VLDP3：データフローを高速実行する大規模アーキテクチャ

入江 英嗣[†]　服部 直也[†]　山口 健輔[†]　谷地田 瞬[‡]　田中 裕治[†]
坂井 修一[†]　田中 英彦[†]

概要

我々は、全結合性がスーパスカラ方式の限界要因の一つであるとの立場から、限定した結合網上に多数の演算器を実装し、データフローを直接実行する ALU-Net 方式を検討している。ALU-Net 実行コアの課題を議論し、これを運用するアーキテクチャ、VLDP3 の提案を行う。シミュレータによる評価を行い、アーキテクチャの特性を議論する。

VLDP3:Large scale architecture which enables high speed data-flow execution.

Hidetsugu Irie[§]　Naoya Hattori[§]　Kensuke Yamaguchi[§]
Shun Yachida[¶]　Yuji Tanaka[§]　Shuichi Sakai[§]　Hidehiko Tanaka[§]

Abstract

We are researching on “ALU-Net”engine which directly executes dataflow with limited connection of a large amount of arithmetic unit. We examine ALU-Net execution core’s problem, and propose VLDP3 architecture which operate ALU-Net engine. We make evaluation and discussion on this architecture.

1　はじめに

コンピューティング能力の向上に関する社会の要請は留まる所を知らず、処理の中核となるマイクロプロセッサには、依然として高い性能向上率が求められている。

現行のプロセッサ性能向上はクロック周波数の向上によって得られているところが大きい。周波数の向上はデバイス技術の進歩によるものだけではなく、より深くパイプラインを設計するスーパパイプライニングとの両輪で支えられている。この結果、1 クロックの間のロジックはゲート 10 段分以下にもなり、理論的な最適点に近づいている。このため、今後は並列処理による処理能力向上が以前にも増して重要となる [1]。

一方、半導体微細化技術の進歩は目覚しく、今後 10 年以上堅調な見通しであり [3][2]、数億ものゲートから成る大規模プロセッサも現実のものとなりつつある。次世代プロセッサとして、このような大規模資源を活かして大規模並列実行を行うプロセッサが考えられる。

しかし、現在主流となっているスーパスカラアーキテクチャでは大規模化に際して以下のような問題があり、大規模資源を高速かつ有効に稼動させることが難しい。すなわち、分岐予測ミスペナルティの問題、issue logic のオーバヘッド、レジスタファイルポート負荷の増大、フォワーディングパスの肥大、メモリアクセスのオーバヘッド等の要素である。スーパスカラでは不用意な大規模化はかえって速度低下を招いてしまう。このため、現状では大規模資源は例えば on-chip の 2 次 cache 等に用いられている。また大規模 chip の将来像としては、効率的な規模のスーパスカラが複数搭載される on-chip multi processor 構成において、TLP 実行を行うモデル等が考えられている。

一方大規模化や高速化に適した新しいアーキテクチャの模索も活発になってきている。本稿では我々

[†] 東京大学
[‡] (株) 日立製作所
[§] Graduate School of Information Science and Technology, University of Tokyo
[¶] Hitachi, Ltd

の提案している VLDP3 アーキテクチャについて、実行方式の利点、導入した要素技術、アーキテクチャの傾向と課題等を示す。

2 関連研究

2.1 VLDP アーキテクチャ

我々は、将来のトランジスタ資源を仮定した大規模並列プロセッサの検討にあたり、プログラム処理の本質はデータフローであるとの立場から、多数の演算器上にデータパスを構築して大規模並列処理を行う、大規模データパス (Very Large Data Path:VLDP) アーキテクチャを提案し、プロセッサ構成や要素技術の研究を続けてきた [11][8]。

これまでの VLDP 研究で提案してきた大規模な “ALU-Net” 上にデータパスを構築する実行方式、静的に切り分けた “命令ブロック” を利用した処理、フィルタリングによる分岐制御などは VLDP3 でも用いられている。

