計算機アーキテクチャ　109－6
（１９９４．１２．１３）

データ駆動プロセッサにおけるスレッド処理についての考察

寺田秀文
静岡大学　工学部
e-mail：terada@jce.shizuoka.ac.jp

データ駆動の動的命令スケジューリングはプログラムに内在する並列性を自然に抽出できる，遠隔メモリ参照のような長いレイテンシの操作を許容できる，などの超並列計算機向きの特徴を持つ．しかし静的命令スケジューリングの欠如に起因する非効率性が認識され，主に制御駆動との融合が検討されてきた．本報告では制御駆動との融合よりはむしろ，静的にスケジュールされた命令がコードブロックの連続位置に配置されることを前提に，多重命令フェッチと命令プリフェッチとでスレッド処理の効率化を考察する．さらに機能ユニットの多重化と共に多重行先トークンを導入することで同時フェッチした命令の並列実行について考察する．

A Study for Threads Processing on Data Driven Processors

Hidefumi TERADA
Faculty of Engineering, Shizuoka University
3-5-1 Johoku, Hamamatsu, shizuoka, 432 Japan
e-mail：terada@jce.shizuoka.ac.jp

The dynamic instruction scheduling on data driven machines has excellent features for massively parallel computers. For example, it can naturally extract parallelisms within programs and it can tolerate long latency operations such as remote memory reference. However, because of the machines without static instruction scheduling have inefficiencies, hybrid architectures are mainly studied so far. In this paper, efficient threads processing by multiple instruction fetch and instruction prefetch is discussed. Furthermore, multiple destination token to support parallel execution is introduced.

1．まえがき

データ駆動計算機[1]～[2]の動的命令スケジューリングは超並列計算機向きの優れた特徴を持つ．例えば，プログラムに内在する並列性を自然に抽出できる，遠隔メモリ参照のような長いレイテンシの操作を許容できるなどが揚げられる．

動的データ駆動を実現した初期の計算機[3]～[5]の連想メモリは明示的に管理するフレームストア[6]～[14]に置き換えられ，同一データ分配の効率化[8]などの改善が行われてきた．しかし，データ駆動計算機がより高い達成度を得るためには根本的な改善，すなわち静的命令スケジューリングの導入が必要である．これによりパイプラインでの逐次処理重畳が容易になるだけでなく，資源管理や危険領域のプログラミングが容易になる．

スケジュールされた命令の実行支援としては，結果行先による先行制御[7]，結果トークンの再循環[9]～[11]，制御駆動との融合[12]～[14]がある．制御駆動との融合には制御駆動的実行の支援から，スレッド実行をＲＩＳＣプロセッサで行うマルチスレッドアーキテクチャまで存在する．この融合段階で，スレッド内に有効なレジスタが導入され，存在ビットによる同期はフレームストア上の同期に置き換えられた．

さて，本報告では制御駆動との融合よりはむしろ，多重命令フェッチ[16]～[17]と命令プリフェッチとで静的にスケジュールされた命令の実行支援を考察する．静的にスケジュールされた命令がコードブロックの連続位置に配置されることを前提に，トークンが指示する命令とそれに続く複数命令を同時にフェッチする．フェッチした命令間にデータ依存関係があれば，これら命令を連続実行する．さらに結果行先命令をプリフェッチすることで，逐次処理の重畳を行う．また多重命令フェッチで本来期待できる並列実行を実現するために，コードブロックの連続位置に置かれた複数命令を行先とする多重行先トークンを導入する．

以降，第２章でデータ駆動計算機を概観することで本報告の基礎モデルを述べ，第３章で多重命令フェッチと多重行先トークンを導入する．第４章で実現例を述べ，第５章でこれらの効果を示し，最後に今後の課題などを述べる．

2．動的データ駆動計算機

2．1　動的データ駆動の実現における改良

（１）待ち合わせ処理実現において，初期の動的データ駆動計算機が採用した連想記憶は明示的に管理されるフレームストアに置き換えられた．

ＥＴＳ[9]はアクティベーションフレームを関数呼び出し時の呼び出し規則の一部として明示的にフレームストアに配置する．アクティベーションフレーム内の各待ち合わせ位置は静的に決定する．図１のようにトークンは行先情報としてフレームポインタＦＰと命令ポインタＩＰを持つ．コードブロック内の命令をＩＰで直接指示し，アクティベーションフレーム内の待ち合わせ位置をＦＰ＋ｒで指示する．ここでｒは命令が保持する待ち合わせ位置と存在ビットのアクティベーションフレーム内変位．また，ｓは結果行先命令の相対アドレスである．

