VLDP3 アーキテクチャに対するコード生成の検討

服部 直也　入江 英嗣　田中 裕治　坂井 修一　田中 英彦

概要

これまでマイクロプロセッサのクロック周波数は飛躍的に向上してきたが、スーパーパイプライニングによるクロック向上は、既に理論限界に近いことが指摘されている。こうした背景から我々は、高クロックかつ高 IPC を実現するために小さな演算ノードを大規模に分散させ、限定的なネットワークで接続する ALU-Net 方式と、それを搭載する VLDP3 アーキテクチャを提案している。VLDP3 では限定的なネットワークで正しく高速なデータ転送を行うために ISA レベルで命令配置ノードを指定する仕様となっているが、具体的なコード生成法は未検討であった。本研究では、VLDP3 の限定的な Bypass Logic に対するコード生成手法を提案・評価し、その基本方針の正しさを確認した。

Preliminary Evaluation on A Code Generation System for VLDP3

Naoya Hattori　Hidetsugu Irie　Yuji Tanaka
Shuichi Sakai　Hidehiko Tanaka

Abstract

The clock rate of microprocessors has improved with deeper pipelining in recent years. But the deeper pipelining technique will reach it's limit in near future. So, next generation microprocessor must archive both high clock rate and high IPC. In this purpose, we had proposed ALU-Net model, composed of many computation nodes and limited bypass logic to connect them, and VLDP3 architecture to support it. In VLDP3 ISA, compiler specifys the execution node for each instruction to run the applications on the limited bypass logic. But such code generation mechanism had not been discussed yet. In this paper, we propose a code generation system for VLDP3 and preliminary evaluate it. And the result shows the adequacy of our strategy.

1　はじめに

これまでマイクロプロセッサの性能は、ムーアの法則通りに年 50%程度の割合で指数関数的に向上してきた。マイクロプロセッサの性能はクロック周波数とクロック当たりに処理する命令数 (IPC) の積に比例するが、ここ数年の性能向上の大部分はスーパーパイプライニングによるクロック周波数向上に支えられてきた。しかしこの傾向は長く続かず、パイプライン化による性能向上の上限が近いことが指摘されている [1]。このことから、今後のプロセッサアーキテクチャには、クロックだけでなく IPC も向上させることが期待されている。

その一方で、近年のマイクロプロセッサの IPC は低下している。これは Register File, Issue Logic, Bypass Logic といった並列性抽出に重要なロジックが配線遅延の影響を受けやすいことに因ると指摘されている [1, 6]。従来型アーキテクチャで高クロック動作のプロセッサを設計する場合、これらのロジックを小規模化するか、レイテンシを容認するかの選択を迫られるが、いずれにしても抽出できる並列度が低下し、高 IPC の実現は難しくなる。半導体技術が微細化するに連れて、ゲート遅延に対する配線遅延の割合は増加することから、この傾向はますます大き

東京大学情報理工学系研究科
Graduate School of Information Science and Technology, University of Tokyo

くなると予想されている。そのため今後の高性能 ILP プロセッサとしては、(1) 多数分散した小規模 Register File と Issue Logic、(2) それらを限定的に接続する簡単かつ高速な Bypass Logic で構成されることが重要になる。

これらの背景から VLDP [10] やその最新モデルである VLDP3 [9]、また GPA-1 [5] といった大規模な汎用プロセッサアーキテクチャが提案されている。VLDP3, GPA-1 のいずれも、限定的な Bypass Logic でアプリケーションを正しく実行するために、コンパイラが座標付きコードを出力することを想定している。しかし具体的なコード生成手法が未だ検討されておらず、これらのアーキテクチャの妥当性を主張する上で問題になっていた。

以上の状況を踏まえて、本稿では VLDP3 のコード要件を述べ、コード生成システムを提示する。そして Bypass Logic を限定したアーキテクチャに対するソフトウェア支援として、グラフ論的な演算ノード割り当てを提案・評価する。

2 VLDP3 アーキテクチャ

2.1 ALU-Net

VLDP3 [9] アーキテクチャの分散基本ユニット (以下ノードと呼ぶ) は、図 1 に示すような、1 命令を処理するための ALU と小規模な Register, Issue Logic から成る。個々のノードはデータが送られることで起動し、その演算結果は Router を介して少数のノードへ Bypass される。このように限定的に接続された多数のノード群を ALU-Net と呼ぶ。初期実装モデルである VLDP 3.0 では、Bypass 先を 図 2 のように下方隣接 3 ノードに限定している。VLDP3 では限定された高速な Bypass Logic を用いて、プログラムを正しく、効率良く実行するために、ISA レベルで命令を割り当てるノード及び演算結果の送り先ノードを指定する。

