計算機アーキテクチャ　89－5
（１９９１．　７．　１９）

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・プロセッサのアーキテクチャ

安藤秀樹　中西知嘉子　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、サイクル・タイムを犠牲にすることなく解決しなければならないスーパスカラの課題である。我々は、単純なハードウェアで命令レベルの並列性を抽出するために、ブースティングを採用した。また、動的にハザードを解消するスーパスカラにおける分岐遅延による性能低下の緩和に、命令キューを用いた遅延分岐方式を採用した。遅延スロットは分岐先命令で埋め、分岐時には遅延が生じない。分岐しない時には、プリフェッチされた遅延スロットに続く命令を実行する。ベンチマークによる評価によって、スカラ・プロセッサの1.6から1.8倍の性能を確認した。

A Superscalar Architecture using Boosting and Delayed Branch

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

Exploiting instruction-level parallelism and alleviation of performance degradation by branch delay are issues which should be solved without cycle time penalty. We adopt instruction boosting[M.Smith 90] to extract instruction-level parallelism with a simple hardware. And a delayed branch with an instruction queue reduces branch delay. Delay slots are filled by branch target instructions, and are executed when branch is taken. Sequential successors after delay slots are pre-fetched in the queue before execution of a branch instruction, and executed when branch is not taken. Performance evaluation shows that the speedup is 1.6-1.8 times over scalar machines.

1. はじめに

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行なわれている。スーパスカラ設計における最初の問題は命令レベルの並列性の抽出である。科学計算プログラムでは、プログラム自身の並列性が高く、また、ループ・アンローリング等のコンパイラ最適化手法も適用しやすく、並列性の抽出は容易である。しかしながら、非科学計算プログラムにおいては、基本ブロックのコード・サイズは小さく、命令の並列性を、基本ブロックの中だけで引き出すだけでは、十分な性能は得られない。従って、基本ブロックを越えてコード移動を行なう必要がある。

スーパスカラ設計の次の問題は、分岐遅延である。スーパスカラでは、1サイクルに複数の命令が発行されるので、理想的なマシンに対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、性能に対する影響が大きい。

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た。本論文では、SARCH (Superscalar ARCHitecture)と呼ぶアーキテクチャを提案し、これらの問題の解決方法を示す。

2. コード・スケジューリング

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・サイズは小さく、中央値は約4である。このように基本ブロックのコード・サイズの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、従って、基本ブロックを越えてコード移動を行なう必要がある。

IBM360/91の[Tomasulo 67]のアルゴリズムのような動的な命令スケジューリングによって、out-of-orderに命令を発行し、さらに、分岐命令を越えて命令実行を行なうために、条件付き実行を行なう方式がある[Murakami 89]。この方式では、スーパスカラ・プロセッサが実行時に命令レベルの並列性を抽出し、命令の同時実行が行なわれることから、すでにあるスカラ・プロセッサのプログラムが高速に処理される可能性はある。しかし、複雑なハードウェアが必要となり、クロック・レートを遅くする原因となり、高速化は難しい。また、フェッチされた命令の中から並列に実行できる命令を発見しなければならないため、限られた範囲内でしかスケージュールできないので、十分な並列性の抽出は難しい。また、基本ブロックのコード・サイズの小さなプログラムにおいては、命令フェッチが十分な速度で行なわれず、性能を著しく下げてしまう[M.Smith 89]。さらに、out-of-orderに命令が実行されるために、正確な割り込みを実現することが難しく[J.Smith 88]、例えば、in-orderに書き込みを行うためにreorder bufferを設けると、そこにボトルネックが生じる[久我 90]。

動的なコード・スケジュールに対して、静的なコード・スケジュールでは、コンパイラがコードをスケジュールし、ハードウェアはフェッチされた順に命令を発行しする[M.Smith 90][原 90]。この方式では、実行時に命令の並列性の抽出の必要がなく、ハードウェアは簡単である。従って、サイクル・タイムを延ばすことなくスーパスカラを実現できる可能性がある。しかし、基本ブロックのコード・サイズは非常に小さいので、コンパイラは、分岐命令を越えてコードを移動し、ブロックのコード・サイズを大きくし、命令レベルの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・パイプライニングを用いて基本ブロックのコード・サイズを大きし、並列度を上げることができる。この方法は、科学計算のプログラムでは非常に有効な手法である[納富 91]。しかし、科学計算以外のプログラムにおいては、実現が難しいだけでなく、ループ自身がプログラムの速度を決めるパスでない場合が多く、適用性にも疑問がある。さらに、積極的にブロックのコード・サイズを大きくしようとするアルゴリズムにTrace Scheduling[Ellis 86]がある。この方法では、分岐予測に基づいていくつかの基本ブロックから、1つの大きなコード・ブロックを作成し、そこでコード・スケジューリングを行なう。しかし、この方法では、実行のパスにかかわらず正しい結果が得られるようにするために補正コードが必要であり、これが困難であるばかりでなく、分岐が予測通りに実行できなかったときのオーバヘッドが大きい。

