計算機アーキテクチャ 139－11
（2000．8．4）

大規模データパス・アーキテクチャにおける命令ブロック構成の検討

塚本泰通† 安島雄一郎† 辻 秀典†
坂井修一† 田中英彦†

大規模データパス・アーキテクチャの処理単位は4つの基本ブロックを合わせた命令ブロックと呼ばれるものである。本論文では、命令ブロックの基本性質を示すとともに、命令ブロックを処理単位とするシミュレータ上で、生成方式の異なる命令ブロックを用い各々の性能評価を行った。その結果、静的分岐予測を用いた命令ブロックの生成方式が効率的であることを示した。

Study of Instruction Block on Very Large Data Path Architecture

YASUMICHI TSUKAMOTO,† YUICHIRO AJIMA,† HIDENORI TSUJI,†
SHUICHI SAKAI† and HIDEHIKO TANAKA†

Very Large Data Path Architecture fetches and executes instructions in a granularity called Instruction Block. An Instruction Block consists of four contiguous basic blocks, each of which can have up to 8 instructions. In this paper, we describe key properties of Instruction Block and evaluated some possible generation techniques. Evaluation showed that we can generate Instruction Blocks in an effective way using static branch prediction.

1. はじめに

現在の汎用プロセッサの主流であるスーパースカラ・プロセッサでは、2～3の命令レベル並列性が抽出可能である。そして、1サイクルに1つの分岐予測を行うことにより毎サイクル1基本ブロックをフェッチできる。基本ブロックは平均5命令から構成されているため、毎サイクル最大で2～3命令を実行可能な現在のプロセッサではフェッチ機構はボトルネックとならない。

しかし、命令スループットが5命令を越すプロセッサにおいては、現在の命令フェッチ機構を用いた場合実行機構に十分な命令を供給できずボトルネックとなる。この問題を解決するためには、毎サイクルに複数の分岐予測を行うことで従来よりも多くの制御依存関係を解消し、複数の連続する基本ブロックを毎サイクルにフェッチすればよい。これを行う命令フェッチ機構にトレースキャッシュ[1)2)]を用いるものがある。

トレースキャッシュは命令フェッチ性能を大きく向上させる。しかし、分岐予測がはずれた場合は分岐命令以降にフェッチした命令がすべて無駄になり、フェッチ能力が向上した分回復ペナルティは大きなものとなる。そこで、大規模データパス (VLDP:Very Large Data Path)・アーキテクチャ[3)4)]では、複数パス実行を行う。これにより分岐予測がはずれた場合でも正しいパスが投機的に実行されている可能性が高くなり、その場合には正しいパスに制御を移すことで回復ペナルティを隠蔽できる。

VLDPアーキテクチャでは複数パス実行を行うためすべてのパスがリタイアされるわけではなく、リタイアする命令数以上の命令をフェッチする必要がある。1サイクルに命令単位でフェッチする場合はフェッチ要求数が増えメモリアクセス数が増加する。また基本ブロック単位でフェッチする場合はIPC8を達成できない。そこで、複数の基本ブロックからなる命令群を単位としてフェッチ操作を行う。この命令群を命令ブロック (IB:Instruction Block) と呼ぶ。このIB単位でフェッチ、実行を行いIPC8を目指す。

本論文では、VLDPアーキテクチャにおけるIBの予備評価としてその構成を考察し、またシングルパス実行のシミュレータ上でいくつかの生成、フェッチ方式の評価を行う。

以下、まず2章ではVLDPアーキテクチャの概要を述べる。3章では命令ブロックの構成を示し、4章で命令ブロックの生成について述べる。5章で評価モデルを示し、6章で評価を行う。最後に7章でまとめる。

2. 大規模データパス・アーキテクチャ

VLDPアーキテクチャは、従来にない全く新しい

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

アーキテクチャである。大規模にハードウェア資源を利用することで命令実行スループットの向上を目指し、その性能目標は実効 IPC8 の実現である。

現在主流のマイクロプロセッサ・アーキテクチャでは、動的分岐予測を用いて制御依存関係を解決し分岐命令を越えて処理を行うことで性能向上を目指している。現在、動的分岐予測成功率は 95%に達しており、多くの制御依存関係を解決可能である。しかし、分岐予測がはずれた場合の回復ペナルティが大きいため、分岐予測の失敗が少ない場合でもプロセッサ性能に与える影響は大きいという問題点がある。

