大規模データパス・アーキテクチャの提案

辻　秀典[†]　安島　雄一郎[†]
坂井　修一[†]　田中　英彦[†]

我々は新しいマイクロプロセッサ・アーキテクチャとして、大規模データパス・アーキテクチャを提案する。これは、将来利用できる大規模なハードウェア資源を有効に活用し、積極的に細粒度並列性を抽出することで、実効 IPC 8 の達成をめざすものである。本アーキテクチャでは、大規模な命令処理と複数パス実行を導入する。本論文では、その大規模な複数パス実行の実現について述べ、性能に関する初期的な検討を行う。

Very Large Data Path Architecture

Hidenori Tsuji,[†] Yuichiro Ajima,[†] Shuichi Sakai[†]
and Hidehiko Tanaka[†]

We propose the Very Large Data Path (VLDP) architecture, a new microprocessor architecture which is expected to effectively utilize the massive hardware resources available in the future. VLDP performs the enormous instruction processing and multiple-path execution to achieve effective IPC of 8 by exploiting fine-grain parallelism aggressively. This paper describes the implementation for large scale multi-path execution mechanism and briefly evaluates its performance.

1. はじめに

マイクロプロセッサの性能向上は留まるところを知らない。その性能向上は、アーキテクチャと半導体プロセス技術に支えられている。常に進歩をとげる半導体プロセス技術によって、より高い集積度が実現され、より多くの利用可能なトランジスタ数が提供されてきた。それが、さまざまな新しい技術の実装を可能とするだけでなく、1GHz を越える高いクロック周波数を実現した。現在主流のスーパースカラで・アーキテクチャでは、さまざまな技術により命令レベル並列性を利用した命令処理が行われている。

しかしながら、スーパースカラをベースとしたアーキテクチャでは、分岐予測性能の限界と分岐予測ミスペナルティの増大、より多くの並列性利用を目的とした命令ウィンドウの拡大の限界など、動的な並列性利用技術による性能向上の限界が指摘されている[6)7)]。そこで、より多くの細粒度並列性を利用するさまざまなアーキテクチャの研究が行われている。その研究の例としては、hydra[3)]、multiscalar[8)]、MUSCAT[11)]、SKY[12)] などの CMP(Chip Multi-Processor) と、simultaneous multithreading (SMT)[9)], M-Machine[2)] などの multithreading がある。スーパースカラが単一スレッドにおける並列性の利用であるのに対し、それらのアーキテクチャは複数のスレッドからより多くの並列性を利用する。

今後も半導体技術の進歩が期待できるならば、ハードウェア資源の投入とともに性能向上が望めるアーキテクチャが必要である。スーパースカラは、より多くのハードウェア資源を投入したとしても、命令ウィンドウの実装の複雑さなどの点で大規模化による性能向上は難しい。multithreading も、構造の複雑さという点では、スーパースカラを改善するものではないため同様である。その観点では、CMP は提供されるハードウェア資源を有効に活用する手段である。しかしながら、さらに多くのハードウェア資源を活用するために、より多くのプロセッサを並列化した場合には、複数のプロセッサ間における制御依存とデータ依存の管理が複雑化し、単純にプロセッサ数を増やすことによる性能向上は難しい。

スーパースカラによる並列性利用が細粒度とすれば、単一スレッドに対する細粒度並列実行の要素プロセッサとそのプロセッサの並列化による中粒度から粗粒度の並列性利用を組合せる技術が CMP の技術である。そのため、CMP は単純にスーパースカラを並列化する技術ではなく、利用可能なハードウェア資源を考慮した、要素プロセッサの規模とそのプロセッサの並列化のバランスが重要である。つまり、細粒度並列利用と CMP による中粒度以上の並列性利用は直交する技

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

術であると考えられ、スーパースカラを越える細粒度並列性を利用するアーキテクチャの研究は必須であるといえる。

そこで我々は、5 年以上先に利用可能なハードウェア資源を背景に、単一スレッドから積極的に細粒度並列性を利用して実効 IPC 8 を達成する、大規模データパス (VLDP: Very Large Data Path)・アーキテクチャを提案する。このアーキテクチャは、スーパースカラや VLIW よりもはるかに大きな幅で命令を並列処理するとともに、並列性抽出の鍵となる大幅な命令ウィンドウの拡大を実現する。また、分岐予測ミスペナルティの増大を避けるため、複数パス実行を導入する。そして、VLDP アーキテクチャを実行機構を含む複数パス実行を実現するアーキテクチャとして提案する。本論文では、VLDP アーキテクチャにおける複数パス実行の実現と大きな幅の命令発行の実現を中心に、アーキテクチャの提案と初期評価を行う。

