計算機アーキテクチャ 141－9
（2001．1．26）

# 動的情報収集機構および投機実行支援機構を備えたチップマルチプロセッサに関する検討

小池汎平

電子技術総合研究所

スレッドレベル投機実行の支援機構を備えたチップマルチプロセッサが注目を集めている．本稿では，このタイプのチップマルチプロセッサとして，投機実行の支援するハードウェアに加えて，プログラム実行情報の収集を行なうハードウェアと実行時支援ソフトウェアを追加したシステムアーキテクチャを提案し，投機実行支援機構，プログラム実行情報収集機構，支援ソフトウェアそれぞれに必要とされる機能についての検討を行なう．また，このようなシステムの実証実験に用いるために，今回我々が開発し利用しようとしている研究ツールについての紹介も行なう．

# On Chip-Multiprocessors with Support Mechanisms for Execution Profiling and Speculative Execution

Hanpei KOIKE

Electrotechnical Laboratory

Chip-Multiprocessors (CMPs) with thread-level speculative execution support mechanism have attracted our current interest. In this paper, the author proposes a novel system architecture for this type of CMP, which includes additional support mechanisms for program execution profiling and runtime support software. The required functions for the speculative execution mechanism, the program execution profiling mechanism and the runtime support software are discussed. The author also introduces a novel tool for computer hardware resarch, which is planned to be utilized in the project.

## 1 はじめに

シングルチップ上にビリオントランジスタを集積して高性能プロセッサを構築することが可能となった現在，プロセッサアーキテクチャ構成法に関する研究のトレンドは大きく変わり，既存設計を最大限利用しつつさらなる性能向上のために比較的独立した複数の支援ハードウェア群を付加していくという方法論に基づいたアーキテクチャの設計が重要視されてきている．これまでに高性能プロセッサハードウェアの構築に費やされてきた多大な努力（チップの設計からソフトウェア環境の整備まで）を継承することなく，ゼロから新たなアーキテクチャを設計し直すことを繰り返すことは得策ではない，との見方が強いのが大きな理由である．このようなコンテクストにおいて，シングルチップ上に複数の高性能プロセッサコアを配置し結合する「チップマルチプロセッサ」が現実的かつ有望な解と考えられており，各所で様々な研究開発が活発に進められてきている [1, 2, 3, 4]．

チップマルチプロセッサの利用法として，並列化の比較的容易なプログラムの実行やタスクレベルの並列性の利用など，従来の SMP の利用法の延長としての，処理スループットを重視しつつ使用する方法が，少なくとも出発点として考えられている．しかし，従来のプロセッサ性能向上技術（クロック高速化，命令レベル並列性の向上）で得られてきた利点をスケーラブルに引き継ぐためには，単一プログラムの性能向上を，出来る限り広い応用対象に対して，しかもできる限り使用者に透過的な形で提供できるよう，新たな技術開発が望まれる．

チップマルチプロセッサにおいて，単一プログラムの実行における性能向上を引き出すためには，そのプログラムからスレッドレベル並列性を抽出し，プログラムを再構成することが必要である．マルチスレッド実行可能なプログラムの開発方法として最も原始的なやり方は，プログラマ自身の手による明示的な並列プログラミングであったが，この手法ではソフトウェア開発コストの上昇等をまねき，チップマルチプロセッサの利用範囲を大幅に限定してしまうことは，現在の SMP の利用形態の現実から容易に想像することができる．

逐次プログラムの静的解析によって並列性を抽出し，並列プログラムを自動生成する並列化コンパイラは，このような問題点に対する有力な解決策とみなされている．しかし静的解析に頼るコンパイラの問題点として，並列化の対象が並列化可能であると静的に判断できる場合に限定される，プログラムのグローバルな構造を考慮した並列化の技術が未開拓である等の問題点が残されており，並列化コンパイラ技術の延長線上において，

適用範囲の拡大とより一層の性能向上を目指すことが望まれている.

静的解析で得られる情報のみに頼るコンパイラの問題点を解消するための提案として, 動的実行情報のプロファイリング結果を最適化のための情報としてコンパイラへとフィードバックするなど, プログラムの実行特性をより加味した並列化手法が検討されてきているが, なかでも, スレッドレベルの投機実行によってプログラムの並列性を最大限抽出することによって実行性能の最大限の向上を目指すマイクロアーキテクチャが注目を集めている [5]. そして, 投機実行支援ハードウェアを備えたチップマルチプロセッサの開発事例も報告されはじめている [1, 2].

