# クロスバスイッチをなくしたマルチバンクキャッシュ

嶋田　創[†]　安藤秀樹[†]　島田俊夫[†]

プロセッサの性能を向上させるためには，命令の実行に必要なデータが遅れることなく供給される必要がある．そのために，現在の多くプロセッサにおいて，マルチバンク化という手法でキャッシュをマルチポート化している．しかし，クロスバスイッチの存在により，従来のキャッシュよりもキャッシュアクセスに時間がかかるという欠点がある．我々はこのクロスバスイッチをなくしたマルチバンクキャッシュの構成と，その実現のために必要なバンク番号の早期計算機構を提案する．また，提案する構成においてより高い性能を得るために，バンク予測による投機的命令発行の併用を提案する．提案の構成に，バンク番号予測による投機的命令発行を適用することにより，平均 4.7%の性能向上を達成できることを確認した．

# Multi-Bank Cache without Crossbar Switch

HAJIME SHIMADA,[†] HIDEKI ANDO[†]
and TOSHIO SHIMADA[†]

Data must be provided without delay when instruction requests them to sustain processor performance. Caches must have sufficient numbers of ports to meet this requirement in wide-issue processors. A general way to organize multiported caches is to separate caches in banks. Although multi-bank caches are more efficient than any other alternatives, cache access latency lengthens due to the crossbar switch that connects load/store units and cache banks. This paper proposes a mechanism that removes the crossbar to reduce the latency. The mechanism includes early calculation of bank numbers, bank number prediction, and speculative instruction execution base on that prediction. Our evaluation results show that our multi-bank cache with the mechanisms achieves 4.7% performance improvements over a conventional multi-bank cache in an eight-issue processor.

## 1. はじめに

近年のプロセッサは，命令レベルの並列性を用いて 1 サイクル中に複数の命令を実行するスーパースカラプロセッサが主流となっている．そして，命令レベル並列性を高めるべく，プロセッサの命令発行幅は増加する傾向にある．プロセッサで実行されるプログラムの命令の 3 分の 1 はロード／ストア命令である[1]．そのため，多命令発行を行うプロセッサが，命令の発行を続けるためには，1 サイクル中に複数のロード／ストアを行えるデータキャッシュが必要となる．この要求に応えるために，プロセッサ内のキャッシュはマルチポート化されている．

実際のプロセッサで利用されているキャッシュのマルチポート化の手法は 2 つある．まず 1 つ目は多重化である．全く同じ内容のキャッシュを必要なポート数だけ用意することにより，複数のロード／ストア命令の要求に応えるものである．この構成は，任意の組合せのロード／ストア命令の要求に応えることができるが，N ポートキャッシュを実現すると，コストはシングルポートキャッシュの N 倍になるという問題がある．この手法は，DEC Alpha 21164[2] で採用されている．

2 つ目はマルチバンク化[3] である． この手法は，キャッシュをバンクと呼ばれるグループに分け，バンクごとにポートを用意する．これにより，異なるバンクにアクセスされた時のみ，複数のロード／ストア命令の要求に応えることができる．追加コストは，バンクごとのデータ選択機構と，バンクとロード／ストアユニットをつなぐクロスバスイッチだけであり，キャッシュの容量にかかわらず，追加コストの大きさは小さい．このため，マルチバンク化は多重化と比較して，大容量キャッシュのマルチポート化を実現をしやすい．このような面での有利さから，Intel Pentium[4] ,MIPS R10000[5] ,Compaq Alpha 21264[6] ,Sun UltraSPARC[7] ,AMD Athlon[8] など，近年の多くのプロセッサにおいて採用されている．しかし，同一バンクへのアクセス競合や，クロスバスイッチによるアクセス時間の増加などの問題がある．

† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

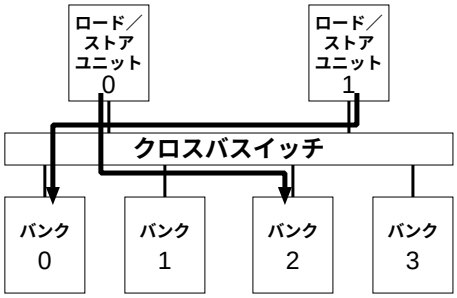


**図 1**　マルチバンクキャッシュの構成

本研究の目的は，このマルチバンクキャッシュにおいて，クロスバスイッチによるアクセス時間の増加の削減である．本稿では，クロスバスイッチをなくしたマルチバンクキャッシュの構成と，その実現に必要なバンク番号の早期計算機構を提案する．さらに，その構成においてより高い性能向上を得るために，バンク番号予測による投機的命令発行の併用を提案する．

