多数演算器方式における演算器利用率の検討

吉瀬謙二[†]　中村友洋[†]　辻　秀典[†]
安島雄一郎[†]　田中英彦[†]

本稿では、非常に多数の演算器をチップ上に集積し、それらの接続形態を動的に変更することでプログラムの最適な実行を目指す多数演算器方式の演算器利用率を検討する。多数演算器方式のフェッチ・ステージでは投機的に分岐成立と不成立の両方の制御流をフェッチするEager Executionにより高い並列性の抽出を目指すが、一方で演算器の利用率を高めるために、実行する確率の低い制御流の枝刈りが必要となる。枝刈りをおこなうEager Executionの評価結果より、フェッチする命令数の爆発的な増加を抑えながら、逐次実行と比較し7.2倍の性能向上率が達成できることがわかった。

Instruction fetch efficiency on ALUS architecture

Kenji Kise,[†] Tomohiro Nakamura,[†] Hidenori Tsuji,[†]
Yuichiro Ajima[†] and Hidehiko Tanaka[†]

We discuss the ratio of ALU usage on the ALUS architecture which integrates many arithmetic logic units in one chip. The ALUS architecture speculatively fetches instructions from both paths of a branch before the condition of the branch has been evaluated (eager execution). In order to avoid the explosion of utilized resources, we adopt pruning eager execution(PEE). Our evaluations showed that 1%-PEE bypassing about 30 branches reaches 7.2 speedup over the scalar machine.

1. はじめに

マイクロプロセッサの性能に対する要求は留まるところを知らない。しかしながら、パイプライン処理やスーパースカラ方式によるマイクロプロセッサの性能向上には限界が見え始めており[2]、VLIW方式[3]、1チップに複数のプロセッサを集積するマルチプロセッサ・オン・チップ方式との融合といった新しい実行方式[6),9),10),14)]が盛んに研究されている。

一方ユーザの立場からは、実行コードの互換性に対する強い要望があり、新しい実行方式の提案には実行コードの互換性をできる限り維持する努力が必要となる。

このような状況を考慮すると、今求められていることは、スーパースカラ方式とのコード互換性を維持しながら数十倍の性能向上を目指す新しい実行方式を議論することである。

我々が注目する多数演算器方式とは、非常に多数の演算器をチップ上に集積し、それらの接続形態を動的に変更することでプログラムの最適な実行を目指す実行方式である。本方式では投機的に分岐成立と不成立の両方の制御流における命令をフェッチし、実行した後に必要となる結果のみを記憶領域に書き戻すことで計算を進める。本稿では、多数演算器方式のフェッチ手法として確率を用いた枝刈りを採用した場合の演算器利用率と性能向上率について検討する。

2. 並列性利用に関する研究

プログラムに内在する並列性に関する初期の研究は、スーパースカラ方式、スーパーパイプライン方式で利用できる並列性を明らかにすることが目的であった。文献[7]では、インオーダー発行を仮定し、投機的実行をおこなわない単純なスーパースカラ・プロセッサが抽出できる並列度を測定し、利用できる並列度が2程度であることを示している。

次の段階は、投機的実行を積極的に利用して並列性を抽出できる能力を調査するというものである。つまり、命令ウィンドウとリザベーション・ステーションに代表される並列性の利用を目指すハードウェアと、それを支援するソフトウェア技術により、さらなる並列性の利用が可能となることが明らかになった[1),5)]。図1に命令ウィンドウを用いて得られる、逐次実行に対する性能向上率の評価結果を示す。この評価はトレースデータを用いて測定したもので、全ての分岐結果が既知であり、演

