計算機アーキテクチャ　71－4
(1988. 7. 7)

並列処理システムＶＰＰのアーキテクチャ

An architecture of parallel processing system VPP

真鍋　俊彦　　井上　淳　　前田　明

Toshihiko MANABE　Atsushi INOUE　Akira MAEDA

（株）東芝　総合研究所

Toshiba Corp. R&D Center

あらまし　我々は画像処理を主目的とした並列処理システムＶＰＰ(Variable Processor Pipeline)を開発している。ＶＰＰは多数のベクトル処理を基本とした単位プロセッサＰＵと高速な結合部から構成されており、ＭＩＭＤ方式で制御されるマルチプロセッサ型の並列処理システムである。また、ベクトルプロセッサＰＵには配列のインデックスを陽に操作する機能を付加し、条件分を含むＤＯループのベクトル化、画像のテーブル変換の高速化を実現している。ここでは、ＶＰＰのアーキテクチャとＰＵの演算パイプラインの整合性を保つための同期方式について説明する。

Abstract　We are developing a flexible parallel processing system VPP (Variable Processor Pipeline) which mainly aims at effective image processings. It consists of many element processor PU's (Processor Unit) and a high speed connection network. Each PU is a kind of vector processor and executes programs in its local memory based on an MIMD manner. PU is able to handle indices of arrays explicitly and easily vectorizes DO-Loops including conditional statements and table look up operations in image processings. In this paper, we describe the architecture of VPP and flexible synchronization mechanism to keep consistency of pipelined operations in PU.

１．はじめに

画像処理を始め、様々な分野で高速処理の要求が高まっている。我々は、通産省工業技術院の大型プロジェクト「科学技術用高速計算システムの研究開発」の一環として、画像処理を応用の一つとする高機能並列処理システムＶＰＰ（Variable Processor Pipeline）の研究開発を進めてきた。ＶＰＰは多数のベクトルプロセッサＰＵ（Processor Unit）を非常に高速な結合部で結合した、ＭＩＭＤ方式の並列計算機である。　現在、我々は、ＰＵの主要部分をゲートアレイ化し一枚の基板にまとめ、このＰＵ８台を用いたＶＰＰパイロットモデルの開発評価を行っている。

本稿では、前半でＶＰＰ全体の構成について、後半ではＰＵの演算パイプラインの整合性を保つための同期方式に焦点を当てて説明する。

２．ＶＰＰの構成

ＶＰＰは、図１の構成のように、ベクトルプロセッサＰＵ、画像メモリ、制御プロセッサＣＰを高速な結合部で結合したマルチプロセッサシステムである。各ＰＵは、プログラムメモリ（ＰＭ）、データメモリ（ＤＭ）を有し、それぞれのプログラムに従い互いに協調しながら処

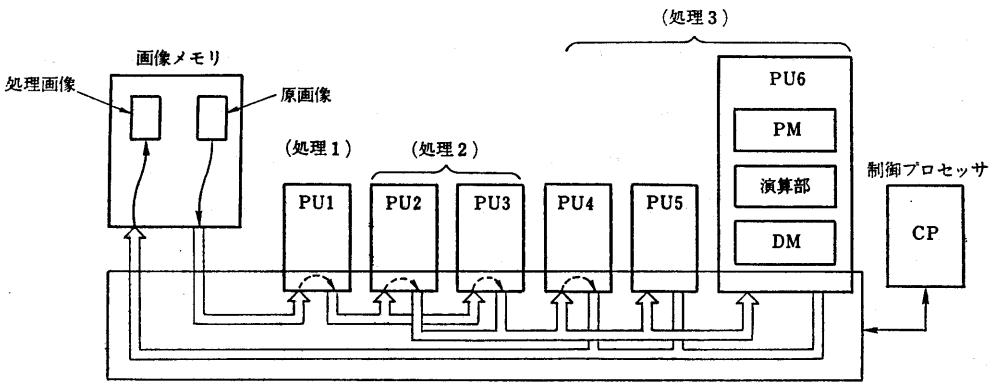

図１．ＶＰＰの構成

理を進めていく。各ＰＵはＣＰにより起動されるが、一旦起動された後はＰＭのプログラムに従い自律的に動作し、ＣＰとのやり取りは不要となる。

ＶＰＰでは、各ＰＵに種々の処理を、例えば処理１－処理３を分担させ、データをＰＵ間で転送しながら処理を進めていく方式を採用している。この場合、ＰＵを遊ばせないためには各処理の負荷の大きさに応じて、ＰＵ台数の割当を行う必要がある。例えば、処理１＜処理２＜処理３の順で負荷が重い場合には、図１に示すようにＰＵ台数を割り当てることが必要である。

