計算機アーキテクチャ　105－6
ハイパフォーマンス
コンピューティング　50－6
（１９９４．　３．　１０）

Para$_{tool}$ によるスーパスカラプロセッサの評価

西本 晴子　志村 浩也　木村 康則

(株) 富士通研究所 プロセサ研究部

マイクロプロセッサのスーパスカラアーキテクチャにおいて、構成要素の違いにより全体性能へどの程度影響を与えるか定量的に解析した。

性能解析を行なうため、実行トレースに基づくスーパスカラ方式のプロセッサの性能評価ツール Para$_{tool}$ を開発し、スーパスカラ方式の命令発行機構、機能ユニットの構成、投機的実行における分岐予測、レジスタリネーミング、待機ステーションなどの効果を測定した。シミュレーション結果よりスーパスカラプロセッサにおいて性能向上に最も寄与する最適化手法は、分岐予測による投機的命令実行であるということが分かった。

Performance Evaluation of a Superscalar Processor Using Para$_{tool}$

Haruko Nishimoto　Kouya Shimura　Yasunori Kimura

The Processor Laboratory, FUJITSU LABORATORIES LTD.

1015, Kamikodanaka Nakahara-ward, Kawasaki 211, JAPAN

This paper reports a performance evaluation of a superscaler processor using Para$_{tool}$. There are some optimal techniques in processor design. We intend to measure the performance quantitatively and investigate how much the individual technique affects a total performance.

In order to analyze a performance, a fast performance evaluation tool - Para$_{tool}$ - has been developed. Para$_{tool}$ simulates a superscalar processor from dynamically execution traces and calculates an IPC (Instruction Per Cycle). We measured the improvement of IPC by individual technique - multiple issue, multiple functional units, register renaming, branch prediction, and out-of-order execution. From the results, we found the speculative execution by branch prediction is the most effective technique.

1 はじめに

近年，高性能マイクロプロセッサの性能向上手段としてスーパスカラ方式が主流になっている。スーパスカラ方式のプロセッサにおいては、性能とデバイスの物理的制約や回路設計、コンパイラ技術などの要素が相互に効率良く機能することが重要である。

また、スーパスカラプロセッサの最適化手法には様々なものが存在し、各々の手法が複雑に関係しあい全体性能に影響を及ぼす。そのため様々な構成を組み合わせたものの評価を行なう必要があり、評価項目は多岐に渡る。

そこで、実行トレースに基づく高速な性能評価ツール **Para$_{tool}$** を開発し [4]、スーパスカラプロセッサの各種最適化項目について評価を行なった。

以下では、第 2 章で **Para$_{tool}$** について簡単に説明する。第 3 章では、今回の評価で用いたスーパスカラプロセサのアーキテクチャについて述べる。次に第 4 章で、今回評価の対象とした命令発行数、分岐予測バッファ、レジスタリネーミング、待機ステーション、についての結果を報告する。最後に第 5 章で今回の評価結果から得られた知見をまとめる。

2 性能評価ツール Para$_{tool}$

2.1 Para$_{tool}$ の構成

Para$_{tool}$ は実行トレースベースに基づくソフトウエアシミュレータである。アプリケーションプログラムを実際にワークステーション上で実行し、その履歴情報を基にスーパスカラプロセッサの性能の算定を行なう。**Para$_{tool}$** の動作について、図 1に示す。

図 1 で、命令トレーサの部分は解析ツール SHADOW を用いた。これは Sun Microsystems, Inc. から提供された解析ツールで、命令レベルでプログラムの動作を追跡できる [2]。SHADOW は、SPARC™アーキテクチャの評価のために利用され、実行した命令、そのプログラムカウンタ、命令がアクセスしたメモリアドレスを生成する。**Para$_{tool}$** (並列度解析モジュール) は、SHADOW の出力と評価するスーパスカラアーキテクチャを規定するパラメータを入力として性能解析を行ない、結果をファイルに出力する。

Para$_{tool}$ はアプリケーションを単体で走らせる時と比較しておよそ 500～1000 倍程度の時間を要する。これはレジスタトランスファーレベルのシミュレーションに比べて非常に高速であると言える。評価速度は Sparc Station 2 上で実行した場合 20KIPS～40KIPS 程度である。

