社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
ICD98-23, CPSY98-23, FTS98-23(1998-04)

大規模データパスプロセッサにおける命令供給システム

中村 友洋, 吉瀬 謙二, 辻 秀典, 安島 雄一郎, 高峰 信, 田中 英彦

東京大学大学院 工学系研究科

東京都 文京区 弥生 2-11-16 工学部 10 号館 360 号室

03-3812-2111 内線 6752

hiro@mtl.t.u-tokyo.ac.jp

あらまし

筆者らが提案をしている次々世代のマイクロプロセッサ・アーキテクチャである大規模データパス・アーキテクチャは、大規模な投機的処理をその特徴としているが、その際に問題となる命令の供給システムについて、問題点を考察する。そして、そのハードウェア的解決方法として、コントロールフロー先行展開機構およびその分岐処理機構について述べる。ここでは、分岐履歴情報の伝搬を使った効率的なフェッチシステムを提案する。またソフトウェア的な解決方法については、プロファイルを用いて命令列をバースト的にメモリから転送できるような形式にするストリーミングについて述べる。それぞれに関して簡単な評価を行なう。

キーワード 次世代マイクロプロセッサ, コントロールフロー先行展開, 大規模データパスプロセッサ

Fetch System of Very Large Data Path Processor

Tomohiro Nakamura, Kenji Kise, Hidenori Tsuji, Yuichiro Ajima,
Makoto Takamine, Hidehiko Tanaka

Graduate school of engineering, University of Tokyo

2-11-16 Yayoi Bunkyo-ku Tokyo-to, JAPAN

+81-3812-2111 ext. 6752

hiro@mtl.t.u-tokyo.ac.jp

Abstract

This paper presents the Fetch System of Very Large Data Path (VLDP) Processor which we have proposed for the next generation micro processor architecture. A special feature of VLDP processor is very large scale speculative processing, which affect its instruction fetch system extremely. In this paper we study hardware and software of fetch system. In hardware feature, we propose Control-Flow Pre-Construction mechanism which use branch history information effectively. In software feature, we propose Instruction Streaming which is able to transmit instructions from memory to processor in burst mode. We evaluate each of them briefly.

key words Next Generation Microprocessor, Control-Flow Pre-Construction, VLDP Processor

1 はじめに

次世代もしくは次々世代のマイクロプロセッサ・アーキテクチャに関する研究が盛んである。マイクロプロセッサの性能向上に対する要求も衰えていない。新しいマイクロプロセッサ・アーキテクチャを考える際には、性能に関する指標を1桁上げる程度のものを考えて行かなければ、時代の流れに乗り遅れてしまう危険が高まっている。

さて、マイクロプロセッサのこれまでの歴史を振り返ってみると、新しいアーキテクチャはつねにデバイス技術の進歩と共にその形を変えてきた。今後のマイクロプロセッサ・アーキテクチャについても、それは変わらない。ここ数年はデバイス技術的には、集積度の向上と動作周波数の向上が当初の予定通り順調に進展してきているが、今後を見ると様々な障壁が待ち構えている。特に、より一層大規模化するデバイスをどのように利用するかは最大の問題である。自由に利用できるデバイスが増加するということは、それだけアーキテクチャ選択の幅の自由度も上がるということであるので、様々な方向性が見られるようになった。

筆者らは、その中で1つのCPUとしての性能向上をより一層目指すという方向性のもと[12]、大規模データパスプロセッサ(Very Large Data Path processor)の研究開発を行なっている。このプロセッサはデータパス部分を大規模化し、理想的にはデータフローグラフをハードウェアにマップして実行を行なっていくことで、各プログラムに対する処理の限界性能を目指すものである[8]。

このようなプロセッサ・アーキテクチャを開発するには、さまざまな問題があるが、本稿ではその中の命令供給システムに関する問題に関して検討し、その問題を解決するための方策について述べる。第一義的な目的は、プロセッサの実行部に有効な命令を可能な限り供給することである。

