計算機アーキテクチャ　91－9
（１９９１．１１．２２）

スーパスカラ・プロセッサーSARCHーの性能評価

中西知嘉子　安藤秀樹　町田浩久　中屋雅夫
三菱電機（株）

命令レベルの並列性の抽出と分岐遅延による性能低下の緩和は、スーパスカラの設計における重要な課題である。我々は、並列性の抽出と分岐遅延の問題を、単純なハードウェアでサイクル・タイムを犠牲にせずに解決する方法として、ブースティング方式と遅延分岐方式を検討し、SARCH (Superscalar ARCHitecture)と呼ぶアーキテクチャを提案し、評価を行なった。SARCHでは、ハードウェアを簡単にするため、コード・スケジューリングを動的に行なわない。計算機の能力を十分に発揮するためには、コンパイラによるコードの最適化が性能向上のためのキーとなる。我々は、コード最適化としてまず、並列度向上のためのコード・スケジュール（基本最適化）を行ない、さらに、ループの最適化、Load/Store命令の最適化をも行なった。Dhrystone ver1.1による性能評価により、基本的最適化を行なったコードで、スカラ・マシンの1.9倍、さらに、ループの最適化、Load/Store命令の最適化を行なったコードで、3.2倍の性能向上が確認できた。これは、動作周波数50MHzで、VAX-11/780相対MIPS値として137MIPSに相当する。

Performance Evaluation of a Superscalar Processor: SARCH

Chikako Nakanishi, Hideki Ando, Hirohisa Machida, and Masao Nakaya
Mitsubishi Electric Corporation
LSI Laboratory
4-1 Mizuhara, Itami, Hyogo, 664 Japan
e-mail: ikenaga@ls2.cll.melco.co.jp

Exploiting instruction-level parallelism and alleviation of performance degradation by branch delay are problem which should be solved in architectural design on superscalar machines. We have proposed a superscalar architecture, SARCH, employing instruction 2 way boosting and a new delayed branch scheme. Code optimization by the compiler is a key to get high performance since SARCH does not schedule the code dynamically. We takes three optimization steps: a basic optimization for exploitation of instruction-level parallelism, a loop optimization, and a load/store optimization. The performance evaluation on Dhrystore ver.1.1 shows that the speedup is 1.9 and 3.2 over scalar machines in the basic optimization code and in the basic, loop and load/store optimization code, respectively. Hence, the performance is 137 VAX MIPS at 50 MHz operation.

1. はじめに

近年、次世代プロセッサのアーキテクチャとして、VLIW、スーパパイプラインと並んで、スーパスカラの研究が盛んに行なわれている[納富 91][中島 91][Sohi 87]。スーパスカラの設計において考慮しなければならない最初の問題は、命令レベルの並列性の抽出である。特に、非科学計算のプログラムにおいては、基本ブロックのコード・サイズが小さいため、スーパスカラとしての特性を十分に引き出せるほどの並列性を見いだすことは困難である。

スーパスカラの設計において考慮しなければならない次の問題は、分岐遅延である。分岐遅延は、スカラ・プロセッサの性能に悪影響及ぼす。特にスーパスカラにおいては、1サイクルに複数の命令が発行されるので、理想的なマシンと比較して、分岐遅延サイクル中に失われる命令は複数存在することとなり、スカラ・プロセッサ以上に性能に対する悪影響は大きい。

我々は、これらの問題をサイクル・タイムを犠牲にせずに解決する方法を検討しSARCH (Superscalar ARCHitecture)と呼ぶアーキテクチャを提案した[安藤 91]。そして、SARCHの性能を確かめるため、シミュレータを作成し性能評価を行なった。

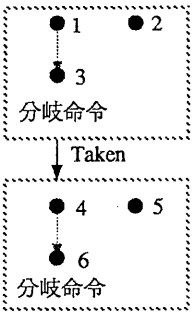
本論文では、まず、SARCHのアーキテクチャとして採用している方式であるブースティング方式、遅延分岐方式について述べ、その実現方法を示す。次に、ハードウェアの概要について述べるとともに、SARCHの能力を十分に発揮するためのコード・スケジューリング方法についても説明する。そして最後に、作成したシミュレータにより行なったDhrystone ver1.1 によるSARCHの性能評価について示す。

