

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 Configuration of MIMD Type Processor with Reconfigurable Interconnection Network

大山 敬三

斉藤 忠夫

猪瀬 博

Keizo OYAMA Tadao SAITO Hiroshi INOSE

東京大学 工学部

University of Tokyo Faculty of Engineering

高度な情報処理の要求に対し、計算機の高性能化が必要となるが、逐次処理による性能向上には限界があり、飛躍的な進歩は望めない。また、柔軟性が高く、しかも効率の良い並列処理システムは未だ実現できる見通しが立っていない。一方で、すでに高い処理能力を実現し、また、実現できる可能性が高いと考えられるシステムに、多数のプロセッシングエレメント(PE)を結合回路網により結合した専用並列処理システムがある。しかし、これらは適応分野に限られ、柔軟性があまり高くない。また、新たな専用機を開発しようとするとは長期の開発期間を必要とする。

これらの問題を緩和するために、筆者等は、アルゴリズムに対応して結合形態を適応できるように結合回路網に柔軟性を持たせた、MIMD型のアレイ状プロセッサシステムについて提案しており、本稿では、このシステムのハードウェア構成について述べる。

1. はじめに

従来のコンピュータ(特にプロセッサ)を、その汎用性と処理能力をパラメータとしてプロットしてみると、図1のような分布になると考えられる。

最も汎用性の高いのは言うまでもなくフォンノイマン型の汎用機であり、世の中に多く出回っていて、技術も確立されているために、方式上の改良による性能の飛躍的な向上は期待できない。

汎用マルチプロセッサは、複数の汎用のプロセッサから構成され、なんらかの形で全てのプロセッサが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*, C.mmpなどがある^[1]。一般に、このような並列処理システムでは、プロセッサ台数が増え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。

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に対抗する汎用性を持っているが、入出力等にまだ解決すべき点が残る^[2]。

ベクトルプロセッサは、最近、スーパーコンピュータとして脚光を浴びている計算機の中核をなすプロセッサである。しかし、スーパーコンピュータの汎用性はスカラプロセッサに依るもので、ベクトルプロセッサはベクトル化された浮動小数演算専用の単能プロセッサである。これは、演算のパイプライン化による高性能化である^[3]。

アレイプロセッサはSIMD型の処理を行ない、行列演算、画像処理、偏微分方程式の数値解法などさまざまな分野において専用機として提案されている。最近、シストリックアレイと

呼ばれるアレイプロセッサが提案され、注目されている。これは行列演算、データベースの関係演算、ソーティングなどを、2方向ないし3方向のデータのストリーム間の繰り返し演算として実現するハードウェア・アルゴリズムを、アレイプロセッサの構造にマッピングしたもので、完全な専用演算装置である^[4]

この図を、汎用性に沿ってながめてみると、専用機においては新しい提案もあり研究も盛んである。一方、データフロー計算機では汎用化の研究が進められている。しかし、これらに挟まれた部分はあまり研究もなく、欠落している。データフロー原理を専用機に用いようという例もあるが、アプリケーションの特質も生かしく、データフロー原理に厳密に従うことも難しいので、あまり有望ではない^[5]

筆者等が提案しているシステムはこの空隙を埋める物として位置付けられる。

専用機としては、画像処理、信号処理、行列演算、数値解析等の分野を対象としたものが提案されているが、これらはいずれも処理の対象とするデータや、処理におけるデータ依存性の形がそれぞれ定まっており、それに基づいて、構造も固定されているため、他のアプリケーションに適應することが困難であったり、非効率のであったりする。

本システムは、プロセッシング・エレメント(P E)と結合回路網(I N)に柔軟な構造を

持たせ、各アプリケーションに適應可能とすることにより、ある程度広範囲の問題に対して専用処理装置の機能を提供する。汎用システムで高い処理能力を実現するのは困難であるので、専用機的な使用形態を仮定し、定型的で大きな処理量を要する、並列性の高いアプリケーションを対象とする。同一のP Eを多数用いることにより、P Eの利用効率よりは並列動作数を高めることで処理性能の向上を図る。