投機実行はプログラムの確率的振る舞いを利用した動的な最適化技法であり, 高い確率で起こることが予測される事象を仮定した並列化などの最適化処理のアグレッシブな適用と, プログラム実行時における予測事象の確認作業 / 予測がはずれた場合の修復作業とを組み合わせたものであると考えることができる [6]. このため, 投機実行を効率良く実現するためには, プログラムの確率的振る舞いの把握が重要となる.

このような考えから, 我々は, チップマルチプロセッサを用いてスレッドレベル投機実行を効率的に行うためには, 従来提案されてきたスレッドレベル投機実行向けチップマルチプロセッサで重視されている

- 投機実行支援ハードウェア

のみならず

- 実行統計情報を効果的に収集するハードウェア

および

- 得られた実行特性情報を利用して, プログラムの実行前ないし実行中にプログラムの静的ないし動的な最適化をはかる支援ソフトウェア

とが連携する複合システムとしての総合的な設計を行うことが, 成功への鍵を握っていると考えている.

そこで, 本稿では, 単一プログラムからスレッドレベル並列性を最大限に引き出して効率良く実行させるための「動的情報収集機構および投機実行支援機構を備えたチップマルチプロセッサ」のアーキテクチャおよび支援ソフトウェアの機能として, どのようなものが必要とされるかについての検討を行なう. また, 本稿後半では, これらの検討に基づいて我々の研究グループが進めているチップマルチプロセッサアーキテクチャ研究開発において我々が採用している, 新しい計算機アーキテクチャ実証的研究のための研究ツールについても紹介する.

## 2 対象とするチップマルチプロセッサシステムの概要

今回我々が開発を目指しているチップマルチプロセッサシステムは, 図1のように, 4台程度のプロセッサコアを結合した共有メモリシステムを核とし, これらに, 投機実行支援ハードウェアと, プログラム実行情報収集ハードウェアを付加したものである.

投機実行支援ハードウェアは, スレッドレベル投機実行を行なう際に大きなオーバヘッドとなることの予想される [11]

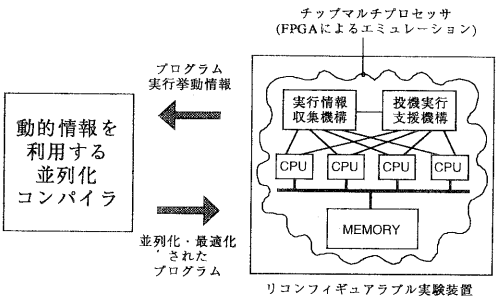


図 1: 実行情報収集機構, 投機実行支援機構, 支援ソフトウェアが連携するチップマルチプロセッサ

- 投機的メモリ操作
- 投機スレッド管理

などを効率的に支援するものである.

いっぽう, プログラム実行情報収集ハードウェアは

- 制御フローのプロファイリング
- データ値のプロファイリング
- データ依存関係のプロファイリング

を行ない, プログラム最適化に有用な動的実行情報の効果的な収集を, 実行時支援ソフトウェアとの連携のもとに行ない, コンパイラへとフォードバックする.

並列化コンパイラは, プログラム実行情報収集ハードウェアの収集した情報のフィードバックを受けて, 最適なスレッドレベル投機実行を行なうコードを生成し, チップマルチプロセッサに提供する. 当初は, コンパイラとして, マルチプロセッサ上のプログラム実行と切り放された, オフライン型のものの実現を目指すが, 今後, 実行時支援ソフトウェアと連携したオンライン型のコンパイラの実現の検討も行なう予定である.

また, 上記システムのハードウェア部は, 後述するFPGA を用いた実験装置上で実現し, 本システムの実証実験に十分なだけの処理速度と実行可能プログラムの規模を達成することを目指している.

以下の各節では

- 投機実行支援ハードウェア
- 実行情報収集ハードウェア
- 実行時支援ソフトウェア

に必要とされる機能についての検討を順次行なっていく.

