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

2008－ARC－179 (24)
2008／8／7

プログラムの繰り返し構造に着目した動的なヘルパースレッディング

安藤 徹[†] 塩谷亮太[†]
五島正裕[†] 坂井修一[†]

マルチスレッド実行プロセッサで単一スレッドのプログラムの実行を高速化する技術としてヘルパースレッディングがある．ヘルパースレッディングではメインスレッドに先行して実行を行うヘルパースレッドにより，ロードのプリフェッチや分岐予測の精度を向上させる．本研究ではプログラムのループ構造に着目して，動的にヘルパースレッドを作成するアーキテクチャを提案する．ヘルパースレッドの実行結果を分岐プレディシジョンの補助として利用することで，分岐先の早期決定を実現する．

Dynamic Helper Threading Based on Loop Structure

TORU ANDO,[†] RYOTA SHIOYA,[†] MASAHIRO GOSHIMA[†]
and SHUICHI SAKAI [†]

Helper threading is one of techniques that increases performance of a single thread program on a multithread processor. Helper threading prefetches data and improves the accuracy of branch predictions by results of pre-execution. This paper proposes the architecture that constructs helper threads dynamically based on loop structure. The early decision of branch instructions comes true by using the result of helper thread as the help of branch predecision.

1. はじめに

近年のプロセッサではパイプラインが深化する傾向にあり，分岐予測ミスペナルティが大きくなってきている．分岐予測ミスペナルティを削減する手法として分岐予測の研究が盛んに行われている．しかし，従来のように過去の分岐パターンや実行パス情報を用いる分岐予測では，分岐結果の規則性が低い場合には予測を行うことが困難となる．

そのような分岐命令に対して分岐予測を用いずに処理する分岐プレディシジョンという手法がある．分岐プレディシジョンはレジスタ間接分岐ターゲットフォワーディング[1)]の考え方を条件分岐にまで一般化した手法で，分岐命令の依存元の命令の実行終了時に分岐命令を早期実行し，その結果をフロントエンドにフォワーディングする．その結果を元にフェッチすることによって分岐予測を用いずに分岐先を決定することができる．しかし，フォワーディングを成功させるには分岐命令とその依存元の命令が十分離れていることが必要であるといった問題がある．

† 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

また，違うアプローチとしてヘルパースレッディングという技術が挙げられる．ヘルパースレッディングは Simultaneous Multithreading(SMT) やチップマルチプロセッサ (Chip Multiprocessor, CMP) のようなマルチスレッド実行を行うプロセッサにおいて，逐次の単一スレッドプログラムの実行を高速化する技術である．通常 SMT や CMP は複数のスレッドを持つ処理において，複数のスレッドを並列に処理することで高い実行効率を得ている．単一のスレッドのプログラムを実行するような状況ではマルチスレッドの実行資源を有効に利用することはできない．ヘルパースレッディングは単一スレッドのプログラムからヘルパースレッドを作成し，メインスレッドと並列に実行することでマルチスレッドの実行資源を有効に活用する．ヘルパースレッディングでは元のプログラムの命令の一部を抜き出し，メインスレッドに対して先行して実行することによって得られる結果を用いてメインスレッドの実行を高速化することが一般的である．つまり，分岐命令の先行実行を行い，その結果を分岐予測に用いることで予測精度を向上させたり，分岐予測ミスを早期に発見することでペナルティが削減できる．

また，ヘルパースレッディングでは分岐命令に対してだけではなく，ロード命令の先行実行を行うことで

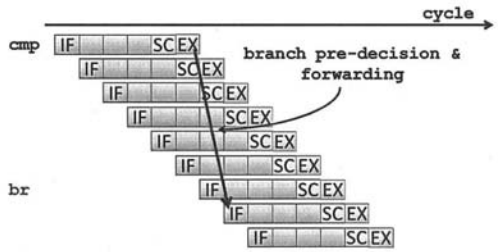

図 1 分岐プレディシジョンの動作

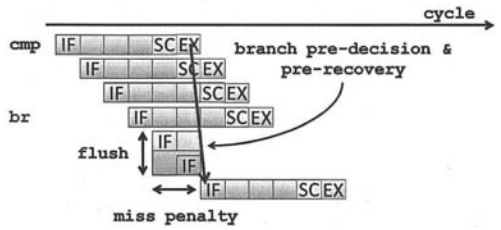

図 2 早期リカバリ

ロードのプリフェッチをし，メモリアクセスのレイテンシを隠蔽することも行われている．

本研究では，プログラムの繰り返し構造に着目し，値予測を用いることで早いタイミングでヘルパースレッドを実行開始できるヘルパースレッディングを提案する．提案手法ではヘルパースレッドの実行結果を，分岐プレディシジョンの補助として用いる．先行実行の結果を用いることで通常の分岐プレディシジョンではフォワーディングが間に合わない場面に対しても効果が得られると考えられる．