これに対して、[M.Smith 90]は、ブースティングという方式をTORCHというプロセッサで提案している。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し、移動された命令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す。ハードウェアは、移動された命令も含めてコードの順に発行していく。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。この方式は、プログラムの性質によらず適応でき、特に、if-then-elseの多い非科学計算のプログラムには有効である。また、移動された命令の副作用の処理はハードウェアの責任となっているが、移動された命令は、コードの中で明示的に示されているので、ハードウェアの負担は少なく、従って、クロック・レートを下げる原因とはなりにくい。

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す。この分岐予測に基づくブースティングは、プロファイルを取る等して、静的な分岐予測が高い精度で行なえることに基づいている[McFarling 86]。しかしながら、静的な分岐予測の精度が高いためには、分岐方向に偏りのあることが必要であり、プログラムに依存する。Stanfordのベンチマークでは、静的分岐予測のヒット率は、平均的には75%であるが、60%から90%の範囲でばらつきがある。また、さらに、より多くの並列度を引き出すには、ブーストする命令をNot Taken側かTaken側かを選択するのではなく、どちらもブーストし、同時実行の機会を多く作りだすことが望ましい。これを行なうことによる欠点は、どちらの基本ブロックからブーストしたかを示す命令コードのビットと、副作用のためのシャドウ・レジスタの増加に、ほぼ留まる。我々のこれまでの評価では、レジスタへの書き込み以前にブースティングされた命令をキャンセルすることが可能であり、シャドウ・レジスタも不要である可能性は高い。SARCHでは、分岐方向の両側からのブースティングによる性能向上は、これを行なうことによるハードウェアの増加の損失を上回ると考え、これを採用した。

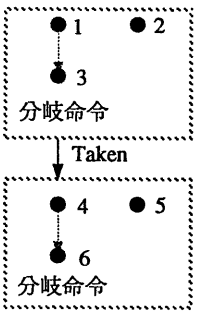
3. 分岐遅延

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。次のような簡単な例を考える。理想的には、CPIが0.5のスーパスカラがあったとする。分岐遅延を1サイクルとすると、3命令に1つの分岐命令を含むプログラムをこのスーパスカラで実行した場合、図1の様に実行される。従って、この場合、CPIは1.0となり性能の低下は50%である。

分岐遅延が性能に及ぼす影響を定量的に解析するために、次のようなモデルを考える：

1) 分岐遅延のない状態では、1命令の実行に必要なサイクル数はCPIidealである。
2) 1サイクルにN命令フェッチし、N命令デコードする。このとき、N命令の中にハザードのある分岐命令があれば、分岐予測に従って命令のフェッチを行なう。
3) 命令のアドレスを命令キャッシュに送り出し、その命令が発行されるまでに少なくとも必要なサイクル数はCbrである。

このマシンにおいては、分岐命令の実行に必要なサイクル数は以下のように計算される：



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

1) Not Taken予測／Not Taken実行
　パイプラインは乱れないので、分岐命令実行に必要なサイクル数は、CPIidealである。
2) Not Taken予測／Taken実行
　分岐先命令をフェッチするため、分岐に必要なサイクル数Cbrである。
3) Taken予測／Not Taken実行
　Not Taken側の命令をフェッチしなおすので、分岐に必要なサイクル数Cbrである。
4) Taken予測／Taken実行
　分岐命令のハザードが解消されたサイクルに分岐先命令が届いているかどうかに依存する。第n番目の命令レジスタに格納されている分岐命令によって分岐予測が行なわれ、分岐先命令のフェッチが開始された場合、その分岐命令のハザードが解消されるまでのサイクル数は、n x CPIideal である。従って、分岐命令の実行に必要なサイクル数はCbr-(n-1)xCPIidealとなる。分岐命令のハザードが解消されたサイクルに分岐先命令が届いていない場合に必要なサイクル数の平均は、

$$\sum_{n=1}^{N}[Cbr-(n-1)xCPIideal]/N = Cbr-(N-1)xCPIideal/2 \quad ..(1)$$

　である。また、分岐命令のハザードが解消されたサイクルに分岐先命令が届いている場合に必要なサイクルは、CPIidealであるので、分岐命令実行に必要なサイクル数は、上式とCPIidealの大きいほうである。