2. アーキテクチャの概要

スーパスカラの設計において性能向上のために考慮しなければならない問題は、並列性の抽出と、分岐遅延である。

一般に、非科学計算のプログラムの基本ブロックのコード・サイズは小さく、中央値は約4であると報告されている[M.Smith89]。このように基本ブロックのコード・サイズが小さいプログラムにおいては、並列発行可能な命令が少なく、基本ブロック内で、並列性を抽出しようとしても限界がある。

また、スーパスカラにおける分岐遅延は、スカラ・プロセッサにも増して深刻な性能の低下を引き起こす。例えば、理想的には、CPIが0.5であるスーパスカラがあるとする。分岐遅延に1サイクルを要すると仮定すると3命令に1つの分岐命令を含むプログラムをこの



サイクル	発行命令
1	1 2
2	3
3	
4	4 5
5	6

図1 理想CPIが0.5のスーパスカラが3命令に1つの分岐命令を含むプログラムを実行した場合の命令発行

スーパスカラで実行した場合、図1のように命令は発行される。この場合のCPIを計算すると1.0となり、スカラ・プロセッサと性能的に同じとなってしまう。

そこで、SARCHでは、並列性を増すためにブースティング方式を採用し、分岐遅延の性能に及ぼす影響を緩和させるために遅延分岐方式を採用した。以下に、SARCHに採用したブースティング方式、遅延分岐方式の概要について説明する。

2.1. ブースティング方式

静的なコード・スケジューリングでは、コンパイラがコードをスケジュールし、ハードウェアは移動された命令も含めてコードの順に命令を実行していく。しかし、基本ブロックのコード・サイズは非常に小さく中央値で約4である。従って、性能向上のため、コンパイラは分岐命令を越えてコードを移動することにより基本ブロックのコード・サイズを大きくし、並列度を上げなければならない。

ループ・アンローリングやソフトウェア・パイプライニング[Lam 88]によるループの最適化を用いて基本ブロックのコード・サイズを大きくし、並列度を上げる方法がある。しかし、これらの方法は、科学計算のプログラムでは有効であるが、非科学計算のプログラムにおいては実現が難しいばかりでなく、有効であるとは限らない。また、コード・サイズが大きくなってしまうという欠点もある。そこで我々は、簡単に実現でき、かつコード・サイズの増加なしに並列度を上げる方式として、ブースティング方式を採用することにした。

ブースティング方式とは、ある基本ブロックの下位の基本ブロック（続いて実行される可能性のあるブロック）に属する命令を上位の基本ブロックに移動させることを特長とし、M.SmithらによってTORCHというプロセッサで提案された[M.Smith 90]（図2 (b)）。ブー

スティングにおいては、分岐命令を越えて命令を移動する。移動された命令（ブーストされた命令と呼ぶ）は、コードの中で、明示的に移動された命令であることを示しておく。移動された命令の実行結果は、ハードウェアが、分岐の成否に従って有効化、または、無効化する。

M.Smithらは、静的な分岐予測に基づいて、ブーストする命令をNot Taken側かTaken側かのどちらかの基本ブロックとしている。どちらからブーストしたかは、分岐命令のコードで明示的に示している。また、ブーストされた命令による副作用を防ぐためには、シャドウ・レジスタを用意し、分岐が決定した段階でシャドウ・レジスタを有効化する。