2 研究の背景

米国半導体工業会調査のデバイス技術に関するロードマップ[4]によれば、今後想定されるデバイス技術は表1の通りである。表1によれば、2007年には現在の主なプロセッサが利用しているトランジスタ数(数百万〜一千数百万)の10倍程度に相当する数億トランジスタを1チップで利用できることになる。このような大規模なデバイスをどのように利用するかが今後のプロセッサを考える上で重要である。

表1: デバイス技術ロードマップ

年	1998	2001	2004	2007
最小加工寸法 (μm)	0.25	0.18	0.13	0.10
MPU Tr 数 (M/chip)	28	64	150	350
chip 内周波数 (MHz)	450	600	800	1000
DRAM bit 数	256M	1G	4G	16G

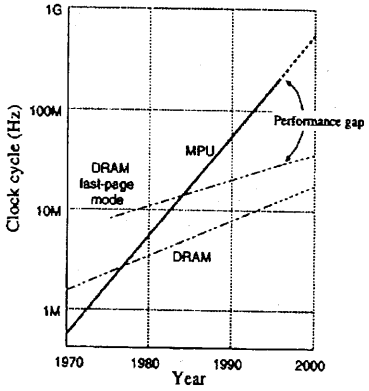

図1: MPUおよびメモリの動作速度の変化

大規模データパス(VLDP)アーキテクチャでは、このような大規模なデバイスを大規模なデータパスをもつプロセッサとして利用する。これによって、それぞれのプログラムに対してその実行における限界性能を引き出すことを目指している。ハードウェア的には、多数の演算器とそれらの結合網をもち、論理的には、このネットワーク(ALU-Net)上にデータフローグラフをマップし、データを流すことで実行を行なう。つまり、プログラムの本質部分を抽出し、それを最適に実行する機構をもたせたものが、大規模データパス・アーキテクチャである。

しかし、ALU-Netを有効して命令レベル並列度を高めるほど、より大量の命令をメモリからプロセッサに供給する必要が生じる。そのためには、効率的なフェッチシステムが必要不可欠である。図1[3]に示すように、DRAMの速度向上率がlogicに比べて低く、今後もこの傾向が続きメモリウォールと呼ばれる問題がさらに大きな影響を及ぼすと考えられる。以上の観点から、本稿では大規模データパス・アーキテクチャに必要なフェッチシステムについて考察する。

なお、本稿で評価に用いたサンプル・プログラムはSPECint92からcc1, compress, espresso, scとdhrystoneベンチマークの5つである。またSPARC V8命令セットを使用している。

3 大規模データパス・アーキテクチャ

大規模データパス・アーキテクチャは大規模なハードウェアを利用して、多数の演算器とその結合網上に命令をデータ依存関係に基づいてマップし、効率的に実行を行なっていくアーキテクチャである。大規模データパス・アーキテクチャのブロック図は図2の通りである。以下では、各ブロックについて簡単に説明をする。

3.1 コントロールフロー先行展開

コントロールフロー先行展開はVLDPプロセッサにおけるフェッチ機構である。コントロールフロー先行展開では、動的な分岐情報を活用し、投機的に資源の許す限り命令をフェッチする。フェッチの際には、コントロールフロー・ツリーの構築を行ない、この情報を更新していくことでフェッチが進む。大量の命令をALU-Netに供給するために、プログラムの実行点に比べてかなり先行してフェッチを行なう。そのため、図3のように、フェッチ開始点に近い部分(図の上部)では可能な限りEagerにフェッチを行ない(図3の実線矢印)、分岐予測ミスによるようなペナルティを回避する。一方、フェッチ開始点からもさらに先の命令については、動的な分岐情報などを活用して選択的にフェッチを行なう。その結果、図3の点線で示されるような、この時点ではまだフェッチしない命令が徐々に出てくることになる。

