Vol.2016-ARC-221 No.11
2016/8/8

アウトオブオーダ命令実行の依存グラフ表現に関する考察

谷本 輝夫[1,a)]　佐々木 広[2,b)]　小野 貴継[1,c)]　井上 弘士[1,d)]

概要：
アウトオブオーダプロセッサでは複数の命令が順序を入れ替えながら同時に実行されるため，実行時間に影響する処理を特定することは容易ではない．これを実現する方法の一つとして依存グラフを用いた動的な命令列のモデル化が提案されている．しかしながら，既存手法は単純なプロセッサ構造を前提としており，異なるマイクロアーキテクチャの特徴や詳細度を持つプロセッサを前提とした場合に正しくモデル化できない．本研究ではモデル化できていないマイクロアーキテクチャの特徴を明らかにし，それらを表現可能な依存モデルを構築した．適用実験として，モデルから得られる実行時間をプロセッサシミュレータを用いて評価した結果，平均相対誤差が 22%から 2.1%となり精度向上が得られた．

1. はじめに

デナードスケーリング [5] は 65nm 以下の半導体プロセスでは成り立たず [6]，チップに搭載されるトランジスタを同時に使用することができない．これによりマルチコアプロセッサを用いた並列処理による性能向上はその限界が指摘されており [7]，アーキテクチャおよびソフトウェアの最適化によるシングルスレッドの実行効率化の重要性が増している．そのためには実行時間を決定する処理の特定が必要不可欠である．しかしながら，アウトオブオーダプロセッサでは複数命令が実行順を入れ替えながら同時実行されるため，これを特定することは困難である．

そこで，マイクロアーキテクチャが与えられた際のプログラムの動的な命令列を依存グラフとして表現（本稿ではこれを**依存モデル**と呼ぶ）するモデル化手法が提案された [10][11][12][15][20]．この依存モデルにより実行時間を律速する命令を動的に特定することができ，投機実行 [11] や，適応的スケジューリング [11][18] などに用いることができる．また，依存モデルにより実行時間を推定することで設計空間探索 [10][15] にも有用となる．

動的な命令列の依存グラフは，抽象化されたマイクロアーキテクチャにおけるプログラムの動的な振る舞いを表現したものとみなすことができる．したがって，依存モデルを構築する際に考慮すべき依存関係は，プロセッサのマイクロアーキテクチャとその抽象化度合いにより異なる．既存の依存モデル [12] は Simplescalar[3] のマイクロアーキテクチャを前提としている．そのため，近年の複雑な構造を持つ高性能マイクロプロセッサアーキテクチャ（たとえば Intel 社の Haswell などをモデル化可能な Gem5 simulator[2] の O3CPU）を前提とした場合には，依存モデルで表現できない実行の振る舞いが生じる．本稿では上記の原因を分析した．その結果以下の点をモデル化できないことがわかった．

- 動的な分岐予測ミスペナルティ（詳細度の違い）
- シニアストアキュー [16] フル時の振る舞い（マイクロアーキテクチャの違い）
- 命令キャッシュ (icache) ミス，命令 TLB (itlb) ミス以外のフロントエンドでの遅延（マイクロアーキテクチャの違い）

そこで，既存の依存モデルを拡張した．そして，適用実験として依存モデルから得られる実行時間の相対誤差をプロセッサシミュレータを用いて評価した結果，平均相対誤差が 22%から 2.1%となり精度向上が得られた．このことから提案モデルにより命令列の振る舞いをより正確に表現できていると言える．

2. アウトオブオーダ命令実行の依存グラフ表現

2.1 依存グラフ表現とクリティカルパス解析

アウトオブオーダプロセッサにおける命令実行を依存グラフとして表現する際には，コミットされた動的な命令ごとかつプロセッサ内における状態ごとにノードを作成し，

[1] 九州大学
Kyushu University
[2] コロンビア大学
Columbia University
[a)] teruo.tanimoto@cpc.ait.kyushu-u.ac.jp
[b)] sasaki@cs.columbia.edu
[c)] takatsugu.ono@cpc.ait.kyushu-u.ac.jp
[d)] inoue@ait.kyushu-u.ac.jp

表 1 FIELDS モデルの制約一覧

名前	制約	エッジの定義
DR	オペランド待ち	$D_i \rightarrow R_i$
RE	オペランド待ち後実行	$R_i \rightarrow E_i$
EP	演算の完了	$E_i \rightarrow P_i$
PC	演算完了後コミット	$P_i \rightarrow C_i$
DD	インオーダディスパッチ	$D_{i-1} \rightarrow D_i$
CC	インオーダコミット	$C_{i-1} \rightarrow C_i$
FBW	フェッチバンド幅	$D_{i-fbw} \rightarrow D_i$ $fbw :=$ フェッチバンド幅
CBW	コミットバンド幅	$C_{i-cbw} \rightarrow C_i$ $cbw :=$ コミットバンド幅
PD	制御依存	$P_{i-1} \rightarrow D_i$ 命令 $i-1$ が予測ミス分岐命令時追加
PR	データ依存	$P_j \rightarrow R_i$ 命令 j が命令 i のオペランド生成時追加
CD	リオーダバッファ エントリ数	$C_{i-w} \rightarrow D_i$ $w :=$ リオーダバッファエントリ数
PP	キャッシュラインの共有	$P_j \rightarrow P_i$ 命令 j が命令 i がロードするブロックの キャッシュミスを起こす時追加