Taken予測／Taken実行の場合で、分岐命令のハザードが解消されたサイクルに分岐先命令が届いているた

表1　分岐命令に関するパラメータ

理想マシンでのCPI	CPIideal	0.5
分岐に必要なサイクル数（分岐遅延+1）	Cbr	2
命令レジスタの数	N	4
分岐命令の出現頻度	BR	0.2
Not Takenと予想される分岐命令の割合	BN	0.3
Takenと予想される分岐命令の割合	BT	0.7
Not Taken予測のヒット率	HN	0.8
Taken予測のヒット率	HT	0.8

めには、式(1)≤CPIidealであることが必要となるが、この場合の命令レジスタの数Nは、N≥4/CPIideal - 1である。CPIidealを例えば0.5とすると、Nは7以上であることが必要となるが、基本ブロックのコード・サイズはその半分程度であるので、たとえ、命令レジスタの数を増やしても効果は少なく、分岐命令のハザードが解消されたサイクルに分岐先命令が届いている確率は低いと考えられる。従って、Taken予測／Taken実行の場合は、分岐命令実行に必要なサイクル数は実質的には、式(1)である。

表1に示すパラメータを用いれば、分岐命令の遅延を考慮したCPIは、

$$CPIreal = (1 - BR) \times CPIideal + BR \times Cbr - BR \times [(Cbr - CPIideal) \times (BN \times HN) + (N - 1) \times CPIideal / 2 \times (BT \times HT)] \quad ...(2)$$

である。実際に値を代入すると、CPIrealは、

$$CPIreal = 0.64$$

である。分岐命令によって性能は、理想マシンの性能の22%低下する。

分岐遅延による性能低下を緩和する別の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。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であるが、この遅延スロットが埋められて、かつ、動的に有意である確率は約0.5である[Patterson 90]。スーパスカラの場合、分岐遅延を1サイクルとすると、遅延スロットを埋めるために必要な命令は少なくとも、1/CPI必要となる。このような複数の遅延スロットを有意に埋められる確率は低いと考えられるが、ブースティングを採用すれば、ブーストされた命令は、元々は移動先の基本ブロックよりも下にあったのであるから、遅延スロットを埋められる可能性は高い。しかしながら、ハザードを動的に解消するスーパスカラでは、分岐遅延スロットの数は動的に変化するため、遅延分岐を適用することは難しい。TOACHでは、フェッチ／実行する命令の数を2に固定して、ハザードの解消も静的に行なっており、そのため、遅延スロットは固定的である。しかし、非科学計算プログラムであっても、命令レベルの並列度は2程度であるので[Jouppi 89]、実行する命令の数を2にすることはプログラムの最大の並列性を引き出すことができない。フェッチする命令の数を2以上にすると（命令キャッシュの構成上、2の次は4であろうが）、ハザードの解消を静的に行なうためには、かなりの数のnopを必要としてコード・サイズが大きくなる。

SARCHでは、ハザードを動的に解消するスーパスカラにおいて遅延分岐を実現している。以下に示す遅延分岐方式を提案する：

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2）。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。
   - 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。
   - 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、

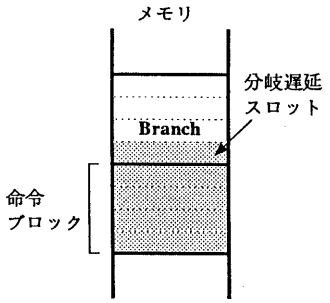


図2　分岐遅延スロット

・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、遅延スロットの次のエントリを指すように移動される。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。

この方式では、遅延スロットはTaken側と固定しており、これを埋めることは容易である。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、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少なくとも遅延スロットがフェッチできれば、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、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。

