2003－ARC－154　(1)
2003／8／5

プログラムの実行時における命令の重要度決定に関する検討

千代延 昭宏† 佐藤 寿倫†‡
九州工業大学 †情報工学部 知能情報工学科 ‡マイクロ化総合技術センター

近年の研究でプログラム中のクリティカルパスに沿って実行を最適化させることで，プロセッサの処理性能向上や省電力化が可能であることがわかっている．これに対し我々は新たなクリティカルパス予測器を提案し評価を行ってきたが，満足のいく結果が得られていなかった．本稿では，クリティカルパス予測器の問題点を明らかにするために行った評価結果について報告する．

On Dynamic Identification of Instruction Criticality

Akihiro CHIYONOBU† Toshinori SATO†‡
† Department of Artificial Intelligence, Kyushu Institute of Technology
‡ Center for Microelectronic Systems, Kyushu Institute of Technology

Recent studies show that microprocessor performance or its energy efficiency can be improved, if we utilize information regarding instruction criticality. This paper investigates why we could not achieve low energy consumption and high performance simultaneously, and reveals there is a fundamental problem in heuristics to identify critical instructions.

1 はじめに

近年携帯情報端末や組み込みシステムにおいても高い処理性能が求められるようになっており，高性能なプロセッサが搭載されている．しかしこれらのシステムには高い処理性能が求められる一方で，その消費する電力も少量であることが要求される．

この問題に対して我々はプログラム実行中のクリティカルパスに着目して研究を行っている [7] が，実行される命令がクリティカルパス上の命令かどうか特定する方法が不十分であるため，性能と省電力の両立に必ずしも成功していない [9]．そこでクリティカルパス予測器の予測精度が向上しない原因を明らかにするため，命令ウインドウ内にある命令間のデータ・フローグラフを作成することでより厳密にクリティカルパスを特定する小林らの提案する命令発行機構 [6] と，クリティカルパス予測器を比較して，クリティカルパス特定方法について調査を行った．

本稿では，満足するクリティカルパス特定精度を得るため検討を行った結果について述べる．

2 クリティカルパスとクリティカルパスを特定する機構

クリティカルパスとは命令間の依存関係を結んだ鎖のうち最長のものを結んだ実行パスであり，プログラムの実行時間を決定する命令列である [6]．図 1 に命令列中に現れるクリティカルパスを表すデータ・フローグラフを示す．図 1 において矢印は命令間の依存関係を示す．つまり，依存している命令は矢印の始点にある依存先の命令実行が終了しない限り実行できない．全ての命令のレイテンシが 1 サイクルであるとすると，最も長いパスである命令 I:0 → I:3 → I:4 → I:6 → I:8 を結んだパスがクリティカルパスとなる．クリティカルパスの情報を利用できると，クラスタ化スーパースカラプロセッサにおける性能低下の低減 [3, 5, 6]，値予測の効率改善 [3, 5]，消費電力の削減 [1, 7] などに応用できる．しかし，そのためにはプログラムの実行時にクリティカルパスを特定する必要がある．続いて，クリティカルパスを特定するための機構を説明する．

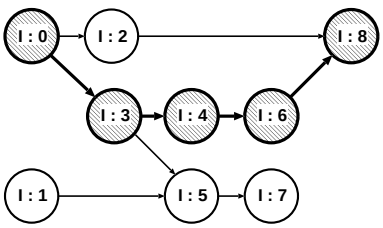

図 1: 命令間の依存関係とクリティカルパス

2.1 クリティカルパス予測器

命令がクリティカルか否かを判断する予測器をクリティカルパス予測器という．クリティカルパス予測に関する研究は近年注目され始めている [3, 5]．Tune らのクリティカルパスヒストリテーブル (CPHT:Critical Path History Table) は各命令ごとのローカルな履歴のみを利用しているが，我々は命令がクリティカルになる要因に命令間の依存関係があることに着目し，命令間のグローバルな相関を予測の際に利用できればクリティカルパス予測の精度が向上するのではないかと考え，命令間の依存関係に着目した 2 レベルクリティカルパス予測器を考案した [7]．しかしそれらの予測器を評価した結果，これまでのところ満足するクリティカルパスの予測精度は得られていない [9]．