2.2 マシンモデル

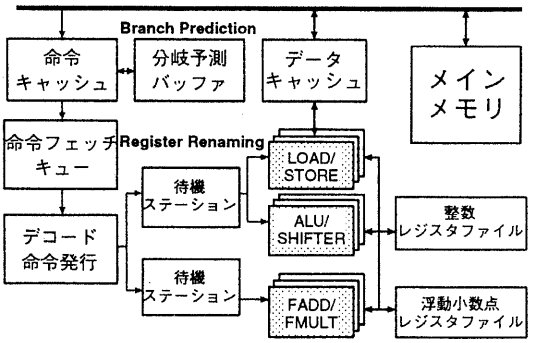

図 2: 基本マシンモデル

Para$_{tool}$ で解析できるスーパスカラプロセサの基本モデルを図 2に示す。複数の演算器を持ち、分岐予測バッファを用いた投機的実行 (speculative execution) をサポートし、非順序 (**out-of-order**) で実行を行なうことが特徴である。すなわち、命令キャッシュから供給された命令は、一旦命令フェッチキューに格納される。この時、分岐命令の taken/not-taken を予測して、命令のフェッチ方向を変える。命令デコードと同時に、逆依存や出力依存の解消のためレジスタの名前つけ替え (レジスタリネーミング) が行なわれ、待機ステーションに **in-order** で発行される。待機ステーション内の命令は、構造ハザードやデータハザードの解消したものから **out-of-order** で実行が開始される (**in-order** 実行のシミュレートも可能)。命令実行の完了は **out-of-order** であるが、プログラムの意味を変えないように、リオーダバッファ等を用いてコミットは **in-order** で行なわれる。

実験は、この基本モデルのパラメタを変化させることにより行なう。具体的に **Para$_{tool}$** では、表 1で示すようなパラメタ設定が可能である。

また、パイプライン各々のフェーズのレイテンシや各演算ユニットの実行レイテンシも設定できる。これらは、図 1のアーキテクチャパラメタ

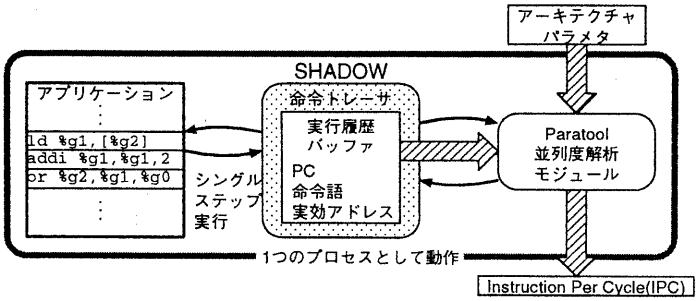

図 1: Para$_{tool}$ の構成

表 1: Para$_{tool}$ で設定できるパラメタ

in-order/out-of-order 実行
命令発行幅
命令供給幅
命令供給整列機構
キャッシュサイズ
リネーミング用代替レジスタ数
分岐予測バッファサイズ
ストアバッファエントリ数
待機ステーションエントリ数
機能ユニット個数
各種レイテンシ

表 2: Para$_{tool}$ で得られる統計情報

総実行命令
総実行サイクル数
命令フェッチ回数
命令キャッシュミス率
データキャッシュミス率
分岐命令頻度
BTB ヒット率
BTB 予測正当率
分岐予測成功率
レジスタリネーミング失敗率
待機ステーション稼働率
演算ユニット稼働率
サイクル当たり実行命令数

に対応し、コマンドラインから Para$_{tool}$ に渡される。また、実験の結果得られる統計情報を表 2に示す。

Para$_{tool}$ の利用者から見た時の特長をまとめると以下のようになる。

- アプリケーションプログラムを実際のプロセッサで実行し、そのトレース結果を基に解析を行なうため、非常に高速である。
- パラメタは全てコマンドラインから容易に設定や変更ができ、様々な最適化手法の評価が簡単に行なえる。
- アプリケーションプログラムは既存のものに手を加える必要なく利用できる。
- 各種の資源 (resource) を無限にあると仮定した測定が可能なため、理想的な場合の性能 (性能限界) の測定が可能である。