2.2 次世代アーキテクチャの研究

スーパスカラ方式や VLIW 方式 [5] の限界を越える新しいアーキテクチャの模索として、Speculative Multithreaded Processor[6] や Instruction-Level Distributed Processing[4] 等、様様な検討が行われている。

GPA[?] は今後配線遅延の影響が支配的になるとの立場から、“ALU-Array” 上に命令を配置し、local-wire を利用した高速な転送を行うことにより、スケーラビリティの高いシステムを提案している。この方式は本稿で述べる VLDP3 の ALU-Net 方式と同様のものであるが、制御面での実装やコード生成面はまだ詳細にはなっていない。性能、評価の傾向等は VLDP3 と異なったものになると考えられる。

3 ALU-Net 方式

3.1 ALU-Net 方式の提案

スーパスカラ方式では、1 命令毎に全ての演算結果がレジスタファイルへ書き込まれる。また演算結果はフォワーディングパスを通じて他の全ての演算器の入力へブロードキャストされる (図 1(a))。このような 1 命令毎のループ、集積は命令ウィンドウにおける発行ロジックにも存在する (図 1(b))。条件が揃い発行された命令のタグは命令ウィンドウの全エントリにブロードキャストされる。これらブロードキャストの存在はスーパスカラ方式の大規模化を妨げる大きな要因となっている。

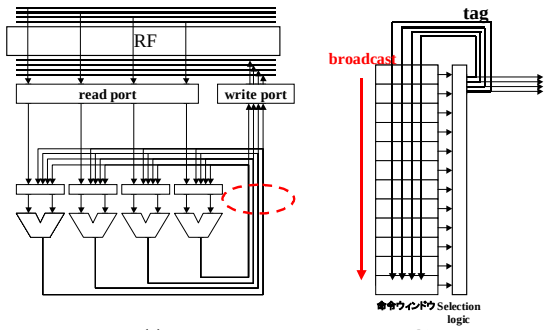

図 1: スーパスカラ方式の全結合性

スーパスカラ方式では全ての命令を “一様に” 扱うためブロードキャストが必要となっている。任意の演算器の出力結果を任意の演算器の入力とすることを可能とするため、あるいはその可能性が排除できないため、オーバヘッドの大きい冗長な転送を設けている。

分散化、階層化したり、コネクションに制限を加えるなどしてスーパスカラ方式のスケーラビリティを高めようとする試みもあるが、この場合、どのように配線等を間引くかということが問題となる。各命令を一様に扱ったままでは、本質的に全結合性が必要とされ、統計的な偏り等に頼らざるを得なくなってしまう。

プログラム処理の本質はデータフローであり、複数命令の連続である。データフローに着目して、必要な転送を必要な演算器にのみ行うことで配線と発行ロジックの負荷を軽減し、演算器数の大規模化を可能とする方式が ALU-Net 方式である。一つ一つの命令を発行し、次にどの命令が発行できるかを解析する、スーパスカラの動的スケジューリングとは異なり、ALU-Net 方式では実行機構上に予めデータパスを形成し、そこにデータを流すことによって演算を駆動させる。

提案する方式では、実行プログラムは複数命令のつながりから構成されるデータフローとして供給、

実行される。実行機構は多数の演算器の結合網で構成されており、ALU-Net と呼ばれる (図 2(a))。データフローは ALU-Net の演算器、配線上にそのまま形成される (図 2(b))。ALU-Net の各 node(演算器) に割り当てられた演算は、入力オペランドが揃い次第発火し、結果を次の node へ送信する。このようにして各 node の演算がカスケードし、データフローが実行される。この実行は、従来の一命令毎に集積ループの入る実行に較べて非常に高速に演算を連続させることができる。

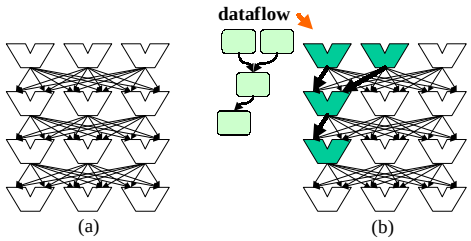

図 2: ALU-Net

1) データフローの特性により、ALU-Net は完全結合ではなく、制限された配線量で構成できること、2) データフロー的な実行により issue logic が ALU-Net 上で吸収できることが大きな利点であり特徴である。