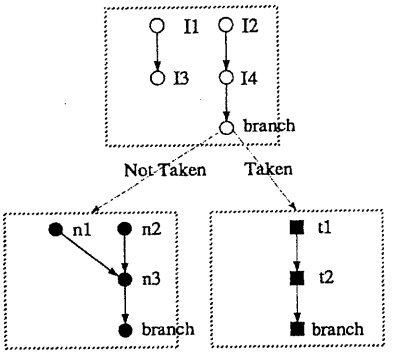
しかしながら、静的な分岐予測の精度はプログラムに依存し、すべてのプログラムにおいて高い精度を得ることは困難である。そこで我々は、ブーストする命令をNot taken側、Taken側の両方から行ない、有効な命令が同時に実行される機会を多く作り出すことにした。そして、どちらの基本ブロックからブーストしたかは、それぞれのブーストされた命令の命令コードの中に明示的に示すことにした（図2 (c)）。また、ブーストされた命令がレジスタ・ファイルを書き換える以前にブーストされた命令をキャンセルすることが可能であると考え、ブーストする範囲に制限を設けることにした。この制限により、シャドウ・レジスタは不要なものとなる。

我々は、これらの方法に従った、分岐方向の両側のブロックからのブースティングによる性能向上は、これを行なうためのハードウェアの増加による損失を上回ると考えている。また、ブーストを制限することによる性能低下は、基本ブロックの走行長から考えて少ないと考えている。

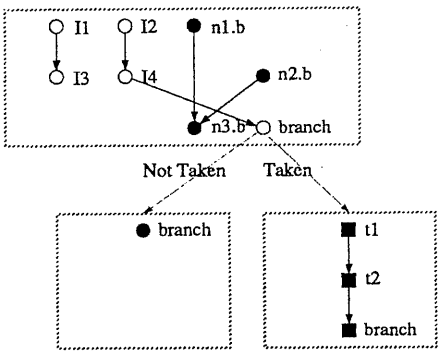
2.2. 遅延分岐方式

スーパスカラにおいて分岐遅延は、スカラ・プロセッサにも増して深刻な性能の低下を引き起こすことは、前に述べた通りである。

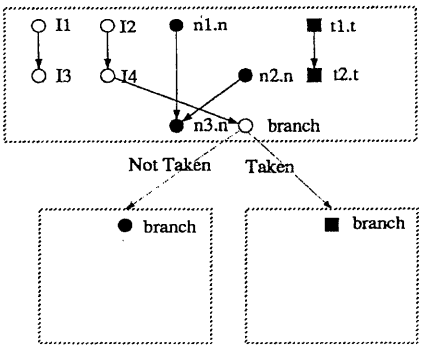
分岐遅延による性能低下を緩和する方法に遅延分岐方式がある。この方式では、分岐先命令をフェッチしているサイクルにも、命令を実行できるようにコード・スケジュールを行なう。そのため、遅延スロットに移動される命令は分岐の方向にかかわらず正しい結果が得られるように選ばなければならない。現在のRISCにおいては、ほとんどの場合遅延スロットは1である。また、この遅延スロットを有意に埋められる命令の存在確率は約0.5である[Patterson 90]。スーパスカラの場合、分岐遅延を1サイクルとしても、遅延スロットを埋めるために必要な命令は複数必要となり、これを有意に埋められる確率は低いと考えられる。



(a) ブースト前



(b) TORCH



(c) SARCH

図2　ブースティングの方式の説明図

SARCHでは、並列に発行できる命令数が一定でないため、必要となるnop命令を動的に挿入し、ハザードを動的に解消している。そのため、1サイクルあたりの発行命令数が固定せず分岐遅延スロットの数は動的に変化する。従って、遅延分岐を適用することは難しい。しかし、SARCHは以下のような方式によって遅延分岐を実現した。

