Vol.2010-SLDM-146 No.5
2010/10/5

# バイパスアーキテクチャ向けコード最適化における演算命令のクラスタリングを利用した改良手法に関する研究

鎌田裕基† 庄司俊寛†† 田金†† 杉野暢彦††

近年、プロセッサの技術進歩に伴い、消費電力の増加が大きな問題となってきている。消費電力の中でも、プロセッサとレジスタとの通信電力が比較的高い割合を占めている。これを削減するために、バイパス構造をもつアーキテクチャ、及びこのアーキテクチャを効率よく使用するための DFG を利用したスケジューリング方法が提案されている。
本稿では提案されている方法における、DFG の再構成の方法について改良し、再構成中に DFG を分割することにより、スケジューリングにかかる時間を削減する方法について提案する。提案方法を適応することで、バイパスの使用効率の上昇と、スケジューリングにかかる時間の短縮が見込める。

# An Improved Code Optimization Method based on DFG Formulation for a Processor Architecture with a Bypass Chain

Yuki KAMADA† Toshihiro SHOJI †† Jin TIAN††
Nobuhiko SUGINO ††

For a processor with a bypass chain, an improved code optimization method based on data flow graph (DFG) form is discussed. A compiler with the existing post-pass code optimization method generates a code, which utilize the bypass chain as much as possible in order to reduce power consumption. In the method, a given DFG is converted into an equivalent DFG by the preprocessing method for associative and communicative operation clusters. In this presentation, vertices in each cluster are evaluated in order to derive an equivalent DFG which gives better result. The proposed method is expected to be effective by the derived codes for example program in term of usage of the bypass chain (hence, power consumption,) and elapsed time for code optimization.

## 1. はじめに

近年、プロセッサの微細化や動作の高速化などの技術進歩とともに、消費電力の増加が大きな問題となってきている。この消費電力には、演算器の処理による消費電力や、プロセッサとレジスタとの通信による消費電力などが含まれている。中でも、このプロセッサとレジスタとの通信電力は、プロセッサ全体における消費電力の中で比較的高い割合を占めていることが知られている**[1]**。これに対して、データフォワーディングを能動的に活用することでレジスタアクセス回数を抑える、バイパス構造を持ったアーキテクチャが提案されている**[2]**。

本来、フォワーディングとは、プロセッサのパイプライン制御でのデータハザードを回避するための機構であるが、ハザードの危険が無い状況でも積極的にデータフォワーディングをすることにより、レジスタからのデータの読み込みによる通信回数の削減が期待できる。バイパスアーキテクチャではフォワーディングを実現するバイパスを複数備えており、これらをソフトウェアで制御することにより、任意のサイクルだけ前のデータまでフォワーディングを可能としている。

また、このアーキテクチャをより有効に利用するため、実行コードからデータの依存関係を解析し、より多くのフォワーディングが利用できるようにデータフローグラフ（DFG）を利用して実行コードを再スケジューリングする手法が提案されている。この再スケジューリング手法においては DFG をコンパイラなどから得られた実行コードから抽出し、可換な演算命令の自由度を考慮して命令をクラスタリングした後、バイパスアーキテクチャにより適した等価 DFG に再構成し､それを利用してスケジューリングを行う。

しかし、このスケジューリング方法では、等価な DFG を再構築する際、クラスタリングされたノードにつながるノード群の差異を考慮に入れていない。また、スケジューリングに必要な時間がかなり長くなる場合もあり、改善の必要がある。

そこで本研究では、クラスタリングされたノードのデータ依存関係を考慮しつつ

† 東京工業大学情報工学科
Department of Computer Science, Tokyo Institute of Technology
†† 東京工業大学大学院総合理工学研究科物理情報システム専攻
Department of Information Processing, Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology

DFG の再構成を行い、また再構成を行う際に DFG を分割することで、スケジューリングにかかる時間を短縮する方法を提案する。本方法を使用することで、スケジューリングにかかる時間の短縮と、バイパスの使用効率の向上が見込める。

