

データ駆動計算機のアーキテクチャ最適化に関する考察†

坂井 修一^{††} 平 木 敬^{††} 山口 喜 教^{††}
児 玉 祐 悦^{††} 弓 場 敏 嗣^{†††}

データ駆動計算機の有効性が、大規模な実機によって検証されつつある。本論文では、データ駆動計算機をさらに高速化するためのアーキテクチャの改良について、主にパイプライン設計の観点から述べる。最初に高速化のための新しい手法として、(1)直接マッチング方式、(2)待ち合せ記憶の多バンク化、(3)循環パイプラインにおける先行制御、(4)強連結枝モデル、(5)レジスタファイルを利用した先行制御パイプライン、などの提案を行う。次に、改良型循環パイプラインと強連結パイプラインの結合方式に関して考察する。さらに、(1)～(5)の実現方式に関して検討し、改良型のデータ駆動計算機アーキテクチャを示す。最後に、以上の提案のうち、現在開発中のデータ駆動計算機 EM-4 において採用されている高速化方式について述べ、そのパイプライン構成の詳細を示す。本論文で示された方式に基づくデータ駆動計算機では、演算処理部の稼働率がよく、かつピッチの細かいパイプラインが実現され、待ち合せオーバーヘッド、パケット転送オーバーヘッド、残留トークン処理のオーバーヘッドが大幅に軽減される。

1. はじめに

データ駆動型の計算機アーキテクチャは、(1)問題の並列性を自然な形で抽出することができる、(2)同種の要素プロセッサ (PE) と同種のデータ交換スイッチからなるシステムによって構成することができ、VLSI を用いた実装に有利である、(3)データ駆動言語は並列に動作するプログラムを書くのに有利である、などの理由から、高並列計算機アーキテクチャとして最も有望なものであると考えられる。その有効性は、数十台から 100 台規模の並列要素数を持つ実機で検証されてきている^{1)~6)}。筆者らは、科学技術計算用データ駆動計算機 SIGMA-1 の実装³⁾、記号処理用データ駆動計算機 EM-4 の設計^{7)~12)}などにより、今後のデータ駆動アーキテクチャの進むべき方向を、高速化、高機能化、ハードウェア軽量化の 3 点から検討してきた。

本論文では、データ駆動計算機を高速化するための、計算モデル、実行モデルとアーキテクチャの改良について述べる。最初に従来の典型的なデータ駆動計算機でとられている様々な高速化技法を述べ、従来方式の問題点を挙げる (2 章)。次にパケットアーキテクチャにおける改良法として、(1)直接マッチング、(2)待ち合せ部および網の多ポート化、(3)先行制御

の導入を提案し、その実現方式に関して述べる (3 章)。続いて、(4)改良型データ駆動モデルである強連結枝モデルを提案し、その実現方式に関して述べる。以上 4 点が本論文の提案する方式であるが、さらにこれら 4 点を無駄なく組み合わせることでシステム全体としての性能向上をはかるための総合的な実現方式について検討する。さらに、現在開発中のデータ駆動計算機 EM-4 で採用した最適化方式について述べ、最適化の効果を例題によって述べる (4 章)。最後に、今後の課題を挙げる (5 章)。

なお、本論文では、主にデータ駆動計算機におけるプロセッサ (結合網とのインタフェース部、パケット転送量の最小化などを含む) の高速化を検討の対象とし、結合網の形状と交換方式、構造データの分配方式、資源管理方式などに関する考察、検討は別途報告することとする。

2. 従来のデータ駆動計算機における高速化手法と問題点

2.1 従来型のデータ駆動計算機

従来型のデータ駆動計算機として、ここで想定しているのは、次のようなものである。MIT データフローマシン¹⁾、Manchester データフローマシン²⁾、SIGMA-1³⁾、DFM⁵⁾などは、すべてこれらの条件を満足しており、“従来のデータ駆動計算機”であると考えられる。

● 動的データ駆動モデルに基づくアーキテクチャを採り、トークンはタグ付きのパケットとして表現される。

† Architectural Optimization of a Dataflow Computer by SHUICHI SAKAI, KEI HIRAKI, YOSHINORI YAMAGUCHI, YUETSU KODAMA (Computer Architecture Section, Computer Science Division, Electrotechnical Laboratory) and TOSHITSUGU YUBA (Intelligent Systems Division, Electrotechnical Laboratory).

†† 電子技術総合研究所情報アーキテクチャ部計算機方式研究室

††† 電子技術総合研究所知能システム部

• パケットは PE 内の循環パイプラインで処理され、結果パケットは相互結合網を介して他の PE に送られるか、自 PE で再び処理される。

• 命令レベルのデータ駆動計算機である。

従来型のデータ駆動計算機で提案されている処理の並列化、重畳化は次のようなものである。

① データ駆動図式によって表現される並列性を、複数の PE 間に展開することによる並列化

② データ駆動図式によって表現される並列性を、PE 内の循環パイプライン²⁾上に展開することによる重畳化

③ データ授受機能と処理機能の並列化

④ 待ち合せと命令フェッチの並列化

⑤ 結合網上でのパケット転送の重畳化

⑥ 関数の部分実行を用いた先行制御

⑦ 擬結果関数を用いた先行制御

このうち、①②⑥はすでにほとんどのデータ駆動計算機上で実現されており、⑦は EM-3⁴⁾、DFM⁵⁾などで実装されている。また、③④⑤は SIGMA-1³⁾で実装されている。

以下では①から⑦を前提として、新たな高速化手法について検討する。

2.2 従来型のデータ駆動計算機の問題点

従来型のデータ駆動計算機の欠点を列挙する。

(1) 循環パイプラインは、本来の意味の先行制御を行わない。

循環パイプラインは、先の②によって処理のスループットを高めるものだが、これは本来の意味の先行制御とは異なり、逐次的な処理に対しては重畳処理を行わない。したがって、並列度が十分に高い計算においては高いスループットが得られるが、パイプライン段数と PE 台数の積より小さい並列度の計算に対しては、パイプライン効果が小さくなる。