## 3 投機実行支援機構の検討

### 3.1 投機的メモリアクセスの支援機構

図2に示されるようなスレッドレベルの投機実行を行う場合に, 投機スレッドにおいて実行されるメモリア

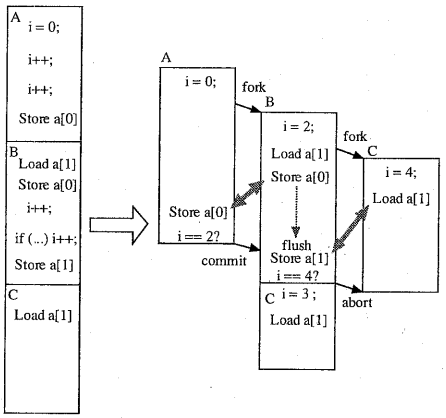


図 2: スレッドレベル投機実行

クセスは, 投機的なメモリアクセスであり, 特別な取り扱いをする必要がある. これをオーバヘッドとなることなく実現するために適切なハードウェア支援が必要となる. 投機的なメモリアクセスにおいて考慮しなければならないのは以下の点である [11].

- 同一メモリロケーションに対するストアよりも前に, 本来ならあとから実行されるロードが実行されたことによって依存関係違反が発生したことを検出する
- 投機的に実行されているスレッドのストアを実メモリへ反映させるのを, スレッドがコミットするまでストアバッファに蓄えるなどして遅延させる
- 投機スレッドが実行したロードが適切なバージョンのメモリセルを参照する

Address Resolution Buffer[7] は, これらの問題を一般的に解決するハードウェアとして最初に提案されたものである. しかし, 集中型構成の複雑なハードウェアとなる問題点が指摘されてきた. この問題点を解決すべく, 上記の機能を, キャッシュコヒーレンスプロトコルの拡張を行なうことによって, 分散型で実現させる提案が Speculative Versioning Cache[8] である. SVC では, 各キャッシュが同一メモリロケーションに対して独自の投機バージョンを保持しつつ, 各キャッシュラインにロード (L) ビットと呼ばれる拡張ビットを保持して, プロセッサによるそのラインからのロードの実行からラインへのストアの実行までのあいだセットして, 親スレッドが同じロケーションに本来ロードすべきであった値をあとからストアする可能性があることを示すことによって, メモリ依存違反の検出のためのバスの監視を行なうものである. しかし, オリジナルの SVP は, 各プロセッサに分散するキャッシュライン間で投機実行開始順序を示す順序リストを保持させなければならないなど, プロトコルとハードウェア実現はまだ複雑であった. HydraCMP[1] では, ライトスルーキャッシュの採用等によって SVC の大幅な実装の簡単化を試みており, その一方で, 簡略化にともなうバストラフィックの増大を補うために書き込み専用バスを用意するという現実的な戦略を用いている. また, 最近の研究としては, スケーラビリティを高める研究 [9, 10] も行われている. 我々は, これらの機能のソフトウェアでの実現の可能性を検討している [11].

### 3.2 投機スレッドの管理の支援機構

スレッドレベルの投機実行では

- 投機スレッドの生成
- 親スレッド実行完了の子スレッドへの通知
- 子投機スレッドの予測誤り検出の親スレッドへの通知

などの各種イベントの発生の時点で適切な処理が必要であり, これにともなって, 各スレッドの投機 / 非投機実行モードの変更や, 投機スレッド間に存在する順序関係の維持をはじめとする投機スレッド管理処理が必要となる [11].

最初の投機スレッドの生成においては, スレッド初期状態の設定を適切かつ効率良く行った上でスレッドの実行を開始させる必要がある. 2 番目の親スレッド実行完了の子スレッドへの通知においては, 子スレッドの投機状態から非投機状態への移行処理 (ストアバッファのフラッシュなど) がともなう. また, 3 番目の子スレッドの予測誤りの検出においては, 子スレッドがすでに生成している孫スレッドの停止を正しく行なわなければならない.

[11] における我々の経験によると, これらの処理はかなりの複雑度を持つものであり, 実現のためにはいずれにしてもソフトウェアの介在が不可欠であるいっぽうで, 必要とされる機能全てを既存のハードウェアを用いてソフトウェアで実現しようとすると, 粒度の細かい排他制御, イベントのポーリング等の追加処理と, それらを実現する際に発生するコヒーレントキャッシュ上の共有変数を介したスレッド間通信 (キャッシュインバリデーションの多発をともなう) のオーバヘッドが無視できないほど大きくなる恐れが大きい. 実際, HydraCMP では, これらを実現させる実行時ソフトウェアのオーバヘッドがかなり大きくなってしまったことが報告されており [12], 処理の粒度を細かくできない, ファンクショナルレベルの並列性の抽出をあきらめざるをえない等, システムの適用範囲を狭めかねない弊害を招いてしまっている. また, 我々の予備的研究でも, 投機実行を行う中間コードインタプリタを全てソフトウェアで実現させるにあたって, 同様の問題点を経験している [11].