## 2. バイパスアーキテクチャ

ある命令で生成された演算結果は、その周辺の命令で使用される割合が高いため**[1]**、直前の命令で生成された演算結果をレジスタに書き戻すのではなく一時的な記憶に格納しておき、後の命令でそのデータが必要になったときに、一時的な記憶からすぐにデータを取り出せるバイパスのような機構があればレジスタアクセスを回避できる。レジスタではなくバイパスからデータを読み込むという点では、これはパイプライン制御におけるデータフォワーディングの機構と同じ原理である。そこで、バイパスを使用したデータフォワーディングを利用することで、レジスタアクセス数を抑えることができる。本研究では、フォワーディングを可能とするプロセッサアーキテクチャとして、バイパスアーキテクチャを対象とする。

バイパスアーキテクチャは、データ依存の関係があれば積極的にフォワーディングを活用し、レジスタアクセス回数を抑えることができるので、この結果としてプロセッサの消費電力の低減が期待できる。

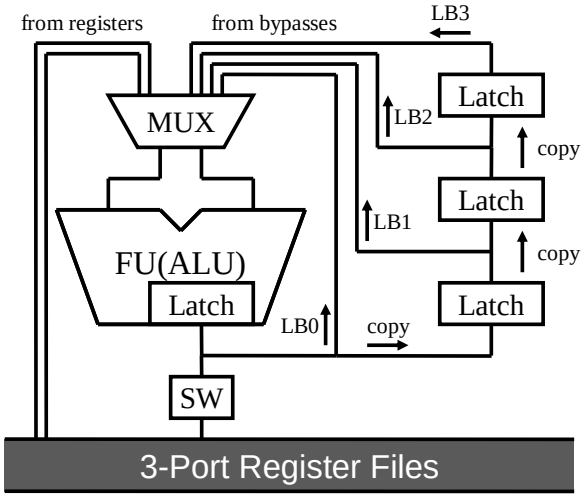



図 1　バイパスアーキテクチャ

### (1) 特徴

バイパスアーキテクチャの基本的な構成を図 1 に示す。このアーキテクチャは IF、ID、EX、WB の 4 ステージからなるパイプライン処理により動作する。図 1 のバイパスアーキテクチャでは 4 つのバイパスを有しており、演算器 FU の出力によるバイパス(LB0)と、3 つのローカルバイパス(LB1～3)により、3 サイクル前までラッチからのデータフォワーディングが可能である。ハザードの危険性をフォワーディングにより回避する以外にも、比較的電力消費の高いレジスタアクセスをフォワーディングにより回避できれば、レジスタにおける熱集を抑えることも期待ができる。

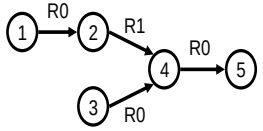
## 3. DFG を利用した再スケジューリング

バイパスアーキテクチャをより有効に使用するために、DFG を利用した再スケジューリング手法が提案されている[3]。以下、(1)~(3)にその概要を示す。

### (1) データ依存解析

アドレスコードの RISC プロセッサを仮定し、予めコンパイラによりアセンブリ言語に変換された実行コードに対し、データ依存関係を解析し、DFG を得る。図 2(a)の実行コードを解析すると、各命令をノードで表し、依存関係を有向エッジで示す図 2(b)の DFG が得られる。

| ID | inst | Dest | S1 | S2 |
|---|---|---|---|---|
| 1 | movi | R0, | 4 | |
| 2 | add | R1, | R0, | R1 |
| 3 | mov | R0, | R2 | |
| 4 | mult | R0, | R1, | R0 |
| 5 | str | R0, | R2 | |