専用であるとは次のような条件を意味する。

- a. 基本的にはシングルプロセスである。
- b. アーキテクチャを考慮したアルゴリズムの変更・開発を必要としても構わない。
- c. 処理の実行のための初期設定にはある程度大きなオーバーヘッドも許す。
- d. ホスト計算機存在を仮定する。
- e. 比較的定型的な処理を高速で実行する。

I Nについては、従来は単純なP Eでもチップ数が多くなり、P E-P E間の配線長が長くなるために、データ転送のクロックレートがあまり高くできなかったが、VLSI化によりこれが改善されてきているため、演算処理と同程度の時間でデータの転送が可能になってきた。このため、従来は問題にならなかったような、汎用の通信回路網におけるスイッチング遅延や、経路選択、競合制御等による通信制御の複

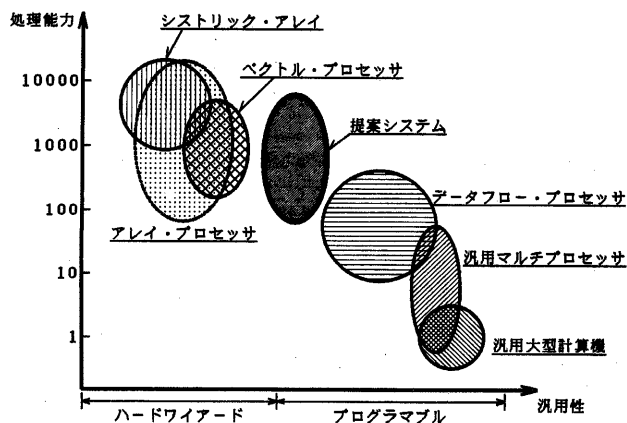


図1. 種々のプロセッサの汎用性と処理能力

雑化に起因するオーバーヘッドが、VLSIシステムでは全体の性能に大きな影響を与えるようになってきている。このレベルの結合回路網には、オーバーヘッドの大きい汎用の結合回路網は適当でない。

従って、パッチボードのように、プログラム実行前に必要なデータの経路をすべて接続しておく、回線接続方式にしてしまう方がシステムが簡単になり、通信オーバーヘッドも極めて小さくすることができる。

2. システムの概念

本システムの特徴は、

- ① プログラムの実行制御を2段階に階層化している。
上位：データ駆動制御
下位：逐次制御
- ② 上位プログラムの論理的なデータ依存関係を物理的な回線にマッピングすることにより、データ転送制御が簡素化され、スループットの向上と遅延の減少が期待でき、高度の並列処理が実現できる。
- ③ 下位プログラムの逐次制御を徹底することにより、実行制御を単純にするとともに、乱れないパイプラインを実現することによりフォンノイマンボトルネックを解消する。
- ④ これらの特徴をいかすため、上位プログラムのデータ駆動制御を極力簡単にし、ハードウェア化して、オーバーヘッドを小さくする。

という点である。

INにはさまざまな形態が考えられる^[6]。しかし、例えば全PEを多段のスイッチ回路網で結合しようとする、図2の様に、PEとINは別々に構成されることになり、配線量が大きくなるため、PE数の上限が低く抑えられてしまう。これはスイッチマトリクス回路網やバスによる結合でも同様である。

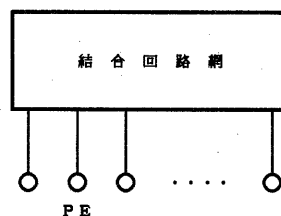


図2. PEとINが分離された構成

これを避ける構成に、階層構成とメッシュ状構成がある。階層構成にはさらにさまざまな形態があり、アプリケーションのデータ依存性により、適・不適があり、また、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ず、本システムの目的に適合する形態を見出すのは難しい。

そこで、本システムではメッシュ状の構成を採用する。メッシュにも何種類かあるが、実装の容易さを考慮し、2次元格子状のメッシュを変形した構成とする。メッシュ構成は、PE間の配線長、配線面積とも小さく、VLSIシステムに向いている。システム全体の大きさもPE数に比例し、大規模システムが容易に作れる。INの機能も各PE内に存在するため、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。