3.2 データパス先行展開

データパス先行展開ではコントロールフロー先行展開から送られてくる制御依存関係の解消された命令列に対してデータ依存解析を行なう。多数の演算器を接続したALU-Net上に命令をマップしデータを流すことで実行を行なうには、従来のようなレジスタを用いた集中型のデータ管理では、スケーラビリティに乏しく、大規模データパス・アーキテクチャには不適当である。そこで、大規模データパス・アーキテクチャでは、ローカルなデータのやりとりをALU-Net上の接続として実現する。このような演算前後にALU-Netの制御を行なうのがデータパス先行展開である。個々の演算は前の演算器からのデータ出力によって開始され、実行時には集中的な管理を行なう必要がなくなる。図4は、複数の分岐命令をまたいで命令フェッチを行なった場合に、フェッチされた命令間に存在するデータ依存関係がそれらの命令間にある割合を調べたものである。これによれば、大規模な命令展開を行なった場合、理想的には70 %以上の命令間データ転送がローカルに行なえることが分かる。

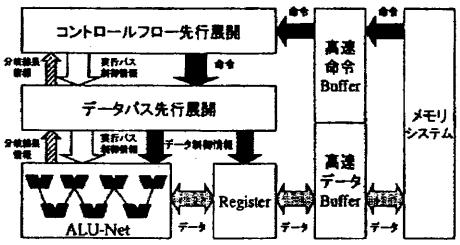

図2: VLDP アーキテクチャ・ブロック図

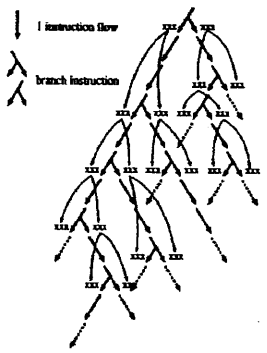

図3: コントロール・フロー先行展開のイメージ

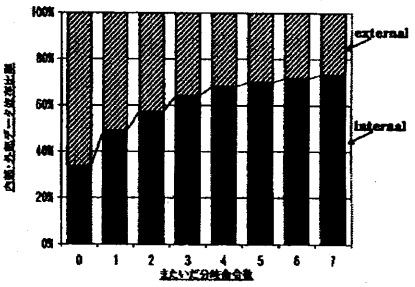

図4: ローカルなデータ・アクセス

3.3 ALU-Net

ALU-Netは多数の演算器とそれらの接続網で構成される。しかし、すべての演算器間に接続をもたせると、その数が爆発的に増えてしまうため、ある限られた自由度のネットワークとすることにしている。命令ウィンドウ・サイズを40、演算器数を100、1つの演算器が最大3つの他の演算器との間に接続網を持っているとした場合に、62 %のデータ転送が接続網に吸収されることが分かっている[6]。

4 大規模命令展開とフェッチシステム

4.1 命令展開方式

大規模データパス・アーキテクチャのような大規模な命令先行展開を行なう場合には、いくつかの展開方式が考えられる。

Single Path 分岐予測機構を用いて、1つのコントロールフロー・パスの命令だけをフェッチする方式。分岐予測成功率が非常に高い場合は、フェッチされた命令が有効な命令である確率が高くなるので命令ウィンドウを有効に利用できるが、分岐予測性能が低い場合には非常に資源の無駄使いとなる。この方式は分岐予測機構の性能に大きく影響を受けるため、現在の80～95％程度の分岐予測成功率しかない分岐予測機構を用いて大規模化することは現実的ではない。

コントロールフロー先行展開 大規模データパス・アーキテクチャのフェッチ機構。選択的に複数パスの投機的フェッチを行なう方式。分岐予測機構から、各分岐先への分岐確率値を動的に予測し、その確率値によって1つ以上のパスの命令展開を行なう。原理的には disjoint eager-execution[5] と同じである。

Eager Fetch 各分岐命令ごとに分岐成立／不成立の両方のパスへ命令を展開し、すべてのパスのフェッチを行なう方式。大規模な展開を行なうと、不要な命令をフェッチしてしまう率が高くなる。分岐予測機構は不要であり、基本的に必要な命令は必ずフェッチされていることになるが、SPARC コードにおける JMPL 命令のようなレジスタ間接アドレッシクングで分岐先が決定されるような分岐命令に対しては、分岐予測を行なわなければならない。