2. 命令ブロックの導入

VLDP が目標とする実効 ILP 8 を達成するためには、毎サイクルに 2 桁命令のフェッチスループットが必要となる。そこで、複数の命令を同時に処理するために、命令ブロック (IB: Instruction Block) を導入する。IB によって処理単位を大幅に拡張することで、高いスループットを確保するとともに、命令管理の単位が大きくなることで処理の複雑化も避けられる。さらに、整数演算系の命令列には分岐命令が 2 割以上存在することから、IB は複数の分岐命令を含む必要があり、複数パス実行における分岐命令の扱いも考慮する。本節ではそのような IB の構成について述べる。

2.1 IB の構成

IB は複数の命令によって構成され、フェッチポイントとなるひとつの PC を与えられる。命令幅は 32 命令の固定長として、その中に存在できる分岐命令数は 4 つとする。図 1に示すように 32 命令のスロットを持ち、これが 8 命令単位の 4 つの field に区切られ、制御フロー順に命令が配置される。ただし、それぞれの field の最後のスロットにだけ分岐命令が配置できるものとする。分岐の区切りにより命令が埋められないスロットは、空きスロットとして NOP 命令を挿入する。なお、分岐命令が 8 命令以上の間隔で出現した場合には、その基本ブロックを複数の field に分割して配置する。

IB は先頭の命令から必ず処理されるが、分岐の結果によって、実際に実行される制御フローは異なるため、IB 内の命令がすべて実行されるとは限らない。そのため、IB は field 単位に実行を区切ることができる。

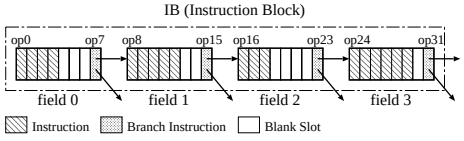

図 1 命令ブロックの構成
Fig. 1 Instruction Block Structure

2.2 IB の構成情報

IB は命令情報とデータ同期情報を持つ。命令情報は 32 命令それぞれの命令コードと入力オペランドにより構成され、ひとつのスロットに相当する情報を図 2に示す。なお、出力オペランドは用意されず後述の Output Register Map を用いる。

ここで注目すべきは、入力に関する情報である。従来の命令の入力は論理レジスタ番号もしくは即値であったが、IB ではこれに加え出力番号が追加される。これは IB 内の 32 の命令に順に与えられた番号であり、IB 内の n 番目の命令の結果を意味する。これを利用すれば、IB ローカルな命令間でのデータの受け渡しには、論理レジスタを介する必要がなくなる。

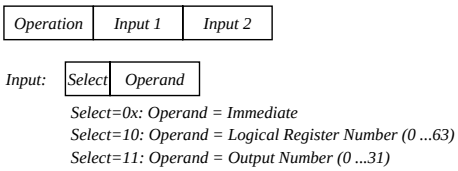

図 2 命令情報
Fig. 2 Instruction Information

VLDP では、IB 内で参照するすべての論理レジスタ、IB の実行の結果として更新するすべての論理レジスタの情報をコードに付加する。この情報をデータ同期情報と呼び、これによってデコード時の論理レジスタと物理レジスタの対応づけの処理を軽減する。これらは、Input Register Mask、Output Register Mask、Output Resgiter Map として表現され、次のような意味を持つ。

Input Register Mask (IRMask): IB 内の全命令が参照する論理レジスタの情報をあらわす。64 ビットで構成され、それぞれの bit が 64 個の論理レジスタに対応し、参照される論理レジスタに対応するビットが 1 となる。

Output Register Mask (ORMask): IB の実行の結果、更新する論理レジスタの情報をあらわす。構成は IRMask と同様であり、更新する論理レジスタに対応するビットが 1 となる。

Output Register Map (IRMask): IB 内の各命令の演算結果に対応する論理レジスタをあらわす。命令番号順に更新する論理レジスタの番号を記述する。これが各命令の出力オペランドに相当する。