よって, 投機スレッド管理処理の適切かつ柔軟性を失わないハードウェア支援と, 支援ソフトウェアがハードウェアを効率良く利用できるための適切なソフトウェアインタフェスの設計が不可欠である.

## 4 動的情報収集機構の検討

### 4.1 プロファイル対象のモデル化

投機実行はプログラムの確率的振る舞いを利用した動的な最適化技法であるとみなすことができ, 高い確

率で起こることが予測される事象を仮定した最適化処理と，予測事象の実行時における確認作業が必要となる[6]．前章で述べた投機実行支援機構の役割は主に後者の実行時の予測事象の確認の処理であったが，投機実行を効率良く行うためには，プログラムの確率的振る舞いの把握が重要であり，プログラムの実行時情報の取得のハードウェアによる支援も劣らず重要であると考えなければならない．

プログラムの実行時情報の取得のハードウェア支援を考える場合，プロファイル対象であるプログラムをどのようにモデル化するかが重要な選択点となる．同様の問題提起をしている[13]では，パイプライン各ステージで発生するイベントと動的命令の時系列を関係表と見なし，これに対して問い合わせを発行して処理させるプロファイルデータモデルを提案している．これはハードウェアに密着したプロファイルデータモデルとして興味深いものであるが，プロファイルデータをコンパイラが最適化に利用することを前提とするならば，この目的に有用な抽象度の高いプログラム動作特性をあらわすモデルは，コンパイラ自身が採用しているプログラムモデルと類似性の高い

> 制御依存グラフとデータ依存グラフのアークに，そのアークを流れる情報の属性を付加したもの

と考えるのがより妥当であると考えられる．これは[6]で我々が提案した「投機的 制御 / データ依存グラフモデル」をより一般化したものととらえることができる．

以下

- 制御フローのプロファイリング
- データ値のプロファイリング
- データ依存関係のプロファイリング

の順で検討を進めていく．

## 4.2 制御フローのプロファイリング

一般にプログラムは，その実行時間の90%がコードの10%で費やされるという，いわゆる「90/10ルール」に従うとされる．したがって，制御フローのプロファイリングによってこのコードの10%の部分を検出すること，すなわちホットスポットの検出が重要となる．

このようなホットスポット / ホットトレースの検出方法として，ブロックカウンタないしエッジカウンタを用いてソフトウェアで行なう方法が従来から用いられている．いっぽう，よりプロファイリングのオーバヘッドを低減させ，より精密なデータの取得と自己改変コードへの対応を可能とすることを目的として，プログラムの重要な実行部分であるホットスポットの検出をハードウェアで行なわせる，ホットスポットディテクタ[14, 15]の研究が行なわれている．ホットスポットディテクタは，分岐命令毎の実行回数とその振る舞いに着目し，実行頻度の高い分岐命令の実行回数を計測するカウンターのテーブルを用意して，プログラムの制御フローのプロファイリングを行なう興味深いハードウェアである．

## 4.3 データ値のプロファイリング

データ値のプロファイリングによって求めるデータ依存アークの属性として，次の2種類が考えられる．

- どの変数はどの値をどのくらいの割合で保持するか
- どの変数はどのような規則性で値が変化するか

変数の値の分布の測定をソフトウェアで実現することによって，値が実際にはほとんど定数であるという性質をもった変数を検出し，検査付きスペシャライゼーションなどの最適化の対象とする研究がこれまでにも行われている[16]．値プロファイルをソフトウェアで実現することによるオーバヘッドは，プログラムの実行時間を数ないし数十倍にしてしまうと報告されており，ハードウェアによる効率的なサポートが望まれる．また，プロファイリングをソフトウェアで実現することによるオーバヘッドを低減させる研究などが行なわれている[17, 18]．

いっぽう，変数の値の変化の規則性の測定し，値予測に基づく投機実行に役立てる目的で，我々は，Java仮想マシンを改造し，プログラム中の全てのデータ依存アークにあらかじめ用意したいくつかのタイプの予測器を割り付け，データ依存アーク毎に値の予測容易性を計測するJava Jog-time Analyzerを用いた実験を行なった[6]．