1) 命令フェッチは命令ブロックと呼ぶ4ワード境界にある4つの命令に対して行なう。命令は命令キューにプリフェッチされる。命令キャッシュから読みだされた命令は、命令キューの書き込みポインタ(queue_top)から始まる4つのエントリに書き込まれる。分岐命令の実行がなければ、queue_topは、命令ブロックのサイズだけインクリメントされる。
2) 命令キューの読みだしポインタ(scope)から始まる4つのエントリの内容は毎サイクル読みだされ、命令デコーダで発行解析が行なわれ、発行可能な命令は機能ユニットに発行される。分岐命令の実行がなければ、発行された数だけscopeはインクリメントされる。
3) 分岐遅延スロットは、分岐命令の直後の命令から分岐命令を含む命令ブロックの次の命令ブロックの最後の命令までである（図3）。また、分岐遅延スロット内の命令はTakenの時のみ有効な命令である。
4) 分岐がTakenの時は、
   - 命令キャッシュに分岐先命令の読みだしを要求する。
   - 分岐遅延スロットの命令がすでに命令キューにフェッチされている場合、queue_topは分岐遅延スロットの次のエントリを指すように移動され、分岐先命令が命令キャッシュから送られるのを待つ。
   - 分岐遅延スロットの命令がまだ命令キューにフェッチされていない場合、分岐命令の実行がないときと同様に命令キューへの書き込みを行なう。
   - scopeは、分岐命令の実行がないときと同様に移動される。
5) 分岐がNot Takenの時は、
   - 命令デコーダ内にある命令のうち、分岐命令よりも後方にある命令を無効化する。
   - queue_topは、分岐命令の実行がないときと同様に移動される。
   - scopeは、分岐遅延スロットの次のエントリを指すように移動される。分岐遅延スロットの命令ブロックのフェッチが完了していない場合、次のサイクルでは、命令が命令キャッシュから送られるのを待つ。

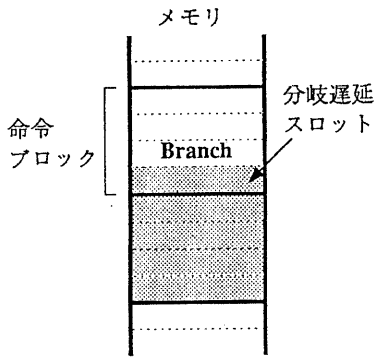


図3　分岐遅延スロット

この方式では、分岐遅延スロットはTaken側と固定している。そのため、この分岐遅延方式を採用したことによる損失は以下のように考えられる。Takenの場合、分岐先命令のフェッチを行なっているサイクルでは、分岐遅延スロットの命令を実行することができ、分岐遅延による性能低下はゼロである。Not Takenの場合は、分岐命令が実行されるサイクルにおいて分岐遅延スロットがフェッチされているなら、分岐しないことによる性能低下は、命令デコーダ内で、分岐命令よりも後方にある命令が無効化されることによる低下のみである。この損失は、分岐命令が実行されるサイクルで、分岐命令の後方に命令が存在しないようにコンパイラがコード移動することにより防ぐことができる。分岐命令が実行されるサイクルで、分岐遅延スロットがフェッチできていなければ、1サイクルの損失が生じるが、命令の発行速度よりも命令フェッチのバンド幅を大きくしておけば、この状況の起こる確率は低い。

3. ハードウェア構成

図4にSARCHのハードウェア構成を示す。5段のパイプラインの機能ユニットを3個持つ。IFステージでは、命令キャッシュから4命令を読みだす。IDステージでは、命令キャッシュから読みだされた命令を命令キューに取り込むと同時に、命令キューから命令を読みだし、命令のデコード、レジスタ・ファイルの読みだし、分岐命令の実行、命令の発行を行なう。命令の発行はin-orderに行なわれ、ハザードの存在する命令はインタロックされる。EXCステージでは、ALU演算系の命令