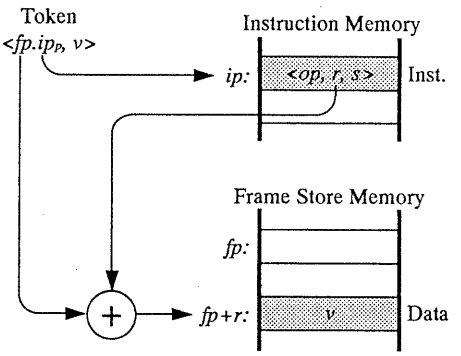

図１：Explict Token Store

またＥＭ－４では待ち合わせ位置をトークンで直接に指示する直接マッチング[6]が採用されている．

（２）Epsilon[8]は多重行先トークンとrepeat-on-inputを使い，データ分配を効率化した．一般的に同一データを複数命令に分配する時，分配命令を使って複数のトークンを生成する．Epsilonにおいて，トークンには繰り返し数と繰り返しステップが明記され，repeat-on-input機構がパイプラインにデータを繰り返し投入する．プログラム中の分配命令数，トークントラフィック，トークン循環オーバーヘッドなどが抑制され，達成度の向上が報告されている．

（３）データ駆動計算機のパイプラインは待ち合わせ部と演算実行部から成る．前者はトークンが指示する命令の実行可能性検査や実行可能命令の発行

などを行い，後者は命令が明記する演算や結果トークンの生成などを行う．これらが同一性能である場合，２入力命令に先着するトークンで起きる待ち合わせの失敗は演算実行部を頻繁にアイドルにする．Monsoonにおける巡回セールスマン問題で全サイクルの約３０％がこのＡＬＵバブルである[10]．待ち合わせ部の相対性能向上とトークン供給量増加の手段として，待ち合わせメモリとネットワークインターフェースの多ポート化がＥＭ－４に対して検討された[7]．

２．２　静的命令スケジューリングの導入

データ駆動は半順序の命令に対して動的に命令をスケジュールする．換言すれば，翻訳時に最適な実行が解析可能な場合でも静的命令スケジューリングを許さない．静的命令スケジューリングの欠如はパイプラインによる逐次処理の重畳を困難にし，資源管理や危険領域のプログラミングを困難にする．従って，静的命令スケジューリングの導入はデータ駆動計算機の達成度改善に本質的である．スケジュールされた命令，すなわちスレッドの実行を支援する方法を次に述べる．

第１に結果行先でプリフェッチを行う方法がある．現在実行中の命令の結果行先が自処理要素内であり，処理を待つトークンが存在しない時，結果出力を待たずに行先となる命令の待ち合わせとフェッチを先行させる方法である．これは静的命令スケジューリングを導入するものではないが，逐次処理の効率化が期待できる．

第２にMonsoonで採用された結果トークンの再循環がある．これはスケジュールされた命令の結果トークンを直接にパイプラインへ再循環する．Monsoonは忠実なデータ駆動の８ステージパイプラインを採るため，スレッド実行は８サイクル毎になる．スレッド実行の効率化のために，テンポラリレジスタの利用や複合命令によるスレッド長短縮を行っている．

第３に制御駆動の取り込み，すなわちアーキテクチャ統合を図るハイブリッドまたはマルチスレッドアーキテクチャがある．Ianuucci[12]はデータ駆動図式にスケジューリングカンタＳＱを定義し，命令ポインタとフレームポインタの対である継続点（continuation）でＳＱを制御するハイブリッドアーキテクチャを提案し，制御駆動的な命令スケジューリング（ＩＰ＋１）やレジスタを導入した．またＥＭ－４はデータ駆動図式に強連結枝ブロックを定義し，通常のデータ駆動処理と排他的に強連結枝ブロックを演算実行部のＲＩＳＣパイプラインで実行する．Ｐ－ＲＩＳＣ[13]や＊Ｔ[14]はＲＩＳＣプロセッサによるスレッド実行，フレームストア上で同期を行うマルチスレッドアーキテクチャを提案している．