３．結合部

図１に示したように処理に応じＰＵを割り当てる方式では、任意のＰＵ間でのデータ転送が必要となり、結合部に大きなデータ転送能力が求められる。ＶＰＰではこの結合部に、以下の特長を持たせている。

（１）競合のないデータ転送が実現できる。全ＰＵが同時にデータ転送を開始しても、転送先が互いに異なる限り、競合による待ち合わせは発生しない。

（２）各ＰＵは演算結果を直接（各ＰＵ内のＤＭにバッファリングすることなく）結合部を介して送ることが可能である。このため、データの転送時間は、その演算の時間とオーバーラップさせることができ、見掛け上陽には現れてこない。

（３）各ＰＵをいくつかのグループに分けそれぞれのグループ内でブロードキャスト転送ができる。すなわち、複数組のブロードキャスト転送を同時に実行することが可能である。

（４）結合部は転送に伴うオーバーヘッドを軽減するため、すべてブロック転送で行われる。

この結合部は図２のようになっており、各ＰＵに対し一つのＢＬ（Bus Logic）を対応させ、このＢＬを格子上に並べた構成を採っている。このＢＬは列方向、行方向にそれぞれS(Source)-loop、D(Destination)-loopによりループ状に結合されており、転送データは、S-loop -> 交点のＢＬ -> D-loopの順に転送される。これらのループ上には、そのループに接続されたＢＬに対応したスロットが回っており、データはこのスロットにより転送される。[1]

列方向のループには転送元に対応したスロットが、行

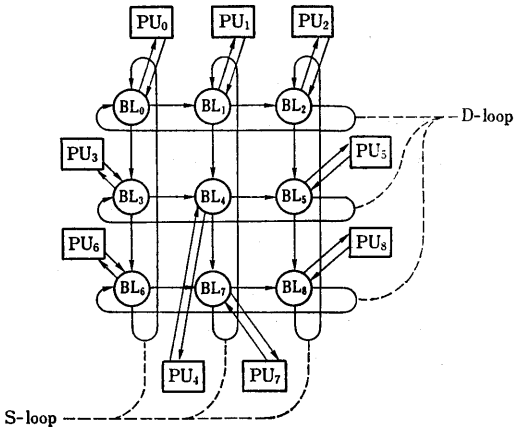

図２．ＶＰＰの結合部

方向のループには転送先に対応したスロットが回っているので、転送先が異なる限り任意の二つの転送間でスロットの奪い合いがなく、競合が発生しない性質を有している。

４．ＰＵの特長

ＰＵは図３に示すように、ＰＭとＤＭの他に、

（１）ＰＵ全体を制御するシーケンス制御部、

（２）ベクトル演算等でのパイプラインを制御するベクトル制御部、

（３）演算データのパスを制御するデータ制御部、

（４）ベクトル演算等でアドレスを生成するアドレス制御部

（５）ベクトル／スカラー演算を実行する演算部

から構成される。このＰＵの特長は次のとおりである。

（１）配列のインデックスの集合をインデックスセットとして定義し、陽に操作できる機能を持つ。このことにより、条件分岐や間接アドレス演算を含む処理を効率よく実行できるようにした。[2],[4]

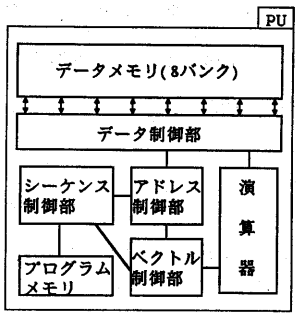

図３．ＰＵの構成

（２）シーケンス制御部とその他の制御部は並列に動作し、ベクトル演算実行中に、次のベクトル演算の準備を行うことができる。この結果ベクトル演算部の稼働率を高めることができる。[6],[7]

（３）画像データの二次元アドレスによりＤＭにアクセスできる。また、ＦＦＴ演算を効率よく実行する機構を内部に持っている。このように、内部に画像処理用の機構を備えている。[3]

（４）バンク選択方式を工夫することにより、ユーザがアクセス競合の発生を考慮せずにプログラミングをできるようにした。例えば、従来よくやられていた、アドレスの下位ビットによる選択であると、バンクの整数倍のアドレスでＤＭをアクセスすると、競合が多発するので、そのようなアクセスパターンを避けるプログラミングを心がける必要があった。本ＰＵでは、アドレスのビットをいくつかのグループに分け、各グループ内のビットの全体の排他的論理和をとり、その値を基にバンクを決めており、一つのバンクにアクセスが集中することを避けている。なお、ＤＭへのアクセスに関しては、生成されたアドレスを格納するためのアドレスレジスタを二重化することにより、たとえ競合が発生しても、それによる性能低下を小さくしている。