(2) 待ち合せ処理のオーバーヘッドが大きい。

パケットの待ち合せ処理は、通常、ハッシュなどによるメモリの連想アクセス、パートナーの存在のチェック、待ち合せフラグのクリアまたはデータ書き込みからなり、時間オーバーヘッドが大きい。この対策として、待ち合せ記憶をキャッシュ化して必要とされる時間を短縮することが考えられる。ただしデータ駆動計算機のように文脈の異なるトークンが同時に非決定的に存在する環境では、スワップの頻度が大きいと予想されるため、キャッシュの有効性については定量的な検討が必要である。

データ駆動計算機では、命令を発火させる前に必ず上の3つの操作が必要になる。実際の命令の発火においては、あるパケットのパートナーの存在は、実行前に決定することができる場合があり、その場合、上記の待ち合せを行うのは無駄となる。

これらの結果として、演算実行部へのデータ供給速度が抑えられることになる。

(3) レジスタの利用が困難である。

データ駆動計算機のように文脈の異なる多数のトークンが同時に存在する環境では、レジスタにトークンを蓄えておくことは、有効性が小さい。

(1)(2)(3)より、次の(4)の問題点が導かれる。

(4) 細かいピッチのパイプラインを作るのが困難である。

(5) 残留トークン処理のオーバーヘッドが大きい。

条件分岐を含むプログラムでは、結果データの生成にかかわらないトークン(残留トークン)が生じる。残留トークンは識別子を持っており、その再利用のために回収されなければならないが、そのための時間オーバーヘッドが大きくなる。

残留トークンの回収方式として、①SWITCH 命令はつねにトークンを生成するとし、不要なトークンを SYNC 命令によって回収していく方式、②関数の終了時に待ち合せ記憶、結合網、各種バッファに存在する当該関数に所属するパケットを全部消去する方式、③残留トークンは回収せず、識別子の再利用もシステムの初期化のときまで行わない方式、などが考えられる。

①では、SYNC 命令の列(木状にして並列処理する)がオーバーヘッドを生む。②では、メモリクリアや結合網のフラッシングに時間がかかる。③は、大きな待ち合せ記憶と大きな識別子を持つパケットを前提としているが、それでも、回収されずに残った不要オペランドによって待ち合せ記憶がオーバーフローすることが予想されるため、現実的ではない。

全クリアメモリやフラッシング機構付きの結合網を持たない場合、①の方式を用いる以外に方法はないと考えられる。

(6) 分岐時にデータ流制御のための命令(SWITCH 命令)が数多く必要になる。

(7) パケットの分配、供給能力に限界がある。

PE から結合網に同時に出力可能なパケットの数は限られており(通常1)、ここで処理が直列化される。また、結合網においては、実行プログラムの並列度が

低い状態ではパケット転送の遅延が、並列度の高い状態では結合網の閉塞などによる網スループットの低下が、全体性能を低下させる。

(2)(7)より、次の問題点が導かれる。

(8) 演算実行部の稼働率が低い。

待ち合せ処理の効率化の限界、パケット出力速度の限界、結合網自体のスループットの限界などにより、演算実行部の空き時間が長くなる。

(9) VLSI 化, RISC 化が困難である。

待ち合せに要するハードウェアが大きい、各種メモリーインタフェースを数多く必要とする、レジスタやキャッシュの利用が困難であることなどから、VLSI 化して物理的な大きさを抑えることによる高速化が困難である。また、個々の命令を単純化することで高速化を図る RISC アーキテクチャの適用は、パケットの転送回数を増すことになり、有効ではないと考えられる。

3. パケットアーキテクチャにおける改良

最初に、循環パイプラインだけを用いたデータ駆動計算機の高速化に関する検討を行う。

3.1 待ち合せの高速化

2.2 節で述べた問題点(2)を解決する手法を考える。

動的データ駆動方式の実現のために待ち合せに連想記憶を用いるのは、1回の連想処理に要する時間(ハッシュを用いた場合)、ミスヒットによる時間オーバーヘッド、ハードウェアコストなどの点で不利である。

一方、静的データ駆動方式を採用した場合、連想処理の必要はなくなるが、再入可能性などの点で柔軟性を失う。

動的データ駆動方式を採りつつ連想処理を必要としない待ち合せとして、われわれはセグメント割り付けによる直接マッチング方式⁹⁾を提案した。本方式では、関数起動時に、当該関数インスタンスに必要な待ち合せ数分だけ、待ち合せメモリの領域(オペランドセグメント)が割り付けられる。データパケットのヘッダには、待ち合せ場所が(PE 番号, セグメント番号, 変位)の3つ組となって、絶対番地で書かれている。当該関数の命令は、命令メモリの中のテンプレートセグメントと呼ばれる領域に格納されている。テンプレートセグ

メントとオペランドセグメントの対応関係は、やはり関数起動時に与えられる。待ち合せと同時に、(テンプレート番号, 変位)という2つ組で示される命令メモリの絶対番地によって、待ち合せと同時に命令フェッチが行われる。本方式によって、メモリの利用効率は低下するが、連想処理やハッシュのミスヒットによる時間オーバーヘッドが解消され、待ち合せに要する論理回路のハードウェア量も減少する。

図1に、固定長セグメントを用いた直接マッチングを示した。図では、オペランドセグメントの先頭にテンプレートセグメントの先頭番地を格納することで、両セグメントの対応をとっている。