2.2 パス情報テーブル

小林らはデータ・フローグラフ (DFG:Data Flow Graph) の最長パスを特定するためのパス情報テーブル (PIT:Path Info Table) を提案している [6]．PIT を用いると DFG は以下のようにして表現される．

(1) DFG に存在する各合流ノードについて，そのノードに入るエッジのうち，実行終了時刻が最も遅いノードからのエッジを残し他を削除する．こうすることにより，DFG は木となる．

(2) 木を互いに重なりのない複数の部分パスに分解し，各部分パスに含まれてるノードを FIFO に記憶する．部分パス間の結合情報は PIT を用意し記憶される．PIT は最長パスを終点から始点までたどるために使用する．

(3) 命令ウインドウ内の全命令のうち，実行終了時間が最も遅い命令を最長パスの終点として別途記憶する．

DFG を作成するために，デコードされた命令はオペランドを生成する命令の直後のエントリか，空の FIFO のエントリに挿入される．また，各命令に対応するエントリを持つテーブルを用意し，オペランドを定義する先行命令へのポインタと，その命令の実行終了時刻を保持する．ディスパッチされる命令はこのテーブルを参照することで，オペランドを生成する命令へのポインタとオペランドが生成される時刻を得る．オペランドが複数存在する場合は，実行終了時刻が最も遅い命令を選択する．当該命令は，FIFO 内のこうして得られた依存先の命令の直後に挿入される．テーブルから得られたオペランドが揃う時刻に当該命令のレイテンシを加えることで，実行終了時刻を得ることができる．当該命令の実行終了時刻が，記憶していた最長パスの終点の命令の実行完了時刻と比較して遅い場合は，当該命令を最長パスの終点として記憶し直す．最長パスの先頭は，最長パスの終点から DFG をたどることで求めることができる．終点は前述したように記憶されており，終点からパス間の結合情報を用いて順に DFG をたどり先頭を求める．

3 評価方法

SimpleScalar ツールセット (version 3.0)[2] のアウト・オブ・オーダ実行を行うシミュレータに，小林らの提案する命令発行機構と低消費電力化アーキテクチャ [7, 9] を組み込んでシュミレータを作成した．比較対象として CPHT[5] と我々の提案している GCPH 型，GBH 型，BOTH 型の 2 レベル型予測器 [7] も評価した．評価にはプロセッサの処理性能とエネルギー遅延積 (EDP:Energy Delay Product) を用いる．

評価に用いたプロセッサ構成は参考文献 [9] と同様である．また，MIPS R10000 を拡張した SimpleScalar /PISA を命令セットとして用いる．本稿では提案するアーキテクチャを整数 ALU に適用して評価する．プロセッサは ALU を 6 個持ち，高速な ALU と低速な ALU のレイテンシはそれぞれ 2 : 1 となるようにする．低速な ALU は，高速な ALU をパイプライン動作させるものに相当し，レイテンシは 2 に増加するが，スループットは 1 のままである．これらの ALU の電源電圧は，高速動作時に 1.1V，低速動作時に 0.7V とする [4]．ALU 構成は以下の様にして決定される．PIT を用いた場合のシミュレーション結果を図 2 に示す．左図に性能を，右図に EDP を示す．性能についてはグラフが高い方が好ましく，EDP についてはグラフが低い方が好ましい．図において `6fast/0slow` と `0fast/6slow` は，クリティカルパス情報を用いた最適化を適用しない場合を示す．前者は全ての整数 ALU が高速型であり，後者は全ての整数 ALU が低速型である．`1fast/5slow`, `2fast/4slow`, `3fast/3slow`, `4fast/2slow`, `5fast/1slow` はそれぞれ高速，低速な ALU が，1 : 5，2 : 4，3 : 3，4 : 2，5 : 1 の場合を示している．容易に判る様に低速な ALU を増すと，性能は単調に悪化する．一方 EDP は，必ずしも単調には改善されない．`bzip` では，`3fast/3slow` で最良となり，以後低速な ALU を増しても EDP は改善されない．これは高速な ALU の数が 3 つ以下になるとプロセッサ性能が著しく低下するためである．以上を考慮して，以後の評価は高速，低速な ALU を 3 つずつの場合で行うこととする．

