計算機アーキテクチャ 149－2
（2002．8．22）

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

千代延 昭宏† 佐藤 寿倫†‡ 有田 五次郎†
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

近年の研究でスーパスカラプロセッサ上でプログラム中のクリティカルパスを効率良く実行することにより，プロセッサの処理性能を向上させることが可能であることが知られている．我々の研究では実行中の命令がクリティカルであるかどうかの情報を用いて，プロセッサの処理性能を低下させることなく消費電力を削減させることを目指す．本稿では，我々が提案する新たなクリティカルパス予測器を用いて低消費電力化の最適化評価を行った結果について報告する．

A Proposal of Critical Path Predictors for Low Power Processor Architecture

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

Recent studies show that microprocessors can be accelerated, if we can identify which instructions are critical. Exploiting information regarding instruction criticality is effective not only for improving processor performance but also for improving energy efficiency. In this paper, we propose three critical path predictors to identify critical instructions. Experimental results demonstarate the future resarch direction of our approach.

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求められている．プロセッサの消費電力と性能はトレードオフの関係にあるため，特に携帯情報端末用プロセッサでは電力消費量が設計における大きな制約となる．マイクロプロセッサの消費電力 P_{active} と遅延時間 t_{pd} はそれぞれ，

$$P_{active} = fC_{load}V_{dd}^2 \quad (1)$$

$$t_{pd} \propto \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \quad (2)$$

で求めることができる．ここで，f はクロック周波数，C_{load} は付加容量，V_{dd} は電源電圧，そして V_{th} はデバイスの閾値電圧である．また，α はキャリア移動度の飽和を表すパラメタである．式 (1) からわかるように電源電圧を下げることが電力削減においては最も効果が高い．しかし，式 (2) に従えば，電源電圧の低下はゲート遅延を増大してしまい，クロック周波数が低下してしまう．このことは，マイクロプロセッサの性能低下につながる．この問題に対して本研究では，プログラム実行中のクリティカルパスに着目した．具体的にはレイテンシと電源電圧の異なる演算器を用意し，実行時間に影響を与えるクリティカルパス上の命令は高速かつ電力消費の大きな演算器に実行させ，クリティカルパス上にない命令は低速かつ消費電力の小さな演算器で実行させる．このことにより，実行時間を増やすことなくプロセッサの消費電力を抑えることが可能となる．本稿では上記のアーキテクチャ実現に不可欠なクリティカルパス予測器について述べる．

2 クリティカル情報を利用した電力削減

現在多くのプロセッサは実行しようとする命令を機能ユニットに対してアウト・オブ・オーダに発行することでプログラムの実行時間を少なくしている [1]．プログラムの実行時間は，プロセッサの処理性能と実行している命令間の依存関係によって決まる．

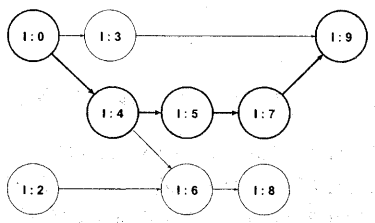

図 1: 命令間の依存関係とクリティカルパス

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 [2]．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:4 → I:5 → I:7 → I:9 を結んだパスがクリティカルパスとなる．

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, 4]．Tune 等の予測器 [3] は各命令の履歴に基づいて予測を行う．図 2 に示すように予測器の主要素であるクリティカルパスヒストリテーブル (Critical Path History Table: CPHT) は，命令のプログラムカウンタでインデックス付けされる飽和型アップ・ダウンカウンタから構成される．ある命令がクリティカルであると判断されると対応するカウンタがインクリメントされ，そうでない場合には対応するカウンタはデクリメントされる．ある命令に対応するカウンタの値が閾値を上回っているならその命令はクリティカルであると予測され，閾値を下回っているならその命令はクリティカルでないと予測される．カウンタの飽和値や予測時に用いられる閾値，カウンタのアップ幅，ダウン幅は予測器の実装に依存する．命令がクリティカルか否かを判断するためのヒューリスティックには，QOLD，ALOLD 等がある [3]．それぞれ，命令キューあるいはアクティブリストの先頭に位置する命令をクリティカルであるとする．命令がクリティカルが否かが予測できれば，その情報を利用してプロセッサの処理性能を向上することが可能である [2, 3, 4]．