本稿の構成は以下の通りである．まず，2 章で分岐プレディシジョンの概要とその問題点について説明する．3 章ではヘルパースレッドを先行スレッドとして扱うヘルパースレッディングについて説明する．4 章で提案手法であるプログラムの繰り返し構造に着目したヘルパースレッディングについて説明し，5 章でまとめと今後の課題を述べる．

2. 分岐プレディシジョン

豊島らは間接分岐に対し，間接分岐命令とそれが依存しているロード命令の間に存在する命令によって遅延を隠蔽する手法として分岐ターゲット・フォワーディング[1] と呼ばれる手法を提案している．この分岐ターゲット・フォワーディングは間接分岐だけでなく，一般の条件分岐に拡張して考えることが可能であり，それを行ったのが分岐プレディシジョンである．

分岐プレディシジョンではある分岐命令 (以下，Consumer 命令と呼ぶ) がフェッチされる際に，Consumer 命令の依存している命令 (以下，Producer 命令と呼ぶ) の実行が終了していれば，その実行結果から Consumer 命令の早期実行を行い，分岐先をフォワーディングする．これにより，Consumer 命令の後続の命令は分岐予測ではなく，フォワーディングされた結果を用いてフェッチ先を決定することができる．

図 1 に分岐プレディシジョンの動作を示す．同図では，比較命令 cmp が Producer 命令，条件分岐命令 br が Consumer 命令となる．この図から分かるようにフォワーディングを行うためには，あらかじめコンパイラ等によって，コード上で静的に Producer 命令と Consumer 命令の距離を話しておく必要がある．Producer 命令と Consumer 命令の距離を十分に離れておらず，フォワーディングが行えない場合には，早期リカバリを行うことで分岐予測ミスペナルティの軽減を図る．

図 2 に早期リカバリの様子を示す．図 1 と同様に，比較命令 cmp が Producer 命令，条件分岐命令 br が Consumer 命令となる．図 2 では，Producer 命令と Consumer 命令が十分に離れていないため，Producer 命令の実行終了が Consumer 命令のフェッチに間に合わない．このため，Consumer 命令の後続の命令は，分岐予測によってフェッチを行う．Producer 命令の実行が終了した場合，Consumer 命令である分岐命令の早期実行を行う．この結果が Consumer 命令の分岐予測結果と異なった場合，その時点でリカバリを行う．

分岐プレディシジョンの問題点

前述したように分岐プレディシジョンでは Producer 命令と Consumer 命令が十分に離れていないとフォワーディングが行えないといった問題がある．Producer 命令と Consumer 命令が十分に離れていない場合には早期リカバリを行うが，フォワーディングに比べ効果は薄い．フォワーディングを行うために必要なコード上の距離は，おおよそパイプライン段数とフェッチ幅の積によって決定され，通常 20 から 40 命令程度となる．一般に，コンパイラによって依存関係にある特定の命令をこのように大きく離してスケジューリングすることは困難であり，フォワーディングできる場面は限られてしまう．

3. ヘルパースレッディング

従来からマルチスレッドプロセッサで単一スレッドのプログラムの実行を高速化する手法としてヘルパースレッディングの研究が行われている．ヘルパースレッディングの研究例の多くはメインスレッドの命令の一部をヘルパースレッドとしてメインスレッドに先行し

て実行することで高速化を実現している．本研究が提案するヘルパースレッディングもヘルパースレッドをメインスレッドの先行スレッドとして用いる．

本章ではヘルパースレッドをメインスレッドに対する先行スレッドとして用いるヘルパースレッディングについて述べる．

3.1 概　　要

先行実行命令列の作成

ヘルパースレッドの実行はメインスレッドの実行より先行させるためには，ヘルパースレッドで先行実行する命令はメインスレッドで実行される命令よりも少なくしなければならない．そのため，ヘルパースレッドで実行を行う先行実行命令列の作成がヘルパースレッディングの性能を決める重要な点の一つとなる．