そこで、VLDP アーキテクチャでは分岐予測を行うだけではなく、大規模な投機的処理を行うことで分岐予測ミスによるペナルティを削減することを目指す。具体的には以下の特徴を持つ。

- 複数パス実行による大規模な投機的実行
- サイクル当たり最大 32 命令の処理スループット
- レジスタを介さないデータアクセスの明示的処理
- 高いデータアクセス処理能力

VLDP アーキテクチャでは、高い処理スループットを確保するため従来の RISC 命令よりも大きい命令ブロックと呼ばれる処理単位を導入し、命令処理スループットの向上をはかる。そして、IB を処理単位とする機能ユニットを複数用意し同時に複数の IB を並列に実行する。

また、VLDP アーキテクチャでは従来の命令キャッシュに当たる命令ブロックバッファ(以下、IB バッファ)を持ち、IB のフェッチ要求はまず IB バッファに出される。IB バッファにない場合はより下位のメモリ階層にアクセスする。

3. 命令ブロックの構成

3.1 VLDP アーキテクチャにおける命令ブロック

VLDP アーキテクチャでは、分岐命令が投機処理の区切りであるため基本ブロック (BB:Basic Block) を最小単位としパスの管理を行う。一方、全命令中約 2 割が分岐命令であるため、BB 内の命令数は平均約 5 命令になる。よって、毎サイクルに最大 32 命令を処理する VLDP アーキテクチャにおいて、BB を処理単位に設定すると IPC8 の目標を達成できない。そこで BB を 4 つ合わせたものを IB とし、これを処理単位とする。また、IB の処理を単純にするため BB 内の最大命令数は 8 とし、IB 内の最大命令数は 32 とする。8 命令以上分岐命令が出現しない場合は 9 命令目から次の BB とする。IB の命令の並びは図 1 のようになる。分岐命令は 8 番目、16 番目、24 番目、32 番目に配置し飛先を計算しやすくする。BB が 8 命令未満の場合、足りない部分には NOP が入る。実際には命令以外に入出力の命令情報およびレジスタ同期情報

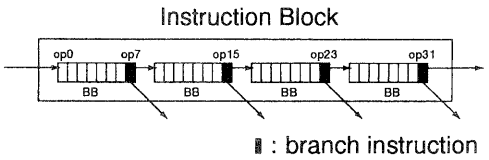

図 1 IB 内の命令の並び

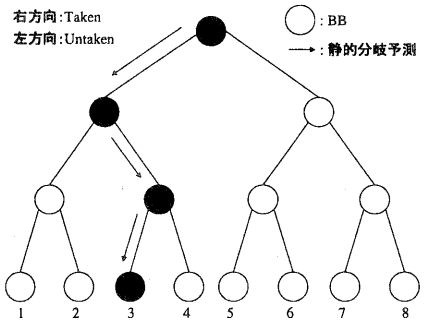

図 2 IB の生成方式

を含んでいる。

3.2 命令ブロックの定義

IB の予備評価を行うにあたり、SPECint95 のベンチマークプログラム go,m88ksim,compress,li,ijpeg,perl,vortex の Alpha AXP[5] バイナリから IB を作成した。以下に、IB 作成時に用いた IB の定義を述べる。

- 「ある PC から始まる IB はただ一つである」
  IB は基本ブロックを最大 4 つ含むため、ある PC から始まる IB は図 2 に示すように最大で 8 つ作成できる。その 8 つの中から静的分岐予測を用いて最も実行確率の高い IB を選び、その一つだけを作成する。図 2 において矢印のような流れで静的分岐予測される場合、黒く塗り潰された 2 番の IB のみを作成する。ここで用いる静的分岐予測は、実行プロファイルからそれぞれの分岐命令の Taken と Untaken の数を集計し多い方を分岐予測結果とするものである。
- 「レジスタ間接分岐命令、Alpha AXP の PAL コード☆を含む場合、その命令までを IB とする」
  レジスタ間接命令の飛先は静的には分からない。また、PAL コードを跨いだ投機実行は行わないものとする。このため、IB を作成する際にこれらの命令が現れた場合、そこまでを IB とする。
- 「分岐命令の飛先命令から始まる IB のみを作成する」