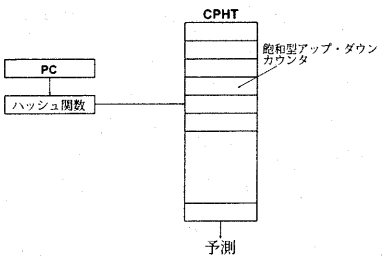

図 2: Tune 等のクリティカルパス予測器

我々は実行サイクル数を増やさずにプロセッサの低消費電力化を実現するため，プログラムの実行時間を決めるというクリティカルパスの特性に着目した [5]．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が，クリティカルパス上にない命令はクリティカルパス上の命令の開始を遅らせない限りプログラムの実行速度に影響を与えない．この点に着目して，レイテンシと電源電圧の異なる演算器を用意し，実行時間に影響を与えるクリティカルパス上の命令は高速かつ電力消費の大きな演算器で実行し，クリティカルパス上にない命令は低速かつ電力消費の小さな演算器で実行させる [5, 6, 7]．こうすることで，プログラム全体の実行時間を変化させることなくプロセッサの消費電力を下げることが可能となり，電源電圧を下げることによる動作速度低下という弊害の影響を隠蔽することが可能となる．

3 2 レベルクリティカルパス予測器

Tune 等の予測器は各命令ごとのローカルな履歴のみを利用している．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上が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．予測に用いるグローバルな命令の履歴によって GCPH 型，GBH 型，BOTH 型の 3 種類に分類する．以下にそれぞれの詳細を示す．

3.1 GCPH 型予測器

この方式では，Tune 等の CPHT 以外に新たにレジスタを追加する．追加するレジスタはシフトレジスタで，最近実行された各命令に対してそれがクリティカルであったか否かを表す 1 ビットの情報を記録する．命令が実行される度に 1 命令分シフトして最後尾に新たな情報を書き込む．命令ごと (local) の履歴ではなく命令を区別しない (global) 履歴を記録するのでこの情報をグローバルクリティカルパス履歴 (Global Critical Path History: GCPH) と呼ぶ．この方式を図 3 に示す．CPHT とは別に記録しておいた GCPH の情報を利用することによって，命令間の相関を利用して予測を行うことが可能となる．命令がクリティカルになるのは他の命令との依存関係が原因なので，命令間のクリティカルパスの相関を予測に利用することは有効であると思われる．

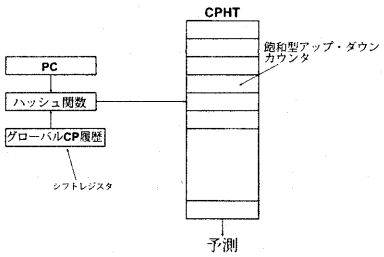

図 3: GCPH 型の構成

3.2 GBH 型予測器

プロセッサは分岐命令によって命令の並列実行性を大きく低下させられる．分岐命令の分岐先が決定しない限り分岐命令より先の実行アドレスが分からないからである．これを避けるため，分岐方向を予測する分岐予測器の研究が盛んに行われている [8]．その中の予測器の 1 つにグローバル履歴 2 レベル適用型分岐予測

器がある．この分岐予測器は分岐命令間の相関を分岐予測に利用するために GCHP と同様の手法で記録された分岐命令のグローバルな履歴を記録する．グローバル履歴 2 レベル適用型分岐予測器は予測器が記録できるグローバル分岐履歴が長ければ遠い分岐履歴との相関も分岐予測に反映されるため，分岐予測精度は高くなる．しかしグローバル分岐履歴が長くなると必然的に分岐予測器が大きくなる．このことにより，プロセッサ全体の消費電力量が増加する．このため低消費電力アーキテクチャではある程度分岐予測器の大きさを縮小せざるをえないが，このことは分岐予測精度の低下につながり命令の並列実行性が下がってしまう．つまり，分岐命令はクリティカルパスに大きな影響を持っていると考えられる．そこで我々は，分岐予測器が生成するグローバル分岐履歴をクリティカルパス予測に利用することを考えた．

