社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－ARC－176 (9)
2008－EMB－ 7 (9)
2008／1／15

# ディペンダビリティに配慮する可変レイテンシ演算器のためのスケジューリング方式

佐藤寿倫[†] 渡辺慎吾[††]

半導体微細化技術の進展に伴い，パラメータばらつきの問題が深刻化している．パラメータばらつきはトランジスタの閾値電圧に影響し，その結果，回路遅延にもばらつきが現れる．この遅延ばらつき問題の解決策として，近年，可変レイテンシ加算器や長レイテンシ加算器が研究されている．残念ながら，遅延ばらつきの悪影響を被った演算器を長レイテンシの演算器で置き換えると，プロセッサ性能が大幅に低下してしまう．演算器の置き換え後にもプロセッサ性能を維持するために，本稿では命令の重要度に着目した動的命令スケジューリングを提案する．重要度の低い命令のみを長レイテンシ演算器で実行することで，プロセッサ性能の維持を図る．シミュレーションによる評価の結果，4 つの整数 ALU のうち 2 つが遅延ばらつきの悪影響を被っても，提案方式は従来方式と比較してプロセッ性能を平均 12.5%改善することが判明した．さらに，遅延ばらつきのない非現実的なプロセッサと比較した際の性能低下は，平均 4.0%に抑えられることが判明した．

# Dynamic Instruction Scheduling for Variable-latency Arithmetic Units in Dependable Processors

Toshinori Sato[†] Shingo Watanabe[††]

The advance in semiconductor technologies presents the serious problem of parameter variations. They affect threshold voltage of transistors and thus circuit delay also has variations. Recently, variable latency adders and long latency adders are proposed to manage the variation problem. Unfortunately, replacement of variation-affected adder with the long latency ones has severe impact on processor performance. In order to maintain performance, this paper proposes an instruction scheduling technique considering instruction criticality. By issuing and executing only uncritical instructions in the long latency ALU, we can maintain processor performance. From detailed simulations, we find that the proposed scheduling technique improves processor performance by 12.5% on average over the conventional scheduling, and performance degradation from a variation-free processor is only 4.0% on average when 2 of 4 ALU's are affected by variations.

## 1. はじめに

半導体製造技術における微細化の進展により，パラメータばらつきが増加している [1]．パラメータばらつきはトランジスタの閾値電圧に影響を与え，その結果，回路遅延にもばらつきが顕現する．ひいてはパラメータ歩留まりにも影響し，半導体メーカの収益を左右することとなる．幸いなことに，チップ上の全ての回路が遅延ばらつきの悪影響を被るわけでは無い．したがって，遅延ばらつきの悪影響を被った回路のみを遅延ばらつきに対処した回路に置き換えることで，そのようなチップであっても出荷可能になる．このような考えに基づいて，可変レイテンシ加算器や長レイテンシ加算器が提案されている [2, 3]．それらは非常に単純な方法であり，遅延ばらつき対策として優れている．残念ながら，遅延ばらつきの悪影響を被った加算器を，それらの加算器で置き換えると，プロセッサ性能が大きく低下する．この問題を考慮して，本稿で，可変レイテンシに配慮する動的命令スケジューリング方式を提案する．命令の重要度に着目するスケジューリング方式では，すでに様々な提案がなされている [4, 5, 6]．しかしそれらのいずれも，回路の単純さと重要命令の特定精度の両方を，同時には満足していない．本稿ではこれら二つの要求を満足するために，非重要度に基づくスケジューリング方式と，そのために用いる孤立命令特定表とを提案する．

本稿の構成は以下のとおりである．2 節で重要度に基づく命令スケジューリングを説明する．3 節で非重要度に基づいて命令を動的にスケジューリングする方式を提案する．4 節で評価方法を述べる．5 節でシミュレーション結果を紹介する．6 節はまとめである．

