大規模投機処理を特徴とする
マイクロプロセッサ・アーキテクチャ
— VLDP —

中村 友洋

概要

マイクロプロセッサは1971年の4004誕生以来これまで四半世紀の間、デバイス技術の進歩とアーキテクチャの進歩という両輪と、そして何より高性能化への強い要求に支えられて発展を続けてきた。デバイス技術に関してはMooreの法則でいわれるように3年間でチップのトランジスタ数が4倍になるというペースがほぼこれまで続いている。それに伴って現在ではオンチップ・キャッシュの搭載、命令レベル並列性を利用するスーパースカラ方式・VLIW方式などのチップが登場してきた。

近年のプロセッサ・アーキテクチャにおいては、スーパースカラ方式が主流となり、ローカルな命令レベル並列性をより利用するために、out-of-order実行、投機的実行、データフォワーディングなどの機構が追加されてきた。さらに2次キャッシュのオンチップ化なども見られる。しかし、数年前からスーパースカラ方式の限界が様々指摘されている。元々スカラ・プロセッサの拡張として作られてきたため、旧来の方式へアドホックに機能を追加してきた面もあるのは否めない。

最近のマイクロプロセッサ市場を見ると、PC市場ではIntel社が市場を独占している状態に近い一方で、ゲーム機・携帯情報家電や組み込み用途のプロセッサなどの市場では、複数のメーカーが競合している状態が見られる。汎用マイクロプロセッサに関しては、明らかに寡占状態になりつつあり、マイクロプロセッサの研究開発者は、この寡占状態の市場に勇気を持って挑戦するか、それとも特定用途向けのプロセッサをターゲットとするかの選択を迫られている。しかしながら、マイクロプロセッサの歴史を見てみると、これだけの短期間に激しい変革を起こしてきた世界である。また性能・コストに非常にシビアな世界でもある。革命的な技術の発明により、ある日市場が急転する可能性も秘めている。より高性能なプロセッサに向けた基礎的な研究はやめるべきではない。

現在、21世紀に向けてより長期的な視点から、大規模なVLSIの利用方法に関する議論が様々行なわれている。オンチップ・マルチ・プロセッサや、DRAM・ロジッ

ク混載などによるメモリとCPUの1チップ化、もしくはI/OとCPUの1チップ化を行なったシステムオンチップと呼ばれるアプローチなどがその代表例である。これらの技術により既存の並列計算機やコンピュータ・システムの機能に相当するものが少数のチップにより構成され、例えば並列計算機における通信オーバーヘッドや、PC・ワークステーションなどにおけるメモリ・ウォール問題などが改善されることで高性能化が期待されている。しかしながら、これらは従来のCPU・メモリ・周辺機器を1チップ化することでシステムとしての性能を上げることがメインで、必ずしもCPUのもつ性能自体を改善するというのではない。CPUのもつ性能自体を上げる努力をしなければ、CPU性能がボトルネックとなりシステム全体の性能が上がらなくなるのは明らかである。

このような現状を鑑み、本研究では、今後おおよそ10年後のデバイス技術を仮定した上で、CPU自体の性能向上を目指した新しいプロセッサ・アーキテクチャとして大規模データパス(Very Large Data Path)の開発を行なう。

マイクロプロセッサの性能向上には、命令レベルの並列性の利用が欠かせない。プログラムの処理の本質はデータフローであり、データーフローをもとに実行を行なえれば、自然と命令レベルの並列性が抽出される。しかし、現在のアーキテクチャの多くはコントロールフローを中心にして実行を行なっているため、命令レベルの並列性の抽出が難しくなっている。本研究では制御依存関係を解消した命令を大量に実行部に送り高速処理を行なう方式の開発を目的としている。実行部としては、多数のALUをネットワーク状に結合したALU-NETと呼ぶ機構を設ける。このような大規模なデータパス部分をもったアーキテクチャを、大規模データパス・アーキテクチャと呼ぶ。

本研究では、大規模データパス・アーキテクチャの概要についてまとめ、その中で特に制御依存関係による制約を可能な限り取り除く方式の検討を行なった。この機構をコントロールフロー先行展開と呼ぶ。コントロールフロー先行展開の特徴は大規模な投機処理にある。マイクロプロセッサにおいて、コントロールフローの複数パスへの投機フェッチを行なうことで、分岐命令による制御依存関係を効率的に解消することを目指している。

複数パスへの投機フェッチにおいては、どのような命令の展開方式をとるかが重要な問題である。単一パスへの命令展開では、分岐予測の影響を受けやすく、分岐予測に失敗したときには大きなペナルティーが課せられる。一方すべてのパスを展開したのでは、投機規模を大きくするに従って不要な命令が爆発的に増加してし

まう。そのため効率的な命令展開を行なうには、選択的な複数パスへの投機フェッチが必要である。

投機フェッチの効率化には分岐予測性能の向上も欠かせない。しかしながら、これまでにさまざまな分岐予測機構の検討が行なわれてきた結果、最近ではさらなる性能向上はあまり望めなくなっている。ただし、これまでの分岐予測機構は単一パスへの投機フェッチを想定して設計されてきたので、コントロールフロー先行展開で行なうような複数パスへの投機フェッチに対応した分岐予測機構の検討が必要である。本研究では、投機フェッチに伴う履歴情報の更新遅れ、複数パスへの分岐予測の 2 点について調査をおこなった。

動的な分岐履歴情報を利用する分岐予測機構では、投機フェッチによって先行する分岐命令の分岐予測を行なう場合、古い履歴情報をもとに分岐予測を行なうことになるため、この影響について調査を行なった。この結果、10 段程度先までの分岐命令に対する予測性能に対して大きな影響は見られなかった。

また、レジスタ間接アドレッシングで分岐先が決まる分岐命令は、その分岐先が 2 つ以上になる可能性をもっているため、複数パスへの投機フェッチに際しては、この複数の可能性の中から、確率の高い複数のパスの候補を取り出す分岐予測機構が有効であると考えられる。本研究では、1 つの分岐先をキャッシングする BTAC(Branch Target Address Cache) を拡張し、複数の分岐先をキャッシングする Multi BTAC を提案し、評価を行なった。その結果従来の BTAC に比べて、分岐予測成功率で 10〜30 %程度の向上が見られ、さらに複数パスへの投機フェッチも Multi BTAC を使って可能になった。

以上のような分岐予測機構を用いて複数パスへの選択的投機フェッチを行なうのがコントロールフロー先行展開であるが、この際にコントロールフローの中からどのパスを選択するかが重要な問題である。理想的には、各パスへの実行確率を計算し最も可能性の高いパスから投機フェッチをするのがよいと思われるが、現実的に実装することは困難である。そこで、コントロールフロー先行展開では実装を考慮し、軽い処理で近似的に効率的な選択的投機フェッチが可能な実行確率管理機構を開発した。

その結果、プログラムや投機規模にもよるが、シングルパスへの投機フェッチに比べて 1.5 倍〜6 倍程度の命令フェッチ量で、シングルパス・モデルで得られる平均並列度とオラクルモデルで得られる平均並列度の中間程度の並列度が得られることを確認した。

全体として、理想的にはスカラプロセッサの6倍～8倍の速度向上が可能であることを示した。

目 次

1 序論 **1**

1.1 研究の背景 1

1.2 研究の目的 2

1.3 本論文の構成 3

2 マイクロプロセッサ **5**

2.1 デバイス技術 5

2.2 アーキテクチャ 9

2.2.1 汎用マイクロプロセッサ 9

2.2.2 メディア・プロセッサ 11

2.2.3 省電力プロセッサ 14

3 高性能化をめざすアーキテクチャ **17**

3.1 スーパースカラ 18

3.2 DRAM・ロジック混載プロセッサ 19

3.3 オンチップマルチプロセッサ 21

3.4 大規模 CPU 25

4 高性能化への問題と最近の研究 **27**

4.1 制御依存関係による制約 27

4.1.1 分岐予測機構 27

4.1.2 条件付き実行 29

4.1.3 複数パスの投機フェッチ 32

4.1.4 命令フェッチ能力 41

4.2 データ依存関係による制約 41

5 大規模データパス・アーキテクチャの提案 45
5.1 概要 45
5.2 コントロールフロー先行展開 47
5.3 データパス先行展開 50
5.4 ALU-NET 52

6 大規模投機処理の分岐予測機構への影響 55
6.1 大規模先行展開と分岐予測機構 55
6.2 分岐履歴情報の更新の問題 56
6.3 更新型 BHT の提案 57
6.4 更新型 BHT の性能評価 58

7 複数パス投機フェッチのための分岐予測機構 67
7.1 レジスタ間接アドレッシングの問題 67
7.2 Multi BTAC の提案 68
7.3 Multi BTAC の性能評価 68
7.3.1 レジスタ間接分岐命令に対する性能 68

8 コントロールフロー先行展開の提案 81
8.1 コントロールフローパス管理 81
8.2 実行予測確率管理 83
8.2.1 Bit Order 方式 84
8.2.2 Bit Cross 方式 86

9 コントロールフロー先行展開の評価 89
9.1 評価環境 89
9.2 投機レベルに関する性能評価 91
9.3 分岐履歴ビット数に関する性能評価 96
9.4 実行確率管理ビット読み出し方法に関する性能評価 104
9.5 BO 方式および BC 方式の性能比較評価 111
9.6 フェッチ命令数制限時の性能評価 115

10 考察 123
10.1 コントロールフロー先行展開に関する考察 123

10.1.1 理想的なモデルとの比較 123
10.1.2 Disjoint Eager Execution との比較 129
10.1.3 Threaded Multiple Path Execution との比較 131
10.1.4 フェッチ能力および実行部限界周波数 132
10.1.5 命令ストリーミング 136
10.2 大規模データパス・アーキテクチャに関する考察 138
10.2.1 データパス先行展開 138
10.2.2 ALU-NET 141
10.2.3 今後の VLDP 142

11 結論 **151**

謝辞 **153**

発表文献 **155**

参考文献 **157**

A コントロールフロー先行展開の性能評価詳細 **163**

図目次

2.1 1チップ当たりのトランジスタ数 6
2.2 MPU およびメモリの動作速度の変化 8
2.3 全トランジスタに対するロジックの占める割合 8
2.4 整数／浮動小数点演算性能 (SPECint/fp95) 10
2.5 PI 命令の例 13
2.6 移動通信機器の加入数 15

3.1 次世代マイクロプロセッサ 17
3.2 DRAM 混載プロセッサ・モデル 19
3.3 各モデルの性能 20
3.4 制御並列処理 24

4.1 3つの投機フェッチ戦略 34
4.2 Static Tree Heuristic 35
4.3 TME のパス管理機構 37

5.1 分岐をまたいだスケジューリングと並列性の関係 46
5.2 VLDP アーキテクチャ・ブロック図 47
5.3 コントロール・フロー先行展開のイメージ 49
5.4 ローカルなデータ・アクセス 51

6.1 命令の先行展開方式 56
6.2 実行完了ポイントとフェッチポイント 57
6.3 分岐履歴情報の更新／非更新 58
6.4 分岐履歴情報 bit の状態遷移 59
6.5 更新型／非更新型 BHT の分岐予測成功率 (CC1) 61
6.6 更新型／非更新型 BHT の分岐予測成功率 (COMPRESS) 62

6.7 更新型／非更新型 BHT の分岐予測成功率 (GO) 63
6.8 更新型／非更新型 BHT の分岐予測成功率 (IJPEG) 64
6.9 更新型／非更新型 BHT の分岐予測成功率 (LI) 65
6.10 更新型／非更新型 BHT の分岐予測成功率 (PERL) 66

7.1 分岐命令における制御の流れ 67
7.2 BTAC と Multi BTAC 69
7.3 Multi BTAC の Multi Level 別性能 70
7.4 BTAC エントリ数と性能 (フルアソシアティブ／ダイレクトマップ) 71
7.5 Multi BTAC の分岐予測成功率 (CC1) 74
7.6 Multi BTAC の分岐予測成功率 (COMPRESS) 75
7.7 Multi BTAC の分岐予測成功率 (GO) 76
7.8 Multi BTAC の分岐予測成功率 (IJPEG) 77
7.9 Multi BTAC の分岐予測成功率 (LI) 78
7.10 Multi BTAC の分岐予測成功率 (PERL) 79

8.1 コントロールフロー先行展開の処理の流れ 82
8.2 Bit Order 方式の実行確率管理 84
8.3 コントロールフロー先行展開のテーブル更新方法 86
8.4 Bit Cross 方式の実行確率管理 87
8.5 Bit Cross 方式の実行確率管理ビット登録方法 87
8.6 Bit Cross 方式の実行確率管理ビット更新方法 88

9.1 投機レベルと平均並列度 (cc1,3bit 履歴,1bit シフト,BO 方式) 92
9.2 投機レベルと分岐予測成功率 (cc1,3bit 履歴,1bit シフト,BO 方式) . 92
9.3 投機レベルと有効命令率 (cc1,3bit 履歴,1bit シフト,BO 方式) 94
9.4 投機レベルとフェッチ命令数 (cc1,3bit 履歴,1bit シフト,BO 方式) . . 94
9.5 投機レベルとフラッシュ率 (cc1,3bit 履歴,1bit シフト,BO 方式) . . . 95
9.6 分岐履歴ビット数と平均並列度 (cc1,2,3,5bit 履歴,0.5bit シフト,BO 方式) 99
9.7 分岐履歴ビット数と平均並列度 (cc1,2,3,5bit 履歴,0.5bit シフト,BC 方式) 99
9.8 分岐履歴ビット数と分岐予測成功率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO 方式) 101

9.9 分岐履歴ビット数と分岐予測成功率 (cc1,2,3,5bit 履歴,0.5bit シフト,BC 方式) 101
9.10 分岐履歴ビット数と有効命令率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO 方式) 102
9.11 分岐履歴ビット数と有効命令率 (cc1,2,3,5bit 履歴,0.5bit シフト,BC 方式) 102
9.12 分岐履歴ビット数とフラッシュ率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO 方式) 103
9.13 分岐履歴ビット数とフラッシュ率 (cc1,2,3,5bit 履歴,0.5bit シフト,BC 方式) 103
9.14 実行確率管理ビットシフト量と平均並列度 (cc1,2bit 履歴,0～1bit シフト,BC 方式) 106
9.15 実行確率管理ビットシフト量と分岐予測成功率 (cc1,2bit 履歴,0～1bit シフト,BC 方式) 106
9.16 実行確率管理ビットシフト量と有効命令率 (cc1,2bit 履歴,0～1bit シフト,BC 方式) 107
9.17 実行確率管理ビットシフト量とフラッシュ率 (cc1,2bit 履歴,0～1bit シフト,BC 方式) 107
9.18 実行確率管理ビットシフト量と平均並列度 (cc1,5bit 履歴,0.5～2bit シフト,BC 方式) 109
9.19 実行確率管理ビットシフト量と分岐予測成功率 (cc1,5bit 履歴,0.5～2bit シフト,BC 方式) 109
9.20 実行確率管理ビットシフト量と有効命令率 (cc1,5bit 履歴,0.5～2bit シフト,BC 方式) 110
9.21 実行確率管理ビットシフト量とフラッシュ率 (cc1,5bit 履歴,0.5～2bit シフト,BC 方式) 110
9.22 BO 方式および BC 方式の平均並列度 (cc1,2,3,5bit 履歴,0.5bit シフト,BO/BC 方式) 113
9.23 BO 方式および BC 方式の分岐予測成功率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO/BC 方式) 114
9.24 BO 方式および BC 方式の有効命令率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO/BC 方式) 114

9.25 BO 方式および BC 方式のフラッシュ率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO/BC 方式) 115
9.26 フェッチ命令数制限時の平均並列度 (cc1,5bit 履歴,0.5bit シフト,BO 方式) 118
9.27 フェッチ命令数制限時の平均並列度 (cc1,5bit 履歴,0.5bit シフト,BC 方式) 118
9.28 フェッチ命令数制限時の分岐予測成功率 (cc1,5bit 履歴,0.5bit シフト,BO 方式) 119
9.29 フェッチ命令数制限時の分岐予測成功率 (cc1,5bit 履歴,0.5bit シフト,BC 方式) 119
9.30 フェッチ命令数制限時の有効命令率 (cc1,5bit 履歴,0.5bit シフト,BO 方式) 120
9.31 フェッチ命令数制限時の有効命令率 (cc1,5bit 履歴,0.5bit シフト,BC 方式) 120
9.32 フェッチ命令数制限時のフラッシュ率 (cc1,5bit 履歴,0.5bit シフト,BO 方式) 121
9.33 フェッチ命令数制限時のフラッシュ率 (cc1,5bit 履歴,0.5bit シフト,BC 方式) 121

10.1 Floating モデルで得られる平均並列度 (cc1) 124
10.2 Floating モデルの有効命令率 (cc1, Δ=0.0) 126
10.3 Floating モデルの有効命令率 (cc1, Δ=0.1) 126
10.4 Floating モデルの有効命令率 (cc1, Δ=0.2) 127
10.5 Floating モデルの有効命令率 (cc1, Δ=0.3) 127
10.6 様々な命令展開方式の平均並列度比較 (cc1) 130
10.7 様々な命令展開方式の有効命令率 (cc1) 130
10.8 必要フェッチ能力 (2001 年モデル) 146
10.9 必要フェッチ能力 (2012 年モデル, 下限) 146
10.10必要フェッチ能力 (2012 年モデル, 上限) 147
10.11最大フェッチ能力 (2001 年モデル) 147
10.12最大フェッチ能力 (2012 年モデル, 下限) 148
10.13最大フェッチ能力 (2012 年モデル, 上限) 148
10.14限界周波数 (2001 年モデル) 149

10.15限界周波数 (2012 年モデル, 下限) 149
10.16限界周波数 (2012 年モデル, 上限) 150
10.17命令ストリームの例 150

A.1 フェッチ命令数制限時の平均並列度 (GO,5bit 履歴,0.5bit シフト,BO 方式) 164
A.2 フェッチ命令数制限時の平均並列度 (GO,5bit 履歴,0.5bit シフト,BC 方式) 164
A.3 フェッチ命令数制限時の分岐予測成功率 (GO,5bit 履歴,0.5bit シフト,BO 方式) 165
A.4 フェッチ命令数制限時の分岐予測成功率 (GO,5bit 履歴,0.5bit シフト,BC 方式) 165
A.5 フェッチ命令数制限時の有効命令率 (GO,5bit 履歴,0.5bit シフト,BO 方式) 166
A.6 フェッチ命令数制限時の有効命令率 (GO,5bit 履歴,0.5bit シフト,BC 方式) 166
A.7 フェッチ命令数制限時のフラッシュ率 (GO,5bit 履歴,0.5bit シフト,BO 方式) 167
A.8 フェッチ命令数制限時のフラッシュ率 (GO,5bit 履歴,0.5bit シフト,BC 方式) 167
A.9 様々な命令展開方式の平均並列度比較 (GO) 168
A.10 様々な命令展開方式の有効命令率 (GO) 168
A.11 フェッチ命令数制限時の平均並列度 (IJPEG,5bit 履歴,0.5bit シフト,BO 方式) 169
A.12 フェッチ命令数制限時の平均並列度 (IJPEG,5bit 履歴,0.5bit シフト,BC 方式) 169
A.13 フェッチ命令数制限時の分岐予測成功率 (IJPEG,5bit 履歴,0.5bit シフト,BO 方式) 170
A.14 フェッチ命令数制限時の分岐予測成功率 (IJPEG,5bit 履歴,0.5bit シフト,BC 方式) 170
A.15 フェッチ命令数制限時の有効命令率 (IJPEG,5bit 履歴,0.5bit シフト,BO 方式) 171

A.16 フェッチ命令数制限時の有効命令率 (IJPEG,5bit 履歴,0.5bit シフト,BC 方式) 171
A.17 フェッチ命令数制限時のフラッシュ率 (IJPEG,5bit 履歴,0.5bit シフト,BO 方式) 172
A.18 フェッチ命令数制限時のフラッシュ率 (IJPEG,5bit 履歴,0.5bit シフト,BC 方式) 172
A.19 様々な命令展開方式の平均並列度比較 (IJPEG) 173
A.20 様々な命令展開方式の有効命令率 (IJPEG) 173
A.21 フェッチ命令数制限時の平均並列度 (LI,5bit 履歴,0.5bit シフト,BO 方式) 174
A.22 フェッチ命令数制限時の平均並列度 (LI,5bit 履歴,0.5bit シフト,BC 方式) 174
A.23 フェッチ命令数制限時の分岐予測成功率 (LI,5bit 履歴,0.5bit シフト,BO 方式) 175
A.24 フェッチ命令数制限時の分岐予測成功率 (LI,5bit 履歴,0.5bit シフト,BC 方式) 175
A.25 フェッチ命令数制限時の有効命令率 (LI,5bit 履歴,0.5bit シフト,BO 方式) 176
A.26 フェッチ命令数制限時の有効命令率 (LI,5bit 履歴,0.5bit シフト,BC 方式) 176
A.27 フェッチ命令数制限時のフラッシュ率 (LI,5bit 履歴,0.5bit シフト,BO 方式) 177
A.28 フェッチ命令数制限時のフラッシュ率 (LI,5bit 履歴,0.5bit シフト,BC 方式) 177
A.29 様々な命令展開方式の平均並列度比較 (LI) 178
A.30 様々な命令展開方式の有効命令率 (LI) 178
A.31 フェッチ命令数制限時の平均並列度 (PERL,5bit 履歴,0.5bit シフト,BO 方式) 179
A.32 フェッチ命令数制限時の平均並列度 (PERL,5bit 履歴,0.5bit シフト,BC 方式) 179
A.33 フェッチ命令数制限時の分岐予測成功率 (PERL,5bit 履歴,0.5bit シフト,BO 方式) 180

A.34 フェッチ命令数制限時の分岐予測成功率 (PERL,5bit 履歴,0.5bit シフト,BC 方式) 180
A.35 フェッチ命令数制限時の有効命令率 (PERL,5bit 履歴,0.5bit シフト,BO 方式) 181
A.36 フェッチ命令数制限時の有効命令率 (PERL,5bit 履歴,0.5bit シフト,BC 方式) 181
A.37 フェッチ命令数制限時のフラッシュ率 (PERL,5bit 履歴,0.5bit シフト,BO 方式) 182
A.38 フェッチ命令数制限時のフラッシュ率 (PERL,5bit 履歴,0.5bit シフト,BC 方式) 182
A.39 様々な命令展開方式の平均並列度比較 (PERL) 183
A.40 様々な命令展開方式の有効命令率 (PERL) 183

表目次

2.1 マイクロプロセッサのデバイス技術予測 5
2.2 Alpha21264 の仕様 11
2.3 ESMMA の仕様 12
2.4 ESMMA の命令セット 13
2.5 メディア処理に適用される命令種類 14
2.6 用途と消費電力 14
2.7 SA-1100 の仕様 15

3.1 HARP1E の仕様 22

6.1 更新型／非更新型 BHT の性能シミュレーション環境 59

7.1 主な分岐命令の特徴 68
7.2 Multi BTAC の Multi Level 別性能シミュレーション環境 69
7.3 投機処理時における Multi BTAC の性能シミュレーション環境 . . . 73

9.1 コントロールフロー先行展開機構のシミュレーション環境 90
9.2 投機レベルに関する性能評価・シミュレーションパラメータ 91
9.3 分岐履歴ビットの読み出し方法 97
9.4 分岐履歴ビット数に関する性能評価・シミュレーションパラメータ 98
9.5 実行確率管理ビットシフト量に関する性能評価・シミュレーション
パラメータ 104
9.6 BO 方式および BC 方式の性能比較評価・シミュレーションパラメータ 111
9.7 フェッチ命令数制限時の性能評価・シミュレーションパラメータ . . 116
9.8 フェッチ命令数制限モデルと無制限モデルの比較 117

10.1 コントロールフロー先行展開実行確率管理部パラメータ 128

10.2 コントロールフロー先行展開とDEEの比較 131
10.3 コントロールフロー先行展開パラメータ 133
10.4 1ストリームのフェッチでの有効フェッチ命令数 138

第 1 章

序論

1.1 研究の背景

マイクロプロセッサは 1971 年の 4004[FJMS96] 誕生以来これまで四半世紀の間、デバイス技術の進歩・アーキテクチャの進歩という両輪と、そして何より高性能化への強い要求に支えられて発展を続けてきた。デバイス技術に関しては Moore の法則でいわれるように 3 年間でチップのトランジスタ数が 4 倍になるというペースがほぼこれまで続いている。それに伴って現在ではオンチップ・キャッシュの搭載、命令レベル並列性を利用するスーパースカラ方式・VLIW 方式などのチップが登場してきた。

近年のプロセッサ・アーキテクチャにおいては、スーパースカラ方式が主流となり、ローカルな命令レベル並列性をより利用するために、out-of-order 実行、投機的実行、データフォワーディングなどの機構が追加されてきた。さらに 2 次キャッシュのオンチップ化なども見られる。しかし、数年前からスーパースカラ方式の限界が様々指摘されている [Mich89]。元々スカラ・プロセッサの拡張として作られてきたため、旧来の方式へアドホックに機能を追加してきた面もあるのは否めない。

最近のマイクロプロセッサ市場を見ると、PC 市場では Intel 社が市場を独占している状態に近い一方で、ゲーム機・携帯情報家電や組み込み用途のプロセッサなどの市場では、複数のメーカーが競合している状態が見られる。汎用マイクロプロセッサに関しては、明らかに寡占状態になりつつあり、マイクロプロセッサの研究開発者は、この寡占状態の市場に勇気を持って挑戦するか、それとも特定用途向けのプロセッサをターゲットとするかの選択を迫られている。しかしながら、マイクロプロセッサの歴史を見てみると、これだけの短期間に激しい変革を起こして

きた世界である。また性能・コストに非常にシビアな世界でもある。革命的な技術の発明により、ある日市場が急転する可能性も秘めている。より高性能なプロセッサに向けた基礎的な研究はやめるべきではない。

現在、21 世紀に向けてより長期的な視点から、大規模な VLSI の利用方法に関する議論が様々行なわれている [坂井 97]。オンチップ・マルチ・プロセッサや、DRAM・ロジック混載などによるメモリと CPU の 1 チップ化、もしくは I/O と CPU の 1 チップ化を行なったシステムオンチップと呼ばれるアプローチなどがその代表例である。これらの技術により既存の並列計算機やコンピュータ・システムの機能に相当するものが少数のチップにより構成され、例えば並列計算機における通信オーバーヘッドや、PC・ワークステーションなどにおけるメモリ・ウォール問題などが改善されることで高性能化が期待されている。しかしながら、これらは従来の CPU・メモリ・周辺機器を 1 チップ化することでシステムとしての性能を上げることがメインで、必ずしも CPU のもつ性能自体を改善するというのではない。CPU のもつ性能自体を上げる努力をしなければ、CPU 性能がボトルネックとなりシステム全体の性能が上がらなくなるのは明らかである。

マイクロプロセッサの性能向上には、命令レベルの並列性の利用が欠かせない。プログラムの処理の本質はデータフローであり、データーフローをもとに実行を行なえれば、自然と命令レベルの並列性が抽出される。しかし、現在のアーキテクチャの多くはコントロールフローを中心にして実行を行なっているため、命令レベルの並列性の抽出が難しくなっている。本研究では制御依存関係を解消した命令を大量に実行部に送り高速処理を行なう方式の開発を目的としている。実行部としては、多数の ALU をネットワーク状に結合した ALU-NET と呼ぶ機構を設ける。このような大規模なデータパス部分をもったアーキテクチャを、大規模データパス (Very Large Data Path)・アーキテクチャと呼び、本研究では、今後おおよそ 10 年後のデバイス技術を仮定した上で、CPU 自体の性能向上を目指した新しいプロセッサ・アーキテクチャとして大規模データパス・アーキテクチャの開発を行なう。

1.2 研究の目的

本研究では、大規模データパス・アーキテクチャの概要についてまとめ、その中で特に制御依存関係による制約を可能な限り取り除く方式の検討を行なった。この機構をコントロールフロー先行展開と呼ぶ。コントロールフロー先行展開の特徴

は大規模な投機処理にある。マイクロプロセッサにおいて、コントロールフローの複数パスへの投機フェッチを行なうことで、分岐命令による制御依存関係を効率的に解消することを目指している。

複数パスへの投機フェッチにおいては、どのような命令の展開方式をとるかが重要な問題である。単一パスへの命令展開では、分岐予測の影響を受けやすく、分岐予測に失敗したときには大きなペナルティーが課せられる。一方すべてのパスを展開したのでは、投機規模を大きくするに従って不要な命令が爆発的に増加してしまう。そのため効率的な命令展開を行なうには、選択的な複数パスへの投機フェッチが必要である。

投機フェッチの効率化には分岐予測性能の向上も欠かせない。しかしながら、これまでにさまざまな分岐予測機構の検討が行なわれてきた結果、最近ではさらなる性能向上はあまり望めなくなっている。ただし、これまでの分岐予測機構は単一パスへの投機フェッチを想定して設計されてきたので、コントロールフロー先行展開で行なうような複数パスへの投機フェッチに対応した分岐予測機構の検討が必要である。本研究では、投機フェッチに伴う履歴情報の更新遅れ、複数パスへの分岐予測の2点について調査および評価をおこなう。

複数パスへの選択的投機フェッチを行なうのがコントロールフロー先行展開であるが、この際にコントロールフローの中からどのパスを選択するかが重要な問題である。理想的には、各パスへの実行確率を計算し最も可能性の高いパスから投機フェッチをするのがよいと思われるが、現実的に実装することは困難である。そこで、コントロールフロー先行展開では実装を考慮し、軽い処理で近似的に効率的な選択的投機フェッチが可能な実行確率管理機構を開発し評価する。

最後に本研究で開発するコントロールフロー先行展開を用いて、理想的な実行部を持ったプロセッサにおける限界性能について考察し、あわせて大規模データパス・アーキテクチャの今後について考察する。

1.3 本論文の構成

本論文の構成は以下の通りである。

第2章では、現在のマイクロプロセッサに関して、デバイス技術、アーキテクチャの両面について概観する。

第3章および第4章では、現在、そして今後のマイクロプロセッサの高性能化に

関する研究について説明する。

第5章では、本研究で提案する大規模データパス・アーキテクチャについて、その概要と各機能モジュールについて説明する。

第6章では、大規模投機処理の際に分岐予測機構が受ける影響について述べる。特にフェッチポイントと実行完了ポイントの差が広がることによる分岐履歴情報の更新遅れの影響をシミュレーションにより調査する。

第7章では、複数パスへの投機フェッチに必要な分岐予測機構について考察し、新しい分岐予測機構としてMulti BTACを提案・評価する。

第8章では、複数パスへの大規模投機処理における命令供給システムについて検討し、大規模データパス・アーキテクチャの命令供給システムであるコントロールフロー先行展開について詳しく述べる。

第9章では、コントロールフロー先行展開について詳細に性能評価を行ない、様々なパラメータについてその最適値を調査する。

第10章では、大規模データパス・アーキテクチャの命令供給システムであるコントロールフロー先行展開について他の方式との比較を行なう。また大規模データパス・アーキテクチャについてまとめ、今後開発していくべき技術課題について述べる。

第11章で結論を述べる。

第 2 章

マイクロプロセッサ

2.1 デバイス技術

マイクロプロセッサの性能向上は、デバイス技術の進歩とアーキテクチャの改良という 2 つの柱で達成されてきた。本章では、まずこのうちのデバイス技術の進歩と今後の予測を、米国半導体工業会 (Semiconductor Industry Association: 以下 SIA) が 3〜4 年おきに発表するロードマップ [fSMP][Sem97] をもとにまとめる。表 2.1は、SIA の予測に基づくハイエンド・マイクロプロセッサのデバイス技術ロードマップである。主な指標に関してまとめる。

表 2.1: マイクロプロセッサのデバイス技術予測

year	1997	2001	2006	2012
rule(nm)	250	150	100	50
wafer(mm^2)	300	385	520	750
layer	6	7	7-8	9
Tr./chip	11M	40M	200M	1.40G
pins	1100	1800	3000	5500
chip I/Os	1450	2400	4000	7300
chip clock(Hz)	750M	1.5G	3.5G	10G
I/O clock(Hz)	750M	1.4G	2.0G	3.0G
voltage(V)	1.8-2.5	1.2-1.5	0.9-1.2	0.5-0.6
power(W)	70	110	160	175

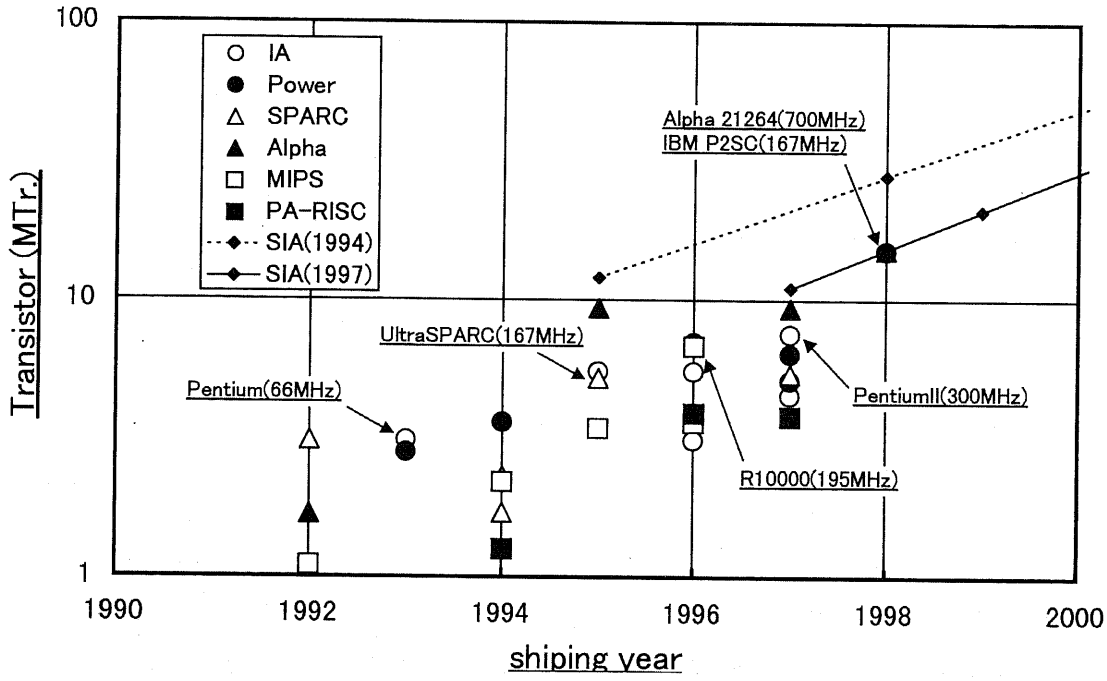


図 2.1: 1チップ当たりのトランジスタ数

- rule(nm) はチップ内のロジック部分の最小加工寸法であり、一般的にこの値が小さいほど高い動作周波数を実現でき、同じ寸法のチップではより多くのロジックを詰め込むことができる。

- wafer(mm^2) はチップの寸法であり、この値が大きいほど大量のロジックを 1 チップに詰め込むことができるが、一般的に寸法が大きくなると歩留まりが悪化しコストがかかる。

- Tr./chip は、1 チップ内に実装できるトランジスタ数であり、この値が多いほど大量のロジックを実装できる。

- chip clock(Hz) はチップ内部の動作周波数で、同じロジックの場合には、一般的に動作周波数が高いほど高性能である。

- voltage(V) はチップ内部の電源電圧で、低電圧になると動作速度が遅くなる場合もあるが、消費電力 (power(W)) を押えるために低電圧化をめざして研究がなされている。消費電力が大きいとチップが高熱になり熱破壊が起こる可能性が高まるためである。

この SIA 予測の中で特に注目すべきは、Tr./chip の項目である。図 2.1は 1 チップ当たりのトランジスタ数を、最近のプロセッサと SIA 予測についてプロットしたものである。縦軸は対数表現である。また SIA 予測については、1994 年版を点線、1997 年版を実線で結んだ。SIA の予測は 2000 年以降の部分もあるが、2000 年以降も直線であるのでグラフには入れていない。このグラフから今後 1 チップで利用できるトランジスタ数が爆発的に増えることが読みとれる。今後はこのような大規模なデバイスをどのように使うのかが問題である。

一方 DRAM とロジックそれぞれを見てみると、そこにはメモリウォールと呼ばれる問題が存在する。図 2.2[Masa95] のように DRAM とロジックのプロセスでは DRAM の速度向上率がロジックに比べて低いため、今後一層その差が広がることも考慮しなければならない。最近のプロセッサにおいても図 2.3[Yu.96] のように大量のキャッシュメモリを載せてメモリウォール問題の緩和を行なっている。

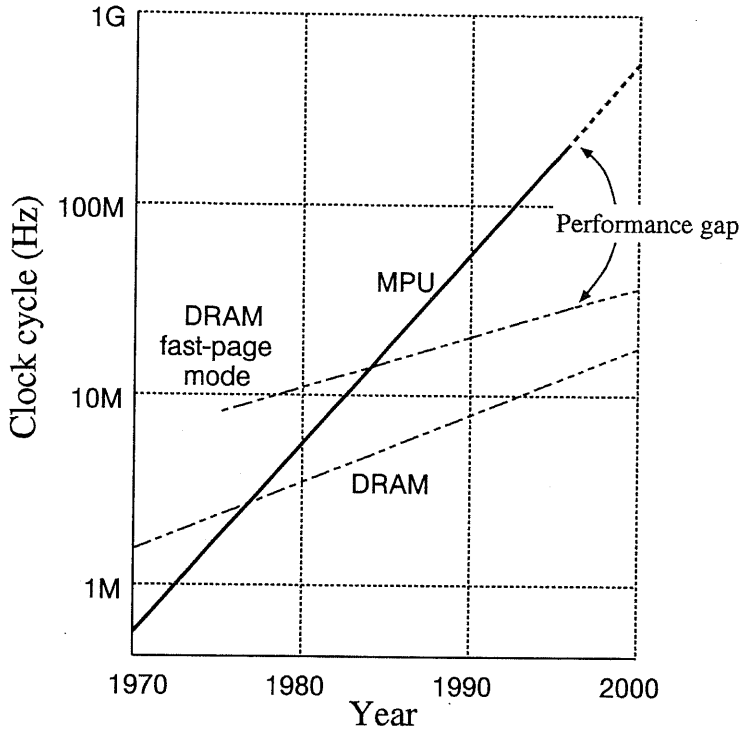


図 2.2: MPU およびメモリの動作速度の変化

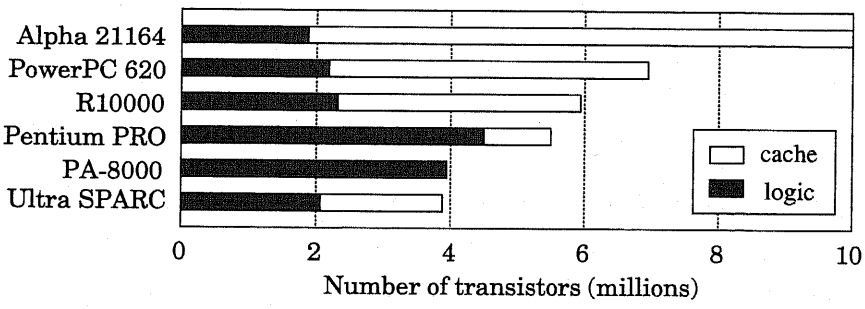


図 2.3: 全トランジスタに対するロジックの占める割合

2.2 アーキテクチャ

最近のマイクロプロセッサは大きく分けて3つの種類がある。1つはワークステーションやPCなどでの使用を想定し高性能化をメインに考えた汎用マイクロプロセッサ、2つ目はマルチメディア処理を想定しマルチメディア処理のために特別なSIMD命令などを用意したメディア・プロセッサ、そして3つ目はハンドヘルドPCや情報家電を想定し省電力化をメインに考えた省電力プロセッサである。本節では、これらの3つのアーキテクチャについて例を挙げて概観する。なお、汎用マイクロプロセッサのアーキテクチャについては、第3章でさらに詳しく述べる。

2.2.1 汎用マイクロプロセッサ

汎用のマイクロプロセッサとしては、Intel系のIA(x86)系、MIPS系、Alpha系、SPARC系、Power系, PA-RISC系などがあるが、最近ではIA系が市場の8割程度を占めるようになっている。しかし、性能的には必ずしもIA系が優れているわけではなく、過去のバイナリとの互換性の問題によってIA系が多く使われていると言っても過言ではない。

図2,4はSPECベンチマーク[tS]を使った、各プロセッサの整数／浮動小数点演算の性能を示したものである。値が大きいほど高性能である。各図の棒グラフはSPEC値つまり絶対性能を示し、折線グラフはSPEC値を周波数で割った値つまり周波数当たりの性能を示している。整数演算性能は拮抗しているが、浮動小数点演算性能についてはかなり差があり、特にIA系はまだ性能が他に比べて低い状態である。ただし、SPECベンチマーク値がプロセッサの性能を正しく評価しているとは限らない。特に大規模科学技術計算やトランザクション処理のような用途に対しては、SPECベンチマークでは計れない部分の性能差が大きく影響する場合があり、例えば、トランザクション処理のベンチマークとしては、TPC(Transaction Processing Council)[tT]などがよく使われている。TPCベンチマークは、よりシステム全体としての性能を評価するので、プロセッサ単体の性能の評価としては利用しにくい。

ここではまず、DECのAlpha 21264を例に高性能汎用マイクロプロセッサ[DEC98]の特徴を見てみる。

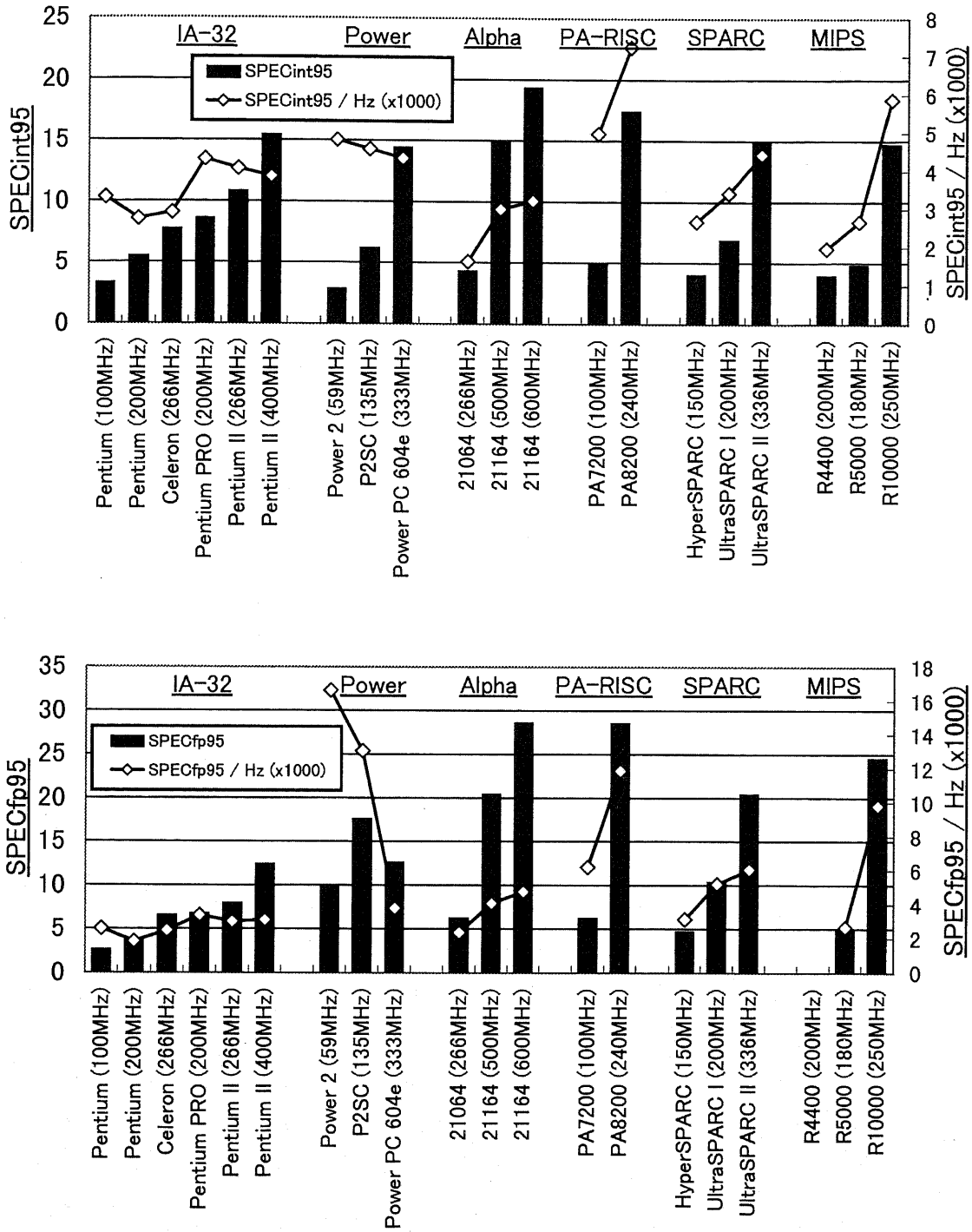


図 2.4: 整数／浮動小数点演算性能 (SPECint/fp95)

Alpha 21264

Alpha シリーズは、常に周波数を上げて性能を稼ぐ方法で開発されてきた。21264 も 1998 年夏は 600MHz 動作であるが、2000 年までには 1GHz 以上にクロックアップする予定になっている。これまで動作周波数は 2 年で 2 倍以上の伸びをしてきており、今後もこのペースが続くとしたら実現不可能な値ではない。しかし、高い動作周波数を実現するにはかなりの困難がつきまとうのも事実である。設計ルールの微細化を行ない、さらに 1 サイクルで行なえる処理を減らさずにサイクル時間を縮めるための回路的な工夫が必要となる。さらに熱破壊を防ぐため周波数に比例して増大する消費電力を押える必要がある。

表 2.2: Alpha21264 の仕様

動作周波数	600MHz～1GHz
消費電力	22W
演算性能	40～100 SPECint95 60～150 SPECfp95
電源電圧	+2.0V(内部), +3.3V?(入出力)
トランジスタ数	15.2M
製造技術	0.35～0.18μmCMOS 6-layer Metal

2.2.2 メディア・プロセッサ

カラーテレビの普及率は 99.2 % (1996 年度) であるが、これから 10 年程度の間にアナログ放送からディジタル放送に移行することが決まり、これらのアナログ放送用テレビが使えなくなることになった。1998 年 5 月現在の日本の総世帯数は 4411 万世帯で、これによると 4375 万台以上のカラーテレビが買い替えになるということになる。デジタル放送用テレビの中にはデコーダ用のメディア・プロセッサが使われることになり、1 チップ当たり 1000 円だとしても 500 億円超程度の市場[1]となるため、現在盛んに研究がなされている。

また、その他の情報家電機器でも MPEG のデコード用などにメディア・プロセッサの活躍場所が増えてきている。このようなメディア・プロセッサの特徴をまとめると次の通りである。

[1]各世帯に 1 台とは限らないため

- 汎用マイクロプロセッサほどの自由度はいらないが、専用 LSI よりも自由度が必要である。最近では情報家電機器にも OS が載ることがある時代である。
- 汎用マイクロプロセッサよりも桁違いに安いことが求められる。
- 製品サイクルが短いので変更が容易な構造にすることが多い。CPU コアは共通にして ASIC[2]展開を考慮に入れた設計が多い。

ここでは、富士通の Embedded Superscalar Microprocessor for Multimedia Applications (ESMMA)[須賀 98] を例にメディア・プロセッサの特徴を見てみる。

富士通 ESMMA

表 2.3は ESMMA の仕様である。この仕様は、MPEG2 のデコードが IPC[3]= 1.4 の状態で可能となるように作られている。

表 2.3: ESMMA の仕様

動作周波数	180MHz
消費電力	1.2W
演算性能	2.16GOPS[4]/720MFLOPS[5]
電源電圧	+1.8V(内部), +3.3(入出力)
トランジスタ数	4.0M
製造技術	0.21μmCMOS 4-layer Metal

表 2.4は ESMMA の全 170 命令の内訳である。ESMMA では、16bit の整数積和および累積演算を 4 個同時に実行できる実行ユニットをもっている。

SIMD[6]命令とは 1 つの命令で複数のデータに対して演算を行なうタイプの命令で、例えば積和演算 (ab + cd) のような演算を 1 命令で実現するようなタイプの命令である。この演算はフィルタ処理でよく使われる。表 2.4の SIMD type の 80 命令にはこのようなマルチメディア処理で良く使われる命令が用意されている。例えば、PI(Packed Integer) 命令は、図 2.5のように、2 つの 32bit のレジスタ上にある

[2]Application-Specific Integrated Circuit
[3]Instruction Per Cycle: 一サイクル当たりの命令数
[4]Giga Operation Per Second: 一秒当たりに実行できる演算数
[5]Mega Floating Operation Per Second: 一秒当たりに実行できる浮動少数点演算数
[6]Single Instruction Multi Data

表 2.4: ESMMA の命令セット

Non-SIMD type		SIMD type	
Arithmetic	19	Arithmetic	34
Logical	4	Logical	4
Compare	20	Compare	20
Shift	4	Shift	4
Move	6	Move	6
Convert	13	Convert	13
PI-sum/merge/sub	4	PI-sum/merge/sub	4
Bit Operation	6	Bit Operation	6
Load/Store	14	Load/Store	14
total	90	total	80

16bit データを 1 つにまとめ、それぞれに対して独立に演算を行なうような命令である。マルチメディア処理においては、このように、精度を落しても高速演算ができればよい場合が多く存在する。その他にもオーバーフローしない加算命令などがよく実装されている。

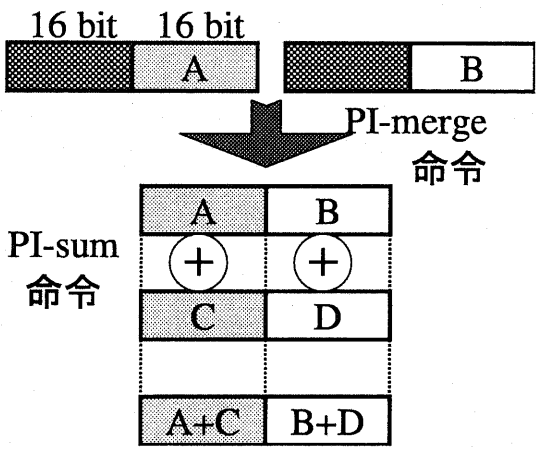


図 2.5: PI 命令の例

主なメディア処理とそれに適用される命令の種類を表 2.5 にまとめておく。

表 2.5: メディア処理に適用される命令種類

	SIMD 8bit	SIMD 16bit	Byte or 16bit align	Shift / Round	Vector Elem. select	飽和演算
Motion Search	○	−	○	−	○	○
Motion Compensation	○	−	○	○	−	○
Audio Filter	−	○	○	○	○	○

2.2.3 省電力プロセッサ

近年の急速なデジタル携帯情報機器やデジタル携帯通信機器の普及にともなって、省電力プロセッサの重要性が高まっている。図 2.6は移動通信機器の加入数の推移を示したものである [移動]。これらに組み込まれているプロセッサだけでも数十億円以上の市場である。この他にも PDA や Mobile PC と呼ばれるノートパソコンなどにも省電力プロセッサが使われており、今後もこれらの用途向けの省電力プロセッサ市場は大きくなっていくと思われる。

現在のところ、消費電力と用途の関係をまとめると表2.6 のようになる。今後の携帯機器用プロセッサの消費電力目標は 100mW を切る線にまでなっている。

表 2.6: 用途と消費電力

消費電力	< 100mW	< 2W	10W <
目的	バッテリーの長寿命化	プラスティック・パッケージの熱限界	セラミック・パッケージの熱限界
用途	携帯機器	民生機器	ハイエンド MPU

ここでは、省電力プロセッサの一例として、ISSCC `98 で発表された、DEC の StrongARM プロセッサ [etc98] を紹介し、省電力化のための工夫について述べる。

DEC SA − 1100

StrongARM(以下SA) シリーズの最新バージョンである。SA シリーズは組み込み用途のマイクロプロセッサとして 1996 年から開発されたアーキテクチャで、高性能と

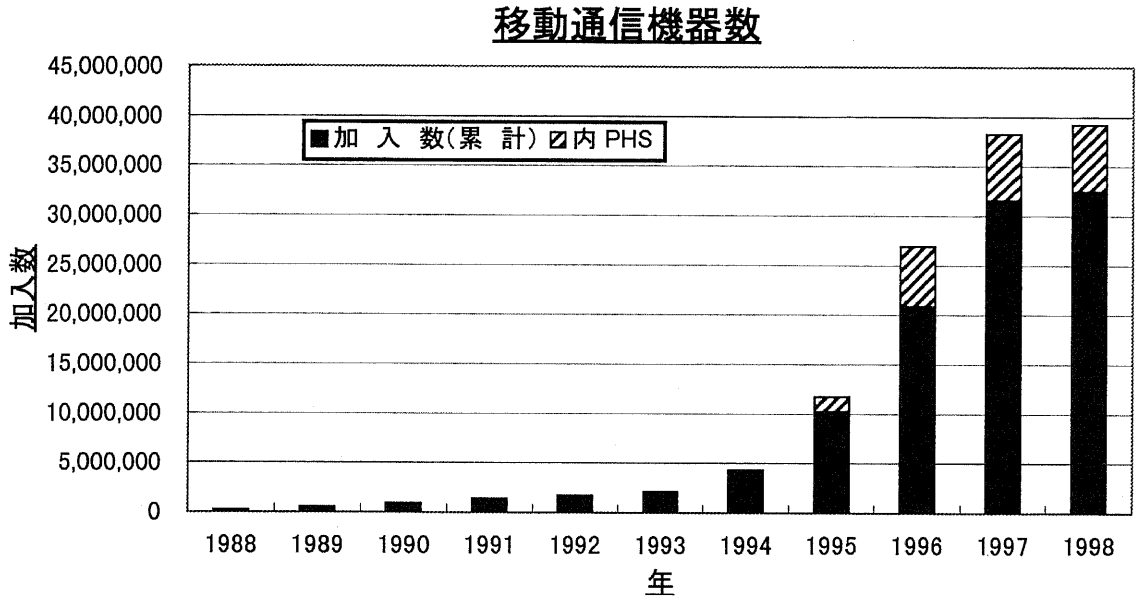


図 2.6: 移動通信機器の加入数

低消費電力という従来は相反する要求を実現することを目標に作られた。SA-1100 がターゲットとしている組み込み機器は、携帯型情報機器や携帯電話、Mobile PC などである。SA-1100 の仕様は表 2.7の通りである。

表 2.7: SA-1100 の仕様

動作周波数	133MHz, 200MHz
消費電力	200mW～250mW
演算性能	230MIPS[7]
電源電圧	+1.5V(内部), +2.7～+3.6V(入出力)
トランジスタ数	2.5M
製造技術	0.35μmCMOS 3-layer Metal

演算性能当たりの消費電力で見ると、SA-1100 は約 1mW/MIPS であり、例えば Hitachi SH-4 の約 5mW/MIPS と比べると 1/5 の省電力化がはかられている。このような低消費電力を実現するための工夫は次の通りである。いずれも最近の省

[7]Mega Instruction Per Second: 一秒当たりに実行できる命令数

電力化において使われるテクニックの代表的なものである。

内部電源電圧の低減 以前のSA シリーズでは内部電源電圧は+2.0V であったので、内部電源電圧を+1.5V にすることで、消費電力は $(+1.5)^2/(+2.0)^2$倍 (≈56 %) になる[8]。しかし、内部電源電圧を低下させるためには、スレッショルド電圧の低いプロセスを使う必要がある。そのような回路技術としては、MT-CMOS などがある (ここでは回路技術の詳細までは触れない)。

3 つの動作モード 通常の処理が行なわれる NORMAL MODE の他に、外部からの入力がない状態のモードとして IDEL MODE、外部入力が長時間ない場合の SLEEP MODE が用意されている。IDEL MODE では CPU コアとキャッシュに対するクロック供給が停止され 40mW まで消費電力が押えられる。SLEEP MODE では内部メモリのデータを保護するだけであり、さらに消費電力が押えられる。

クロック供給の部分的停止 クロック信号の分配は非常に電力を消費する。また動作周波数に消費電力は比例するので、高い周波数で動作する最近のプロセッサではさらにこの傾向が強くなっている。そのため、必要ない部分へのクロック供給を一時的に停止させて省電力をはかることが最近よく行なわれるようになった。SA-1100 では内部回路を 150 もの区画に分けてクロック供給を ON/OFF できるようになっている。

この他にも、負荷容量の低減を目指してゲート長微細化やダイナミック回路の利用などが考えられる。省電力化は主に製造プロセスの改良によってなされるが、クロック供給の部分的停止などは、設計側での工夫である。

[8]消費電力は電圧の 2 乗に比例する

第 3 章

高性能化をめざすアーキテクチャ

本章では、最近の高性能化を目指したアーキテクチャからスーパースカラについて、現在研究が進められている次世代マイクロプロセッサ・アーキテクチャから DRAM・ロジック混載プロセッサ、オンチップマルチプロセッサ、大規模 CPU について述べる。第 2.1章で述べたように、次世代のマイクロプロセッサを考える際には、1 チップに集積可能な大規模なデバイスをどのように利用するかが最も重要な視点である。本章で述べる次世代マイクロプロセッサ・アーキテクチャのそれぞれの構成を摸式的に示したのが図 3.1である。DRAM・ロジック混載プロセッサ

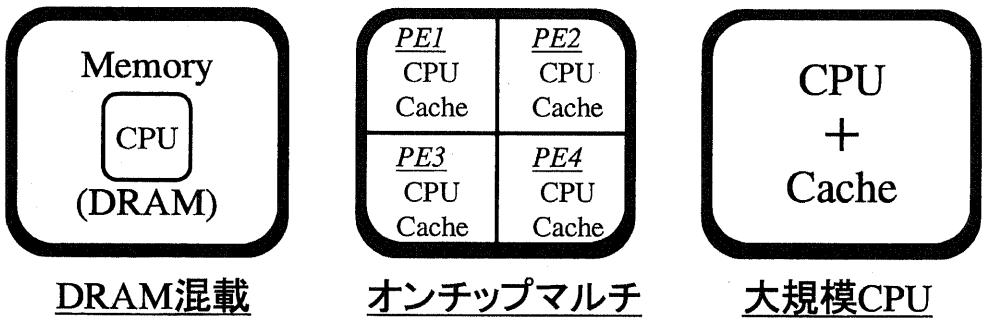


図 3.1: 次世代マイクロプロセッサ

は、DRAM セルの上、もしくは横に小規模な CPU を搭載し、メモリ・CPU 間のデータ転送能力を上げることでメモリウォール問題の影響を押え高性能化を目指したアーキテクチャである。オンチップマルチプロセッサは、マルチプロセッサ・システムの CPU 部分を 1 チップ化したものである。1 チップ化によって CPU 間のネットワーク性能が向上し、高性能化が期待できる。大規模 CPU は、基本的に