以下に本論文の構成を示す．2 章では本研究の基礎となるマルチバンクキャッシュについて詳しく述べ，3 章で提案するクロスバスイッチをなくしたマルチバンクキャッシュの構成を示す．また，提案する構成の実現に必要なバンク番号の早期計算と，さらなる性能向上を目指すためのバンク番号予測による投機的命令発行について説明する．4 章で本研究で提案する手法の評価方法と評価に用いるベンチマークを説明し，評価によって得られた結果を示し，評価結果の考察を行う．最後に 5 章で本研究によって得られたことをまとめる．

## 2. マルチバンクキャッシュ

### 2.1 マルチバンクキャッシュの構成

マルチバンクキャッシュの構成を **図 1** に示す．キャッシュは，キャッシュラインやワードなどを単位として，バンクと呼ばれるグループに分けられており，各バンクごとにポートを持っている．各バンクのポートはクロスバスイッチを介してロード／ストアユニットと接続されている．このように，クロスバスイッチを用いてロード／ストア命令の要求ごとにロード／ストアユニットとキャッシュのバンクをつなぐことにより，1 サイクル中に複数のロード／ストア命令の実行を可能としている．ただし，同一サイクル中に複数のロード／ストア命令の要求が 1 つのバンクに集中した場合は，複数のロード／ストア命令の要求のうち，1 つにしか応えることができない．

キャッシュにおいて，ライン番号，ライン内オフセットは，アドレスの一部を用いて指定する．マルチバンクキャッシュで新たに必要となるバンク番号は，**図 2** のように，従来のキャッシュのライン内オフセットの下位ビットをバンク番号として用い，複数のロード／ストア命令の要求が同一バンクへ集中するのを防ぐ．


**図 2**　バンク番号の確保

### 2.2 マルチバンクキャッシュの欠点

マルチバンクキャッシュは，現在の数多くのプロセッサで採用されているが，利点と同時にいくつかの欠点もある．以下，欠点について述べる．

(1)　バンク競合

マルチバンクキャッシュでは 1 つのバンクに対してポートは 1 つしかない．したがって，ロード／ストア命令の要求が 1 つのバンクに集中した場合は，複数の要求のうち，1 つの要求にしか応えることができない．これをバンク競合と呼ぶ．

(2)　クロスバスイッチによるレイテンシの増加

マルチバンクキャッシュのキャッシュアクセス時には，シングルポートキャッシュと比較して，クロスバスイッチを通過する時間が余分にかかる．これによるクロックサイクル時間の増加を避けるため，通常，データキャッシュのアクセスは 2 段以上のパイプラインで構成されている．

バンク競合の問題については文献 9) にておいて競合を緩和する研究がされている．しかし，クロスバスイッチによるレイテンシの増加の問題について，レイテンシを削減する研究はされていない．我々は，このクロスバスイッチによるレイテンシを削減する研究を行なった．

## 3. クロスバスイッチをなくしたマルチバンクキャッシュの提案

本章では，まず最初に，クロスバスイッチをなくしたマルチバンクキャッシュの構成を提案する．次に，その構成の実現に必要となるバンク番号の早期計算機構，および，さらなる性能向上を得るための，バンク番号の予測による投機的命令発行について述べる．この機構により，従来のクロスバスイッチのあるマルチバンクキャッシュと比較して，ロード／ストア命令の実行サイクルが短くなる．これにより，後続命令が先行するロード／ストア命令に依存関係がある命令列では，従来よりも依存解消が早くなるため，性能向上が得られる．

### 3.1 クロスバスイッチをなくしたマルチバンクキャッシュの構成

従来のキャッシュでは，ロード／ストアの要求を各バンクに振り分けるため，ロード／ストアユニットとキャッシュの間にクロスバスイッチをいれる構成を取っている．我々は，クロスバスイッチにおけるロード／ス

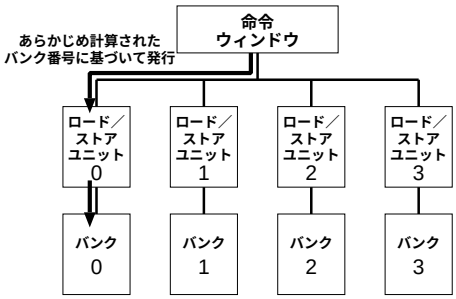


**図 3**　クロスバスイッチをなくしたマルチバンクキャッシュの構成

トアの要求を振り分ける作業が，命令ウィンドウから各機能ユニットに命令を発行する作業と似ている点に着目した．そして，命令ウィンドウに命令が入る前にバンク番号を求め，命令ウィンドウから各機能ユニットに命令を発行する作業に，各バンクへの振り分け作業を兼ねさせることを考えた．