図 4 に示すように分岐命令のグローバルな履歴と，フェッチされてきた命令のアドレスを用いて CPHT のインデックスを作成する．こうすることで，CPHT のインデックスに分岐命令との相関情報が反映されるのでクリティカルパス予測精度の向上が期待できる．

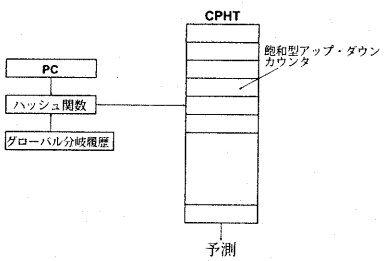

図 4: GBH 型の構成

3.3 BOTH 型予測器

この方式は，3.1 節で述べたクリティカルパス履歴と，3.2 節で述べたクリティカルパスに影響を与える分岐命令との相関を両方利用する．2 つの情報を用いることで，それぞれの予測器が持つ特徴を両方とも活用できる．このことはクリティカルパス予測に対してよい結果をもたらすと思われる．

図 5 にこの方式を示す．GCPH とグローバル分岐履歴の記録の仕方は以前述べたとおりである．予測器は GCPH とグローバル分岐履歴，命令アドレスの 3 つの情報を用いて CPHT のインデックスを作成する．

3.4 予測器更新のタイミング

予測をより確実なものにするために CPHT の内容は動的に更新されなければならない．Tune 等の予測器では CPHT の更新を命令コミットのタイミングで行っ

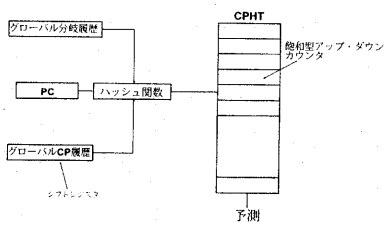

図 5: BOTH 型の構成

ているので，コミットされた命令の情報しか反映されない．したがって，早い時期にクリティカルパス上の命令だと判断できてもコミットされるまではその情報が予測に使用できず，仮にある命令の直後に同じ命令が再び現れても先行する命令がコミットされていないため後続の命令にはこの情報が適用できない．つまり，新たにクリティカルと判断された命令の情報が予測器に反映される前に同じ命令が出現した場合，期待通りの命令のスケジューリングが行われないためプログラム全体の実行時間に悪影響を与えかねない．我々はこの問題を解決するため，まだコミットされない命令に対しても予測器の更新を行うことを提案する．もし命令の完了後にクリティカルと判定されたなら，そのサイクルでその命令がコミットされていなくても対応する CPHT のエントリの更新を行う．

この方式では命令の持つクリティカルであるという情報が迅速に次の予測に利用される反面，クリティカルと判定された命令がコミットされるまで何度も CPHT の同一エントリを更新し続けてしまう恐れがある，これを避けるため一度 CPHT を更新した命令にはフラグをセットして CPHT を一度しか更新しないようにしなければならない．また，実行中の命令には分岐予測によって投機的に実行されている命令も含まれている．この投機実行中の命令はクリティカルパス上の命令と判断されて CPHT が更新された後その投機実行が失敗だったと判明することもある．この場合，予測に失敗した分岐命令とその後続命令は実行途中であってもその実行が破棄される．これと同様に CPHT の内容を信頼性の高いものに保つために実行が破棄された命令によって更新された CPHT のエントリも更新される前の状態に戻さなければならない．