† 九州大学
Kyushu University
†† 九州工業大学
Kyushu Institute of Technology

## 2. 重要度に基づくスケジューリング

近年では組み込みプロセッサの中にも，性能向上の目的でアウトオブオーダー実行を採用しているものが存在する [7, 8]．本稿で提案する方式は，それらの組み込みプロセッサでの利用を想定している．プログラムの実行時間は，プロセッサが本来備えている処理性能と，プログラム内の命令間に存在する依存とで決定される．データフローグラフ（DFG）では，各節点が

命令を，各枝が命令間の依存関係を表している．クリティカルパスとは DFG 上で最長の経路であり，プログラムの実行時間を定める．図 1 に DFG の例を示す．全ての命令の実行レイテンシが 1 であると仮定すると，この DFG におけるクリティカルパスは I0→I3→I4→I6→I7 となる．

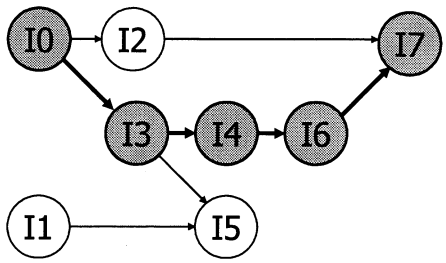


図 1　DFG とクリティカルパス

DFG とクリティカルパスから得られる情報を利用して，短レイテンシと長レイテンシの二つの異なる演算器を持つプロセッサを研究してきた[6]．クリティカルパス上の命令はプログラムの実行時間を決定するので，短レイテンシである高速な演算器上で実行される．クリティカルパス上に無い命令は，長レイテンシである低速な演算器上で実行される．その様子を図 2 に示す．このスケジューリング方式は，プロセッサ性能に悪影響を及ぼさないことが期待される．この，命令の重要度に基づくスケジューリング方式を採用するためには，クリティカルパスを特定する何らかの機構が必要になる．

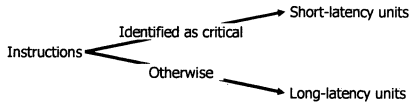


図 2　重要度に基づく命令スケジューリング

### 2.1　関連研究

クリティカルパスは互いに依存関係にある命令からなる鎖であり，プログラムの実行サイクル数を決定する．つまり，クリティカルパスに沿って実行する速度でプロセッサ性能が制約される．クリティカルパスを特定できれば，実行速度を改善できる可能性がある．

クリティカルパス予測器(CPP:Critical path predictor) [4, 5, 6] は，クリティカルパス上の命令を動的に特定する機構である．そこから得られる命令の重要度に関する情報を利用して，プロセッサ性能を改善出来る．CPP はクリティカルパスの特定に利用可能であるが，単純な回路で高精度な予測が可能なものはまだ存在しない [9]．複雑な回路は予測精度が高いが，余計な電力を消費する．逆に予測精度が低いと，プロセッサ性能を悪化させてしまう．

## 3.　非重要度に基づくスケジューリング

### 3.1　概要

これまで数年に渡ってクリティカルパスを特定する方式の検討を進めてきたが，いまだに単純な回路で高精度な特定が可能な方式の考案には至っていない [9]．本稿では重要度ではなく非重要度に着目して，プロセッサ性能の低下を引き起こさない方式を提案する．それは，図 3 に示されるように，クリティカルパス上に無い命令だけを長レイテンシである低速な演算器上で実行するものである．

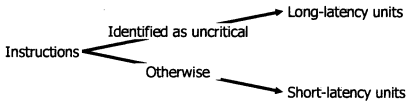


図 3　非重要度に基づく命令スケジューリング

クリティカルパス上にある命令を特定する目的には CPP が存在するが，クリティカルパス上に無い命令を特定する方式は存在しない．CPP では回路的な単純さと特定における高精度とを両立出来ないので，クリティカルパス上に無い命令を特定するための機構も，これらの二つの要求を両立は出来ないと想像される．しかし実際には，そのような機構を容易に実現出来る．