**図 3** に，提案するクロスバスイッチをなくしたマルチバンクキャッシュの構成を示す．この構成では，バンクと同数のロード／ストアユニットを用意し，ロード／ストアユニットとバンクを一対一で接続する．ロード／ストア命令は，あらかじめ計算されて，命令ウィンドウに入っているバンク番号を元に，各バンクにつながっているロード／ストアユニットに発行される．

バンク番号の計算は，デコードステージでレジスタを読みだした直後に行われる．計算を行う機構を **図 4** に示す．バンク番号は，レジスタファイルより読み出されたレジスタ値と，命令レジスタから読み出されたオフセット値を加算して求められ，命令ウィンドウに書き込まれる．図 2 より，バンク番号はアドレスの下位にあるため，レジスタ値，およびオフセット値を全て加算しなくても求めることが可能である．そのため，バンク番号計算時に加算するレジスタ値およびオフセット値は，バンク番号を求めるのに必要な下位数ビットのみ計算する．

この構成によって，従来の構成に存在していたクロスバスイッチは不要となる．デコードステージでバンク番号が計算できるなら，クロスバスイッチがなくなったことにより，ロード／ストア命令の実行時のパイプラインは図 5 (b) のようになり，図 5 (a) に示す従来のクロスバスイッチのあるマルチバンクキャッシュと比較して，命令の完了までのサイクル数は 1 サイクル短くなる．

問題となる点として，デコードステージにおいてバンク番号の計算を行えない場合がある．それは，ロード／ストア命令のベースレジスタがまだ計算されておらず，利用可能でない場合である．この時は，計算されたレジスタ値と命令ウィンドウ中のオフセット値からバンク番号を計算した後，ロード／ストア命令を発行する．この計算の様子を **図 6** に示す．まず，図 6 (a) に示すように，依存するレジスタをライトバックするサイクルにおいて，ライトバックされるレジスタ値をバンク番号計算器にフォワーディングする．これを，命令ウィンドウより送られて来るオフセット値と加算し，バンク番号を計算して命令ウィンドウに書き込む．そして，図 6 (b) に示すように，後のサイクルで，計算で得られたバンク番号に基づいて命令を発行する．この時のロード／ストア命令の実行時のパイプラインは図 5 (c) になる．命令の実行完了までのサイクル数は，図 5 (a) の，従来のクロスバスイッチを用いたマルチバンクキャッシュと同じになる．ただし，従来の構成では，依存している命令の実行が行なわれるサイクルの，次のサイクルにはロード／ストア命令のアドレス計算が行なえたが，提案する構成では，アドレス計算の前にバンク番号計算と発行のサイクルが必要になる．そのため，**図 7** のように，従来の構成より，キャッシュアクセスの完了が 1 サイクルの遅れとなる．

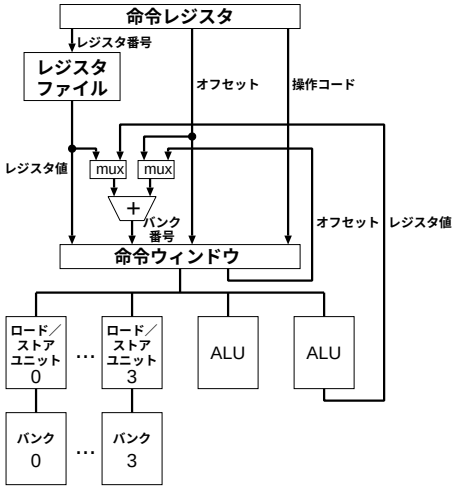


**図 4**　バンク番号の早期計算

### 3.2　バンク番号予測による投機的命令発行

3.1 節では，デコードステージにおいてバンク番号を計算できない場合は，バンク番号の計算を行なうためのサイクルを設けることにした．本節では，このバンク番号の計算を行うサイクルを省くために，バンク番号を予測し，それをもとに投機的にロード／ストア命令を発行することを考える．