CPU 単体の性能向上を目指したアプローチである。情報処理の基本は演算であり、演算部分の高性能化を追求する方式である。

3.1 スーパースカラ

今日の命令レベル並列性を利用するマイクロプロセッサ・アーキテクチャの始まりはスーパースカラ・アーキテクチャにある。スーパースカラはスカラプロセッサの各パイプライン・ステージを1サイクルあたりに複数命令を処理できるとしたものである。このような命令レベル並列性の利用を行なうことにともなって、新たに性能を制約する条件が出てきた。

資源競合 スカラプロセッサでは、あるサイクルでフェッチ・デコード・実行・完了される命令は1命令だけであるので、各資源、例えば機能ユニットやメモリバスなどの利用において競合が起こることはなかったが、スーパースカラプロセッサでは、同じ資源、例えば同じ種類の命令を同時に実行使用とした場合などには機能ユニットが競合を起こす。これを回避するには、資源の多重化をするか、もしくは同時実行をやめる必要がある。

命令の実行順序及び例外回復 従来のプログラムは先頭から逐次に実行されることを仮定しているため、複数の命令を同時に実行するには、プロセッサが先の命令を見て実行可能なものを探す必要がある。すると、本来プログラムに書かれていた命令の順序とは異なる順序で命令が実行されることが起こる。どの程度プロセッサが先見能力を持つかは得られる命令レベル並列度に影響を与える。一方、先見によって実行された命令よりも前の命令で例外が起こった場合の例外回復機構も考える必要があり、正確な割り込みを実現するためにリオーダーバッファやヒューチャファイルなどの機構が開発された。

制御依存関係 プログラムには多数の分岐命令が含まれているため、プロセッサの先見能力を上げて命令レベル並列度を上げようとすると、分岐命令をまたいで実行可能な命令を探す必要が出てくる。これを可能にする機構として、分岐予測機構が考えられた。

3.2 DRAM・ロジック混載プロセッサ

図 2.2に示すように、近年プロセッサとメモリの速度差が広がっており、メモリからのデータ供給がプロセッサの速度向上に追い付けない状態になっている。このメモリ・レイテンシを隠蔽するために、最近のプロセッサではキャッシュを内蔵するのが当たり前になり、さらに大容量の 2 次キャッシュをもうけるものも多い。第 2.1節で見たように、今後もプロセッサはある程度の動作周波数の向上と、アーキテクチャ的な改良による IPC の向上により、より一層メモリ・プロセッサ間の速度差による問題 (メモリ・ウォール問題) が深刻になると考えられる [Masa95]。そこで、大規模 LSI の中にメモリを取り込んでしまうというアプローチが研究されている。ここでは、PPRAM^R[村上 94][村上 95] の研究の中で行なわれた、DRAM 混載モデルに関する性能評価を紹介する。 図 3.2は DRAM 混載のプロセッサ・モ

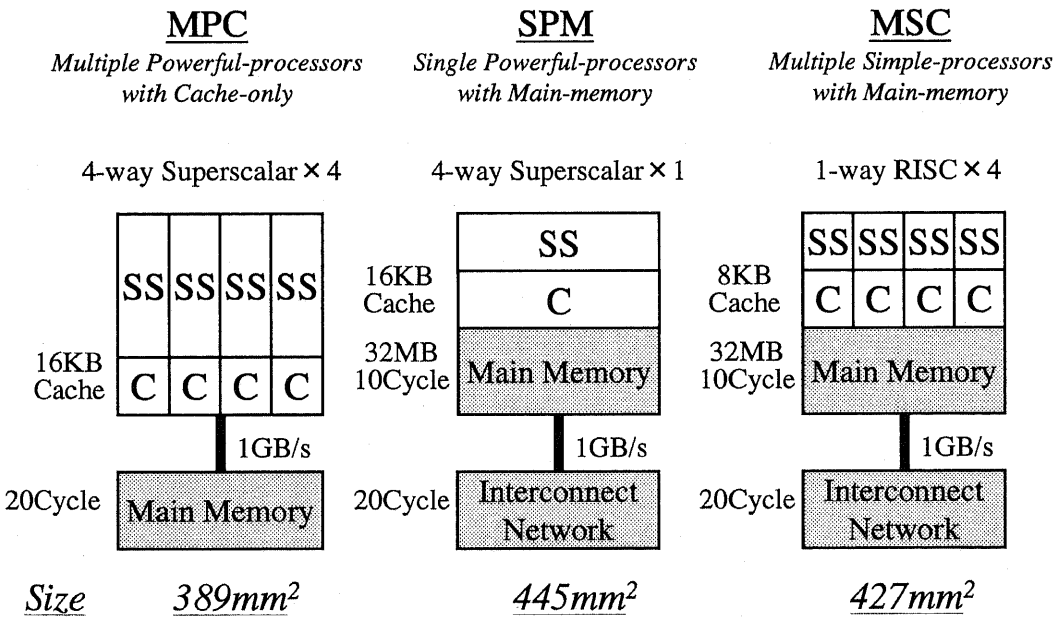


図 3.2: DRAM 混載プロセッサ・モデル

デルと比較用のモデルの定義である。ここでは、各プロセッサ・モデルのダイ・サイズがほぼ同じ位になるようにモデルを定めている。

MPC オンチップ・マルチプロセッサのモデルで、最大同時実行命令数が 4 のスーパースカラ・プロセッサと 16KB のキャッシュの組を 4 セット持ち、メイン

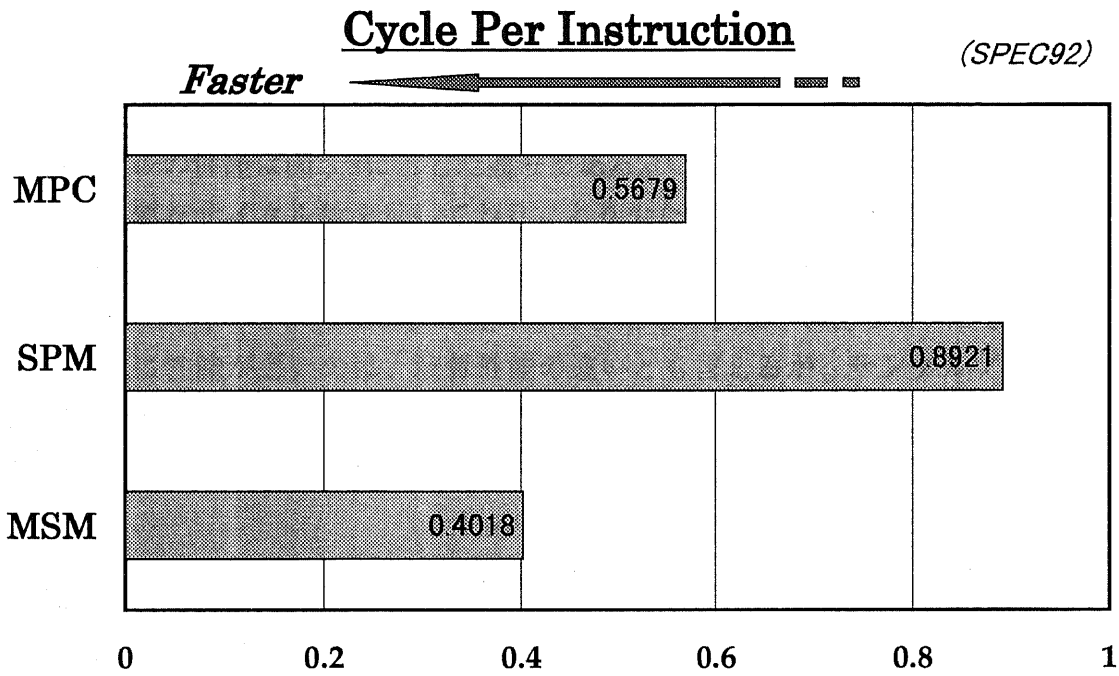

Cycle Per Instruction
(SPEC92)
Faster
MPC
SPM
MSM
0.5679
0.8921
0.4018
0
0.2
0.4
0.6
0.8
1

図 3.3: 各モデルの性能

メモリは外部に 1GB/s のバスで結合されている。メインメモリへのアクセスには 20 サイクル必要となる。

SPM このモデルは DRAM 混載の基本モデルで、MPC と同様のスーパースカラ・プロセッサと 16KB のキャッシュのセットを 1 つだけもち、残りを DRAM としたものである。DRAM は 32MB で、オンチップになったことにより 10 サイクルでアクセスできる。

MSM 小さな RISC プロセッサと 8KB のキャッシュの組を 4 セット持ち、残りを 32MB の DRAM(10 サイクル・アクセス) としたモデルで、オンチップ・マルチプロセッサに DRAM を混載した形になっている。

各モデルの性能は図 3.3に示した通りである。このグラフで横軸は 1 命令実行するのに必要なサイクル数を示しており、より少ないサイクル数 (グラフでは左) で 1 命令を実行できる方が高性能である。これによれば、MSM モデルが最も高い性能を示していることが分かる。メインメモリをオンチップに持っている SPM モデルが意外に性能が悪いのは、演算性能の違いとも言えるが、サンプルプログラムが SPEC92 であり、ほとんどのデータがキャッシュにヒットしてしまうため、メモリ・ウォール問題の影響があまり大きくないという可能性も考えられる。

3.3 オンチップマルチプロセッサ

マルチプロセッサの 1 チップ化を行なうのが、オンチップ・マルチプロセッサのアプローチである。テクノロジ的にどの程度のものが実現可能かどうかについては、文献 [坂井 97] などを始めとして様々な検討が行なわれており、近い将来にオンチップ・マルチプロセッサの実現が見込まれている。

具体的に、文献 [坂井 97] では 10 年後の 2007 年におけるプロセッサ VLSI(VLSI2007) として、

- 内部トランジスタ数: 1.15G
- クロック周波数: 1.5GHz
- パッケージ信号線数: 2500

という仮定をおいた上で、表3.1に示した仕様のマイクロプロセッサHARP1E[1]を要素としたオンチップ・マルチプロセッサに関する検討が行なわれており、これによれば、

(HARP1E + 4MB 2nd Cache) × 5

の構成が可能であるとされている。

表 3.1: HARP1E の仕様

動作周波数	150〜200MHz
消費電力	10W
演算性能	0.3GFLOPS
電源電圧	+2.5V
トランジスタ数	4.5M
製造技術	0.3μmCMOS

オンチップ・マルチプロセッサの構成は、大きく分けて疎結合型と密結合型の2種類[2]がある。それぞれについてまとめる。

疎結合型 おおよそ現在のSMP[3]を1チップにまとめたものであり、現在のSMPとの互換性が高いので、並列科学技術計算などの分野に適用性が高い。しかし、PE[4]間の結合が密結合型ほど強くなく、シングルJOBにおける性能向上は、現在のSMPにおいてシングルJOBを高速化する場合と同じく、並列化コンパイラに頼る部分が大きく制約となる。どちらかというと、疎結合型では、2次キャッシュのオンチップ共有化などによって、大容量メモリをチップ内に取り込むことで性能向上を目指す意味合いが強いと思われる。その他、コスト的に1チップに集積した方がよいかもしれない、SMPに対してPE間・メモリ間の結合強化が行なえるため、スケジュール支援がしやすくなるのではないかという利点が考えられている。

[1]超並列機 Hitachi SR2201 の要素プロセッサ

[2]文献 [西 98] ではさらに時間多重型・非対称型のあわせて4種類に分けられているが、ここでは疎結合型と密結合型のみを取り上げる

[3]Symmetric Multi Processor

[4]Processing Element

密結合型 疎結合型がSMPの流れを汲んでいるのに対して、密結合型は現在のスーパースカラ方式のマイクロプロセッサの次を睨んで検討されているものである。従来のSMPではプログラムに内在するような粒度の小さい並列性を利用しようとすると、並列化のためのオーバーヘッド[5]の影響が大きく、必ずしも高い性能が得られなかったが、1チップに集積することで、PE間の結合が強化でき、このオーバーヘッドが小さくなることが期待されている。

ここでは、より次世代のマイクロプロセッサとしての意味あいが強い密結合型の例として、NEC MUSCAT[鳥居 97] を取り上げる。

NEC MUSCAT

MUSCAT(MUlti-Stream Control ArchiTecture) では、オンチップ・マルチプロセッサによって従来のSMPに比べてPE間の距離が短くなり通信コストが低くなるという利点を使って、制御並列処理 (Control Parallel) を行なうのが特徴である。図3.4は制御並列処理のイメージを示したものである。ベーシックブロックA～Iが図のような制御フローで結ばれているとき、ベーシックブロックAの実行が確定すると、ベーシックブロックEの実行も確定する。また同様の関係がEとHの間にもある。よって、A～D, E～G, H～Iをそれぞれスレッド1,2,3とすると、スレッド間でデータ依存関係がなければ、次々と実行することができる。MUSCATではこれらの制御並列処理を他のPEへのスレッド作成命令を使って実現する。スレッド間にデータ依存関係がある場合には、データ依存制御命令を使って、データの受渡しを行なう。

MUSCATでは、コンパイラが静的にプログラムを解析して、スレッド制御命令やデータ依存制御命令を明示的に入れることによって制御並列処理を実現する[鳥居 96]。さらに、コンパイラは、次の2つのスレッドスペキュレーションをおこなうための命令を使うことができる。

制御スペキュレーション 実行する可能性の高いスレッドを実行確定前に投機的に実行する。

データスペキュレーション スレッド間のデータ依存がないと仮定して、投機的にロード／ストアを実行する。

[5]例えばPE間のデータ転送や同期によるもの

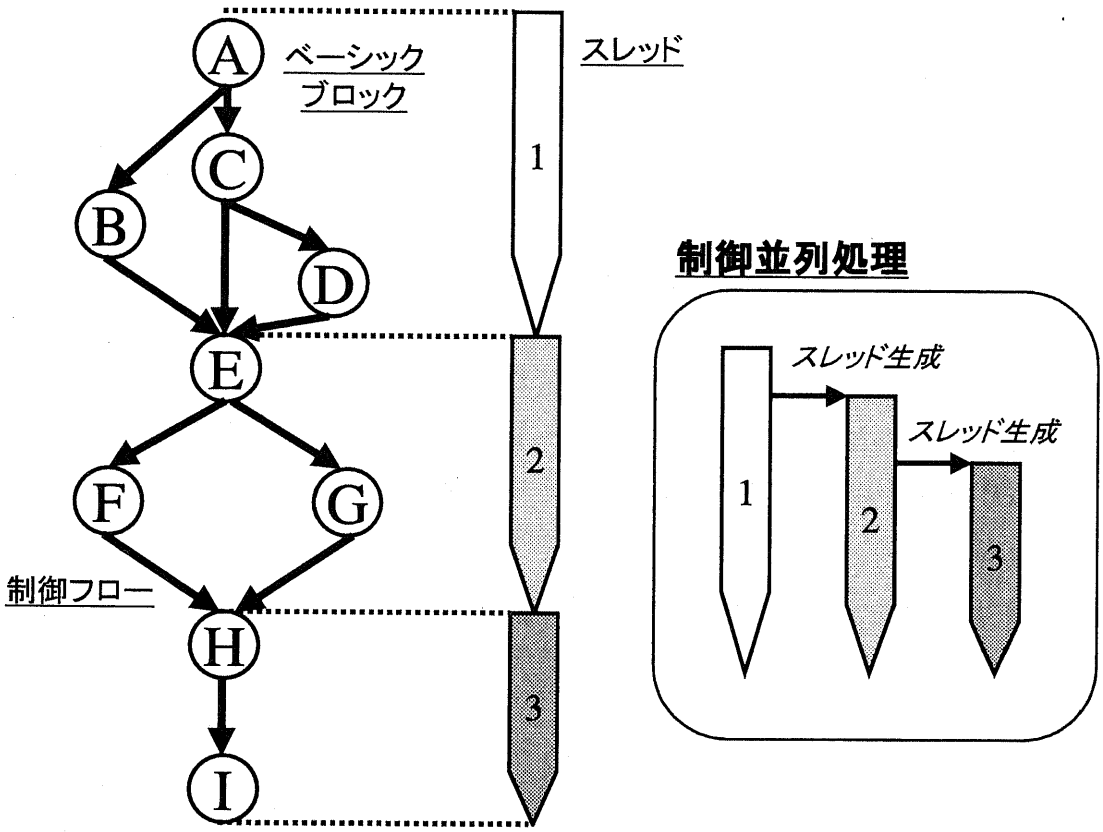

A
ベーシック
ブロック
B
C
D
E
F
G
H
I
制御フロー
スレッド
1
2
3
制御並列処理
スレッド生成
スレッド生成
1
2
3

図 3.4: 制御並列処理

性能的には4PEを1チップにおさめた場合には、MPEG2のデコードで1PEの場合に比べて約3倍の性能向上が得られることが分かっている。

密結合型のオンチップ・マルチプロセッサの実用化には、コンパイラがどこまで並列性を抽出し、その制御を最適に行なうかにかかっているといえる。

3.4 大規模 CPU

マイクロプロセッサの高性能化において究極的に必要なのは、シングルJOBの高速処理である。DRAM混載のアプローチはメモリ・ウォール問題の解消を目指して高性能化をはかっているが、プロセッサの演算性能自体をあげるものではない。オンチップ・マルチプロセッサのアプローチは、密結合型ではシングルJOBの高性能化を目指しているが、疎結合型はSMPの延長技術であり、シングルJOBの高性能化が必ずしも実現できるわけではない。

大規模CPUのアプローチは、CPU自体の性能向上を目指すもので、より下位のレベルでの高性能化を実現するものである。よって、DRAM混載やオンチップ・マルチプロセッサの要素プロセッサとしての利用も将来的には可能になることが考えられる。本研究で提案するVLDPアーキテクチャはこのアプローチである。

次章以降では、大規模CPUを開発する上で、どのような技術課題があり、VLDPではどのようなアプローチでそれを解決していくかを示す。

第 4 章

高性能化への問題と最近の研究

本章では既存プロセッサの代表としてスーパスカラプロセッサを考える。スーパースカラプロセッサは命令レベル並列性を利用して性能を得る構造である。しかし、様々な問題によりさらなる性能向上に限界が見え始めている。その原因を制御依存関係による制約とデータ依存関係による制約の 2 点に分けて述べ、あわせて最近の研究から、これらの制約を削減もしくは隠蔽する技術について述べる。

4.1 制御依存関係による制約

プログラム中の分岐命令による制御依存によって命令レベルの並列性抽出に制限が加わる [SrP92]。これを緩和するためこれまでに様々な分岐予測機構 [YN92][DF95] や条件付き実行 [ea92] などが研究されてきた。しかし、それぞれの性能には限界が見えてきており、より積極的な投機実行機構を設けたり、より大きなレベルでの並列性の利用を考えなければならない。また積極的な投機実行機構を用いた場合には、従来よりも高い命令転送能力が必要となる。

4.1.1 分岐予測機構

分岐予測とは実際の分岐先が判明する前に、分岐命令の分岐先を予測して分岐遅延を削除することで、命令の流れに沿って命令のフェッチを継続させるものである。分岐予測には大きく分けて次の 2 つのアプローチがある。

分岐予測

- 静的:コンパイル時

- 動的:ハードウェアを用いて動的な分岐動作を獲得しようとする

いずれにおいても予測失敗によるペナルティとして、間違ってフェッチされた命令の取り消しと正しいパスへの復帰が必要となる。

ここでは、主な 6 つの分岐予測アプローチについて説明する。この内 5 つは製品として現存するものである。平均的な分岐予測成功確率は、単純な静的な方法で 60 %程度、洗練された動的な方法で最大 97 %程度に及ぶ。

1.Branch Always:常に分岐成立

全ての分岐が成立と予測, 常に分岐先のターゲットから命令をフェッチする必要

2.PA-RISC(HP):Branch Backward(後方への分岐を仮定する静的な分岐予測)

前方への分岐=不成立, 後方への分岐=成立
→前方への分岐が不成立となる確率の高いアプリケーションでは分岐予測成功率向上

3.Alpha:2 つの分岐方法 (後方への分岐,1〜2 ビットの分岐履歴表)

後方への分岐:PA-RISC のものと同じ
1 ビットの分岐履歴表:ダイレクト・マップされた 1 ビット,2048 エントリー, 分岐命令のアドレスの下位ビットから表を引く, 後で実際の分岐条件を反映させる

4.Pentium:256 エントリーの分岐ターゲット・バッファ(BTB)

各エントリーには分岐のターゲット・アドレスと 2 ビットの以前の分岐状況を含む,BTB は 4 ウェイ・セット・アソシアティブとランダム置き換え戦略を使用,BTB 中にない分岐は不成立と予測

5.PowerPC604:フル・アソシアティブ 64 エントリーの BTB と 512 エントリーの 2 ビット履歴表

BTB とは分離されたダイレクト・マップの 512 エントリーの 2 ビット履歴表: 分岐命令に当たると、まず分岐アドレスによって BTB が検索されエントリーが見つかると成立し、ない場合には不成立と予測し、シーケンシャルに命令フローが続く,1 度分岐結果が判明すると、分岐履歴表が更新され、履

歴表の値が次の分岐実行で成立を予測するなら、分岐アドレスがBTBに追加され、そうでなければ、BTBからアドレスが削除される

6.2 レベル適応戦略 (Yeh& Patt)

この戦略は他の分岐予測機構に比べて多くのハードウェア資源を必要とするが、分岐予測性能は高い

分岐履歴パターン表に加えて分岐履歴レジスタのセットをもつ, 分岐命令の実行の際には分岐アドレスの下位ビットをインデックスとして履歴レジスタをひく, シフトレジスタとして実装されている履歴レジスタにはその分岐の分岐履歴についての情報が含まれる, この情報は分岐履歴パターン表を引くインデックスとして利用され、現在の分岐予測を決定するのに必要な情報を取り出す

現在のマイクロプロセッサでは一般的にパイプライン処理が行なわれている。しかし、分岐命令の存在のために、パイプライン処理がストールすることがある。これは動的に分岐先の計算がなされる場合に、この計算結果を待ってから分岐先の命令をフェッチしようとするためである。特にパイプライン段数が深く、命令のフェッチ幅・実行幅が広く、さらに分岐先の計算が分岐命令の直前でなければ行なえないような場合に最悪の状況が生じる。これを回避するために、先に挙げたような分岐予測機構を備えるのが普通である。このように静的分岐予測機構や、動的な分岐履歴を利用するような動的分岐予測機構が提案され実装されてきたが、これらの分岐予測性能はほぼ飽和してしまったのが現状である。これに対して現在では、プロファイル情報などを利用して、分岐命令の特性に応じた分岐予測を行なう方式[MN96]などのように、ハードウェアコストと予測性能のバランスを改善する研究や、複数の分岐予測戦略を用意し、分岐命令ごとに適当な戦略を選択して利用する方法[LKM98]なども研究れている。

4.1.2 条件付き実行

命令ストリームから完全に条件分岐を取り除くテクニックとして条件付き実行がある。CRAYのようなベクトル・マシンでは、ベクトルマスクによって、以前からこの種の実装はされていた。この方法は、命令ストリームから分岐を削除するだけでなく、スケジューリング能力を向上させるという利点もある。

分岐予測と条件付き実行を同時に使うことで、分岐による性能への影響を低減することがある程度可能である。例えば、分岐予測ミスによるペナルティよりも、誤った予測に基づいて実行される命令数の方が少ない場合には、分岐予測ミスによって失うサイクル数は軽減できる。さらに全体的な動的分岐予測の成功率は、予測が難しい分岐のいくつかを条件付き実行によって取り除くことができれば、いくらか改善される。

本節では条件付き実行モデルについて概説する。条件付き実行とは論理的な条件の成行きに基づいた命令の条件付き実行 (完了) のことである。大きく分けて、「制限モデル (restricted)」「無制限モデル (unrestricted)」の 2 つに分類できる。

制限モデル

限られた数の新しい条件付き命令が使われ、明示的にプログラム変数に対するステートメントの実行による影響を遅らせるために使用する, 問題となっているステートメントを分岐の前に持ってきて、プログラム変数の代わりにあいているレジスタに書き込むことで実現する, そして特別な条件付きオペレーションのいずれかを使って、条件的にアクティブなプログラム変数を更新する,Alpha と HP の PA-RISC で使用されている

無制限モデル

すべての命令について条件付き実行できるモデル, 様々な実現方法があるが、その 1 つは、それぞれの命令にオペランド・フィールドを追加することである, 他に、特別な命令を導入して次の (条件付き実行でない) 命令の条件的な実行を制御する方法, Pnevmatikatos と Sohi の提案した"guarded execution model"がその一例で、次の命令を実行すべきかそうでないかの指定を行なう特別な命令を導入するものである

以下では、このような特別な命令のことを補償コードと呼ぶ。

ここでは実際に 3 つの条件付き実行モデル (Alpha, HP-RISC, guarded execution model) について説明する。

1.Alpha: 制限モデル, 条件付き移動命令

条件付き移動命令 (conditional move instruction) で制限モデルの条件付き実行をサポートする, 通常の move 命令の 2 つのオペランド (ソース, ディスティネーション) に加えて 1 つの条件値を指定するフィールドを追加したものを

条件付き移動命令という, 条件を満たした場合にレジスタの移動が可能だが、そうでない場合には阻止される, コンパイラでの条件付き移動命令の使用法は次のように行なう:

- 式計算を条件分岐の前に持ってきて本来使われるべきレジスタではなく空いているレジスタに結果を書き込む
- 条件付き移動を本来の分岐命令のところで使ってテンポラリ値を本来のレジスタに移動する
- 条件が成立したら計算式の本来の書き込み先に結果がある
- 条件不成立の場合には条件付き移動は実行されず実行中のアプリケーションの状態は変わらない

この条件付き実行方式にはいくつかの制限がある:

- コンパイラは式計算のコードの押し上げによって例外が生じないことを保証しなければならない (たとえば 0 での除算)
- メモリ・アクセス命令とフロー制御命令は条件付き実行できない
- コンパイラは条件付き移動命令に対して前もって結果をストアするための空きレジスタを割り当てなければならない, レジスタがあふれるようなときにはうまく行かない

2.HP: 命令無効化 (instruction nullification)

HP の PA(Precision Architecture) では、制限の少ない条件付き実行を実現するのに命令無効化を利用する。このアーキテクチャでは、制御フロー (分岐) と算術命令は、続く命令を実行すべきかどうかを指定することができる。算術命令が、それに続く命令の無効化をできることによって、コンパイラは分岐命令を削除することができる。Alpha の条件付き実行と同じような変換を、例外を発生するような命令をスキップするような命令も含めて行なえる能力をもつ。

3.Pnevmatikatos& Sohi: ガード命令 (guard instruction)

命令シーケンスの実行を制御するのにガード命令を使用する。ガード命令は 2 つのものを指定する。—条件レジスタとマスク値で、マスク値はその条件レジスタの内容に続く命令のどちらが依存しているかを示している。プロセッ

サ・ハードウェアはこの情報を用いて、与えられた命令がプロセッサの状態を変更してもよいかどうかを決定するのに使うダイナミック・スカラ・マスクを作成する。多重ガードのサポートは、以前に削除とマークされていなかったスカラ・マスクのエントリーを変更できるようにガード命令をすることで可能である。このアプローチは、ベクトル命令の ALU オペランドの代わりに、命令流の中の命令シーケンスの発行を制御するビット・マスクを備えていた以前のベクトル・プロセッサで使われていたベクトル・マスク・レジスタを思い起こさせるものである。ガードされた分岐命令をどのように扱えるのか、またガードすることでフェッチ・レーテンシにどのような影響があるのかについては不明瞭な点がある。

4.Cydra5:

もっとも一般的な条件付き実行のフォームをサポートしている。それぞれの Cydra 5 オペレーションは、128 の論理条件付き実行レジスタが目的の実行条件を含んでいるかを指定することで、条件付き実行を行なうことができる。選択した条件付き実行レジスが 0 でない場合には、オペレーションがマシンの状態を完了 (ライトバック)、変更することができる。すべての命令が条件付き実行レジスタを参照できる (しなければならない) ので、すべての命令シーケンスについて条件付き実行が可能である。

4.1.3 複数パスの投機フェッチ

第 4.1.1節、第 4.1.2節で述べた分岐予測と条件付き実行は、単一パスの実行を行なう場合を想定して開発された技術である。つまり、分岐予測は、2 つ以上のパスの中から 1 つだけを選択することで、プロセッサをあたかも分岐命令がない単一パスの実行を続ける状態にすることができる。しかし分岐予測がはずれれば、その分岐命令より先の命令が無駄になるだけでなく、場合によってはその命令の処理を取り消すためにプロセッサ性能に対してマイナスの影響を与えることがある。一方、条件付き実行は、複数の分岐パスの実行をまとめて 1 つのパスの実行に置き換えることで分岐予測がはずれた場合のペナルティをなくすことができる。しかし、必然的に無駄な命令の実行が含まれるため、分岐予測ミスによるペナルティが大きく、無駄な実行を行なってもなお性能的にプラスであるようなアーキテクチャ、つまり複数命令を同時に実行可能なアーキテクチャでのみ使われる。また、条件付き実行が行なわれるのは、分岐予測確率が低い分岐命令に限られ、通常はその分岐

命令の分岐先パスが2つであるようなTaken or Not-Taken型の分岐命令にのみ使われる。

条件付き実行も一種の複数パスの投機フェッチであるが、条件付き実行の場合には、補償コードを利用して複数パスの投機フェッチを安全に行なっているため、その投機展開規模に限界がある。通常は制御の分岐・合流がごく近い部分で使われるものである。これまでのプロセッサでは大規模な投機フェッチ機構を設けるだけの物理的な余裕がなかったため、このような小規模な複数パスの投機フェッチにとどまっていたが、今後のプロセッサでは物理的な余裕が見込まれ、分岐処理によるペナルティの軽減にこの余裕を利用することがプロセッサ性能に対してプラスになると考えられるため、より大規模な複数パスの投機フェッチ機構が検討されている。

ここでは、複数パスの投機フェッチに関する関連研究として、Disjoint Eager Execution (DEE)[US95], Threaded Multiple Path Execution (TME)[WCT98]について述べる。

Disjoint Eager Execution (DEE)

DEEでは複数パスの投機フェッチにおける基本的な展開方法について検討している。

複数パスへの命令展開の方式は図4.1のような3つに分けられる。図4.1では、各矢印が1つのベーシックブロック(実線はフェッチされたベーシックブロック、点線はフェッチされていないベーシックブロック)、●が分岐命令を表している。またここでは各パスへの遷移確率は、すべて70 %としており、2段分岐命令を進むと70 %×70 %＝49 %となる。各矢印の横の数字はこの確率であり、丸囲みの数字はフェッチ順序である。ここの例では6つのベーシックブロックをフェッチする際の各戦略の違いを示している。

Single Pathは各分岐命令で確率の高いパスのみを展開していく方式である。よって名前の通り、命令展開は一方向のパスのみとなる。よって展開規模を大きくすると、確率のより低いパスがフェッチされる可能性があり、図4.1でも4番目にフェッチされたベーシックブロックへの遷移確率24 %は、最初の分岐命令でフェッチされなかった方向への遷移確率30 %よりも低くなっている。

Eager Fetchは分岐命令ごとにすべてのパスへ命令を展開していく方式である。よって分岐予測ミスはなくなるが、不要な命令が爆発的に増える。図4.1でも6つのフェッチされたベーシックブロックのうち、正しいパスにあるベーシックブロッ

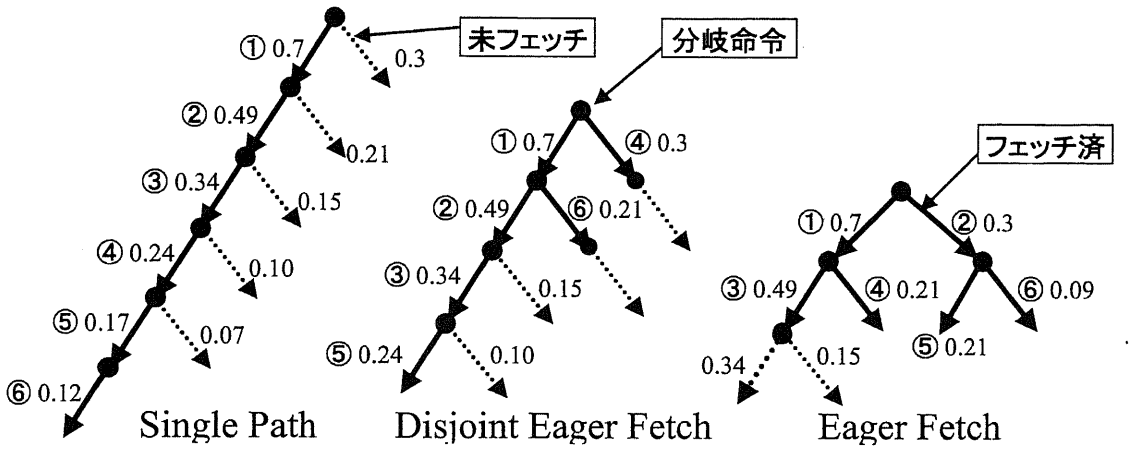


図 4.1: 3 つの投機フェッチ戦略

クは 2 つだけである。これは展開規模を大きくするとより顕著になり、正しいパスにあるベーシックブロックと不要なベーシックブロックの比は、N 段の分岐命令をまたいだ場合、N : $2(2^N\text{-}1)$ になる。正しいパスにあるベーシックブロックの割合は、例えば 3 段分岐命令をまたぐと 21 %、4 段で 13 %、5 段で 8.1 %、8 段では 1.6 %となってしまい、大規模化には向かない。

Disjoint Eager Execution は Single Path と Eager Fetch の中間の方式である。つまり、実行される確率のより高いパスにあるベーシックブロックを優先してフェッチする。図 4.1では、まず左下方向へ 3 つのベーシックブロックをフェッチした後、先頭の分岐命令のもう一方のパスにあるベーシックブロックをフェッチしていく。このようにすることで、大規模化をしてもそれほど多くの命令をフェッチせずに効率的な複数パスへの投機フェッチができると考えられる。図 4.1の例のように、各分岐命令における分岐確率がそれぞれ 70 %と 30 %の場合には、5 段分岐命令をまたいでもフェッチするベーシックブロック数は 8 個、8 段でも 23 個である。

次に問題となるのは、DEE においてどのようにして未フェッチのパスの中から実行される確率の高いパスを見つけ出すかである。DEE では static tree heuristic と呼ぶ方式でこのパス選択を行なっている。static tree heuristic の概略は次の通りである。

1. 複数の典型的なベンチマークプログラムを利用して、DEE を行なおうとしているプロセッサが利用している分岐予測機構の平均性能 (分岐予測成功率 p)

を測定する。

2. そのプロセッサで実行されるすべての分岐予測成功率が p であると仮定する。

3. 次に式4.1～式4.3で示される計算式によって計算された E_T, h_{DEE}, ℓ に従ってDEEの形を決定する。

4. DEEの形に従って、プロセッサ資源(管理できるパスの数(E_T、ALUの数など)を決定し、プロセッサを設計する。

DEEの形を決定する計算式は次の通りである。

$$E_T = log_p(1-p) + \frac{1}{2}h_{DEE}^2 + \frac{3}{2}h_{DEE} - 1 \quad (4.1)$$

$$h_{DEE} = -\frac{3}{2} + \frac{1}{2}\sqrt{8E_T - \frac{8log(1-p)}{logp} + 17} \quad (4.2)$$

$$\ell = h_{DEE} + log_p(1-p) - 1 \quad (4.3)$$

この関係式は $(1-p) > p^{\ell} > (1-p)^2$ の関係を満たすものである。この関係を図

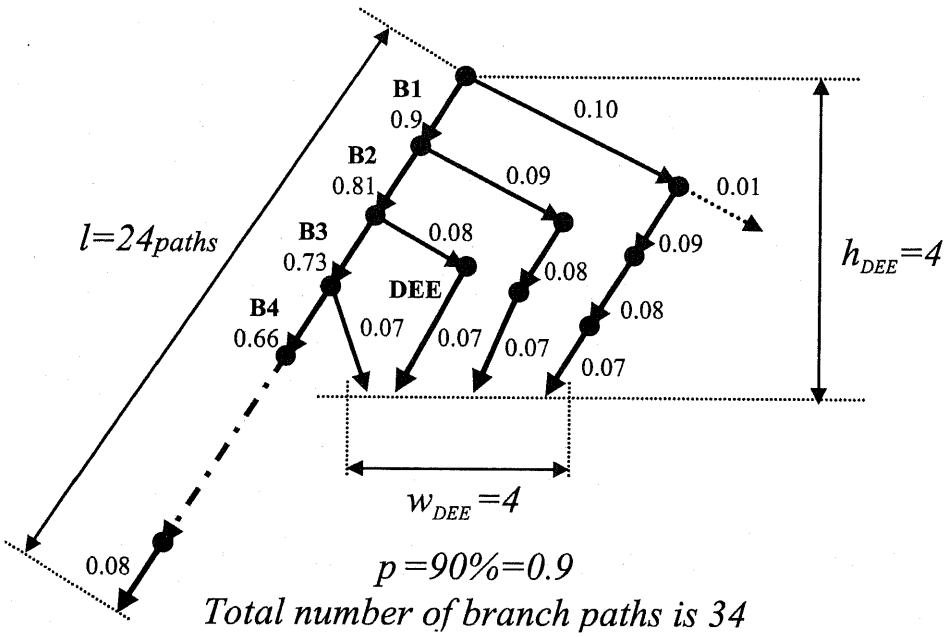


図 4.2: Static Tree Heuristic

示したのが図4.2である。図4.2は分岐予測成功率 $p = 90$ %の場合であり、$p = 0.9$

から式4.1～式4.3によって、$E_T = 34, h_{DEE} = 4, \ell = 24$ と計算される。つまりメインパスの長さが24、投機パスの深さと幅が4、パスの総数は34である。DEEでは$w_{DEE} = h_{DEE}$としているため、投機パスの深さと幅が同じになっている。この理由は次の通りである。

- メインパスの実行は、通常の分岐予測に基づく単一パスの実行と同様の方法で行なわれる。
- 投機パスの実行は、メインパスから分岐した後それぞれの投機パス毎に単一パスの実行と同様の方法で行なわれる。これは投機パスからさらに別の投機パスが生成されることを許すとより多くの資源が必要となり、制御も複雑化する。
- すべての分岐予測確率をpという単一の値に統一しているため、関係式$(1-p) > p^\ell > (1-p)^2$をみたすℓに対して、p^ℓとほぼ等しい遷移確率をもつ投機パスは必然的に$h_{DEE} = w_{DEE}$の関係を満たす。

図4.2においても、メインパス(左下に伸びる24paths)の先端への遷移確率が.08であり、DEEの投機パスの先端のパスへの遷移確率.07とほぼ等しくなっている。

DEEの性能評価はSPECint92に対して行なわれており、256の分岐パスを管理できる資源を仮定したときにスカラプロセッサに比べて31.9倍の性能が得られるとしている。しかし実際には256の分岐パスを管理できるとした場合、$p = 0.9$を仮定すると式4.1～式4.3から$\ell = 40, h_{DEE} = 20$となり莫大な資源が必要となる。ただしこの評価では分岐先が3つ以上になるような分岐命令およびトラップ命令に関する部分について言及されておらず、この性能評価は本来複数パスの投機フェッチにおいて性能を制約する大きな要因となるこのような分岐命令の影響がないものと仮定されている可能性がある。

Threaded Multiple Path Execution (TME)

Threaded Multiple Path Execution (TME)[WCT98]は、Simultaneous Multithreading Processorにおいて複数パスの投機実行を行なうことを目指した研究である。これによってマルチスレッド・プロセッサにおいて、分岐予測成功率が悪い分岐をまたぐ実行パスを複数のスレッドに分けることで同時に実行し、分岐予測ミスによるペナルティを隠蔽することができる。ここでは、TMEのパス選択機構について述べる。

複数パスの投機フェッチを行なう場合には、どのパスをフェッチするかが重要である。制御の複雑さの面では、できる限り Single Path に近い方式で命令展開した方がよいが、性能を稼ぐには Eager Fetch をした方がよい。TME では制御の複雑さを考慮し 1 つの primary path といくつかの alternate path という形で命令の展開を行なう。つまり alternate path からさらに alternate path を作ることはない。この点では第 4.1.3節で述べた DEE と命令展開の形は同じである。

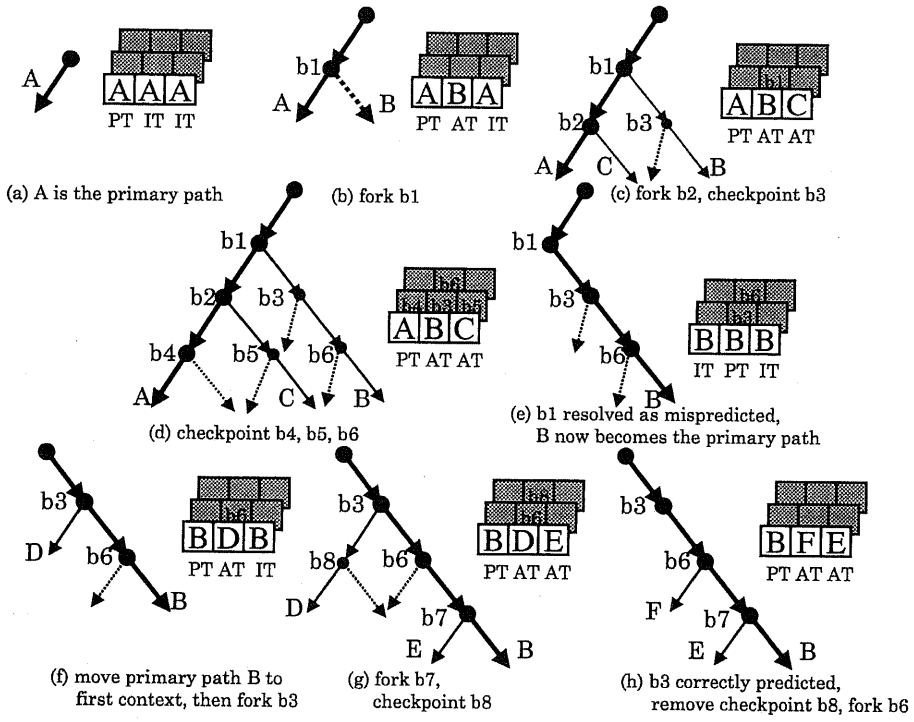


図 4.3: TME のパス管理機構

図 4.3は TME のパス管理機構を示したものである。図中の矢印はフェッチパスを示し、手前の 3 つのボックスはスレッドの管理テーブル、後ろの 6 つのボックスは各スレッドのチェックポイントの管理テーブルである。ボックスの下の文字は、PT = Primary Thread, AT = Alternate Thread, IT = Idle Thread を意味している。よって図 4.3の例では 3 つスレッドの各状態は PT が 1 つと AT もしくは IT が 2 つになる。

各図について簡単に説明する。

(a) まず最初のパスがPrimary Threadとなり、残りの2つのスレッドはIdle Threadになる。

(b) 最初の分岐ポイント (b1) でB方向のパスを新しいスレッドとして作るとAlternate Threadとして登録される。なおb1は両方向のパスがThreadとして作られているためチェックポイントとして登録されない。

(c) Primary ThreadのAの2つ目の分岐ポイント (b2) でさらに新しいC方向へのスレッドが生成されると、CがAlternate Threadとして登録される。さらにBのスレッドでは1つ目の分岐ポイント (b3) で一方向へのみ命令を展開していく。これはスレッドBがAlternate Pathであるので、新たなAlternate Pathを作ることができないためである。このように展開されないパスがある分岐ポイントではこの点をチェックポイントとして記録しておく必要がある。図4.3ではBの後ろのボックスにb3が登録されている様子が分かる。

(d) A, B, Cの各スレッドがそれぞれ次の分岐ポイント (b4, b5, b6) の先に命令を展開すると、すでに管理できるスレッド数3になっているため、すべての分岐ポイントがチェックポイントとして記録される。

(e) (d) の状態から分岐ポイントb1の分岐方向が決定しB方向に制御が移った場合には、分岐予測が外れたことになり、Primaryスレッド Aが捨てられる。またB以外のスレッドはPrimary Thread Aから作られたものであるから (なぜならば、Alternate Threadから新しいスレッドは作れないため) スレッド Cも捨てられる。つまり残ったスレッドはBだけとなりBがPrimary Threadとなる。

(f) Primary ThreadとなったBは次のフェッチすべきパスを選択して命令の展開を行なっていく。図4.3の例では1つ目のチェックポイント、つまり分岐ポイントb3のもう一方のパスDを新しいAlternate Thread Dとして生成している。

(g) 必ずしもルート (コントロールフローの最初の点、図4.3(g)ではb3) に近い分岐ポイントから命令の展開を行なう必要はない。例えば分岐ポイントb6の分岐方向に偏りがあり、まずは分岐確率の高いパスだけを展開し、より分岐方向が不明確な分岐ポイントb7で両方向に命令の展開を行なった場合が図

4.3(g) である。分岐ポイント b7 で新しい Alternate Thread E が生成されている。

(h) 分岐ポイント b3 の分岐方向が決定し Primary Thread が正しいことが分かると Alternate Thread D が捨てられ、次の分岐ポイントから命令展開が再び始まる。

さて、図 4.3の説明でも分かるように、まずどの分岐ポイントから命令の展開を行なうかの選択が命令の展開形に直接影響し重要な問題であることが想像される。TME ではまず各分岐命令に branch confidence counter というカウンタを設けてどの分岐命令の展開を優先するかを決定している。branch confidence counter は分岐予測が連続して当たった回数をカウントしているものである。優先順位の決定は次の 4 方式が検討されている。

oldest 最も古い未解決の分岐命令を次の展開候補として選択する。つまり最もルートに近い分岐ポイントからの展開を最優先にする。

latest 最近に登録された分岐ポイントを次の展開候補として選択する。つまり展開されたツリーの先頭に近い部分が最優先となる。

next アイドル・スレッドができた場合、チェックポイントに登録されている分岐ポイントを除いて、次にフェッチされた分岐命令を次の展開候補として選択する。

lowest もっとも branch confidence counter が低い値の分岐命令を次の展開候補として選択する。つまり連続して分岐予測が当たりにくい分岐ポイントからの展開が最優先となる。

これらの方式の評価によれば、next を除いて他の 3 方式はほぼ同様の性能であり、いずれのサンプルプログラムに対しても next の性能に比べて他の 3 方式の性能が上回っている。

次に、フェッチの優先順序を決める方式について述べる。各パスに対して fetch priority counter と呼ばれるカウンタを設けて、この値が小さいほどフェッチの優先度が高いとしている。このカウンタはそのスレッドのフェッチ命令数を表している。このカウンタを用いた選択方式として、TME では大きく分けて次の 4 つの方式を検討している。

high-constant Alternate-Path のスレッドの fetch priority counter には、Primary-Path のスレッドの fetch priority counter の最大値より大きい値になるように大きな値を加える。つまり Alternate-Path のスレッドは、Primary-Path のスレッドが命令キャッシュミスなどによって命令フェッチを行なえない場合にのみフェッチが行なわれる。この戦略では Alternate-Path のスレッドの fetch priority counter はスレッド生成時にある大きな値の定数に設定される。そしてこの Alternate Thread が Primary Thread になった時にこの定数分が減算される。ここではこの定数は 64 である。

low-constant high-constant とほとんど同じ方式であるが、Alternate-Path のスレッド生成時に加えられる定数がより小さい値である。ここではこの定数は 16 である。

variable スレッドの生成時にその分岐ポイントの分岐命令の branch confidence counter に応じた値が fetch priority counter に加算される方式である。つまり branch confidence counter の値が小さい (確信度が低い) 場合には、相対的に fetch priority counter の値が小さくなり、よりフェッチ優先度が高くなる。ここでは 3bit の branch confidence counter を利用し、ある定数 (ここでは 0, 8, 16 のいずれか) に 4 × (branch confidence counter) を加えた値としている。

primary-variable Alternate-Path のスレッド生成時における Primary-Path の fetch priority counter に応じた値が Alternate Thread の fetch priority counter に加算される方式である。ここでは、Primary-Path の fetch priority counter に (そのスレッドが生成された分岐ポイントの分岐命令の branch confidence counter) × (confidence multiplier) を加えた値としている。ここで confidence multiplier は 2, 4, 6, 8 のいずれかである。

これらの評価の結果はサンプルプログラムによって異なるものとなり、優劣はつけ難い。しかし複数プログラムを同時に実行するような場合には、ランタイム情報を最も使っている primary-variable 方式が全体的によい性能を示すことが示されている。

4.1.4 命令フェッチ能力

より高い命令レベル並列性の利用には、よりバンド幅の広い命令フェッチが必要となる。これに対して、例えばトレースキャッシュ[RBS96] を使った実現方法が研究されている。トレースキャッシュでは、プログラム中のあるポイントとそのポイントから先の複数の分岐命令の予測値をタグにして、命令ストリームを特殊なキャッシュ上に格納しておくことで、命令フェッチの際に、あるポイントと分岐予測値の組が一致した命令流を高い転送速度でデコーダに供給することができるようにするものである。

4.2 データ依存関係による制約

データ依存関係のうち、大きな問題となっているのは、メモリとの間のデータ転送である。図 2.2に示したようにCPU とメモリの速度差のために十分なデータがCPU に供給できない状況が生じてきた[Wil94]。並列性を利用するほどこの影響は大きく現れる。プリフェッチ技術 [CB94] の利用等によりこれを緩和することが行なわれている。さらに最近では、真のデータ依存関係をも解消する試みがなされており、従来言われていたプログラムから抽出できる並列性の限界を越える可能性も出てきた。ここでは、真のデータ依存関係を解消する試みである Value Prediction に関して、いくつかの方式について述べる。

データを予測するのには、マイクロアーキテクチャのステート情報からデータテーブルを引き予測値を引き出す。ステート情報としては様々なものが考えられるが、通常はプログラムカウンタを使う場合が多い。そのような意味では、キャッシュアクセス方式や、データのプリフェッチなどに関する研究と共通する部分がある。

Value Prediction の方式を大きく分けると次の 3 つになる。

Last Value Predictors 最も基本的な予測方式で、前回の値を予測値として用いるものである。これはデータ値に関する時間的な局所性に基づいている。Last Value Predictors にはいくつかの類似方式がある。例えば、予測の成功／失敗によって変化するカウンタを持ち、このカウンタがある閾値を下回ると予測値の置換えをするようにする方式などがそれである。

Stride Predictors 最近の複数回のデータ値から次の予測値を作り出す方式である。例えば、最近2回のデータ値がa_1, a_2の場合、次のデータ値を$a_1+(a_1-a_2)$

とするような方式である。これはループ内のデータなどに多く見られる、定数の増減が繰り返されるパターンや、一定値に保たれる場合に有効に働くことが期待される。ただし、この方式の場合には、大規模な投機処理を行なっている場合など、最近のデータがテーブルに反映されずに古いデータを元に予測を行なうと、予測成功率が大幅に下がることが考えられる。

その他の方式 より複雑な計算によって予測値を出す方式で、例えば、2つの予測値を持ちこれらを選択的に取り出すような方式がそれである。この方式ではループがネストしている場合に、内側のループと外側のループで異なる値を予測値とするなどという制御が可能となる。さらに、複数の予測方式を用い、各データ予測において最もよい成功率の方式を使い分ける方式 (例えば Last Value Predictors と Stride Predictors のハイブリッド [WF98]) もある。文献 [WF98] では Two-Level Value Prediction という方式が提案されている。これは、最近数回のデータ値を保存しておき、その出現パターンを合わせて記録し、このパターンを元に複数の保存されているデータ値の中の 1 つを予測値として選択する方式である。文献 [SE97] では、Finite Context Method Predictor という方式が提案されている。この方式も複数のデータ値を保存しておき、あわせてそのデータ間の出現関係を記録し、これらを使って次のデータ値を予測するものである。例えば、A, B, C という 3 つのデータがあるとして、これが A-A-A-B-C-A-A-A-B-C-A-A-A という順序で出現したとする。データ間の出現関係には A の後には A が 6 回、B が 2 回、C が 0 回などと記録されている。すると次は A の後なので、A の確率が高いと予測される。このテーブルはより次元を多くすることもできる。例えば、過去 3 回の履歴で引くようなテーブルを仮定すると、A-A-A の次の値を予測することになり、この例では B が予測されることになる。

Value Prediction については、実際のデータ値の予測だけでなく、Value Prediction を行なうかどうかの判定も重要である。データの性質として、Value Prediction が当たるものとまるで当たらないものがかなりはっきり分かれると思われ、さらに Value Prediction が外れた場合の回復機構がかなり大規模なものになると思われるため、予測成功確率の高いデータについてのみ予測を行なうことが重要になるのである。そのような意味では、Value Prediction の性能評価はどれだけのデータに関して良い確率で予測が可能となるのかが指標となるであろう。Value Prediction に関する研究は最近の研究であり、性能評価が統一されておらず、どの方式が最も

優れているかなどの優劣は現状ではつかない。また新しい方式の開発もこれから行なわれていくであろう。

第 5 章

大規模データパス・アーキテクチャの提案

5.1 概要

本論文では、CPU 自体の性能向上を目指す次世代のマイクロプロセッサ・アーキテクチャとして大規模 CPU のアプローチをとる。これは第3章で述べたように、プロセッサの基本性能をあげるためのアプローチであるからである。大規模 CPU としては、多数の演算器を持つプロセッサを考える。このようなプロセッサを従来のスーパースカラ構造の大規模化として考えるのは困難である。すでに数個の ALU をもったスーパースカラ・プロセッサにおいても性能向上限界が見えており、より多くの ALU を用意しても有効に利用できるとはいえない。そこで、プログラムの実行方式から考え直す必要がある。プログラムの処理の本質はデータフローであり、これを効率的に実行することを考えなければならない。現在のスーパースカラ等では、プログラムのコントロールフローを中心にして実行を行なっているため制御依存関係による制約等で性能向上は限界にきていることは先に述べた通りである。そこで、多数の演算器を相互に接続し、そこにデータパスを構築する実行方式を考える。このような多数の演算器によってデータパスを構築可能なプロセッサ (大規模データパス–VLDP–プロセッサ) を開発していく。VLDP アーキテクチャでは、可能な限りデータフローをハードウェアで実行することを考える。データパスを多数の演算器の相互接続によって構築し、相互接続網でデータフロー・グラフを実現することで、オーバーヘッドの少ない処理ができる。このような多数の演算器の相互接続網を ALU-NET と呼ぶことにする。

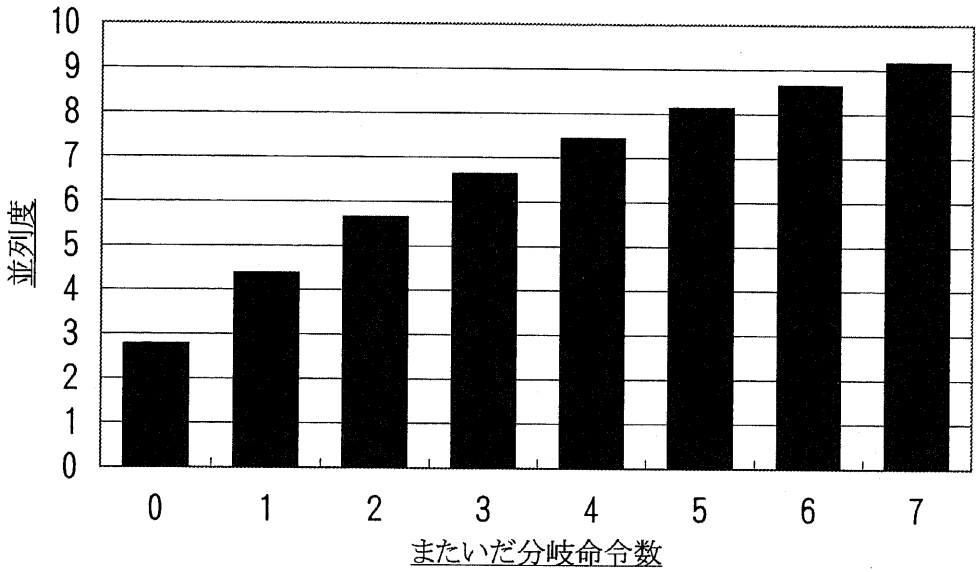


図 5.1: 分岐をまたいだスケジューリングと並列性の関係

図 5.1は、複数の分岐命令をまたいで命令スケジューリングを行なった場合に抽出可能な平均命令レベル並列性である[1]。図 5.1では、理想的に分岐予測が 100%成功して必要なパスのみを実行できた場合を想定している。図 5.1によれば、理想状態で 7 個の分岐命令をまたげば並列度が 9 程度になることが分かる。よって ALU-NET を使って効率的に処理を行なうには、制御依存関係が解消された状態のデータパスを多数用意し、各パスごとにデータ依存関係に基づいてデータパスを作り、それを ALU-NET 上でまとめて処理を行なえばよい。そのためには、まず制御依存関係を解消し、多数のデータパスを用意するための機構。次にこれらのデータパスのデータ依存関係に基づいて ALU-NET 上にデータパスを構築するための機構と、それを実際に実行する機構が必要である。これらの機構を VLDP アーキテクチャではそれぞれ次のように呼ぶことにする。

1. コントロールフロー先行展開
2. データパス先行展開
3. 実行（ALU-NET）

[1]SPEC92 を用いたシミュレーションによる

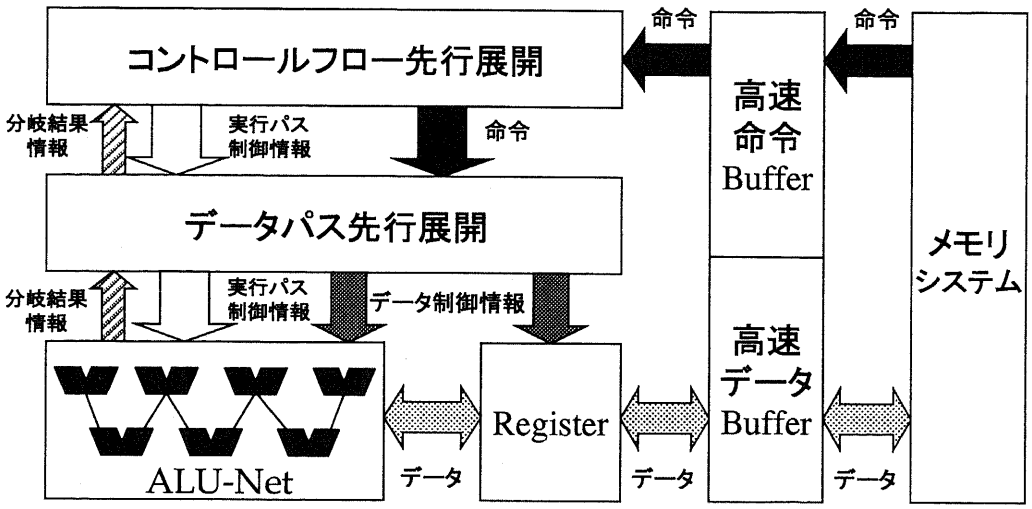


図 5.2: VLDP アーキテクチャ・ブロック図

図 5.2は VLDP アーキテクチャのブロック図である。コントロールフロー先行展開では、命令キャッシュから命令を読み込み、実行パス制御情報をつけてデータパス先行展開に命令を渡す。データパス先行展開では、実行パス制御情報と命令からデータ制御情報を作りだし、ALU-NET およびレジスタにおけるデータ転送制御を行なう。また実行パス制御情報によって ALU-NET 上にデータパスが構成される。分岐結果は ALU-NET から各制御部に返される。

