計算機アーキテクチャ　102－1
（1993．10．21）

スーパスカラプロセッサの性能評価
－Para$_{tool}$－

志村 浩也　西本 晴子　江口 剛　木村 康則

(株) 富士通研究所 プロセサ研究部

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$_{tool}$を開発した．プロセッサ開発において，マイクロ・アーキテクチャの設計およびコンパイラの開発には性能評価は必須であり，高速で柔軟な性能評価ツールが必要とされる．Para$_{tool}$の開発には，シミュレーションの精密さよりも，評価の高速性に重きを置いた．本稿では，まず実行トレースを基にPara$_{tool}$がどのように性能評価を行なうかを説明し，扱うことのできるプロセッサのモデルについて説明する．トレースベースのシミュレータによる限界もあり，その問題点を述べ，最後に参考結果を示す．

Performance Evaluation of a Superscalar Processor

Kouya Shimura　Haruko Nishimoto　Takeshi Eguchi　Yasunori Kimura

PROCESSOR LABORATORY, FUJITSU LABORATIRIES LTD.

1015, KAMIKODANAKA NAKAHARA-KU, KAWASAKI 211, JAPAN

This paper describes the performance evaluation tool of a superscaler processor. The performance evaluation is indispensable for designing a processor and developing the compiler. So, we developed the fast and flexible performance evaluation tool – Para$_{tool}$– . Here we explain how to evaluate the performace from dynamically executed traces and the machine model which Para$_{tool}$ can handle. Then the problem caused by the trace simulator and some preliminary results are shown.

1 はじめに

近年，マイクロプロセッサの性能向上手法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開発において，マイクロ・アーキテクチャの設計には，性能とデバイスの物理的制約や設計の困難さなどの多くのトレードオフがある．そのため，プロセッサの設計の初期段階において予め性能を評価することが必須である．また，プロセッサとコンパイラを合わせたシステム性能を向上させるためには，実際のシステムが稼働する前に，十分な性能評価を行なうことが大切である．

従来，このような性能評価は，レジスタ・トランスファー・レベルでのアーキテクチャ・シミュレータを作成することにより行なわれていた．スーパスカラ方式のプロセッサの最適化手法には様々なものが存在し，評価項目は多岐に渡る．更に，各々の評価項目はお互いに影響を与えるため，様々な構成を組み合わせたものを評価する必要がある．従って性能評価を少しでも高速に行なうことは非常に重要である．

そこで，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評価ツール Para$_{tool}$を開発した．Para$_{tool}$の開発には，シミュレーションの精密さよりも，評価の高速性に重きを置いた．第2章で Para$_{tool}$について詳しく述べる．第3章では，Para$_{tool}$で扱うプロセッサのモデルについて説明する．トレースベースのシミュレータによる限界もあり，第4章でその問題点を述べる．第5章で参考結果を示す．

2 Para$_{tool}$

Para$_{tool}$は SHADOW の解析モジュールとして開発した．SHADOW は Sun Microsystems, Inc. から提供された解析ツールであり，命令レベルでプログラムの動作を追跡でき，SPARC™アーキテクチャの評価に利用されている．図 1に Para$_{tool}$の様子を示す．

SHADOW を使って解析を行なう際，アプリケーション・プログラムと命令トレーサ，それに Para$_{tool}$が UNIX の 1 つのプロセスのメモリ空間に同時にマッピングされる．命令トレーサと Para$_{tool}$はアドレスの高位置にマッピングされるため，アプリケーションはこれらと無関係に動作できる．実行トレーサは SPARC のレジスタ・ウィンドウにより処理の切替を行なうため，コンテキスト・スイッチの負荷がなく非常に高速に実行トレースを生成する．評価に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．評価速度は Sparc Station 2 上で実行した場合 20KIPS～40KIPS 程度である．