この遅延分岐方式の効果を定量的に解析する。追加のパラメータとして、Not Takenの場合に遅延スロットがフェッチできていない確率をWとする。Not Takenの場合は、命令デコーダ内の分岐命令の後方の命令が無効化されるので、命令デコーダ内の第n番目の命令が分岐命令であったとき、この分岐命令の実行に必要なサイクル数は1-(n-1)xCPIidealである。従って、分岐命令の命令デコーダ内での位置をコンパイラが制御しない最悪の場合において、Not Takenで、遅延スロットの命令がすでにフェッチされている場合の分岐命令の実行サイクルは、

$$\sum_{n=1}^{1/CPIideal}[1 - (n-1)xCPIideal] \times CPIideal = (1 + CPIideal) / 2$$

となる。Not Takenで、分岐命令実行のサイクルで遅延スロットの命令がまだフェッチされていない場合、分岐命令実行サイクル数は、上記の値にさらに1サイクルかかる（分岐遅延は1サイクルとしている）。よって、CPIrealは、

$$CPIreal = [CPIideal \times (1-BRxBN/2) + BRxBN/2] + WxBRxBN \quad ...(3)$$

となる。実際のパラメータを代入すると、

$$CPIreal = 0.52 + 0.06 \times W$$

となる。Wの値は、プログラムやコンパイラやパイプラインの本数に大きく依存するが、これまでのところ0.03から0.63の範囲を取っている。例えば、平均を取ってWを0.33とすれば、CPIrealは、

$$CPIreal = 0.54$$

であり、分岐命令による性能低下は、理想マシンの7%に押さえることができる。

4. ハードウェア構成

図3にSARCHのハードウェア構成を示す。パイプラインは5段である。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

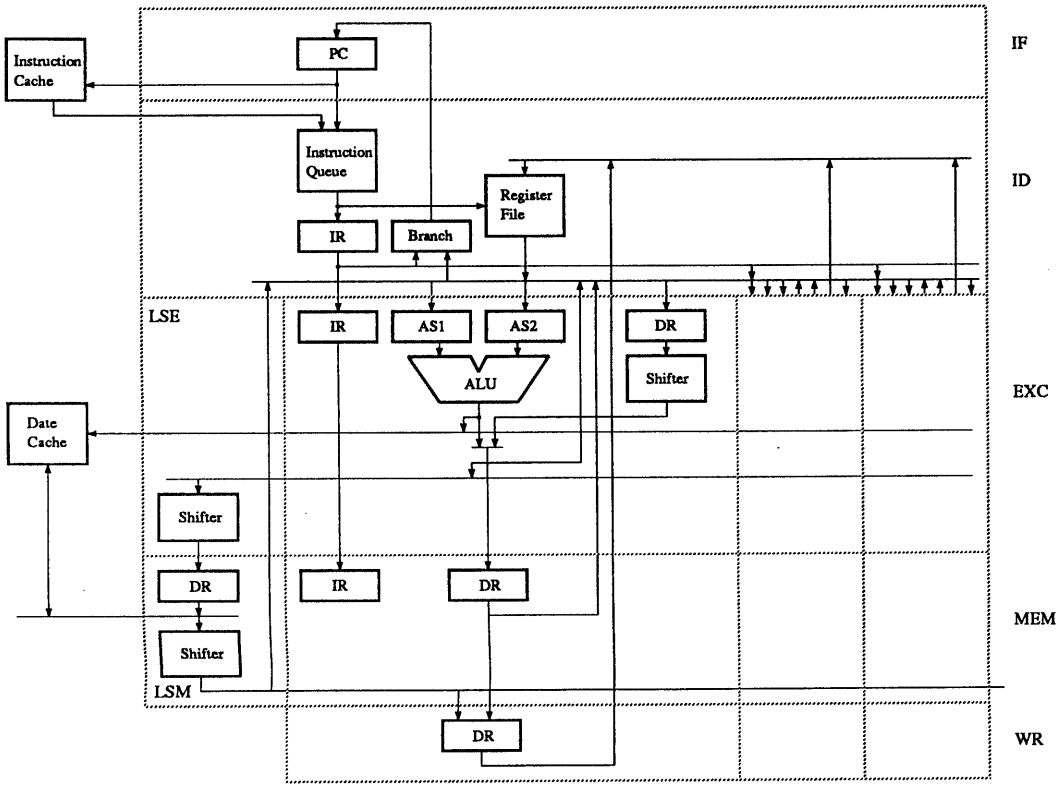


図3　SARCHのブロック図

は、再び、元のパイプラインに戻される。LSE/LSMは1つである。従って、この構成においては、例えば、3本のパイプラインでは、2つのALU演算命令と1つのLoad/Store命令、あるいは、3つのALU演算命令という組み合わせで実行を行なう。