3 シミュレーション

3.1 ベンチマークプログラムとコンパイラ

シミュレーションには表 3に示すベンチマークプログラムを使った。整数系、浮動小数点系ともSPEC92 から任意に抜粋したものである。

コンパイラは、Cプログラムに対しては SunOS 4.1 に附属の cc、また FORTRAN プログラムに対しては f77 を用いた。最適化のレベルは O2 とした。

3.2 評価項目と評価条件

今回の評価では、スーパスカラに特徴的な以下の最適化項目について、その有用性、ハードウエアとして必要な量、組み込み方式の違いによる全体性能の差、などについて調べた。

1. 命令発行機構

表 3: ベンチマークプログラム

SPECint	SPECfp
008.espresso	015.doduc
022.li	047.tomcatv
023.eqntott	048.ora
026.compress	089.su2cor
072.sc	090.hydro2d
085.gcc	094.fpppp

2. 分岐予測バッファ

3. レジスタ・リネーミング

4. 待機ステーション

今回のシミュレーションでは、キャッシュや命令整列機構についての評価結果は省略した。また種々の評価において、性能の尺度としては IPC (Instruction Per Cycle：1 サイクルあたりに実行した命令数）の値を用いた。

また、Para$_{tool}$ では、注目している項目以外にも演算ユニットの数や各々の実行レイテンシについて値を設定しなければならない。このためのデフォルト値として表 4と表 5に示す値を使う。これを「基本モデル」と呼び、以後のシミュレーションでは特に違いを述べない限り全てこの値を使う。

表 4: 基本モデル

項目	設定値
実行形式	out-of-order
命令発行幅	4 命令
命令供給幅	8 命令
命令整列機構	なし
分岐予測バッファ	32 バイト、4 ウェイ 128 エントリ、2bit 予測
レジスタリネーミング	16 個/16 個（int/float） 4 個/4 個（int cc/float cc）
ストアバッファ	8 エントリ
キャッシュ	32K/32K (Data/Inst) (512 エントリ、2 ウェイ)
ミスヒットペナルティ	3 サイクル (2 次キャッシュを想定)
待機ステーション	集中型 16 エントリ

表 5: 演算ユニット数と各種レイテンシ

演算ユニット	個数
BR(分岐)	1
ALU(算術論理演算)	2
LDST(ロードストア)	1
SFT(シフト)	1
FADD(浮動小数点加算)	2
FMUL(浮動小数点乗算)	1
FDIV(浮動小数点除算)	1
演算	**レイテンシ (cycle)**
命令キャッシュ	2
命令供給	2
デコード	2
分岐予測	1
ロード命令	3
ストア命令	1
分岐命令	1
浮動小数点加算	4
浮動小数点乗算	4
浮動小数点除算	20
浮動小数点平方根	20
浮動小数点比較	2
その他	1

4 シミュレーション結果

4.1 命令発行機構

スーパスカラマシンでは、1 サイクルあたり複数の命令を複数の演算器に発行して、性能を向上させている。そこで、命令供給幅/命令発行幅と演算器の数の関係を調べた。発行幅と供給幅に関しては以下のように設定した。演算器の数については、ALU が 1 個と 2 個の場合について評価を行なった。この結果を図 3に示す。

実験 A 命令発行幅：2 命令、命令供給幅：4 命令

実験 B 命令発行幅：4 命令、命令供給幅：8 命令

実験 C 命令発行幅：無限、命令供給幅：無限

図中の X_aY は X が上述の命令発行幅と供給幅を示し、Y が ALU の個数を表す。この図 3から、B と C では IPC に顕著な違いは見られなかった。これは演算器の数を比較的小さく設定して実験を行なったためであり、この演算器数の設定では命令発行数は 4 命令で充分であることを示している。命令発行幅 4 命令供給幅 8 命令は、今日のプロセッサ開発においては充分実現可能であり、以後 B をベースモデルにして評価を続行する。