各ステージの概要・実現のために必要な技術については次節以降で述べる。

5.2 コントロールフロー先行展開

コントロールフロー先行展開は VLDP プロセッサにおけるフェッチ機構である。VLDP プロセッサのような大規模なデータパス部をもつプロセッサでは、特に命令供給機構の構成が重要である。大規模な実行部を有効に活用するには、できる限り多くの有効な命令を安定的に供給する必要がある。つまり、コントロールフロー先行展開に求められる機能をまとめると次のようになる。

できる限り正しい命令の割合を多くする 理想的な命令フェッチを行なうには、分岐予測成功率 100 %の分岐予測機構を用意すれば良い。しかしながら、現実的には分岐予測成功率 100 %の分岐予測機構は用意できず、高く見積もって

95 %程度である。分岐方向の正しい予測ができるほど、フェッチ命令に含まれる有効な命令の割合が増えるので、より効果的に大規模な実行部が使えることになる。

できる限りストールをおこさない 大規模な実行部をもつプロセッサでは、その前段であるフェッチ・デコード機構もそれにあわせて大規模化するため、分岐予測に失敗したときのペナルティも大きくなる。例えば、フェッチ・ステージから実行ステージまでの長さが長くなったり、フェッチ命令バンド幅の増加により、実行前のフェッチ済み命令数が増えるので、分岐予測失敗時に削除される命令数(フラッシュ命令数)が増加する。またそれらの命令の管理情報も複雑化するため、回復機構などのレイテンシも大きくなる。つまり、大規模化においては、分岐予測失敗時のペナルティを削減もしくは隠蔽する技術が必要不可欠である。

高い命令フェッチ能力 大規模な実行部において高並列実行が行なわれると、それにあわせて命令フェッチ能力に対する要求も高くなる。特にできる限りストールをおこさないようにするために投機フェッチを行なうと、さらにフェッチ命令数が増えることが推定される。そのため大きいバンド幅、高いスループットが必要である。さらに、分岐予測失敗時の回復を早く行なうためには、レイテンシの削減も必要であるが、図 2.2によると大幅なレイテンシの削減はあまり期待できない。

以上のような要求から、コントロールフロー先行展開の動作は以下の 3 点を満たすものを考える。

動的な分岐情報の利用 分岐予測のための確率情報をコントロールフロー・パス別に管理することによって分岐予測を効率的に行うものである。

資源が許す限り複数パスへの投機フェッチ動作を行う 効率的分岐制御に基づき多くの分岐をまたがって資源が許す限り投機的に命令をフェッチするものである。多くの命令をフェッチすることよってコントロールフローの構築を行い制御依存関係を解消した命令列をできるだけ多く抽出することが目的である。さらに、多くの分岐を越えた投機的フェッチにより複数のパスが構築されるため分岐予測がはずれた場合でもその影響が小さいと考えられる。

命令列のストリーミング 高い命令フェッチ能力を得るためには、メモリデバイスの高スループット化を利用することが必要であろう。命令列を1つの命令ストリームとしてバースト的にフェッチできるようにするのが理想である。場合によっては複数パスの命令列を1つの命令ストリームとして構成することも必要であるかもしれない。

コントロールフロー先行展開の動作は 図5.3 に示すように、命令をフェッチしつつコントロールフロー・パスの構築を行い、このコントロールフロー・パスに基づき情報の管理を行うものである。命令フェッチ時、分岐命令に到達した場合のその先の投機的フェッチは分岐予測の確率情報に基づいて行われる。結果として、投機的にフェッチされるパスとされないパスが存在する図5.3 に示すようなコントロールフローパスが構築される。このパスのうち命令実行の結果、実行パスが確定し不要となったパスが切り捨てられるとともに、分岐確率情報の更新が行われる。

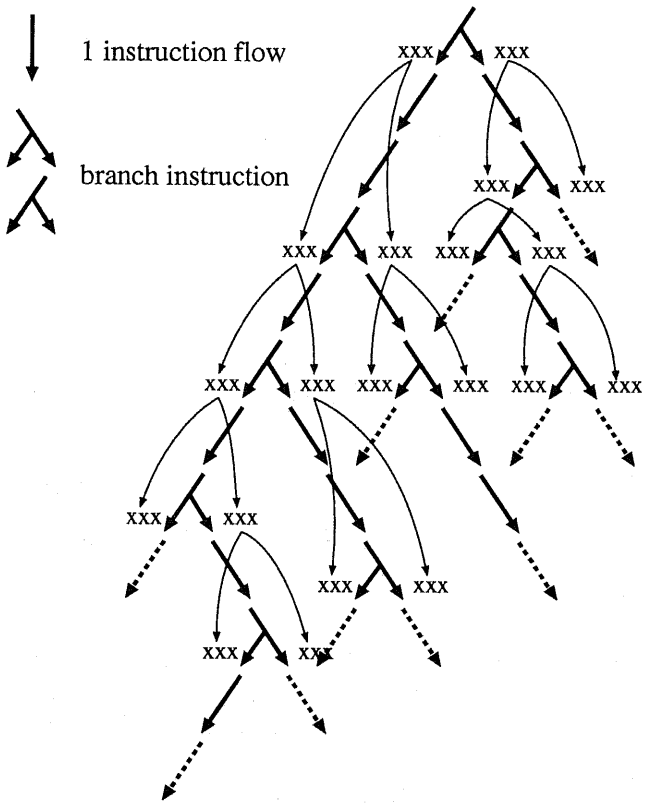


図 5.3: コントロール・フロー先行展開のイメージ

具体的に 図 5.3 において矢印は 1 命令であり、実線が実際にフェッチされた部分、点線が命令は存在するがその時点ではフェッチされていない部分である。さらに、xxx は分岐命令における分岐確率情報を意味し、細い線の矢印はこの分岐確率情報の依存関係を表している。図 5.3 は、命令のフェッチ状況（コントロールフロー）を表すもので、データ依存関係（データパス）を表しているものではない。さらにフェッチは制御パスの構築に行うものであるため、フェッチに伴いデータ処理は行われない。

コントロールフロー先行展開において管理される情報は、コントロールフローパスの情報（パスの切捨てを行う場合に必要）と分岐確率情報である。これらを総合して環境情報と呼ぶことにする。

コントロールフロー先行展開の実現において開発すべき技術課題は次の通りである。

複数パスへの分岐予測機構 従来の分岐予測機構はシングルパスへの命令展開をするための機構として設計されてきた。そのため VLDP がおこなうような複数パスへの命令展開における分岐予測機構を新たに検討する必要がある。

分岐パスの実行確率の管理 複数パスへの投機フェッチを行なうには、次にどの分岐ポイントから投機的に命令をフェッチするかが重要な課題である。このパス選択機構の性能により複数パスへの命令展開性能が大きく変わってくると思われる。

命令転送能力の向上 メモリからの命令フェッチ能力の向上を目指したハードウェア、およびソフトウェア的な新規技術が必要になってくると思われる。

本論文では、「複数パスへの分岐予測機構」に関して、第 6章および第 7章で定量的に考察を行なう。また「命令転送能力の向上」に関しては第 10章で簡単に考察する。

5.3 データパス先行展開

VLDP では ALU-NET に命令を割り当て接続網上にデータパスを構築し、データを流すことで実行を行う。ここでは、実行すべき命令を解析し ALU-NET への割り当てを行うステージをデータパス先行展開と呼ぶ。

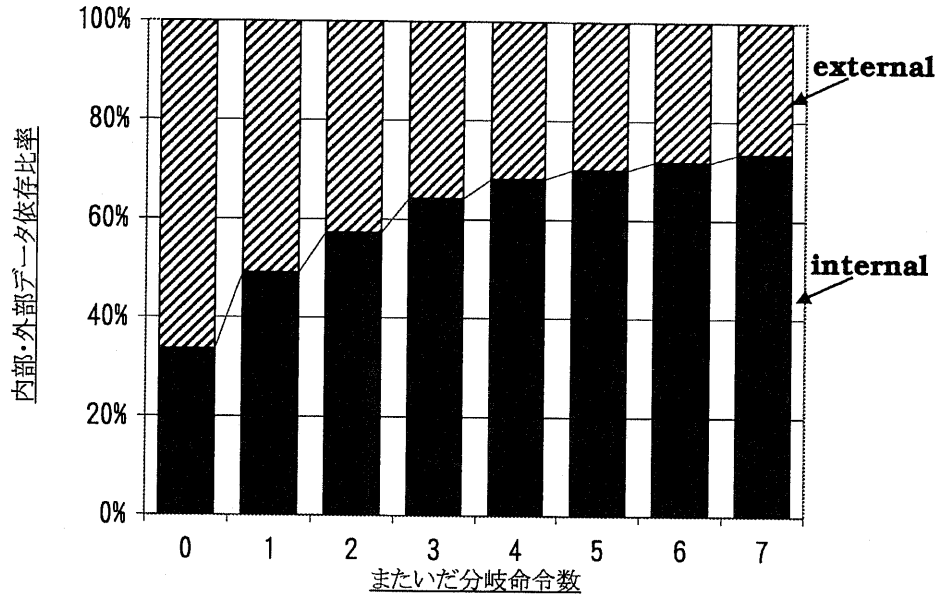


図 5.4: ローカルなデータ・アクセス

従来のレジスタオペランドによる演算はレジスタを介した集中型のデータ交換と見ることができる。しかし、レジスタアクセスに制御が集中することは、将来多数の演算器を利用する際にハードウェア的な障害となると考えられる。特にVLDPでは、コントロールフロー先行展開により複数実行パスの命令がフェッチされるため、その影響はさらに大きくなると思われる。ここで従来のプロセッサと同様にレジスタによるオペランド供給のみで実行を行うと仮定する。フェッチ命令数の増加や、全後続実行パスで解放されるまでリネームレジスタを保存しなければならないために起こるリネームレジスタのライフタイム伸長によって、必要レジスタ数が増加することが予想される。

そこでVLDPでは、ローカルなアクセスをALU-NET上の接続として吸収し、データを直接演算器間で転送する。図5.4は複数の分岐命令をまたいで命令フェッチを行なった場合に、それらの命令間に存在するデータ依存関係を調べたものである。internalとはデータ依存がフェッチされた命令間にあるもの、externalとはデータ依存がフェッチされた命令以外の命令との間にあるものである。図5.4によれば、分岐命令を多くまたぐ程、データ依存がフェッチされた命令間にある確率が高くなり、ALU-NET上の接続で吸収できる確率が高くなる。このようなデータ転送を行なうことで、従来では演算前後に制御を行って解決しなければならなかっ

たレジスタアクセスを、演算前に解決可能な ALU-NET の制御に置き換えることができ、制御に関するレイテンシを削減できる。ここでさらに演算器がデータを出力する際に次の演算器の発火を行うことにより制御の分散化を実現できるため、多数の演算器を持つ大規模 CPU の実装が可能になる。

一方、ALU-NET は加算器、乗算器、論理演算器、シフタ等基本的な回路を多く集め、相互に接続したものである。レイテンシによる実行速度の低下を防ぐため、接続網にはゲート数が少ないことが要求される。

データパス先行展開の実現において開発すべき技術課題は次の通りである。

データ依存関係の効率的抽出 データパス先行展開では、データ依存関係に基づいてデータ転送および命令のマッピングを行なう。そのため、データ依存関係の抽出においては、その効率化が必要である。最終的には、ソフトウェア的なサポートが必要になると思われる。

データ転送の効率化 図 5.4で示したように、VLDP では効率的なデータ転送もその特徴の 1 つである。しかし、例えば実際のデータ転送において、そのデータの生存期間 (life) を動的に求めることは一部を除いて容易ではなく、ソフトウェア的なサポートが必要になると思われる。その他、メモリ階層間におけるデータ転送などにおいても、既存のキャッシュ技術などでは、大規模化に応じた性能向上が得られるか疑問がある。

命令の演算器へのマッピング 次節で述べる ALU-NET は、VLDP プロセッサの特徴の 1 つであるが、ALU-NET を有効に利用するためには、命令の演算器へのマッピングが効率的に行なわれなければならない。演算器資源の管理も含めてデータ依存関係と ALU-NET の管理におけるアルゴリズムが必要である。

5.4 ALU-NET

ALU-NET は多数の演算器とそれらの接続網で構成される。しかし、すべての演算器間に接続をもたせると、その数が爆発的に増えてしまうため、ある限られた自由度のネットワークとすることにしている。命令ウィンドウ・サイズを 40、演算器数を 100、1 つの演算器が最大 3 つの他の演算器との間に接続網を持っているとした場合に、62 %のデータ転送が接続網に吸収されることが分かっている [吉瀬 98]。

ALU-NET の実現方法については、今後の課題である。ALU の相互結合によって、例えば、スーパースカラ方式などのフォワーディングパスを扱いやすく、また効率的にしたり、非同期的動作の可能性によって従来のパイプラインステージごとの同期的処理におけるオーバーヘッドの削減などが期待できる。ALU-NET の規模については、コントロールフロー先行展開やデータパス先行展開部の性能による部分が大きいが、簡単な考察は第10章でおこなっている。

第 6 章

大規模投機処理の分岐予測機構への影響

本章では、大規模な投機処理のための命令供給システムに関して、命令展開の展開方式と、大規模な投機処理を行なう場合の分岐予測機構への影響について検討し、新しい分岐予測機構の提案および評価・考察をする。

6.1 大規模先行展開と分岐予測機構

大規模データパス・アーキテクチャのような大規模な命令先行展開を行なう場合には、いくつかの展開方式が考えられる。

Single Path 分岐予測機構を用いて、1 つのコントロールフロー・パスの命令だけをフェッチする方式。分岐予測成功率が非常に高い場合は、フェッチされた命令が有効な命令である確率が高くなるので命令ウィンドウを有効に利用できるが、分岐予測性能が低い場合には非常に資源の無駄使いとなる。この方式は分岐予測機構の性能に大きく影響を受けるため、現在の 80〜95 %程度の分岐予測成功率しかない分岐予測機構を用いて大規模化することは現実的ではない。

コントロールフロー先行展開 大規模データパス・アーキテクチャのフェッチ機構。選択的に複数パスの投機的フェッチを行なう方式。分岐予測機構から、各分岐先への分岐確率値を動的に予測し、その確率値によって 1 つ以上のパスの命令展開を行なう。命令展開の形としては disjoint eager-execution[US95] とほぼ同じである。

Eager Fetch 各分岐命令ごとに分岐成立／不成立の両方のパスへ命令を展開し、すべてのパスのフェッチを行なう方式。大規模な展開を行なうと、不要な命令をフェッチしてしまう率が高くなる。分岐予測機構は不要であり、基本的に必要な命令は必ずフェッチされていることになるが、SPARC コードにおける JMPL 命令のようなレジスタ間接アドレッシクングで分岐先が決定されるような分岐命令に対しては、分岐予測を行なわなければならない。

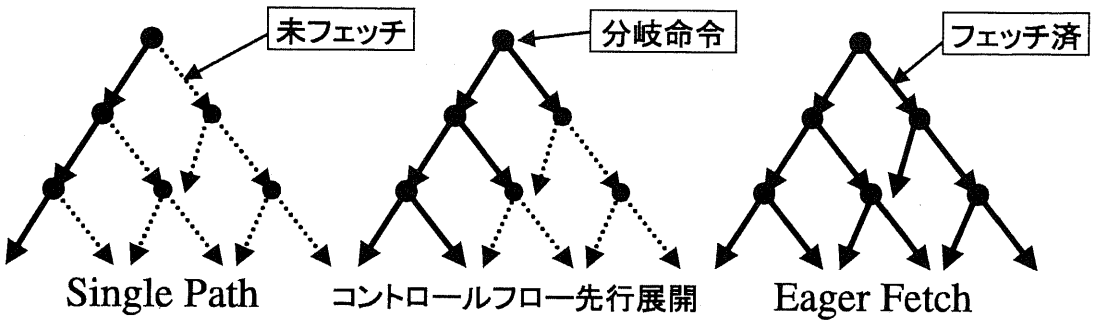


図 6.1: 命令の先行展開方式

コントロールフロー先行展開が行なうような大規模な命令先行展開において、従来ではあまり問題とならなかった新たな問題が分岐予測機構に生じると考えられる。その１つとして動的な分岐履歴情報の更新が投機フェッチによって分岐予測ポイントと分岐結果が履歴表に反映されるポイントが離れることに起因する分岐履歴情報の更新遅れに関する問題について考察を行なう。

6.2 分岐履歴情報の更新の問題

動的な分岐予測においては、通常 BHT(Branch History Table) や BTAC のように分岐履歴情報を使って分岐予測を行なう。しかし大規模な命令先行展開をおこなうコントロールフロー先行展開のような機構でこのような動的な分岐予測機構を利用する場合、予測時に使用する分岐履歴情報が古いものである可能性がある。つまり、図 6.2のような命令流が実行される場合、同じ分岐命令が 3 回繰り返し実行されるが、フェッチポイントにある一番下の分岐命令の分岐予測は、真中の分岐命

令の分岐結果が未定のため、一番上の分岐命令の分岐結果までしか反映されていない分岐履歴情報をもとに分岐予測をしなければならない。

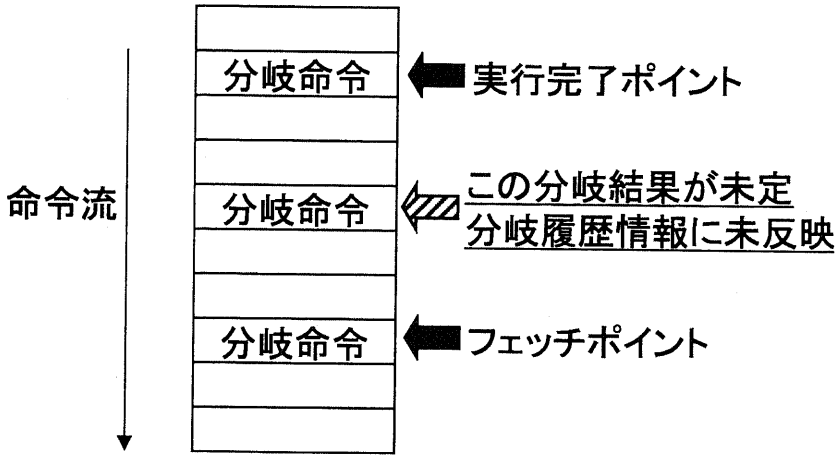


図 6.2: 実行完了ポイントとフェッチポイント

6.3 更新型 BHT の提案

第 6.2節で述べたように、大規模な投機処理を行なう場合には、分岐予測の際に最新の状態ではない古い状態の分岐履歴情報を用いて分岐予測を行なうことになる。条件分岐命令 Bicc などの分岐予測には、BHT (Branch History Table) などがよく用いられるが、このような古い情報を用いた分岐予測を行なうと、分岐予測精度が落ちる可能性がある。そこで、投機的にフェッチしたパスの分岐履歴も、投機的に履歴情報として登録することを考えこれを更新型 BHT、通常の BHT を非更新型 BHT と呼ぶことにし、両者の性能差について調査した。

図 6.3は更新型と非更新型の違いを摸式的に示したものである。この例では分岐履歴 bit として 2bit をもち、このコントロールフローグラフでは、命令先行展開の際にコントロールフローパスごとに分岐履歴情報を更新しながらフェッチをする場合 (図 6.3左) が更新型、フェッチ開始ポイントにおける分岐履歴情報をそのまま使ってフェッチをする場合 (図 6.3右) が非更新型である。図 6.3の四角に囲まれた “00” などの数字は分岐履歴情報 bit を表しており (この例では 2bit)、ここでは● はすべて同じ分岐命令、矢印はコントロールフローを表しており、左側が分岐成立

(履歴 bit に 1 が登録される)、右側が分岐不成立 (履歴 bit に 0 が登録される) ことを示している。

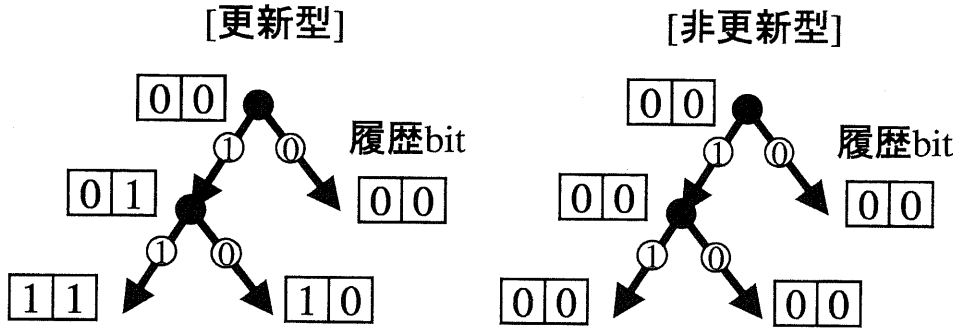


図 6.3: 分岐履歴情報の更新／非更新

6.4 更新型 BHT の性能評価

本節では、第 6.3節で提案した更新型 BHT の性能について調査し、従来の非更新型 BHT と比較をする。

図 6.5～図 6.10に示したのは、各サンプルプログラムに対する更新型 BHT の分岐予測成功率である。各グラフでは上図が PowerPC 型、下図が UltraSPARC 型の BHT に関する評価である。この 2 つの違いは履歴 bit の状態遷移にある。図 6.4 はさまざまな履歴 bit の状態遷移を示したものである。ここでは、2bit PowerPC mode と 2bit Ultra SPARC mode について調査した。各図の横軸は投機処理の規模および BHT のエントリ数を示している。ここで、1024の上の 0 は、投機処理をしていない状態と同じ状態の分岐履歴表を使って分岐予測を行なうことを示しており、更新型 BHT の分岐予測性能を表している。一方 1～8 は、その数だけの分岐命令をまたいだ先の分岐予測を行なう場合を示しており、例えば 8 は 8 つの分岐命令をまたいだ先の分岐予測を現時点での分岐履歴表を使って行なった場合の分岐予測性能を示している。つまり、8 段分の分岐履歴情報が不足している状態の古い分岐履歴表を使って分岐予測をしたことになり、投機レベル[1]N の非更新型 BHT

[1]投機レベルとは、最大何段の分岐命令をまたいで投機処理ができるかを言う

の分岐予測性能は、横軸が N の部分のグラフになる。シミュレーション環境は表

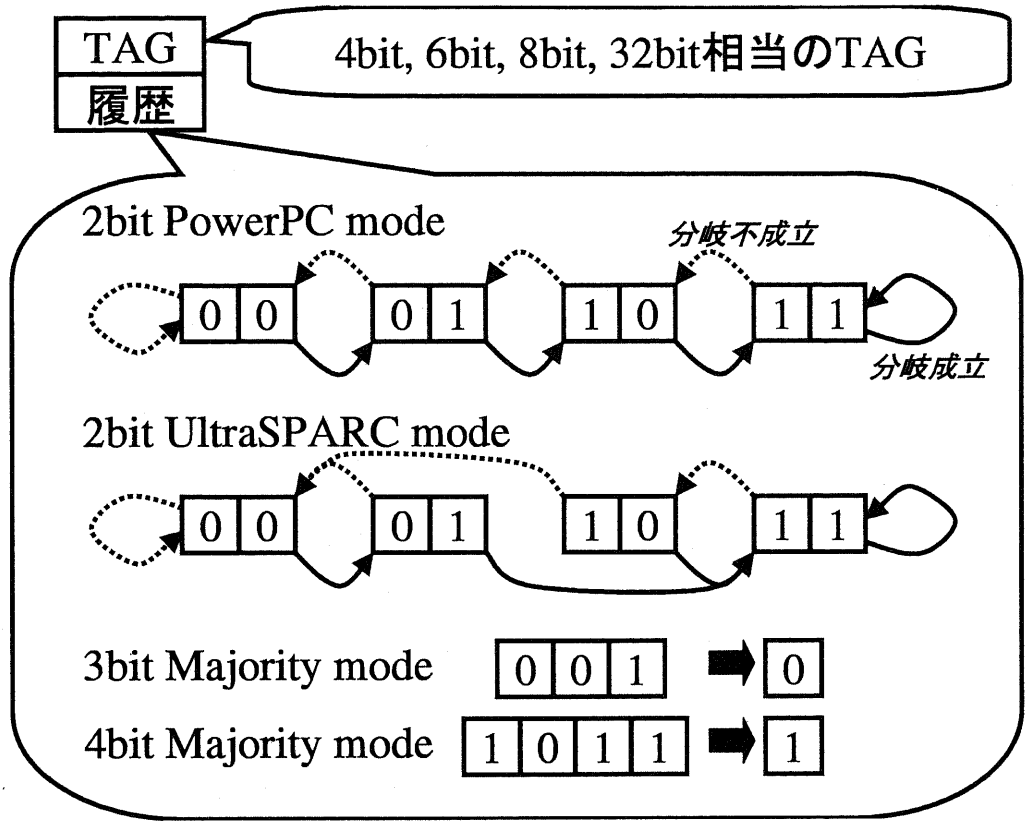


図 6.4: 分岐履歴情報 bit の状態遷移

6.1の通りである。

サンプル・プログラム	cc1, compress, go, ijpeg, li, perl (SPECint95)
命令トレース規模	10 億命令トレースを使用 (SPARC)
(非) 更新型 BHT 設定	1,024～8,192 エントリ エントリ連想度 ダイレクトマップ エントリ置換戦略 FIFO

表 6.1: 更新型／非更新型 BHT の性能シミュレーション環境

図 6.5～図 6.10で黒い棒は分岐予測が成功した割合、灰色の棒は分岐予測に失敗した割合、斜線の棒は該当分岐命令に対するエントリがないために分岐予測がで

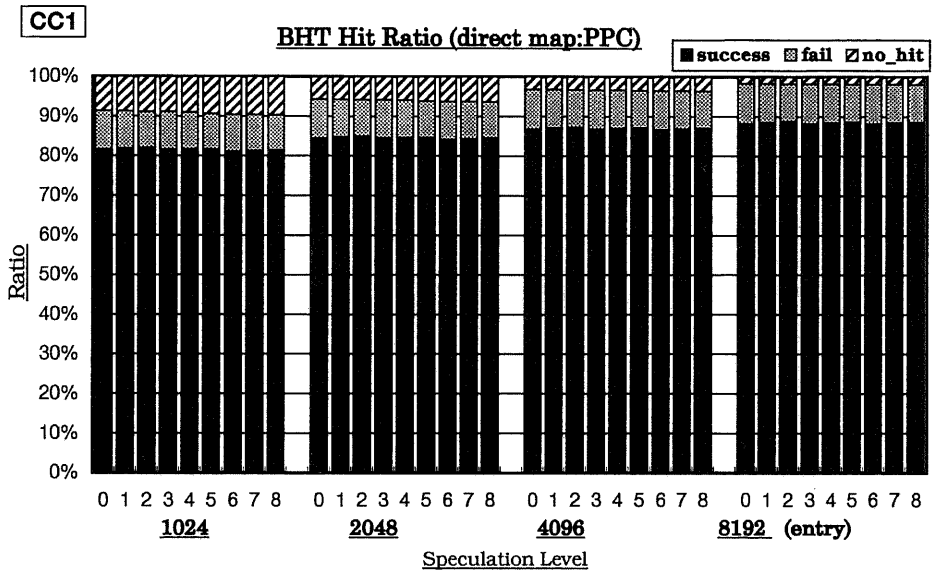
きなかった割合を示している。全体的な傾向としては、投機レベルが大きくなると分岐予測成功率が若干低下し、失敗率とエントリがない率が若干上がるが、大きな違いはなく、場合によっては逆の現象が起こったり、ランダムに変化している場合もある。また、エントリ数が1024エントリと8192エントリでは性能に大きな差が出る場合もあり、その差が大きいものでは10％近い場合もある。

以上から、更新型BHTでも非更新型BHTでも、その分岐予測性能については大きな違いがないことがわかる。つまり、分岐履歴の更新遅れによる分岐予測性能への影響はあまりないと言える。この理由としては、プログラムのトレースデータなどの調査から次の2点が主な原因であると考えられる。

- 同じ分岐命令が時間的にごく近い部分で繰り返し出てこない場合には、もともと更新遅れによる影響はでず、このような分岐命令も相当数存在する。

- 同じ分岐命令が時間的にごく近い部分で繰り返し出てくるような場合としては、ループの終端にある分岐命令が考えられるが、通常このような分岐命令は繰り返し実行され、かつ分岐方向に偏りがあるため、分岐予測成功率が高く、かつ、たとえ分岐履歴情報の更新遅れによって1～2回程度分岐予測ミスが増加しても全体性能に与える影響はわずかである。さらに、実際には分岐履歴の更新遅れによる影響は分岐予測ミスをするタイミングがずれるだけである場合がほとんどである。

よって、大規模投機処理においても従来のままのBHTを用いることが可能であると言える。また、BHTエントリ数に関しては、サンプル・プログラムにより異なるが、4096～8192エントリ程度あれば、ほぼ性能に変わりがないことが分かる。



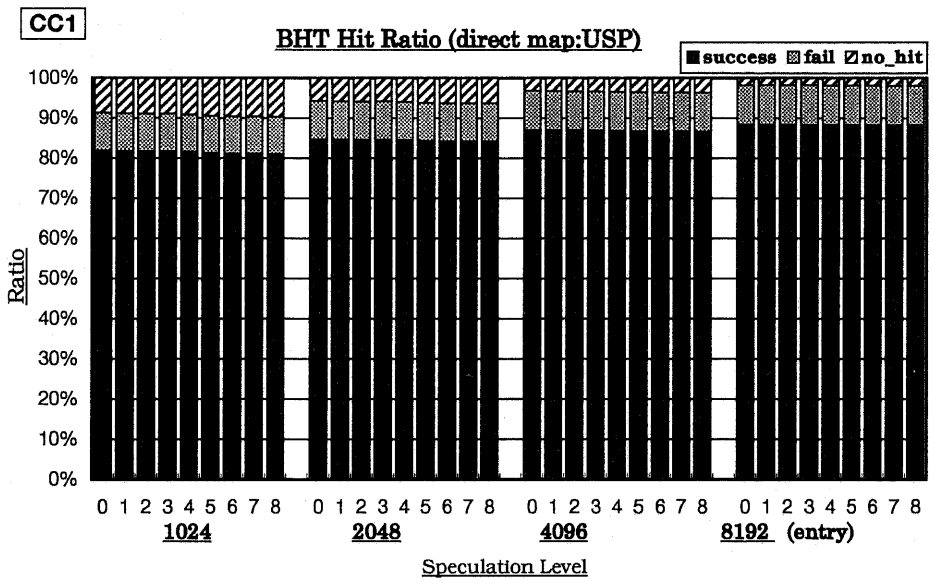
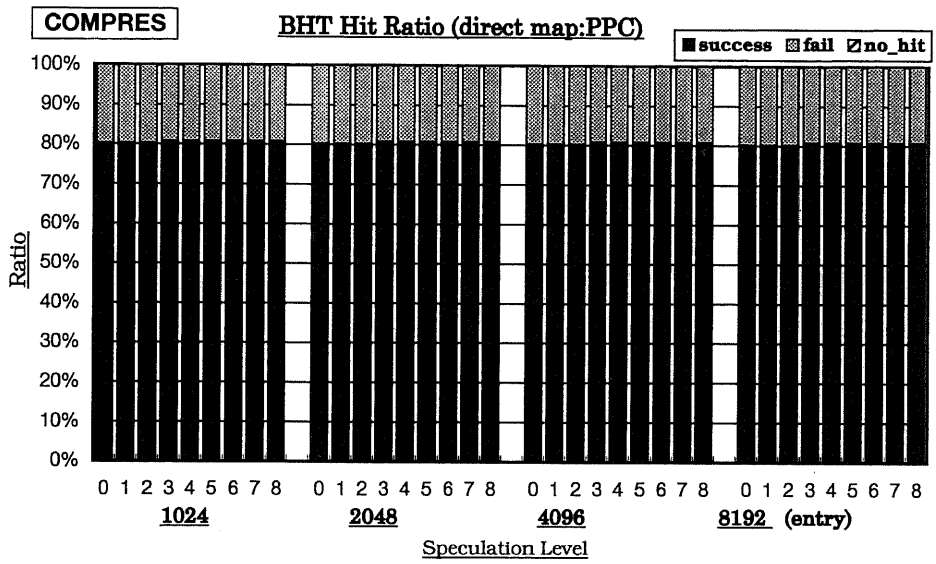


図 6.5: 更新型／非更新型 BHT の分岐予測成功率 (CC1)



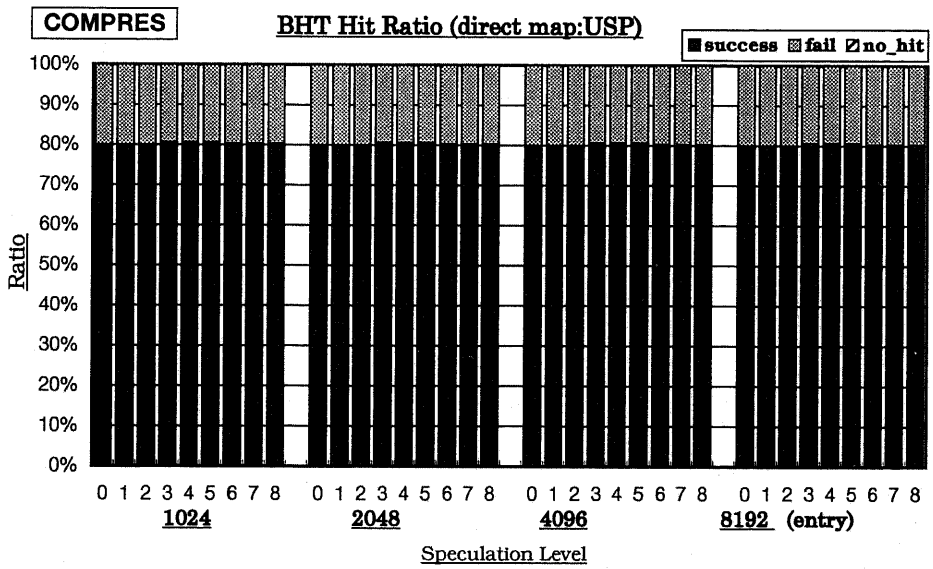
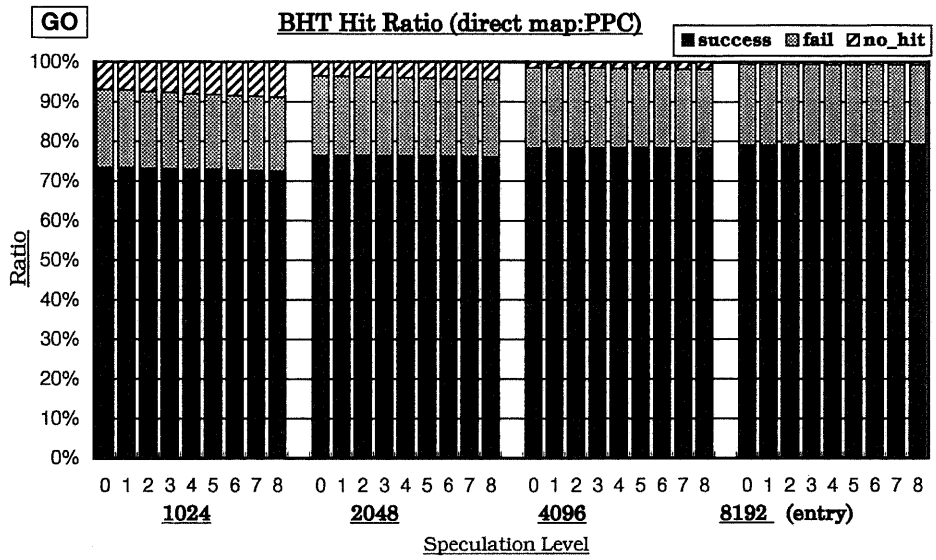


図 6.6: 更新型／非更新型 BHT の分岐予測成功率 (COMPRESS)



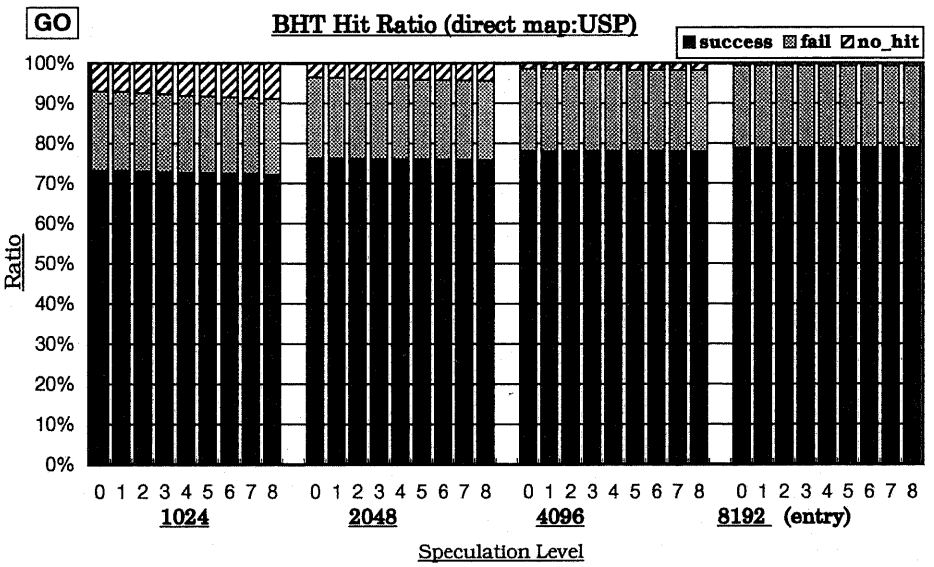
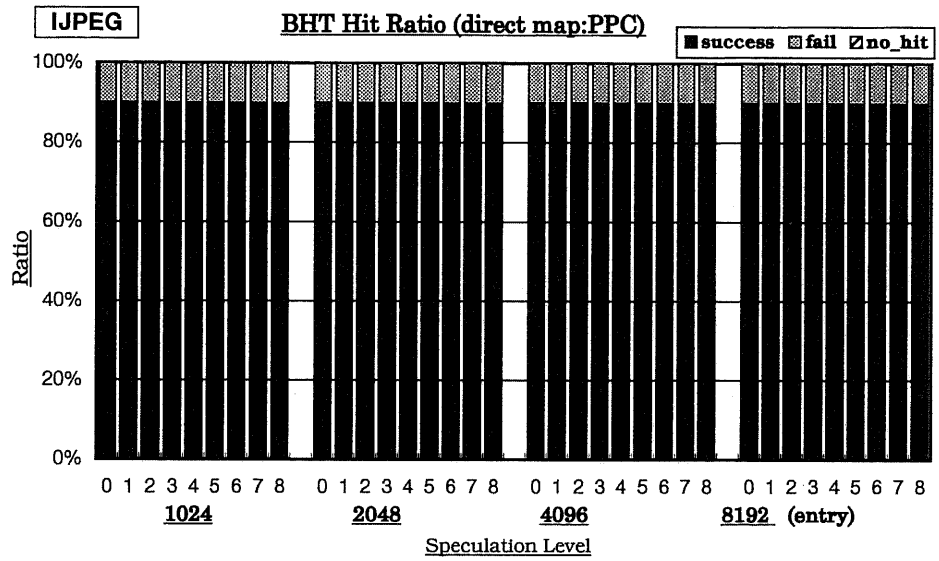


図 6.7: 更新型／非更新型 BHT の分岐予測成功率 (GO)



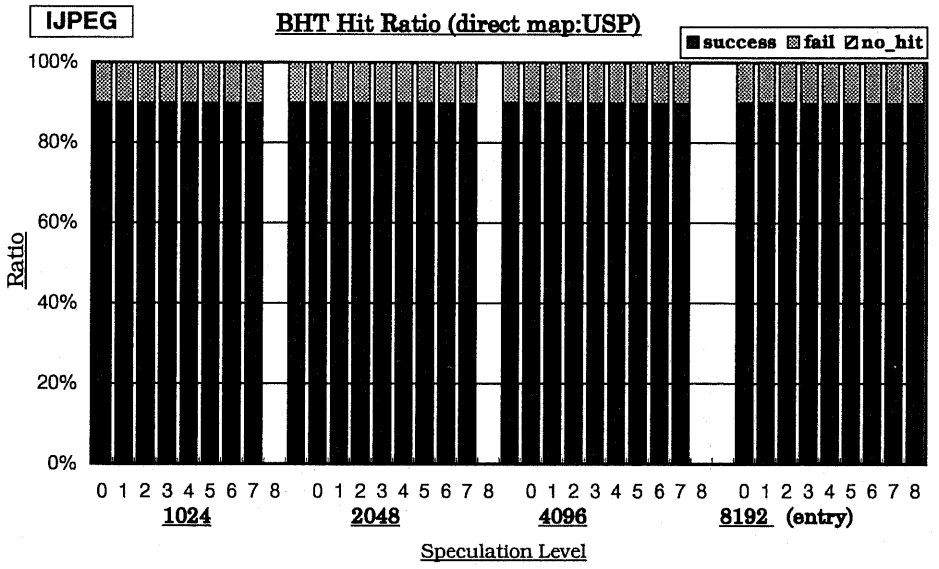
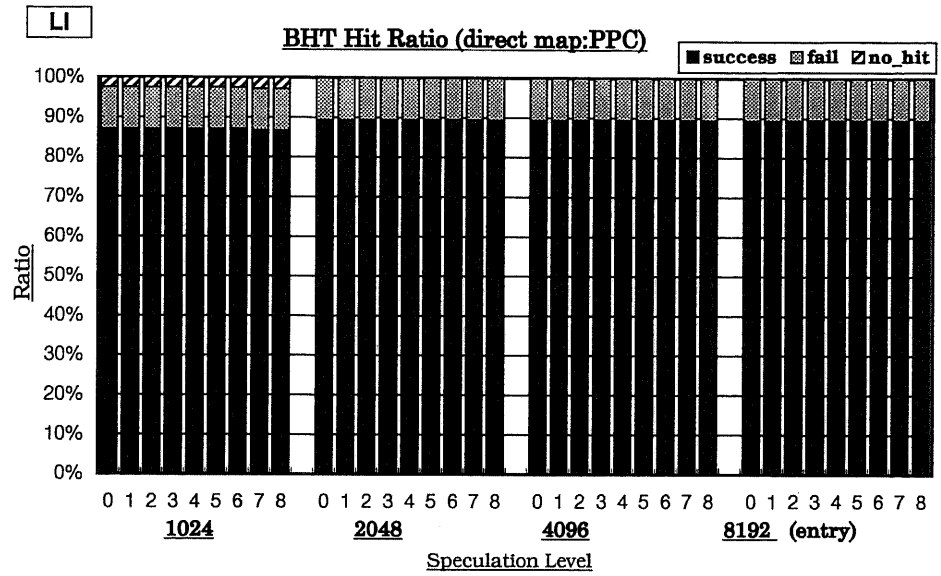


図 6.8: 更新型／非更新型 BHT の分岐予測成功率 (IJPEG)



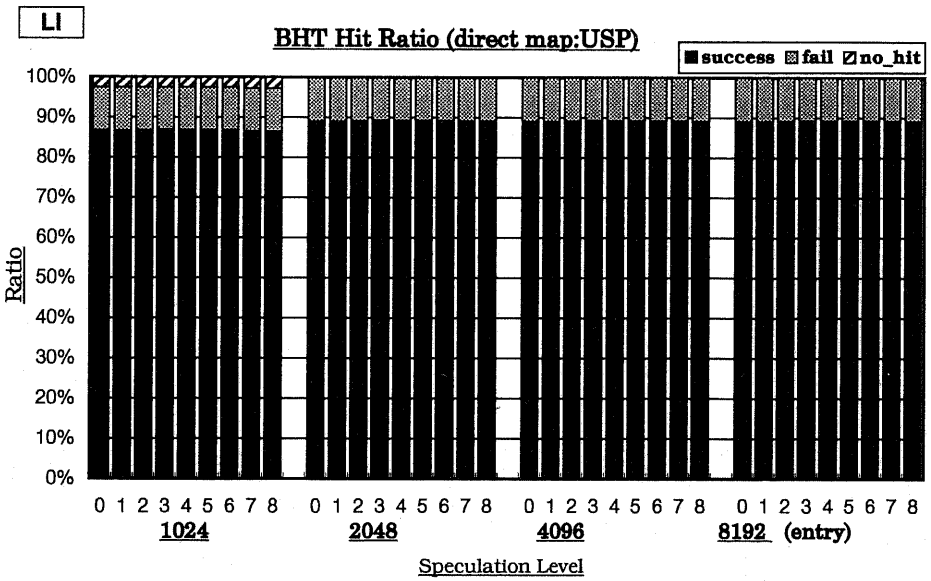
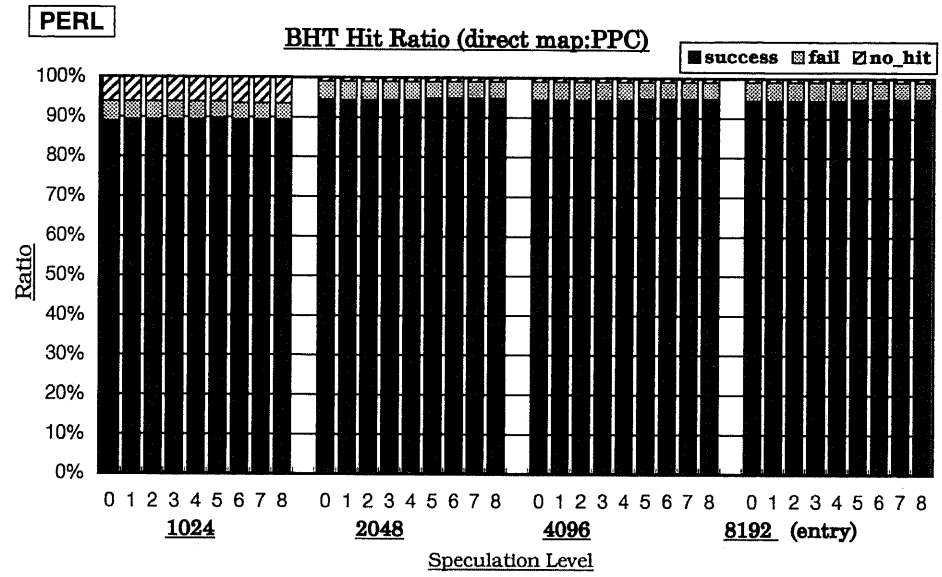


図 6.9: 更新型／非更新型 BHT の分岐予測成功率 (LI)



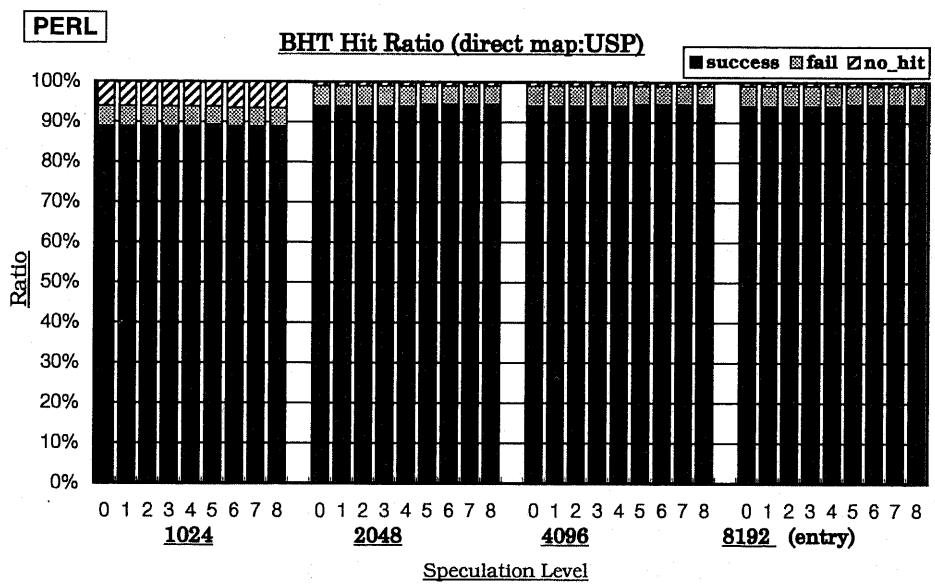


図 6.10: 更新型／非更新型 BHT の分岐予測成功率 (PERL)

第 7 章

複数パス投機フェッチのための分岐予測機構

本章では、複数パスへの大規模な投機処理を行なう場合に必要な分岐予測機構の提案および評価・考察を行なう。

7.1 レジスタ間接アドレッシングの問題

本研究でシミュレート環境としているSPARCアーキテクチャの分岐命令には表7.1のような4種類がある。ここで特に問題となるのは、この中のアドレス計算がレジスタ間接で行なわれるJMPL, Ticc命令である。Bicc, CALL命令の分岐先が図7.1(a)のように2つ以下であるのに対して、JMPL, Ticc命令の場合、分岐先がその時点でのあるレジスタの値によって計算される場所となり、図7.1(b)のように分岐先が複数になることがある。よって、命令展開の際にフェッチすべき命令をどのように選択するかが大きな問題となる。なお、Ticc命令はトラップ命令であり、Ticc命令で分岐条件が成立した場合は、命令展開がTicc命令で切れるので、以下ではJMPL命令だけを対象として考える。

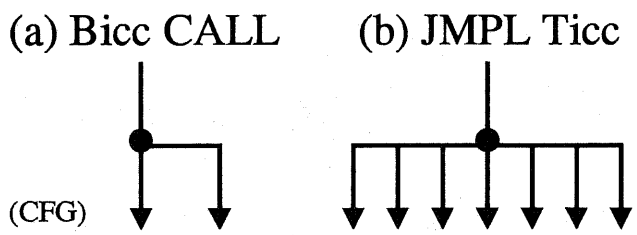


図 7.1: 分岐命令における制御の流れ

分岐命令	分岐条件	アドレス計算	分岐方向
Bicc	条件付き	PC 相対	図 7.1(a)
CALL	無条件	PC 相対	図 7.1(a)
JMPL	無条件	レジスタ間接	図 7.1(b)
Ticc	条件付き	レジスタ間接	図 7.1(b)

表 7.1: 主な分岐命令の特徴

7.2 Multi BTAC の提案

大規模なコントロールフローの展開を行なう場合、レジスタ間接アドレッシングの分岐命令をまたぐ命令展開において、分岐先を複数の可能性の中から正しく予測することが必要となってくる。そこで、これらの分岐命令に対する分岐予測機構として Multi BTAC (Multi Branch Target Address Cache) を提案する [中村 97]。通常 JMPL 命令のようなレジスタ間接アドレッシングの分岐命令の分岐先予測には BTAC と呼ばれるテーブルが用意され、これを用いて分岐先を予測することが多い。BTAC は図 7.2のように、JMPL 命令の PC 値などを使った TAG エリアとその分岐命令の前回の分岐先を保存しておく Target#1 エリアの 2 エリアを 1 エントリとして、128〜2048 エントリ程度がハードウェアに用意されていることが多い。しかし、BTAC を用いたのでは第 5.2節で述べたコントロールフロー先行展開が行なう図 6.1のような命令展開の際に、JMPL 命令をまたいだ命令展開が 1 方向だけに限定されてしまい、展開効率が悪化することが考えられる。そこで、図 7.2の下のような Target エリアが#1〜#n まで n 個あるようなテーブルをもつ Multi BTAC を用意することでこの問題を解決する。なお、Target エリアが n 個ある Multi BTAC を n-Level Multi BTAC と呼ぶことにする。

7.3 Multi BTAC の性能評価

7.3.1 レジスタ間接分岐命令に対する性能

本節では、第 7.2節で提案した Multi BTAC の性能について、レジスタ間接分岐命令に対する性能を評価する。

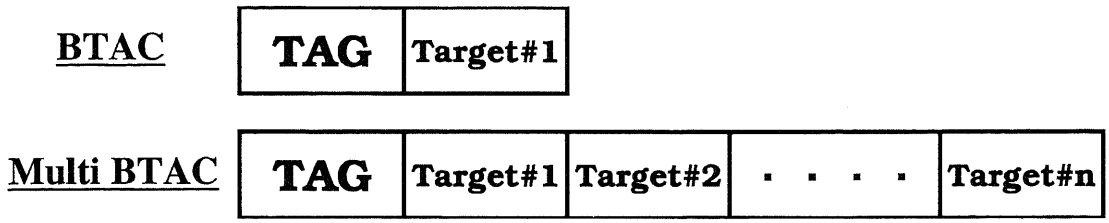


図 7.2: BTAC と Multi BTAC

図 7.3に示したシミュレーション結果は、Multi BTAC の Target エリアの個数によって分岐予測成功率がどのように変わるかを示したものである。ここでは JMPL 命令についてのみ Multi BTAC で予測した場合を示している。つまり横軸が n-Level Multi BTAC の n 、縦軸が分岐予測成功率を示している。ここでは、Multi BTAC の限界性能を得るために、十分大きなエントリ数を用意した。シミュレーション環境は表 7.2の通りである。 図 7.3によれば、Multi Level=1、つまり BTAC の場合

サンプル・プログラム	cc1, compress, dhrystone, espresso, sc (SPECint92)
命令トレース規模	1,000 万命令トレースを使用 (SPARC)
Multi BTAC 設定	10,240 エントリ エントリ連想度 フルアソシアティブ エントリ置換戦略 LRU

表 7.2: Multi BTAC の Multi Level 別性能シミュレーション環境

は平均で 76 %の予測成功率であるが、Multi Level=2 となるとこれが 85 %になることが分かる。つまり、コントロールフロー先行展開のような複数のパスを選択的に投機フェッチしていく命令展開機構では、Multi BTAC のような分岐先予測機構が有効であることが分かる。Multi Level = 3 では平均して 90 %以上の予測成功率を得ており、通常の BTAC(Multi Level = 1 の Multi BTAC) に比べて 15 %以上の性能向上が見られた。ハードウェアへの実装の可能性などを考慮して、以下では Multi Level = 1, 2, 3 の Multi BTAC について、エントリ数と分岐予測性能の関係について調査した。図 7.4は BTAC エントリ数の違いによる性能差を示したものである。各線は Multi Level = 1, 2, 3 のもので、Multi Level = 2 では平均で 86 %, 3 では 91 %程度の予測成功率が得られる。BTAC のエントリの連想度が

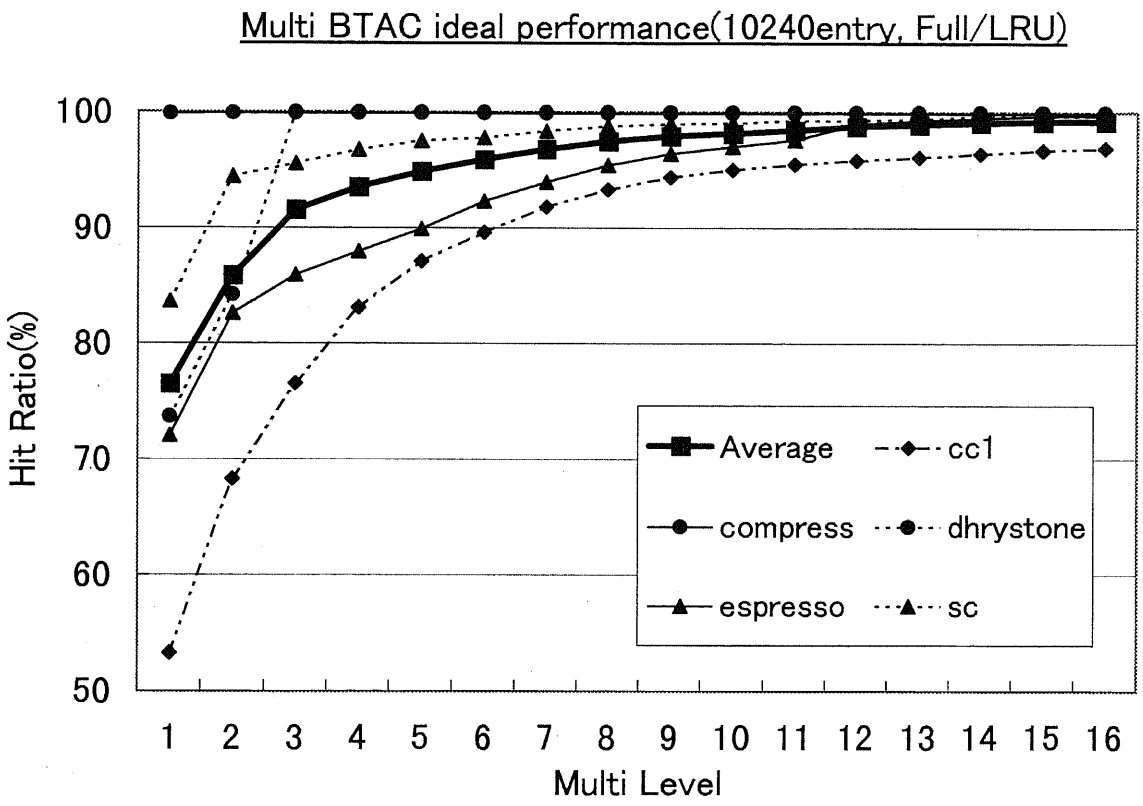


図 7.3: Multi BTAC の Multi Level 別性能

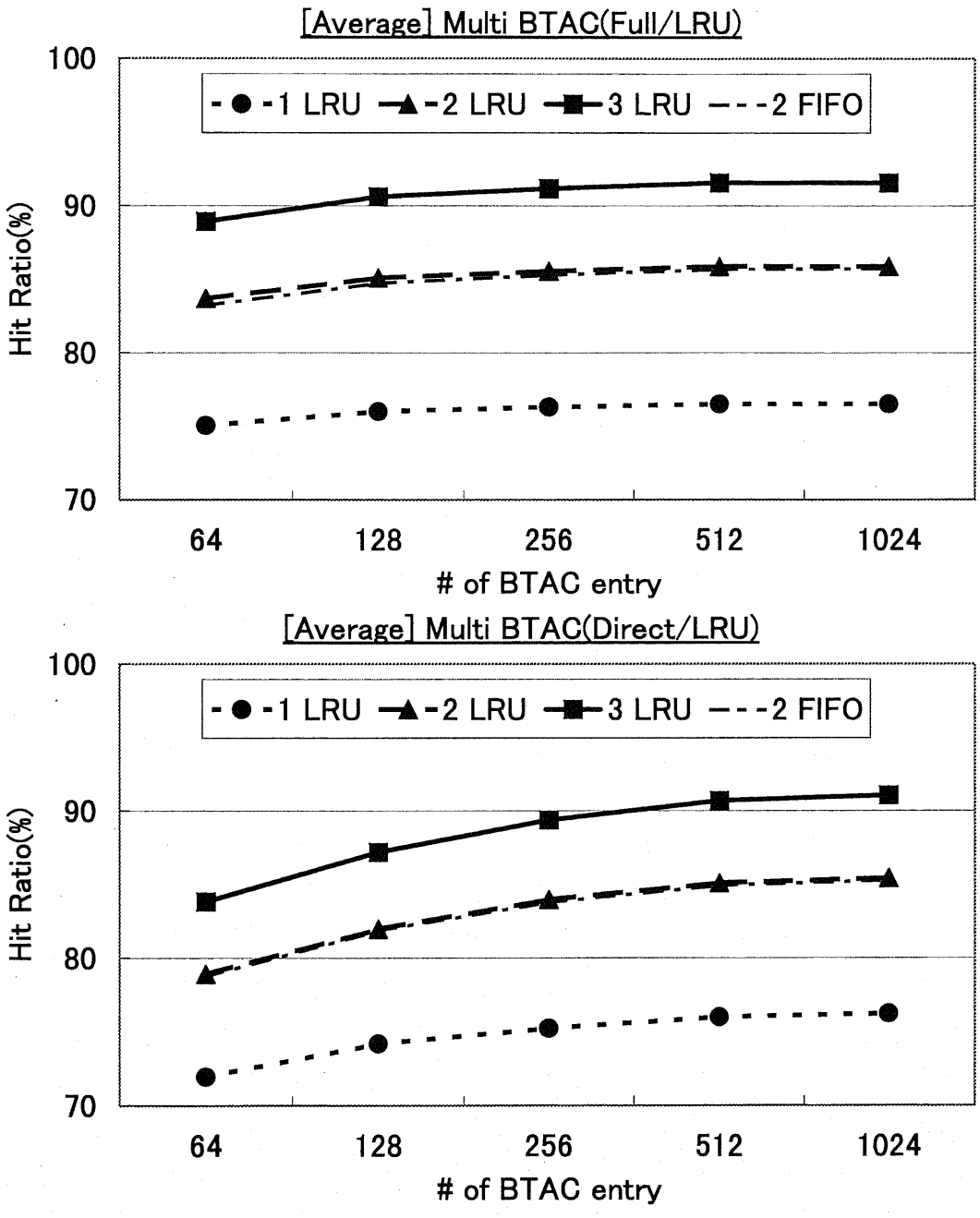


図 7.4: BTAC エントリ数と性能 (フルアソシアティブ／ダイレクトマップ)

