VLDP アーキテクチャにおける実行パス制御機構

高峰　信[†]　中村友洋[†]　吉瀬謙二[†]
辻　秀典[†]　安島雄一郎[†]
坂井修一[†]　田中英彦[†]

従来のプロセッサ・アーキテクチャでは、並列性の利用に関して性能の限界が指摘されている。我々は、投機的に多数の命令をフェッチし命令間の制御依存関係・データ依存関係を解消した後、ネットワークで接続された複数の演算器で命令を実行する VLDP プロセッサ・アーキテクチャを提案し研究・開発を行なっている。VLDP プロセッサ・アーキテクチャの中で、命令間のデータ依存関係を解析し、演算器への命令の割り付けと実行を制御する実行パス制御機構をデータパス先行展開と呼ぶ。本稿では、データパス先行展開を従来の技術で実現した場合のモデルを示し、その動作と実装時の問題点について考察を行う。また、データ依存解析のソフトウェア支援の可能性について述べる。

The data path construction unit on VLDP architecture

Makoto Takamine,[†] Tomohiro Nakamura,[†] Kenji Kise,[†]
Hidenori Tsuji,[†] Yuichiro Ajima,[†] Shuichi Sakai[†]
and Hidehiko Tanaka[†]

The performance of usual processors shows a tendency to saturation because of the limit in exploiting instruction level parallelism. Our research project aims at developing a new architectural model: Very Large Data Path architecture with technology which fetches many instructions speculatively, dissolves the control and data dependences and executes instructions in many ALUs connected by a network. VLDP architecture has the unit which dissolves the data dependences, controls allocation and controls execution of the instructions. We call the unit the data paths construction unit. In this paper we show a model of the data paths construction unit based on the traditional technologies and discuss the problems of the model. And we show the possibility of the software support of the data dependence dissolution.

1. はじめに

近年、プロセッサの高速化に対する要求は高まるばかりであるが、従来のアーキテクチャでは、命令レベル並列度の利用に関する限界が指摘されている。3 年で 4 倍と言われているような、半導体デバイス技術の進歩がほぼ続いている現状を見ると、この限界を破るには、大規模な VLSI を積極的に活用する新しいアーキテクチャを考えることが重要である。そのような背景から、我々は、ネットワークで接続した多数の演算器 (ALU-Net) 上に大規模なデータパスを構築し、複数の命令を並列処理するプロセッサ (VLDP プロセッサ) を提唱し、研究・開発を行なっている[1)]。VLDP プロセッサ中の機構で、投機的にフェッチした命令間のデータ依存関係を解析し、ALU-Net 上に複数のデータパスを構築する機構をデータパス先行展開と呼ぶ。本稿では、データパス先行展開を従来のアーキテクチャ技術で実現した場合のモデルを示し、その動作と実現時の問題点について考察する。また特に、データパス先行展開の役割の 1 つであるデータ依存解析のソフトウェア支援の可能性について述べる。

2. VLDP アーキテクチャ

VLDP アーキテクチャは、多数の演算器をネットワークにより相互接続した ALU-Net を用いて効率の良いデータ処理を行なうことを目指す。従来のアーキテクチャでも、独立した命令を並列に演算するために、数個の ALU を同時に利用するスーパスカラ方式や VLIW 方式が提案されてきた。一方、VLDP アーキテクチャはネットワークで結んだ数十〜数百の ALU を持ち、より大規模な並列処理を目指している。VLDP アーキテクチャでは、スループットの大きい ALU-Net に対して十分な命令を供給するため、依存関係が解消され

† 東京大学大学院 工学系研究科
Graduate school of Engineering, The University of Tokyo

た多数の命令列を実行に先駆けて作成する。また、投機実行の規模が大きくなるにつれ、分岐予測ミス時のペナルティも大きくなる。このペナルティを削減するため、複数パスの投機実行を行う。このような面からは、VLDP アーキテクチャとは、積極的に複数パスの大規模な投機実行を行うアーキテクチャであるといえる。VLDP アーキテクチャの実現には、制御依存関係を解消した複数のパスを用意する投機フェッチ機構と、各パス毎のデータ依存関係を解析して、ALU-Net 上に実行パスの構築を行なう実行パス制御機構が必要である。図 1に VLDP アーキテクチャのブロック図を示す。