クリティカルパス予測器のテーブルサイズは 2K エントリとし，その飽和型アップ・ダウンカウンタの更新値は命令がクリティカルと判定された場合に 4，クリティカルではないと判定された場合に −1 とした．またしきい値は 4 とし，GCPH 型，BOTH 型の予測器が利用する GCPH は過去 8 命令分とした．ベンチマークは SPEC 2000 CINT である．どのプログラムも，先頭の 1B 命令をスキップし，続く 100M 命令をシミュレーションした．

4 シュミレーション結果

4.1 省電力アーキテクチャ

これまでの評価 [9] から，現状のクリティカルパス予測器と省電力アーキテクチャの組み合わせでは，期待される効果が得られていないことが判っている．この原因が，(1) クリティカルパス予測器にあるのか，それとも (2) 省電力アーキテクチャにあるのかを特定するために，まず，より厳密にクリティカルパスを特定

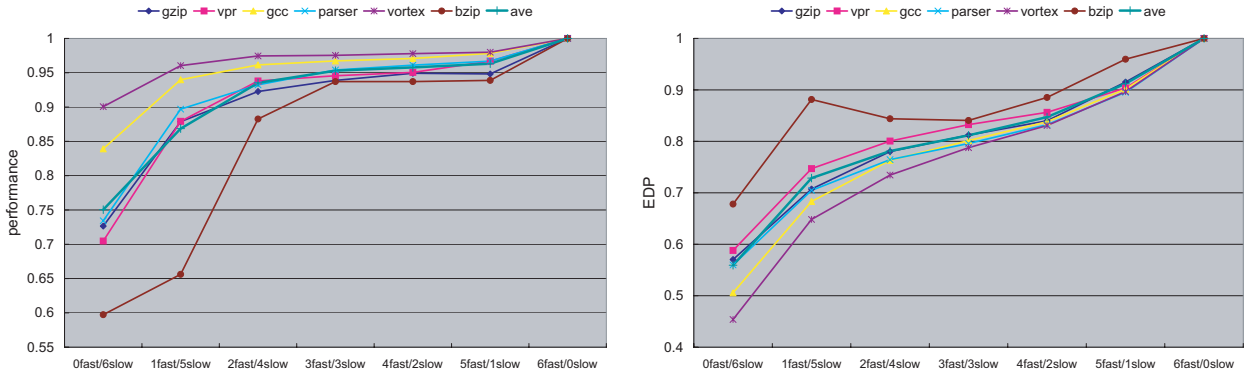

図 2: 最適な演算器の割合の評価

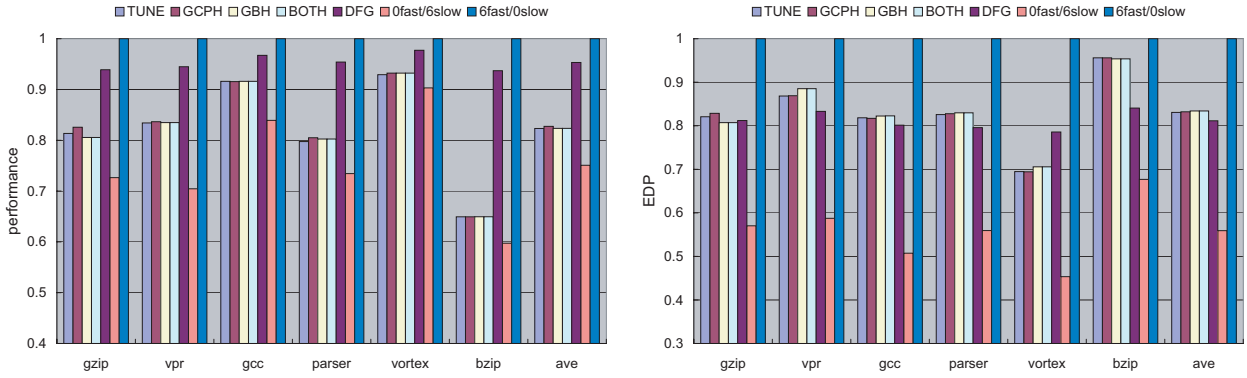

図 3: 省電力アーキテクチャの効果

できる PIT を用いて省電力アーキテクチャを評価した [8]．シュミレーション結果を図 3 に示す．クリティカルパス予測器の更新は命令の実行時 (issue 時) に行っている．評価にはプロセッサの処理性能とエネルギー遅延積 (EDP) を用いる．左図に性能を，右図に EDP を示す．性能についてはグラフが高い方が好ましく，EDP についてはグラフが低い方が好ましい．それぞれの図において `DFG` は PIT を用いる小林らの機構を表し,`TUNE` は Tune らの CPHT を，`GCPH`，`GBH`，`BOTH` はそれぞれ GCPH 型，GBH 型，BOTH 型の 2 レベル型予測器を表す．クリティカルパス予測器を利用しない場合を `6fast/0slow` と `0fast/6slow` と表す．前者は全ての整数 ALU が高速型であり，後者は全ての整数 ALU が低速型である．

