列ごとに、上端のBC端子群から b_j 行を1行ごとに入力していく(i=1～n, j=1～n)。 計算ステップは以下のサブステップより実現される。

1. 右下端子から右下隣接プロセッサの出力値、 行BC端子から a_i 列の各要素、列BC端子から b_j 行の各要素を読み込み r_i, r_1, r_2 とする。
2. $R=r_i+r_1 \cdot r_2$
   の計算を行う。

第 $\min(p_1, q_2)$ ステップ目から図5.6 のように左端および上端の出力端子群から c_{ij} が出力されてくる。最初の2クロックの動作を

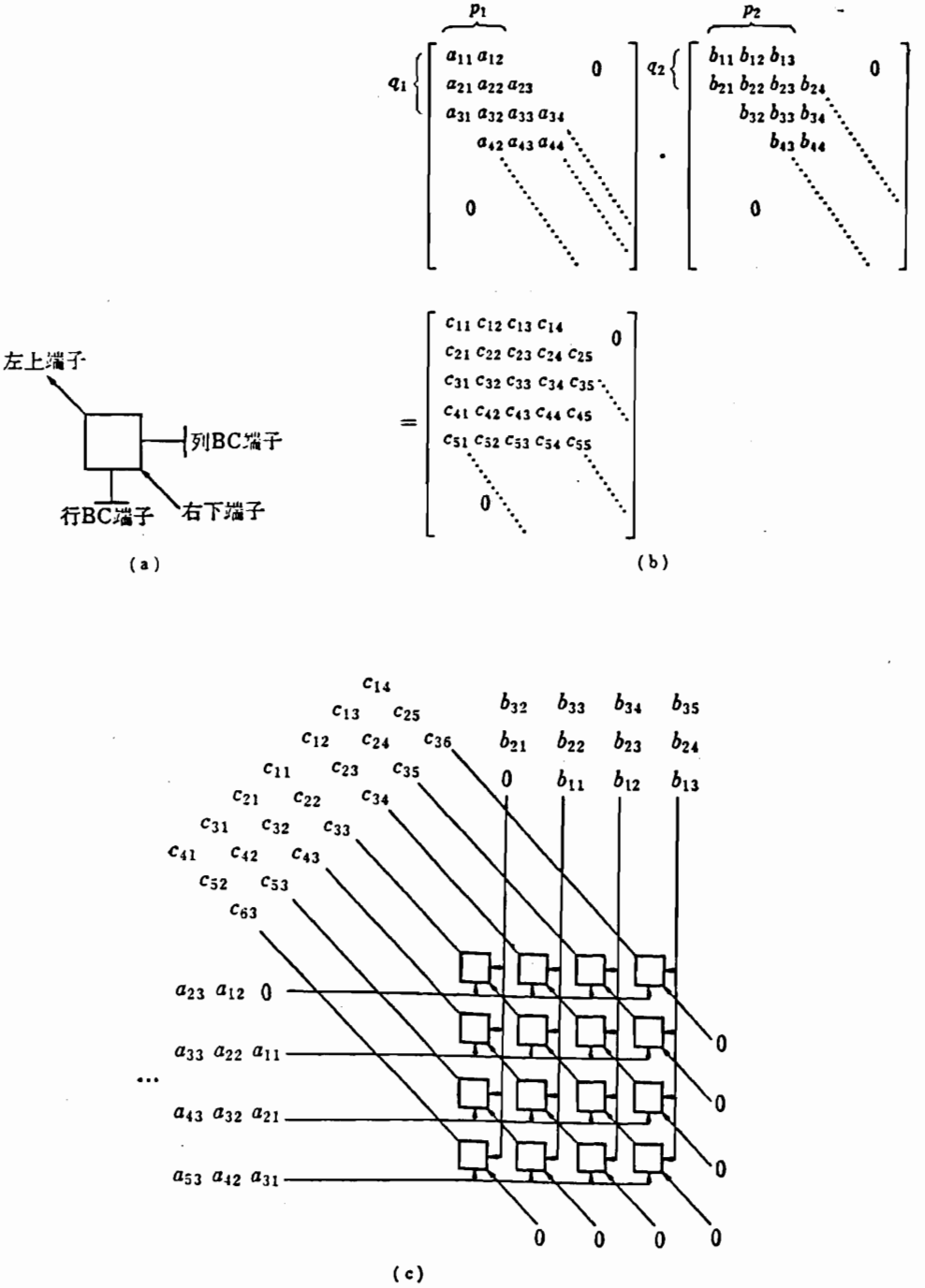
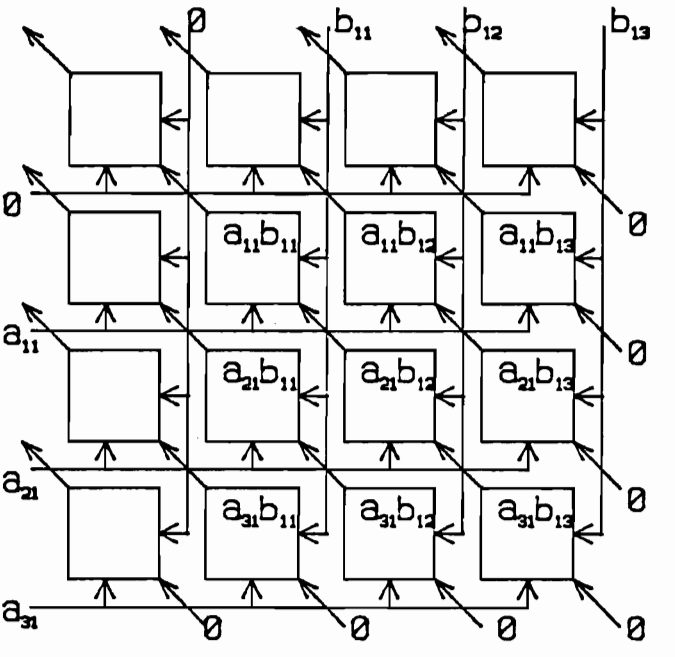