IB 内には最大 4 つの分岐命令が存在するため、それぞれの分岐命令の確定によって、更新するレジスタの情報は異なる。そのため、ORMask と ORMap については、それぞれの分岐命令をチェックポイントとして図 3に示すように 4 つ用意する。

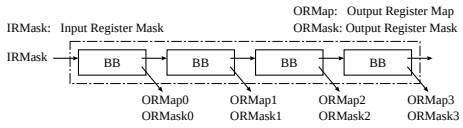

図 3 データ同期情報
Fig. 3 Data Synchronus Information

2.3 IB の生成

IB 内には 3 つの分岐ポイントが存在するため、その組合せは最大 8 とおりとなる。しかし、同じ基本ブロックから始まる IB を 8 とおり用意すると、1. 命令列が冗長となる 2. 同じ PC からスタートする複数の IB を区別する機構が必要になるという問題が生じるため、ひとつの PC からスタートする IB はひとつに限定する。実行される命令列の制御フローには局所性があるため、実際の IB の構成では、なるべく IB 内の命令が多く実行されるように、実行される確率が高い組み合せで命令列を生成する。これによって、コード量の増大とフェッチ機構の複雑化を避ける。

```
int loop, n;
void livermore05(long *x, long *y, long *z){
  int l, i;
  for (l=1; l<=loop; l++) {
    for (i=1; i<n; i++) {
      x[i] = z[i] * (y[i] - x[i-1]);
}}}
```

図 4 サンプルプログラムの C ソースコード
Fig. 4 C Source Code of Sample Program

次に、実際に IB の生成例を示す。サンプルプログラムとして、簡単なループ演算の livermore loop 5 番を取り上げた。その C のソースプログラムを、図 4に示す。これを Alpha AXP アーキテクチャのコードに gcc の -O2 オプションでコンパイルしたコードを基本ブロックに分割して、その制御フローの関係を示したものが図 5である。図中の BBxx は基本ブロックの番号を示す。また、矢印の太さは分岐先の実行確率を示しており、ループする方向に確率が高いと仮定した。

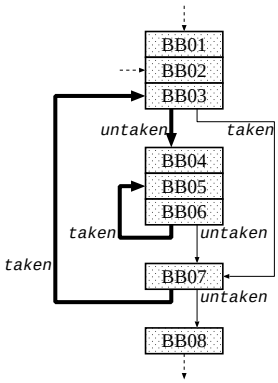

図 5 サンプルプログラムのコントロールフロー
Fig. 5 Contorl Flow of Sample Program

これに基づき、より実行される確率の高い命令の組合せで、各基本ブロックから始まる IB を生成する。図 6 における括弧内は、IB に含まれる命令の数を示している。

2.4 IB のストリーミング

容量の大きなメモリはレイテンシが大きいために、ランダムアクセスの高速化によりスループットを稼ぐ

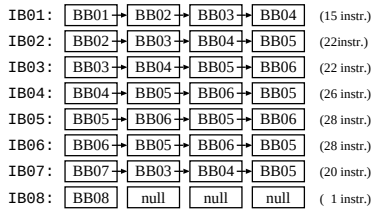

図 6 IB の生成例
Fig. 6 Example of IB Creation

ことは難しい。そこで、メモリデバイスのバースト転送能力に注目し、連続化した IB 列を転送することで要求されるフェッチ能力を達成する。VLDP ではコードの連続化をストリーミングと呼び、コントロールフローが連続する複数の IB をまとめ、IB よりもより大きな単位で命令列を転送する。さらに、IB 内の NOP を圧縮しメモリの利用効率を上げる。ストリームコードは、コンパイラによって生成され、メモリ上にその形で格納される。メモリ上における命令転送のスループットを確保するために、オフチップのメモリ (メインメモリや外部キャッシュ)、オンチップのキャッシュ上は、すべてストリームコードが転送される。

3. 複数パス実行

VLDP では毎サイクルに最大 4 つの分岐命令を処理するため、分岐予測ミスの影響は従来よりもはるかに大きい。そこで、分岐ペナルティを削減するアプローチとして複数パス実行を採用し、分岐先が確定していない分岐命令の複数の分岐候補を投機的に処理する。従来より、複数パス実行に関する研究は多く行われている[10)4)5)]。しかしながらこれらの研究では、複数パス実行における命令フェッチの戦略について主に議論されているにとどまっている。複数パス実行を実現する場合には、パスのフェッチの戦略にとどまらず、制御依存とデータ依存の管理が大きな課題となる。この節では、複数パス実行の実現のために解消しなければならない課題を列挙し、VLDP がこれをどのように解決しているかについて述べる。