３．多重命令フェッチと多重行先トークン

本報告はＥＴＳに基づく動的データ駆動プロセッサにおいて，多重命令フェッチと命令プリフェッチを導入する．これらの導入で逐次処理の重畳が可能になる．さらに機能ユニットの多重化を前提に多重行先トークンの導入を行い，並列実行を実現する．これらの導入で待ち合わせ部の相対性能向上やトークン循環オーバーヘッドの削減などが行われる．

３．１　命令とデータの配置

静的にスケジュールされた命令（スレッド）はコードブロックの連続位置に配置する．図３は灰色部分をスレッドとした場合の命令配置を示す．他の命令の配置については次節で述べる．

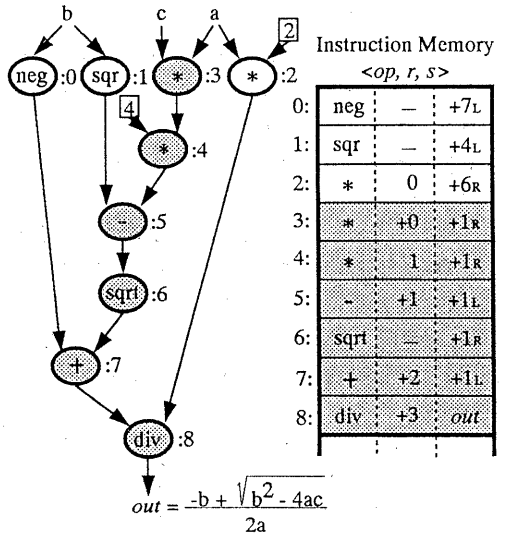

図２：命令メモリへの命令配置例

静的命令スケジューリングに従う命令配置は対応する存在ビットと待ち合わせ位置の配置にも反映させる．これらの配置は待ち合わせを必要とする命令の出現順序でその命令のアクティベーションフレーム内変位を決めれば良い．ここで，存在ビットと待ち合わせ位置はそれぞれのメモリ上で同一のアドレス，すなわちＦＰ＋ｒである．ＦＰはトークンが持つフレームポインタ，ｒは命令が持つアクティベーションフレーム内変位である．なお，スレッド化処理を含むプログラム翻訳については本報告の範囲を超えるために別途報告とする．

３．２　多重命令フェッチ

前節の命令配置を前提にトークンの命令ポインタが指示する命令とそれに続く複数の命令を同時にフェッチする．多重命令フェッチで期待できることはフェッチした命令間の逐次実行であり，次節で述べる並列実行である．本報告は議論の簡潔性から同時フェッチ命令数を２とする．

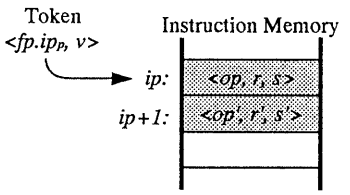

図３：２命令フェッチ

命令ポインタＩＰが指示する命令（以降，第１命令）とその次位置（ＩＰ＋１）の命令（以降，第２命令）を同時にフェッチする．この２命令にデータ依存関係があり，静的にスケジュールされている（すなわち，第１命令の結果行先命令の相対アドレスｓが＋１）ならば，第１命令の演算結果を受けて第２命令が実行可能になることがある．第２命令が第１命令の結果で実行可能であれば，第２命令の遅延発行と第１命令出力のフォーワーディングでこの逐次実行を重畳処理する（多重命令フェッチによる逐次重畳）．

第２命令が実行可能かつその次位置（ＩＰ＋２）の命令と静的にスケジュールされている場合，トークンによる通常の命令フェッチの代わりにプリフェッチを行う．プリフェッチはデータ値未定の擬似的なトークン（以降，プリフェッチトークン）を命令フェッチユニットに直接に送ることで行う．プリフェッチした第１命令が先の第２命令の実行結果で実行可能であれば，この逐次実行も重畳処理する（プリフェッチによる逐次重畳）．