ALU-Net 方式では、スーパスカラ方式における全結合性を排することができ、高速大規模化が可能である。無論 ALU-Net には実行時に稼動しない冗長な配線、node が生ずるが、スーパスカラにおける冗長性（ブロードキャスト等）とは異なり、実行時のクリティカルパスに大きな影響を与えない。

ALU-Net 方式は、従来の少数の演算器を効率よく使用するための動的スケジューリング手法から、大量の演算器を実装可能にするための手法、大量の演算器を利用して演算を高速化する手法に転換している。この転換は、演算器を多数実装することがテクノロジー的に可能になること、また TLP 実行やメディア演算など、多数の演算器を駆動する利点が生じてくること等により今後より進むと考えられる。

また、ALU-Net 方式の大きな利点として、レジスタファイルのアクセス負荷を減らすことができる。従来方式では、生成された演算結果が即参照され、その後は参照されないといった、一時変数にもレジスタが用いられるが、ALU-Net 方式ではそのような値は node と node を結ぶ local-wire 転送として処理され、レジスタファイルを圧迫しない。

3.2 ALU-Net 方式の問題点

このように、演算器数の拡張と高速稼動を両立させ、また配線資源や転送レイテンシを節約できる ALU-Net 方式だが、汎用プロセッサの実行コアとして導入するためには、以下のような問題を解決する要素技術が必要である。

node への命令割り当て：ALU-Net 方式の場合、実行しようとするデータフローを解析し、制限された結合網である ALU-Net 上に命令を配置しなければならない。

ALU-Net 各 node の制御：多数 node を制御し、データフローおよびコントロールフローを高速に処理しなければならない。特に汎用プログラムでは分岐処理を ALU-Net コアで高速に行う技術が必要となる。これには、分岐予測に基づく投機実行や、その巻き戻しによる演算の部分破棄などの複雑な制御を導入しなければならない。また、例外処理にも対応できなければならない。

多数演算器の稼働：ALU-Net 方式によって大規模化が可能となった演算器資源について、稼働率を確保し、実行並列度向上を達成しなくてはならない。

4 VLDP3 アーキテクチャ

この節では、前節に挙げたような問題を解決し、ALU-Net 実行コアを運用するプロセッサモデル、VLDP3 アーキテクチャを提案する。

4.1 静的解析の利用

IB の導入：動的解析による実行前処理の増加や、ALU-Net トポロジで対応できないデータフローの発生を避けるため、VLDP3 ではコンパイラによる静的解析を用いる。

分岐命令、ジャンプ命令などによる制御依存、メモリアクセス命令による曖昧な依存関係の存在のため、実行プログラム全体のデータフローを静的に解析することは一般に困難である。しかし、ジャンプ命令を跨がないプログラム断片であれば、断片内のデータフローはレジスタについては完全に解析する

ことができる。VLDP3 ではこのような断片に、静的解析して得た配置情報を加えたものを命令ブロック (以降 IB) と呼び、ALU-Net の制御単位とする。実行プログラムは多数の IB の集合として与えられる。IB 実行時には、静的解析による配置情報、入出力情報をそのまま ALU-Net にアサインすることで、高速にデータフローが形成される。

一般に IB の大きさを大きくするほど Local-Wire による高速な転送の割合を増やすことができ、効率的な実行が期待できる。このため、IB は制御流に関して単純な BB（ベーシックブロック：汎用プログラムでは数命令程度の大きさ）ではなく、内部に複数の分岐命令と複数の制御流を含むハイパブロック [7] で構成される。

4.2 IB 単位の実行

VLDP3 では 1IB をまとめてフェッチ、デコードし、ALU-Net 上に展開する。1 命令単位でデータパスに追加配置するような制御は行わず、異なる IB ではデータパスは切れているものとして別々に処理する。異なる IB 間にデータ依存がある場合、ALU-Net の外にあるレジスタファイルを利用してデータを転送する。