3.1 複数パス実行の課題

複数パス実行では、これによって生み出される複数の制御流に対する制御依存とデータ依存を管理しなければならない。具体的には次にあげる処理である。

(1) 命令間の順序関係の管理
(2) 分岐の確定による不用な命令の削除
(3) 異なる制御流におけるデータ依存性の保証

複数パス実行では、すべての命令の親子関係を管理するとともに、複数の制御流間での依存関係を管理する必要がある。これが、(1) の命令の順序関係の管理である。また、パスが投機的に処理されているので、分岐の確定により実際には必要のないパスを削除する必要がある。これが、(2) の分岐の確定による不用な命令の削除である。(1) の情報と (2) の操作は、プロセッサ内部の全ての処理に必要とされるため、これが

処理のクリティカルパスとならない実装を提案する必要がある。

そして、(3) は特に大きな課題である。制御流の分岐よってデータ流も分岐するため、複数の制御流間で独立したデータ依存性を保証しなければならない。単純にデータ依存性を保証するための手法として、制御流の分岐ポイントにおけるデータの複製があげられる。しかしながら、プロセッサにおけるデータはレジスタとメモリ上に存在し、それを分岐のたびに複製することは実質的に不可能である。そのために、これを仮想的に実現する、レジスタアクセス機構とメモリアクセス機構が必要である。また、これらについても処理のクリティカルパスとならないために、パス管理機構と新和性の高い手法をとる必要がある。

3.2 複数パス実行の実現

VLDP では大規模に複数パス実行を行う現実的な手法を提案する。それらは大きくパス管理、レジスタアクセス管理、メモリアクセス管理に分けられる。

3.2.1 パス管理

複数パス実行におけるパス管理を実現する場合、フェッチしたパスに対してタグを与え、そのタグを表で管理することで命令の順序関係を管理する。VLDP ではタグの与え方を工夫し、タグ同士の比較により順序関係の判定が行えるようにする。このタグを BHTag と呼び、フェッチ時に IB 内の各 field に与える。パス管理はすべて BHTag を用いて行い、BHTag の比較だけでパスが親子関係にあたるのか、異なる制御流のものであるかを比較できるようにする。これによって、パス管理の表へのアクセスは、フェッチ時と完了時、それに伴うパス無効化時だけとなる。

3.2.2 レジスタアクセス管理

VLDP では、物理レジスタと論理レジスタの対応を、Register Map Set (RMS) という形で保存する。フェッチ時にフェッチポイントにおける RMS が与えられ、デコード時に実行に物理レジスタへのアクセス情報を生成する。同時に、その IB を実行した後の状態の RMS を生成する。IB 内には最大 4 つの分岐命令が存在し、4 つの新たなフェッチポイントを持つため、4 つの RMS が生成される。RMS は分岐ポイントにおけるデータ流のチェックポイントであり、これによって複数パス実行におけるレジスタのデータ依存を保証する。VLDP では、IB 内のレジスタ同期情報を用いることにより、物理レジスタへのアクセス情報の生成と新たな RMS の生成の処理を簡単化している。

3.2.3 メモリアクセス管理

メモリアクセスにおける制御依存性とデータ依存性は、ロードストアユニットによって保証される。ロードストアユニットは、実行ユニットからのメモリアクセスのリクエストを保持し、ストア命令に関しては、そのストアがリタイアするまで保持して依存性を解消する。ロードに関しては、依存性をロードストアユニットで解析し、保持されているストア命令からフォワーディングできるものはフォワーディングする。複数パス実行により、リタイアしない命令からのメモリアクセスも処理されるが、これはすべてロードストアユニットにおいて吸収する。VLDP では大規模なロードストアユニットを構成することで、投機的メモリアクセス、依存性の解消、ロードストア間のデータフォワーディングを実現する。