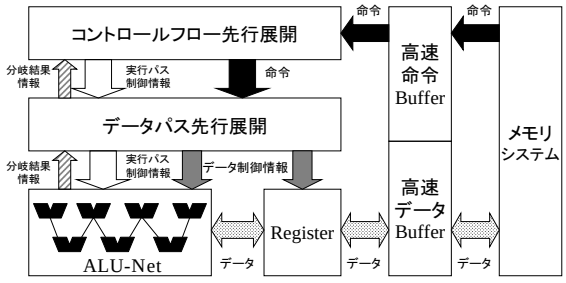

図 1　VlDp アーキテクチャのブロック図

VLDP アーキテクチャの各機構[1)]を簡単に説明する。

(1)　コントロールフロー先行展開
(2)　データパス先行展開
(3)　ALU-Net

コントロールフロー先行展開では、複数のパスを投機的にフェッチし、制御依存関係に基づいた情報を付加する。データパス先行展開では、コントロールフロー先行展開で付加された制御情報をもとに、各制御パス毎に命令間のデータ依存関係を解析し、多数のデータパスを作成する。さらに、作成したデータパス情報に従って各命令の実行情報を得た後、ALU-Net への命令の割り付けと実行パス制御を行なう。ALU-Net は多数の ALU がネットワークによって接続された演算機構である。1 つの ALU が作成した演算結果を、レジスタやフォワーディングの機構を介さずに別の ALU で利用することができ、高速なデータ転送が可能となる。ALU-Net 内でのデータの受渡しについては、10×10 の ALU-Net で命令ウィンドウを 40、次段の ALU への接続の個数を 3 とした場合には、全体の 60% 程度が ALU-Net 内部で吸収できることが報告されている[2)]。次の章では、データパス先行展開について、その動作を詳しく述べる。

3. データパス先行展開

3.1 基本動作と基本モデル

図 2 にデータパス先行展開の動作を模式的に表す。

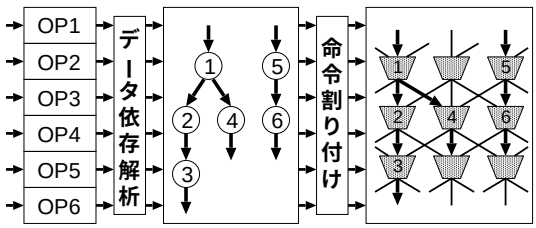

図 2　データパス先行展開のイメージ

コントロールフロー先行展開では、各命令がどの制御パス上のどの位置にある命令であるかを示した制御パス情報が付加される。データパス先行展開の動作の 1 つが、この制御パス情報をもとにしたパス毎の命令に対するデータ依存解析である。さらに、データ依存関係が解消された命令について、ALU-Net の状態を示す情報を参照して、ALU-Net への割り付けを行う。ここで、ALU-Net の状態を示す情報とは、現在命令が割り付けられていない空き ALU についての情報、および、既に ALU に割り付けられている命令の実行結果がどこに書き戻されるかについての情報である。データパス先行展開は、この一連の動作を ALU での命令実行に先駆けて、複数パスを対象に行う。

ALU-Net において、ネットワークによるデータの授受を効率良く行うには、従来のアーキテクチャにおける、フェッチ、デコード、発行、実行、レジスタへの書き戻し、という動作に加えて、次のような動作が必要となる。

(1)　デコードされた命令間でのデータ依存解析
(2)　ALU-Net への命令の割り付け
(3)　ALU-Net での実行パスの作成

データパス先行展開とは、これらの動作の中でデコード、データ依存解析、ALU-Net への割り付け、実行パス作成の動作を制御する機構である。以降では、命令はすでにデコードされ、制御依存情報が付加されたものとして、データ依存解析、命令割り付け、実行パスの作成について述べる。