フルアソシアティブ (図 7.4上) とダイレクトマップ (図 7.4下) のいずれにおいても BTAC エントリ数が 256 もしくは 512 エントリでほぼ性能向上が飽和する。また両者では BTAC エントリ数が少ないときには性能に若干の差があるが、ある程度のエントリ数が確保できる場合には大きな差はない。4way セットアソシアティブについても調査を行なったが、両者の中間的な性能であった。また、分岐先アドレスの置き換え方式についても LRU と FIFO で大きな差はみられなかった。

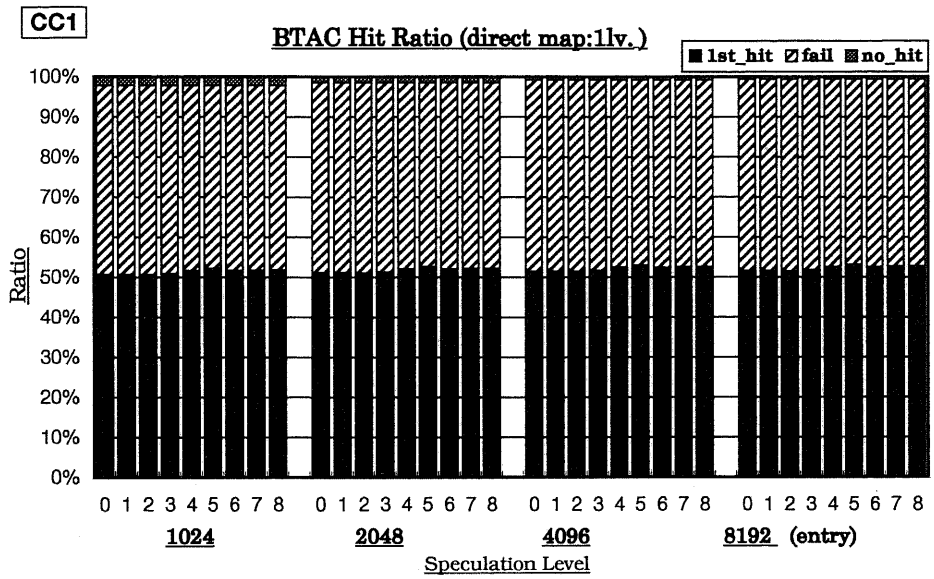
さらに、投機処理時における Multi BTAC の性能を調査した。図 7.5～図 7.10に示したのは、各サンプルプログラムに対する Multi BTAC の分岐予測成功率である。各グラフでは上図が通常の BTAC[1]であり、下図が 2-Level の Multi BTAC である。各図の横軸は投機処理の規模および Multi BTAC のエントリ数を示している。例えば、1024の上の 0 1 2 3 4 5 6 7 8 は、何段の分岐命令をまたいで投機処理を行なった場合かを示したもので、例えば 8 の場合には、分岐を 8 つまたいだ先のレジスタ間接分岐命令に対して分岐予測を行なった場合である。つまり、Multi BTAC の各エントリに格納されている分岐履歴情報は、現時点までに分岐結果の判明している部分までであるので、最大 8 つの間にある分岐命令の分岐結果が欠落している状態の Multi BTAC で分岐予測を行なったことになる。横軸が 0 の場合には、すべての分岐結果が反映されている最新の Multi BTAC で分岐予測を行なった状態である。縦軸は分岐予測結果の分布を示しており、1st_hit は 2-Level Multi BTAC の場合、第 1 エントリの予測結果が正しかった確率、2nd_hit は第 2 エントリの予測結果が正しかった確率である。なお、第 1 エントリは最も近い時点の分岐先アドレスを格納している。fail は Multi BTAC 中に該当する分岐命令のエントリは見つかったが、第 1、第 2 エントリともに正しい分岐先を格納していなかった確率である。no_hit は該当する分岐命令のエントリが見つからなかった確率である。シミュレーション環境は表 7.3の通りである。図 7.5～図 7.10を見ると、プログラムによって特性に大きな違いがあることが分かる。全体的には、BTAC の分岐予測成功率 (ヒット率) に比べて、2-Level Multi BTAC では 2 つのエントリの合計ヒット率が最大 30 %ほど大きくなっていることが分かる。またヒット率自体も、最も性能の低い cc1 でも 70 %程度になっていることが分かる。特徴的なデータ傾向を示しているのは、ijpeg である。ijpeg では、投機レベルが上がるほど分岐予測性能が上がる傾向を示している。この原因としては、最内周ループにおいて、2 つ以上の分岐先をもつレジスタ間接アドレッシングの分岐命令が存在し、その周期がうま

[1]分岐先のキャッシュエントリ数が 1 つのみ

サンプル・プログラム	cc1, compress, go, ijpeg, li, perl (SPECint95)
命令トレース規模	10 億命令トレースを使用 (SPARC)
Multi BTAC 設定	1,024～8,192 エントリ エントリ連想度 ダイレクトマップ エントリ置換戦略 FIFO

表 7.3: 投機処理時における Multi BTAC の性能シミュレーション環境

くタイミング的にあった場合に性能が上がっていると考えられる。



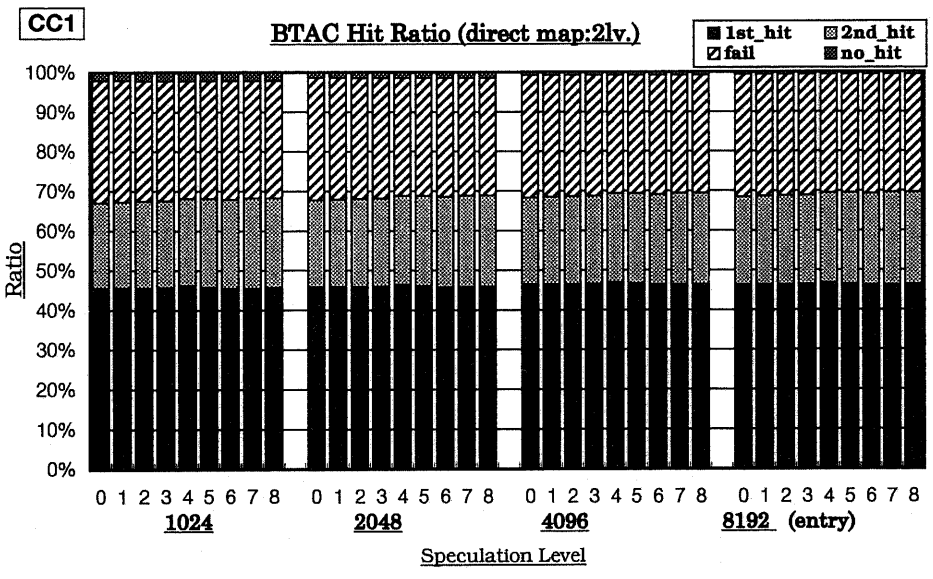
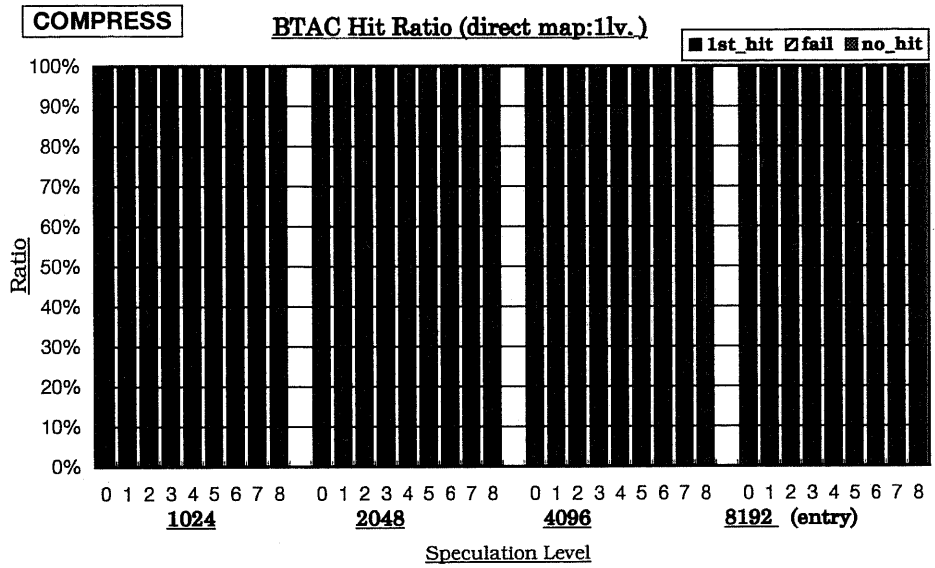


図 7.5: Multi BTAC の分岐予測成功率 (CC1)



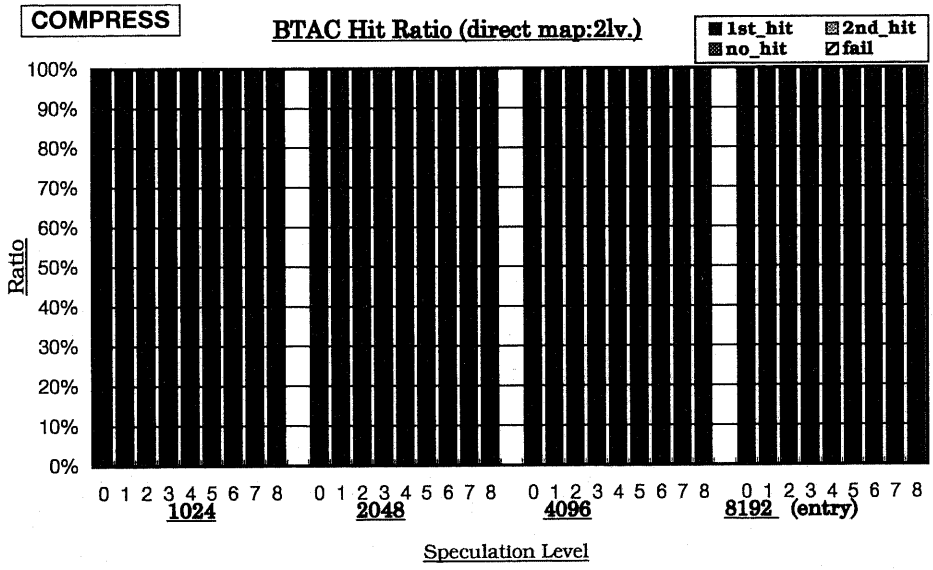
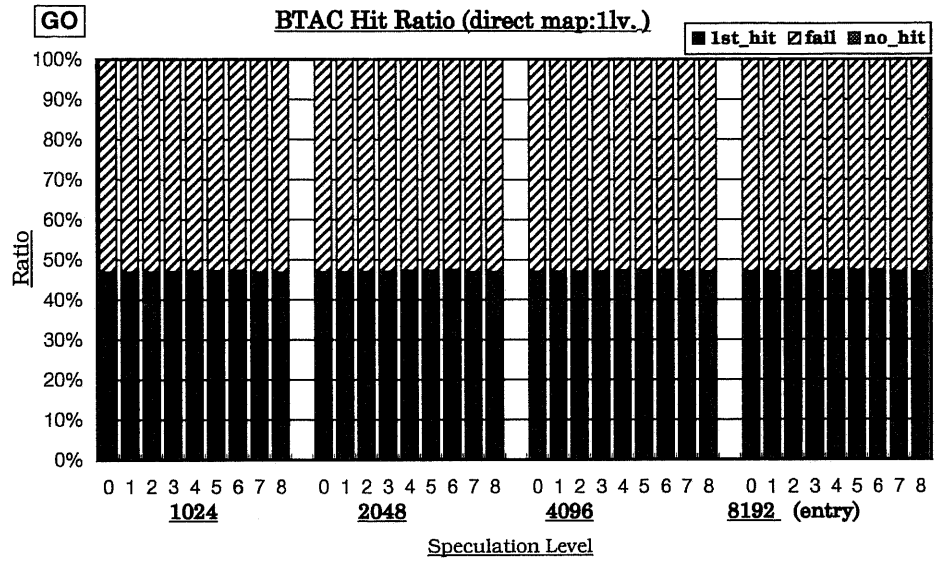


図 7.6: Multi BTAC の分岐予測成功率 (COMPRESS)



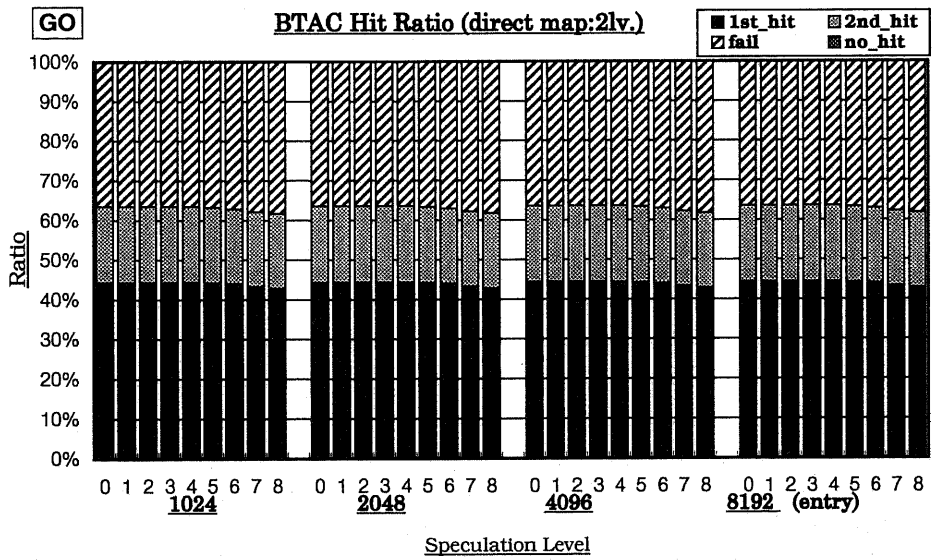
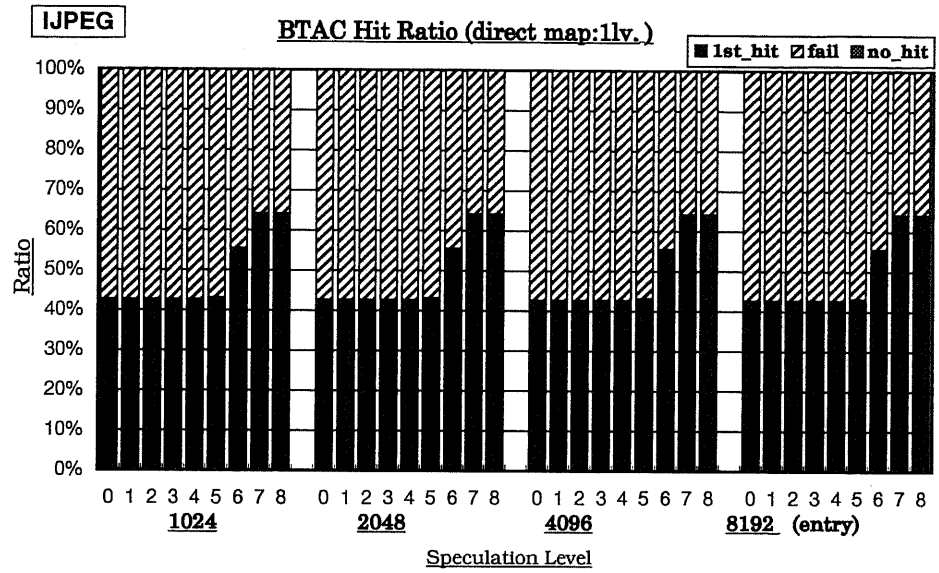


図 7.7: Multi BTAC の分岐予測成功率 (GO)



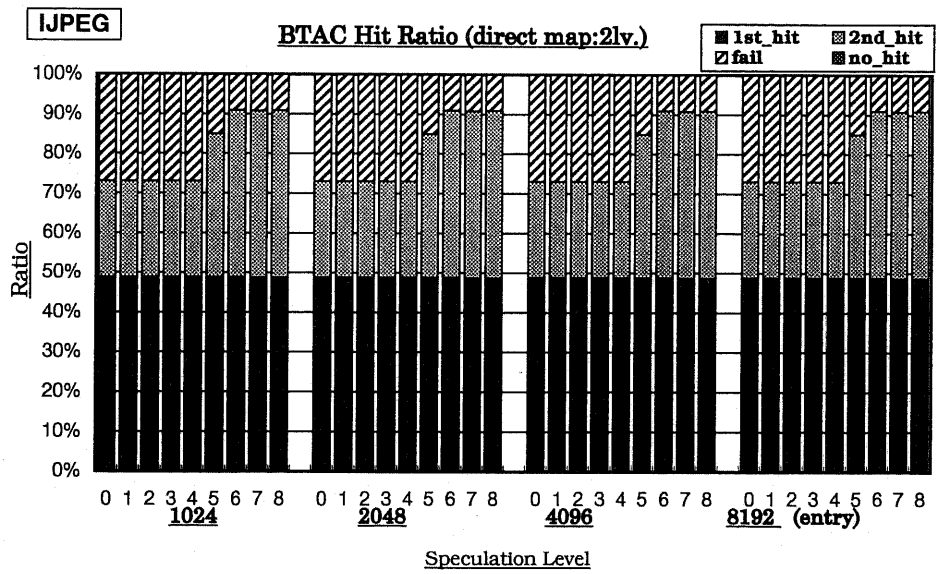
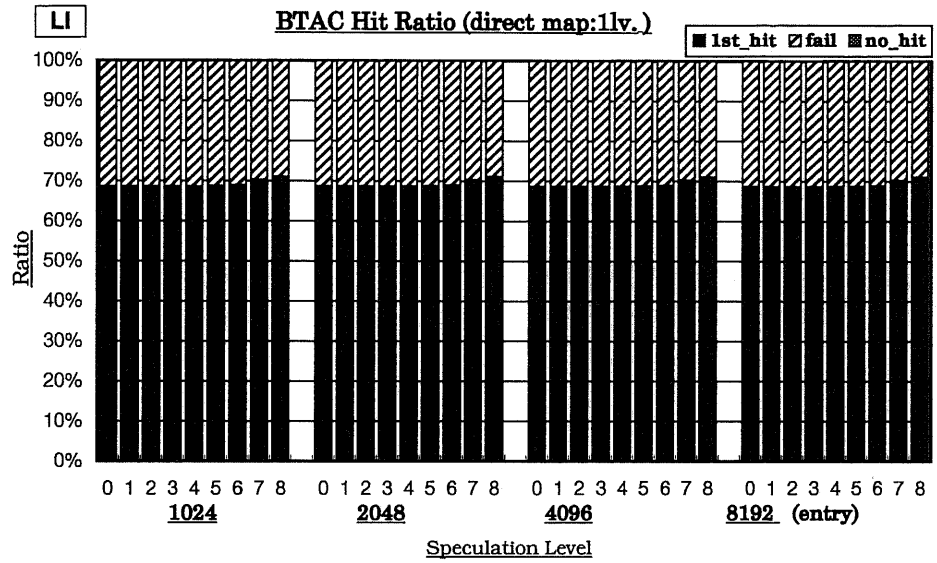


図 7.8: Multi BTAC の分岐予測成功率 (IJPEG)



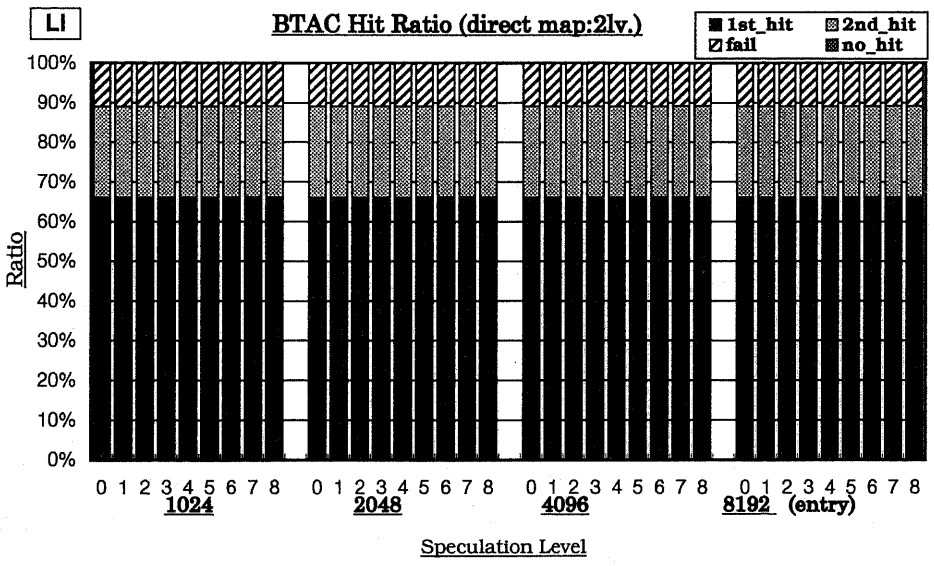
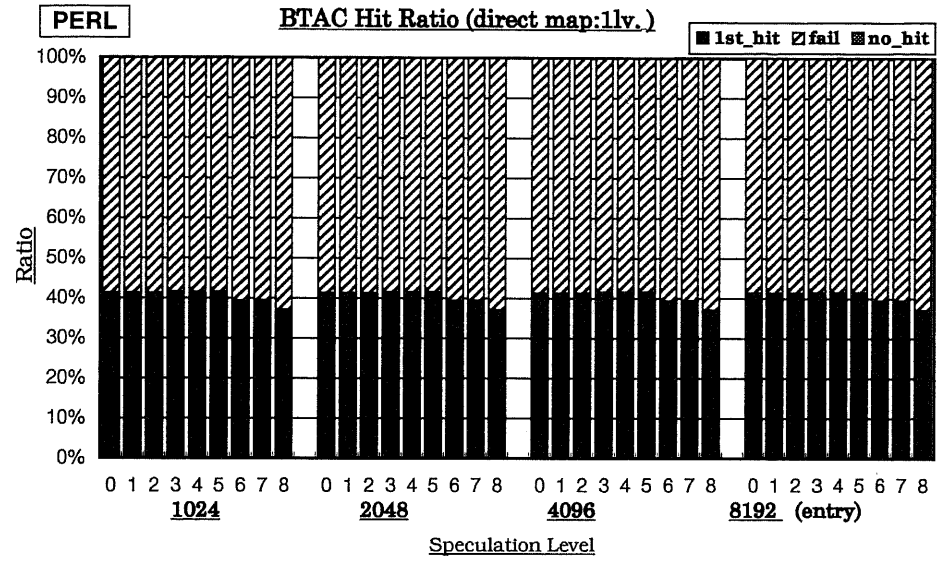


図 7.9: Multi BTAC の分岐予測成功率 (LI)



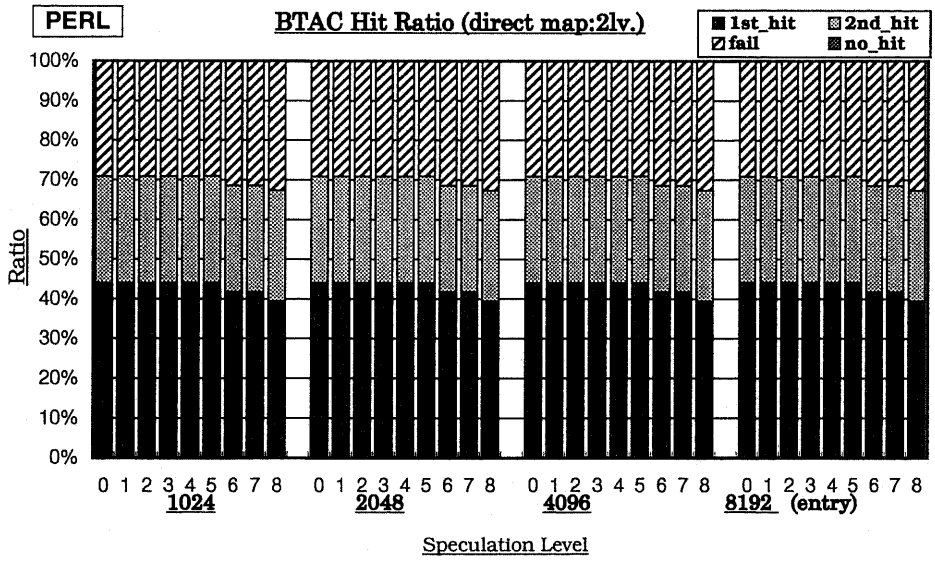


図 7.10: Multi BTAC の分岐予測成功率 (PERL)

第 8 章

コントロールフロー先行展開の提案

本章では大規模データパス・アーキテクチャにおけるフェッチ機構としてコントロールフロー先行展開機構を提案し、その詳細について述べる。図 8.1は、コントロールフロー先行展開における処理の流れを摸式的に示したものである。図中、実線の矢印は命令の移動を、網掛け矢印は制御情報の移動を表している。

8.1 コントロールフローパス管理

コントロールフロー先行展開の 1 つの役割は、フェッチされた命令のコントロールフローを保存しておくことである。その動作は次の通り。

1. フェッチされた命令に対してコントロールフロー・パスごとに“色付け”をし、パスごとの制御依存関係情報を登録する。

2. 制御依存関係が解消された命令流をデータパス先行展開に送る。この際に、各パスごとにフェッチ・発行を確認するための情報を登録しておく。

3. 実行が完了し分岐結果が判明したら、不要となったパスを制御依存関係情報から特定し、データパス先行展開や ALU-NET から不要となったパスを削除し、あわせて制御依存関係情報を更新する。

ここでポイントとなるのは、制御依存関係が解消された命令流をそれぞれデータパス先行展開に送ることにより、複数のコントロールフロー・パスに対して別々の資源を割り当てて[1]並列に投機実行することが可能となることである。

[1]例えば別々のリネームレジスタ・セット

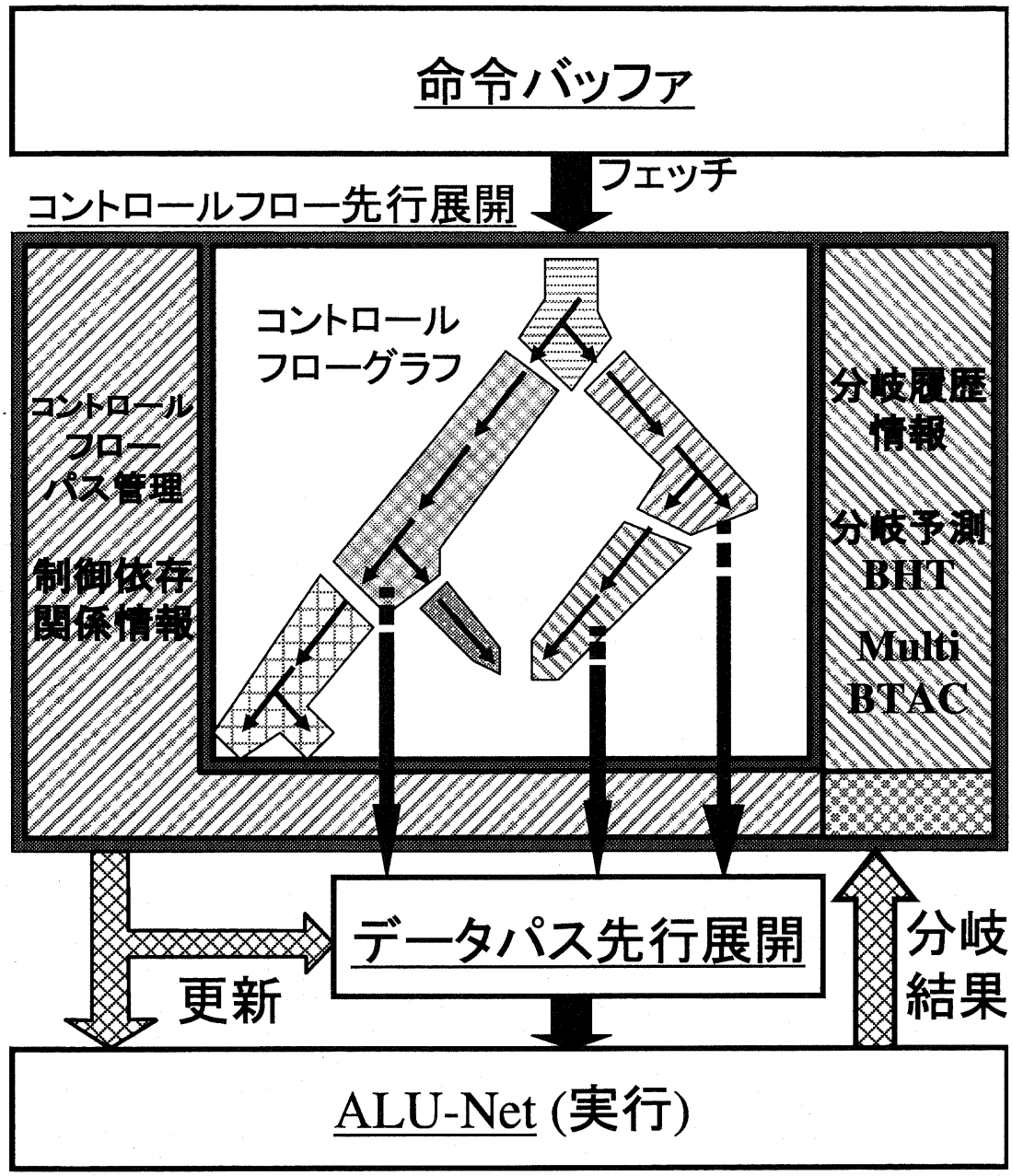


図 8.1: コントロールフロー先行展開の処理の流れ

8.2 実行予測確率管理

コントロールフロー先行展開では第6.1節で述べたように、選択的な複数パスへの投機フェッチを行なうことで、

- Single Path の投機フェッチに比べて分岐予測の影響を受けにくく、分岐予測失敗によるペナルティーが小さくでき、

- Eager Fetch の投機フェッチに比べて不必要な命令のフェッチによる影響 (投機規模の拡大が困難、命令フェッチ能力の不足) を受けにくく、かつ Eager Fetch の利点である分岐予測性能に影響を受けないフェッチ機構に近い性能を得ることを目標とする。

このような選択的な複数パスへの投機フェッチにおいては、どのパスを次にフェッチすべきかを正しく予測することが非常に重要である。図5.3に示した命令展開において、xxx と表示した各パスへの実行予測確率情報を厳密に計算して、正確に実行予測確率の高いパスを選択することが理想かもしれないが、実際にこのような計算を行なうとすると、確率値の伝搬・再計算が命令展開処理のボトルネックとなりかねない。

つまり、実行予測確率の管理・計算機構には、

- 処理的に軽い計算量で近似的に確率値を求められ、

- 分岐結果を得てコントロールフロー・ツリーの更新の際に簡単な処理で更新が行なえる

ことが求められる。

ここでは、コントロールフロー先行展開の実行確率管理方式として次の 2 方式を比較検討する。

Bit Order (BO) 方式 コントロールパス上の分岐命令がもつ分岐履歴ビットを結合したものをそのコントロールパスの実行確率とする方式

Bit Cross (BC) 方式 コントロールパス上の分岐命令がもつ分岐履歴ビットをビットごとに結合したものをそのコントロールパスの実行確率とする方式

8.2.1 Bit Order 方式

まず、BO 方式について説明する。BO 方式は図 8.2のような実行確率管理方式である。図 8.2の例では、各分岐命令においてそれぞれのパスへの分岐確率は 2bit の分岐履歴ビットで表現されている。図 8.2の上の図は例として取り上げるコント

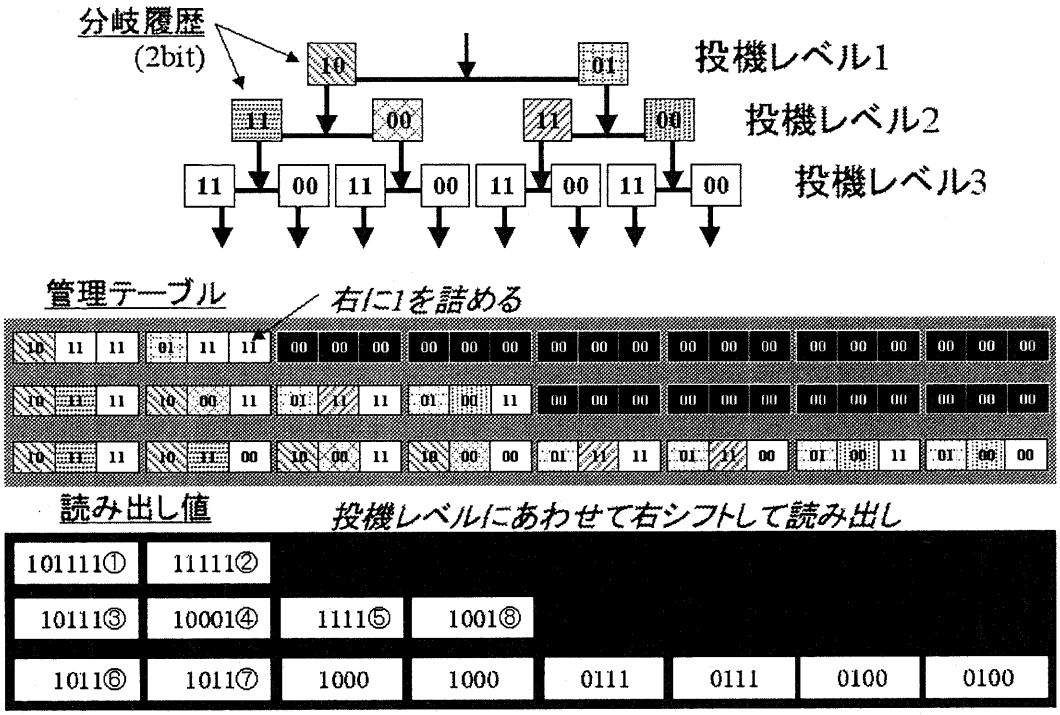


図 8.2: Bit Order 方式の実行確率管理

ロールフロー・ツリーである。ここでは 3 段の投機フェッチであり、上から投機レベル 1, 2, 3 と呼ぶことにする。中の図は、このコントロールフローに対する実行予測管理テーブルの状態で、1 行目が投機レベル 1, 2 行目が投機レベル 2, 3 行目が投機レベル 3 に対応しており、ここでは 6bit を 1 つの単位として各分岐パスへの実行予測確率を管理している。つまり、管理テーブルの左上のビット列 101111 は上の図の投機レベル 1 における左側のパスの実行予測確率を管理するビット列である。管理テーブルの作成方法は次の通りである。

1. 各パスの投機レベルの最も若い分岐履歴ビットを最上位に登録する

2. 次の投機レベルの分岐履歴ビットを使って順に下位ビットを埋めていく

3. 下位の投機レベルの分岐がなくなったら1を詰める

ここで、下位の投機レベルの分岐がなくなったら1を詰めるのは、投機レベルの小さい、つまりより実行完了ポイントに近い部分のパスは、そのパスがフェッチされていない状態で実際の実行がそのパスに移った場合、まずそのパスへのフェッチから処理を再開することになり、ペナルティーが大きいため、そのパスへの実行予測確率はより高い確率値に設定すべきだからである。

図8.2の下の図は管理テーブルを使って実際に実行予測確率を求めたものである。この図中丸囲みの数字は、優先順位を表している。この読み出しは次のように行なわれる。

1. 管理テーブルに登録されている各パスへの実行確率管理ビットを投機レベルにあわせて右シフトして読み出す

2. 未フェッチのパスの中で最も上位に1が立っているパスを次のフェッチ候補とする

ここで投機レベルにあわせて右シフトしてデータを読み出すのは、管理テーブルの作成において、下位の投機レベルの分岐がなくなったら1を詰めるのと同じ理由である。

次に1つの分岐命令の結果が判明し、コントロールフロー・ツリーの更新を行なう方法について述べる。図8.3は更新の方法を摸式的に示したものである。図の左側はコントロールフロー・ツリーの状態、右側はそのときの管理テーブルの状態を示したものである。ここで、図の上から下の状態にコントロールフロー・ツリーを更新することを例に実行確率管理テーブルの更新方法を説明する。管理テーブルには各行ごとに1つのカレント・ポインタ (スタート・ポインタ) が用意されている。また縦方向に別の1つのカレント・ポインタ (レベル・ポインタ) が用意されている。このポインタを移動させる簡単な操作でテーブルの更新が可能になっている。具体的な更新方法は次の通りである。

1. レベル・ポインタを1つ下に移動する (すでに最も下の場合は一番上に移動する)

2. コントロールフローツリーが右方向に更新された場合には、スタート・ポインタを投機レベルにあわせて移動する (投機レベルがNであった行は2^{N-1}だけ右に移動する, ただし投機レベル1であった行はスタート・ポインタを左端に移動する)

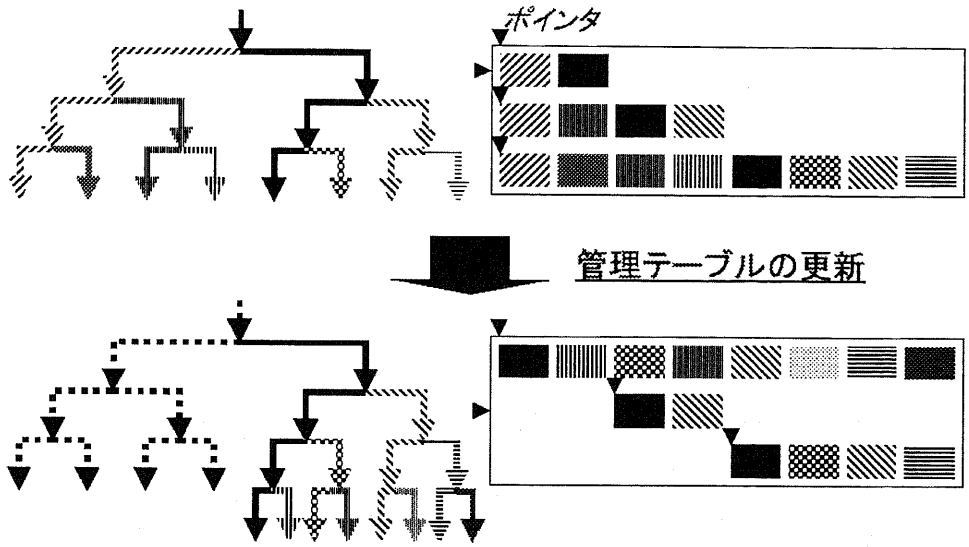


図 8.3: コントロールフロー先行展開のテーブル更新方法

3. 各エントリのビット列を左にシフトする (分岐履歴ビットが 2bit の場合には 2bit シフトになる)

8.2.2 Bit Cross 方式

BC 方式は図 8.4のような実行確率管理方式である。図 8.4の例では、各分岐命令においてそれぞれのパスへの分岐確率は 2bit の分岐履歴ビットで表現されている。図 8.4の見方は図 8.2と同様である。BC 方式と BO 方式の相違点は、管理テーブルの作成方法にある。

1. 各パスの投機レベルの最も若い方から順番に分岐履歴ビットの上位ビットをとりだし登録する

2. 投機レベルが小さい場合には最大投機レベルになるまで 1 を登録する

3. 最大投機レベルに達したら分岐履歴ビットの次のビットを順に登録していく

4. 分岐履歴ビット数分だけこれを繰り返す

この方法を簡単に示したのが図 8.5である。

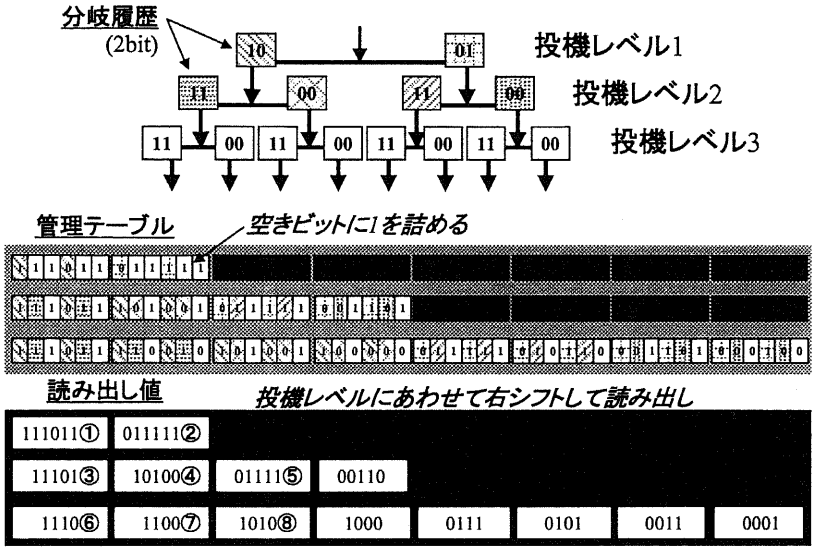


図 8.4: Bit Cross 方式の実行確率管理

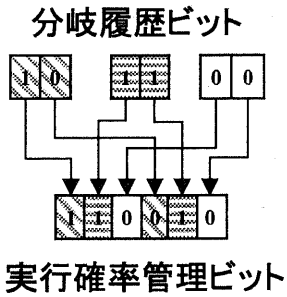


図 8.5: Bit Cross 方式の実行確率管理ビット登録方法

図8.4の下の図は管理テーブルを使って実際に実行予測確率を求めたものである。この図中丸囲みの数字は、優先順位を表している。この読み出し方法はBO方式と同じである。また、コントロールフロー・ツリーの更新を行なう方法についてもBO方式と同じく、図8.3に示した通りである。ただし各エントリのビット列は単純な左シフトではなく、実行確率管理ビットを分岐履歴ビット数に分割しそのそれぞれの組の中で左に1bitだけシフトすることになる。この様子を図示したのが図8.6である。

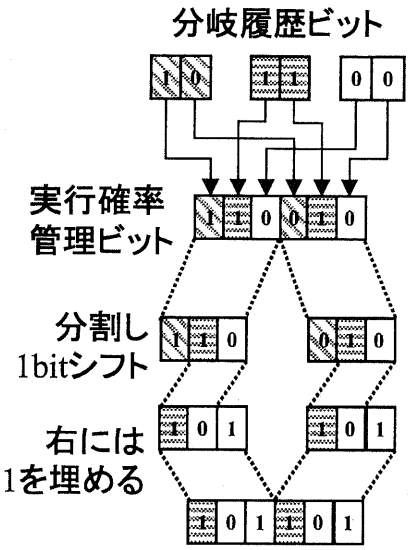


図 8.6: Bit Cross 方式の実行確率管理ビット更新方法

以上のような簡単な処理で各パスの実行予測確率を管理し、これを利用して複数パスへの選択的投機フェッチを行なうのがコントロールフロー先行展開機構である。

第 9 章

コントロールフロー先行展開の評価

本章では、BO 方式および BC 方式の実行確率管理機構をもったコントロールフロー先行展開機構に関する性能評価を行なう。コントロールフロー先行展開機構の構成には多くのパラメータがあり、これらのパラメータの違いによる性能の違いなどについてシミュレーションを行なった。

9.1 評価環境

シミュレーションには SPARC V.7 のコードを利用し、作成した SPARC シミュレータによるトレース駆動のシミュレーターを利用した。サンプルプログラムとしては SPECint95 から cc1, compress, go, ijpeg, li, perl の 6 つのプログラムを利用した。ただし、本論における評価では cc1 を主に利用する。これは cc1 が最もプログラムサイズ的に大きく、プログラムの実行特性に偏りが少ないと思われるためである。また compress については、シミュレーションを行なったが、分岐命令の特性に偏りがあり、他のプログラムによる性能評価と大きく特性が異なるため、対象から外し、cc1, go, ijpeg, li, perl の 5 つのプログラムによる評価とした。go, ijpeg, li, perl のデータについては、一部を Appendix A に載せた。

シミュレーションに利用した命令トレース規模は 1 億命令である。これは主にシミュレーション時間による制約である。一部のシミュレーションにおいてプログラム終了まで実行をした場合のトレースを利用してシミュレーションを行なったが、1 億命令トレースを利用したシミュレーション結果との違いは少なかった。

コントロールフロー先行展開機構に内蔵する分岐予測機構は、第 6 章と第 7 章において行なった複数パスへの大規模投機フェッチに適した分岐予測機構の検討を

元に、Taken or Not-Taken 型の分岐命令に対して 8192 エントリの 1-Level Multi BTAC、レジスタ間接アドレッシングの分岐命令に対して 4096 エントリの 3-Level Multi BTAC を仮定した。いずれの Multi BTAC に関しても、連想度は 4way セットアソシアティブ、エントリの置換え戦略については FIFO を用い、各エントリには数ビットの分岐履歴ビットを設けた。この分岐履歴ビット数はパラメータであり、2〜5bit の場合の評価を行なった。また、1 つの分岐ポイントからの命令の展開方向は 2 方向に限定した。

評価値としては、命令フェッチ量、分岐予測成功率、フェッチされた命令の中に正しいパスがなく、フェッチ済みの命令を削除 (Flush) し、フェッチをやり直した割合 (Flush 率) などを使った。また最終的にフェッチされた命令から得られる最大並列度についても評価を行なった。得られる並列度の計算においては、複数パスへの大規模投機実行を行なう実行部にさまざまな方式が考えられ、データ転送などのパラメータも複雑にからみあうため、本論文では理想的な実行部 (オラクル・モデル) を仮定した場合の並列度を示すこととした。ここで理想的な実行部とは、資源競合が起こらないだけの多重化されたレジスタウィンドウ、キャッシュ・ポート、無限の ALU およびフォワーディングパス、データバスなどをもつものである。なおキャッシュヒット率は命令・データともに 100 %を仮定し、アクセスタイムは 1 サイクルとした。

表 9.1はシミュレーション環境をまとめたものである。

サンプル・プログラム	cc1, go, ijpeg, li, perl (SPECint95)
命令トレース規模	1 億命令トレースを使用 (SPARC)
分岐予測機構 (レジスタ間接アドレッシング用) (即値アドレッシング用)	非更新型 Multi BTAC 4way セットアソシアティブ, FIFO, 4,096 エントリ, 3-Level Multi BTAC 4way セットアソシアティブ, FIFO, 8,192 エントリ, 1-Level Multi BTAC
命令展開幅	2 方向
実行部	オラクル・モデル

表 9.1: コントロールフロー先行展開機構のシミュレーション環境

9.2 投機レベルに関する性能評価

まずコントロールフロー先行展開の性能と投機レベルの関係を示す。以下の評価では、投機レベルは最小 3、最大 8 とする。これは投機レベルが小さすぎると Single Path や Eager Fetch と同様の命令展開となることが多く、選択的な複数パスへの投機フェッチに関する評価にならないためである。また最大の投機レベルを 8 にしたのは、これ以上の投機レベルのシミュレーションは計算量が莫大となり、現実的な時間でシミュレートするのが困難であるからである。また以下に示すように、投機レベルが大きくなると、得られる性能向上量が減少してくるため、投機レベル 8 程度までを評価するのが妥当である。また、cc1 の平均ベーシックブロックサイズは 4〜5 命令程度であり、投機レベルが 8 段になるとシングルパスであっても管理する命令量が 40〜50 命令となり、Eager Fetch ではこれが 2500 命令程度にまでなってしまう。よって、Single Path と Eager Fetch の差が十分にあるといえるので、選択的な複数パスへの投機フェッチの評価としては、十分な状況である。

本節で用いたシミュレーションパラメータは表 9.2の通りである。

サンプル・プログラム	cc1
分岐履歴ビット数	3bit
実行確率管理ビットの読み出し時シフト量	1bit
実行確率管理ビット更新方式	Bit Order
評価するパラメータ	投機レベル (3, 4, 5, 6, 7, 8)

表 9.2: 投機レベルに関する性能評価・シミュレーションパラメータ

図 9.1は投機レベルと得られる平均並列度の関係を示した図である。投機レベルが大きくなると、それに応じて平均並列度も向上しているが、このシミュレーションパラメータではその向上率は投機レベル 3 と 8 で 30 %程度である。ただし、投機レベル 3 から 5 にかけては 2 レベルの増加で 22 %の性能向上が得られている。

図 9.2は投機レベルと分岐予測成功率の関係を示した図である。本論文での分岐予測成功率は、通常のシングルパスへの命令展開時の分岐予測成功率とは若干意味合いが異なる。シングルパスの場合には、分岐予測によって展開すべきパスが決定されるが、コントロールフロー先行展開のような複数パスへの命令展開を行なう可能性がある場合には、分岐予測が成功したというのは、「最も遷移確率が高い

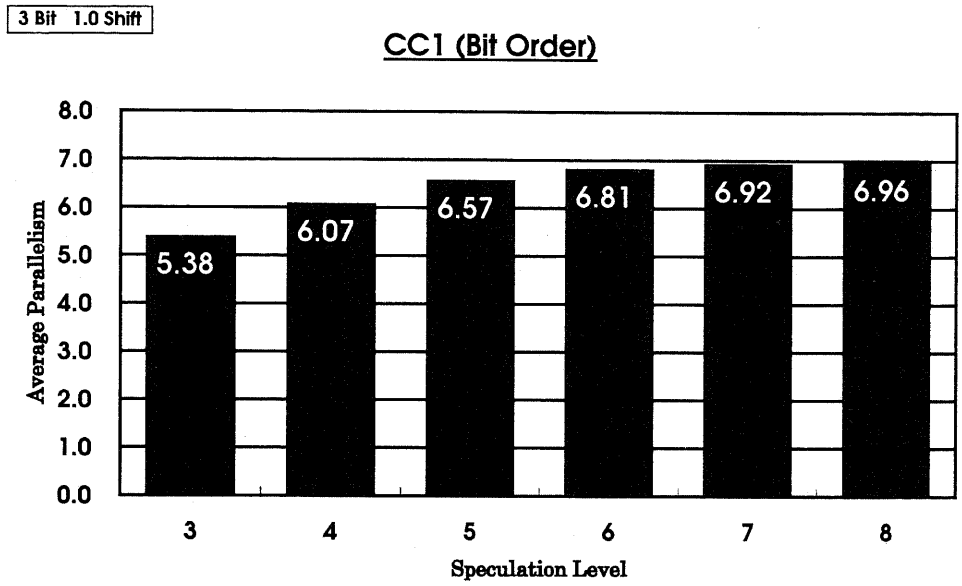


図 9.1: 投機レベルと平均並列度 (cc1,3bit 履歴,1bit シフト,BO 方式)

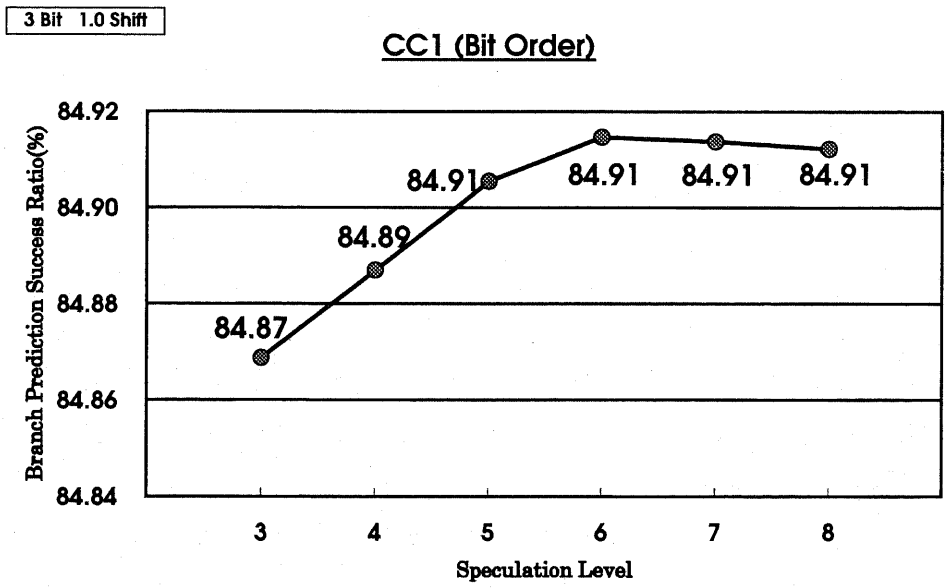


図 9.2: 投機レベルと分岐予測成功率 (cc1,3bit 履歴,1bit シフト,BO 方式)

と予測したパスへ実際に制御が分岐した」ということを意味する。よって、分岐予測が外れた場合でも、フェッチされている命令すべてがフラッシュされるとは限らない。これに関しては、図9.5の説明でもう少し説明する。当然であるが、コントロールフロー先行展開やEager Fetchのような複数パスへの命令展開を行なう方式の利点はこのようにフラッシュされる可能性が低いもしくはないという点にある。

図9.2で示された分岐予測成功率は、85 %弱であり一般的に見て必ずしも高いとは言えない。しかしながら、ここで示した分岐予測成功率はすべての分岐命令に対する予測成功率であり、SPARCのJMPL命令のようなレジスタ間接アドレッシングで分岐先が決まるような分岐命令も含まれていることを考えると、決して低い値ではない。

図9.2の全体的な傾向としては、投機レベルに応じて分岐予測成功率もあがっているように見えるが、縦軸のスケールが非常に小さく、分岐予測成功率が一番低い投機レベル3と一番高い投機レベル6での差は0.04 %しかない。よって投機レベルによる分岐予測性能への影響はないと考えてよい。これは第6章や第7章での考察結果とも一致している。

図9.3および図9.4は投機レベルとフェッチ命令数の関係を2通りに表現した図である。図9.3はフェッチした全命令を100 %としたときに、その中で正しいパスであった命令の割合を示したものである。投機レベル3では38 %、投機レベル5では17 %、投機レベル8では9 %というように、投機レベルが上がると正しいパスの命令の比率が下がっていることが分かる。図9.4はフェッチした命令の絶対数でグラフをかいたものである。黒い部分が正しいパスの命令(1億命令)であり、灰色の部分が不要なパスの命令である。投機レベルが7, 8と大きくなると10億以上の命令をフェッチすることになる。大規模な投機フェッチにおいては、フェッチ命令数をいかに押えて性能を稼ぐかが重要であるが、フェッチ命令数を押える、つまりフェッチするパスをうまく選択するという点に関してついて言えば、コントロールフロー先行展開がある程度働いていることが分かる。なぜならば、Eager Fetchであれば、投機レベルの増加にともなって、指数関数的にフェッチ命令数が増えるはずであるが、図9.4のフェッチ命令数の増加は1次関数的である。それだけパスの選択をうまく行なっているということが読みとれる。

図9.5は、投機レベルとフラッシュ率の関係を示したものである。フラッシュとは、フェッチされている命令がすべて無効化されることであり、これは次のような場合に起こる。

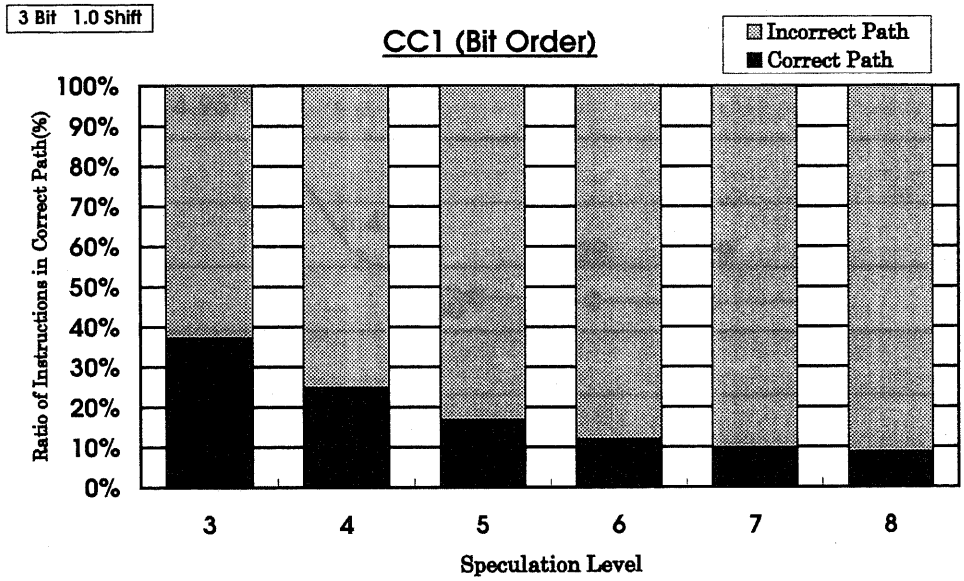


図 9.3: 投機レベルと有効命令率 (cc1,3bit 履歴,1bit シフト,BO 方式)

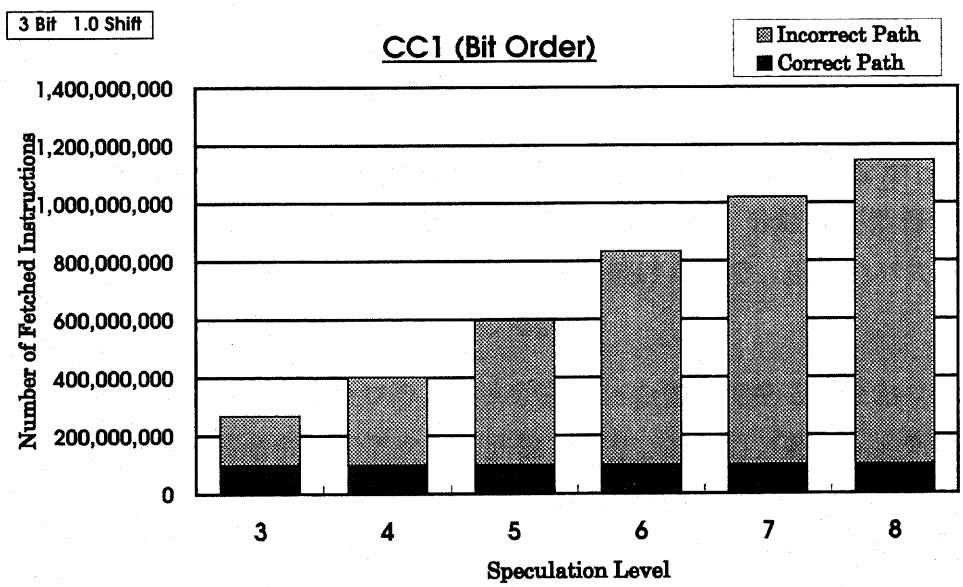


図 9.4: 投機レベルとフェッチ命令数 (cc1,3bit 履歴,1bit シフト,BO 方式)