図2に直接マッチングを用いた循環パイプラインの例を示す。図は、2入力命令用のパケットが4個連続

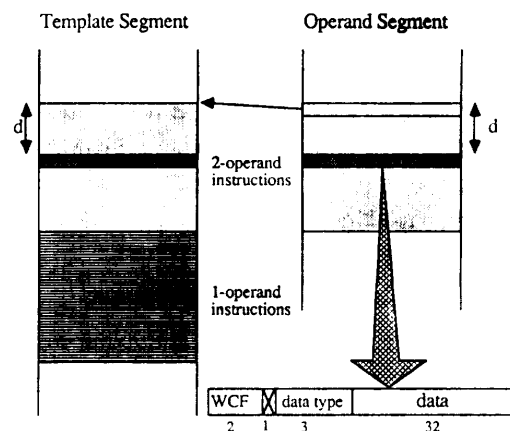
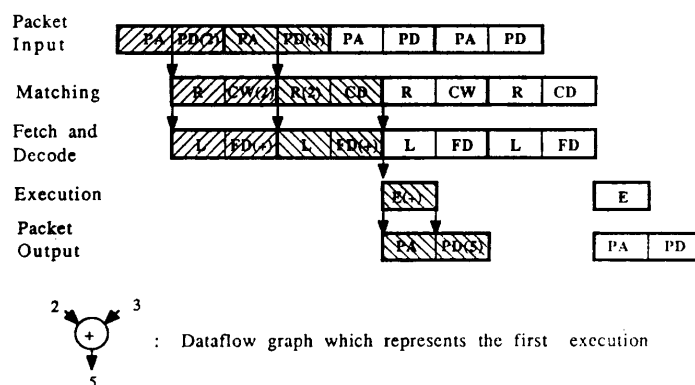


図1 直接マッチング
Fig. 1 Direct matching scheme.



PA: Packet address part, PD: Packet data part, R: Matching memory read, CW: Matching check and data write, CD: Matching check and flag delete, L: Template segment top fetch, FD: Fetch and Decode, E: Execution.

図2 直接マッチングを用いた循環パイプライン
Fig. 2 Circular pipeline with a direct matching scheme.

して到着する場合で、横軸はパイプラインの段数に対応するクロックを表す。図では、各パケットごとに待ち合せメモリの読みだし (R)、パートナ不在の確認とデータの書き込み (CW) またはパートナの確認と待ち合せメモリの語消去 (CD) という 2 段階の操作の後、パートナが存在する場合は、命令が実行される (E)。図の例では、R とテンプレートセグメント番号のフェッチ (L)、CW または CD と命令フェッチおよびデコード (FD) の操作が並列化されている。最初のパケットの処理における FD は、パートナが存在しないため放棄される。E の操作前には、当該オペランドデータの当該命令がそろっている。なお、入出力のパケットは 2 語とし、前半は行先番地の入っているアドレス部 (PA)、後半はデータの入っているデータ部 (PD) とした。命令実行とアドレス部の転送は重畳化されている。

3.2 データ流の高速化、高並列化

2.2 節で述べた問題点(7)(8)の検討を行う。

演算実行部の稼働率を高くするためには、演算実行部へのデータ入力および演算実行部からのデータ出力を高速化する必要がある。

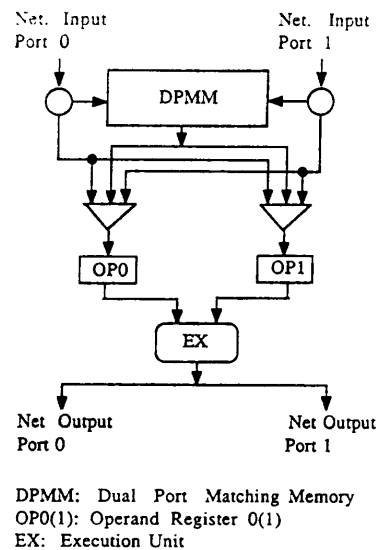
最初に、演算実行部へのデータ入力の高速化について考える。図 2 では、2 入力命令の実行手順が示されていたが、同図において、実行部が稼働するのは、4 クロックにつき 1 クロックしかない。これは、パートナ不在で待たされるトークンについては、演算実行部が使用されないためである。すなわち、2 入力命令の実行を考えると、2 クロック (1 パケットの移動時間) に 1 回演算を行うためには、待ち合せ処理部の処理速度は演算実行部の 2 倍必要であることがわかる。

その実現法として、待ち合せメモリを 2 ポート化し、入力パケットの各ポートへの到着頻度がほぼ同じになるように調整する方式が考えられる。この方式では、片方のポートのオペランドのパートナが存在しても、同時に入力された他方のポートのオペランドのパートナが存在すればデータが供給されるため、図 2 の 2 倍近い演算実行部の稼働率が期待される。

2 ポートの待ち合せメモリを有効に使うためには、それぞれのポートにパケットが入力される速度が、図 2 の 1 オペランドの入力される速

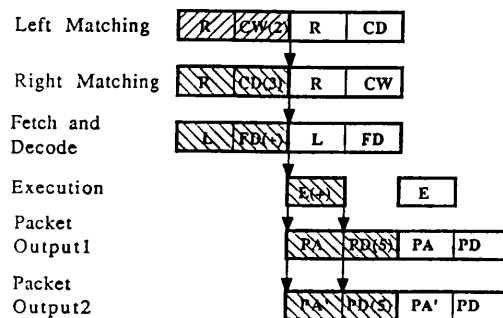
度と同程度以上でなくてはならない。すなわち、相互結合網からのデータの供給能力が、以前の 2 倍かそれ以上必要になる。その実現法として、パケット生成および転送の速度を演算速度の 2 倍にすることが考えられる。しかし、演算処理が 1 チップの LSI で実現可能であるのに対し、パケット転送は通常チップ間にまたがる信号伝達を含んでおり、これは現実的ではない。

これに対して、結合網から PE への入力ポートを 2 つ設け、それぞれを待ち合せメモリのひとつのポー



(a) PE の構成

(a) PE organization.