☆ 特権アーキテクチャ・ライブラリ (PAL コード) は、各 Alpha AXP オペレーティングシステムを実現するために使われるサブルーチンの集合である。

分岐命令の飛先となる命令から始まるIBに対してのみフェッチが要求される。そのため、それ以外のIBはコードサイズを減らすため作成しない。しかし、レジスタ間接分岐命令やPALコードの飛先は静的には分からないため、実行プロファイルを用いてその飛先のIBを作成する。ただし、実際にはレジスタ間接分岐命令の飛先判定などを静的に行うことは困難なため、IBを作成する際に飛先の候補をヒントとして与える必要がある。

また本論文で評価に用いたIBには以下の情報を付加する。

- 先頭命令のPC(IBを識別するためのもの)
- IB内の4つの分岐命令の静的分岐予測結果(分岐命令がない場合はNot Takenとする)
- IB内最後の命令の次のPC(分岐命令なら静的分岐予測先、それ以外なら次のPC)

3.3 コードサイズ

本節ではオリジナルの命令列とIBのコードサイズの比較を行う。但し、オリジナルの命令とはIBに含まれる命令列を指し、IB作成時に使われない命令は勘定しない。それぞれのコードサイズを表1に示す。IBのコードサイズは平均で約5.6倍増加している。しかし、1つのIBに最大32個の命令が含まれていることを考えれば、それほど増加していない。

3.4 命令ブロックの充填率

IBは最大32命令からなる命令列であるが、実際には8命令より少ない間隔で分岐命令が出現することから全部埋まるとは限らない。基本ブロックは約5命令から構成され、IBは4つの基本ブロックから構成されることから平均約20命令を含むと予想される。実際に32命令中いくつの命令で埋まっているのかを図3に示す。図3を見ると、どのベンチマークプログラムのIBも20命令(充填率60%)前後から構成されており、予想通りの結果となっている。

また、1~32命令それぞれから構成されるIBの分布を図4に示す。一桁の命令しか埋まっていないIBは全体の中では少ない。19~28命令を含むIBが多いが、特に注目すべき点は32命令全部埋まっているIBが一番多い点である。32命令すべてが埋まっているIBは分岐命令を含まない連続した命令列であることが多く、最も効率的な命令実行が可能となる。このようなIBが全体の1割を占めていることを考えると、IBを処理単位にすることは効率的であると期待できる。

4. 命令ブロックの生成

IBを生成する際にいくつかの手法が考えられる。本章では、本論文の評価に用いた生成方式について述べ、定性的に評価する。

あるPCから始まるIBは3章で述べたように複数

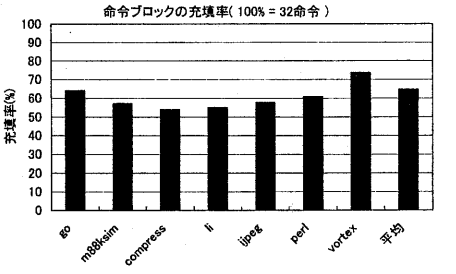

図3 IBの充填率

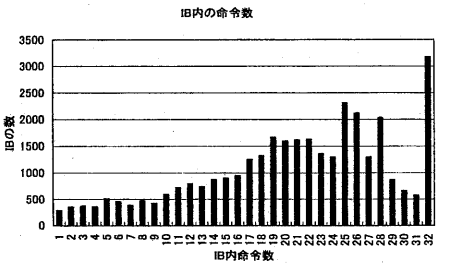

図4 IBに含まれる命令数別分布

(最大8つ)存在する。その点に注目し、あるPCから始まるIBを作成する際、以下の2つを考える。1つは作成可能なIBすべてを作成する方式である(以下、Eager方式)。つまり、図2において1~8番のIBすべてを作成する。もう1つは作成可能な複数のIBの中から静的分岐予測により1つだけ選択し作成する方式であり、VLDPアーキテクチャで用いる手法である(以下、VLDP方式)。図2においては3番のIBのみを作成する。

VLDP方式とEager方式それぞれの長所と短所を以下に述べる。

- VLDP方式
  - 長所
    (1) コードサイズがEager方式より小さい
    (2) (1)から、あるPCから始まるIBのIBバッファ占有率がEager方式に比べて低いため、IBバッファのヒット率が高くメモリアクセス回数が少ないと期待できる
  - 短所
    (1) 静的分岐予測と動的分岐予測が異なる場合、その分岐命令以降にはフェッチ要求を出した命令とは異なるものが入っており、IB内の命令利用率が低くなる(但し、動的分岐予