先行実行を行う命令の数は，メインスレッドの実行を妨げないという点では少なければ少ないほどよく，理想的には先行実行の結果を用いることでメインスレッドの実行が高速化される命令のみ先行実行を行えられればよい．しかし，対象の命令だけを先行実行することは不可能で，メインスレッドの実行に対してほとんど先行させることはできないため，対象の命令の依存元の命令を加えて複数の命令を先行実行することが必要となる．先行実行命令列の作成ではメインスレッドに対する先行度と先行実行命令の数との兼ね合いが問題になってくる．

また，作成するタイミングとしては実行前にコンパイラなどを用いて作成しておく方法と実行時の情報を用いて動的に作成する方法が考えられる．前者ではプロファイリング情報を基に複雑な処理や解析が行えるため効率的な命令列の選択ができると考えられるが，再コンパイルの必要があるといった欠点がある．後者はあまり複雑な処理を行うことはできないが，入力データなど実行時の情報を用いることができるという利点がある．

ヘルパースレッドの実行開始タイミング

ヘルパースレッドの実行開始のタイミングは先行実行命令列の作成と合わせてヘルパースレッドの先行度に大きく影響してくる．一般にはヘルパースレッドの実行に必要なデータを与える命令をトリガとして実行開始されることが多い．

実行結果の利用

ヘルパースレッドの実行結果の利用法としては分岐予測器の補助，ロードのプリフェッチが一般的である．

分岐予測は過去の分岐パターンや実行パス情報を用いて行われるため，それらに相関のない分岐命令は分岐予測が困難である．そこでヘルパースレッドで実際に実行した結果を用いて予測することで予測精度の向上を図っている．

ロードのプリフェッチでは実際に実行を行うことで，従来のプリフェッチャでは困難であった不規則パターンのメモリアクセスに対応できると考えられる．

いずれにしてもヘルパースレッドの実行はメインスレッドの実行にたいして十分前に終了している必要がある．そのため前述したように先行実行命令列の作成方法とヘルパースレッドの実行開始タイミングが重要となってくる．

3.2 従 来 研 究

本項では実際の過去の研究例を紹介する．

Simultaneous Subordinate Microthreading

Chappell らの提案している Simultaneous Subordinate Microthreading (SSMT)[2] ではヘルパースレッドで実行される先行実行命令列をコンパイラによって作成する．コンパイラによって挿入される SPAWN 命令によってヘルパースレッドの実行が開始される．先行スレッドによって分岐予測の精度向上，ロードのプリフェッチ，キャッシュ管理の効果が得られると述べられており，その中で分岐に対する評価が行われている．プロファイリング情報によってコンパイル時に gshare 分岐予測器では予測精度が低くなってしまう，他の分岐命令との相関がない分岐命令に対してヘルパースレッドを作成する．ヘルパースレッドの実行によって得られた分岐履歴を利用し，分岐予測を行うことで分岐予測精度の向上を図っている．

Slipstream Processor

Slipstream Processor[3),4)] では 2 つのスレッドを並列に実行する．まず，実行開始時には 2 つとも元のプログラムを実行する．実行していく中で，片方のスレッドは動的に制御フローに関係のない命令を投機的に取り除いていく．そうすることで命令を取り除くスレッドはもう一方のスレッドよりも先行して実行が行われる．Slipstream Processor は通常の分岐予測器の他に IR-predictor という gshare 分岐予測器を元にした分岐予測器を持つ．IR-predictor のエントリは 1 つの基本ブロックに割り当てられるため，基本ブロックごとの分岐予測器といえる．先行スレッドの分岐結果が IR-predictor に入力され，それをメインスレッドが分岐予測器として利用することで分岐予測精度が向上する．また，先行スレッドで生成された値を用いてメインスレッドで値予測を行う．測定結果ではほとんどのプログラムで，分岐予測の精度の向上よりも，この値予測によって性能が向上していると述べられている．