多重命令フェッチと命令プリフェッチとで効率的なスレッド実行が実現可能である．しかし，これだけでは多重命令フェッチの効果は十分ではなく，次節ではフェッチした２命令の並列実行が支援される．

３．３　多重行先トークン

Epsilonの多重行先トークンは繰り返し数と繰り返しステップを持つ．多重行先トークンの支援のためにrepeat-on-input機構が付加されており，この機構がトークンデータをパイプラインに繰り返して投入する．従って，比較的多数の命令へのデータ分配が可能である．

本報告で導入する多重行先トークンはコードブロックの連続位置にあるｎ命令を行先とする．ここで，ｎは命令フェッチの多重度に等しい定数である．制約の強いデータ分配ではあるが，多重命令フェッチとの組み合わせにより特別な機構は不要である．多重行先トークンの導入で同時フェッチした命令の逐次重畳可能な機会が増え，機能ユニットが多重化された場合には並列実行が実現できる．本報告では前節と同様に２行先として議論を進める（図４）．２行先トークンの導入はトークンと命令の行先情報を２ビット（トークン種別ビットと第２命令に対するポートビット）長くする．従って，トークン長は２ビット，命令長は４ビットの増加となる．

さて，３．１節の命令配置で述べていない命令の配置を説明する．図２において，２行先トークンはａとｂの入力に使う．従って，"0:neg"と"1:sqr"，"2:*"と"3:*"の命令はそれぞれ連続位置に配置することになる．前者の配置順は任意であるが，後者はスレッドを構成するために順序が制限される．

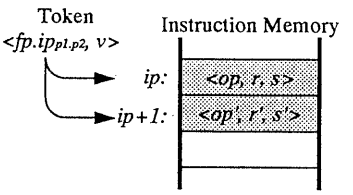

図４：２行先トークン

4．データ駆動プロセッサ

4．1　パイプライン構成

図５に２命令フェッチと２行先トークンを支援するプロセッサパイプラインを示す．ＥＴＳを採用するためにMonsoonパイプラインと同様な構成になるが，テンポラリレジスタや結果トークンの再循環などのない６ステージパイプラインである．次に各ステージの基本的な操作を述べる．

（ＩＦ）命令フェッチステージはトークンかプリフェッチトークンかのいずれかの命令ポインタＩＰが指示する命令とその次位置にある命令の２命令を命令メモリから同時フェッチする．プリフェッチトークンはＰＢステージから直接与えられる擬似的なトークンであり，通常のトークンよりも優先的に処理される．

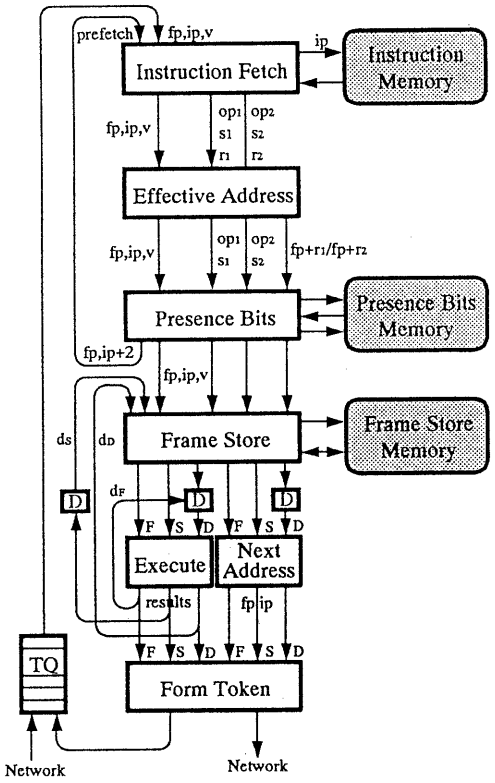

図５：パイプライン構成

（ＥＡ）実効アドレスステージは実効アドレスＦＰ＋ｒ$_1$とＦＰ＋ｒ$_2$を計算する．ｒ$_1$とｒ$_2$はそれぞれ第１命令と第２命令が持つ待ち合わせ位置のアクティベーションフレーム内変位である．ここで２命令が共に待ち合わせを必要とする時，ｒ$_2$はｒ$_1$＋１に等しい．

