受理　59.12.22

学位請求論文

問題に適応して再構成可能な構造を持つ高並列処理システムの構成に関する研究

昭和５９年１２月２２日

指導教官　斉藤忠夫 助教授

東京大学大学院 工学系研究科 電気工学専門課程

大山 敬三

目　次

第1章．序　論

1.1.研究の背景と目的

さまざまな分野において，計算機の応用が進むにつれ，処理装置には高い処理能力が要求されるようになってきている．これに対応する一つの方法として並列処理方式の研究が進められている．

一方，ＶＬＳＩ技術の進歩により，同一の高機能な論理装置を大量に安価に製造することが可能となってきており，高度の並列処理装置の実現に必要な技術的および経済的背景が整ってきている．そこでこのＶＬＳＩの量産性を生かし，同一の処理要素を多数，有機的に結合し，協調して一つの処理を実行するようなシステムの実現が重要な課題となる．

並列処理方式はこれまで大きく分けて専用処理システムと汎用処理システムにおいて検討されてきた．しかし，汎用処理システムでは結合形態が汎用であり，応用ごとの特徴を生かせないために高並列化による性能向上は難しい．一方，専用処理システムでは高並列化は比較的容易だが結合は単純で柔軟性に乏しく，適応範囲が限られる．また各アプリケーションにおいても処理内容の量と共に質の変化も激しく，この傾向は将来一層大きくなると考えられ，従来のような専用処理システムでは質的変化に対応しきれず，また，汎用処理システムでは充分な処理能力が実現できない．このような状況に対処するために，専用処理システムと同程度の処理能力を持ち，かつさまざまな応用分野に対応できる柔軟性の高い並列処理システムのアーキテクチャが求められている．

本研究で提案しているシステムは，柔軟な構造を持つ高並列処理システムであり，さまざまな応用分野において専用処理システム的な機能を提供することを目的とする．定型的で大量の処理を要する並列性の高い応用を対象とし，同一の処理要素を多数用いることにより高並列動作による処理性能の向上を図る．

専用処理システム的な機能とは次のようなことをいう．

- アーキテクチャを考慮したアルゴリズムを必要とする．
- プログラム開発などを行なうためにホスト計算機を仮定する．
- 処理の実行のための初期設定にはある程度のオーバヘッドを許す．
- プログラムの構造が動的に変化しない比較的定型的な処理を対象とする．
- 専用処理システムと同程度の処理能力を実現する．

本システムは汎用処理システムと専用処理システムとの間に位置づけられる．

汎用処理システムが万能性を求められるのに対し，本システムでは応用範囲をある条件を満たす範囲に限定することにより，専用処理システムと同程度の処理能力を実現する．これにより従来汎用処理システムでは処理能力不足だが，専用処理システムを構成するには需要が小さかったりあるいは処理の定型化が進んでいないような応用分野においても，専用処理システム開発のリスクを冒すことなく要求に見合う処理性能のシステムを利用することができるようになる．

従来は論理機械を製造するためのコストが高かったため，並列処理システムにおいても少数のプロセッサをできるだけ効率よく使用する構成が研究されてきた．少数のプロセッサによる並列処理システムでは共通バスやスイッチマトリクス，多段スイッチのような汎用の結合回路網におけるスイッチング，経路選択，競合制御等による通信制御の複雑化に起因するオーバヘッドや配線上の伝送遅延はあまり問題にならなかった．しかし，ＶＬＳＩ技術の進歩にともない論理機械の製造コストは著しく低下してきており，多数のプロセッサを組み合わせて用いることができるようになってきた．このようなシステムでは結合回路網も大規模になり，オーバヘッドや遅延が全体の性能に大きな影響を与えるため，より効率の良い結合回路網形態が求められる．

このために，本システムでは結合回路網を静的な回線接続と局所性の抽出が可能で，各アプリケーションごとに特有のデータ依存性に対応してシステムの再構成を可能とする，いわば問題に適応可能な構成とし，また，演算実行制御にデータ駆動制御を導入し，プログラムの開発を容易とすると同時に実行制御のオーバヘッドを小さく抑えるような構成を採用する．これにより，高度の並列処理能力が実現でき，その柔軟性と高いスループットにより，従来アレイプロセッサ，シストリックアレイ，トリーマシン等で検討されてきた応用に対応でき，さらに，不規則であっても静的な構造を持つ応用に対しては適応可能となり，高並列処理システムとしては従来にない柔軟性を実現できる．その反面，プログラム開発における負荷が増大する可能性があり，この点に関する検討はまだ充分ではない．しかし，これについても結合回路網と演算処理を物理的，論理的に分離したアーキテクチャを採用することにより問題の困難さを低減するよう考慮しており，同じクラスに属する他のシステムに比べ解決は容易である．

本論文ではこの結合回路網の構造について述べ，データ駆動制御を実現するた

めの処理要素の構造を明らかにし，プログラム開発と資源割付けに関する検討をし，システムの適応性と性能の評価を行なう．

1.2. 論文の構成

本論文は序論を含めて９章より構成されており，図１－１にその全体構成を示す．第１章において本研究の目的と背景について述べ，第２章で現在の並列処理システムの問題点を明らかにするとともに本研究の位置付けを明確にし，第３章では提案システムの全体的な概念と構成について述べ，第４章では結合回路網，第５章では処理要素の演算部のハードウェア構成の検討を行ない，第６章では基本言語によるプログラムの記述について述べ，第７章では資源割付け方法とシステムの適応性に関する検討，第８章ではシステム全体のハードウェアの性能評価を行ない，第９章において本研究の成果をまとめるとともにいくつかの将来における検討課題について言及する．

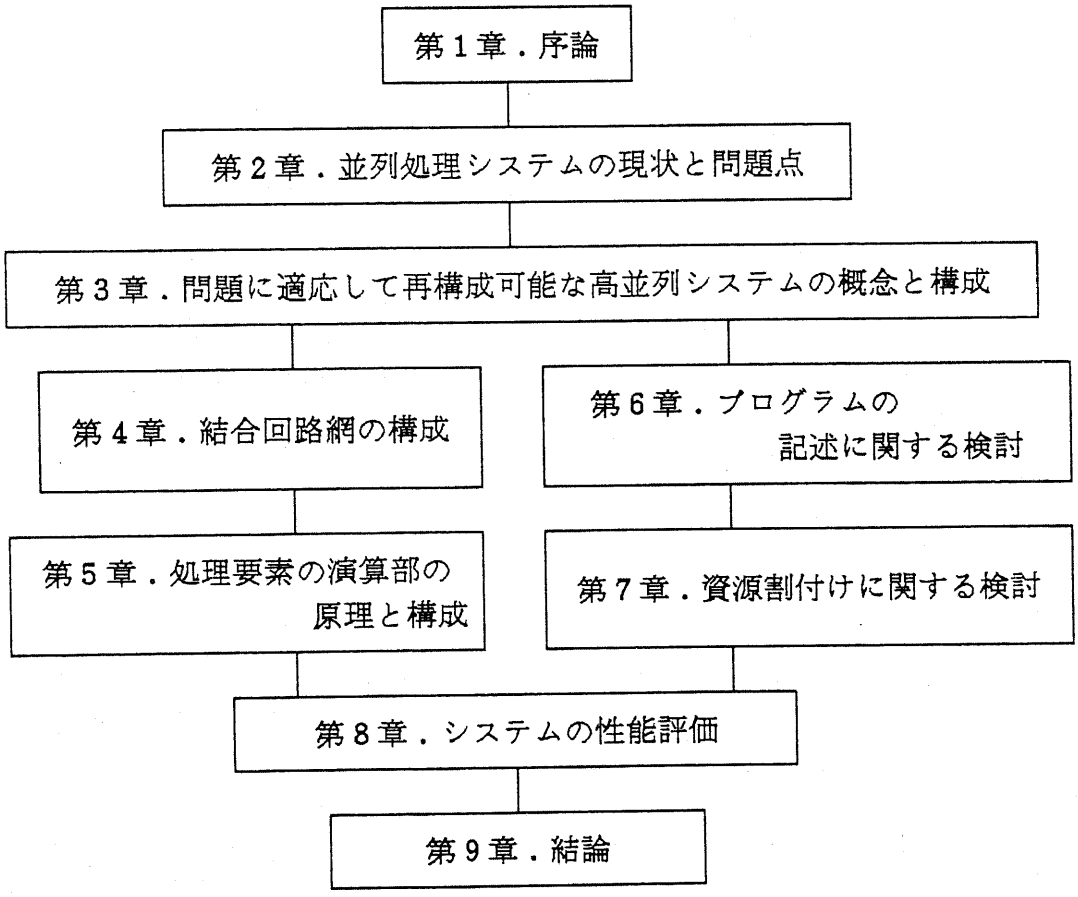


図１－１．論文の構成

第2章

並列処理システムの現状と問題点

本章では現在の並列処理技術とその問題点を明らかにし，本研究の位置付けを行なうために各種の並列処理アーキテクチャを，その汎用性と処理能力に着目して整理し検討を行なう．次にこの問題点を解決するためのアプローチを示し，最後に計算処理の統合化の立場から本研究の重要性を明らかにする．

2.1.各種の並列処理アーキテクチャ

各種の並列処理アーキテクチャを汎用性と処理能力をパラメータとして整理すると図2-1のような分布になると考えられる．

汎用処理システムとしては汎用マルチプロセッサやデータフロープロセッサ，専用処理システムとしてはアレイプロセッサやシストリックプロセッサ，ベクトルプロセッサなどがある．以下に，これらの各プロセッサについてその構成と具体例を示す．

(1)汎用マルチプロセッサ

汎用マルチプロセッサは複数の汎用プロセッサから構成され，何らかの形で全てのプロセッサからアクセス可能なメモリを持っている．代表的な例としてCm*やC.mmpなどがある[Jones 1980]．一般にこのような並列処理システムではプロセッサ台数が多くなると共有メモリへのアクセスがボトルネックになって処理能力が飽和してしまう．プロセッサ台数は物理的にも100台程度が限度である．

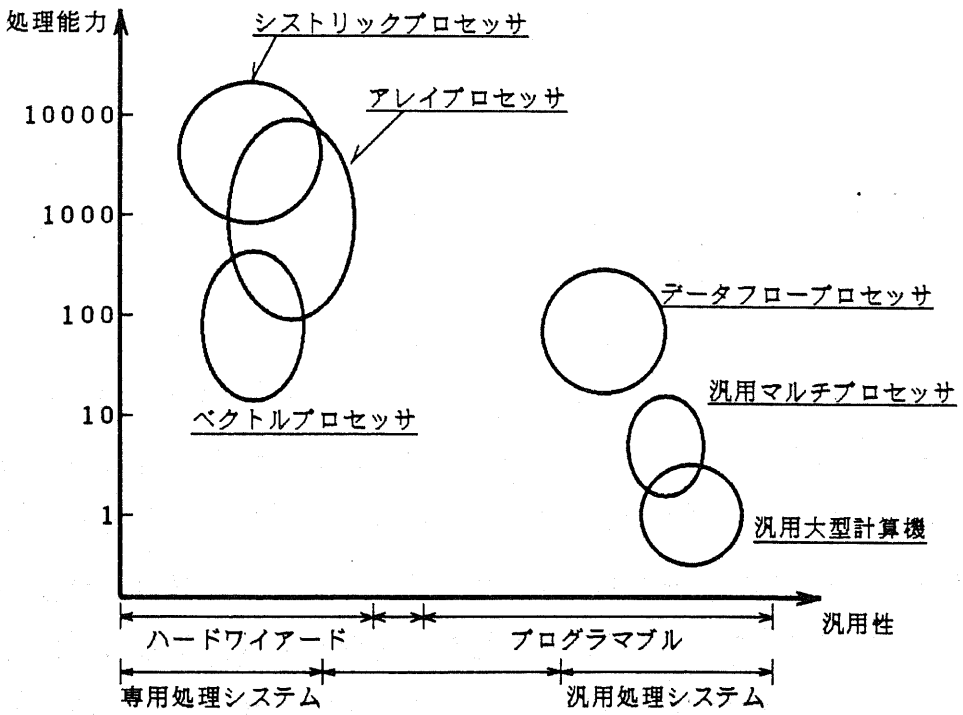


図2-1．各種の並列処理プロセッサの汎用性と処理能力

⑵データフロープロセッサ

データフロープロセッサは，従来のプロセッサとは異なる動作原理を持つ．これはデータ駆動原理と呼ばれ，処理単位ごとに必要なデータが揃ったものから非同期に実行が開始されるため，自然な形で並列処理が実現できるというものである．

処理単位のレベルが通常の演算程度であるシステム例としては，

①ＤＦＭ[Dennis 1980]：ＭＩＴ（図２－２）
②Ｕ-Interpreter[Gostelow 1980]：ＵＣ Irvine（図２－３）
③データフロープロセッサアレイ計算機[Takahasi 1984]：武蔵野通研（図２－４）
④ＩｍＰＰ[Nikkei-E 1984.4]：日本電気（図２－５）

また，例えばＤＯループのようなより大きな処理を単位としたマクロデータフローシステムの例としては，

⑤Cedar[Kohara 1984]：Illinois 大学（図２－６）

がある．

①はデータフロー原理を追求し，科学技術計算向けに汎用性を求めたものであるが，制御が集中しているために，実質的な処理能力はあまり高くはならず，並列動作の多重度で数十～数百が限度と考えられる．

②は，階層構造の下位でアプリケーションの局所性を生かし，上位で柔軟性を高めようという方針で検討されているため，グローバルな構造を持つアプリケーションに対しては著しく効率が低下する可能性があり，また，上位レベルで数百もの並列性を持つアプリケーションはそれほど多くないと考えられるので，一般的な応用に関しては数十程度の多重度が限界であろう．ただし，プログラムの構造は実現しやすくなっているので，汎用性はかなり高い．

③はトポロジーだけ見るとアレイプロセッサと区別がつかないが，データの自動的な転送機能がシステムに作り込まれている点で区別できる．応用を科学技術

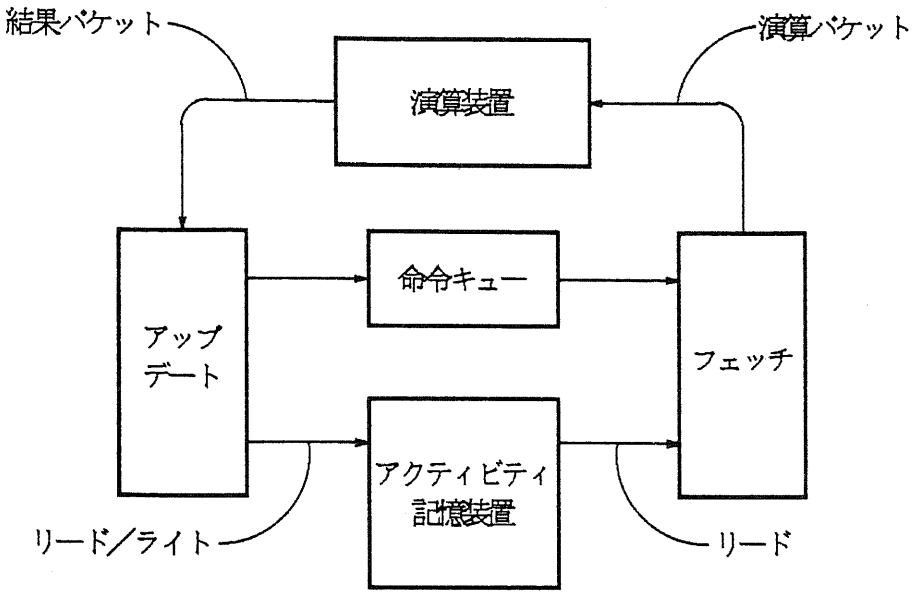


図２－２．ＤＦＭの基本構成

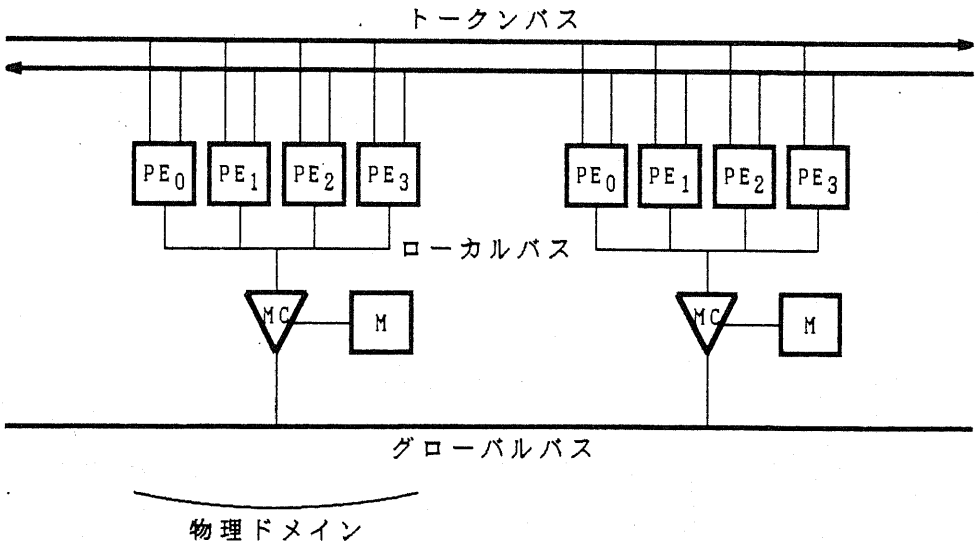


図２－３．U-Interpreterの構成例

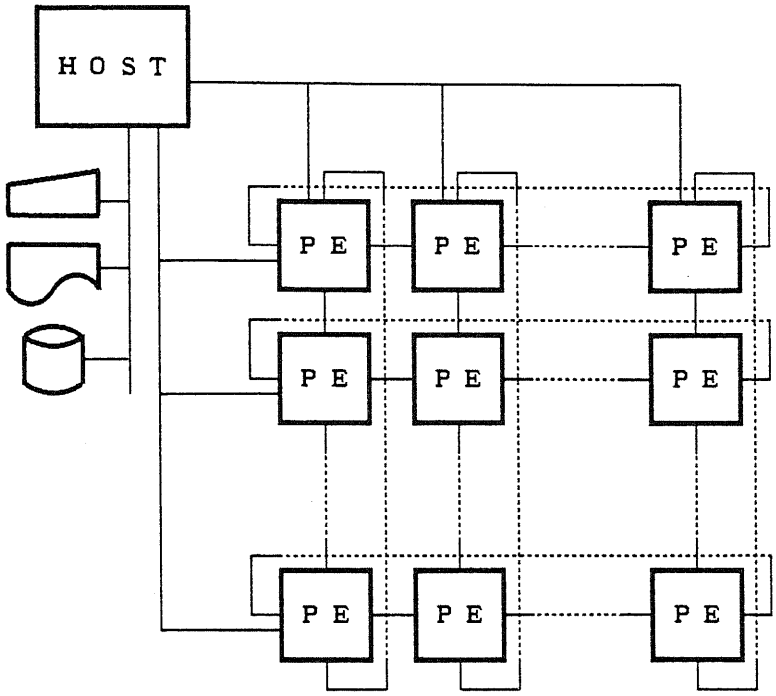


図2-4．データフロープロセッサアレイ計算機の構成

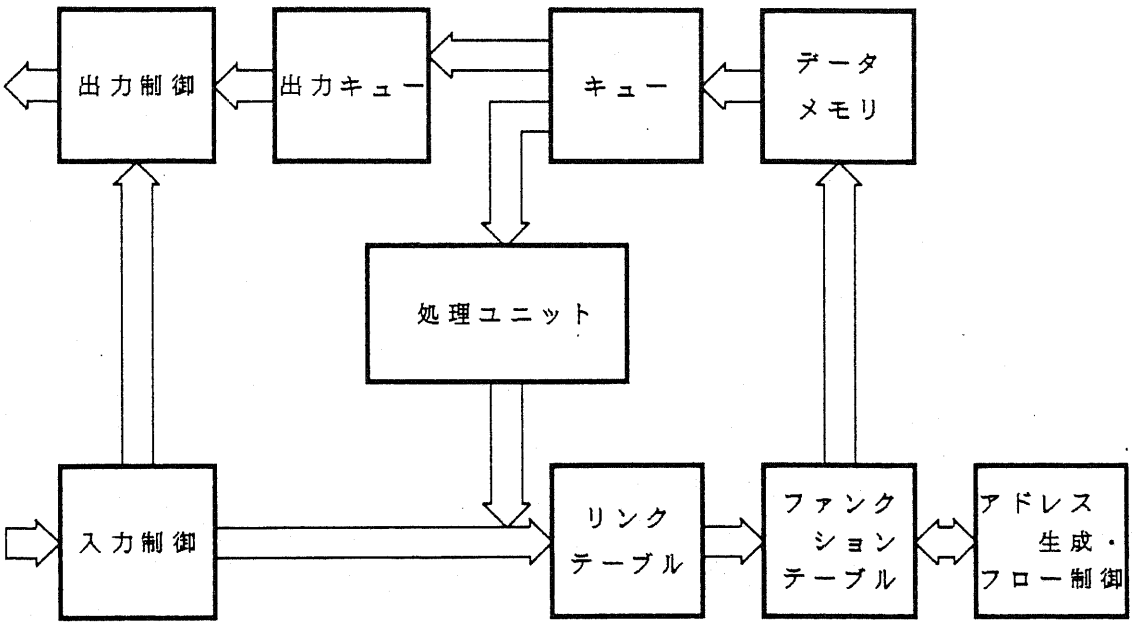


図2-5．ＩｍＰＰの構成

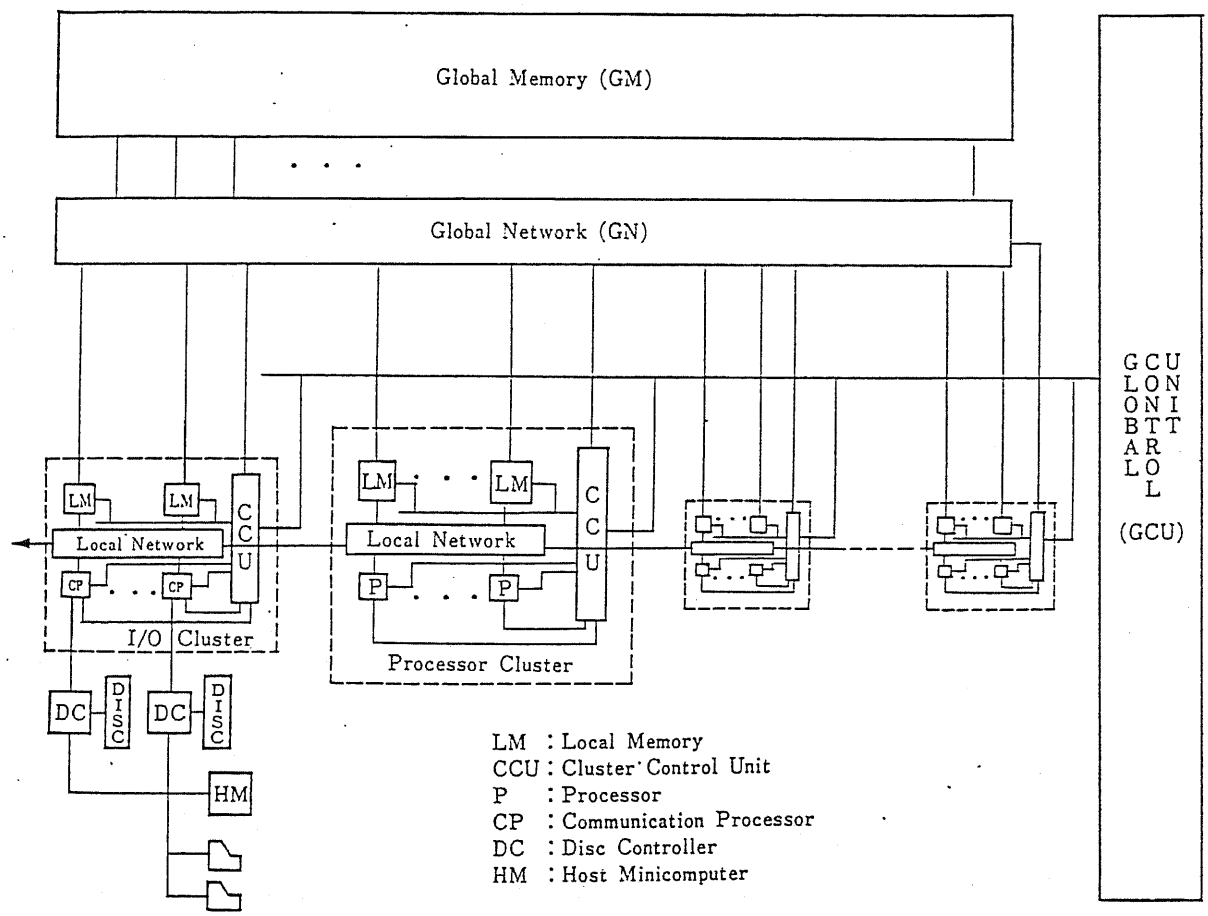


図２－６．Ｃｅｄａｒの構成

計算，特に偏微分方程式に絞り込んでいるので，この応用には高い能力を示すと予想される．

④は画像処理専用のプロセッサとして開発されたものだが，データフロー原理に従うため，他の応用にも使用できる．しかし，構造が，基本的には一次元のパイプラインであるので，画像データの高い並列性（例えば1024×1024以上）を活かしきることができない．また，画像処理のように処理要素の演算レベルが低い応用には，データフローはオーバヘッドが大き過ぎ，処理内容に比較してプロセッサの構造が複雑になっている．さらに，厳密な意味ではデータフロー原理に従っていない点もあり，アプリケーションの構造をうまく利用した専用機を構成するためには，厳密なデータフロー原理を用いることがあまり効率的でないことを示している例と言える．

⑤は高並列プロセッサ間の同期にデータフロー原理を用いようというもので，各処理要素内は逐次処理である．どのようなアプリケーションを想定しているのか不明である．

データフロープロセッサは演算処理に関してはフォンノイマン型の計算機と同程度の汎用性を持っているが，入出力等にはまだ解決すべき問題点が残る[Oyama 1984.1][Gostelow 1979]．プロセッサの台数は主に結合回路網により制限を受けるため，これが性能上のボトルネックとなる．汎用の結合回路網を用いたものでは数百台程度が限界である．

⑶ベクトルプロセッサ

ベクトルプロセッサは最近スーパコンピュータとして脚光を浴びている計算機の中枢をなすものである[Kozdrowicki 1980]．

この分類に属するシステムとしては，1970年代後半にCDCのSTAR-100やTIのASCが登場し，1975年にCRAY-1，1976年にCDCのCyber203，1977年にはバロースのBSPがアナウンスされ，商用化の時代に入った．最近ではCRAY X-MP，Cyber205，富士通のFACOM VP，日立のHITAC S-810，日本電気のSX-2やCRAY-2が登場してきている[Nikkei-E 1983.4][Nikkei-E 1983.5][Nikkei-E 1984.11]．

基本的構造を図２－７に示す．

上記のコンピュータは，それぞれＯＳを持ち，汎用システムであると考えられるが，この汎用性はスカラプロセッサによるものであり，ベクトルプロセッサ自体はパイプライン処理によりベクトル化された浮動小数演算を高速に実行する単能プロセッサである．

高速化の原理はＭＩＳＤ型処理の一種の演算内部のパイプライン化である．浮動小数演算を細かい処理のステップに分け，データを次々と流し込んでパイプライン動作をさせるとともに，各ステップの処理時間（マシンサイクル）を短くして，等価的に１マシンサイクルに１個の演算を実行させるものである．

パイプラインを並列に複数持つ構成を取るものの多いが，高々４～８本で，またパイプラインのステップ数も十数～数十が限度で，並列化は逐次処理の100倍程度が限界である．しかし，データの流れが一次元的であり，従来の逐次処理型のプロセッサとのなじみも良いので，実用化は最も進んでいる．

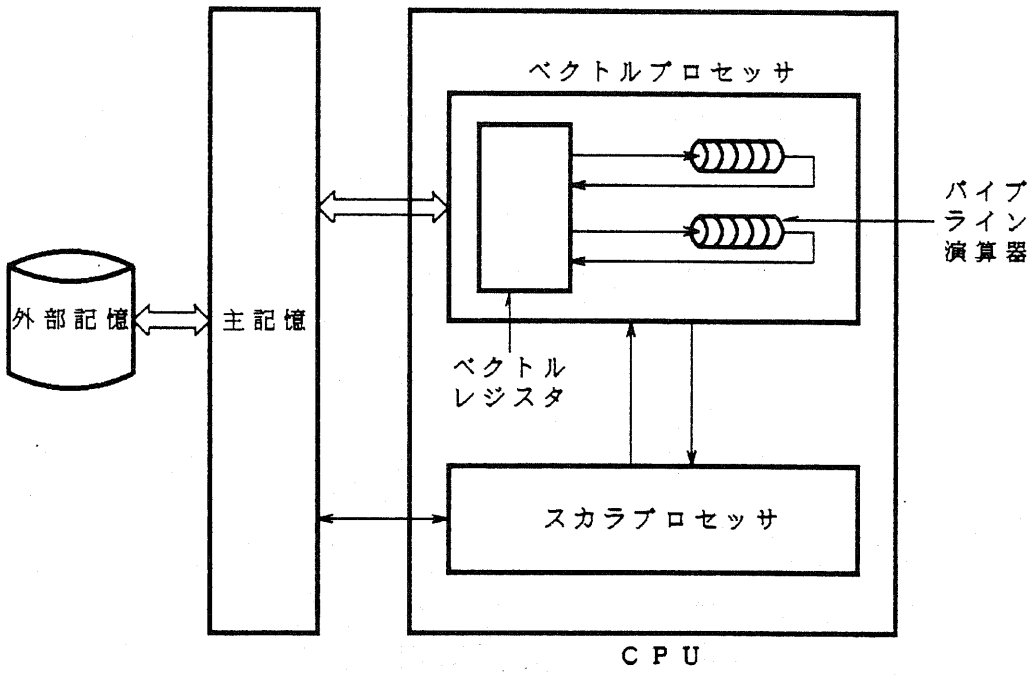


図２－７．ベクトルプロセッサの構成

(4)アレイプロセッサ

アレイプロセッサの概念は古くからあり，実際のインプリメントの例として，ILLIAC IVが有名である[Bouknight 1972]．このILLIAC IVは実用的にはあまり成功しなかったため，アレイプロセッサの潜在的な能力は認識されながらも，その後しばらくはあまり活発な研究は行なわれていなかった．しかし1970年代終わり頃から半導体技術の向上と高並列処理に対する要求により再び注目されるようになり，新しいアーキテクチャも提案され，大規模なシステムのインプリメントも始まった．

アレイプロセッサは基本的にはＳＩＭＤ型のものが大部分であり，図２－８のような構造を持つ．

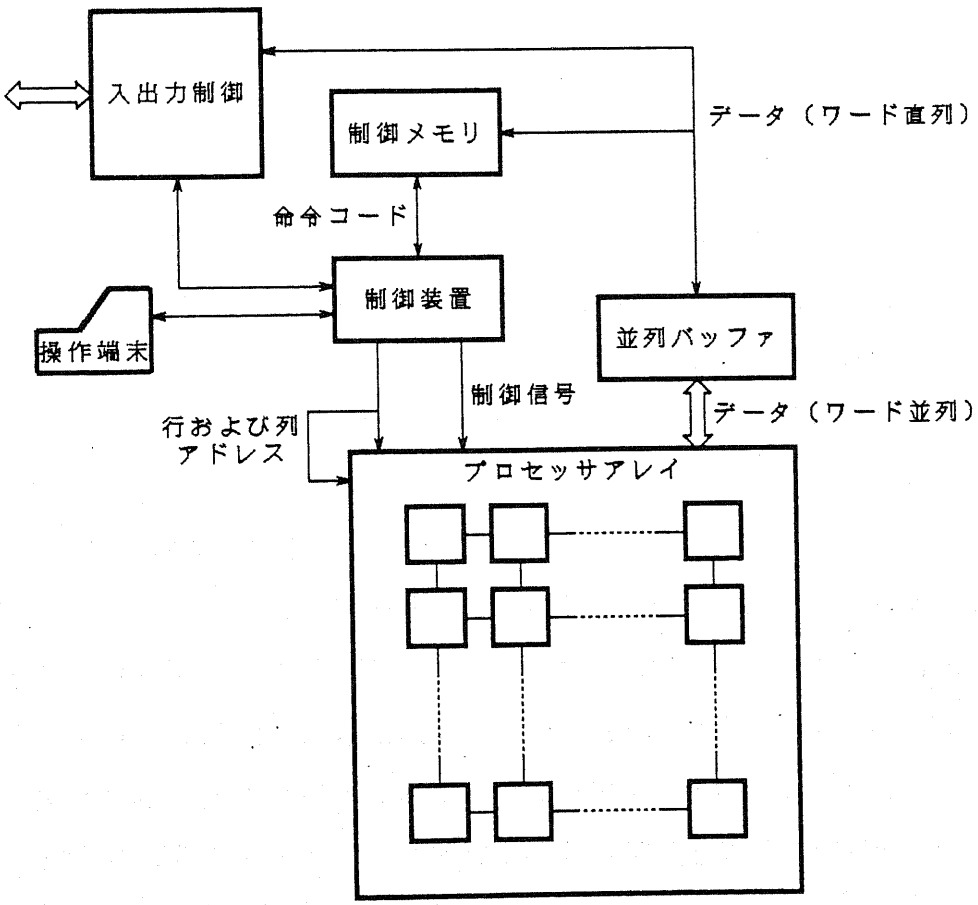


図２－８．アレイプロセッサの全体構成

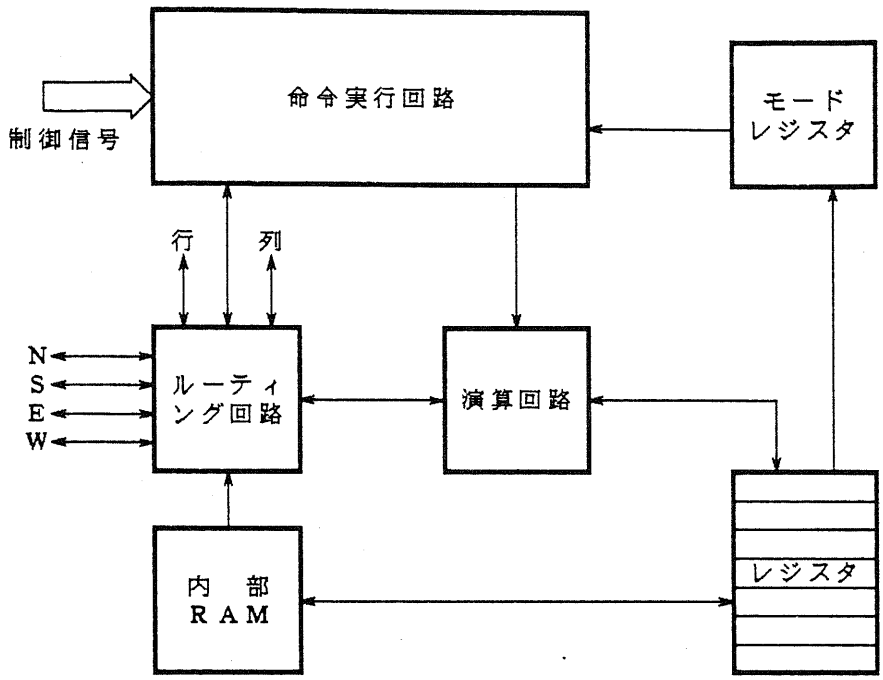


図2-9．アレイプロセッサの処理要素

同一の多数の処理要素がアレイ状に配置され，同一の命令を並列に複数のデータに対して実行する．多くのシステムでは処理要素は隣接するもの同士が結合され，データの通信を行なう．多数の処理要素は集中制御されるため，各処理要素は命令フェッチや命令解析機能を必要としない．また，同一の処理要素が多数結合されているため，耐故障性を高く保てる可能性を有する．各処理要素は典型的には図2-9のような構造になっている．

アレイプロセッサの応用分野として最も良く研究されているのは画像処理への応用であり画像処理専用マシンとして提案されているものも多い[Ericsson 1983][Sakaue 1984]．画像処理では，局所的な処理が多いため，アレイプロセッサの構造とのマッチングが良いことがこの理由として考えられる．しかし，その他にも信号処理やデータベース管理，行列演算，偏微分方程式などさまざまなアプリケーションも研究されている．

現在研究が進められているアレイプロセッサの代表的な例としては，

①ＭＰＰ[Batcher 1982][Potter 1983][Schaefer 1982]：ＮＡＳＡおよび Goodyear Aerospace，128×128，ＳＩＭＤ（図２－１０）

また，全体を分割し，それぞれの部分においてローカルなＳＩＭＤ型処理ができるように構成したものとして，

②ＰＭ'[Briggs 1979]：Purdue大学，ＳＩＭＤ～ＭＩＭＤ（図２－１１）

さらに，単純で規則的な処理要素配置はそのままに，各処理要素を独立動作可能としたＭＩＭＤ型の並列処理システムとして提案されているものに，

③ＣＨｉＰ[Snyder 1982]：Purdue大学，16×16～256×256，ＭＩＭＤ（図２－１２）

といったシステムがある．

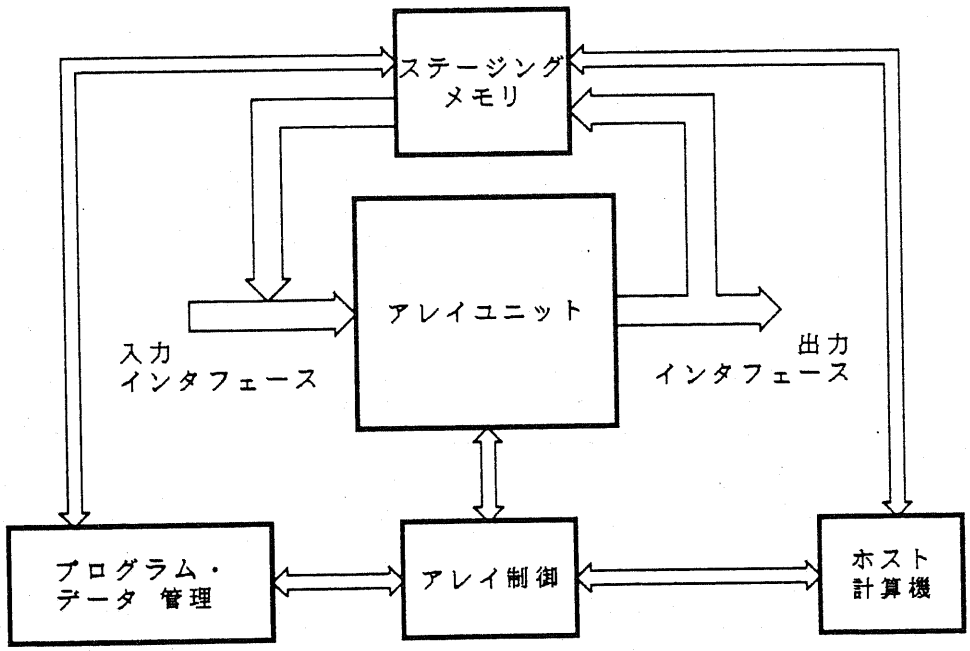


図２－１０．ＭＰＰの構成

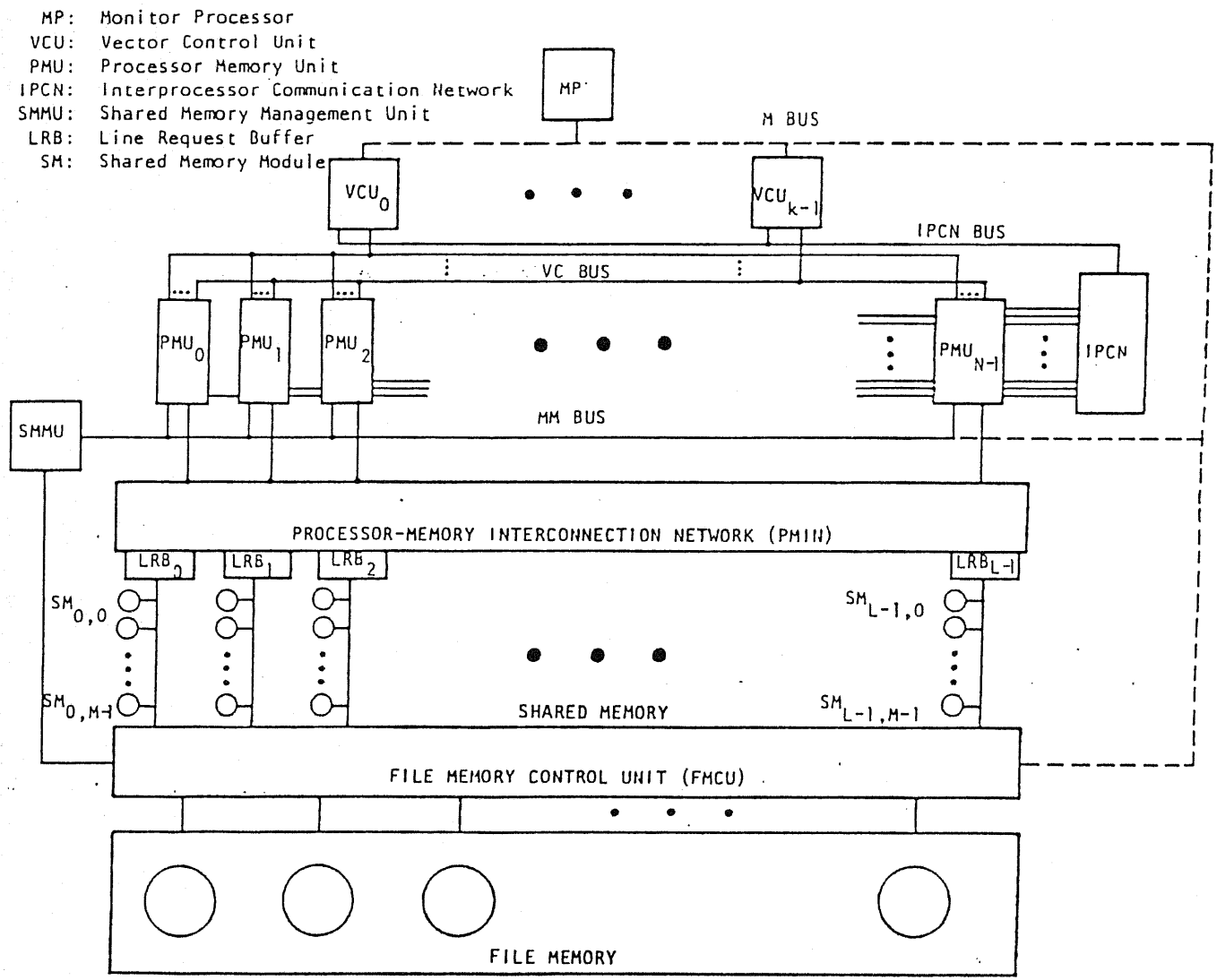
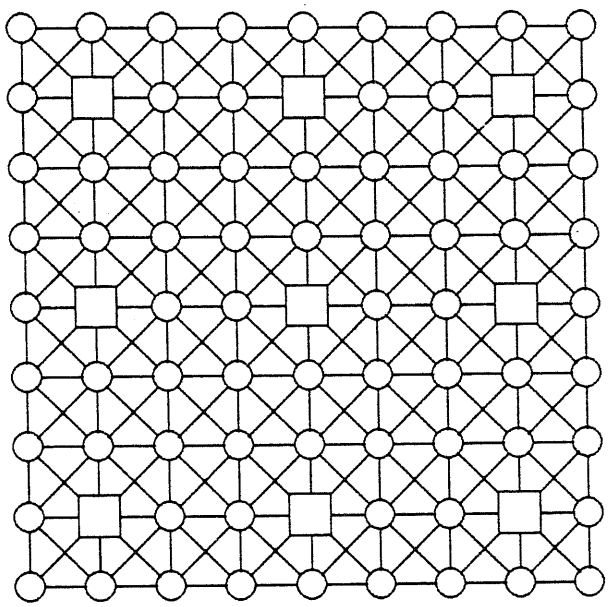

MP: Monitor Processor
VCU: Vector Control Unit
PMU: Processor Memory Unit
IPCN: Interprocessor Communication Network
SMMU: Shared Memory Management Unit
LRB: Line Request Buffer
SM: Shared Memory Module
MP
M BUS
VCU0
VCUk-1
IPCN BUS
VC BUS
PMU0
PMU1
PMU2
PMUN-1
IPCN
MM BUS
SMMU
PROCESSOR-MEMORY INTERCONNECTION NETWORK (PMIN)
LRB0
LRB1
LRB2
LRBL-1
SM0,0
SM0,M-1
SM L-1,0
SM L-1,M-1
SHARED MEMORY
FILE MEMORY CONTROL UNIT (FMCU)
FILE MEMORY

図2-11．PM⁴の構成



○ : switch

□ : processing element

図2-12．CHiP計算機の構成

この他にもアレイプロセッサに関する提案は数えきれないほどある．

①のタイプでは，制御は完全に集中化した制御装置（CU）が行なうため，制御装置が制御できる範囲内で任意の処理が行なえるという意味では，ベクトルプロセッサよりは汎用性が高い．しかし，制御装置が制御できたとしても，並列性が得られないような使い方では，かえって通常のSIMD型のプロセッサの方が処理能率が高いので意味が無くなる．また，処理要素間の接続が局所的なので，局所性のない応用に対しては効率がひどく低下する．従って，少なくとも数十～数百の局所的なデータに対して同一の命令を実行することが意味を持つような応用に適応範囲が限定されるため，あまり汎用性は高くない．しかし，処理要素の構造が簡単であり，拡張も比較的容易であるので，1024×1024位のアレイは近い将来に可能であると考えられ，極めて高い処理能力が得られる．

②のタイプでは①のような構造をより細かい部分に分割し，各部を独立に制御できるようにすることにより適応範囲を広げようというものであり，基本的な概念や動作原理は①と変わりない．

③のタイプでは，制御は各処理要素に分散しているため，プログラムさえ記述できれば，並列性を持つさまざまな応用に対応できる．データ依存性等が必ずしも規則的でなくとも効率は高く保たれる．しかし，データの局所性は①ほどではないにしろ重要であり，また構造が動的に変化するような応用に対しては効率が著しく低下する可能性がある．処理要素の構造も通常のマイクロプロセッサほどではないにしろ複雑になるため，①ほどは大規模なアレイは作れず処理能力は数分の一～数十分の一になるが適応範囲はかなり広くなる．

⑸シストリックプロセッサ

シストリックプロセッサは，シストリックアルゴリズムと呼ばれる2ないし3方向のデータストリーム間の繰り返し演算を，1あるいは2次元のプロセッサアレイ上で行なうもので，行列演算，データベースの関係演算，ソーティングなどのように処理を回帰式で表現できる問題を効率よく実行する[Kung 1982]．処理がいずれも回帰式で表現され，数学的な裏付けがなされているため専用システムとしてインプリメントしやすい．例として行列演算等に対して非常に強力な処理能力を持つ六方格子状配列のシステムを図2-13に示す[Mead 1980]．回帰式の

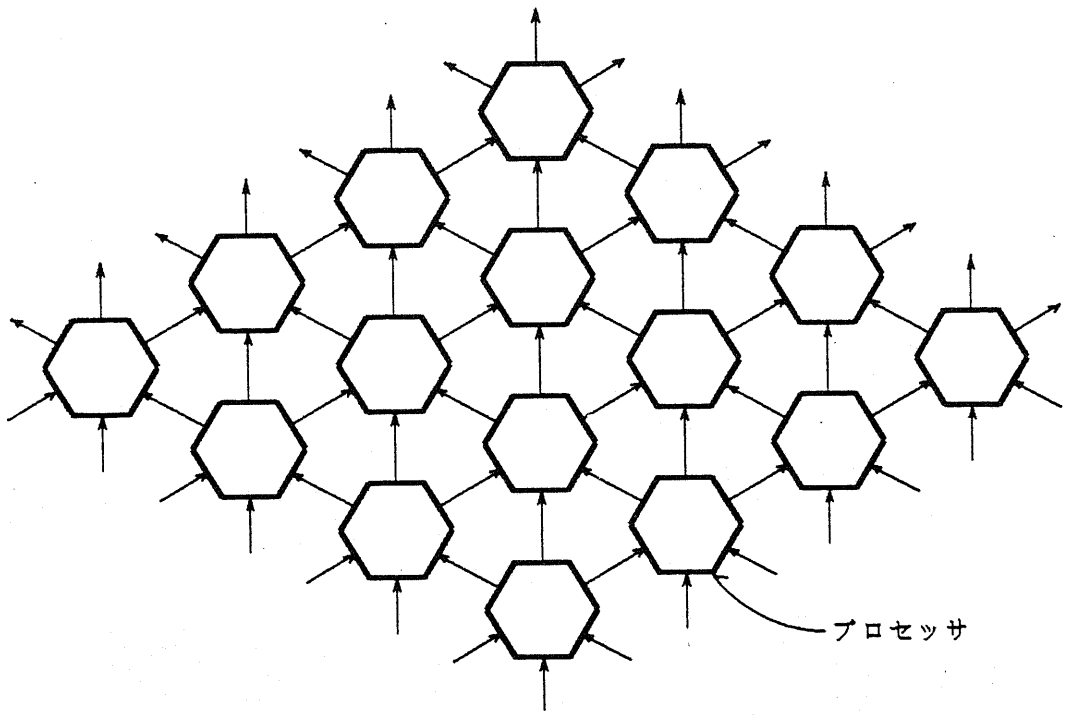


図2-13．6方格子状シストリックプロセッサの構成

形式によりプロセッサアレイの結合形態が決まっており，プロセッシングエレメントをプログラマブルにすることで同一の形式の回帰式で表現される範囲の問題には対応することができる[Fisher 1983-1]．しかし，結合形態や，データの流れの方向が固定されているため，同一のシステムを異なるデータ依存関係を持つアプリケーションに使用することは困難であり，専用処理システムに分類される．構造が単純で，入出力と処理がオーバラップしていることにより，大規模化が容易で，極めて高並列なシステムの実現も可能であることから，特化したアプリケーションに対しては高い処理能力が得られると考えられる．