Speculative Data-Driven Multithreading

Roth らは Speculative Data-Driven Multithreading (DDMT)[5] を提案している．DDMT はメインスレッドと最大 16 個のヘルパースレッドを並列に実行する．DDMT では頻繁に分岐予測ミスをする分岐命令とキャッシュ・ミスをするロード命令に対してヘルパースレッドを作成する．先行実行命令列は，まずプロファイリング情報を元にそれらの命令を特定し，次にその命令の依存している命令を追加していくことで作成される．ヘルパースレッドは作成された先行実行命令列の最初の命令のソースオペランドの値を生成する命令がリネームされた時点をトリガとして実行開始される．先行実行で得られた結果はレジスタに保存され，レジスタ統合[6] という技術でメインスレッドや他のヘルパースレッドで再利用される．レジスタ統合では，ある命令がリネームされる時に先行スレッドに同一の命令がないかを検証する．同一の命令があり，実行を完了していた場合には，その命令を再実行することなく実行を終了する．分岐命令の場合には，ヘルパースレッドでの分岐結果がリネーム時に分かるため，分岐予測が間違っていた場合にはリカバリを行い，分岐予測ミス・ペナルティを削減することができる．

Dynamic Speculative Precomputation

Collins らの提案している Dynamic Speculative Precomputation[7] では，ヘルパースレッドによってキャッシュ・ミスを頻繁に起こすロード命令を先行実行し，メモリアクセスのレイテンシを隠ぺいする．そのようなロード命令をハードウェアで動的に特定し，それが依存する命令を集めて p-slice と呼ばれる命令列を作成する．p-slice はその最初の命令のソースオペランドの値を与える命令をトリガとして実行が開始される．

3.3 従来研究のまとめ

先行実行命令列の作成

SSMT，DDMT はプログラムの実行前にコンパイラやプロファイリング情報を利用してヘルパースレッドで実行する先行実行命令列を作成する．一方，Slipstream Processor，Dynamic Speculative Precomputation では実行時にヘルパースレッドを作成する．

先行実行命令列の作成方法は SSMT，DDMT，Dynamic Speculative Precomputation が同じような形をとっており，まず先行実行するべき命令と特定し，その命令の依存元の命令を加えていくことで作成する．この方法はどこまでの命令を先行実行命令列に加えるかが重要な点になる．

Slipstream Processor は他の手法とは違っていて，始めは全く同じプログラムを実行して，実行中に片方のスレッドから命令を取り除いていく．この方法では取り除く命令に限界があり，ヘルパースレッドで実行する命令の数が多くなってしまうという問題点がある．

ヘルパースレッドの実行開始タイミング

従来の研究例ではコンパイラよって実行開始命令が挿入されているか，先行実行命令列の最初の命令のソースオペランドの値を与えるメインスレッドの命令がトリガとなって実行を開始する．そのため，ヘルパースレッドの実行開始のタイミングは限定されており，メインスレッドに対して十分先行して実行完了するように先行実行命令列を作成しなければならない．メインスレッドに対してより先行させようとすると，先行実行命令の数を多くしなければならなくなり，ヘルパースレッドの実行に資源をより割かなければならなくなるといった欠点が考えられる．

ヘルパースレッドの実行結果の利用

SSMT と Slipstream Processor ではヘルパースレッドでの先行実行の結果を分岐予測に用いることで分岐予測の精度向上を図っている．Dynamic Speculative Precomputation ではロードのプリフェッチによってメモリアクセスのレイテンシを隠蔽している．DDMT は分岐命令に対しては分岐予測ミスを早期に発見することで分岐予測ミスペナルティを削減している．ロード命令に対してはプリフェッチの効果がある．また，ヘルパースレッドで実行された命令の結果は他のスレッドで再利用することで同じ命令を再実行しない．しかし，このためにハードウェアが複雑になってしまっている．

4. 繰り返し構造に着目したヘルパースレッディング

本研究で提案するヘルパースレッディングは，プログラムの繰り返し構造に着目し，値予測を利用してヘルパースレッドの作成と実行を行う．ヘルパースレッドは実行時に動的に作成，実行開始され，キャッシュミスをするロード命令と分岐予測ミスをする分岐命令をメインスレッドに対して先行して実行する．ロード命令の先行実行では関連研究と同様にプリフェッチの効果があると考えられる．分岐命令の先行実行では実行結果を分岐プレディシジョンの補助として使うことにより，分岐予測ミスペナルティを削減する．

本章では，値予測を用いたヘルパースレッドの作成と実行と，分岐プレディシジョンの補助について説明する．

4.1 ヘルパースレッドの作成と実行

4.1.1 先行実行命令列の作成

先行実行命令列の作成は分岐予測ミスをする分岐命令やキャッシュミスをするロード命令を検出するところから始まる．これは実行時に分岐予測ミスとキャッシュミスをカウントする表を用意することで実現する．