ノードの添字は動的な命令のシーケンス番号

ノード間の順序関係を有向エッジで結ぶ．順序関係は，ある処理が終わった後でないと次の処理が行えないことを表す．順序関係が循環することはないので，このグラフは閉路を持たない有向グラフとなる．エッジは依存を解決するために要するサイクル数を重みとしてもつ．

このグラフにおいて，先頭命令の初期状態のノードから末尾命令の最終状態までの最長経路が依存モデルにおけるクリティカルパスである．仮に依存モデルがプロセッサ内のすべての依存を表現できているとすると，この実行パスに要するサイクル数はプログラムの実行サイクル数に一致する．

2.2 FIELDS モデル

表 1 に Fields らによる依存モデル [12] の制約一覧を，**表 2** に FIELDS モデルにおけるエッジの重みの定義を示す．本稿ではこの依存モデルを FIELDS モデルと呼ぶ．この依存モデルは各動的命令の実行において以下の状態をノードとして定義している．

- ディスパッチ（D）：フロントエンドからバックエンドへディスパッチされた後の状態
- レディ（R）：ディスパッチ後ソースオペランドが利用可能になった状態
- 実行（E）：命令の演算が開始された状態
- 完了（P）：命令の演算が完了した状態
- コミット（C）：命令がコミットされた状態

n 個の動的命令の実行をモデル化の対象とする際には $5n$ 個のノードを要する．

ノード間の依存は表 1 に示した 12 種類の依存関係により表現する．DR，RE，EP，PC によりそれぞれの命令実行に関する状態遷移の順序関係を，DD ならびに CC により，異なる 2 つの命令間におけるインオーダディスパッチならびにインオーダコミットをモデル化している．また，FBW，CBW はスーパスカラの幅分前の命令から重み 1 のエッジを引くことにより 1 サイクルにディスパッチ／コミット可能な命令数の制約を表現している．PD は制御依存を，PR はデータ依存を表す．CD は有限のリオーダバッファエントリ数の制約をモデル化する．リオーダバッファはリネーム時にエントリが作成され，コミット時に開放される．リネーム・コミットはインオーダに実行されるため，リオーダバッファエントリ数分前の命令がコミットされた後でしか命令はリネームできない．PP は同じキャッシュブロックに対するロード命令間で，先行するロード命令がキャッシュミスを起こし下位のメモリ階層からデータをキャッシュへ運んだ場合に生じる依存関係である．後続のロード命令は先行するロード命令の完了を待つ必要がある．

表 2 FIELDS モデルのエッジの重みの定義

名前	重み
DR	固定値 (パイプラインの遅延)
RE	演算器の競合
EP	演算器の遅延
PC	固定値 (パイプラインの遅延)
DD	icache ミス，itlb ミス
CC	store バンド幅競合
FBW	固定値 (1 サイクル)
CBW	固定値 (1 サイクル)
PD	固定値 (分岐ミスペナルティ)
PR	固定値 (0 サイクル)
CD	固定値 (0 サイクル)
PP	固定値 (0 サイクル)

エッジの重みは静的に与えるものと，動的に値を得る必要があるものがある．静的に値を与えられるエッジはプロセッサの設計パラメタをより直接的にモデルに反映することができ，このようなパラメタに関してはモデルを利用して実行時間に対する影響を短時間で計算することができる．

図 1 に FIELDS モデルによる命令列の表現の例を示す．この例では，フェッチバンド幅およびコミットバンド幅が 2（$fbw = cbw = 2$），リオーダバッファのエントリ数が 4，DR および PC の重みが 1，PD の重みが 6 である場合を表している．実線で示したエッジはそれぞれのノードに入るエッジの中でもっとも遅く解決するエッジを示し，中でもクリティカルパスであるエッジを太線で示している．それ以外のエッジは点線で示している．i_4 は分岐予測ミスした分岐命令であり，i_4 から i_5 へ PD エッジが引かれている．この例では D_0 から C_8 までのクリティカルパスのサイクル数は 28 である．