表 1　コードサイズの比較 (単位：Byte)

	go	m88ksim	compress	li	ijpeg	perl	vortex	TOTAL
Original	208624	44396	16860	34084	70968	87368	317376	778676
IB	1056640	346624	178048	249984	432768	658304	1433088	4355456
増加率 (倍)	5.09	7.81	10.6	7.33	6.10	7.53	4.52	5.59

測が誤りで静的分予測が正しい場合、フェッチした IB 内には正しい命令列が入っており実行を継続できる)

(2) (1) のため新たな IB をフェッチする必要が多くなり、フェッチ要求数が多くなる

- Eager 方式
  - 長所

    動的分岐予測通りの IB が存在するため、フェッチ要求通りの命令列をフェッチ可能となり、IB 内の命令利用率が高い
  - 短所

    (1) VLDP 方式に比べコードサイズが最大 8 倍になる

    (2) (1) のため、ある PC から始まる IB の IB バッファの占有率が高いため、IB バッファのヒット率が下がりメモリアクセス回数が多いと予想される

VLDP 方式と Eager 方式の性能は、コードサイズの大/小とフェッチ要求通りの命令列の取り出し可能/不可能と IB バッファのコンフリクションによるメモリアクセス数の大/小のトレードオフにより決まると予測される。

5. 評価モデル

本章では、評価に用いる 3 つのモデルについて述べる。これらのモデルは IB の生成方式とフェッチ方式がそれぞれ異なる。

5.1 VLDP モデル

IB は VLDP 方式により生成し、フェッチ要求はフェッチする IB の開始 PC とその先 4 つの分岐命令に対する動的分岐予測結果の 2 つから行う。同じ開始 PC を持つ IB が IB バッファにない場合メインメモリにアクセスする。その次に投機的にフェッチする IB は、3.2 節で述べた IB に記述されている 4 つの静的分岐予測結果と 4 つの動的分岐予測結果を比較して決める。結果の異なる分岐命令がある場合はその動的分岐予測先から始まる IB を、すべて一致した場合は 3.2 節で述べた IB に記述されている次にフェッチする IB を投機的にフェッチする。

実行例を図 5 に示す。IB1 の開始 PC とその先 4 つの動的分岐予測からフェッチ要求を出す。IB1 をフェッチしてから、図 5(a) のように、IB に付加された静的分岐予測と動的分岐予測を比較する。この場合、2 番目の分岐命令の予測が異なっている。IB は静的分岐予測に基づいて生成されるため、3 番目の BB は 2 番目の分岐命令の Not Taken の飛先の命令から始まる。しかし、動的分岐予測により Taken の飛先の命令が要求されている。そこで、図 5(b) のように Taken の飛先の命令から始まる IB2 を投機的にフェッチする。飛先は、2 番目の分岐命令はデコードした時に判明している。全体としては、図 5(c) に示すようなパスを実行する。このモデルは、4 章で述べた長所、短所を持つ。

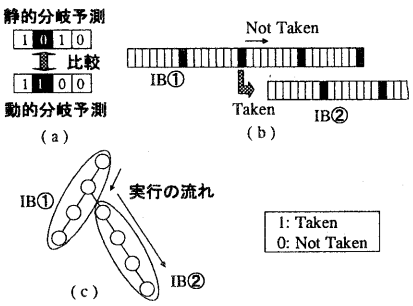

図 5　VLDP モデルのフェッチの様子

5.2 完全一致モデル

IB は Eager 方式により生成し、フェッチ要求は開始 PC とその先 3 つの分岐命令に対する動的分岐予測結果の 2 つから行う。開始 PC が一致し、分岐予測結果が完全一致する IB が IB バッファにある場合のみ IB バッファからフェッチする。IB バッファに存在しない場合はメインメモリにアクセスする。その IB の次に投機的にフェッチする IB の開始 PC は、先にフェッチした IB に付加されている。

このモデルは、4 章で述べた長所、短所を持つ。

5.3 部分一致モデル

VLDP モデルと完全一致モデルの中間の性質を持つモデルである。IB の生成方式とフェッチ要求は完全一致モデルと同様に行う。しかし、開始 PC の一致する IB が IB バッファ内にあれば少なくとも 1 つの同じ BB を含むので、その IB をフェッチする (部分一致)。但し、複数候補がある場合はより一致している数が多いものをフェッチする。完全一致する場合は完全一致モデルと同様に、部分一致する場合は VLDP モデルと同様に、一致する IB が IB バッファにない場合はメインメモリから完全一致する IB をフェッチ