先行実行を行うべき命令が検出した後，従来研究と同様にその命令の依存元の命令を先行実行命令列に加えていくことで作成していく．命令を加えていく時にその命令が値予測可能かどうかを調べ，可能であればその命令の依存元の命令を加えないという点が従来と違っている．具体的な手順は以下のようになる．

(1) 分岐予測ミスする分岐命令，またはキャッシュ・ミスをするロード命令を検出する．
(2) (1) で検出した命令のソースレジスタを探索するソースレジスタ群に加える．
(3) 検出した命令から上に見ていき，探索するソースレジスタ群をデスティネーションとする命令を探す．
(4) (3) で見つけた命令のデスティネーションレジスタを探索するソースレジスタ群から外す
(5) (3) で見つけた命令が値予測可能でなければ，そのソースレジスタを探索するソースレジスタ群に加える．
(6) 探索するソースレジスタ群の数が 0 になるまで，3-5 を繰り返す．

(3) においてループを一回りして，手順 1 で特定した命令まで達したときに探索するソースレジスタ群の数が 0 にならなければ，ヘルパースレッドの生成が失敗したとする．

従来との違いを図 3，図 4 を例にして説明する．どちらの図も左側の命令列がメインスレッドで実行される命令で，右側の命令列がヘルパースレッドで実行される命令である．まず，このループで I8 の分岐命令が頻繁に分岐予測を失敗する命令であると分かったとする．従来のヘルパースレッドではこの分岐命令の依存元の命令を見ていくと I7 が見つかる．さらに上に見ていくと I4，I2，I1 という順に依存元の命令が見つかる．ここでは I2 までを先行実行命令列に加えて I1 をヘルパースレッドの実行のトリガとなる命令とする．

値予測を用いた手法においても I8 の分岐命令の依存元の命令を探索していく．ここでは I1 と I4 の命令が値予測可能であるとすると依存元の命令は I7，I4 と順に見つかるが，I4 は値予測可能であるため I4 の依存元の命令である I2 は先行実行命令列に加えない．

この例においては従来のヘルパースレッドではメインスレッドが I2 から I8 まで 7 命令のところ，ヘルパースレッドは 4 命令となり 3 命令分先行して実行を行うことが期待できる．一方，値予測を用いた場合には値予測を用いて好きなタイミングで実行を開始できるため，従来よりも先行して実行を行うことができると考えられる．

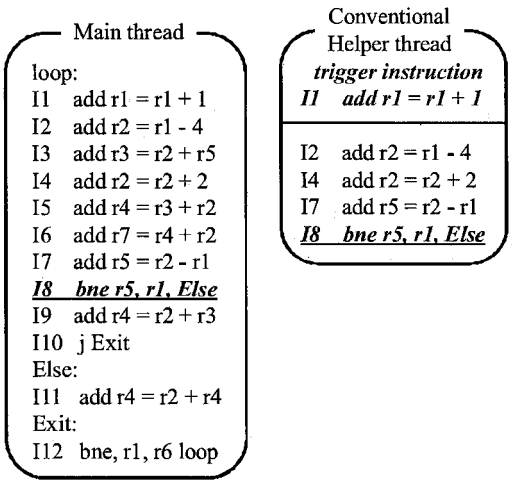

図 3 従来のヘルパースレッドの例

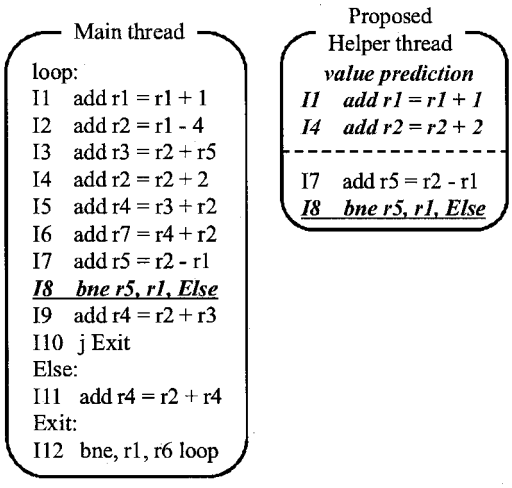

図 4 値予測を用いたヘルパースレッドの例

4.1.2 ヘルパースレッドの実行開始タイミング

従来研究では先行実行命令列のはじめの命令が依存している命令がトリガとなってヘルパースレッドが実行開始されることが多い．そのため，メインスレッドに対してどの程度先行できるかはある程度限定されてしまう．