各種の最適化手法に関するパラメタは，アーキテクチャ・パラメタとしてコマンド・ラインから Para$_{tool}$に渡す．命令トレーサは SPARC のアプリケーション・プログラムを 1 命令ずつ実行し，UNIX のユーザ・モードで実行された全ての命令の実行履歴をバッファに貯めていく．バッファが一杯になると，命令トレーサは Para$_{tool}$を呼びだす．プログラム・カウンタ，命令語，それにロード/ストア/分岐等の命令の実効アドレス実行が，履歴として Para$_{tool}$に受け渡される．Para$_{tool}$はこれらの逐次的に実行された履歴を元に，スーパスカラ方式で実行した場合の IPC(Instruction Per Cycle: 1 サイクル当たりの平均実行命令数) を計算する．IPC 以外に，アーキテクチャ評価のため表 1に示すような統計情報を測定することもできる．

表 1: 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
命令フェッチ・レート
レジスタ・ウィンドウ overflow/underflow 回数
命令キャッシュ・ミス率
データ・アクセス回数
データ・キャッシュ・ミス率
ストア・バッファ・フル率
分岐命令頻度
レジスタ相対分岐頻度
BTB ヒット率
BTB 予測正解率
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・フル率
演算ユニット・フル率

Para$_{tool}$の特徴をまとめると，

- 命令を実際のプロセッサで実行してそのトレース結果を基に IPC を計算するため，非常に高速

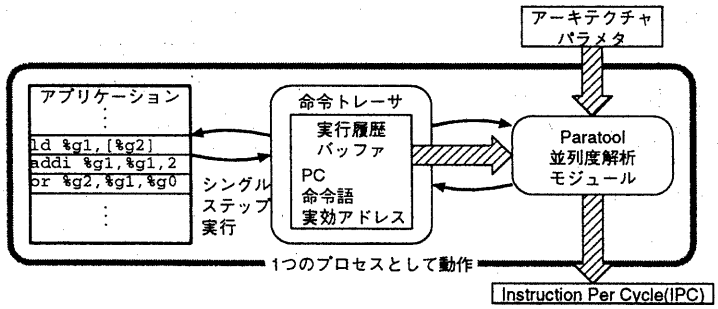

図 1: SHADOW による並列度解析

- パラメタは全てコマンド・ラインから設定でき，様々な最適化手法の評価が簡単
- 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) を無限に設定可能なため，理想的な場合のシミュレートが可能

のようになる．一方，次のような制限がある．

- SPARC 以外のプロセッサに適用不可能
- fork するアプリケーションを評価不可能
- system call 中の評価不可能

3 評価モデル

Para$_{tool}$は SPARC プロセッサをスーパスカラ方式で実行した場合の動作をシミュレートする．図 2に評価モデルのパイプライン構成を示す．分岐予測による投機的実行 (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．

分岐予測ミス，割り込みなどによる復元処理は reoder buffer と future file によるものを仮定している．復元処理の際はパイプラインのフラッシュを行なう．

表 2にパイプラインの各々のフェーズのレイテンシを示す．これらのレイテンシは任意に設定することが可能である．また，演算バイパス機能のある/なしについてシミュレートすることができる．演算結果バイパスのレイテンシは，演算結果が得られてから何サイクル後にその結果を使う演算が開始できるかを表す．

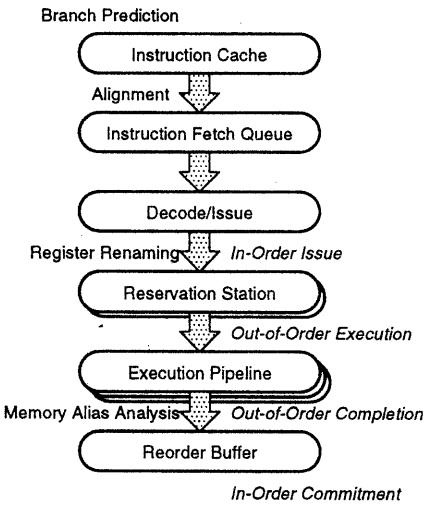

図 2: 評価モデルのパイプライン構成

表 2: パイプラインのレイテンシ

レイテンシの種類	cycle
命令キャッシュ・アクセス	2
BTB のアクセス	1
命令のフェッチ	1
命令デコード/発行	1
オペランド・フェッチ	1
演算結果バイパス	0
レジスタ書き戻し	1
キャッシュ書き戻し	2