（5）パイプラインの空きを最小限に抑えるために、パイプラインレベルでの同期のための様々な機構を用意している。これらの機構により、演算間のデータ依存等による同期を細かく制御できるようにしている。

これらの特長の内、（5）については次節で説明し、ここでは（1）のインデックスセットについて簡単に説明する。

インデックスセットは、上記のように、条件分岐や間接アドレス演算を効率よく実行し、ベクトル演算の柔軟性を上げるために導入された。例として、条件分岐をインデックスセットを用いて実現した場合について示す。図4（a）は、条件分岐（if文）を含んだFortranプログラムの一部である。多くのベクトルプロセッサでは、図4（a）のようなプログラムはマスクベクトルやギャザー／スキャター機能により処理される。これに対して、PUでは、インデックスセットを用いて、図4（b）のように処理される。図4（b）中のII,IX,IYはインデックスセットを表現している。図4（b）中の文(sb1)は、図4（a）中の文(sa1)に対応し、II中のインデックスを条件により、IXとIYに分類している。条件を満たしたインデックスはIXに、満たしていないインデックスはIYに振り分けられる。新しく分類されたインデックスセットを用いて文(sb2),(sb3)に示すようにベクトル演算を実行できる。この分類により、不要な演算やデータの再構成を行うことなく、条件分岐の処理を実現できる。このインデックスセットの機能を容易に実現するため、および、PUの1ボード化のために、PUではベクトルレジスタを設けない、メモリーメモリ演算を採用している。

```
      DO 30 I=1,1000
      IF (D(I) .GT. 0) GOTO 20   (sa1)
      A(I) = B(I)                (sa2)
      GOTO 30
   20 A(I) = B(I) * C(I,I)       (sa3)
   30 CONTINUE
```

（a）FortranのDOループ

```
      II = (1:1000)
      IX,IY = IF(D(II) .GT. 0)   (sb1)
      A(IY) = B(IY)              (sb2)
      A(IX) = B(IX) * C(IX,IX)   (sb3)
```

（b）インデックスセットの例

図4．インデックスセットを用いた演算例

5．パイプラインレベルの同期制御

5．1　同期制御の必要性

PUでは、次の三つの場合にパイプラインレベルの同期が必要となる。

（1）近接する二つの演算の間にデータの依存があるとき

図5がこの場合の一例である。図5ではs1の演算結果をs2が参照しているので、s1の演算結果が格納されるまでs2の演算を開始できない。RISC方式のマイクロプロセッサ等では、この種の同期をソフトウェアでとっていた。しかし、ソフトウェアでこの種の同期を行うと、次の問題が生じる。

（a）図5の例をソフトウェアで解決すると、s1とs2の間にパイプラインを空にするための同期命令を

生成することになる。しかし、多くの場合ｓ１とｓ２はまったくオーバーラップできないわけではなく、ｓ２のオペランドのアドレス計算等はｓ１の演算結果が格納される前に実行可能であり、ｓ１とｓ２の間でパイプラインを空にすると演算能力の低下を招く。

（ｂ）（ａ）の問題を避けるために、コンパイラで命令の並び換えを行う手法が一般に採られている。データ依存のある二つの演算をパイプラインの段数より離すことができれば、そのデータの依存は考慮しなくてよい。ＲＩＳＣのようにレジスタ間演算を基本とするプロセッサでは、レジスタに着目してデータ依存の解析を行えばよいので、この手法は比較的有効である。しかし、ＰＵのようにメモリーメモリ演算を基本とするプロセッサでは、別名問題(aliasing problem)により正確なデータ依存の解析が困難なので、抜本的な解決にならない。

これらの問題を考慮し、ＰＵでは、ハードウェアで図５のようなデータ依存を検出し、その依存が意味をもつ間、後続の命令をパイプライン中で待たせる機構を導入した。その内容については、５．２節で説明する。