本研究のヘルパースレッドは4.1.1項で述べたように値予測を前提として先行実行命令列を作成するので，メインスレッドのある特定の命令をトリガとする必要がない．先行実行命令列を作成した時点で対象の分岐命令やロード命令はメインスレッドで近い将来に再度実行されると予想して，ヘルパースレッドの実行を開始する．これによってヘルパースレッドの実行はメインスレッドに対して十分先行して完了することが期待できる．

4.2 分岐プレディシジョンの補助

2章で述べたように分岐プレディシジョンでは，分岐命令とそれが依存している命令が十分離れていないと効果が薄い．そこでヘルパースレッドの実行結果を分岐プレディシジョンの補助として利用することを考える．4.1節で説明したように提案するヘルパースレッディングではヘルパースレッドの実行に値予測を用いることによって，メインスレッドと独立に実行開始できる．したがって，ヘルパースレッドの分岐命令の実行がメインスレッドの分岐命令のフェッチよりも前に完了するようにヘルパースレッドの実行を開始することが可能である．メインスレッドではヘルパースレッドの実行結果を利用することによって，より通る確率の高いパスをフェッチすることが期待できる．

分岐プレディシジョンでは実際の実行結果をフォワーディングするため，対象の分岐命令のフェッチに間に合えば確定したパスをフェッチすることができる．一方，ヘルパースレッドの結果を利用する場合にはヘルパースレッドの実行に値予測を用いているため確定したパスをフェッチすることにはならない．そのため，従来の分岐予測器による予測精度よりも，その分岐命令が依存している命令の値予測の予測精度が高い場合に効果があると考えられる．

5. まとめと今後の課題

本研究ではプログラムの繰り返し構造に着目してヘルパースレッドを生成するアーキテクチャを提案した．ヘルパースレッドはループ中の分岐予測ミスをする分岐命令とキャッシュ・ミスをするロード命令に対して生成され，その実行に値予測を利用する．それによってメインスレッドとは独立に実行を開始することができ，分岐先の早期決定やロードのプリフェッチが実現される．

今後は，本研究室で開発したプロセッサ・シミュレータ「鬼斬弐」[8]に提案アーキテクチャを実装し，評価を行う．

謝辞 本論文の研究の一部は，文部科学省 科学研究費補助金 No. 20300015 による．

参 考 文 献

1) 豊島隆志, 入江英嗣, 五島正裕, 坂井修一: レジスタ間接分岐ターゲットフォワーディング, 先進的計算基盤システムシンポジウム (SACSIS), Vol.2006, No. 5, pp. 325–332 (2006).
2) Chappell, R., Stark, J., S. Kim, S. R. and Patt, Y.: Simultaneous Subordinate Microthreading (SSMT), *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pp. 186–195 (1999).
3) Purser, Z., Sundaramoorthy, K. and Rotenberg, E.: A Study of Slipstream Processors, *Proceedings of the 33rd Annual International Symposium on Microarchitecture* (2000).
4) Sundaramoorthy, K., Purser, Z. and Rotenberg, E.: Slipstream Processors: Improving both Performance and Fault Tolerance, *Proceedings of Architecture Support for Programming Language and Operating Systems* (2000).
5) Roth, A. and Sohi, G.: Speculative Data-Driven Multithreading, *Proceeding of the 7th International Symposium on High-Performance Computer Architecture* (2001).
6) Roth, A. and Sohi, G.: Register Integration: A Simple and Efficient Implementation of Squash Re-Use, *Proceedings of the 33rd Annual International Symposium on Microarchitecture* (2000).
7) Collins, J., Tullsen, D., Wang, H., Lee, Y., Lavery, D., Shen, J. and Hughes, C.: Speculative precomutation: Long-range prefetching of delinquent loads, *Proceedings of the 28th Annual International Symposium on Computer Architecture* (2001).
8) 渡辺憲一, 一林宏憲, 五島正裕, 坂井修一: プロセッサ・シミュレータ「鬼斬」の設計, 先進的計算基盤システムシンポジウム SACSIS 2007 (ポスター), pp. 194–195 (2007).