2.2 ALU-Net と外部ユニット

VLDP3 には演算装置である ALU-Net の他に、命令供給ユニットレジスタ管理ユニットやメモリ管理ユニット等の外部ユニットが存在する (図 3)。ALU-Net に割り当てられた命令は、ネットワークを介してレジスタ・メモリへの読み書きが可能である。尚、ALU-Net と外部ユニットを結ぶネットワークに関しては現在検討中であるため、本稿では全てのノードが外部ユニットと間接的に通信可能であることを仮定する。

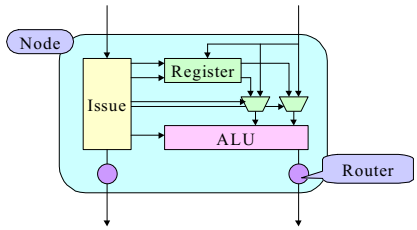

図 1: ALU-Net のノード構成

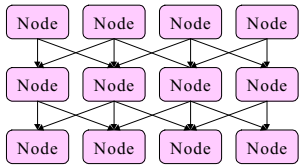

図 2: VLDP 3.0 における ALU-Net トポロジ

2.3 ALU-Net と Data Flow

各ノード及びその集合である ALU-Net は、データ駆動式に動作するので、命令配置を考える際は部分プログラム中の Data Flow に着目する。例えば $c = (a - b) \times 3$ という演算からは図 4 左 のような Data Flow Graph (DFG) が得ら

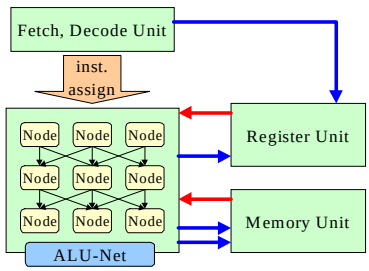

図 3: ALU-Net と外部ユニット

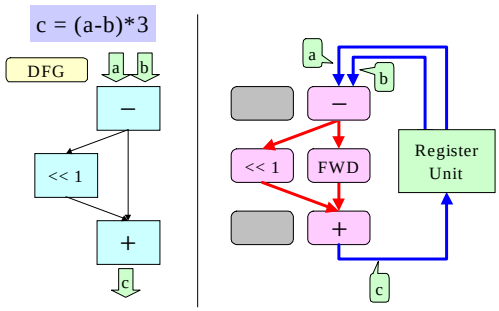

図 4: ALU-Net の使用例

(左図) c=(a-b)*3 のデータフロー解析結果。
(右図) ALU-Net への配置結果。

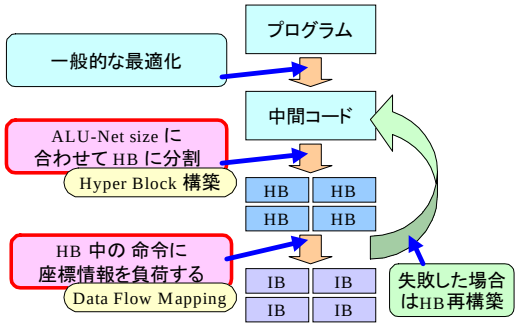

図 5: VLDP3 コード生成システム

IB は座標情報を付加した Hyper Block のこと。

れる。この Data Flow を ALU-Net 上に配置する場合、減算と加算は “高さ” に 2 段分の差があるため、VLDP 3.0 のトポロジでは直接結合することはできない。このような場合には適宜中継命令 (図 4 の FWD) を挟むことが必要になり、最終的に 図 4 右 のように配置することができる。この部分プログラムを実行する際は、レジスタユニットから a, b の値が到着するのに伴って減算が発火し、その結果は高速 Bypass Logic を経て左シフトと中継ノードへ送られる。続いて左シフト、加算が連鎖的に発火し、最終的にレジスタユニットへ c の値を出力する。

2.4 ALU-Net と Control Flow

VLDP3 では、ALU-Net への命令供給、演算終了判定、予測ミスに伴う演算破棄などを命令単位ではなく Control Flow の塊を単位として制御する。この塊が大きい程 (1) 命令当たりの制御オーバーヘッドが軽減できる。(2) 多くのデータを高速に Bypass できる。という利点がある。そのため、VLDP3 では ALU-Net 制御単位として Hyper Block を採用している。Hyper Block は単一の入口のみを持つ連続した Control Flow で、分岐・合流や複数の出口があっても構わない [8] 。Hyper Block の大きさは上限は ALU-Net のノード数で決まっているため、限られたノードを如何に有効な (retire する) 命令で埋めるかが重要になる。