以上に高並列処理システムのアーキテクチャについて述べたが，図2-1の各分類を汎用性を基準に比較すると，専用処理システムおよび汎用処理システムの分野でさまざまなシステムが研究されているが，これらの中間に位置するシステムはあまりなく，ようやく研究が始まった段階である．この部分に対しては汎用処理システム側と専用処理システム側からのアプローチがあり，これについては2．2および2．3節において述べる．

2.2.汎用処理システムのからのアプローチ

一般に汎用マルチプロセッサやデータフロー計算機のような汎用処理システムは現在のメインフレームと同程度の汎用性を目標とし，かつ高い処理能力を目指している．しかし，これらを両立させるのは非常に難しい．この主な制約条件は汎用の結合回路網にある．汎用結合回路網では任意の処理要素間の通信を完結した機能として提供せねばならず，応用分野に内在するデータ依存性の特質をいかせないため遅延が大きくなると同時にスループットも低く抑えられてしまうことがその理由である．

前節②③，④のシステムはこの汎用性を犠牲にし，より高い処理能力を実現することで，前節で述べたような領域のシステムに対するアプローチを試みたものといえる．このようなアプローチでは共通して，応用範囲を特定分野に限定することによりそれに適合した結合回路網を採用して処理能力を高めようとしている．しかし，同期制御や排他制御のようなプロセッサ間の実行制御は汎用処理システムにおける方式をそのまま継承しているため，専用的な用途に用いるには必要以上に柔軟性が高く，オーバヘッドが大きいため，高並列で効率の良いシステムの実現を困難にしている．

これらのことから，汎用処理システムでは，第一に汎用結合回路網が，第二に汎用制御機構が性能の限界を抑えていることがわかる．

2.3.専用処理システムからのアプローチ

アレイプロセッサやシストリックプロセッサのような専用処理システムでは非常に高い処理能力が期待できるにもかかわらず，比較的限られた分野でしか実用化されていない．この理由としては，

①問題の定型化，すなわち並列処理アルゴリズムが確立していなければならない．

②専用処理システムの開発には高いコストとリスクが伴う．

③狭い応用分野においてコストに見合う需要が存在しなければならない．

という点が挙げられる．①はシステムが存在しない状況で，紙の上で問題の解析が必要なうえ，必ずしも計算機ハードウェアの専門家でない者がこれを行なわなければならない点に問題がある．②の問題は，ＬＳＩの自動設計技術やプロセス技術と結びついた製造の自動化が進むにつれて低減されてきてはいるが，ソフトウェア開発に比べると格段に大きい．③は限られた狭い応用分野においてある程度定常的な処理要求を持つアプリケーションあるいはユーザでなければ専用処理システムを利用できない点が問題である．

②を解決するために，プログラム可能なシストリックプロセッサチップが研究されている[Fisher 1983-1]が，これだけでは適用可能な応用の範囲を広げるためには不充分である．

このような問題点を解決するために，2.1.節で述べたような領域のシステムに対するアプローチが必要となる．これには，アレイプロセッサにおいて結合回路網を柔軟な構造にし，同時に演算実行制御機能を並列化することにより適応可能な応用範囲を広げ，システム全体のスループットを上げようというものがある[Briggs 1979]．これらのシステムでは演算実行制御機能は基本的には従来と変わりなく，結合回路網の柔軟性を充分生かせるかどうか疑問である．本論文で提案しているシステムもこのアプローチをとるものであるが，結合回路網と演算実行制御機能の独立性を高め，柔軟性を増している点が特徴である．

2.4.問題に適応可能な結合回路網による汎用化アプローチ

前述のように並列処理システムの効率は結合回路網に大きく依存する．そこで専用処理システムの結合回路網における固定接続方式の利点を持ち，かつある程度柔軟性の得られる構成として，静的な回線接続方式が考えられる．

従来の汎用処理システムは蓄積交換か回線交換の方式を採る，いわば動的な接続方式がほとんどであった．しかし，高度の並列性を有する応用においては，問題に対して並列処理アルゴリズムを選択した段階で，データ依存関係が定まる場合が多い．このような条件を満たす問題に対しては，プログラム実行前に，通信する必要のある処理要素間に予め静的に物理回線を接続しておけば動的接続は必要なく，専用処理システムの固定接続に近い高いスループットが実現できる．

このような思想に基づくシステムとして，2.1節④③に示したCHiPコンピュータと呼ばれるシステムが研究されている[Snyder 1982]．図2-12に示したように処理要素とスイッチが格子状の配列をなし，スイッチをプログラムすることにより処理要素間を回線接続する構造になっている．処理要素と処理要素の間に置くスイッチの数により，接続の柔軟性を変えることができる．

このシステムの特徴としては，

①静的な回線接続による通信
②回線の両端における同期式通信
③格子状配置
④処理要素とスイッチの分離
⑤ＭＩＭＤ動作

が挙げられる．この格子状配置での一つの問題は，回線長が一定でなく，距離の離れた処理要素間の信号遅延が大きくなり，システムの動作クロックもそれに伴って遅くせざるを得ないという点である．

また，スイッチはトランジスタのみでできるとしているが，距離の離れた処理要素間の接続には，途中にバッファが必要になるので，そう簡単ではない．

しかし，この構成は充分定型化された並列処理アルゴリズムへの適応性が高く，このアプローチの有効性を示している．

2.5.計算処理サービスの統合化における位置付け

本章で述べてきたように，さまざまな並列処理アーキテクチャが研究されているがそれぞれに問題点もある．これらの問題点を補う方向として，さまざまな汎用あるいは専用処理システムを一つの制御機構に組み込んで，複合計算処理サービスシステムを構築することが考えられる．

これまでは，多くの並列処理システムは特定のホスト計算機の内部バス，あるいはチャネルに直結され，他の並列処理システムと組み合わされて使用されることはほとんどなかった．この理由の一つとしては，システム間を結合する高速の媒体が無かったことが挙げられる．このため，各システムは閉じた世界を形成してしまい，広い分野へ適用される機会が奪われるばかりでなく，汎用ではあっても専有処理システムとなっている例が多い．また，比較的小規模な処理から段々と大規模に発展してゆくような応用分野における発展過程に対しても，処理システム側からのアプローチはほとんどなかった．

しかし，ローカルエリアネットワークに代表されるように，システム間の高速通信媒体が一般的になってきたことにより，さまざまな汎用処理装置や専用処理装置を統合化できる可能性が生じてきた．

このようなシステムが可能となると，さまざまな分野において，さまざまな人々が，処理の規模や内容に応じた装置の選択をすることが可能となり，資源の有効利用と処理装置の発展に対して新しい展望が開けると考えられる．

ここで，2.1節で述べたような汎用処理システムと専用処理システムとの格差が問題になる．応用によっては，汎用処理システムでは処理能力が不足だが需要がそれほど大きくなかったり，あるいは需要は大きくとも処理内容が充分に定型化されていなかったりして，専用処理システムの構築が不適当であったりあるいは不可能である場合もある．また，専用処理システムを作れる程度に応用分野が発達してきても，専用処理システムの開発には膨大な時間とコストがかかり，常に危険が伴う．

本論文で提案しているシステムはこの格差を埋めるものとして位置付けられ，計算処理の統合化という立場から，より一層の重要性を持つものである．

第3章

問題に適応して再構成可能な高並列処理システムの概念と構成

本章では，システム各部の詳細な検討に先立ち，システム全体の構成上の概念を明確にし，基本的な方針を明らかにする．まず本システムのハードウェアおよび機能上の全体構成について述べ，結合回路網と演算部の構成方法に関する基本的方針と概念を示す．次にデータの入出力に対する考察を行ない，ホスト計算機による制御と故障への対応について簡単に触れ，最後にシステムソフトウェアの全体的構成を示す．

3.1. システムの構成と動作原理

本システムは，データ依存関係を抽出することにより各問題に適応可能な結合回路網を持ち，処理要素間の演算実行制御にデータ駆動制御を採用した点を特徴とする，問題に適応して再構成可能な高並列処理システムである．

処理要素の配置と結合方法についてはさまざまな形態が考えられる．従来は配線コストに比べスイッチのコストが高かったためにスイッチを集積化する傾向にあった．これは制御は容易であるが，主に配線の複雑さの問題により処理要素の数が制限され，高並列処理システムの実現を困難にしていた．しかし半導体の高集積技術によりスイッチのコストが低くなったため，多数のスイッチを分散して使用することが容易になってきた．

一方，問題が内包するデータ依存関係はランダムではなく，一般には規則性や局所性を持っているものが多い．高い処理能力を実現するには，この性質をうまく利用することが重要である．

これらのことを考慮すると，高並列処理システムには階層構成と2次元格子状構成が適する．階層構成にはさまざまな形態があるが，一般に上位の回路網では配線長が長くなるため，データ伝送があまり高速化できない．さらに応用問題のデータ依存性とシステムの構成のマッチングが悪いと上位回路網にトラヒックが集中し，ボトルネックとなりやすい．

そこで，本システムでは2次元格子状の静的な回線接続方式をとることにする．2次元格子状構成は処理要素間の配線長，配線面積とも小さく，ＶＬＳＩシステムに適している．システム全体の大きさも処理要素数に比例し，大規模システムが容易に作れる．結合回路網の機能も各処理要素内に分散するため，完全に単一のチップでシステムを構築できる点もＶＬＳＩシステム向きである．

この全体的構成と結合回路網形態についての検討は第4章に詳しく行なう．

システムの全体的構成を図3－1に示す．2次元格子状に配置した処理要素と，それらを結ぶ結合路により構成される．

システムの規模は，処理要素数にして64×64ないし256×256程度を考えており，8×8あるいは16×16程度をドメインと呼び，各ドメインにはドメインコントローラ（ＤＣ）を置く．

各処理装置には8ビット程度の演算器を持つ専用プロセッサを想定する．ドメ

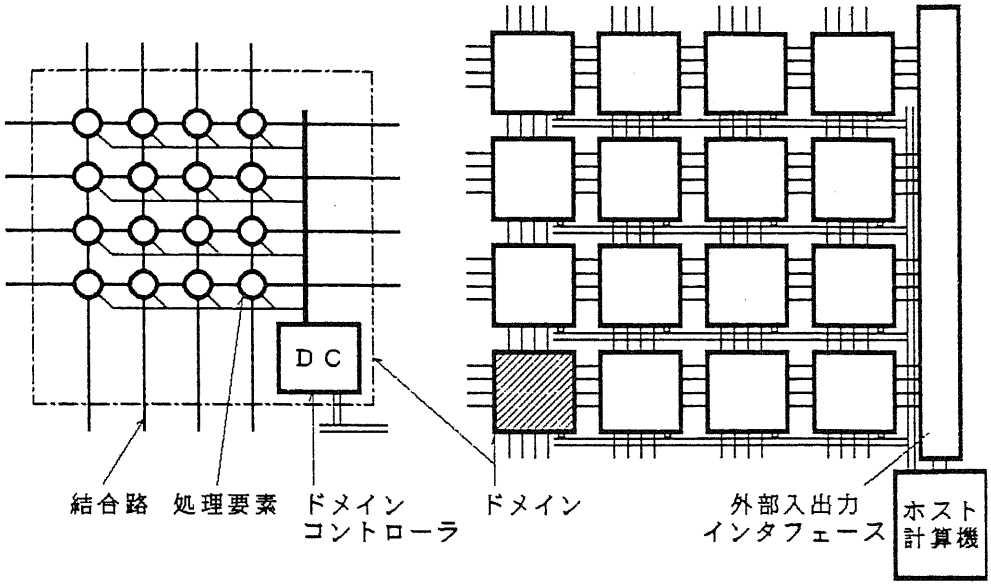


図3-1．システムの全体的構成

インコントローラには16ビットないし32ビット程度の汎用のマイクロプロセッサが適当であろう．

現時点では1処理要素を1チップ上に実現することを仮定しているが，将来的には1チップまたは1ウェーハ上に1ドメインを実装することも考えられる．

本システムを構成する機能を大別すると，

(1)ホスト制御機能
(2)回線接続機能
(3)通信制御機能
(4)演算実行制御機能
(5)演算処理機能
(6)外部データ入出力機能

となる．

(1)のホスト制御機能には，システムの初期化と後処理および故障時の制御が含

まれ，ホスト計算機からドメインコントローラを介して行なう．

⑵と⑶の通信に関する機能と，⑷と⑸の演算に関する機能は論理的に互いに独立しており，⑵と⑶を実現する部分を結合回路網，⑷と⑸を実現する部分を演算部とよぶ．結合回路網は通信のための回線を保持し，ある処理要素の演算部からデータを受け取って回線上を伝送し，目的の処理要素の演算部へ渡す．この動作は入出力ポート，スイッチマトリクスと結合路により実現され，プログラム実行中は演算部とは独立して動作する．各処理要素の演算部は結合回路網によってのみ互いに通信可能であり共通メモリやバスのようなものは持たない．また，プログラムやデータは演算部内に記憶されており，プログラム実行中にホスト計算機やドメインコントローラがプログラムの実行に関与することはない．

⑹の外部データ入出力機能はホスト計算機の管理下にある外部入出力装置とプロセッサアレイ間のデータの入出力を実現するもので，入出力インタフェースに対してホスト計算機が指示を与えることにより実現する．

プログラムの実行のためにはさまざまな前処理と後処理が必要であり，それに応じてシステムの動作モードには以下のようなものがある．

・プログラムロードモード
・データロード，アンロードモード
・ネットワークモード
・演算モード

ドメインコントローラはこれらのモードを指定して，処理要素をプログラムしたり演算実行させたりする．

プログラムロードには

・スイッチマトリクスに対するスイッチパターンのロード
・入出力ポートに対する入出力管理表のロード
・ＰＭ管理レジスタおよびＰＭ管理装置に対するプログラムモジュールのロード

・命令メモリに対する命令ストリームのロード

が含まれる．

データロード，アンロードには

・ＰＭ管理装置に対する定数データのロード
・入出力レジスタに対する初期データのロード
・入出力レジスタからの結果データのアンロード

が含まれる．これら各部の機能については3.2および3.3節に述べる．

ネットワークモードでは，プロセッサアレイの演算部は動作させず，結合回路網のみを動作させる．このモードを用いることで外部入出力インタフェースを通じて全処理要素にデータを転送したり，逆に全処理要素からデータを回収したりすることが容易になる．これは特にデータの配置が複雑な場合に，並べ替えを行ないながら入出力を行なえる点が有効である．

演算モードでは通常のプログラムの実行を行なう．

3.2.問題に適応可能な結合回路網の概念

並列処理システムの結合回路網にはさまざまな構成が考えられるが，各処理要素の規模が小さいため処理単位も小さくなり，密な結合が必要になることと，処理要素数が非常に大きいことを考慮し，本システムでは2.4節に述べたような静的な回線接続によるデータ依存関係への適応のアプローチをとる．

各処理要素にはスイッチマトリクスが内蔵され，隣接する処理要素のスイッチマトリクスを結合路でつなぐことにより2次元格子網を形成する．また，各スイッチマトリクスには演算部とのインタフェースである入出力ポートも接続する．この構成を図3－2に示す．

通信を行なう処理要素間を結ぶために，結合路をスイッチマトリクスで次々に接続してゆくことにより静的な回線を設定し，これを用いて通信を行なう．

プログラムの実行が進むに従い，一本の回線上をさまざまなデータが流れる．そこで，単に処理要素と処理要素を接続するという意味においては，これを物理回線と呼び，流れるデータの順番や内容までを含めた意味においては，これを論理回線と呼ぶことにする．

物理回線は主にスイッチマトリクスと結合路により構成される．

論理回線の制御は入出力ポートが行なう．各ポートは入出力管理表を持ち，こ

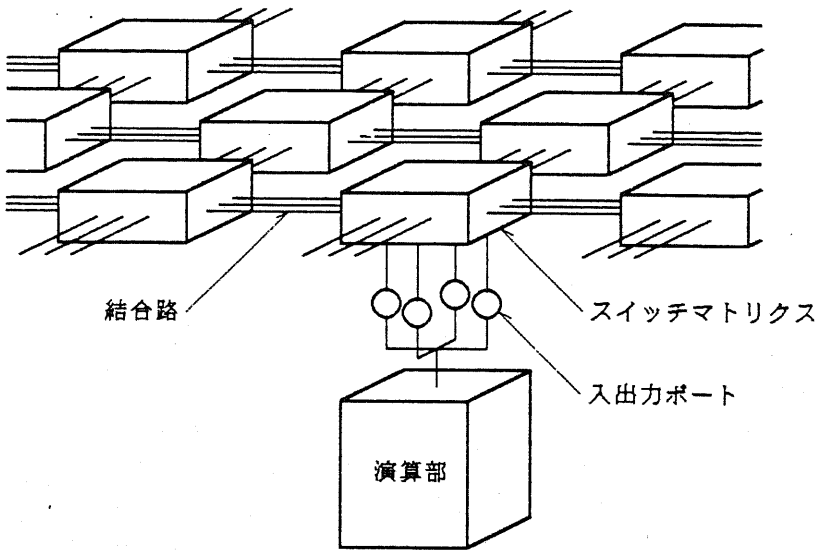


図3－2．結合回路網の構成

の内容に従い順に入出力レジスタにアクセスし，データ長や伝送方向の制御を行ないながらデータを送受信する．

スイッチマトリクスの制御は，プログラムロード時にホストからスイッチ制御レジスタにスイッチパターンをロードすることにより行ない，入出力ポートの入出力管理表も同様にホストからロードするため演算部がこれに関与することはない．動作はすべてハードウェアにより行ない中継によるオーバヘッドを極力小さくする．

これらにより，各物理回線は完全に独立に動作することができ，データ伝送レートも高くすることができるため，複雑なデータ依存関係を持つアプリケーションに対しても，この結合回路網では非常に高いスループットを実現することが可能となる．

3.3.演算部の構成と動作原理

処理要素の演算部の構成を図3-3に示す．

演算部は結合回路網の高速性を生かすよう，実行制御のオーバヘッドを極力小さく抑えた構成とする．

並列処理システムでも，各プロセッサで実行する処理単位のレベルが高ければ，プロセッサ間の通信の頻度はそれほど多くなく，プロセッサの制御のもとに通信を行なってもあまりオーバヘッドは問題にならない．しかし，プロセッサの規模が小さく，従って処理単位のレベルも低いようなシステムでは，全体の処理における通信の比重が大きくなる．

アレイプロセッサやシストリックプロセッサのように，全ての処理要素が同一の処理を同期して実行するようなシステムでは通信も完全に制御された状態にあり，処理要素間の演算の同期や実行制御は考えないで良い．しかし，本システムのようにデータ依存関係がより複雑なアプリケーションまでも扱おうとすると，演算や通信は各処理要素間で非同期に行なわれる必要がある．従って，各処理要素におけるデータの送受信の順序や時期を予め予測することは極めて難しく，これにともなって起動される演算は本質的にデータ駆動的になる．このような演算

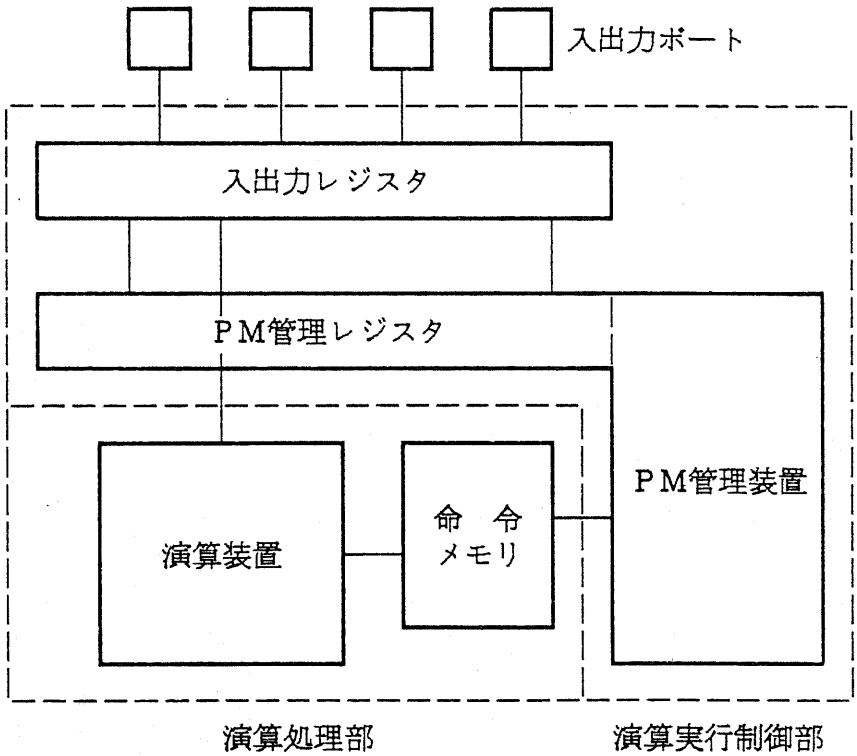


図3-3．演算部の構成

の実行制御を逐次的な制御方法で行なうことは極めて不効率であり，より直接的な制御方法が必要となる．

そこで本システムではプログラムの実行制御を2段階に階層化し，上位の演算実行制御はデータ駆動制御，下位の演算処理は逐次制御とする．通信にともなう演算実行同期制御を上位で行ない，通信を必要としない連続的な処理を下位で高速に実行する．上位プログラムの処理単位をプログラムモジュールと呼び，プログラムモジュールに対応して実行される下位の逐次処理演算の命令列を命令ストリームと呼ぶ．

データ駆動制御の処理単位であるプログラムモジュールではそれぞれデータの入出力に使う入出力レジスタのセットが決められている．データ駆動原理に従い，各レジスタが読み出し・書き込み可能になってプログラムモジュールが実行可能となると，そのプログラムモジュールに対応した命令ストリーム番号がＰＭ（プログラムモジュール）管理装置により命令メモリに渡され，命令ストリームが命令メモリから演算装置に対して順次送られる．演算装置は入出力レジスタや内部レジスタにアクセスしながら与えられた演算を実行してゆく．

各部の詳細な構成と動作については第5章に述べる．

3.4.外部入出力インタフェースの検討

従来の構成の専用処理システムにおいては，演算処理装置の動作速度が速くなるに従いデータの入出力がボトルネックとなってシステム全体の性能を規定してしまうようになってきている．

シストリックプロセッサにおいてはデータの入出力と演算処理が同時に実行され，データの入力が始まると同時に演算も始まり，演算が終わると同時にデータの出力も終わる．従って，見かけ上入出力のオーバヘッドは無い[Kung 1982]．しかしこれは演算装置の能力を入出力に合わせて限定して用いることにより実現されており，性能を規定しているのはやはり入出力である．さらに，例えば2次元六方格子状のシストリックアレイで行列AのLU分解を行なうようなな場合，図3-4に示すように必ずしも一次元的に連続していないデータを並列にかつシステムクロックに同期して入出力しなければならず[Mead 1980]，ホストシステムとのインタフェースが複雑になる．このため，アレイの外にバッファを置くような形をとることになり，このバッファへの入出力がボトルネックになる．

アレイプロセッサにおいては入出力も演算もアレイ全体を集中制御する逐次制御装置から与えられる命令により行なうため，これらは逐次的に行なわれることになる．従って，アレイに対する入出力はそのまま処理のオーバヘッドとなる．また，m×nのデータをアレイにロードするためにはm個のデータを並列にかつ

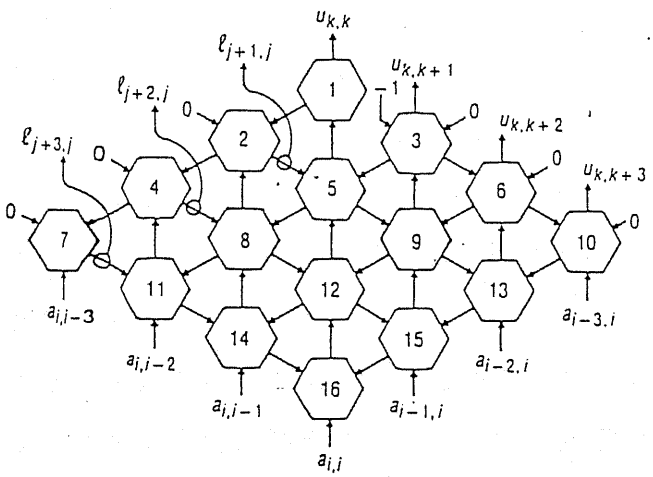


図3-4．六方格子状シストリックアレイによるLU分解

同期してn回入力する必要があり，これを実現するために，例えばＭＰＰではステージングメモリを設けてバッファリングを行なっている[Batcher 1982]．さらにアレイプロセッサでは扱うデータ幅を1ビットとしているものが多いため，ホストシステムの語構成との整合性が悪く，バッファリングを複雑にしている．入出力はホストシステムとバッファ間，およびバッファとアレイ間について行なわれ，両方とも高速化する必要があるため実現が難しい．

ベクトルプロセッサにおいてはフォンノイマン型の計算機の中にベクトル処理装置を組み込んだ形で使用されており[Nikkei-E 1983.4][Nikkei-E 1983.5][Nikkei-E 1984.11]，メモリバスに直結され，主記憶が一種のバッファとして用いられているため，主記憶の語構成やメモリバス速度の制約を受けることになる．メモリバスの制約を越えて，より高速化をはかるためにはパイプラインを縦列に接続する以外に方法がなく，アプリケーションによっては性能を充分に引き出すことが一層困難になる．しかしベクトルプロセッサは1次元的なパイプラインにより構成されるためフォンノイマン型との整合性がよく，商用化されているシステムも多い．また，この種のシステムは総合的なスループットを高くするために半導体拡張記憶装置を実装しているものが多く，外部とバッファとの間の入出力に関しては高速化がはかられている．だが，あくまでもバスという一本の経路を通して全ての入出力を行なっているため，性能の上限は低く抑えられる．

以上に述べてきたように，専用処理システムでは何らかの形でバッファメモリを持っており，外部とバッファメモリとの間，およびバッファメモリとプロセッサとの間の入出力がボトルネックになる．これらはいずれも大量のデータに対して低級な処理を大量にほどこすという高性能化の方針に基づいていることに起因している．本論文の中で提案しているシステムにおいても，上述のようなシステムと同じような使用形態をとるならば同様の問題が発生し，対処方法も類似の方式にならざるを得ないであろう．

しかし，提案システムでは上述のシステムと二つの点で大きく異なる．第一点は演算および通信が全て非同期である点であり，第二点は処理内容のレベルが高い点である．

第一の特長により，外部装置との入出力の目的には大容量のバッファは不要となる．外部装置のデータラインとプロセッサアレイの物理回線とを結ぶことによ

り直接処理要素に入出力できるからである．これにより本システムではＭＰＰにおけるようなバッファとアレイとの間の入出力によるオーバヘッドは無くなり，外部装置とバッファとの間の入出力によるオーバヘッドに相当する部分のみとなる．従って，入出力速度は外部装置との間のデータ転送レートによって決まることになる．外部装置が二次記憶であれば並列化により高速化が可能であるが，ホスト計算機の主記憶の場合はメモリバス速度により制限を受ける．

また，第二の特長により，一回の入出力データに対して行なえる処理内容が多くなるため，相対的に入出力の頻度が下がり，オーバヘッドが性能に与える影響が小さくなる．例えば，行列演算を行なうシストリックプロセッサにおいては，１回の入出力に対して行なえる処理は行列乗算，ＬＵ分解，コンボリューション等の単一の演算であり[Tokura 1983]，入出力のオーバヘッドが全体の処理時間に占める割合を下げることは難しい．これに対して，本システムでは結果が出力ではなくプロセッサ内に残るような並列処理アルゴリズムを採用することにより，一度入力されたデータに対して異なる種類の演算を繰り返し行なったり，前回の演算の結果を次回の演算のデータとして用いたりすることができる．このような使用方法をすると，入出力のオーバヘッドが全体の処理時間に占める割合を等価的に下げることができる．

これらの特長により，シストリックプロセッサやアレイプロセッサと同様の考え方に基づいて外部入出力インタフェースを設計しても，システム全体のスループットを高く保つことができる．

3.5.ホスト計算機の制御構造

本システムでは3.1節に述べたように，ホスト計算機の制御を容易とするために，処理要素を8×8あるいは16×16程度のドメインにまとめ，このドメインごとにドメインコントローラを設ける．

ドメインコントローラは，プログラム走行管理ソフトウェアにより，

①システムの動作モードの切り換え
②プログラムのロード
③初期データ，定数データのロード
④プログラムの実行開始の制御
⑤プログラムの実行終了の制御
⑥スイッチパターンのモード切り換え
⑦故障に対するドメインの切り離し制御

等を実行する．

①は3.1節に述べたシステムの動作モードの切り換えを意味する．

③は外部入出力を伴わないようなデータの扱いに関するもので，例えば処理要素番号や全処理要素に共通のデータや，あるいは少数の処理要素のみに必要なデータなどに関するものである．

④はプログラムの一斉開始の制御を意味する．本システムのような大規模のプロセッサアレイの同期をとるのには難しい問題が含まれる．これには，システム全体に対するクロックの同期の問題[Fisher 1983-2]と各処理要素の動作開始の同期の問題，すなわちいわゆる一斉射撃問題[Umeo 1982]とがある．

前者の問題は主に通信に関係するので第4章で述べる．後者の問題は本システムではドメインコントローラを設け，システムの動作モードを分けているために比較的簡単化される．全処理要素の動作モード切り替えをクロックレベルで同期させるのは現実的には非常に難しい．そこで，まずシステム全体をネットワークモードにした上で，次にこれを確認した後に演算モードにする．このようにすることにより各処理要素が非同期的に処理を開始しても全体としての整合性が保証される．

⑤はプログラムの実行終了の検出および制御を意味する．各処理要素はデータ駆動制御により動作しているため処理要素内だけではプログラムの実行の終了を検出することができない．また各処理要素内において演算処理が全て終了してもシステム全体のプログラムの終了を検出することが難しい．これに対処するにはプログラムの中に終了を示すフラグの設定を埋め込む方法が最も容易な解決法である．しかし，プログラム中で全ての処理要素の終了信号を一箇所に集めることは非常に大きなオーバヘッドになる．これは各処理要素ごとに終了フラグを設け，ドメインコントローラが必要な処理要素の終了フラグを見ることにより全体のプログラムが終了したことを判定することにより解決できる．また，演算処理中にプログラムのランタイムエラーが発生したような場合にもドメインコントローラによる制御が有効である．

⑥は，全ての回線を同時に設定することができない時に，処理全体をいくつかのステップに分割して，それぞれに異なるスイッチパターンを設定することにより対処するためのものである．ただし，これを実現するには各処理要素は複数のスイッチ制御レジスタセットを持たなければならない．

⑦については3.6節で述べる．

ドメインコントローラは演算実行処理内容には直接は関与せず，全体の制御に関する事柄だけを行なうことにより，集中化によるボトルネックが生じないようにする．

3.6.故障に対する柔軟性

前節に述べたように，故障に対する制御をドメインコントローラに行なわせる．基本的にはプロセッサアレイに冗長性を持たせておき，故障時に切り離すことにより故障回復を行なう．

冗長性の単位には小さい方から処理要素，処理要素行または列，ドメイン，ドメイン行または列の4種類が考えられる．しかし，処理要素単位あるいはドメイン単位で切り離しを行なおうとすると，2次元格子状のトポロジーが崩れてしまい，割付けのやり直しが必要となるため，オーバヘッドが大きい．従って，処理要素行または列単位か，ドメイン行または列単位の冗長性が現実的である．この場合，切り離すためには処理要素あるいはドメイン周辺から入出力する結合路を列または行方向にバイパスすれば良い．しかし，このバイパスで信号伝搬遅延が大きくならないよう実装上の工夫が必要である．

通常，アレイプロセッサ等では故障した処理要素を含む行または列を切り離すようにしているが，これは処理要素間の結合が4または8近傍の範囲に限られているために可能である．しかし本システムではより広い範囲の処理要素間あるいはドメイン間で結合路をもつことも考えており，行または列単位での切り離し制御では近傍の行または列にも影響を及ぼすのでバイパスする信号線数が多くなる．

従って，冗長性の単位は結合路形態の検討を行なってから定める必要があり，これについては第4章で述べることにする．

3.7. システムのソフトウェア構成

本システムでは，プログラム開発，実行制御などの種々のソフトウェアがホスト計算機に必要となる．図3-5にこれらソフトウェアの構成を示す．

ソフトウェアには，大きく分けて，

①コンパイラ（C-1，C-2）
②資源割付けソフトウェア（A-1，A-2，A-3）
③プログラムローダ（L）
④入出力制御ソフトウェア（I）
⑤プログラム走行管理ソフトウェア（M）

がある．

コンパイラは，高級言語用と基本言語用の二段階となる．本論文では，研究の目的からそれるので高級言語については検討せず，基本言語についても，記述に関して述べるだけにとどめる．

割付けソフトウェアはシステムのさまざまな資源の割付けを行なうソフトウェアである．本システムは，静的に構造の定まっている応用を対象としているので割付けについても静的であり，

①論理的処理要素の割付け
②物理的処理要素の割付け
③回線設定

の設定の3段階となる．①で1処理要素に実行させるべきプログラムモジュールを組み合わせてグループ化し，②でこれらを実行する処理要素の物理的な位置を決定しレジスタセットやPM管理レジスタを割り当て，③では②で配置されたプログラムモジュール間の通信を行なうために物理回線を設定し入出力レジスタを割り当てる．これらのアルゴリズムについては第6章で述べる．

プログラムローダは，コンパイラにより作成された機械語プログラムを割り付けに従ってシステムにロードするためのソフトウェアで，ハードウェアの各部の

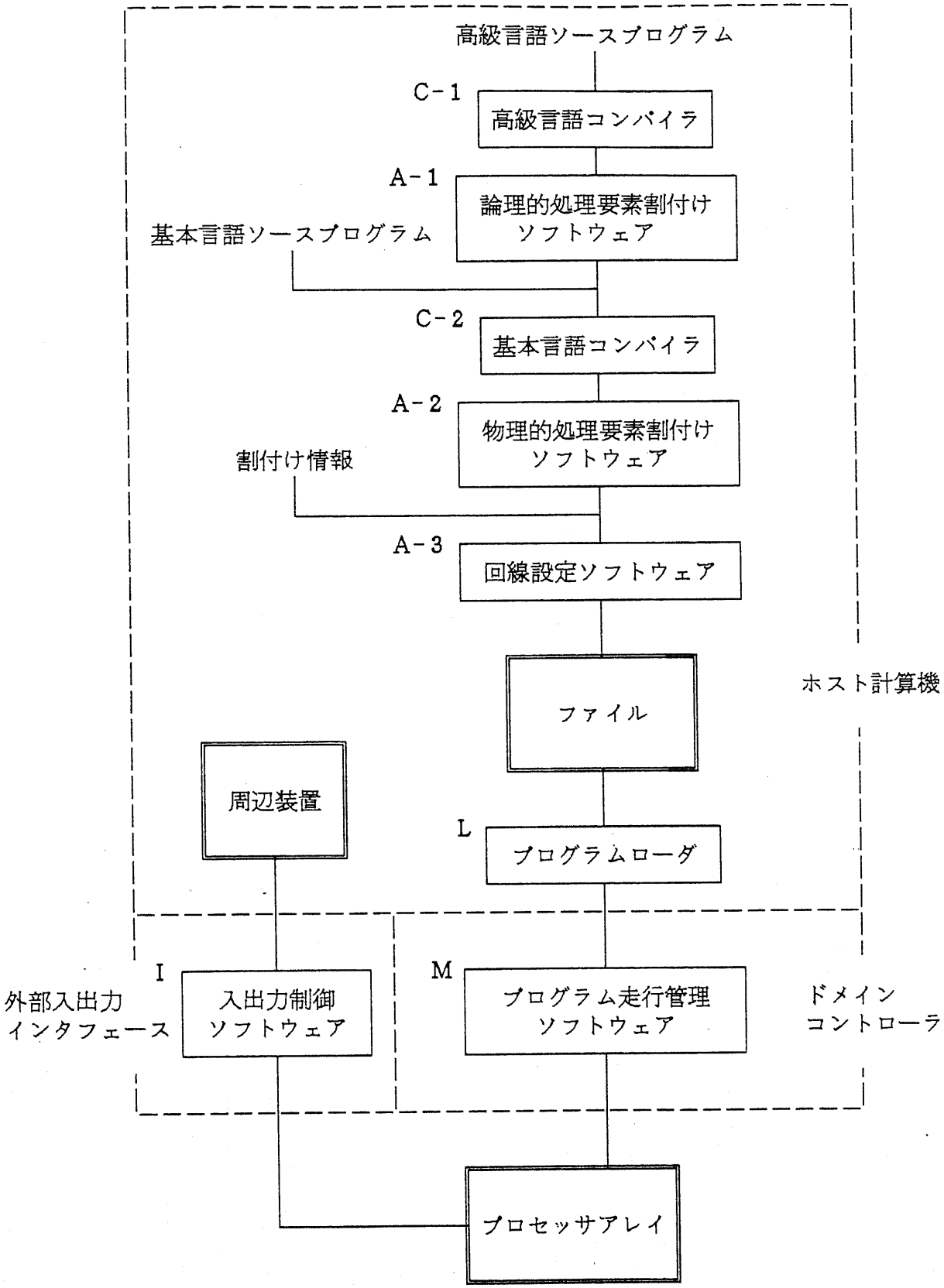


図3-5．ソフトウェア構成

機能に応じて，スイッチマトリクスのスイッチパターン，入出力ポートの論理回線の制御を行なうための入出力管理表，演算実行制御のためのデータ駆動に関するプログラムモジュール，演算処理のための逐次処理プログラムである命令ストリームを，それぞれドメインコントローラを通じてロードする．

入出力制御ソフトウェアは，外部入出力インタフェースを制御して周辺装置とプロセッサアレイの間のデータの入出力を行なう．

プログラム走行管理ソフトウェアはホスト計算機によるプロセッサアレイの制御を実現するために，ドメインコントローラ上で動作するソフトウェアである．

第4章

結合回路網の構成

本章では結合回路網の構成について述べる．まず結合回路網に要求されるスループットや諸条件を明らかにし，代表的な結合回路網の諸構成について比較検討し，2次元格子網による問題に適応可能な構成の有効性を示す．次に，この概念的構成を実現するための諸方式について検討し，空間分割回線接続方式の有効性を示し，具体的な構成方法を明らかにする．

4.1.結合回路網に要求される条件

本節では非常に多数（10^3～10^5個）のプロセッサにより構成される並列処理システムの実現のために結合回路網に要求される構成上の諸条件について検討する．

4.1.1.スループット

アプリケーションの問題を並列処理により解く場合，並列性の高い処理アルゴリズムを得るためにはできるだけ小さい単位に処理を分割したほうが有利である．一方，処理単位を小さくすると処理要素間の通信においても小さな単位のデータが頻繁に送受信されるようになる．従って，均質で小規模のプロセッサを多数用いて高並列なシステムを構成しようとする場合，多数の短いデータを効率よく処理する結合回路網の構成が求められる．

画像処理や偏微分方程式の数値解法などにおいては，最も小さい処理単位は近傍の３～９個のデータ間の四則演算であり，８ビット程度のマイクロプロセッサにおいてはマシンサイクルにして10^2～10^3と考えられる．また，これにともなって送受信されるデータの数は，２～８となり，各データの大きさは８～32ビットと考えられる．現在のＭＯＳ技術においては演算器のマシンサイクルは100ns程度と仮定できるので，最小の処理単位の実行時間は10～100μsとなり，この間に２～８個のデータの転送が必要である．これを１処理要素当りの通信に要求されるスループットに換算すると，毎秒10^5～10^6個のデータとなり，情報量としては約２Mbpsとなる．この通信容量を10^3～10^5個の処理要素に対して時間的，空間的に平均して提供しなければならない．従って結合回路網全体のスループットとしては毎秒10^7～10^{11}個のデータ，２～200Gbpsの通信容量が必要となる．

4.1.2.ハードウェア量

一般に論理装置では規模が大きくなるに従い，システムを構成するサブシステム間の結合のためのハードウェア量が増える．これは並列処理システムの処理要素間の結合にもあてはまる．ここで結合回路網を構成するハードウェアとして考慮しなければならないものに論理部品と配線がある．従来は主に論理部品に注意が向けられ，配線についてはあまり配慮がなされていなかったが，処理要素数が

多くなるに従いこれが無視できないようになってきており，現在検討している程度の規模のシステムになるとむしろ配線の方が重要となる場合が多い．

ハードウェア量を表現する場合に，処理要素数Nに対するオーダで示すことが多い．これは確かに一つの目安であるが，実現を考えた場合，定数倍の影響も同程度に重要となる．

厳密な評価は難しいが，実現性を考えた場合結合回路網のハードウェア量は処理要素自体のハードウェア量と同程度か，多くとも数倍程度が限度であろう．

4.1.3.故障に対する柔軟性

高並列処理システムでは処理要素数が10^3～10^5個程度となり，処理要素のハードウェアの故障の頻度もその数に応じて高くなる．このようなシステムでは処理要素が1箇所でも故障を起こすとシステム全体が止まってしまうようでは実用上問題が多い．そこで故障に対する何らかの自動回復機能を持たなければならない．

高並列処理システムでは同質のプロセッサを多数用いているため，故障した処理要素を他の処理要素で代替することは比較的容易にできる．しかし，故障回復に伴ってプログラムの変更あるいは再コンパイルが必要になるようでは大規模なアプリケーションに用いることはできない．そこで，ハードウェア的に全ての回復が可能であって，プログラムに対して影響のないような機能が必要である．

回復の方法としては，予備の処理要素を用意しておき故障発生時に処理要素ごと切り換える方式と，予備処理要素は用意せず故障発生時にその処理要素を切り離す方式が考えられる．切り換え方式の場合，予備処理要素は正常動作時は使用されないため，資源の有効利用の面では不利であるが，故障発生時の再割付け等の処理が不要になる．予備処理要素数は通常の処理要素数に対しそれほど多数は必要ない．

いずれの方式にせよ，結合回路網としてはハードウェア上，処理要素のシステムからの切り離しと接続を容易とするような機能が必要である．

また，結合回路網自体の故障も考慮する必要がある．例えば，図4-1のように結合回路網にクロスバ交換機等の集中制御の装置を用いる場合，その制御部分が故障を起こすとシステム全体の機能が停止してしまう．そこで，このように何

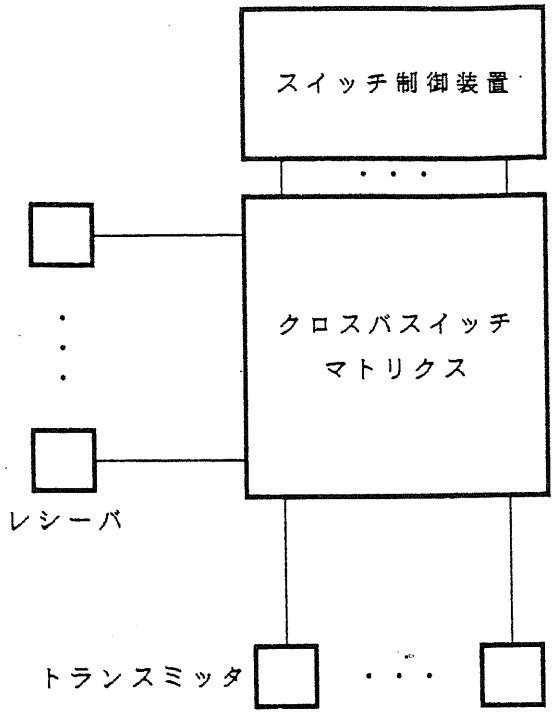


図4-1．集中制御を含む結合方式

らかの形で集中制御をする部分が存在するアーキテクチャでは，その部分の故障回復を容易とするような機能あるいは特徴を備えていなければシステム全体の信頼性を上げることはできない．

以上のような意味において結合回路網には故障に対する柔軟性が要求される．

4.1.4.結合の均質性

提案システムではプログラムの実行が，結合回路網上を物理的にデータが流れることにより進められる．従って，データの流れ方に制約があると，それはそのまま並列処理アルゴリズムの実行上の制約になってしまう．例えば図4-2に示すような階層構成の結合回路網では，構成とマッチングの良い並列処理アルゴリズムを開発できる場合は効率の良いプログラムが実現できるが，画像処理のように全てのデータが均質な位置付けを持っているような場合は階層構成とのマッチングはとれず，上位の回路網における通信がボトルネックとなってしまう．これは階層化により処理要素がグループに分割されてゆくために起こるもので，下位の結合において不連続性があることが原因である．

そこで広範囲の応用への適応性を高めるために結合回路網が備えるべき条件と

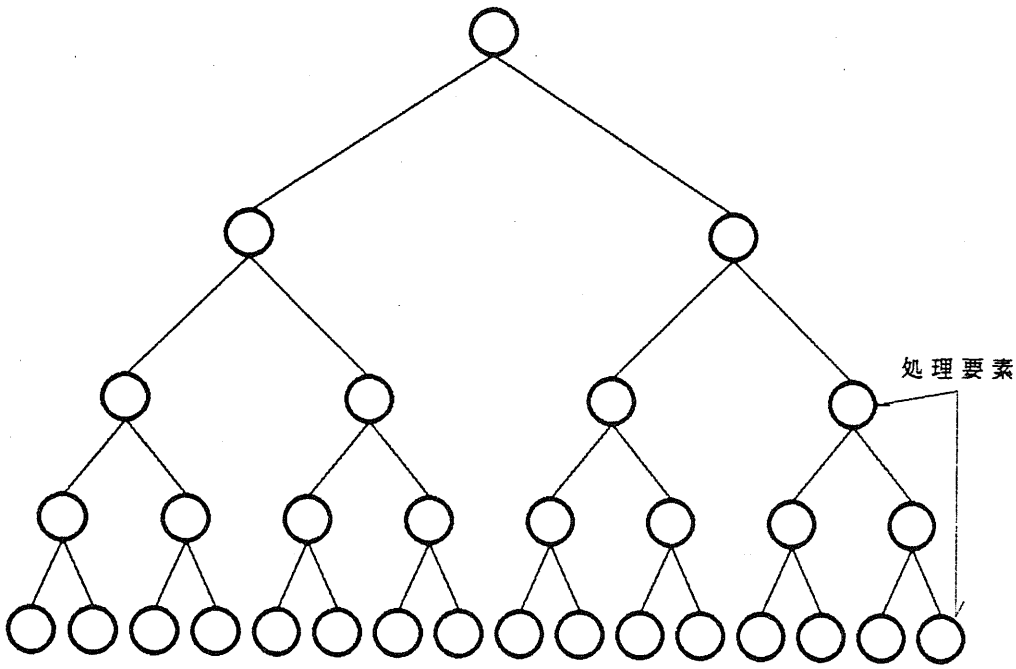


図4－2．均質でない結合方式

して，結合の均質性が必要である．

4.1.5.制御の分散化

提案システムでは演算実行制御機能を分散化することにより高性能化を図っている．しかし，ここで新たに結合回路網がボトルネックとなる可能性がある．また，処理要素は比較的簡単な専用プロセッサで実現することを仮定しているが，結合回路網に関してのみに高性能なものを用いる構成ではシステムの処理速度を高速化しようとする場合，この部分だけ特に高性能なものが要求され，システム全体の能力が制限されることになる．従って，システムは全て同レベルのハードウェア技術で実現されることが望ましい．

上記のような理由により，結合回路網は極力分散制御されることが望ましい．

4.2. 各種の結合方式に関する検討

並列処理システムのための代表的な結合方式としては，(a)共通バス，(b)多段スイッチ網，(c)スイッチマトリクス，(d)木構造回路網，(e) 2 次元格子網等がある．本節ではこれらの回路網について，前節で述べたような各種の条件に関して検討を加える．

4.2.1. 共通バス結合方式

共通バス結合方式の構成例を図 4 - 3 に示す．アービタは処理要素間の競合を制御するもので，先着順であることが望ましい．バスはアドレスバスとデータバスとに分けられる．

本方式では，アービタに制御が集中し，これが故障するとこのバスを用いた通信はできなくなる．これに対する有効な対策は特に無く，二重化等の通常の方法による．

以下ではバス幅は通信される最大データ長と等しいものとして検討する．

(1) スループット

共通バスのサイクルタイムを100nsとすると，データの送受信を最大で毎秒10^7回行なうことができる．一方，1 処理要素当りでは最大で毎秒10^4～10^5回の送受信が発生するため，1 本のバスには最大で10^2～10^3個の処理要素を接続することができる．処理要素数がこれより大きくなると，もはや単一の共通バスでは結合できなくなり，階層構成にしたりあるいはバススイッチを用いてバス相互を動的