以下，本研究で適用するバンク予測の手法について述べる．

(1)　Last Bank

メモリアクセスには空間的局所性があり，最近アクセスしたデータの周辺は再びアクセスされる可能性が高いことはよく知られている．Rivers ら[9] によって，4 バンクのマルチバンクキャッシュにおいて，連続するロード／ストア命令が同一バンクをアクセスする確率は 30%から 55%であると示されている．したがって，前のサイクルで最後のロード／ストア命

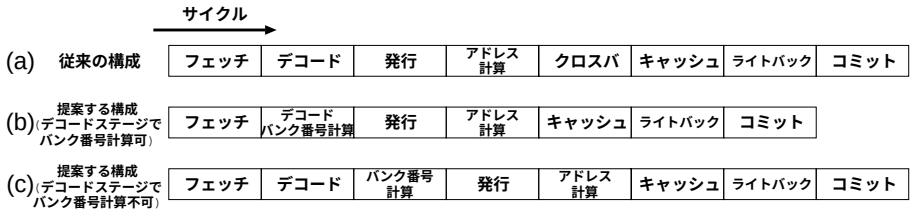


**図 5**　実行パイプラインの比較

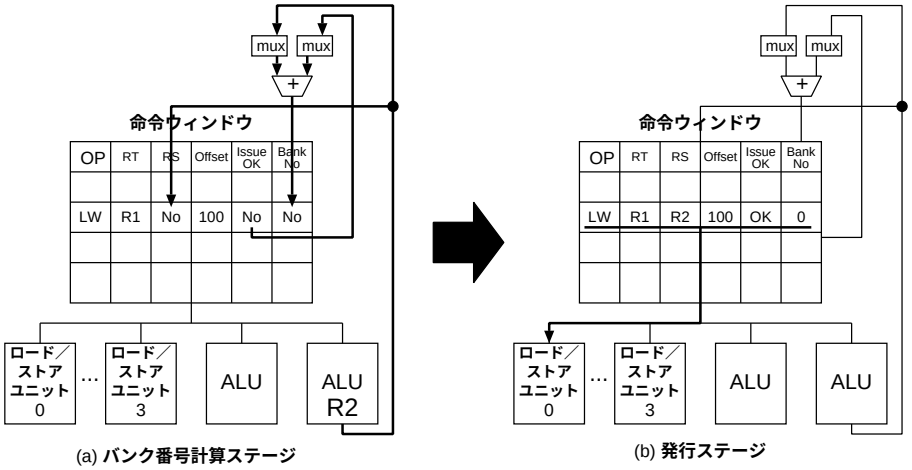


**図 6**　フォワードされたレジスタ値によるバンク番号計算

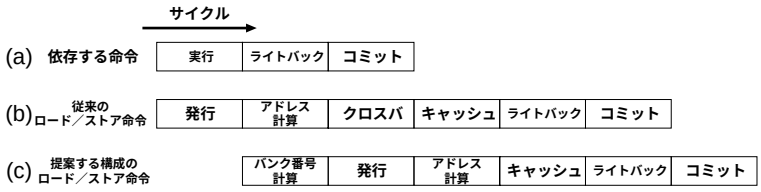


**図 7**　RS に依存がある時のパイプライン

令がアクセスしたバンクを予測バンク番号とすれば，高い確率で予測が成功すると考えられる．そのため，この手法では，最後にアクセスしたバンクの番号を保持するためのレジスタを準備しておく．このレジスタの値は，ロード／ストア命令の実行ごとに更新される．そして、命令発行時にレジスタを参照し，前のサイクルで最後のロード／ストア命令がアクセスしたバンクを予測バンク番号とする．

しかし，この手法には欠点がある．それは，同一サイクル中に，Last Bank で求めた予測バンク番号を用いた投機を，複数行なえないという点である．なぜならば，同一サイクル中に行なわれる 2 回目以降のバンク番号予測を行なおうとすると，2 回目の予測バンク番号は，1 回目の予測バンク番号と同じになるため，バンク競合を起こしてしまうからである．

(2)　Old Register

この手法では，デコードステージにてレジスタファイルに入っている古いレジスタ値を用い，通常のバンク番号と同様の計算を行なう．この計算により得られる値をアドレスと推定し，バンク番号になる桁の値を，予測バンク番号とする．

これは，計算後のレジスタ値は計算前のレジスタ値に対して，全ての桁が異なることは少ないという仮定に基づいている．計算後のレジスタ値において，バンク番号の計算に影響する桁が計算前のレジスタ値と変化しない場合，この手法によって得られる予測バンク番号は的中することになる．

このようにして得られる予測バンク番号が的中する

例として，以下の命令列をあげることができる．

```
LOOP:
        addi R2,R2,1
        lb R3,0(R2)
        add R4,R4,R3
        subi R5,R5,1
        bne R5,R0,LOOP
```