図 3 は、スーパスカラ・アーキテクチャ[3)]を基に、ALU-Net への割り付けを行うための機構を加えて、データパス先行展開を構成した単純なモデルである。図中、空き ALU テーブルと ALU-Net 接続テーブルが割り付けのために拡張されたテーブルである。この 2 つのテーブルは ALU-Net への命令割り付け時に参照され、命令が発行される際と ALU-Net より演算が終了した ALU についての解放情報を得た際に更新される。フューチャ・ファイルはアーキテクチャ・ステートを保持するためのテーブルで、従来のフューチャ・ファイルの機能に加え、ALU-Net 上で実行を待ち合わせている命令についてのタグも保持することができる。フューチャ・ファイルは常に最新の演算結果を保持しているため、通常、命令バッファへのデータの供給はフューチャ・ファイルより行われる。これに対して、

リオーダバッファ、レジスタファイルは例外回復の際に用いられる。図中で、命令バッファ、フューチャ・ファイル、リオーダバッファが複数パスに対応するために多重化した部分である。点線で囲んだ部分が本稿で取り扱う部分である。3.2で、このモデルの動作と問題点を詳しく述べる。

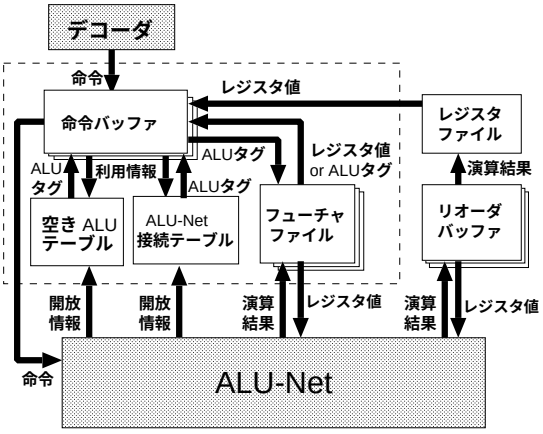

図 3　単純なデータパス先行展開のモデル

3.2　データ依存解析

VLDP アーキテクチャでは、大規模な命令の先行割り付けを行う。現実的な ALU-Net の構成を考えた場合、ALU 間の結合能力には制限が加わるため、全てのデータ依存関係をネットワークで吸収することはできない。オペランドの供給をできる限りネットワーク上で吸収するためには、割り付けを行う前に命令間でのデータ依存解析が重要となる。命令間のデータ依存解析を行わないとすれば、各命令は適当な ALU に割り付けられ、データ供給のほとんどはレジスタを介して行われることになり、ALU-Net を用いる利点を得ることができない。

データ依存解析の定義

本稿においてデータ依存解析とは、(1) 解析の対象となる命令が含まれている制御パス中で、当該命令のソースオペランドに対応する最新のデータか、または (2) そのデータを作成する命令が存在する場所を検索する動作であると定義する。図 3に示しているモデルでは、命令バッファか ALU-Net 中の命令に対応するタグ、またはフューチャ・ファイル中の実際のデータが検索の結果として得られる。同時にフェッチされた命令間でデータ依存関係があれば、最新のデータを作成する命令は命令バッファ中にある。データパス先行展開では、現在実行されている命令に先行して ALU-Net に新たな命令を割り付けるので、ALU-Net 上でソースオペランドのデータが作成されるのを待ち合わせている命令がある。ALU-Net 上の命令に対してデータ依存関係がある場合とは、これらの実行を待ち合わせている命令に対して依存関係がある場合を指す。実行を終えた命令はフューチャ・ファイルに値が書き戻される。命令バッファ、ALU-Net 上の命令に依存関係がなければ、ソースオペランドへのデータはフューチャ・ファイルから読み出される。

データ依存解析の実現

このモデルにおいては、データ依存解析は次のようにして実現している。まず、命令バッファ中の命令間で、ソースオペランドとデスティネーションオペランドの比較を行う。ここでデータ依存関係があれば、解析を行っている命令は依存関係にある命令の命令バッファ中の位置に対応するタグを得る。同時に、フューチャ・ファイルの参照も行い、現在 ALU-Net 上に割り付けられている命令のタグか、既に演算を終了している命令の場合は演算結果のデータを得る。命令バッファ中に依存関係がなければ、フューチャ・ファイルの検索結果を選択する。このデータ依存解析によって得られる値は、命令バッファ中の他の命令もしくは ALU-Net 上の命令に対応するタグ、または実際のデータである。テーブルの更新は、命令バッファ中の命令が ALU-Net に割り付けられた時と、ALU-Net 上の命令の実行が完了した時にそれぞれ行われる。