図5.6　行列乗算の場合

①



②

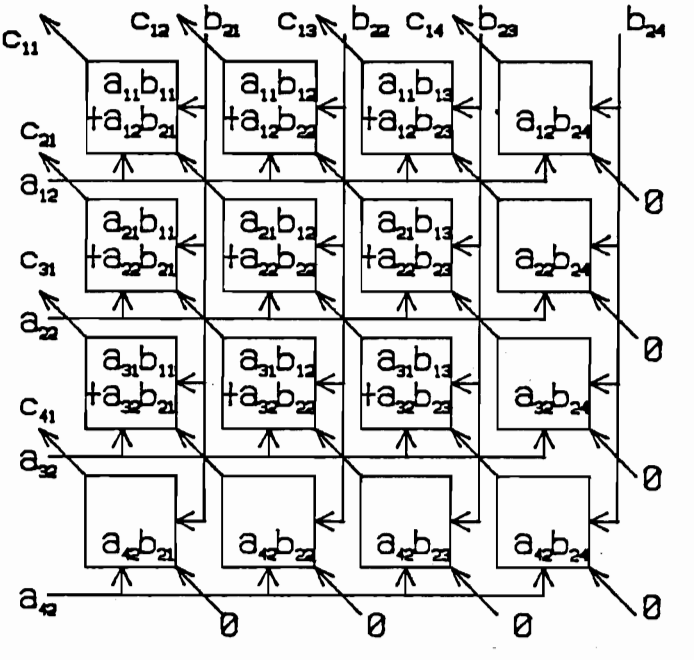


図5.7　行列乗算の実行ステップ

図5.7 に示す。

合計 $n+\min(p_1,q_2)$ 時間後にマトリクスCが得られる。シストリックアレイにおいては $3n+\min(w_1,w_2)$時間が必要となる。

5.3.3 LU分解

LU分解は行列AをA=L Uなる下三角行列Lと上三角行列Uとに分解する計算で、Aの逆行列を求めたり線形方程式A x=Bを解く過程で用いる。軸選択なしで計算できるとすると、L=(l_{ij})、U=(u_{ij})は K=1～n に対して次の漸化式で計算できる。

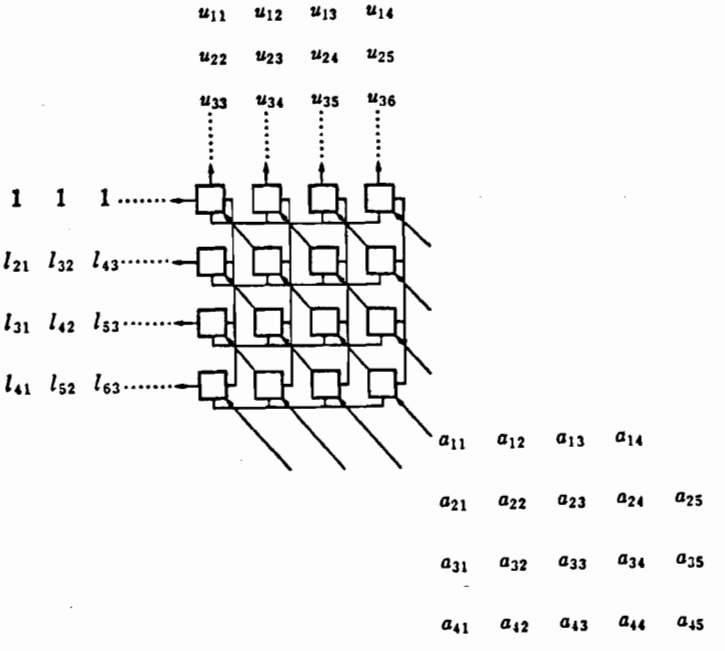


図5.8 行列のLU分解

$$a_{ij}^{(1)}=a_{ij}$$

$$a_{ij}^{(k+1)}=a_{ij}^{(k)}-l_{ik}\cdot u_{kj}$$

$$l_{ik}=\begin{cases}0 & i<k \text{ の場合}\\ 1 & i=k \quad 〃\\ a_{ik}^{(k)}\cdot u_{kk}^{-1} & i>k \quad 〃\end{cases}$$

$$u_{kj}=\begin{cases}0 & k>j \quad 〃\\ x_{kj}^{(k)} & k\leqq j \quad 〃\end{cases}$$

Aを帯幅 w=p+q-1 の行列であるとすると計算は 図5.8 に示される 2BC端子、1入力、1出力端子をもつBCプロセッサ p・q 台から構成されるプロセッサアレイで実現される。

行列Aの帯の部分は、図5.8 に示すように右ななめ下方から入力端子群を通して入力されてくる。 min(p・q)ステップ後にプロセッサ p_{ij}に a_{ij}($1\leqq i\leqq q, 1\leqq j\leqq p$) が伝搬してくる。

以後の計算ステップは、 k=1～n に関して以下の一連のサブステップにより進められる。

1. プロセッサアレイの第1行が

   $u_{kj}=a_{kj}$ ($k\leqq j\leqq k+p-1$)

   とし、 u_{kj}をそれぞれの上出力端子に出力するとともに列方向のBC端子に出力し、u_{kj}を列方向に放送する。 この際 p_{11}は特別のプロセッサで 、u_{kk}の逆数 u_{kk}^{-1}を計算し列方向に放送する。

2. プロセッサアレイの第1列が u_{kk}^{-1}を列BC端子から読み込んで

$$l_{ik}=\begin{cases}1 & (i=k)\\ a_{ik}^{(k)}\cdot u_{kk}^{-1} & (k<i\leqq k+p-1)\end{cases}$$