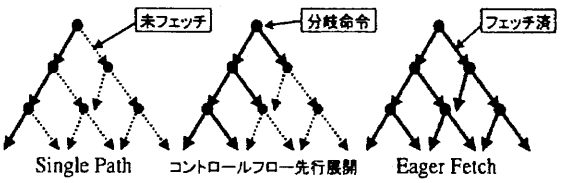

図 5: 命令の先行展開方式

4.2 大規模先行展開と分岐予測機構

コントロールフロー先行展開が行なうような大規模な命令先行展開において、従来ではあまり問題とならなかった次のような新たな問題が分岐予測機構に生じる。

4.2.1 レジスタ間接アドレッシングの問題

分岐命令には表2のような4種類がある。ここで特に問題となるのは、この中のアドレス計算がレジスタ間接である JMPL, Ticc 命令である。Bicc, CALL 命令の分岐先が図6(a) のように2つ以下であるのに対して、JMPL, Ticc 命令の場合、分岐先がその時点でのあるレジスタの値によって計算される場所となり、図6(b) のように分岐先が複数になることがある。よって、命令展開の際にフェッチすべき命令をどのように選択するかが大きな問題となる。なお、Ticc 命令はトラップ命令であり、Ticc 命令で分岐条件が成立した場合は、命令展開が Ticc 命令で切れるので、以下では JMPL 命令だけを対象として考える。

分岐命令	分岐条件	アドレス計算	分岐方向
Bicc	条件付き	PC 相対	図 6(a)
CALL	無条件	PC 相対	図 6(a)
JMPL	無条件	レジスタ間接	図 6(b)
Ticc	条件付き	レジスタ間接	図 6(b)

表 2: 主な分岐命令の特徴

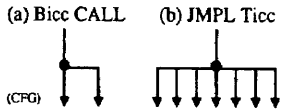

図 6: 分岐命令における制御の流れ

そこで、筆者らはこれらのレジスタ間接アドレッシングで分岐先が決まるような分岐命令に対する分岐予測機構として Multi BTAC(Multi Branch Target Address Cache) を提案している [9]。通常 JMPL 命令のようなレジスタ間接アドレッシングの分岐命令の分岐先予測には BTAC と呼ばれるテーブルが用意され、これを用いて分岐先を予測することが多い。BTAC は図7のように、JMPL 命令の PC 値などを使った TAG エリアとその分岐命令の前回の分岐先を保存しておく Target#1 エリアの2エリアを1エントリとして、128～2048 エントリ程度ハードウェアに用意されていることが多い。しかし、BTAC を用いたのでは第4.1節で述べたコントロールフロー先行展開が行なう図5のような命令展開の際に、JMPL 命令をまたいだ命令展開が1方向だけに限定されてしまい、展開効率が悪化することが考えられる。そこで、図

5の下のようなTargetエリアが#1～#nまでn個あるようなテーブルをもつMulti BTACを用意することでこの問題を解決する。図8は横軸にMulti BTACの1エントリに含まれるTargetエリアの数(Multi Level)を取り、縦軸に分岐先予測成功率を取ったものである。ここでは、限界性能を求めるために、十分なエントリ数をとり、フルアソシアティブとしている。図8によれば、Multi Level=1、つまりBTACの場合は平均で76 %の予測成功率であるが、Multi Level=2となるとこれが85 %になることが分かる。つまり、コントロールフロー先行展開のような複数のパスを選択的に投機フェッチしていく命令展開機構では、Multi BTACのような分岐先予測機構が有効であることが非常に分かった。なお、詳細な評価は文献[9],[11]を参照して頂きたい。