データ依存解析は複数パスについて行われるので、フューチャ・ファイルはパス毎に多重化する。その場合、新しい制御パスがフェッチされる場合を想定して、分岐命令を跨ぐ毎に新たなフューチャ・ファイルを作成する必要がある。さらに、データ依存解析を高速に行うために、できるだけ多くの命令について、並列に解析が行われることが望まれる。1 つの制御パスに属する多数の命令が同時にフューチャ・ファイルを参照するために、複数パスに対応するため多重化した各フューチャ・ファイルについて、さらにその複製を用意する。

データ依存解析の問題点

図 4 は、データ依存解析を同時に行うことができる命令数に対する、1 サイクル当たりの命令実行数である。これは SPEC92 のベンチマークプログラムである cc1、compress、espresso、sc について、1 千万命令のトレースデータを解析したものである。この解析では、ALU-Net の数とネットワークを理想的なものとしている。命令もデータ依存解析に十分な速度で供給されるとして、純粋なデータ依存解析の能力による影響を調べている。各動作はパイプライン的に処理される。1 つの命令は ALU に発行された後、データ依存関係のある命令が終了した次のサイクルで実行できるとしている。ロード命令については、実行のための遅延が 3 サイクル多くかかるものとする。これを見ると、データ依存解析命令数について、IPC はほぼ比例的に上昇している。取り出すことのできる並列度に限界があるため、ある値で IPC の上昇は止まりグラフは平坦になる。このグラフは、十分な実行能力とフェッチ能力を持つアーキテクチャでの、理想的な実行速度を

表していると見ることもできる。つまり、VLDP アーキテクチャにおいて、ALU-Net 中の ALU の数を増加させて演算能力を向上させた場合、それに十分な命令を供給できるだけのフェッチ能力の向上、および命令バッファの大規模化が必要となる。その場合、命令バッファ中の命令間についてデータ依存解析を行うには、多数の比較器が必要となる。また、既に発行された命令についてのデータ依存解析のため、フューチャ・ファイルへのアクセスも増大する。これらの理由により、データ依存解析のためのハードウェアが大規模化し、ボトルネックになる可能性がある。これを解消するためには、コンパイラによる静的な依存関係の解析といった支援技術や、新しい検索機構の提案が必要となる。

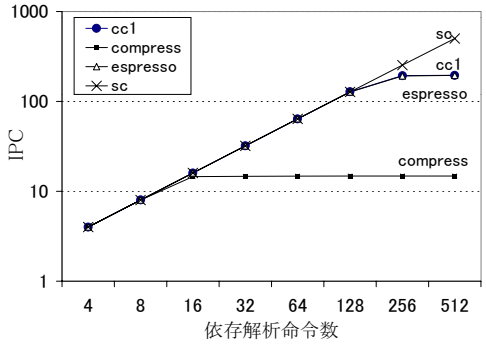

図 4　データ依存解析の性能による IPC の変化

3.3　ALU-Net への割り付け

データ依存解析のところでも述べたように、現実的な ALU-Net のネットワーク接続には制限がある。できるだけ多くのデータの授受をネットワーク上で行うためには、データ依存関係がある命令同士を接続された ALU に割り付ける機構が必要となる。

割り付けの定義

ALU-Net への割り付けとは、データ依存解析によりソースオペランドへのデータ、もしくはデータを作成する ALU-Net 上の命令に対応するタグを得て、全てのデータが供給可能となった命令を ALU-Net に割り付ける動作である。ここで、命令バッファ中の命令にデータ依存関係がある命令は、オペランドとなるデータがどの ALU で作成されるかがこの時点では分からないため、割り付けは次サイクルに持ち越される。