システムの全体の構成を図3に示す。2次元格子状あるいはトーラス状に配置されたPEと、それらの間をむすぶ信号路により構成される。

信号路の具体的な結合方法については6.で

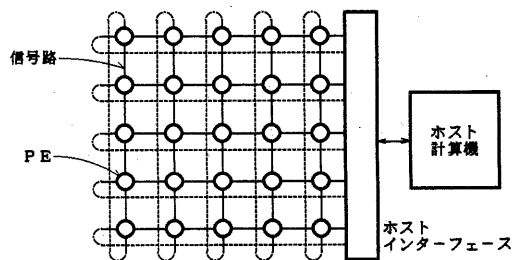


図3. システムの全体構成

述べる。

各 P E は、図 4 に示すように、処理部 (P U) から結合回路網 (I N) へのデータの出入口であるポート、および、隣接 P E 間の信号路の相互の接続を行なうためのスイッチマトリクスを持つ。

仮定として、各 P E は V L S I 1 チップ上に実現されているとする。

P E の処理部の構成を図 5 に示す。I N の高速性を活かすよう、実行制御のオーバヘッドを極力小さく抑えた構成となっている。

基本的な動作としては、上位プログラムの処

理単位である処理プログラムモジュール (P M) 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おり、データ駆動原則に従い各レジスタが読み出し・書き込み可能になると、P M 管理装置により命令メモリにその P M 番号が渡され、命令メモリから演算装置に対して順次命令ストリーム (通常のフォンノイマン型プロセッサの機械語命令、あるいはマイクロ命令に相当する) が送られる。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。入出力ポートと入出力レジスタとのデータの転送は入出力制御装置が制御を行なう。

各部の詳細な構造および動作については後述する。

プログラムの記述は各処理プログラムモジュールの内容と、処理プログラムモジュール内のデータの流れについて行なうことになる。処理プログラムモジュールを、P E に割り付け、各物理回線を設定するための経路割付けを行なうことによりプログラムが完成する。

下位プログラムレベルでは、ジャンプやサブルーチンコールのような制御構造は排除し、上位プログラムレベルでサポートする。

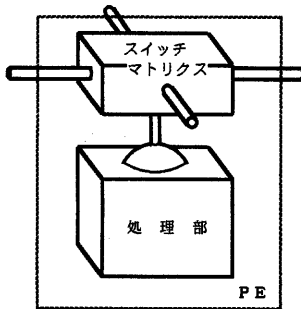


図 4 . P E の構成

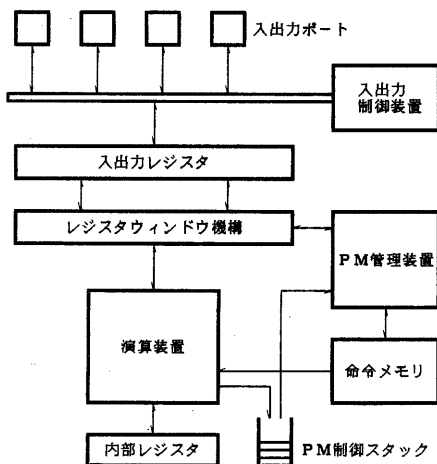


図 5 . P E の処理部の構成

3 . 通信方式

P E 間の通信方式にはバケット交換と、空間分割および時間分割の回線交換が考えられる。しかし、図 3 の様な構成で動的に回線交換を行なうことは実際上不可能であるので、回線交換においては処理の実行中は回線を固定し、これを回線接続と呼ぶこととする。以下では、本システムに適した方式の選択を行なう。

3 - 1 . データ転送遅延

バケット交換と回線接続の両方式のデータ転送遅延を簡単に評価する。

バケット交換

バケット交換では、待ち時間を無視し、バッファ制御やアクノリッジを考えないと、

- ・データバケット長は、データ L バイト、ヘッダ 3 バイト、合計 (L + 3) バイト

程度，

- ・ 1 P E 当たりの転送のオーバーヘッドは，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，合計 5 クロック程度