する。また、完全一致した場合はIBに付加されている次にフェッチすべきIBとし、部分一致した場合はVLDPモデルと同様にして次にフェッチすべきIBを決定する。

これにより、完全一致モデルに比べフェッチ要求通りのIBはフェッチできない場合が多くなるが、IBバッファのヒット率は向上しメモリアクセス数の削減が期待できる。

6. 評　　価

マルチパス実行プロセッサでは、パス管理の戦略が性能に大きく影響を及ぼす。本論文の目的はIBの基本的性質の予備評価を行うことにあるため、パス管理を行わなくてよいシングルパス実行プロセッサのシミュレータ上で、5章で述べた3つのモデルの評価を行う。性能はIBバッファのヒット率、Fetched IPC(以下FIPC)、Retired IPC(以下、IPC)で比較する。

6.1 評価環境

6.1.1 動的分岐予測

動的分岐予測は、IBのフェッチ性能に大きな影響を及ぼす。本シミュレーションでは、gshare方式とbimodal方式を併せたCombining Branch Predictor[6]を用い、あるPCを与えるとその先4つの分岐命令の分岐予測を1サイクルで出るものとする。9命令以上分岐命令が出現しない場合は8命令目を分岐命令と見なし、分岐予測はNot Takenとする。動的分岐予測の精度を上げるため、インデックスの大きさは1Mエントリとする。

6.1.2 プロセッサモデル

プロセッサモデルは、シングルパス実行で毎サイクル1IBをフェッチ、処理可能なフェッチユニットと実行ユニットを持つものを想定する。フェッチしてから5サイクル後にIBは実行完了し、分岐結果などはそのサイクルに判明する。判明するまでは動的分岐予測により投機的にIBをフェッチし、分岐結果が判明して予測が正しければ実行を継続、誤っていれば投機的にフェッチしたIBをフラッシュし正しいIBからフェッチする。

IBバッファは、1ライン1IBで4way set associativeとし置換方式はLRUとする。また、1サイクルでIBをフェッチ可能とする。2次キャッシュは想定せず、IBバッファでミスした場合はメインメモリからフェッチする。このメインメモリアクセスレイテンシは30サイクルとする。

またIBバッファの容量をパラメータとし、純粋な命令領域の大きさを32KByte～1MByteで変化させる。

6.2 結　　果

図6にヒット率、図7にFIPC、図8にIPCの各モデルにおけるIBバッファの各容量における変化を示す。横軸が命令ブロックバッファの容量で縦軸がそれぞれヒット率とFIPC、IPCの値となっている。結果は、ヒット率、FIPC、IPCともにVLDPモデルが一番性能がよい。

また、図9に命令ブロックバッファの容量を256KBにした時の、各モデルのメモリアクセス回数とIB内の命令利用率を示す。IB内の命令利用率とは、実行したIBのなかで実際に実行された命令の割合のことである。

6.3 考　　察

図9に示す通り、VLDPモデルのメモリアクセス回数は部分一致モデルの約2分の1、完全一致モデルの約3分の1になっている。これは用意するIBの数がもともと他の2つに比べ少ないためIBバッファのコンフリクションが少ないからであり、図6に示すIBバッファのヒット率がVLDPモデルが一番高いことからも分かる。ヒット率が高いとメモリアクセスによる数～数十サイクルのミスペナルティを減らすことができ、フェッチ性能向上に繋がる。そのため、図7に示すようにFIPCにおいてもVLDPモデルが一番高い値になっている。一番ヒット率の悪い完全一致モデルはメモリアクセス回数が多いためミスペナルティが大きくなり、結果としてFIPCが一番低くなっている。VLDPモデルと完全一致モデルの中間の性質を持つ部分一致モデルでは、ヒット率、メモリアクセス回数ともに中間の値となっており、その結果FIPCも中間の値になっている。