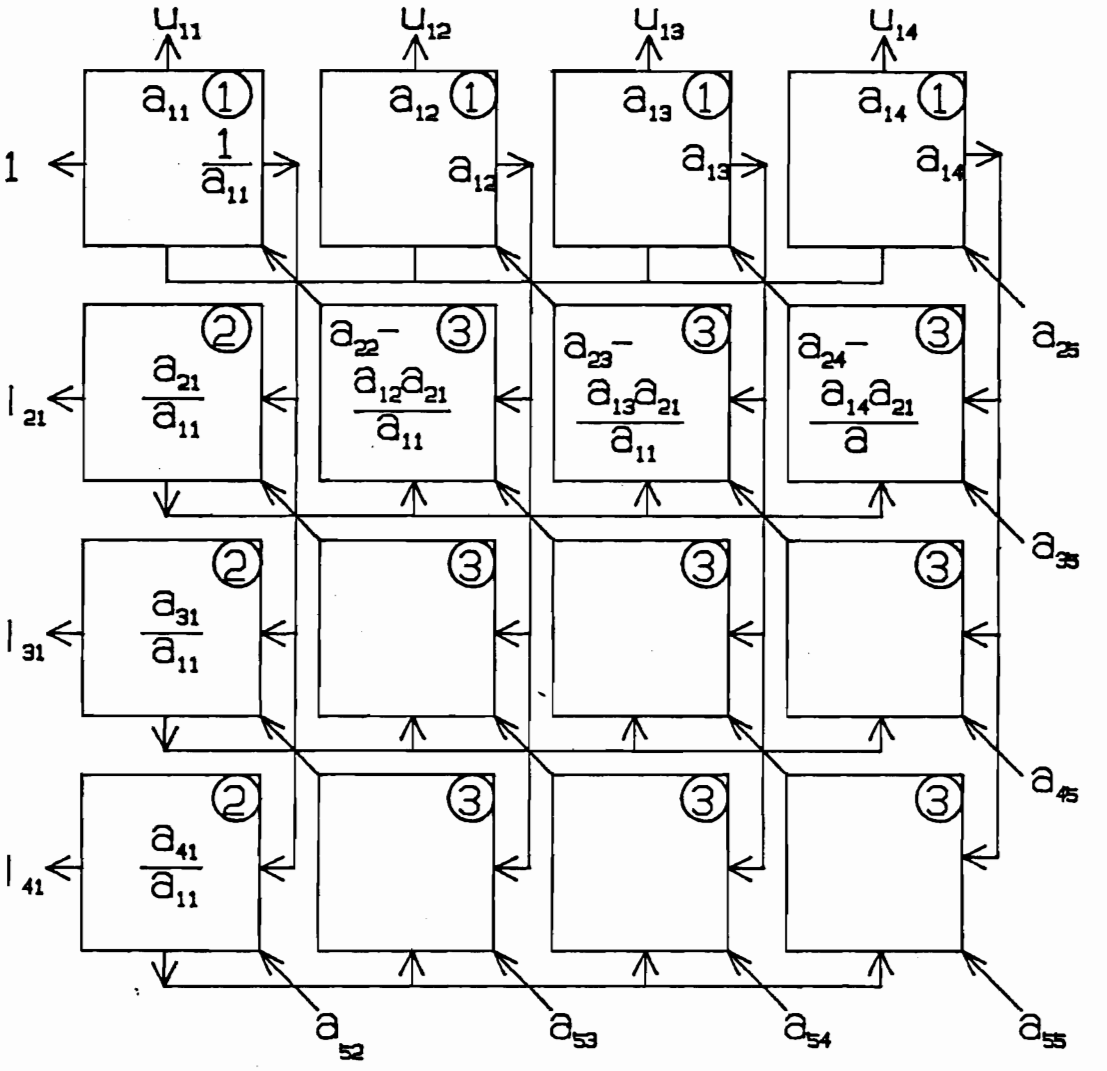


図5.9　LU分解の実行ステップ

の計算を行う。求めた l_{ik}を左出力端子から出力するとともに、行方向のBC端子に出力し行方向に放送する。この際、p_{11}は1を左出力端子に出力する。

3. プロセッサアレイの第1行と第1列とを除いた全プロセッサが

$$a_{ij}^{(k+1)}=a_{ij}^{(k)}-l_{ik}\cdot u_{kj}$$

の計算を行う。l_{ik}は行BC端子から、u_{kj}は列BC端子から入力する。

4. $a_{ij}^{(k+1)}$は左上方の出力端子に出力され、左上方の隣接プロセッサへ伝搬される。

a_{11}が左上すみに来たときの動作を、図5.9 に示す。 図中の番号は各サブステップに対応している。

上記の一連のサブステップを単位時間と考えると、 全体としてn+min(p,q) 時間でLU分解が完了する。 シストリックアレイにおいては 3n+min(p,q) 時間を要する。

5.4 連立一次方程式の計算

5.4.1 ガウス消去法(前進消去)

後退代入は後述する修正コレスキー法による計算の場合にも共通となる計算過程であるから 5.4.3節で論じることとし、ここでは前進消去についてのみ論じる。対象とする方程式はＡｘ＝Ｂで、Ａはn×n のサイズで帯幅 w=p+q-1 の帯行列であるとし、Ｂは n×l の密行列であるとする。

図5.10 に BCプロセッサアレイによるガウス消去法の実行の例が示してある。計算は図5.10 に示すように、Ｂに関しては p_{p1}～p_{pl}

の入力端子に下方から入力されたデータを、上方に向けて各ステップ1行ずつ伝搬させていく。 行列Aのほうは帯部分が、右半分のプロセッサ群の右下入力端子群から、図5.10 に示すように1ステップ1区画（点線で区切られた区画)ずつ入力され、左ななめ上方に伝搬されていく。計算はqステップまで伝搬を行ってから、k=1～n に対して以下の手順で実行される。