となる．ヘッダにはデータ長やアドレス情報が含まれる．各信号線のデータビット幅を 8 ビット，平均の転送 P E 数を N と仮定すると，1 パケット当たりの合計の遅延 D_p は，

$$D_p = 5N + L + 3 \quad \text{クロック} \quad (1)$$

となる．

空間分割回線接続

空間分割の回線接続では，転送はビットシリアル，制御ビットに 2 ビット使うとして，1 データ当たりの合計の転送遅延 D_s は，

$$D_s = N + 8L + 2 \quad \text{クロック} \quad (2)$$

となる．ただし，後述の様に，各 P E ごとに信号同期のために 1 クロックの遅延があると仮定している．

時間分割回線接続

時間分割の回線接続では，転送はデータ 8 ビットと制御線 1 ビットをパラレルとし，8 回線を多重化すると仮定すると，経路の設定が最適であれば，1 P E を経由することの遅延は 1 クロックですむので，1 データ当たりの合計の平均転送遅延 D_t は，

$$D_t = N + 8L \quad \text{クロック} \quad (3)$$

となり，空間分割の場合とほとんど差は無い．ただし，最適なスイッチの設定が必ずしも可能とは限らないので，実際は回線が混雑すると，式③の第 1 項の係数が大きくなる．

パケット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 10 MHz 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，通常の MOS 技術でも 20 MHz 程度のクロックレートは得られる．

D_p と D_s の比較を図 6 に示す．この図から明らかな様に，パケット交換は長いデータを近くの P E に送るのに適し，回線接続は短いデータを遠くの P E に送るのに適している．

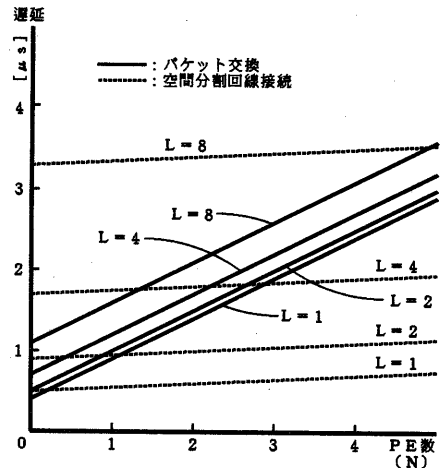


図 6．パケット交換と回線接続の遅延特性

3-2．ハードウェア機能

以下に，両方式において必要とされる機能を示す．

パケット交換では，

- ・ バッファリングとバッファ制御
- ・ 送受信制御とハンドシェーク
- ・ 経路選択とスイッチング

等が必要である．このうち，バッファリングが最も大きなハードウェア量を必要とする．各入力信号路対応に最低でも 1 パケット分のバッファが必要である．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．また，送信制御では，出方路が同じパケットが複数の入力から到着した場合，定めら

表 1 . パケット交換と回線接続の特質の比較

	パケット交換	回線接続
オーバーヘッド	大 : ヘッダ、ハンドシェーク、バッファリング等	小 : 同期ビット程度
ブロッキング	なし : 任意 P E 間で通信可能	あり : 隣接 P E 間のリンク数は物理的な回線数で制限される
スイッチング	動的 : 各 P E ごとに行なう	静的 : 実行開始前にホストで行なう

れた規則に従い、競合制御を行ない、順にパケットを送出せなければならない。パケット長の管理も必要であり、かなり複雑な処理を要するため、通信処理部に必要となるハードウェア量が大きくなる。

一方、回線接続では、

- ・スイッチマトリクス、または、時間分割スイッチ
- ・スイッチ制御レジスタ

等が必要である。これらは構成が単純であり、ハードウェア量もパケット交換に比べると小さくなる。

3-3. その他の特質

上記の他に両方式の特質として表 1 のようなものがあげられる。

パケット交換では、アドレス情報やパケット長等のヘッダがデータ以外に付加され、これがオーバーヘッドとなる。回線接続では同期のためにスタートビットが必要になるだけで、その他には本質的にオーバーヘッドは無い。

スイッチングは、パケット交換では各 P E 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。

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 P E 間では全て回線を接続しておく必要があるが、互いに隣接する P E 間の回線数は、パッケージのピン数により、高々十数本に制限されるため、特定の P E 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つたりすると、

回線の設定ができない可能性がある。これは、通常回線交換でいうところの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呼ぶことにする。パケット交換では、このようなブロッキングは存在しない。このため、回線接続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、処理のモジュールへの分割、処理モジュールの P E への割付け、P E 間のリンクの物理的回線への割付けをそれぞれ最適化する必要がある。

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。また、VLSI 技術の進歩により集積度があがったとはいえ、ハードウェア量や制御構造の複雑さは小さい方が良く、この点からも回線接続が望ましい。