（２）シーケンス制御部とベクトル制御部が同期しなければならないとき

シーケンス制御部は、すでに述べたように、ＰＵ全体の制御を行っており、演算実行上の主な役割は次の通りである。

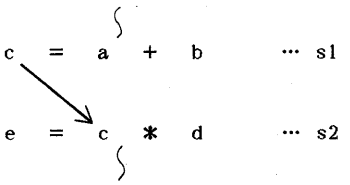

図５．単純なデータ依存の例

（ａ）配列要素等のアドレス計算を行う。例えば、ａ（ｉ，ｊ）をスカラー値としてアクセスする場合でのアドレスについては、シーケンス制御部　がｉとｊの値をＤＭから読み出し、それらとａのベースアドレスから計算する。この計算はシーケンス制御部内部の演算器を使用する。したがって、その前にｉやｊに値を格納する演算があれば、それが終了するのをシーケンス制御部は待たなくてはならない。ｉやｊに値を格納する演算は図３の演算部を用いて、ベクトル制御部の制御のもとで実行される。

（ｂ）スカラー演算での条件分岐やループ等の、命令の制御の流れを司る。この場合も（ａ）と同様に制御の分岐の判断の元となるデータを、シーケンス制御部がＤＭから読んでこなくてはならず、そのときに同期の必要性が生じる。

シーケンス制御部とベクトル制御部の同期の最も単純な方法は、上記の（ａ）や（ｂ）でシーケンス制御部がＤＭからデータをよむのを、そのときパイプラインの中で実行中のすべての命令が終了するまで待つやり方である。しかし、このやり方では、もちろん頻繁にパイプラインを空にしなくてはならず、全体の性能が大幅に低下する。そこで、ＰＵでは、さらに細かいレベルで同期のとれる命令を用意することにした。その内容は、５．３節で説明する。

（３）分類された結果のインデックスセットを参照するとき

図４（ｂ）の(sb1)はインデックスセットの分類の例で

あるが、そのような分類の結果を利用するには、分類が終っていなくてはならない。また、本ＰＵではベクトル演算を起動するとき、ベクトル長を指定する方式を採用しているため、分類結果のインデックスセットに何個のインデックスが集まったか調べておかなくてはならない（ＰＵは分類したインデックスを数えるレジスタを持っている）。さもなければ、(sb2)や(sb3)のような演算でベクトル長を設定できない。すなわち、(sb1)が終了するまで(sb2)や(sb3)を実行できない。ベクトル長を設定するのはシーケンス制御部であるので、ここでの同期の問題は、上記の（２）の特別な場合と考えることができる。そのため、この問題は５．３節の手法でも解決可能であるが、その他にこの問題特有の同期方式をＰＵに用意した。その内容については、５．４節で説明する。

５．２　自動パイプラインロック

５．１で説明したように、スカラー／ベクトル演算でのデータ依存のための同期は、ＰＵではハードウェアで自動的に行っている。パイプライン中で、同期の必要なデータ依存を検出するために、ＰＵのアドレス制御部の中にアドレス比較器を設けている。データ依存を検出するための機構を図６に示す。

図６のアドレスＡとアドレスＢは演算の対象となるデータのアドレスであり、アドレスＤは演算結果の格納先のアドレスである。これらの三つのアドレスは、同時に、ベースアドレス、増分、各々のインデックスをもとに生成される。パイプライン方式で演算を行っているので、アドレスＡとアドレスＢによりデータがメモリから読み出されてから、演算が終りアドレスＤの位置にデータが格納されるまで時間がかかる。その間アドレスＤは図６中のＦＩＦＯに格納されて、出番を待っている。したがって、アドレスＡ、アドレスＢとＦＩＦＯ中のアドレスが一致するか調べれば、パイプライン中でのデータ依存を検することができる。アドレスの一致を調べるためにアドレスＡ用とアドレスＢ用の比較器を設けている。どちらかの比較器が一致を検出すれば、アドレスＡとアドレスＢによるアクセスを待たせればよい。このことにより、パイプライン中でデータ依存が問題となる間、後続の命令は停止している。このように、パイプラインを自動的にロックすることから、この機構を自動パイプラインロックと呼んでいる。

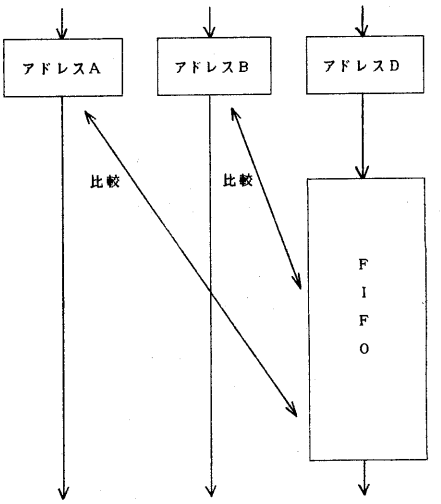

図６．自動パイプラインロックの機構

自動パイプラインロックは、図６のようなスカラー演算のデータ依存発生時の効率化だけでなく、巡回演算のベクトル実行時にも役立つ。巡回演算とは、

$$a(ix) = a(ix-1)*b(ix)+c(ix) \quad \cdots \quad (1)$$

のような演算のことである。（１）式は、一次の巡回演算である。通常、巡回演算はベクトル実行できない。最先端のスーパーコンピュータでも、たかだか一次の巡回演算がベクトル実行できるのにすぎない。ＰＵでは、自