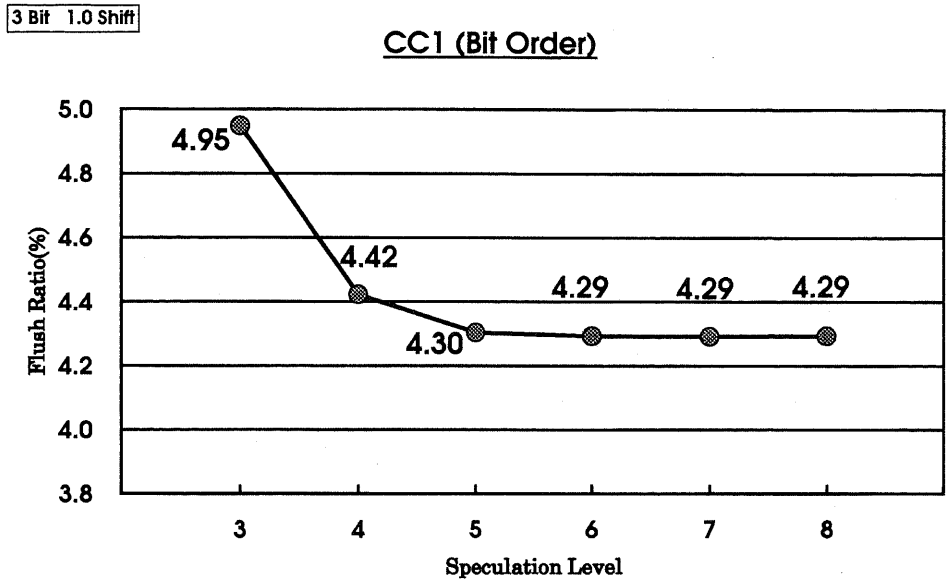


図 9.5: 投機レベルとフラッシュ率 (cc1,3bit 履歴,1bit シフト,BO 方式)

- 次の分岐命令の履歴に偏りがあり、ある一方向にのみ命令が展開されていたが、例外的にもう一方のパスに制御が移った場合。
- 次の分岐命令がレジスタ間接アドレッシングで分岐先が決まる分岐命令で、Multi BTAC の予測に基づいて 1 もしくは 2 方向に展開されていたが、そのいずれとも違う方向に制御が移った場合。
- 次の分岐命令に関する分岐履歴情報がなく、分岐予測ができなかった場合。ただし、即値アドレッシングの分岐命令の場合には、分岐履歴情報がなくとも通常、分岐が Not Taken としてフェッチを続けるので、正しいパスが Taken 側であった場合にフラッシュが起こる。

フラッシュ率とは、このようなフラッシュが起こった割合である。なおフラッシュ率の計算の分母は分岐命令へのアクセス回数であり、総実行／総フェッチ命令数ではない。よってフラッシュ率が 10 %というのは、分岐命令の 10 個に 1 つの割合でフラッシュが起こったことを意味する。図 9.5によると、投機レベルが上がるとフラッシュ率が下がっている。これは、図 9.5の例が、実行確率管理ビットの読みだしにおいて 1bit シフトして読み出すことが影響している。つまり、投機レベルが大きいパスの実行確率管理ビットは右にシフトされ読み出されるので、相対的に

投機レベルが小さいパスの実行確率が大きくなるため、投機レベルが小さい部分の分岐ポイントではEager Fetchが行なわれる可能性が高いためである。

図9.5では、投機レベルが6を越えるとフラッシュ率が4.29％で一定になっているが、この確率は上で述べたフラッシュが起こる場合の例からも分かるように、分岐履歴情報が不足している確率と複数パスへ分岐する分岐命令の予測に失敗をした確率が主な内訳である。

図9.2で示した分岐予測確率とこのフラッシュ率の合計が100％にならないのは、第2候補の分岐パスがフェッチされており、第1候補の分岐先は外れたものの、第2候補の分岐先に制御が移動した場合がどちらの確率値にも入っていないからである。分岐予測成功率が約85％、フラッシュ率が約4.3％であるので、10％強の確率でこのようなことが起こっていることが分かる。

9.3 分岐履歴ビット数に関する性能評価

分岐履歴ビット数とコントロールフロー先行展開の性能の関係について述べる。分岐履歴ビットは最近の分岐方向を記録するビット列である。図8.2や図8.4のように分岐履歴ビットを用いて各コントロールフローパスの実行確率管理ビットを作成し、確率の高いパスを選択的にフェッチしているので、分岐履歴ビットを何ビットでどのように表現するかは性能に影響を与えるパラメータである。

本節では、分岐履歴ビット数を2bit, 3bit, 5bitの3通りに設定した場合の性能比較についてまとめる。なお、分岐履歴ビットの登録・更新方法および実行確率管理ビットを作成する場合の読み出し方法は次のようになっている。

分岐履歴ビットの登録・更新方法 2bit, 3bit, 5bitのいずれもFIFOで登録する。つまり分岐成立の場合を1と表現し、2回連続して分岐成立の場合は、11(2bitの場合)と登録される。次に分岐不成立となると01(2bitの場合)となる。なお、レジスタ間接アドレッシングの分岐命令に対する分岐履歴ビットは、3-Level Multi BTACで管理されており、3-Level Multi BTACが保持する分岐先キャッシュ#1, #2, #3のそれぞれに分岐履歴ビットがある。よって最近の分岐先履歴が#1, #1, #2, #1, #3であった場合、分岐履歴ビットが5bitであるとすると、分岐先キャッシュ#1の分岐履歴ビットは11010, #2は00100, #3は00001となる。次に分岐先が#3になると、それぞれ01101, 00010, 10000に更新される。

分岐履歴ビットの読み出し方法 実行確率管理ビットを作成する場合の読み出し方法は、分岐履歴ビットのビット数によって異なる。各ビット数ごとの読み出し方法は表9.3の通りである。

分岐履歴ビット数	
2bit	分岐履歴ビットをそのまま読み出す
3bit	111 → 111,110 → 110,101 → 101,100 → 011 011 → 100,010 → 010,001 → 001,000 → 000 と変換する
5bit	5bit のうちの 1 が立っている数 (count) に応じた値を返す count=5 → 11111, count=4 → 11001, count=3 → 10011 count=2 → 01100, count=1 → 00110, count=0 → 00000

表 9.3: 分岐履歴ビットの読み出し方法

このように読み出し時にビット列の変換を行なうのは、より確からしい実行確率を得るためである。例えば、5bit の分岐履歴ビットが 10000 の状態の場合、分岐履歴としては 1 回だけ分岐が成立しただけにもかかわらず、このまま実行確率管理ビットの作成にこの分岐履歴ビットを使うと、最上位ビットが 1 になっているため、優先度が高いパスとして登録されてしまう。そこで実際の実行確率の予測値に近い 20 %程度 (5 回に 1 度であるため) を 5bit で表現するように 00110 に変換する。これは 00110 / 11111 = 0.19 である。また、この変換においては、例えば分岐履歴ビット数が 5bit の場合、count が (5,0), (4,1), (3,2) の組で全ビットが 1 になるようになっている。この対称関係は分岐履歴ビット数が 3bit の場合も同様である。

本節で用いたシミュレーションパラメータは表9.4の通りである。なお、実行確率管理ビットの読み出し時シフト量の 0.5bit とは、(int)((投機レベル-1) × 0.5) だけ右シフトして実行確率管理ビットを読み出すことを意味している。つまり、投機レベルが 1,2 の場合はシフトせず、投機レベル 3,4 では 1bit シフト、投機レベル 5,6 では 2bit シフトなどとなる。また、以下の評価では投機レベル 5 と 8 の 2 つについてデータを示すこととする。これはシミュレート量の問題もあるが、データの見やすさを考慮し、中規模投機フェッチとして投機レベル 5, 大規模投機フェッチとして投機レベル 8 を選択した。

サンプル・プログラム	cc1
投機レベル	5, 8
実行確率管理ビットの読み出し時シフト量	0.5bit
実行確率管理ビット更新方式	Bit Order および Bit Cross
評価するパラメータ	分岐履歴ビット数 (2, 3, 5)

表 9.4: 分岐履歴ビット数に関する性能評価・シミュレーションパラメータ

図 9.6, 図 9.7は分岐履歴ビット数と平均並列度の関係を示した図である。図 9.6 が Bit Order (BO) 方式、図 9.7が Bit Cross (BC) 方式である。それぞれ白い棒が分岐履歴ビット数 2bit, 薄い灰色の棒が 3bit, 濃い灰色の棒が 5bit の場合である。いずれの方式・投機レベルの場合においても分岐履歴ビット数が多くなると得られる平均並列度が大きくなっている。

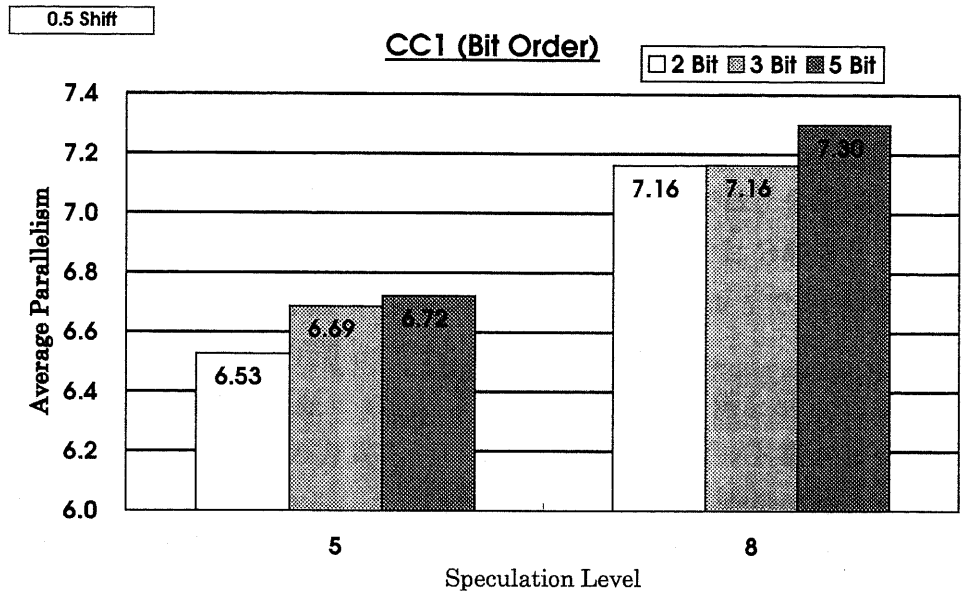


図 9.6: 分岐履歴ビット数と平均並列度 (cc1,2,3,5bit 履歴,0.5bit シフト,BO 方式)

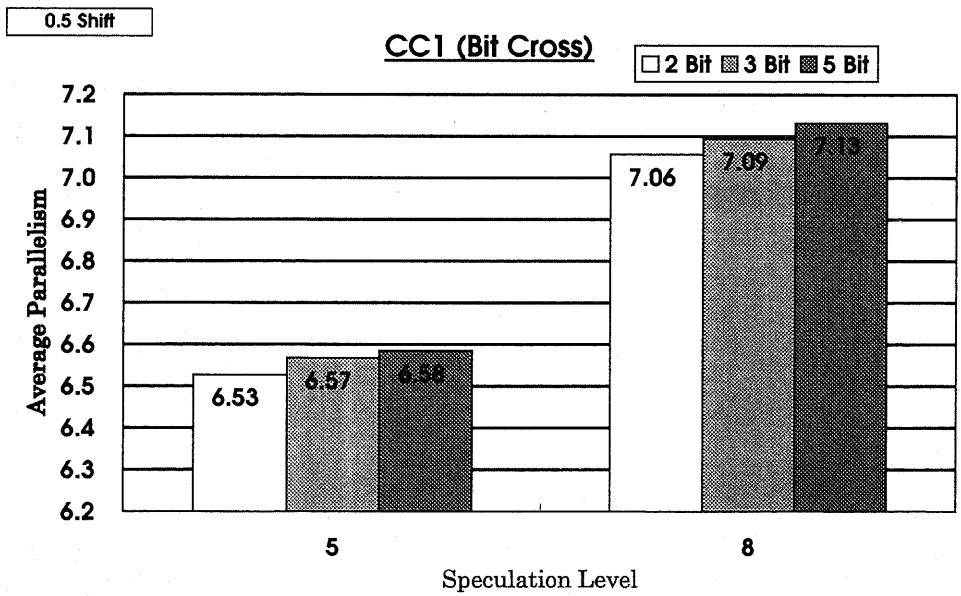


図 9.7: 分岐履歴ビット数と平均並列度 (cc1,2,3,5bit 履歴,0.5bit シフト,BC 方式)

図 9.8, 図 9.9は分岐履歴ビット数と分岐予測成功率の関係を示した図である。分岐履歴ビット数が多いほど分岐予測成功率も上がっている。分岐履歴ビット数が多いということは、それだけ古い分岐履歴も含めて分岐予測が可能であるためであると考えられる。

図 9.10, 図 9.11は分岐履歴ビット数と有効命令率の関係を示した図である。これはBO 方式と BC 方式で結果に差が出ている。BO 方式では、分岐履歴ビット数が多い方が有効命令率が高い傾向が見られるが、BC 方式では分岐履歴ビット数による差はほとんどない。全体的な傾向としては、投機レベル 5 で 14〜18 %程度、投機レベル 8 で 9〜12 %の有効命令率となることが分かる。

図 9.12, 図 9.13は分岐履歴ビット数とフラッシュ率の関係を示した図である。BO 方式と BC 方式のいずれの方式においても、分岐履歴ビット数が多い方がフラッシュ率が低い傾向が見られる。これは図 9.8, 図 9.9に示した分岐履歴ビット数と分岐予測成功率の関係のちょうど逆の関係にある。先に述べたように、分岐予測成功率とフラッシュ率の和が 100 %になるわけではないが、分岐予測成功率が高いほどフラッシュされる可能性が小さくなるのは確かである。ここで注目すべきは、BO 方式の投機レベル 8 の場合である。分岐履歴ビットが 2bit と 5bit の場合を比較すると、フラッシュ率に 1.15 %の差がある。2bit の場合のフラッシュ率が 5.25 %であるから、5bit にすることで 2bit のときに比べて 22 % (= 1.15 / 5.25) もフラッシュ回数が減ることになる。これはシングルパスの命令展開に例えるならば、分岐予測成功率が 94.75 %だったものが、95.90 %になったこととほぼ同じであり、両者には大きな違いがあると言える。

以上の考察から分岐履歴ビット数に関しては、5bit が優れていると言える。

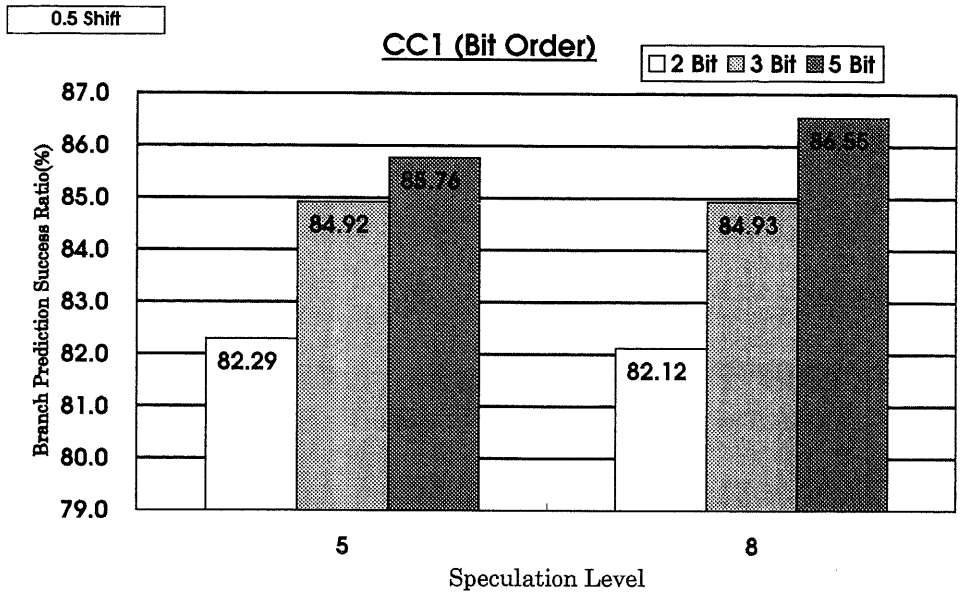


図 9.8: 分岐履歴ビット数と分岐予測成功率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO 方式)

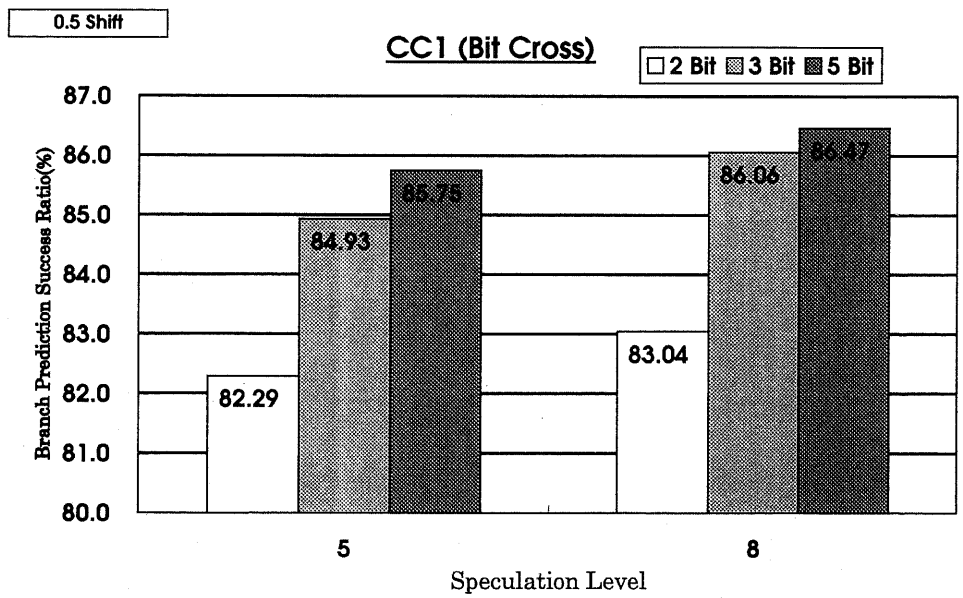


図 9.9: 分岐履歴ビット数と分岐予測成功率 (cc1,2,3,5bit 履歴,0.5bit シフト,BC 方式)

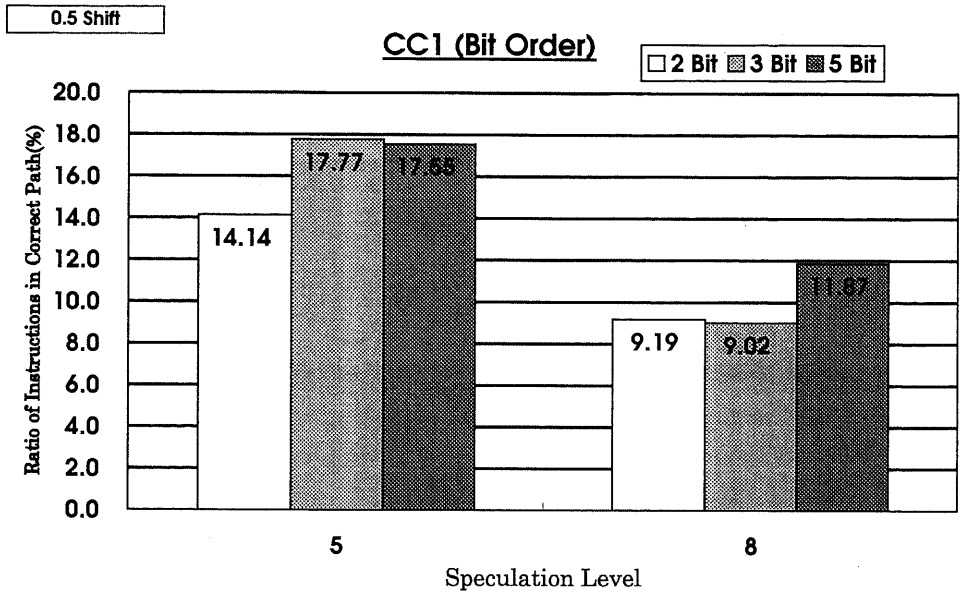


図 9.10: 分岐履歴ビット数と有効命令率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO 方式)

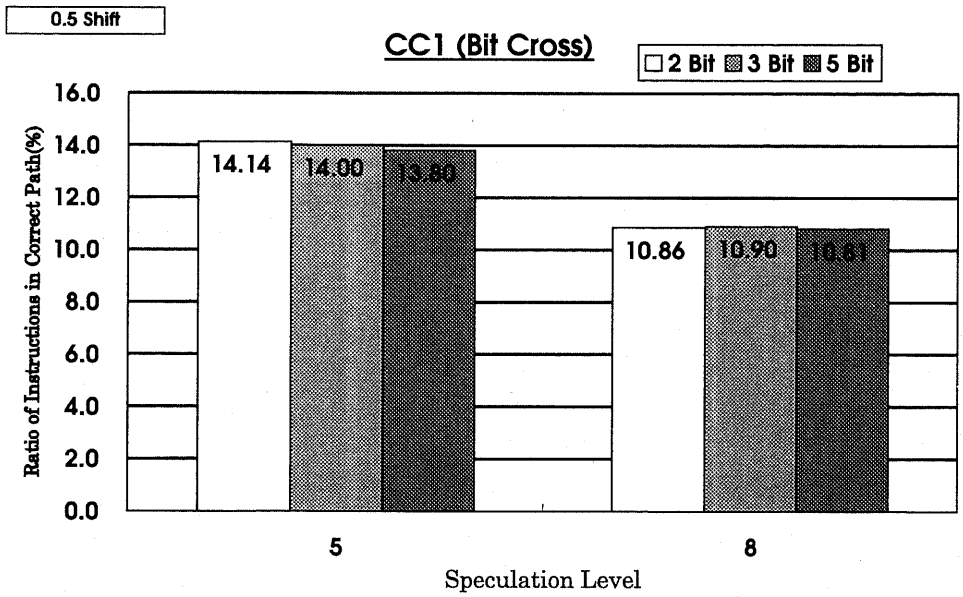


図 9.11: 分岐履歴ビット数と有効命令率 (cc1,2,3,5bit 履歴,0.5bit シフト,BC 方式)

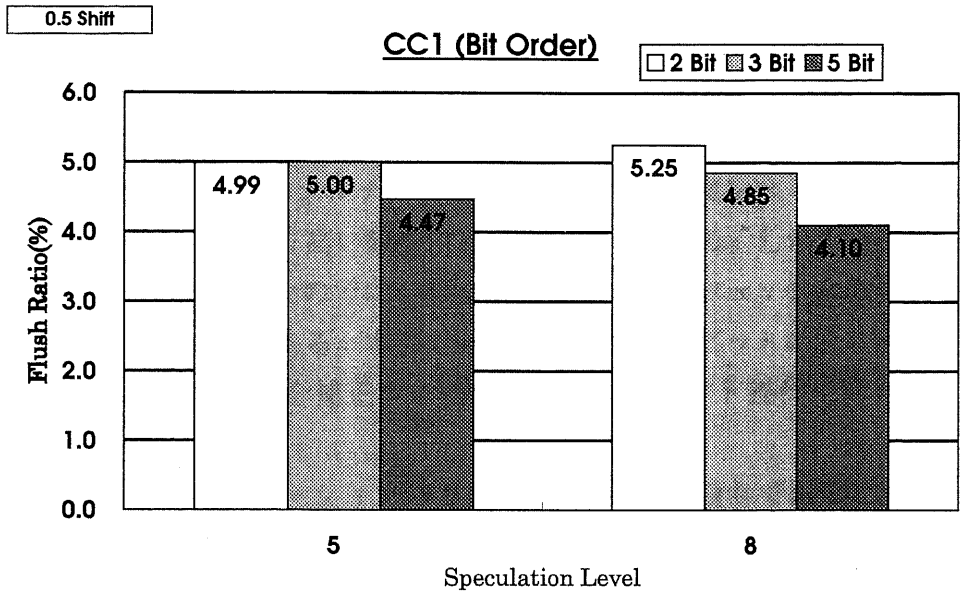


図 9.12: 分岐履歴ビット数とフラッシュ率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO 方式)

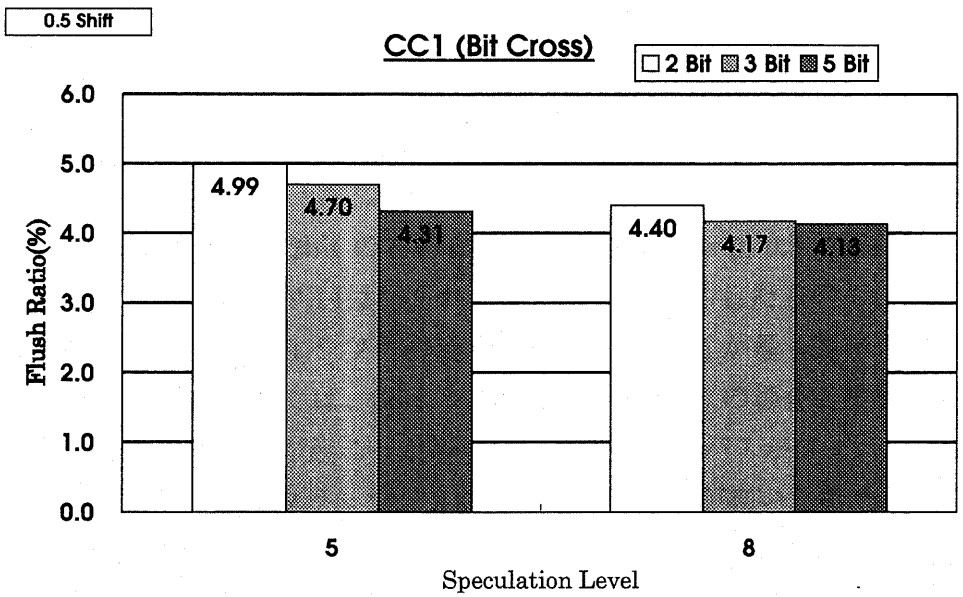


図 9.13: 分岐履歴ビット数とフラッシュ率 (cc1,2,3,5bit 履歴,0.5bit シフト,BC 方式)

9.4 実行確率管理ビット読み出し方法に関する性能評価

第8.2節で述べたように、投機レベルの小さい、つまりより実行完了ポイントに近い部分のパスの部分で、フェッチされていないパスに実行が移った場合、まずフェッチされていた命令を廃棄してそのパスへのフェッチから処理を再開 (フラッシュ) することになり、ペナルティーが大きいため、投機レベルの小さい部分のパスへの実行予測確率はより高い確率値に設定すべきである。しかしながら、無闇に多くの命令をフェッチしたのでは有効命令率が下がり、フェッチ命令総数が爆発的に増えてしまう可能性がある。本節では、実行確率管理ビットの読み出し時に投機レベルにあわせて確率値を補正するためにおこなうシフト操作に関して評価を行なう。

本節で用いたシミュレーションパラメータは表9.5の通りである。

サンプル・プログラム	cc1
投機レベル	5, 8
分岐履歴ビット数	2 および 5
実行確率管理ビット更新方式	Bit Cross
評価するパラメータ	実行確率管理ビット読み出し時シフト量 (0, 0.5, 1.0, 2.0)

表 9.5: 実行確率管理ビットシフト量に関する性能評価・シミュレーションパラメータ

まず分岐履歴ビットが2bit のものについてシフト量を 0, 0.5, 1.0 に変化させた場合のデータを示す。図 9.14は得られる平均並列度を示したもので、いずれの投機レベルにおいても 0.5bit シフトの場合 (白い棒グラフ) が最も高い平均並列度を得ている。この理由としては、0bit シフト (シフト無し) の場合、投機レベルの小さい部分で分岐予測ミスが起こった場合に、もう一方のパスがフェッチされていない確率が高く、そのままフラッシュされてしまう可能性が高いためである。これは図 9.17に示した実行確率管理ビットのシフト量とフラッシュ率の関係を示したグラフからも読みとれる。シフト無しの場合は他の場合に比べて突出してフラッシュ率が高く、他の 4〜5 倍もある。さらに図 9.15と図 9.17をあわせてみると、シフト無しの場合は、分岐予測成功率とフラッシュ率の和が 99 %程度となり、分岐予測ミスが起こるとフラッシュされる可能性が極めて高いことが分かる。図 9.16 に示

した実行確率管理ビットのシフト量と有効命令率の関係では、シフト無しの場合が最も優れている。これはシフト無しの場合には、ほとんどシングルパスと同じような形の命令展開になるためである。

以上の結果、シフト無しでは十分な複数パスへの投機フェッチが行なえず、性能が上がらないことが言える。0.5bit シフトと 1bit シフトについては、平均並列度以外のデータに大きな差はない。よって平均並列度のより高い 0.5bit シフトが実行確率管理ビットの読み出し時のシフト量と適当である考えられる。

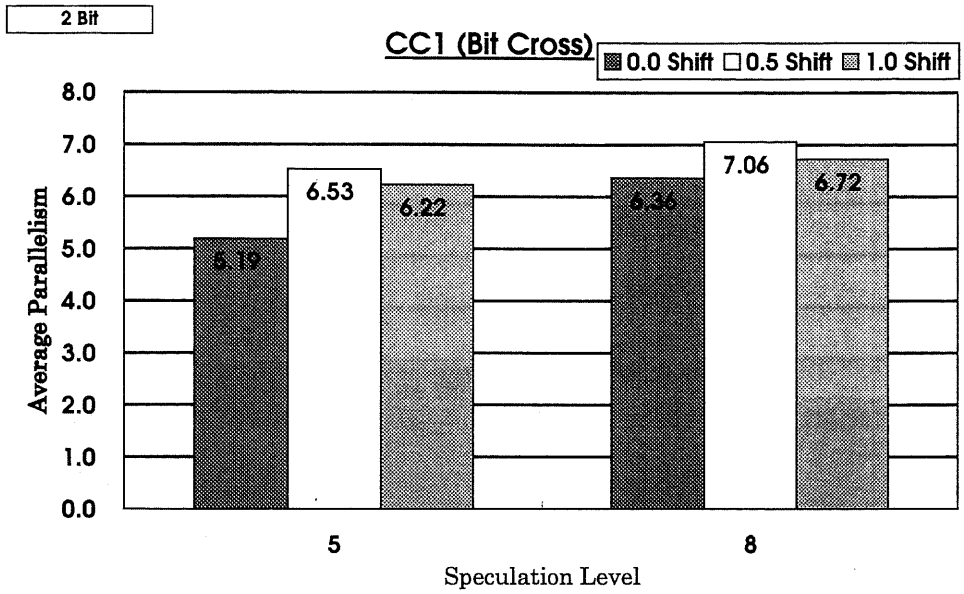


図 9.14: 実行確率管理ビットシフト量と平均並列度 (cc1,2bit 履歴,0～1bit シフト,BC 方式)

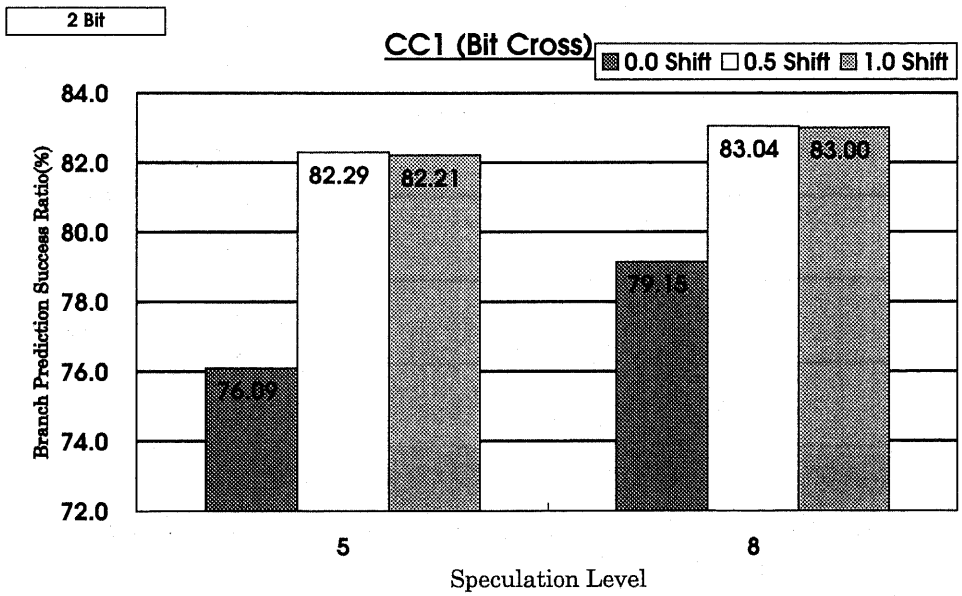


図 9.15: 実行確率管理ビットシフト量と分岐予測成功率 (cc1,2bit 履歴,0～1bit シフト,BC 方式)

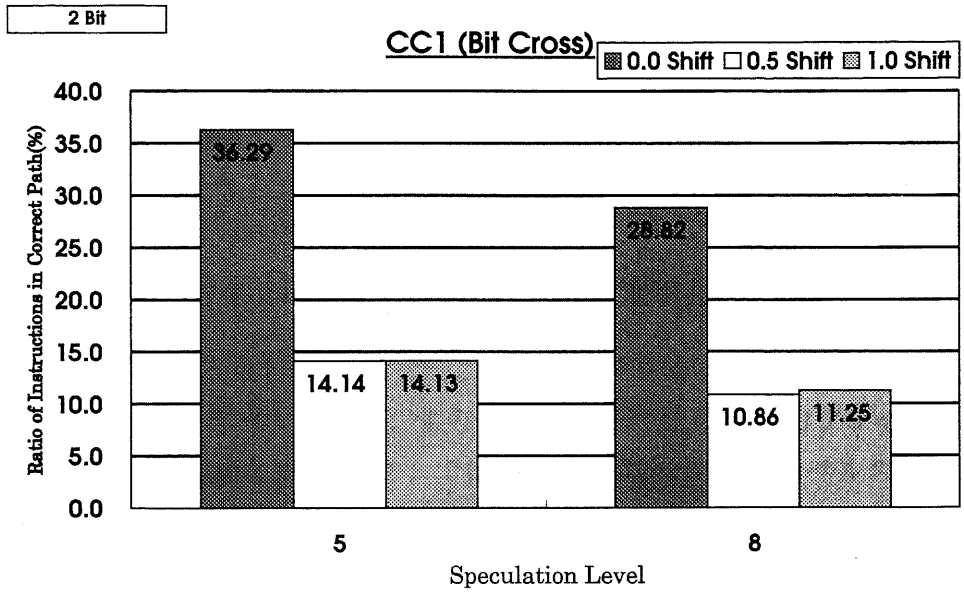


図 9.16: 実行確率管理ビットシフト量と有効命令率 (cc1,2bit 履歴,0～1bit シフト,BC 方式)

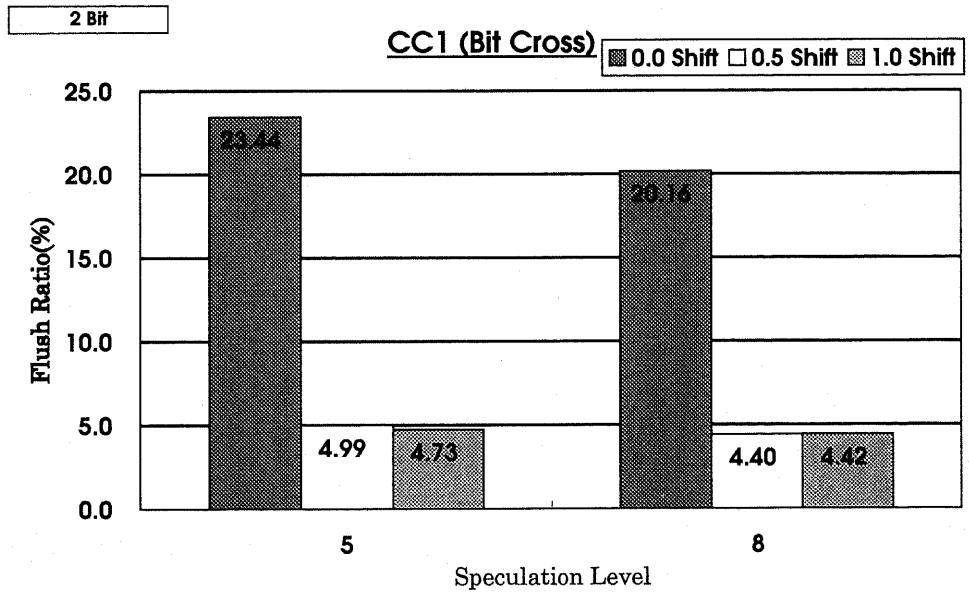


図 9.17: 実行確率管理ビットシフト量とフラッシュ率 (cc1,2bit 履歴,0～1bit シフト,BC 方式)

図 9.18～図 9.21は同様の評価を履歴ビットが 5bit のものに対して行なったグラフである。なお、シフト無しは得られる性能が低いため、今回の評価では外し、かわりに 2bit シフトの場合を調査した。平均並列度に関しては、0.5bit シフトが 1bit, 2bit シフトのものに比べて高くなっている。しかしその他のデータについて、若干ではあるが、1bit, 2bit シフトの方が 0.5bit シフトに比べてよい結果となっている。例えば、図 9.21のフラッシュ率に関しては、0.2～0.4 %程度の違いが出ている。図 9.20によれば、フェッチ命令量はそれぞれ大きな差はないので、フラッシュ率が若干高いにも関わらず 0.5bit シフトの方が平均並列度が高いというのは、次のようなことが原因であると考えられる。

- シフト量が大きい場合には、より投機レベルの低いパスのフェッチ優先度があがるため、1bit, 2bit シフトでは命令の展開形が投機レベルの低い側により偏っていると考えられる。すると同じだけの命令をフェッチしても、フェッチされているコントロールフローツリーの深さ方向が 0.5bit シフトの場合に比べて浅いので、フェッチされている正しいパス上の命令の長さも短いと考えられる。そのかわりに、コントロールフローツリーの幅方向は 0.5bit シフトの場合よりも広くフェッチされているので、フラッシュされる可能性は低くなる。

以上の考察より、0.5bit シフトがバランスのよいシフト量であると考えられる。

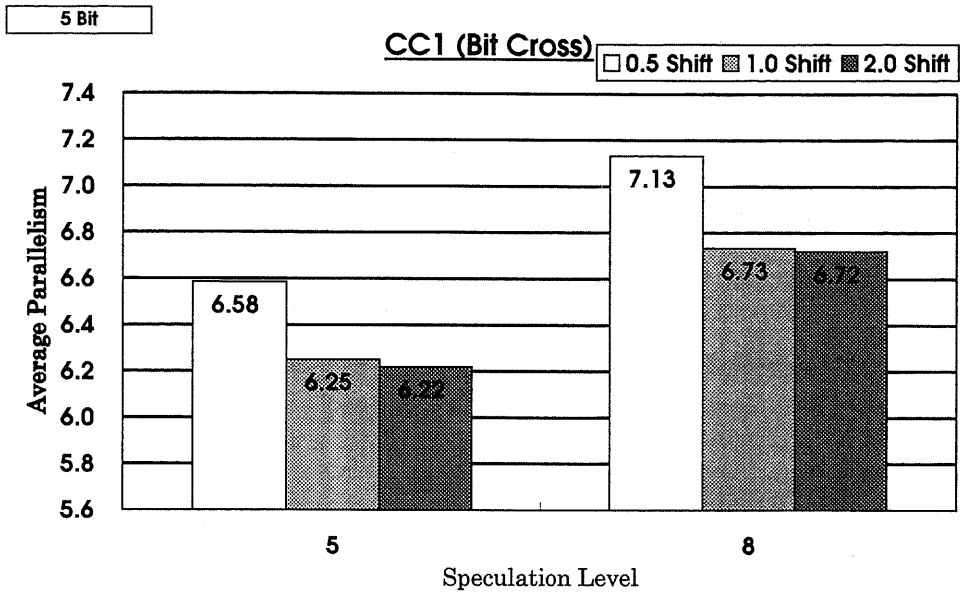


図 9.18: 実行確率管理ビットシフト量と平均並列度 (cc1,5bit 履歴,0.5～2bit シフト,BC 方式)

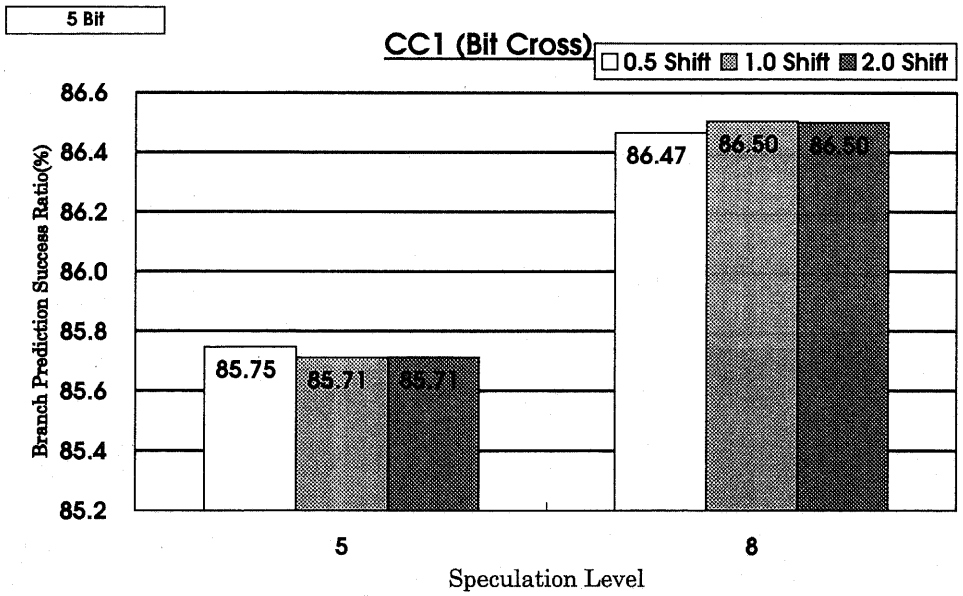


図 9.19: 実行確率管理ビットシフト量と分岐予測成功率 (cc1,5bit 履歴,0.5～2bit シフト,BC 方式)

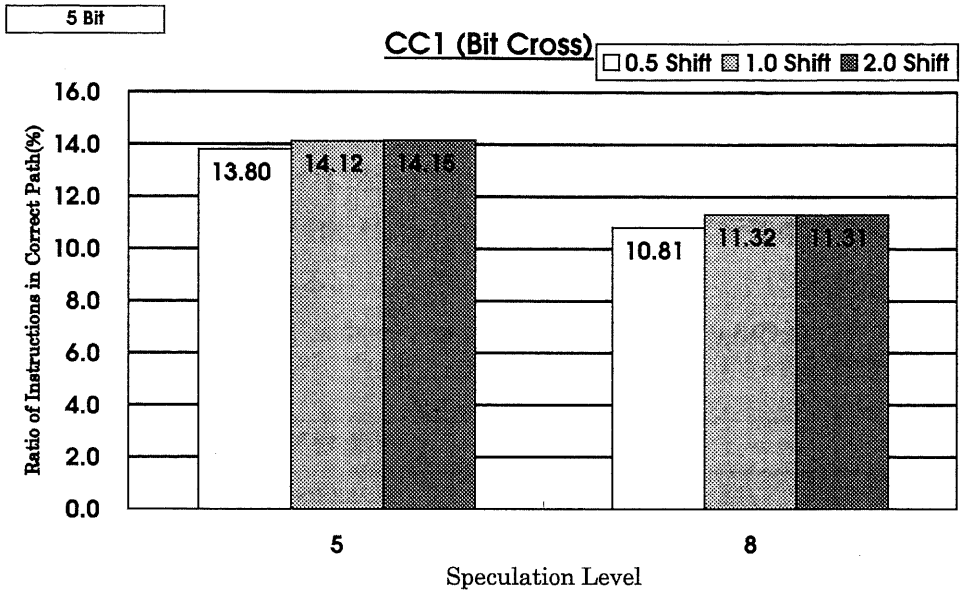


図 9.20: 実行確率管理ビットシフト量と有効命令率 (cc1,5bit 履歴,0.5～2bit シフト,BC 方式)

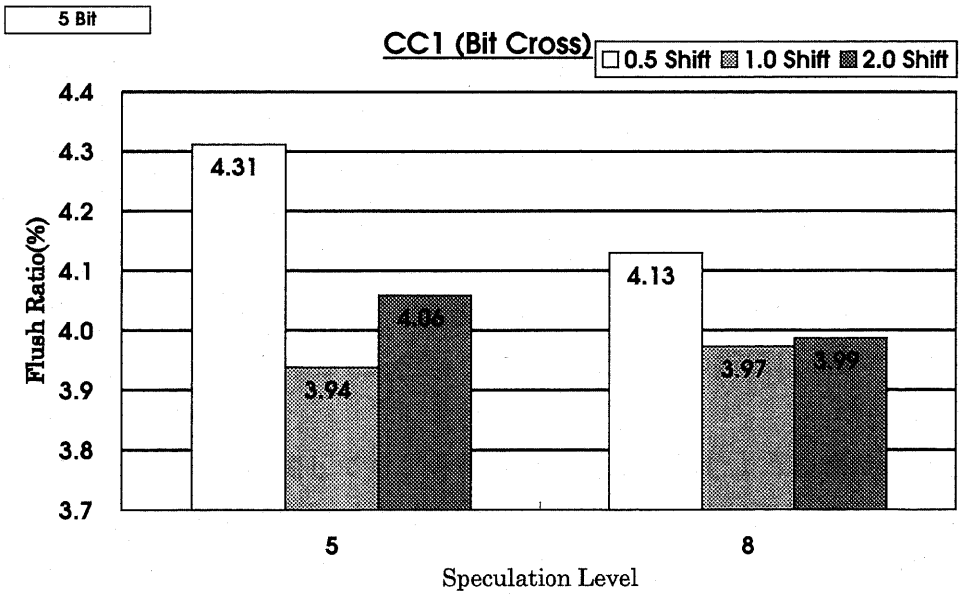


図 9.21: 実行確率管理ビットシフト量とフラッシュ率 (cc1,5bit 履歴,0.5～2bit シフト,BC 方式)

9.5 BO 方式および BC 方式の性能比較評価

図 9.22～図 9.25は、Bit Order (BO) 方式と Bit Cross (BC) 方式について、比較検討のために図 9.6～図 9.13 をそれぞれ 1 枚にまとめたものである。それぞれのグラフは 6 本の棒グラフが 2 組で構成されており、左から分岐履歴ビットが 2bit, 3bit, 5bit のものが 2 本ずつになっている。またその 2 本のうち左側のものが Bit Order 方式、右側のものが Bit Cross 方式である。なお実行確率管理ビットの読み出し時シフト値は 0.5 である。

なお、本節で用いたシミュレーションパラメータは表 9.6の通りである。

サンプル・プログラム	cc1
投機レベル	5, 8
分岐履歴ビット数	2, 3, 5
実行確率管理ビット読み出し時シフト量	0.5
評価するパラメータ	実行確率管理ビット更新方式 Bit Order および Bit Cross

表 9.6: BO 方式および BC 方式の性能比較評価・シミュレーションパラメータ

まず図 9.22の平均並列度に関しては、いずれの投機レベル、どの分岐履歴ビット数においても BO 方式の方が BC 方式を上回っている。特に 5bit の分岐履歴ビットを利用する場合には、特にその差が大きくなっている。より多くの正しいパスが BO 方式によってフェッチされていることになる。

次に図 9.23の分岐予測成功率に関しては、分岐履歴ビットが 2bit, 3bit のときは BC 方式が、5bit のときは BO 方式が優れている。特に 2bit, 3bit に比べて性能が高い 5bit で分岐予測成功率が勝っている BO 方式がこの指標についても優れていると言って良いであろう。

図 9.24の有効命令率に関しても、分岐予測成功率と同じような傾向が見られる。ここでも 5bit の分岐履歴ビットを使う場合には BO 方式が優れており、さらに 6 本の棒グラフの中で最も高いものが BO 方式であることも考えれば、BO 方式が優れている。

図 9.25のフラッシュ率に関しては、分岐履歴ビット 5bit の投機レベル 8 の場合を除いて BC 方式が BO 方式に勝っている、もしくは同じ性能をもっている。しか

しいずれも4〜5％程度であり、分岐履歴ビット5bitの投機レベル8の場合にBO方式が勝っていることを考えれば、大きな差はないと言える。

BO方式とBC方式については、ハードウェアへの実装上もBO方式の方が特に更新操作の実装で簡単であり、性能的にもほとんどの指標で優れており、とりわけ得られる平均並列度が勝っていることを考えればBO方式の方がよいといえる。

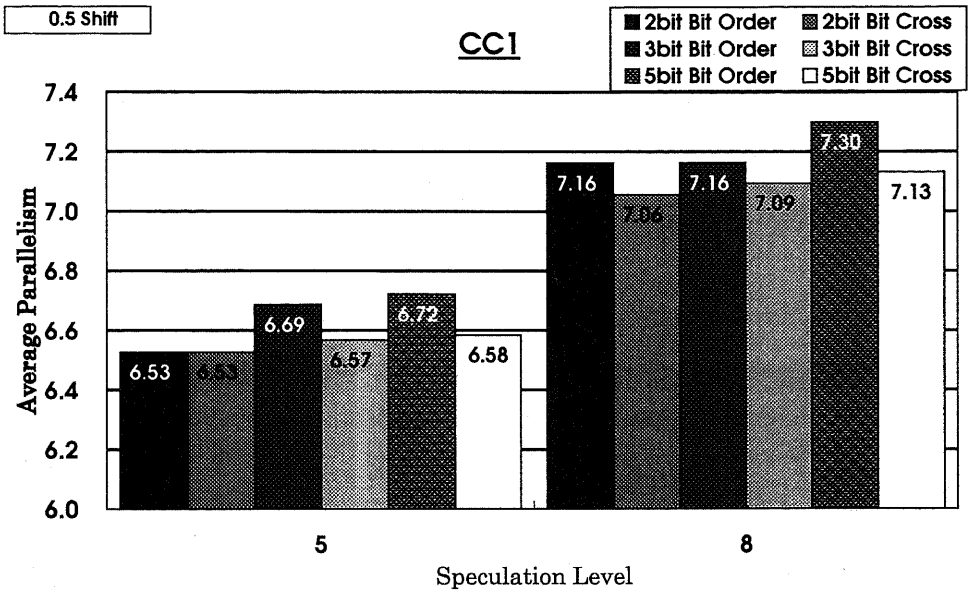


図 9.22: BO 方式および BC 方式の平均並列度 (cc1,2,3,5bit 履歴,0.5bit シフト,BO/BC 方式)

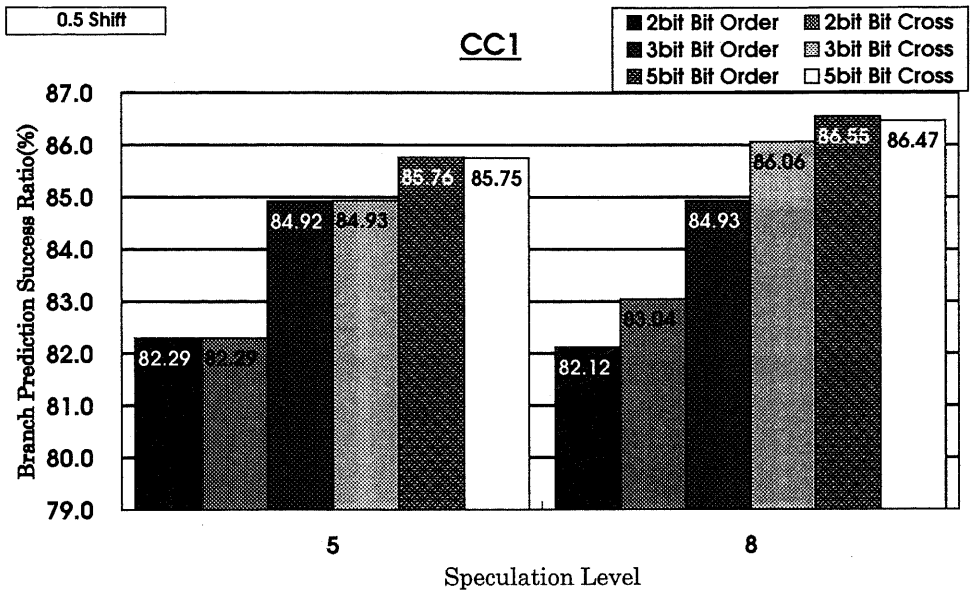


図 9.23: BO 方式および BC 方式の分岐予測成功率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO/BC 方式)

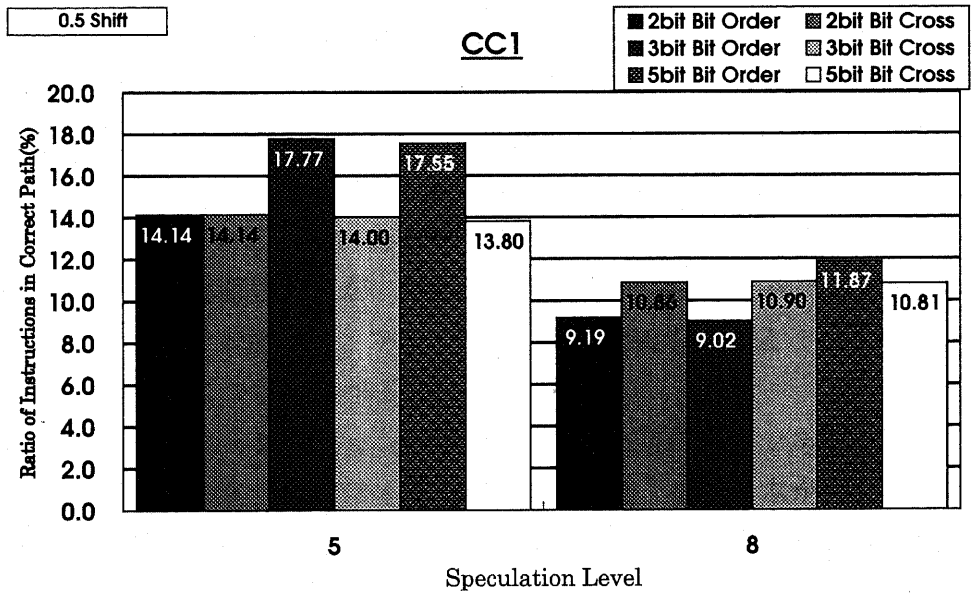


図 9.24: BO 方式および BC 方式の有効命令率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO/BC 方式)

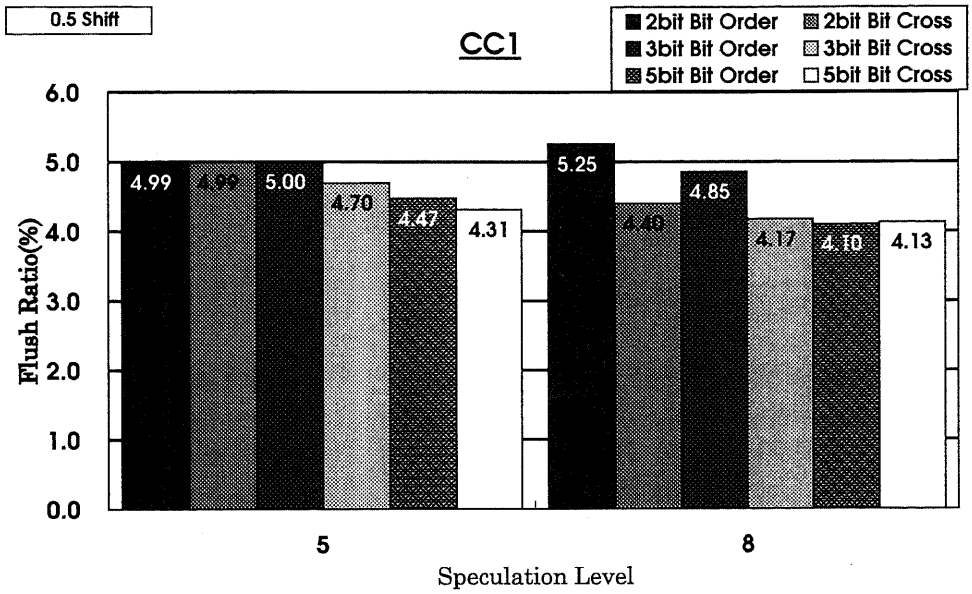


図 9.25: BO 方式および BC 方式のフラッシュ率 (cc1,2,3,5bit 履歴,0.5bit シフト,BO/BC 方式)

9.6 フェッチ命令数制限時の性能評価

これまでのシミュレーション結果を見ると、全フェッチ命令に含まれる有効命令の割合は、投機レベル 5 のときで 15 %程度、投機レベル 8 のときで 10 %程度であった。複数パスへの分岐予測を行なう Multi BTAC、および複数パスの中から次にフェッチすべきパスを決定する実行確率管理機構について、これまでに選択的フェッチがなされていること、ある程度の並列性を取り出せるような命令をフェッチしていることが確認されたが、フェッチ命令数の削減については、あまり触れてこなかった。そこで本節では、フェッチ命令数に制限を加えることで性能にどのような影響が及ぶかについて述べる。

本節で用いたシミュレーションパラメータは表 9.7の通りである。

図 9.26～図 9.33の各グラフは 4 本の棒グラフ 2 組、合計 8 本からなっている。それぞれの組は投機レベル 5 と 8 の場合であり、4 本の棒グラフは左から順に、

2x フェッチ命令数を最大投機レベルの 2 倍までのベーシックブロック数に制限する。つまり最大投機レベルが 8 の場合には、ベーシックブロック 16 個分までにフェッチ命令数が制限される。

サンプル・プログラム	cc1
投機レベル	5, 8
分岐履歴ビット数	5
実行確率管理ビット読み出し時シフト量	0.5
実行確率管理ビット更新方式	Bit Order および Bit Cross
評価するパラメータ	フェッチ命令数制限 最大投機レベル× 2,3,5 ベーシックブロック分

表 9.7: フェッチ命令数制限時の性能評価・シミュレーションパラメータ

3x フェッチ命令数を最大投機レベルの 3 倍までのベーシックブロック数に制限する。つまり最大投機レベルが 8 の場合には、ベーシックブロック 24 個分までにフェッチ命令数が制限される。