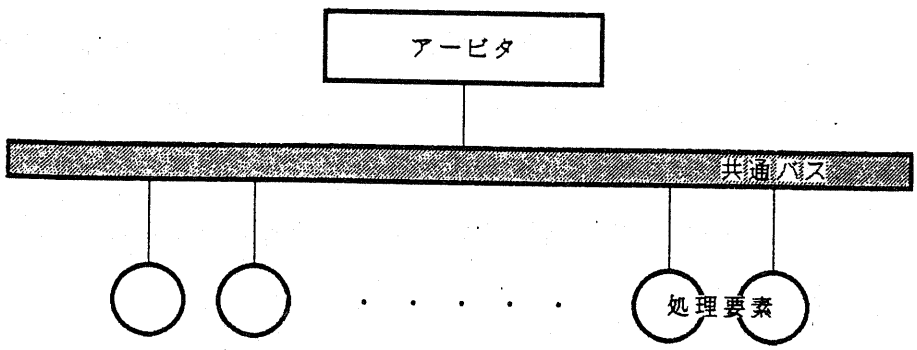


図 4 - 3 ．共通バス結合方式の構成例

にスイッチングしたりする必要が生じ非常に複雑なシステムになるため，簡単には評価できない．

⑵ハードウェア量

バスの特徴として，小規模のシステムではハードウェア量は処理要素数に比例し$O(N)$となる．しかし，単一のバスで収容しきれなくなるとバス相互間の接続のためのハードウェア量が大きくなり，少なくとも$O(N\log N)$となる．特に共通バスの特長を損なうことなく複数のバス間を接続しようとすると，$O(N^2)$のハードウェア量が必要となる．しかし，Nが10^3～10^5程度では絶対量はあまり多くはならないと考えられる．

⑶故障に対する柔軟性

予備処理要素のシステムへの追加は容易であり，処理要素の故障についてはトランスミッタが正常であれば切り離すことにより容易に予備処理要素に代替可能である．しかし，アービタの故障に対しては特に有効な対策はない．

⑷結合の均質性

単一の共通バスは全ての処理要素が任意の処理要素と均一なコストで通信可能であり，均質性は極めて高いと言える．しかし階層構成にしたりバススイッチを用いてバス間を相互接続したりするとこの均質性は失われてしまう．

⑸制御の分散化

単一の共通バスに接続できる処理要素数が比較的少ないため，集中制御となるアービタがボトルネックとなることはないと考えられる．しかし，バスを階層構成にしたりバススイッチで接続したりすると，バス相互間の制御が集中化する可能性があり，これがボトルネックになる可能性が大きい．

4.2.2.多段スイッチ結合方式

多段スイッチ結合方式の構成例を図4-4に示す．スイッチングモジュールとしては2×2や8×8のものが代表的である．この分類にはオメガネットワーク，キューブネットワーク等，さまざまな構成がありうる[Broomell 1983]．制御方式にもさまざまな構成が可能である．高並列処理システムの結合回路網ではデータ長の短い多数のデータを効率よく伝送できなければならないので，これらの組合わせのうち，ハードウェア化が容易な構成が望ましい．

この方式では通常1つの処理要素から複数の処理要素に回線を張ることは不可能であるため静的な回線接続は実現できず，動的な回線接続では回線設定のオーバヘッドが大きすぎて現実性に乏しいため，蓄積交換とならざるをえない．蓄積

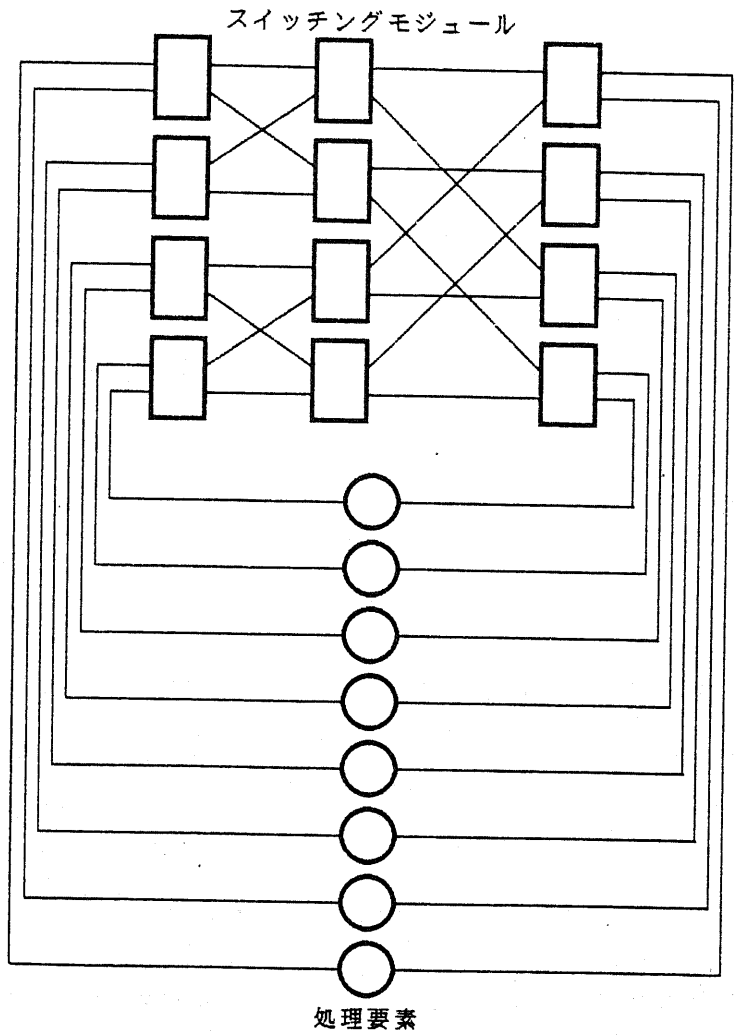


図4-4．多段ネットワーク結合方式の構成例

交換では経路選択は各スイッチングモジュールで動的にローカルに行なえるので分散制御が容易に実現できる．

そこで，以下では2×2のスイッチングモジュールを用いた分散制御の蓄積交換方式について検討する．

(1)スループット

多段スイッチ網の場合，構成にもよるが，各処理要素は独立にデータの送受信を行なうことができると考えられ，1処理要素当りで毎秒10^4～10^5回のデータの送受信は充分可能である．

(2)ハードウェア量

多段スイッチ網では，スイッチングモジュールのハードウェア量は一般に$O(N\log N)$であるが，配線量は$O(N^2)$となる．Nを10^3～10^5と仮定すると，絶対量は処理要素自体の10～100倍となると考えられる．

(3)故障に対する柔軟性

回路網のトポロジーにより，処理要素数とスイッチングモジュールの段数が定まるため予備処理要素の追加は容易でなく，処理要素の故障については安価で効率の良い回復方法はない．また，スイッチングモジュールの故障に関しても有効な対策はない．

(4)結合の均質性

多段スイッチ網においてもさまざまな構成があり必ずしも一概には言えないが，一般的には全ての処理要素が任意の処理要素と同じコストで通信可能であり，均質性は高い．

(5)制御の分散化

2×2のスイッチングモジュールを用い，図4-4に示したようなトポロジーとすると，経路選択は各スイッチングモジュールで完全にローカルに行なうことができ，分散化が可能でボトルネックになることはない．

4.2.3. スイッチングマトリクス結合方式

スイッチングマトリクス結合方式の構成例を図4-5に示す．各スイッチングモジュールはオン，オフの2状態をとるスイッチと制御回路から成る．以下では同一のトランスミッタに接続されたスイッチの集合を列，同一のレシーバに接続されたスイッチの集合を行と呼ぶことにする．

この方式には集中制御と分散制御が考えられる．しかし集中制御では毎秒10^7～10^{11}ものスイッチ切り換えを制御するのは不可能であり，分散制御とする必要がある．また，各スイッチングモジュールにバッファを持たせることもハードウェアが膨大になり不可能なので回線接続となる．そこでここでは行あるいは列ごとの分散制御による回線交換方式について検討する．

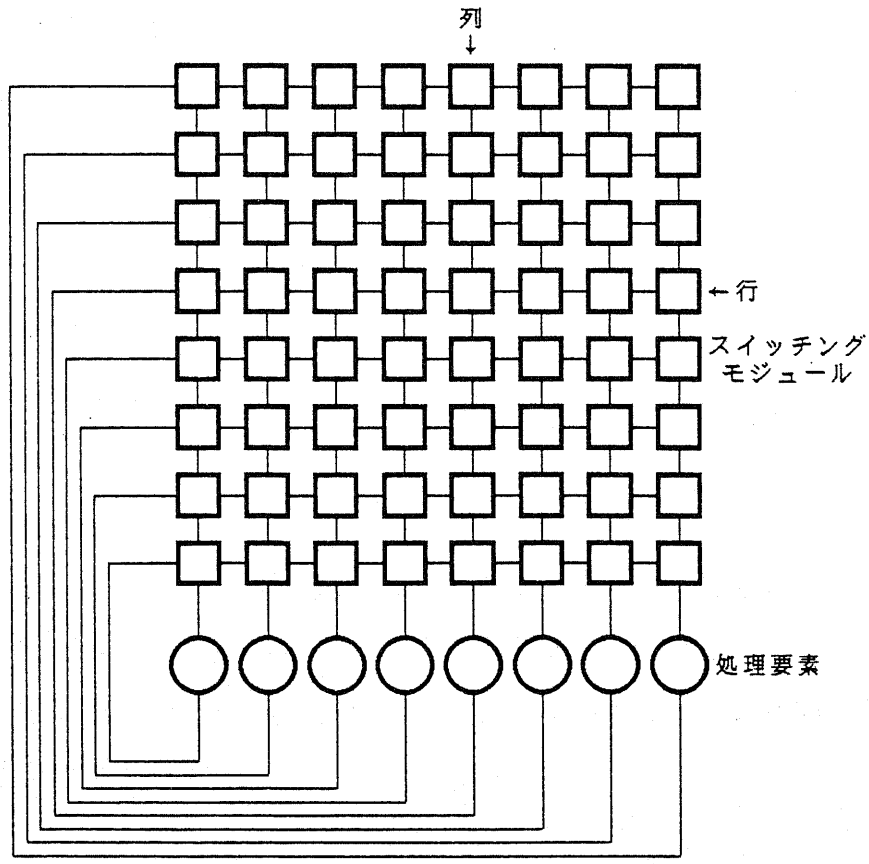


図4-5．スイッチングマトリックス結合方式の構成例

(1)スループット

スイッチングマトリクスの場合，多段スイッチ網と同様に各処理要素は独立にデータの送受信を行なうことができるので1処理要素当り毎秒10^5～10^6回のデータの送受信は充分可能である．

(2)ハードウェア量

スイッチングマトリクスでは，全処理要素を一つのマトリクスで結合するには$O(N^2)$のハードウェア量が必要である．これを軽減するために，小規模のマトリクスを多段に接続することも考えられるが制御が複雑になる．Nを10^3～10^5と想定しているためスイッチの数は10^6～10^{10}にもなり，実現性はほとんどない．

(3)故障に対する柔軟性

予備処理要素を追加する場合，マトリクスの行および列をそれぞれ拡張する必要があるが，構成上は比較的容易である．従って，処理要素の故障に対しては処理要素の切り離しにより対処できる．また，スイッチングモジュールの故障についても，そのモジュールが含まれる行あるいは列を切り離すことにより対処できる．

(4)結合の均質性

スイッチングマトリクスでは全ての処理要素が完全に均質に結合されている．

(5)制御の分散化

各スイッチングモジュールが独立に分散制御することは難しいが，行あるいは列ごとの分散制御は比較的容易にできる．各行あるいは列においては，複数の処理要素が同時にデータの送受信を行なうことはないため，この制御がボトルネックになる可能性はない．

4.2.4.木構造結合方式

木構造結合方式の構成例を図4－6に示す．図には2分木構造の最も下位に位置する“葉”のノードのみに処理要素を配置した例を示したが，全てのノードに処理要素を配置する構成も考えられる．また，4分木構造にしたり，あるいは上位の回路網がボトルネックとなることを避けるためバイパスを設けることも考えられる．いずれにしろ，この方式では回線接続は明らかに不適当であり，蓄積交換とするのが適当である．そこで，以下では図に示したような構造の蓄積交換方式について検討する．

(1)スループット

木構造結合方式の場合，アプリケーションのデータ依存性によりスループットが大幅に変化する．並列処理アルゴリズムが木構造を持ち，結合回路網形態とマッチングが良い場合は高いスループットが得られる．しかし，画像処理や偏微分方程式の数値解法のようにデータ依存性に局所性があっても階層的でないような応用では木構造回路網とのマッチングがとれず上位の回路網にトラヒックが集中しボトルネックとなってスループットが大幅に低下する．例えば平面を格子点に分割し，各格子点の演算をそれぞれの処理要素に割り付けるような場合，4近傍

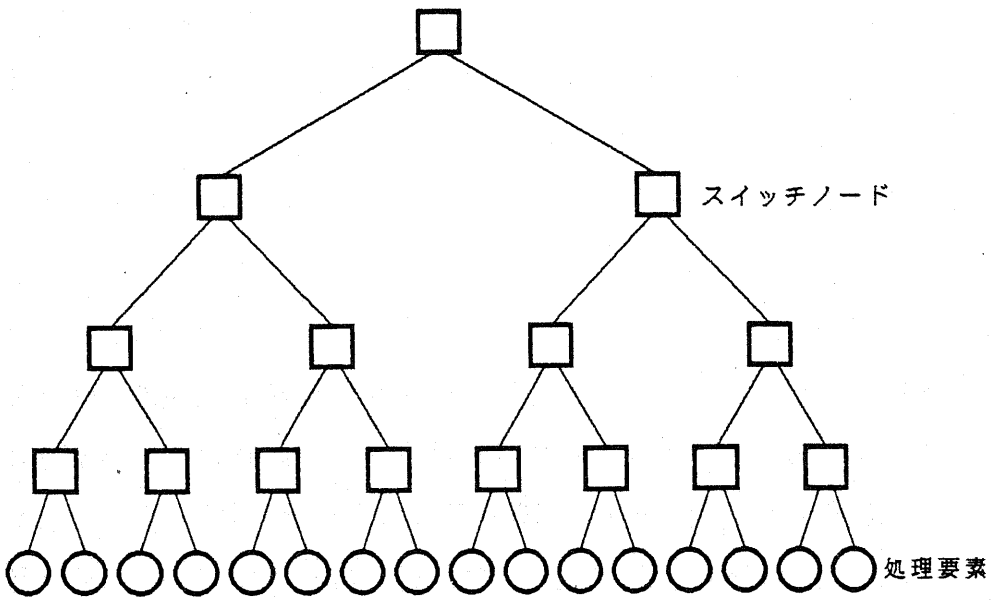


図4－6．木構造結合方式の構成例

との通信を1回行なうには最上位のスイッチノードでは$O(N^{1/2})$個のデータを転送しなければならず，Nが10^5程度では約600個となる．これを10～100μsで処理するには相当の高速性が要求される．さらに，上位になるに従い信号配線も長くなるため伝送速度を高くすることが難しくなり一層解決を困難にしている．従って，木構造をもつ並列処理アルゴリズム向きの専用機としては良いが，より広範囲な分野への応用を考えるとスループットは不充分である．

⑵ハードウェア量

スイッチノードの数は（N－1）個であり，配線量は実装方法によるが$O(N)$から$O(N\log N)$の間であって，前述のネットワークに比べ少ない．

⑶故障に対する柔軟性

処理要素の数は木構造の種類により決まり，冗長な予備処理要素の追加は困難であるため，処理要素の故障に対する有効な対策はない．また，スイッチノードの故障が起こると，その下位のサブシステムは全て使用不能となり，これに対する有効な対策もない．

⑷結合の均質性

この結合方式では構成が階層的になっているため，物理的に隣り合っているノード間でもそれらが属する部分木が異なる場合は上位回路網を経由しなければならず，システム内の位置により通信のコストが大幅に異なる．これは部分木間において結合が不連続であるためであり，この構成は結合の均質性に欠けている．

⑸制御の分散化

図4－6の例では，処理要素に左から0～（N－1）までの番号をつけるとこれは$\log N$ビットで表現でき，上からk段のスイッチノードではこのうち上位のkビットでその下位の部分木の番号を表現できる．データの転送においては，自分の部分木に含まれない処理要素番号を宛先にもつデータは上位に転送し，含まれる場合はその下の1ビットが0なら左，1なら右に転送すればよい．これにより，制御は完全に分散化できる．

4.2.5. 2次元格子網結合方式

2次元格子網結合方式の構成例を図4-7に示す．前述の方式はいずれも処理要素を1次元的に配置し，それらを結合回路網で結合するものであったが，この方式では処理要素は2次元的に配置され，空間的に処理要素は結合回路網に含まれている点が特徴である．

処理要素相互間の結合は4近傍のほかに8近傍のものも考えられるが，ここでは最も単純な4近傍の結合に限って検討する．

提案システムでは，処理要素は独立した演算実行制御機能を持ち，互いに非同期に動作するので，アレイプロセッサのようにプロセッサが通信制御も行なうというようなことはできない．従って，各処理要素は図4-8に示すようにスイッチングモジュールと演算部に分けられることになる．

(1)スループット

2次元格子網においてはアプリケーションの持つデータ依存性に局所性がないような場合，データの転送が非常に多くなり通信トラヒックが増大する．そこで以下ではデータの送受信を行なう処理要素間の距離の平均値をdとして検討する．

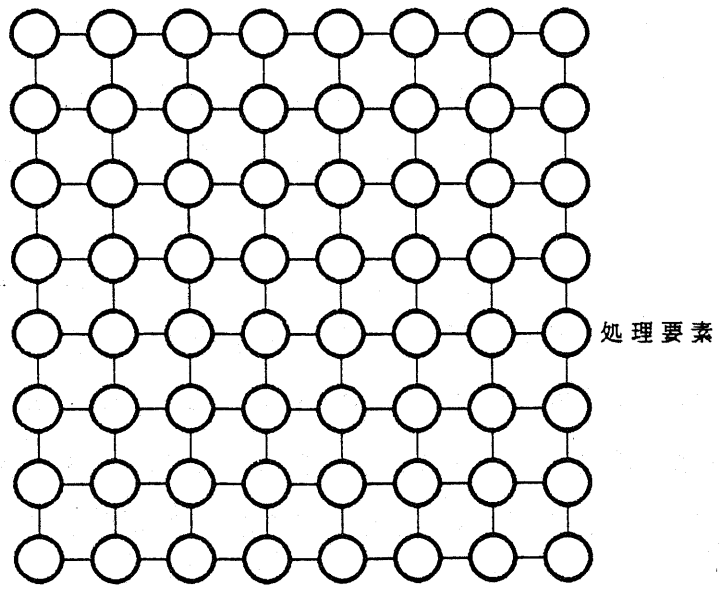


図4-7．2次元格子網の構成例

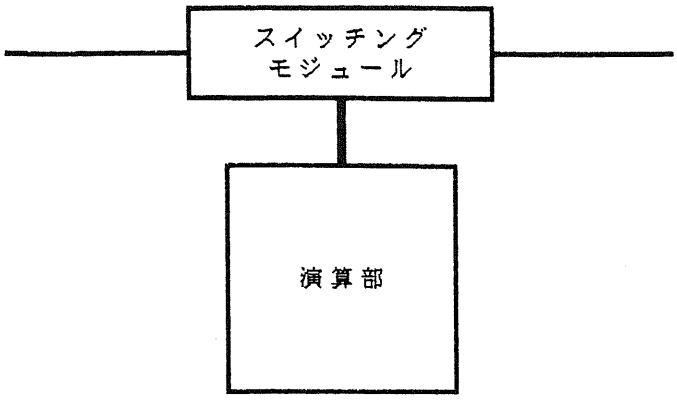


図4-8．2次元格子網の処理要素の構成

各処理要素が送受信するデータ量は約2Mbpsと想定しているので，各処理要素のスイッチングモジュールにかかるトラヒックは平均で$2d\times10^6$bpsと考えることができ，スイッチングモジュールのデータ処理能力を100Mbpsと仮定すると，dが50程度のアプリケーションにまで対応できる．しかし，一般にはトラヒックは中央部に集中するため，現実にはこの半分程度が限界であろう．

(2)ハードウェア量

2次元格子網では，スイッチングモジュールは各処理要素内に含まれており，その他には処理要素相互間の結合配線だけであるので，O(N)のハードウェア量ですむ．また，絶対量も各処理要素内のスイッチングモジュールの大きさで決まり，演算部に比べ同程度あるいはそれ以下になる．

(3)故障に対する柔軟性

予備処理要素の追加は格子網の行または列の拡張という形で行なわれ，実現は容易である．従って，演算部の故障に対しては処理要素を切り離すことで対処できる．スイッチングモジュールの故障に対しては，処理要素相互間の結合配線をバイパスすることにより対処できるが，このためにはハードウェア機能の追加が必要である．

(4)結合の均質性

2次元格子網では全ての処理要素の結合は均質であるが，局所性があり処理要素間の距離により通信コストが変わる．しかし，階層構成のような不連続性はない．

(5)制御の分散化

制御は完全に各スイッチングモジュール内で実現でき，本質的に分散化に適している．

4.2.6.結合方式の選択

上述の各種の結合方式に関する検討の結果をまとめると表4-1のようになる．

(a).共通バスでは要求されるスループットを単一のバスで実現することは不可能であり，高々10^2～10^3個の処理要素が限度である．処理要素数を増やす場合はバスを並列に張る必要があり，これら相互間の結合は特殊な形をとることになり，構成が複雑になるため，本システムの結合回路網には不適当である．

(b).多段スイッチ網はスループットは充分高いが，ハードウェアの絶対量が多く10^5個の処理要素を接続するためには10^6個近いスイッチングモジュールが必要であり，さらに配線は量が多いばかりでなく複雑である．故障に対しても柔軟性に欠ける．これらの理由により，この方式は中程度の並列処理システムには適用できる可能性があるが，並列度が高くなると実現は困難になる．

(c).スイッチングマトリクスはスループットは高いが，ハードウェア量が$O(N^2)$と極めて多く，高並列処理システムには非現実的である．

表4-1．各種の結合方式の比較

	スループット	ハードウェア量	故障に対する柔軟性	結合の均質性	制御の分散化
共通バス	×	△	△	○	△
多段スイッチ網	◎	○	×	◎	◎
スイッチングマトリクス	◎	×	○	◎	○
木構造	△	◎	×	×	◎
2次元格子網	○	◎	◎	△	◎

(d). 木構造回路網では，その構造に対するマッチングの悪い並列処理アルゴリズムにおいては上位回路網がボトルネックとなりスループットが上がらないため適応範囲が狭くなる．また故障に対する柔軟性に乏しく，結合の均質性も持たないため，本システムの目的には不適当である．

(e). 2次元格子網ではこれらに対し，スループットについてはアプリケーションにもよるが一応の要求を満たし，ハードウェア量もO(N)と小さく，故障に対する柔軟性は最も高く，制御の分散化も容易であるので，結合に局所性はあるが，本システムには最も適した結合方式である．

以上のような理由から，本提案システムでは2次元格子結合方式を採用することにした．

4.3.処理要素間の通信方式

4.1および4.2節において，高並列処理システムの各種の結合方式について検討し，2次元格子結合方式が最も適当であるという結論を得た．本節ではこの結合方式の基本的概念に基づき，処理要素間の通信方式の検討を行なう．

処理要素間の通信方式には蓄積交換と，空間分割および時間分割の静的な回線接続が考えられる．本システムでは通信制御は分散化させるため，動的な回線交換はオーバヘッドが大きく現実的でない．

以下で，これらの方式の比較を行ない，本システムで空間分割の回線接続方式を選択した理由を明らかにする．

4.3.1.データ転送遅延

蓄積交換と空間および時間分割の回線接続の各方式のデータ伝送遅延を簡単に評価する．遅延に影響を与えるパラメータとして，データ長をLバイト，平均経由処理要素数をNとおく．図4-9に示すように，結合路は4近傍の処理要素間のみで，これらの各処理要素間には並列，直列にかかわらずデータ転送用に16本の配線が可能であるとする．なお，この評価には待ちによる遅延は考慮していない．

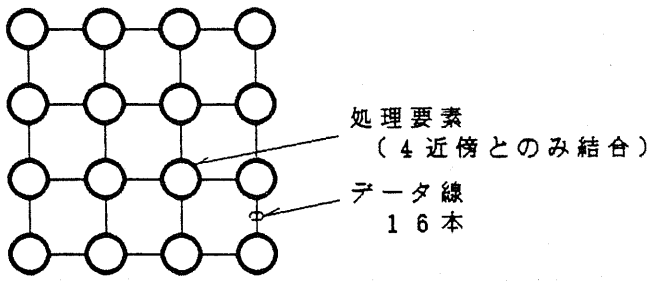


図4-9．データ転送遅延評価のための結合路の配線に関する条件

蓄積交換

蓄積交換では，結合路を双方向に各８本ずつ用いて８ビットの並列伝送を行なうことができる．バッファ制御や受信応答を考えないと，

・メッセージ長は，データＬバイトの他にヘッダに５バイト程度必要で，合計で（Ｌ＋５）バイト程度となる．ヘッダにはソースおよびデスティネーションの処理要素番号２バイトずつと，データ長１バイトが含まれる．
・１処理要素経由するごとの転送のオーバヘッドは，デスティネーションアドレス入力に２クロック，スイッチングに２クロック，送出制御に１クロック程度必要で，合計５クロック程度となる．

従って，１メッセージ当たりの合計の遅延Ｄpは，

$$Dp = 5N + L + 5 \quad \text{クロック} \qquad (4-1)$$

となる．

空間分割回線接続

空間分割の回線接続では，複数の回線を同時に張ることができるよう，伝送はビット直列と仮定する．ヘッダや動的なスイッチングは必要ないので，

・メッセージ長は，データ８Ｌビットの他に，制御ビットに２ビット使うとして，全体で合計して（８Ｌ＋２）ビットとなる．
・１処理要素経由するごとのオーバヘッドは，後述のように信号同期のために１クロックだけ必要である．

従って１メッセージ当たりの合計の転送遅延Ｄsは，

$$Ds = N + 8L + 2 \quad \text{クロック} \qquad (4-2)$$

となる．この場合，隣接処理要素間の結合路には双方向合わせて16本の回線を通すことができる．

時間分割回線接続

時間分割の回線接続では，蓄積交換と同様に８ビットの並列伝送が可能である．この場合，各方向に制御線１本程度が必要である．ｎ回線を多重化すると仮定すると，

・メッセージ長はデータＬバイトのみであり，ヘッダ等は不要である．
・１処理要素経由するごとのオーバヘッドは経路の設定が最適であれば１クロックですむが，タイムスロットの割当てがうまくいかないとこれが大きくなる．

このオーバヘッドの平均値をｃクロックとおくと，１データ当たりの合計の平均転送遅延Ｄｔは，

$$Dt = cN + nL \quad \text{クロック} \qquad (4-3)$$

となる．隣接処理要素間の回線数を空間分割と等しくするためにはｎを８とすることになり，ｃが１の時は空間分割の場合とほとんど差はない．ただし，最適なスイッチの設定が必ずしも可能であるとは限らないので，実際は回線数が増えると，ｃの値が大きくなる．しかし，これは蓄積交換においては待ちとして現れるものに相当するため，ここではｃ＝１として比較する必要がある．

蓄積交換ではバッファリングが必要であり，転送のクロックレートは通常のＭＯＳ技術で10MHz程度が限度であろう．回線接続では制御が単純であり，信号が経由するゲートの段数も少なくできるため，20MHz程度は得られる．

ＤｓとＤｔはほぼ等しいので，ＤｐとＤｓの比較を図４－１０に示す．この図から明らかなように，蓄積交換は長いデータを近くの処理要素に送るのに適し，回線接続は短いデータを遠くの処理要素に送るのに適している．

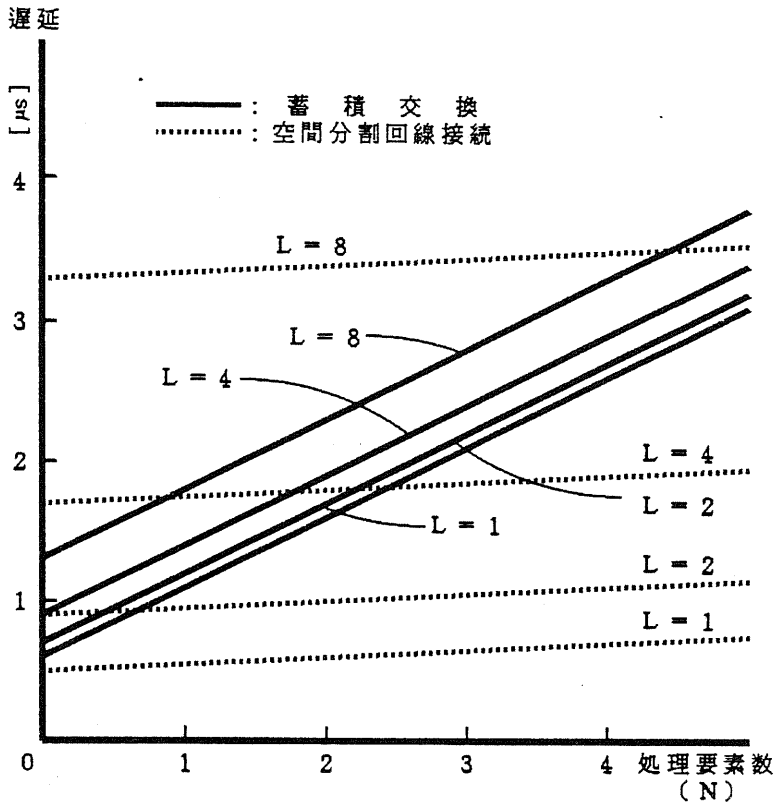


図4-10．蓄積交換と回線接続の遅延特性

4.3.2. ハードウェア機能

以下に，各方式においてスイッチングモジュールに必要とされる機能を示す．

蓄積交換

・バッファリング：

受信したメッセージを一時的に蓄積する機能であり，各入力結合路ごとに，最低でも1メッセージ分のバッファ容量が必要である．トラヒックが大きい場合の効率を高めるには，バッファの容量を増やす必要があり，これにともないバッファ制御も複雑になる．

・送信制御：

出力方向が同一のメッセージが複数の入力から到着して競合した場合，定められた規則に従い，競合制御を行ない，順にメッセージを送出する機能である．メッセージ長の管理も必要であり，かなり複雑な処理を要するためハードウェア量が大きくなる．

・ハンドシェーク：

スイッチングモジュールはバッファに空きがある時に受信可能，空きがない時に受信不能となり，これを隣接処理要素に知らせる機能がハンドシェークである．これをメッセージで行なう方法と信号線で行なう方法があるが，後者のほうが簡単で高速性が得られる．

・経路選択とスイッチング：

メッセージに付加されたヘッダ中のデスティネーションアドレスを見て送信方向を決定し，スイッチングする機能である．経路を固定とすると，経路選択にはテーブルルックアップか演算による方法が考えられるが，処理要素数が多いためテーブルルックアップによる方法は実現が難しく，演算による方法では演算機能が必要である．

このうち，バッファリングと経路選択が最も大きなハードウェア量を要する．

空間分割回線接続

・空間スイッチ：

各方向の結合路間を接続するもので，スイッチマトリクスにより構成できる．

・スイッチ制御：

空間スイッチのオン・オフを制御する機能で，動的な切り換えをしないため単なるフリップフロップで実現できる．

これらは構成が非常に単純であり，ハードウェア量も蓄積交換に比べると小さくなる．

時間分割回線接続

・時間スイッチ：

各方向からの入力信号をタイムスロットから取りだし出力方向のタイムスロットにのせる機能である．

・スイッチ制御：

各方向の入力信号のタイムスロットと，出力方向および出力のタイムスロットの対応を記憶し，スイッチを制御する機能である．

・フレーム同期：

システムの立ち上げ時に入力と出力の間のフレーム同期をとる機能であり，一度同期が確立されればあとは不要である．しかし，この機能は少々特殊であり，ハードウェアが複雑になる可能性が高い．

この方式は蓄積交換に比べると簡単であるが，空間分割回線接続に比べるとかなり複雑である．

4.3.3.その他の特質

上記に2次元格子網の通信方式についてデータ転送の遅延特性とスイッチングモジュールに要求される機能について述べたが，この他に両方式の特質として表4-2のようなものがあげられる．

蓄積交換では，ソースおよびデスティネーションの処理要素においてアドレス情報やメッセージ長などのヘッダの付加・削除の処理が必要であり，またソースアドレスからデータを識別することも必要であるため，これらがオーバヘッドとなる．回線接続では同期のための制御ビットが必要になるだけで，その他には本質的にオーバヘッドはない．

スイッチングは，蓄積交換では各処理要素ごとに動的に行なう必要があるのに対して，回線接続では実行処理の開始以前，通常はコンパイル時に接続表が作られ，プログラムロード時に設定されることになり静的である．動的なスイッチングの方が柔軟性は高いがそれに伴いハードウェアが複雑になる．

回線接続での最大の問題はブロッキングである．通信を行なう可能性がある処理要素間では全て回線を接続しておく必要があるが，隣接する2つの処理要素間の結合路の回線本数は，空間分割ではパッケージのピン数により，また時間分割ではタイムスロット数とスイッチ制御レジスタ数により，高々十数本に制限されるため，特定の処理要素に多数の回線を接続したり，データ依存関係の局所性をうまく抽出できなかったりすると，回線の設定ができない可能性がある．これは

表4-2．蓄積交換と回線接続の特質の比較

	蓄積交換		回線接続	
オーバヘッド	大	ヘッダの付加・削除，ソースアドレスによるメッセージの識別	小	同期ビット程度（時間分割ではシステム立ち上げ時にフレーム同期が必要）
スイッチング	動的	各処理要素ごとに行なう	静的	プログラム実行開始前にホスト計算機からパターンをロードする
ブロッキング	なし	任意処理要素間で通信可能	あり	隣接処理要素間のリンク数は物理的な結合路数で制限される

通常の回線交換におけるブロッキングとは異なるが，回線が必要数だけ無いことから生じる問題であり，ここではこれをブロッキングとよぶことにする．蓄積交換でこのブロッキングに対応して発生する減少は待ちによる遅延の増大である．このため，静的な回線接続方式のの適応範囲の限界はこのブロッキングにより決まる．

ブロッキングを避けるためには，処理のモジュールへの分割，処理モジュールの処理要素への割付け，処理要素間のデータ依存性の物理回線への割付けを，それぞれ最適化する必要がある．

4.3.4.通信方式の選択

本システムは専用処理装置的な用途を目的としているので，比較的短いデータが多いと考えられ，転送遅延特性では回線接続が望ましい．

ＶＬＳＩ技術の進歩により集積度が上がったとはいえ，ハードウェア量や制御構造の複雑さは小さいほうが良く，この点からも回線接続が望ましい．

回線接続においては，空間分割では任意の回線を任意の方向に使うことができ，さらに半二重双方向伝送も可能であるのに対して，時間分割では伝送方向は一方向に固定となる．時間分割では回線の多重度を増やすと，ブロッキングは減るが，遅延が大きくなり，さらに，フレーム同期のためにハードウェアが複雑になる可能性がある．

これらのことから，ブロッキングの問題はあるが，本システムの通信方式には空間分割の回線接続通信方式が適当である．

4.4.同期方式

処理要素間の通信のための信号同期について以下に述べる．

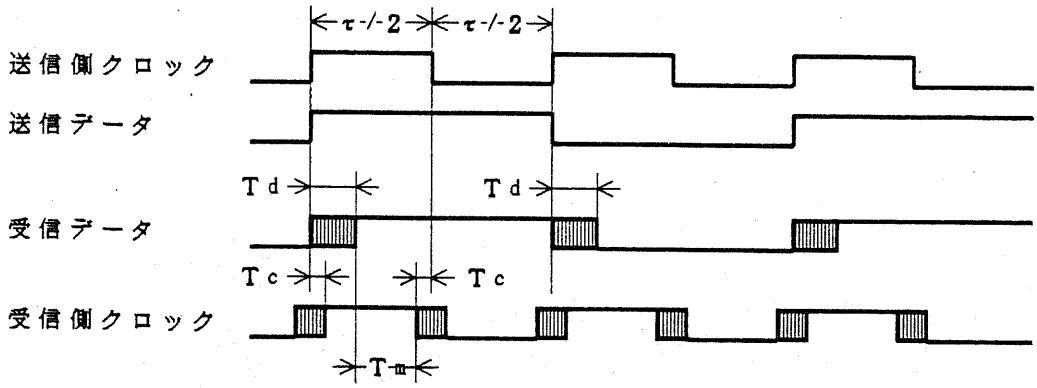
通信は空間分割回線接続で行なうため，回線の両端で同期が取れれば良いのであるが，回線長が一定ではなく，従って信号遅延も一定ではないため，両端の処理要素間で直接同期を取るのは難しい．また，伝搬遅延を小さくするだけでなく，転送のビットレートも高くする必要があるため，本システムでは，回線が処理要素を経由するごとに同期を取ることにする．

この方式では各処理要素ごとに1クロックの遅延を生ずることになる．しかしシステムクロックは隣接処理要素間の位相のずれがある範囲内にあればよいので実現は比較的容易である．

システムクロックサイクルをτ，隣接処理要素間のクロックのずれの最大値をTc，信号の伝搬遅延をTdとおくと，通信の信号タイミングは図4-11のようになり，Tm＞0であることより信号のクロック周波数fは

$$f = \frac{1}{\tau} = \frac{1}{2(Tc+Td)} \qquad (4-4)$$

で抑えられる．



τ :システムクロックサイクル
Tc:クロックのずれ(最大値)
Td:信号伝搬遅延(最大値)
Tm = τ/2 - Td - Tc

図4-11．結合路の信号同期タイミング

Tdは，2つの処理要素が同一の基板上にある場合は数ns以下で小さいが，別々の基板上にある場合は配線長が長くなり，浮遊容量も大きくなるため遅延が大きくなるので実装上の工夫が必要になる．しかし，10ns程度に抑えることは可能であろう．

また，システム全体のクロックをそろえることは難しいが，Tc は結合路の両端のクロックのずれであり，局所的であるので比較的容易に小さくすることができる．クロックの配線長をそろえたり，その他の調整により，5ns程度とすることは可能であろう．

これらのことから，通信に関してはシステムクロックを30MHz程度とすることが可能であると考えられる．以下ではこれを20MHzとして検討を行なう．

この同期方式を実現するための入出力回路の構成を図4-12に，動作のタイミングを図4-13に示す．同期を取るために入力信号はシステムクロックの立ち下がりによりラッチされる．また，双方向のデータを各方向に切り分けるために付属的な回路が必要になる．

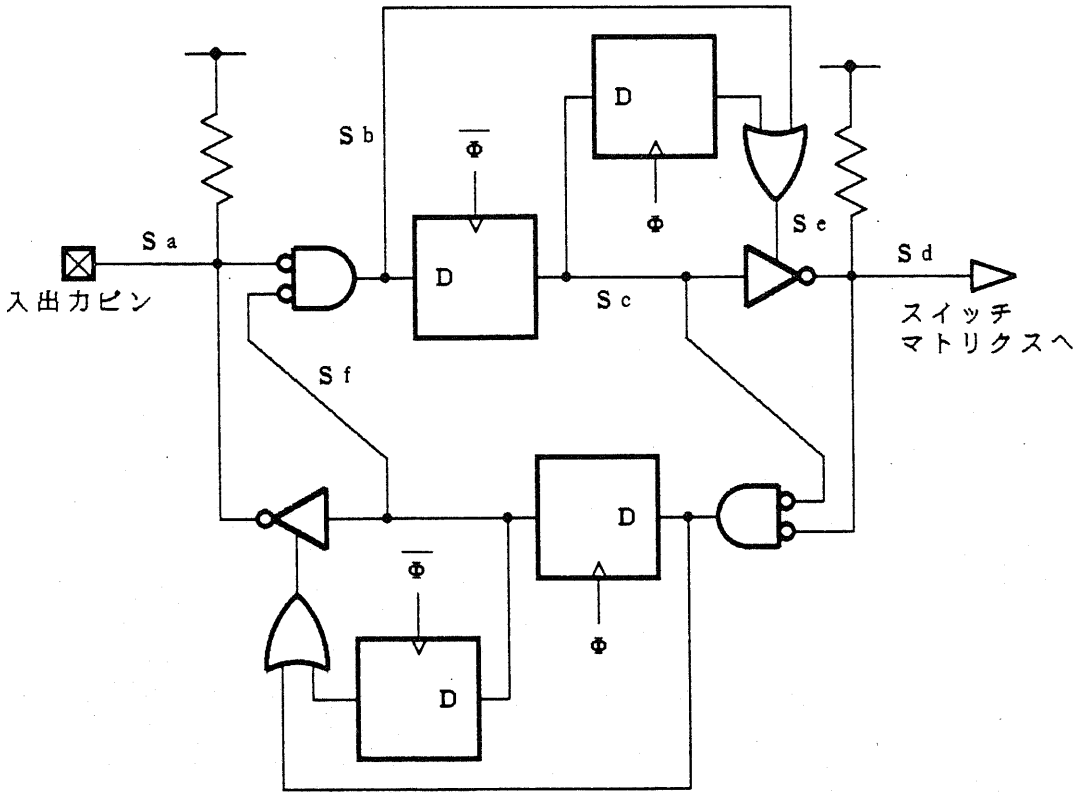


図4-12．入出力回路

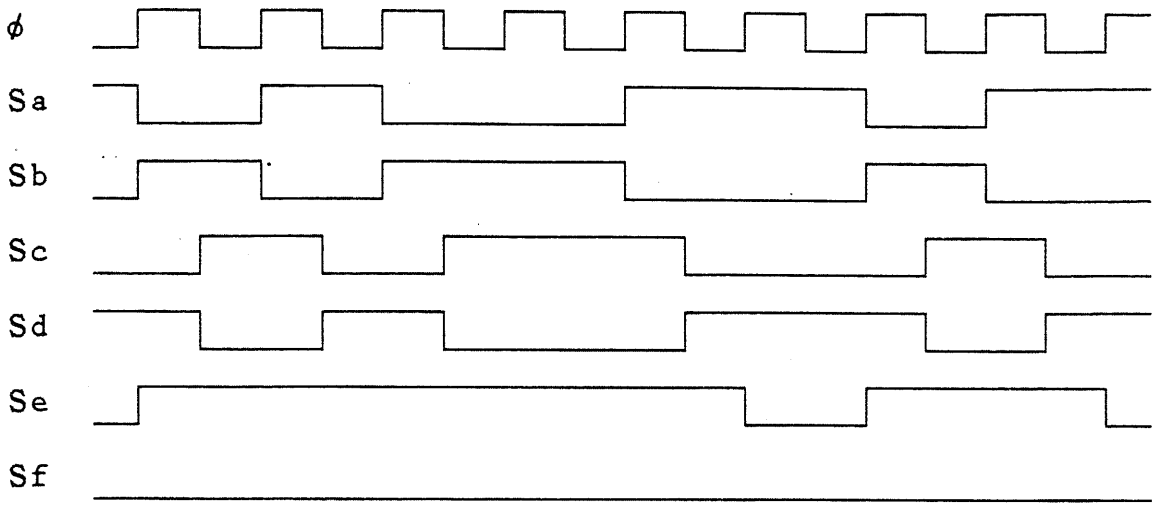


図4-13．入出力回路タイミング（入力時）

4.5.結合路のトポロジー

結合回路網は各処理要素内のスイッチマトリクスと入出力ポート，それに隣接処理要素間を結合する結合路により構成されるが，問題への適応可能性を主に決めるのが結合路の本数やトポロジーである．合計の本数はパッケージのピン数により抑えられるので，この結合路のトポロジー，すなわち，ある処理要素から近傍のどの処理要素との間に何本結合路を配線するかが重要となる．

通常のアレイプロセッサでは，4または8近傍の処理要素との間に1本ずつ結合路が張られている．2.4節で紹介したＣＨｉＰコンピュータでは処理要素とスイッチが別々であるので直接比較はできないが，前後左右の処理要素との間には2～10本，斜めの処理要素との間には1本程度の結合路を持つ構成と見なすことができる．

しかし，本システムではより高い柔軟性と広い分野への適応可能性を得るため，より距離の大きい処理要素との間にも結合路を設けることが考えられる．

処理要素の配置が2次元格子状なので，結合路のトポロジーも，90度の回転に対して不変であると仮定する．そこで，以下では，ある特定の処理要素に注目し，その処理要素を原点とし格子方向にx軸とy軸をとり，格子上の隣り合った処理要素間の距離を1とおいて検討することにする．

処理要素をＶＬＳＩ1チップ上に実装する場合を考えると，入出力ピン数は100本位が妥当な値であり，結合路として使えるのは64本程度となる．

結合路の接続能力の指標としては，

$$P = \sum_{i,j} (i + j) \cdot \ell(i,j) \qquad (4-5)$$

（ただし，$\ell(i,j)$は(i,j)の処理要素に対する結合路の本数）

で表わされるＰを用いることが考えられる．

実際の問題を考えた場合，256×256個程度の処理要素を効率良く使うためにはＰ≧128位が必要である．

この条件を満たす一例として，図4-14のような結合路のトポロジーが考えられる．このトポロジーでは，

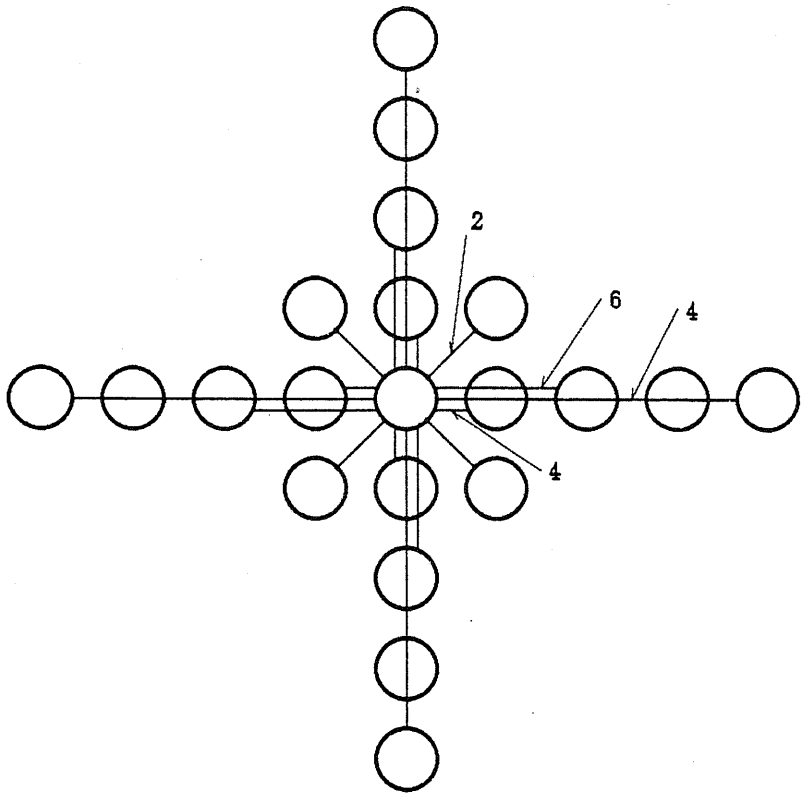


図4-14．処理要素間の結合路のトポロジー

$$P = 4\times(1\times4+2\times6+4\times4+2\times2) = 144 \qquad (4-6)$$

である．

Pが大きいほど広範囲の回線設定が可能となるが，結合路の総数が限られているために，あまりPを大きくとると近傍との接続ができなくなる．実際のさまざまな並列処理アルゴリズムに対して，それぞれ最適なトポロジーが存在すると考えられ，すべてに対して適するトポロジーを実現することは困難である．図4-14のトポロジーでは，24近傍の処理要素との間の直接的な回線設定も可能であり，Pの値も144と比較的大きいので，1処理要素を1チップに実装する場合，この程度の結合路トポロジーが現実的であると考えられる．

しかし，将来的に1ドメインを1チップまたは1ウェーハに実装できるようになった場合，結合路はチップ内またはウェーハ内の配線となり，LSIの多層配線技術が進歩するにしても処理要素を飛び越すような結合路の配線はコストが高くつく．そこで，これを考慮し，かつ広範囲の回線を接続できるような結合路ト

ポロジーが必要になる．このために，図4-15のような構成が考えられる．各ドメイン内は最も単純な4近傍接続とし，ドメイン間の結合路にドメイン単位の飛び越しを可能とするトポロジーを導入する．ドメイン内ではピン数の制約が無くなるため，より多数の結合路を設けることが可能となり，4近傍接続でも図4-14に示したのと同程度以上の接続能力を実現できると考えられる．ドメイン間の結合路はいったんスイッチアレイを通して他のドメインに接続されているため，長距離の回線はドメイン内を通らずに設定でき，混雑の解消と遅延の低減に有効である．ただし，このスイッチアレイにおいても，4.4.5節に述べたようなクロック同期が必要である．一方，故障時の回復動作においては，ドメイン行または列を切り離す制御機能をスイッチアレイに内蔵することにより，実現が容易になる．