アウトオブオーダー実行するプロセッサには，命令ウインドウが備わっている．そこでは，オペランドの揃っていない命令が待機している．演算のためのオペランドが揃うと，その命令は演算器に発行され実行に移る．ここではそれらの命令を**レディ**命令と呼ぶ．命令ウインドウ中には二種類の命令が混在している．ひとつは，それに依存する命令が命令ウインドウ中に存在する命令．もう一つは，そのような依存命令が存在しない命令である．ここでは後者を**孤立**命令と呼ぶ．孤立命令を急いで実行する必要は無い．なぜなら，その結果を直ちに必要とする命令が存在しないからである．つまり孤立命令はクリティカルパス上に無い命令であり，長レイテンシである低速な演算器上で実行可能である．

図4に非重要度に基づく命令スケジューリングの例を示す．図1と同じDFGを用いている．時刻#0においては，命令I0〜I3の4命令がウインドウ中に存在すると仮定している．点線の四角が命令ウインドウを表しており，薄い灰色の節点は将来ウインドウにディスパッチされる未来の命令を表している．命令I0とI1がレディ命令である．命令I0には二つの依存命令I2とI3が存在するが，命令I1には無い．命令I1は孤立命令であり，低速な演算器に発行される．一方I0は高速な演算器に発行される．時刻#1では，命令I0とI1はウインドウから消えており，命令I4とI5がウインドウにディスパッチされている．命令I2とI3がレディ命令であり，I3だけが孤立命令である．I2が高速演算器へ，I3が低速演算器へ発行される．この例から，非重要度に基づく命令スケジューリングの動作が理解出来るだろう．

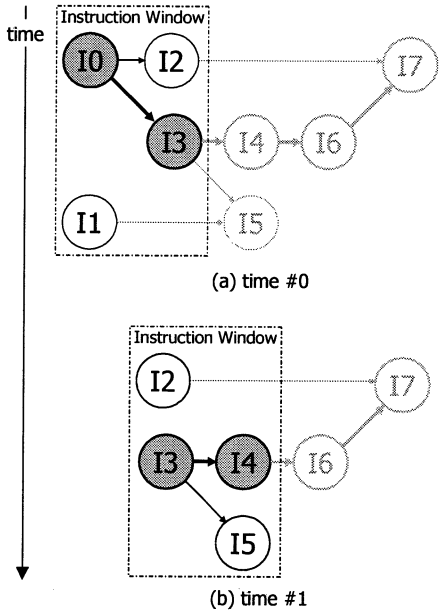


図4　非重要度に基づく命令スケジューリング例

### 3.2　孤立命令の特定法

続いて，孤立命令を特定するための機構を提案する．準備として，まずレジスタリネーミングを説明する．

命令レベル並列性を抽出する妨げとなる逆依存と出力依存とを解消するために，アウトオブオーダー実行プロセッサではレジスタリネーミングが活用されている．レジスタリネーミングの実装には二つの方法がある．ひとつはリネーミング専用のレジスタを備える方法である．例えばリオーダーバッファがこれに相当する．もう一つは，命令セットで定義されるレジスタとリネーミング用のレジスタを統合する方法である．本稿では後者を採用する．このリネーミング法では，レジスタをマップする機構が必要になる．それは，マップ表，アクティブリスト，そしてフリーリストから構成される．マップ表を使って，各論理レジスタにそれぞれ物理レジスタを割り付ける．ソースレジスタはマップ表に基づいて割り付けられるが，デスティネーションレジスタはフリーリストから提供される空のレジスタに割り付けられる．その同じ論理デスティネーションレジスタに割り付けられていたレジスタは，アクティブリストに退避される．命令がコミットすると，アクティブリストに退避していたレジスタが開放され，フリーリストに追加される．

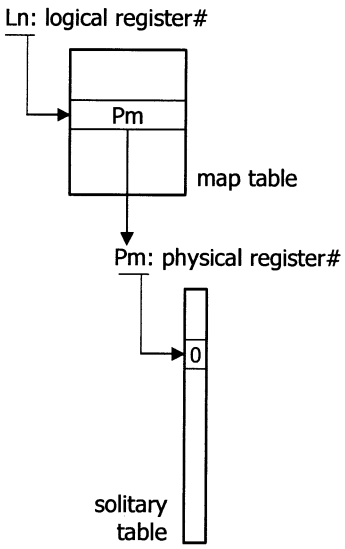


図5　孤立命令特定表

孤立命令を特定するために，上記のマップ表を利用する．図5が孤立命令を特定する機構である．マップ表に，極めて小さな表が追加されている．この表を，孤立命令特定表と名づける．孤立命令特定表の各エントリは1ビットであり，物理レジスタと同じ数のエントリが備わっている．たいていのプロセッサは高々数十の物理レジスタを備えるだけなので，孤立命令特定表のハードウエア規模が小さいことが容易に理解されるだろう．見方を変えると，各物理レジスタが1ビッ