R: Matching Memory Read CW: Matching Check and Data Write
CD: Matching Check and Flag Delete L: Template Segment Top Fetch
FD: Fetch and Decode E: Execution

(b) 改良型循環パイプライン

(b) Improved Circular Pipeline.

図 3 改良型循環パイプラインの実現

Fig. 3 Realization of an improved circular pipeline.

ト専用として用いる方式が考えられる。本方式によれば、ハードウェア量は大きくなるが、高いスループットのデータ転送が可能である。ここでは、後者の方式を採用することを考える。

図2では、パケットを1つだけ出力する演算を扱ったが、実際には複数の出力パケットの出力終了まで演算部がアイドル状態を続けるような事態がしばしば生じる。これには、PE から結合網への出力ポートを複数設け、さらに結合網に放送機能を設ける方式で対処することを考える。

以上の改良を取り入れた PE の構成を図3(a)に、パイプライン構成を図3(b)に示す。2入力命令では、演算実行部は2クロックに1クロックの割合で稼働することになり、図2と比較して約2倍の処理性能が得られる。また、放送機能を用いて同時に2つのパケットを出力することができ、実行部に待ちをつくらない。ハードウェアとしては、待ち合せメモリの2ポート化、同時に2つの待ち合せを行うための制御機構、結合網との間の入出力ポートの数を増したことによる PE の網インタフェース部の拡張、結合網の拡張と放送機能、などがあらたに必要となる。

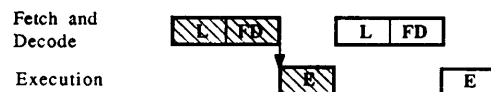
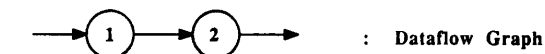
さらに、演算実行部の稼働率を1に近づけるための手段として、①図3のようにパケットを2語に分けず、結合網の1ポートあたりの線数を増して、1クロックに1パケットを転送するようにし、②待ち合せ処理の並列度を4に上げて、常時データ供給できるようにする、という方式が考えられる。本方式は、ハードウェアが大幅に増加し、制御回路やデータマルチプレクサなどの遅延が増す欠点がある。

3.3 先行制御の導入

2.2 節で述べた問題点(1)の解決法を考える。

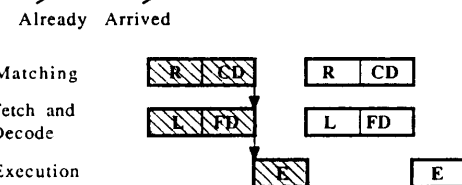
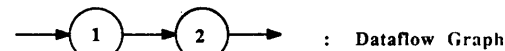
循環パイプラインでは、次に実行されるトークンがないとき、待ち合せ処理部や命令フェッチ部は、先行制御をせずにアイドル状態に入る。たとえば図4(a)(b)のように、逐次的な処理の場合、この種のアイドル時間が生じる(処理方式は図3と同じとした)。これに対して、フォンノイマン方式の計算機では、逐次処理においても次の命令のフェッチやデコードを行っている。この点がデータ駆動方式のもっとも不利な点のひとつとなっている。

図4(a)(1入力命令が連続する場合)は、図5(a)のように処理されるのが望ましい。すなわち、最初の命令実行のときに2番目の命令のフェッチとデコードを同時に行うようにする(以下同様)。この場



(a) 1 入力命令が連続するとき

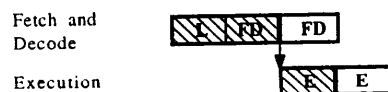
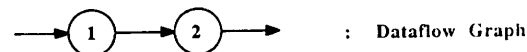
(a) Continuous one-operand nodes.



(b) 2 入力命令が連続するとき

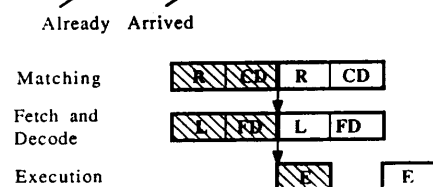
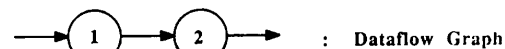
(b) Continuous two-operand nodes.

図4 逐次処理における実行手順
Fig. 4 Executing sequential nodes.



(a) 1 入力命令が連続するとき

(a) Continuous one-operand nodes.



(b) 2 入力命令が連続するとき

(b) Continuous two-operand nodes.

図5 循環パイプラインにおける先行制御
Fig. 5 Advanced control in a circular pipeline.

合、先行制御の重畳効果のほかに、連続する処理でテンプレートセグメントは同じであるから、Lの作業は省略可能となり、スループットがさらに向上する。同様に図4(b) (2入力命令が連続する場合)は、図5(b)のように処理されるのが望ましい。スループットは(a)の処理が連続する場合3倍に、(b)の処理が連続する場合1.5倍になる。

次に図5の処理のために必要な機構を考える。

現在実行中の命令の結果パケットの行先が自PEであり、かつ次のパケットの到着がないときは、結果が出ないうちに行先の待ち合せとフェッチを先行処理する。その結果、パートナーが不在のときは、次のステップで当該命令の結果トークンを待ち合せメモリに書き込み、フェッチした命令は放棄する。パートナーが存在するときは、次の命令を実行する。以上の操作を行うことができる制御機構があれば、図5の処理は実現される。この場合、1つ先の待ち合せ、フェッチの先行処理は比較的簡単に実現できると考えられるが、2つ以上先の待ち合せ、フェッチは困難である。したがって、フォンノイマン型の大型計算機に見られるような、段数の細かいパイプラインを、データ駆動計算機上の先行制御に適用するのは不向きと考えられる。