2.3 FIELDS モデルの表現力

サイクル精度で実行可能なプロセッサシミュレータである Gem5 に FIELDS モデルを適用し SPEC CPU2006 ベン

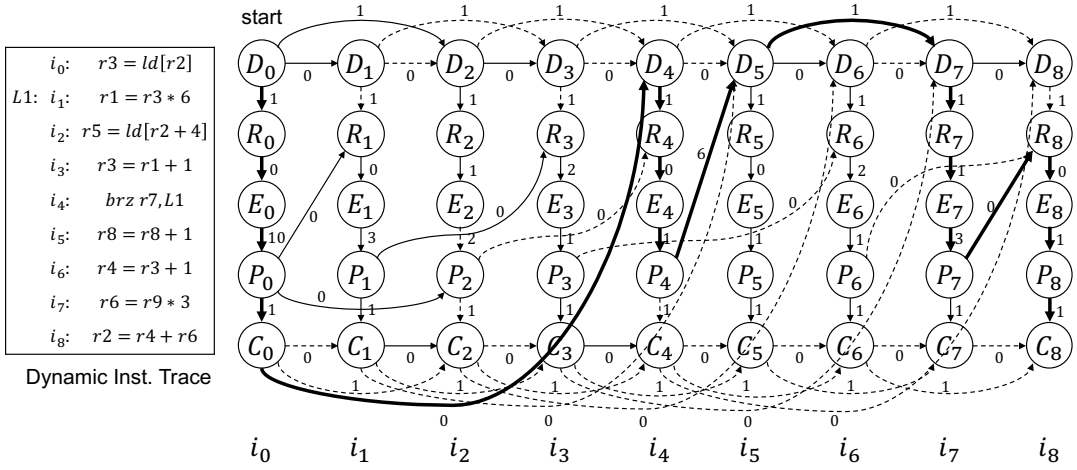

図 1 FIELDS モデルによる命令列のモデル化の例

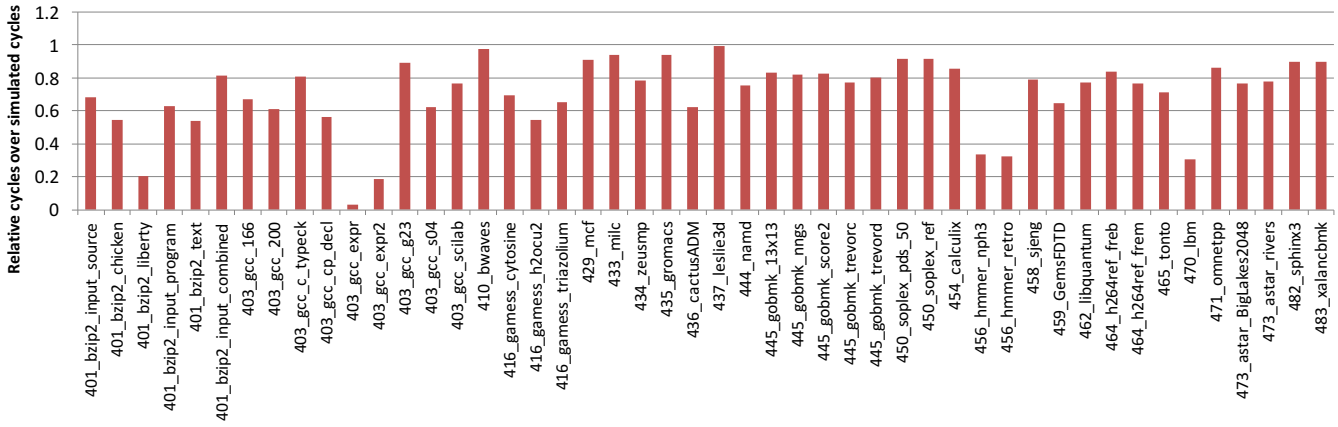

図 2 SPEC CPU2006 における FIELDS モデルのクリティカルパスのサイクル数の割合

チマークを実行して，クリティカルパスのサイクル数とプログラムの実行サイクル数を比較した結果を**図 2** に示す．シミュレータのパラメータは 4.2 節のパラメタと同様である．Gem5 ではディスパッチと同じサイクルに命令実行を開始できる実装のため，*DR* エッジの重みは 0 サイクル，*PC* エッジの重みは 4 サイクルとした．また，命令セットとして x86 を用いたため，マイクロオペレーションを考慮したモデルの変更を行っている．詳しくは 3.2 章で述べる．縦軸は各プログラムの実行サイクル数に対するクリティカルパスのサイクル数の相対値であり，1 に近いほどモデルがプログラム実行をよく表現できていることを示す．`leslie3d` は 0.99 で最も数値が高く，`gcc` で `expr` を入力とした時が 0.03 でもっとも数値が低い．平均相対誤差は 22%であった．命令実行をよく表現できているプログラムがある一方で，ほとんど表せないプログラムも存在する．

クリティカルパスのサイクル数と実行サイクル数の差は大きく以下の 2 つの理由により生じる．一つはモデルが表現する制約とプロセッサのマイクロアーキテクチャの不一致であり，もう一つはエッジに与える重みが適切でないことである．既存の依存モデルが表現できていない点を調査した結果，以下が原因であると分かった．

- 固定値の分岐予測ミスペナルティでは実行時の動的なペナルティを表現できない（不適切な重みの設定）
- ストアキューはライトバックバッファと一体化したシニアストアキューとして実装されているため，ストアバンド幅での競合を *CC* で表現できない（マイクロアーキテクチャの不一致）
- icache ミス，itlb ミス以外のフロントエンドでの遅延をモデルに反映できない（不適切な重みの設定）

これらの FIELDS モデルの課題を解決するためのモデルの変更を考え，それらを適用した際のモデルのクリティカルパスのサイクル数への影響を調査する．

3. Gem5 のアウトオブオーダプロセッサ（x86 命令セット）を表現可能な依存モデル

3.1 提案モデルの概要

提案モデルは FIELDS モデルをベースとして，Gem5 におけるアウトオブオーダプロセッサの詳細なプロセッサモデルである O3CPU を表現するための追加・変更を行う．**表 3** に FIELDS モデルに対する追加および変更点を示す．