まずプロセッサ性能について考察すると，いずれのクリティカルパス予測器を用いた場合と比較しても，PIT を用いた場合の方が，性能低下の度合いが極めて小さい．平均で見ると，全ての ALU を低速にした場合で約 25%，予測器を用いた場合でいずれも約 20%の性能低下であるのに対して，PIT を用いた場合は約 5%の性能低下に過ぎない．また，小林らの機構を用いた場合では性能低下の度合いにばらつきが見られない．いずれの場合でも約 5%の低下である．それに対して予測器を用いた場合の性能低下は 7%〜35%とベンチマークプログラムによって大きくばらついている．

EDP についても，多くのベンチマークプログラムにおいて PIT を用いた場合の方が省電力達成の割合が高い．平均で見ると，全ての ALU を低速にした場合で 44%の省電力化，予測器，PIT を用いた場合で 17%の省電力化を達成している．ベンチマークプログラムごとに見てみると，予測器を用いた場合はプロセッサ性能同様，省電力化の割合にバラつきがみられる．これに対し，PIT を用いた場合の省電力化達成の割合はほぼ一定である．

以上の結果から予測器の予測精度を考えてみる．予測器を用いて命令のスケジューリングを行った場合，EDP ではどのベンチマークプログラムにおいても PIT を用いた場合と同程度，もしくは上回る省電力を達成している．しかし，プロセッサ性能は最大で 28%も低下している．このことから，予測器を用いた場合は積

極的に低速な ALU を使用して省電力化を図っていることが判る．クリティカルパス上の命令を高速な演算器に，クリティカルパス上にない命令を低速な演算器に正しく発行していれば，プロセッサ性能を維持しつつ省電力化が実現できるはずである．このことを考えると，現状の予測器は実行しようとしている命令がクリティカルパス上にあるかどうか正しく予測できていないことが判る．

一方，PIT を用いた場合はどのベンチマークプログラムにおいても一定の処理性能低下で EDP の削減を実現しており，ベンチマークプログラム間で顕著な結果の差は見られない．このことから，PIT を用いた場合は実行しようとしている命令がクリティカルパス上にあるかどうか正確に特定できているといえる．

以上のことから，クリティカルパスを正確に特定できれば，高速，低速な ALU を使い分ける方式は省電力目的に有効であることが判った．

4.2 クリティカルパス判定基準

前節までの検討結果から，提案しているクリティカルパス予測器の予測精度を改善する必要があることが判った．クリティカルパス予測では，命令がクリティカルであるかどうかの判定にヒューリスティックを用いている．従って，(1) 予測機構に問題があるのか，それとも (2) ヒューリスティックを用いた判定基準に問題があるのか，を特定しなければならない．続いて我々は，クリティカルパス予測器の予測精度が満足できるほど高くない原因を調査するため，クリティカルパス判定基準の判定結果と予測器の予測結果を，PIT を用いた場合のクリティカルパス特定結果と比較する．これらの比較を行うことで，クリティカルパス予測器を更新する情報の正確性とその情報を用いた際のクリティカルパス予測器の予測精度を知ることができる．クリティカルパス判定基準は Tune らによって提案されたもの [5] で，詳細を表 1 に示す．また，比較結果を図 4 に示す．図において，左図はクリティカルパス判定基準と小林らの機構の特定結果との一致率を，右図はクリティカルパス予測器の予測結果と小林らの機構の特定結果の一致率を示している．また，図中の `commit`, `issue` はそれぞれクリティカルパス予測器の更新を命令のコミット時，実行時に行った場合を示す．