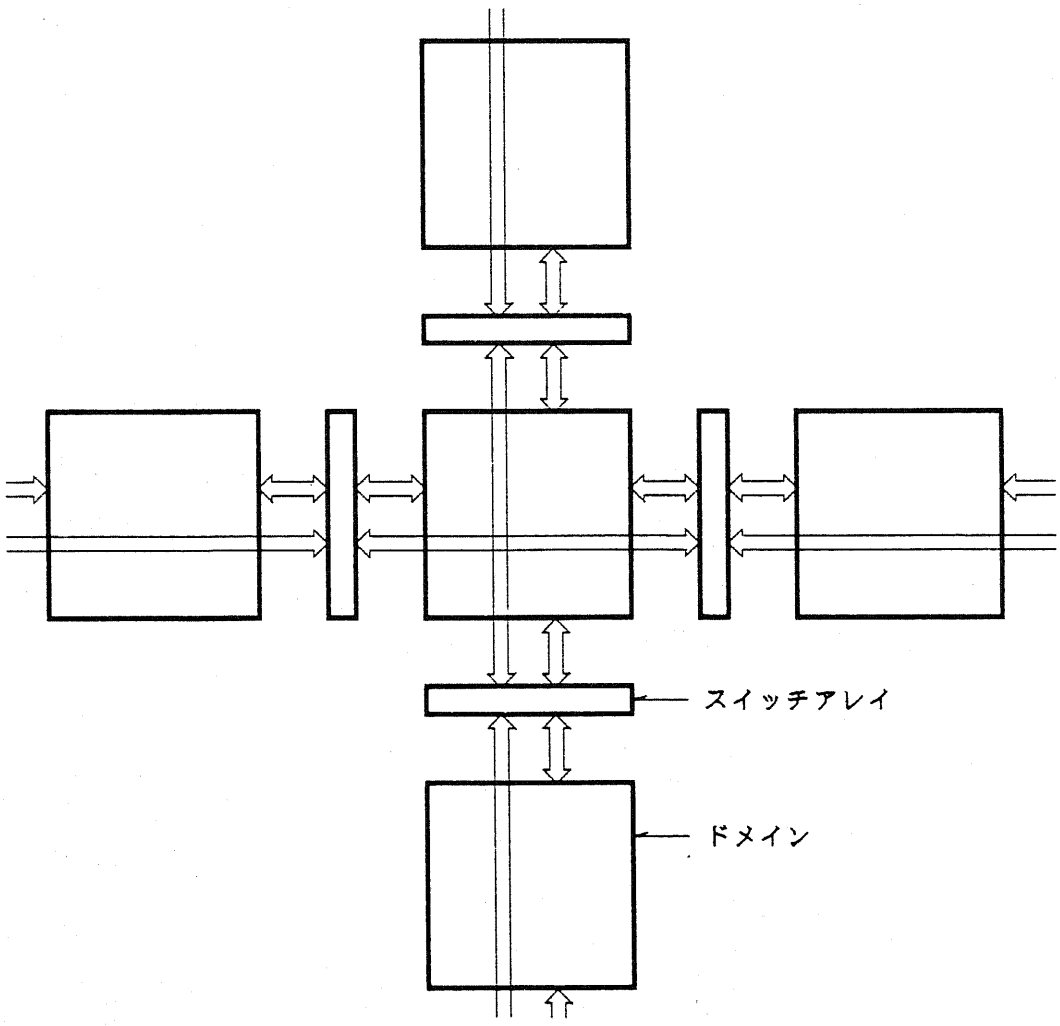


図4-15．ドメイン間の結合路のトポロジー

4.6.回線の接続制御

物理回線の経路設定は，プログラム開発時にホスト計算機において行ない，各処理要素のスイッチマトリクスのスイッチパターンとして記憶しておく．接続制御はスイッチのオン・オフだけでよく，タイミングや動的な切り換えの制御は不要であり，極めて単純化できる．制御データは，スイッチのオン・オフを制御するスイッチ制御レジスタに，3.5節に述べたように，ホストからドメインコントローラを通じてロードする．

スイッチのモード切り換えの実現方法には2通り考えられる．1つは各処理要素にスイッチ制御レジスタを数セット内蔵し，ドメインコントローラが指定するスイッチ制御レジスタセットを選択してスイッチマトリクスの各スイッチのオン・オフを行なう方法であり，もう1つはドメインコントローラからスイッチ制御レジスタに対して直接制御データをロードしなおす方法である．第1の方法は処理要素のハードウェア量が増えるのが欠点であり，第2の方法は切り換えのオーバヘッドが大きくなるのが欠点である．しかし，第2の方法でも，ドメインコントローラに予め各モードのスイッチパターンをロードしておけば，プログラム実行の初期化に比べればはるかにオーバヘッドは小さくなり，特殊な制御も必要ないので，この方法の方が適当である．

スイッチマトリクスの1スイッチ分の構成を図4-16に示す．信号配線は双方向データ伝送をするためスイッチも双方向である必要があり，MOSでは比較的容易に実現できるがバイポーラでは実現が難しい．スイッチ制御レジスタは単なるフリップフロップでよく，縦列に接続することでシフトレジスタ構成とすることができ，スイッチパターンのロード用の配線量を小さく抑えることができる．

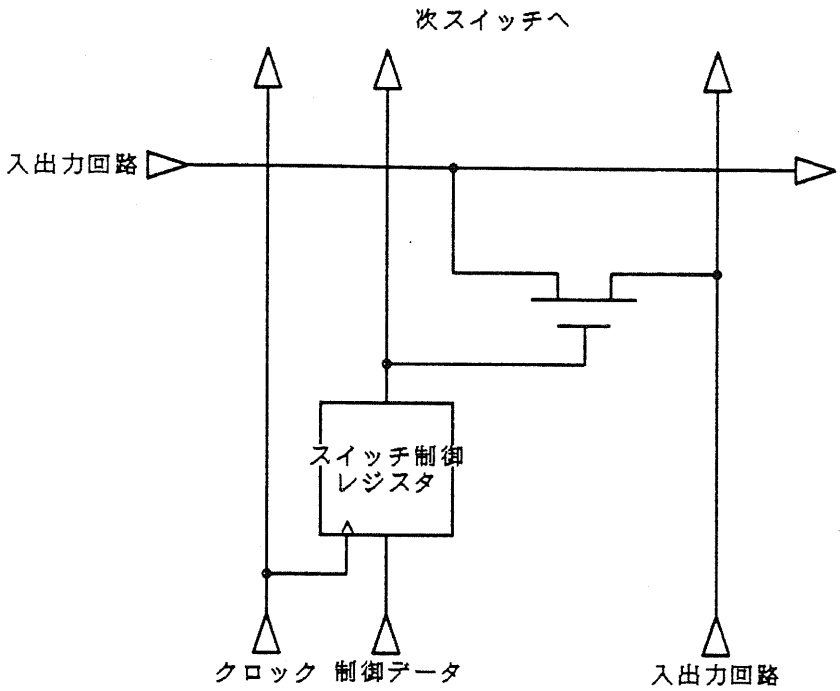


図4-16．スイッチマトリクスのスイッチの構成

4.7. 結合回路網の送受信制御

回線の両端のＰＥにおいて送受信制御を行なうのは入出力ポートである．入出力ポートの構成を図４-１７に示す．

送信側の入出力ポートでは，入出力レジスタからデータを取り出して，並直列変換をしてから回線に送出する．受信側では，到着したデータを直並列変換してから入出力レジスタに格納する．

送受信の制御にはデータの送受の別，入出力レジスタ番号，データ長に関する情報が必要である．これらはプログラム開発時に得られ，プログラムロードと同時にホストからドメインコントローラを介して入出力ポートの入出力管理表にロードされる．シーケンサは，この表を順次読み出して入出力ポートの送受信を行なう．このため，回線上にはデータ以外の制御情報を流す必要がない．

受信応答が必要な場合は空のデータの転送という形で行なう．

シフトレジスタは最大データ長分の大きさを持っている必要があるので，64ビット程度が必要である．

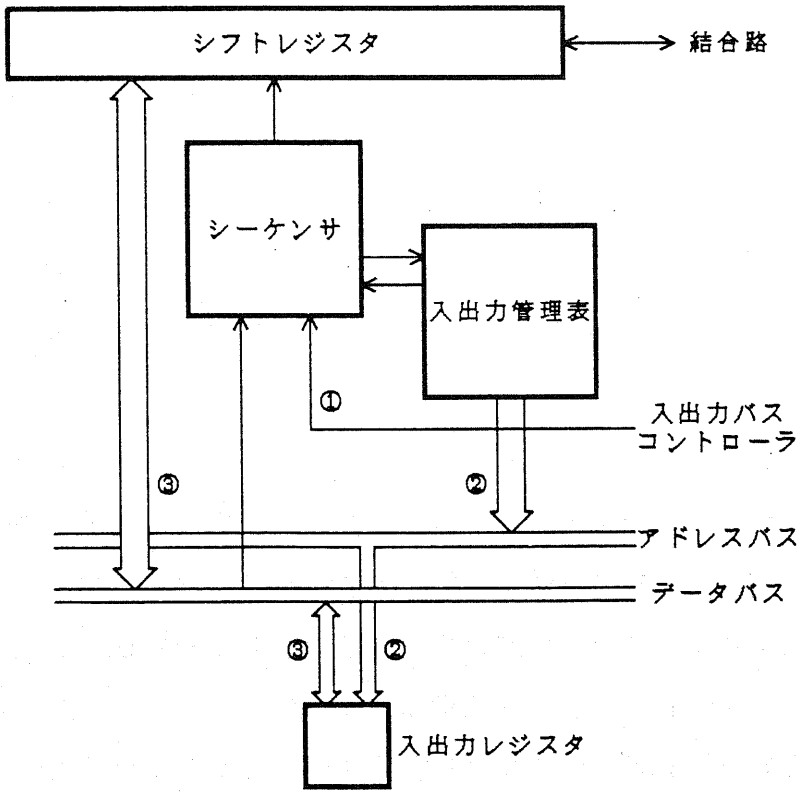


図４-１７．入出力ポートの構成

4.8. 入出力制御

入出力レジスタは複数の入出力ポートと演算装置からアクセスされるため，このアクセス制御が問題である．

入出力ポートにデータが到着した場合，入出力レジスタが空なら問題ないが，空でなければ空になるまで待つ必要がある．その間ポートをふさいでしまうこと自体は問題ではないが，レジスタが空になったのをどのようにして入出力ポートが検出するのかが問題である．

同様に，入出力ポートが送信可能になった時に入出力レジスタにデータが書き込まれていれば問題ないが，入出力レジスタが空の場合はデータが書き込まれるまで待つ必要がある．この場合も，入出力レジスタにデータが書き込まれたことをどのようにして入出力ポートが検出するかが問題となる．

入出力ポート数を16～32，入出力レジスタ数を64～256程度と想定しているので，これらの間の接続にはバス形式が現実的である．

入出力ポートあるいは入出力レジスタのどちらかに能動的動作をさせることが必要だが，数の上からは入出力ポートの方が少ないので，こちらに制御をさせる方が実現が容易である．

そこで，多数の入出力ポート間のアクセス制御としては，

①固定タイムスロット割当て
②先着順制御
③ラウンドロビン制御

等が考えられる．

入出力ポートから入出力レジスタへのアクセスには，データの転送の他に状態のセンスもあり，このためにバスの使用率が極めて高くなる．例えばデータの出力の場合，入出力ポートは入出力管理表に従い入出力レジスタからのデータを読み出そうとするが，この時点でまだ演算が終了していない場合，入出力ポートは演算が終了するまで入出力レジスタをセンスしつづける必要がある．1つのデータを演算により消費または生成するための時間と1つのデータを送受信するための時間の比は通常1より大きく，従って上記のような状況はしばしば起こり，バ

スの使用率が100%近くなるものと考えられる．

バスの使用率が低い時は②や③の制御方式は効率が良いが，負荷が大きくなると競合制御のオーバヘッドのために効率が悪くなる．これに対し，①の制御方式は負荷によらず効率が一定であり，特に高負荷時においても他の方式に比べ効率が良いため，本システムでは①の方式が適当である．

①の方式では，ポート数を16としても20MHzで動作させれば入出力ポートからの出力レジスタへの平均アクセス時間は0.5μs以下になる．

演算装置も入出力レジスタにアクセスするが，これについては入出力レジスタを2ポートレジスタとして構成しておくことにより入出力ポートとは独立にアクセスできる．

各入出力レジスタに対応して状態フラグを設け，入出力ポート，演算装置いずれからのアクセスに対しても，読み出し時はリセット，書き込み時はセットする．

入出力ポートからのアクセスのタイミングは図4-17において次のように3段のパイプライン動作をさせる．

①入出力制御装置から入出力ポートへ選択信号を出す．

②選択されたポートからアドレスバスにレジスタ番号とリード／ライト信号を出す．

③指定されたレジスタは状態フラグを返し，データバスを介して，リードならポートからレジスタに，ライトならレジスタからポートに，データを転送する．

この①，②，③の動作をそれぞれ別々のバスを用いてタイミング的にオーバラップさせ，20MHz程度のクロックで動作させれば1ポート当たり1Mワード／秒程度のデータ転送レートが得られると考えられる．

第5章

処理要素の演算部の原理と構成

本章では処理要素の演算部の動作原理と構成について述べる．まず処理要素全体の機能上の構成を示し，次に演算実行制御方式の検討を行ない，各部の構成についての概念設計を行なう．

5.1.処理要素の全体的構成

処理要素の機能は，回線接続機能，通信制御機能，演算実行制御機能，演算処理機能に分けられる．処理要素の構成を，この機能に基づいて整理したものを図5－1に示す．

これらの各機能は，プログラム実行の階層構造の各階層に対応する．

回線接続機能および通信制御機能については第4章に述べた．

演算実行制御機能および演算処理機能においては，各処理要素に割り付けられた処理のプログラムを記憶し，通信機能により送受信されたデータを用いて処理の実行を行なう．

演算実行制御機能では回線接続および通信制御の各機能を用いて転送されるデータを管理することにより他の処理要素との処理の同期制御とスケジューリングを行なう．

演算処理機能では命令メモリから与えられる命令ストリームに従い，受信されたデータおよび内部に記憶されているデータを用いて実際の演算を行なって結果を入出力レジスタに格納する．これらの実現方式はシステム全体のプログラム実行効率に大きく影響する．

以下では演算実行制御および演算処理の方式に関する検討を行なう．

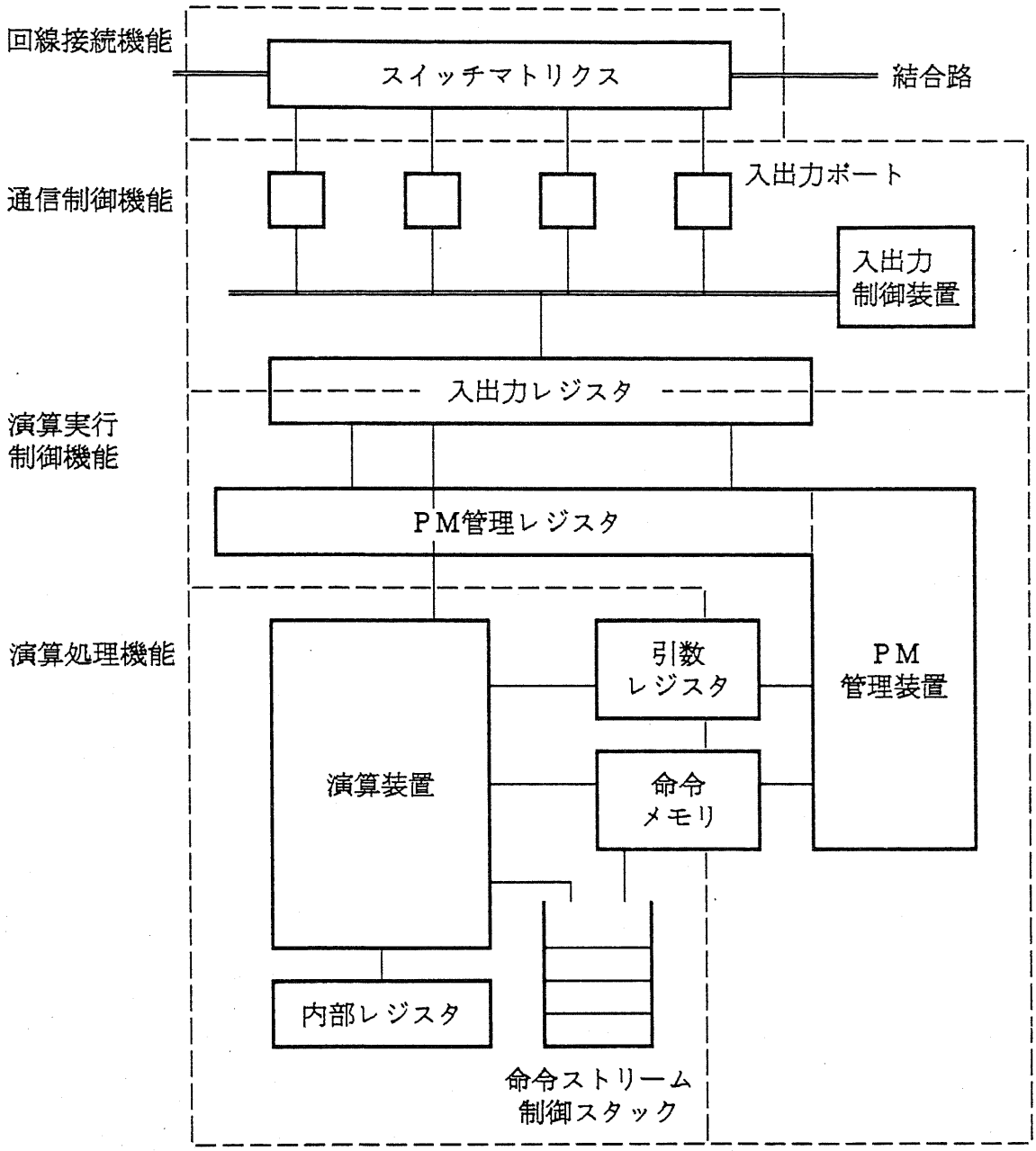


図5-1．処理要素の機能と構成

5.2.演算実行制御方式

演算実行制御機能では，他の処理要素との処理の同期や処理要素内の処理のスケジューリングが主な機能となる．

処理要素は物理的に分散しており，動作は非同期であって，共通メモリのように全ての処理要素からアクセスできる資源は存在しないため，処理要素間の処理の同期方式は必然的にデータ駆動的になる．これを逐次制御により実現しようとすると，データの到着や送出の状態の管理をプログラムにより行なわねばならず，全てのデータの送受信や処理の実行の順序が定まっている場合は良いが，そうでない場合はポーリング等を行なってデータの状態のテーブル管理を行なう必要があり，オーバヘッドが大きくなる．これに対し，データ駆動制御をハードウェアで実現すれば上記のようなオーバヘッドは極めて小さくできる．本システムでは通信はすべて回線接続で行なっており，各回線上のデータの順序等も定まっているため，このような制御方法とのマッチングが良く，汎用のデータフロー計算機の制御構造[Dennis 1980][Arvind 1982][Oyama 1984]に比べ極めて簡単化できる．

一方，スケジューリングに関しては，逐次制御では細かい制御が可能であるのに対し，ハードウェアによる制御では固定的になり柔軟性に欠ける．中間的な方法として，データ駆動制御をハードウェアで行ない，スケジューリングをプログラムによる逐次制御で行なうことも考えられる．しかし，本システムではマルチプロセスのような複雑なスケジューリングを必要とする使用形態は考えておらず，固定的で一次元的な単純な優先順位制御で充分であると考えられ，演算実行制御機能ではハードウェアによりデータ駆動制御と固定優先順位を実現することにする．

演算実行制御レベルの逐次性については，入出力レジスタを介さずに演算装置からＰＭ管理装置に直接的にフィードバックをかける手段を設けることで明示的に行なうこともできるが，実行制御の統一性が乱れ構造が複雑になる．このような制御は同一の処理要素内においてあるプログラムモジュールの出力レジスタを別のプログラムモジュールの入力レジスタに指定し，この入出力レジスタへのデータの書き込みを制御することにより間接的にではあるが実現できるので，プログラムモジュールレベルにおける演算同期制御およびスケジューリングはすべ

て入出力レジスタを経由してデータ駆動制御により行なうことにする．入出力レジスタを介することによる遅延は後述のように2～4クロック程度であるので，このオーバヘッドは無視し得る程度に小さい．

また，データ駆動制御においても履歴依存性を完全に排除して，全ての情報を入出力レジスタを介して渡すことにすると，定型的であって密に関連し合うプログラムモジュール間の通信にはオーバヘッドが大きくなるので，演算装置内に内部レジスタを設けて履歴を許し，同一処理要素内のプログラムモジュール間の通信のオーバヘッドの低減をはかる．

5.3.演算処理

本システムでは，5.2節で述べたように演算実行制御機能を実現するためにデータ駆動制御を採用するため，演算処理機能にはフォンノイマン型のような汎用性は要求されず，むしろ完全に固定的な処理シーケンスを高速に実行できる方が良い．そこで演算処理機能ではプログラムを完全な命令のストリームと考え，分岐やサブルーチンのような制御構造を取り除き，プログラムカウンタの概念を排除することにより逐次制御の高速化を図る．代わりに，条件フラグにより実行したりしなかったりする条件付き演算を用いて簡単な条件文はそのまま記述できるようにする．

しかし，実際のプログラム時には各種の乗除算のようにプログラムモジュールよりも低いレベルで，入出力を伴わない定型的な処理シーケンスも存在し，これをマクロ命令のようにプログラム中で呼び出せるような機構があると命令ストリームをより短く記述することができる．そこで，このような処理単位をサブ命令ストリームと呼ぶことにし，このサブ命令ストリームを起動するための手段として命令ストリーム制御スタックを設ける．逐次的な命令ストリームに完全には展開できないループや分岐，サブルーチン等のプログラム制御構造に代わるものとしてこのサブ命令ストリームを用いる．命令ストリーム制御スタックを用いたサブ命令ストリームの起動方法については後述するが，これにより非常に単純な構造でありながらフォンノイマン型プロセッサの基本的機能と同程度の機能を提供することができる．

フォンノイマン型ではプログラムの命令コードと処理の対象であるデータとが論理的に同一の経路を通してアクセスされるうえ，演算と逐次制御が同レベルで扱われているために命令フェッチとデータアクセスのパイプライン化が複雑になり，高速化方式を複雑化している．これに対し，本システムでは命令コードは命令メモリ内に記憶され，データとは独立してアクセス可能であるうえに，演算を行なう演算装置と逐次制御を行なう命令メモリとが独立しており，制御のフィードバックの経路は命令ストリーム制御スタックだけであるため，命令メモリの構成が単純になる．

5.4.演算部の構成

本節では上述の演算実行制御および演算処理の各機能を実現するための処理要素の演算部の各部の構成について概念設計を行なう．

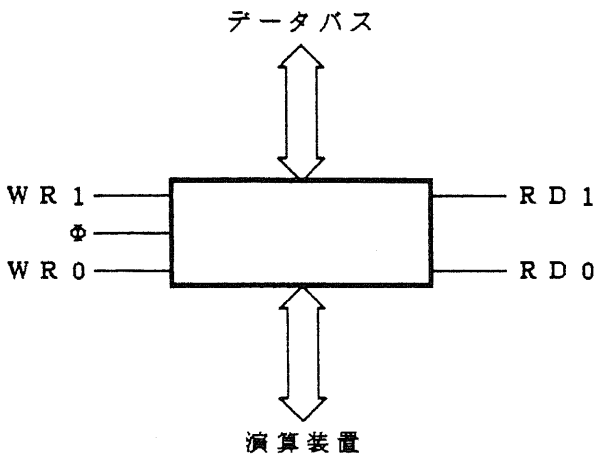
5.4.1.入出力レジスタ

5.3節に述べたように入出力レジスタは2ポートレジスタとする．また，演算実行制御による実行可能性の管理や入出力ポートからのセンスのために，各入出力レジスタごとに状態フラグを設置する．これを実現するための入出力レジスタの構成を図5－2に示す．PRSは入出力ポートからの入出力レジスタの選択信号であり，レジスタアドレスを用いて各入出力レジスタごとあるいは入出力バス制御装置において作る．ORSは演算装置からの入出力レジスタの選択信号である．PRWは入出力ポートからのリード／ライト信号，ORWは演算装置からのリード／ライト信号である．アクセスは演算装置を優先させており，状態フラグはいずれからにおいても書き込み時にセット，読み出し時にリセットするようになっている．

5.4.2.ＰＭ管理レジスタ

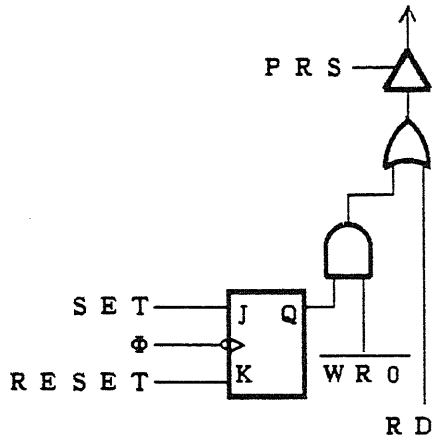
データ駆動制御においては，入力レジスタセットがすべて読み出し可能，出力レジスタセットがすべて書き込み可能になった時にプログラムモジュールが実行可能になる．この実行可能ＰＭ管理の実現上，最も問題になるのが，プログラムモジュールが実行可能であることを検出する方法である．これにはデータフロー計算機で行なっているように[Gostelow 1980][Nguen 1982][Oyama 1984.1]，データが到着するたびにテーブルを更新して実行可能性を調べるというような方法は，オーバヘッドが大きいため現実的でない．従って，実行可能性の検出はハードウェアで行なう必要がある．これには図5－3に示すようにＰＭ管理レジスタをプログラムモジュールの数だけ置いて管理させれば良い．

ＰＭ管理レジスタの各ビットと，対応する入出力レジスタの状態フラグとのＯＲを取り，各レジスタごとに全ビットのＡＮＤを取ると，出力が｀１´のＰＭ管理レジスタに対応するプログラムモジュールが実行可能であることになる．ただし，出力レジスタに関しては論理を反転して同様の処理を行なう．



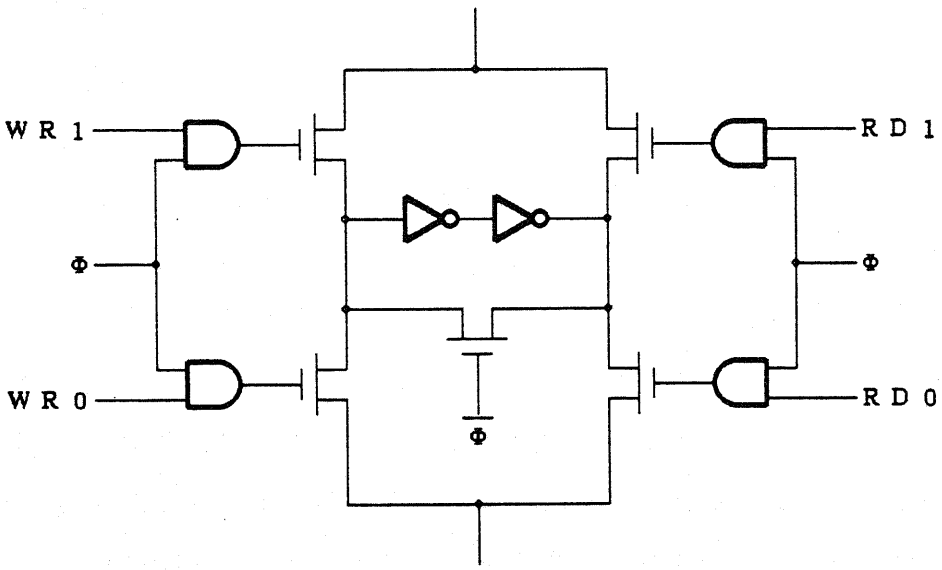
$$WR0 = ORS \cdot \overline{ORW}$$
$$WR1 = \overline{ORS} \cdot PRS \cdot \overline{PRW}$$
$$RD0 = \overline{ORS} \cdot ORW$$
$$RD1 = \overline{ORS} \cdot PRS \cdot PRW$$

(a) 2ポートレジスタ



$$SET = OWR + PWR \cdot \overline{ORD}$$
$$RESET = ORD + PRD \cdot \overline{OWR}$$

(b) 状態フラグ



(c) 2ポートレジスタの回路

PRS：入出力ポートからの選択信号，PRW：入出力ポートからのリード／ライト信号，
ORS：演算装置からの選択信号，ORW：演算装置からのリード／ライト信号

図5-2．入出力レジスタの構成

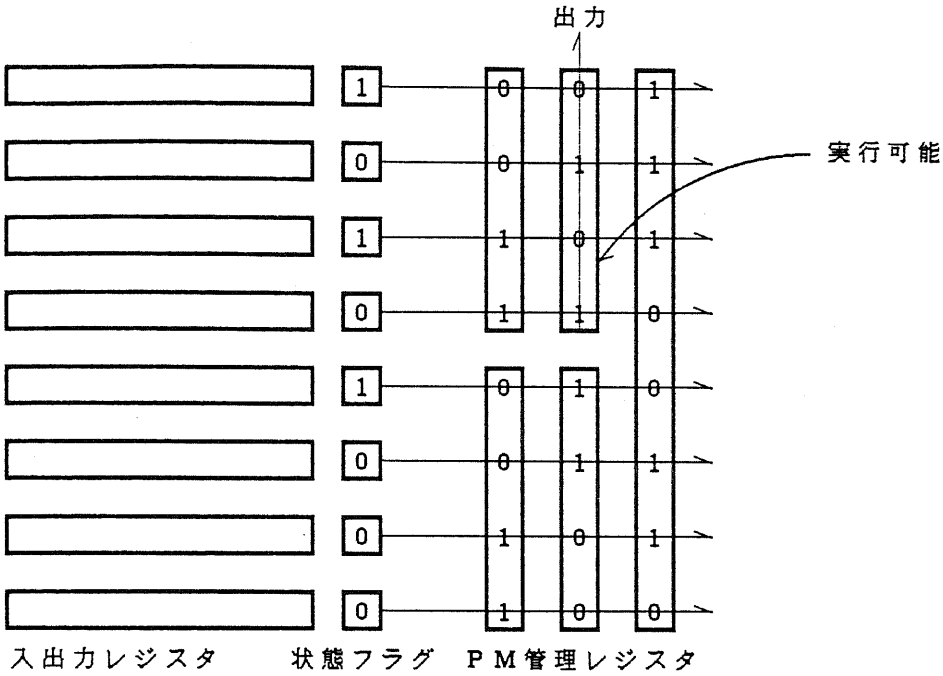


図５－３．実行可能ＰＭ管理方式

5.4.3. ＰＭ管理装置

ＰＭ管理装置はデータ駆動制御レベルのプログラムをＰＭ管理表に記憶し，ＰＭ管理レジスタによって検出された実行可能プログラムモジュールの番号をＰＭ選択装置を介して受け取ると，実行中のプログラムモジュールの終了を待ってＰＭ管理表を引き，引数レジスタに引数を設定してから命令ストリーム番号を命令メモリに与えて演算処理を開始させる．ＰＭ管理装置の構成を図５－４に示す．

ＰＭ選択装置はＰＭ管理レジスタによって複数の実行可能プログラムモジュールが検出されている場合にこれらの内から１つだけ選択する．これには固定優先順位制御，先着順制御，ラウンドロビン制御等が考えられる．これらの制御方法はスケジューリングとも関係するが，前節にも述べたようにスケジューリングにはあまり複雑な制御を行なう必要はないと考えられるので，ハードウェアの実現が最も容易な固定優先順位とする．この構成例を図５－５に示す．ＰＭ管理レジスタ数を256個としても16入力のプライオリティエンコーダを２段にすれば実現できる．選択のための遅延も２～４クロック程度と小さくすることができる．

ＰＭ管理表には各プログラムモジュール番号に対応して引数と命令ストリーム

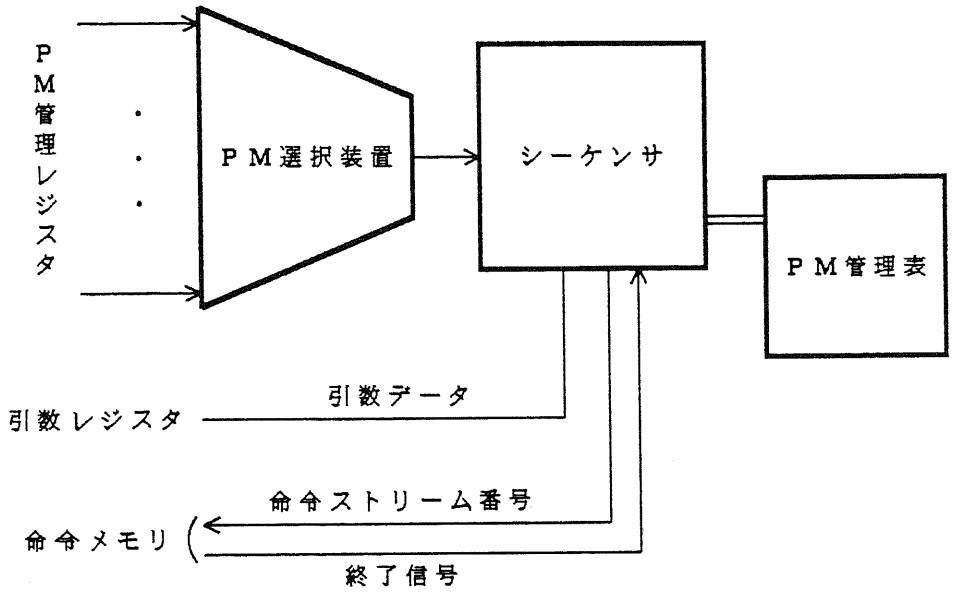


図5-4．PM管理装置の構成

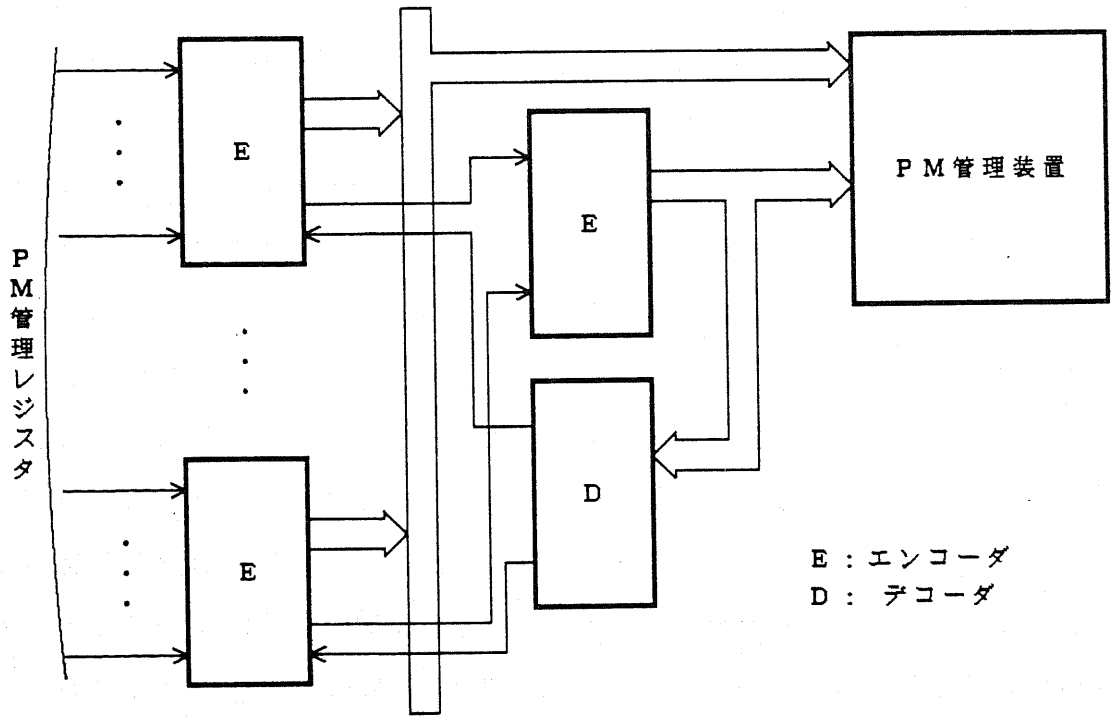


図5-5．PM選択装置

要素ごとあるいはプログラム実行ごとに異なるような定数値，同一処理要素内でのデータの授受のための内部レジスタ番号などが含まれる．複数のプログラムモジュールが同一の命令ストリームを共有することも可能である．

シーケンサの実現はハードウェアによっても可能であるが，テーブルアクセスや可変個の引数を扱う必要性があるため，マイクロプログラム制御とする方が実現は容易である．

5.4.4.命令メモリ

命令メモリは逐次制御レベルのプログラムである命令ストリームの内容を記憶している．ＰＭ管理装置から実行可能となって選択されたプログラムモジュールの命令ストリーム番号を受け取ると，この番号に対応した命令ストリームを順次取り出して演算装置に与える．

命令ストリームは完全に一次元的な命令列であり，読み出される順番が決まっているため，これを記憶するための記憶装置も逐次的なアクセスが可能な構造を持っている必要がある．このためにはＲＡＭとアドレスカウンタの組み合わせ，シフトレジスタ，磁気バブル，ＣＣＤ等による構成が考えられるが，処理要素を従来の技術を用いて１チップで実現するためにはＲＡＭとアドレスカウンタを用いて構成する方式が現実的である．シフトレジスタを用いる方式も構成を工夫することにより実現可能であると考えられ，ＲＡＭを用いる方式より効率の良く実現できる可能性もある．

ＲＡＭを用いる方式による命令メモリの構成を図５－６に示す．シーケンサは命令ストリーム番号を用いて命令ストリームエントリ表を引き命令ストリームの開始番地を取り出し，これをアドレスカウンタにセットしてＲＡＭに与えることにより演算装置に１命令ずつ与える．

命令ストリームの終了の検出は，命令ストリーム長を用いてカウントする方法と，命令ストリームの最後に終了コードを付加してこれを検出する方法があるが，後者の方が回路が簡単で実現が容易である．

命令ストリームが終了したときに命令ストリーム制御スタック中にサブ命令ストリーム番号が入っている場合は対応するサブ命令ストリームを起動する．サブ命令ストリームは起動方法が異なるだけで，その他の扱いは命令ストリームと同

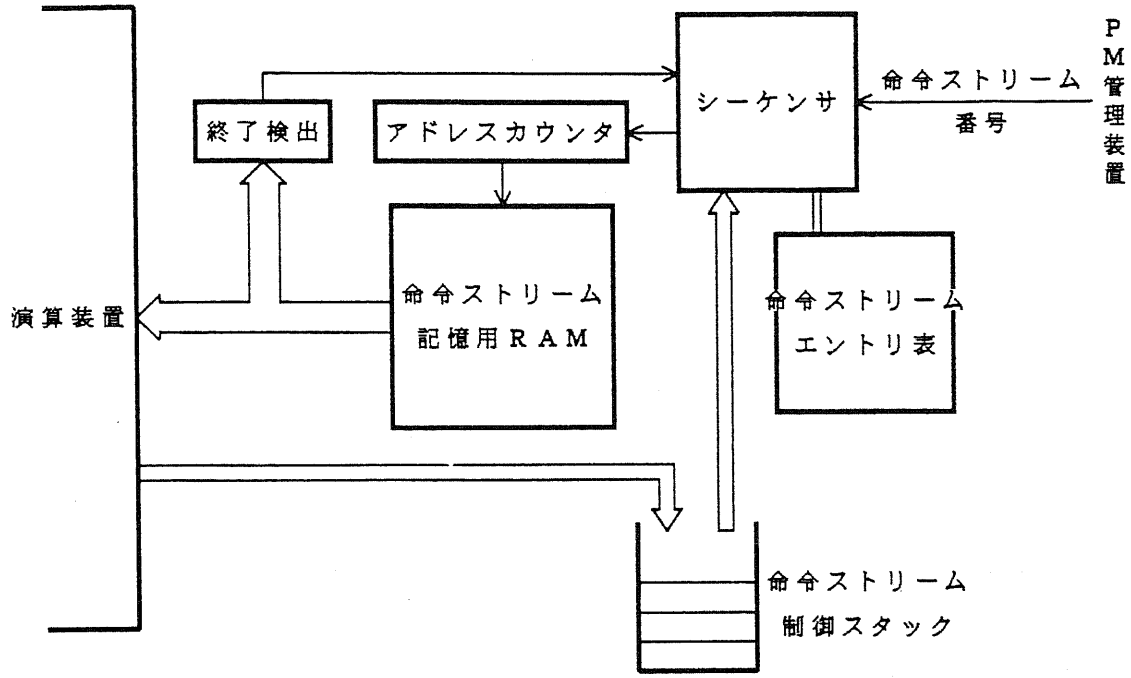


図5-6．命令メモリの構成

等である．起動の動作は以下のようになる．演算装置が次に実行すべきサブ命令ストリーム番号を命令ストリーム制御スタックに積む．実行中の命令ストリームが終了したらシーケンサはスタックの先頭のサブ命令ストリーム番号を取り出して，命令ストリームの実行と同様に命令ストリームエントリ表を引き，アドレスカウンタに値を設定して演算装置に命令ストリームを与える．複数のサブ命令ストリームを連続的に実行させる場合はこれらをすべて命令ストリームに積んでおくこともできる．

演算装置においては命令が全て1マシンサイクルで実行を終了するように命令セットを設計することにより命令メモリと演算装置の間のハンドシェークが不要となり，ハードウェア量の低減と処理の高速化が可能になる．

5.4.5.引数レジスタ

引数レジスタはPM管理装置から書き込み，演算装置から読み出しが可能な通常のレジスタである．レジスタ数は命令ストリームへの引数の最大数となるが，16位が適当であろう．

5.4.6.演算装置

演算装置は命令メモリから与えられた命令に従い，入出力レジスタや内部レジスタにアクセスしながら命令ストリームを実行し，演算処理を行なう．演算装置の構成を図5－7に示す．演算装置には逐次制御装置は置かず，すべての命令は1マシンサイクルで終了しなければならない．命令メモリから与えられる命令は命令デコーダによりレジスタレベルの制御信号に変換され演算装置の各構成要素に与えられる．従って，命令は垂直型マイクロプログラムレベルに近いものとなる．

命令メモリから与えられる命令に条件付きビットが付加されている場合は，条件フラグ（ＣＦ）がオンの時は命令デコーダを動作状態，オフの時は停止状態とすることにより条件付き演算を実現する．

算術論理演算装置（ＡＬＵ）や各種のレジスタのワード幅は半導体集積技術にもよるが，現在の水準では8ビット程度は充分に実現可能であろうと考えられるので，演算に関してはデータはすべて8ビットと仮定する．

ＡＬＵの機能としては加減算，論理演算，1ビットシフト程度を想定しているが，可能であればバレルシフタや並列乗算器も実装する．

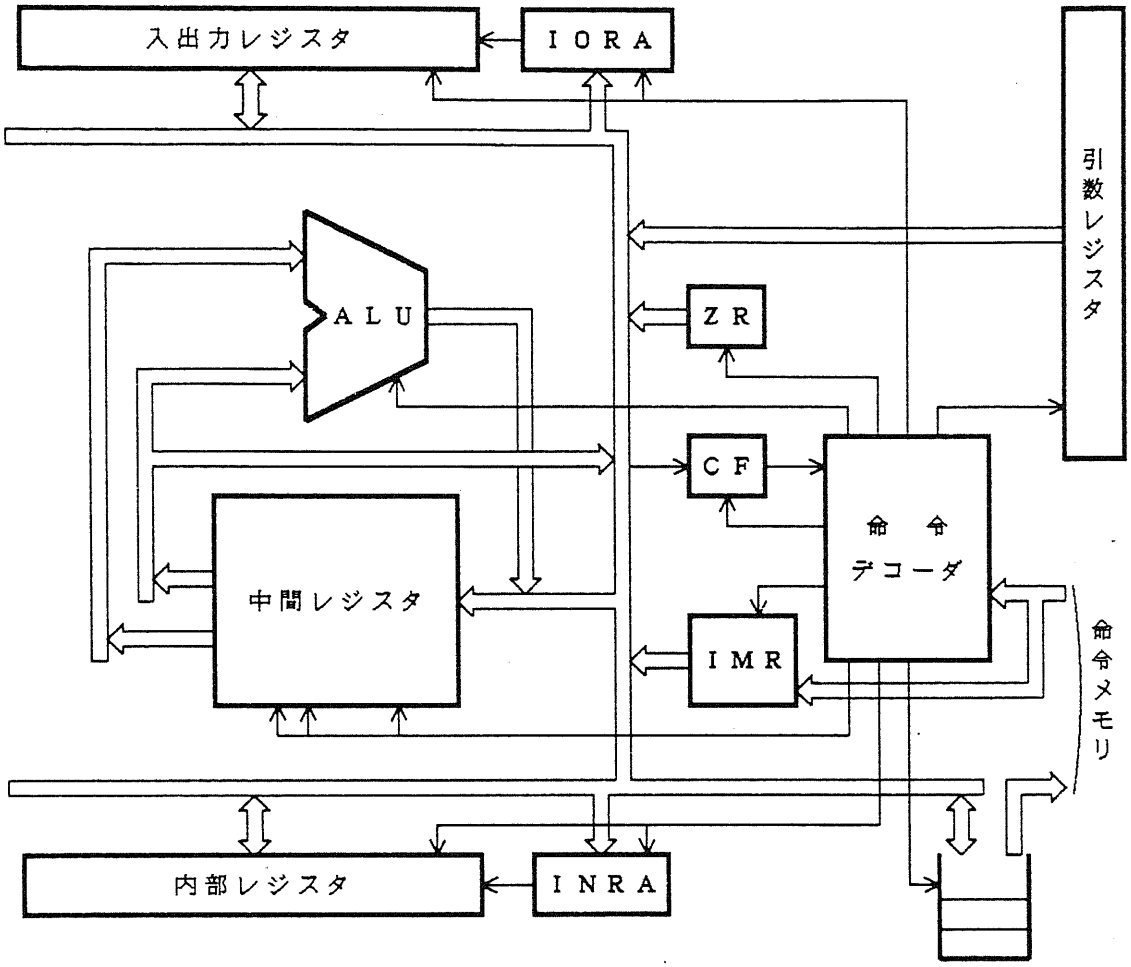
中間レジスタファイルは，命令ストリームの中で中間的に使用するもので，アキュムレータを兼ねる．

入出力レジスタと内部レジスタへのアクセスはＩＯＲＡおよびＩＮＲＡにレジスタ番号をロードしてから入出力レジスタあるいは内部レジスタへリード／ライトを行なう形となる．

命令ストリーム制御スタックは演算装置からはプッシュ／ポップ，命令メモリからはポップが可能なハードウェアスタックで，この実現には双方向のシフトレジスタを用いることができる．

イミディエットレジスタ（ＩＭＲ）は命令ストリームに固有の定数を命令ストリーム中に記述するために用いられ，ロードイミディエットレジスタ命令のコード中の下位8ビットをそのまま保持する．

データの移動は任意のレジスタおよび命令ストリーム制御スタック間で可能であるとする．中間レジスタを32個，引数レジスタを16個とすると，指定すべきレジスタの総数は56個となり，ソースとデスティネーションの指定にそれぞれ6



IMR：イミディエットレジスタ
INRA：内部レジスタアドレス
IORA：入出力レジスタアドレス
ALU：算術論理演算装置
CF：条件フラグ
ZR：ゼロレジスタ（値が常に0のレジスタ）

図5-7．演算装置の構成

ビット程度必要になる．

また，演算は2個の中間レジスタのデータ間に対して実行可能であるとする．中間レジスタを32個とすると，ソースとデスティネーションの指定にそれぞれ5ビット程度必要で，演算の種類を32とするとこれに5ビット必要である．従って条件付きビットを加えると，命令ストリームの命令ワード幅は16ビット程度となる．

命令セットとしては表5-1に示すようなものを想定している．命令にはシングルオペランドとダブルオペランドの演算，条件フラグ関係とステータス関係の命令，転送命令(mov)，イミディエットレジスタへのロード命令(imld)，命令ストリームの終了命令(end)がある．演算命令のオペランドは全て中間レジスタであり，転送命令は任意のレジスタと命令ストリーム制御スタックをオペランドのとりうる．オペランドを表5-2に示す．なお，条件付き命令は命令名のあとに".c"を付けて示す．

SINGLE OPERAND OPERATION

clr	Clear destination
com	Complement destination
inc	Increment destination
dec	Decrement destination
neg	Negate destination
tst	Test destination
asr	Arithmetic shift right
asl	Arithmetic shift left
lsr	Logical shift right
lsl	Logical shift left
rsr	Rotate shift right
rsl	Rotate shift left

DOUBLE OPERAND OPERATION

cmp	Compare source to destination
add	Add source to destination
addc	Add source to destination with carry
sub	Subtract source from destination
subc	Subtract source from destination with carry
bit	Bit test
bic	Bit clear
bis	Bit set
eor	Exclusive OR