以上の更新で常に新しい情報をもとに予測を行うことが可能となる．このことは予測精度に良い効果をもたらすと考えられる．

4 評価方法

シミュレーションに用いたプロセッサモデルとベンチマークプログラムについて説明し，どのような環境

表 1: プロセッサ構成

Fetch Bandwidth	8 instructions
Branch Predictor	1K-set 4-way set-associative BTB, 4K-entry 8-history-length gshare predictor, 64-entry return address stack, 6-cycle miss penalty, updated at commit stage
Insn. Windows	32-entry instruction queue, 32-entry load/store queue
Issue Width	8 instructions
Commit Width	8 instructions
Functional Units	6 Int, 3 FP, 4 Ld,St
Latency (total/issue)	iALU 1/1, iMUL 8/1, iDIV 32/1, fADD 4/1, fCMP 4/1, fCVT 4/1, fMUL 4/1, fDIV 32/1, fSQRT 32/1, Ld/St 2/1
Register Files	32 32-bit Int registers, 32 32-bit FP regsters
Insn. Cache	64KB, 2-way, 64B blocks, 18-cycle miss penalty
Data Cache	64KB, 2-way, 64B blocks, 4-port, write-back, non-blocking load, hit under miss, 18-cycle miss penalty
L2 Cache	unified, 1MB, 4-way, 64B blocks, 80-cycle miss penalty

で予測器を評価したかを述べる.

SimpleScalar ツールセット (version 3.0b)[9] のアウト・オブ・オーダ実行を行うシミュレータに，提案する各クリティカルパス予測器と低消費電力化アークテクチャを組み込んでシミュレータを作成した．評価するクリティカルパス予測器の各テーブルサイズは 2K, 4K, 8K, 32K, 64K とし，その飽和型アップ・ダウンカウンタは 6 ビットで，カウンタの更新値は命令がクリティカルな場合に 4, 6, 8 の 3 通りを，そうでない場合には −1 とした．予測の閾値は，命令がクリティカルな場合の更新値に一致させた．GCPH 型，BOTH 型の 2 レベル型予測器が利用する GCPH は過去 4, 6, 8 命令分とした．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，レイテンシと電源電圧がそれぞれ 2 : 1 の比となる高速な ALU と低速な ALU を 6/0, 1/5, 2/4, 3/3, 4/2, 1/5, 0/6 の全ての組み合わせで評価した．本稿では紙面の都合上 3/3 の場合のみを紹介する．プロセッサ構成の詳細を表 1 に示す.

命令セットは MIPS R10000 を拡張した SimpleScalar /PISA である．使用したベンチマークプログラムは SPEC 2000 CINT の中から選んだ 6 本で，それぞれの入力は表 2 に示す通りである．いずれのプログラムも，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.

表 2: ベンチマークプログラム

Benchmark	*input set*
164.gzip	input.compressed
175.vpr	net.in arch.in
176.gcc	cccp.i
197.parser	test.in
255.vortex	lendian.raw
256.bzip2	input.random

5 シミュレーション結果

前節で述べた様々な構成でシミュレーションしたうち，最もサイクル時間への影響が小さかったクリティカルパス履歴長が 8 で予測の閾値が 8 の予測器について結果を紹介する.

図 6〜9 にシミュレーション結果を示す．評価には実行サイクル数とエネルギー遅延積 (ED 積) を用いる．それぞれ左図に実行サイクル数を，右図に ED 積を示す．実行サイクル数についてはグラフが高い方が好ましく，ED 積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 `Tune` は Tune 等の予測器を，`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．`6fast/0slow` と `0fast/6slow` はクリティカルパス予測を利用しない場合を表す．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.

5.1 サイクル数と ED 積への影響

まず CPHT のテーブルサイズを 64K，予測器の更新のタイミングを命令のコミット時とし，QOLD を用いて命令のクリティカルを判定した場合の結果を図 6 に示す．クリティカルパス予測を利用しないで全て高速な ALU で実行した場合と比較して，どの予測器を用いた場合も平均で約 20%強実行サイクル数が増加している．2 レベル型予測器を Tune 等の予測器と比較すると，GBH 型と BOTH 型で，実行サイクル数がやや増加するものの ED 積が改善されている．一方，GCPH 型では逆の結果となっている．これは GCPH 型予測器がより多くの命令をクリティカルと予測したために，高速な ALU を積極的に使用していることが原因である．総じて 2 レベル型予測器と Tune 等の予測器の効果の違いは小さい.