3.1 命令供給機構

命令フェッチ幅 (1 サイクルに命令キャッシュから供給できる命令数) を任意に設定できる．その他に整列機構のある/なしや分岐予測機構について設定可能である．命令キャッシュのラインクロスに関しては考慮していない．

図 2で示した Instruction Fetch Queue(命令バッファ) はダブルバッファであり，命令フェッチ幅の倍の命令を保持することができる．命令バッファに命令フェッチ幅以上の空きがあれば，命令キャッシュから次の命令列が取り出される．

3.1.1 整列機構

命令の整列 (alignment) については，整列を行なう場合と行なわない場合について評価することが可能である．図 3は命令フェッチ幅が 4 の場合の命令フェッチの整列機構を説明したものである．

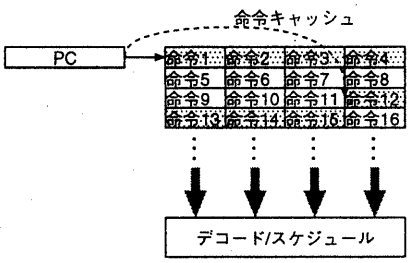

図 3: 命令フェッチ・アラインメント

命令キャッシュの構成によっては，命令キャッシュから一度に取り出すことのできる命令に制限がある．命令フェッチ幅が 4 命令で整列機構がない場合，4 命令毎に境界ができる．この場合，図 3において，命令 1～4 は同時にフェッチ出来るが，命令 3～6 は境界をまたぐため同時にフェッチ出来ない．従って，命令 12 への分岐が起きると命令 9～12 をフェッチするが，そのうち命令 9～11 は捨てられるため性能の低下を招く．整列機構によって境界にまたがる命令 12～15 が同時にフェッチできるようになる．

3.1.2 分岐予測

Para$_{tool}$では分岐予測を用いて投機的実行 (speculative execution) を行なう．分岐予測が外れた際は，パイプライン中の命令の無効化を行ない，分岐の条件コードが確定した時点から新たに命令のフェッチを開始する．命令の実行制御は out-of-order のため条件コードがいつ確定するかが動的に変化する．従って分岐ミスによるペナルティは一定ではない．

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，セット数，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 の分岐のみ BTB のタグに登録), 1bit，2bit の 3 通りの方法が選択できる．2bit の場合，始めて BTB に登録するときに 2 という値を履歴情報に登録する．以降，履歴情報を，taken の場合 1 つ値を増やし，not taken の場合 1 つ値を減らす．履歴情報が 2 以上のとき taken と予測し，1 以下のときは not taken と予測する．

BTB に登録できる分岐命令は，PC 相対分岐と call 命令である．命令フェッチ時に分岐先アドレスが定まらないサブルーチンからのリターンや多方向分岐等のレジスタ相対分岐は登録しない．Tag には分岐命令のアドレスを登録し，Dest に分岐先のアドレスを登録する．登録時の置き換えアルゴリズムはLRU 方式である．BTB がミスヒットした場合は，分岐しない方向に命令のフェッチを行なう．

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．予測なしの場合は常に not taken の方向に命令をフェッチする．完全分岐予測の場合，レジスタ相対分岐についても予測できるものとして解析を行なう．

3.2 命令発行/Renaming

命令のデコード・フェーズで命令の発行 (Issue) を行なう．命令発行の幅は任意の値に設定可能である．

命令間の依存関係によるハザードを解消するための register renaming の手法の評価が可能である．renaming を行なわない，インタロック制御についても評価できる．

renaming 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

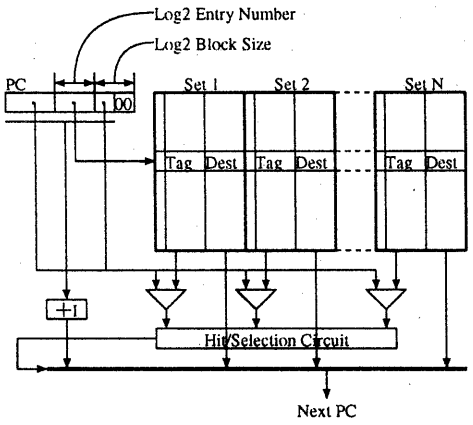

図 4: BTB の構成

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 renaming 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．

renaming 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が止められ，後続の全ての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，renaming 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，renaming を行なわない方が性能がよい場合が考えられる．