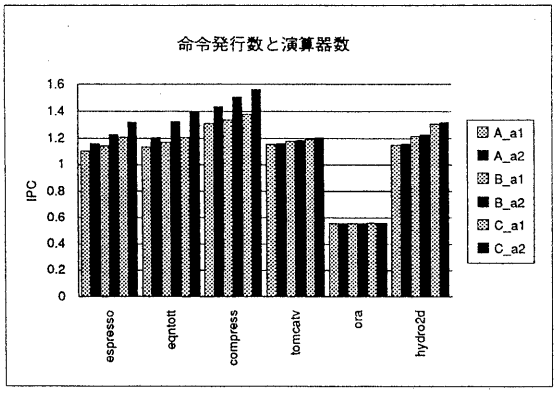

図 3: 命令発行幅と IPC の関係

4.2 分岐予測バッファ

Para$_{tool}$ では分岐予測を用いて投機的実行 (speculative execution) を行なう。分岐予測にはセット・アソシアティブ方式の BTB(分岐先バッファ: Branch Target Buffer) を用いて行なう。図 4に BTB の構成を示す。BTB の構成パラメタはブロックサイズ、エントリ数、ウェイ数、履歴ビット数でそれぞれ任意に設定できる。履歴ビット数には, 0bit (taken の分岐命令のみ BTB のタグに登録。not-taken になるとエントリから外す)、1bit (taken、not-taken ともに登録。一度外れれば予測の方向を変える)、2bit (taken、not-taken ともに登録し、二度続けて外れない限り予測の方向を変えない) の 3 通りの方法が選択できる。また、BTB を使った分岐予測以外に、分岐予測なしの場合と完全な (常に正しい) 分岐予測ができる理想的な場合の評価を行なうことも可能である。完全分岐予測とは、レジスタ相対も含め、全ての分岐命令に対して 100%の予測を行なうことができるとする。

シミュレーションでは、BTB のエントリ数とウェイ数を変えて IPC の変化を見た。また、BTB による予測を行なわない場合と、完全に分岐方向を予測できる場合 (分岐予測完全と呼ぶ) のデータも併せて取った。結果を図 5に示す。

この図中で、nopred とは分岐予測を全くしない場合、perf は完全な分岐予測を行なった場合の IPC である。サイズの略称で eXXwYY となっているのは BTB のバッファサイズを表し、XX がエントリ数で YY がウェイ数を表す。

図 5から以下のことが言える。まず、BTB の

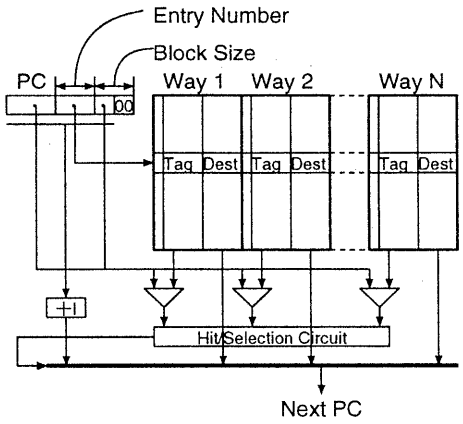

図 4: BTB の構成

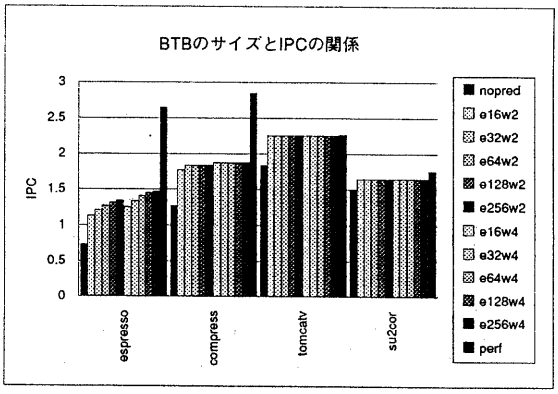

図 5: BTB と IPC の関係

物理的なサイズが同じ場合、エントリ数よりはウェイ数を多く取った方が効果が大きい。それはエントリの置き換え頻度が下がるため、BTB のヒット率がよくなり、その結果分岐先を正しくフェッチできることが多くなるからである。