表5-1．演算装置の命令コード（続く）

CONDITION CONTROL

scf Set condition flag (unconditional)
scne Set condition flag if not equal (to zero)
sceq Set condition flag if equal (to zero)
scpl Set condition flag if plus
scmi Set condition flag if minus
scvc Set condition flag if overflow is clear
scvs Set condition flag if overflow is set
sccc Set condition flag if carry is clear
sccs Set condition flag if carry is set
scge Set condition flag if greater than or equal
sclt Set condition flag if less than
scgt Set condition flag if greater than
scle Set condition flag if less than or equal
schi Set condition flag if higher
scls Set condition flag if lower or same
schs Set condition flag if higher or same
sclo Set condition flag if lower

STATUS CONTROL

clc Clear carry flag
clv Clear overflow flag
clz Clear zero flag
cln Clear negative flag
sec Set carry flag
sev Set overflow flag
sez Set zero flag
sen Set negative flag

MISCELLANEOUS

mov Move source to destination
imld Load immediate register
end End of instruction stream (no operation)

表5-1．演算装置の命令コード

r n	(n=0~31)	中間レジスタ
ar n	(n=0~15)	引数レジスタ
iora		入出力レジスタアドレス
iord		入出力レジスタデータ
inra		内部レジスタアドレス
inrd		内部レジスタデータ
imr		イミディエットレジスタ
iscs		命令ストリーム制御スタック
zr		ゼロレジスタ（常に0であるレジスタ）

表5-2．オペランド

第6章

プログラム開発に関する検討

本システムのソフトウェア構成については3.7節に述べた．このうち，高級言語および基本言語コンパイラ，論理的および物理的処理要素割付け，回線設定がプログラム開発に関するシステムソフトウェアであり，これらに対する入力としてプログラムを記述する必要がある．

プログラムは高級言語または基本言語で記述されることを想定しているが，高級言語の仕様やコンパイラの実現については本研究の主目的からはずれ，また，基本言語コンパイラについても，システムのインプリメントが行なわれていない現状では開発する意義に乏しいので，対象から除外し，本章では基本言語によるプログラムの記述について述べる．

基本言語によるプログラムの記述においては3.1節に述べたシステムの全体構成の各機能に対応してプログラムも階層をなしており，まずこのプログラム階層について述べ，次に各階層における記述の内容や方法を実例を用いて示す．

6.1. プログラムの階層

プログラムとして記述しなければならない情報は，システムのプログラム実行機能の階層に対応して，

- 命令ストリームの内容（逐次処理レベル）
- プログラムモジュールの入出力レジスタセット，内部レジスタセット，定数および命令ストリーム名（データ駆動レベル）
- 入出力ポートと入出力レジスタの対応（通信制御レベル）
- 処理要素間の接続関係（回線接続レベル）

がある．

基本言語によるプログラムの記述では，論理的処理要素の割付けを意識して入出力レジスタや論理回線の接続を記述しなければならない．つまり，どのプログラムモジュールを同一の処理要素で実行させるか決めておかなければならない．

6.2.逐次処理レベルの記述

逐次処理レベル，すなわち，命令ストリームは，命令ストリーム名と，入出力レジスタ，内部レジスタのローカル名，レジスタの型宣言，演算処理内容により記述される．

高級言語レベルの記述例を図6-1に示す．これはＦＦＴバタフライに相当する命令ストリームである．

第1行の"butterfly"が命令ストリーム名，括弧内が引数である．引数は入力レジスタ，出力レジスタ，内部レジスタ，定数の4つのフィールドより成り，これらを';'（セミコロン）で区切って記す．この例では定数引数は使われていない．

第2行は引数の型の宣言である．

命令ストリームの処理内容は第4～8行の'{'と'}'で囲まれた部分に記述されている．第4行は中間レジスタの宣言，第5～8行は演算である．

これをアセンブラレベルにコンパイルする場合，単一の命令ストリームに翻訳することもできるが，いくつかのサブ命令ストリームの組合わせとすることも可能である．例えばcomplexの乗算を行なうサブ命令ストリーム"cmpmul"により実現するような場合を考えてみる．図6-2に示すように，"cmpmul"を呼び出すときはまず続きの処理を記述したサブ命令ストリーム"s1_butterfly"の番号を命令ストリーム制御スタックに積み，次にサブ命令ストリーム"cmpmul"の番号を命令ストリーム制御スタックに積んで命令ストリームを終了する．これにより，命令

```
1       i_stream butterfly(x1,x2; y1,y2; w; )
2       complex x1,x2,y1,y2,w;
3       {
4               complex s,t;
5               s = x1;
6               t = x2 * w;
7               y1 = s + t;
8               y2 = s - t;
9       }
```

図6-1．命令ストリームの記述例（高級言語）

```
.i_stream _butterfly

    （処理内容）

imld    s1_butterfly  (続きの処理のサブ命令ストリーム番号をimrにロード)
mov     imr,iscs      (imrの内容を命令ストリーム制御スタックに積む)
imld    cmpmul        (complex乗算のサブ命令ストリーム番号をimrにロード)
mov     imr,iscs      (imrの内容を命令ストリーム制御スタックに積む)
end                   (命令ストリームの終了コード)

.i_stream s1_butterfly(続きの処理のサブ命令ストリームの定義)

    (続きの処理内容)

end
```

図６－２．サブ命令ストリームの組合わせによる命令ストリームの記述例

ストリーム"butterfly"が終了すると自動的にサブ命令ストリーム"cmpmul"が実行され，これが終了すると続きのサブ命令ストリーム"s1_butterfly"が実行される．

条件付き演算を用いた命令ストリーム例として，入力データを条件に応じてスイッチする命令ストリームの高級言語とアセンブラレベルの記述例を図６－３に示す．引数は順に引数レジスタar0～ar3に割り当てられている．

第102，103行で"cond"を読み出し，第104，105行で制御フラグの設定，第106～109行で"in"を"out1"に代入，第110行でelseに対応して制御フラグの設定，第111～114行で"in"を"out2"に代入している．

```
1       i_stream switch( in,cond; out1,out2; ; )
2       int in,cond,out1,out2;
3       {
4               if (cond==0)                    ①
5                       out1 = in;              ②
6               else                            ③
7                       out2 = in;              ④
8       }
```

(a)スイッチ命令ストリーム（高級言語）

```
101     .i_stream _switch
102     mov     ar1,iora        ┐
103     mov     iord,r0         │ ①
104     tst     r0              │
105     sceq                    ┘
106     mov.c   ar0,iora        ┐
107     mov.c   iord,r1         │ ②
108     mov.c   ar2,iora        │
109     mov.c   r1,iord         ┘
110     scne                    ③
111     mov.c   ar0,iora        ┐
112     mov.c   iord,r1         │ ④
113     mov.c   ar3,iora        │
114     mov.c   r1,iord         ┘
115     end
```

(b)スイッチ命令ストリーム（アセンブラレベル）

図６－３．条件付き演算を用いた命令ストリーム例

6.3. データ駆動レベルの記述

データ駆動レベルでは，各論理的処理要素ごとに，入出力レジスタと内部レジスタの型と名前，プログラムモジュールの命令ストリーム名と引数を記述する．

記述例を図６－４に示す．これは64個の処理要素による128点ＦＦＴのプログラムである．

第１行で論理処理要素番号を指定する．この例ではプログラムモジュールは全処理要素に共通である．第２～４行では以下のプログラムモジュールで使用する入出力レジスタおよび内部レジスタの型と名前を宣言する．第５～７行と第１０行は各行が１つのプログラムモジュールに対応している．第８，９行では第９行の"i"の値を０～４のそれぞれの値に置き換えた形のプログラムモジュールを定義する．各プログラムモジュールの定義では，対応する命令ストリーム名と，括弧の中に入力レジスタ，出力レジスタ，内部レジスタ，定数を記述する．

この例では，"ivar"は初期データであり，この入力レジスタへのロードについてはここには含まれていない．"w"は各処理要素に固有の定数であり，予め計算かホストからのロードにより内部レジスタに記憶されているとする．また"fvar"は結果データであり，この出力レジスタからのアンロードについてもここには含まれていない．命令ストリーム"copy"は単に入力レジスタから出力レジスタへ値を移す命令ストリームである．

```
1       p_module (0..63) {
2               input complex ivar[2],brin[2],shin[6,2];
3               output complex brout[2],shout[6,2],fvar[2];
4               internal complex w[7];

5               copy(ivar[0]; brout[0]; ; );
6               copy(ivar[1]; brout[1]; ; );
7               butterfly(brin[0],brin[1]; shout[0,0],shout[0,1]; w[0]; );
8               foreach i (0..4)
9                   butterfly(shin[0,0],shin[0,1];shout[1,0],shout[1,1];w[1];);
10              butterfly(shin[5,0],shin[5,1]; fvar[0],fvar[1]; w[6]; );
11      }
```

図６－４．プログラムモジュールの記述例

6.4.通信制御レベルの記述

通信制御レベル，すなわち，入出力ポートと入出力レジスタとの対応関係は，各論理的処理要素ごとに，入出力ポート名とアクセスされる入出力レジスタ名を記述する．1つの入出力ポートが複数の入出力レジスタに対して順次送受信を行なう．これらには送信と受信が混在することもあり，この別も記述する必要がある．

記述例を図6－5に示す．前節同様，64処理要素による128点ＦＦＴのプログラムである．第1行で論理処理要素番号を指定する．この例では通信制御は全処理要素に共通である．第2～9行が入出力ポートと入出力レジスタの対応関係を記述している．入出力ポートごとに入出力ポート名と，括弧の中にはそれに対応する入出力レジスタの入出力の別と名前をアクセスされる順に記述する．第6行は

```
p_shout0 = {o:shout[0,0],o:shout[1,0],･･･,o:shout[5,0]};
```

と同じ意味である．

```
1     port (0..63) {
2             p_brout0 {o:brout[0]};
3             p_brout1 {o:brout[1]};
4             p_brin0 {i:brin[0]};
5             p_brin1 {i:brin[1]};
6             p_shout0 {foreach n (0..5) o:shout[n,0]};
7             p_shout1 {foreach n (0..5) o:shout[n,1]};
8             p_shin0 {foreach n (0..5) o:shin[n,0]};
9             p_shin1 {foreach n (0..5) o:shin[n,1]};
10    }
```

図6－5．通信制御レベルの記述例

6.1.5.回線接続レベルの記述

回線接続レベル，すなわち通信を行なう論理的処理要素の入出力ポート間の接続関係は，回線の両端の入出力ポート名と論理的処理要素番号により記述する．

記述例を図6-6に示す．これはＦＦＴのためのビット逆順の並べ換えの回線と，シャフルエクスチェンジネットワークの回線である．

第1～20行はビット逆順，第21～39行はシャフルエクスチェンジの接続を記述したもので，それぞれ論理処理要素番号の組合わせにより4つにグループ分けされている．まず回線の両端に位置する入出力ポート名の組を指定し，次にこれに相当する回線を持つ論理処理要素の組を番号で記述する．なおこの例では回線の両端が同一の処理要素であったり方向が逆であるだけで処理要素が同じであるような回線は冗長であるがそのまま記述してある．

```
1       link (p_brout0,p_brin0) {
2               ( 0, 0),( 1,16),( 2, 8),( 3,24),( 4, 4),( 5,20),( 6,12),( 7,28),
3               ( 8, 2),( 9,18),(10,10),(11,26),(12, 6),(13,22),(14,14),(15,30),
4               (16, 1),(17,17),(18, 9),(19,25),(20, 5),(21,21),(22,13),(23,29),
5               (24, 3),(25,19),(26,11),(27,27),(28, 7),(29,23),(30,15),(31,31)}
6       link (p_brout0,p_brin1) {
7               (32, 0),(33,16),(34, 8),(35,24),(36, 4),(37,20),(38,12),(39,28),
8               (40, 2),(41,18),(42,10),(43,26),(44, 6),(45,22),(46,14),(47,30),
9               (48, 1),(49,17),(50, 9),(51,25),(52, 5),(53,21),(54,13),(55,29),
10              (56, 3),(57,19),(58,11),(59,27),(60, 7),(61,23),(62,15),(63,31)}
11      link (p_brout1,p_brin0) {
12              ( 0,32),( 1,48),( 2,40),( 3,56),( 4,36),( 5,52),( 6,44),( 7,60),
13              ( 8,34),( 9,50),(10,42),(11,58),(12,38),(13,54),(14,46),(15,62),
14              (16,33),(17,49),(18,41),(19,57),(20,37),(21,53),(22,45),(23,61),
15              (24,35),(25,51),(26,43),(27,59),(28,39),(29,55),(30,47),(31,63)}
16      link (p_brout1,p_brin1) {
17              (32,32),(33,48),(34,40),(35,56),(36,36),(37,52),(38,44),(39,60),
18              (40,34),(41,50),(42,42),(43,58),(44,38),(45,54),(46,46),(47,62),
19              (48,33),(49,49),(50,41),(51,57),(52,37),(53,53),(54,45),(55,61),
20              (56,35),(57,51),(58,43),(59,59),(60,39),(61,55),(62,47),(63,63)}
21      link (p_shout0,p_shin0) {
22              ( 0, 0),( 2, 1),( 4, 2),( 6, 3),( 8, 4),(10, 5),(12, 6),(14, 7),
23              (16, 8),(18, 9),(20,10),(22,11),(24,12),(26,13),(28,14),(30,15),
24              (32,16),(34,17),(36,18),(38,19),(40,20),(42,21),(44,22),(46,23),
24              (48,24),(50,25),(52,26),(54,27),(56,28),(58,29),(60,30),(62,31)}
25      link (p_shout0,p_shin1) {
26              ( 1, 0),( 3, 1),( 5, 2),( 7, 3),( 9, 4),(11, 5),(13, 6),(15, 7),
27              (17, 8),(19, 9),(21,10),(23,11),(25,12),(27,13),(29,14),(31,15),
28              (33,16),(35,17),(37,18),(39,19),(41,20),(43,21),(45,22),(47,23),
29              (49,24),(51,25),(53,26),(55,27),(57,28),(59,29),(61,30),(63,31)}
30      link (p_shout1,p_shin0) {
31              ( 0,32),( 2,33),( 4,34),( 6,35),( 8,36),(10,37),(12,38),(14,39),
32              (16,40),(18,41),(20,42),(22,43),(24,44),(26,45),(28,46),(30,47),
33              (32,48),(34,49),(36,50),(38,51),(40,52),(42,53),(44,54),(46,55),
34              (48,56),(50,57),(52,58),(54,59),(56,60),(58,61),(60,62),(62,63)}
35      link (p_shout1,p_shin1) {
36              ( 1,32),( 3,33),( 5,34),( 7,35),( 9,36),(11,37),(13,38),(15,39),
37              (17,40),(19,41),(21,42),(23,43),(25,44),(27,45),(29,46),(31,47),
38              (33,48),(35,49),(37,50),(39,51),(41,52),(43,53),(45,54),(47,55),
39              (49,56),(51,57),(53,58),(55,59),(57,60),(59,61),(61,62),(63,63)}
```

図６－６．回線接続レベルの記述例

第7章
資源割付けに関する検討

本章では，システムソフトウェアのうち，資源割付けに関するものについて述べる．3.7節に述べたようにプログラムを実行可能形式に直すためには高級言語および基本言語コンパイラと論理的および物理的処理要素割付け，および回線設定が必要になる．以下ではこれらのうち論理的および物理的処理要素割付けと回線設定について述べる．またこれらの評価についてはそれぞれ単独で行なうことは難しいので，物理的処理要素割付けと回線設定を合わせていくつかの実際の並列処理アルゴリズムのデータ依存関係に対して適用することにより評価する．また，現在は基本言語によるプログラムについてのみ考えているので，論理的処理要素割付けはすでに並列処理アルゴリズムに含まれた形となっているため，これについての評価は本論文では行なっていない[Hama 1984]．

7.1. 論理的処理要素割付け

論理的処理要素の割付けの目的は与えられた並列処理アルゴリズムの各プログラムモジュールをシステムの処理要素数に対応して効率的にグループ化し，処理要素の並列動作数を高めることによりシステムの処理能力の向上を図ることである．従って論理的処理要素割付けはシステムの処理能力に大きな影響を与えることになる．割付けにあたって考慮すべきこととしては以下のような事柄がある．

(1)並列処理アルゴリズムそのものの効率を損なわないために同時刻に処理を実行する可能性のあるプログラムモジュールはできる限り異なる処理要素へ割り付けることにより並列動作数の増加を図る．

(2)資源の有効利用のため，同時刻に動作することのないプログラムモジュールを同一の処理要素に対して重複して割付け，各処理要素の処理量の均等な分散を図る．

以上の2点は並列処理を目的とするマルチプロセッサシステムに共通した事柄である．本システムでの論理的処理要素割付けの際にはもう1点重要なことがある．処理要素間の通信を考えた場合，処理要素間の回線は処理実行前に全て設定され，処理実行中は固定であるため，1つの処理要素が通信を行なえる処理要素数には物理的に限界があることである．回線が一ケ所でもブロッキングを起こすとそのプログラムの実行は不可能となってしまう．そこで以下のような事柄も考慮する必要がある．

(3)プログラムモジュールの重複割付けに際して，プログラムモジュール間のデータ依存関係を抽出し，回線を接続する処理要素数ができるだけ少なくなるようにグループ化を行ない割り付ける．

以上の3点を基本とした具体的な論理的処理要素割り付けの方法を以下に述べる．

本システムではその適用範囲をある一定のデータ依存関係を持った並列性の高いアプリケーションに限るので，プログラムモジュールは比較的均質なものが多

数存在すると仮定する．

割付け方法は次に示すように2段階に分ける．

(1)並列処理アルゴリズムの効率を損なわない最小数の処理要素への割付け

(2)実際の処理要素数への対応

この割付け方式はプログラム実行開始時に全てのデータがそろっている場合に関するものである．処理実行中に逐次的にデータが与えられるような場合は動作がパイプライン的になり，データ入力速度とパイプラインのステップを合わせておけば全体が並列に動作するので割付けは単純になる．ここではこのような場合については考慮しないことにする．

次にこれらについて詳しく述べる．

(1)最小数の処理要素への割付け

この段階では1つの処理要素に重複した割付けを行なっても並列処理アルゴリズムの効率を損なう可能性のないプログラムモジュールだけを対象として重複割付けを行なう．

プログラム実行開始時に全てのデータが与えられている場合，重複割付けが可能なプログラムモジュールのグループとして次の3つのタイプが考えられる．

タイプ1：

図7-1に示すように処理が複数のプログラムモジュールに確率的に分散し，同時に動作するプログラムモジュールが高々1つであるようなグループ．

タイプ2：

図7-2に示すようにそれぞれの処理が時間的にオーバラップすることがあり得ないプログラムモジュールのグループ．

タイプ3：

図7-3に示すようにクリティカルパスに属さないプログラムモジュールのグ

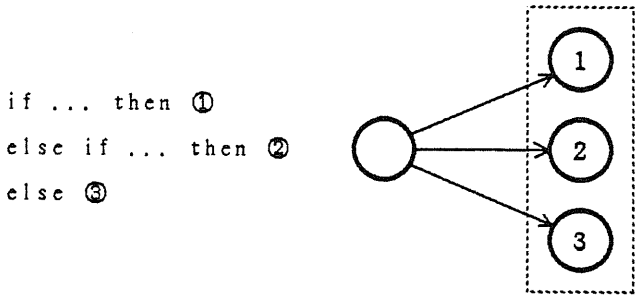


図7-1．タイプ1の例

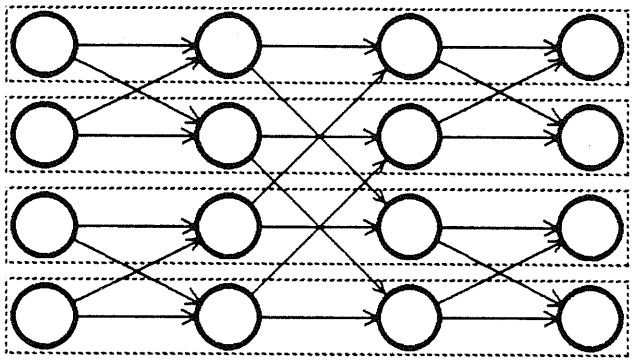


図7-2．タイプ2の例

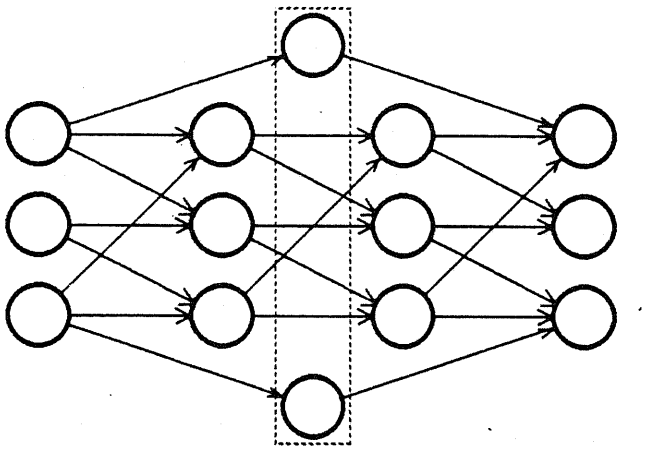


図7-3．タイプ3の例

ループ．

処理開始時にデータが全て与えられている場合は以上のようなグループに対して重複割付けを行なったとしても並列処理アルゴリズムの効率を損なうことはない．

本システムでの割付けにおいては各処理要素の回線数に上限があり，全てのタイプに共通して処理要素の結合数がある上限値を越える場合にはその処理要素へのプログラムモジュールの重複割付けはそれ以上行なえない．しかし，これはまず上限を設けずに割付けを行ない，その後で分割を行なえばよいので，ここでは上限については考えないことにする．

以下に各タイプ別の重複割付け方法を述べる．

タイプ１：

このタイプでは無条件に１つのグループを１つの処理要素に割り付けても良い．

タイプ２：

並列になっているプログラムモジュールのうちから一つずつを選択してグループとしてまとめて同一の処理要素に割り付けることができるが，このタイプの場合，重複割付けによって結合数が増加するので同一の処理要素に割り付けるプログラムモジュールの選択を最適化する必要がある．そこで次のような方法が考えられる．

(1)処理が最初に行なわれるプログラムモジュールに区別のため形式的に番号を付け（図７－４(a)），そのうちから適当な１つを選ぶ（図７－４(a)①）．

(2)処理の流れに沿って，(1)で選ばれたプログラムモジュールと結合を持つ次のステップのプログラムモジュール１つを選択して同じ番号を付ける．これを最後のステップまで行なう（図７－４(b)(1)～(3)）．

(3)番号の確定しているプログラムモジュール間に結合が存在する場合（図７－５(b)A），それと同じ結合関係を持つ未確定のプログラムモジュールを１つ

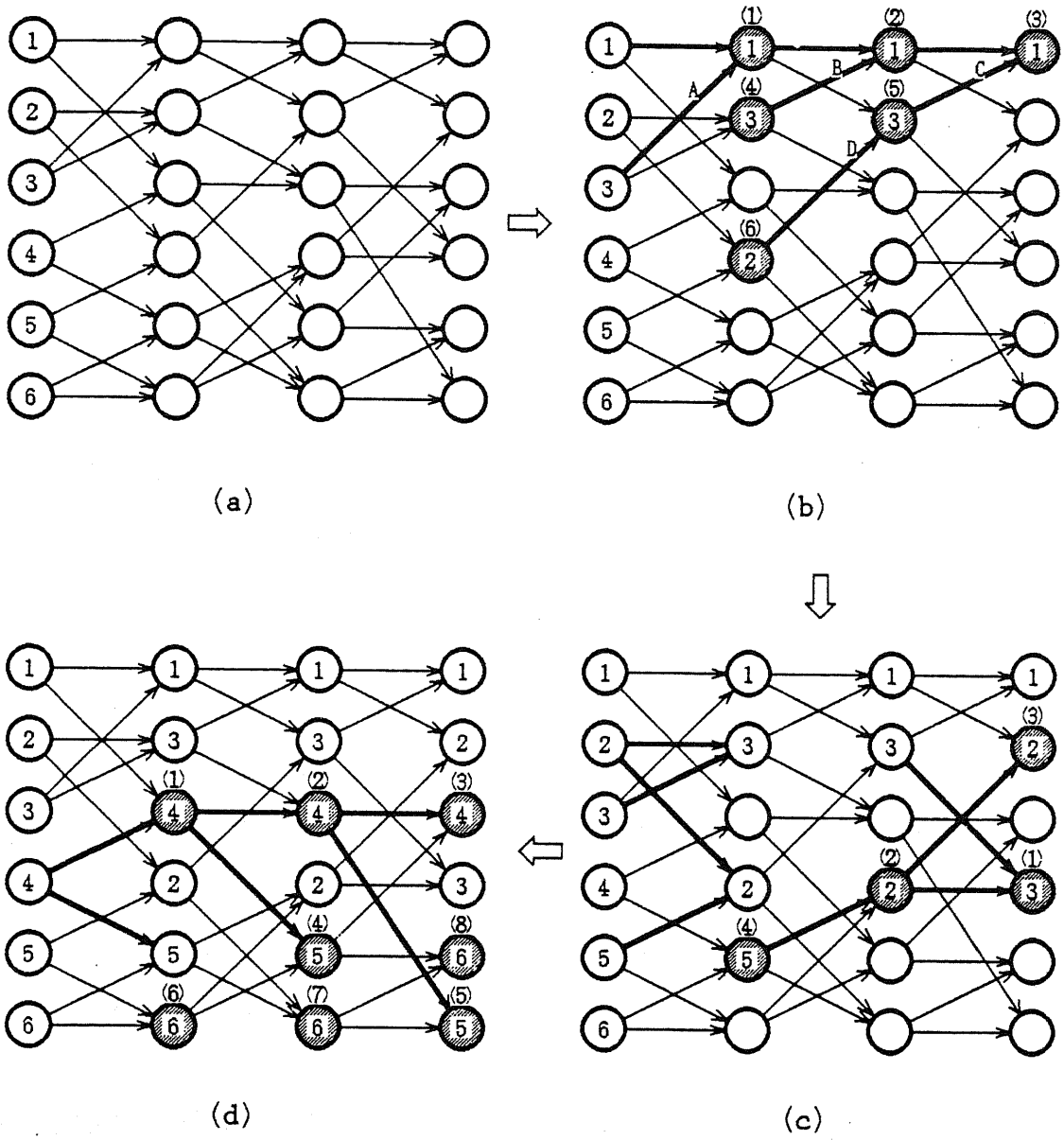


図7－4．タイプ2の重複割付けの例

見つけ（図7－4(b)B,C），同一の結合関係となるように番号を付ける（図7－5(b)⑷,⑸）．これを対象となる未確定のプログラムモジュールが無くなるまで繰り返す（図7－4(c)）．

⑷最初のステップのプログラムモジュールのうちで未使用のプログラムモジュールを1つ選び（図7－4(d)③），すべてのプログラムモジュールの番号が確定するまで⑵，⑶を繰り返す（図7－4⑴～⑻）．

⑸同一の番号のプログラムモジュールを同一の処理要素に割り付ける．

タイプ3：

このタイプでは全体の処理時間がクリティカルパスの実行時間より短くなることがないことを利用し，他のパス上のプログラムモジュールをいくつか組み合わせて処理要素に割り付けることができる．組み合わせるプログラムモジュールとしては同一のプログラムモジュールやグループと結合関係にあるプログラムモジュールを優先的に選択する．

各タイプ別の重複割付けについては以上のような方法により行なうことができる．しかし，プログラムモジュール間の結合関係が複雑な場合，1つのプログラムモジュールが複数のタイプのグループに属することがある．このような場合はタイプ1，2，3の順で重複割り付けを行ない，各タイプの割付け方法においてプログラムモジュールの代わりにプログラムモジュールのグループに対して割付けを行なえばよい．

⑵実際の処理要素数への対応

以上の方法による割付けによりできたグループ数が，実際にシステム内に存在する処理要素数より小さい場合は，処理の効率上からは1グループを1処理要素に割り付けて良い．しかし分割することによって1処理要素当りの結合数を減らすことができるならば分割を行なったほうが後の回線設定が容易になると考えられる．

一方，グループ数がシステムの処理要素数より大きい場合はできるだけ効率の低下を避けながらグループを組み合わせて割り付ける必要がある．組み合わせる

グループの基準として次のようなものがある．

・相互に結合を持つグループ
・同一の結合先を持つグループ

これらのグループのうちから処理量の小さいものを組み合わせて１処理要素に割り付ける．

7.2.物理的処理要素割付け

物理的処理要素割付けの目的は論理的処理要素割付けの終了した処理要素間の結合関係から，総回線長を最短とし回線設定の段階でブロッキングが起こりにくいような最適に近い物理的位置を決定することである．

これらの問題には，ＬＳＩの設計におけるセルの配置や配線等と共通の要素があるため，ＬＳＩの自動設計技術[Kambe 1982]が応用できる可能性がある．本システムの物理的割付け問題とＬＳＩの配置問題の違いは，

①本システムでは論理的処理要素の大きさは全て等しいが，ＬＳＩのセルの大きさはさまざまである．

②本システムでは回線接続は結合路の占有となるが，ＬＳＩの配線は空間的な面積の占有となる．また，ＬＳＩでは配線の交差のためには層間の接続が必要になる場合がありコストが高くつくが，本システムでは交差の影響は全く無い．

という点である．①の理由から，本システムの割付けは論理的な結合関係のみを考慮すればよく，ＬＳＩのセルの配置問題より解決は簡単となる．しかし，基本的考え方はそのまま応用できるので，以下ではセル方式のＬＳＩの自動配置方法を参考に物理的処理要素割付け方式を検討する．

物理的処理要素割付けの際に考慮すべきこととして重要な問題が２つある．１つは回線長の問題であって，回線長が長くなるほどデータの転送遅延が大きくなり処理速度に影響を与える．もう１つは回線の分散の問題であって，回線設定の段階でブロッキングを起こさないように回線の局所的混雑を避けて割り付けなければならない．

これらのうち，本システムではブロッキングが起こるとプログラムの実行ができないため回線の分散の問題の方が重要となるが，この２つは互いに密接な関係にあり，回線長を短くすれば設定に使用される結合路数も減少しブロッキングを起こす確率も小さくなると考えられる．

そこで，物理的処理要素割付けの基本的方針として図７－５に示すような手順を踏む．まず回線の分散を考慮した初期設定を行なう．次に回線長が短くなるこ

とを目的として初期設定に対する改善を加える．

以下に物理的処理要素割付けのアルゴリズムについて述べる．図7－6に示すように基本的には論理的処理要素の部分的な入れ替えの繰り返しによる改善法である．まず，何らかの方法で論理的処理要素を物理的処理要素に仮に割り付け

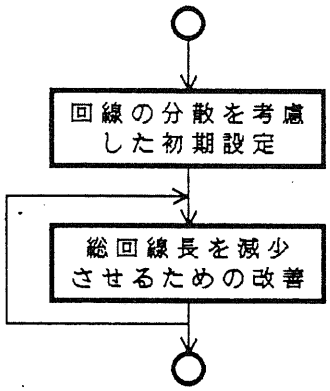


図7－5．物理的処理要素割付けの基本方針

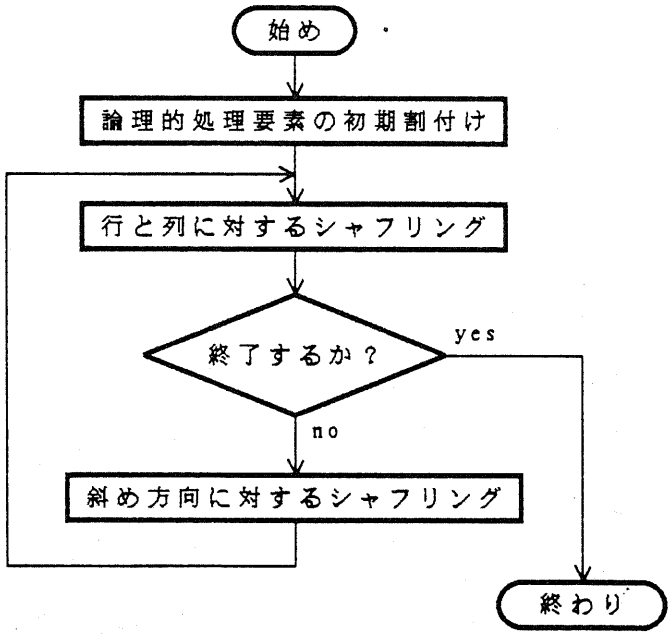


図7－6．物理的処理要素割付けのアルゴリズム

る．次に行と列に対するシャフリングを繰り返す．シャフリングの方法については後述する．ここで適当な条件を用いて終了するかどうかを判定する．続ける場合は，斜め方向に対するシャフリングを行なってから再び行と列に対するシャフリングを行なう．

以下に，初期割付け，行および列に対するシャフリング，終了条件，および斜め方向に対するシャフリングについて述べる．

(1)初期割付け

初期割付けの方法は，シャフリングによる改善の速さや最終的な割付けの良さに影響を与えると考えられる．従って，データ依存関係の局所性をある程度考慮して行なったほうが良い．しかし，実際にはシャフリングによる収束は比較的速いので初期割付けは簡単な方法でもあまり問題ない．最も単純には論理的処理要素の番号順に割り付ける方法が考えられるが，結合関係が一見ランダムに見えるようなデータ依存関係ではこれでも充分な場合が多い．そこで7.4節に示す割付けの例では，初期設定はこの方法によっている．

(2)行および列に対するシャフリング

シャフリングは図7-7に示すように隣り合う処理要素をある評価関数を用いて順に入れ替えるものである．この操作により徐々に回線長を減少させてゆくので各処理要素は1つずつ行または列を移動してゆく．

評価関数は対象とする2つの処理要素を入れ替えるかどうかの判断基準となり，これによりシャフリングの性能が大きく変化する．評価関数としては処理要素を入れ替えることによりどの程度回線長を短縮できるかを表現する必要があり，最も簡単でかつ効率の良いものとして，次のような値が考えられる．

$$E_{AB}=(C_{AS}-C_{AR})-(C_{BS}-C_{BR}) \qquad (7-1)$$

ただし，C_{AR}は図7-8に示すように処理要素Aから領域Rの処理要素に対する結合数を表わす．このE_{AB}はほぼAとBを入れ替えたときに行αと行βの間で減少する結合路数を示しており，

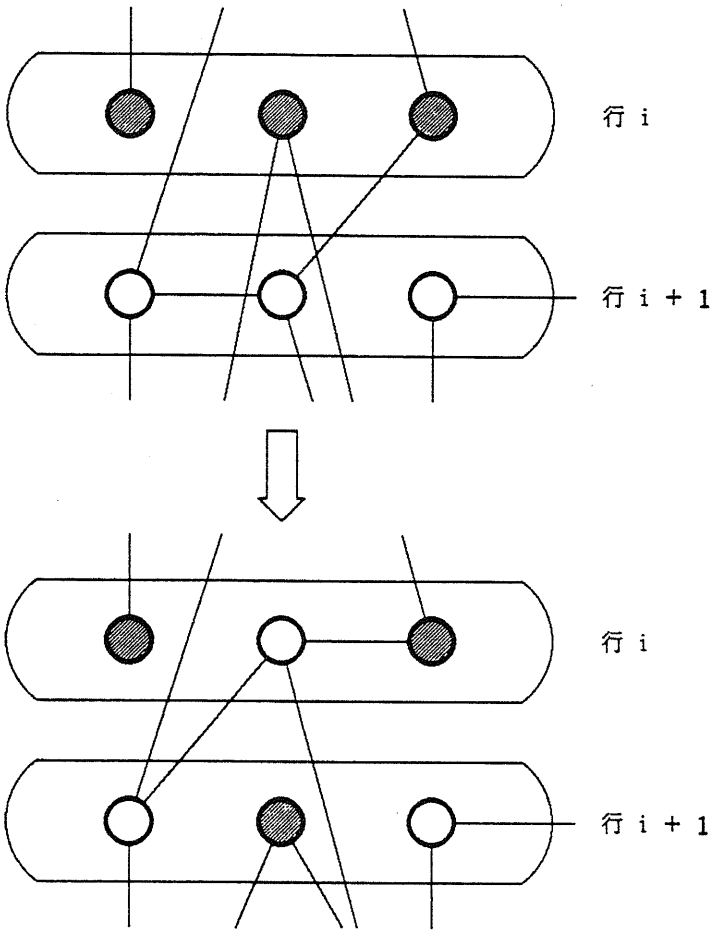


図7-7．行に対するシャフリング

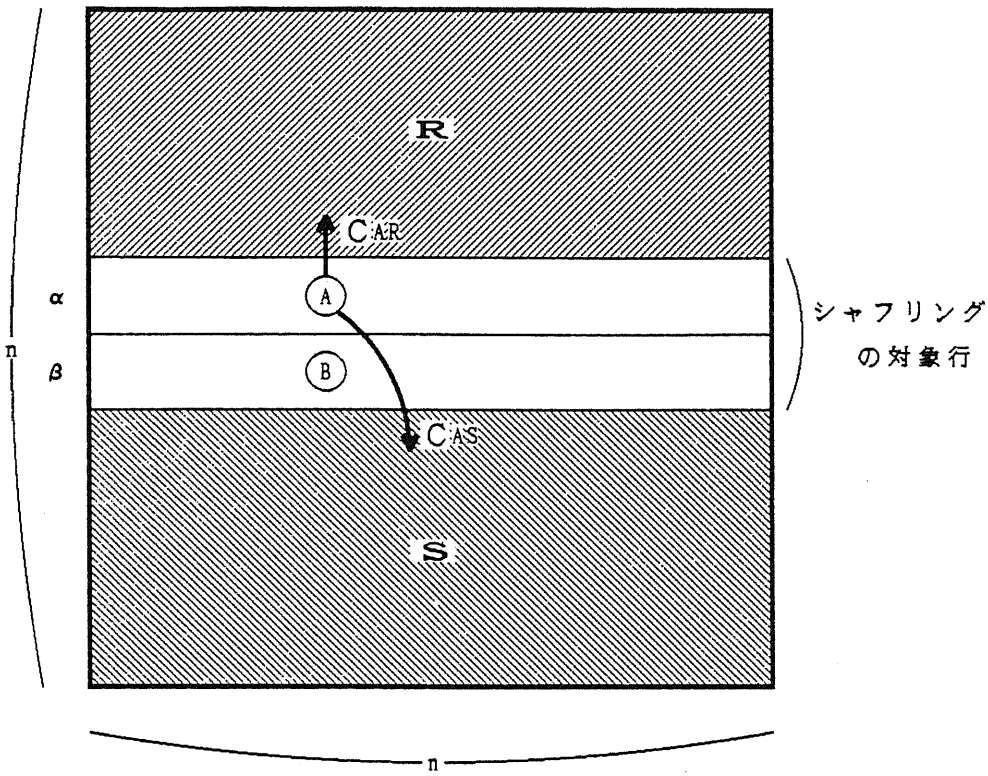


図7-8．シャフリングの評価関数

$$E_{AB} > 0 \qquad (7-2)$$

の条件が成り立つときは入れ替えを行なう．

プロセッサアレイをn×nとすると，この入れ替えを第（0，1）行間から第（n－2，n－1）行間までのすべての処理要素に対して順に行なってゆくことにより，1回分の行に対するシャフリングが完了する．任意の処理要素が任意の行に移り得るためにこれを（n－1）回繰り返す．

列に対しても同様にシャフリングを（n－1）回行なう．

③終了条件

最適な割付けであるかどうかを判定するのは不可能であるので，現実的にはループ1回分のシャフリングによりどの程度の改善が得られたかにより判断する．改善の尺度も厳密には難しいが，簡単には総回線長の減少率を用いることが

できる．減少率が一定の値以下，例えば５％以下になったら終了とする．これにより得られた割付けに対して次節にのべる回線設定を行ない，接続不能な回線が生じた場合はさらにシャフリングを繰り返す．

⑷斜め方向に対するシャフリング

行および列に対するシャフリングだけでは，結合パターンによっては改善できない場合があるので，このパターンを崩す目的で斜め方向に対するシャフリングを行なう．図７－９に示すように，格子方向に対して45度および135度をなす処理要素の組を対象として，行および列に対するシャフリングと同様の方法で行なう．これは結合パターンに変化を与えることが目的であるので，全ての処理要素に対して１回だけ行なえばよい．

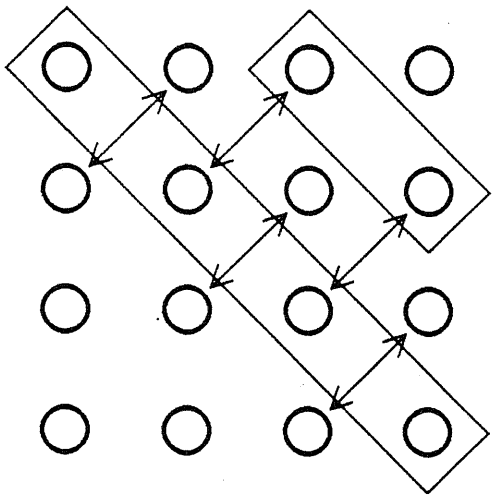


図７－９．斜め方向に対するシャフリング（45度方向）

7.3. 回線設定

本システムでは，多数の処理要素を2次元格子状に配置し，処理要素間の通信は空間分割回線接続により行なう．このため，回線の経路設定が重要となる．ある問題に関して並列処理アルゴリズムが与えられた場合，これを本システム上で実現できるかどうかは主に回線設定ができるか否かにより決まる．しかし，逆に回線設定ができさえすれば結合路の使用状態が空間的に偏っていてもかまわないため，最適な回線設定を求めることは必ずしも必要ない．本節では回線設定を行なうための1つのアルゴリズムを示す．この評価は前節の物理的処理要素割付けと合わせて次節で行なう．

並列処理アルゴリズムはプログラムモジュールに分解され，すでに物理的処理要素に割り付けられているものと仮定し，これらの間の回線設定のアルゴリズムについて検討する．

結合路のトポロジーは処理要素の実現方法により変わり得るものであるため，任意の形態における回線設定を同一のアルゴリズムで行なうことは困難である．そこで，第1段階として結合路を最も基本的なトポロジーである4近傍接続とした構成に対して回線接続を行ない，第2段階としてこれを実際の結合路のトポロジー上にマッピングする．

このうち，第2段階は結合路のトポロジーが決定しないとアルゴリズムも定まらないので本論文では省略する．しかし，本システムのような構造では結合路は比較的狭い範囲に限られるため，4近傍接続からのマッピングは比較的容易であると考えられる．

以下では，第1段階の4近傍接続のトポロジーに対する回線設定アルゴリズムについて述べる．

回線設定アルゴリズムとして考慮すべきことには次のような事柄がある．

①回線の両端の処理要素の位置により融通性が異なる．処理要素の格子上の座標を(m_1, n_1)，(m_2, n_2)とおくと，この2つの処理要素間の回線rが取り得る最短経路の総数ℓ_rは

$$m = |m_1 - m_2|,\ n = |n_1 - n_2| \qquad (7-3)$$

とおいて，

$$\ell_r = {}_{m+n}C_m \qquad (7-4)$$

となる．このℓ_rの値が大きいほど回線設定に対する融通性が高いと言える．処理要素間の距離は(m+n)であるから，この融通性は回線長とは直接対応しないが，一般的には回線長が長い方が融通性が高い．回線設定に当ってはこの融通性を考慮する必要がある．

②回線の分布には空間的な分散がある．一般に回線がランダムに近いとアレイの周辺部よりは中央部に回線が集中しやすいので，回線はできるだけ中央に近い部分を避けて設定した方が良い．しかし，場合によっては部分的に集中することもあるので，このような空間的な分布を考慮して回線設定を行なう必要がある．

③並列処理アルゴリズムによっては結合路に縦方向と横方向の偏りが存在する場合がある．例えばシステム全体を並列パイプラインのようにして動作させる場合，データはアレイの一辺から入力し反対の辺から出力するので，回線もこの方向のものが多くなる．従って，空間的な分布を考える場合，回線が多い方向の分布を中心に考慮して経路設定する必要がある．

④回線接続ができるかどうかは各結合路を通過する回線数の最大値で決まる．アレイをマクロ的にとらえると総回線長や各結合路を通る回線の平均や分散が特徴となるが，実際の設定ではたとえ平均や分散が小さくとも1ケ所でも回線数が物理的に配線されている結合路数より大きければ接続できないので，この点が重要になる．

①については設定を行なう回線の順番を最適化することにより対処する．②と③については，各結合路を通る回線数の期待値を求めておくことにより，全体的な分布を考慮する．④については経路選択する際に最も混雑しそうな場所を避け

ることにより対応する．

これらを考慮した回線設定アルゴリズムを図7-10に示す．

各結合路を通る回線数の期待値は以下のようにして求める．

処理要素の位置を格子上の座標で表わすことにし，図7-11に示すように処理要素(m_1, n_1)と(m_2, n_2)を接続する回線ℓ_rが結合路cを通過する期待地$e_{r,c}$を

$$e_{r,c} = \frac{{}_{i+j}C_i \cdot {}_{m+n-i-j-1}C_{m-i-1}}{{}_{m+n}C_m} \qquad (7-5)$$

と定義する．ただし，

$$m = |m1-m2|,\ n = |n1-n2|,\ 0 \leq i \leq m,\ 0 \leq j \leq n \qquad (7-6)$$

これは回線rが両端の処理要素間の全ての最短経路を等確率でとると仮定した場合に結合路cを通過する確率に等しい．この定義に従うと全ての回線が結合路cを通過する期待値E_cは

$$E_c = \sum_r e_{r,c} \qquad (7-7)$$

で与えられる．これは縦の結合路に対しても同様に定義できる．E_cはcに対して求められるものであり，空間的な分布をもつ．

次に，回線の経路選択の方法を図7-12に示す．まず全体の期待値から，これから設定しようとする回線rの期待値を差し引く．次に図7-11に示した最短経路が通過する範囲の結合路のうち，E_cが最大の結合路を除外して最短経路設定ができるかどうか調べる．設定ができる場合はこれを設定できなくなるまで繰り返す．設定できなくなった時に，最も最近除外した結合路は実は除外できなかったことがわかり，この結合路を経路として登録する．回線の残りの部分はより短い2つの回線に分割されることになる．これを全ての経路が確定するまで繰り返す．この方法により，通過する回線数が大きくなると予想される結合路をで