4. 基本構成

VLDP の基本構成を図 7 に示す。その構成は大きく Contorl Section と、Exectuion Section、Memory Access Section に別れ、Control Section ではフェッチとパス管理、Execution Section ではデコードと実行、Memory Access Section では Load/Store 命令の処理を行う。

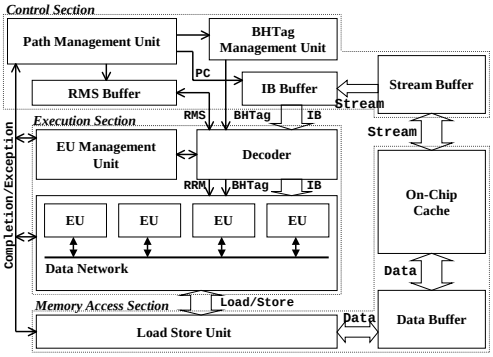

図 7 VLDP の基本構成

Fig. 7 Fundamental Structure Block Diagram

4.1 命令処理

VLDP における命令の処理は IB の単位で行われ、フェッチとデコードは直列、実行が並列に処理される。ひとつの IB はひとつの EU(Eexecutin Unit) に割り当てられて実行され、EU が複数存在することで IB を並列に実行する。EU 間でのレジスタアクセスのために EU 間を接続する Data Network が存在する。Path Management Unit (PMU) は IB のフェッチと完了を管理する機構であり、RMS Buffer と BHTag Management Unit はフェッチした IB に RMS と BHTag を与える。Execution Section において、命令のデコードと物理レジスタへのアクセス情報が生成され、EU Management Unit(EUMU) によって指示された EU に IB を割り当てて実行する。また、EUMU は EU における IB の実行完了と EU の解放の管理も行う。分岐の確定により不用となったパスの削除の管理は PMU で行われ、その指令を全機構に送ることで各機構が命令の削除を行う。EU 内にはメモリアクセス機構は持たず、ロード・ストア命令は Load Store Unit に直接発行される。

4.2 レジスタアクセスの効率化

VLDP では、処理命令数の大幅な増大とともにレジスタアクセス数も多くなるため、集中化したレジスタファイルでは大規模かつ複雑化する。そこで、レジスタファイルを分散させ各 EU に配置する。

「短い距離で命令間の一時的なデータ転送に使われることが多い」[1)]というレジスタアクセスの性質に注目すると、IB 内で生成されたデータを IB 内で消費する IB 内レジスタアクセスを、IB 間レジスタアクセスと分離できる。特に IB 内レジスタアクセスのうち特に IB 内で生成され、IB 内で消費されてしまうレジスタを”Ephemeral Value” と定義し、論理レジスタを消費しないデータ転送を実現する。これは、IB に情報を付加することで行い、データの消費者が生成者の IB 内命令番号を指定することで実現する。(図 2 おける input field の select = 11 がこれに相当) このように、局所的なレジスタアクセスを最適化して高速化するとともに、大域的なレジスタアクセス数を減らすことで、平均的なレジスタアクセス時間をを低下させることなく、分散レジスタ構成により仮想的に大規模なレジスタファイルを実現する。

さらに、IB 間のレジスタアクセス性能を低下させないために、他の IB に対するレジスタアクセス要求はデコード時に生成される。図 8に示すように、IB の割り当てと同時に他の EU に対して Register Request Map (RRM) が発行される。RRM を受け取った EU では、指定されたレジスタ値が準備でき次第値を転送する。