値プロファイルをハードウェアで実現する場合，変数(メモリロケーション / レジスタ)を読み書きする命令を命令アドレスとタイプによって特定し，その命令によって読み書きされた値について，簡略化された頻度グラフを求めるハードウェア，および，幾つかの種類の値予測器との比較を行ない予測の正否の統計を測るハードウェアを統計情報収集ユニットとして用い，これをハードウェア規模が許す数量だけ用意することになる．

前節で述べたモデルに基づくならば，値プロファイリングは，基本的に，全ての変数に対して，頻度グラフ，ないし複数のタイプの値予測器を割り当てるということになるが，いうまでもなく，これは時間的 / 空間的 / ハードウェア的オーバヘッドが禁止的大きさとなる．変数自身の示す振る舞いと，得られる特性の有用性(それを用いた最適化の効果の度合い)に基づいて，対象とする変数を絞り混む技術の開発が重要となる[17, 18]．特性の有用性の判断には，コントロールフローのプロファイリングによるホットスポットの絞り込みが重要な役割を果たすことになるであろう．これらの絞り込の機能を，実行時支援ソフトウェアによって実現することを考えている．

## 4.4 データ依存関係のプロファイリング

並列化コンパイラにとって有用なもう1つの実行時情報として

- どのストア命令で書かれたメモリロケーションはどのロード命令で読み出されるか

があり，投機スレッド実行の正否を判断する実行時統計として用いられる．これの効果的なハードウェアサポートを考えている．

## 5 実行時支援ソフトウェアの役割

近年，実行時オプティマイザ[19, 20, 21]や，実行時支援ソフトウェアを用いた仮想アーキテクチャの実現[22, 23]などが注目を集めている．プログラムの静的解析のみでは知ることの難しいプログラムの動的特性(例えばホットスポットやホットトレースの把握)を加味することによって，より効果的なプログラム最適化が可能となると考えられていることが理由の一つである．

プログラム中の実行頻度が高く最適化の効果の高いことが期待される部分に，最適化のリソース，すなわち，コンパイラ最適化時間や，数量の限定された実行情報収集ハードウェアなどを割り当てることによって，より効果的なプログラム最適化を行なうことが可能となることが期待される．また，現在広く行なわれているモジュール単位での独立したプログラム開発において，実行直前にはじめてリンクされるプログラム全体にわたる最適化が可能となり，プログラム実行の実体に則した最適化スコープの拡大がはかれるという利点ももたらされることが指摘されている．

今回我々が開発するシステムにおいても，実行時支援ソフトウェアの当初の役割として，プログラム中の実行頻度が高く最適化の効果の高いことが期待される部分の特定，数量の限定された実行情報収集ハードウェアを適切に割り当てることなどを考えている．

## 6 新しいツールを利用したアーキテクチャ研究

最後に，本節では，これまでに検討した，動的情報収集機構および投機実行支援機構を備えたチップマルチプロセッサを，我々がどのように研究開発を進めていこうと考えているかについて述べる．

従来，計算機アーキテクチャ研究において，ハードウェアの実現を伴う実証的研究が行なわれることが普通であった．しかし，研究対象のプロセッサアーキテクチャの指数的複雑化や動作速度の高速化などに伴い，ハードウェア(特に集積回路)を研究の一部として実現することに多大な手間がかかるようになり，現在ではむしろソフトウェアシミュレーション手法を主体とした研究が中心となってきている．

しかしその一方で，ソフトウェアシミュレーションで得られるデータの精度と信頼性の問題，シミュレーション実験において時間的制約のためにたびたび行われるプログラム実行の部分的な切り出しの問題点や，オペレーティングシステムの動作など総合システムとしての動作を考慮した性能評価の重要性の度重なる指摘などから，ハードウェア実装を伴う研究手法の重要性が再び見直されようとしている[24, 25]．

我々は，今回の研究において「リコンフィギュアラブル実験装置」を用いる．同装置は，開発時点で入手可能であった最大規模のFPGA(約250万ゲート相当)を中心としつつ，プロセッサエミュレーションという主用途に特化して，実用規模のプログラムの実行に十分なサイズのメモリ(DRAMおよび高速SRAM)や通信インタフェースなどを周辺に配置して，幅広い計算機アーキテクチャ研究を可能とするテストベッドとすべく現在我々が開発を進めている実験装置である．