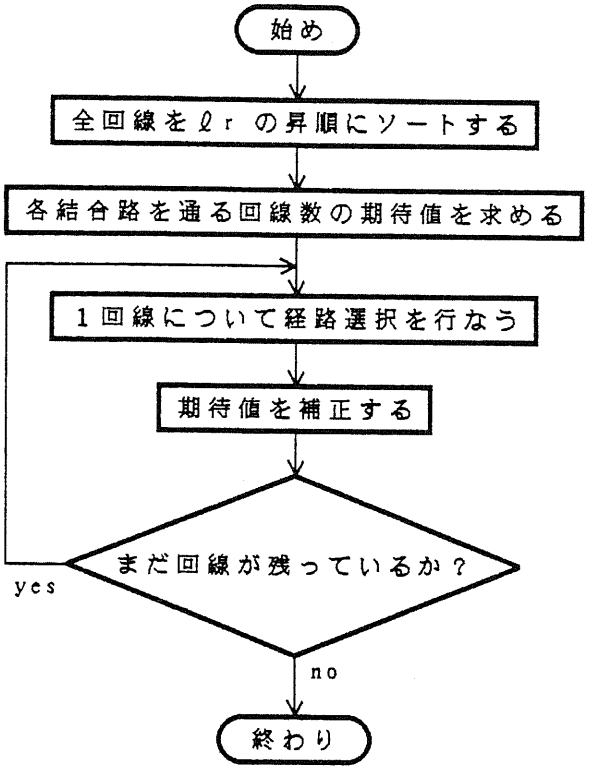


図7-10．回線設定アルゴリズム

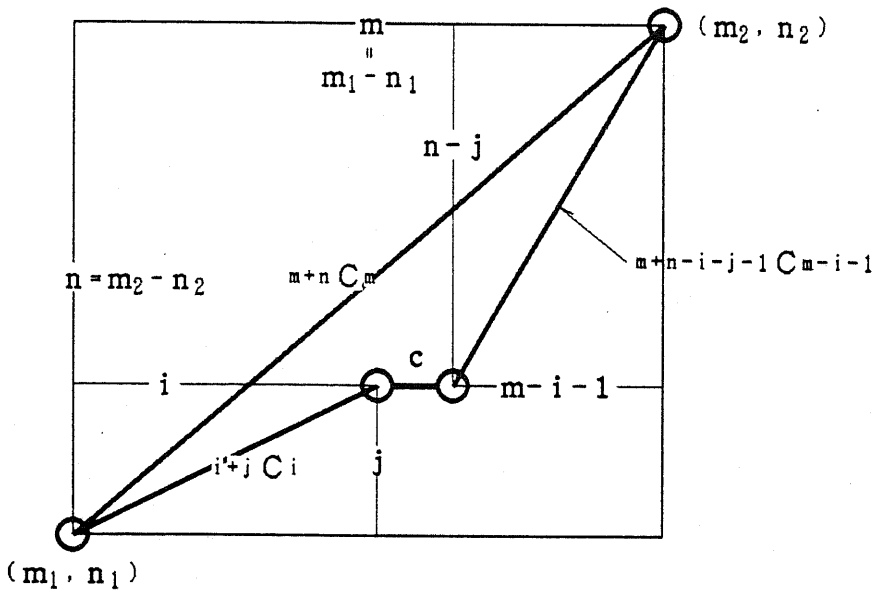


図7-11．回線の最短経路数

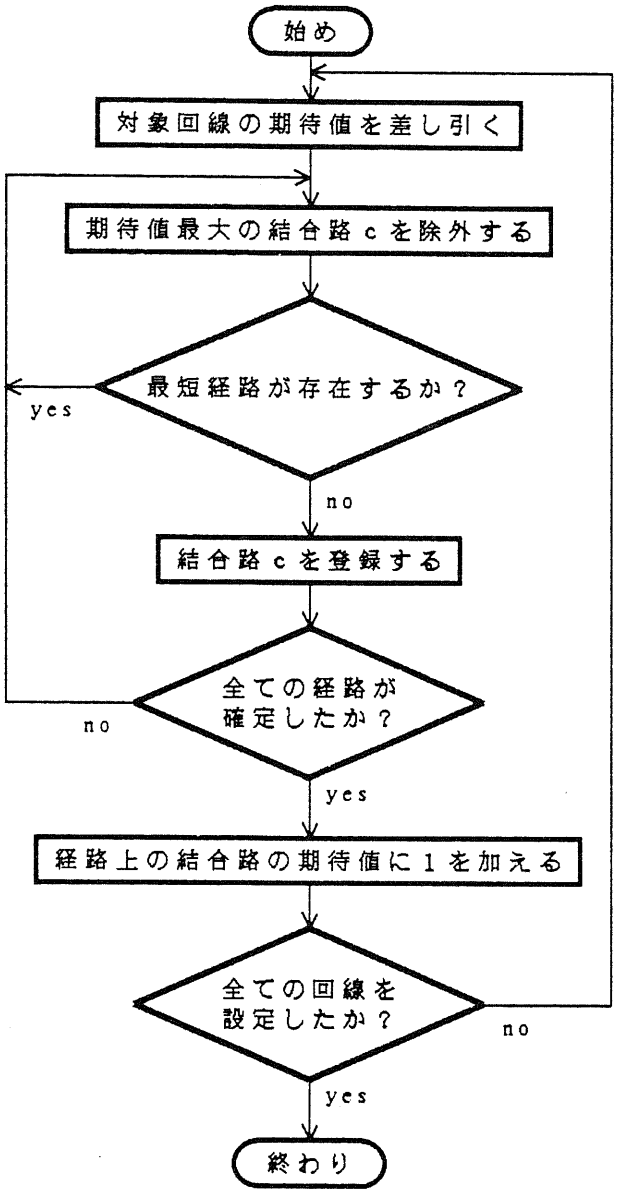


図7－12．回線の経路選択アルゴリズム

きるだけ避けるような経路選択が実現できる．

回線の経路が確定したら，この経路に相当する各結合路の期待値に1を加える．

これを全ての回線について繰り返すことにより，かなり良好な回線設定ができる．

7.4.資源割付けの評価

高級言語プログラムに対する資源割付けは，これまで述べてきた3つの段階を経て完了する．各段階は最終的な割付け結果に対して独立ではなく相互に係わり合いを持って影響する．従って，評価は総合的な結果をふまえて各段階について行なわなければならない．

資源割付けによる並列処理アルゴリズムのプロセッサアレイに対するマッピングの評価の指標として次のことが挙げられる．

(1)システムがどの程度の処理能力を発揮できるか
(2)システムがどの程度の適応性をもつことができるか

(1)は主に論理的処理要素割付けに関するものである．しかし現在は比較的規則的なアプリケーションについてしか検討しておらず，論理的処理要素割付けは並列処理アルゴリズムに含まれた形となっているため，この評価は本論文には述べない．

(2)は主に物理的処理要素割付けと回線設定に関するものである．適応性は，いかに多数で広範囲にわたるデータ依存関係を限られた結合路を用いてマッピングできるかにかかっている．

本節では，これらのことを考慮に入れ，いくつかの結合関係に対して実際に資源割付けを行なうことにより各アルゴリズムを評価する．

7.4.1.結合関係のモデル

評価に用いた結合関係のモデルは以下の7つである．

①2分木構造：図7-13

2分木構造はソーティングやサーチングなどのさまざまな応用が考えられる結合関係であり，logN段，(N－1)ノードの2分木構造において，各処理要素間の回線は上位から下位へと，下位から上位への2本ずつであるとする．

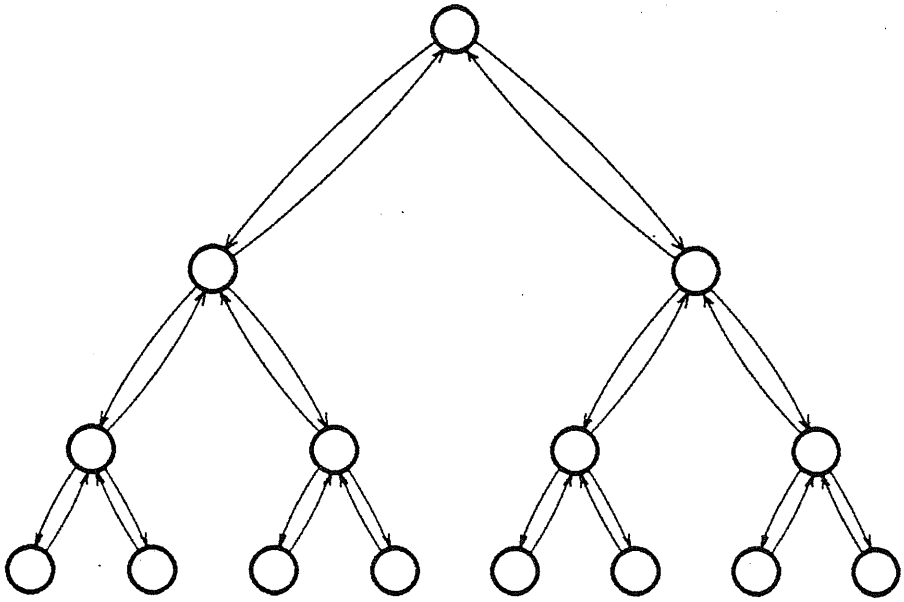


図7-13．2分木構造

②シャフルエクスチェンジ：図7-14

シャフルエクスチェンジはさまざまな並列処理アルゴリズムに対して有効な結合関係であり，番号の2進数表現が$\langle b_n b_{n-1} \cdots b_1 b_0 \rangle$ (b_i=0,1) である処理要素を$P\langle b_n b_{n-1} \cdots b_1 b_0 \rangle$と記すことにすると，

シャフル：　$P\langle b_n b_{n-1} \cdots b_1 b_0 \rangle \rightarrow P\langle b_0 b_n b_{n-1} \cdots b_1 \rangle$　（7－8）

エクスチェンジ：$P\langle b_n b_{n-1} \cdots b_1 b_0 \rangle \rightarrow P\langle b_n b_{n-1} \cdots b_1 b_0^* \rangle$　（7－9）

（b_0^*はb_0の反転）

で表わされる処理要素間の結合である．

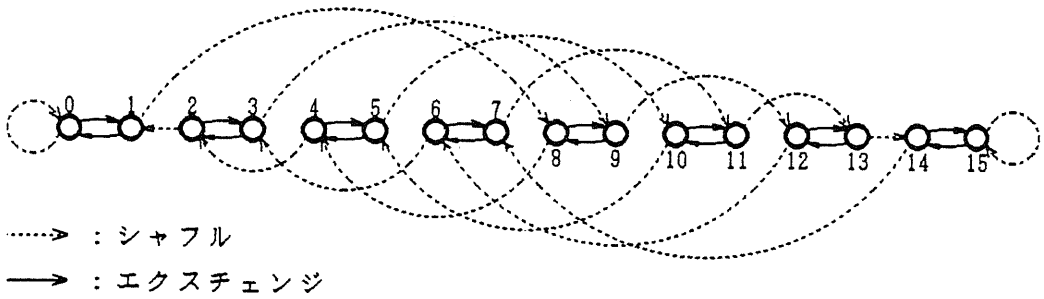


図7-14．シャフルエクスチェンジ

③ＦＦＴ：図７－１４＋図７－１５

ＦＦＴ用の結合関係は②のシャフルエクスチェンジと図７－１５に示したビット逆順の結合を組合わせたものである．ビット逆順は

$$P\langle b_n b_{n-1} \cdots b_1 b_0 \rangle \rightarrow P\langle b_0 b_1 \cdots b_{n-1} b_n \rangle \qquad (7-10)$$

で表わされる処理要素間の結合である．

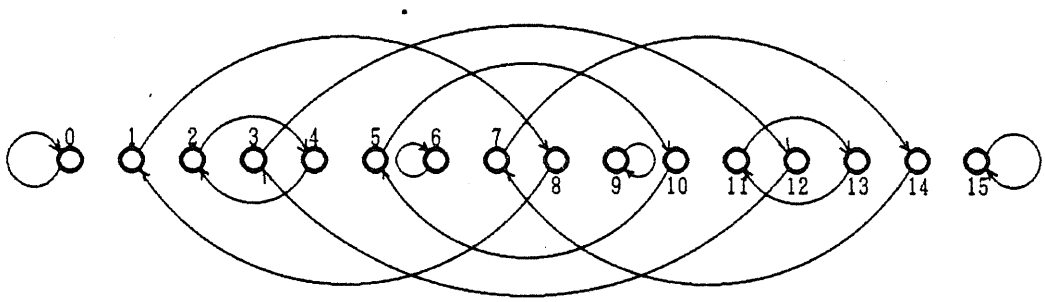


図７－１５．ビット逆順

④ 3次元立方格子：図7-16

3次元立方格子は物理現象の解析に用いられる偏微分方程式などの各種の応用の基本となる結合関係であり，座標(x,y,z)の格子点に対応する論理的処理要素をP(x,y,z)と記すことにすると，

$$P(x,y,z) \rightarrow P(x',y',z') \quad (|x-x'|+|y-y'|+|z-z'|=1) \qquad (7-11)$$

で表わされる処理要素間の結合である．

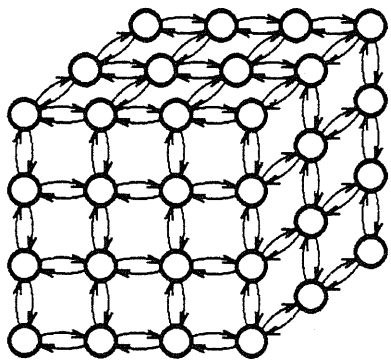


図7-16．3次元立方格子

⑤ランダムⅠ：

全ての回線の一端は回線番号とmodNで等しい番号を持つ論理的処理要素で，もう一端はランダムな処理要素であるような結合関係である．従って，回線数をL＝kNとすると，各論理的処理要素には少なくともk本の回線が接続されることになる．確率的に0から（N－1）の整数値をとる一様乱数をR_Nとし，便宜的に回線と論理的処理要素に0から順に番号を付けて$\ell(i)$，P(i)と記すことにすると，この結合関係は次のように表現できる．

$$\{\ \ell(i):\ P(i \bmod N) \rightarrow P(R_N(i))\ |\ 0 \leq i < L\ \} \qquad (7-12)$$

これは回路シミュレーションのように結合関係が規則的でないようなネットワーク状のアプリケーションを想定したものであり，従来の高並列処理システムでは適応できなかった新しい分野にこのようなものが多いと考えられる．

⑥ランダムⅡ：

全ての回線の両端がランダムな処理要素である結合関係であり，各処理要素に接続する回線数は任意の値を取り得る．R_N'をR_Nと独立な乱数とすると，この結合関係は次のように表現できる．

$$\{\ \ell(i):\ P(R_N(i)) \rightarrow P(R_N'(i))\ |\ 0 \leq i < L\ \} \qquad (7-13)$$

これは仮想的なデータ依存関係であり，特に実際のアプリケーションをモデル化したものではないが，処理要素により接続される回線数のばらつきが大きいため物理的処理要素割付けが難しい形態である．

⑦ランダムⅢ：

処理要素間の全ての結合はランダムであるが，回線数をL＝kNとすると，各論理的処理要素には必ず2k本ずつの回線が接続されているような結合関係である．0から（L－1）の範囲の整数の任意の置換をσ_Lとおくと，この結合関係は次のように表現できる．

$$\{\ell(i): P(i \bmod N)) \rightarrow P(\sigma_L(i) \bmod N) \mid 0 \leq i < L\} \quad (7-14)$$

これも⑥と同様に仮想的なデータ依存関係であり，回線長はランダムであるが処理要素による回線数のばらつきはない．

なお，本評価においては⑤，⑥，⑦ともL＝2Nとしている．

これらの結合関係に対して実際に割付けを行なった結果を用いて，以下に物理的処理要素割付けおよび回線設定について評価を行なう．

7.4.2.資源割付け結果

本節では前節の各種の結合関係に対して物理的処理要素割付けおよび回線設定アルゴリズムを適用した結果を示す．表7-1にはそれぞれについて処理要素数を8×8～64×64とした場合の総回線数，割付け後の平均回線長，割付け後に各隣接処理要素間の結合路を通過する回線数の最大値を示す．

資源割付けにより得られた結合路の使用状態の例を図7-17に示す．図中“#”が処理要素を示し，それらの間の数字が処理要素間の結合路を通過する回線数を示す．これは処理要素数を16×16としてシャフルエクスチェンジを割り付けた結果である．x方向とy方向で回線数に差が現れているのがわかる．また，周辺部よりも中心部に回線が集中している様子もうかがえる．

この結果をすべての結合関係に対して示すことはできないので，これを回線数について整理した結果を図7-18～24に示す．各図には，処理要素数を64×64とした場合の各結合関係に対する資源割付けを行なった結果から，各結合路を何本の回線が通過するかを回線数に対して示している．これらについての検討は次節以下で行なう．

表7-1．各種の結合形態に対する割付け結果
（回線数／平均回線長／最大結合路数）

処理要素数	64	256	1024	4096
2分木構造	124/ 2.0/ 5	508/ 2.2/ 7	2044/ 2.3/ 6	8188/ 2.5/ 8
シャフルエクスチェンジ	126/ 2.8/ 5	510/ 3.2/ 6	2056/ 2.3/ 6	8190/ 7.0/16
FFT	182/ 2.4/ 7	750/ 3.0/ 8	3038/ 4.5/12	12222/ 6.5/20
3次元立方格子	288/ 2.5/10	1280/ 3.0/14	5504/ 3.0/16	23040/ 4.3/24
ランダムⅠ	128/ 3.0/ 6	512/ 4.3/ 8	2048/ 6.3/10	8192/ 9.1/15
ランダムⅡ	128/ 3.0/ 7	512/ 4.5/ 9	2048/ 7.5/14	8192/14.3/27
ランダムⅢ	128/ 3.0/ 6	512/ 4.0/ 7	2048/ 6.6/11	8192/10.1/17

```
# 3 # 2 # 5 # 0 # 5 # 3 # 5 # 3 # 4 # 3 # 5 # 4 # 4 # 2 # 2 #
3   3   3   3   5   4   2   0   3   3   4   3   4   4   4   4
# 3 # 1 # 4 # 4 # 6 # 4 # 5 # 3 # 4 # 3 # 5 # 2 # 5 # 2 # 4 #
0   3   2   1   3   2   3   2   2   2   2   2   1   1   2   2
# 5 # 2 # 4 # 3 # 5 # 4 # 3 # 3 # 5 # 4 # 3 # 1 # 3 # 0 # 4 #
5   4   4   4   3   3   4   2   2   3   5   4   1   2   2   2
# 3 # 2 # 4 # 3 # 5 # 1 # 4 # 3 # 4 # 3 # 4 # 1 # 3 # 2 # 5 #
2   3   4   3   5   5   5   5   5   0   4   3   5   5   5   5
# 4 # 0 # 4 # 2 # 6 # 2 # 6 # 5 # 4 # 4 # 4 # 3 # 5 # 1 # 4 #
6   5   6   5   5   3   3   4   2   4   2   4   3   3   4   3
# 2 # 3 # 5 # 4 # 5 # 3 # 3 # 4 # 5 # 4 # 5 # 4 # 3 # 2 # 2 #
4   4   4   4   4   1   5   5   5   5   5   5   4   4   4   5
# 4 # 1 # 5 # 3 # 5 # 1 # 2 # 4 # 2 # 4 # 4 # 4 # 1 # 0 # 3 #
6   5   6   6   6   5   4   3   3   3   1   5   5   5   5   4
# 5 # 1 # 4 # 1 # 5 # 5 # 4 # 3 # 2 # 4 # 3 # 4 # 0 # 2 # 3 #
3   3   3   5   6   3   3   4   4   5   4   2   3   3   4   3
# 4 # 2 # 4 # 3 # 4 # 0 # 4 # 3 # 3 # 3 # 2 # 4 # 2 # 2 # 3 #
3   3   3   4   5   3   3   5   2   1   3   4   3   5   5   4
# 2 # 3 # 3 # 1 # 5 # 3 # 5 # 3 # 2 # 4 # 2 # 4 # 0 # 2 # 2 #
5   6   5   6   5   5   5   5   5   5   5   6   5   3   5   6
# 4 # 0 # 4 # 3 # 6 # 4 # 4 # 5 # 3 # 5 # 1 # 4 # 5 # 4 # 3 #
3   2   3   3   4   5   5   4   1   3   5   5   4   4   4   5
# 3 # 1 # 3 # 1 # 5 # 4 # 2 # 4 # 3 # 5 # 5 # 4 # 2 # 5 # 1 #
4   4   5   5   4   4   3   2   4   3   5   2   2   1   4   4
# 3 # 1 # 4 # 0 # 5 # 1 # 3 # 3 # 4 # 1 # 5 # 2 # 1 # 5 # 2 #
1   2   2   1   3   4   3   2   3   4   5   5   5   5   5   6
# 3 # 1 # 3 # 3 # 5 # 4 # 3 # 4 # 5 # 3 # 4 # 2 # 3 # 5 # 4 #
0   2   2   1   1   3   2   1   2   4   4   1   2   1   4   2
# 3 # 4 # 2 # 4 # 5 # 4 # 4 # 3 # 5 # 4 # 4 # 4 # 3 # 2 # 3 #
3   1   4   3   4   4   4   4   4   5   4   3   3   4   3   3
# 1 # 4 # 2 # 1 # 5 # 3 # 5 # 3 # 5 # 4 # 4 # 1 # 0 # 2 # 3 #
```

図７－１７．割付け後の結合路使用状況（16×16処理要素，シャフルエクスチェンジ）

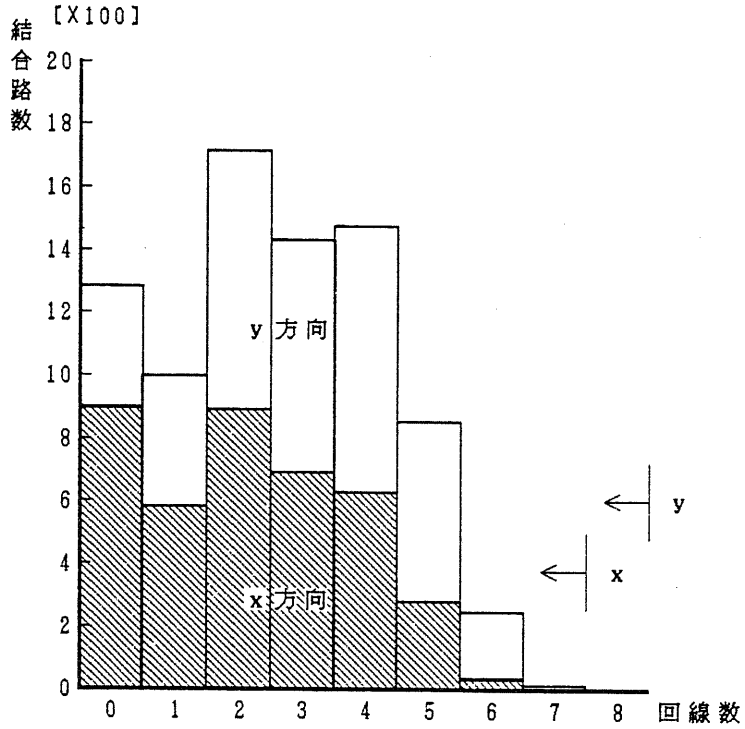


図7-18．結合路を通過する回線数（２分木構造）

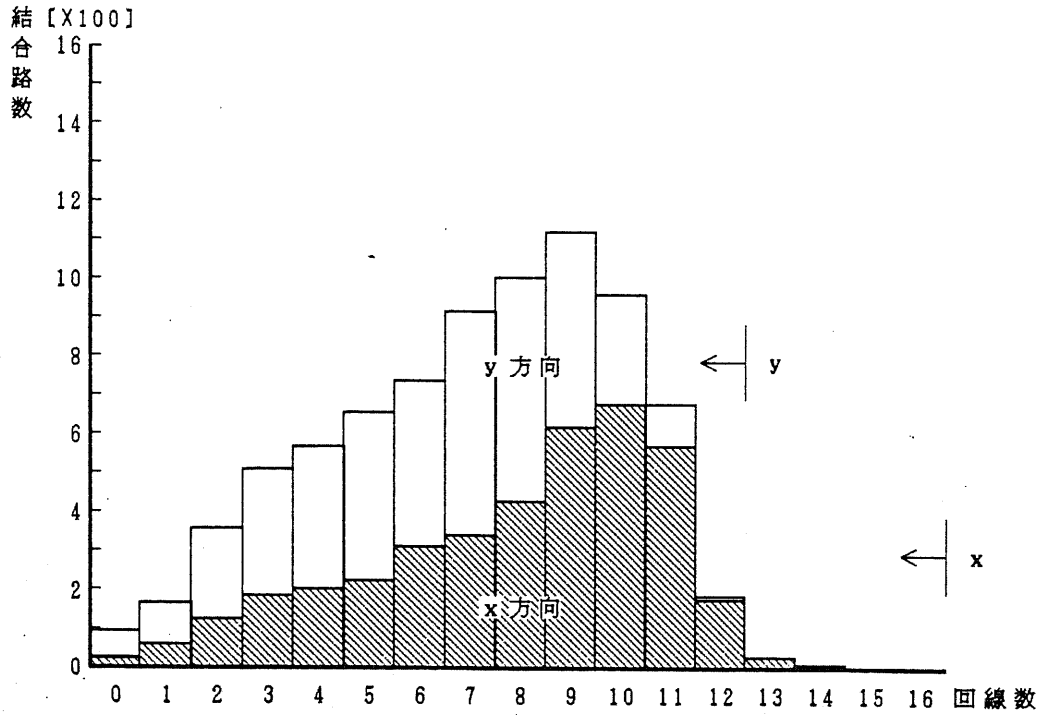


図7-19．結合路を通過する回線数（シャフルエクスチェンジ）

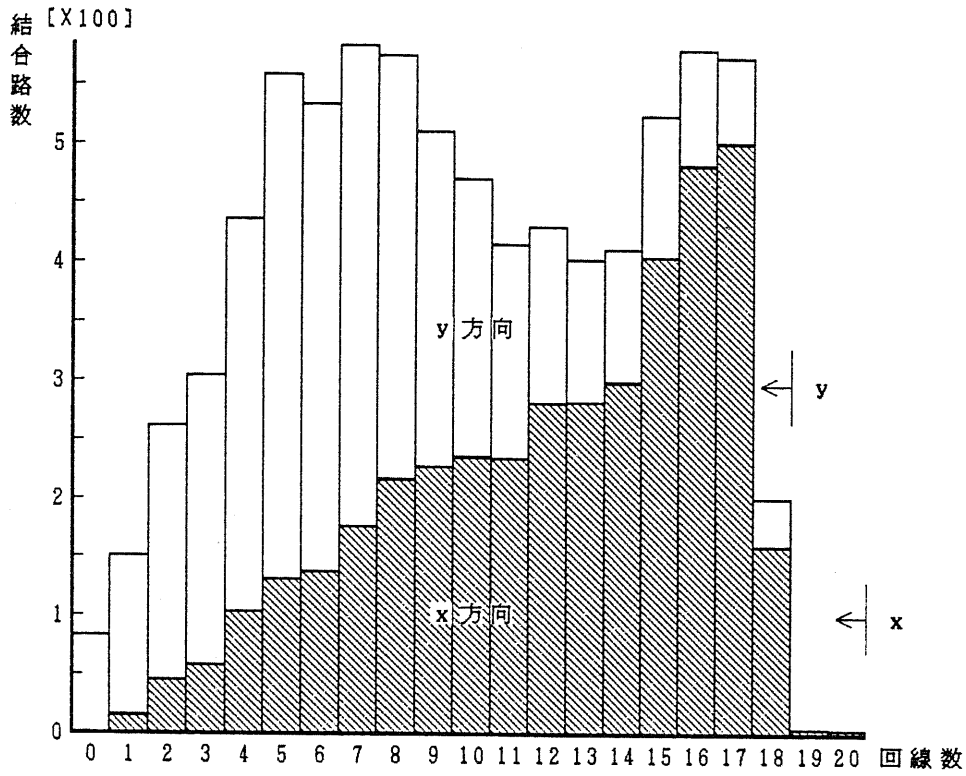


図7-20．結合路を通過する回線数（ＦＦＴ）

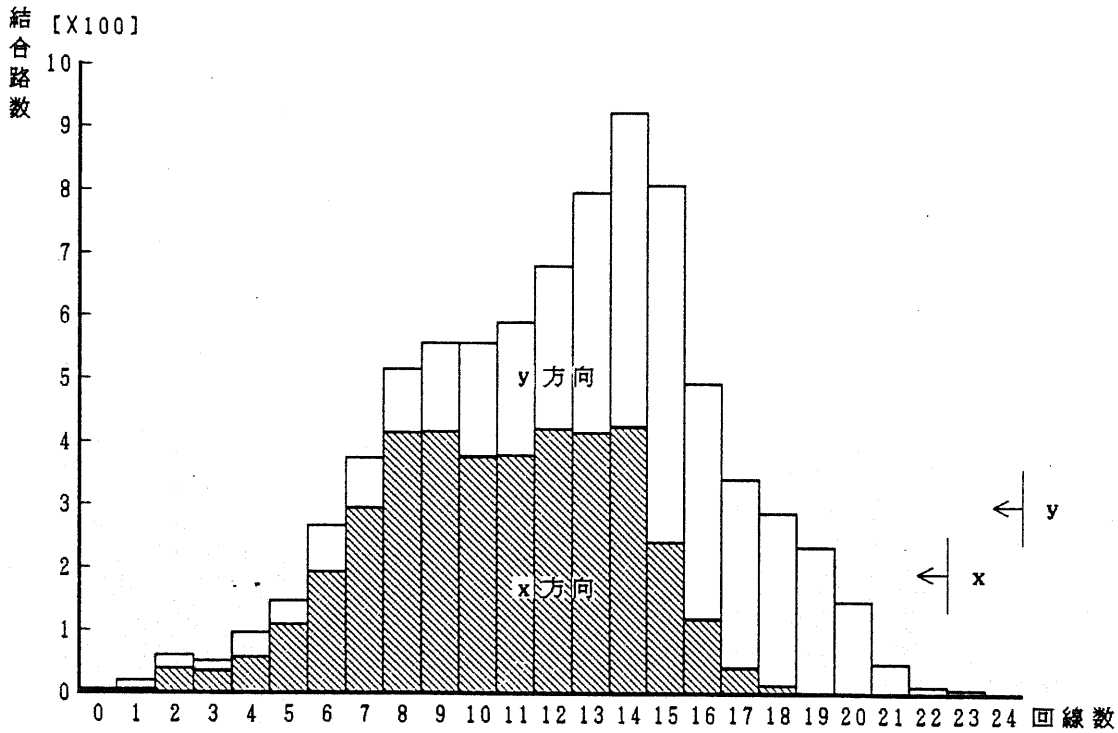


図7-21．結合路を通過する回線数（３次元立方格子）

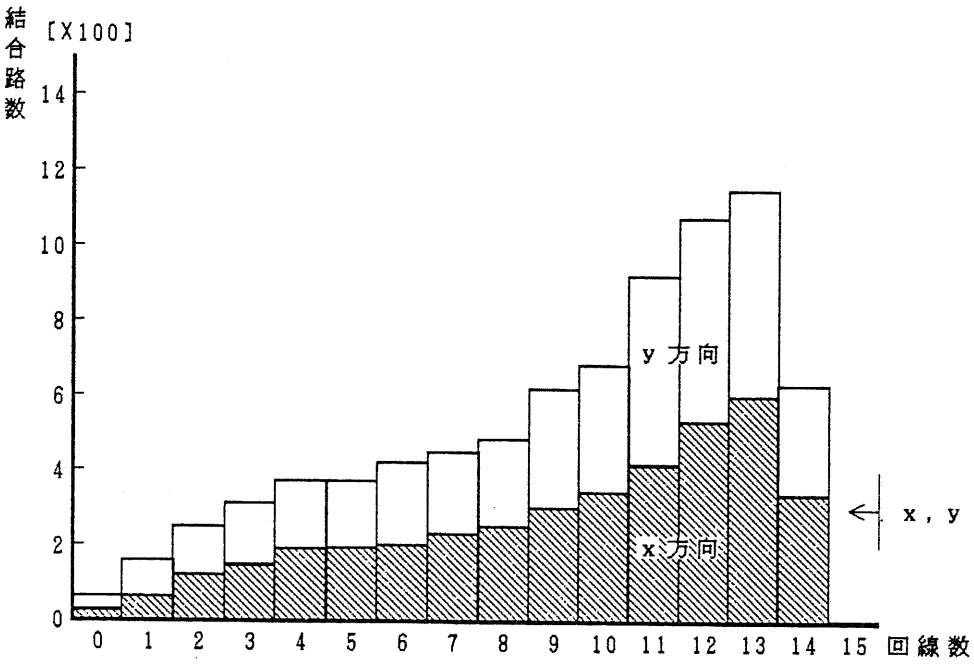


図7-22．結合路を通過する回線数（ランダムⅠ）

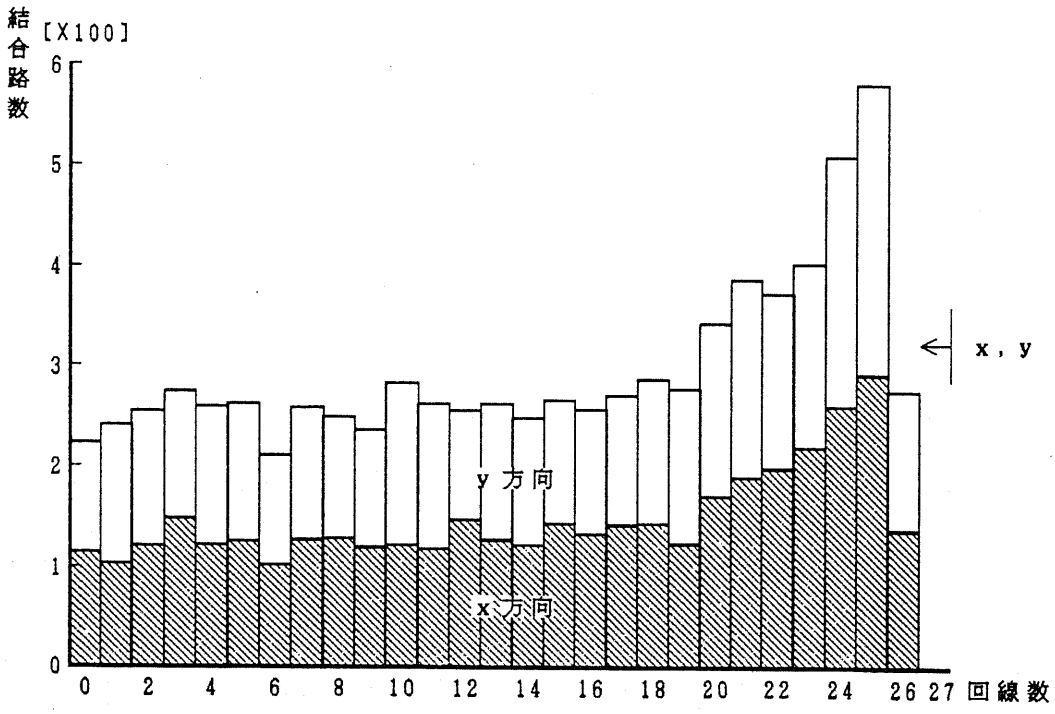


図7-23．結合路を通過する回線数（ランダムⅡ）

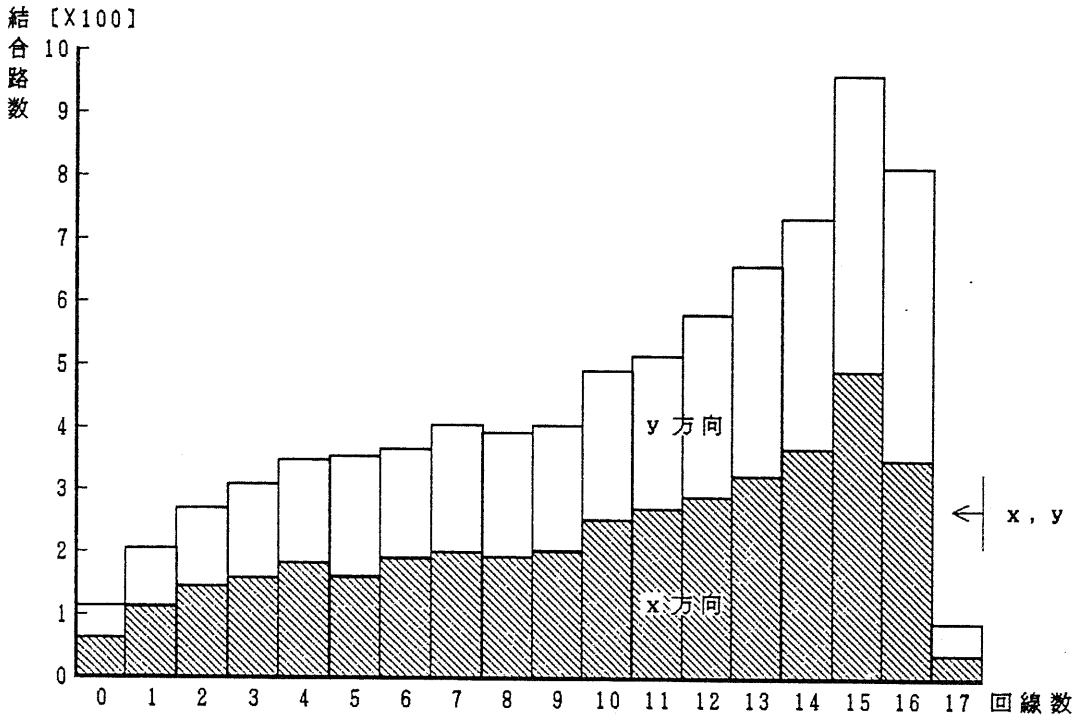


図7-24．結合路を通過する回線数（ランダムⅢ）

7.4.3.物理的処理要素割付けに関する評価

割付け方法の評価には，割付けのための処理量と割付け結果を考慮する必要がある．本節では7.2節に述べた物理的処理要素割付けアルゴリズムについて計算量と割付け結果について評価する．

(1)計算量

計算量に関連するパラメータとして，全処理要素数をN，全回線数をLとして評価を行なう．

初期割付けはここでは単に論理的処理要素番号順としているので計算量の評価の対象とはならない．

行および列に対するシャフリングは，隣り合う全ての処理要素の入れ替えにより1回分となる．2つの処理要素間の入れ替えに要する計算量はNやLによらず一定であり，評価関数の値により入れ替えを行なわない場合もあるので全処理要素に対してはO(N)以下となる．また，評価関数の計算量は，最初に一回だけ全回線に対して処理要素ごとのリストを作るためにO(L)，シャフリングの際に2つの処理要素間で行なう評価関数の計算は処理要素当りの回線数に比例するのでO(L/N)であり，処理要素全体ではO(L)となる．ここで一般にL＞Nであり，また入れ替えの回数はNよりかなり小さくなると考えられ，さらに入れ替えよりは評価関数の計算の方が計算量が大きいので，1回のシャフリングにおける計算量はほぼO(L)となる．行および列のシャフリングは図7-6の1回のループ中で($N^{1/2}-1$)回ずつ行なうので，1ループ分を合計すると$O(LN^{1/2})$以下となる．

終了条件の判定では総回線長を用いるため，この計算量はO(L)となる．

斜め方向に対するシャフリングの計算量は，行および列に対するシャフリングと同様であり，これは1ループ中に1回だけ行なうのでO(L)である．

以上を全て合計すると，1回のループの計算量はほぼ$O(LN^{1/2})$となる．

7.2節に述べたように1回のループによる総回線長の改善率を終了条件に用いた場合，ループの繰り返し回数を理論的に推定することは困難であるので，前述の各結合関係に対して物理的処理要素割付けを実際に行なった様子を図7-25～31に示す．各図には比較が容易となるよう平均回線長を示した．シャフリン

グによっても回線数は変化しないので，総回線長の変化の様子もこれと同じである．

初期状態と最終的な状態との差は各結合関係により異なるが，いずれにおいてもループ３回以内でほぼ収束していることがわかる．この収束の速さは処理要素数Ｎや回線数Ｌには依存しておらず，さまざまな結合関係に対してほぼ一定であると考えられる．

従って，物理的処理要素割付け全体に要する計算処理量は$O(LN^{1/2})$であると推定される．

これを実際にプログラムにより記述し，前述の各種結合関係に対して実行した場合の処理時間の測定値を図７－３２に示す．これはいずれもループを８回実行した時のＣＰＵタイムの測定値である．

使用した計算機は住友電工社製ＵＳＴＡＴＩＯＮ，ＣＰＵは１０ＭＨｚ 68000マイクロプロセッサ，主記憶１ＭＢ，キャシュなし，バスはＶＥＲＳＡバス，ＯＳはUniplus＋（ＵＮＩＸ System Ⅲ相当），使用言語はＣである．

測定結果によると，物理的処理要素割付け処理時間は傾き１の直線上に極めて良く一致していることから，$O(LN^{1/2})$になっていることがわかる．この測定結果から物理的処理要素割付け処理時間T_{PA}は

$$T_{PA}=0.067LN^{1/2}+0.91 \quad [s] \qquad (7-15)$$

と推定できる．プログラムの改良により処理時間の短縮も可能であろうが，実際的には64×64処理要素，10000回線で数十分程度と考えられる．これは，本システムの目的からすると許容できる時間である．