† 東京大学大学院工学系研究科
Graduate school of Engineering, The University of Tokyo

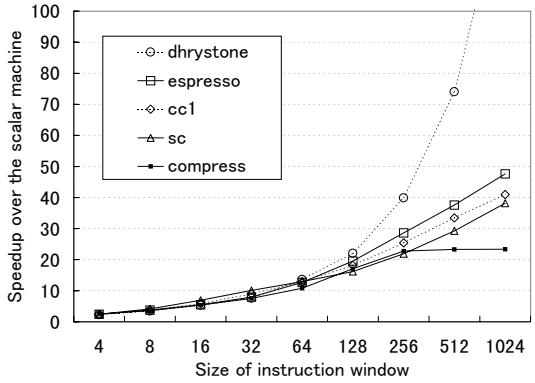

図 1 命令ウィンドウを用いた場合の最大性能向上率
Fig. 1 maximum speedup with instruction window

算器の数を制限しないという仮定を用いている。この評価より、多くの命令をフェッチし、解析することで数十倍の性能向上が得られることがわかる。

分岐予測機構の性能を見ると、動的分岐予測を用い 92〜97% 程度のヒット率が達成できることが示されている[12]。しかし 97% というヒット率でさえ決して高い値ではない。なぜなら、図 1 の命令ウィンドウを用いた評価において 64 のエントリを満たし 10 倍程度の性能向上を得るためには、10 段以上にわたる分岐命令の結果を予測する必要がある☆からである。このような背景から分岐の成立と不成立の両方の制御流を投機的にフェッチする Eager Execution に関する研究もおこなわれている[11]。

現実的な実装という視点から離れ、資源の制約を取り払い、実行前に全分岐の結果が既知とした理想的な計算機 (オラクル・マシン) における並列性を調査する研究もおこなわれてきた。理想的な計算機を仮定した場合の並列性は並列性利用の限界を示すという意味で重要である。文献[1),4)] では、理想的な計算機を仮定することで、汎用プログラムからでも数十から数百の並列性を抽出できることを示した。

多数演算器方式では、プログラムが持つ様々なレベルの並列性を利用することで、逐次実行と比較し、数十倍の性能向上率を目指す。

3. 多数演算器方式

多数演算器方式は、文献[13)] で検討されている様々なレベルの並列性を効率的に引き出すアーキテクチャを実現する試みの一つである。

スーパースカラ・プロセッサの実行コードには、レジスタ使用の競合により発生するデータ依存関係やレジスタ数の不足により必要となるロード / ストア命令といった、ハードウェアの資源競合が原因で加えられた処理が多数存在する (**図 2**-a)。また、演算結果を利用するためには、レジスタ・ファイルを介してデータを移動する必要があり、この枠組が持つ無駄が存在する☆☆。多数演算器方式では、処理装置 (Execution Unit) と呼ばれる動的に接続形態を変える多数の演算器を用いて計算の本質を表すデータフローグラフの最適な実行を目指す (図 2-b)。

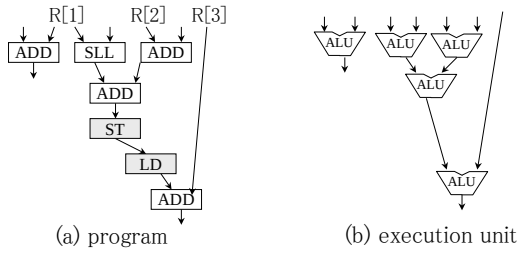

図 2 理想的な処理装置
Fig. 2 ideal execution unit

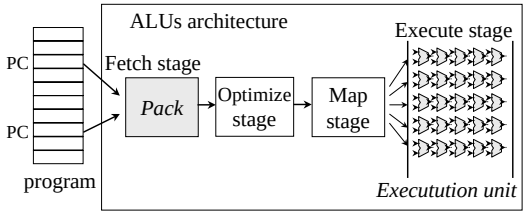

図 3 多数演算器方式の実行ステージ
Fig. 3 execution stage of ALUS architecture

多数演算器方式の実行ステージの構成を**図 3** に示す。

フェッチ・ステージでは、単一制御流のフェッチではなく、分岐成立と不成立の両方の制御流における命令を投機的にフェッチする。本方式では、複数のプログラム・カウンタを用い広い範囲からの並列性抽出を目指す。フェッチ・ステージにおけるパラメータとして、フェッチ段数と呼ぶ値を定義する。これは、跨ぐことのできる分岐命令の数に 1 を足した値であり、フェッチ段数 n とは、それぞれの制御流に存在する基本ブロックの上限が n 個であることを意味する。フェッチ・ステージでフェッチした命令の集合をパックと呼ぶことにする。

最適化ステージではレジスタ・リネーミングにより不要なデータ依存関係を解消した後に、パック内の命令が持つ真のデータ依存関係を解析し並列性を抽出する。

マッピング・ステージでは実行ユニットの構成と実行するデータフローグラフとの対応をとる。

実行ステージではパック内の命令を実行する。パックの実行に必要となるサイクル数は、正しい制御流の命令

☆ dhrystone,espresso,cc1,sc,compress それぞれのプログラムについて 1 千万命令のトレースデータを解析した結果、基本ブロック内の平均命令数は 5.1 だった。