ハードウェア記述言語によって記述され合成されたプロセッサ設計データをこの実験装置のFPGA内に書き込むことによって，実際にハードウェアを開発したのと比べてはるかに少ない手間で，実際の10分の1程度のクロック速度[26]で動作する計算機ハードウェアを何通りもの構成で実現して比較実験に用いることができる．通常の実験に用いるのに十分な処理性能が得られることを利用して，ソフトウェアシミュレータを用いた現在主流の研究手法では達成の難しい

- オペレーティングシステムを含むプログラム全体の現実的な時間での実行
- 設計パラメータを変化させた数多くの試行
- 必要に応じた様々な統計収集ハードウェアの埋め込み
- 実用コンパイラの開発プラットフォームとしての利用
- プロセッサ設計情報の実ハードウェア化へのスムーズな転用

などを可能としてくれる，極めて有用な研究ツールとして役立ってくれることを期待している．実時間で動作する実ハードウェアでないことに起因する，クリティカルパス等の見積もりの不正確さ等の問題点については，実際のASIC向けライブラリを用いた合成結果と順次比較することによって，常に校正を心掛けることを考えており，そのような環境も並行して準備している．

本実験装置が有効に活用されるために重要な役割を占めるのは，ホスト計算機と実験装置の連携によって動作するクロスコンパイラ/クロスデバッガなどの支援ソフトウェア，プロセッサコアをはじめとする各種ハードウェアライブラリなどの周辺実験環境であり，現在これらの整備を鋭意進めている．本実験装置の詳細については，別途報告を予定している．

## 7 おわりに

本稿では，単一プログラムからスレッドレベル並列性を最大限に引き出して効率良く実行させるための「動的情報収集機構および投機実行支援機構を備えたチップマルチプロセッサ」のアーキテクチャおよび支援ソフトウェアの機能としてどのようなものが望まれるかを検討し，さらに，これらの検討に基づいて我々が進めているチップマルチプロセッサアーキテクチャ研究開発に用いる研究ツールについての紹介を行なった．

現在，チップマルチプロセッサ基本部の設計が進んでおり，実験装置ハードウェアの完成を待って，今回検討した支援ハードウェアおよび支援ソフトウェアの詳細な設計を行なっていく予定になっている．これらの詳細については，別途報告を予定している．

## 謝辞

大蒔和仁 情報アーキテクチャ部長，並びに，日頃より熱心にご討論を頂いている同僚諸氏に感謝します．なお，リコンフィギュアラブル実験装置は，科技庁(2001年1月6日からは文部科学省)COEプロジェクト「大域情報処理」および科学技術振興事業団の科学技術特別研究員制度の一環として，佐谷野健二氏を中心として開発が進められているものであり，また，本研究中の並列化コンパイラに関する技術検討は，通産省(2001年1月6日からは経済産業省)産技プロジェクト「アドバンスト並列化コンパイラ技術開発」の一環として行われたものである．これら関係各位に感謝します．

## 参考文献


[1] Hammond, L., Willey, M. and Olukotun, K.: Data Speculation Support for a Chip Multiprocessor, *the 8th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 58–69 (1998).

[2] 鳥居淳: 1チップ制御並列プロセッサMerlotのアーキテクチャ, 並列処理シンポジウムJSPP2000 (2000).

[3] Petrovick, J. and Seymour, E.: IBM POWER4 Chip Integratrion, *Hot Chips 12 Conf. Record* (2000).

[4] Barroso, L., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadder, S., Sano, B., Smith, S., Stets, R. and Verghese, B.: Piranha: A Scalable Architecture Based on Sigle-Chip Multiprocessing, *the 27th Annual Intl. Symp. on Computer Architecture*, pp. 282–293 (2000).

[5] Pollack, F.: New Challenges in Microarchitecture and Compiler Design, *Keynote at Intl. Conf. on Parallel Architectures and Compilation Techniques* (2000).

[6] 小池汎平, 山名早人, 山口喜教: 投機的制御 / データ依存グラフとJava Jog-time Analyzer, 情処論: プログラミング, Vol. 40, No. SIG1(PRO2), pp. 32–41 (1999).

[7] Franklin, M. and Sohi, G. S.: ARB: A hardware mechanism for dynamic reordering of memory references, *IEEE Trans. Compt.*, Vol. 45, No. 5, pp. 552–571 (1996).