3.4 パイプラインの細分化

2.2節の問題点(4)について検討する。

循環パイプラインのパイプラインピッチは、①パケットが入ってから出るまでの処理をいくつかの重畳化可能な部分処理に分割できるか、②逐次処理において先行制御はどの程度可能であるか、また全体の処理の中で逐次処理が支配的なところはどれくらいあるか、の2つによって決まる。これまでの検討により、逐次処理においても先行処理ができ(3.3節)、全体の処理量もセグメント割り付け方式による直接マッチング(3.1節)によって減少させることができ、また待ち合せ処理の並列化(3.2節)が可能であることがわかった。以上を考慮すると、従来型のデータ駆動計算機よりは段数の多いパイプライン構成が適当であると考えられる。しかし、先行制御の限界(3.3節)から、処理の分割によって増やせる段数は、たかだか1、2段であろうと考えられる。

4. 強連結枝モデルの導入とその実現法

4.1 実行順序の制約

3.1節では、データ駆動図式の処理を、パケットアーキテクチャと循環パイプラインの上でデータ駆動

モデルに忠実に実現する場合における、処理方式とアーキテクチャの改良について述べた。その結果、①直接マッチングによって待ち合せ処理が簡単化、高速化され、②待ち合せメモリと結合網の入出力の多ポート化により、演算実行部の稼働率が向上し、③1、2段程度の先行制御によってスループットが向上し、④パイプラインの若干の細分化が可能になった。しかし、3段以上の先行制御は難しく、レジスタやキャッシュの利用は依然として困難であり、したがってパイプラインの細分化には限界がある。また、残留トークンの処理、分岐によるデータ流変更命令のオーバーヘッドなどの問題も残されている。

これらの問題を解決するためには、ノード間のデータ依存関係だけではない実行順序の制約や発火の優先度を設けることが有利なことが多い。すなわち、データ依存関係にない2つのノード間の実行順序あるいは発火の優先度を指定することで、先行制御が可能になり、残留トークンの数を減らすことができる。データを記憶に蓄えて移動させない履歴依存型の処理も、実行順序を制約したときの利点を生かすものと考えられる。

実行順序を制限し、文脈を記憶して処理を進める典型的な例が、マクロデータ駆動計算機¹³⁾である。マクロデータ駆動計算機では、マクロブロック内の順序制御はフォンノイマン的に実現され、マクロブロックの発火はデータ駆動的に実現される。マクロブロックは、関数インスタンスそのものであってもよい、関数インスタンスの内部にブロックを作ってもよい。

われわれは、データ駆動モデルの枠内で発火の優先度を設けることを可能にする改良をあらたに加え、これに基づいた効率的な処理をデータ駆動計算機に行わせることを考案した。次節以下で述べる強連結枝モデル⁷⁾がこれである。

4.2 強連結枝モデルの定義と適用

最初に強連結枝モデルの定義を示す⁷⁾。

【定義】 データ駆動図式において枝(arc)に2種類の区別を設ける。一方を常連結枝と呼び、他方を強連結枝と呼ぶ。強連結枝で結合されたデータ駆動部分図式を強連結ブロックと呼ぶ。

【発火規則】 データ駆動図式上のノードは、そのすべての入力枝にトークンが到着したとき発火可能となる(通常のデータ駆動)。強連結ブロック内のノードが発火したとき、この強連結ブロック内のノードを実行するPEはすべて当該ブロックだけを排他的に実

行するモードに入る。すなわち、強連結枝を出力枝とするノードが発火し実行されると、同じ強連結ブロック内の発火可能なノードだけが次に発火する。

【強連結ブロックの実行終了】 ブロック外部から当該ブロックへのトークンがすべて到着し、かつ発火可能なブロック内のトークンがなくなると、強連結状態は解除され通常の処理に戻る。

図6に強連結ブロックの例を示す。図中で細い矢印が常連結枝、太い矢印が強連結枝である。図では、ブロックAとブロックBの2つの強連結ブロックがある。ノード5またはノード6が発火すると、ブロックAまたはブロックBの排他実行のモード（強連結モードと呼ぶ）になる。ここでは、1つの強連結ブロックは、1つのPEで実行されるとする。

次に強連結枝モデルを用いることの利点を列挙する。

(1) ブロック外部のノードの実行を除外することで文脈が制限されるため、先行制御がやりやすくなる(2.2節の(1)の解決)。

(2) ブロック内の処理では、待ち合せの空間が絞りこまれている(関数の内部)ため、待ち合せが高速かつ簡単に実行される(2.2節の(2)の解決)。

(3) 強連結枝モデル内のトークンを格納するためにレジスタファイルを用いることができ、命令処理時間とパイプラインピッチが短縮される(2.2節の(3)の解決)。

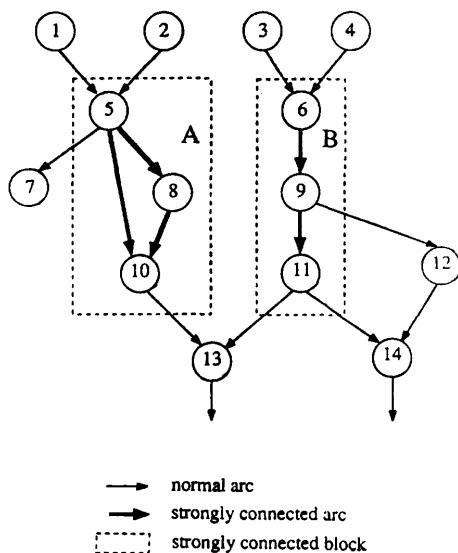


図6 強連結枝と強連結ブロック

Fig. 6 Strongly connected arcs and strongly connected blocks.