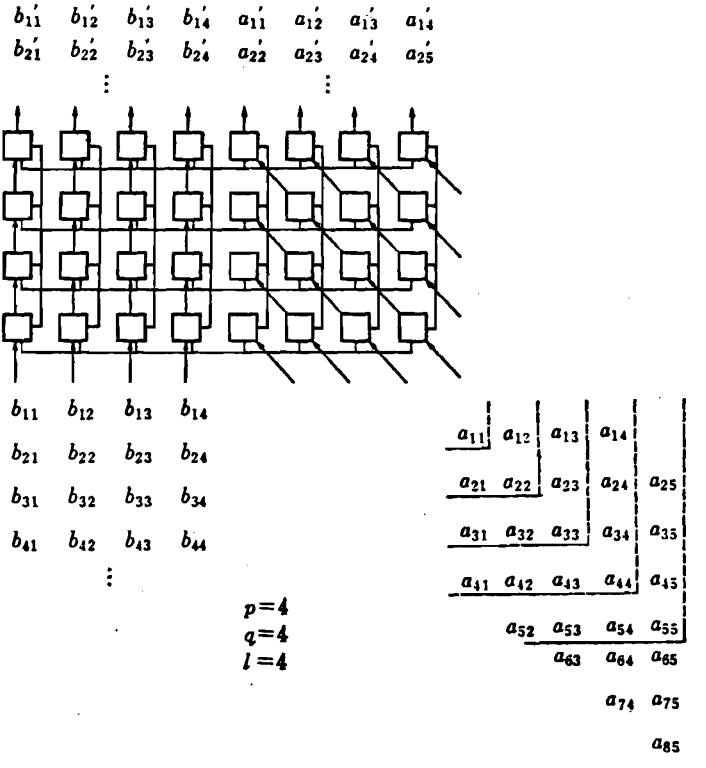


図5.10　ガウス消去法の前進消去

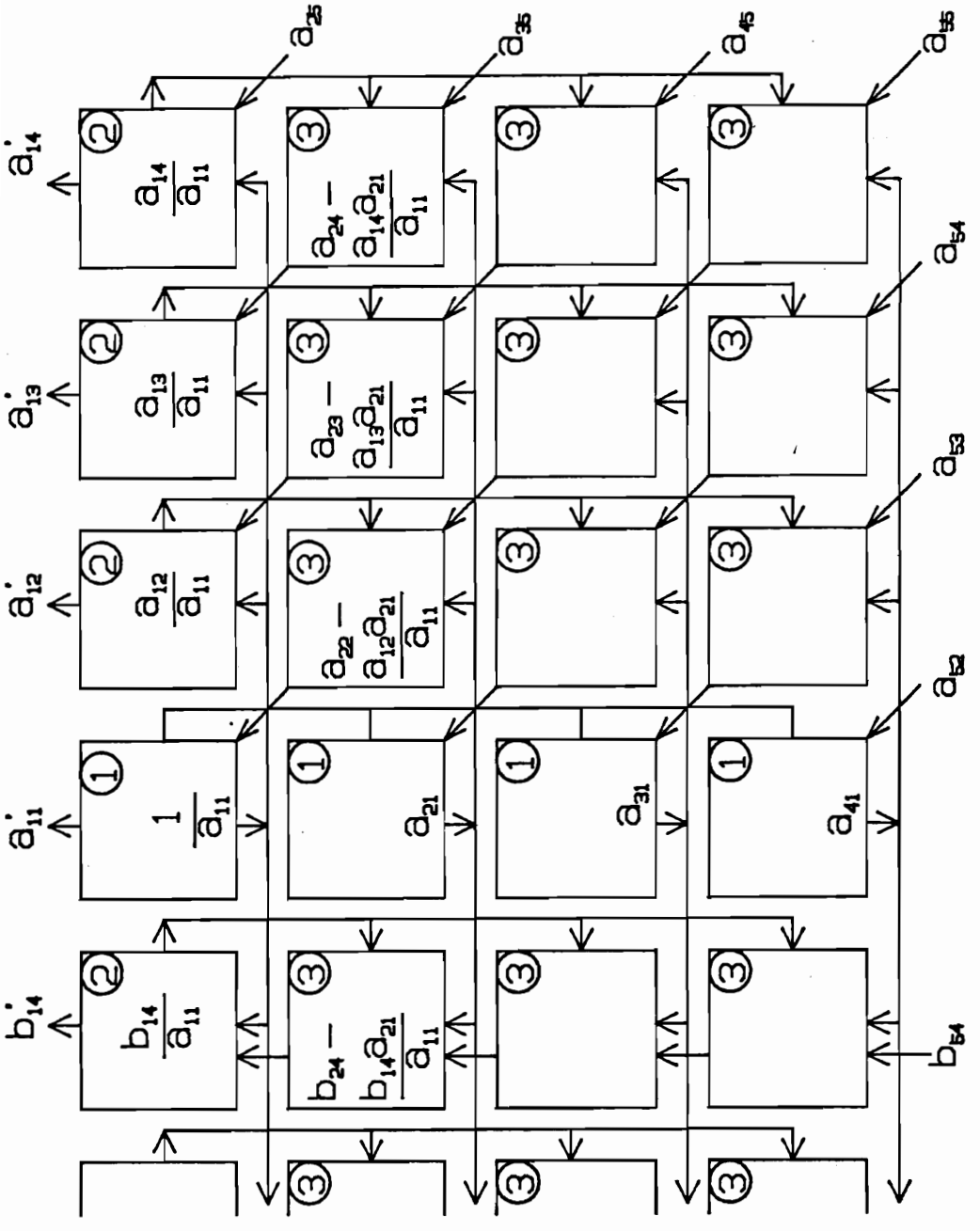


図5.11　前進消去の実行ステップ

1. $p_{1\ i+1}$がa_{kk}^{-1}の値を行BC端子に出力し、行方向に放送する。
2. プロセッサアレイの第1行が

$$a_{kj}=a_{kj}/a_{kk}\ (k+1\leqq j\leqq k+q)$$

$$b_{kj}=b_{kj}/a_{kk}\ (1\leqq j\leqq l)$$

の計算を行い a_{kj},b_{kj}の値を列BC端子に出力して列方向に放送する。

3. 全プロセッサが並行して

$$a_{ij}=a_{ij}-a_{ik}a_{kj}$$

$$b_{ij}=b_{ij}-a_{ik}b_{kj}$$

を行う。

4. a_{ij}を左ななめ上方に、b_{ij}を上方に伝搬させる。

a_{11}が第1行に到着した後の BCプロセッサアレイの動作を 図5.11に示す。

以上の計算を進めていくことにより前進消去のプロセスが、初期設定を含めて n+q-1 時間で実現できる。

5.4.2 修正コレスキー法

計算はＡｘ＝ＢをＡ'ｘ＝Ｂ'に変換するもので、 Ａ'は上三角行列となる。 Ａを、対角要素を含んで半帯幅がpの正定値対称行列であるとすると、計算は i=1～n について

$$a'_{ij}=a_{ij}-\sum_{k=1}^{i-1}a_{ki}a_{kj}/a_{kk}\ (i\leqq j\leqq \min(n,i+p-1))$$