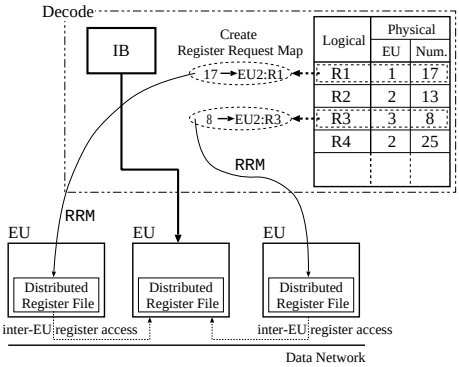

図 8 EU 間レジスタアクセス
Fig. 8 Inter-EU Register Access

5. 命令実行の流れ

この節では、VLDP の命令実行の流れについて、サンプルプログラムのパイプラインフローの例をあげて説明する。

5.1 パイプライン構成

フェッチとデコードのパイプラインステージ構成を図 9に示す。

フェッチ処理には 2 ステージを要し、IB のフェッチと RMS、BHTag の取得を行う。PMU は、分岐命令の履歴とすでにフェッチしたパスの情報を管理し、その情報に基づいて次にフェッチする IB を予測する。予測の結果フェッチ候補となる PC は優先順位を付けてバッファリングされており、このバッファから次にフェッチする IB の PC を取得する。IB が展開されている IB Buffer に対して、取得した指定することで、新たな IB をフェッチする。このとき、フェッチする IB の親にあたる IB の BHTag と予測された IB のフェッチポイントを BHTag Management Unit と RMS Buffer に送り、フェッチポイントにおける RMS と新たな BHTag を取得する。

デコード処理には 3 ステージを要し、IB を割り当てる EUID の指定、IB のデコード、RRM の生成が行われる。EUMU は EU の実行状況を把握しており、新たな IB が割り当て可能な EU の EUID を指定する。また、指定された EU に従い RMS と IRMask より RRM を生成し、ORMask と ORMap を参照することで RMS の更新を行う。

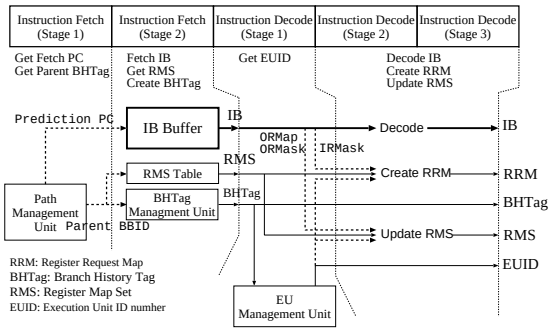

図 9 フェッチとデコードの処理
Fig. 9 Fetch and Decode Process

EUMU で指定された EU に対して、IB が割り当てられることで IB は実行される。IB の割り当てとともに、他の EU に対しては RRM が発行される。それぞれの EU は RRM に従って、値が準備できたものからレジスタ値を返す。IB は EU 内の 32 命令幅の命令ウィンドウに格納され、実行可能な命令が out-of-order に発火され、命令レベル並列処理される。そのため、実行ステージのサイクル数は IB により異なる。

5.2 パイプラインイメージ

次に、2.3 で用いたサンプルプログラムを実行したときの、パイプラインフローを図 10に示した。EU における実行サイクルとは、データ依存グラフの段数をもとに設定し、メモリアクセスについては理想化した。また、分岐命令が確定するサイクルも同様に設定している。

図 10 中の矢印は、分岐命令の確定とパスの削除の関係を示している。この例では、サンプルプログラムの内側のループを 4 まわす例にすぎないが、途中までの実行を見ると、外側のループ 2 回に相当する 20 サイクル目までに実行した、有効な総命令数は 147 命令に相当し、単純計算で $147/20 = 7.35$ という実行 IPC になる。

6. 性能に関する考察

VLDP アーキテクチャは実効 ILP にして 8 という値を達成する。これについて、図 11 にスループットベースの性能について示した。VLDP は、フェッチ、デコード、EU に対する IB の割り当てのスループッ

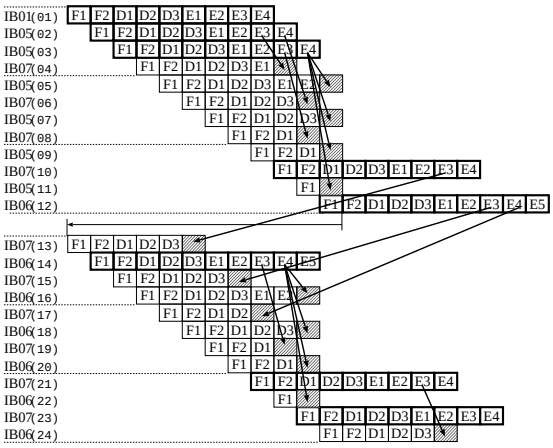

図 10　パイプラインフロー
Fig. 10　Pipeline Flow Image

トは毎サイクル 1IB となる。IB の実行には複数サイクル要し、複数の IB が並列に処理される。IB は実行の結果、リタイアするものと破棄されるものが存在する。VLDP における複数パス実行では、リタイアするパスと投機的処理の結果不用となるパスの割合を 1:1 としており、フェッチスループットの 50%をリタイアスループットとする。IB の平均命令長さは 16 命令以上であるため、リタイアスループットとして 8 命令以上を達成する。