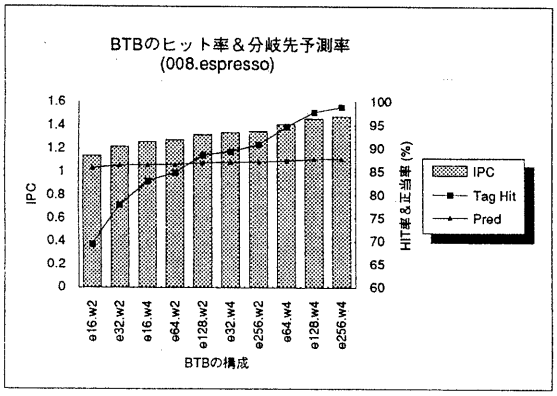

図 6: BTB の分岐正当率と HIT 率

また、整数系プログラムと浮動小数点系プログラムで BTB の効果に特徴がある。整数系プログラムでは BTB のサイズを大きくしても分岐予測完全の場合に近づかないのに対し、浮動小数点系プログラムでは比較的小さな BTB を用意するだけで分岐予測完全に近い効果を得ることが出来る。さらに整数系プログラムでは図 6から、BTB のサイズを上げることにより BTB のヒット率は向上するが、予測率はそれほど向上しない。これは、浮動小数点系プログラムではループ構造が多く、BTB の予測が当たりやすいのに対し、整数系プログラムでは分岐の偏りが小さく予測が当たりにくいためと考えられる。(図 7参照)

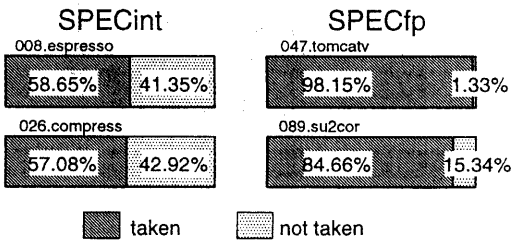

図 7: 分岐命令の偏り

4.3 レジスタリネーミング

リネーミングを行なう場合、命令セットで用意されているレジスタ数よりも多くの物理的なレジスタを用意する必要がある。Para$_{tool}$ では代替レジスタ数を指定して性能を評価することができる。レジスタ以外に条件コード (condition code) のリネーミングを行なうこともでき、整数レジスタ、整数条件コード、浮動小数点レジスタ、浮動小数点条件コードについてそれぞれ代替レジスタ数を設定することができる。また、物理的なレジスタが無限にあるという理想的な条件について性能評価することも可能である。

リネーミングは命令デコードフェーズで行なう。利用できる物理的なレジスタに空きがない場合は、デコードフェーズでその命令のデコードが止められ、全ての後続命令のデコードがその次のサイクル以降に持ち越される。一方、リネーミングを行なわないインタロック制御を選択した場合は、命令はデコードされた後、待機ステーション内で逆依存や出力依存がないことを調べてから演算フェーズに移される。従って代替レジスタの数が少ない場合は、リネーミングを行なわない場合より性能が悪くなると考えられる。

図 8にリネーミング用の代替レジスタの個数を変化させた場合の IPC を示す。図右の数字は代替レジスタの個数を示す。

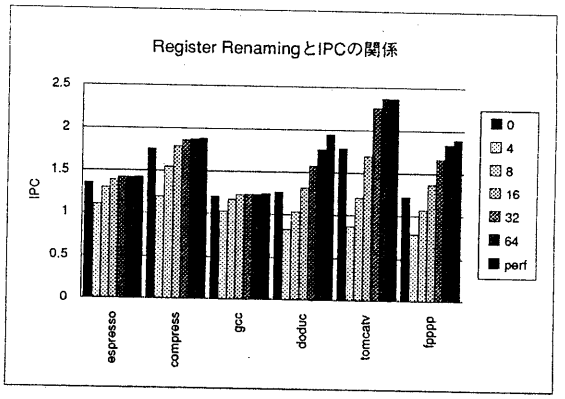

図 8: リネーミングと IPC の関係

この図で特徴的なのは、少数の代替レジスタでリネーミングを行なった場合、レジスタリネーミングを行なわない場合に比べて IPC が下がってしまうことである。この原因はリネーミングの実現方式にある。Para$_{tool}$ では前述のように、レ