$$b'_{ih}=b_{ih}-\sum_{k=1}^{i-1}a_{ki}b_{kh}/a_{kk}\ (i\leqq h\leqq l)$$

を行うことになる。ここではこの計算を若干変更し、以下のようにする。新たにベクトル a''を導入し、k=1～n について順次

$a''_{kj}=a_{kj}/a_{kk}$

$a_{ij}=a_{ij}-a''_{ik}a_{kj}$ $(k\leqq i\leqq \min(n,k+p),$

$k+1\leqq j\leqq \min(n,k+p))$

$b_{ih}=b_{ih}-a''_{ik}b_{kh}$ $(k\leqq i\leqq \min(n,k+p),1\leqq h\leqq l)$

の計算を進めていけば a_{ij}は a'_{ij}に、b_{ih}は b'_{ih}に変換される。

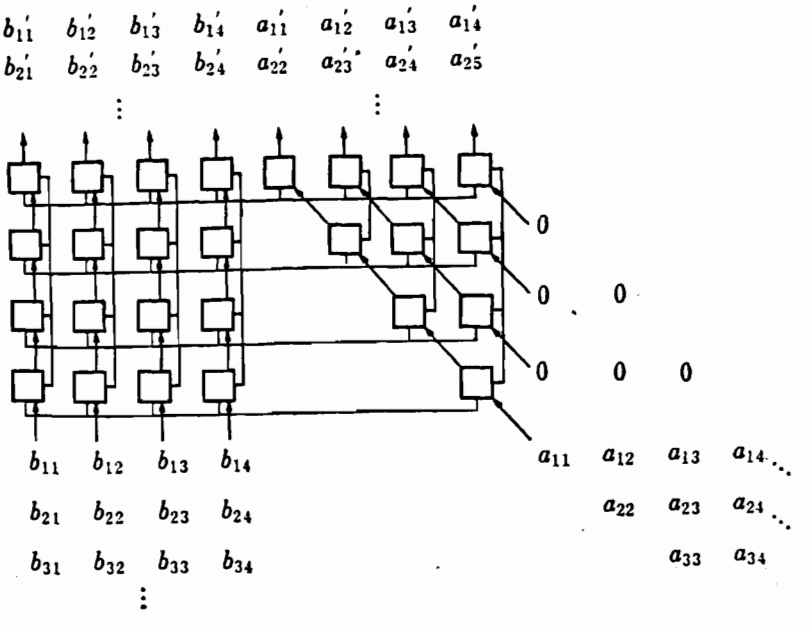


図5.12　修正コレスキー分解

図5.12 に示されているように、Bは第1行から順に1ステップ1行ごとに下端子から入力され、 上方に1行ずつ伝搬されていく。Aは上三角部分のみ1ステップ1列ごとに右ななめ下方から入力され、左上方に伝搬されていく。 最初のpステップで b_{ih}は $p_{ih}(1 \leqq i \leqq p, 1 \leqq h \leqq l)$ に、a_{ij}は $p_{i\ j+i}(1 \leqq i \leqq p, 1 \leqq j \leqq p)$ に到着する。

以後 k=1～n に関して以下の計算サブステップを進めていく。

1. $p_{1\ i+1}$が $1/a_{kk}$ の値を計算して行BC端子に出力し、 行方向に放送する。
2. $p_{1\ i+1}$～$p_{1\ i+p}$ が $a_{kj}/a_{kk}(k \leqq j \leqq \min(n, k+p))$ を求め、結果を列BC端子を介して列方向に放送する。
3. 対角上のプロセッサ $p_{1\ i+1}, p_{2\ i+2}, \cdots, p_{p\ i+p}$ が 2.で計算された a_{kj}/a_{kk}の値を行BC端子に出力し、行方向に放送する。この値が a''_i となる。 同時に第1行のプロセッサが列BC端子に a_{kj} および b_{kj} を出力し、列方向に放送する。
4. 全プロセッサが並行して

   $a_{ij}=a_{ij}-a''_i a_{kj}$

   $b_{ih}=b_{ih}-a''_i b_{kh}$

   の計算を行う。
5. a_{ij} を左ななめ上方に、b_{ij}を上方に伝搬させる。

初期設定のステップが終わり、a_{11}が第1行に到着してからの動作を図5.13 に示す。

計算は、初期設定のステップを含め n+p 時間で終了する。

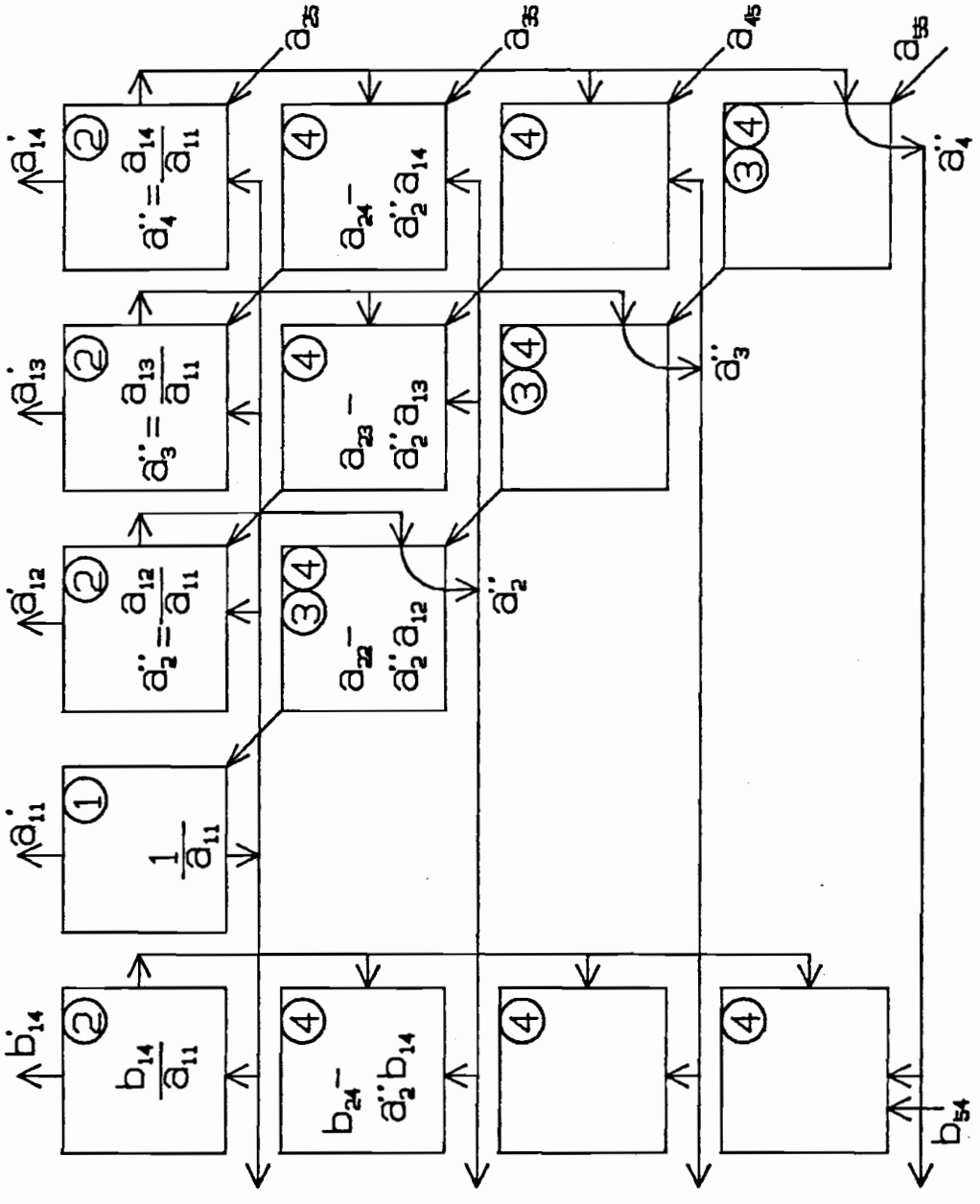


図5.13　修正コレスキー分解の実行ステップ

5.4.3 三角線形方程式

Ｕｘ＝Ｂなる連立一次方程式でＵが n×n の上三角帯行列で、対角要素を含んだ帯幅がpであるとする。 またＢは n×l の定数項行列であるとする。 この連立一次方程式の解は図5.14 に示す構成のBCプロセッサアレイで、以下の手順で計算できる。

Ｂの第n行 b_nから順に、 プロセッサアレイの第1行の上入力端子群から1ステップ1行ずつ入力し、 下方に伝搬させていく。pステップ後に b_n行がプロセッサアレイの第p行に到着する。以後計算は以下のように、k=1～n に対して実行されていく。