割り付けの実現

図 5に命令が割り付けられる様子を示す。命令を新しく割り付けるには、現在割り付けが行われていない空き ALU を示すテーブルと、ALU 同士の接続を示したテーブルを用意する。この 2 つのテーブルは同時に参照される。ALU の接続を示すテーブルの参照は、データ依存解析の結果で得られた、ALU-Net 上の命令に対応するタグを用いて行われる。新しく割り付ける命令 (当該命令) がソースオペランドとして持つ ALU のタグを入力すると、その ALU に接続され利用されていない ALU の番号を示すタグが返される。うまく空いている ALU があれば、その ALU に対して当該命令は割り付けられる (図 5(1))。この場合、空き ALU テーブルから得られた値は利用しない。命令バッファ中には、当該命令にデータ依存関係を持つ命令が存在するので、発行の際にはそれらの命令に対して、新しく割り付けた ALU に対応するタグを渡す。テーブル中の 1 つの ALU に対して 2 つ以上の問い合わせが重なった場合、命令間に優先順位を設けて、複数の命令が同じ ALU に割り付けられないようにする。

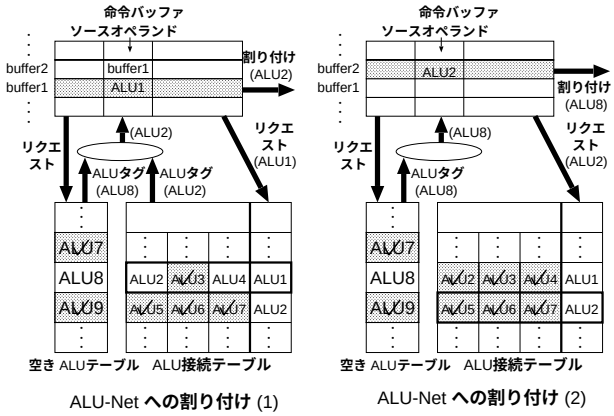

図 5　命令の割り付けプロセス

ALU-Net 上の命令に依存関係を持たない命令、または依存関係のある ALU-Net 上の命令に接続された ALU の空きがなかった命令は、ALU の空きを示すテーブルを参照し、空いている ALU の中で適当なものに割り付けられる (図 5(2))。ソースオペランドへのデータはフューチャ・ファイルより供給される。この場合、依存関係がある命令がインオーダに実行される保証がなくなるため、解消したデータ依存関係が崩れてしまう。図 3のモデルでは、この問題にレジスタ名前替えの機構を用意して対応する。複数実行パスを扱うため、多重化したリオーダ・バッファによるレジスタ名前替え[4]を用いている。空き ALU がなくなった場合は、命令の割り付けを中断する。演算が終了した命令は、演算結果をフューチャ・ファイルとリオーダ・バッファに書き戻す。同時に ALU を解放し、空き ALU テーブルと接続テーブルを更新する。

割り付けの問題点

ALU-Net への割り付けでは、1 つの ALU のファンアウトの制限が問題となる。全ての ALU の接続が可能なモデルでは、データ依存関係のある命令はネットワーク上の全ての ALU で接続することができる。その場合、テーブルを引くといった割り付けのための

機構は必要がなく、データ依存解析をした全ての命令を同時に ALU-Net に割り付けることができる。現実的なモデルではネットワークの制限から、レジスタを介するデータの授受が避けられない。図 3のモデルでは、データ依存関係のある先行命令の割り付けが行われないと、演算結果を利用する側の命令の割り付けが行われず、割り付けが遅れる。1 サイクルで割り付けた命令は、そのほとんどが次の 1 サイクルで演算が完了する。データ依存解析が十分な速度で行われ、かつ ALU-Net の演算能力が十分にあるとすれば、依存関係のある命令を同時に割り付ける機構がないと、割り付け性能が全体のクリティカルパスになると考えられる。

3.4 実行パスの作成

ALU-Net 中の 1 つの ALU の出力は、複数の ALU の入力に接続されている。データパス先行展開では、ある ALU の演算結果を、その値が必要な ALU に供給されるように、ALU 間の接続の制御を行ってデータの流れを管理し、実行パスを作成する。

ソースオペランドにデータが供給されるパスは、ネットワークで接続された他の ALU から直接データが供給されるパスと、フューチャファイルよりデータが供給されるパスがある。命令は割り付け時に ALU 間の接続も考慮して配置されているため、ALU 間でデータの授受を行う命令については、ALU 間の接続の ON/OFF を行って前段の ALU の出力を供給する。