3.3 待機ステーション

RS(Reservation Station:待機ステーション) の構成には集中型，分散型の 2 つの方式がある．

集中型 命令の種類に関係なく 1 つの RS により演算の発火を制御する

分散型 各々の演算ユニット毎に RS を用意する

RS のエントリ数が同じ場合は，集中型の方が分散型に比べると RS を有効に利用できるため性能はいいが，ハードウェアの実現コストは高い．

Para$_{tool}$では，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 のエントリ数を無制限にした理想的な場合の評価もできる．

RS では，renaming 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の演算ストールはない．

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．

3.4 演算ユニット

Para$_{tool}$で用意した演算ユニットの種類を表 3 に示す．演算ユニットの個数はそれぞれ個別に設定可能である．また各々の演算ユニットの個数を無制限に設定することもできる．

一般的には，レジスタのポート数や内部バスなどの制限のため各種の演算を 1 つのパイプラインで構成する．しかし，現在の Para$_{tool}$ではこのような構成はサポートしていない．従って算術論理演算とシフトを同一の演算パイプラインで行なうようなプロセッサのシミュレーションを行なうことはできない．

表 3: 演算ユニット

演算ユニット名	説明	個数
BRU	分岐	1
LSU	ロード/ストア	2
ALU	算術論理演算	3
SFT	シフト	1
FADD	浮動小数点加算	1
FMUL	浮動小数点乗算	1
FDIV	浮動小数点除算	1

3.5 演算パイプライン

各命令について演算に必要なサイクル数(レイテンシ)を設定することができる．表4に指定できる演算の種類とそのレイテンシを示す．このレイテンシの値は，オペランドがそろって演算結果が得られるまでに必要なサイクル数である．

Para$_{tool}$では，解析処理を高速に行なうために，演算パイプラインのストールは一切考慮していない．このため例えばロード命令においてキャッシュ・ミスが起こった場合でも，後続のロード/ストア命令はブロックされることなく演算パイプラインを流れる．

同様に浮動小数点除算は，実際のプロセッサではパイプライン処理できないことが多いが，Para$_{tool}$では，除算命令を実行中に次の除算命令をブロックするようなことはできない．

表 4: 演算のレイテンシ

命令の種類	レイテンシ (cycle)
ロード命令	3
ストア命令	1
PC 相対分岐命令/call 命令	1
レジスタ相対分岐命令	1
浮動小数点加算	4
浮動小数点乗算	4
浮動小数点比較	2
浮動小数点と整数の変換	2
浮動小数点除算	20
浮動小数点平方根	20
その他の命令	1

3.6 キャッシュ

現在のモデルでは，1次キャッシュのみをサポートしている．キャッシュの構成は命令キャッシュ，データ・キャッシュ別個に設定可能である．命令とデータを共用するユニファイド・キャッシュはサポートしない．キャッシュの構成については，ブロックサイズ(ラインサイズ)，エントリ数，セット数(ウェイ数)をそれぞれ任意に設定可能である．ブロックを更に分割して管理するサブブロックの概念はない．置き換えアルゴリズムはLRU方式のみをサポートする．キャッシュ・ミスのペナルティについては任意の値を設定することができる．この値を ロードの実行サイクルに加算することで，キャッシュ・ミスをシミュレートする．ストア・バッファを持つためストア命令のミス時のペナルティはない．また，ストア時のミスヒットは，キャッシュ・ブロックを必ず割り当てられる．