また、回線接続において、空間分割では双方向の通信が可能であるのに対して、時間分割では同一の信号路は片方向にしか使えない。回線の多重度を増やすと、ブロッキングは減るが、遅延が大きくなる。さらに、時間分割ではフレーム同期の問題もあり、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を採用する。

4. 同期方式

P E 間の通信のための信号同期について以下に述べる。

通信は回線接続で行なうため、両端間で同期が取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、両端間のみで同期を取るのには難しい。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、各中継PEごとに同期を取ることにする。

この方式では、PEを経由することにより1クロックの遅延を生ずることになる。しかし、クロックは隣接PE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。

隣接PE間のクロックのずれの最大値を T_c 、信号の伝搬遅延を T_d とすると、信号のクロック周波数 f は

$$f \leq \frac{1}{2(T_c + T_d)} \quad (4)$$

で抑えられる。

5. INのトポロジカルな構成

INは、各PE内のスイッチマトリクスと、各PE間を接続する信号路とからなる。

スイッチマトリクスは、PE内の処理部(PU)の出力ポートおよび隣接PEからの入力信号路を、入力ポートおよび出力信号路へ接続する。

各方向に16本ずつの入出力信号路と、16個の入出力ポートがある場合、完全接続とするためには、 16×64 のスイッチマトリクスが4個必要である。

2. で述べたように、PEの配置は2次元格子状のメッシュの変形とする。メッシュでは通常、格子上の距離が1である様なPE間に信号路を張るが、本システムでは、距離が2以上のPE間にも信号路を張ることとする。

その理由は、接続回線が経由するPE数を減らすことにより、同期のための伝送遅延が減り、同時に、PEの入出力信号路数を一定とした時の接続可能な回線数が増えることにある。

変形の例としては、図7の様な信号路の張り方が考えられる。

また、回線設定の自由度を増すため、各回線は、半二重の双方向伝送を行なう。

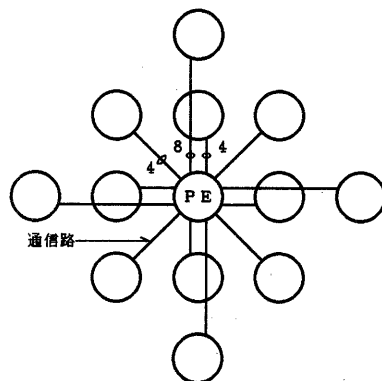


図7. 通信路の張り方

6. 入出力制御

PUからのINへの入出力は、全て入出力レジスタを介して行なう。入出力レジスタの状態は、レジスタウィンドウ機構により監視され、プログラムモジュールの実行制御にも用いられる。

入出力ポートは、データのバラレルーシリアル変換、双方向半二重転送制御のほか、1本の物理回線上を流れる複数のデータの順序制御を行なう。

1本の回線上を流れるデータは、論理的に順序が一定であることが保証されていなければならない。送信・受信の別、および、そのデータのレジスタ番号とデータ長の表が、送信および受信ポートの双方に蓄えられていることが必要である。この条件が満たされれば、回線にはデータ以外の余分な情報を一切流す必要がなくなる。アクノリッジは、空のデータの転送という形で扱うことにする。

この表に従い、入出力ポートは、受信の場合は到着したデータを指定されたレジスタに書き込み、送信の場合は指定されたレジスタからデータを読み出して送信する。

入出力レジスタは入出力ポートと演算装置からアクセスされる。さらに入出力ポートは16～32個位と考えているので、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うやって入出力ポートが検出するかが問題である。

同様に、入出力レジスタにデータが書き込まれた時に、入出力ポートが送信可能なら問題ないが、送信不可の場合は送信可能になるまで待つ必要がある。やはり、入出力レジスタにデータが書き込まれたことをどうやって入出力ポートが検出するかが問題となる。

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

入出力ポートか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、数の上からゆくと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

等が考えられるが、実現の容易さ、デッドロックの危険防止、高速性を考慮して、①を選択する。(②、③は低負荷時は効率がよいが、高負荷時は①がよい。入出力ポートが入力と出力に半々に使われているとすると、出力の時は出力レジスタのセンスが頻繁に起こることになり、負荷が50%近くになると考えられる。)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。