フューチャ・ファイルから値が参照されるものは実行パスの構成が複雑になる。フューチャ・ファイルから ALU-Net へのバスは本数が限られるため、同じバスを使って複数の ALU へデータを転送しなければならない。命令を発行する際に、データを読み出すタイミングについての情報を命令に付加し、その情報に基づいてデータの授受の同期をとる必要がある。

4. データ依存解析に対するソフトウェア支援

前章で、データパス先行展開について単純な 1 モデルを示し、そのモデルを実現した場合の問題点について述べた。ALU-Net への割り付けと実行パスの作成は、ALU-Net 自体の規模や構成と深い関わりを持つため、動作を詳しく解析するためには、ALU-Net のモデルを定義しなければならない。しかし、ALU-Net の規模や構成はデバイス技術の推移に伴い変化するため、最適な ALU-Net を現状で定義するのは困難である。以降では、ALU-Net の構成とは独立して考えることができるデータ依存解析について、ソフトウェアによる支援の可能性について考察を行う。

4.1 従来のモデルによるデータ依存解析

先に述べたモデルにおけるデータ依存解析は、フューチャ・ファイルの参照と、命令バッファ中のオペランドの比較で実現している。フューチャ・ファイルの検索は、発行済みの命令に依存関係のある命令について同時に行う必要があり、命令バッファ中での比較は、バッファ中の全ての命令が比較の対象となる。大規模な投機的フェッチを行う VLDP アーキテクチャでは、その規模や動作速度を考えた場合に、これを実現するハードウェアのコストが問題となる。次では、ハードウェアコストの削減を目指した、ソフトウェアによるデータ依存解析のアプローチを示す。

4.2 ソフトウェア支援に対する考察

新しいアーキテクチャを考える時に、他モデルとのコード互換性を維持するという点から、全ての処理をハードウェアで行うという立場がある。その場合、動的に動作が決まる部分にも処理が行える、という利点もある。対して、静的に処理を行える部分はできるだけソフトウェアで行う、というアプローチがある。ソウトウェアによる処理では、ある程度の処理をソフトウェアで未然に行うため、ハードウェアが簡単化・高速化できる、また、処理の仕様の変更に対して、柔軟に対応できるという利点がある。ソフトウェアで行うことの欠点は、ハードウェア処理の特徴であった、コードの互換性、動的な解析、という利点を得ることが難しくなることである。

先に VLDP において、データ依存解析の重要性と、そのハードウェアにかかるコストがボトルネックとなる可能性を挙げた。そこで、ソフトウェアによってデータ依存解析を行うことで、ハードウェアの負担を減らすアプローチを考える。コード互換性という点でも、従来のコードに依存情報を付加するという手法を取れば、大幅な変更を避けることができる。コンパイラレベルでの解析を行えばより良い最適化も期待できるが、その場合異なったモデル間でのコード互換性が失われてしまう。データパス先行展開を実現する際には、コード互換性を保つという立場と、コードの最適化による高速化という立場の 2 通りのアプローチから考える必要がある。

表 1 は命令間のデータ依存関係が、基本ブロックをいくつまたいでいるかを示した表である。これは SPEC92 のベンチマークプログラムである cc1、compress、espresso、sc について、1 千万命令のトレースデータを解析したものである。例えば距離が 2 とは、その命令が含まれている基本ブロックから見て、2 つ前に実行された基本ブロックに含まれる命令について、データ依存関係があることを示している。距離が 0 とは、同じ基本ブロック中に依存関係があることを示す。間接アドレッシングによる分岐命令をまたぐ依存関係は、この表の中ではカウントされていない。割合の合計が 1.0 になっていないのは、間接アドレッシングをまたいだ依存関係も含めた全依存数に対する割合となっているからである。この表を見ると、データ依存関係は時間的に近く発行された基本ブロック間に多く存在している。VLDP アーキテクチャにおける