5x フェッチ命令数を最大投機レベルの 5 倍までのベーシックブロック数に制限する。つまり最大投機レベルが 8 の場合には、ベーシックブロック 40 個分までにフェッチ命令数が制限される。

No Limit これまでのシミュレーションで使ってきた制限のないモデル。つまりフェッチは 1 つのパスが最大投機レベルに達するか、最も実行確率の高いパスから先にパスをのばせなくなるまで (例えば、分岐予測エントリがないなど) 行なわれる。

図 9.26および図 9.27によると、BO 方式、BC 方式ともに投機レベル 8 で、 2x 時には無制限の時に比べてとりだし得る並列度が 94 %程度に、 3x 時には 96 %程度に、 5x 時には 98.5 %程度に減少することが分かった。一方、図 9.30および図 9.31によると有効命令率は、BO 方式、BC 方式ともに投機レベル 8 で、2x 時には無制限の時に比べて 1.8 倍程度に、 3x 時には 1.45 倍程度に、 5x 時には 1.2 倍程度に増加することが分かった。

2x 制限のときに有効命令率が 50 %とはならないのは、必ずしもフェッチされた命令の 50 %が有効な命令であるとは限らないためで、例えばフラッシュされた場合には、フェッチされている命令すべてが不要な命令となってしまうなどの場合があるからである。

ここで注目すべきは、投機レベル 5 のときの無制限モデルと、投機レベル 8 の時の 2x 制限モデルである。両者の比較をすると、表 9.8のようになる。表 9.8から

モデル	投機レベル 5, 無制限	投機レベル 8, 2x 制限
平均並列度	6.72	6.87
有効命令率 (%)	17.55 %	21.52 %

表 9.8: フェッチ命令数制限モデルと無制限モデルの比較

わかることは、投機レベル 8 の 2x 制限モデルの方が、より少ないフェッチ命令数でより大きな平均並列度が得られると言うことである。

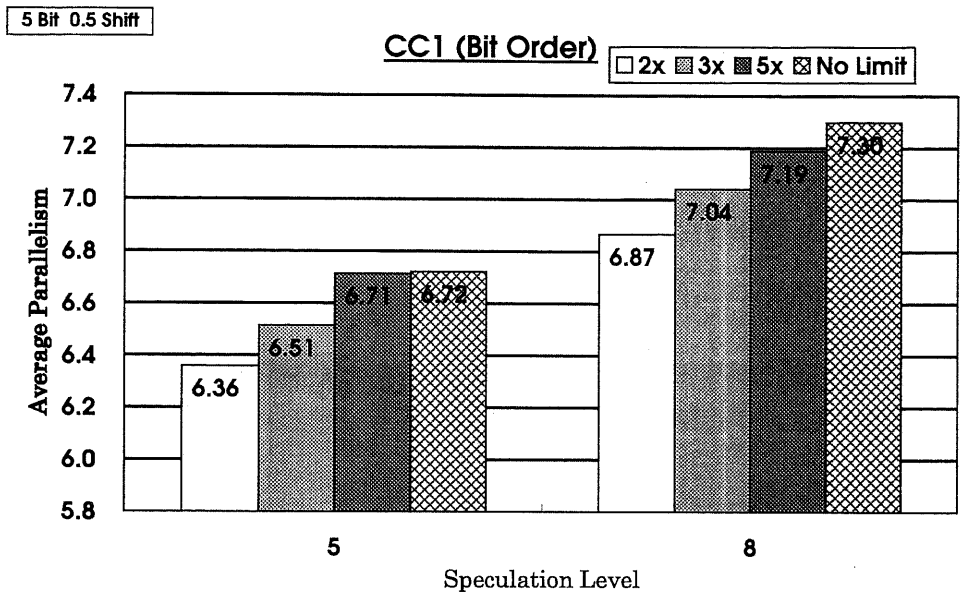


図 9.26: フェッチ命令数制限時の平均並列度 (cc1,5bit 履歴,0.5bit シフト,BO 方式)

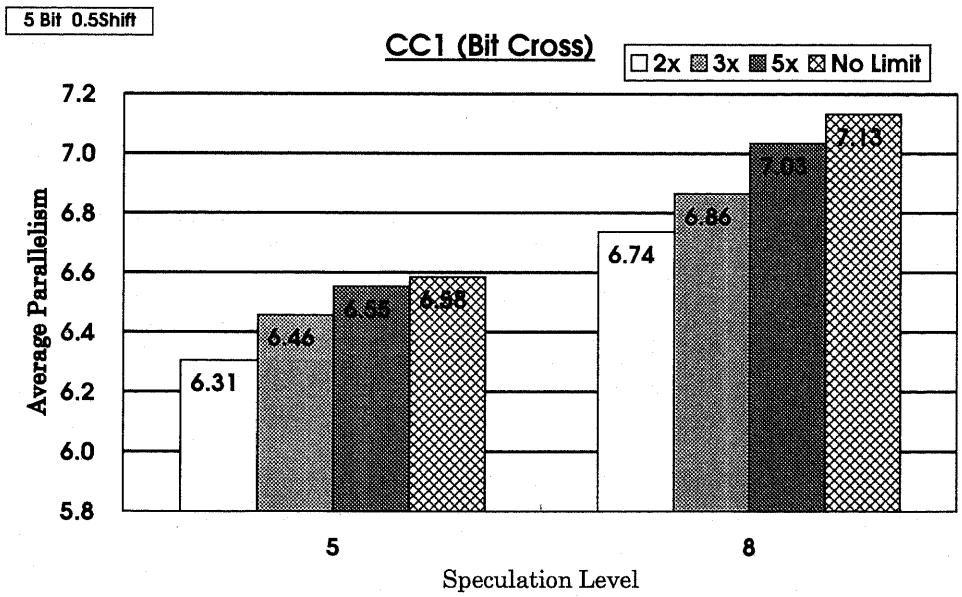


図 9.27: フェッチ命令数制限時の平均並列度 (cc1,5bit 履歴,0.5bit シフト,BC 方式)

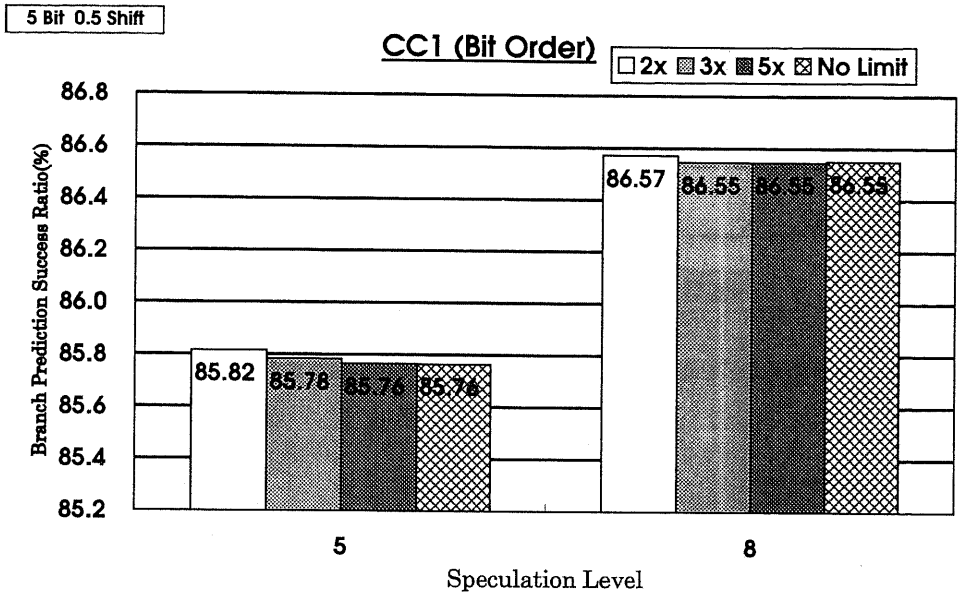


図 9.28: フェッチ命令数制限時の分岐予測成功率 (cc1,5bit 履歴,0.5bit シフト,BO 方式)

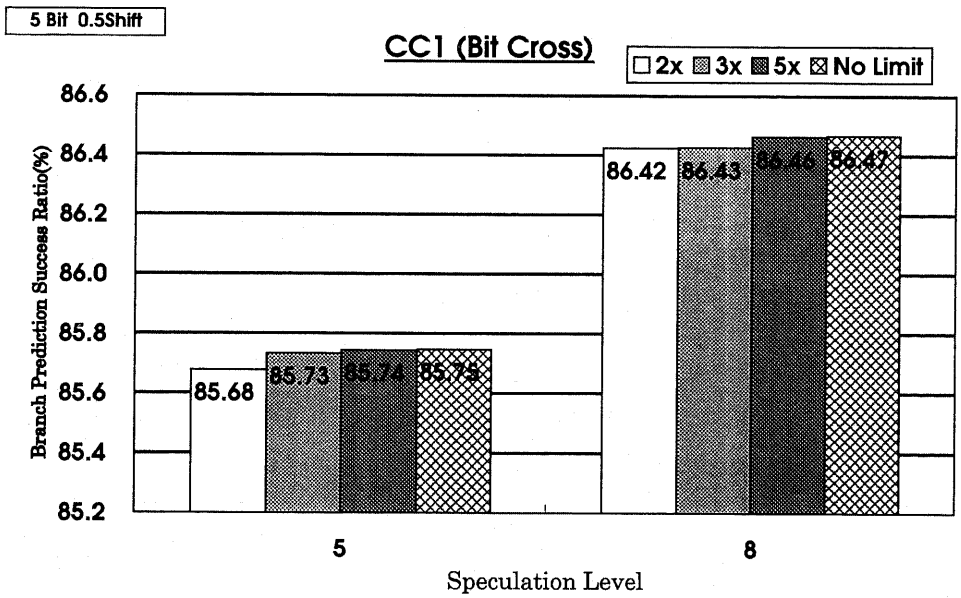


図 9.29: フェッチ命令数制限時の分岐予測成功率 (cc1,5bit 履歴,0.5bit シフト,BC 方式)

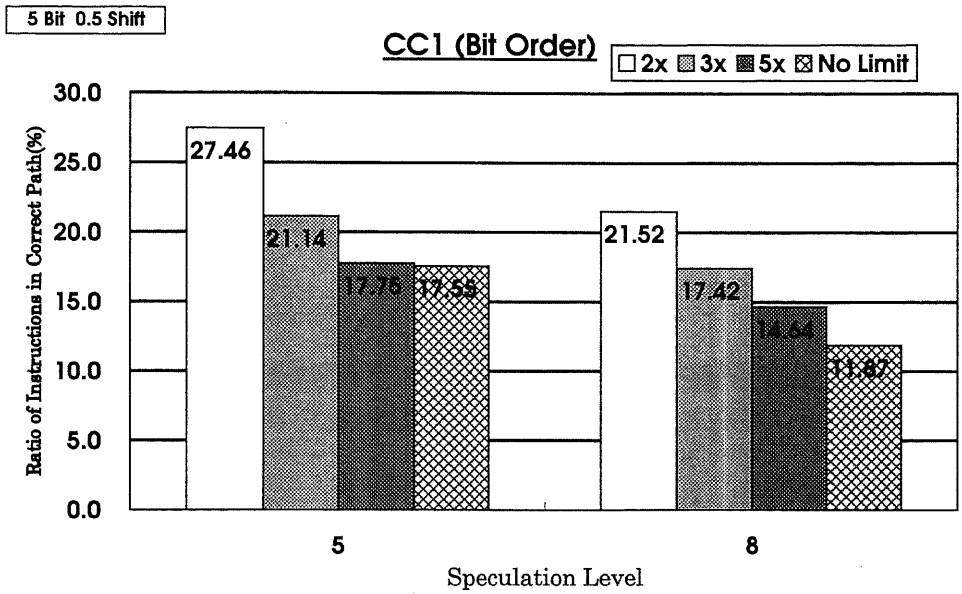


図 9.30: フェッチ命令数制限時の有効命令率 (cc1,5bit 履歴,0.5bit シフト,BO 方式)

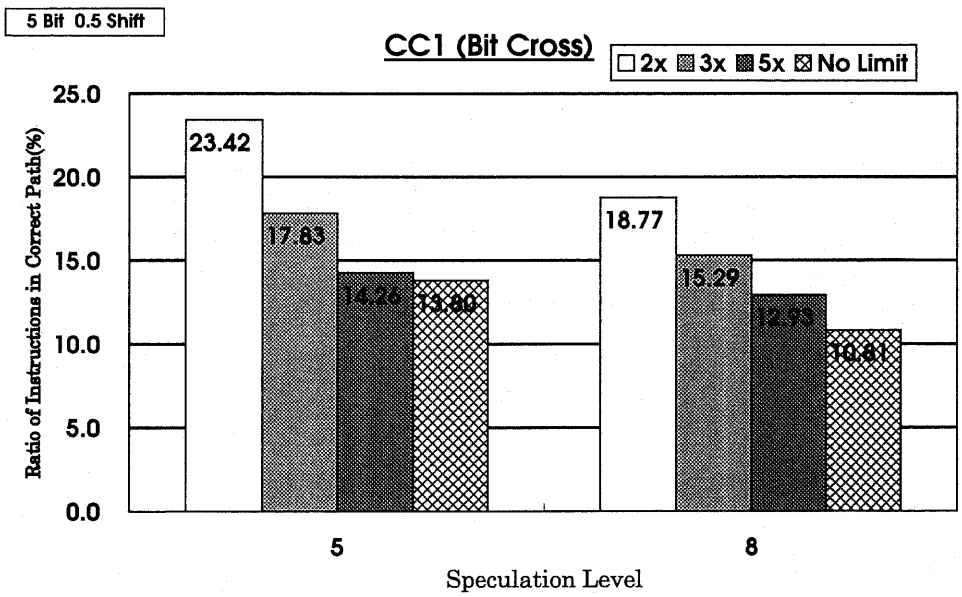


図 9.31: フェッチ命令数制限時の有効命令率 (cc1,5bit 履歴,0.5bit シフト,BC 方式)

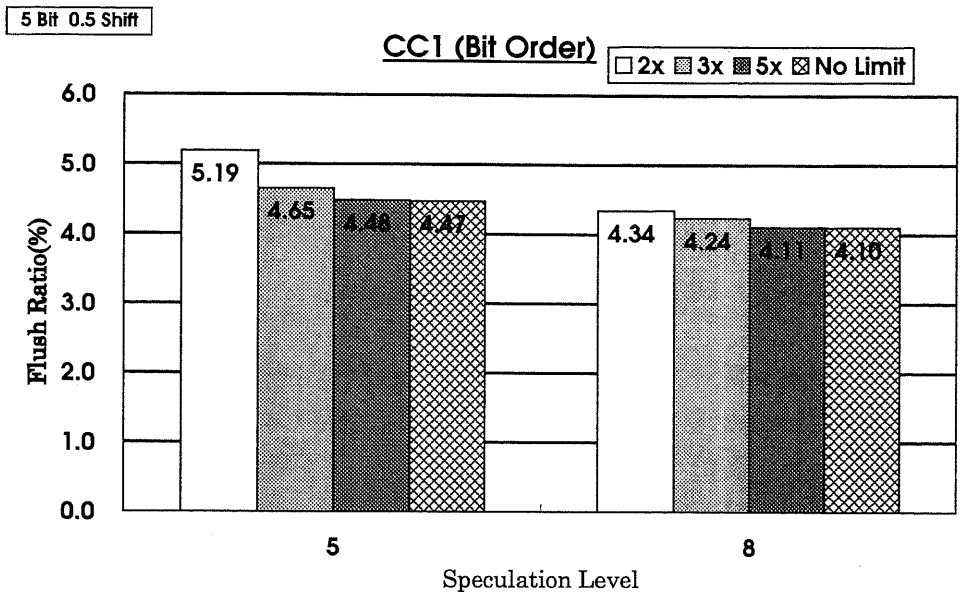


図 9.32: フェッチ命令数制限時のフラッシュ率 (cc1,5bit 履歴,0.5bit シフト,BO 方式)

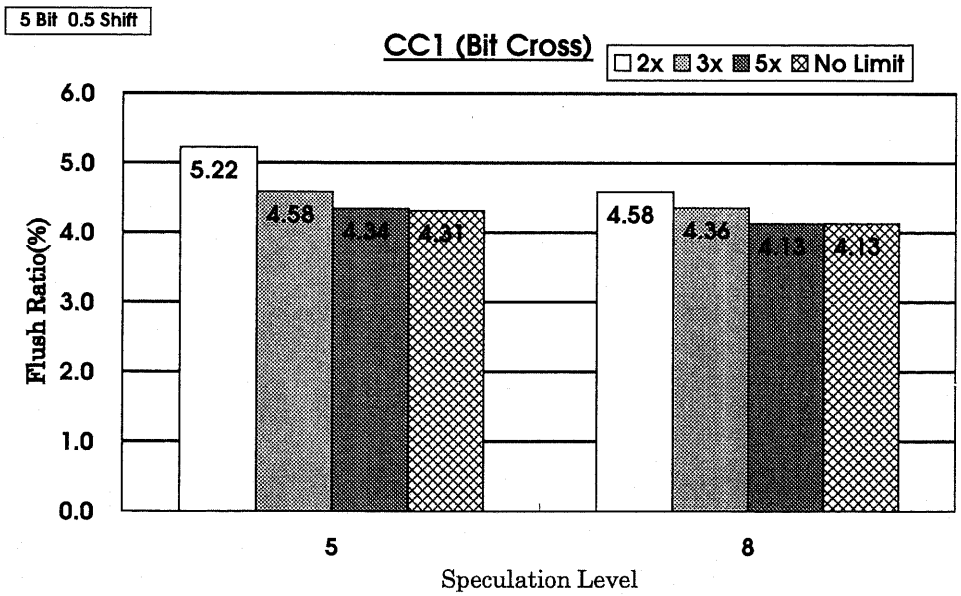


図 9.33: フェッチ命令数制限時のフラッシュ率 (cc1,5bit 履歴,0.5bit シフト,BC 方式)

第 10 章

考察

本章では、大規模データパス・アーキテクチャの命令供給システムであるコントロールフロー先行展開について他の方式との比較を行なう。また大規模データパス・アーキテクチャについてまとめる。

10.1 コントロールフロー先行展開に関する考察

10.1.1 理想的なモデルとの比較

第8章において、コントロールフロー先行展開機構の提案を行なったが、その際に示した2つの実行確率管理機構はいずれもハードウェアへの実装を考慮して、簡単なビット操作で近似的に実行確率の管理を行なうことを目指したものであった。この機構を用いて、選択的な複数パスへの投機フェッチが可能であることを第9章で示したが、ここではソフトウェア・シミュレーションにより実行確率の管理を浮動小数点計算で行なう Floating モデルとの比較を行なう。

Floating モデル Floating モデルとは次のようなモデルである。

- すべての分岐履歴を無制限に記録し、Taken or Not-Taken 型の分岐命令の場合には分岐成立回数と分岐不成立回数のカウンタをもつ。レジスタ間接アドレッシングで分岐先が決まる分岐命令については、すべての分岐先への記録を保持し、さらにその分岐先への分岐回数をカウントするカウンタをもつ。
- 分岐予測は最も大きなカウンタ値をもっている分岐先を第一候補とする。

- 各分岐ポイントから分岐パスへの遷移確率は、(そのパスへ遷移したカウンタ値) ÷ (その分岐命令が実行されたカウンタ値) の浮動小数点で表現し、コントロールフロー・ツリーを実行完了ポイントからフェッチポイントに向かって浮動小数点のかけ算として各パスの先端への遷移確率を求める。
- 各パスの先端の中で最も大きい遷移確率をもつパスを次のフェッチポイントとする。

なお、Floating モデルについても、コントロールフロー・ツリーの上部、実行完了ポイントの方が遷移確率が高くなるような補正値 (Δ) を設けた。これはコントロールフロー・ツリー上で 1 段分岐命令をまたぐたびに補正値Δだけ遷移確率を減算することで実現した。よってΔ=0.10 の場合に、投機レベル 8 の Floating モデルでは、8 段分岐を跨ぐと遷移確率が 0.80 だけ減算されることになる。

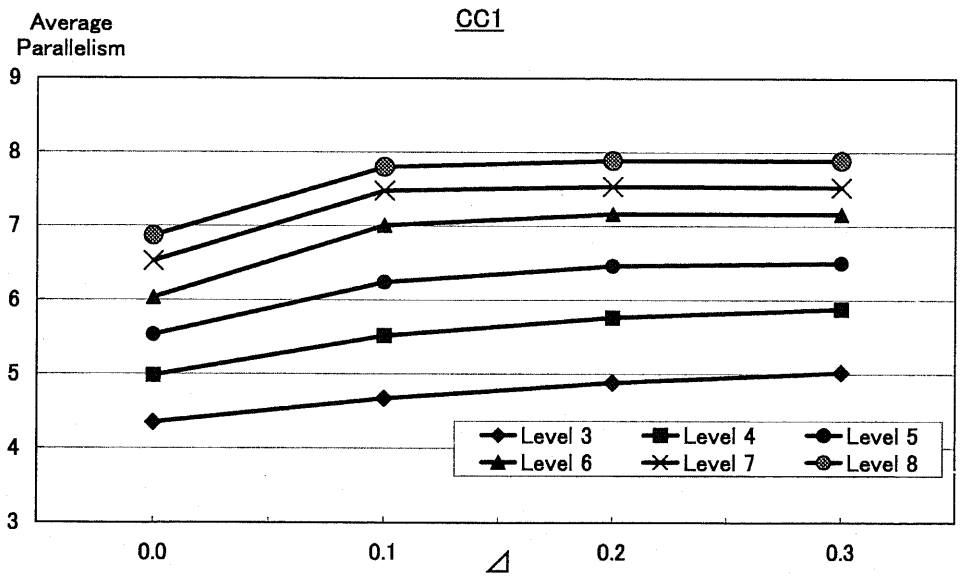


図 10.1: Floating モデルで得られる平均並列度 (cc1)

図 10.1は、Floating モデルで得られる平均並列度を示したものである。投機レベルが小さい部分ではΔの影響が大きいが、投機レベルが大きくなると 0.1〜0.3 はほとんど同じ値になり、Δ=0.0 の場合とΔ=0.1 の場合のみに違いが現れている。

図 10.2は、Floating モデルの Δ=0.0〜0.3 のそれぞれに対して、フェッチした全命令の中の有効な命令の割合を示したものである。Δ の増加と共に有効命令率が下がることが分かる。

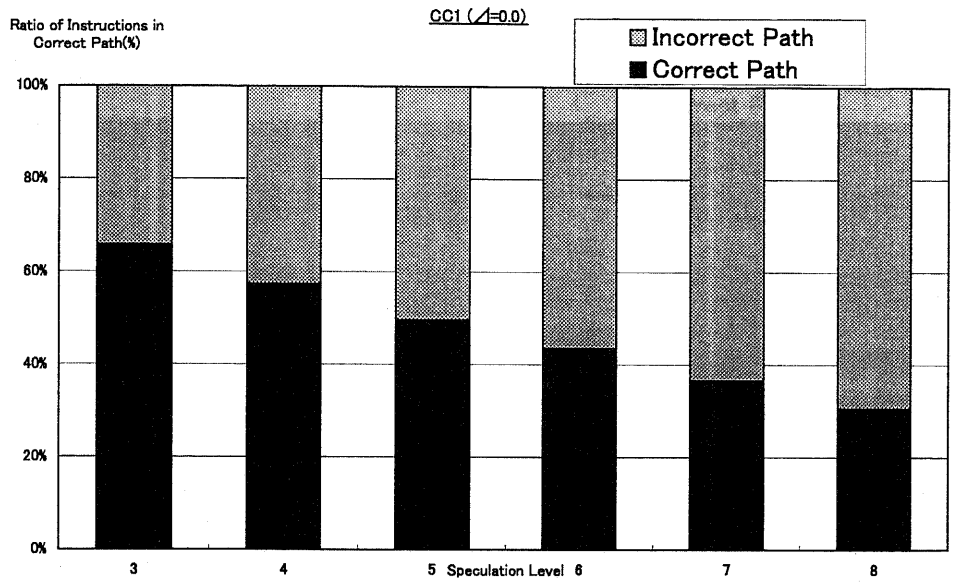


図 10.2: Floating モデルの有効命令率 (cc1, Δ=0.0)

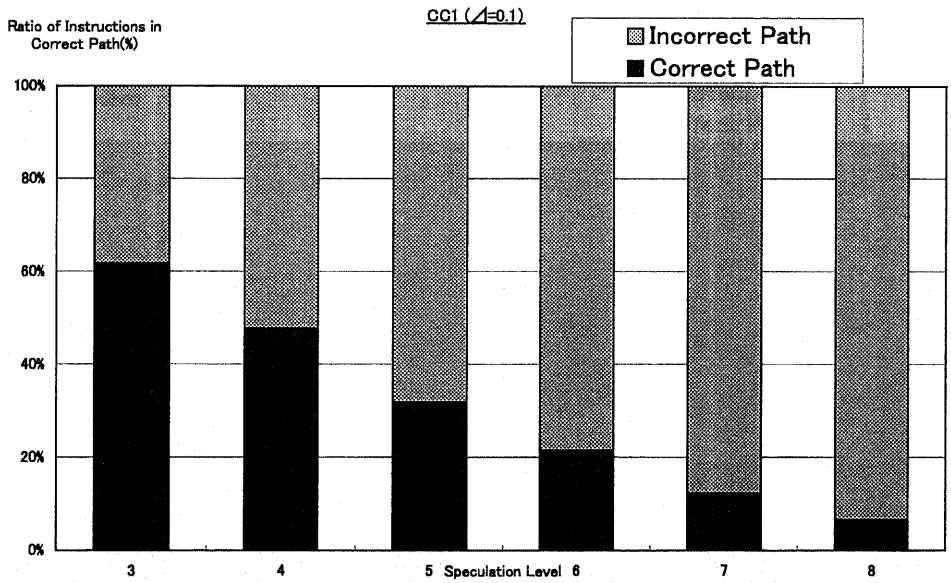


図 10.3: Floating モデルの有効命令率 (cc1, Δ=0.1)

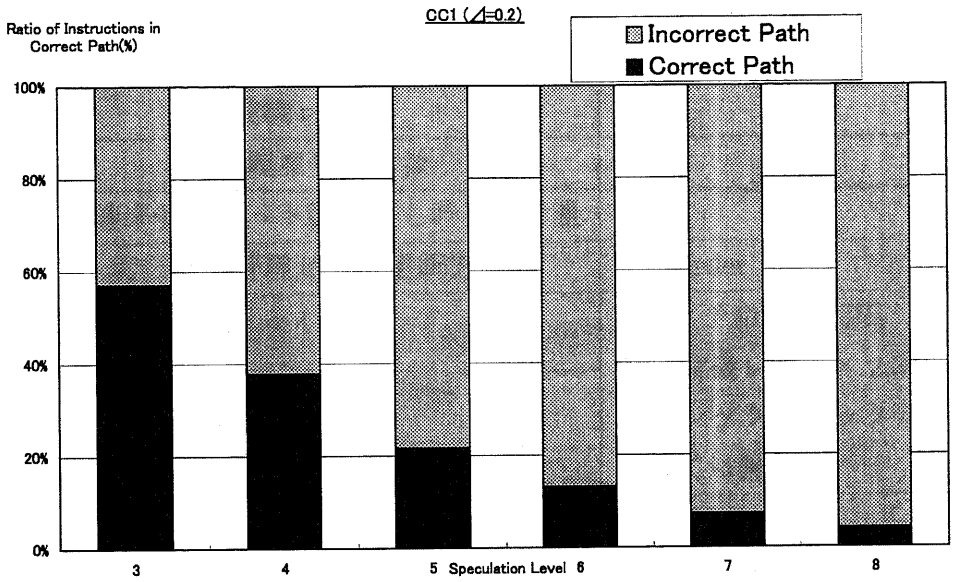


図 10.4: Floating モデルの有効命令率 (cc1, Δ=0.2)

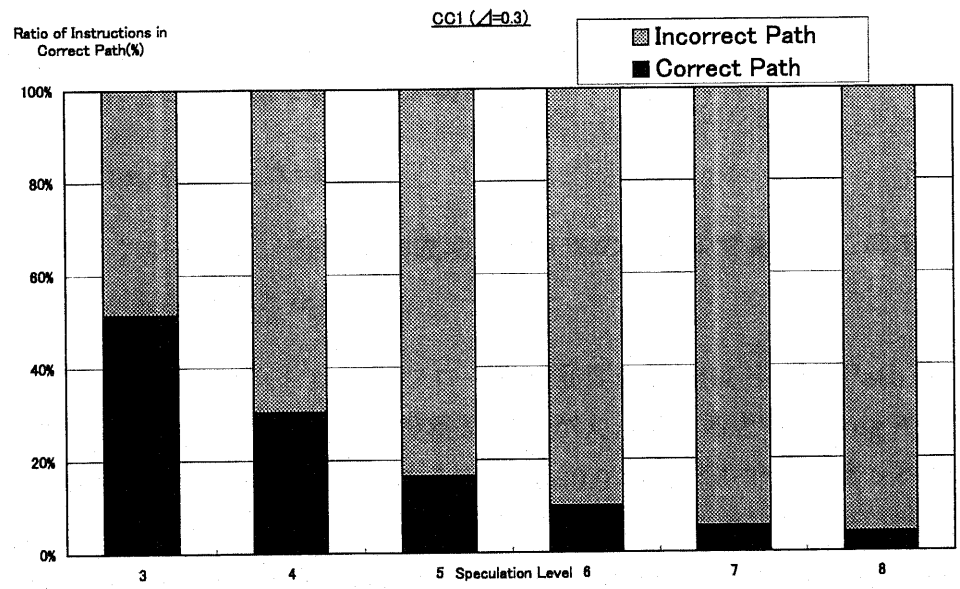


図 10.5: Floating モデルの有効命令率 (cc1, Δ=0.3)

次に、コントロールフロー先行展開とオラクルモデルなどとの性能比較を行なう。第9章での評価から、コントロールフロー先行展開の実行確率管理部は表10.1に示す通りとする。

投機レベル	5, 8
分岐履歴ビット数	5
実行確率管理ビット読み出し時シフト量	0.5
実行確率管理ビット更新方式	Bit Order
評価するパラメータ	フェッチ命令数制限 最大投機レベル× 2,3,5 ベーシックブロック分 もしくは無制限

表 10.1: コントロールフロー先行展開実行確率管理部パラメータ

図10.6は次の6つの命令展開方式を利用して複数パスへの投機フェッチ (一部単一パス) を行なった場合に得られる平均並列度である。

Single Path シングルパスへの命令展開

2x, 3x, 5x, No Limit 表10.1のパラメータをもつコントロールフロー先行展開，それぞれフェッチ命令数制限が異なる

Float Floating モデル (Δ=0.0)

Oracle オラクルモデル，分岐予測成功率100 %のモデルであり、このモデルによる性能が限界性能となる

図10.6によると、コントロールフロー先行展開の性能はSingle Path とOracleの中間に位置することがわかる。投機レベルが大きくなるとFloatingモデルは実行確率の精度がよいため高性能となるが、Δ=0.0 のモデルでは投機レベルが小さい場合にはそれほどの性能は出ない。コントロールフロー先行展開は、Single PathとOracleモデルの性能差を、おおよそ40〜60 %程度縮めていることが分かる。

Floatingモデルの性能があまり高くない理由は次のように考えられる。Δ=0.0のモデルでは、命令の展開形がSingle Pathに近くなる。これは多くの分岐命令の

分岐方向が偏っており、ある程度の実行回数を経ると、Taken 方向に 990 回, Not Taken 方向に 10 回などのように、それぞれのパスの実行予測確率が 99 %:1 % のようになるものがかなりあるためである。すると Single Path モデルの性能が低いのと同じ理由で (分岐確率の低いパスへ実際の実行が移った場合に正しいパスからフェッチされている命令がほとんどなくなってしまう)、性能が低くなるものと考えれる。そのような意味では Δ=0.1, 0.2, 0.3 のモデルの性能はかなり改善されている。ただし Δ のような形で補正値を加えるのがよいのか、他の形で補正を加えるべきであるかどうかはさらに検討が必要である。

もう一つ性能が出ない大きな理由があると考えられる。Floating モデルでは、ある分岐命令の分岐先への遷移確率を、過去の「すべての」履歴をもとに計算している。そのため、ある分岐命令がまず 10000 回だけ A 方向に分岐し、そのあとずっと別の B 方向に分岐し続けたとすると、過去の「すべての」履歴を見ている限り、10001 回～20000 回までの間は A 方向に分岐する確率が高いと計算されてしまう。これを例えば「最近 5 回の」履歴をもとに計算するとすれば、10004 回目の予測からは A 方向に分岐する確率が高いと予測することができる。このような性質を示す分岐命令がどの程度あるかは未調査ではあるが、一般に分岐予測においては、時間方向のローカル性を利用した方がよいとされていることからも、過去のすべての履歴情報を利用することが必ずしも理想的ではないといえる。

図 10.7は同じ 6 つの命令展開方式を利用して複数パスへの投機フェッチ (一部単一パス) を行なった場合の有効命令率である。命令フェッチ量で見ると 2x のフェッチ命令数制限をかけたコントロールフロー先行展開で、Single Path に比べておよそ 2 倍強の命令フェッチ量となっている。

10.1.2 Disjoint Eager Execution との比較

コントロールフロー先行展開の選択的な複数パスへのフェッチ機構に近い命令展開方式をするものとして、Disjoint Eager Execution(以下 DEE)[US95] として提案されているが、コントロールフロー先行展開と DEE では複数パスの中からフェッチすべきパスを選択する機構が根本的に異なる。簡単に言えば、コントロールフロー先行展開は動的な選択機構であるが、DEE は静的な選択機構である。表 10.2 はコントロールフロー先行展開と DEE の比較である。

- 各分岐命令に対する分岐予測性能は著しく異なることが多いため、DEE のようにプロファイル情報から得た平均分岐成功率という 1 つの値によってすべ

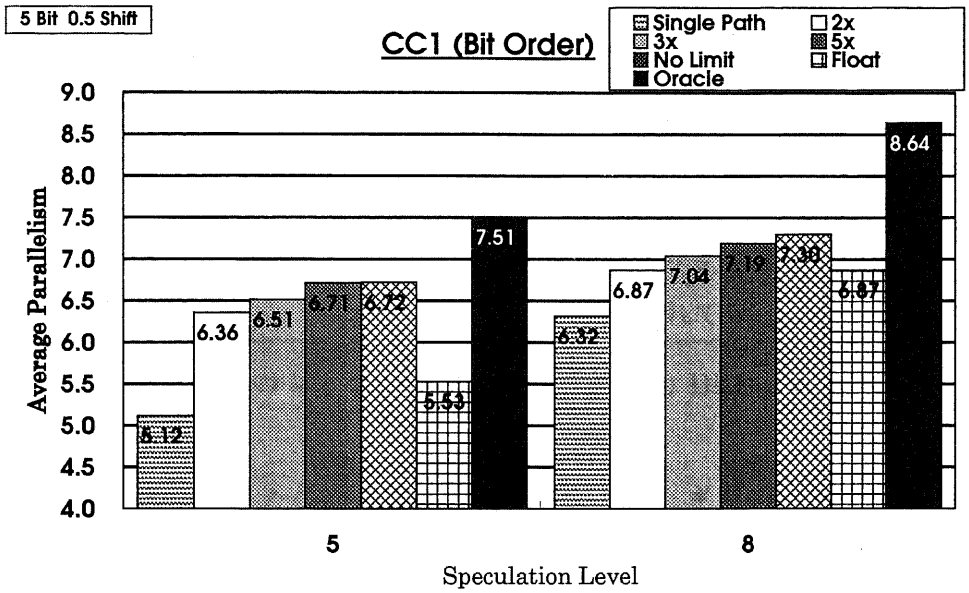


図 10.6: 様々な命令展開方式の平均並列度比較 (cc1)

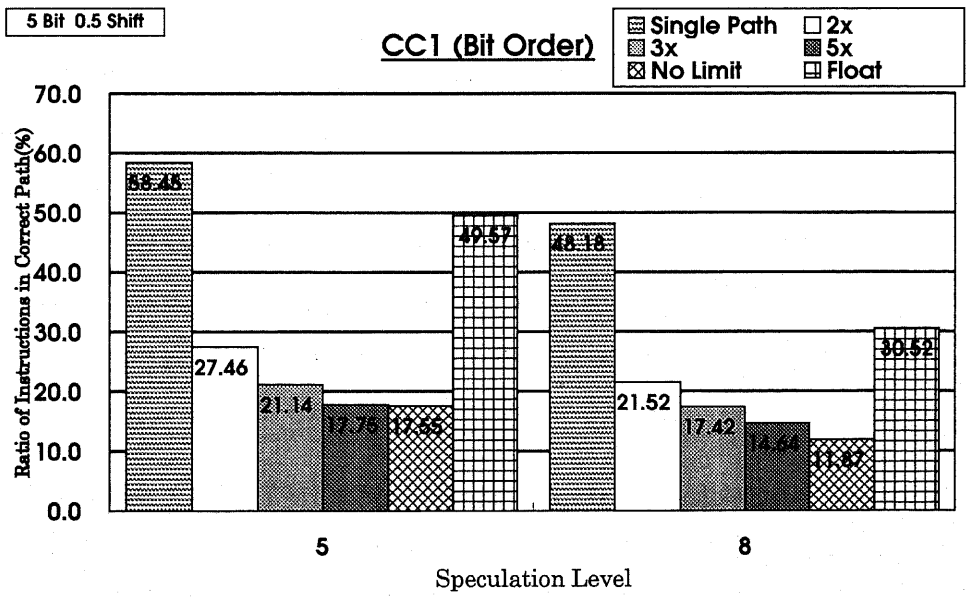


図 10.7: 様々な命令展開方式の有効命令率 (cc1)

	コントロールフロー先行展開	DEE
パスの選択	動的, 各パスの実行予測確率を動的に求め最も確率の (必要性の) 高いパスを選択する	静的, プロファイルを取り平均分岐予測成功率から静的に決めた命令展開方式で行なう
分岐予測	Multi BTAC のような大規模な投機フェッチにあわせた分岐予測機構を用いる	従来の分岐予測機構を利用しており、レジスタ間接アドレッシングの分岐命令に関して対応されていない

表 10.2: コントロールフロー先行展開と DEE の比較

ての分岐命令をまたぐ命令展開方式を規定してしまうのは不合理であると言える。

- 複数パスへの分岐の可能性があるレジスタ間接アドレッシングの分岐命令はほとんどのアーキテクチャに含まれる命令であり、その出現頻度も低くない。大規模な投機処理を行なう場合には、このような分岐命令に対する高性能な分岐予測は必要不可欠である。

以上の点で、DEE に比べてコントロールフロー先行展開は優れていると考えられる。DEE ではレジスタ間接アドレッシングの分岐命令に対する対応が不明であり定量的な比較は困難である。

10.1.3 Threaded Multiple Path Execution との比較

第 4.1.3節で述べた Threaded Multiple Path Execution (TME) は複数パスへの投機フェッチを行なう点でコントロールフロー先行展開と同様の機能をもつ。ただし、以下の点でコントロールフロー先行展開と異なる。

命令展開に制限がある TME では Primary Thread と Alternate Thread の 2 つでパスを区別しており、Alternate Thread がさらに Alternate Thread を作ることを許さない。そのため、命令の展開形は Primary Thread から横にのびるパスだけとなる。コントロールフロー先行展開では、必要なパスの展開を行なえるようになっており、今回の例ではバイナリ・ツリーであるという制限

を設けていたが、それなりの資源を使えば1箇所の分岐ポイントから複数のパスの展開を行なうこともできる。

次にフェッチすべきポイントの決定法 TMEでは、次にフェッチすべきポイントの決定において、それぞれの分岐命令が独立にもつbranch confidence counterというカウンタを使って行なう。さらにそのカウンタを使った選択方法はoldest / latest / next / lowestなどの静的な方法である。コントロールフロー先行展開では、命令のパスの情報を実行予測確率ビットとしてもち、分岐命令をまたぐ毎に近似的な計算で実行予測確率を上位から下位に伝搬させている。

つまり、TMEに比べてコントロールフロー先行展開は動的な情報を活用しており、より柔軟性があるといえる。性能に関する定量的な評価は、対象とする実行部が異なり、評価方法も違うため簡単には比較できないが、TMEではベースラインマシンに比べてTMEによる性能向上は14〜23 %であるとしている。コントロールフロー先行展開ではベースラインマシンをスカラープロセッサにしているため、6倍〜8倍の性能向上が得られている。ベースラインマシンを「理想的な実行部をもつSingle Path命令展開のマシン」とすると3〜30 %程度の性能向上率である。

10.1.4 フェッチ能力および実行部限界周波数

本節では第2.1節で示した今後のデバイス技術予測などをもとに、3つのモデルに対してコントロールフロー先行展開を使って命令フェッチをした場合の命令フェッチ能力および実行部の限界周波数について考察する。なお、本節では命令バッファのhit率は100 %を仮定し、実行部は第9.1節で述べたオラクル・モデルを仮定する。また、コントロールフロー先行展開パラメータは表10.3の通りとする。

投機レベル	5, 8
分岐履歴ビット数	5
実行確率管理ビット読み出し時シフト量	0.5
実行確率管理ビット更新方式	Bit Order
フェッチ命令数制限	最大投機レベル× 2,3,5 ベーシックブロック分 および無制限
分岐予測機構 (レジスタ間接アドレッシング用) (即値アドレッシング用)	非更新型 Multi BTAC 4way セットアソシアティブ, FIFO, 4,096 エントリ, 3-Level Multi BTAC 4way セットアソシアティブ, FIFO, 8,192 エントリ, 1-Level Multi BTAC
命令展開幅	2 方向

表 10.3: コントロールフロー先行展開パラメータ

メモリ～CPU 間のモデルとしては、次の 3 つのモデルを検討した。

(1)2001 年モデル 2001 年におけるデバイス技術の予測値に基づいて、バッファ～CPU 間の命令転送能力を仮定したモデル。転送幅 1024bit、転送クロック 1.5GHz(0.67ns) 動作を仮定。

(2)2012 年モデル (下限) 2012 年におけるデバイス技術の予測値 (下限) に基づいて、バッファ～CPU 間の命令転送能力を仮定したモデル。転送幅 4096bit、転送クロック 3.0GHz(0.33ns) 動作を仮定。

(3)2012 年モデル (上限) 2012 年におけるデバイス技術の予測値 (上限) に基づいて、バッファ～CPU 間の命令転送能力を仮定したモデル。転送幅 4096bit、転送クロック 10GHz(0.10ns) 動作を仮定。

図 10.8, 図 10.9, 図 10.10は各命令展開戦略ごとの必要なフェッチ能力を示したものである。ここでは、実行部の動作クロックを命令転送クロックと同じと仮定した。つまりバッファ～CPU 間のサイクル時間と実行部のサイクル時間が 1:1 であると仮定をした。よって、例えば2001 年モデルでは実行部の動作クロックは 1.5GHz

となる。各グラフ中の横軸は命令展開戦略を示しており、CFP はコントロールフロー先行展開を表し、(x2) や (x3) はそれぞれコントロールフロー先行展開のフェッチ命令制限モデルを示している。また、各グラフの左側の 7 本の棒グラフと右側の 7 本の棒グラフは、それぞれ投機レベルが 5 と 8 の場合を表している。

グラフを見ると、Single Path のフェッチ能力に対する要求はかなり低い水準になっている。これは、命令展開が 1 方向であるのでもともとフェッチ命令数が少なくてよい上に、実行部の平均並列度が低いため、単位時間当たりに必要な命令数がさらに減っているためである。Single Path と CFP(x2) モデルを比較すると、おおよそ必要な命令フェッチ能力が 2.5 倍程度違うことになる。

また、投機レベルが 5 の場合と 8 の場合を比較すると、どのモデルも 1.2〜1.8 倍程度命令フェッチ能力に対する要求が高くなっているのが分かる。オラクルモデルにおいても投機レベルが 5 と 8 の場合で 1.1 倍程度フェッチ能力に対する要求が高くなっているのは、平均命令並列度が上がるためである。

各モデルにおける転送能力の限界もあわせて示した。2001 年モデルでは、コントロールフロー先行展開の x3, x5, No Limit 戦略がこの限界を越えてしまっているが、x2 モデルまではこの限界内に収まっている。2012 年モデルになると、転送バンド幅が 4096bit になることが大きく効いて、各モデルの転送能力の限界が問題になることはなくなる。

次に、各モデルにおける最大フェッチ能力のグラフを示す。最大フェッチ能力とは無限に速い実行部があると仮定し、各モデルの命令転送能力を最大限利用して命令をフェッチした場合に、1秒間にどれだけの有効命令が供給できるかを示したものである。図10.11, 図10.12, 図10.13は最大フェッチ能力のグラフである。このグラフは、各メモリ～CPU間モデルにおいて各命令展開戦略で可能な最大の命令フェッチ・スループットを示している。つまり、オラクルモデルの値は、命令転送能力をすべて有効な命令の転送に使った場合であり、これが限界能力である。他のモデルとオラクルを比較すると、他のモデルが命令転送能力のどの程度の割合を有効な命令の転送に使っているかがわかる。

また、図10.8, 図10.9, 図10.10に示したupper limit of this modelの値はこのオラクルモデルにおける転送能力である。

ただし、この棒グラフが高ければ良いと言うわけではない。例えばSingle Pathでは、有効な命令で全転送能力の約半分を使うことができるが、実際には平均並列度がそれほど取り出せないので、これほどの転送能力を必要とすることはない。これは図10.8, 図10.9, 図10.10 を見るとわかる。

図 10.14, 図 10.15, 図 10.16は各モデルの限界周波数のグラフである。ここで限界周波数とは、理想的な実行部があると仮定した場合に、各命令展開戦略によって駆動できる実行部の限界動作周波数を示している。つまり、実行部の動作周波数がこの値を越えると、フェッチ能力が不足して十分な命令供給が行なえなくなることを意味する。

各グラフには、実行部の動作クロックが各モデルの命令転送クロックと同じ場合を点線で示した。一つの基準がこの線よりも上にあることである。例えば2001年モデルでは、命令転送クロックが1.5GHz であるので、実行部の動作クロックも1.5GHz であると仮定すると、コントロールフロー先行展開のx2 モデルは約 2GHz の動作クロックの実行部に必要な命令フェッチ能力を持っているので、1.5GHz の動作クロックならば、命令フェッチ能力の不足が原因でプロセッサ性能を落す可能性は少ないことになる。2012 年モデルでは、いずれの命令展開戦略においても十分な命令フェッチ能力があることが分かる。

10.1.5 命令ストリーミング

コントロールフロー先行展開をサポートするソフトウェア技術についても考察を行う。コントロールフロー先行展開では、大規模な命令先行展開を行なうため、メモリとの間に高スループットで広いバンド幅をもったバスを必要とする。

ところが、バンド幅については、off chip のメモリデバイスを使う場合、今後も大幅な向上は望めない。現在、これを解決するための研究が様々行なわれている [DB97]。例えば、PPRAM[村上 94] では、Logic と DRAM を 1 チップにインテグレートすることで、メモリ・バンド幅を稼ぐことが可能である。DRAM の混載に限らず、大規模化するデバイスを大容量のオンチップ・キャッシュに利用するアプローチも同じである。

しかし、いずれにしてもいずれにしても、メモリ・アクセス・レイテンシの問題が生じる可能性がある。DRAM の場合は構造上レイテンシが大きく、大容量のオンチップ・キャッシュでは、大容量化に伴う複雑なアドレス変換によってこれまでのような 1 サイクル・アクセスが実現できるかどうかは疑問である。ラインサイズを大きくしてこれを回避することは可能であるが、この場合キャッシュミスが増大する恐れがある。

一方、スループットに関しては、今後の向上が期待できる。例えば Direct RAMBUS を利用すれば off chip でも 1.6GByte/sec のスループットが得られる [Cri97]。

現在でもSDRAMなどのように、最初のアクセスを除いて1サイクル・アクセスが可能なメモリもあり、今後もこの傾向は続くと考えられる。

つまり、コントロールフロー先行展開を使った命令フェッチにおいても、メモリへのアクセス・レイテインシを隠蔽するために、高いスループットを使って命令をバースト転送することが有効であることが分かる。

通常の命令フェッチでは図6.1のSingle Pathのような命令展開が行なわれることが多いため、コンパイラによってうまくコードを生成することができれば、命令フェッチにおけるメモリ・アクセスは逐次的になる。しかし、コントロールフロー先行展開のようにコントロールフロー・ツリーの各部分を実行される確率の高い順にフェッチするような場合には、命令フェッチにおけるメモリ・アクセスはメモリの各所に分散し、バースト転送可能なサイズもベーシック・ブロック程度(通常5命令程度)にとどまり、バースト転送には向かない可能性が高い。

そこでバースト転送に向いたコードを考える。図10.17はその一例である。図10.17はコントロールフロー・グラフを表しており、それぞれの網掛け部分が命令ストリーム・コードである。この網掛け部分の命令を1セットの命令流としてまとめておくことで、大きなバースト転送が可能となる。

命令ストリームの作り方は様々考えられるが、1つの方法はプロファイルを利用し、静的にコントロールフロー先行展開と同様の処理をして実行予測確率の高いフローを1セットにしていくことである。その他にも、コンパイラによるソースコードの情報を用いた最適化戦略も考えられる。

ここでは、プロファイルを用いてストリームコードを作り、メモリ・アクセス回数の減少を調べた。ここでは、1ストリームのフェッチによって、実際に実行された命令数(有効命令数)を示す。なお、各パラメータを次のように設定した。

- 1ストリームのサイズは1024Byte以内(最大256命令に相当)
- 1000万命令のトレースデータをプロファイル・データとして利用
- パスごとの実行予測確率は浮動小数点計算

この結果は表10.4の通りである。なお、この評価については、SPEC92intから5プログラムを使っている。

命令ストリームに関しては、今後様々な展開が期待できる。ストリームの作り方もここであげた方法は一例にすぎず、より良い方法の検討が必要であろう。

サンプルプログラム	有効命令数
cc1	23.2
compress	52.3
dhrystone	40.1
espresso	40.5
sc	25.6

表 10.4: 1 ストリームのフェッチでの有効フェッチ命令数

10.2 大規模データパス・アーキテクチャに関する考察

前節では、本論文で詳しく述べたコントロールフロー先行展開についての考察を行なったが、本節では、大規模データパス・アーキテクチャその他の部分および全体について考察を行ない、今後の課題について述べる。

10.2.1 データパス先行展開

データパス先行展開では、データ依存関係の抽出をもとにデータ転送および命令の ALU-NET へのマッピングを行なうのが主な役割である。図 5.2に示した VLDP ブロック図を見ると分かるように、データパス先行展開への入力は、命令と実行パス制御情報である。このうちコントロールフロー先行展開で作り出される実行パス制御情報とは、実行が進み分岐結果によって必要な命令と不必要な命令の区別を行なう際に参照される情報であるが、データパス先行展開では、さらにこの情報を使うことで制御依存関係の解消された命令列の集合を見つけ出すことができる。なぜならば、各々の命令がどの制御パスに乗っているかを示す識別子が実行パス制御情報であるからで、この識別子が異なる、つまり別々のパス上にある命令はある条件においてデータ依存関係がないとして扱える。この条件とは、次の 2 点である。

制御の合流を認めない コントロールフロー先行展開において (実行パス制御情報において) 制御の合流を認めない。これは、プログラムの論理構造上で制御の合流を認めないわけではない。コントロールフロー先行展開の命令展開において、例え制御の合流があってもそれぞれの合流先を別々のパスとして (複製して) 命令展開を行なうことを意味している。これによって一度異なる分

岐先のパスに乗った命令間でデータの交換を行なうことはなくなる。よってデータ依存関係の解析を行なうべき命令は親子関係にある命令だけでよい。制御の合流を認めなければ、コントロールフロー先行展開における命令展開も簡単化される。ただし、制御の合流がある場合には、少なくとも合流先のパスに関して、2つ以上の複製が作られることになるので、命令の展開効率は落ちる。ただし、コンパイラ・サポートによりこの影響は低く押えることが可能である。制御の分岐・合流が近いようなコントロールフロー・ツリーの場合には、第4.1.2節で述べた条件付き実行を利用すればよい。つまり制御の分岐・合流をセットで削除してしまうことで、この問題はなくなる。一方、制御の分岐・合流が遠い場合には、合流先の命令が投機フェッチされるまでにその制御の分岐方向が確定している可能性が高い。以上のような考察から、制御の合流を認める必要はない。

メモリ・コンシステンシを保つ機構を用意する 各パスごとに別々のレジスタセットおよび独立なストアバッファを設けることが必要である。分岐ポイントごとにレジスタセットのコピーを作ったのでは、ハードウェアコストがかかるばかりでなく、処理のオーバーヘッドが大きくなりすぎる。そこで必要に応じてリネームレジスタを利用することになる。リネームには実行パス制御情報を付加することで、レジスタにも実行パス制御情報をつけるとよいであろう。これは、分岐結果の確定によって削除されるのは投機的にフェッチされた命令だけでなく、ある意味投機的に割り当てられた(もしくは投機的にデータが格納された)レジスタの解放も行なう必要があるからである。このようにレジスタに色をつけることを考えると、レジスタの管理および物理的なレイアウトについてもある程度集中制御が必要でなくなることが分かる。りネームレジスタの解放はコントロールフロー先行展開から送られてくる実行パス制御情報を何らかの手段でレジスタに送れば、そのレジスタの色が不要であると通知されたときにそのレジスタ自身で解放が行なえる[1]。またレジスタに色がつくことで、そのレジスタにアクセスする命令が限られることになる。よってレジスタを物理的に分散配置して利用することも考えられる。これによってハードウェアの設計上、レジスタがクリティカルパスになる可能性を低く押えられることになる。メモリのストアバッファに関してもレジスタと同様である。ストアバッファをパスごとに設けることでメモリを介したデー

[1]ただし、空きレジスタのリストを管理する部分は集中制御が必要であろう。

タ依存関係を正しく保てる。

このように、データ依存解析に関しては、いかに実行パス制御情報を利用するかが今後の課題である。また例外回復機構もこの実行パス制御情報を利用しておこなう重要な処理である。

データ転送の効率化に関しては、データの生存期間 (life) に関する調査が必要であろう。一般に命令列のデコードによりデータの誕生に関してはその時間が確定できるが、データが不必要になる時間については簡単に確定できない。命令列のデコードから分かることは、データの消費時間および消滅 (上書きされること) 時間だけである。つまり、データの消費・消滅時間が確定されるまでの間、そのデータはいつまで必要であるかを確定できない状態にあることになる。プログラム中には一時変数など、データの生存時間が短いものが数多く存在する。このようなデータについて、データの生存期間を明示的に与えることは、VLDP の実行部である ALU-NET においては大変に有用である。なぜならば、データが ALU-NET 上で転送され、レジスタやメモリに転送する必要がなくなるためである。データの生存期間の与え方として、データの誕生の際にそれを与える場合には、命令列で何命令分というような生存期間の与え方が適当であろう。サイクル数で与えるのは、割り込みや計算資源の関係でサイクル数が伸びる可能性を考えると危険であるし、例えば、2 回参照された不必要になるというような参照される回数で生存期間を与える方法では、VLDP のような複数パスへの投機フェッチを行なう場合にはやはり危険である。生存期間は必ずしもデータの誕生時に与えなくてもよい。データの消費時に、この命令が「最後の消費者」であることを示すビットがあってもよい。この方法の利点は生存期間がそれほど短くないデータに関しても生存期間を与えることができる点である。ただし、この方法では生存期間を与えられない場合がある。例えば、次の分岐命令で A 方向に分岐すれば現在の命令が「最後の消費者」であるが、B 方向に分岐するとまだ「別の消費者」がいるような場合である。現在の消費者が「最後の消費者」であると宣言してしまうと B 方向に分岐した場合にデータが不足するし、宣言をしないと A 方向に分岐した場合にこのデータは本来の生存期間を越えて生存することになる。現在 short-lived(短命) 変数の検出については、研究が勧められているところであるが、変数の生存期間の利用方法、情報の付加方法などについては今後の課題である。

命令の演算器へのマッピングに関しては、多くの課題がある。これは次節で考察する ALU-NET の構成方法にも大きく依存しており、特にマッピングのアルゴ

リズムは ALU-NET の構成によってさまざま検討されるべきであろう。マッピング・アルゴリズム以外にも、データパス先行展開では資源管理の問題がある。マッピング・アルゴリズムと資源の管理は互いに深く関係している。マッピング・アルゴリズムにおいては、いくつかの最適化基準が考えられる。例えば、可能なかぎり ALU-NET 内でデータ転送が行なわれるようなマッピングであったり、可能な限り深くもしくは幅広くマッピングすることであったりする。しかし、このようなマッピングの自由度を保つためには、ALU-NET 上にある程度の空き演算器のかたまりが必要である。よって資源管理においては、空き演算器のガーベッジコレクションのような機能が必要になるかも知れない。もしくは、マッピングの段階で ALU-NET のスラッシングが起こらないような工夫が必要かも知れない。

10.2.2 ALU-NET

ALU-NET は VLDP の特徴の 1 つであり、スーパースカラプロセッサなどと一線を画す機構である。ALU-NET の 1 つの特徴は、横方向だけでなく縦方向にも演算器をつなげることができることである。例えば、この直列接続の演算器を使って非同期的動作をさせることで処理時間の短縮などが得られると考えられる。しかしより重要なのは、大規模なデータパスの制御の複雑さを低減する能力がある点である。これは、演算器を含むデータパスの制御が分散化できるためである。データパス先行展開によってデータ依存解析された命令列は ALU-NET にマッピングされる。このとき直接接続の演算器の後段の演算器はその入力データの到着によって計算を開始する。よって実行を管理・監視する側 (データパス先行展開) では、命令のマッピングが終れば、最終的な出力が出るのを待てば良く、中間状態での制御は命令のマッピング時にすでに終っている。演算器資源にある程度の余裕があると考えれば、キャッシュミスなどによりデータの到着が遅れた場合でも、特別な操作は必要でなく、ただ同じように最終的な出力だけを監視していればよい。

ALU-NET の設計においては、考えるべき点が多く存在する。ALU-NET の構成として理想的なのは、完全結合網ではあるが、現実的には不可能であるため、実装において ALU-NET の自由度をどの程度にするかは非常に重要な問題である。この設計はデータパス先行展開における命令のマッピング・アルゴリズムにも大きな影響を与える。

まず考えるべきことは、ALU-NET の各演算器としてどのようなものを用意すべきか、またその組合せはどのようなものが良いのか、という基礎調査である。

これは最近のマルチメディア拡張命令などの設計に思想的には近いものがあるが、対象とすべきはVLDPプロセッサが対象とするプログラムすべてである。幅広くALU-NETのあるべき姿を調査することは、最終的な実行部がALU-NETであり、コントロールフロー先行展開やデータパス先行展開を生かすも殺すもALU-NETであることを考えれば決して無駄なことではない。

ALU-NETの実際の設計においては、演算器間のデータ線だけを考えていたのでは、設計が破綻する恐れがある。例えば、ALU-NETの各ノードである演算器には加算器・乗算器などの複数の基本的な回路があると思われ、この選択を行なうためには制御線が必要であるし、空き演算器資源の管理のための制御線も必要である。更に重要なのは、外部データとのデータ線である。図5.4に示したように、投機レベルがあがるにつれて演算器間でのローカルなデータ転送が「割合的には」増えるのは確かであるが、外部とのデータ転送も相当な量存在する。ALU-NETの大規模化を考慮すれば、外部とのデータ転送可能な演算器の数を限定するのがよいと思われる。これはメモリシステムの設計の問題でもある。レジスタの扱いをどのようにするかなど、データ転送に関する研究は今後の重要課題である。