a)アセンブリコード　　(b)DFG

図 2. 実行コードからのデータ依存の解析

### (2) DFG における並列度の解析

(1)により生成された各 DFG に対して、次に定義する深さ $D$ と並列度 $P_D$ を求める。

$D$ …各 DFG の頂点に位置するノードをルートノードと呼び、ルートノードからの木の深さを $D$ とする。また、ルートノードの深さは 0 とする。

$P_D$…各深さ $D$ において同じ深さに並ぶノードの数を並列度 $P_D$ とする。

そして、DFG における最大の深さを $D_{max}$ とし、全ての $P_D$ が次の条件式を満たすとき、その DFG 内で(3) による最適化を行う。

**条件式**：$P_D$≦バイパスの本数 $(0 \leqq D \leqq D_{max})$

ある深さで並列度が高く上式を満たさないような DFG では、保持できるデータ数の限界を超えてしまい、DFG 内でバイパスの使用ができない命令が存在することになる。その場合はグラフ内で 2 つのエッジを入力としているノード(2 入力ノードと呼ぶ)に注目し、グラフ内の 2 入力ノードに向かうエッジの中で、どちらか 1 方のエッジを切断する。そして分割された全てのグラフに対し先ほどの条件式を満たすかどうか判定する。このエッジの切断は、全ての分割されたグラフが条件式を満たすまで続ける。切断するエッジの数が最小になるよう切断箇所を全試行しているので、切断候補の多いプログラムは実行に長い時間がかかる。

**(３) DFG 内における最適化**

(2)で分割された全ての DFG に対して、幅方向の ALAP アルゴリズムによる命令のスケジューリングを施す。これにより、バイパスの使用効率の向上が見込める。そして最適化の済んだ DFG のスケジューリング結果を、切断する前の実行順序・データの整合性などに注意して全ての DFG を結合していき、最終的に 1 つの実行コードにする。

## 4. 命令の自由度を考慮したＤＦＧの再構成

3 節の再スケジューリング手法では、DFG は実行コードから生成されている。しかしながら、命令に内在する実行順番の自由度を考慮すると、等価な DFG が多く存在し、これらの中には並列度の低い DFG も存在する可能性がある。3 節の方法の前処理として、演算命令のクラスタリングを利用して自由度のある演算命令の依存関係を変更することで、より並列度の低い DFG を得る。

**(１) 演算命令の自由度**

演算命令の中には、add 命令や mult 命令のような、データ依存関係をある程度変更することができる命令が存在する。例えば、X=a+b+c+d という計算は、X=(a+b)+(c+d)と計算しても X=a+(b+(c+d))と計算しても正しいが、それぞれの計算を DFG で表現すると違ったものになる。一般的に、可換則と交換則が成り立つ演算においては、命令の実行順番にある程度の自由度が存在するので、DFG の自由度のある演算命令の部分をクラスタリングした後、DFG を再構成することで、実行コードからより並列度の低い等価な DFG を得る。

**(２) DFG の再構成**

DFG の再構成は、3 節の再スケジューリング方法のデータ依存解析のステップで行う。例題として図 3(a)～図 3(f)を示す。最初に、実行コードから DFG を生成する(図 3(a))。DFG の各ノードを順に調べていき、自由度のある演算命令であった場合、その子ノードをクラスタとしてまとめる。子ノードから解析された命令へのエッジを切断し、クラスタから解析された命令にエッジを作る(図 3(b))。その後、クラスタ内のノードを調べていく。クラスタの親ノードと調べたノードが同じ命令である場合、調べたノードの子ノードをクラスタに入れ、子ノードから調べたノードへのエッジを切断する。その後、調べたノードを取り除く。これをクラスタの中にまだ調べていないノードがなくなるまで繰り返す(図 3(c))。その後、クラスタリングしたグラフを、2 分木のＤＦＧに再構成する。クラスタからノードを 1 つ選びクラスタから出す。クラスタの親と同じ命令を表すノードを追加する。クラスタの親ノードからその親ノードへのエッジを切断し、それぞれのノードの間に図 3(d)と図 3(e)のようにエッジを作る。これをクラスタの中にノードがなくなるまで繰り返し、最終的に再構成された DFG を得る(図 3(f))。

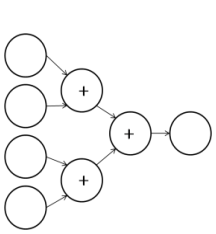


図 3(a)

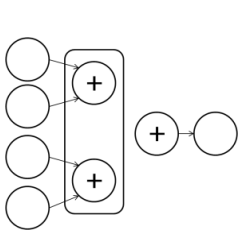


図 3(b)

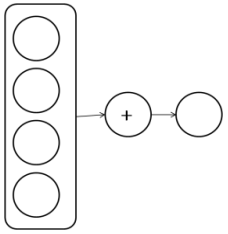


図 3(c)