表 3 FIELDS モデルに対する追加・変更点

	名前	制約	エッジの定義
追加	UOP	マクロオペレーション	$P_j \rightarrow C_i$ 同一のマクロオペレーションの uOP (j) から先頭 uOP (i) へ
変更	PD	制御依存	$P_{i-1} \rightarrow D_i$ スクアッシュ遅延 + フロントエンド遅延 + icache ミス，itlb ミス遅延
追加	CS	コミット後ライトバック	$C_i \rightarrow S_i$
追加	SS	インオーダライトバック	$S_j \rightarrow S_i$ j,i はストア命令
追加	SD	ストアキューエントリ数	$S_i \rightarrow D_{i-s}$ $s :=$ ストアキューエントリ数
変更	DD	インオーダディスパッチ	$D_{i-1} \rightarrow D_i$ icache ミス，itlb ミス + フロントエンドブロック遅延

分岐予測ミスペナルティの動的な変動に対応するために PD エッジの重みの与え方を固定値から動的な値へ変更し，シニアストアキューにおける制約を表現するために，ストア命令にストア状態 (S) を追加する．また，フロントエンドにおける icache ミス，itlb ミス以外の遅延を依存モデルに反映するため DD エッジの重みの与え方を変更する．

本稿では実行プラットフォームとして Gem5 を利用し，その上での表現するための依存モデルを構構構築したが，他のプラットフォームでも同様のアーキテクチャの特徴を持つ場合には同じモデルへの変更が利用できる．また，別のアーキテクチャの特徴を持つ場合にも，本稿と同様のアプローチにより依存モデルの構築が可能である．

3.2 マイクロオペレーションへの対応

本研究では x86 命令セットを対象とする．x86 命令セットでは，命令（以下マクロオペレーション）がプロセッサ内のデコーダにより複数のマイクロオペーレーション（以下 uOP）に分割され，uOP 単位でスケジューリングや実行される．Gem5 は x86 命令セットをサポートしており，内部で uOP へ変換し実行する．そのため，依存グラフを作成する際にはマイクロオペレーションを 1 命令として扱う．この際，コミットはマクロオペレーション単位で行う必要があるため，マクロオペレーション中の最後の uOP がコミット可能となった時点で，同じマクロオペレーションに含まれる uOP をコミットする．この制約を表現するために Lee らの依存モデル [15] を参考に UOP 依存を追加する．UOP は処理の順序関係のみを表現すればよく，重みは 0 サイクルとする．

3.3 分岐予測ミスペナルティの動的重み付け

FIELDS モデルでは，分岐ミスペナルティを PD エッジにより表現している．この重みは分岐ミスを検出してから次の正しいパスの命令をディスパッチするまでのレイテンシを意味する．FIELDS モデルの定義では重みに静的な値を用いている．これは，一般に分岐予測ミスペナルティは実行ステージで分岐予測ミスを検出した後，正しいパスの命令をフェッチし実行ステージまで進めるのに要するレイテンシ，つまりフロントエンドのレイテンシであると考えられるためである．しかしながら，実際には分岐予測ミスが生じた際のペナルティは以下の 2 つの理由で動的に変動する [8].

- 新しい命令をフェッチする前に誤ったパスの命令を無効化（スクアッシュ）する必要がある
- 新しい命令をフェッチする際に icache ミス，itlb ミスが起きるとその解消に要する遅延が分岐ミスペナルティに追加で生じる

前者については，マイクロアーキテクチャの実装方式により実行時間への影響が異なるが，Gem5 においては，リオーダバッファに含まれる誤ったパスの命令，つまり予測ミスした分岐命令よりも後に追加された命令を一旦全てスクアッシュしてから新しい命令のフェッチを再開する方式を採っている．1 サイクルにスクアッシュできる命令数には限りがあるため，リオーダバッファのエントリ数が大きいとスクアッシュに数サイクルから数十サイクルかかることがある．

また，後者については icache ミス，itlb ミスの遅延は DD エッジにより依存モデルに反映されるが，分岐予測ミスが起きた場合には PD エッジがクリティカルパスとなり，icache ミス，itlb ミスの遅延がクリティカルパスに含まれなくなってしまう．そのため，本稿では PD エッジに icache ミス，itlb ミスの遅延を加算することでこの問題を解決した．

3.4 シニアストアキューへの対応

シニアストアキューはストアキューとライトバックバッファを一体化したキューである．Alpha 21264 microprocessor[4] や Intel の P6 microarchitecture[19] で採用されている他，プロセッサシミュレータでは Gem5 以外に Zesto[16] で採用されている．キュー内にはストア命令がプログラムオーダーに格納されており，ヘッドとテイル以外に最も最近コミットされた命令を示すポインタが存在する．ヘッドからコミットポインタまでの命令はコミット済みでライトバック待ちのストア命令であり，コミットポインタの次の命令からテイルの一つ前までの命令はコミット前のストア命令である．