1. 行BC端子群に u_{n-k+1}列を入力する。
2. 第p行のプロセッサが

   $$x_{n-k+1\ j}=b_{n-k+1\ j}/u_{n-k+1\ n-k+1}\quad(1\leqq j\leqq l)$$

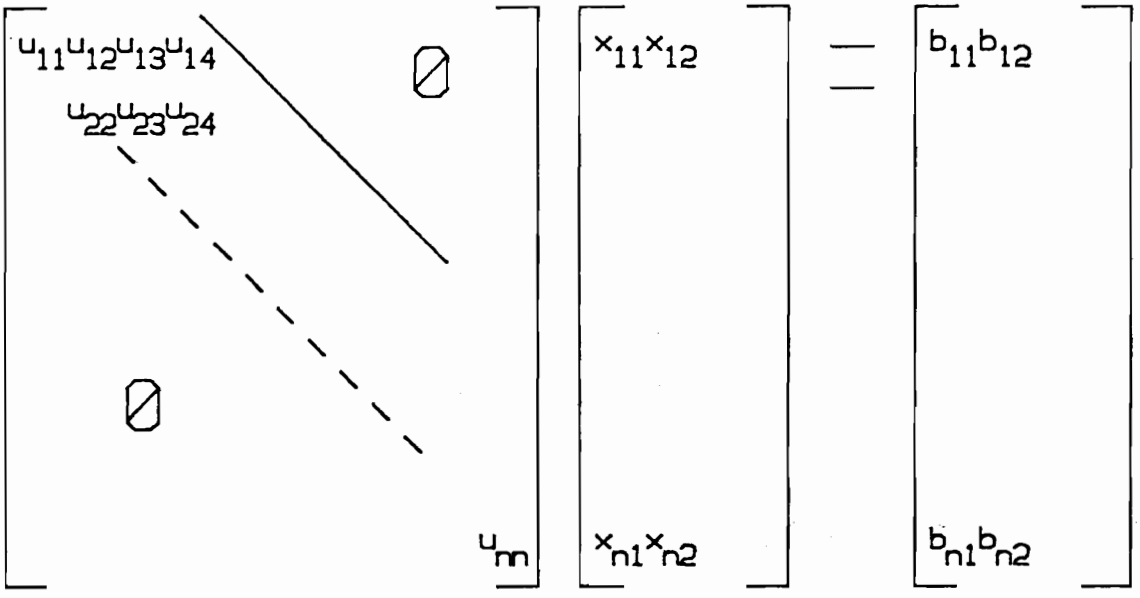
   の計算を行い、$x_{n-k+1\ j}$ を下方端子に出力するとともに列 BC 端子を介して列方向に放送する。
3. $p_{i\ j}(1\leqq i\leqq p-1, 1\leqq j\leqq l)$ が