まずクリティカルパスの判定時に用いる判定基準ごとのクリティカルパス判定結果と PIT を用いたクリティカルパス特定結果とを比較する．更新のタイミングの違いで予測器の予測結果が異なるため結果に若干の差が見られるが，おおむね結果とその傾向は同じである．判断基準別について見てみる，ALOLD の一致率が最も高く 68%，次に QOLD の 57%，最も低い一致率を示したのは FREED3 の 22%であった．これは ALOLD，QOLD の判定結果が PIT の特定の結果と高く一致することを示している．一方 QCONS, FREED3 でクリティカルと判断された命令は PIT もクリティカルと特定はしているが，これらの基準の条件を満たす命令数が少ないためにクリティカルと判断する命令数が少なくなってしまい，結果として PIT の特定結果と差がでている．

表 1: クリティカルパス判定基準

基準名	内容
QOLD	各サイクルで，未発行命令のうち最古の命令をクリティカルと判定
ALOLD	各サイクルで，命令ウインドウ内の最古の命令をクリティカルと判定
QCONS	各サイクルで，最も多くの命令に生成する値が使われる命令をクリティカルと判定
FREED3	命令ウインドウ内で 3 つ以上の命令に依存されている命令をクリティカルと判定

次に予測結果と PIT による特定結果との一致率を見る．こちらは更新のタイミングの違いによって QOLD, ALOLD にやや差がみられるものの，全体の傾向としては判定結果の一致率同様予測器更新のタイミングは予測精度に影響していない．しかし以前の評価 [9] から，更新のタイミングはプロセッサ性能と EDP に影響することが判っている．従って，予測器更新のタイミングが予測に何らかの影響を与えていることは間違いない．よって，今後は予測器更新のタイミングが予測に与える影響について詳しく調査する必要もある．

以上から，ヒューリスティックを用いたクリティカルパス判定基準が好ましくないと予想される．

4.3 クリティカルパス予測器

前節で述べたように，クリティカルパス判定基準の判定結果とその情報を利用したクリティカルパス予測器の予測結果が PIT の特定結果と一致する確率は低い．我々は，各クリティカルパス判定基準の判定結果と PIT の特定する結果に差があることから，クリティカルパス予測器の更新に用いられる情報の精度が低かったためにクリティカルパスの予測精度も低くなったのではないかと考えた．そこでクリティカルパス予測器を更新する際の情報として，PIT の特定結果を用いることを考えた．これまで用いてきたクリティカルパス判定基準の代わりに，PIT が特定するクリティカルパス情報をクリティカルパス予測器更新の情報として用