ストアキューとライトバックバッファが分離されている場合は，ストア命令をコミットする際にストアキューからライトバックバッファへストアアドレスとストアデータを移動する．もしもライトバックバッファがフルの場合はストア命令をコミットできずにプロセッサのコミットが停止する．しかしながら，シニアストアキューでは，キュー内でコミットポインタを進めることによりストア命令のコミットができるため，ストアによるプロセッサのコミット

停止を無くすことができる．また，ストアキューとライトバックバッファのエントリを共有できるためエントリ数をより有効に利用できる．

通常のストアキューと同様，シニアストアキューはストア命令をディスパッチする時にエントリを作成する．そのため，ライトバック待ちのストア命令がキュー内にたまりフルになると新しい命令をディスパッチできない．FIELDS モデルは通常のストアキューを想定しているため，シニアストアキューを持つプロセッサではストアバンド幅での競合を表現することができない．そこで，ストア命令にストア状態 (S) を追加する．これはストアデータがライトバックされた状態を表す．CS エッジはコミット後ライトバックの順序関係を，SS エッジはライトバックがインオーダに実行される順序関係を表し，SD エッジは有限のシニアストアキューエントリ数の制約を表している．CS エッジの重みはコミットからライトバックまでのサイクル数，SS および SD エッジの重みは 0 サイクルとする．

3.5 フロントエンドにおける遅延の動的重み付け

FIELDS モデルではフロントエンドにおける遅延は icache ミスと itlb ミスのみが考慮されているが，実際には他にも遅延が生じる場合がある．命令キューやロードキュー，ストアキューのいずれかがフルになるなどによりディスパッチが停止すると，リネーム，デコード，フェッチの各ステージが順次ブロックされていく．Gem5 の実装ではこれらのステージはディスパッチが再開されてもすぐには処理を再開できず，直後のステージにおいて中断されていた処理が終わった後でなければ再開できない．そのため，各ステージの処理に要するサイクル数（ここではリネーム，デコード，フェッチの各ステージがパイプライン化されていることを想定している）のバブルが入ってしまう．

この問題に対処するため，リネーム，デコード，フェッチの各ステージがブロックされた場合，処理を中断された先頭の命令とその 1 つ前の命令間の DD エッジに対し，直後のステージの処理に要するサイクル数をペナルティとして加算するものとした．

4. 提案モデルの評価

4.1 評価方法

提案モデルを評価するため，分岐予測ミスペナルティの動的重み付け，シニアストアキューへの対応，フロントエンドにおける遅延の動的重み付けのそれぞれおよび全てを適用した場合を評価した．マイクロオペレーションへの対応は命令セットの変更に起因するため，比較対象である FIELDS モデルに対しても適用した．本稿では依存モデルの評価指標として，プログラムの実行サイクル数に対するクリティカルパスのサイクル数の割合を用いた．また，依存モデルの変更前と変更後のクリティカルパスに占めるエッジ種ごとの内訳を比較し，クリティカルパスの変化の有無を調べた．

表 4 評価に用いたシミュレータのパラメタ

Number of cores	1
CPU type	O3CPU
Frequency	2 GHz
ROB / Issue Queue / LQ / SQ entries	192 / 60 / 72 / 48
Pipeline width	8, Issue width のみ 6
L1 icache / dcache size	32KB / 32KB
L1 dcache access latency	4 cycle
L2 cache size	256KB
L2 cache access latency	12 cycle
DRAM	DDR3-1600 11-11-11

4.2 評価環境

評価には Gem5 の SE (Syscall Emulation) モードを用い，SPEC CPU2006 ベンチマーク [13] から実行できた 25 個のプログラム（入力データの違いも考慮すると 48 種類）を使用した．**表 4** に評価に使用したシミュレータのパラメタを示す．

4.3 クリティカルパスのサイクル数の評価

図 3 に本稿における 3 つの制約の追加・変更の影響を示す．FIELDS は FIELDS モデルの結果であり，PD+は分岐予測ミスペナルティの動的重み付け，S+はシニアストアキューへの対応，DD+はフロントエンドにおける遅延の動的重み付けを，ALL は 3 つ全てを同時に適用した場合の結果である．平均相対誤差は FIELDS は 22%，PD+は 15%，S+は 8.9%，DD+は 15%であり，ALL は 2.1%である．

PD+では aster において rivers を入力とした 473_aster_rivers で 19%，sjeng (458_sjeng) で 14%改善している．また，S+では多くのプログラムで大きな改善が見られるが，中でも gcc で expr を入力とした 403_gcc_expr で 96%，expr2 を入力とした 403_gcc_expr2 で 79%の改善が見られる．S+はストアキューフルイベントが起きた時のみにクリティカルパスに影響するため，そのようなイベントが起きない bwaves (410_bwaves) や gromacs (435_gromacs) では FIELDS と全く同じ結果である．DD+では，gcc で 166 を入力とした 403_gcc_166 で 24%，cp_decl を入力とした 403_gcc_decl で 20%の誤差減少が見られる．

ALL では 473_aster_rivers など値が 1 を超えているプログラムが見られる．クリティカルパスのサイクル数が実際の実行サイクル数を超える原因として，モデル中に偽の依存の存在および重みを実際よりも大きく与えていることが考えられる．これらの解決は今後の課題である．