一方、演算装置が入出力レジスタにアクセスするときは、入出力レジスタは必ずアクセス可能であるはずなので(アクセス可能でなければPMは実行可能にはならない)、入出力ポートよりも優先させる。

レジスタの状態フラグは、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→リセット、書き込み→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3段のバイプライン動作をさせることとし、

- 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read/write信号を出す。
- ③ 番号を指定されたレジスタは状態フラグを返し、入力ならポートからレジスタに、出力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ーラップさせることにより、20M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。

7. 実行可能PM管理

本システムでは、2.で述べたように、プログラムの実行制御を2段階の階層構造としている。多くのマルチマイクロプロセッサシステムも何らかの形で階層構成となっているが、下位レベルはフォンノイマン型の制御であるのが普通である。しかし、本システムでは、上位にある程度柔軟性のあるデータ駆動制御を採用しているため、下位レベルには、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よい。そこで、下位レベルでは、プログラムは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は取り除き、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、条件付き演算(それ以前の演算の実行結果により実行したりしなかったりする演算)を用いて上位レベルのプログラムモジュール(PM)の実行の制御を行なう。すなわち、条件に応じて、出力データを出力レジスタに書き込んだり書き込まなかったりすることにより、次に実行可能になるPMの選択をして、プログラムの制御構造を実現する。

上位レベルの逐次性については、データ駆動制御においても、データの依存関係により、暗黙的に記述できるが、これは効率が悪い。

また、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PM間の通信にはオーバーヘッドが大きくなりすぎる。

そこで、PMの逐次的な実行制御をもっと明示的に行なうための手段として、PM制御スタックを導入し、これを用いた tail jump 機構により、密な関連を持つPM間の実行制御を可能とする。

tail jump の基本的動作は以下のようになる。演算装置が、条件に応じて、現在実行中のPMの次に実行すべきPM番号をスタックに積む。命令ストリームが終わったら、PM管理装置は、スタックの先頭のPM番号により指定されたPMを選択して実行させる。これにより、PMの連続的な実行順序を制御する。命令ストリームの途中でその実行を中断し、他のPMを実行するようなことは無い。

データ駆動制御においては、PMが実行可能になるのは、

入力レジスタ：読み出し可能

出力レジスタ：書き込み可能

が満たされた時である。実行順序はスケジューラが決定する。

上述のように、スケジューラは、PM制御スタックによるPMの指定を優先する。すなわち、PM制御スタックが空でなければ、スタックからPM番号を取りだし、そのPMが実行可能でなければ実行可能になるまで待つて命令メモリに渡す。これにより、逐次的な入出力データやデータストリームも効率良く扱え、また、同一PE内に限り、サブルーチンと論理的に等価な制御構造の実現もできる。

PM制御スタックが空の場合、複数のPMが実行可能であつたら、何らかの方法でこれらのうちから1つだけ選択しなければならない。

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。1つのPMを実行中に次のPMを選択すれば良いので、数クロック程度のオーバーヘッドは問題にならない。どの方式も一長

一短であるので、ハードウェアの実現の容易さを考慮して、固定優先順位制御とするのが良いであろう。プライオリティ・エンコードを2段にすれば実現できる。

次に、PMが実行可能であることをいかにして検出するかが問題になる。これにはデータフロー計算機で行なっているように、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ような方法は、オーバーヘッドが大き過ぎてできない。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8のように実行可能PM管理レジスタをPMの最大数だけ置き、管理させれば良い。

実行可能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`の実行可能PM管理レジスタに対応するPM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フラグを反転して同様の処理を行なう。

このような機能をレジスタウィンドウ機構の中に作り込む。

前述のPM制御スタック、あるいはプライオリティ・エンコードにより選択されたPM番号を命令メモリに与えることにより、そのPMに対応する命令シーケンスの実行を開始させる。

命令メモリは下位プログラムの命令シーケンスの記憶媒体により、さまざまな構成が考えられるが、例えばRAMを使うことにすると、PM開始番地テーブル、アドレスカウンタと若干の制御回路により実現できる。