3.5節で述べたように，LSUユニットのパイプラインのストールをシミュレートできないため，ノン・ブロッキング・キャッシュとなる．すなわち，キャッシュ・ミスが連続して(同時に)起きたとしても，キャッシュのミスのペナルティは一定である．また，キャッシュとメモリ間の転送サイクルなどは考慮していない．

3.7 ストア・バッファ

ストア命令は必ずストア・バッファを経由してキャッシュ・メモリにデータを書き込む．ストア・バッファのエントリ数は任意に設定できる．

キャッシュへのデータの書き込みの順番は必ずin-orderで行なう．キャッシュへデータを書き込むとストア・バッファを解放して後続のストア命令で利用できるようになる．ストア・バッファが一杯になると後続のロード/ストア命令はRSでブロックされる．

ストア・バッファを用いて，メモリ・エイリアス解析を行なう．ロード命令は，アドレス計算後にストア・バッファを走査してアドレスの衝突を確かめ，衝突した場合はストアの実行終了を待った後レジスタに値をセットする．LSUユニットを複数個設定した場合，ストア・バッファによりアドレスの衝突がない限り複数のロード/ストア命令が同時に実行可能である．またロード/ストア命令の実行順序の逆転もあり得る．

4 問題点

現在分かっているPara$_{tool}$による評価の問題点について説明する．

4.1 浮動小数点レジスタ

SPARCの浮動小数点レジスタは，1つのレジスタが単精度，2つの連続するレジスタを使って倍精度を表す．しかし，Para$_{tool}$では単精度と倍精度とのレジスタを区別して扱っていない．

4.2 分岐ミス時の副作用

Para$_{tool}$は実行トレースに基づいた解析を行なうため，分岐予測において予測が外れた場合に外れた方向の命令の実行トレースが得られない．従って本来は間違って実行するべき命令についての解析は全く行なわない．このためプロセッサ内に副作用を残す場合において，正確に動作をシミュレートするができないということである．副作用としては，

- BTB への登録
- renaming テーブル
- キャッシュの置き換え

などがあるが，実際のプロセッサでは，予測ミス時の動作としてこれらを壊す方へ働くと考えられるので，Para$_{tool}$による評価は実際のプロセッサより高めに出ると推測される．

4.3 各種テーブルへの登録のタイミング

Para$_{tool}$は実行トレースを 1 命令ずつ逐次 (in order) に解析する．従って，各種テーブルなどの更新も逐次に行なう．このため命令をフェッチした段階で，各種テーブルなどの更新がなされてしまうような動作になる．一方，実際にプロセッサで実行した場合，テーブルへの更新は実行フェーズで行なわれる．命令の実行には何サイクルも必要であるし，実行順序も Out-of-Order であるので，Para$_{tool}$は実際のプロセッサの動きと異なった解析を行なう場合がある．例えばキャッシュについては，同一キャッシュ・ラインにアクセスするロード命令が連続していて最初の命令がキャッシュ・ミスした場合，以下のように解析してしまう．

(1) 最初のロード命令を解析．キャッシュミス．タグ・テーブルに登録．ロードの実行サイクルを加算．

(2) 次のロード命令を解析．テーブルに登録されているのでヒット．ミスのペナルティがなく最初のロード命令より先に実行終了．

本来ならば後続のロード命令は先のロード命令のキャッシュ・ミスが終った後に実行できる筈である．

BTB に関しても同じような問題があるが，分岐予測に失敗した場合は命令のフェッチからやり直すため殆んど影響はないと考えられる．

4.4 レジスタ・ウィンドウ

SPARC アーキテクチャの特徴としてレジスタ・ウィンドウがある．レジスタ・ウィンドウのオーバフロー/アンダーフローが起きた際，本来はスタックへレジスタのセーブ/リストアを行なう必要がある．このためのオーバヘッドは現在特に考慮していない．

5 参考結果

SPEC92 ベンチマークから SPECint(整数系) を使って，簡単な評価を行なってみた．表5に示したパラメタを基準にして，命令発行幅，RS のエントリ数，演算ユニット数を変化させ測定した．コンパイラには SunOS4.1 に附属のものを使用した．