(4) 以上の(1)(2)(3)の結果として、細かいピッチのパイプラインが容易に設計されるようになる(2.2節の(4)の解決)。

(5) レジスタファイルを用いることで、パケット転送がなくなり、2.2節の(7)の欠点が解消される。また、分岐があったときも、分岐するデータはレジスタファイル上に保存されているため、SWITCH 命令をいちいち発行する必要がない。したがって、2.2節の(6)の欠点が解消される。

(6) 残留トークンの収集が効率化する。これは、強連結ブロックの終了にあたって、レジスタファイルのフラグクリアだけで、残留トークンの消去を完了させることができるからである(2.2節の(5)の解決)。

(7) レジスタファイルの利用、パケット転送がなくなること、待ち合せの単純化などにより VLSI 化、RISC 化が行いやすくなる(2.2節の(9)の解決)。

このほかにも、本モデルの導入によって資源管理が容易に行えるようになる、などの利点があるが、本文の範囲を出るので省略する。

強連結枝モデルを採用した場合、プログラムの強連結化処理は次のような手順になる。

① プログラムをデータ駆動図式に展開する。

② データ駆動図式の部分部分を、並列度を損なわない範囲で強連結ブロック化する。

③ 並列度を損なう部分も、ブロック化による重量効果や残留トークンの処理の単純化が並列効果より大きいところは、強連結ブロック化する。

④ 実際のデータ駆動計算機の上で強連結ブロックを実行する。

①②③の処理は、すべてコンパイラで実現することが望ましいが、一部をプログラマに負担させる方法も考えられる。

強連結ブロックの自動生成アルゴリズムは、現在検討中であり、別途報告する予定である。

4.3 強連結枝モデルの実現方式

ここでは、前節と同様に、1つの強連結ブロックは1つのPEで実行されるとする。

強連結枝モデルを実現するハードウェアとして必要なものは、①各語に同期用のプレゼンスビットのついたレジスタファイル、②発火可能なノードを発火し、発火可能なノードがなければ次に発火可能になると予想されるノードを先行発火する機構、の2つである。強連結ブロック内のノードに全順序を与えれば、レジスタファイルのプレゼンスビットは不要になり、先行

制御も容易になるが、コンパイラの負担が重くなる。

図6で見たように、強連結ブロックの発火は通常の packets によって行われる。ブロックが発火すると当該 PE は強連結モードとなり、当該強連結ブロックの入力データを持つ packets 以外は受け付けなくなる。実際には強連結ブロックの入口ノードを1個に限定し、ここで多数個(3個以上)の入力の待ち合せを行う機構⁹⁾を設けることで、強連結ブロックの内部では packets を受け付けないようにする。この方式が、packets 形式の簡単さ、ハードウェアの簡単さ(優先度つきキュー、コンテキストスイッチなどが不要になる)、処理効率などの点から有利であると考えられる。強連結ブロック内のノードの結果データは、外部のノードの入力となるもの以外は packets 化されず、一時的にレジスタファイルに格納される。したがって、強連結ブロック内のノードでは、入出力ともにレジスタファイル上で実現される。

4.4 循環パイプラインと強連結パイプラインの結合

本節では、3章で述べた packets アーキテクチャにおける改良と前節までで述べた強連結枝モデルに基づく改良とを結び付ける方式について検討する。すなわち、改良型循環パイプラインと、前節までで述べた強連結ブロック内のパイプラインの結合を考える。その際考慮しなくてはならないのは次の諸点である。

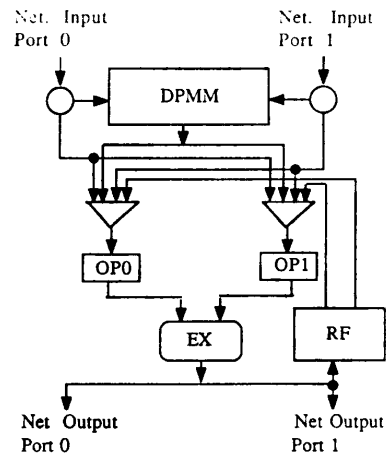
(1) 強連結枝モデルを適用した上で、さらに 3.2 節で述べた待ち合せ記憶と結合網インタフェースの多ポート化が必要か否か。

(2) 強連結枝モデルを適用した上で、さらに 3.3 節で述べた循環パイプラインの先行制御が必要か否か。

(3) 2つのパイプラインがともに行う命令フェッチ、デコード、演算処理、packets 出力の各処理を、両者がパイプラインの段として共有するか否か。

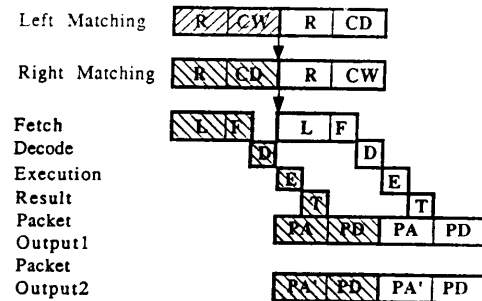
(1)については、待ち合せ記憶と結合網インタフェースの多ポート化は、ハードウェア量と信号線の増加の点で LSI 化に不向きである。しかし、全体性能の点から、LSI 化可能な範囲で実現することが望ましい。一方、強連結枝モデルの適用によって、循環パイプラインの上の待ち合せの回数が減少するため、多ポート化の必要性は従来型のデータ駆動計算機にくらべて小さくなっている。

(2)の循環パイプラインの先行制御は、原則として逐次的な処理を強連結ブロック化し、強連結枝の先行



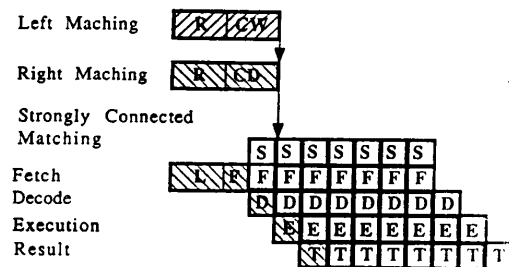
DPMM: Dual port matching memory, OP 0(1): Operand register 0(1), EX: Execution unit, RF: Register file.

図7 改良型 PE アーキテクチャ
Fig. 7 Improved PE architecture.



(a) 循環パイプライン

(a) Circular pipeline.



(b) 強連結パイプライン

(b) Strongly connected pipeline.