4.4 クリティカルパスの変化の評価

図 4 に FIELDS モデルと提案モデル (ALL) から得られ

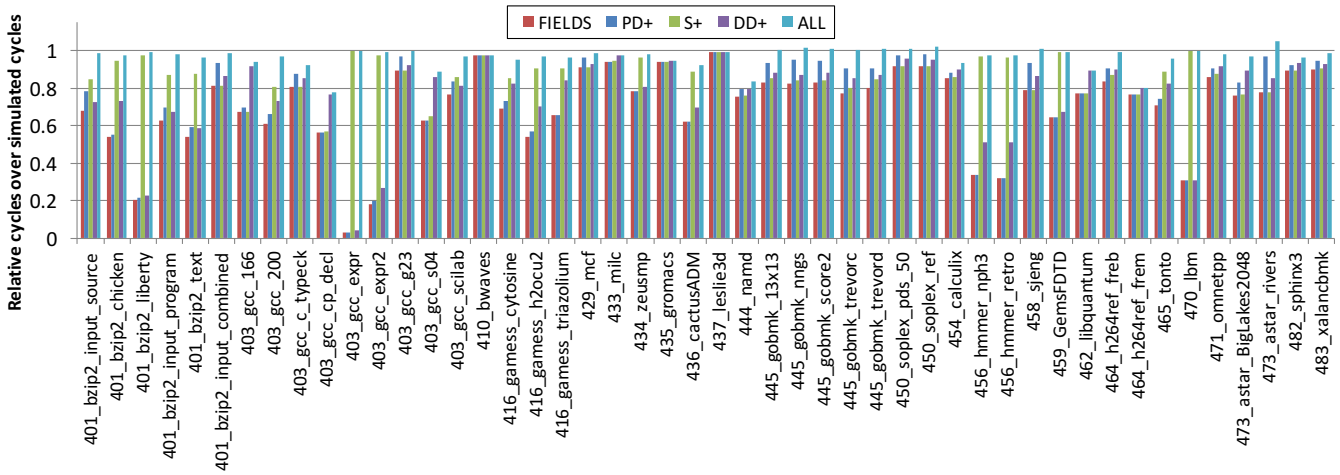

図 3 モデルの変更のクリティカルパスのサイクル数への影響

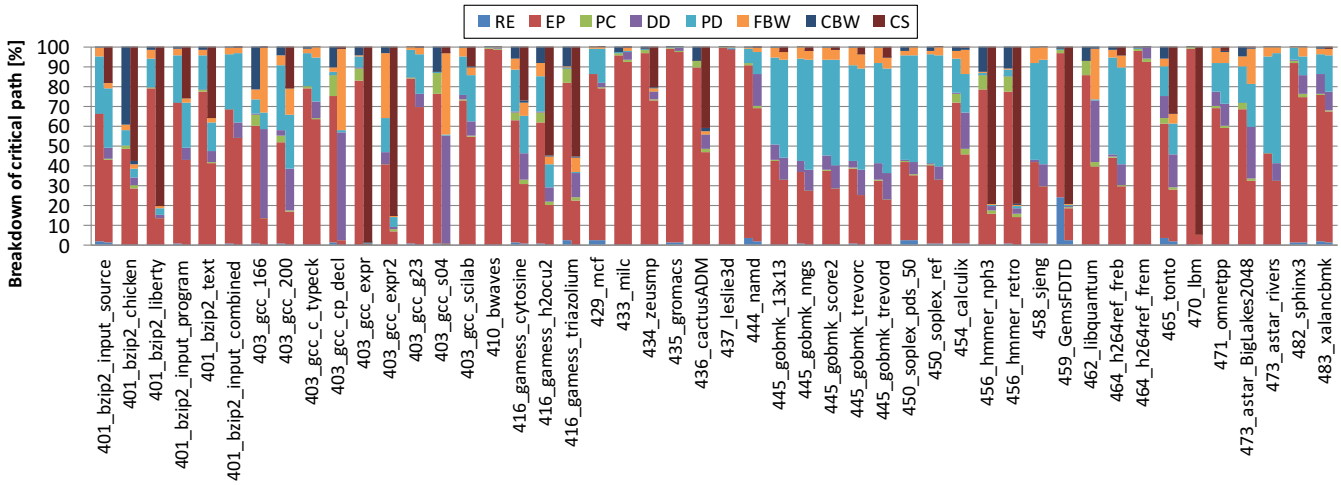

図 4 モデルの変更によるクリティカルパスの変化．各プログラムについて左側が FIELDS モデル，右側が提案モデル (ALL)

るクリティカルパスに占めるエッジ種ごとの内訳を示す．凡例には非ゼロの重みを持つエッジのみ示している．`403_gcc_expr` や `403_gcc_expr2` といった，FIELDS と ALL の差が大きいプログラムでは内訳も大きく変化している．S+により大きく改善しているプログラムに関しては *CS* エッジが大きな割合を占めており，モデルの変更によりクリティカルパスが大きく変化していることがわかる．