ジスタリネーミングを実現する場合、命令中の全てのレジスタのリネーミングを行なおうとする。この場合、リネーミング用の代替レジスタ数が少ないとリネーミング用のレジスタが枯渇し、実行が中断してしまう。これはその命令がデコードの段階で止まってしまい、後続命令が発行できなくなるためである。一方、リネーミングを行なわない場合には、命令間にデータ依存が存在する時のみ、実行が中断する。これはその命令が待機ステーション内で依存が解消されるまで待つことになるが、命令の発行は止まることなく続けられる。この差が図 8の IPC の差に反映していると考える。また図 9にリネーミング失敗率を示す。失敗率とは、総実行サイクル数に対して、リネーミング用レジスタに空きがなく命令のデコードが止まったサイクル数の割合である。この図からも、リネーミングの失敗が主たる原因であることが分かる。

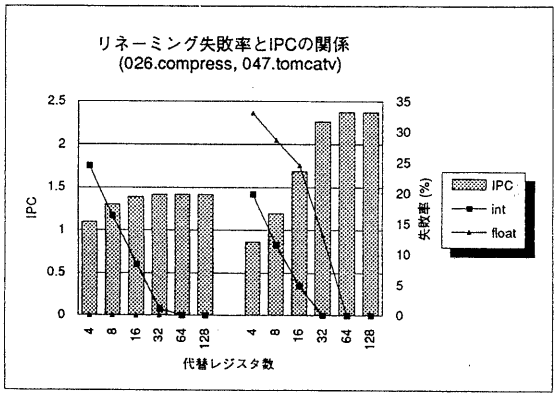

図 9: リネーミングの失敗率

さらに、代替レジスタを多くしていった場合の IPC の向上率は、浮動小数点系のプログラムの方が大きい。これは、整数系のプログラムでは多くの命令が 1 サイクル程度で実行できるのに対し、浮動小数点系プログラムでは実行に複数サイクルかかる命令が多いためで、それらの命令間の依存を解消できることが向上率を大きくしているためと考える。

4.4 待機ステーション

Para$_{tool}$ では、待機ステーション (Reservation Station、以下 RS) の構成を柔軟に設定できる。演算ユニット毎に RS のエントリ数を設定できる他、いくつかの演算ユニットの間で RS を共有するように設定することが可能である。例えば、整数演算と浮動小数点演算用に 2 つの RS を用意するというような設定ができる。全ての演算ユニットの RS を 1 つの RS に共有させることで、集中型の RS をシミュレートする。また、RS のエントリ数を無限に設定した理想的な場合のシミュレートもできる。

RS では、リネーミングを行なう場合は命令の真依存だけを調べて演算の発火を行なう。インタロック制御を行なう場合は、RS において更に出力依存と逆依存を調べて演算を発火する。依存関係の解消は全て RS で解消するため、演算パイプライン中で演算ストールは起きない。なお、RS が一杯の場合は、命令デコードフェーズで命令はブロックされる。

RS のサイズを変えて IPC を測定した結果を図 10に示す。今回は集中型 RS のバッファサイズについてのみ評価した。図右の数字は RS のエントリ数を示す。

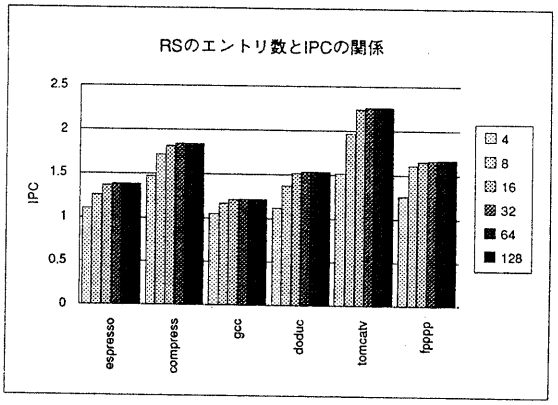

図 10: RS と IPC の関係

図 10からも、RS のサイズが大きくなった場合の IPC の向上率は、浮動小数点系のプログラムの方が大きい。これは、命令ウインドウを大きく取ることにより、より広い範囲での out-of-order 実行を可能にしているためである。一方、整数系のプログラムについては、比較的小さなサイズの RS で IPC 向上率が飽和している。これはそもそもプログラムに真依存が多く存在して、ウインドウを大きくしても効果がないことを示している。ただし、現在はコンパイラとして SunOS の cc を使っており、将来広域命令スケジューリングを行なうコンパイラで生成したコードで実験すれば、IPC が飽和する RS のサイズが大きくなると考える。