（ＰＢ）存在ビットステージは待ち合わせ処理，命令発行制御，プリフェッチ制御を行う．待ち合わせ処理と命令発行制御の結果はＦＳステージ以降の操作を決め，プリフェッチ制御は次サイクルのＩＦステージの操作を決める．待ち合わせは実効アドレスに従って存在ビットをフェッチし，更新して書き戻すことで行う．命令発行制御において，第１命令に対しては一般的なデータ駆動と同様であるが，第２命令に対しては同時実行可能か遅延実行可能かで異なる発行タイミングを制御する．プリフェッチ制御はプリフェッチが有効な時，プリフェッチトークンをＩＦステージに送ることで行う．

（ＦＳ）フレームストアステージはＰＢステージの指示に従い，フレームメモリ操作と命令発行を行う．これらの操作で対象となるデータはトークンデータとフレームストア上のデータの他に，次のＥＸステージからフォーワーディングされる２つのデータがある．プリフェッチを行った場合，それを起動した前命令のトークンは抑制される．抑制したトークンの代わりにこれらデータをＦＳステージに戻す．

（ＥＸ）実行ステージでは第１命令，第２命令，遅延実行命令の３命令が多重化された機能ユニットに入力される．それぞれ，図５のＦ，Ｓ，Ｄの各パスに対応する．第１命令の演算結果（ｄ$_F$）はＥＸステージに，第２命令の演算結果（ｄ$_S$）と遅延命令の演算結果（ｄ$_D$）はＦＳステージに送る．また，結果行先（ＩＰ＋ｓ）は演算実行と並行して計算する．ｓは命令が持つ結果行先命令の相対アドレスである．

（ＦＴ）フォームトークンステージはＥＸステージから与えられる演算結果と結果行先などでトークンを生成し，トークンキューかネットワークに送る．１サイクルの最大トークン数は６（２出力命令が同時に３命令実行された場合）である．しかし，本報告ではトークン出力の多重化を行わないため，トークンは逐次化されることになる．

4．2　命令発行とプリフェッチの制御

逐次処理の重畳を実現する命令発行制御とプリフェッチ制御を述べる．これらの制御はＰＢステージで行い，前者がＦＳステージ以降の操作を，後者がＩＦステージの操作を決める．

命令発行制御はＦＳステージからの第２命令発行タイミングを制御する．第２命令が第１命令と同時実行可能であれば，第２命令をＥＸステージのＳパス上に発行する．また，第２命令が第１命令の結果で実行可能（すなわち，逐次実行可能）である場合，第２命令をＤパス上に発行する．すなわち，第２命令は１サイクル後に発行（以降，遅延命令）され，フォーワードされる第１命令の演算結果（d_F）と共にＥＸステージに入力され，実行される．この時，第１命令から遅延命令へのトークンを抑制しなくてはならない．トークン抑制はＦＴステージで行う方法とこれ以前に第１命令の結果行先を無効にする方法がある．ここでは後者の方法をＰＢステージで行うこととする．

プリフェッチ制御はプリフェッチトークンの送出，第２命令からのトークン抑制，プリフェッチした命令へのデータ供給制御を行う．プリフェッチトークンはデータ値が未定，命令ポインタがＩＰ＋２のトークンであり，ＩＦステージでは通常のトークンに優先してプリフェッチトークンを扱う．第２命令の結果トークンの抑制はＰＢステージで結果行先を無効にすることで行う．第２命令からプリフェッチした命令へのデータ供給は，ＥＸステージからＦＳステージに行われなくてはならない．なぜなら，プリフェッチした命令が待ち合わせに失敗した時，供給されるデータをフレームストアに残す必要があるからである．ＦＳステージへのデータ供給は第２命令が並列実行か遅延実行かで異なる．遅延実行した場合は遅延命令出力（d_D），並列実行した場合は第２命令出力（d_S）である．