このように、VLDP3 では IB 内については静的にデータフローを保証し、データフローマシン的な実行を行う。一方で、IB の単位でスーパスカラ的な実行を行う。異なる IB 同士も out-of-order に実行され、レジスタは IB 単位でリネーミングされる。

node の稼働率を高めるため、ALU-Net 上では同時に複数の IB が多重実行される。これは、各 node の待ち合わせ用バッファを複数エントリにし、送信データに、どの IB に対するものかを示す IBID を加えることで実現できる。この多重化は SMT 処理にも適用でき、シングルスレッドに潜在する並列性の限界を越えて node の稼働率を高めることができる。

4.3 分岐制御の実現

IB には分岐命令が含まれるため、IB 内のどの命令が分岐によって有効となり、どの命令が無効となるかは実行時まで判明しない。そこで、IB は予想される分岐パターンの数だけ OMask というフィルタを用意している。OMask は対応する分岐パターンのときに、IB で生成する全てのレジスタ/メモリ書き込みのうち、どれが有効となり、どれが無効となるかを表す bit 列である。

決定したフィルタを通してレジスタ/メモリ書きこみを行うことで IB 内の部分破棄が、フィルタを通して NPC を set することで次に実行されるアドレスの分岐が、それぞれ実現される。

このようなフィルタリング方式により predicate や multipath といった複雑なコントロールフローをそのまま ALU-Net 上で実行することができる。

4.4 node 間相互通信の排除

一度アサインされた命令は、分岐結果に関わらず基本的に全実行される。各 node では実行中の演算が有効になるかを把握する必要はなく、それらのフィルタリング処理は ALU-Net 外部の RWU(read write unit) や LSU(load store unit) で行われる。node から演算結果を即出力させ外部でフィルタリングすることにより、ALU-Net の制御は、全 node につき IB アサインと、全 node につき IBflush の 2 種類だけとなる。

VLDP3 では IB 多重実行の効率を高めるために、OMask、NPC を予測し、積極的に投機実行を行う。node から出力されたレジスタ値等は投機的フィルタリングを通して後続 IB に転送され、分岐決定前に実行を先へ進めることを可能としている。

5 評価

VLDP3 プロセッサ初期モデルを策定し、その実行タイミングをサイクル単位で再現するシミュレータおよび、IB を生成するコンパイラを作成し、評価を行った。

5.1 入力

SPEC95 から compress,go,li,m88k-sim,perl を用い、それぞれのプログラムについて IB を作成し、シミュレータへの入力とした。入力パターンは train を用い、最後まで実行させた。

IB は最大 64 命令の制約で作成したものを用いた。またここではモデルの基本的な傾向を調べるため、ALU-Net トポロジを完全結合であるとして IB を作成した。VLDP3 コンパイラの作成と、ALU-Net ト

ポロジによる IB の品質については、関連研究 [10] で述べている。

5.2 シミュレーションモデル

シミュレーションモデルのブロック図を図 5.2 に示す。策定した仕様に基づき、シミュレータは以下

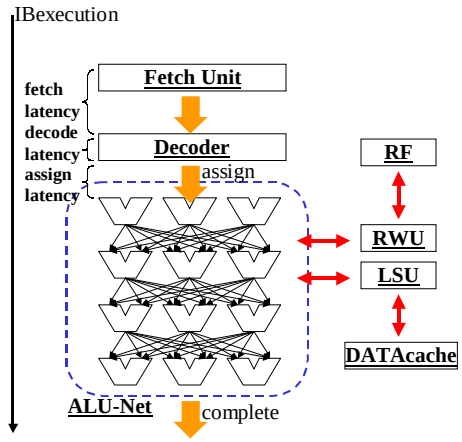

図 3: シミュレーションモデル

のような実行を再現する。クロック周波数は現行のスーパスカラと同様に高クロックを想定している。これは IPC を落とすことになるが、スーパスカラではこのような細かいパイプラインが性能の最適解となっている。VLDP3 における clock-IPC セッティングの最適化は今後の課題である。