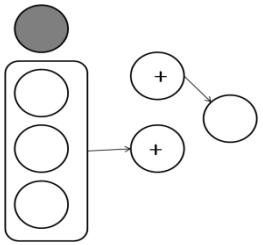


図 3(d)

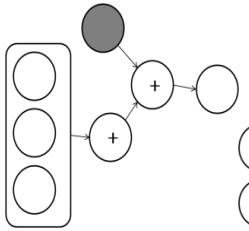


図 3(e)

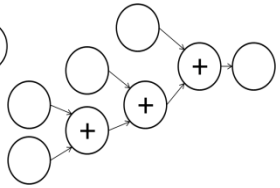


図 3(f)

## 5. 提案方法

クラスタからノードを選ぶ時、従来はすべてのノードから 1 つを無作為に抽出していたが、クラスタ内のノードにつながるノード群の差異を考慮してノードを選び出し

DFG を再構成することで、クラスタ内のノードをランダムに選ぶ場合よりラッチを有効に利用できる可能性がある。また、ノードを選ぶ際に DFG の分割を並行して行うことで、切断箇所の探索する際に、探索空間の削減が期待できる。

例題として、図 4(a)のようなクラスタを再構成していく過程を示す。

最初にクラスタからノードを選ぶ際には、最もノード数の多い DFG を持っているノードを選ぶ(図 4(b))。次のノードからは、3 節(2)で示した条件式($P_D$≦バイパスの本数)を満たす範囲で最大のノードを選んでいく(図 4(c))。クラスタの中に条件を満たすノードが無い場合は、クラスタの親ノードの出力エッジを切断し、再び最初に戻ってノードを選んでいく(図 4(d))。これを繰り返し、DFG を再構成する。

このアルゴリズムを使用して DFG を再構成することで、ノードをランダムに選択する場合と比べて、同等以上の DFG が得られる可能性が高い。また、あらかじめ DFG の分割を行うことでこの後に行う再スケジューリングの際、切断するエッジの本数が少なくなるので、プログラムの実行時間が短くなることが期待できる。ただし、このアルゴリズムを使用すると、DFG の分割が最善のものではなくなる場合もあるので、切断箇所が増え、レジスタアクセスが増加する場合もある。

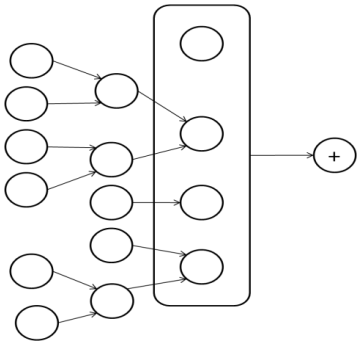



図 4(a)

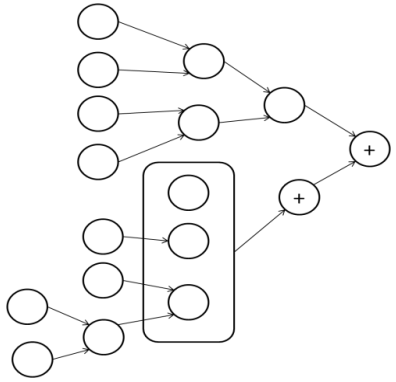



図 4(b)

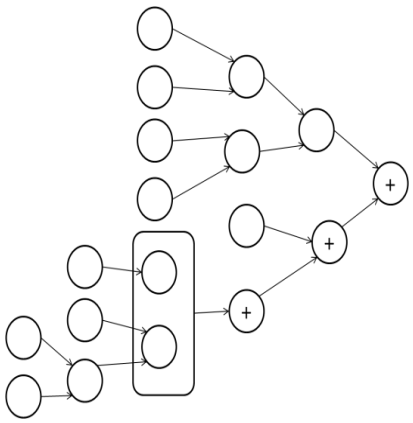



図 4(c)

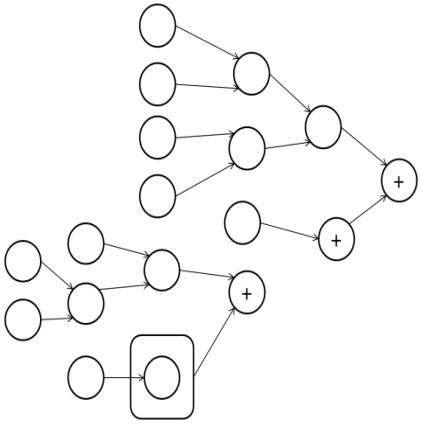



図 4(d)