   $$b_{n-k-i+1\ j}=b_{n-k-i+1\ j}-u_{n-k-i+1\ n-k+1}\cdot x_{n-k+1\ j}$$

   の計算を行う。
4. 新しく計算した $b_{n-k-i+1\ j}$ を下方に1行伝搬させる。

pステップ後、 b_n行が最も下の行に到達してからのプロセッサアレイの動作を図5.15 に示す。

初期化にp、x群計算にn の合計 n+p 時間の計算で解が得られる。シストリックアレイでは 2n+p 時間を要する。



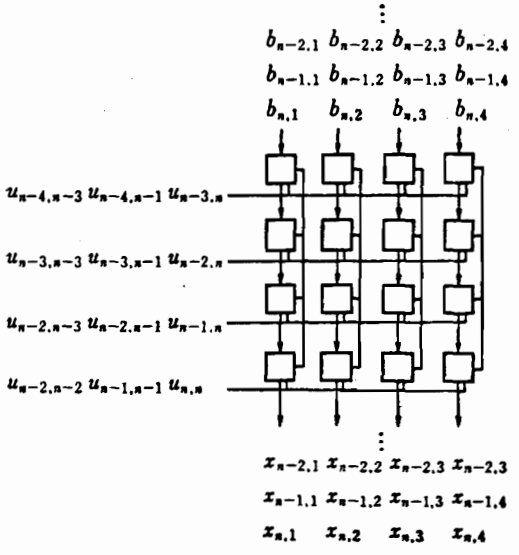


図5.14　三角線形方程式の計算

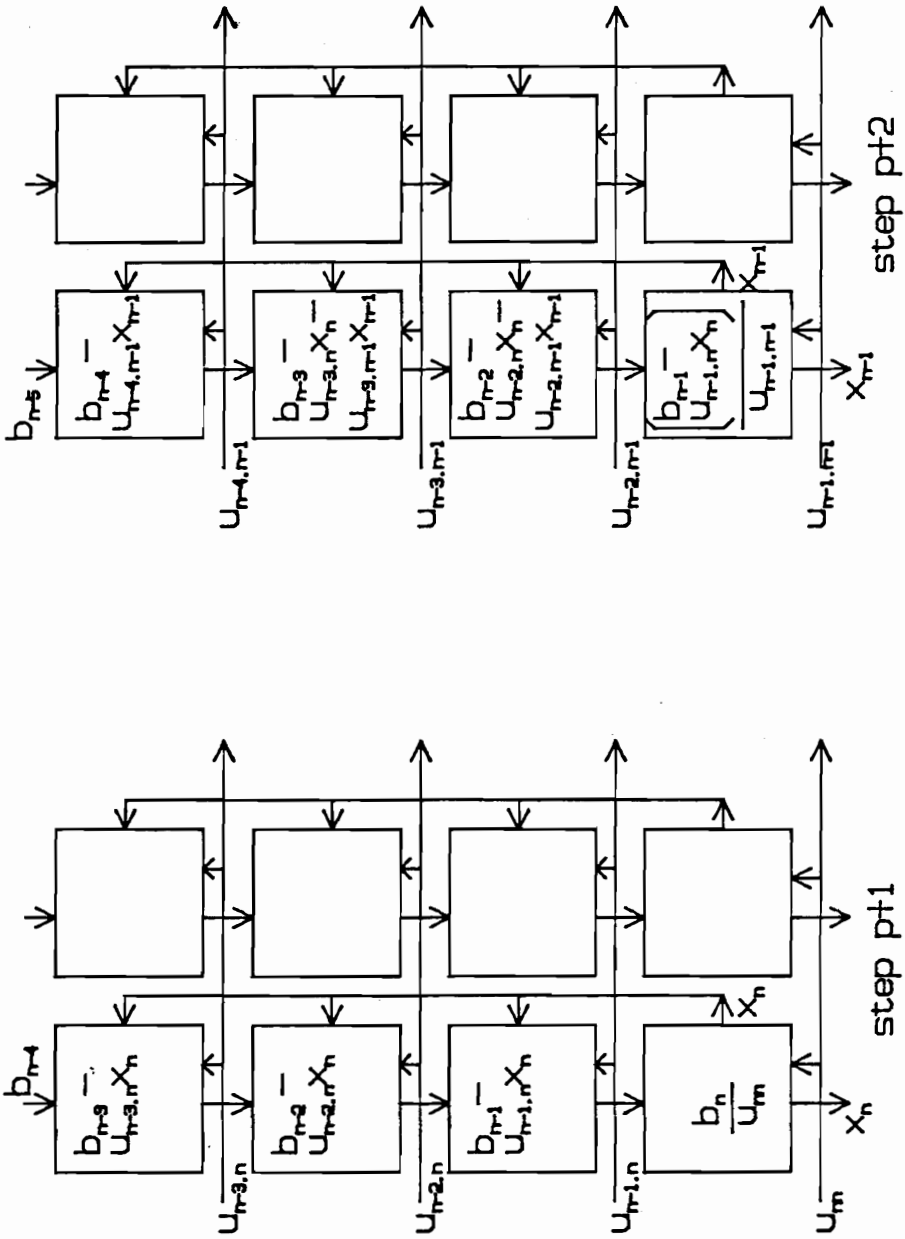


図5.15　三角線形方程式の実行ステップ

5.5 結言

プロセッサ数を数千台以上用いるような超高並列計算機の研究において、 Kung らが提案しているシストリックアレイは高い評価を受けている。

本章で提案したBCプロセッサアレイはシストリックアレイと同様にVLSI向きの構成をしているのみならず、シストリックアレイと比べて以下の点ですぐれている。

(1) シストリックアレイでは、ベクトルと行列の積および三角線形方程式の計算に約2nステップ、行列の乗算およびLU分解に約3nステップを要するのに対して、BCプロセッサアレイではいずれの場合でも約nステップで実現でき、 演算プロセッサの稼働率が高い。

(2) 放送機能を有するため、ガウス消去法や修正コレスキー法のアルゴリズムが効率よく実現できる。

(3) シストリックアレイでは伝搬してきたデータを保持しておくための内部レジスタが必要となるが、BC端子から放送されるデータは共通バスに保持されているので、BCプロセッサ内にデータを保持しておくための内部レジスタを省くことが可能となる。

第6章

結論

多重マイクロプロセッサによる行列計算能力の現状を見てみると、いわゆるパイプライン制御方式を採用したスーパーコンピュータに対して、その速度の点で太刀打ちできるところまでいっていないが、次のような点で将来性が期待できる。

(1) VLSI 化によるコストの低下と超多重化の可能性が出てきた。

(2) 専用プロセッサによる、数値演算能力の飛躍的向上が期待できる。

(3) パイプライン処理では十分に並列性を引き出せないような不規則なデータに対しても柔軟に対応できる。

特に、チップあたり数十万ゲートを越える VLSI が実現してくると、その応用としての超高並列計算の分野が重要な目標となってくる。超高並列計算は 10^2～10^5 オーダの数のプロセッサを用いるもので、プロセッサ間結合方式と並列計算アルゴリズムがきわめて重要である。

本論文では、以上のような将来性をもつ並列計算機システムの実現に対する、ブロードキャストメモリの有効性について論じた。具体的には、ブロードキャストメモリによって結合した三つの並列計算機システムと、その上での行列計算の並列処理方法を提案し、数値計算分野での並列計算機の実用化に対する一つの解答を与えた。

結論

第3章で述べたシステム はブロードキャストメモリ結合システムの最も基本となるものであり、試作によってブロードキャストメモリの実現方法と有効性を示した。ハードウエアでは特にブロードキャストメモリを実現するためのバス構造を中心に述べ、ソフトウエアでは数値計算問題向きに開発した並列プログラミング言語について述べた。また、連立一次方程式の解法に対する並列計算の実現方法と実行結果を示した。これらの結果より、ブロードキャストメモリはバス結合の持つ問題点であるアクセス競合を低減し、簡単なシステムでも容易に並列計算の効果が得られることを示した。

第4章では、 マトリクスブロードキャストメモリ結合形システムを提案した。 このシステムはブロードキャストメモリを2次元格子状に組み合わせたもので、この上でガウス消去法と修正コレスキー法の計算時間を $O(n)$ とすることが可能であることを示した。
また、 ガウス消去法に軸選択を取り入れた場合でも、計算時間を $O(n \log_2 n)$ とできることを示した。

第5章では、 ブロードキャスト機能とパイプライン機能とを合わせ持つBCプロセッサと、これを構成要素とするBCプロセッサアレイを提案した。そして、BCプロセッサアレイによる行列計算アルゴリズムを示し、同様の構成を持つシストリックアレイに対して2〜3倍の高並列性を持つことを示した。

今後の課題として、

(1) 並列計算機上での並列プログラミング・実行・デバッグをサポートするオペレーティングシステムやソフトウエアツールなど、並列プログラムの開発・実行環境を整備すること。

(2) 数値計算以外の分野も含めたより多くの問題に対して、ブロードキャストメモリの有効性を調べること。

などがあげられる。

謝　辞

本研究の全過程を通じて、直接理解ある御指導、御教示を賜わった神戸大学大学院自然科学研究科システム科学専攻　前川禎男教授（神戸大学工学部システム工学科）に心から感謝の意を表する。また、熱心な御指導、御教示を賜わったシステム科学専攻　金田悠紀夫助教授（システム工学科）に深く感謝する。

大学院博士過程において、本研究に対して理解ある御指導、御教示をいただいた、自然科学研究科システム科学専攻　松本治弥教授（計測工学科）、瀬口靖幸教授（システム工学科）、平野浩太郎教授（電子工学科）、池田雅夫助教授（システム工学科）に心から御礼申し上げる。また、筆者が神戸大学工学部電子工学科に在学中より、御指導ならびに激励をいただいた　高橋豊助手（電子工学科）に厚く御礼申し上げる。

本研究を進めるにあたって、有益な御助言、御討論をいただいた新世代コンピュータ技術開発機構の滝和男氏、神戸大学大学院自然科学研究科の和田耕一助手、ならびに自然科学研究科博士過程の田村直之氏、松田秀雄氏に深く感謝する。また、本研究において、多大なる御協力をいただいた元神戸大学工学部システム工学科第４講座　西野佐登史氏、角木裕成氏、田中敏幸氏、藤川昌彦氏、山元賢治氏、橋本篤男氏、久保誠一氏に深く感謝する。最後に、本論文作成に多大なる御尽力をいただいた、システム工学科第４講座の小畑美保子技官に感謝する。