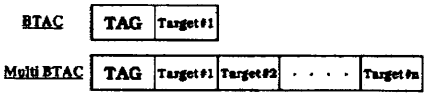

図 7: BTACとMulti BTAC

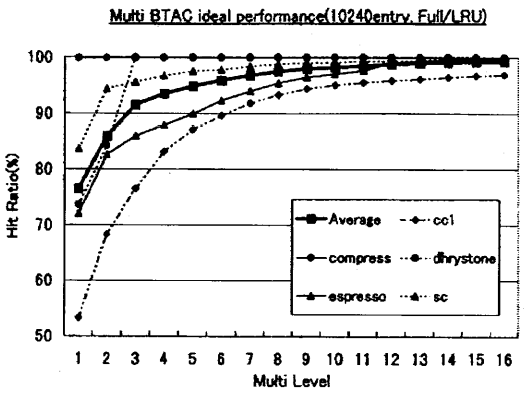

図 8: Multi BTACのmulti level別性能

4.2.2 分岐履歴情報の更新の問題

動的な分岐予測においては、通常BHT(Branch History Table)やBTACのように分岐履歴情報を使って分岐予測を行なう。しかし大規模な命令先行展開をおこなうコントロールフロー先行展開のような機構でこのような動的な分岐予測機構を利用する場合、予測時に使用する分岐履歴情報が古いものである可能性がある。つまり、図9のような命令流が実行される場合、同じ分岐命令が3回繰り返し実行されるが、フェッチポイントにある一番下の分岐命令の分岐予測は、真中の分岐命令の分岐結果が未定のため、一番上の分岐命令の分岐結果までしか反映されていない分岐履歴情報をもとに分岐予測をしなければならない。

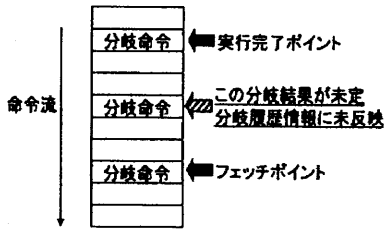

図 9: 実行完了ポイントとフェッチポイント

そこで筆者らは、命令先行展開の際にコントロールフローパスごとに分岐履歴情報を更新しながらフェッチをする場合(図10左)と、フェッチ開始ポイントにおける分岐履歴情報をそのまま使ってフェッチをする場合(図10右)の比較を行なった[10]。図10の四角に囲まれた"00"などの数字は分岐履歴情報bitを表しており、ここでは●はすべて同じ分岐命令、矢印はコントロールフローを表している。

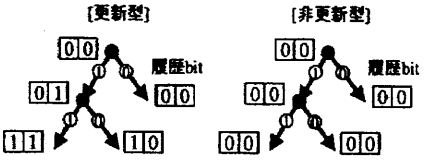

図 10: 分岐履歴情報の更新／非更新

図11は、いずれも2048エントリ／4ウェイ・セット・アソシアティブのBTACとBHTを利用した場合に、図10の更新型と非更新型における分岐予測成功率の差を示したものである。縦軸は更新型の分岐予測成功率を基準にしている。よってグラフの下の方にいくほど更新型が非更新型に対して優れていることを示している。横軸は命令の展開規模で、ここではまたいだ分岐命令の数をとっている。

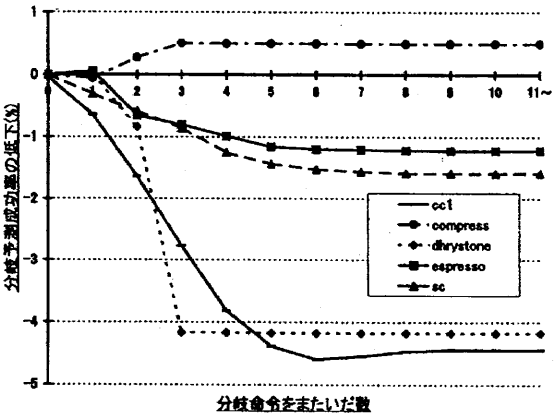

図 11: 更新型と非更新型の性能差

図11によれば、ほとんどの場合、非更新型にくらべて更新型の方が分岐予測成功率の点で1～4 %程度優れており、この差は命令先行展開の性能に大きく影響を与えることが分かっている[10]。

5 コントロールフロー先行展開

第4節での考察から、コントロールフロー先行展開のハードウェア的な構成は図12のように考えられる。図中、実線の矢印は命令の移動を、網掛け矢印は制御情報の移動を表している。