5.2 クリティカル判定基準の影響

上記と同様の構成でクリティカル判定基準を ALOLD に変更した結果を，図 7 に示す．図 6 に比べさらに積極的に高速な ALU を使用しているため，全体的にサ

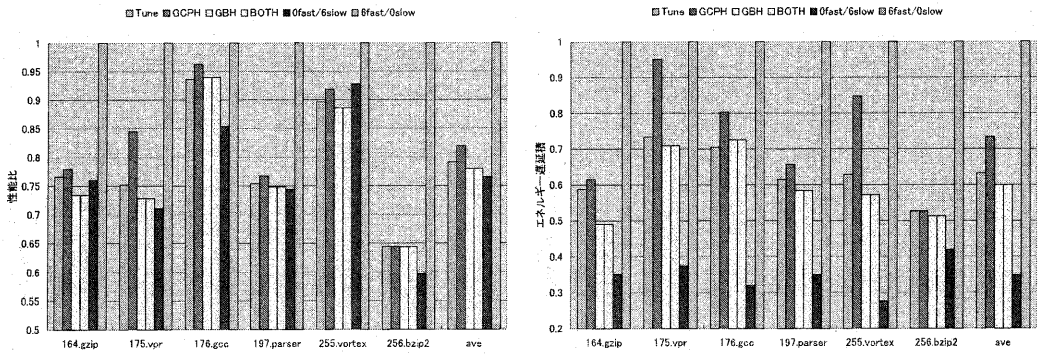

図 6: 64K エントリ, QOLD, コミット時更新

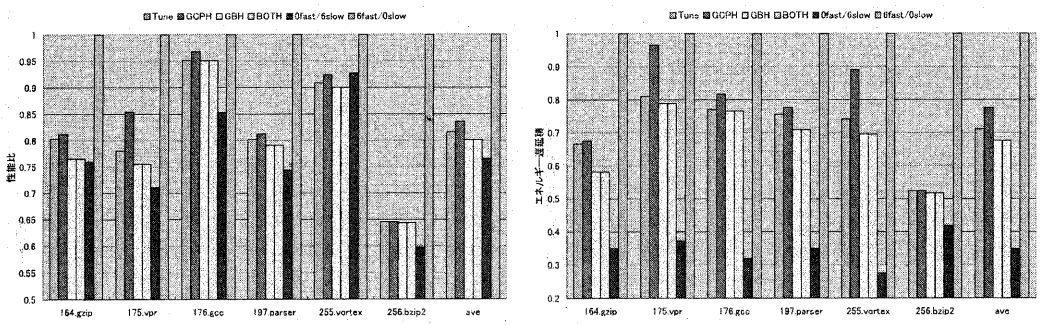

図 7: 64K エントリ, ALOLD, コミット時更新

イクル数の改善が見られるが，ED 積は増加している．サイクル数の改善とエネルギー利用効率の悪化の度合いを比較してみると，サイクル数は平均で約 2%の改善でエネルギー利用効率は平均で約 7%悪化している．このため，以後クリティカルの判定には QOLD を用いることにする．

5.3 予測器更新タイミングの影響

図 8 に予測器更新のタイミングを命令の実行時に行った場合の結果を示す．この結果は図 6 に比べ，実行サイクル数では最大で約 6%，平均で約 4%増加しているが，ED 積は平均で約 17%改善している．実行サイクル数の増加よりもエネルギー利用効率の改善の効果が大きいため，予測器の更新は命令の実行中に行った方がよいことがわかる．

5.4 テーブルサイズの影響

図 9 に CPHT のテーブルサイズを 2K に削減した場合の結果を示す．図 8 の場合と比べると実行サイクル数では平均で約 2%の改善を示しているが，ED 積は約 5%増加している．この原因としては，CPHT のテーブルサイズが小さくなったためにエントリの競合が起きてしまい予測器が命令を多くクリティカルと予測してしまうことが挙げられる．この影響は特に 2 レベル型予測器で大きい．

以上のように，現段階では実行サイクル数が短いと ED 積の面で良い結果が得られていない．この理由として，いずれの予測器も，実行サイクル数を改善しようとすると，高速な ALU を積極的に利用する傾向にあることが挙げられる．つまり，クリティカルパス予測器の予測精度が低いために，クリティカルではない命令が高速な ALU で実行されているためである．