表 5のプロセッサの構成はかなり強力な構成である．しかし，図5を見ると強力な構成にも関わらず並列度は 2 に満たないものが殆んどである．やはり整数系のプログラムではアプリケーション自体に内在する並列度が低いことが伺える．図 6 からは，この構成においては待機ステーションのエントリ数は 16 で十分であることが分かる．また，図 7を見ると，演算ユニットを増やしてもあまり性能には貢献していない．

表 5: 基準パラメタ

レイテンシ	表 2，表 4を参照
演算ユニットの個数	表 3を参照
命令フェッチ幅	8inst.
命令発行幅	4inst.
BTB	128entry, 4set, 2bit 履歴
代替レジスタ数	整数:32, CC:8
待機ステーション	集中型 32entry
ストアバッファ	8entry
命令キャッシュ	64bytes/line, 128entry, 4set (32Kbytes)
データキャッシュ	64bytes/line, 128entry, 4set (32Kbytes)
キャッシュ・ミス ペナルティ	3cycle

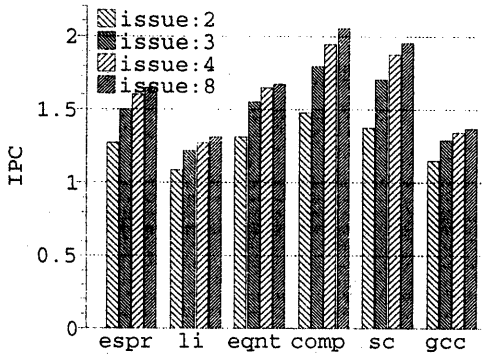

図 5: 命令発行幅

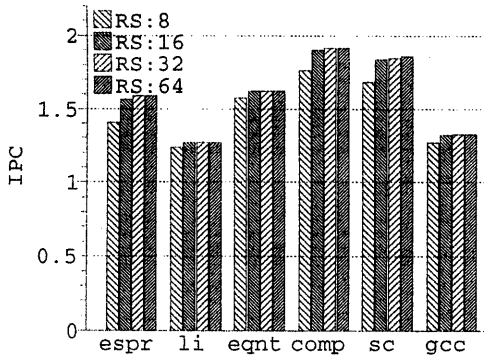

図 6: 待機ステーション・エントリ数

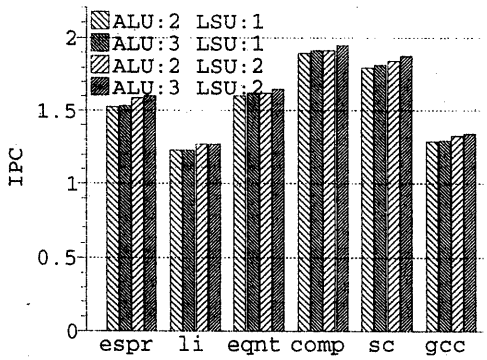

図 7: 演算ユニット数

6 おわりに

スーパスカラプロセッサの高速性能評価ツール，Para$_{tool}$の開発を行ない，Para$_{tool}$の扱うプロセッサのモデルと参考結果を示した．

Para$_{tool}$は現実のプロセッサを抽象化したモデルを扱うため，得られる性能は正確に実際のプロセッサの反映しているとはいえない．また，いくつかの問題があることも示した．しかし，シミュレーションの精密さと評価時間はトレード・オフであり，現在扱っているモデル程度が丁度良いと考えている．これ以上に精密な性能評価が必要ならば，レジスタ転送レベルのシミュレータを用意すべきであろう．

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踏まえて評価データを採取し，最適なアーキテクチャ構成を検討して行きたい．

末筆ながら，本研究を遂行するにあたり御指導，御支援して下さった関係各位に厚く感謝の意を表します．

参考文献

1) Mike Johnson : Superscalar Microprocessor Design

2) FUJITSU MICROELECTRONICS,INC. : SPARC™ REFERENCE GUIDE