参 考 文 献

1) 元岡:スーパーコンピュータの現状と展望,情報処理,Vol.22,No 12.pp1103-1110,(1981)

2) D.L.Slotnick , W.C.Borck , R.C.McReynolds : The Solomon Computer,FJCC.22 (1962)

3) 加藤,苗村:並列処理計算機,オーム社 (1976)

4) J.Holland : Universal Computers Capable of Executing and Arbitrary Number of Sub-progrems Simultaneously, EJCC,16 (1959)

5) 坂井,稲垣,加藤: パターン理解マルチマイクロプロセッサシステムMACSYMの設計と画像処理への応用例,信学技報,EC81,pp49-60,(1981)

6) A.K.Jones, P.Schwarz : Experience Using multiprocessor Systems - A Status report,Computing Surveys,Vol.12,No.2, pp121-165,(1980)

7) 星野ほか: 科学技術専用並列計算機 PACS の開発,昭和55年,信学総全大,pp1400-1401

8) 西村ほか:LINKS-1 コンピュータグラフィックスシステム,情報処理学会マイクロコンピュータ研究会資料,24,pp1-7,(1982)

9) 村岡,板間:並列処理技術,信学誌,Vol.63,NO.10,pp1064-1071, (1980)

10) 西山,山根,高橋: 2進木構造並列処理システムCORALによる関数型プログラムの処理,信学技報,Vol.82,EC82-61,(1982)

11) 中川,小林,相磯: データ駆動型離散系シミュレータ KDSS-1,信

学論,Vol.J65-D,No.3,pp386-393,(1982)

12) 田中: 並列処理システムの性能を左右する相互結合ネットワーク,日経エレクトロニクス,1981.12.21号,(1981)

13) L.D.Wittie : Communication Structure for Large Networks of Microcomputers, IEEE Trans. on Comuter,Vol.c-30,No.4, pp264-272,(1981)

14) D.M.Dias, J.R.Jump : Analysis and Simulation of Bufferd Delta Networks ,ibid.,pp273-282

15) M.A.Franklin : VLSI Performance Comparison of Banyan and Crossbar Communication Networks,ibid.,pp283-291

16) H.J.Siegel : Interconnection Networks for SIMD Machines, COMPUTER,Vol.12,No.6,pp57-65,(1979)

17) P.B.Hansen : Concurrent Programming Concepts, Computing Surveys,Vol.5,No.4,pp223-245,(1973)

18) J.L.Baer: A Survey of Some Theoretical Aspects of Multi-processing,Computing Surveys,Vol.5,No.1,pp31-80,(1973)

19) E.W.Dijkstra : Co-cperating Sequential Processes, InProgramming Languages, Academic Press,New York,(1968)

20) C.A.R.Hoare : Monitors: An Operating System Structuring Concept,Communications of ACM, Vol.17, No.10, pp549-557, (1974)

21) INMOS Limited : occum プログラミングマニュアル,啓学出版, (1984)

22) P.B.Hansen : The Architecture of Concurrent Programs, Prentice-Hall,(1977)

23) L.G.Testler: A Language Design for Concurrent Processes, AFIPS proc., SJCC-68,Vol.32,pp403-408,(1968)

24) G.W.Andrews, F.B.Scheneder : Concepts and Notation for Concurrent Programming, Computing Surveys, Vol.15,No.1, pp3-43,(1983)

25) D.D.Chamberlin : The Single Assignment Approach to Parallel Processing,AFIPS proc.,Vol.39,pp263-269,(1971)

26) E.A.Ashcroft, W.W.Wadge : Lucid, A Nonprocedual language with Iteration,Comm.of ACM,Vol.20,No.7,pp519-526,(1977)

27) J.B.Denis:First Version of Data Flow Procedure Language, Lecture Notes in Computer Science, 19G,Springer-Verlag, New York,pp362-376(1974)

28) J.B.Denis, D.P.Misunas : A Preliminary Architecture for a Basic Data Flow Processor, Proc. 2nd Annual Simposium on Computer Architecture,Jan.1975,pp126-132

29) J.E.Rumbawgh : A Data Flow Multiprocessor,IEEE Trans. on Computer,Vol.c-26,pp138-146,(1977)

30) Z-80 テクニカルマニュアル,シャープ

31) The 8086 Family user's Manual,Intel co.

32) CP/M User's Guide,Digital Research

33) CP/M-86 User's Guide,Digital research

34) 戸川:マトリクスの数値計算,オーム社,(1971)

35) H.T.Kung : The Structure of Parallel Algorithm, Advances in Computers,Vol.19,pp65-112,Academic Press,(1980)

36) マイクロプロセサの数値演算能力を強化する浮動小数点演算

LSI,日経エレクトロニクス,No.327,pp129-140,(1983)
37) 金田,小畑,前川: マトリクスブロードキャストメモリ結合形並列計算機による n元連立一次方程式の O(n) 時間計算,情報処理学会論文誌,Vol.23,No.5,pp570-575,(1982)
38) 金田,小畑,前川:BCプロセッサアレイと高並列マトリクス計算,同上,Vol24,No.2,pp175-181,(1983)
39) 小畑,金田,前川: ブロードキャストメモリ結合形マルチマイクロプロセッサシステムの試作, 同上, Vol.24,No.3,pp351-356,(1983)
40) 金田,小畑,角木: マトリクスブロードキャストバス結合形並列プロセッサによる軸選択形ガウス消去の並列計算法,同上,Vol.25,No.3,pp495-498,(1984)
41) Y.Kaneda, M.Kohata : Highly Parallel Computing of Linear Equationson the Matrix-Broadcast-Memory Connected Array Processor System,10 IMACS,(1982)
42) 小畑ほか: ブロードキャストメモリ結合形並列計算機による行列計算の試み, 情報処理学会計算機アーキテクチャ研究会資料48,pp1-9,(1983)
43) 小畑ほか: 並列プログラミング言語の設計と実現,信学技報,EC84-5,pp43-48,(1984)
44) 小畑ほか: ブロードキャストメモリ結合形並列計算機システムの試作, 情報処理学会全国大会, 第25回, pp103-104, (1982)
45) 小畑ほか: ブロードキャストメモリ結合形並列計算機によるガウス消去法の並列計算, 同上, 第26回, pp191-192, (1983)
46) 小畑ほか: 並列計算機上での動的計画法の実行とその評価, 同

上，第27回，pp73-74，(1983)
47) 小畑ほか：並列計算機用プログラム言語Q，同上，第28回，pp407-408，(1984)
48) 小畑ほか：放送メモリ結合形並列マシン上でのＤＰ計算の並列化，同上，第29回，pp201-202，(1984)