トだけ拡張されたと考えることも出来る．孤立命令特定表は以下のように動作する．①新しい物理レジスタがデスティネーションとして割り当てられると，孤立命令特定表中の該当するエントリがセットされる．②後続の命令がマップ表により論理レジスタ番号（図中 Ln）を物理レジスタ番号（図中 Pm）に変換すると，その物理レジスタ番号で指定される孤立命令特定表のエントリがリセットされる．③命令の発行時に孤立命令特定表を参照し，物理デスティネーションレジスタ番号で参照されるエントリがセットされたままであれば，孤立命令として特定出来たことになる．

この方法の特定精度は 100%である．なぜなら，命令ウインドウ中の全命令がディスパッチ時に孤立命令特定表を更新しているからである．したがって以上の説明から，孤立命令特定表が単純なハードウエアで高精度に非重要命令を特定可能であることが理解出来るだろう．

### 3.3 可変レイテンシ演算器向けスケジューリング

パラメータばらつきの影響下でプロセッサ性能を維持するために，可変あるいは長レイテンシ加算器と，上述の非重要度に基づく命令スケジューリングとを組み合わせる．クリティカルパス上に無い命令だけを，遅延ばらつきの悪影響を被った演算器を置き換える，可変あるいは長レイテンシ演算器に発行する．

## 4. 評価方法

表 1　プロセッサ構成

| | |
|---|---|
| フェッチ幅 | 8 命令 |
| L1 命令キャッシュ | 16K, 2 ウエイ, 1 サイクル |
| 分岐予測器 | gshare＋bimodal |
| gshare 予測器 | 4K エントリ, 履歴 12 ビット |
| bimodal 予測器 | 4K エントリ |
| 分岐ターゲット表 | 1K セット, 4 ウエイ |
| ディスパッチ幅 | 4 命令 |
| 命令ウインドウ | 32 エントリ |
| 発行幅 | 4 命令 |
| 整数 ALU | 4 個 |
| 整数乗算器 | 2 個 |
| 浮動小数点 ALU | 1 個 |
| 浮動小数点乗算器 | 1 個 |
| L1 データキャッシュポート | 2 ポート |
| L1 データキャッシュ | 16K, 4 ウエイ, 2 サイクル |
| 共有 L2 キャッシュ | 8M, 8 ウエイ, 10 サイクル |
| メモリ | 容量無限, 100 サイクル |
| コミット幅 | 8 命令 |

SimpleScalar/PISA ツールセット[10]を用いてシミュレータを作成した．表 1 にプロセッサ構成をまとめる．これから先では，演算器として整数 ALU（iALU）のみに注目することにする．

SPEC2000 CINT からの 6 つのプログラムと，MediaBench [11]からの 6 つのプログラムとを使用する．SPEC では，最初の 2 億命令をスキップし，そのあとの 1 億命令をシミュレーション対象とする．MediaBench では，最初から最後までシミュレーションを実施する．NOP 命令は命令数に数え入れていない．以下の 3 つの想定ケースを評価する．iALU がそれぞれ 1 個，2 個，3 個，遅延ばらつきの悪影響を被る場合である．その際には，レイテンシが 1 である iALU を，レイテンシが 2 である低速な iALU で置き換えることにする．

## 5. 結果

図 6 は，提案する命令スケジューリング方式による性能改善を示している．実行サイクル数の削減率を指標として用いている．3 本の棒のうち左端は，遅延ばらつきのために一つの iALU が低速な iALU に置き換えられた場合の結果である．中央は二つの iALU が，右端は三つの iALU が，それぞれ置き換えられた場合の結果である．一つの iALU が置き換えられたプロセッサ上で epic を実行する場合を除いて，全ての場合でプロセッサ性能が改善されていることが確認出来る．3 つの想定ケースでそれぞれ，平均 9.9%，12.5%，11.1%の性能改善が達成されている．