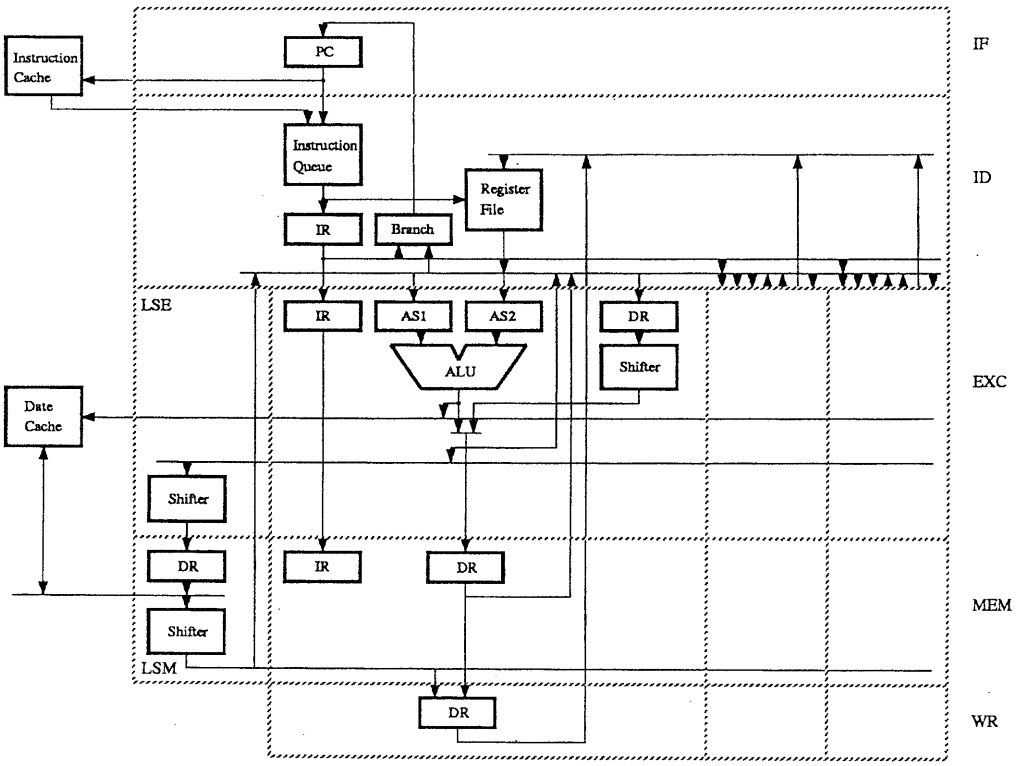


図4　SARCHのブロック図

は実行が行なわれ、Load/Store命令はアドレスが計算される。MEMステージでは、Load/Store命令はデータ・キャッシュをアクセスし、WBステージでは、レジスタ・ファイルへ書き込む。整数演算用には、同一のパイプラインを複数持つ。Load/Store命令は、EXCステージの後半のサイクルから、データ・キャッシュのアクセスを行なうLSEに送られ、データ・キャッシュのアクセスの後は、再び、元のパイプラインに戻される。LSE/LSMは1つである。この構成では、3個のALU演算命令、1個のLoad/Store命令、1個の分岐命令のうち、3個を同時に実行できる。

4. コード・スケジュール

SARCHでは、ブースティング方式や、遅延分岐方式を用いて静的にコード・スケジュールを行なうことを前提としている。そのため、計算機の能力を十分に発揮するためには、コンパイラ技術が非常に重要となる。ソース・プログラムを単にコンパイルしただけでは並列度の高い命令列を得ることができない。アーキテクチャの能力を十分に発揮するコードを得るためには、コンパイラは、プログラムの実行結果が変わらない範囲で、命令列をスケジューリングしなければならない。本章においては、SARCHのアーキテクチャに適したコード・スケジューリングについて述べる。また、より性能向上を目指し、ループの最適化、Load/Store命令の最適化を行なったので、それについても説明する。

4.1 並列度の向上

並列度向上のためのスケジューリングを行なった。コード・スケジューリングの主な手順を以下に示す。

1）基本ブロック毎にデータ依存グラフを作成する。

2）データ依存グラフにしたがって、
(i) データ依存の解析
(ii) サイクル数の判定
(iii) 資源の衝突の判定
を行ない、分岐命令までに要するサイクル数の多い命令から優先的にスケジュールをする。

3）基本ブロック内の命令において、1サイクル3命令以下の命令しか実行されないサイクルに、続いて実行される基本ブロックからのブースト可能な命令を上記1)、2)の手順にしたがって、優先度の高い命令からブーストする。