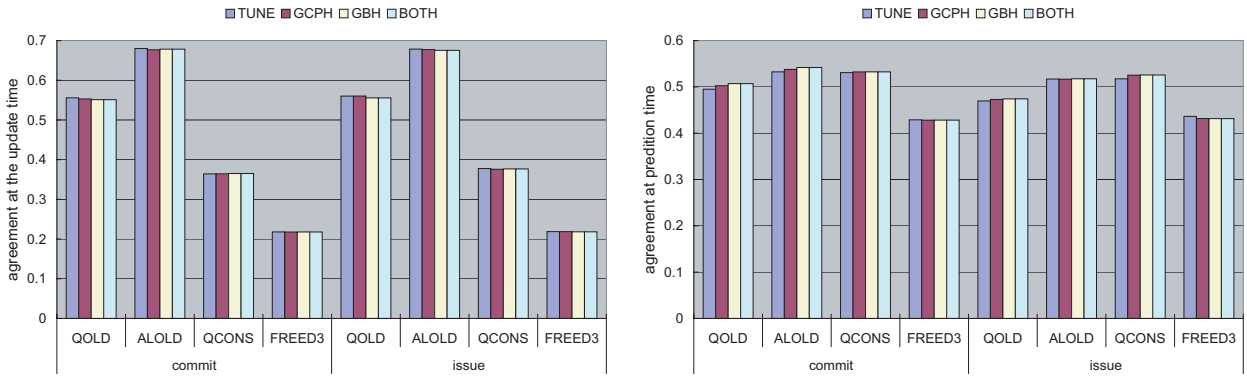

図 4: クリティカルパス判定結果と PIT の情報との比較結果

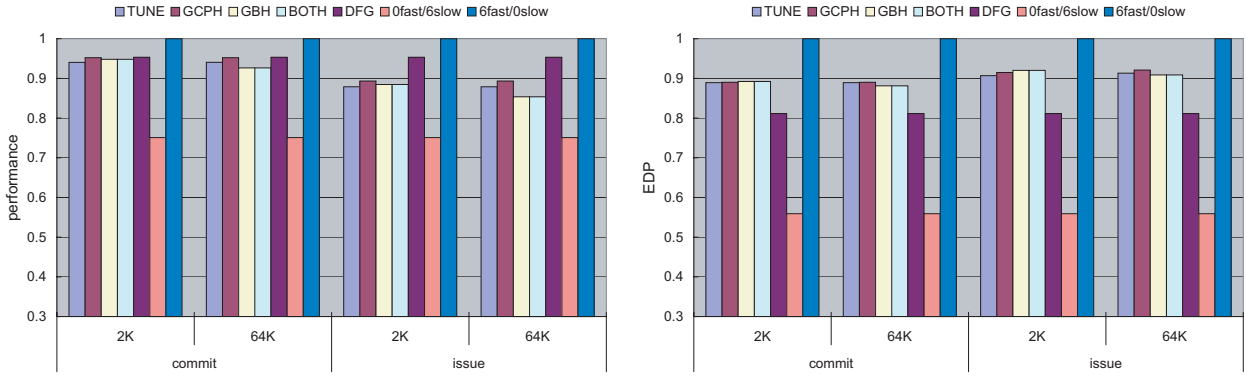

図 5: PIT の情報で予測器を更新した場合の省電力化の効果

いる．発行先の ALU の選択にはこれまで同様，クリティカルパス予測器の予測結果を用いる．この評価を行うことで，クリティカルパス判定基準の正確性がクリティカルパス予測器の予測結果に及ぼす影響を確認できる．

プロセッサ性能と EDP を表す評価結果を図 5 に，予測一致率を図 6 に示す．理想的なクリティカルパス予測器の状態も評価できるよう，テーブルサイズは 64K エントリの場合も評価した．`commit` は命令のコミット時にクリティカルパス予測器の更新を行った場合を，`issue` は命令の実行時にクリティカルパス予測器の更新を行った場合を示している．

プロセッサ性能と EDP について見てみる．まず 64K のエントリの理想的なテーブルを持つ場合について見てみる．コミット時更新の場合，プロセッサの性能低下の割合が各予測器とも 10%以内に抑えられている．実行時更新の場合においても各予測器とも 15%以内の性能低下に抑えている．しかし，PIT を用いた場合のプロセッサ性能と比較すると，コミット時更新の GCPH 型予測器を除いてどちらの場合においても，匹敵するプロセッサ性能を達成できていない．EDP はコミット時更新の場合で最大 13%，実行時更新の場合で最大 9%の省電力化しか実現できていない．プロセッサ性能で PIT を用いた場合と同等の結果を示したコミット時更新の GCPH 型は 11%の削減率を達成しているものの，PIT を用いた場合に約 20%の省電力化を実現していることを考慮すると，クリティカルパス予測器の予測精度は向上していないようである．

次にテーブルサイズを 2K エントリまで減らした場合について結果を見てみる．コミット時更新の場合，各予測器とも PIT を用いた場合とほぼ同等のプロセッサ性能を実現しており，64K エントリの場合よりもプロセッサ性能は向上している．同様に実行時更新の場合も，各予測器とも PIT を用いた場合よりは劣るが 64K エントリの場合よりもプロセッサ性能は向上している．EDP を見てみると，コミット時更新の場合最大 12%と 64K エントリの場合とほぼ同等の省電力化を実現している．実行時更新の場合も，最大 9%と 64K エントリの場合とほぼ同等の省電力化を達成している．しかし両者とも PIT を用いた場合と比較すると，そ

の省電力性はかなり劣る．エントリ数，予測器更新のタイミングを変化させた場合の結果から判断すると，少ないエントリ数で高いプロセッサ性能と省電力性を両立させている 2K エントリ，コミット時更新が最適だと思われる．しかし，EDP の削減率は PIT を用いた場合に比べ悪いことから正確にクリティカルパスを予測できている可能性は低い．