コントロールフローパス管理 フェッチされた命令のコントロールフローを保存しておく機構である。その動作は次の通り。

1. フェッチされた命令に対してコントロールフロー・パスごとに"色付け"をし、パスごとの制御依存関係情報を登録する。
2. 制御依存関係が解消された命令流をデータパス先行展開に送る。
3. 実行が完了し分岐結果が判明したら、不要となったパスを制御依存関係情報から特定し、データパス先行展開やALU-Netから不要となったパスを削除し、あわせて制御依存関係情報を更新する。

ここでポイントとなるのは、制御依存関係が解消された命令流をそれぞれデータパス先行展開に送ることにより、複数のコントロールフロー・パスに対して別々の資源を割り当てて[1]並列に投機実行することが可能となることである。

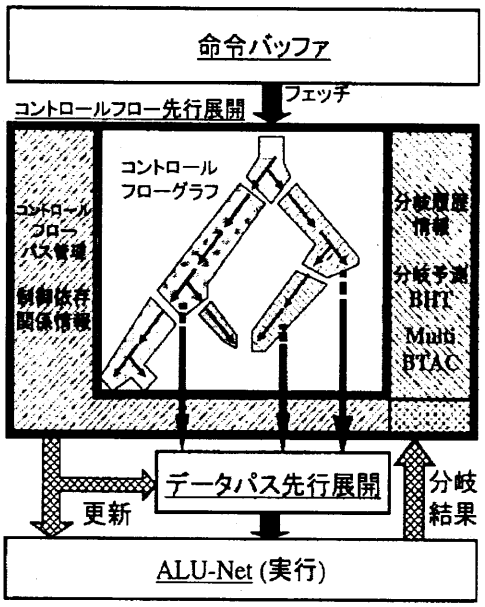

図 12: コントロールフロー先行展開の処理の流れ

[1]例えば別々のリネームレジスタ・セット

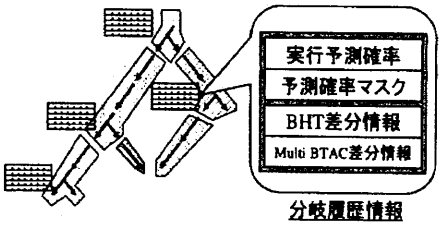

図 13: 分岐履歴情報

分岐履歴情報管理／分岐予測 コントロールフロー・パスごとの分岐履歴情報を管理し、それに基づいて分岐予測を行なう機構である。分岐履歴情報は各分岐命令ごとに作成され、図13のように、そのパスが正しいパスである予測確率(実行予測確率)、BHT, Multi BTACの各パスごとの差分情報などが格納される。分岐予測としては、第4.2.1節と第4.2.2節で述べたBTACを拡張したMulti BTAC, 更新型のBHTを用いる。その動作は次の通り。

1. 実行予測確率の高いポイントから分岐予測に基づいてフェッチを行ない、分岐履歴情報を更新する。
2. 分岐予測テーブル(BHT, Multi BTAC)は更新されないので、BHTやBTACを仮想的に更新した場合の差分情報を登録する。
3. 実行が完了し分岐結果が判明したら、それに基づき分岐予測テーブル(BHT, Multi BTAC)を更新し、あわせてコントロールフロー先行展開中の各分岐命令について分岐履歴情報・差分情報の更新を行なう。

実行予測確率を正確に計算するには浮動小数点計算が必要となるが、処理が重くなり現実的ではないので、コントロールフロー先行展開ではシフト・マスク計算のみで近似的に実行予測確率を計算する。その計算方法は図14の通りである。

1. 各分岐命令ごとに、実行予測確率にマスク値とのANDをとり、右シフト(上位bitには0を埋める)した値を現時点での実行予測確率とし、この値の大きいものから後続命令のフェッチを行なう。図14の例では、各エントリを8bitで表現している。
2. フェッチされたパスに対して実行予測確率を分岐予測機構から求め、各パスの実行予測確率に登録する。