`calculix`（`454_calculix`）は図 3 では FIELDS と ALL で大きな違いはないが，クリティカルパスの内訳をみると *EP* エッジの割合が減少し，*DD* エッジの割合が増加している．このことはモデルから得られるクリティカルパスが変化していることを示している．クリティカルパスに基づいた最適化を行う際にはクリティカルパスを誤って推定すると悪影響を及ぼしてしまう可能性があり，依存モデルにより適切にクリティカルパスを判定可能になることは極めて重要である．

5. 関連研究

5.1 プログラムの振る舞いの解析方法

プログラムをアウトオブオーダプロセッサで実行した際の振る舞いの解析方法は 2 種類に大別できる．一つはキャッシュミスや分岐予測ミスといったペナルティイベントに関しイベントの 1 回あたりのペナルティサイクル数とイベントの発生回数を用いて実行時間を定式化する方法である．もう一つは，依存グラフにより動的な命令列を表現する方法である．本章ではそれらの関連研究について述べる．

5.2 実行時間の定式化

キャッシュミスや分岐予測ミスといったペナルティイベントの回数と，イベント 1 回当たりのペナルティサイクル数を乗じることでイベントのサイクル数を計算し，それらを足し合わせることで実行時間を定式化する方法が知られている [1][14]．しかしながら，アウトオブオーダプロセッ

サでは複数命令が実行順を入れ替えながら同時実行されるため，複数のペナルティイベントのオーバラップを考慮する必要がある．また，イベントによるペナルティサイクル数が一定でない場合には上記のような定式化ができない．これらを考慮することで，実行時間に占めるペナルティイベントの内訳を得る手法が提案されている [9]．これらの方法はモデル式の各項の数値を得ることが容易な点で優れている．汎用プロセッサの多くはパフォーマンスモニタリングユニット を備えており，ペナルティイベントの発生回数を低オーバヘッドに取得することができる．しかし一方で情報取得の時間粒度は最小でも 1,000 サイクル以上 [22] であり，命令単位での解析はできない．

5.3 依存グラフによる動的な命令列の表現

命令単位で動的な情報を取得し，プログラムおよびマイクロアーキテクチャに起因する依存関係をモデル化する方法が知られている．[11] では 1 命令をディスパッチ，実行，コミットの 3 状態で表す依存モデルを提案している．また，[12] では 1 命令を 5 状態で表しており，[15] では 1 命令を 13 状態で表す詳細な依存モデルを構築している．本稿の提案モデルではこれらの依存モデルのいずれも表現できていないマイクロアーキテクチャの特徴の表現を実現している．これらのモデル化方法は動的な命令列の情報を命令単位で表現することができ，実行パスの情報を含んでいる点で優れている．一方で，依存グラフを構築するために必要な情報量が多く，情報取得のオーバヘッドが大きい．オーバヘッド削減のため，サンプリングにより得た断片的な情報から依存グラフを構築する提案 [17] がなされている．

依存モデルから得られる情報として，以下のものが提案されている．

- 実行時間を決定する実行パス（クリティカルパス）[11]
- クリティカルパスでない命令の実行を遅らせた時にクリティカルパスとなるサイクル数 [10]
- クリティカルパスに乗っている命令の実行に要するサイクル数を減じた時にクリティカルパスでなくなるサイクル数 [21]
- 複数のハードウェアパラメタを同時に変更した時の実行時間への影響 [12]

依存グラフの解析により得られるこれらの情報は，プログラムの特徴解析やアーキテクチャの設計空間探索に役立つ．さらに，値予測などのペナルティの大きな投機実行や不均一性を持つアーキテクチャにおいて投機実行を行うかどうかの動的な制御にも利用することができる．

6. おわりに

アウトオブオーダプロセッサでは命令順を入れ替えながら複数同時に実行されるため，動的な命令列の解析は困難である．これを解析するために命令列の依存グラフによる表現が提案されている．しかしながら，既存の依存モデルは単純なプロセッサ構造を前提としており，異なるマイクロアーキテクチャの特徴や詳細度を持つプロセッサを前提とする場合うまく表現できない．そこで，本稿では Gem5 におけるアウトオブオーダプロセッサを前提とした場合に表現できていないマイクロアーキテクチャの特徴を明らかにし，それらを表現するための依存モデルを提案した．依存モデルから得られるクリティカルパスのサイクル数を評価した結果，従来の依存モデルでは実行サイクル数に対する平均相対誤差が 22% であったが提案モデルでは 2.1%となり，20%の改善が見られた．さらなる依存モデルの改善，依存モデルを用いたプロセッサの最適化は今後の課題である．

謝辞 本研究は一部，JST CREST「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」の支援を受けたものである．計算機の利用にあたり九州大学情報基盤研究開発センターの協力を得た．

参考文献

[1] Ailamaki, A., DeWitt, D. J., Hill, M. D. and Wood, D. A.: DBMSs on a Modern Processor: Where Does Time Go?, *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., pp. 266–277 (1999).

[2] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The Gem5 Simulator, *SIGARCH Computer Architure News*, Vol. 39, No. 2, pp. 1–7 (online), DOI: 10.1145/2024716.2024718 (2011).

[3] Burger, D. and Austin, T. M.: The SimpleScalar Tool Set, Version 2.0, *SIGARCH Computer Architecture News*, Vol. 25, No. 3, pp. 13–25 (online), DOI: 10.1145/268806.268810 (1997).