あるパイプラインが処理できる命令のグループを命令クラスと呼ぶと、ある命令クラスを処理するパイプラインの数が同時に発行できる命令の数より少ない場合、同一の命令クラスに属する命令は、一度には、パイプラインの本数しか発行できない。このクラス競合は性能を数十%にわたって低下させる[Jouppi 89]。しかしながら、Load/Store命令に関しては、クラス競合を回避しようとすると、データ・キャッシュのアクセスというリソース競合が生じる。例えば、市販のSRAMを用いて、データ・キャッシュの2ポート化を行なうことは可能であるが、データ・キャッシュのアクセスはマシンのクリティカル・パスであるから、サイクル・タイムを延ばさずに実現することは難しい。また、LSIパッケージのピン数の制限もある。チップに内蔵することも可能であるが、キャッシュ・メモリの2ポート化は、1ポートのメモリに対して倍近い面積ペナルティがあり、データ・キャッシュのアクセス競合を解消するコストとしては、あまりに大きすぎる。ALU演算命令に関しては、次章の評価においてパイプラインの本数を変化させて評価を行なっている。

5. 性能評価

ベンチマークには、Stanfordのベンチマーク・プログラムを用いた。Stanfordのベンチマーク・プログラムは、quick sort, bubble sort, queens, towers, perm, tree

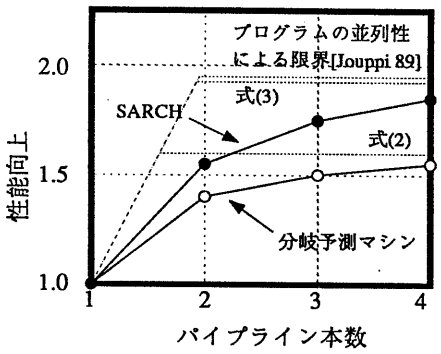


図4　性能向上

sort, intmmからなるが、SARCHは、非科学計算用途において性能を向上させることを第1の目的としているので、この中から、行列乗算を行なうintmmを除いた。

性能比較を行なうために、同一のパイプライン構造で、命令ブースティングを行なわず、4つ先までの分岐命令を予測により実行するスーパスカラについても評価を行なった。この「分岐予測マシン」のパイプライン本数が1の場合を基本マシンとした。性能比較は、MIPSのCコンパイラで最適化を行なったコードを基本マシンで実行した場合の性能を基準とした。基本マシンでは、CPIは1.04であり、現在のスカラ・マシンの性能に近いものである。

図4に性能向上を示す。[Jouppi 89]によれば、Stanfordのベンチマーク・プログラムの並列度は1.9程度であるので、SARCHはそれに近い値を示している。また、分岐予測マシンに対して、SARCHは13-20%性能が勝る。さらに、SARCH、分岐予測マシン共に、式(2), (3)による見積値と一致した性能を示しており、見積の妥当性を示している。

図5に、quick sortにおいて、同時に発行された命令数の分布を示す。分岐予測マシンでは、予測によって分岐が実行されることがあるので、パイプライン本数より1多い命令発行の状況がある。SARCHでは命令をブースティングしているので、パイプラインの使用率が高い。また、分岐遅延が主たる原因である発行命令数がゼロであるサイクルは、SARCHでは、分岐予測マシンに対して、12-22%減少している。

図6にquick sortにおけるハザードの分布を示す。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：

br delay：分岐遅延
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
inst align: 命令キャッシュは4命令境界からしかアクセスできない。この制限は命令キャッシュの構成を簡単化し、サイクル・タイムが延びることを防ぐ。4命令境界にない命令をフェッチしようとした場合不要な命令が送られてくるが、不要な命令は命令キューに取り込むときに無効化する。ハザードinst alignの数とは、このような制限がなかったときに、あった場合に比べて余分に発行できた命令数である。