3. 1つの分岐命令の結果が判明したら、マスク値をそれぞれ1bitずつ右にシフトし、最上位bitに1を埋める。

この方式のポイントは、実行予測確率の伝搬がマスク値のシフトだけで実現できる点にある。これによって、大規模な命令先行展開も可能となる。

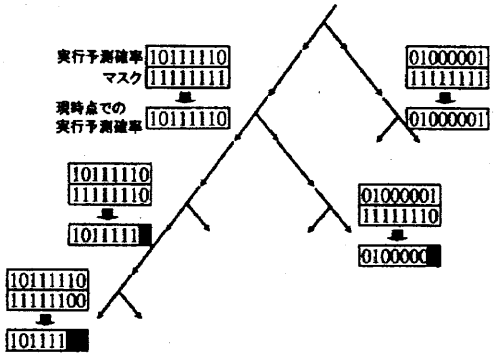

図 14: 分岐履歴情報の計算

6　命令ストリーミング

コントロールフロー先行展開をサポートするソフトウェア技術についても考察を行なっている。コントロールフロー先行展開では、大規模な命令先行展開を行なうため、メモリとの間に高スループットで広いバンド幅をもったバスを必要とする。

ところが、バンド幅については、off chipのメモリデバイスを使う場合、今後も大幅な向上は望めない。現在、これを解決するための研究が様々行なわれている[2]。例えば、PPRAM[7]では、LogicとDRAMを1チップにインテグレートすることで、メモリ・バンド幅を稼ぐことが可能である。DRAMの混載に限らず、大規模化するデバイスを大容量のオンチップ・キャッシュに利用するアプローチも同じである。

しかし、いずれにしてもいずれにしても、メモリ・アクセス・レイテンシの問題が生じる可能性がある。DRAMの場合は構造上レイテンシが大きいし、大容量のオンチップ・キャッシュでは、大容量化に伴う複雑なアドレス変換によってこれまでのような1サイクル・アクセスが実現できるかどうかは疑問である。ラインサイズを大きくしてこれを回避することは可能であるが、この場合キャッシュミスが増大する恐れがある。

一方、スループットに関しては、今後の向上が期待できる。例えばDirect RAMBUSを利用すればoff chipでも1.6GByte/secのスループットが得られる[1]。現在でもSDRAMなどのように、最初のアクセスを除いて1サイクル・アクセスが可能なメモリもあり、今後もこの傾向は続くと考えられる。

つまり、コントロールフロー先行展開を使った命令フェッチにおいても、メモリへのアクセス・レイテインシを隠蔽するために、高いスループットを使って命令をバースト転送することが有効であることが分かる。

通常の命令フェッチでは図5のSingle Pathのような命令展開が行なわれることが多いため、コンパイラによってうまくコードを生成することができれば、命令フェッチにおけるメモリ・アクセスは逐次的になる。しかし、コントロールフロー先行展開のようにコントロールフロー・ツリーの各部分を実行される確率の高い順にフェッチするような場合には、命令フェッチにおけるメモリ・アクセスはメモリの各所に分散し、バースト転送可能なサイズもベーシック・ブロック程度(通常5命令程度)にとどまり、バースト転送には向かない可能性が高い。

そこでバースト転送に向いたコードを考える。図15はその一例である。図15はコントロールフロー・グラフを表しており、それぞれの網掛け部分が命令ストリーム・コードである。この網掛け部分の命令を1セットの命令流としてまとめておくことで、大きなバースト転送が可能となる。

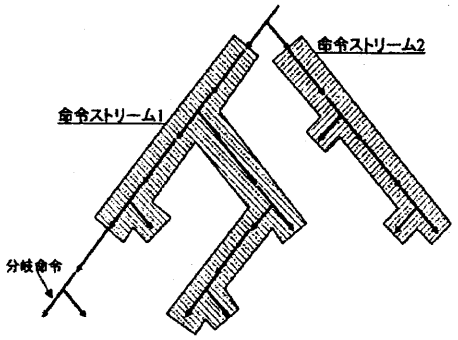

図 15: 命令ストリームの例

命令ストリームの作り方は様々考えられるが、1つの方法はプロファイルを利用し、静的にコントロールフロー先行展開と同様の処理をして実行予測確率の高いフローを1セットにしていくことである。その他にも、コンパイラによるソースコードの情報を用いた最適化戦略も考えられる。