実行前処理:予測アドレスに基づき毎サイクル 1IB を一度にフェッチする。分岐予測機構として、単純な LastValuePredictor を用い分岐方向 (OMask) の予測を行っている。デコード、アサインとも数サイクルかかるがパイプライン化されている。ALU-Net 上には、設定された多重度の数だけ複数の IB をアサインできる。

実行：アサインされた各 node はオペランドが揃うと演算を行い、結果を出力する。演算結果の転送は、IB 内の Loacl-Wire を用いた転送と、ALU-Net 外のユニットを経由する転送の 2 種類が存在する。IB 内の全ての演算が終了すると、IB は ALU-Net から flush され、新しい IB がアサイン可能となる。

メモリアクセス：メモリリクエストは LSU に集積され、依存関係のチェックが行われる。load 命令は先行ストア命令の計算を待たずに依存関係投機アクセスを行っている。このシミュレータではキャッシュは 100%hit を仮定している

5.3 転送レイテンシによる影響

VLDP3 では IB 内のデータ依存は ALU-Net 内の Local-Wire による高速な転送が行われる。一方、IB を跨いだレジスタ依存やメモリアクセスは、各 node から ALU-Net 外のユニットへのアクセスが必要となる。初期モデルでは ALU-Net 外へアクセスするための線が全 node に存在する。しかし、このような全結合に近いアクセスには Local-Wire による転送と違い、ネットワークのスイッチングや通過に関するレイテンシが避けられない (図 4)。

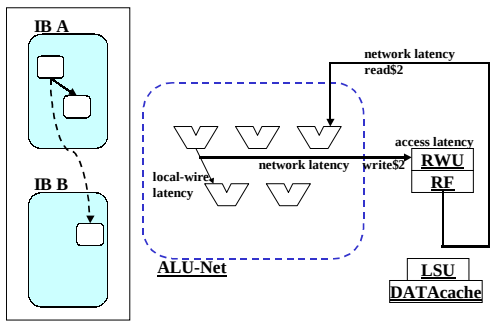

図 4: データ転送

この特性を調べるために、外部ユニットへの転送レイテンシを変化させて IPC を評価し影響を調べた。また、転送レイテンシが一様なモデルを用意し、スーパスカラに近い値として比較した。パラメタの諸元を表 1 に示す。

表中の network latency は往路復路共に影響する。つまり、VLDP-latency2+2 パラメタでは、IB 間データ依存の転送には、レジスタアクセス 2 サイクルに加えて network latency2 サイクルが往復 2 回、計 6 サイクルかかることになる。同様にメモリアクセスには $2\times2+3=7$ サイクルが少なくとも必要となる。メモリアクセスの場合、先行ストアの待ち合わせ等があった場合更にレイテンシは大きくなる。一方の SS パラメタは network latency のようなペナルティはなく、全ての転送が、設定された local latency で行われる。これは全ての演算器を全結合、ブロードキャストすることに相当し、64 演算器という規模では、ここで設定した SS1〜3 サイクルという数字は

共通パラメタ

IB最大多重度	**8**
実行前レイテンシ (fetch-decode-assign)	**6**
RF access latency	**2**
cache access latency	**3**

転送レイテンシ

	local-wire-latency	network-latency
VLDP-latency1+1	0	1
VLDP-latency2+2	0	2
VLDP-latency3+3	0	3
SS-latency1	1	0
SS-latency2	2	0
SS-latency3	3	0