4）遅延スロットに命令を上記1)、2)の手順にしたがって移動する。

3)、4)についての詳細な説明を以下に示す。

4.1.1 ブースティング方式に対するコード・スケジューリング

SARCHのアーキテクチャにおけるブースティングの特長は以下に示す2項目である。

1）ブーストする命令をNot taken側、Taken側の両方から行なう。

2）シャドウ・レジスタを不要にするため、ブーストされた命令がレジスタファイルに書き込まれる以前にブーストされた命令をキャンセルする。

以上の特長にしたがってコード・スケジューリングを行なうには、以下のような制約がある。

(a) ブーストできる命令は、メモリ・アクセス命令、及び演算命令である。

(b) ブースト命令をおくことのできる範囲は、分岐命令の直前のスロットから、
(i) Load命令、または演算命令の場合は分岐が実行されるサイクルの2サイクル前までに実行されるスロット
(ii) Store命令の場合は分岐が実行されるサイクルの1サイクル前までに実行されるスロットまで、
と限定する。

4.1.2 遅延スロットに対するコード・スケジューリング

遅延スロットに命令を移動する手順は以下に示すとおりである。

1）遅延スロットの分岐先命令の移動
遅延スロットには、分岐先である基本ブロックから、分岐命令までに要するサイクル数の多い命令から優先的に行なう。

2）nop命令の挿入
遅延スロットを図3のように固定したため、分岐先命令で満たすことができない場合、空となるエントリにnop命令を挿入する。

4.2 ループの最適化

ループの最適化としては、ループ展開やソフトウェア・パイプライニングなど多くの研究がなされている[Lam 88][Charlesworth 81]。我々は、その中から、ループ展開による最適化を行なった。ループ最適化の中でも、ループ展開はかなり効果的なループ最適化技術である。しかし、ループ展開の回数が少ないと十分な並列性が得られない。逆に展開し過ぎるとコード・サイズが増大する。今回我々は、コード・サイズの増大を防ぐため、展開回数を2回と限定し、簡単化のため、展開するループをループ・ボディーが基本ブロックからなるループだけに限定した。

4.3 Load/Store命令の最適化

文字列操作などで頻繁に出現するコードであるバイト単位のLoad/Store命令は、性能向上の妨げになる一因となりうる。そこで、我々は、バイト単位のLoad/Store命令の中で、定数データなど、静的にメモリ・アクセス回数がわかるデータについて、可能なかぎりワード単位で、メモリ・アクセスを行なうように命令の変換を行なった。

5. 性能評価

性能評価には、Dhrystone ver.1.1用いた。まず、上記に示したように並列度向上のための処理を行なったコード（以下、基本最適化を行なったコードと呼ぶ）について評価を行なった。また、上記に述べた最適化の効果を見るために、ループ最適化を行なったコード、Load/Store最適化を行なったコードについても評価を行なった。

まず、Dhrystoneのプログラムの特徴を見るために命令出現頻度を調べた。結果を表1に示す。一般に非科学計算プログラムでは、メモリのアクセスを行なう命令の頻度は約30%程度である[Patterson 90]。表1からもわかるように、Dhrystoneでは40%と高い。これは、文字列操作命令が多いためである。SARCHは、1サイクルに1つのメモリ・アクセスしか実行できない。そのため、Dhrystoneは、性能向上が難しいプログラムである。

ループ最適化を行なうに当たって、ループ展開によ

表1　Dhrystoneの命令別出現頻度 (%)

最適化	基本	基本 +ループ	基本 +ループ +Load/Store
arithmetic	42.3	39.1	46.8
load/store	38.6	40.9	40.3
control	19.1	20.0	12.8
総コード長	679	683	679

るコード長の増加が懸念されたが、結果は数コードの増加にとどまった(表1)。これは、ループ展開を適用できたループが少なかったこと、遅延スロットが効率良く利用できたためである。