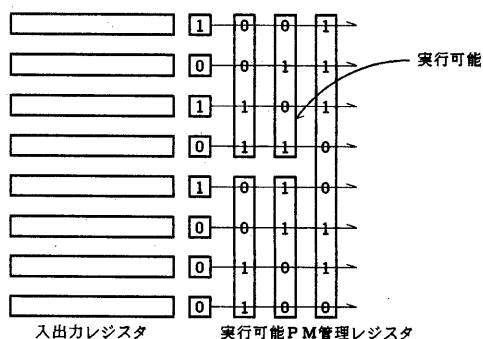


図8．実行可能PM管理方式

PM管理装置から与えられたPM番号からテーブルを引き、そのPMに対応する命令シーケンスの開始番地をアドレスカウンタにロードして、あとはRAMから演算コードを順次読み出しては演算装置に送る。終了コードが検出されたらPM管理装置に信号を送る。

全ての種類の演算が同一の時間で終了するように演算セットを設計しておけば、演算装置との間のハンドシェイクも不要となるので、制御は非常に簡単である。

演算装置はALUと簡単な制御回路で実現できる。

8. システム実現上の制約と問題点

提案システムのPEは全て均一な構造をしており、信号路の配線長はPE数に依存しないので、PE数は物理的な空間の大きさにより最終的な制約を受ける。

実際にPEの数に制約を与えるものとしては、

- ・プログラムロードのオーバーヘッド
- ・データ分配および回収のオーバーヘッド
- ・クロック、その他の共通信号の分配

等がある。また、いくらPE数を増やしても、それに見合うだけの並列性を持ったアプリケーションが無ければ役に立たない。

システムを実現する上での問題点は、ハードウェアに近い方から、

①クロックの分配

システム全体に同一のクロックを与える際の遅延をいかにして抑えるか。
→隣接PE間の位相のずれをある範囲内に収めれば良いので、可能である。

②サイクリックなトポロジーの実現

トラス状にするためには基板を立体的に実装しなければならない。

③外界とのインターフェース

ホストからの入出力、カラーモニタへのビデオ信号の供給、ディスクからの直接入出力等の汎用のインターフェー

スの定義が必要である。

④プログラムロード方法

セットアップ時間の短縮のためにも、またシステムの大規模化のためにも効率の良いプログラムロードが必要である。

⑤割付けアルゴリズム

プログラムモジュールのPEへの割付けと、PM間のデータ依存性のマッピングを自動的に行なうか、あるいは人手で行なうのを支援する道具が必要である。

⑥下位、および上位プログラムの記述法

下位プログラムは逐次的なので従来のプログラミング言語と大差ない。

上位プログラムはデータの依存性を表現するので、規則的な場合は式で表現できるだろうし(例えばFFT)、不規則な場合は羅列的な記述になるだろう(例えば待ち行列系シミュレーション)。これらの統一的な記述法が必要である。

また、これらとは独立に、信頼性、保守性についても検討する必要がある。

9. おわりに

さまざまな分野において、専用処理装置的に用いることを目的とし、高速でかつ柔軟性を持った結合回路網と、2段階の階層的実行制御構造を持った演算装置により構成される、MIMD型のプロセッサレイについて提案し、そのハードウェア上の構成を述べた。ソフトウェア面については、現在検討中であり、別の機会にご報告する予定である。

< 参考文献 >

- [1] Jones, A. K. and Schwarz, P.: "Experience Using Multiprocessor System - A Status Report", ACM Computing Surveys, Vol. 12, No. 2, pp. 121-165 (1980).
- [2] 大山, ゲン, 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", 情報処理, Vol. 25, No. 1, pp. 101-108 (1984).
- [3] 田中善一郎: "500MFLOPS級の商用スーパーコンピュータが稼動へ", 日経エレクトロニクス, No. 314, pp. 108-126 (1983. 4. 11).
- [4] 坂村, 石川: "VLSI コンピュータアーキテクチャ", 情報処理, Vol. 24, No. 2, pp. 156-175 (1983).
- [5] 松本他: "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, No. 340, pp. 181-218 (1984. 4. 9).
- [6] Siegel, H. J., McMillen, R. J. and Mueller, P. T.: "A Survey of Interconnection Methods for Reconfigurable Parallel Processing System", Proc. of National Compt. Conference, pp. 529-542 (June, 1979).