これは，ストライドの大きさが 1 バイトである，連続した要素へのアクセスである．この場合のロード／ストア命令はオフセット値を固定し，ベースレジスタ値をインクリメントしながらアクセスする．この時，図 2 のように，ライン内オフセットが 3 ビット取られているキャッシュでは，インクリメントによってバンク番号の値が変化するのは，8 回のインクリメントに 1 回となる．そのため，このような命令列に Old Register を適用すると，高い確率で予測が成功すると考えられる．

また，前の例とは逆に，ストライドの大きな連続した要素へのアクセスも，Old Register を用いることで，高い確率で予測できる考えられる．なぜなら，ストライドが大きければ，アドレスの上位が主に変化し，バンク番号のある下位は変化しない場合が多いと思われるからである．このようなストライドの大きな例として，大きな構造体の配列の中の要素に連続してアクセスする場合が考えられる．

(3) Last Bank + Old Register

この手法は，Last Bank による予測の欠点である複数投機を行なえないという点を改良するため，Old Register と組み合わせたものである．基本的に Last Bank を使い，Last Bank では投機を行なえない同一サイクル中の 2 回目以降のバンク番号予測に，Old Register を用いるという手法である．

投機的命令発行を行なうに当たり，発行ミス時からの回復手法も考えなくてはならない．発行ミスは，投機的に発行された命令が，ロード／ストアユニットにおいてアドレス計算を行なった時に確定する．もし，正しいアドレスのバンク番号と，予測されたバンク番号が異なる場合，発行ミスである．発行ミスした命令は，次のサイクルにおいて，命令ウィンドウから正しいバンク番号をもとに，正しいロード／ストアユニットに再発行される．この時に必要となる正しいバンク番号は，投機的命令発行を行なった命令がアドレス計算を行なうのと並行して，3.1 節と同様に，フォワーディングされたレジスタ値を用いてバンク番号を計算することによって得られる．

## 4. 評価結果および考察

### 4.1 評 価 方 法

評価は SimpleScalar Tool Set[10] 中の out-of-order 実行シミュレータを使用した．命令セットは MIPS R10000[5] である．ベンチマークプログラムは SPECint95 の compress95, gcc, go, ijpeg, li, m88ksim, perl, vortex の 8 本を用いた．それぞれのベンチマークプログラムへの入力は，実行命令数が 100M 命令から 200M 命令になるように調整した．

**表 1** 評価に用いるプロセッサの構成

| | | |
|---|---|---|
| フェッチ幅 | | 8 命令 |
| デコード幅 | | 8 命令 |
| 発行幅 | | 8 命令 |
| 機能ユニット | 整数 | 4 個 |
| | 整数乗除算 | 1 個 |
| | 浮動小数点 | 2 個 |
| | 浮動小数点乗除算 | 1 個 |
| TLB | 命令 | 64 エントリ |
| | データ | 128 エントリ |
| 分岐予測 | | 完全 |

**表 2** 評価に用いるキャッシュの構成

| | L1 命令 | L1 データ | L2 |
|---|---|---|---|
| 容量 | 64KB | 64KB | 2MB |
| ラインサイズ | 32B | 32B | 64B |
| 連想度 | 1 | 1 | 4 |
| バンク数 | 1 | 4 | 4 |
| ポート数 | 1 | 4 | 4 |
| ミスレイテンシ | 6 | 6 | 18 |

仮定するプロセッサの構成を **表 1** に，キャッシュの構成を **表 2** に示す．表 1 にロード／ストアユニット数が無いが，これは，SimpleScalar では整数ユニットがロード／ストアユニットを兼ねているためである．また，L2 キャッシュは命令・データ混合型である．

### 4.2 クロスバスイッチをなくしたマルチバンクキャッシュの評価結果

従来のマルチバンクキャッシュの場合を基準とし，投機を併用しない提案する構成による性能向上を 8 に示す．棒グラフの左は提案する構成によって得られた性能向上で，右は全てのロード／ストア命令がデコードステージでバンク番号の計算が可能と仮定した理想的な状態での性能向上である．提案する構成により，go において最大の 6.3%を得られた．しかし，compress95, ijpeg, vortex では，デコードステージでバンク番号が計算できない時のペナルティが多く，性能が低下してしまっている．このため，平均の性能向上は，1.6%に留まっている．