[8] Gopal, S., Vijakumar, T. N., Smith, J. E. and Sohi, G. S.: Speculative Versioning Cache, *the 4th Intl. Symp. on High-Performance Computer Architecture*, pp. 195–205 (1998).

[9] Steffan, J., Colohan, C., Zhai, A. and Mowry, T.: A Scalable Aproach to Thread-Level Speculation, *the 27th Annual Intl. Symp. on Computer Architecture*, pp. 1–12 (2000).

[10] Cintra, M., Martinez, J. and Torrellas, J.: Architectural Support for Scalable Speculative Parallelization in Shared-Memory Multiprocessors, *the 27th Annual Intl. Symp. on Computer Architecture*, pp. 13–24 (2000).

[11] 小池汎平, 山名早人, 山口喜教: 逐次プログラムの投機並列実行を行なう中間コードインタプリタの構成法, 情処論: プログラミング, Vol. 40, No. SIG10(PRO5), pp. 64–74 (1999).

[12] Olukotun, K., Hammond, L. and Willey, M.: Improving the Performance of Speculatively Parallel Applications on the Hydra CMP, *1999 Intl. Conf. on Supercomputing*, pp. 21–30 (1999).

[13] Heil, T. and Smith, J.: Relational Profiling: Enabling Thread-Level Parallelism in Virtual Machine, *the 33th Intl. Symp. on Microarchitecture*, pp. 281–290 (2000).

[14] Merten, M. C., Trick, A., George, C., Gyllenhaal, J. and Hwu, W.: A Hardware-Driven Profiling Scheme for Identifying Program Hot Spots to Support Runtime Optimization, *the 26th Annual Intl. Symp. on Computer Architecture*, pp. 136–147 (1999).

[15] Merten, M. C., Trick, A., Nystrom, E., Barnes, R. and Hwu, W.: A Hardware Mechanism for Dynamic Extraction and Relayout of Program Hot Spots, *the 27th Annual Intl. Symp. on Computer Architecture*, pp. 59–69 (2000).

[16] Calder, B., Feller, P. and Eustace, A.: Value Profiling, *the 30th Annual Intl. Symp. on Microarchitecture*, pp. 259–269 (1997).

[17] Burrows, M., Erlingson, U., Leung, S., Vandevoorde, M., Waldspurger, C., Walker, K. and Weihl, B.: Efficient and Flexible Value Sampling, *the 9th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 160–167 (2000).

[18] Watterson, S. and Debray, S.: Goal-Directed Value Profiling, *the 3rd ACM Workshop on Feedback-Directed and Dynamic Compilation*, pp. 21–30 (2000).

[19] Bala, V., Duesterwald, E. and Banerjia, S.: Dynamo: A Transparent Runtime Optimization System, *the 2000 Conf. on Programming Language Design and Implementation* (2000).

[20] Gordon, R.: Wiggins/Redstone: An on-line Program Specializer, *Hot Chips 11 Conf. Record* (1999).

[21] Chen, W., Lerner, S., Chaiken, R. and Gillies, D.: Mojo: A Dynamic Optimization System, *the 3rd ACM Workshop on Feedback-Directed and Dynamic Compilation*, pp. 81–90 (2000).

[22] Ebcioglu, K. and Altman, E.: DAISY: Dynamic Compilation for 100% Architectural Compatibility, *the 24th Annual Intl. Symp. on Computer Architecture*, pp. 26–37 (1997).

[23] Ditzel, D.: Transmeta's Crusoe: Cool Chips for Mobile Computing, *Hot Chips 12 Conf. Record* (2000).

[24] Heinrich, M., Gibson, J., Kunz, R., Ofelt, D., Horowitz, M. and Hennessy, J.: FLASH vs. (Simulated) FLASH: Closing the Simulation Loop, *the 9th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 49–58 (2000).

[25] Nanda, A., Mak, K.-K., Sugavanam, K., Sahoo, R. K., Soundararajan, V. and Smith, T.: MemorIES: A Programmable, Real-Time Hardware Emulation Tool for Multiprocessor Server Design, *the 9th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 37–48 (2000).

[26] 佐谷野健二他: MIPSベースマルチスレッドプロセッサのFPGAによる実装と評価, 情報処理学会アーキテクチャ研究会報告, No. 74 (2000).