R: Matching memory read, CW: Matching check and data write, CD: Matching check and flag delete, L: Template segment top fetch, F: Fetch, D: Decode, E: Execution, T: Result transfer, PA: Packet address part, PD: Packet data part.

図8 2種のパイプライン
Fig. 8 Two types of pipelines.

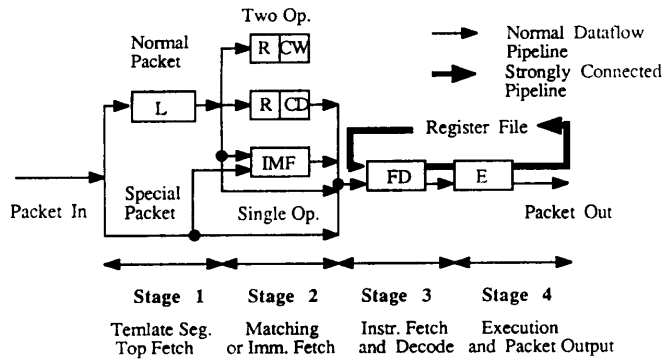


図 9 EM-4 のパイプライン構成
Fig. 9 EM-4 pipeline design.

制御機構を働かせることで、不要になると考えられる。また、これにより一般の逐次処理が高速化されるため、循環パイプライン自身も細分化が可能となる。したがって、2つのパイプラインのピッチは同じにするのが適切であり、(3)のパイプライン段の共有も採用してよいと考えられる。

4.5 改良型アーキテクチャ

図 7 に、前節までで述べた内容を実現するための PE のアーキテクチャを示す。本 PE では、通常データ駆動の待ち合せに直接マッチングが採用され、マッチングメモリと結合網インタフェースは 2ポート化されている。通常データ駆動のモードでは、トークンはパケットとして PE に入り、待ち合せ、演算処理を経て、結果トークンは再びパケットの形で出力される。強連結モードでは、トークンはレジスタファイルからロードされ、結果は再びレジスタファイルに格納されるか、パケットの形で他の PE に送られる。

図 8 に図 7 の PE におけるパイプライン処理の様子を示す。図 8 (a) は通常データ駆動モードにおける処理を示したものである。図 3 (b) に対応するが、命令フェッチ以下のパイプラインが細分化されている (1 クロックに必要な絶対時間が約 2 分の 1 となる)。図 8 (b) は、強連結ブロック内のパイプライン処理の

様子を示したものであるが、先行制御とレジスタファイルの利用によって高いスループット (1 クロックに 1 つの結果) が得られていることがわかる。

4.6 EM-4 における高速化の実現

図 9 に、現在開発中のデータ駆動計算機 EM-4 におけるパイプライン構成を示す。EM-4 では、前節までで述べてきた方式のうち、セグメント割り付けによる直接マッチング方式を採用し、強連結枝モデルを導入した。本マシンの PE は、通信機能も含めてシングルチップ化されており、高密度化により、1,000 台規模のシステムの実装が可能となっている。待ち合せメモリおよび結合網インタフェースの多ポート化は、シングルチップ化を困難にするために採用しなかった。また、強連結ブロック内のノードには全順序を与える方式を採って、先行制御をやりやすくしている。図 8 (b) と比較してパイプライン段数が少ないのは、パイプラインレジスタを入れるためのゲート数が確保できなかったためである。EM-4 の PE チップは、入出力信号線数 255、ゲート数約 46,000 の CMOS ゲートアレイであり、1 台あたりの最大性能は約 12.5 MIPS である。

図 10 に EM-4 において階乗計算をループ方式で行う 2 種類のプログラムを示す。前者は図 2 のパイプラインに対応する通常データ駆動型のプログラムであり、後者は強連結枝モデル導入後の最適化されたアーキテクチャに対応するプログラムである。

前者はループ一周の演算実行時間が全部で 22 クロック (1,760 ns) であり、クリティカルパスの実行時間が 13 クロック (1,040 ns) である。すなわち、PE を 2 台以上用いる場合でも 13 クロック未満でこれを実行することはできない。それに対して後者のループ一周の実行時間は 4 クロック (320 ns) であり、3 倍以上速い。

このように最適化されたアーキテクチャを用いる場

```
FACT: (COPY<EQ1 imm1><SW1 r>)
EQ1: (EQ '1<SW1 l><SW2 l>)
SW1: (SW(F<SUB1 imm1><MULT l>))
SW2: (SW(T<END>)(F<MULT r>))
SUB1: (SUB '1<SYNC1 l>)
MULT: (MUL<SYNC1 r>)
SYNC1: (SYNC(L<FACT s>)(R<SW2 r>))
```

(a) A normal dataflow program.

```
FACT: [BEQ '1(RESULT)(R0: NL)]
[SUB '1 R0(NL: NL)(NL: R1 $)]
[BALL(FACT)(R0: R1)]
[MUL R1 (R0: NL)]
RESULT: [MKPKT<END>]
```

(b) A strongly connected dataflow program.

図 10 階乗計算のプログラム
Fig. 10 Two factorial programs.