また図 11に RS の稼働率と IPC の関係を示す。RS 稼働率とは、総実行サイクル数に対して、RS のエントリに空きがなく命令の発行が止まったサイクル数の割合のことである。この図から以下のことが言える。

整数系プログラムでは RS のエントリを大きくとると、RS のエントリが飽和する時間が減少し、演算器に対してはほぼ充分な命令発行ができていると思われる。しかし IPC がそれほど上がらないのは、そもそも命令供給数が少なく RS が空いている時間の方が長くなっているからだと考えられる。つまり命令発行数を多くすると IPC は向上する可能性がある。また浮動小数点系プログラムに対しては、RS のエントリをかなり大きくしても RS のエントリが空く時間は少なく、そのため IPC は上がらなくなる。これは浮動小数点演算は時間がかかるため演算器の稼働率が上がり、演算器に発行されることができない命令が RS 内で長い間留まってしまうためと考える。したがってこの場合、同じ RS のサイズでも演算器の数を増やせばより IPC は向上すると考えられる。

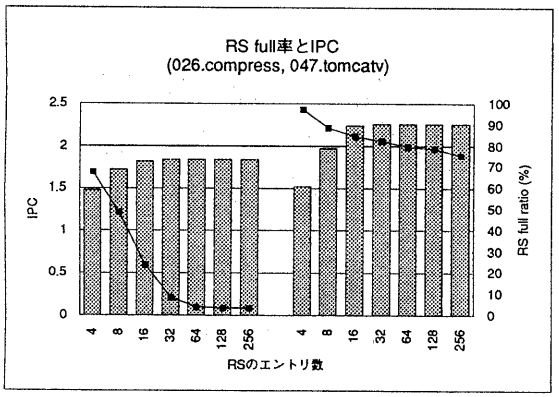

図 11: RS 稼働率と IPC の関係

5 おわりに

スーパスカラプロセッサの性能評価ツール **Para$_{tool}$** を使って、最適化手法の構成の違いが全体性能へどのような影響を与えるかについて評価した。その結果以下のような知見が得られた。

- BTB は、浮動小数点系のプログラムでは、かなり小さな BTB でも分岐の正当率は高いが、元来分岐のペナルティの割合が大きくないためあまり効果がない。整数系のプログラムでは、分岐ペナルティの割合が大きいためあまり正当率が高くなくても効果が得られる。
- レジスタリネーミングは実現の方式にもよるが、ある程度の大きさ (命令から指定できるレジスタ数と同程度) の代替レジスタを用意しないと効果が見られない。
- 待機ステーションは、浮動小数点系のプログラムに対して効果が大きい。整数系のプログラムに対してはプログラム自身の持つ並列性が比較的小さいと予想されることやコンパイラの影響により、期待した程の効果はなかった。

今回は、かなりハードウエアを限定してシミュレーションを行なった。演算器の数や命令発行数を変えれば、また違った傾向を示すことも考えられる。今後は、コンパイラでの最適化やハードウェアでの実現コストをふまえた実験、評価を行なって行きたいと考えている。

謝辞

本研究を遂行するにあたり御指導御支援して下さった、林部門長、津田部長、服部主管研究員をはじめとする関係者各位に感謝します。また、論文の草稿を読んで数多くの貴重なコメントを下さった ICOT の久門耕一氏に厚く感謝の意を表します。

参考文献

[1] Mike Johnson : Superscalar Microprocessor Design. Prentice Hall Series in Innovative Technology, Prentice Hall, 1991.

[2] Sun Microsystems, Inc. : Introduction to SHADOW, Peter Yan-Tek Hsu, 1989

[3] FUJITSU MICROELECTRONICS,INC. : SPARC™ REFERENCE GUIDE.

[4] 志村, 西本, 江口, 木村: スーパスカラプロセッサの性能評価 – **Para$_{tool}$**–. 情報処理学会研究会報告 93-ARC-102-1, pp.1–8, Oct. 1993.