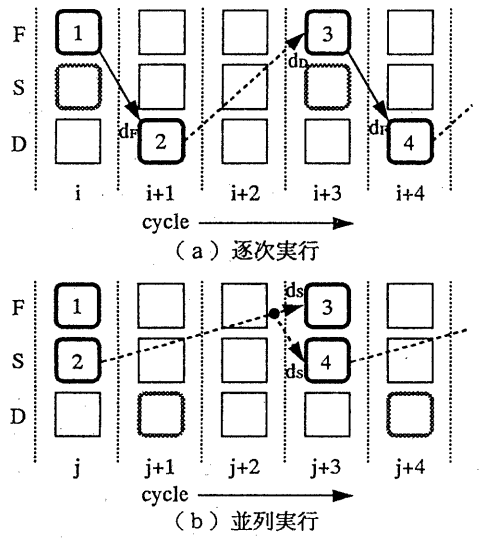

図６：スレッド実行パターン

図６はスレッド実行タイミングを逐次実行と並列実行のそれぞれについて示す．図において，実線矢印はフォーワーデングによるデータ供給，波線矢印はプリフェッチによるデータ供給を表わし，並列実行は２行先トークンが供給される場合である．

各スレッドは現サイクルのＦとＳ，次サイクルのＤを占有する．ただし，ＰＢステージでプリフェッチ制御を行うため，この占有は３サイクル毎である．従って，本パイプラインは逐次実行と並列実行が混在する３スレッドで満たされる．

5．評価

本報告で提案した多重命令フェッチとプリフェッチ，多重行先トークンによる効果をソフトウェアシミュレータを使用して概観する．シミュレータはスケジューリングを施したアセンブリプログラムの実行をトレースし，統計データを出力する．ただし，１プロセッサモデルのシミュレーションであり，機能ユニットにおける資源競合やマルチサイクル命令に対する配慮は行われていない．

図７は一般的なデータ駆動（<1.-.->：単一命令フェッチ，プリフェッチなし，２行先トークンなし）を基準に，２命令フェッチ<2.-.->，２命令フェッチ＋プリフェッチ<2.p.->，２命令フェッチ＋２行先トークン<2.-.d>，２命令フェッチ＋プリフェッチ＋２行先トークン<2.p.d>の各モデルの実行サイクル数比較を示す．例題プログラムはFib10と4Qucenである．ここで，２行先トークンを使用するモデル（<2.-.d>，<2.p.d>）と使用しないモデルとではプログラムサイズが異なり，実行された命令数はFib10で８６％，4Queenで７８％に減少している．また表１にプログラムの特性を示す．

Fib10：スレッド長が多重度の２に近いスレッドがほとんどであることから，２命令フェッチによる逐次重畳の効果が顕著である．プリフェッチによる効果が小さいが，これは実行時に十分なトークン量が存在することにもよる．また，２行先トークンによる効果はプログラムサイズの縮小が主要因であり，並列実行された命令は全実行命令数の８％程度である．結果的に本報告が提案する<2.p.d>では<1.-.->に比べ，４２％にまで改善された．

4Queen：逐次的アルゴリズムを使用したためにトークン量が少なく，ＩＦステージの稼働率はプリフェッチを行わない時に５５％程度，プリフェッチを行った時に６６％程度である．従って，Fib10に比べてプリフェッチによる効果が現われている．結果的に<2.p.d>では<1.-.->に比べ，４８％にまで改善された．

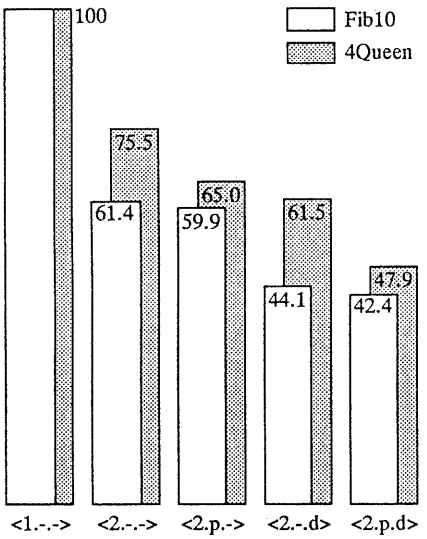

図７：実行サイクル数の比較

表１：FIb10と4Queen

プログラム特性	Fib10	4Queen
逐次・並列実行命令数／実行命令数	42~46%	34~38%
プリフェッチ数／命令フェッチ数	15~21%	27%
２行先トークン数／トークン数	43%	30~32%