☆☆ スーパースカラ・プロセッサではデータ・フォワーディング等の技術を用いレジスタを経由することの無駄を省いているが、これらの制御は複雑になる。

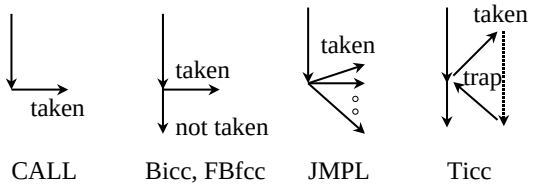

図 4 4 種類の制御転送命令における制御の流れ
Fig. 4 control flow of control transfer instructions

を実行するために必要なサイクル数として計算する。

多数演算器方式の実行イメージはパックを単位としたパイプライン処理と捉えることができる。

4. フェッチ・ステージの問題点と初期評価

ここでは、制御依存関係を作る 4 種類の分岐命令を整理した後に、演算器利用率の評価の際に生じる問題点を議論する。

それぞれの分岐命令が作る制御の流れを図 4 に示す。

CALL 命令☆は無条件分岐命令であり分岐先アドレスは静的に決めることができる。

Bicc,FBfcc 命令は共に条件分岐命令であり、分岐先アドレスは静的に決めることができる。

JMPL 命令は無条件で分岐するが、分岐先アドレスはレジスタ間接アドレッシングにより計算される。これより、分岐先アドレスの候補が多数になる場合があり、分岐先アドレスの予測を複雑にする。

Ticc 命令はソフトウェアトラップを引き起こす。本解析では Ticc 命令を跨いだ命令スケジューリングを禁止するが、一般に、プログラム中に占める Ticc 命令の実行割合は 0.01% 未満であることが多く、その影響は小さい。

4.1 Eager Execution における資源の爆発

従来おこなわれてきた単一パスに対する投機的実行ではなく、複数のパスを投機的に実行する Eager Execution を考える。Eager Execution では、分岐成立と不成立の両方のパスを投機的に実行するため、図 5 に示すようにフェッチ段数を大きくすると、フェッチする命令数が急激に増加する。フェッチ段数を大きくするためには制御流の枝刈りが必要となる。

4.2 JMPL 命令の飛び先予測

Bicc, FBfcc 命令は分岐成立と不成立の両方のパスをフェッチすることで制御依存関係を解消できる。しかし、レジスタ間接アドレッシングで分岐先を決める JMPL 命令は、分岐先が多数になる場合があり、可能性のある全てのパスをフェッチする訳にはいかない。

☆ 各命令の名前は以下による。CALL(Call), Bicc(Integer Conditional Branch), FBfcc(Floating-Point Conditional Branch), JMPL(Jump and Link), Ticc(Trap on integer condition codes)

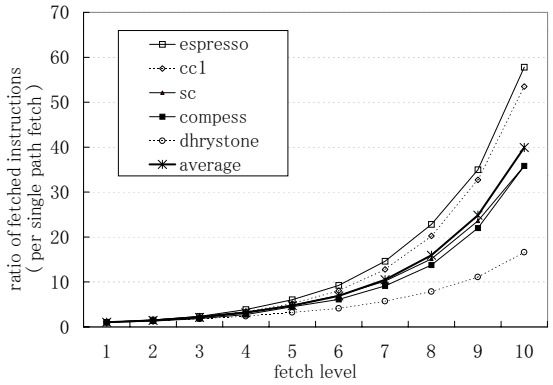

図 5 Eager Execution におけるフェッチ命令数の増加
Fig. 5 explosion of fetched instructions in EE