6 まとめと今後の課題

今回の評価では，我々が提案した予測器と Tune 等の予測器との明確な違いを明らかにはできなかった．サイクル数と ED 積での評価結果はどの予測器でも傾

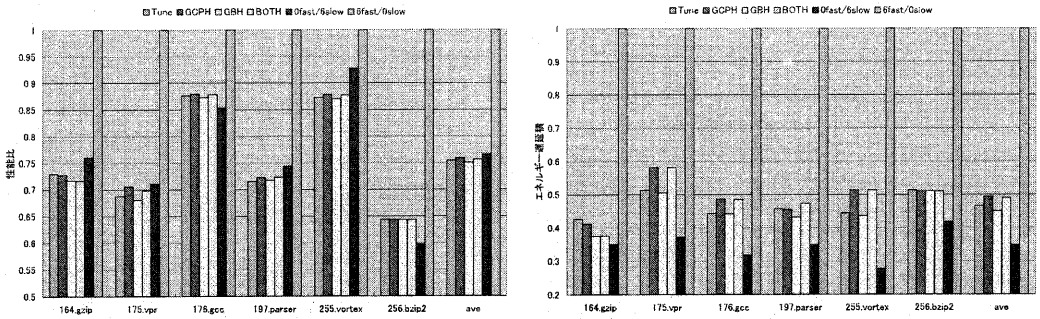

図 8: 64K エントリ, QOLD, 実行時更新

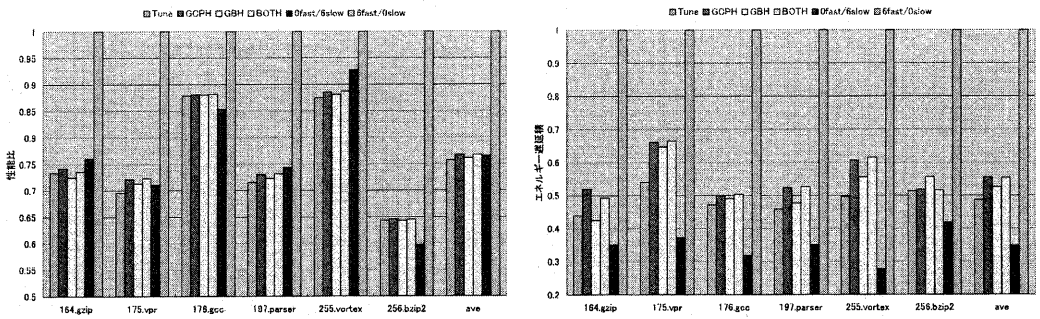

図 9: 2K エントリ, QOLD, 実行時更新

向は同じであった．クリティカル判定基準，予測器更新タイミング，予測器のエントリ数についての調査では，QOLD で実行時に更新する大容量の予測器が好ましいという結果となった．今後は予測の精度を向上させるために命令がクリティカルかどうかの判断をするヒューリスティックの研究や，命令中のクリティカルパスの分布，出現頻度を調査してどこまでサイクル数を増加させることなく低消費電力化が可能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 基盤研究 B(2) 展開 課題番号 13558030 の援助によるものです．

参考文献

[1] マイク・ジョンソン: “スーパースカラプロセッサ マイクロプロセッサ設計における定量的アプローチ”, 日経 BP, 1994 年.

[2] 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.

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

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

[5] T. Sato, T. Koushiro, A. Chiyonobu, I. Arita: “Power and Performance Fitting in Nanometer Design”, 5th International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, January 2002.

[6] R. Pyreddy, G. Tyson: “Evaluating Design Tradeoffs in Dual Pipelines”, Workshop on Complexity-Effective Design, June 2001.

[7] J. S. Seng, E. S. Tune, D. M. Tullsen: “Reducing Power with Dynamic Critical Path Information”, 34th International Symposium on Microarchitecture, December 2001.

[8] 斎藤史子, 山名早人: “投機的実行に関する最新技術動向”, 情処研報 2001-ARC-145-11, 2001 年 11 月.

[9] 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.