3 VLDP3 コード生成システム

3.1 システムの概要

前章で述べた VLDP3 の仕様から、VLDP3 コンパイラでは一般的なコード最適化の他に、

- 実行確率の高い命令を多数含んだ Hyper Block を構築する
- Hyper Block 内の Data Flow を ALU-Net の形に整形し、座標情報を付加する

といった、IB 中の有効命令数を増やすための最適化が重要になる。

これらの要件は、一般プロセッサ用コンパイラの後段に、図 5 に示すコード生成システムを追加することで実現できる。尚我々は、座標情報を付加した Hyper Block のことを Instruction Block (IB) と呼んでいる。

Hyper Block 構築では、コンパイラ中間コードの Control Flow を解析し、ALU-Net の大きさを越えない範囲で Hyper Block を構築する。Hyper Block には分岐の両パス (複数パス) を含めることが可能であるが、パス選択の際には Loop などの分岐確率の高いパスを優先する。また、必要に応じて Predicate などのコード変形も行って無効命令数の増加を防ぐ。

そして Data Flow Mapping では Hyper Block 内の Data Flow を解析し、座標情報を付加するために ALU-Net で実行可能な形に整形する。Data Flow Mapping の具体的な処理内容は次節で述べるが、Mapping は常に成功するわけでは

なく、ALU-Net に収まらない Hyper Block も存在し得る。そのような Hyper Block を構築してしまった場合には、より小さな、或いは他のパスを選択して Hyper Block を再構築する。

3.2 本稿で提案する Mapping 手法

Data Flow Graph を特定の形に変形する手法としては、(1) Simulated Annealing [2] や Genetic Algorithm [4] といった、CAD 等で用いられている確率的なアプローチ、(2) 有向グラフを人間の見やすい形に変形して表示する [7] 場合などに用いられる、グラフの形そのものを最適化するグラフ論的なアプローチ、の 2 種を考えることができる。いずれのアプローチもその有効性は確立されているが、本研究では、(1) 扱う対象 (DFG) が有向グラフであること、(2) 確率的なアプローチは収束に時間を要すること、(3) グラフ論的なアプローチは、確率的なアプローチに対する初期配置としても使えること、などの理由から、手始めにグラフ論的なアプローチを検討した。

この Mapping 手法の中核はグラフ最適化であるが、入力が実プログラムの Data Flow Graph であり、出力が ALU-Net 内の位置情報であることに合わせて、前処理と後処理が必要になる。これらを加味すると Mapping の全体像は以下の手順になる。

1. Fanout 調整 (前処理)
2. Layering (y 座標決定)
3. Crossing Reduction
4. Coordinates Calculation (x 座標決定)
5. Layout 決定 (後処理)

Fanout 数調整 VLDP 3.0 では演算結果を Bypass できるノードを下方隣接 3 方向に限定しているため Fanout (Bypass 先の数) が 4 以上である命令を配置するには 図 6 のように中継命令を挟むことが必要になる。この目的で Data Flow Graph 中のクリティカルパスを解析し、非クリティカルパスに優先して中継命令を挟入する前処理を行う。

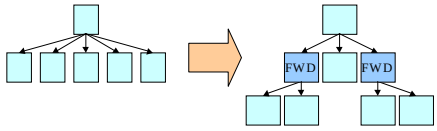

図 6: Fanout 数調整
出力数が多い場合は中継ノードを挟んで 3 以下にする。

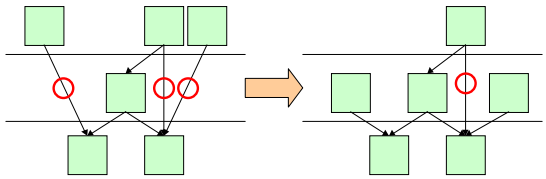

図 7: Layering
エッジ両端の y 座標を可能な限り近づける。
○は中継ノードが必要なエッジ。

Layering Layering はグラフ中のノードに “高さ” を定義する処理であり、この操作で (相対的な) y 座標が決定する。Layering の各種アルゴリズムは文献 [3] で比較されている。本研究では ALU-Net に対する無駄な命令配置を避けるため、中継ノードを最小にするアルゴリズム (図 7) を採用した。