図5に性能評価結果を示す。性能向上率は、このスカラ・マシンの結果を1として各々のスピードの性能向上を示したものである。また、並列度は実行全体を通して平均何個の命令が同時に実行されたかを表したものである。図5に示したように、性能向上率は基本最適化を行なったコードで1.93倍、さらにループ最適化を行なったコードで2.09倍、そして前2項目に加えLoad/Store最適化を行なったコードでは3.2倍となった。

また、どのようなハザードのために命令が発行されなかったかを把握するために、ハザードの分布を調査した。結果を図6に示す。ハザードの数は、ハザードによって発行できなかった命令の数である。ハザードは以下のように分類した：

data　：RAW及びWAWハザード
ld delay：ロード遅延
DC　：データ・キャッシュのアクセス競合
br delay：分岐遅延

基本最適化において、データ・キャッシュ競合やロード遅延によるメモリに関するハザードが約69%も占めている。この原因としては、この原因は、Dhrystoneにおいて、メモリ・アクセスの命令の頻度が高いのに対して、SARCHでは1サイクルに実行できるメモリ・アクセス命令が1個であるためである。

ループの最適化を行なったコードでは、では、最適化を行なう前と比較して約8.3%の性能向上が見られた。これは、図6からわかるようにロード遅延によるハザードの減少が大きく影響していると考えられる。ロード遅延が減少した理由は、ループの最適化が適用できたところは、文字列操作部分であったためであったこと、そして、ループ展開によりロード命令の遅延スロットをうまく使用できたためである。基本ブロックが短く、遅延スロットを埋めることができないようなループに特に効果があると考えられる。

Load/Store命令の最適化により約54%の性能向上が得られた。このような大きな性能向上が得られた第1の原因は、Load/Store命令の減少による実行コード数の減少であると考えられる。特にDhrystoneは、文字列操作命令が多いため、このような大きな効果となって現われた。SARCHでは、メモリ・アクセスに制限があるので、メモリ・アクセス命令の減少は、性能向上に大きく寄与する。第2の原因は、分岐回数の減少にあると考えられる。図6からわかるように分岐遅延によるハザードが約半分に減少している。

静的にコード・スケジュールを行ない、イン・オーダに命令を発行するSARCHでは、コンパイラによるコード・スケジュールが性能に及ぼす影響は大きい。以上述べたように基本最適化を行なったコードの性能と比較して、ループの最適化を行なったコードでは8%、さらにLoad/Storeの最適化を行なったコードでは54%の性能向上が得られた。