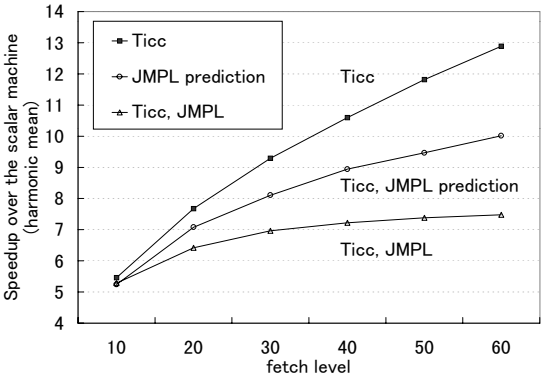

図 6 Eager Execution における JMPL 命令の影響
Fig. 6 speedup by JMPL prediction strategies

図 6 では Eager Execution における JMPL 命令の影響を調査した。横軸がフェッチ段数、縦軸は逐次実行に対する性能向上率である。一番下の曲線は JMPL 命令でフェッチを断念する場合であり、フェッチ段数を増加させても数十という値で性能向上率が飽和している。最も高い性能向上率を得ている曲線は JMPL 命令を正確に予測して、フェッチを続けることができると仮定した場合の曲線であり、両者の開きは大きい。

本章では、演算器利用率の評価の際に生じる問題点を議論し、Eager Execution においてフェッチ命令数の爆発を抑える必要があること、JMPL 命令の分岐先を予測することの重要性を確認した。

5. 評価する多数演算器方式の構成と環境

評価は SPEC92 で用いられる cc1,compress,espresso, sc と、合成ベンチマークの dhrystone を加えた 5 種類を対象に、各プログラム 1 千万命令のトレース・データを用いておこなう。使用したオペレーティング・システムは SunOS Release 4.1.4 であり、用いたアーキテクチャは SPARC architecture version 8[8] である。

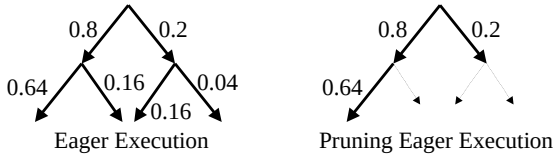

図 7 Eager Execution and Pruning Eager Execution

図 8 分岐命令の履歴を保存するテーブル
Fig. 8 tables to keep branch history information

5.1 Prunign Eager Execution

フェッチ・ステージでは確率を用いて制御流の枝刈りをおこなう Pruning Eager Execution(PEE) を用いる。PEE では全ての分岐先に到達する確率を計算し、その確率が閾値に満たない部分をフェッチしないことで、フェッチ命令数の増加を防ぐ。確率 p を用いた枝狩りを p-PEE と表すことにする。**図 7** にフェッチ段数 2 における Eager Execution と Pruning Eager Execution の様子を示す。矢印は制御流を表しており、分岐成立 80% として、それぞれの制御流に到達する確率を付けた。右の図は、確率 20% より小さい制御流の命令をフェッチしない場合で 20%-PEE に対応する。

5.2 JMPL 命令のアドレス予測

JMPL 命令の飛び先を予測しない Eager Execution では性能向上率が頭打ちとなることを考慮し、本評価では JMPL 命令の飛び先アドレスを予測し、フェッチを続けることができるようにする。予測方法としては、前回の分岐先と同じ分岐先を予測アドレスとする単純な予測方法を用いる。履歴を格納するテーブルは JMPL 命令のアドレスをタグとしたセットアソシアティブ方式でエントリ数を制限しない理想的なテーブルを用いる。**図 8** に示すように、各エントリには前回の予測アドレスを格納し、次回の予測を可能とする。この方法による分岐予測の正解率は 73% 程度であった。この方法を用いた場合の性能向上率を図 6 の中間の線に示す。より複雑な予測方式の採用は今後の課題である。

5.3 分岐確率の計算

Pruning Eager Execution を実現するためには、それぞれの制御流に到達する確率を計算する必要がある。

CALL,JMPL,Ticc 命令では必ず分岐が成立するので分岐成立の確率を 100% とすればよい。

Bicc,FBfcc 命令では、分岐成立と不成立の確率を計算する必要がある。図 8 に示す様に、分岐履歴を保存するテーブルに Bicc/FBfcc 命令のプログラム・カウンタをタグとして過去の分岐成立の数と分岐実行数を保存する。このテーブルも理容的なテーブルを仮定しており、ヒット率は初期参照を除き 100% である。確率は履歴テーブルの情報を元に計算する。テーブルの内容は分岐命令が実行された時に変更する。