図 7 は，遅延ばらつきのプロセッサ性能への影響を，提案方式が抑制している様子を示している．実行サイクル数の増加を指標として採用している．6 本の棒のうち，左の 3 本（図中 Conv:X）が従来のスケジューリング方式を採用した場合の結果であり，右の 3 本(図中 UCB:X）が提案方式を採用した場合の結果である．それぞれの 3 本の棒のうち左端は，遅延ばらつきのために一つの iALU が低速な iALU に置き換えられた場合の結果である．中央は二つの iALU が，右端は三つの iALU が，それぞれ置き換えられた場合の結果である．X が置き換えられた iALU の数を表している．

容易に見て判るとおり，従来のスケジューリング方式ではプロセッサ性能を大幅に悪化させてしまう．特に，遅延ばらつきの影響が深刻で 3 つの iALU が置き換えられる場合に顕著である．その場合には，最大で 32.6%，平均で 21.8%もの性能低下が確認出来る．対照的に，提案方式ではプロセッサ性能の維持に成功して

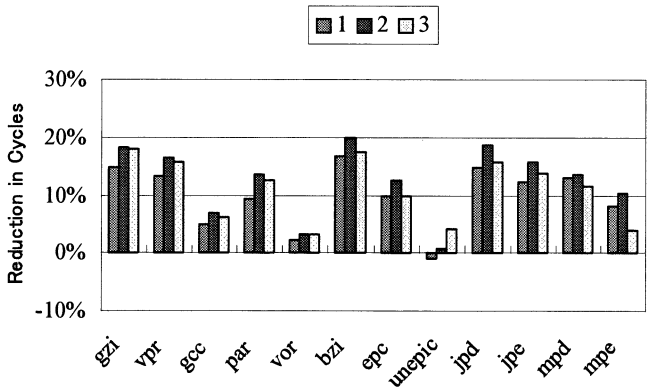


図6　性能改善

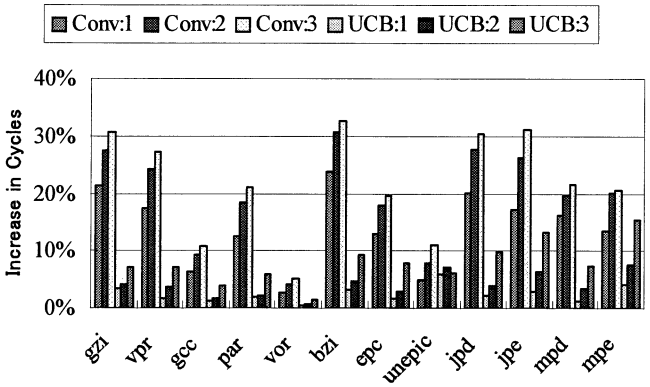


図7　実行サイクル数の増加

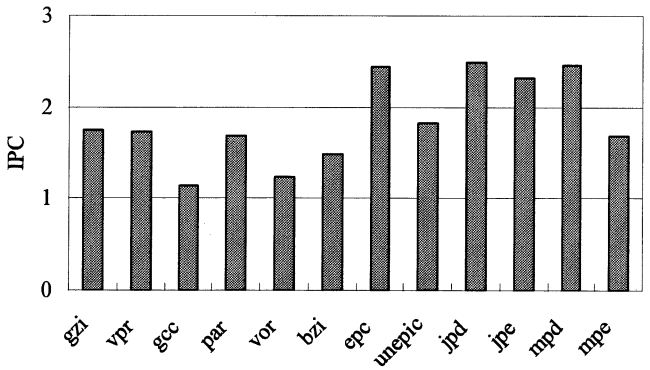


図8　ベース性能

いる．3つのiALUが低速になっても，平均で7.9%の性能低下に過ぎない．遅延ばらつきの影響が一つのiALUのみであれば，わずか2.5%の性能低下に過ぎない．

これらの結果から，提案する命令スケジューリング方式は，可変レイテンシ加算器あるいは長レイテンシ加算器を有効に活用して，遅延ばらつきに対処出来るということが判明した．