参考のために、VAX-11/780相対MIPS値を示す。但し、動作周波数を50MHzとした。基本最適化を行なっ

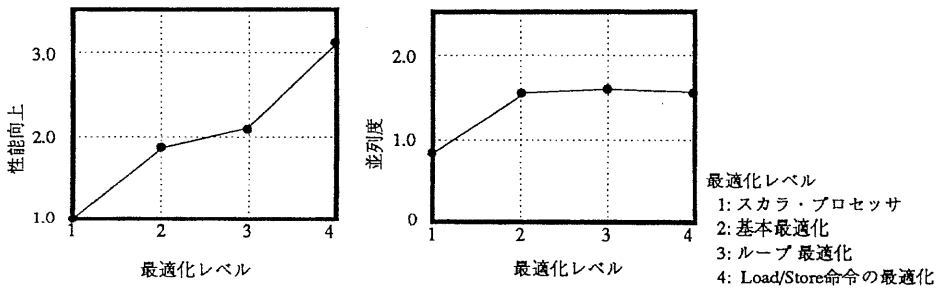


図5　性能向上

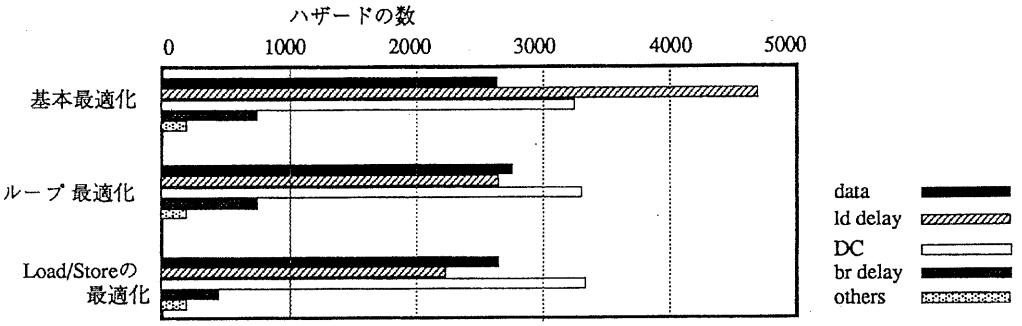


図6　ハザードの分布

たコードで83MIPS、さらにループの最適化、Load/Store命令の最適化を行なったコードでは、137MIPSの性能となる。

7. おわりに

非科学計算用途を目指したスーパスカラでは、命令レベルの並列性の抽出と、分岐遅延による性能低下の緩和は設計のキーになる。本論文では、これらの問題をブースティングと遅延分岐により解決したアーキテクチャを用いたSARCHの概要について述べ、実現方法について示した。性能評価により、SARCHは、基本的最適化を行なったコードで、スカラ・マシンの1.9倍、ループの最適化、Load/Store命令の最適化を行なったコードで、3.2倍の性能向上が確認できた。ループの最適化、Load/Store命令の最適化を行なうことにより、基本最適化の約54%の性能向上が得られた。

動作周波数を50MHzとした場合、VAX-11/780相対MIPS値で137MIPSの性能を有していることが確認できた。

参考文献

[安藤 91] 安藤、中西、中屋：" ブースティング及び命令キューを用いた遅延分岐方式によるスーパスカラスカラのアーキテクチャ" 情報処理学会「1991年並列/分散/強調処理に関する「大沼」サマー・ワークショップ(SWoPP 大沼 '91)」、情処研報,vol.91, no.64, pp.33-40 (1991年7月).

[Chralessworth 81] A.E.Charlessworth, "An Approach to Scientific Array Processing : The Architectural Design of the AP-120 B/FPS-164 Family", IEEE Computer, vol.14, no.9, pp.18-27 (Sept. 1981).

[Lam 88] M. Lam, "Software Pipelining: An Effective Scheduling Technique for VL IW Machines", Proceeding of ACM SIGPLAN '88 Conference on Programming LanguageDesign and Implementation , pp.318--328 (June. 1988).

[M.Smith 89] M. D. Smith, M. Johnson, and M. A. Horowitz, "Limits on Multiple Instruction Issue", Proc. 3rd Int'l Conf. Architectural Support for Programming Languages and Operation Systems , pp.290-302 (Apr. 1989).

[M.Smith 90] M.D. Smith, M. S. Lam, and M. A. Horowitz, "Boosting Beyond Static Scheduling in a Superscalar Processor," Proc. 17th Annu. Int'l Symp. Comput. Architecture , pp.344-354 (May 1990).

[中島 91] 中島、中野、中倉、吉田、後井、中居、瀬川、岸田、廉田：" スーパスカラスカラ・マイクロプロセッサOHMEGAにおける動的ハザード解消機構と高速化手法" 情報処理学会「1991年並列/分散/強調処理に関する「大沼」サマー・ワークショップ(SWoPP 大沼 '91)」、情処研報,vol.91, no.64, pp.25-31 (1991年7月).

[納富 91] 納富、原、久我、村上、富田：" DSN型スーパスカラ・プロセッサ・プロトタイプーアーキテクチャ及び静的コード・スケジューリングに関する総合評価ー、" 並列処理シンポジウムJSPP'91論文集、pp.125-132（1991年5月).

[Hennessy 90] J. Hennessy and D.A. Patterson, "Computer Architecture: A Quantitative Approach," Morgan Kaufmann Publisher, Inc. (1990).

[Sohi 87] G. S. Sohi and S. Vajapeyam, "Instruction issue logic for high-performance, interruptable pipelined processors", Proc. 14th Annual International Symposium on Computer Architecture(1987).