### 4.3 バンク番号予測を用いて得られた結果

従来のマルチバンクキャッシュの場合を基準とし，投機を併用した提案する構成による性能向上を 9 に示す．図のグラフは左からそれぞれ，投機を用いない場合，Last Bank, Old Register, Last Bank + Old Register, 理想的での性能向上である．それぞれの平均の性能向上は，Last Bank で 3.9%，Old Register で平均 3.1%，Last Bank + Old Register で平均 4.7%であ

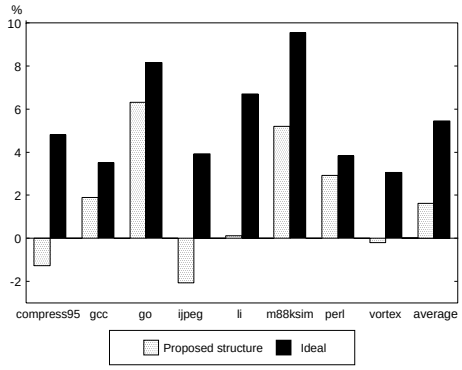


**図 8**　投機を併用しない提案する構成により得られた結果

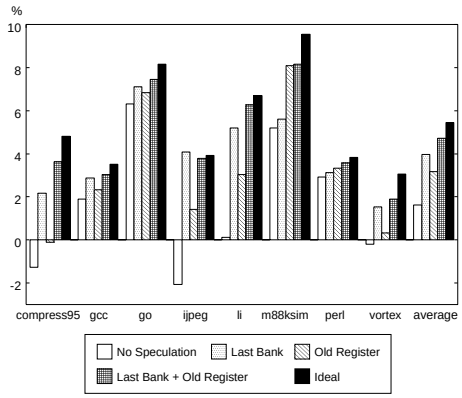


**図 9**　投機を併用した提案する構成により得られた結果

る．投機を用いることにより，用いない場合より平均で Last Bank で 2.3%，Old Register で 1.5%，Last Bank + Old Register で 3.1% 性能を向上させることができた．

## 5. ま と め

現在のプロセッサではキャッシュの容量や必要とされるポート数は増大する傾向にある．キャッシュの容量やポート数を増加させると，キャッシュのレイテンシが増加することが避けられない．本研究では，マルチバンク化によるポート数の増加のために，従来では必要とされたクロスバスイッチをなくした構成を提案することにより，クロスバスイッチによるレイテンシを削減した．

この提案する構成を評価したところ，Last Bank + Old Register バンク番号予測を用いた投機的命令発行により，平均 4.7%の性能向上を達成した．

**謝辞**　本研究の一部は，文部省科学研究費補助金基盤研究 (C)「広域命令レベル並列によるマイクロプロセッサの高性能化に関する研究」(課題番号 1068034) 及び文部省科学研究費補助金基盤研究 (C)「分岐予測と投機的実行に関する研究」(課題番号 11680351) の支援により行った．また，本研究を進めるにあたり、終始御指導頂いた片山清和氏に深く感謝の意を表します。

## 参 考 文 献

1) J.L.Hennessy and D.A.Patterson: *Computer Architecture A Quantitative Approach*, Morgan Kaufmannn Publishers, Inc. (1990).
2) Digital Equipment Corporation: *Alpha 21164 Microprocessor Hardware Reference Manual*.
3) G.S.Sohi and M.Franklin: High-bandwidth Data Memory Systems for Superscalar Processors, *In proceedings of 4th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 52–62 (1991).
4) Intel Corporation: *Pentium(R) Processor Family Developer's Manual Vol.1* (1995).
5) MIPS Technologies, Inc.: *MIPS R10000 Processor User's Manual*.
6) B.A.Gieseke, R.L.Allman and D.W.Bailey: A 600MHz Superscalar RISC Microprocessor with Out-Of-Order Execution, *Digest of IEEE International Solid-State Circuits Conference*, pp. 176–177 (1997).
7) P.Song: Hal Packs SPARC64 on to Single Chip, *Micro Processor Report Vol.11 No.16* (1997).
8) Advanced Micro Device, Inc.: *AMD Athlon Processor Technical Brief* (1999).
9) J.A.Rivers, G.S.Tyson, E.S.Davidson and T.M.Austin: On High-Bandwidth Data Cache Design for Multi-Issue Processors, *In proceedings of 30th Annual International Symposium on Microarchitecture*, pp. 46–56 (1997).
10) D.Burger and T.M.Austin: The SimpleScalar Tool Set, Version 2.0, Technical report, University of Wisconsin-Madison Computer Sciences Department (1997).