一方、図8に示す通りIPCにおいてはそれぞれのモデルの差が小さくなっている。これは図9に示す通り、VLDPモデルと部分一致モデルはIB内の命令利用率が悪いのに対し完全一致モデルはIB内の命令利用率が良いためと考えられる。VLDPモデルと完全一致モデルの差は縮まったが、3倍ものメモリアクセス回数の違いが15%の命令利用率の違いより性能に影響し、VLDPモデルが優位を保っている。しかし、部分一モデルと完全一致モデルではほぼ同じ性能まで近づき、IB内の命令利用率の悪さがメモリアクセス回数の少なさを打ち消す結果となった。

以上から、VLDP方式のようにあるPCから始まるIBは1つのみにしIBバッファのヒット率を高くした方がメモリアクセス回数を大幅に減らすことができ、比較方式のように全部生成するより高い性能を出すことが可能であることが分かった。VLDPモデルではIB内の命令利用率が低くなるがメモリアクセス回数の減少を上回る性能低下はなく、また複数パス実行を行う場合はIB内の命令利用率のシングルパス実行よりも高くなることが予想されるため、それほど問題はないと考えられる。

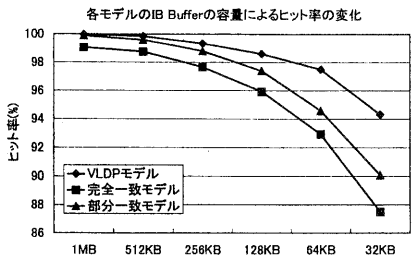

図 6　ヒット率の変化

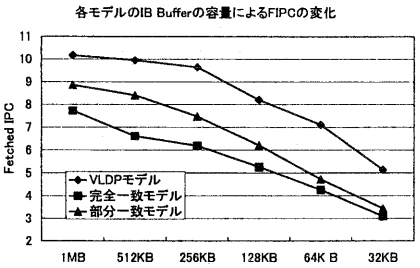

図 7　FIPC の変化

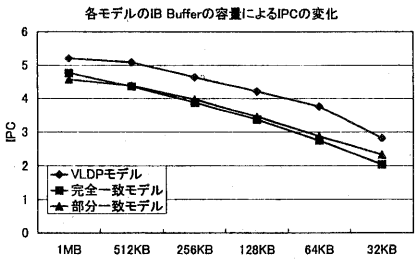

図 8　IPC の変化

7. おわりに

本論文では、VLDP アーキテクチャにおける IB の予備評価として IB の基本的な性質を示した。また、ある命令から始まる IB を生成する際、可能性のある 8 つすべて生成する方式と静的分岐予測を用いて 1 つだけ生成する方式を考え、シングルパス実行プロセッサ・モデルのシミュレータ上で性能評価を行った。その結果、静的分岐予測を用いて 1 つだけ生成するという VLDP アーキテクチャで採用した方式の方が性能が高いことを示した。

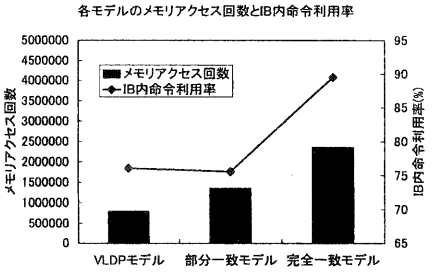

図 9　メモリアクセス回数と IB 内命令利用率

今後の方針は、シングルパス実行プロセッサ・モデル上で VLDP アーキテクチャで採用した IB の生成方式の有効性が示されたことから、複数パス実行プロセッサ上でも評価を行いその有効性を示していく。

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) Eric Rotenberg, Steve Bennett, and James E. Smith. Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching. *29th intl,Symp. on Microarchitecture*, pp. 24–34, December 1996.
2) S.Jeram Patel, D.Holmes Friendly, and YaleN. Patt. Critical Issues the Trace Cache Fetch Mechanism Technical Report. *CSE-TR-335-97*, 1997.
3) 辻秀典, 安島雄一郎, 坂井修一, 田中英彦. 大規模データパス・アーキテクチャの提案. *SWoPP2000*, 2000.
4) 安島雄一郎, 辻秀典, 坂井修一, 田中英彦. 大規模データパス・アーキテクチャの実行機構. *SWoPP2000*, 2000.
5) 日本デジタル イクイップメント株式会社監訳・編, 高澤 嘉光訳. 64 ビット RISC Alpha AXP アーキテクチャ概要. 1993.
6) S.McFarling. Combining Branch Predictors. Technical report. *WRL Technical Note TN-36*, 1993.