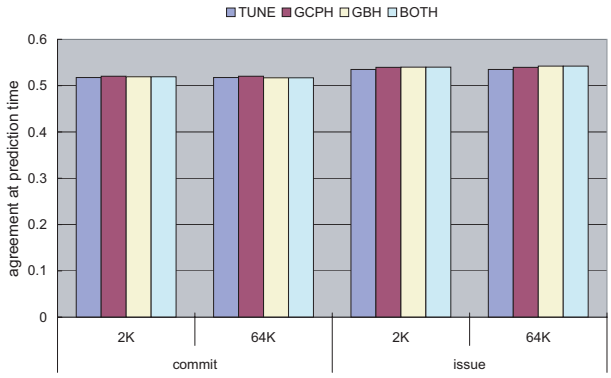

図 6: PIT の情報で予測器を更新した場合の一致率

次に図 6 に示すクリティカルパス予測器の予測結果と PIT の特定結果の一致率を見てみる．プロセッサ性能と EDP 削減率で最も良い結果を出した 2K エントリ，コミット時更新の場合，各予測器とも 51%の一致率，プロセッサ性能と EDP 削減率であまり良い結果を得られなかった実行時更新の場合は 53%の一致率とややコミット時更新よりも高い一致率を示している．

以上の結果からクリティカルパス予測器の更新時に用いる情報に PIT のクリティカルパス特定結果を用いても予測精度が改善しないということが判明した．この原因を考察する．PIT を用いると，各命令がクリティカルであるか否かの特定は毎サイクル更新される．一方で予測器を用いると，デコード時に予測された結果はそれ以後変更されない．つまり PIT を用いる場合ではプログラム実行時の動的な DFG の変化を追跡できるのに対し，現状のクリティカルパス予測器を用いた場合はそのようなことをしていない．また，現状の予測器ではそのようなことは不可能である．このことが期待とは異なる結果の原因と考えられる．

5 まとめ

クリティカルパス予測器を用いた省電力アーキテクチャから期待する効果が得られていない原因を調査した．その結果，

1. 厳密にクリティカルパスを特定できれば，提案している省電力アーキテクチャは省電力化に効果がある．
2. 命令がクリティカルパス上にあったか否かを判定するめのヒューリスティックを用いた判定では，十分な精度ではクリティカルパスを特定できない．
3. 判定基準を改善しただけでは，現状のクリティカルパス予測機構では十分な予測精度が得られない．

ことが判った．

今後は予測時のクリティカルパス予測器の振る舞いやスケジューリング時の命令の振る舞いを詳細に調査し，我々の求める正確な予測が可能なクリティカルパス予測器とクリティカルパス情報に基づいた最適なスケジューリングを行える機構の研究を行っていく予定である．

また現在命令のスケジューリングに用いると高い効果を発揮することが判っている PIT についても，その消費電力などを評価しクリティカルパス予測器の代わりに用いることが可能であるか研究する必要がある．

謝辞

本研究の一部は，科学研究費補助金，および北田奨学会記念財団の援助によるものです．

参考文献

[1] A.Braniasadi, A.Moshovos,“Asymmetric-Frequency Clustering: A Power-Aware Back-End for High-Performance Processors”, International Symposium on Low Power Electronics and Design, August 2002.

[2] D.Burger, T.M.Austin, “The SimpleScalar Tool Set, Version 2.0”, Technical Report CS-TR-97-1342, Computer Science Department, University of Wisconsin Madison, June 1997.

[3] B.Fileds, S.Rubin, R.Blodik: “Foucusing Processor Policies via Critical-Path Prediction”, the 28th International Symposium on Computer Architecture, July 2001.

[4] M.Levy: “SAMSUNG Twists ARM Past 1GHz”, Information Quarterly, vol.1, no.1, 2002.

[5] E.Tune, D.Liang, D.M.Tullsen, B.Calder: “Dynamic Prediction of Critical Path Instructions”, the 7th International Symposium on High Perdormance Computer Arcitecture, January 2001

[6] 小林良太郎, 安藤秀樹, 島田俊夫: “データフロー・グラフの最長パスに着目したクラスタ化スーパースカラ・プロセッサにおける命令発行機構”, 2001 年並列処理シンポジウム JSPP2001, 2001 年 6 月.

[7] 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 , 2002 年 8 月.

[8] 千代延昭宏, 佐藤寿倫, 有田五次郎: “省エネルギー応用におけるプログラムの最長パス特定方法に関する考察”, 先進的計算基盤システムシンポジウム SACSIS2003, 2003 年 5 月.

[9] 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評価”, 電子情報通信学会論文誌　和文論文誌 C, Vol.J86-C, No.8, 2003 年 8 月.