## 6. 実験・評価

4 節で述べた提案方法の有効性を評価するため、サブバンドフィルタープログラムを例題として、計算機実験により提案方法を適応した再スケジューリングを行った場合と、従来の再スケジューリングを行った場合とでの比較を行った。まず、実行プログラムのアセンブリコードを得るために、サイクルベースアーキテクチャシミュレーション環境 MICS**[5]**に含まれる簡易 RISC プロセッサ用コンパイラを用いて、簡易な RISC プロセッサ「SimpleProcessor32」向けのアセンブリコードに変換する。次に、このアセンブリコードに対し、従来の再スケジューリング手法を適応した場合と、提案手法を適応した再スケジューリング手法を適法した場合とで、それぞれバイパスアーキテクチャ上で実行したときのバイパスの使用回数を比較する。想定するバイパスアーキテクチャは、4 ステージのパイプライン処理を仮定し、EX ステージにフォワーディング可能な 3 つのバイパスを備えているシングルコアプロセッサとする。このアーキテクチャ上でアセンブリコードを実行させたときの、コード全体におけるバイパスの使用回数を算出する。

表 1. バイパスの使用効率

| 方式 | ランダム | 提案方法 |
|---|---|---|
| レジスタアクセス | 283 | 283 |
| バイパス使用回数 | 202 | 205 |

表 1 に結果を示す。提案方法によってノードを選んだ場合、ランダムにノードを選んだ時よりバイパスの使用回数が若干多い。

また、提案方法を使用した場合、DFG の再構成の段階であらかじめ DFG が分割されているため、クラスタリング後の切断箇所の探索を実行した場合、１ヶ所切断すれば DFG が条件を満たす場合が多く、探索空間があまり広くならないが、従来の方法では２ヶ所以上の切断が必要な場合もあり、再スケジューリングの実行時間が長くなる傾向がある。

## 7. おわりに

本研究では、クラスタリングされたノードの持つデータ依存関係を考慮しつつ DFG を再構成する方法と、DFG の再構成中に DFG を分割する方法を提案した。また、提案した方法が、ラッチの使用回数の増加と、スケジューリングの時間短縮に有効であることを示した。

今後の課題として、基本ブロックをまたぐようなデータ依存を考慮して、最適化を行う方法についても考慮する必要がある。

そして、バイパスアーキテクチャをマルチコア化したアーキテクチャも発表されており、再スケジューリング手法をマルチコア向けに拡張することを検討していく予定である。

## 参考文献

**[1]** Manoj Gupta, et al. “Energy Based Design Space Exploration of Multiprocessor VLIW Architectures,” Proc 10th European Conf, vol. 00, pp. 207-310, Aug. 2007.

**[2]** 田他，“バイパス構造をバス接続したマルチプロセッサによる消費電力低減の検討,” 2007 ソサイエティ大会投稿論文

**[3]** 庄司他, “データフローグラフに基づくバイパスアーキテクチャ向けコード最適化方法” 第 22 回 回路とシステム軽井沢ワークショップ C2-1-1

**[4]** EEMBC “The Embedded Microprocessor Consortium” http://www.eembc.org/home.php

**[5]** 三好他，“MICS:システム設計のためのフレキシビリティの高いシミュレーション環境,” 情報処理学会論文誌(IPSJ Journal), Vol.49 No.10, pp. 3482-3492, 2008

**謝辞** 本研究を進めるにあたり、多大なご指導、ご協力を頂いた杉野暢彦准教授には深く感謝の意を表すと共に、厚くお礼申し上げます。また、研究を進める上で数々の貴重な意見を頂いた同研究室の皆様にもこの場をお借りしまして、お礼申し上げます。

**著者紹介**

**鎌田裕基**
東京工業大学情報工学科
**庄司俊寛**
東京工業大学大学院総合理工学研究科物理情報システム専攻
**田金**
東京工業大学大学院総合理工学研究科物理情報システム専攻
**杉野暢彦**
東京工業大学大学院総合理工学研究科物理情報システム専攻