６．あとがき

データ駆動の超並列処理向きな特性を失うことなく，スレッド実行の効率化を行うことが本報告の主旨である．本報告ではデータ駆動プロセッサにおけるスレッド処理の効率化を多重命令フェッチとプリフェッチとで支援する方法について考察し，さらに機能ユニットの多重化を前提に同時にフェッチした命令の並列実行を支援する多重行先トークンの導入を行った．多重度２のこれらの導入により，単一命令フェッチの一般的なデータ駆動プロセッサ上のプログラム実行サイクルを５０％以下にまですることが可能である．これはスレッド処理におけるトークン循環オーバーヘッドの抑制や２行先トークンによるプログラムサイズの縮小が主な要因と考えられる．

本研究の次段階としては，現実的な課題（機能ユニットの多重化選択と実行時の資源競合回避など）を含むマルチプロセッサ上でのシミュレーション，テストベッド上での詳細な評価である．さらに重要な課題として，構造体メモリや相互結合網などのデータ駆動システムとしての検討が必要である．またプログラミング言語と最適化を中心としたプログラム翻訳については，TAM[15]や他の計算機上における研究が本モデルに適用可能であると考える．しかしながら，これらの重要性に対する認識は十分にある．

参考文献

[1] Veen,A.H. : "Dataflow Machine Architecture", ACM Computing Surveys, **18**,4, pp.365-396(Dec. 1986).

[2] Gaudiot, J-L. and Bic, L.(eds) : "Advanced Topics in Data-Flow Architecture", Prentice-Hall(1991)

[3] Gurd, J.R., Kirkam, C.C. and Watson, I. : "The Manchester Prototype Dataflow Computer", CACM, **28**, 1, pp.34-52(Jan. 1985).

[4] Arvind and Nikhil, R.S. : "Executing a Program on the MIT Tagged-Token Dataflow Architecture", IEEE Trans. Comput., **39**, 3, pp.300-318(March 1990).

[5]Hiraki,K., Nishida,K., Sekiguchi,S., Simada,T. and Yuba,T.: "The SIGMA-1 DataflowSupercomputer: A Challenge for New Generation Supercomputing Systems", J. Inf. Process., **10**, 4, pp.219-226 (1987).

[6]Sakai, S., Yamaguchi, Y., Hiraki, K., Kodama, Y. and Yuba, T. : "An Architecture of a Dataflow Single Chip Processor", ISCA89, pp.46-53(June 1989).

[7]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アーキテクチャ最適化に関する考察", 情処学論, **30**, 12, pp.1562-1572 (1989-12) .

[8]Grafe, V.G., Davidson, G.S., Hoch, J.E. and Holmes, V.P. : "The Epsilon Dataflow Processor", ISCA89, pp.36-45(June 1989).

[9]Papadopoulos, G.M. and Culler, D.E. : "Monsoon : an Explict Token-Store Architecture", ISCA90, pp.82-91(May 1990).

[10] Papadopoulos, G.M., and Traub, K.R. : "Multithreading : A Revisionist View of Dataflow Architectures", ISCA91, pp.342-351(May 1991).

[11] Papadopoulos, G.M. : "Implementation of a General-Purpose Dataflow Multiprocessor", Pitman(1991).

[12] Iannucci, R.A. : "Toward a Dataflow/von Neumann Hybrid Architecture", ISCA88, pp.131-140(June 1988).

[13] Nikhil, R.S. and Arvind : "Can dataflow subsume von Neumann computing?", ISCA89, pp.262-272 (June 1989).

[14] Nikhil, R.S., Papadopoulos, G.M. and Arvind : "*T : A Multithreaded Massively Parallel Architecture", ISCA92, pp.156-167(May 1992).

[15] Culler, D.E., Sah, A., Schauser, K.E., von Eicken, T. and Wawrzynek, J. : "Fine-grain Parallelism with Minimal Hardware Support : A Compiler-Controlled Threaded Abstract Machine", ASPLOS91, pp.164-175(April 1991).

[16] 寺田秀文 : "並列処理計算機ＭＤＦＭにおける命令レベル並列処理の検討", 43回情処全大, pp.129-130 (1991-3) .

[17] Johnson,M. : "Superscalar Processor Design", Prentice-Hall(1991).