[4] Compaq Computer Corporation: *Alpha 21264/EV67 Microprocessor Hardware Reference Manual*, Shrewsbury, Massachusetts, 1.5 edition (2002).

[5] Dennard, R. H., Gaensslen, F. H., Yu, H.-N., Rideout, V. L., Bassous, E. and Leblanc, A. R.: Design Of Ion-implanted MOSFET's with Very Small Physical Dimensions, *Proceedings of the IEEE*, Vol. 87, No. 4, pp. 668–678 (online), DOI: 10.1109/JPROC.1999.752522 (1999).

[6] Dennard, R. H., Cai, J. and Kumar, A.: A perspective on today's scaling challenges and possible future directions, *Solid-State Electronics*, Vol. 51, No. 4, pp. 518–525 (online), DOI: http://dx.doi.org/10.1016/j.sse.2007.02.004 (2007).

[7] Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K. and Burger, D.: Dark Silicon and the End of Multicore Scaling, *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, New York, NY, USA, ACM, pp. 365–376 (online), DOI: 10.1145/2000064.2000108 (2011).

[8] Eyerman, S., Smith, J. E. and Eeckhout, L.: Characterizing the branch misprediction penalty, *IEEE International Symposium on Performance Analysis of Systems and Software*, IEEE Computer Society, pp. 48–58 (online), DOI: 10.1109/ISPASS.2006.1620789 (2006).

[9] Eyerman, S., Eeckhout, L., Karkhanis, T. and Smith, J. E.:

A performance counter architecture for computing accurate CPI components, *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ASPLOS '06, Vol. 41, No. 11, New York, New York, USA, ACM Press, p. 175 (online), DOI: 10.1145/1168857.1168880 (2006).

[10] Fields, B., Bodík, R. and Hill, M. D.: Slack: Maximizing Performance Under Technological Constraints, *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ISCA '02, Washington, DC, USA, IEEE Computer Society, pp. 47–58 (online), DOI: 10.1109/ISCA.2002.1003561 (2002).

[11] Fields, B., Rubin, S. and Bodík, R.: Focusing Processor Policies via Critical-path Prediction, *Proceedings of the 28th Annual International Symposium on Computer Architecture*, ISCA '01, New York, NY, USA, ACM, pp. 74–85 (online), DOI: 10.1145/379240.379253 (2001).

[12] Fields, B. A., Bodík, R., Hill, M. D. and Newburn, C. J.: Using Interaction Costs for Microarchitectural Bottleneck Analysis, *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, Washington, DC, USA, IEEE Computer Society, pp. 228–239 (2003).

[13] Henning, J. L.: SPEC CPU2006 Benchmark Descriptions, *SIGARCH Comput. Archit. News*, Vol. 34, No. 4, pp. 1–17 (online), DOI: 10.1145/1186736.1186737 (2006).

[14] Keeton, K., Patterson, D. A., He, Y. Q., Raphael, R. C. and Baker, W. E.: Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads, *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA '98, Washington, DC, USA, IEEE Computer Society, pp. 15–26 (online), DOI: 10.1145/279358.279364 (1998).

[15] Lee, J., Jang, H. and Kim, J.: RpStacks: Fast and Accurate Processor Design Space Exploration Using Representative Stall-Event Stacks, *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '14, Washington, DC, USA, IEEE Computer Society, pp. 255–267 (online), DOI: 10.1109/MICRO.2014.26 (2014).

[16] Loh, G. H., Subramaniam, S. and Xie, Y.: Zesto: A cycle-level simulator for highly detailed microarchitecture exploration, *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS '09, IEEE Computer Society, pp. 53–64 (online), DOI: 10.1109/ISPASS.2009.4919638 (2009).

[17] Salverda, P., Tu ker, C. and Zilles, C.: Accurate Critical Path Prediction via Random Trace Construction, *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '08, New York, NY, USA, ACM, pp. 64–73 (online), DOI: 10.1145/1356058.1356068 (2008).

[18] Semeraro, G., Magklis, G., Balasubramonian, R., Albonesi, D. H., Dwarkadas, S. and Scott, M. L.: Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling, *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, IEEE Computer Society, pp. 29–40 (online), DOI: 10.1109/HPCA.2002.995696 (2002).

[19] Shen, J. P. and Lipasti, M. H.: *Modern processor design : fundamentals of superscalar processors*, McGraw-Hill Higher Education, Boston (2005).

[20] Tune, E., Liang, D., Tullsen, D. M. and Calder, B.: Dynamic Prediction of Critical Path Instructions, *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, HPCA '01, Washington, DC, USA, IEEE Computer Society, pp. 185–195 (2001).

[21] Tune, E., Tullsen, D. M. and Calder, B.: Quantifying Instruction Criticality, *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, PACT '02, Washington, DC, USA, IEEE Computer Society, pp. 104–113 (2002).

[22] Yang, X., Blackburn, S. M. and McKinley, K. S.: Computer Performance Microscopy with Shim, *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, New York, NY, USA, ACM, pp. 170–184 (online), DOI: 10.1145/2749469.2750401 (2015).