表 1　基本ブロック間のデータ依存関係

距離 [BB]	cc1		compress		espresso		sc	
0	5056700	0.4330	7422579	0.6712	6380928	0.5435	4809133	0.3758
1	2358081	0.2019	1839330	0.1663	2463513	0.2098	2498939	0.1953
2	596761	0.0511	435691	0.0394	883374	0.0752	759392	0.0593
3	264904	0.0227	369005	0.0334	233298	0.0199	360111	0.0281
4	153675	0.0132	195234	0.0177	80059	0.0068	185306	0.0145
5	100091	0.0086	82189	0.0074	66013	0.0056	104616	0.0082
6	83309	0.0071	77572	0.0070	84901	0.0072	62197	0.0049
7	59911	0.0051	30359	0.0027	66972	0.0057	46429	0.0036
8	43807	0.0038	2	0.0000	81779	0.0070	42885	0.0034
9	28668	0.0025	12634	0.0011	68385	0.0058	38945	0.0030
10	22571	0.0019	22860	0.0021	15514	0.0013	32719	0.0026
∼20	96101	0.0082	39134	0.0035	186930	0.0159	178682	0.0140
∼30	28770	0.0025	10134	0.0009	111969	0.0095	60724	0.0047
∼40	13498	0.0012	3044	0.0003	16745	0.0014	35661	0.0028
∼50	7312	0.0006	2539	0.0002	5576	0.0005	26330	0.0021
∼60	3632	0.0003	3453	0.0003	4874	0.0004	20269	0.0016
∼70	1877	0.0002	146	0.0000	4349	0.0004	17823	0.0014
∼80	1675	0.0001	16	0.0000	4448	0.0004	15064	0.0012
∼90	1164	0.0001	5	0.0000	4612	0.0004	14220	0.0011
∼100	710	0.0001	4	0.0000	3644	0.0003	12078	0.0009
100∼	150986	0.0130	7	0.0000	349002	0.0297	594307	0.0464
total	9074203	0.7641	105459370	0.9537	111168850	0.9172	99158300	0.7285

フェッチ機構は、投機的に大規模なフェッチを行い制御依存関係に基づいた情報付けを行う。基本ブロックを最小単位とし、連続したいくつかの基本ブロックをまとめてフェッチ、デコードするという仕様が考えられる。そのため、同時にフェッチされた命令、もしくは ALU-Net 上で演算の待ち合わせをしている命令に対して、データ依存関係がある可能性が高い。表 1 によると、連続する 3 つの基本ブロックについて依存解析を行えば、最も多い compress で 88%、最も少ない sc で 63% のデータ依存関係がある。フェッチが大規模になるにつれ、これらの依存関係を持つ命令を同時にフェッチする可能性が大きくなり、フューチャ・ファイルや命令バッファ内の比較機構の負担は大きくなる。この理由により、ソフトウェアによる依存解析が有効であると考える。

以上で述べたようにデータ依存関係については、ソフトウェアでの静的な解析を行うことで、大規模化によるハードウェアコストの増大を抑えることができる。

5. ま と め

VLDP アーキテクチャの中で、データ依存解析と ALU-Net への割り付け、および実行パスを作成する機構であるデータパス先行展開について、従来のスーパスカラ・プロセッサの拡張によって実現したモデルを示し、その動作の説明と実現しようとした場合に生じる問題を明らかにした。本稿で示したモデルでは、データパス先行展開の最低限の動作は保証できるが、ハードウェアのコストが増加し、十分な動作性能を得ることができない可能性がある。データ依存解析部では、効率的な依存解析を行う新しい機構を導入する必要があり、本稿ではソフトウェアによる解析支援の可能性を示した。今後はデータパス先行展開の実装に向けて、クリティカルパスとなる部分を定量的に評価し、ハードウェアとソフトウェアの統合的なアプローチを進めて行く予定である。

参 考 文 献

1) 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, ARC124-3, pp. 13-18, 1997.
2) 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. 情報処理研究会 ARCH 125-21, Vol.1, No.2N-1, Mar. 1998.
3) マイク・ジョンソン. スーパースカラ・プロセッサ. 日経 BP センター.
4) 安島 雄一郎, 中村 友洋, 吉瀬 謙二, 辻 秀典, 田中 英彦. スーパスカラ・アーキテクチャのための複数パス実行機構の提案, 並列分散シンポジウム JSPP'98, Vol.98, No.7, pp.23-30, Jun. 1998.