図8には，遅延ばらつきの影響が無いときのプロセッサ性能がまとめられている．サイクルあたりの命令数（IPC）を指標として用いている．興味深いことに，グラフの形状が図6の形状と似ていることがわかる．つまり，元の性能が高ければ高いほど，提案方式による性能改善の度合いが大きいということである．例外はvortexとepicである．

## 6. おわりに

半導体における微細化技術が進展するにつれ，パラメータばらつきに起因する回路遅延のばらつきが問題視されるようになってきた．近年提案されている可変レイテンシ加算器や長レイテンシ加算器は，遅延ばらつきに対するひとつの解であるが，それらの採用はプロセッサ性能に深刻な影響を及ぼす．本稿では，非重要度に基づく動的命令スケジューリングを提案し，それを遅延ばらつきへの対処に応用した．クリティカルパス上に無い命令のみを低速な演算器で実行することで，プロセッサ性能の維持を図っている．シミュレーションによる評価の結果，例えば4つの加算器の二つが遅延ばらつきの悪影響を被る場合には，従来のスケジューリング方式と比較して本稿での提案方式は，プロセッサ性能を平均12.5%改善出来ることが判明した．遅延ばらつきの無い非現実的なプロセッサと比較しても，平均で4.0%の性能低下に抑えることが可能である．以上から，非重要度に基づく動的命令スケジューリング方式は，耐遅延ばらつきマイクロアーキテクチャとして有望であると言える．

**謝辞** 本研究は一部，科学研究費補助金・基盤研究Aの課題「価値と信用を搭載するディペンダブルなLSIの設計手法の研究」（課題番号19200004）および，科学技術振興機構・戦略的創造研究推進事業（CREST）における領域「ディペンダブルVLSIシステムの基盤技術」の課題「統合的高信頼化設計のためのモデル化と検出・訂正・回復技術」の支援による．

## 参考文献

1) S. Borker: Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation, IEEE Micro, Vol. 25, No. 6 (2005)
2) Y. Chen, H. Li, J. Li, and C.-K. Koh: Variable-latency Adder (VL-adder): New Arithmetic Circuit Design Practice to Overcome NBTI, International Symposium on Low Power Electronics and Design (2007)
3) D. Mohapatra, G. Karakonstantis, and K. Roy: Low-power Process-variation Tolerant Arithmetic Units Using Input-based Elastic Clocking, International Symposium on Low Power Electronics and Design (2007)
4) E. Tune, D. Liang, D. M. Tullsen, and B. Calder: Dynamic Prediction of Critical Path Instructions, 7th International Symposium on High Performance Computer Architecture (2001)
5) B. Fields, S. Rubin, and R. Bodik: Focusing Processor Policies via Critical-Path Prediction, 28th International Symposium on Computer Architecture (2001)
6) 千代延昭宏，佐藤寿倫，有田五次郎：低消費電力プロセッサアーキテクチャ向けクリティカルパス予測器の提案，情処研報 2002-ARC-149-2 (2002)
7) V. Rajagopalan: New Area and Power-Efficient MIPS Processors Achieve High Performance, Microprocessor Forum (2007)
8) T. Sartorius: The Scorpion Mobile Application Microprocessor, Microprocessor Forum (2007)
9) 千代延昭宏，佐藤寿倫：新しいクリティカルパス判定基準を用いたクリティカルパス予測器の評価，情処研報 2006-ARC-169 (2006)
10) T. Austin, E. Larson, and D. Ernst: SimpleScalar: an Infrastructure for Computer System Modeling, IEEE Computer, Vol. 35, No. 2 (2002)
11) C. Lee, M. Potkonjak, and W. H. Mangione-Smith: MediaBench: a Tool for Evaluating and Synthesizing Multimedia and Communications Systems, 30th International Symposium on Microarchitecture (1997)