表 1: パラメタ諸元

実装不可能な値に近い。

それぞれのベンチマークについて IPC を測定した結果を図 5 に示す。

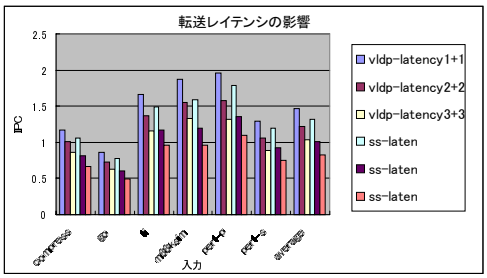

図 5: レイテンシパラメタと IPC

グラフではレイテンシの増大に伴い両方式とも同程度の傾きで IPC を低下させているが、VLDP パラメタの方は 2 倍の増加であることを考えると、大規模化時の耐性は network latency のペナルティを容認しても、VLDP3 方式が有利であると言える。

6 まとめ

本稿では、スーパスカラ方式の限界の要因として全結合性を挙げ、制限された結合網上にデータパスを形成、実行する ALU-Net 方式を示した。また、新しい実行方式である ALU-Net 方式について、それを運用するための VLDP3 モデルを提案し、提案モデルによる実行のシミュレート結果を議論した。

本稿で示したように、VLDP3 モデルでは ALU-Net と外部ユニットを結ぶ network-latency が性能に大きな影響を及ぼす。これは、この部分に全結合性が排除されていないためであり、高速化する forwarding 手法を検討中である [12]。同様に外部ユニットを用いるメモリアクセスは、更に性能に与える影響は大きく、新しいアーキテクチャによりこの課題を解決することが VLDP3 の大きな目標である。ALU-Net の特性を利用した wire-promotion[9] 等を検討中である。

また、コンパイラ技術が VLDP3 では非常に重要となっている。コンパイラによる VLDP3 コード最適化の研究や、コンパイラ支援を前提としたアーキテクチャ技術の検討など、アーキテクチャ-コンパイラ co-design による研究を今後も続けていく。

参考文献

[1] Vikas Agarwal, M.S. Hrishikesh, Stephen W. Keckler, and Doug Burger. Clock Rate versus IPC The End of the Road for Conventional Microarchitectures. *ACM ISCA 2000*, pp. 248–259.

[2] Alan Allan, Don Edenfeld, William H.Joyner Jr., Andrew B.Kahng, Mike Rodgers, and Yervant Zorian. 2001 technology roadmap for semiconductors. *IEEE Computer*, Vol. 2002-02, pp. 42–53.

[3] Semiconductor Industry Association. Summary of the international technology roadmap for semiconductors. *February 6,2002*.

[4] James E.Smith. Instruction-level distributed processing. *IEEE Computer*, Vol. 2001-04, pp. 59–65.

[5] Michael S.Schlansker and B.Ramakrishna Rau. Epic:explicitly parallel instruction computing. *IEEE Computer*, Vol. 2000-02, pp. 37–45.

[6] Gurindar S.Sohi and Amir Roth. Speculative multithreaded processors. *IEEE Computer*, Vol. 2001-04, pp. 66–73.

[7] 中田育男. コンパイラの構成と最適化. 朝倉書店, 1999.

[8] 辻秀典, 安島雄一郎, 坂井修一, 田中英彦. 大規模データパス・プロセッサの提案. 情報処理学会研究報告 計算機アーキテクチャ研究会 2000-ARC-139, Vol. 2000, No. 74, pp. 55–60.

[9] 谷地田瞬, 山口健輔, 田中裕治, 服部直也, 入江英嗣, 飯塚大介, 坂井修一, 田中英彦. VLDP3 アーキテクチャの構想 (4) ～メモリ依存に関する初期検討～. 第 1 回 情報科学技術フォーラム FIT 2002, No. C-17.

[10] 服部直也, 入江英嗣, 田中裕治, 坂井修一, 田中英彦. VLDP3 アーキテクチャに対するコード生成の初期検討. *SHINING2003*.

[11] 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-128, Vol. 97, No. 61, pp. 13–18.

[12] 田中裕治, 山口健輔, 谷地田瞬, 服部直也, 入江英嗣, 飯塚大介, 坂井修一, 田中英彦. VLDP3 アーキテクチャの構想 (3) ～レジスタフォワーディング機構の初期検討～. 第 1 回 情報科学技術フォーラム FIT 2002, No. C-16.