動パイプラインロックにより、n次の巡回演算をベクトル実行できる。また、巡回演算か否か、実行時でないと判らないような場合、たとえば、

a(ix) = a(ix+n)*b(ix)+c(ix) … (2)

のような場合でも、ベクトル実行できる。

5．3 番号札方式の同期

番号札方式とは、演算命令一つ一つに順番号を与えることによって同期をとる方式である。番号札により命令単位の同期の制御が可能になっている。このことを利用して、シーケンス制御部は、着目している命令が終了したことを確認した後、DMにアクセスしにいく。

PUではこの機能を二つのカウンタ（発行番号用カウンタ、終了番号用カウンタ）により実現している。発行番号用カウンタは、パイプラインに投入される命令をカウントするもので、このカウンタ上の番号より若い番号に対応する命令は、パイプライン中で実行中か、終了しているかのいずれかである。終了番号用カウンタは、パイプラインから抜け出てきた命令をカウントするもので、このカウンタ上の番号より若い番号に対応する命令はすでに終了している。

これらのカウンタを用いた同期のやり方は以下のとおり。まず、同期をとりたい命令の前に発行番号用カウンタの値を読み取る。この値が、同期をとりたい命令に対応する番号札になる。つぎに、その命令の結果を参照する時点で、終了番号用カウンタを監視し、その値が読み取っていた番号札より大きくなれば、その命令は終了したと判断できる。終了したことを確認して、シーケンス制御部は演算の結果を読みにいく。PUの命令としては、発行番号用カウンタを読む命令と、指定した値より終了番号用カウンタの値が小さい間ｗａｉｔする命令を用意している。

5．4 VALIDビットを用いた同期

インデックスセットの分類の際の同期は、5．3の方式でも実現できるが、その他に、分類が終了したか否かを示すビット（VALIDビット）を用意し、それを用いても同期をとれるようにした。

分類の際の同期の対象になるのは、分類の結果のインデックスの個数が入るレジスタである。このレジスタの中身をみてシーケンス制御部はベクトル長の設定を行う。したがって、このレジスタの中身が正しいことを、すなわち、インデックスを分類するベクトル演算が終了したことを、保証する必要がある。

このことから、PUでは、このレジスタに、中身が正しいかを示すVALIDビットを設けることにした。分類の実行中には、VALIDビットが1、終了したら0になるようにしてある。このVALIDビットをレジスタのMSBにすることにより、読み出してきた個数が負であれば、分類が終了していないことが、正であれば分類が終了したことが判るようになっている。

ここでの同期方式が5．3の番号札方式の同期と異なる点は、分類されたインデックスの個数の格納されたレジスタを読む命令があればその他に特別な命令が不要であることと、ソフトウェアでポーリングを行う点である。

6．おわりに

VPPの全体構成と、その単位プロセッサであるPUの処理方式について説明した。VPPは、非常に高速な結合部に複数のベクトルプロセッサを結合した並列処理システムであり、多くの特長を有している。本編では、その中で、PUのパイプラインレベルの同期方式に焦点

を当て説明し、併せて、その有効性も示した。

現在、ＶＰＰについては、８台のＰＵから構成されるパイロットモデルを開発し、評価を進めている。今後、さらに実アプリケーションにより、システム全体の性能の評価を進めていく計画である。

参考文献

［１］前田他：並列処理装置のプロセッサ結合方式、第３１回情処全大２Ｄ－２、１９８５

［２］前田他：インデックスセット処理機能をもったベクトル計算機のアーキテクチャ、第３０回情処全４Ｂ－５，１９８５

［３］井上他：ＶＰＰ（画像処理用ベクトルプロセッサ）の性能評価、第３２回情処全４Ｑ－６、１９８６

［４］前田：画像処理マシン、情報処理、vol.28、pp.19-26、１９８７

［５］橋本他：ＶＰＰシステムにおける画像処理、第３４回情処全６Ｑ－２、１９８７

［６］岩佐他：ベクトルプロセッサＰＵのＬＳＩ化アーキテクチャ、第３５回情処全４Ｃ－７、１９８７

［７］岩佐他：ベクトルプロセッサＶＰＰによる処理方式、第３６回情処全６Ｃ－２、１９８８