合、データ駆動図式の中で並列度の低い部分やスイッチ命令が多くある部分は強連結化するのが有利である。

5. おわりに

データ駆動計算機のアーキテクチャ最適化として、主にパイプライン設計の観点から、(1)直接マッチング方式の導入と従来の循環パイプラインの改良、(2)強連結枝モデルの導入と強連結パイプラインの設計、(3)改良型パイプラインと強連結パイプラインの結合、(4)実現機構とアーキテクチャ、(5)EM-4における実現方式、という順序で論じた。その結果、提案した方式によって、演算処理部の稼働率が高くピッチの細かいパイプラインが実現され、待ち合せオーバーヘッド、パケット転送オーバーヘッド、残留トークンオーバーヘッドがあまり生じないデータ駆動処理が実現されることがわかった。今後の課題として、相互結合網構成の最適化、強連結ブロック生成アルゴリズムの確立、コンパイラにおける最適化スケジューリングを含む種々のソフトウェア技法の確立、本論文で述べた方式の様々な応用問題への適用と評価などがあげられるが、これらはすべて EM-4 の実機の上で行われる予定である。

謝辞 本研究を遂行するにあたり御指導、御討論いただいた棟上情報アーキテクチャ部長、島田計算機方式研究室長ならびに計算機方式研究室の同僚諸氏に感謝いたします。

参 考 文 献

- 1) Arvind, Dertouzos, M. L. and Iannucci, R. A.: A Multiprocessor Emulation Facility, MIT-LCS Technical Report 302 (1983).
- 2) Gurd, J. R., Kirkham, C. C. and Watson, I.: The Manchester Prototype Dataflow Computer, *CACM*, Vol. 28, No. 1, pp. 34-52 (1985).
- 3) Hiraki, K., Sekiguchi, S. and Shimada, T.: System Architecture of a Dataflow Supercomputer, *TENCON 87*, Seoul, pp. 1044-1049 (1987).
- 4) 山口, 戸田, 弓場: 先行制御をもつデータ駆動計算機 EM-3 の評価, 電子情報通信学会論文誌 D-1, Vol. J72-D-1, No. 3, pp. 182-195 (1989).
- 5) Amamiya, M., Takesue, M., Hasegawa, R. and Mikami, H.: Implementation and Evaluation of a List-Processing-Oriented Data Flow Machine, *Proc. of the 13th Annu. Symp. on Comp. Arch.*, pp. 10-19 (1986).
- 6) 能美, 菊池, 中田: NEDIPS のデータフロー制御方式, 電子情報通信学会データフローワークショップ予稿集, pp. 239-246 (1987).
- 7) 坂井, 山口, 平木, 弓場: データ駆動型シングルチッププロセッサ EMC-R における強連結枝モデルの導入, 電子情報通信学会データフローワークショップ予稿集, pp. 231-238 (1987).
- 8) 坂井, 山口, 平木, 児玉, 弓場: データ駆動型シングルチッププロセッサ EMC-R のアーキテクチャ, 信学技報, CPSY 88-9, pp. 17-24 (1988).
- 9) 山口, 坂井, 平木, 児玉, 弓場: データ駆動計算機 EM-4 における待ち合せ機構, 第 37 回情報処理学会全国大会論文集, IN-7 (1988).
- 10) Sakai, S., Yamaguchi, Y., Hiraki, K., Kodama, Y. and Yuba, T.: An Architecture of a Dataflow Single Chip Processor, *Proc. of the 16th Annu. Symp. on Comp. Arch.*, pp. 46-53 (1989).
- 11) Yamaguchi, Y., Sakai, S., Hiraki, K., Kodama, Y. and Yuba, T.: An Architectural Design of a Highly Parallel Dataflow Machine, *IFIP Congress '89*, pp. 1155-1160 (1989).
- 12) 坂井, 山口, 児玉, 平木: データ駆動計算機の究極の高速化に関する検討, 並列処理シンポジウム JSPP '89, pp. 71-78 (1989).
- 13) 戸田, 内堀, 島田: マクロデータフローアーキテクチャの検討, 並列処理シンポジウム JSPP '89, pp. 79-84 (1989).

(平成元年 5 月 10 日受付)

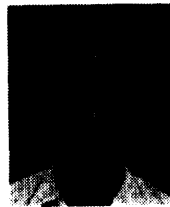
(平成元年 9 月 12 日採録)

坂井 修一 (正会員)



昭和 33 年生。昭和 56 年東京大学理学部情報科学科卒業。昭和 61 年同大学院工学系研究科情報工学専門課程修了。工学博士。同年電子技術総合研究所入所。並列処理計算機、特に相互結合網、スケジューリング、データベースマシン、データフローマシン、並列 AI マシンなどの研究に従事。本学会平成元年度研究賞受賞。現在、情報アーキテクチャ部計算機方式研究室に所属。電子情報通信学会会員。

平木 敬 (正会員)



昭和 51 年東京大学理学部物理学科卒業。昭和 57 年同大学院理学系研究科博士課程修了。同年電子技術総合研究所入所。理学博士。計算機アーキテクチャ全般、特にリスト処理計算機、データフローマシン、スケジューリングなどの研究に従事。元岡賞、市村賞各受賞。情報アーキテクチャ部計算機方式研究室主任研究官。現在 IBM ワトソン研究センター招聘研究員。

山口 喜教 (正会員)

昭和 24 年生。昭和 47 年東京大学工学部電子工学科卒業。同年通商産業省工業技術院電子技術総合研究所入所。以来、高級言語計算機、データ駆動型計算機などの研究に従事。

現在、情報アーキテクチャ部、計算機方式研究室主任研究官。電子情報通信学会会員。

児玉 祐悦 (正会員)

昭和 37 年生。昭和 61 年東京大学工学部計数工学科卒業。昭和 63 年同大学院工学系研究科情報工学専門課程修士課程修了。同年通商産業省工業技術院電子技術総合研究所入

所。以来、データ駆動計算機などの並列計算機システムの研究に従事。現在、情報アーキテクチャ部計算機方式研究室に所属。

弓場 敏嗣 (正会員)

1941 年生。昭和 39 年神戸大学工学部卒業。昭和 41 年同大学院修士課程修了 (電気工学専攻)。(株)野村総合研究所を経て、昭和 42 年通商産業省工業技術院電気試験所 (現、

電子技術総合研究所) に入所。以来、計算機のオペレーティングシステム、見出し探索アルゴリズム、データベースマシン、データ駆動型並列計算機などの研究に従事。工学博士 (情報工学)。計算機方式研究室室長を経て、現在、知能システム部部长。筑波大学併任教授 (電子情報工学系)。電子情報通信学会, Association for Computing Machinery 各会員。