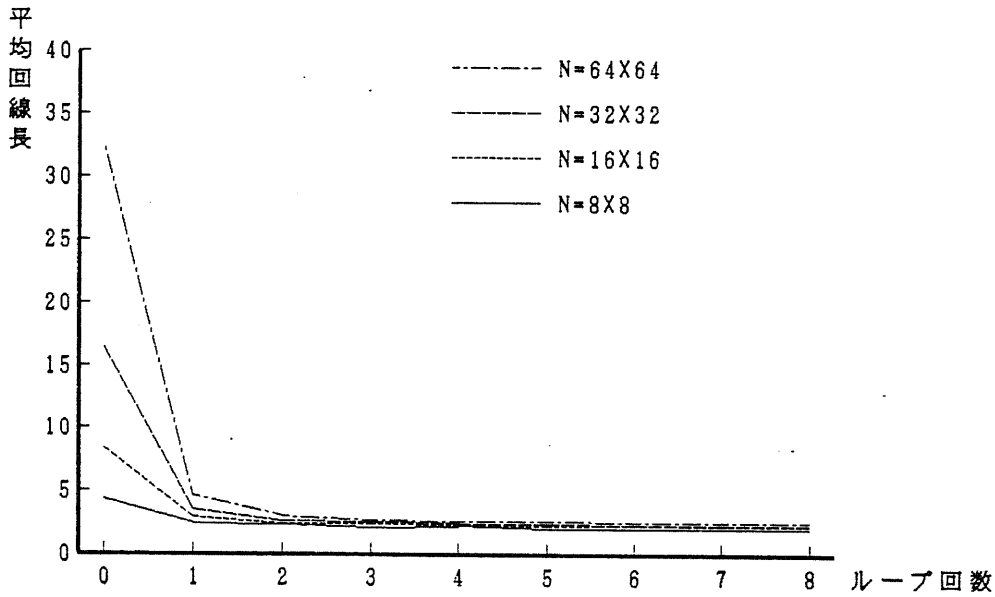


図7-25．シャフリングによる平均回線長の変化（２分木構造）

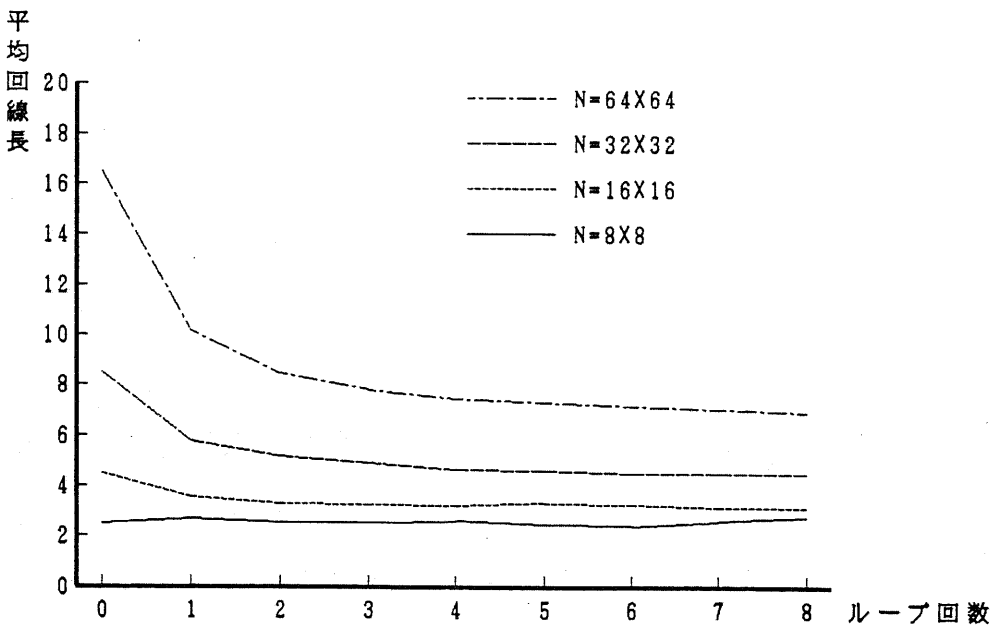


図7-26．シャフリングによる平均回線長の変化（シャフルエクスチェンジ）

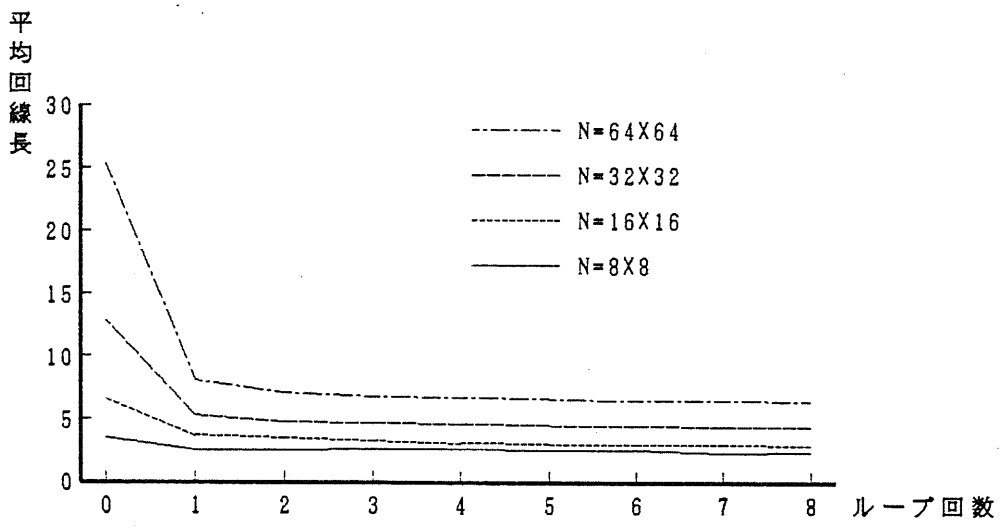


図7-27．シャフリングによる平均回線長の変化（ＦＦＴ）

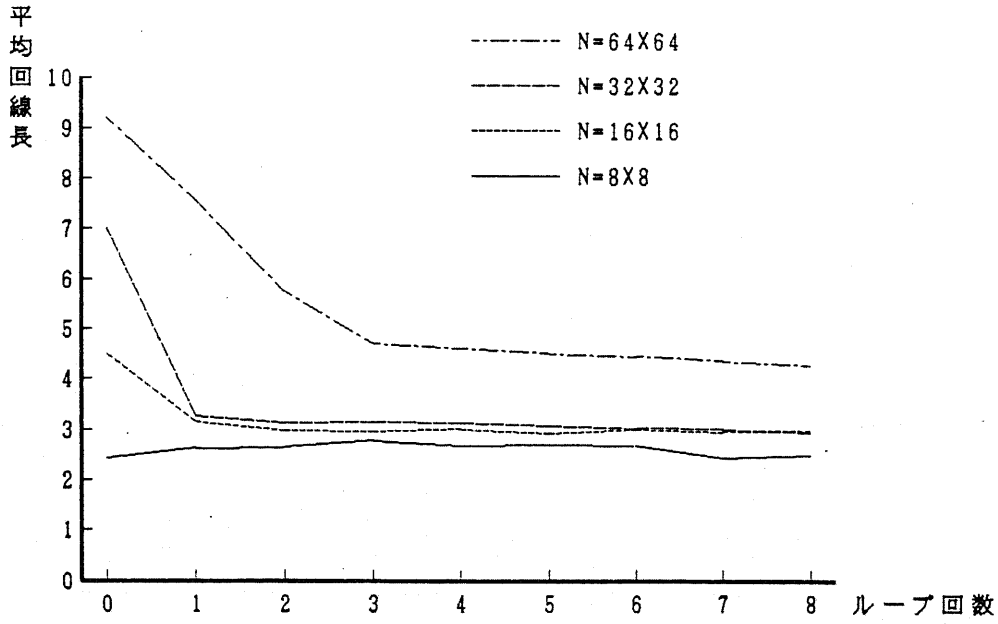


図7-28．シャフリングによる平均回線長の変化（３次元立方格子）

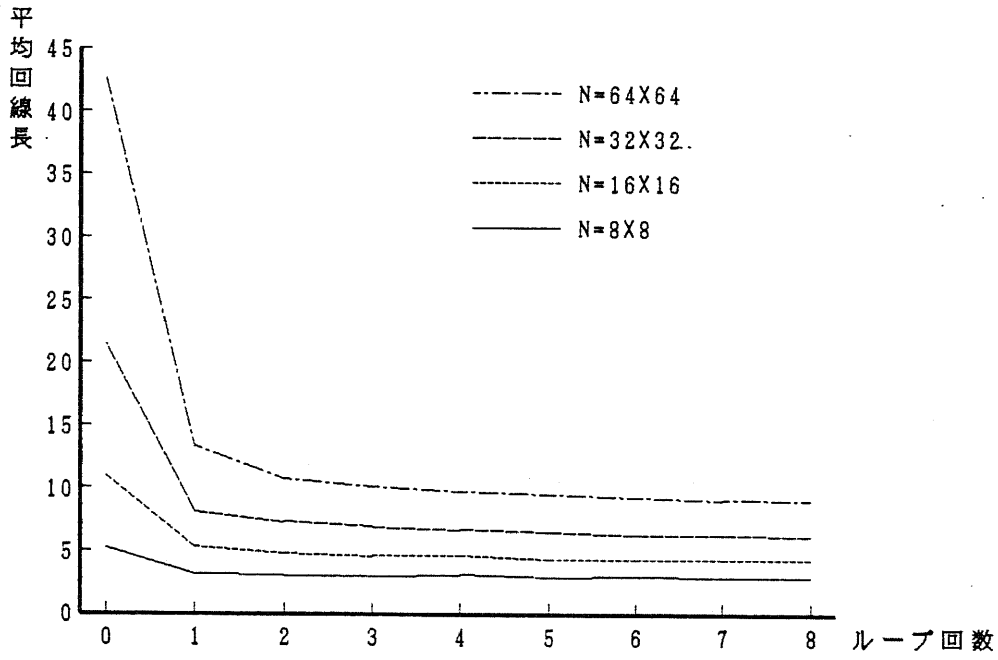


図7-29．シャフリングによる平均回線長の変化（ランダムⅠ）

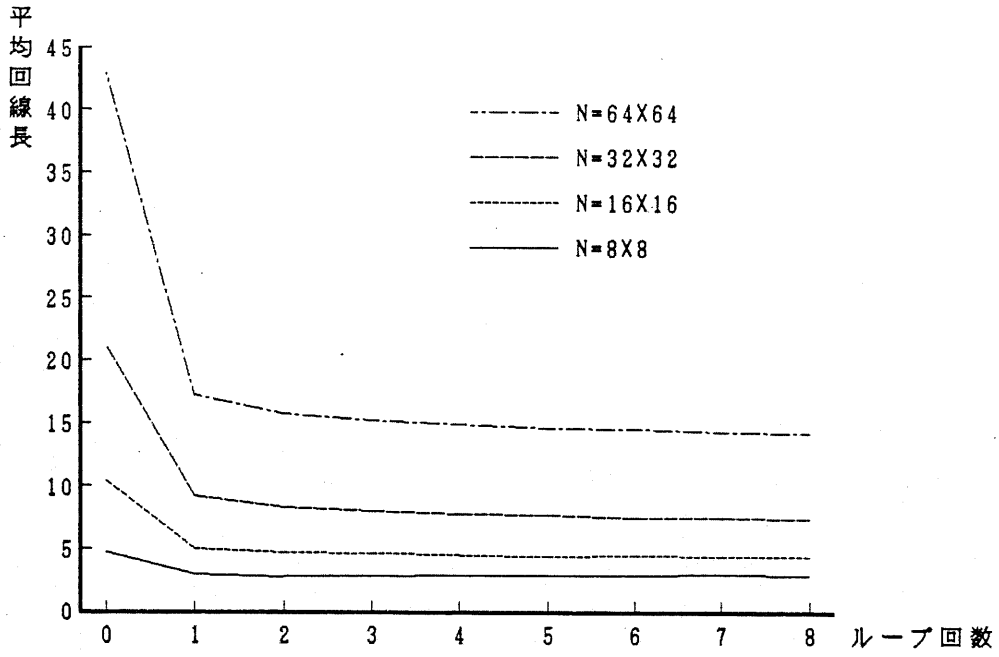


図7-30．シャフリングによる平均回線長の変化（ランダムⅡ）

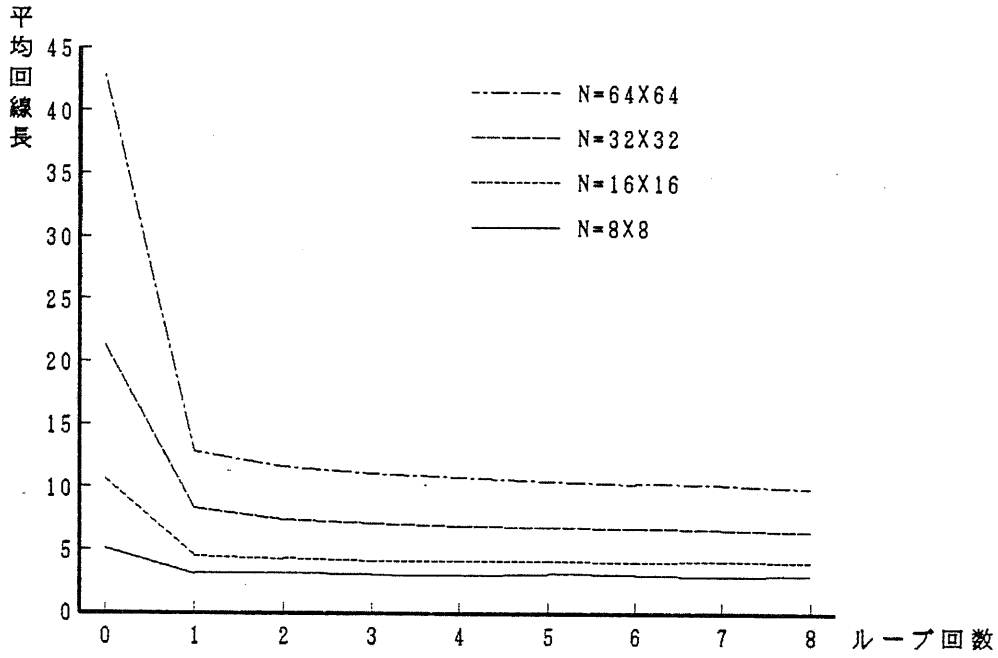


図7-31．シャフリングによる平均回線長の変化（ランダムⅢ）

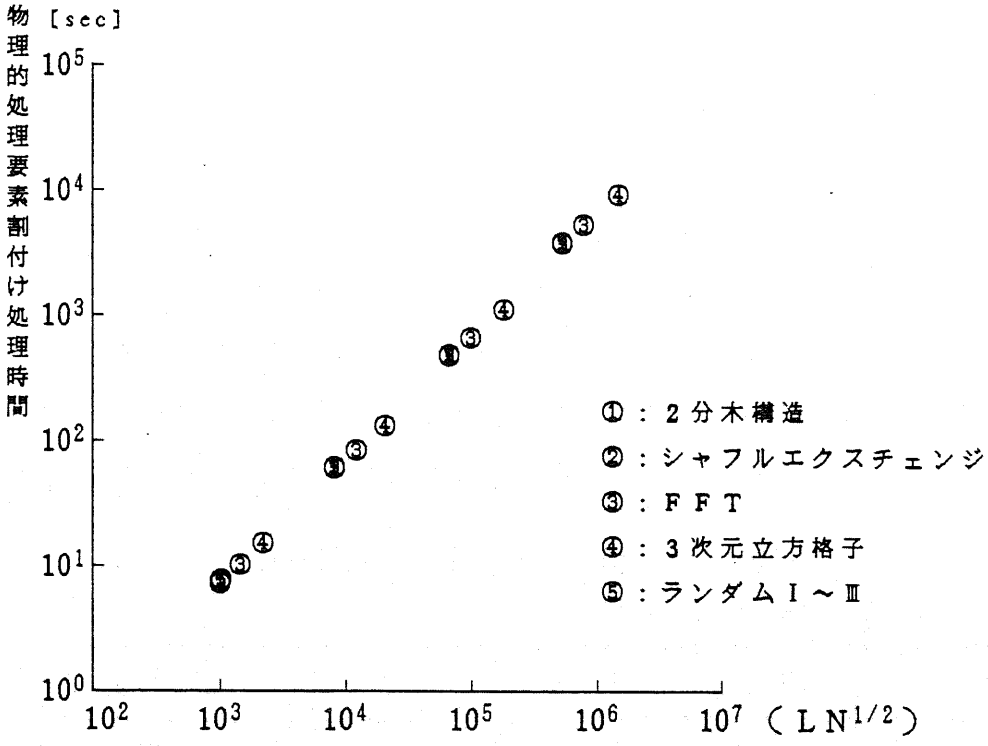


図7-32．物理的処理要素割付けの処理時間

⑵割付け結果

物理的処理要素の割付け結果を評価するには，最適な割付けとの比較を行なうことが望ましいが，一般的な結合関係については必ずしも可能ではない．

そこでここでは各結合関係に対する物理的処理要素割付け結果を個別に検討する．

①２分木構造

図７－１８においては，ｘ方向とｙ方向の結合路の使用状態はほぼ等しい．回線数の少ない結合路が比較的多く，有効に利用されていないとも考えられるが，この結合関係は本質的に下位では局所的で，上位になるに従って広範囲になるためにこのような分布になると考えられる．この結合関係においては図７－３３に示すHyper-H treeを変形した再帰的な配置によりほぼ最適な割付けが得られる[Snyder 1982]．この最適割付けと物理的処理要素割付けアルゴリズムによる割付け結果について回線長の平均値と２乗平均値を表７－２に示す．回線長の平均値は最適割付けに比較的近い値となっているが，２乗平均値は処理要素が多くなるとかなり大きくなっている．これはシャフリングの評価関数に回線長に関する因子が含まれていないために回線長のばらつきが大きくなることが原因であり，回線長を考慮に入れた評価関数を導入すれば改善されると考えられるが，本システムでは回線長のばらつきが通信に与える影響は小さくあまり重要でないので，現在の評価関数でも充分である．この最適割付けに対して回線設定アルゴリズムを適用した結果を図７－１８と同様にまとめて図７－３４に示す．これは12段4095ノードの２分木を64×64のプロセッサアレイにマッピングしたものである．これら両図の比較からは直接的な物理的処理要素割付けの様子はわからないが，回線数から判断すると比較的良好な割付け結果となっている．

表７－２．最適割付けと提案方法の比較（２分木構造）

（回線長　平均／２乗平均）

処理要素数	６４	２５６	１０２４	４０９６
最適割付け	1.76/ 4.26	1.77/ 4.52	1.83/ 5.63	1.86/ 6.72
提案方法	2.02/ 4.69	2.23/ 6.89	2.33/ 8.61	2.52/13.09

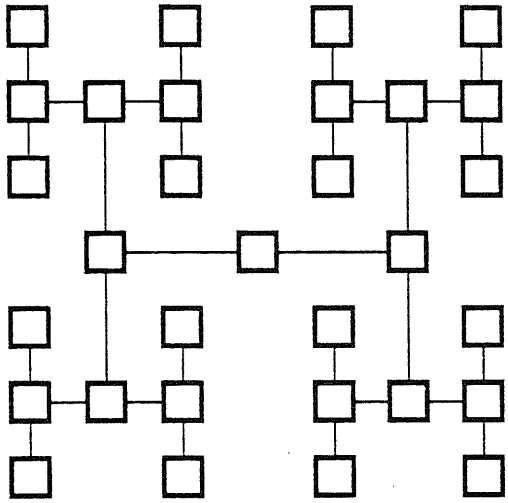


図7-33．Hyper-H tree の配置

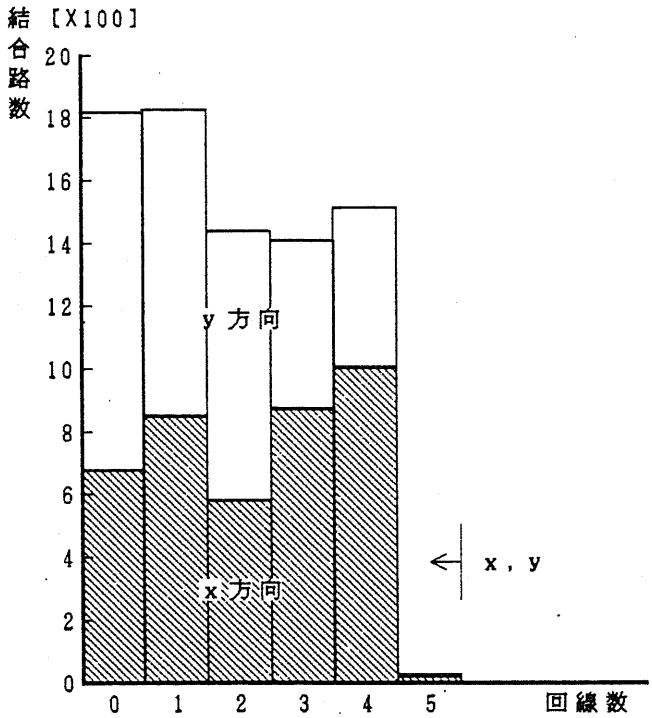


図7-34．結合路を通過する回線数（2分木構造　最適割付け）

②シャフルエクスチェンジ

図7－19においては，全体的にy方向よりx方向の回線数が多くなっており，回線数の最大値もy方向の結合路では12であるのに対してx方向では16となっており偏りが大きい．この結合関係においては，シャフルは広範囲な結合であるのに対して，エクスチェンジは隣り合う処理要素間という非常に局所的な結合となっており，このエクスチェンジの結合がx方向となっているためにこのような差が生じてきたものと考えられる．初期割付けにおいてはx方向に連続した番号の処理要素を配置したため，これがシャフリングによっても崩されずにそのまま保存されたことがこの原因である．これはかなり特殊な状況であり，この結果からすぐにシャフリングによるアルゴリズムが不適当であるとは言えない．改善のためには，斜め方向に対するシャフリングの評価関数に，回線のx方向とy方向の距離に関する因子を加える方法と，初期割付けにおいてx方向とy方向の回線長ができるだけ等しくなるような配置とする方法とが考えられる．いずれも比較的簡単に実現できる．

③ＦＦＴ

図7－19においてはシャフルエクスチェンジに比べ，x方向の回線数がさらに大きくなっている．シャフルもビット逆順も初期割付けにおいてはx方向とy方向は均等であるので，これは単にエクスチェンジの結合関係が保存されただけでなく，シャフルあるいはビット逆順に対しても影響を与えてx方向にそろってしまったためと考えられる．これを改善するには上記の斜め方向に対するシャフリングの評価関数に変更を加える方法が有効であると考えられる．

④3次元立方格子

この結合関係は自動的な割付けにとっては最も困難なものの1つである．図7-21においてはy方向の回線数が非常に多く，このために結合路を通過する最大回線数も大きな値となってしまっている．これもやはり②に述べたような方法により改善可能であると考えられる．この結合関係においては図7-35に示すように論理的処理要素を配置することにより，最適に近い割付けが可能である．これは回線設定アルゴリズムを適用するまでもなく回線設定が可能であり，これを図7-21と同様にまとめたものが図7-36である．これは16×16×16の3次元立方格子を64×64の処理要素に割り付けたものである．図7-21をこれと比較すると，物理的処理要素割付けはあまりうまくいっていないことがわかる．最適割付けでは各結合路を通過する回線数の最大値がx方向，y方向ともに10であるのに対し，物理的処理要素割付けアルゴリズムによる割付け結果ではx方向では22，y方向では24と，2倍以上になっており，これをアルゴリズムの改良により最適に近付けることは困難であると考えられる．これは自動的な方法で規則的な結合関係に対して全体的な特徴を捉えることは極めて困難であるためで，何らかの方法によりこの情報を外から与えてやることが必要であろう．例えば同一のx，y座標を持つ論理的処理要素はできるだけ近くにまとめるといった情報を加えることができれば自動的割付けによっても図7-35に近い結果が得られるであろう．このことから，この物理的処理要素割付けアルゴリズムは3次元立方格子のように規則的で最適な割付けが容易に推定できるような結合関係に適用しても効果は期待できない．

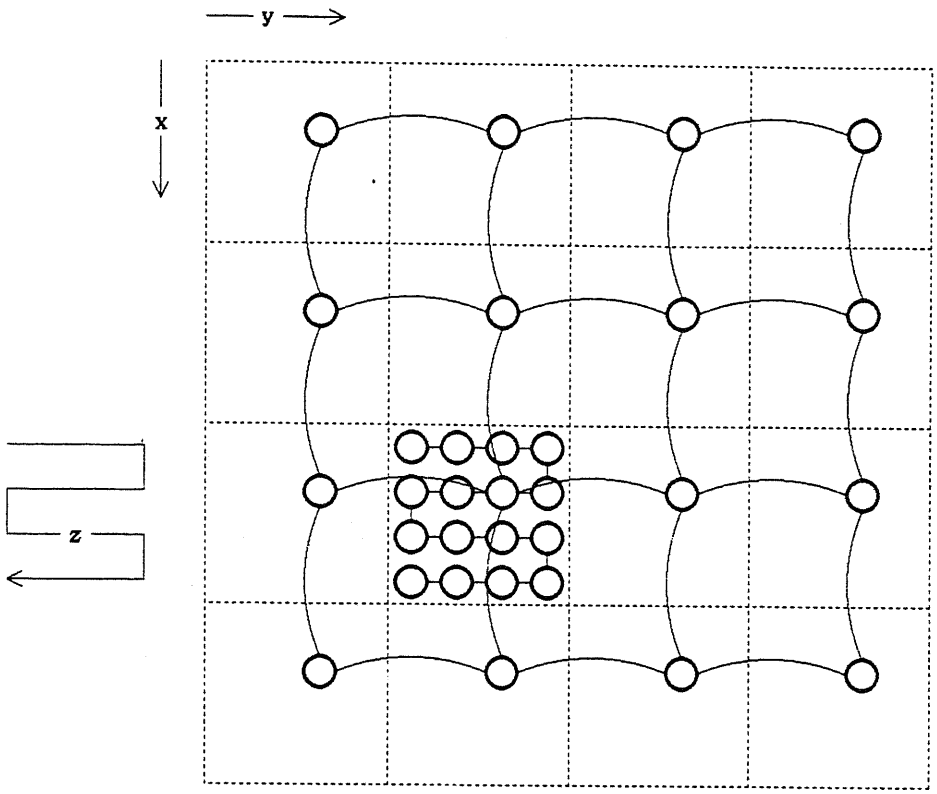


図7-35．3次元立方格子の最適割付け

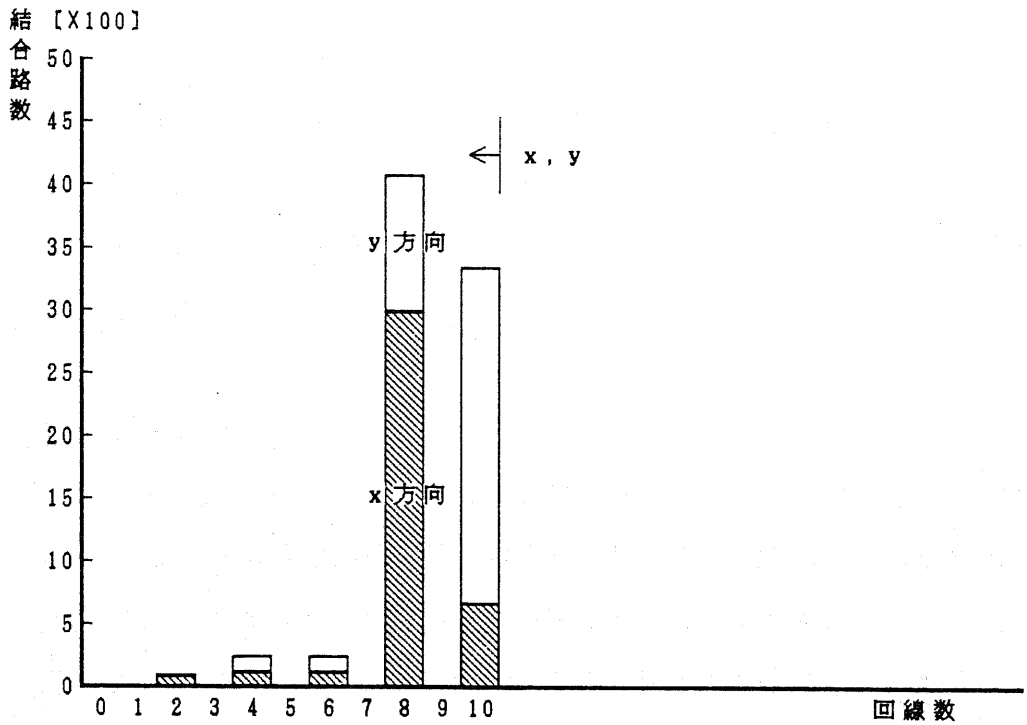


図7-36．結合路を通過する回線数（3次元立方格子　最適割付け）

⑤ランダムⅠ

この結合関係は3次元立方格子と異なり，最適割付けの推定が困難であり，客観的に評価することは難しい．図7－22においては，x方向とy方向の回線数がほとんど同じであることから，初期割付けにおいてx方向とy方向が均等であればシャフリングによっても回線方向が偏ることはないと考えられる．

⑥ランダムⅡ

図7－23においては回線数の広い範囲に対して結合路が分散している．すなわちこれは各結合路を通過する回線数が場所により全くまちまちであることを示している．この結合関係は回線数は前項のランダムⅠと同じであるが，各処理要素に接続される回線数には何らの制限もなく，従って1本も回線が接続されていない処理要素も存在する．このような結合関係に物理的処理要素割付けアルゴリズムを適用すると，接続回線数の多い論理的処理要素はアレイの中心部に集中し，接続回線数の少ない論理的処理要素は周辺部に分散する．このため，周辺部の結合路にはほとんど回線は通らず，中心部に近づくに従い回線数が増えることになり，図7－23のような分布になると考えられる．このように，本物理的処理要素割付けアルゴリズムは各処理要素に接続する回線数のばらつきが極端に大きいような結合関係に対しては効果的な割付けを行なうことができない．この改善のためには，シャフリングの評価関数にマクロ的な結合路の混雑度を導入することが考えられる．これにより接続回線数の異なる処理要素を空間的に適当に分散させることが可能となれば前項のランダムⅠと同程度の回線設定が可能となると考えられる．

⑦ランダムⅢ

図7－24においては，ランダムⅠの図7－22とほぼ同じような傾向を示している．この結合関係では各処理要素はすべて4本ずつの回線が接続されており，ランダムⅠでは各2本ずつ以上の任意本数の回線が接続されていることから，接続回線数の少々のばらつきは物理的処理要素割付け結果にはほとんど影響を与えないことがわかる．このことから逆にランダムⅡはかなり特殊な状況であると言える．

以上をまとめると，物理的処理要素割付けアルゴリズムの改善点として以下のような事項が考えられる．

(a). 回線方向を考慮した初期割付け
(b). x，y 方向の結合路の使用状態を考慮に入れた評価関数
(c). マクロ的な結合路の混雑度と接続回線数を考慮した評価関数

これらの改善により物理的処理要素割付けはより良い結果を得ることができると考えられる．しかし，結合関係に何らかの規則性がある並列処理アルゴリズムに対してはできるだけその特徴を利用して最適に近い割付けを外部から与えるようにした方が効率がよいであろう．

7.4.4.回線設定に関する評価

本節では，7.3節に述べた回線設定アルゴリズムに対して計算量と設定結果について評価する．

(1)計算量

計算量に関連するパラメータとして，総回線長をℓ，i番目の回線のx方向，y方向の距離をそれぞれx_i，y_iとおいて評価を行なう．

回線設定には，まず各所で使う組合わせの値

$$ {}_mC_n \quad (0 \leq m \leq 2N^{1/2},\ 0 \leq n \leq m/2) \qquad (7-16) $$

を求めるためにO(N)の計算量が必要である．

全回線のソーティングには，式(7－4)のℓ_iの計算は1回線当りO(1)であるので，全体ではソーティングのためにO(LlogL)の計算量が必要である．

各結合路を通る回線数の期待値の計算には，式(7－7)を全ての結合路に対して求めることになり，式(7－5)の評価回数をλとおくと

$$ \lambda = \sum_{i=0}^{L-1} (2x_i y_i + x_i + y_i) \qquad (7-17) $$

となり，これはO(λ)の計算量になる．

i番目の回線が各結合路を通過する期待値を差し引くためには，式(7－5)を($2x_iy_i + x_i + y_i$)回評価する必要がある．従って，全回線では式(7－17)と同じになり，O(λ)の計算量を要する．

経路選択では，各結合路の期待値が大きい方から順に除外してゆく方法をとるため，単純にソーティングを行なうと計算量が多くなるが，次のようにすると計算量は減る．

(1)試行錯誤的に任意の経路を1本選択し，その経路中で期待値が最大の結合路

をさがし，それより大きい期待値を持つ結合路全てを除外する．

(2)試行錯誤的な経路選択においてバックトラッキングを用い，1度失敗した結合路を除外することにより再試行を避ける．

この方法では経路選択が完了するまでに（$2x_i y_i + x_i + y_i$）ケ所の結合路を除外することになり，全回線でも$O(\lambda)$以下になる．

期待値の補正は回線が通過する結合路に対してのみ行なえば良いので，全体で$O(\ell)$となる．

従って，以上を全て合計すると，回線設定の全計算量は$O(\lambda)$となる．

この経路設定アルゴリズムを実際にプログラムにより実行させた場合の処理時間の測定値を図7－37に示す．

使用した計算機は前節で述べたものと同じである．

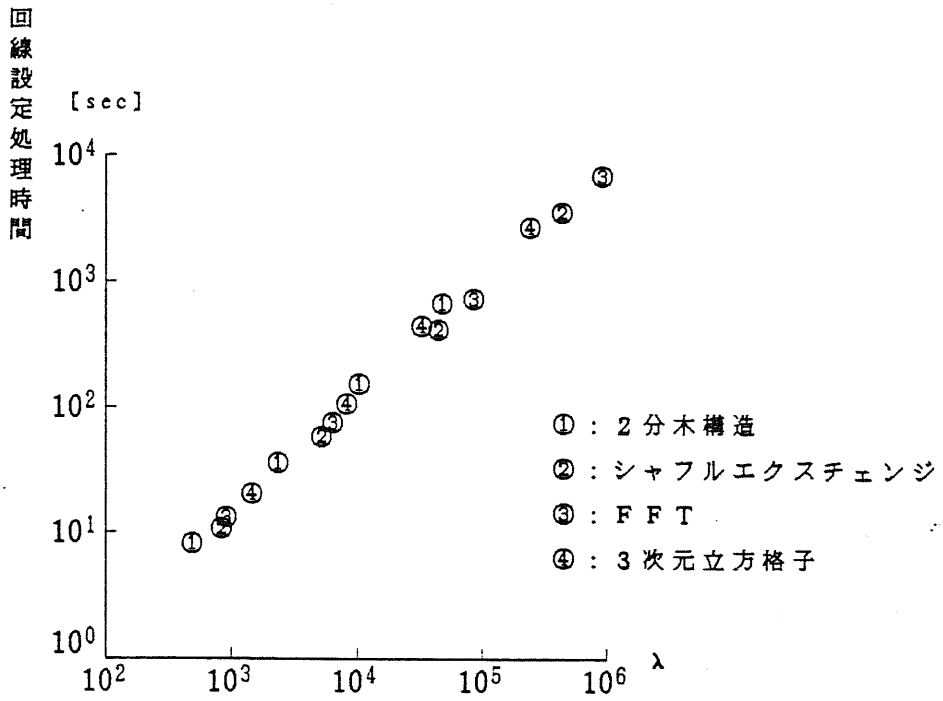
測定結果によると，回線設定処理時間はいずれも傾き1の直線にほぼ一致していることから，$O(\lambda)$になっていることがわかる．この測定結果から回線設定処理時間T_{LS}は

$$T_{LS} = 1.02\times10^{-5}\lambda + 1.14 \quad [\,s\,] \qquad (7-18)$$

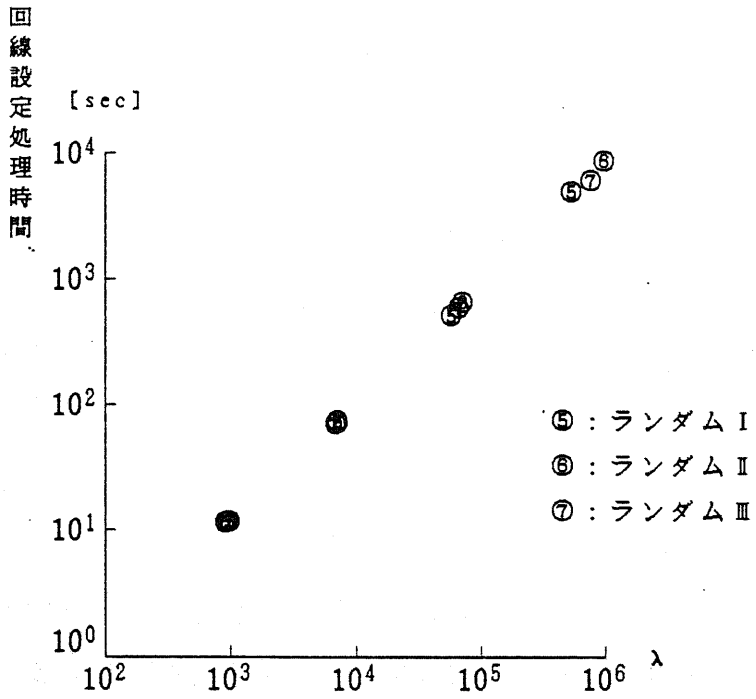
と推定できる．実際的には64×64処理要素，10000回線程度の結合関係でほぼ数十分～数時間である．このプログラムでは倍精度浮動小数を用いて演算を行なっているが，使用した計算機には浮動小数演算器は装備されておらず，ソフトウェアにより処理を行なっているために計算時間が長くなっている．従って，浮動小数演算器付きのプロセッサを用いれば処理時間は少なくとも数分の1には改善されると考えられる．この処理量は本システムの目的からして許容できる範囲にある．

(2)回線設定の結果

最適な回線設定を求めることは，極めて規則的な物理的処理要素割付けに対しては可能であるが，一般の場合は非常に困難であり，回線設定結果を客観的に評



(a). 結合関係①～④



(b). 結合関係⑤～⑦

図７－３７．回線設定処理時間

価することは難しい．そこで，ここでは設定結果の結合路の使用状態をもとに検討を行なう．

物理的処理要素割付けが適当でありかつ回線設定がうまくゆくと，図7-36の例のように，各結合路における回線数がその最大値に近い値に集中する．

2分木構造は回線数も少なく，局所的な結合が多いため最適な物理的処理要素割付けに対しても図7-34のように回線数にばらつきが生ずる．この場合，回線数に対する結合路数の分布が回線数の大きい方にどの程度伸びているかが重要である．図7-34ではこの点からは良い結果となっているが，物理的処理要素割付けアルゴリズムによった場合は図7-18に示すように少々分布が伸びている．

現在，回線設定アルゴリズムでは最短経路のみを対象とし迂回路を考慮していないので，迂回路も許すようにアルゴリズムを変更することによりこの点についてはある程度改善されると考えられる．しかしこれを最初から行なうと全体としては悪い結果となると考えられ，現在のアルゴリズムによる設定後にその結果の修正として迂回路を導入すべきであろう．

シャフルエクスチェンジは図7-19に示すように回線数が比較的多いところに分布の中心があり，全体的には望ましい設定となっているが，分布がかなり伸びている．これを改善するには部分的な迂回や他の回線との入れ替えだけでは対処できず，物理的処理要素割付けの改善が必要である．

ＦＦＴに関しては図7-20に示すようにx方向とy方向の差が大きく，分布にもかなり差がある．しかし，回線設定では各結合路を通過する回線数の最大値が問題であり，この場合結合路の使用数が多いx方向の分布が重要であり，この分布から判断すると回線設定としては比較的良い設定結果となっていると判断できる．

3次元立方格子では物理的処理要素割付けがうまくいっていないため，図7-21に示すように回線設定もあまり良い結果が得られていない．

ランダムⅠおよびランダムⅢでは回線数の最大値に近いところに結合路が集中しており，分布もあまり伸びていないことから良い設定結果が得られていると判断できる．

ランダムⅡは物理的処理要素割付けにより接続回線数の多い論理的処理要素が

中央部に集中してしまっているため，周辺部の結合路では回線数が非常に小さいのに対し，中心部では多数の回線が集中して結合路を通過する回線数が多くなっている．このため，図７－２３では回線数が小さい値から最大値に近い値まで結合路が一様に分布している．これは回線設定では対処できない問題であり，物理的処理要素割付けの改善が必要である．

以上をまとめると，回線設定アルゴリズムでは与えられた物理的処理要素割付けに対しては比較的良い設定結果を示しており，改善点としては迂回路の導入が考えられる．

また，回線設定結果を物理的処理要素割付けにフィードバックして修正を加えることで局所的な混雑を避けることができるようになり，分布が回線数の大きい方に伸びることを抑えることが可能であろう．

7.4.5.資源割付けの総合的評価

本節では資源割付け方法に対して総合的な評価を加える．

前節にも述べたように，回線設定アルゴリズムは良好な設定結果を与えるが，物理的処理要素割付けアルゴリズムについてはより多くの検討と改善が必要である．

表７－１には資源割付けの結果のみを示したが，これらのうち資源割付けの最終的な結果として最も重要なのは各結合路を通過する回線数の最大値であり，これは各隣接処理要素間の物理的な結合路の配線と直接関係する．4.5節の式（４－５）で定義したＰの値がこの最大回線数の４倍以上であることが回線設定が実際に行なえるための必要条件である．図４－１４に示した結合路トポロジーではＰ＝144であるから，最大回線数が36より大きくなるような割付けではこのような結合路トポロジーを持つシステム上に並列処理アルゴリズムをマッピングすることはできないことになる．

２分木構造は結合関係が単純であるため，容易にマッピング可能である．

シャフルエクスチェンジは比較的複雑であるにもかかわらず4096個の論理的処理要素を64×64のプロセッサアレイに比較的容易にマッピングできることがわかる．

ＦＦＴはシャフルエクスチェンジにビット逆順を追加したものであり，それだけ各結合路を通過する回線数も多くなるが，現在の資源割付けのままでもある程度対応できる．

３次元立方格子は物理的処理要素割付けアルゴリズムの適用の効果が少なく，自動的な割付けには不適当であるが，最適割付けが容易に得られ，これによればプロセッサアレイに対するマッピングは非常に容易である．

ランダムＩとランダムⅢは処理要素間の結合がランダムではあるが，ランダムⅢでは全ての処理要素に接続される回線数はすべて４本であり，ランダムＩでも各２本以上となっているため，１つの処理要素に回線が集中することはあまりないために比較的良好な割付け結果となっている．

ランダムＩは回路シミュレーションにおける各セル間の結合関係と比較的近いと考えられ，結合関係に必ずしも局所性が仮定できない応用に対しても本資源割付けアルゴリズムを用いることでシステムへのマッピングが可能であることを示

している．

ランダムⅡは今回用いた結合関係中では最も割付けが難しい結合関係であり，本方式のような単純なアルゴリズムでは最適化できない．これは主に物理的処理要素割付けに問題があるためで，アレイ全体の回線密度を適当に分散させるような割付け方法が必要とされる．しかしこのモデルは現実的にはかなり特殊なものであり，本システムの適用範囲に制約を与えるものではないと考えられる．むしろ，密な結合関係にある少数の論理的処理要素を，それより大規模なプロセッサアレイにマッピングするような状況に近いと考えられ，これは7.1節で述べた論理的処理要素割付けで対処すべき問題である．

以上の検討により，本章で述べた物理的処理要素割付け方法は結合関係が複雑で最適割付けを求めることが困難であるような場合に有効であり，規則的な結合関係を持つ応用にはできるだけ最適に近い割付けを外部から与えるべきである．また，回線設定についてはさまざまな場合に対して最適に近い結果を得ており，実際の問題に対しても有効であると考えられる．

7.5. システムの適応性の検討

前節において，資源割付けに関する検討を行なった結果，処理要素数64×64程度の規模のシステムに対してはさまざまな結合関係のマッピングが可能であることがわかった．システムの規模を256×256程度とした場合にこれを最大限に利用したマッピングが可能であるかどうかは単純には推定できないが，2分木構造や3次元立方格子に対しては適応可能であり，その他の応用に対しても割付け方法の改善により対応できると思われる．

また，このような方法で回線設定が不可能な場合でも，プログラムに少々手を加えることにより適応可能となる．これには，適当な処理要素においてデータを中継することにより回線を多重化して回線数を減らす方法と，並列処理アルゴリズムを複数のステップに分けて，それぞれでスイッチパターンを切り換える方法とがある．第一の方法はデータの転送順序が定まっている場合に図7-38に示すように処理要素の入出力レジスタを介することで蓄積交換を行なって回線を多重化するものであり，演算部とは独立にデータ転送ができるため部分的あるいは規則的な場合に有効である．第二の方法はステップが終了するごとにドメインコントローラが介入してスイッチマトリクスの設定を切り換えるものであり，切り換えにはオーバヘッドが伴うが処理中は本来の動作速度が得られるのでＦＦＴにおけるビット逆順とシャフルエクスチェンジのようにステップ分けにより結合関係が単純になるような場合に有効である．

このように，応用のデータ依存関係がそのままプロセッサアレイにマッピングできない場合でもプログラム上の工夫により対応できるため，本システムはかなり広い範囲の応用問題に適応可能である．

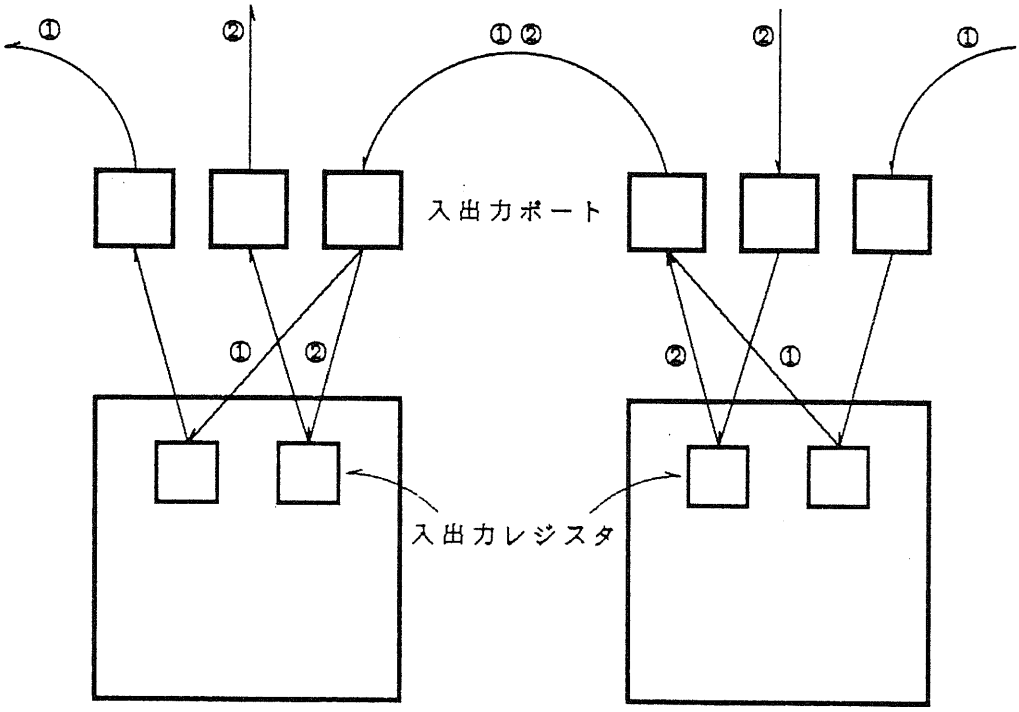


図7-38．入出力レジスタを用いた回線の多重化

第8章
システムの性能評価

本章では，第4章および第5章に示したシステムの構成により実現可能な処理能力を明らかにし，システムの性能評価を行なう．8.1節では結合回路網のデータ転送のオーバヘッド，8.2節では演算実行制御のオーバヘッドを検討し，8.3節では処理要素の基本的演算能力を示し，8.4節で実際の並列処理アルゴリズムに対する処理時間を求めることによりシステムの性能評価を行なう．

8.1. データ転送のオーバヘッド

本節では第4章に述べた結合回路網の構成においてデータを転送する際のオーバヘッドを求める．これは，演算装置が入出力レジスタにデータを書き込んでから，入出力ポートを通してデータを転送し，目的の処理要素の入出力レジスタにデータが到着するまでの遅延時間を言う．

データ長をLバイト，入出力ポート数をPとおくと，入出力レジスタと入出力ポートの間の結合は4.8節に述べたように入出力ポートに対する固定タイムスロット割当ての共通バスによるため，図8-1に示すように入出力レジスタから入出力ポートへのデータ転送時間D_{RP}は

$$D_{RP}=\left(L-\frac{1}{2}\right)P+3 \quad \text{クロック} \qquad (8-1)$$

となる．ただしこれは入出力ポートが送信データ待ちの状態である場合についての式であり，データバス幅は8ビットと仮定している．これには入出力レジスタにデータが書き込まれてからこの入出力レジスタに対して入出力ポートがアクセスするまでの平均時間P／2クロックと，Lバイトのデータ転送を行なうための時間（L－1）Pクロックおよび共通バスのパイプラインのステップ数3クロックが含まれている．

回線の中継処理要素数をNとおくと，結合路はビットシリアルであり，処理要素を1回経由するごとに同期のために1クロックの遅延が生ずるので，入出力ポートのシフトレジスタから目的の処理要素の入出力ポートのシフトレジスタまでの転送時間D_{PP}は

$$D_{PP}=8L+N+1 \quad \text{クロック} \qquad (8-2)$$

となる．第1項はデータ送出時間でデータビット数に等しい．第2項は中継処理要素における同期のための遅延であり，第3項はスタートビットの送出時間である．

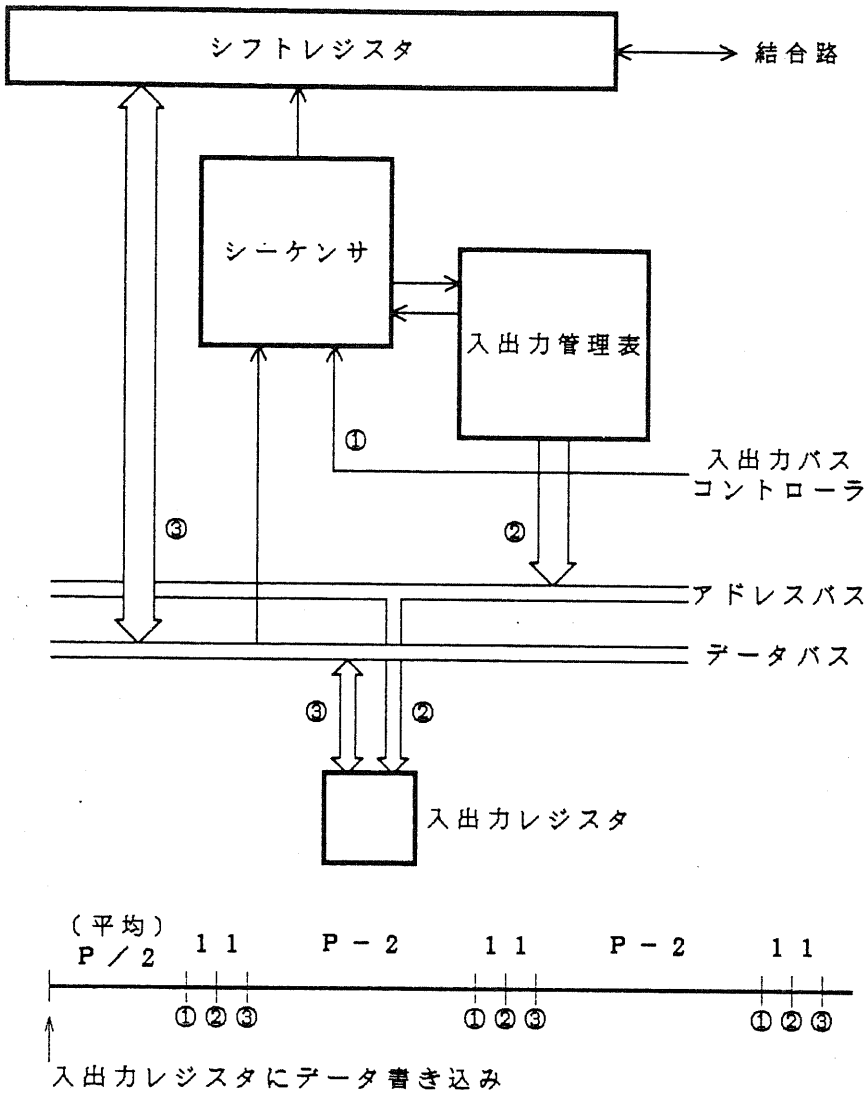


図8－1．入出力レジスタと入出力ポートの間のデータ転送時間

入出力ポートから入出力レジスタへのデータ転送時間D_{PR}はD_{RP}と等しく

$$D_{PR}=\left(L-\frac{1}{2}\right)P+3 \quad \text{クロック} \qquad (8-3)$$

である．

従って，これらのオーバヘッドを合計したデータ転送遅延時間D_Tは

$$D_T=(2L-1)P+8L+N+7 \quad \text{クロック} \qquad (8-4)$$

となる．

例えば入出力ポート数Pを16，データ長Lを2バイト，中継処理要素段数Nを32とすると，演算装置でデータが作られてから，これが目的の入出力レジスタに到着するまでに103クロックかかることになる．処理要素の動作クロックを20MHzとすると，これは約5μsである．

8.2.演算実行制御のオーバヘッド

本節では第５章に述べた演算部の構成における演算実行制御のオーバヘッドを求める．これは，あるプログラムモジュールが実行可能になってから命令ストリームの実行を開始するまでの時間である．

演算実行制御の動作タイミングを図８－２に示す．入出力レジスタにデータが全部そろってプログラムモジュールが実行可能となってから，これをＰＭ管理装置が検出するまでの遅延時間D_Dは図８－２の①→③に相当し，

$$D_D = 3 \quad \text{クロック} \qquad (8-5)$$

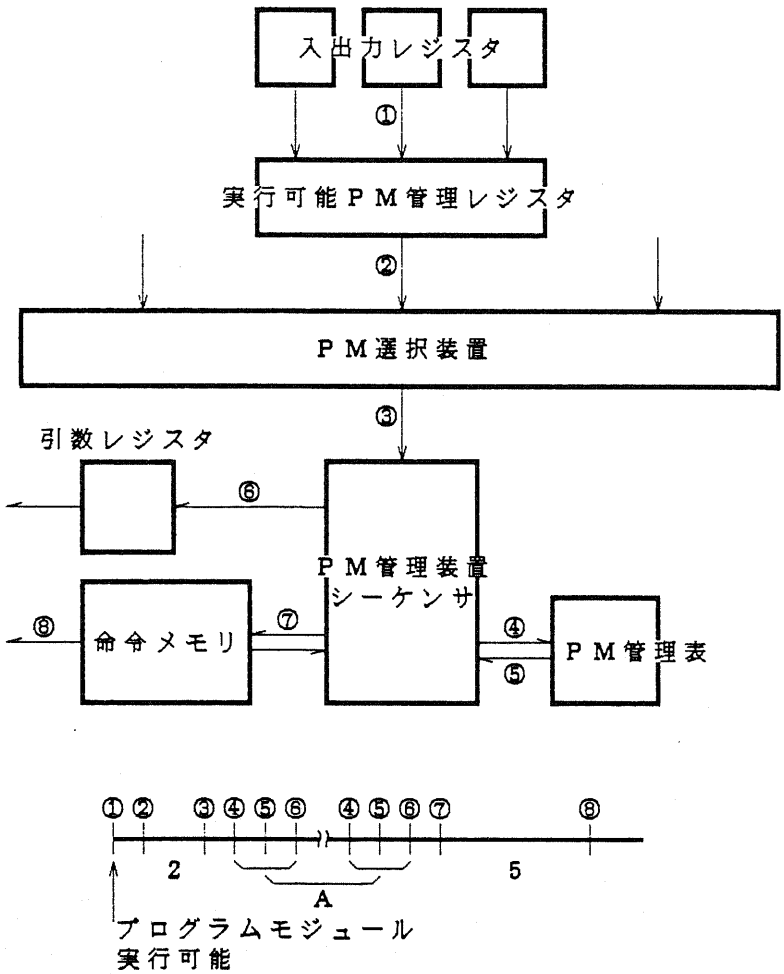


図８－２．演算実行制御のオーバヘッド

である．これにはＰＭ選択装置のプライオリティエンコーダにおける遅延が含まれる．

プログラムモジュールが実行可能になったことをＰＭ管理装置が検出してから演算装置が命令ストリームの実行を開始するまでの遅延時間D_Pは図８－２の③→⑦に相当し，プログラムモジュールの引数の数をＡとおくと

$$D_P = 3A + 9 \quad \text{クロック} \qquad (8-6)$$

となる．③は選択されたプログラムモジュール番号であり，④でこれに対応してＰＭ管理表に引数のエントリ番号を与え，⑤で引数を取り出して⑥で引数レジスタにこれを書き込む．引数の数だけこれを繰り返した後，ＰＭ管理表から命令ストリーム番号を取り出して⑦で命令メモリにこれを与える．命令メモリでは命令ストリーム番号から命令ストリームエントリ表を引いてアドレスカウンタを設定し，⑧で命令ストリームを演算装置に与える．

ただし，以上の値は演算装置が空き状態である場合にプログラムモジュールが実行可能になってから実際に実行が開始されるまでの時間である．

このD_Pは例えば引数の数Ａが４である場合，21クロックとなり，処理要素の動作クロックを20MHzとすると約１μsとなる．

8.3.演算装置の処理能力

本節では演算装置の演算能力を，基本的な処理のサブ命令ストリームを実際に記述することにより明らかにする．

演算装置は基本的な演算機能と記憶機能を有する．演算機能には８ビットのＡＬＵ，記憶機能としては８ビットの中間レジスタ32個と内部レジスタ256個を持つ．

命令セットとしては表５－１に示したように，８ビットマイクロプロセッサ程度の演算内容を持ち，全ての命令について条件フラグによる実行制御が可能である．

各命令は全て１マシンサイクルで実行が終了する．命令のデコードと実行をパイプライン化すれば１マシンサイクルを２クロックで実行することができる．クロックは20MHzと想定しているので，処理要素当りの演算能力は毎秒10×10^6命令となる．

ここでは，整数型および浮動小数型データの加算と乗算のサブ命令ストリームを実際に記述し実行時間の推定を行なうことにより基本演算に対する処理要素の処理能力を示す．

⑴８ビット整数加算

８ビット整数間の加算はＡＬＵにより１マシンサイクルで直接演算できるため，実行時間は0.1μsである．

⑵16ビット整数加算

16ビット整数間の加算はＡＬＵが８ビットであり，２回に分けて演算する必要がある．addとaddcを用いればこれは２マシンサイクルで実行できる．

⑶単精度浮動小数加算

32ビットＩＥＥＥフォーマットの浮動小数間の加算を行なうサブ命令ストリームを図８－３に示す．オペランドデータは＜r0｜r1｜r2｜r3＞と＜r4｜r5｜r6｜r7＞に格納されており，結果データは＜r8｜r9｜r10｜r11＞に格納するものと仮定する．

このサブ命令ストリームは全部で160ステップであり，実行時間は16.0μsである．

⑷ 8ビット整数乗算

8ビット整数間の乗算を行なうサブ命令ストリームを図8-4に示す．オペランドデータはr0とr1に格納されており，結果データは<r2|r3>に格納するものと仮定する．

このサブ命令ストリームは全部で48ステップであり，実行時間は4.8μsである．

⑸16ビット整数乗算

16ビット整数間の乗算を行なうサブ命令ストリームを図8-5に示す．オペランドデータは<r0|r1>と<r2|r3>に格納されており，結果データは<r4|r5|r6|r7>に格納するものと仮定する．