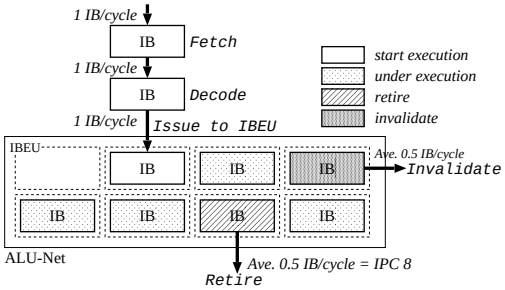

図 11　命令処理のスループット
Fig. 11　Instruction Process Throughput

細粒度並列性を利用するマイクロプロセッサの ILP は、基本的には依存性解析を行う命令ウィンドウの大きさにより決定される。VLDP においては、IBEU における命令レベル並列処理は 32 命令のウィンドウにより実現され、この EU の並列処理によりより大きな並列度を利用可能とする。仮想的には、32 × IBEU の数だけの命令ウィンドウの拡大を行うことに相当し、EU の数を 16 としたとき命令ウィンドウの数は 512 命令に相当する。

7. 結　　論

本論文では、細粒度並列性利用の必要性を述べた上で、積極的に細粒度並列性を利用する VLDP アーキテクチャの提案を行った。そして、VLDP における大規模な複数パス実行の実現について説明した上で、ターゲットとしている ILP 8 の実現について議論した。今後は、アーキテクチャの実装と、シミュレーションによる性能の裏付けを行っていく。また、専用コンパイラの研究も行い一層の性能向上を目指す。

謝辞　本研究の一部は、文部省科学研究費補助金 (基盤研究 (B) 課題番号 11480066) および、(株) 半導体理工学研究センターとの共同研究によるものである。

参 考 文 献

1) C., L. A. L. and Gao, G. R.: Exploiting Short-Lived Variables in Superscalar Processors, *Proc. of the 28th MICRO*, pp. 292–302 (1995).
2) Fillo, M. and Keckler, S. W.: The M-Machine multicomputer, *Proc. of the 28th MICRO*, pp. 146–156 (1995).
3) Hammond, L., Hubbert, B., Siu, M., Prabhu, M., Chen, M. and Olukotun, K.: The Stanford Hydra CMP, *IEEE MICRO Magazine March-April*, pp. 250–259 (2000).
4) Heil, T. H. and Smith, J. E.: Selective Dual Path Execution, *Technical Report, University of Wisconsin-Madison* (1996).
5) Klauser, A., Paithankar, A. and Grunwald, D.: Selective Eager Execution on the PolyPath Architecture, *Proc. of the 25th ISCA*, pp. 250–259 (1998).
6) Lam, M. S. and robert P. Wilson: Limits of Control Flow on Parallelism, *Proc. of the 19th ISCA*, pp. 46–57 (1992).
7) Palacharla, S., Jouppi, N. P. and Smith, J. E.: Complexity-Effective Superscalar Processors, *Proc. of the 24th ISCA*, pp. 206–218 (1997).
8) Sohi, G. S., Breach, S. E. and Vijaykumar, T. N.: Multiscalar Processor, *Proc. of the 22th ISCA*, pp. 414–425 (1995).
9) Tullesen, D. M., Eggers, S. J. and Levy, H. M.: Simultaneous Multithreading: Maximizing On-Chip Parallelism, *Proc. of the 22th ISCA*, pp. 392–403 (1995).
10) Uht, A. K. and Sindagi, V.: Disjoint Eager Execution: An Optimal Form of Speculative Execution., *Proc. of the 28th MICRO*, pp. 313–325 (1995).
11) 鳥居淳, 近藤真己, 木村真人, 西直樹, 小長谷明彦: On Chip Multiprocessor 指向制御並列アーキテクチャ MUSCAT の提案, 並列処理シンポジウム JSPP'97, pp. 229–236 (1997).
12) 小林良太郎, 岩田充晃, 安藤秀樹, 島田俊夫: 非数値計算プログラムのスレッド間命令レベル並列を利用するプロセッサ・アーキテクチャSKY, 並列処理シンポジウム JSPP'98, pp. 87–94 (1998).