Crossing Reduction と Coordinates Calculation VLDP 3.0 では各ノードの出力を隣接 3 箇所にしか Bypass できないため、Data Flow Graph のエッジ両端ノードを極めて近くに配置する必要があり、そのためにはエッジの交差が少ない方が望ましい。一方グラフ描画の分野でも、人間が見やすい図を示すために、エッジの交差数を減らすグラフ最適化 (Crossing Reduction) やエッジ両端のノードを近くに配置する座標計算方法が研究されている [7]。両者は目的が似ているため、Data Flow Mapping にもグラフ描画の手法が応用可能であると考えられる。

Crossing Reduction は同 Layer 内のノードの並び順を変えて交差を減らす最適化で、文献 [7] では各ノードを、「接続ノードの順番」の平均値が小さい順に並び替えている (図 8) 。この最適化は並び順が収束するまで、上下方向から繰り返し実行する。

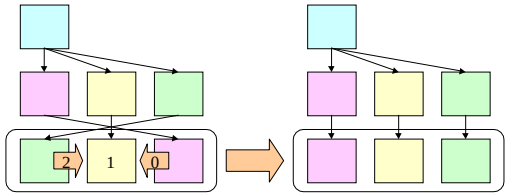

図 8: Crossing Reduction

左図の最下段のノードはそれぞれ 2, 1, 0 番目のノードと接続しているため、並び替えの後は右図のように整形される。

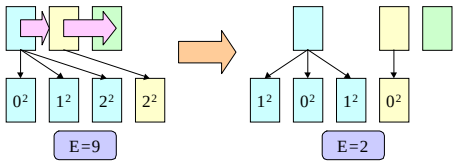

図 9: Coordinates Calculation

各エッジはバネのような位置エネルギーを持ち、ノードはその合計が最小になる位置に落ち着く。

続く Coordinates Calculation では Layer 内の並び順を崩さずに、各ノードに x 座標を定義する。文献 [7] のアルゴリズムでは、各エッジがバネのように両端ノードの Δx^2 に比例する位置エネルギーを持つとして、エネルギーを最小にする x 座標を算出する (図 9)。

Layout 決定 ここまでで Data Flow 中の命令位置が決定するが一般に、Hyper Block 中には複数の Data Flow が存在するため、それらを ALU-Net に配置する処理が必要になる。この問題の解法はいくつか考えられるが、今回は最も単純に、座標を順に探索して配置可能な最初の位置に配置する (first hit) アルゴリズムを、ノード数が多い Data Flow から順に適用した (図 10)。

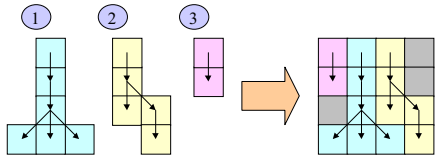

図 10: Layout 決定

数値は配置順。ノード数の多い Data Flow を優先する。

4 VLDP3 コード生成の評価

4.1 評価環境

提案したシステムの評価として、SPECint95 の compress, li, go, ijpeg, m88ksim, perl (primes), perl (scrabbl) に対して実際に Hyper Block 構築と Data Flow Mapping を行い、VLDP 3.0 で想定している 8×8 の ALU-Net 対するコードと、比較対象として 64 ノードの完全結合 ALU-Net に対するコードを生成した。そしてシミュレータを用いて全命令を実行し、Retire 命令数 / IB を測定した。この数値は大きい程多くのデータを高速に Bypass できるため、Hyper Block の作り方や Mapping の品質を表す指標として用いた。尚、8 ノードで構成される Layer 8 段並べた ALU-Net を、8×8 ALU-Net と略記した。

今回の評価では、Predicate は全く使用していない。しかし、Hyper Block 構築時に実行プロファイルを用いて最適なパス選択を行ったため、Predicate を使わないという制限の下で最適な Hyper Block を構築している。

Data Flow Mappr として、前述の提案手法のうち Fanout 調整以外を実装した。そのため今回の評価では、Fanout 4 以上の命令を含む Hyper Block の Mapping は確実に失敗する。

4.2 比較結果

提案手法を用いて、64 ノードの完全 Bypass Logic を仮定した、“完全結合 ALU-Net” に対するコードと、8×8 の限定 Bypass ALU-Net に対して作成したコードの品質を、図 11 に示す。

アプリケーション毎に異なるが、8×8、3 方向の限定 Bypass Logic を用いることで、完全結合網と比べて 3 割から 5 割以上もコードの質が低下していることが明らかになった。

8×8 限定 Bypass ALU-Net コードの品質が低下している原因としては、(1) 8 Layer で構成されているため、8 段以上の Data Flow を Mapping できないこと。(2) Fanout 調整が未実装であるために、Fanout が 4 以上の命令を Mapping で