10.2.3 今後のVLDP

最後に、今後のVLDPについてより大きな視点から述べる。

今日のプロセッサ市場を見ると、個人的には5極化しているように思われる。かなり区分レベルが異なっているが、大型計算機用プロセッサ、組み込み用ASIC、メディアプロセッサ、省電力プロセッサ、そして汎用プロセッサである。

大型計算機用プロセッサ ダウンサイジングにより、その相対的な市場規模は減少したが、今後ともある一定の量を占める。

組み込み用ASIC 工場機械や測定機などに利用される。カスタマイズが容易な構造にすることが必要で、ある程度のモジュールのライブラリができている場合が多い。十分強力で安価な汎用プロセッサができれば置き換えられる可能性もあるが、特にリアルタイム処理などの分野でまだしばらくはある程度の市場規模を保つと思われる。

メディアプロセッサ 最近になって非常に注目を集めているプロセッサであるが、実際には汎用プロセッサにDSP的要素を追加、もしくはその逆の経緯をもつプロセッサである。ここでは、DSPもメディアプロセッサの一種であると

する。今後さらなるメディアのデジタル化が進むと考えられており、市場規模を拡大すると思われる。最近では情報家電やゲーム器などに搭載され、数量的には相当な数になっている。

省電力プロセッサ 一般に省電力プロセッサといえば、省電力の他に小型・軽量・低コストである場合が多い。省電力プロセッサの用途としては携帯機器が多いためで、ある程度性能を犠牲にしても省電力・小型・軽量・低コストを求められることが多い。

汎用プロセッサ ワークステーションやＰＣなどで利用されているプロセッサ。事実上の寡占状態である。汎用プロセッサという分類ではあるが、最近ではサーバー機でも利用され、組み込み用途でもある程度のシェアをもつようになった。またメディアプロセッサがもつような特殊命令も一部取り込んでおり、省電力版の開発も行なわれることが多くなった。

それぞれに固有の用途ができつつあり、今後もしばらくはこのような状況が続くと思われる。さて、このようなプロセッサ市場において、VLDP のターゲットは汎用プロセッサの中核である。組み込み用 ASIC やメディアプロセッサ、省電力プロセッサはそれぞれ性能に対する要求水準が現状程度で均衡している部分もあるが、汎用プロセッサに関しては、大規模シミュレーションや CAD, CG などその性能要求はとどまるところを知らない。

VLDP の実現には、まだ多くの壁が残っている。データパス先行展開や ALU-NET の設計やメモリシステムの検討である。しかし、これらの設計に関してはこれまでに多くのアイディアが出され、方向性は固まってきたように思う。次に着手すべきはコンパイラの検討である。本論文でも、「ソフトウェアサポート」や「コンパイラによるサポート」が必要であると述べてきたが、ここでコンパイラ・サポートについていくつか述べる。

- 命令ストリーミングは是非とも取り組むべきである。ストリームの構成方法は非常に多く考えられる。例えば、複数パスの命令をどのように１つのストリームに格納するかという問題１つにしても、色々とバリエーションがあるであろう。究極的な命令ストリーミングの形としては、動的な命令ストリーミングであるかもしれない。トレースキャッシュにその基本的なアイディアはある。これにコントロールフロー先行展開やデータパス先行展開での解析結果を付加した形で動的に命令ストリーミングを行なえば、よい命令ストリー

ムができると考えられる。もう一度、コンパイラ・サポートに戻ると、命令ストリーミングとは静的なコントロールフロー先行展開であり、静的なデータパス先行展開であるとも言える。

- 静的に行なえるデータ依存解析はできるだけ解析結果をコードに反映させるべきである。命令ストリーミングにデータ依存解析結果を反映させることで、データパス先行展開の処理を軽くすることが必要であろう。VLIW の研究にそのアイディアがある。VLIW よりも柔軟性のある表現方法を確率するとよい。ALU-NET にもそれだけの柔軟性が備わっている (ように設計すべき)。

- データ転送に対するサポートは大変に重要である。メモリを介したデータ転送に対して、動的なデータ依存解析が容易となるようなサポートはできないであろうか。またデータの生存期間をできるだけ示すことは、ALU-NET の有効利用には絶対に必要である。データプレディクションについては、その方法論もこれからの研究課題であるが、VLDP 上でどのように実装可能かを考えるとよい研究になるかもしれない。

最後に私見であるが、ハードウェア規模などを含めた VLDP のロードマップを示しておく。

1998–2000 : 0.25〜0.18μm : 28〜64MTr./chip 10〜30ALUs 程度の規模の ALU-NET を構成可能と思われる。小規模な VLDP プロセッサの詳細なシミュレータを作成するが、この段階での性能向上は大きくないと思われる。まず、サポートするコンパイラがなく、大規模な命令の投機的展開についても、この程度の ALU-NET の規模では、十分な処理規模とは言えない。また、この段階では、コントロールフロー先行展開／データパス先行展開部の詳細設計を行なっていると思われる。この研究において、より効率的な命令展開の方式や、スケーラビリティをもった ALU-NET の実装方式、ALU-NET への命令マッピング・アルゴリズムなどが提案される。またコンパイラ・サポートに関しては、その可能性について述べられるであろう。

2001-2004 : 0.13μm : 150MTr./chip 100ALUs 規模の ALU-NET を構成可能と思われる。この規模では、ILP(Instruction Level Parallelism) として、5〜10 程度が得られる。クロック周波数が 1GHz 超と考えて、現在のプロセッサ性能の約 10 倍程度が得られると思われる。この規模の ALU-NET になると、

命令およびデータのフェッチ能力が現状のままでは相当不足するため、フェッチ能力向上のための手法を確立する必要がある。1つはメモリのon-chip化であるが、これはALU-NET規模が大幅に縮小されない範囲に限られる。メモリデバイスの最近の傾向などから考えると、レイテンシの短縮はあまり期待できないが、スループットの向上は期待できるため、連続転送に適したフェッチ方式によってある程度のメモリネックは解消できる。データ転送については画期的な方法が見つかるとよいが、小さなコンパイラ・サポートの積み重ねで性能を稼ぐ方向であろう。

2004-2007：0.10μm：350MTr./chip 200～300ALUs規模のALU-NETを構成可能と思われる。ILPは現在の検討ではコンパイラ・サポートを含めて10以上が可能であると考えている。動作クロックの向上もあわせて現在のプロセッサ性能の10倍以上の性能が得られると考える。さらに、最適化コンパイラにより、より大きなレベルの並列性利用、VLDPプロセッサを要素プロセッサとした並列計算機により、100倍以上の性能向上が見込まれる。ただし、これだけ大規模なALU-NETになると、ALU-NET内の設計が非常に困難になり、ALU-NETの構成の自由度がより制限されることが十分に考えられるため、ALU-NETの規模に完全に比例して性能が得られるとは限らなくなる。いずれかの規模のALU-NETで性能向上が飽和する可能性も相当ある。

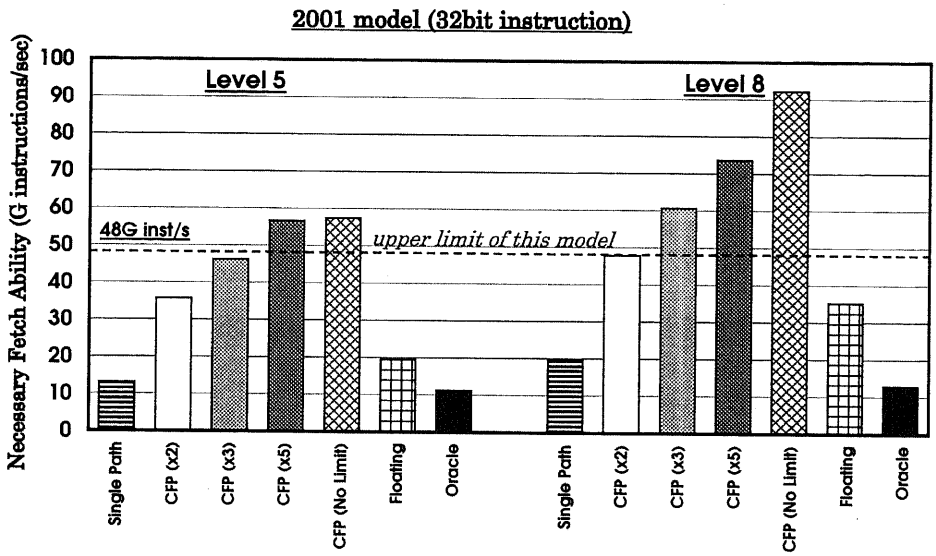


図 10.8: 必要フェッチ能力 (2001 年モデル)

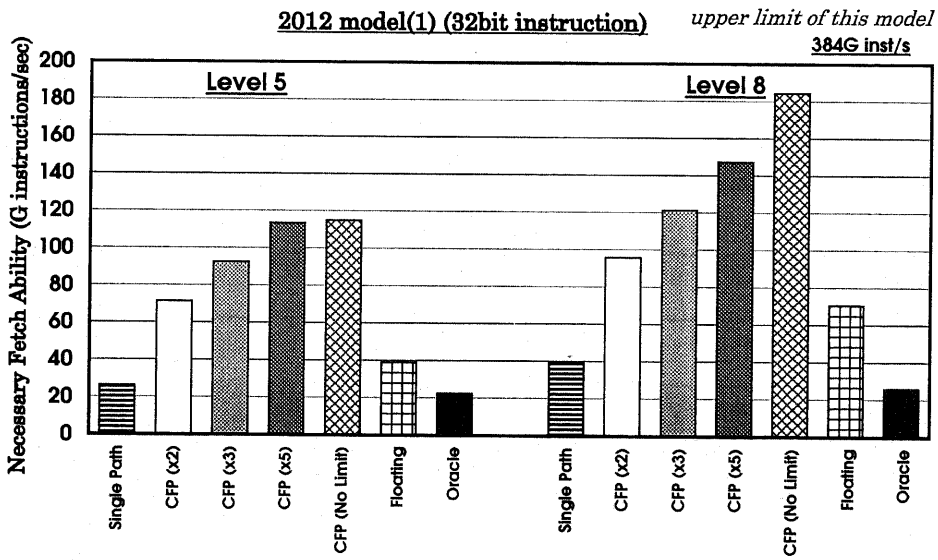


図 10.9: 必要フェッチ能力 (2012 年モデル, 下限)

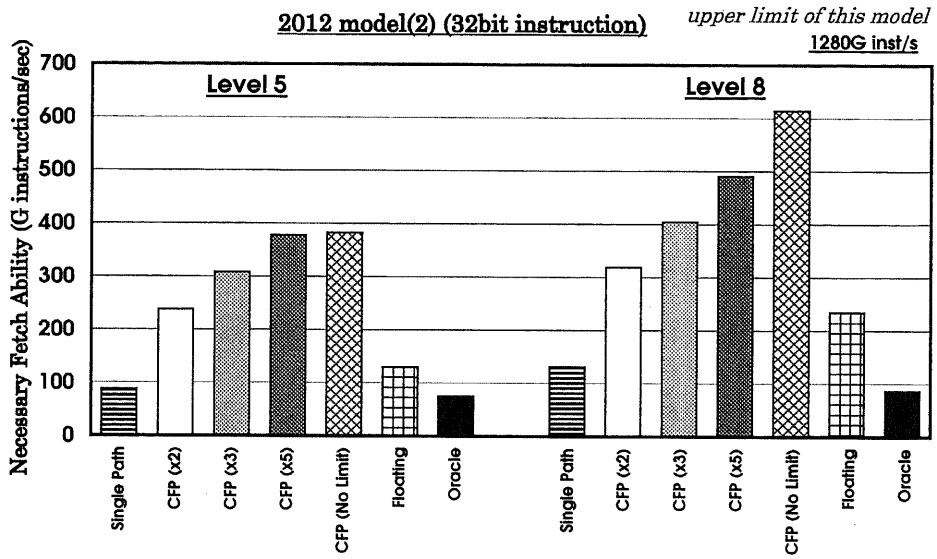


図 10.10: 必要フェッチ能力 (2012 年モデル, 上限)

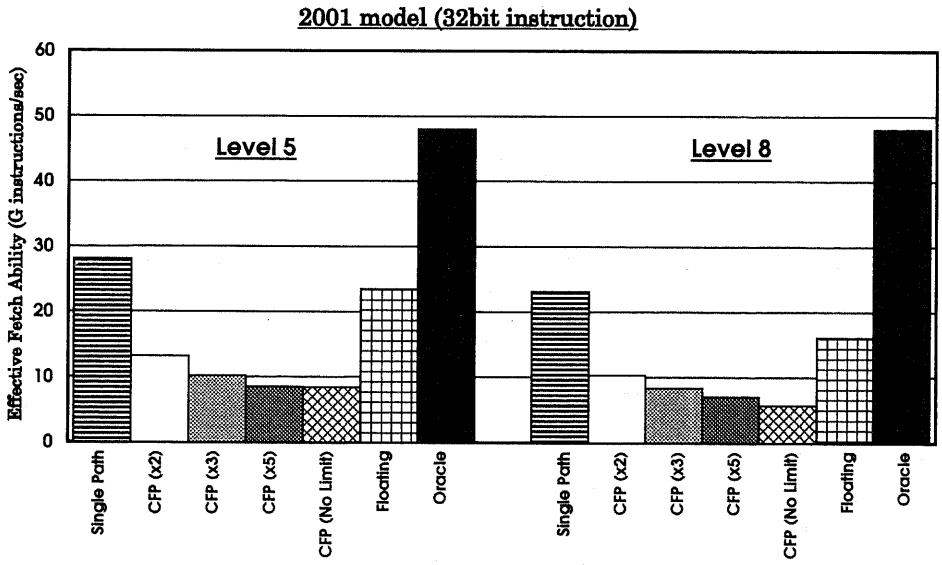


図 10.11: 最大フェッチ能力 (2001 年モデル)

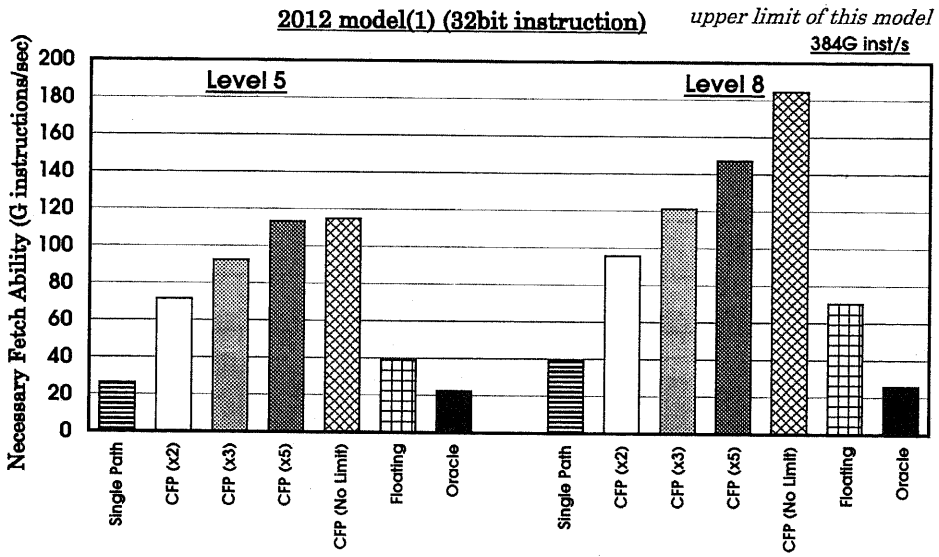


図 10.12: 最大フェッチ能力 (2012 年モデル, 下限)

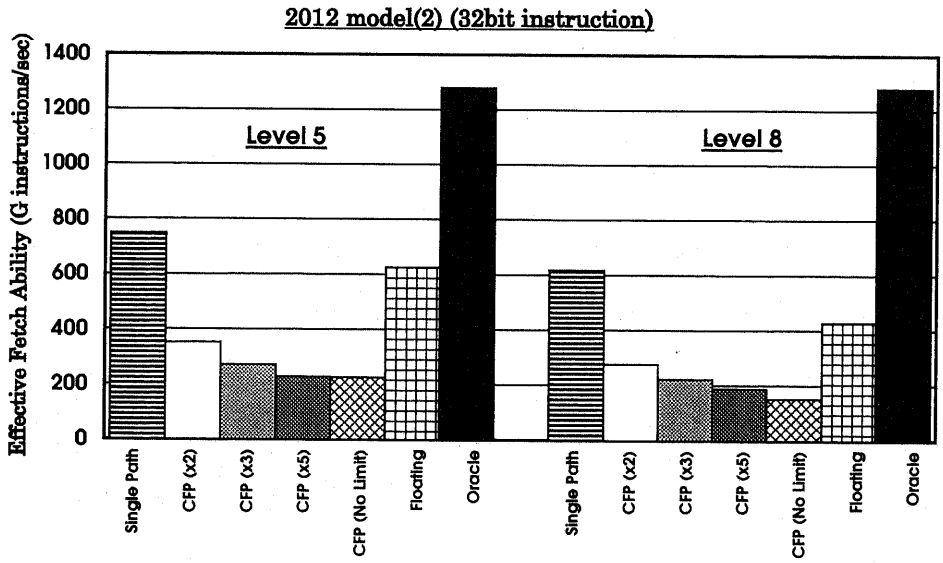


図 10.13: 最大フェッチ能力 (2012 年モデル, 上限)

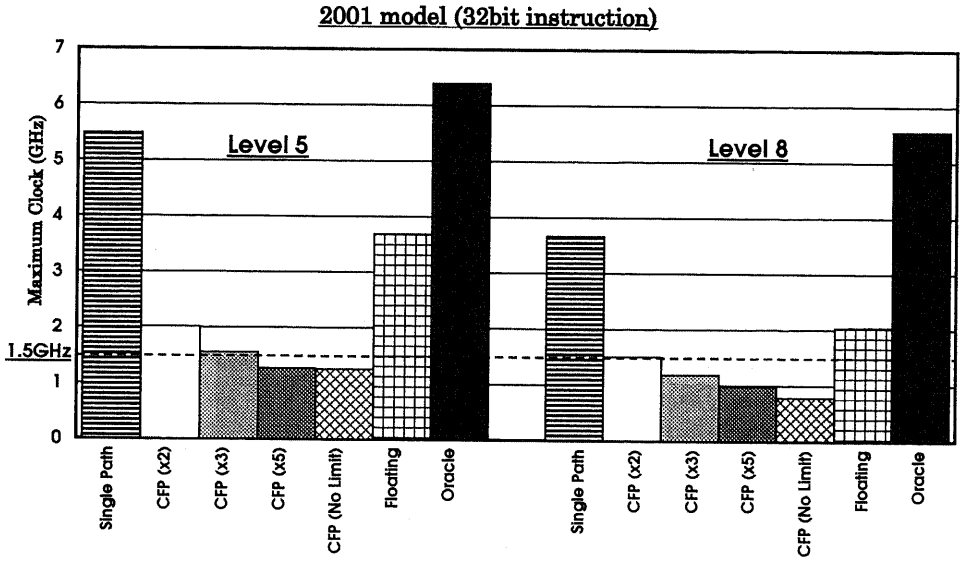


図 10.14: 限界周波数 (2001 年モデル)

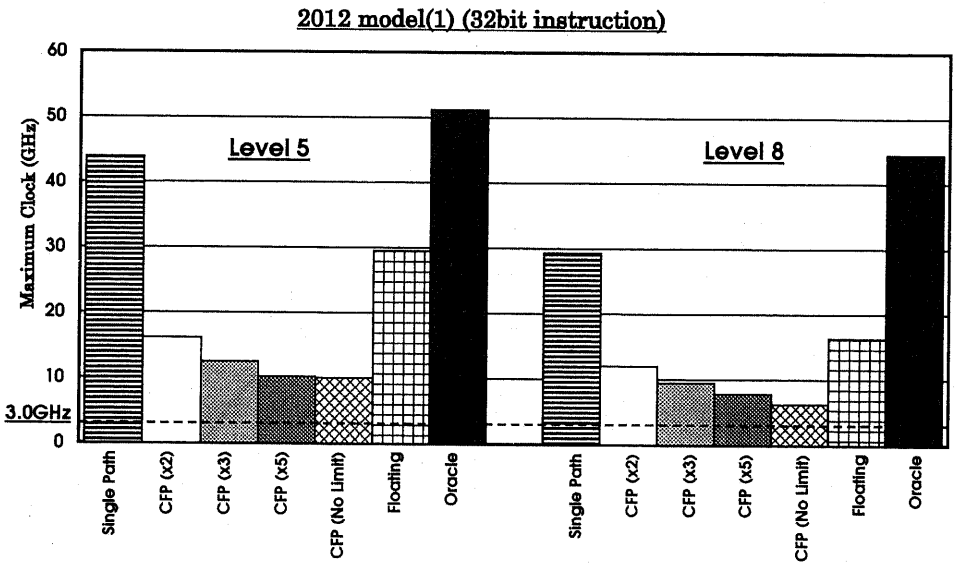


図 10.15: 限界周波数 (2012 年モデル, 下限)

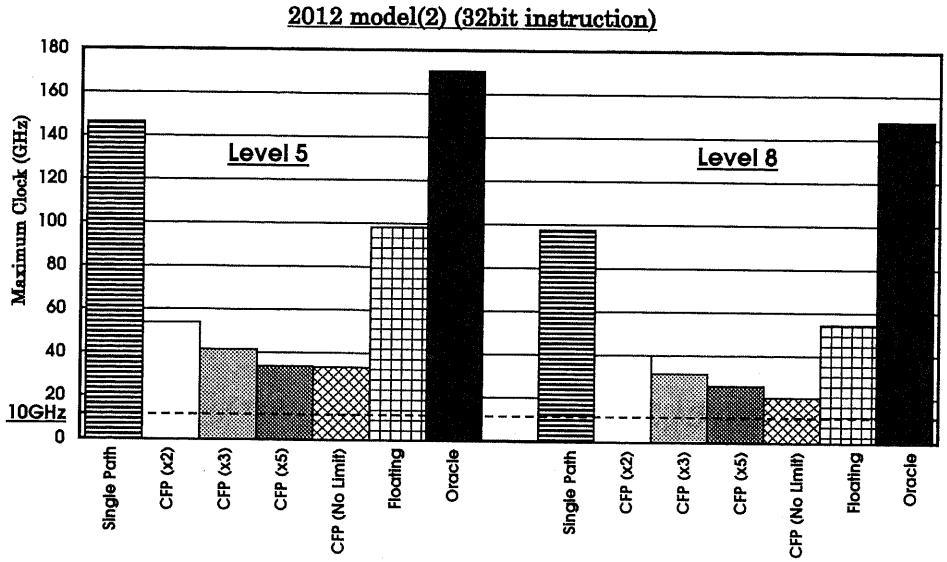


図 10.16: 限界周波数 (2012 年モデル, 上限)

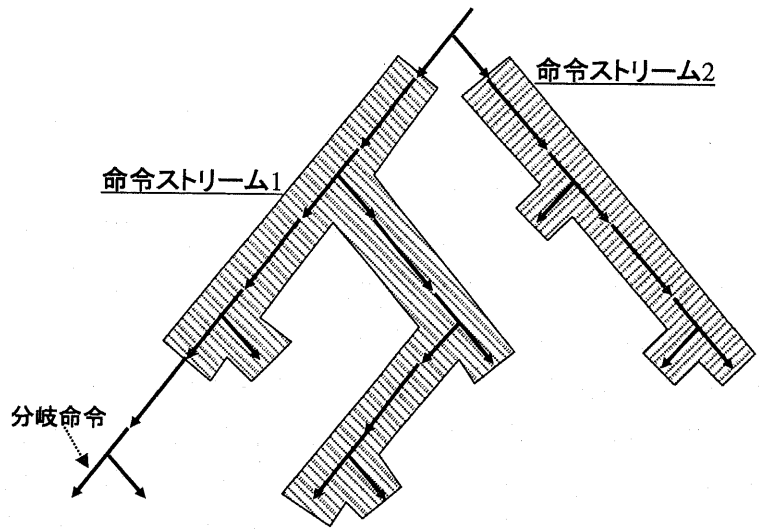


図 10.17: 命令ストリームの例

第 11 章

結論

本論文では、新しいプロセッサ・アーキテクチャとして大規模データパス (VLDP)・アーキテクチャの提案を行ない、その概要を説明した。また、VLDP アーキテクチャの特徴の 1 つである大規模投機処理のための命令フェッチ機構である、コントロールフロー先行展開の詳細について述べた。

コントロールフロー先行展開に用いる大規模投機フェッチに適した分岐予測機構の検討では、投機フェッチに伴う履歴情報の更新遅れ、複数パスへの分岐予測の 2 点について調査をおこなった。

動的な分岐履歴情報を利用する分岐予測機構では、投機フェッチによって先行する分岐命令の分岐予測を行なう場合、古い履歴情報をもとに分岐予測を行なうことになるため、この影響について調査を行なった。この結果、10 段程度先までの分岐命令に対する予測性能に対して大きな影響は見られなかった。

また、レジスタ間接アドレッシングで分岐先が決まる分岐命令は、その分岐先が 2 つ以上になる可能性をもっているため、複数パスへの投機フェッチに際しては、この複数の可能性の中から、確率の高い複数のパスの候補を取り出す分岐予測機構が有効であると考えられる。本研究では、1 つの分岐先をキャッシングする BTAC(Branch Target Address Cache) を拡張し、複数の分岐先をキャッシングする Multi BTAC を提案し、評価を行なった。その結果従来の BTAC に比べて、分岐予測成功率で 10～30 %程度の向上が見られ、さらに複数パスへの投機フェッチも Multi BTAC を使って可能になった。

そして、以上のような分岐予測機構を用いて複数パスへの選択的投機フェッチを行なうコントロールフロー先行展開について評価を行なった。その結果、プログラムや投機規模にもよるが、シングルパスへの投機フェッチに比べて 1.5 倍～6 倍程

度の命令フェッチ量で、シングルパス・モデルで得られる平均並列度とオラクルモデルで得られる平均並列度の中間程度の並列度が得られることを確認した。全体として、理想的にはスカラプロセッサの6倍～8倍の速度向上が可能であることを示した。

謝辞

本研究について至らぬ筆者を親身に指導して下さり、また多くの大変に有益な助言を頂いた田中英彦教授に心から深く感謝致します。研究面のみならず、生活面でも研究室の環境整備や、さまざまな書類・推薦状などの作成など、お忙しいところを御尽力頂きましたことに対しても心から御礼申し上げます。田中英彦研究室に来て早 6 年、長いようで短い 6 年間でした。それもひとえに田中教授のお人柄と研究室運営のおかげだと思っております。今後とも様々な場面でお世話になることがあると思いますが、どうぞよろしくお願い致します。

また博士 2 年次から坂井修一助教授には研究について具体的な御助言を多数頂きました。大変に経験豊富で博学な坂井助教授の御助言は大変に貴重でした。心から感謝致します。

佐藤充さん ((株) 富士通研究所) には、研究面のみならずさまざまな面で大変にお世話になりました。豊富な経験から様々な御助言を頂きました。

同学年の荒木拓也君 (情報工学専攻 博士課程 3 年)、渡辺正泰君 (情報工学専攻 博士課程 3 年) には、日常的によい相談相手として、色々とお世話になりました。また同学年の井手一郎君 (情報工学専攻 博士課程 3 年) にも、さまざまな面でお世話になりました。

筆者は VLDP プロジェクトの一員として研究を進めていますが、このプロジェクトのメンバである吉瀬謙二君 (情報工学専攻 博士課程 2 年)、辻秀典君 (情報工学専攻 博士課程 1 年)、安島雄一郎君 (情報工学専攻 修士課程 2 年)、高峰信君 (電気工学専攻 修士課程 2 年) には研究の細部に渡って議論を重ねることができ、有益な意見を多く頂きました。VLDP プロジェクトを進めてこれたのも、みなさまのお蔭です。今後ともよろしくお願い致します。

また馬場恒彦君 (電気工学専攻 博士課程 2 年)、滝田裕君 (情報工学専攻 博士課程 1 年) には、日常面で色々とお世話になりました。筆者の知らない世界について色々と勉強させて頂きました。

近山研究室の方々には、筆者に近山研究室の計算機資源をお貸し頂きました。多くのシミュレーションを行なえたのも近山研究室の方々のおかげです。特に大内拓実君 (情報工学専攻 博士課程 2 年) には、筆者のために様々な便宜をはかって頂きました。ここに感謝いたします。

(株) 日立製作所 中央研究所 の方々には、今後の研究活動についてご相談にのって頂きました。卒業後も色々な面でお世話になることと思います。今後ともよろしくお願い致します。

その他、名前を挙げればきりがありませんが、研究室のメンバ、そして OB の方にはさまざまな面でお世話になりました。個々に述べられないのは恐縮しますが、ここに感謝致します。

日本学術振興会からは経済的な支援を賜わり、筆者の生活にゆとりを与えて頂きました。大変お世話になりました。ここに深く感謝致します。

最後になりましたが、いつも筆者の我儘を通して、さまざまな面で支えとなってくれた両親 (中村尚之, 中村清子) に、そして婚約者 (押上朋子) に感謝致します。

1998 年 12 月 18 日

発表文献

[1] 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 田中英彦. 大規模データパスプロセッサの構想. 情処研究会 ARCH 124, Vol. 97, No. 61, pp. 13–18, Jun 1997.

[2] 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 高峰信, 田中英彦. 大規模データパスプロセッサにおける命令フェッチ機構. 電子情報通信学会集積回路研究会 (ICD), コンピュータシステム研究会 (CPSY), フォールトトレラントシステム研究会 (FTS) 合同研究会, 信学技法, Vol. 98, No. 23, pp. 93–100, Apr 1998.

[3] 中村友洋, 吉瀬謙二, 金指和幸, 田中英彦. トレース・ドリブン・シミュレーションによる分岐予測機構の検討. 第 51 回 情処全国大会, Vol. 6, No. 4P-7, pp. 13–14, Sep 1995.

[4] 中村友洋, 吉瀬謙二, 金指和幸, 田中英彦. プログラム解析に基づく分岐予測機構に関する問題点の検討. 第 52 回 情処全国大会, Vol. 6, No. 3K-5, pp. 47–48, Mar 1996.

[5] 中村友洋, 吉瀬謙二, 辻秀典, 田中英彦. 明示的な分岐制御の可能性の検討および制御機構の提案. 第 53 回 情処全国大会, Vol. 1, No. 4F-2, pp. 115–116, Sep 1996.

[6] 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 田中英彦. 分岐アドレス予測機構の比較検討. 第 55 回 情処全国大会, Vol. 1, No. 3F-5, pp. 20–21, Sep 1997.

[7] 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 田中英彦. 大規模な投機的処理における分岐制御機構. 第 56 回 情処全国大会, Vol. 1, No. 5N-8, pp. 173–174, Mar 1998.

[8] 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 高峰信, 坂井修一, 田中英彦. VLDP アーキテクチャの性能に関する初期的考察. 第 57 回 情処全国大会, Vol. 1, No. 1Q-7, pp. 34–35, Oct 1998.

[9] 吉瀬謙二, 中村友洋, 辻秀典, 田中英彦. 制御依存関係による並列度利用の限界に関する定量的評価. 第 53 回 情処全国大会, Vol. 1, No. 4F-1, pp. 113–114, Sep 1996.

[10] 吉瀬謙二, 中村友洋, 辻秀典, 安島雄一郎, 田中英彦. 多数演算器方式における演算器利用率の検討. 情処研究会 ARCH 125, Vol. 97, No. 62, pp. 121–126, Aug 1997.

[11] 吉瀬謙二, 中村友洋, 辻秀典, 安島雄一郎, 田中英彦. Alu-net を用いることによるデータ移動の効率化. 第 56 回 情処全国大会, Vol. 1, No. 2N-1, pp. 109–110, Mar 1998.

[12] 辻秀典, 中村友洋, 吉瀬謙二, 安島雄一郎, 田中英彦. マルチレベル分岐予測の検討と評価. 第 55 回 情処全国大会, Vol. 1, No. 3F-6, pp. 22–23, Sep 1997.

[13] 辻秀典, 中村友洋, 吉瀬謙二, 安島雄一郎, 田中英彦. 大規模データパス・プロセッサにおけるフェッチ機構の検討. 情処研究会 ARCH 126, Vol. 97, No. 63, Oct 1997.

[14] 辻秀典, 中村友洋, 吉瀬謙二, 安島雄一郎, 田中英彦. 実行パス予測における確率伝搬手法の検討. 第 56 回 情処全国大会, Vol. 1, No. 2N-2, pp. 111–112, Mar 1998.

[15] 木庭優治, 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 田中英彦. 複合ストライド法によるロードアドレス予測. 第 56 回 情処全国大会, Vol. 1, No. 5N-6, pp. 169–170, Mar 1998.

[16] 安島雄一郎, 中村友洋, 田中英彦. 複数パスの投機的実行を考慮した例外回復機構. 第 56 回 情処全国大会, Vol. 1, No. 5N-7, pp. 171–172, Mar 1998.

[17] 安島雄一郎, 中村友洋, 吉瀬謙二, 辻秀典, 田中英彦. 例外回復可能な複数パス実行機構の提案. 情処研究会 ARCH 129, Vol. 98, No. 66, May 1998.

参考文献

[CB94] Tien-Fu Chen and Jean-Loup Bear. A performance study of software and hardware data prefetching. *21st ISCA*, pp. 223–232, May 1994.

[CP94] Po-Yung Chang and Yale Patt. Branch classification: A new mechanism for improving branch predictor performance. *Proceedings of the 27th International Symposium on Microarchitecture*, pp. 22–31, 1994.

[CPC94] Thomas M. Conte, Burzin A. Patel, and J. Stan Cox. Using branch handling hardware to support profile-driven optimization. *Proceedings of the 27th International Symposium on Microarchitecture*, pp. 12–21, 1994.

[Cri97] Richard Crisp. Direct rambus technology: The new main memory standard. *IEEE MICRO*, Vol. 17, No. 6, pp. 18–28, Nov 1997.

[CWM+95] Pohua P. Chang, Nancy J. Warter, Scott A. Mahlk, William Y. Chen, and Wen mei W. Hwu. Three architectural models for compiler-controlled speculative execution. *IEEE Transactions on Computers*, Vol. 44, No. 4, pp. 481–494, 1995.

[DB97] Alain Kagi Doug Burger, James R.Goodman. Limited bandwidth to affect processor design. *IEEE MICRO*, Vol. 17, No. 6, pp. 55–62, Nov 1997.

[DEC98] DEC. 600mhz out-of-order instruction-queue circuit. *1998 IEEE International Solid-State Circuits Conference*, p. ?[FP15.2], Feb 1998.

[DF95] Simonjit Dutta and Manoj Franklin. Control flow prediction with tree-like subgraphs for superscalar processors. *28th MICRO*, pp. 258–263, Nov 1995.

[DL87] John A. DeRosa and Henry M. Levy. An evaluation of branch architectures. *Proceedings of the 14th International Symposium on Computer Architecture*, Vol. 14, No. 14, pp. 10–16, 1987.

[DM87] David R. Ditzel and Hubert R. McLellan. Branch folding in the CRISP microprocessor reducing branch delay to zero. *Proceedings of the 14th International Symposium on Computer Architecture*, Vol. 14, No. 14, pp. 2–9, 1987.

[DSS89] Kaeli D.R., Kirkpatrick S., and Ong S. Pc workload characterization. *Proceedings of the ACM Sigmetrics and Performance*, p. 220, 1989.

[ea92] S.A. Mahlke et al. Effective compiler support for predicated execution using the hyperblock. *25th MICRO*, pp. 45–54, Nov 1992.

[etc98] Lynn Comp etc. Highly-integrated low-power 200mhz, 0.5w strong arm microprocessor. *1998 IEEE International Solid-State Circuits Conference*, p. ?[FP15.5], Feb 1998.

[FJMS96] Federico Faggin, Marcian E.Hoff Jr., Stanley Mazor, and Masatoshi Shima. The history of the 4004. *IEEE MICRO*, Vol. 16, No. 6, pp. 10–20, Dec 1996.

[fSMP] 1997 National Technology Roadmap for Semiconductors Member Page. http://notes.sematech.org/ 1997mem.htm.

[KE91] David R. Kaeli and Philip G. Emma. Branch history table prediction of moving target branches due to subroutine returns. *Proceedings of the 18th International Symposium on Computer Architecture*, No. 3, pp. 34–42, 1991.

[Lil84] David J. Lilja. Exploiting the parallelism available in loops. *IEEE COMPUTER*, Vol. 27, No. 2, pp. 13–26, 1984.

[LKM98] Chih-Chieh Lee, I-Cheng K.Chen, and Trevor N. Mudge. The bi-mode branch predictor. *30th Annual IEEE/ACM International Symposium on MICROarchitecture*, pp. 4–13, 1998.

[LW92] Monica S. Lam and Robert P. Wilson. Limits of control flow on parallelism. *Proceedings of the 19th International Symposium on Computer Architecture*, Vol. 19, No. 19, pp. 46–57, 1992.

[Masa95] Masaki Kumanoya, Toshiyuki Ogawa, Kazunari Inoue. Advances in DRAM interfaces. *IEEE MICRO*, Vol. 15, No. 6, pp. 30–36, Nov 1995.

[MDA93] James D. Meindl, Vivek K. De, and Bhavna Agrawal. Prospects for gigascale integration(GSI) beyond 2003. *IEEE ISSCC Digest of Technical Papers*, pp. 124–125, 1993.

[Mich89] Michael D. Smith, Mike Johnson, Mark A.Horowitz. Limits on multiple instruction issue. *3rd ASPLOS*, Vol. 3, pp. 290–302, 1989.

[MN96] S. Mahlke and B. Natarajan. Compiler synthesized dynamic branch prediction. *29th Annual IEEE/ACM International Symposium on MICROarchitecture*, pp. 153–164, Dec 1996.

[mWHCC89] Wen mei W. Hwu, Thomas M. Conte, and Pohua P. Chang. Comparing software and hardware schemes for reducing the cost of branches. *Proceedings of the 16th International Symposium on Computer Architecture*, Vol. 16, No. 16, pp. 224–233, 1989.

[PSR92] S.-T. Pan, K. So, and J. T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. *ASPLOS-V*, pp. 76–84, 1992.

[RBS96] E. Rotenberg, S. Bennett, and J. E. Smith. Trace cache : A low latency approach to high bandwidth instruction fetching. *29th Annual IEEE/ACM International Symposium on MICROarchitecture*, pp. 24–34, Dec 1996.

[SE97] Tiannakis Sazeides and James E.Smith. The predictability of data values. *30th Annual IEEE/ACM International Symposium on MICROarchitecture*, pp. 248–257, 1997.

[Sem97] Semiconductor Industry Association(SIA). *The National Technology Roadmap for Semiconductors*, 1997.

[SrP92] Monica S.Lam and robert P.Wilson. Limits of control flow on parallelism. *19th ISCA*, pp. 46–57, May 1992.

[tS] Welcome to SPEC. http://www.spec.org.

[tT] Welcome to TPC. http://www.tpc.org.

[US95] Augustus K. Uht and Vijay Sindagi. Disjoint eager execution: An optimal form of a speculative execution. *IEEE 28th International Symposium on Microarchitecture*, pp. 313–325, 1995.

[WCT98] Steven Wallace, Brad Calder, and Dean M. Thllsen. Threaded multiple path execution. *Proceedings of the 25th International Symposium on Computer Architecture*, pp. 238–249, 1998.

[WDB90] Jack W.Davidson and Whalley David B. Reducing the cost of branches by using registers. *Proceedings of the 17th International Symposium on Computer Architecture*, Vol. 17, No. 17, pp. 182–191, 1990.

[WF98] Kai Wang and Manoj Franklin. Highly accurate data prediction using hybrid predictors. *30th Annual IEEE/ACM International Symposium on MICROarchitecture*, pp. 281–290, 1998.

[Wil94] Maurice V. Wilkes. The memory wall and the CMOS end-point. *Computer Architecture News*, Vol. 23, No. 4, pp. 4–6, Feb 1994.

[WL94] Youfeng Wu and James R. Larus. Static branch frequency and program profile analysis. *Proceedings of the 27th International Symposium on Microarchitecture*, pp. 1–11, 1994.

[YGS94] C. Young, N. Gloy, and M.D. Smith. A comparative analysis of schemes for correlated branch prediction. *ISCA-22*, pp. 276–286, 1994.

[YN92] Tse-Yu. Yeh and Yale N.Patt. Alternative implementations of two-level adaptive branch prediction. *19th ISCA*, pp. 124–134, May 1992.

[Yu.96] Albert Yu. The future of microprocessors. *IEEE MICRO*, Vol. 16, No. 6, pp. 46–59, Dec 1996.

[安藤 94] 安藤秀樹, 中西知嘉子, 原哲也, 町田浩久, 中屋雅夫. 投機的実行を行なうマシンにおける例外処理方式. 情報処理学会計算機アーキテクチャ研究会報告, No. 107-13, pp. 97–104, 1994.

[安倍 94] 安倍正人, 岡部公起, 根元義章. ソフトウェアパイプラインによる条件分岐の際のハザードの影響の 削減. 情報処理学会 計算機アーキテクチャ研究会報告, No. 104-11, pp. 81–88, 1994.

[移動] 移動電気通信事業加入数の現況 (1998.4). http://www.mpt.go.jp/policyreports/japanese/stats/handy-phone.html.

[吉瀬 98] 吉瀬謙二, 中村友洋, 辻秀典, 安島雄一郎, 田中英彦. Alu-net を用いることによるデータ移動の効率化. 第 56 回情報処理学会全国大会 2N-1, Vol. 1, pp. 109–110, Mar 1998.

[原哲 93] 原哲也, 安藤秀樹, 中西知嘉子, 町田浩久, 中屋雅夫. スーパースカラ・プロセッサにおける分岐命令の並列実行. 情報処理学会 計算機アーキテクチャ研究会報告, No. 101-9, pp. 65–72, 1993.

[坂井 97] 坂井修一. オンチップマルチプロセッシングに関する初期的検討. 情処研究会 ARCH 122-7, Vol. 97, No. 15, pp. 33–38, Feb 1997.

[山名 94] 山名早人, 佐藤三久, 児玉祐悦, 坂根広史, 坂井修一, 山口喜教. 投機的実行の現状と unlimited speculative execution scheme の提案. 情報処理学会計算機アーキテクチャ研究会報告, No. 107-14, pp. 105–112, 1994.

[須賀 98] 須賀敦浩 etc. 1.2W 2.16GOPS/720MFLOPS マルチメディア用組み込みスーパースカラプロセッサ. 信学技報, No. 22, pp. 33–40, Apr 1998.

[西 98] 西直樹. オンチップマルチプロセッサの研究動向. 並列処理シンポジウム JSPP '98 ワークショップ, pp. 43–52, Jun 1998.

[村上 94] 村上和彰, 吉井卓, 岩下茂信. 21 世紀に向けた新しい汎用機能部品 PPRAM の提案. 情処研究会 ARCH 108-8, Vol. 94, No. 91, pp. 1–8, Oct 1994.

[村上 95] 村上和彰, 吉井卓, 岩下茂信. 次々世代汎用マイクロプロセッサ・アーキテクチャPPRAM の概 要. 情処研究会 ARCH 113-1, Vol. 95, No. 80, pp. 1–8, Aug 1995.

[中村 97] 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 田中英彦. 分岐アドレス予測機構の比較検討. 第 55 回情報処理学会全国大会 3F-5, Vol. 1, pp. 20–21, Sep 1997.

[鳥居 96] 鳥居淳, 木村真人, 近藤真己, 鈴木研司, 小長谷明彦. 順序付きマルチスレッドアーキテクチャのプログラミングモ デルと評価. 並列処理シンポジウム JSPP '96, pp. 299–306, June 1996.

[鳥居 97] 鳥居淳, 近藤真己, 本村真人, 西直樹, 小長谷明彦. On chip multiprocessor 指向 制御並列アーキテクチャmuscat の提案. 並列処理シンポジウム JSPP '97, pp. 229–236, May 1997.

[日経 93] 日経エレクトロニクス. 特集:ISSCC93. 日経エレクトロニクス, No. 575, pp. 85–117, 1993.

[日経 94] 日経エレクトロニクス. ニュース・レポート [DRAM]. 日経エレクトロニクス, No. 621, pp. 21–24, 1994.

Appendix A

コントロールフロー先行展開の性能評価詳細

ここでは、go, ijpeg, li, perl の各プログラムについて、フェッチ命令数制限時の平均並列度、分岐予測成功率、有効命令率、フラッシュ率および、様々な命令展開方式の平均並列度比較、有効命令率の各グラフを示す。

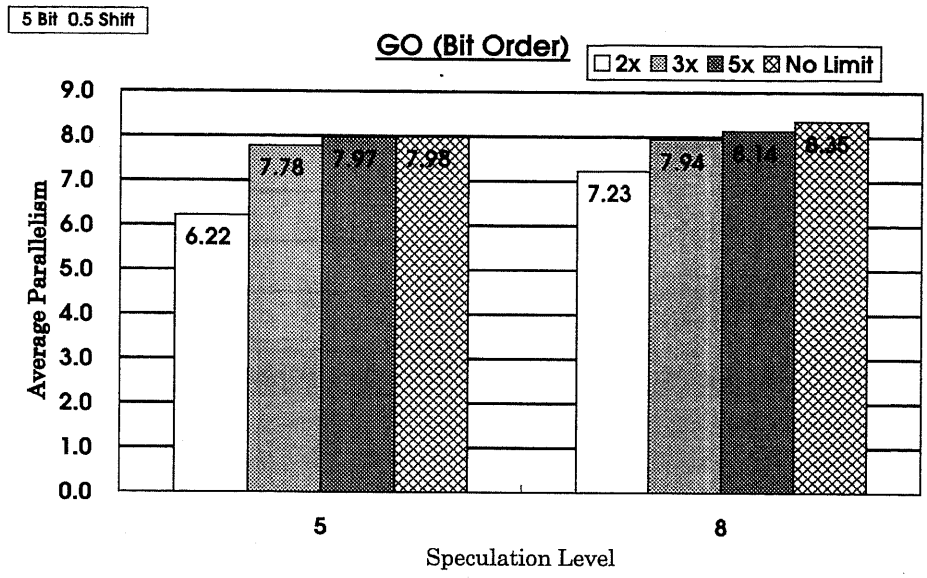


図 A.1: フェッチ命令数制限時の平均並列度 (GO,5bit 履歴,0.5bit シフト,BO 方式)

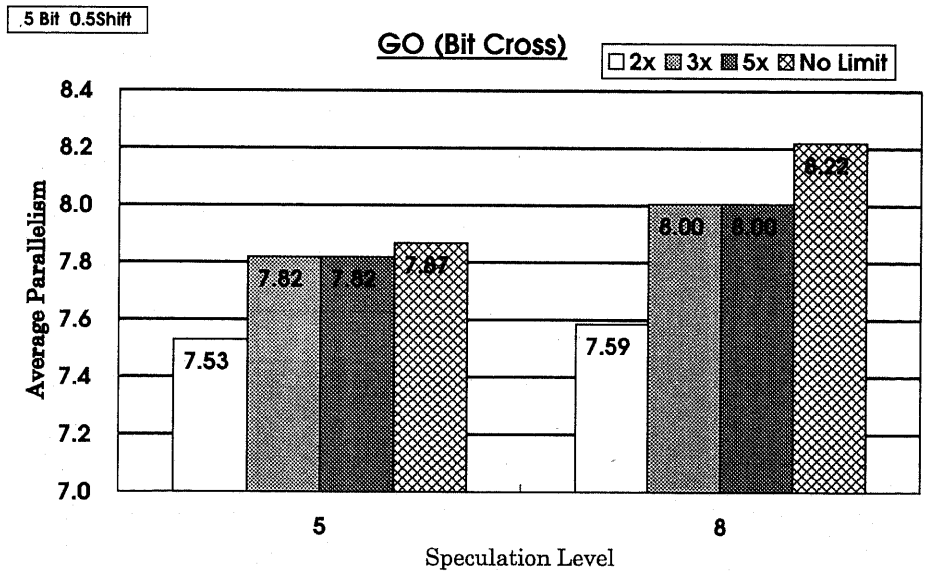


図 A.2: フェッチ命令数制限時の平均並列度 (GO,5bit 履歴,0.5bit シフト,BC 方式)

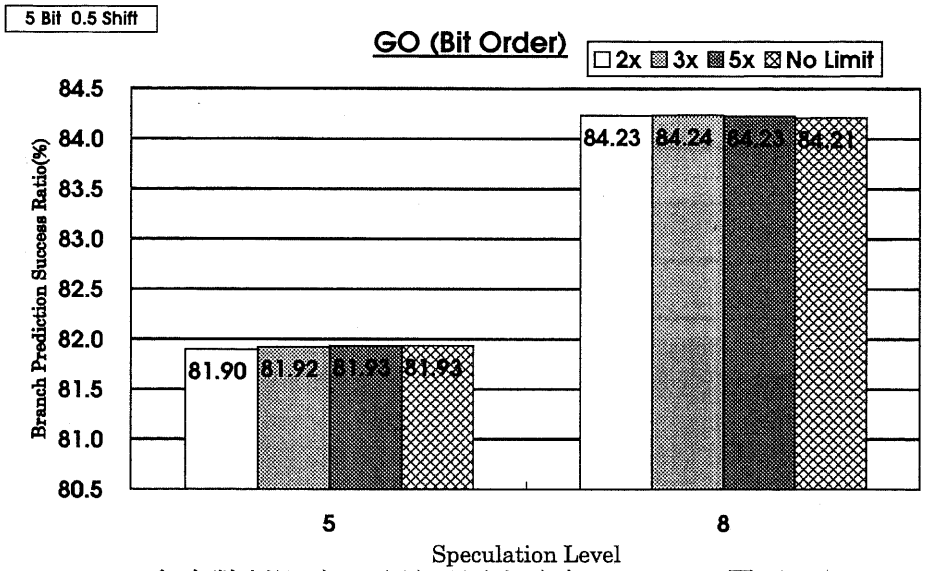


図 A.3: フェッチ命令数制限時の分岐予測成功率 (GO,5bit 履歴,0.5bit シフト,BO 方式)

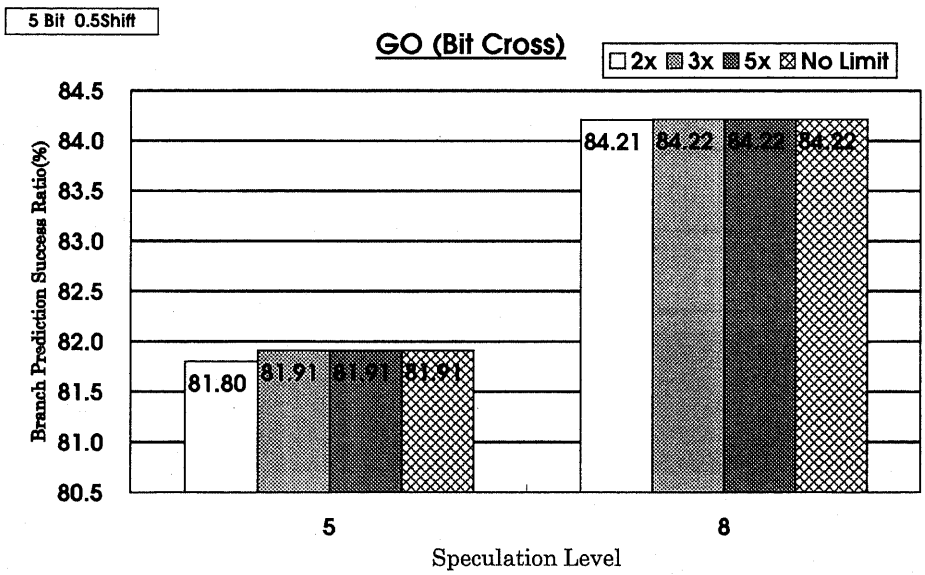


図 A.4: フェッチ命令数制限時の分岐予測成功率 (GO,5bit 履歴,0.5bit シフト,BC 方式)

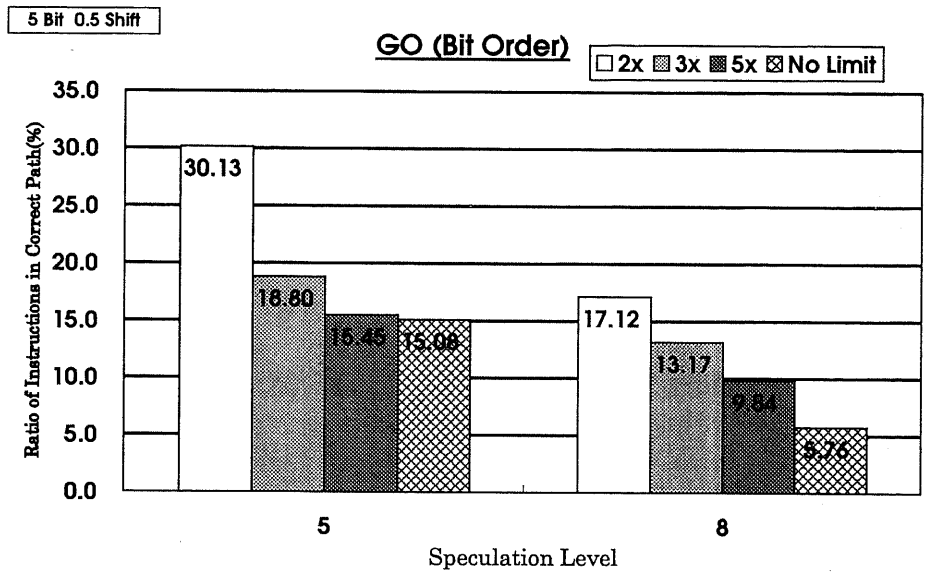


図 A.5: フェッチ命令数制限時の有効命令率 (GO,5bit 履歴,0.5bit シフト,BO 方式)

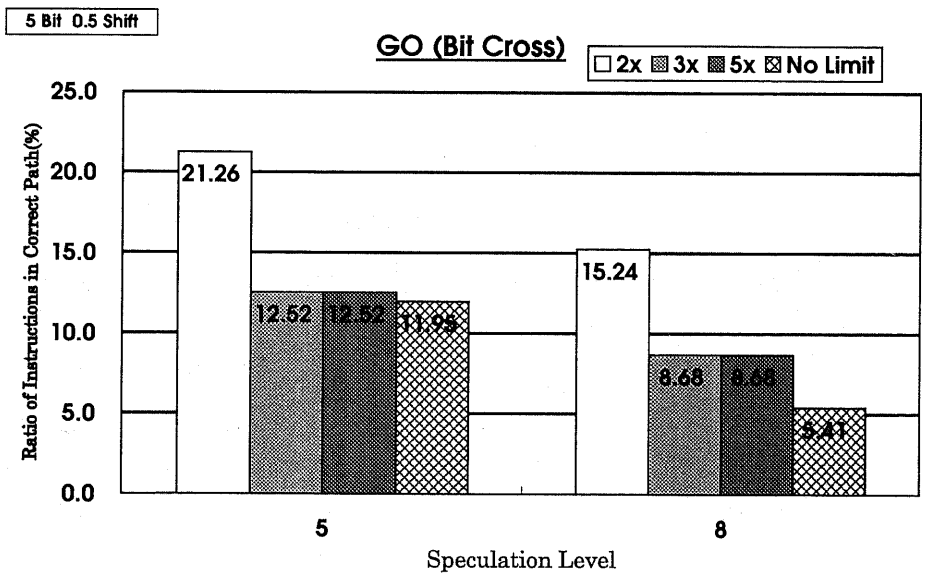


図 A.6: フェッチ命令数制限時の有効命令率 (GO,5bit 履歴,0.5bit シフト,BC 方式)

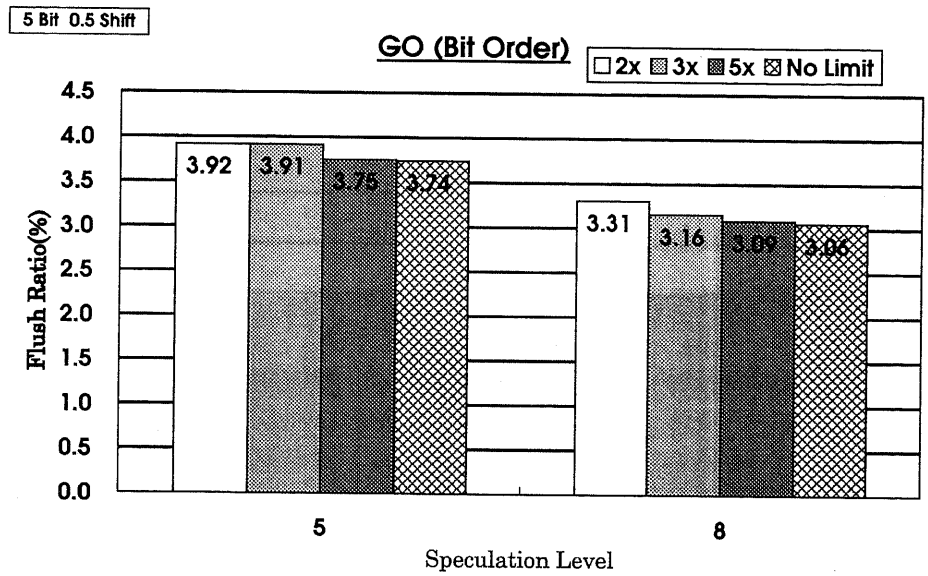


図 A.7: フェッチ命令数制限時のフラッシュ率 (GO,5bit 履歴,0.5bit シフト,BO 方式)

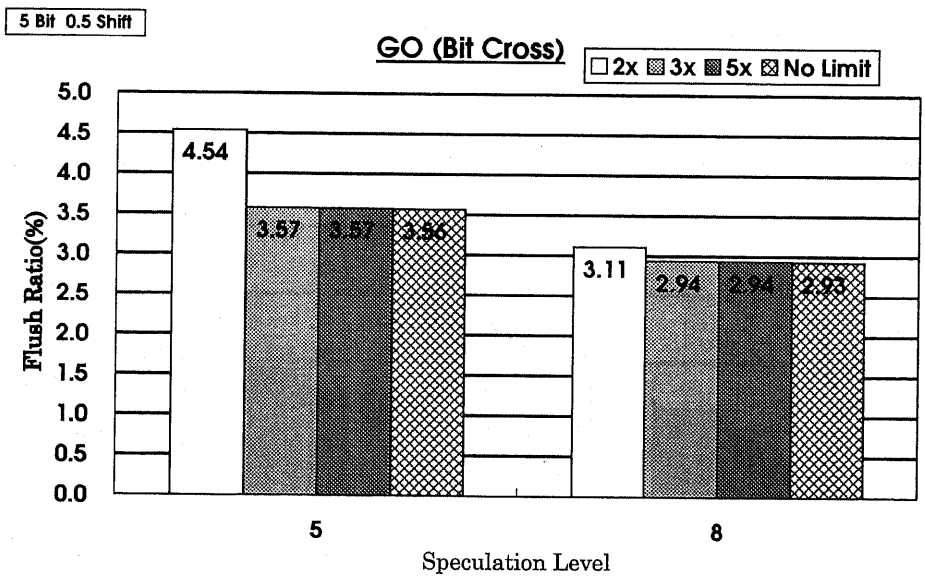


図 A.8: フェッチ命令数制限時のフラッシュ率 (GO,5bit 履歴,0.5bit シフト,BC 方式)