このサブ命令ストリームは全部で112ステップであり，実行時間は11.2μsである．

⑹単精度浮動小数乗算

32ビットＩＥＥＥフォーマットの浮動小数間の乗算を行なうサブ命令ストリームを図8-5に示す．オペランドデータは<r0|r1|r2|r3>と<r4|r5|r6|r7>に格納されており，結果データは<r8|r9|r10|r11>に格納するものと仮定する．

このサブ命令ストリームは全部で189ステップであり，実行時間は18.9μsである．

以上をまとめると，各処理要素の基本的な演算の処理時間は表8-1に示すようになる．比較のため，第2章で述べたＭＰＰにおける処理時間を（ ）中に示した[Potter 1983]．

```
 1  clr      r2          /clear r2
 2  clr      r3          /clear r3
 3  asl      r1          /shift left arithmetic
 4  sccs                 /load carry to control flag
 5  add.c    r0,r2       /add r0 to r2 conditional
 6  addc.c   zr,r3       /add carry to r3 conditional
 7  asl      r2
 8  rsl      r3
 9  asl      r1
10  sccs
11  add.c    r0,r2
12  addc.c   zr,r3
13  asl      r2
14  rsl      r3
15  asl      r1
16  sccs
17  add.c    r0,r2
18  addc.c   zr,r3
19  asl      r2
20  rsl      r3
21  asl      r1
22  sccs
23  add.c    r0,r2
24  addc.c   zr,r3
25  asl      r2
26  rsl      r3
27  asl      r1
28  sccs
29  add.c    r0,r2
30  addc.c   zr,r3
31  asl      r2
32  rsl      r3
33  asl      r1
34  sccs
35  add.c    r0,r2
36  addc.c   zr,r3
37  asl      r2
38  rsl      r3
39  asl      r1
40  sccs
41  add.c    r0,r2
42  addc.c   zr,r3
43  asl      r2
44  rsl      r3
45  asl      r1
46  sccs
47  add.c    r0,r2
48  addc.c   zr,r3
```

図8-4．8ビット整数乗算サブ命令ストリーム

```
 1  mov     r2,r12      41  clr.c   r5          81  addc.c r1,r5      121  rsl.c  r4
 2  rsl     r12         42  imld    24          82  addc.c r2,r6      122  rsl.c  r5
 3  mov     r3,r12      43  cmp     imr,r14     83  addc.c zr,r16     123  rsl.c  r6
 4  rsl     r12         44  sccc                84  clc.c            124  rsl.c  r4
 5  mov     r6,r13      45  clr.c   r4          85  sccs              125  rsl.c  r5
 6  rsl     r13         46  imld    4           86  sub.c  r0,r4      126  rsl.c  r6
 7  rsl     r13         47  bit     imr,r14     87  subc.c r1,r5      127  rsl.c  r4
 8  mov     r7,r13      48  scne                88  subc.c r2,r6      128  rsl.c  r5
 9  cmp     r12,r13     49  lsr.c   r6          89  sccs              129  rsl.c  r6
10  sccs                50  rsr.c   r5          90  neg.c  r4         130  rsl.c  r4
11  mov.c   r12,r14     51  rsr.c   r4          91  neg.c  r5         131  rsl.c  r5
12  mov.c   r13,r12     52  lsr.c   r6          92  neg.c  r6         132  rsl.c  r6
13  mov.c   r14,r13     53  rsr.c   r5          93  cmp.c  r7         133  imld.c 4
14  mov.c   r0,r14      54  rsr.c   r4          94  clr    r17        134  sub.c  imr,r13
15  mov.c   r4,r0       55  lsr.c   r6          95  tst    r16        135  imld   0xcc
16  mov.c   r14,r4      56  rsr.c   r5          96  scne              136  bit    imr,r6
17  mov.c   r1,r14      57  rsr.c   r4          97  sec.c             137  sceq
18  mov.c   r5,r1       58  lsr.c   r6          98  rsr.c  r6         138  rsl.c  r4
19  mov.c   r14,r5      59  rsr.c   r5          99  rsr.c  r5         139  rsl.c  r5
20  mov.c   r2,r14      60  rsr.c   r4         100  rsr.c  r4         140  rsl.c  r6
21  mov.c   r6,r2       61  imld    2          101  imld   1          141  rsl.c  r4
22  mov.c   r14,r6      62  bit     imr,r14    102  add.c  imr,r13    142  rsl.c  r5
23  mov.c   r3,r14      63  scne               103  addc.c zr,r17     143  rsl.c  r6
24  mov.c   r7,r3       64  lsr.c   r6         104  tst    r6         144  imld.c 2
25  mov.c   r14,r7      65  rsr.c   r5         105  sceq              145  sub.c  imr,r13
26  mov     r12,r14     66  rsr.c   r4         106  mov.c  r5,r6      146  imld   0xaa
27  sub     r13,r14     67  lsr.c   r6         107  mov.c  r4,r5      147  bit    imr,r6
28  imld    0x80        68  rsr.c   r5         108  mov.c  zr,r4      148  sceq
29  bis     imr,r2      69  rsr.c   r4         109  tst    r6         149  rsl.c  r4
30  bis     imr,r6      70  rsr     r14        110  sceq              150  rsl.c  r5
31  imld    8           71  sccs               111  mob.c  r5,r6      151  rsl.c  r6
32  cmp     imr,r14     72  lsr.c   r6         112  mov.c  zr,r5      152  dec.c  r13
33  sccc                73  rsr.c   r5         113  tst    r6         153  rsl    r6
34  mov.c   r5,r4       74  rsr.c   r4         114  sceq              154  rsl    r7
35  mov.c   r6,r5       75  mov     r3,r15     115  mov.c  zr,r13     155  rsr    r13
36  clr.c   r6          76  eor     r7,r15     116  mov.c  zr,r7      156  rsr    r6
37  imld    16          77  rsl     r15        117  mov    r6,r14     157  mov    r13,r11
38  cmp     imr,r14     78  clr     r16        118  imld   0xf0       158  mov    r6,r10
39  sccc                79  sccc               119  bit    imr,r6     159  mov    r5,r9
40  mov.c   r5,r4       80  add.c   r0,r4      120  sceq              160  mov    r4,r8
```

図８－３．単精度浮動小数加算サブ命令ストリーム

```
 1  clr     r4
 2  clr     r5
 3  clr     r6
 4  clr     r7
 5  asl     r2
 6  sccs
 7  add.c   r0,r4
 8  addc.c  r1,r5
 9  addc.c  zr,r6
10  asl     r3
11  sccs
12  add.c   r0,r5
13  addc.c  r1,r6
14  addc.c  zr,r7
15  asl     r4
16  rsl     r5
17  rsl     r6
18  rsl     r7
19  asl     r2
20  sccs
21  add.c   r0,r4
22  addc.c  r1,r5
23  addc.c  zr,r6
24  asl     r3
25  sccs
26  add.c   r0,r5
27  addc.c  r1,r6
28  addc.c  zr,r7
29  asl     r4
30  rsl     r5
31  rsl     r6
32  rsl     r7
33  asl     r2
34  sccs
35  add.c   r0,r4
36  addc.c  r1,r5
37  addc.c  zr,r6
38  asl     r3
39  sccs
40  add.c   r0,r5
41  addc.c  r1,r6
42  addc.c  zr,r7
43  asl     r4
44  rsl     r5
45  rsl     r6
46  rsl     r7
47  asl     r2
48  sccs
49  add.c   r0,r4
50  addc.c  r1,r5
51  addc.c  zr,r6
52  asl     r3
53  sccs
54  add.c   r0,r5
55  addc.c  r1,r6
56  addc.c  zr,r7
57  asl     r4
58  rsl     r5
59  rsl     r6
60  rsl     r7
61  asl     r2
62  sccs
63  add.c   r0,r4
64  addc.c  r1,r5
65  addc.c  zr,r6
66  asl     r3
67  sccs
68  add.c   r0,r5
69  addc.c  r1,r6
70  addc.c  zr,r7
71  asl     r4
72  rsl     r5
73  rsl     r6
74  rsl     r7
75  asl     r2
76  sccs
77  add.c   r0,r4
78  addc.c  r1,r5
79  addc.c  zr,r6
80  asl     r3
81  sccs
82  add.c   r0,r5
83  addc.c  r1,r6
84  addc.c  zr,r7
85  asl     r4
86  rsl     r5
87  rsl     r6
88  rsl     r7
89  asl     r2
90  sccs
91  add.c   r0,r4
92  addc.c  r1,r5
93  addc.c  zr,r6
94  asl     r3
95  sccs
96  add.c   r0,r5
97  addc.c  r1,r6
98  addc.c  zr,r7
99  asl     r4
100 rsl     r5
101 rsl     r6
102 rsl     r7
103 asl     r2
104 sccs
105 add.c   r0,r4
106 addc.c  r1,r5
107 addc.c  zr,r6
108 asl     r3
109 sccs
110 add.c   r0,r5
111 addc.c  r1,r6
112 addc.c  zr,r7
```

図8－5．16ビット整数乗算サブ命令ストリーム

```
 1  mov     r3,r12
 2  eor     r7,r12
 3  mov     r2,r13
 4  asl     r13
 5  mov     r3,r13
 6  rsl     r13
 7  clr     r15
 8  sceq
 9  inc.c   r15
10  mov     r6,r14
11  asl     r14
12  mov     r7,r14
13  rsl     r14
14  sceq
15  inc.c   r15
16  imld    127
17  sub     imr,r14
18  sub     imr,r13
19  add     r14,r13
20  scvs
21  add     imr,r13
22  clr     r16
23  inc.c   r16
24  imld    0x80
25  bis     imr,r2
26  bis     imr,r6
27  clr     r8
28  clr     r9
29  clr     r10
30  clr     r11
31  asl     r4
32  sccs
33  add.c   r2,r11
34  addc.c  zr,r8
35  asl     r5
36  sccs
37  add.c   r1,r11
38  addc.c  r2,r8
39  addc.c  zr,r9
40  asl     r6
```

```
    41  sccs
    42  add.c   r0,r11
    43  addc.c  r1,r8
    44  addc.c  r2,r9
    45  addc.c  zr,r10
    46  asl     r11
    47  rsl     r8
    48  rsl     r9
    49  rsl     r10
    50
    .
    .        *
    .
    68
    69
    .
    .        *
    .
    87
    88
    .
    .        *
    .
   106
   107
    .
    .        *
    .
   125
   126
    .
    .        *
    .
   144
   145
    .
    .        *
    .
   163
*  164  asl     r4
```

```
165  sccs
166  add.c   r2,r11
167  addc.c  zr,r8
168  asl     r5
169  sccs
170  add.c   r1,r11
171  addc.c  r2,r8
172  addc.c  zr,r9
173  asl     r6
174  sccs
175  add.c   r0,r11
176  addc.c  r1,r8
177  addc.c  r2,r9
178  addc.c  zr,r10
179  asl     r2
180  asl     r12
181  mov     r13,r11
182  rsr     r11
183  rsr     r2
184  tst     r15
185  scne
186  clr.c   r8
187  clr.c   r9
188  clr.c   r10
189  clr.c   r11
```

図８－６．単精度浮動小数乗算サブ命令ストリーム

表８－１．処理要素の基本演算時間［単位 μs］

（括弧内はＭＰＰ）

	加算	乗算
８ビット整数	0.1（ 2.5）	4.8（ 8.8）
16ビット整数	0.2（ 3.7）*	11.2（18.0）*
単精度複素数	16.0（34.9）	18.9（53.0）

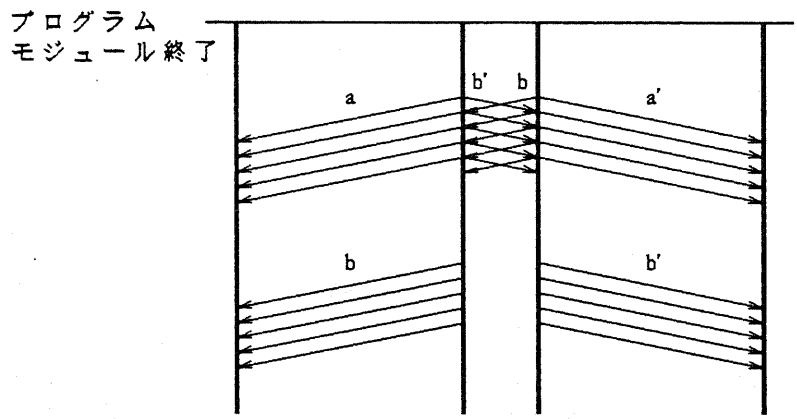
＊ ＭＰＰでは１２ビット

8.4. 実際の問題における処理性能

本節ではいくつかの実際の並列処理アルゴリズムの処理時間を求めることによりシステム全体の処理能力の評価を行なう．ここではプロセッサアレイの大きさ n×n を64×64および256×256として検討を行なう．

(1)繰り返し型ＦＦＴ

これはシャフルエクスチェンジの結合関係を用い，１処理要素に１点を対応させて演算を繰り返すＦＦＴの解法である．64×64処理要素の場合，4096点ＦＦＴとなり，１ステップ当り１点につき複素数乗算１回と複素数加算１回の演算を12ステップ繰り返すので，合計では実数乗算48回，実数加算48回の演算が必要である．256×256処理要素では65536点ＦＦＴとなり，ステップ数は16となる．また，データの転送は各ステップ間でシャフルと，エクスチェンジ→シャフルを行なうため，図８－７に示すようにシャフルの転送の２回分の時間が必要である．各処理要素ともシャフルとエクスチェンジは入出力ポートが異なるので同時に送信が可能であり，エクスチェンジの回線長は短いのでシャフルのデータ送信中にエクスチェンジのデータは受信される．このためデータ長をＬとおくと，各ステップ間のデータ転送時間は式（８－４）においてデータ長を２Ｌとしたデータ１個分



a, a'：シャフル
b, b'：エクスチェンジ→シャフル

図８－７．繰り返し型ＦＦＴのデータ転送動作

のシャフルに対する転送時間と等しくなる．シャフルの回線長は最大でnすなわち64である．演算実行制御時間は式（８－５）および（８－６）において，引数の数は５である．複素数の実部と虚部を16ビット整数および単精度浮動小数として64×64処理要素に対し，この実行時間を求める．

<u>16ビット整数</u>

演算時間　(11.2×4+0.2×4)×12＝547.2μs

データ転送時間　L=8バイト，P=16，N=64

∴ D_T＝15×16+64+64+7＝375　クロック

＝18.75　μs

∴全体では　12×D_T＝225μs

演算実行制御時間　D_P＝3×5+9＝24

∴全体では　12×(D_D+D_P)＝324　クロック

＝16.2　μs

総処理時間　788.4μs

処理性能　4096×(4+4)×12/788.4＝499 MOPS

(Mega Operation per Second)

<u>単精度浮動小数</u>

演算時間　(18.9×4+16.4×4)×12＝1646.4μs

データ転送時間　L=16バイト，P=16，N=64

∴ D_T＝31×16+128+64+7＝695　クロック

＝34.75　μs

∴全体では　12×D_T＝417μs

演算実行制御時間　　$D_P = 3\times5+9 = 24$

∴全体では　$12\times(D_D+D_P) = 324$　クロック

$= 16.2$　μs

総処理時間　　2080μs

処理性能　　4096×(4+4)×12/2080＝189 MOPS

同様に256×256処理要素に対しても処理能力を求めると，

16ビット整数　　：6963 MOPS

単精度浮動小数　：2872 MOPS

となる．

⑵パイプライン型ＦＦＴ

繰り返し型は各点に処理要素を対応させた解法であるのに対し，パイプライン型ではＦＦＴグラフをそのままプロセッサアレイにマッピングし，アレイの一辺からデータを入力して反対側の辺から結果を出力する．64×64の処理要素では256点ＦＦＴ，256×256の処理要素では4096点ＦＦＴまで実現できる．64×64の場合について以下に検討する．

256点ＦＦＴは８段のパイプラインにより実現される．図８－８に示すようにアレイ全体を厚さ４，幅64のスライスに分割し，図８－９に示すようにＦＦＴグラフの各ステップに対応する回線の接続を各ステップ間で行なう．

１組のデータに対して結果を得るまでの演算回数は繰り返し型でもパイプライン型でも同様なので１ステップあたり加算４回，乗算４回で，これを８ステップ行なう．データの転送距離はステップ間によりまちまちであるが，全体としては入出力間での最長のデータ経路が問題となり，これはアレイの大きさをｎ×ｎとおくと２ｎ，すなわち128である．演算実行制御時間は繰り返し型と同様であ

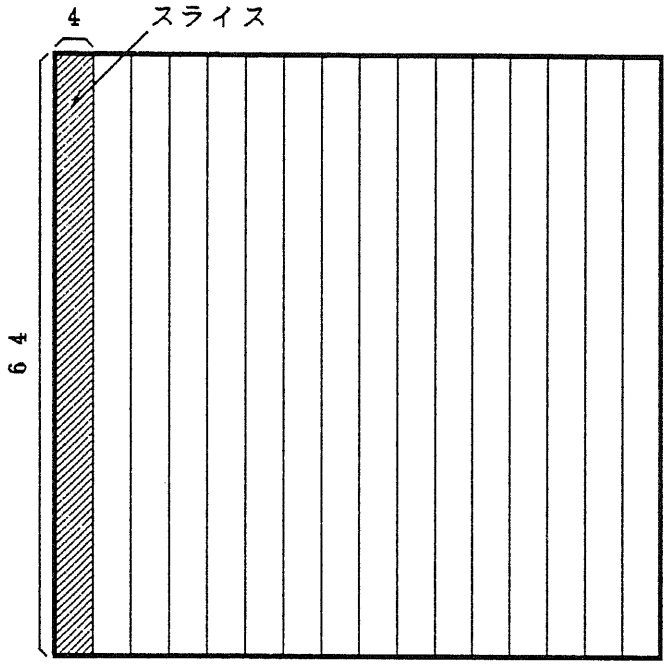


図8-8．256点パイプライン型ＦＦＴのスライス

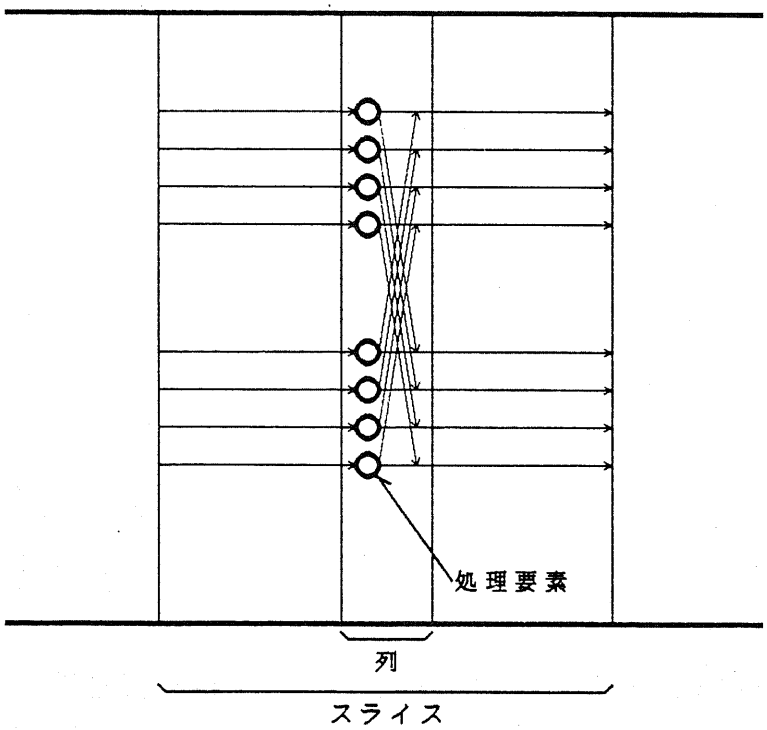


図8-9．パイプライン型ＦＦＴの回線接続

る．データを16ビット整数および単精度浮動小数としてこの実行時間を求める．

16ビット整数

演算時間　(11.2×4+0.2×4)×8＝364.8μs

データ転送時間　L＝4バイト，P=16，∑N=128

∴全体では

8((2L-1)P+8L+7)+∑N＝1336　クロック

＝66.8　μs

演算実行制御時間　8(D_D+D_P)＝216　クロック

＝10.8　μs

(総処理時間　442.4 μs)

パイプライン　(1ステップ演算時間)＋

ステップ　(1ステップ演算実行制御時間)＝47.0μs

(演算と通信はオーバラップ可能)

処理性能　256×(4+4)×8/47.0＝349 MOPS

(ただしデータが定常的に供給されている場合)

単精度浮動小数

演算時間　1129.6μs

データ転送時間　L＝8バイト，P=16，∑N=128

∴全体では　130.8μs

演算実行制御時間　10.8　μs

（総処理時間　　1271.2 MOPS）

パイプライン　　142.6 μs
　　ステップ

処理性能　　115 MOPS

同様に256×256処理要素では，

16ビット整数　　：8371 MOPS
単精度浮動小数　：2757 MOPS

となる．

③畳込み

画像処理で良く用いられる畳込みでは，全ての画素P($,$)について

$$C(\$,\$)=\sum_{i=-r}^{+r}\sum_{j=-r}^{+r}M(i,j)\cdot P(\$+i,\$+j) \qquad (8-7)$$

を計算する．3×3の畳込みではr＝1，7×7ではr＝3となる．

3×3の畳込みのプログラムは図8-10のようになり，データを8ビット整数とすると実行のタイミングは図8-11のようになる．実行時間は1360クロックで68μsとなり，処理能力としては64×64処理要素で1200MOPS，256×256では19200MOPSである．MPPでは128×128のアレイプロセッサで1320MOPSである[Potter 1982]．

7×7の畳込みでは実行時間は7251クロックで363μsとなり，処理能力としては64×64処理要素で1096MOPS，256×256では17534MOPSである．MPPでは1382

MOPSである．

copy5(init;or,ol,ou,od,P)	①	……初期データを上下左右に転送する．
copy3(il;oru,ord,Pl)	②	……左からのデータを上下に転送する．
copy3(ir;olu,old,Pr)	③	……右からのデータを上下に転送する．
mul(M,P;R)	④	……自データにM(0,0)を掛ける．
mul(Mr,Pr;Rr)		
…	⑤×8	……各データにM(i,j)を掛ける．
mul(Mld,Pld;Rld)		
add9(R,Rr,…,Rld;C)	⑥	……和を求める．

図8-10．3×3畳込みプログラム

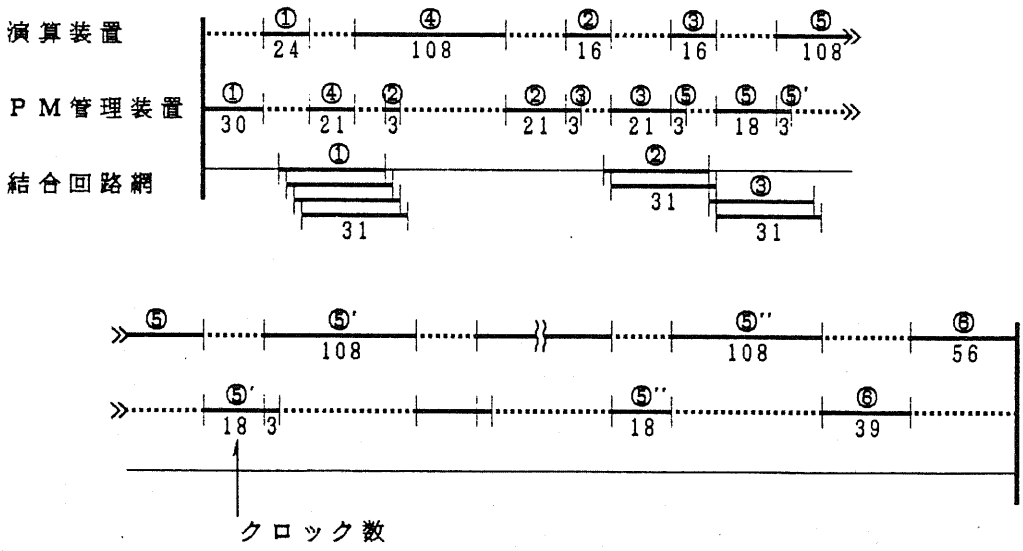


図8-11．3×3畳込みの実行タイミング

(4)総和

n個のデータの総和を求めるプログラムでは，(n－1)回の加算を行なう必要があるが，並列処理アルゴリズムでは演算の順序により処理時間が異なる．ここでは，図8-12に示す2分木構造を用いた方法による処理性能を求める．

各ステップ内における加算は並列に実行でき，1ステップの演算時間は加算2回分の演算時間である．ステップ数は全体で(log n－1)である．最長のデータ経路はほぼ3 $n^{1/2}$であるので，全処理時間はn＝4096に対して16ビット整数では70μs，単精度浮動小数では282μs，n＝65536に対しては16ビット整数では120μs，単精度浮動小数では676μsとなり，処理能力はそれぞれ59MOPS，14.5MOPS，546MOPS，97MOPSという結果になる．

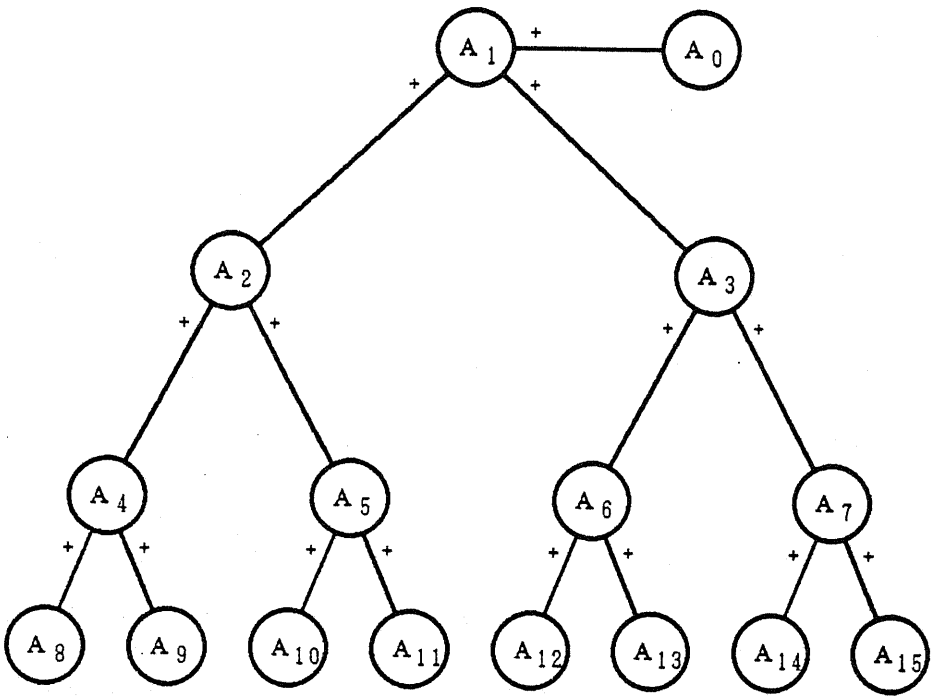


図8-12．総和の計算方法

⑸論理シミュレーション

ディジタル回路の論理シミュレーションではゲート間の結合関係が規則的でなく，また演算の単位がゲートの論理演算であり，ビット単位の論理演算を極めて大量に行なわなければならないため汎用機では効率が悪い．ここでは，ゲートのファンイン，ファンアウトをそれぞれ3とし，1ゲートに対して1プログラムモジュールを用いることを仮定して評価する．なお，ここでは資源割付けについては考えないことにする．

各命令ストリームのステップ数は，入出力レジスタへのアクセスに12，論理演算に2となり，合計で14ステップ，1.4μsである．演算実行制御には各プログラムモジュールごとに30クロック，1.5μs必要である．データ転送は回線長に依存するが回線長が30程度までは演算部とは並列に動作するのでオーバヘッドとはならない．従って処理時間は1ゲートあたり3μs程度となり，処理要素数が64×64では毎秒1.4×10^{9}ゲート，256×256では毎秒2.2×10^{10}ゲートの論理シミュレーションが可能である．論理シミュレーションの応用にはＩＢＭ，ベル研，日本電気等において専用マシンが研究されており，日本電気のシステムでは3台の専用プロセッサにより毎秒2×10^{8}ゲートの処理能力が得られるとしており[Koike 1982]，３０台程度による並列処理では毎秒2×10^{9}ゲート程度の処理能力と予想される．

以上に，いくつかの応用例に対するシステムの処理能力を検討した．これをまとめて表8－2に示す．比較のために他のシステムにおける性能を合わせて記した．

この結果から本システムはデータ依存関係および演算内容がそれぞれ単純なものから複雑なものまで，さまざまな応用に対して高い処理能力を示すことが明らかになった．

表８－２．各種の応用における処理性能

［単位 MOPS］

	64×64処理要素	256×256処理要素	参　考
繰り返し型ＦＦＴ			
4096点整数	499		
浮動小数	189		87.8[1]
			98.7[2]
65536点整数		6963	
浮動小数		2872	
パイプライン型ＦＦＴ			
256点整数	349		
浮動小数	115		
4096点整数		8371	
浮動小数		2757	
畳込み　3×3	1200	19200	1320[3]
7×7	1096	17534	1382[3]
総和　整数	59	546	
浮動小数	14.5	97	121.4[4]
論理シミュレーション	1400[6]	22000[6]	2000[5][6]

[1] HITAC S-810/20 512点倍精度浮動小数［Karaki 1984］.
[2] HITAC S-810/20 1024点倍精度浮動小数［Karaki 1984］.
[3] MPP 124X124 アレイプロセッサ［Potter 1982］.
[4] HITAC S-810/20 倍精度浮動小数［Karaki 1984］.
[5] 日本電気専用マシン 30台並列処理［Koide 1982］.
[6] 単位は10^6ゲート/秒.

第9章．結　論

本論文は均質で小規模な処理要素を多数用いることにより極めて高度の並列処理による高性能システムを実現するための構成に関する研究について述べたものである．

第2章ではまず，現在の並列処理技術とその問題点を明らかにし，本研究の位置付けを行なうために各種の並列処理アーキテクチャを，その汎用性と処理能力に着目して位置付けを行なった．この結果，

(a). 汎用並列処理システムでは汎用結合回路網と汎用制御機構が性能の限界を規定している．
(b). 専用並列処理システムでは処理要素の機能およびそれらの間の結合形態が固定的で適応範囲が狭い．
(c). この両者の間には大きな格差があり，これを埋めるシステムの実現により各種の応用に対する高度の並列処理への道が開ける．

という点が明らかになった．

このような認識に基づき，第3章では問題に適応して再構成可能な高並列処理システムについて，各部の詳細な検討に先立ち全体的概念を明らかにするために以下のような基本方針を示した．

(ｲ). 処理要素の配置を2次元格子状とする．
(ﾛ). 処理要素間の通信は静的な回線接続による．
(ﾊ). 処理要素間の演算実行制御にはデータ駆動制御方式を採用する．

これにより各種の専用並列処理システムと比肩しうる性能を持ち，かつさまざまな応用分野に対応可能なシステムの実現が可能となる．
このような構成のシステムにおける機能としては

(1). ホスト制御機能

(2).回線接続機能

(3).通信制御機能

(4).演算実行制御機能

(5).演算処理機能

(6).外部データ入出力機能

の各機能が必要であり，これらを実現するための構造あるいは概念の概略について述べた．またこれらを支援，運用するためのシステムソフトウェアは

①コンパイラ

②資源割付けソフトウェア

③プログラムローダ

④入出力制御ソフトウェア

⑤プログラム走行管理ソフトウェア

の各ソフトウェアにより構成されることを述べた．

第4章では結合回路網についての詳細な検討を行なった．まず結合回路網の条件として

i.結合回路網全体として2～200Gbpsのスループット

ii.数千～数万個の処理要素に対し，それと同等以下のハードウェア量

iii.故障に対する柔軟性

iv.結合の均質性

v.制御の分散化

が要求されることを明らかにし，並列処理システムの代表的結合形態として

(a).共通バス

(b).多段スイッチ網

(c). スイッチマトリクス

(d). 木構造回路網

(e). 2次元格子網

を選択し上記の条件に対する適性を検討した．この結果，(e)の2次元格子網が本システムの目的に最も適合することが明らかになり，処理要素の配置も2次元格子状とすることが適当であることが示された．

次に処理要素間の通信方式に関し，蓄積交換と空間分割および時間分割の静的な回線接続に対しデータ転送遅延とハードウェア機能およびその他の特質について検討を行なった．この結果，データ転送遅延特性としては回線接続方式が適当であり，ハードウェア量は空間分割回線接続が最も小さいことが示された．そこで，ブロッキングの問題はあるが，本システムの通信方式には空間分割の回線接続を採用することにした．空間分割回線接続では実現の容易さと高い転送のビットレートを得るため，処理要素を経由するごとに信号同期を取ることにし，この場合各回線のデータ伝送レートは最大で30Mbpsとすることができることを示した．

結合路のトポロジーについては接続能力の指標を導入し，現在のＬＳＩ技術による実装を想定した処理要素単位のトポロジーと，将来のＶＬＳＩ技術を仮定したドメイン単位のトポロジーについて概念を示した．

次にこの結合回路網の実現のための構成を検討し，

(1). スイッチマトリクスとスイッチ制御レジスタによる半二重双方向通信のための回線接続制御方式

(2). 入出力ポートによるデータの送受信方向，順序および長さの制御方式

(3). 入出力ポートと入出力レジスタとの間の固定タイムスロット割当て共通バス結合によるデータ入出力制御方式

について述べた．

第5章ではまず処理要素の全体的構造を示し，結合回路網に対する演算部の関

係を明らかにし，各部を回線接続機能，通信制御機能，演算実行制御機能，演算処理機能に分類した．次に高度に並列で非同期動作を行なうシステムにおける演算実行制御機能の効率よい実現方法としてデータ駆動制御が有効であることを示した．また演算処理機能においては高速化の要求を満たすために完全に逐次的な処理シーケンスである命令ストリームの概念を導入した．さらに，これらの各機能を実現するための各部の構成に関する詳細な検討を行ない，入出力レジスタ，実行可能ＰＭ管理レジスタ，ＰＭ管理装置，命令メモリ，引数レジスタ，演算装置の概念設計を行ない，処理要素の実現性を示した．

第６章では応用プログラムの記述に関し，まずシステムの構造に対応してプログラムも

①逐次処理レベル
②データ駆動レベル
③通信制御レベル
④回線接続レベル

の４レベルに階層化されることを述べ，次に各レベルにおける基本言語による記述方法を具体的に示した．本論文ではこれらに対するコンパイラについては触れておらず，これはシステムのインプリメント時の課題である．また，高級言語についてはその妥当性や記法，処理系についての検討は行なっておらず，本システムをより広範で複雑な応用に対して適用する際の重要な技術として今後の検討が必要である．

第７章では資源割付けのシステムソフトウェアに関する検討を行なった．資源割付けを論理的処理要素割付け，物理的処理要素割付け，回線設定の３段階に分け，まず論理的処理要素割付けについて，並列処理アルゴリズムの効率をできるだけ損なわないため同時刻に実行可能となりうるプログラムモジュールは異なる処理要素に割付け，資源の有効利用のため同時刻に動作する可能性のないプログラムモジュールを同一の処理要素に割り付けるという方針に従った割付け方法の

考え方を示した．

物理的処理要素割付けについては，初期割付けに対してシャフリングを繰り返すことにより改善を行なうという基本的な手順を述べ，シャフリングの評価関数を２つの論理的処理要素の入れ替えによる結合路使用数の減少という単純なものとすることでマクロ的な総回線長の減少を図るアルゴリズムを示した．回線設定についてはブロッキングを発生させないことを第一の目的とし，通過する回線数の期待値が大きい結合路を避けて通るという経路選択方法を経路選択の範囲の狭い回線から順に適用するアルゴリズムを示した．次に物理的処理要素割付けと回線設定アルゴリズムをいくつかの具体的な結合関係に対して適用することにより評価を行なった．

物理的処理要素割付けアルゴリズムの計算量は全処理要素数をＮ，全回線数をＬとおくと$O(LN^{1/2})$であり，64×64処理要素，10000回線で数十分程度である．割付け結果については，初期割付けがランダムに近く，各論理的処理要素に接続されている回線数のばらつきが通常のアプリケーションで考えられる程度の大きさである場合に対しては良好な結果が得られるが，初期割付けにおいて回線方向に偏りがある場合，これを最適化することができず，また接続されている回線数のばらつきが非常に大きい場合はこの接続回線数が大きい処理要素は中心部に集中し，接続回線数が小さい処理要素は周辺部に分散するため中心部に回線が集中することが明らかになった．これに対するアルゴリズムの改善点としては，

(a). 回線方向を考慮した初期割付け

(b). ｘ，ｙ方向の結合路の使用状況を考慮に入れた評価関数

(c). マクロ的な結合路の混雑度と接続回線数を考慮した評価関数

を導入することが考えられる．

回線設定アルゴリズムの計算量はｉ番目の回線のｘ方向，ｙ方向の距離をそれぞれx_i，y_iとおくと$O(\sum(2x_iy_i+x_i+y_i))$であり，64×64，10000回線程度の結合関係に対しては数十分～数時間程度であるが浮動小数演算器を用いれば大幅に短縮可能である．回線設定結果については与えられた物理的処理要素割付けに対してはほぼ最適に近い結果が得られており，改善点としては部分的な迂回

路の導入が考えられる．この他に，回線設定結果を物理的処理要素割付けにフィードバックする方法も検討の必要がある．しかし総合的には結合関係に規則性がある場合はこれを利用した物理的処理要素割付けを外部から与えることによりかなり広範囲な結合関係に対応できることが明らかになった．

第８章ではシステム全体の性能評価を行なった．まずデータ転送に伴い通信制御機能と回線上の伝送遅延において発生するオーバヘッドを求めた．この結果，データ長をＬバイト，入出力ポート数をＰ，回線が経由する処理要素数をＮとおくと合計のデータ転送遅延は((２Ｌ－１)Ｐ＋８Ｌ＋Ｎ＋７)クロックとなることが明らかとなった．

また演算実行制御に伴うオーバヘッドはプログラムモジュールの実行可能性の検出に３クロック，さらに命令ストリームの実行開始までには引数の数をＡとおくと(３Ａ＋９)クロックとなることが明らかとなった．

次に各処理要素の基本的な演算能力の検討を行なうために実際に演算を行なうサブ命令ストリームを記述し，表８－１に示したような結果を得た．

最後に実際の問題に対するシステム全体の処理性能を明らかにするために，いくつかの実際の物理的処理要素割付けの処理時間を推定した結果，表８－２に示したような性能が得られることが示された．この結果から本システムはデータ依存関係および演算内容がそれぞれ単純なものから複雑なものまで，さまざまな応用に対して高い処理能力を示すことが明らかになった．

以上に述べたように，高度の並列処理をある程度広範囲な応用分野に提供するための並列処理アーキテクチャとして，２次元格子状配置の静的な回線接続方式が有効であることが示された．

今後このようなアーキテクチャを持つ並列プロセッサを実現するためにはさらに次のような検討を重ねる必要がある．

ｉ．応用に対するより詳細な検討を加えることにより処理要素各部に要求される記憶容量を明らかにする．

ii．基本言語処理系および高級言語の記述と処理系の設計を行なう．

ⅲ．使用可能な回路技術と実装方法から各処理要素の結合路数を決定し，結合路トポロジーを定める．

ⅳ．プログラムの作成，デバッグを支援するために必要な機能を検討し，実現方法を明らかにする．

このような検討を重ねることにより本システムが実現可能となるが，実用化にはさらに利用形態の検討が必要であり，いかにして多数の一般ユーザの使用に供するかといった問題も解決しなければならない．また各種の専用マシンとの使い分けも重要な課題である．

いかに素子技術が進歩しようとも，計算機の高性能化は最終的には並列処理技術に依存するものであり，今後ともより高性能でより柔軟性の高いシステムの研究が続けられる必要があり，これにより計算機の応用分野も飛躍的に拡大することが期待される．

参考文献

[Arvind 1982]

Arvind, Gostelow, K. P. : "The U-Interpreter", Computer, Vol. 15, No. 2, 1982, pp. 42-49.

[Batcher 1982]

Batcher, K. E. : "MPP: a supersystem for satellite image processing", AFIPS Conf. Proc. 1982 NCC, 1982, pp. 185-191.

[Bouknight 1972]

Bouknight, W. J., Denenberg, S. A., McIntyre, D. E., Randall, J. M., Sameh, A. H., Slotnick, D. L. : "The Illiac IV System", Proc. IEEE, Vol. 60, No. 4, 1972, pp. 369-388.

[Briggs 1979]

Briggs, F. A., Fu, K., Hwang, K., Patel, J. H. : "PM^4-A Reconfigurable Multiprocessor System for Pattern Recognition and Image Processing", AFIPS Conf. Proc. 1979 NCC, 1979, pp. 255-265.

[Broomell 1983]

Broomell, G., Heath, J. R. : "Classification Categories and Historical Development of Circuit Switching Topologies", Computing Surveys, Vol. 15, No. 2, June 1983.

[Dennis 1980]

Dennis, J. B. "Data Flow Supercomputers", Computer, Vol. 13, No. 11, 1980, pp. 48-56.

[Ericsson 1983]

Ericsson, T., Danielsson, P. E. : "LIPP-A SIMD Multiprocessor Architecture for Image Processing", Conf. Proc. 10th Annual International Conference on Computer Architecture, 1983, pp. 395-400.

[Fisher 1983-1]

Fisher, A. L., Kung, H. T., Monier, L. M.:"Architecture of the PSC: A Programmable Systolic Chip", Conf. Proc. 10th Annual International Comference on Computer Architecture, 1983, pp. 48-53.

[Fisher 1983-2]

Fisher, A., Kung, H. T.:"Synchronizing Large VLSI Processor Arrays", Conf. Proc. 10th Annual International Comference on Computer Architecture, 1983, pp. 54-58.

[Gostelow 1979]

Gostelow, K. P., Thomas, R. E.:"A View of Dataflow", AFIPS Conf. Proc. 1979 NCC, 1979, pp. 629-636.

[Gostelow 1980]

Gostelow, K. P., Thomas, R. E.:"Performance of a Simulated Dataflow Computer", IEEE Trans. Computers, Vol. C-29, No. 10, 1980, pp. 905-919.

[Hama 1984]

濱:"再構成可能なアレイ状MIMD型プロセッサの割付け方式", 東京大学工学部電気工学科卒業論文, 1984. 3.

[Jones 1980]

Jones, A. K. and Schwarz, P.:"Experience Using Multiprocessor System - A Status Report", ACM Computing Surveys, Vol. 12, No. 2, pp. 121-165(1980).

[Kambe 1982]

Kambe, T., Chiba, T.:"A Placement Algorithm for Polycell LSI and its Evaluation", Proc. 19th Design Automation Conf., IEEE 1982, pp. 655-661.

[Karaki 1984]

唐木:"スーパコンピュータ S-810における数学ライブラリの性能", センターニュース, 東京大学大型計算機センター, Vol. 16, No. 4, 1984, pp. 23-34.

[Kohara 1984]

小原:"階層構造のMIMD型スーパコンピュータ", 情報処理, Vol. 25, No. 5, 1984, pp. 480-490.

[Koike 1982]

小池, 大森, 近藤, 佐々木:"論理シミュレーションマシン", 信学技報EC82-42,

1982.10,pp.35-42.

[Kozdrowicki 1980]

Kozdrowicki,E.W.,Theis,D.J.:"Second Generation of Vector Super-computers",Computer,Vol.13,No.11,1980,pp.71-83.

[Kung 1982]

Kung,H.T.:"Why Systolic Architectures?",Computer,Vol.15,No.1,1982, pp.37-46.

[Mead 1980]

Mead,C.A.,Conway,L.A.:"Introduction to VLSI Systems",Addison-Wesley, Reading,Mass.,1980.

[Nguyen 1982]

グェン,浅田,大山,斉藤,猪瀬:"環状結合回路網を用いた分散型データフロー計算機の一方式",信学論,Vol.J65-D,no.12,1982.

[Nikkei-E 1983.4]

"速さを競うスーパコンピュータ",日経エレクトロニクス,1983.4.11,pp.105-184.

[Nikkei-E 1983.5]

"最大性能1.3GFLOPS,マシンサイクル6nsのスーパコンピュータ",日経エレクトロニクス,1983.5.9,pp.74-76.

[Nikkei-E 1984.4]

"画像処理分野をねらったデータフロー型プロセッサLSI",日経エレクトロニクス,1984.4.9,pp.181-218.

[Nikkei-E 1984.11]

"最大性能1.3GFLOPS,マシン・サイクル6nsのスーパコンピュータＳＸシステム",日経エレクトロニクス,1984.11.19,pp.237-272.

[Oyama 1984.1]

大山,グェン,ビノッド,斉藤,猪瀬:"分散形データフロー計算機の構成と実験的評価",情処論,Vol.25,No.1,1984,pp.101-108.

[Potter 1983]

Potter,J.L.:"Image Processing on the Massively Parallel Processor",

Computer, Vol. 16, No. 1, 1983, pp. 62-67.

[Sakaue 1984]

坂上, 木戸出:"イメージプロセッサの最近の動向", 信学誌, Vol. 67, No. 1, 1984, pp. 90-98.

[Schaefer 1982]

Schaefer, D. H., Fischer, J. R.:"Beyond the Supercomputer", Spectrun, March 1982, pp. 32-37.

[Snyder 1982]

Snyder, L:"Introduction to the Configurable, Highly Parallel Computer", Computer, Vol. 15, No. 1, 1982, pp. 47-56.

[Takahashi 1984]

高橋, 雨宮:"超高速科学技術計算向きデータフローアレイ計算機のアーキテクチャ", 信学論(D), Vol. J67-D, No. 1, 1984, pp. 62-69.

[Tokura 1983]

都倉, 萩原:"VLSIの行列計算法", 情報処理, Vol. 24, No. 4, 1983, pp. 558-562.

[Umeo 1982]

梅尾, 菅田:"一斉射撃問題に基づいたセル空間上での発火周期関数の特徴付け", 信学論, Vol. J65-D, No. 10, 1982, pp. 1227-1234.

発表文献

論文誌

[1] "環状結合回路網を用いた分散型データフロー計算機の一方式",信学論(D), Vol.J65-D,No.12,1982.

[2] "分散形データフロー計算機の構成と実験的評価",情処論,Vol.25,No.1, 1984.

[3] "分散形データフロー計算機の待ち行列モデルを用いた理論解析",信学論(D),Vol.J67-D,No.4,1984.

[4] "分散形データフロー計算機のシミュレーションによる設計と評価",情処論 Vol.25,No.5,1984.

国内大会

[5] "分散形データフロー計算機の制御機能",昭和56年信学全大,情報・システム部門510,1981.10.

[6] "分散形データフロー計算機の通信機能",昭和56年信学全大,情報・システム部門511,1981.10.

[7] "分散形データフロー計算機(ＥＤＡＣ)の待行列モデルによる理論解析", 昭和57年信学全大,電子計算機Ａ部門1414,1982.3.

[8] "データ駆動制御マルチプロセッサシステムの各種のネットワーク構成に関する検討",昭和57年信学全大,伝送方式部門309,1982.8.

[9] "分散形データフロー計算機の構成に関する検討",昭和57年情処全大,2F-2, 1982.10.

[10] "試作分散形データフロー計算機(ＥＤＡＣ)の処理ノードの評価",昭和58年信学全大,電子計算機Ａ部門1528,1983.4.

[11] "アクタ割り付けアルゴリズムの評価",昭和58年信学全大,電子計算機Ａ部門1529,1983.4.

[12] "On the Effective Organization of the Control Unit in a Distributed Data-Flow Computer",昭和59年情処全大,4F-11,1984.3.

[13] "再構成可能な結合回路網を持つＭＩＭＤ型プロセッサ,昭和59年情処全大,
5B-8,1984.9.

研究会

[14] "分散型データフロー計算機のプログラムモデルを用いた性能評価",信学技
報EC80-14,1980.

[15] "再構成可能な結合回路網を持つＭＩＭＤ型プロセッサ構成方式",信学技報
EC84-1,1984.5.

謝辞

学位論文を書き終えるにあたり，修士課程の2年間および博士課程の3年間にわたりご指導を賜わりました猪瀬博教授ならびに斉藤忠夫助教授に心より感謝の意を表します．

また日頃の研究生活においてご協力を戴いた，猪瀬・斉藤研究室の職員の富山忠宏氏，坂田彰一郎氏，もと大学院生および現大学院生の浅田邦博氏，安達淳氏，荻野長生氏，谷中一寿氏，本山崇三郎氏，グェンニュット氏，加藤聰彦氏，堀浩一氏，橋爪宏達氏，小林進一氏，和田正裕氏，越田一郎氏，ヴィノッド・プラサッド・シュレスタ氏，堀本徹氏，斉藤隆氏，村上和隆氏，村井俊雄氏，中島周氏，相原達氏，藤田悟氏，河合智明氏，計宇生女史，ならびに秘書の小林るり子女史，斉藤洋子女史に感謝いたします．

さらに研究の一部を分担して戴いた卒論生の濱利行氏，研究生活を楽しいものとしてくださった松原章雄氏に深謝いたします．

昭和59年12月22日