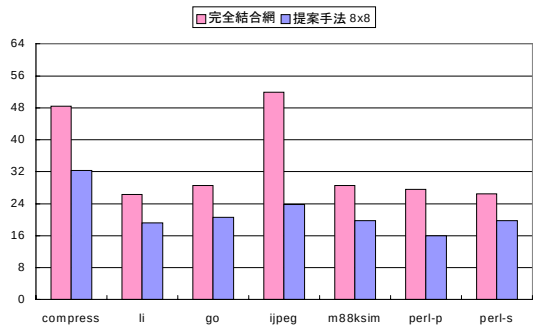

図 11: 完全結合 ALU-Net と VLDP 3.0 の ALU-Net に対するコード品質の比較

縦軸は IB 当たりの retire 命令数 (大きいほど高品質)

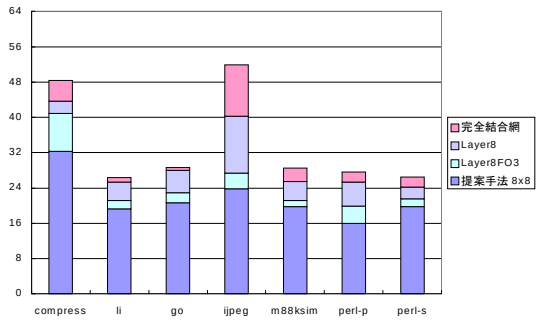

図 12: 各種 ALU-Net に対するコード品質の比較

縦軸は IB 当たりの retire 命令数 (大きいほど高品質)

きないこと。(3) 実装した提案手法自体の問題。の 3 つの要因が考えられる。個々の要因を調査するために、完全結合網に 8 Layer 制限を加えた仮想的な ALU-Net (Layer8 と略記)、Layer8 に Fanout ≤ 3 制限を加えた仮想的な ALU-Net (Layer8FO3 と略記)、に対して同様の調査を行った。結果を 図 12 に示す。

コード品質の低下が最も激しかった ijpeg を始め、compress 以外のアプリケーションでは品質低下要因の大部分が 8 Layer 制限と Fanout3 制限によるものであった。ここから、ALU-Net の形には再考の余地が大きいこと、Fanout 調整は重要な最適化であることが明らかになった。また、Layer8FO3 と提案手法の差は 3 命令分程度に抑えられており、改善の余地はあるものの、今回提案した Data Flow Mapping の方向性は正しかったと考えている。

5 まとめ

本稿では高クロックと高 IPC を両立させるための ALU-Net 方式を紹介し、ALU-Net の限定された Bypass Logic を用いてプログラムを実行するためのコード生成システムとして、グラフ論的アプローチを採用した Data Flow Mapper を提案した。提案システムを評価した結果、Bypass Logic を限定しない場合と比べて大幅なコード品質の低下が見られたが、その主な原因は想定したトポロジと未実装最適化にあることを確認した。今後は、明らかになった問題点に対して更に検討を進めていく。

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] Pei-Ning Guo, Toshihiko Takahashi, Chung-Kuan Cheng, and Takeshi Yoshimura. Floorplanning Using a Tree Representation. *IEEE Trans. CAD 2001*, Vol. 20, No. 2, pp. 281–289.

[3] Patrick Healy and Nikola S. Nikolov. How to Layer a Directed Acyclic Graph. *Graph Drawing: 9th International Symposium, GD 2001*, Vol. 2265, pp. 16–30.

[4] Pinaki Mazumder and Elizabeth M. Rudnick. *Digital Systems Design and Prototyping Using Field Programmable Logic*. Prentice Hall PTR, 1999.

[5] Ramadass Nagarajan, Karthikeyan Sankaralingam, Doug Burger, and Stephen W. Keckler. A design space evaluation of grid processor architectures. *ACM MICRO 2001*, pp. 40–51.

[6] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. Complexity-Effective Superscalar Processors. *ACM ISCA 1997*, pp. 206–218.

[7] G. Sander. Graph Layout through the VCG Tool. *Technical Report A03-94, Universität des Saarlandes, FB 14 Informatik*, 1994.

[8] 中田育男. コンパイラの構成と最適化. 朝倉書店, 1999.

[9] 入江英嗣, 山口健輔, 谷地田瞬, 田中裕治, 服部直也, 飯塚大介, 坂井修一, 田中英彦. VLDP3 アーキテクチャの構想 (1) ～プロセッサ構成～. 第 1 回 情報科学技術フォーラム FIT 2002, No. C-14.

[10] 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.