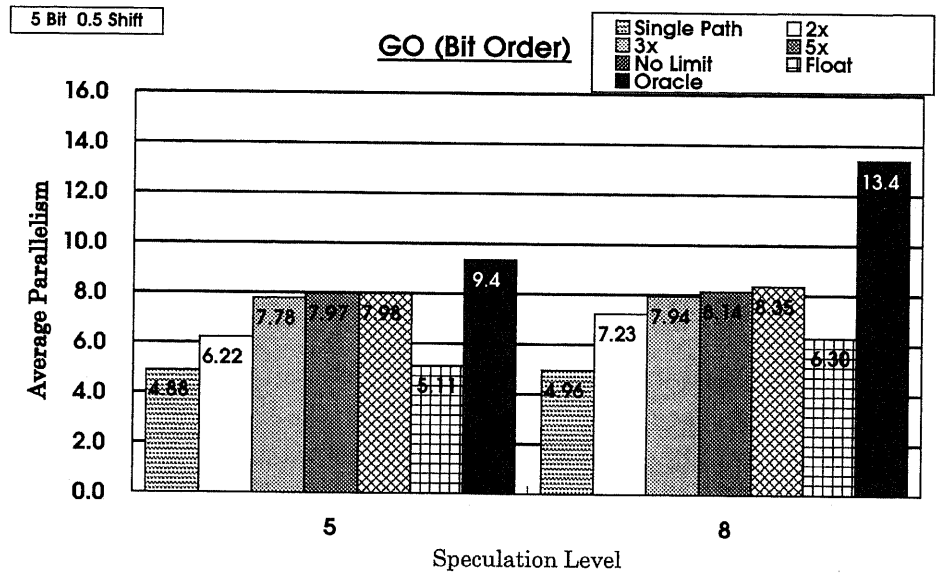


図 A.9: 様々な命令展開方式の平均並列度比較 (GO)

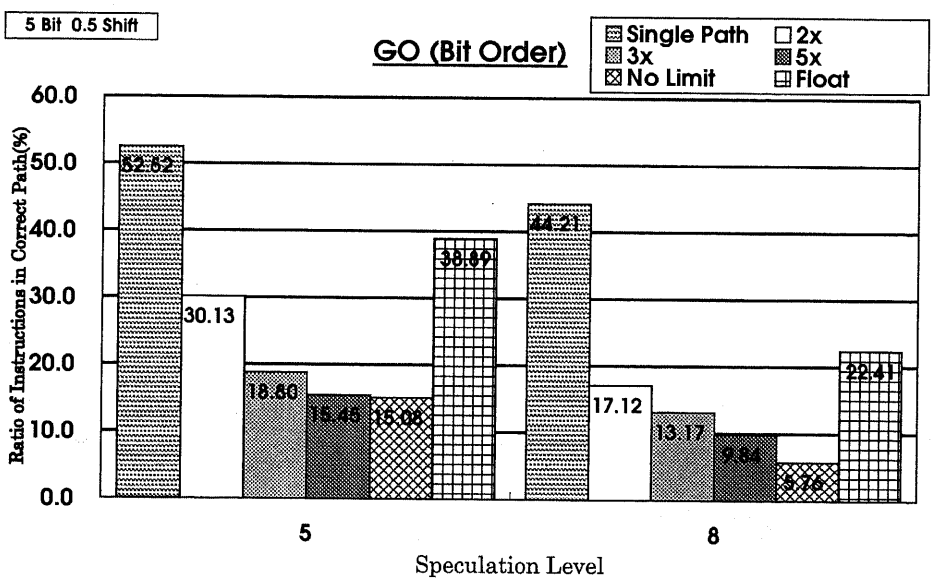


図 A.10: 様々な命令展開方式の有効命令率 (GO)

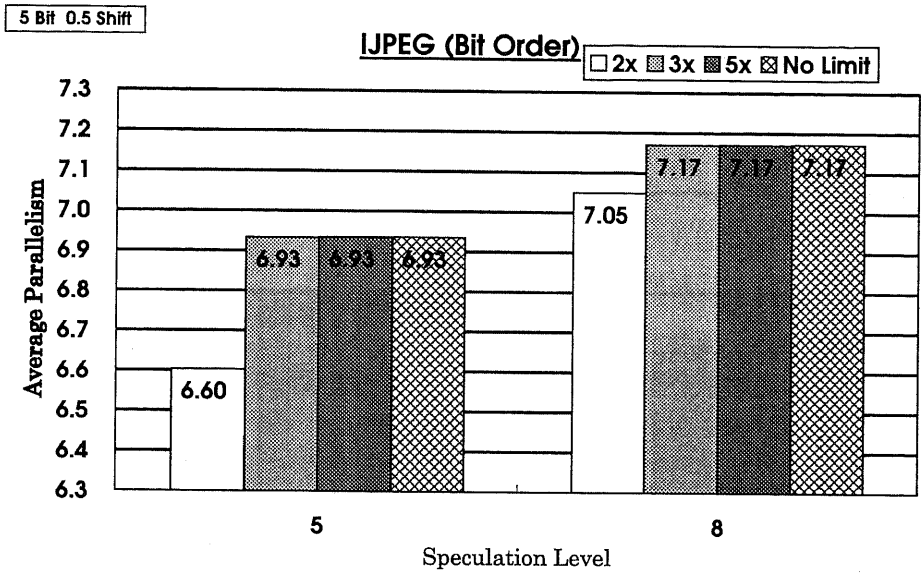


図 A.11: フェッチ命令数制限時の平均並列度 (IJPEG,5bit 履歴,0.5bit シフト,BO 方式)

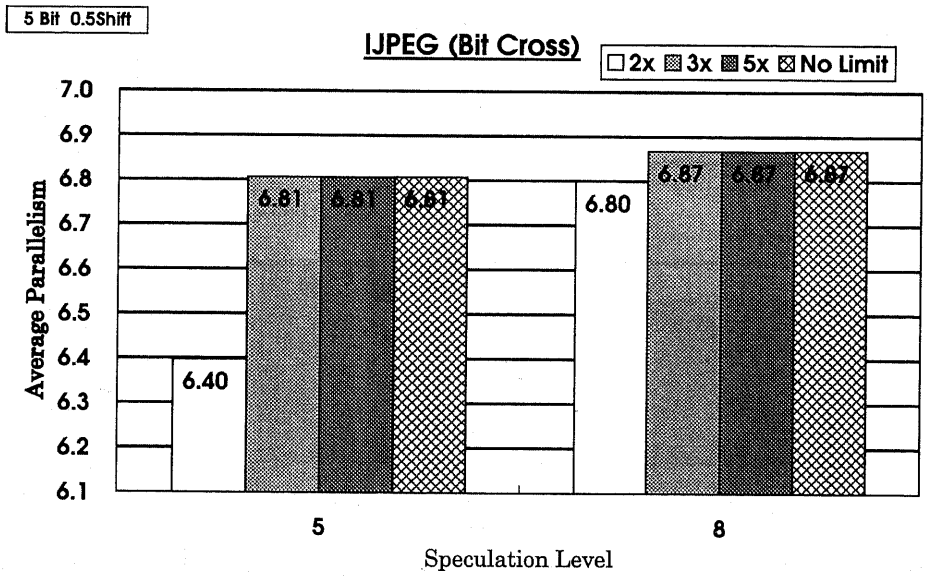


図 A.12: フェッチ命令数制限時の平均並列度 (IJPEG,5bit 履歴,0.5bit シフト,BC 方式)

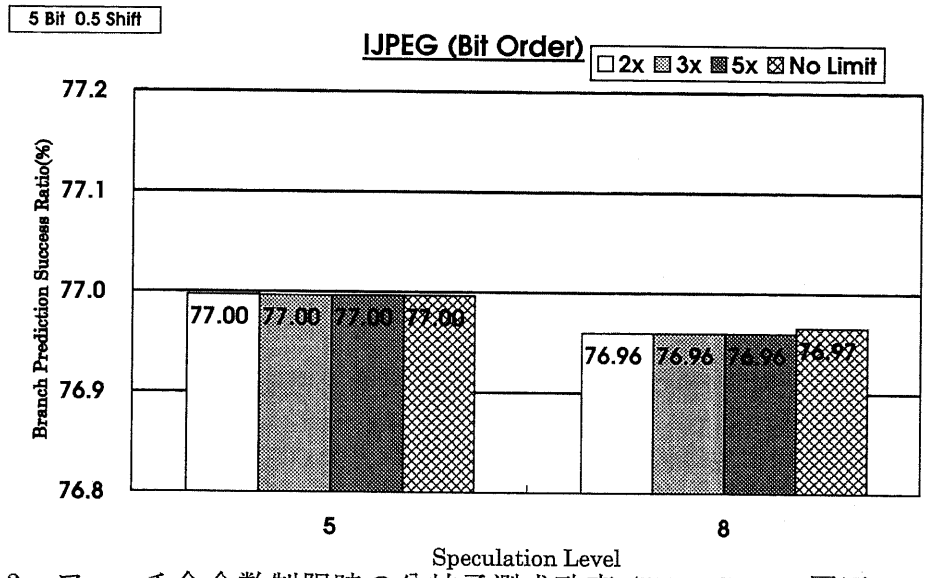


図 A.13: フェッチ命令数制限時の分岐予測成功率 (IJPEG,5bit 履歴,0.5bit シフト,BO 方式)

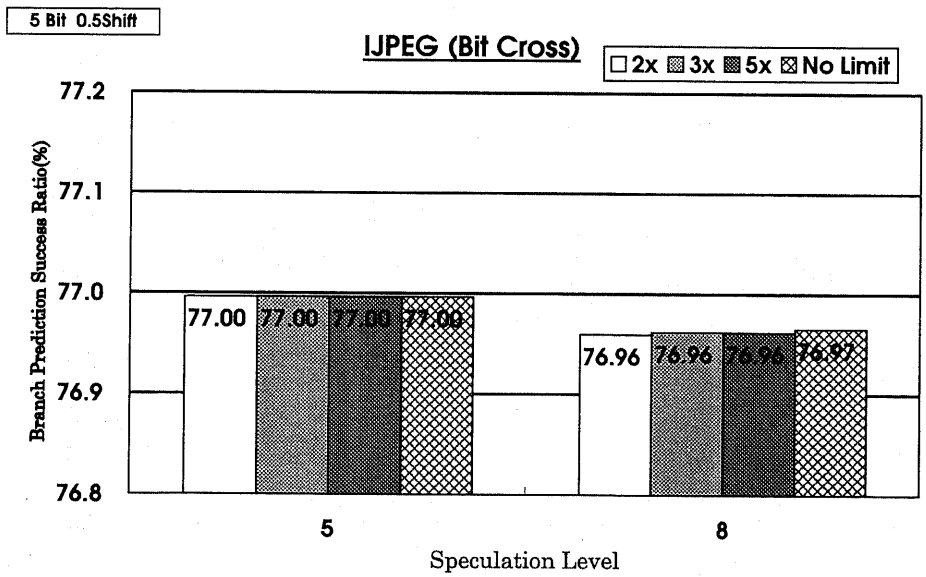


図 A.14: フェッチ命令数制限時の分岐予測成功率 (IJPEG,5bit 履歴,0.5bit シフト,BC 方式)

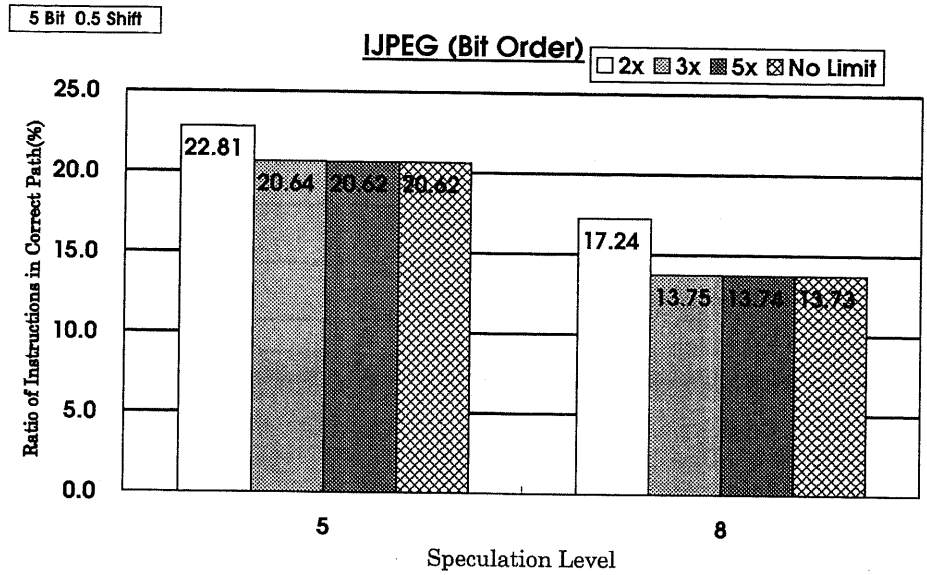


図 A.15: フェッチ命令数制限時の有効命令率 (IJPEG,5bit 履歴,0.5bit シフト,BO 方式)

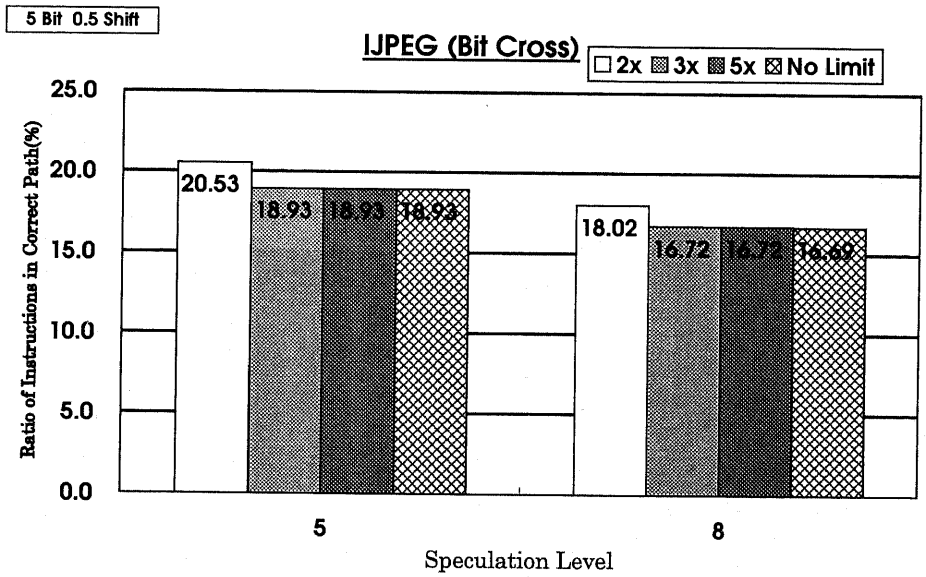


図 A.16: フェッチ命令数制限時の有効命令率 (IJPEG,5bit 履歴,0.5bit シフト,BC 方式)

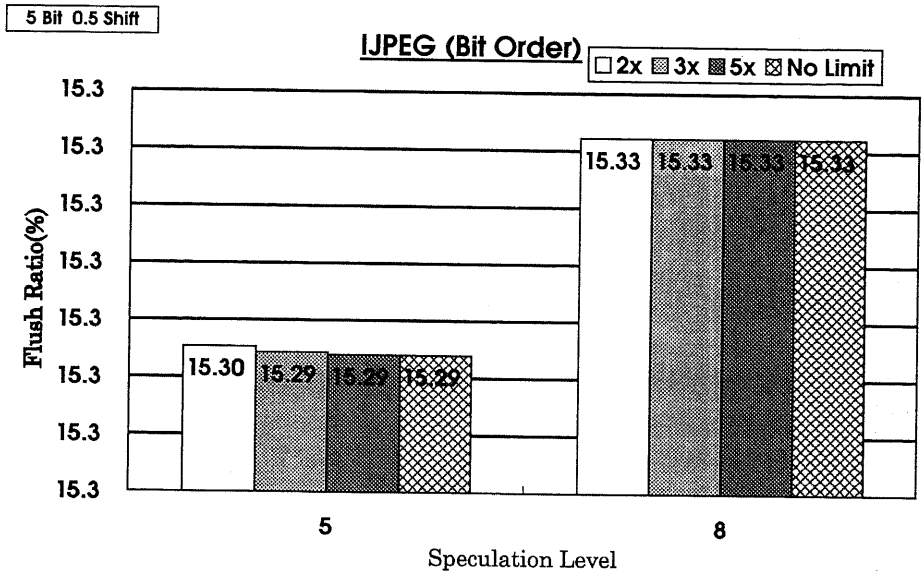


図 A.17: フェッチ命令数制限時のフラッシュ率 (IJPEG,5bit 履歴,0.5bit シフト,BO 方式)

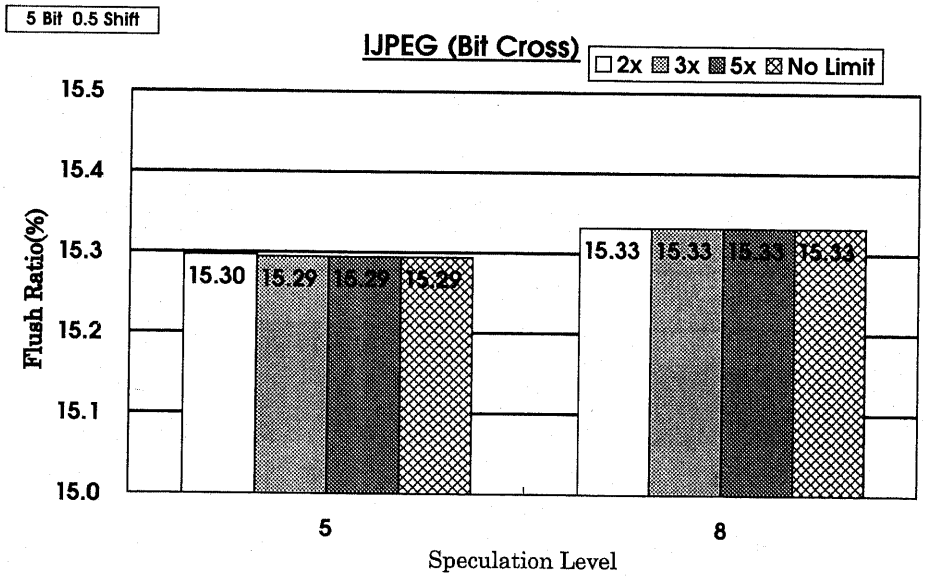


図 A.18: フェッチ命令数制限時のフラッシュ率 (IJPEG,5bit 履歴,0.5bit シフト,BC 方式)

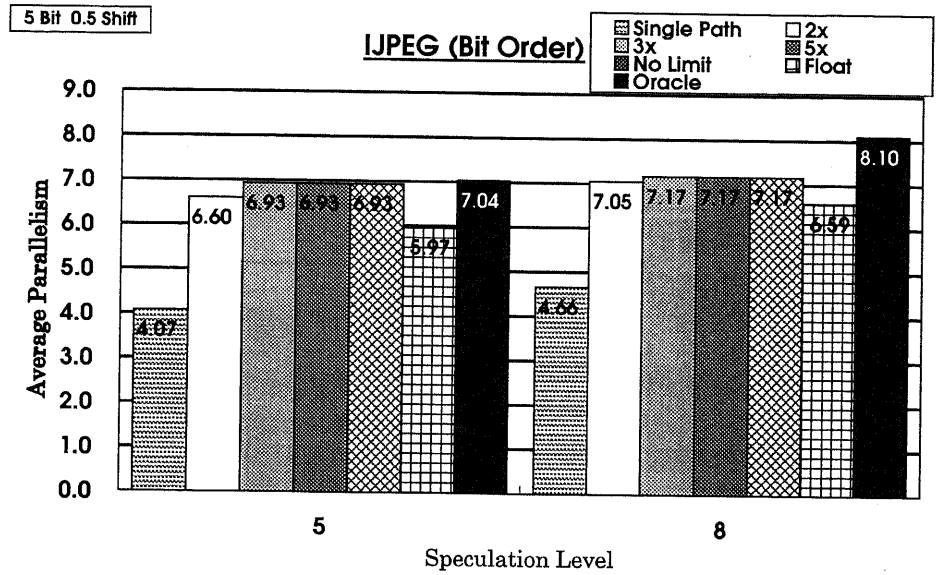


図 A.19: 様々な命令展開方式の平均並列度比較 (IJPEG)

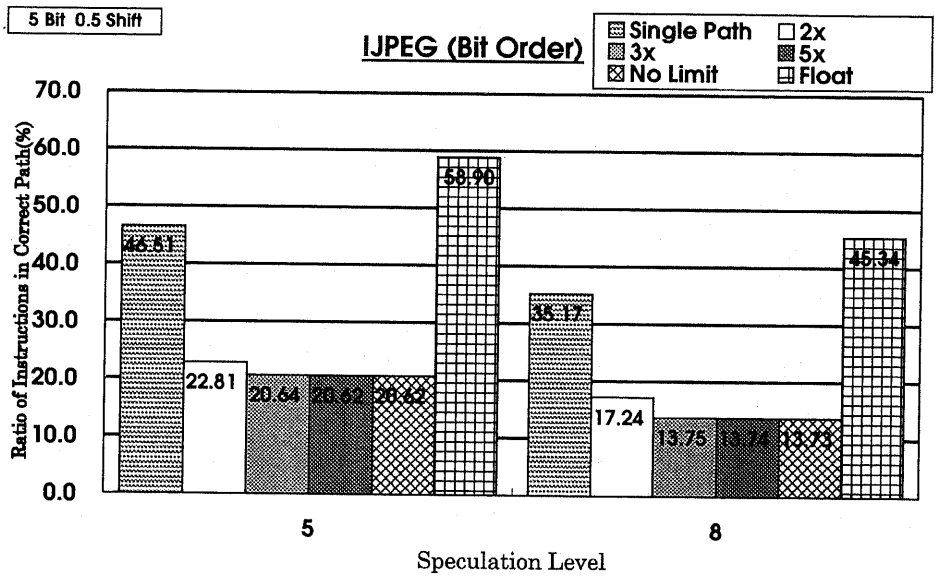


図 A.20: 様々な命令展開方式の有効命令率 (IJPEG)

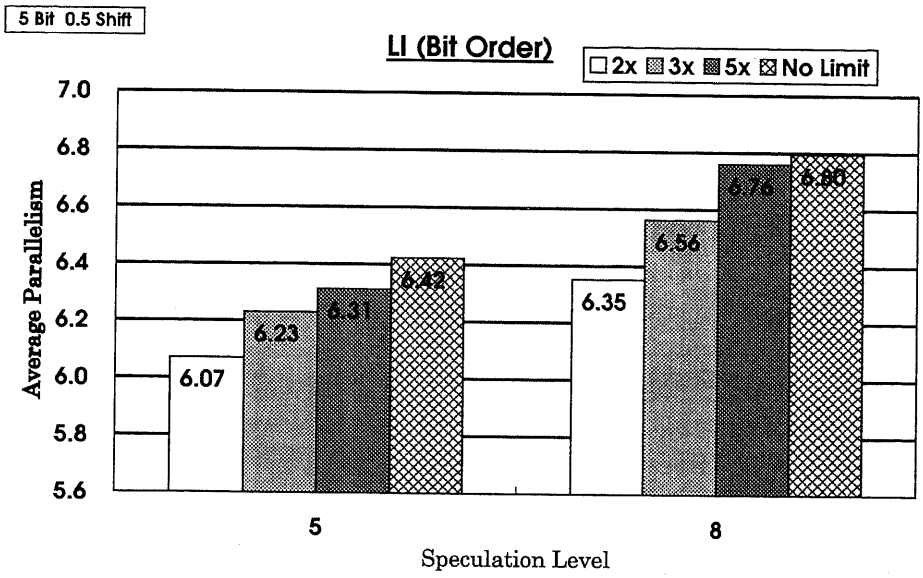


図 A.21: フェッチ命令数制限時の平均並列度 (LI,5bit 履歴,0.5bit シフト,BO 方式)

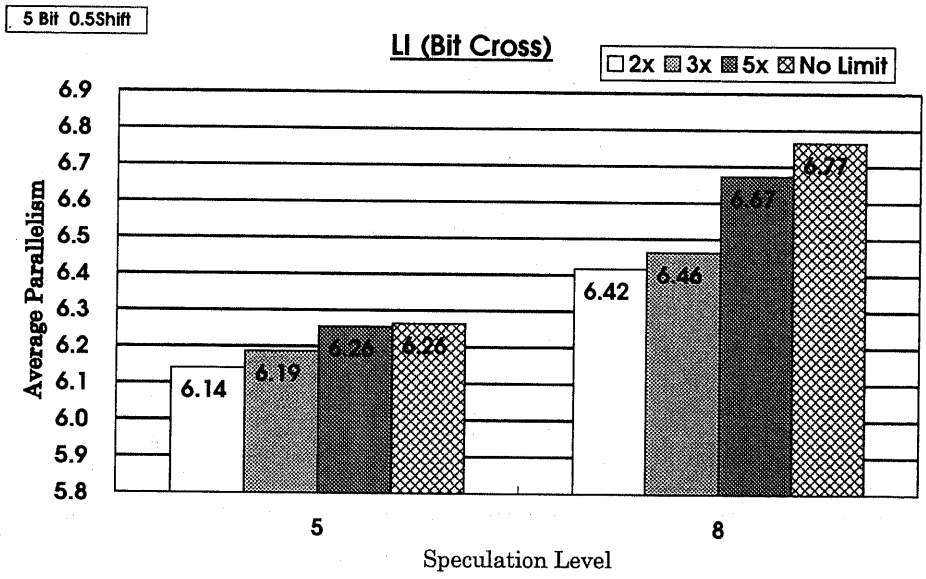


図 A.22: フェッチ命令数制限時の平均並列度 (LI,5bit 履歴,0.5bit シフト,BC 方式)

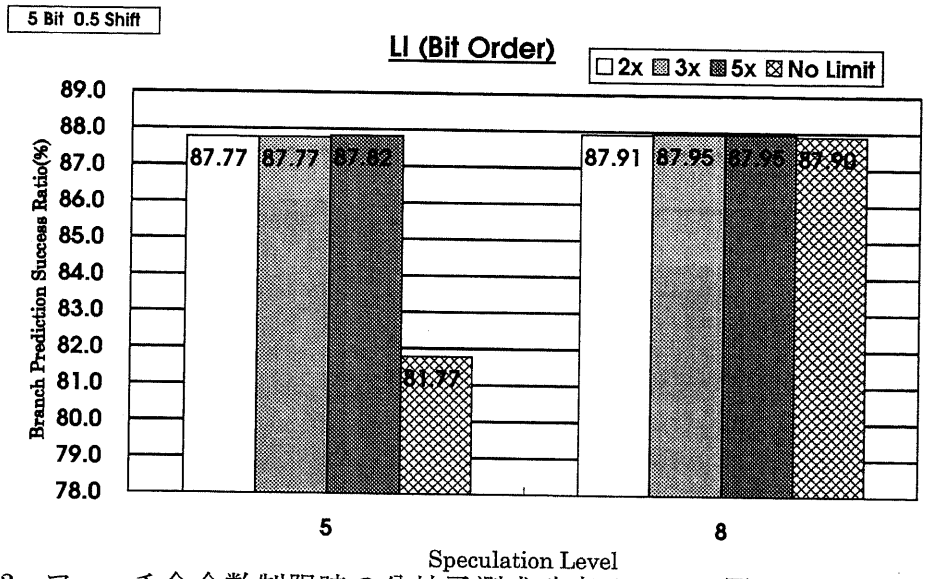


図 A.23: フェッチ命令数制限時の分岐予測成功率 (LI,5bit 履歴,0.5bit シフト,BO 方式)

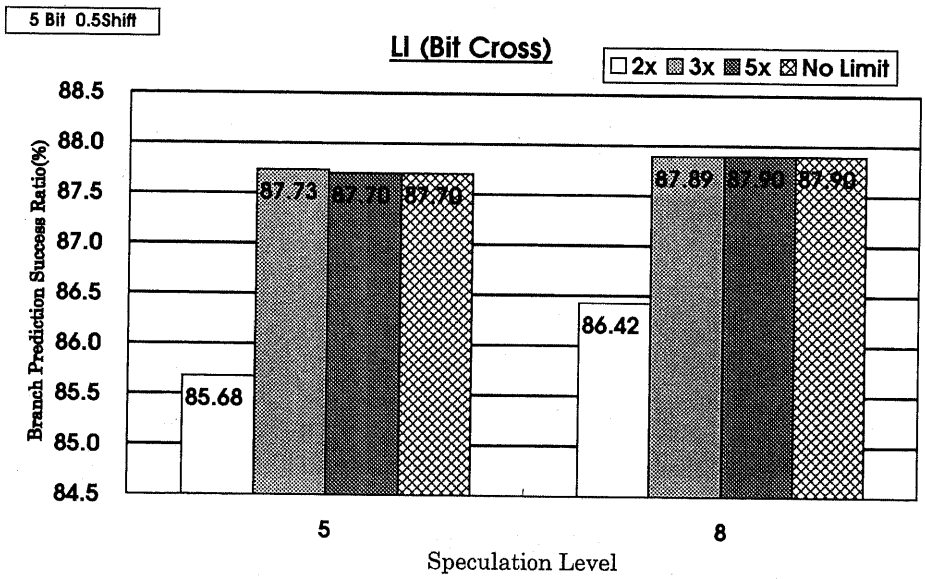


図 A.24: フェッチ命令数制限時の分岐予測成功率 (LI,5bit 履歴,0.5bit シフト,BC 方式)

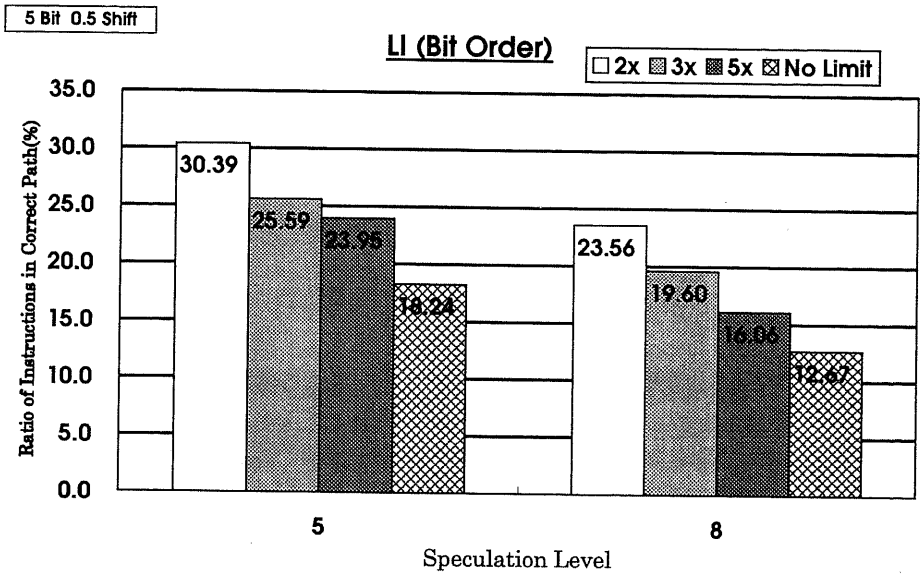


図 A.25: フェッチ命令数制限時の有効命令率 (LI,5bit 履歴,0.5bit シフト,BO 方式)

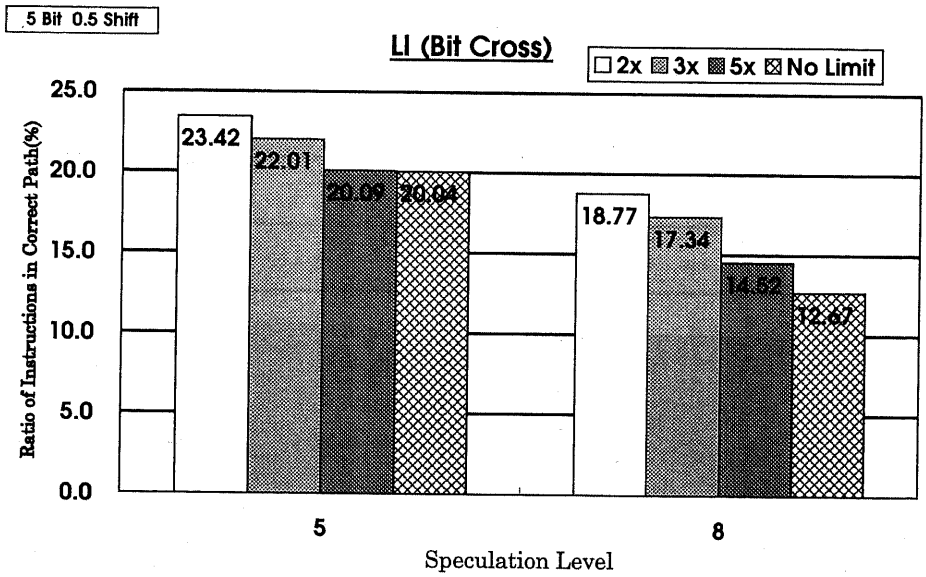


図 A.26: フェッチ命令数制限時の有効命令率 (LI,5bit 履歴,0.5bit シフト,BC 方式)

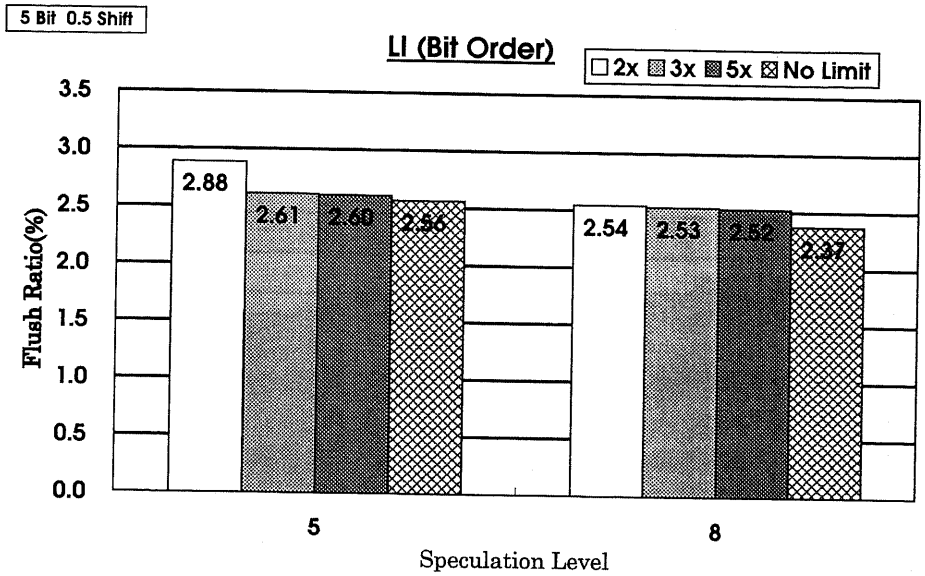


図 A.27: フェッチ命令数制限時のフラッシュ率 (LI,5bit 履歴,0.5bit シフト,BO 方式)

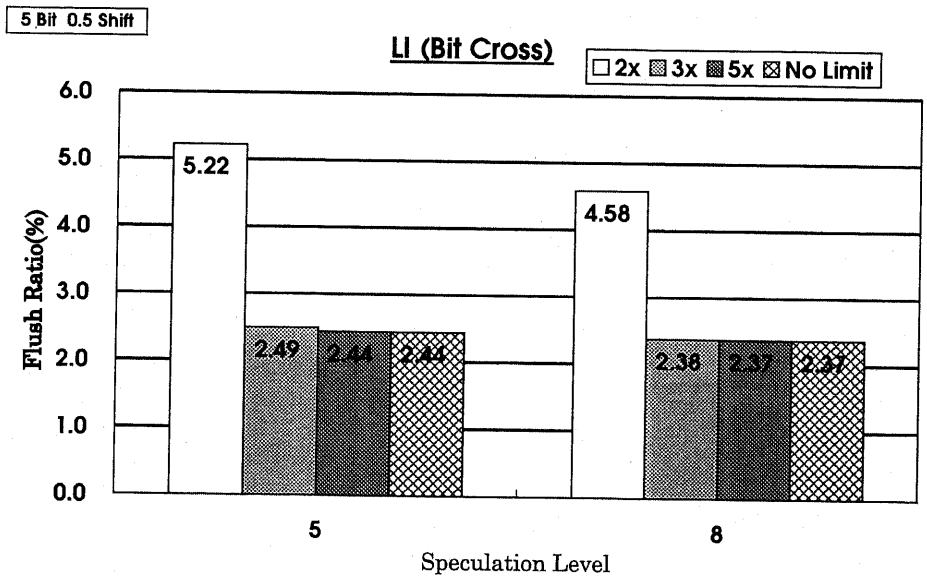


図 A.28: フェッチ命令数制限時のフラッシュ率 (LI,5bit 履歴,0.5bit シフト,BC 方式)

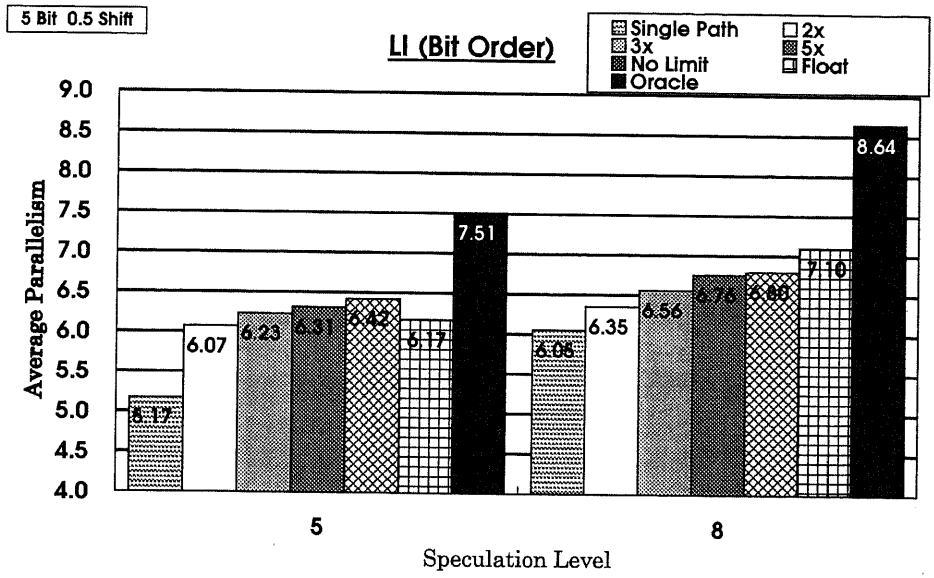


図 A.29: 様々な命令展開方式の平均並列度比較 (LI)

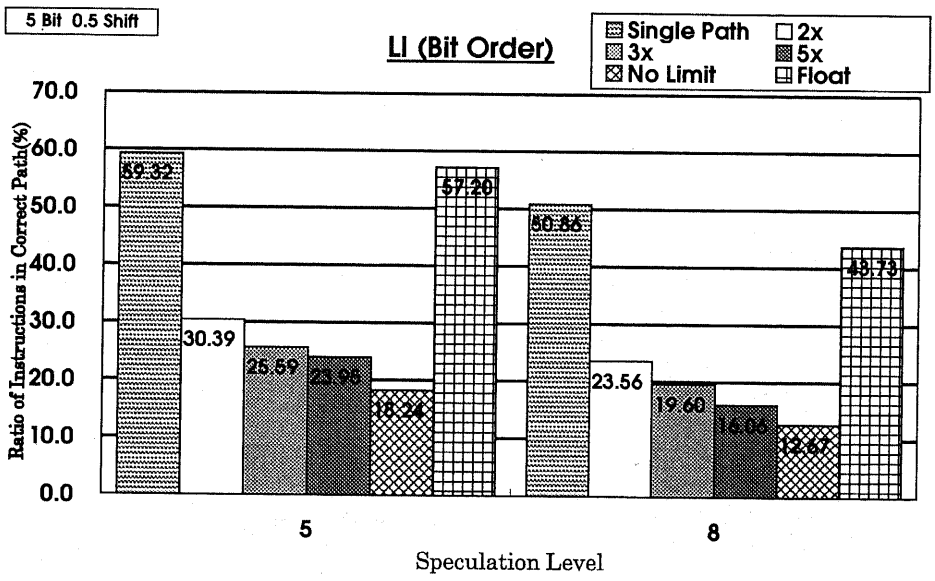


図 A.30: 様々な命令展開方式の有効命令率 (LI)

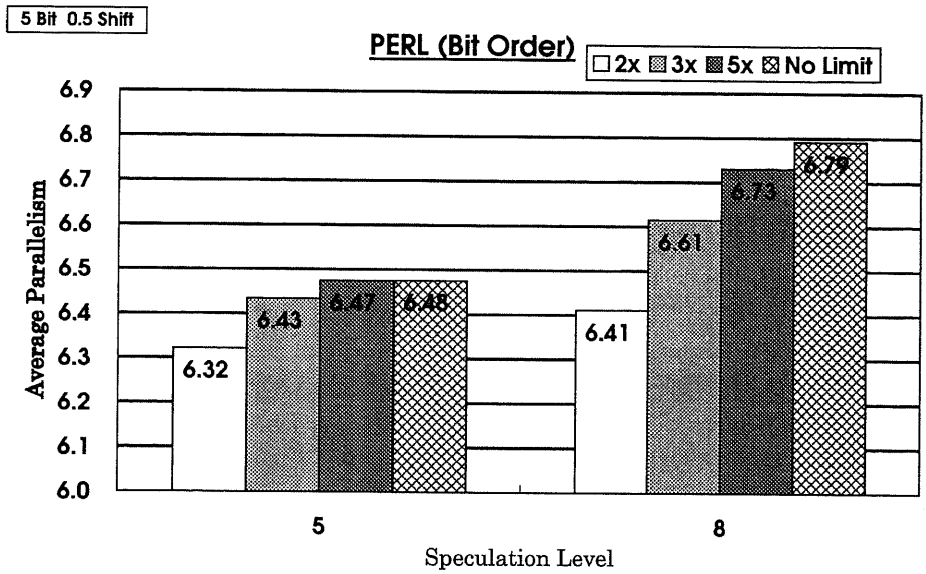


図 A.31: フェッチ命令数制限時の平均並列度 (PERL,5bit 履歴,0.5bit シフト,BO 方式)

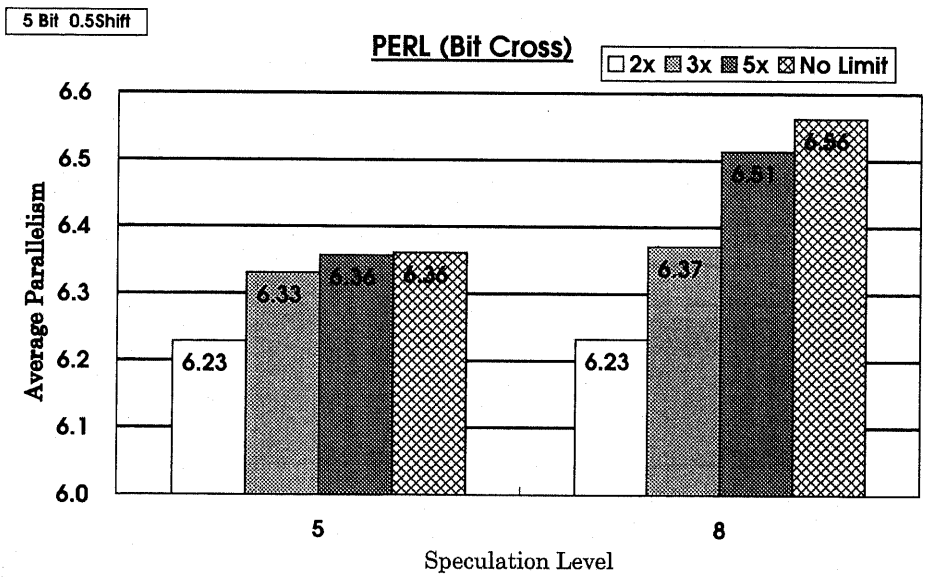


図 A.32: フェッチ命令数制限時の平均並列度 (PERL,5bit 履歴,0.5bit シフト,BC 方式)

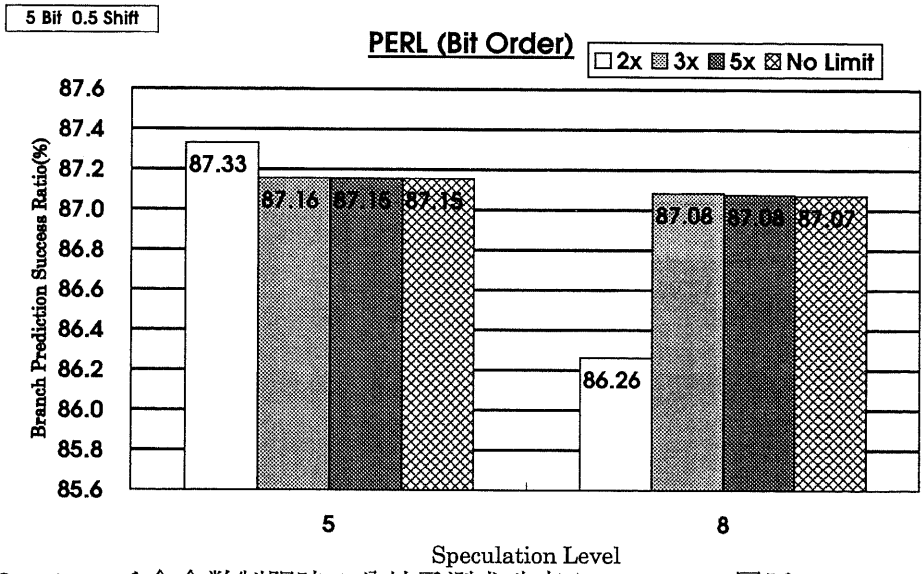


図 A.33: フェッチ命令数制限時の分岐予測成功率 (PERL,5bit 履歴,0.5bit シフト,BO 方式)

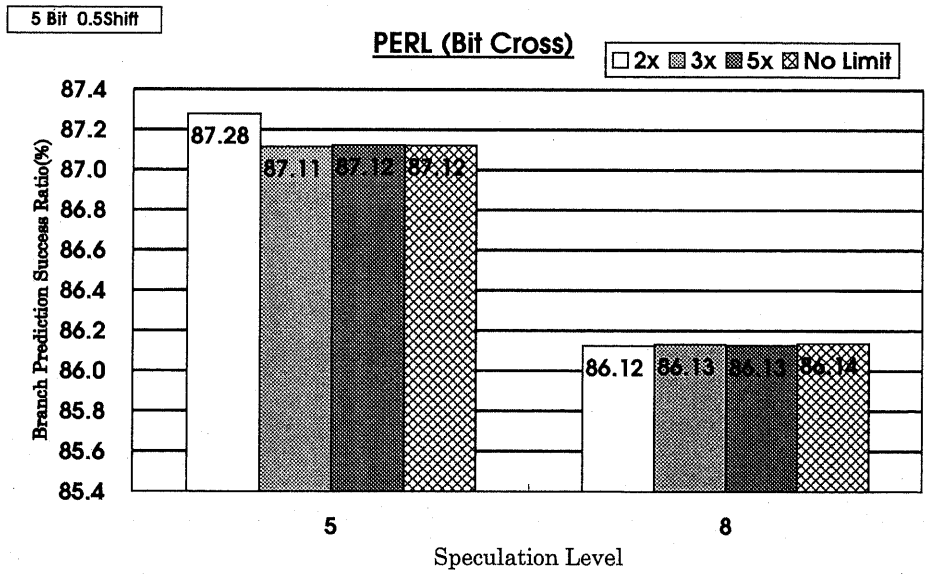


図 A.34: フェッチ命令数制限時の分岐予測成功率 (PERL,5bit 履歴,0.5bit シフト,BC 方式)

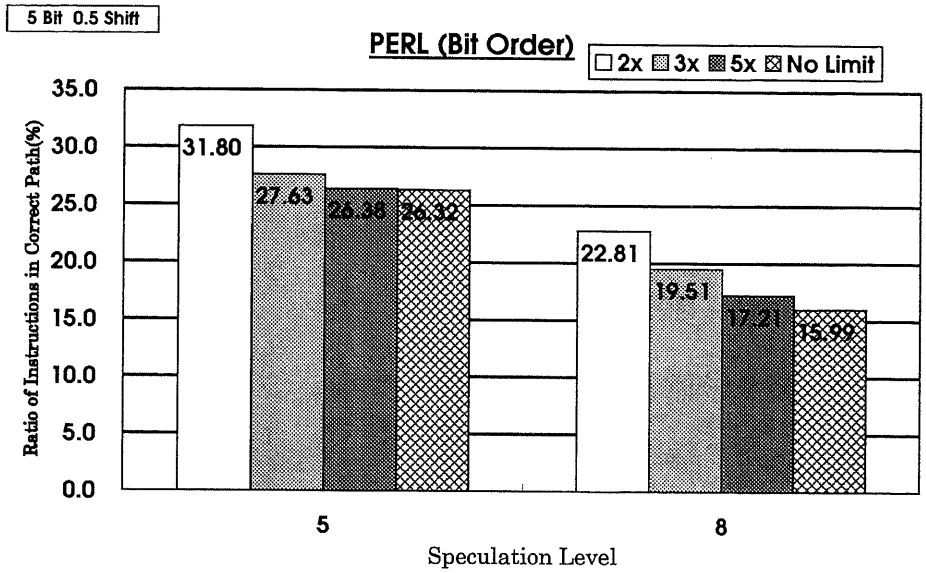


図 A.35: フェッチ命令数制限時の有効命令率 (PERL,5bit 履歴,0.5bit シフト,BO 方式)

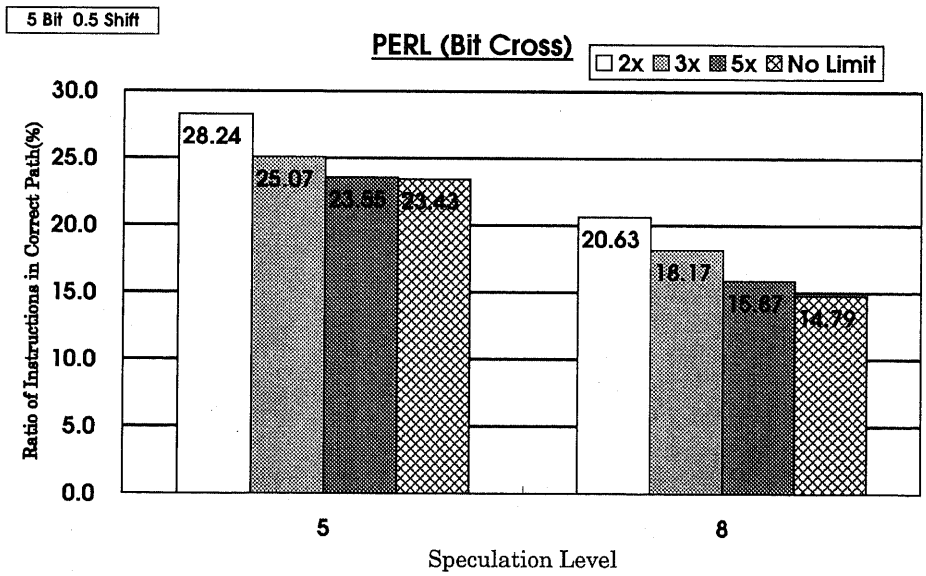


図 A.36: フェッチ命令数制限時の有効命令率 (PERL,5bit 履歴,0.5bit シフト,BC 方式)

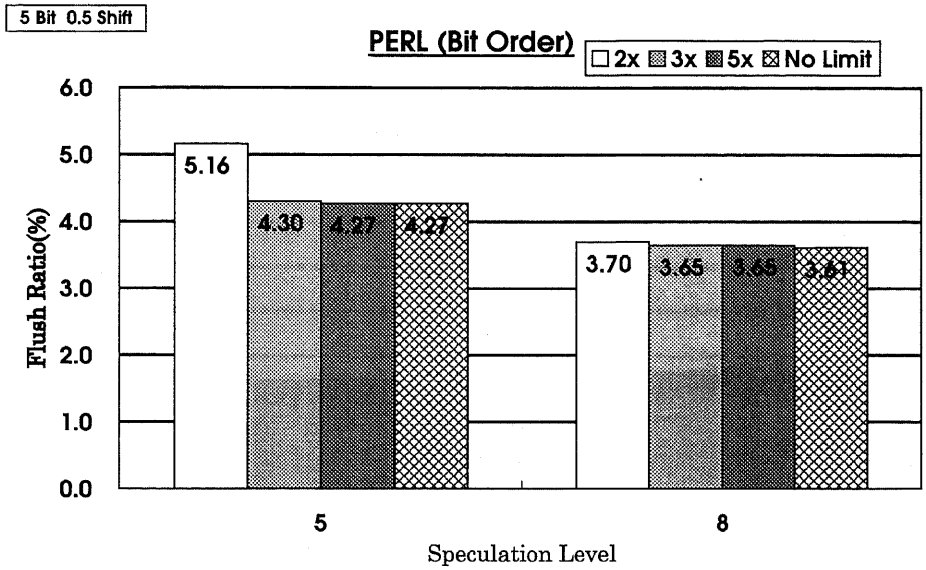


図 A.37: フェッチ命令数制限時のフラッシュ率 (PERL,5bit 履歴,0.5bit シフト,BO 方式)

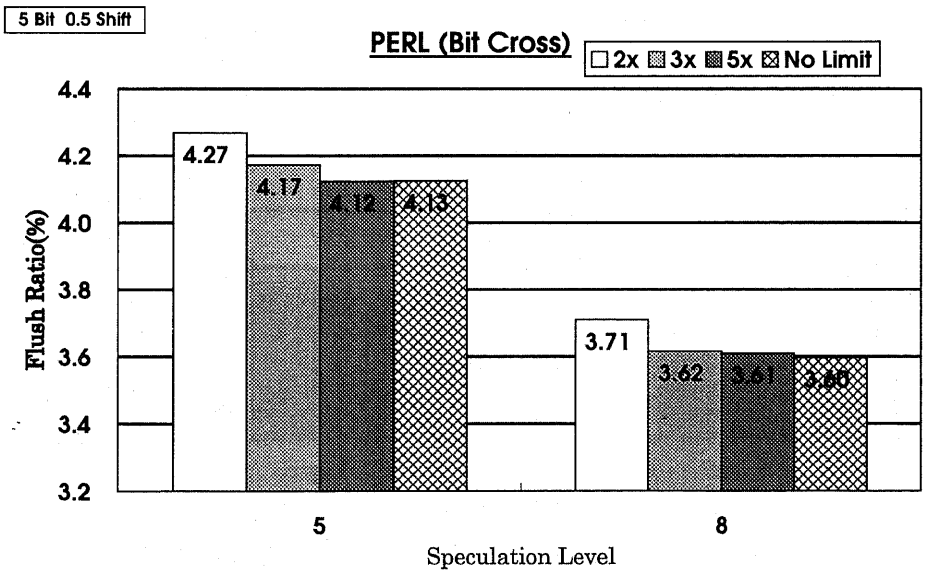


図 A.38: フェッチ命令数制限時のフラッシュ率 (PERL,5bit 履歴,0.5bit シフト,BC 方式)

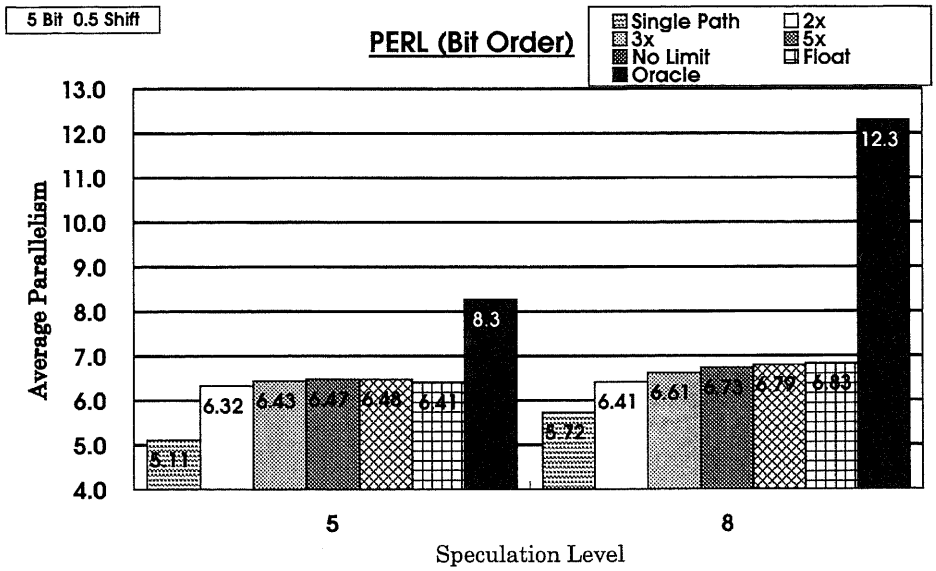


図 A.39: 様々な命令展開方式の平均並列度比較 (PERL)

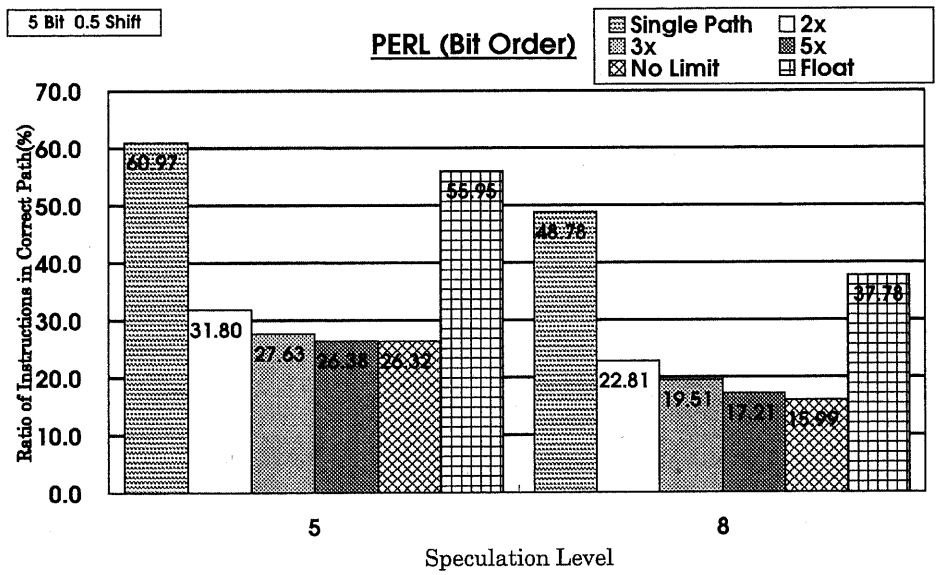


図 A.40: 様々な命令展開方式の有効命令率 (PERL)