ここでは、プロファイルを用いてストリームコードを作り、メモリ・アクセス回数の減少を調べた。ここでは、1ストリームのフェッチによって、実際に実

行された命令数(有効命令数)を示す。なお、各パラメータを次のように設定した。

- 1ストリームのサイズは1024Byte以内(最大256命令に相当)
- 1000万命令のトレースデータをプロファイル・データとして利用
- パスごとの実行予測確率は浮動小数点計算

この結果は表3の通りである。

サンプルプログラム	有効命令数
cc1	23.2
compress	52.3
dhrystone	40.1
espresso	40.5
sc	25.6

表3: 1ストリームのフェッチでの有効フェッチ命令数

7 おわりに

本稿では、筆者らが研究開発を行なっている大規模データパス・プロセッサのフェッチシステムの設計について考察を行なった。大規模な投機フェッチは、命令レベルの並列性をより多く利用しようとする場合には不可欠であり、本稿ではそのような投機フェッチを行なう際に生じる問題を調査した上で、選択的な投機フェッチを効率的に行なうコントロールフロー先行展開について述べた。

今後はすべての機能モジュールを統合したシミュレーションによる評価を行なう。

謝辞

本研究の遂行にあたり、日本学術振興会の特別研究員制度(「プログラム解析に基づく次世代マイクロプロセッサアーキテクチャの研究」)および文部省科学研究費(一般研究(B)課題番号07458052「大規模データパスプロセッサの研究」)のご支援を頂きました。ここに感謝の意を表します。

参考文献

[1] Crisp, R.: Direct RAMBUS Technology: The New Main Memory Standard, *IEEE MICRO*, Vol. 17, No. 6, pp. 18–28 (1997).

[2] Doug Burger, James R.Goodman, A. K.: Limited Bandwidth to Affect Processor Design, *IEEE MICRO*, Vol. 17, No. 6, pp. 55–62 (1997).

[3] Kumanoya, M., Ogawa, T., Inoue, K.: Advances in DRAM Interfaces, *IEEE MICRO*, Vol. 15, No. 6, pp. 30–36 (1995).

[4] Semiconductor Industry Association(SIA): *The National Technology Roadmap for Semiconductors* (1994).

[5] Uht, A. K. and Sindagi, V.: Disjoint Eager Execution: An Optimal Form of A Speculative Execution, *IEEE 28th International Symposium on Microarchitecture*, pp. 313–325 (1995).

[6] 吉瀬謙二, 中村友洋, 辻秀典, 安島雄一郎, 田中英彦: ALU-NETを用いることによるデータ移動の効率化, 第56回情報処理学会全国大会 2N-1, Vol. 1, pp. 109–110 (1998).

[7] 村上和彰, 吉井卓, 岩下茂信: 21世紀に向けた新しい汎用機能部品PPRAMの提案, 情処研究会 ARCH 108-8, Vol. 94, No. 91, pp. 1–8 (1994).

[8] 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 田中英彦: 大規模データパス・プロセッサの構想, 情報処理学会研究報告計算機アーキテクチャ研究会 97-ARC-124, Vol. 97, No. 61, pp. 13–18 (1997).

[9] 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 田中英彦: 分岐アドレス予測機構の比較検討, 第55回情報処理学会全国大会 3F-5, Vol. 1, pp. 20–21 (1997).

[10] 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 田中英彦: 大規模な投機的処理における分岐制御機構, 第56回情報処理学会全国大会 5N-8, Vol. 1, pp. 173–174 (1998).

[11] 辻秀典, 中村友洋, 吉瀬謙二, 安島雄一郎, 田中英彦: 大規模データパス・プロセッサにおけるフェッチ機構の検討, 情報処理学会研究報告計算機アーキテクチャ研究会 97-ARC-126, Vol. 97, No. 102, pp. 37–42 (1997).

[12] 田中英彦: ここいらで、計算機アーキテクチャを再考しよう, 情処研究会 ARCH 108-6, Vol. 94, No. 91, pp. 30–40 (1994).