図6より、分岐予測マシンに比べて、SARCHは分岐遅延によるハザードbr delayが大幅に減少していること

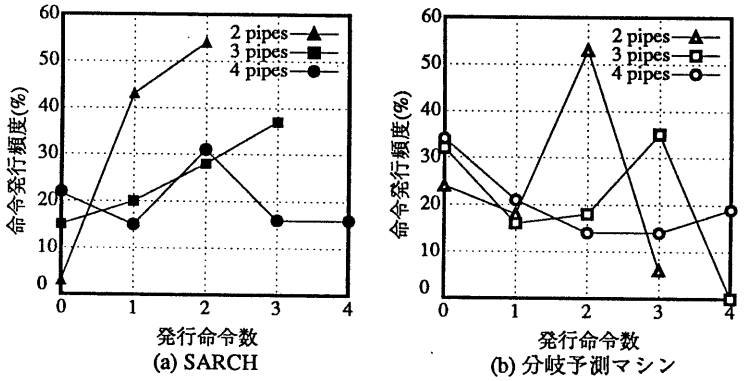


図5　並列に発行された命令数の分布(quick sort)

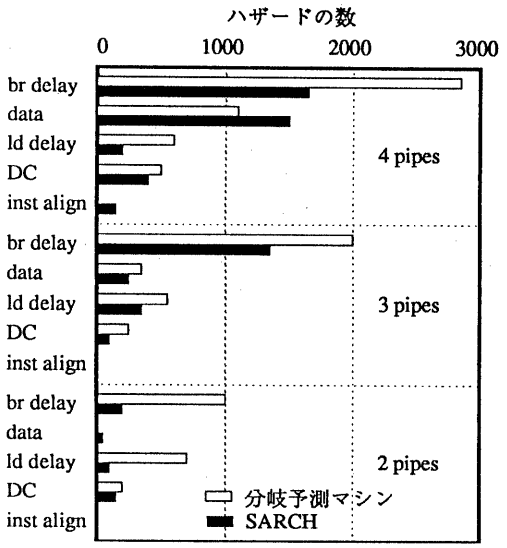


図6　ハザードの分布(quick sort)

がわかる。例えば、3本のパイプラインでは、68%に減少している。また、br delay以外のハザードは、3本のパイプラインでは、56%に減少している。これは、ブースティングにより命令レベルの並列度を上げた効果である。

6. 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が設計のキーになることを示し、本論文では、ブースティングと遅延分岐によりこれらの問題を解決したことを示した。分岐遅延に関しては、予測分岐方式と遅延分岐方式における性能を見積り、その上で、SARCHの採用した遅延分岐方式の優位性を示した。SARCHの遅延分岐方式は、動的にハザードを解消するスーパスカラに適応した遅延分岐方式であり、命令をプリフェッチするキューを用いて実現した。ベンチマークによる性能評価により、SARCHは、スカラ・マシンの、1.6から1.8倍の性能であることが確認できた。

参考文献

[Tomasulo 67] R. M. Tomasulo, "An Efficient Hardware Algorithm for Exploiting Multiple Arithmetic Units," IBM J. Res. Develop., pp.25-33 (Jan 1967).

[Murakami 89] K.Murakami, N. Irie, M. Kuga, and S. Tomita, "SIMP(Single Instruction stream / Multiple instruction Pipelining): A novel High-Speed Single--Processor Architecture," Proc. 16th Annu. Int'l Symp. Comput. Architecture , pp.78-85(May 1989).

[久我90] 久我、入江、村上、富田：" SIMP（単一命令流／多重命令パイプライン）方式に基づくスーパスカラ・プロセッサ「新風」の性能評価、" 並列処理シンポジウムJSPP'90論文集（1990年5月），pp.337-344 .

[原 90] 原、納富、久我、村上、富田：" SIMP（単一命令流／多重命令パイプライン）方式に基づく改良型スーパスカラ・プロセッサの構成と処理、" 信学技報「1090年並列処理に関する「琉球」サマー・ワークショップ（SWoPP琉球'90）」、CPSY-90-55（1990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）.

[Ellis 86] J. R. Ellis, "Bulldog: A Compiler for VLIW Architectures," MIT Press, 1986.

[McFarling 86] S. McFarling and J. Hennessy,"Reducing the Cost of Branches," Proc. 13th Annu. Int'l Symp. Comput. Architecture , pp.396-404 (June 1986).

[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).

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

[Jouppi 89] N. P. Jouppi, "The nonuniform Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance," IEEE trans. Comput., vol.38, No.12, pp.1645-1658 (Dec 1989).

[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).

[J.Smith 88] J. E. Smith and A. R. Pleszkun, "Implementing Precise interrupts in Pipelined Processors," IEEE Trans. Comput., vol.37, pp.562-573 (May 1988).