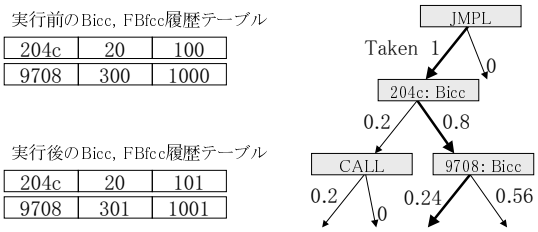

図 9 履歴テーブルと到達確率の計算例
Fig. 9 an example of the probability calculation

フェッチ段数 3 のパックにおける到達確率の計算例を**図 9** に示す。プログラム・カウンタ 204c の Bicc 命令は、履歴テーブルを引くことで過去に 100 回実行され分岐成立の回数が 20 回だったことがわかる。これにより分岐成立確率 20% と計算できる。同様に 9708 の Bicc の分岐成立確率は 30% となる。これらの情報を用いて、全ての制御流に至る到達確率を計算した結果が図 9 右となる。このパックの実行結果が分岐成立、不成立、成立と続き、太い矢印で示した制御流となった場合、実行後の履歴テーブルは図 9 左下の様に変化することになる。

本評価では実装にとらわれずに、到達確率を 32 ビットの浮動小数点数として計算している。

5.4 本評価で用いる理想化

多数演算器方式の実行に関して以下の仮定を用いる。

- フェッチ・ステージでは、単一のプログラム・カウンタからの命令フェッチのみを検討の対象とする。
- 最適化ステージにおいて、真の依存関係が無い場合には、ロード命令は先行するストア命令を越えてスケジュール可能とする。
- 実行ユニットの資源に制限を加えていない。これより、フェッチされた命令は全て実行ユニットにマップされ、処理される。
- パック間の制御依存が解消可能で、パックを単位としたパイプライン処理に乱れがないとする。

5.5 演算器の利用率と性能向上率

フェッチした全ての命令が処理装置にマップされ、有効な演算結果以外は無視されると想定し、演算器の利用率を式 1で定義する。

$$\text{演算器の利用率} = \frac{\text{有効命令の数}}{\text{フェッチした命令数}} \qquad (1)$$

演算器の利用率は、単一制御流から命令フェッチをおこなった時が 100% となり、投機的に多数の制御流から命令をフェッチするに従い低下していく。

$$\text{性能向上率} = \frac{\text{逐次実行のサイクル数}}{\text{多数演算器方式のサイクル数}} \qquad (2)$$

性能向上率を式 2で定義する。多数演算器方式のサイクル数は、全パックの実行サイクル数の和として計算する。本稿で求める性能向上率は、フェッチ以外のステージを理想化した場合の性能向上率となる。

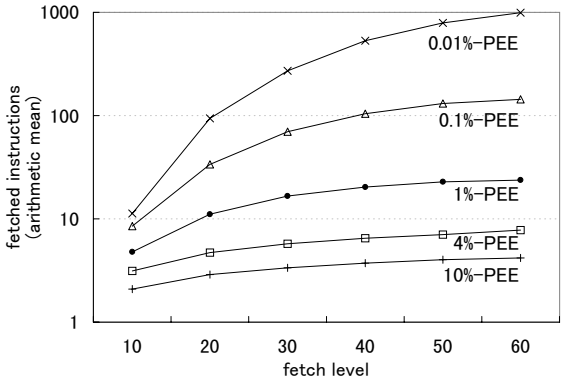

図 10 Pruning Eager Execution におけるフェッチ命令数
Fig. 10 fetched instructions by Pruning Eager Execution

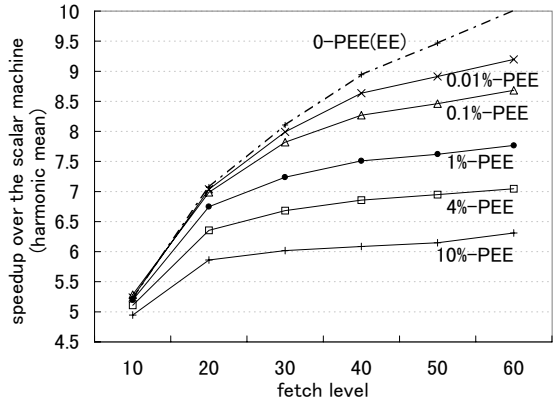

図 11 Pruning Eager Execution における性能向上率
Fig. 11 speedup by Pruning Eager Execution

6. 評価結果と考察

枝刈りのための閾値を 10%,4%,1%,0.1%,0.01% に設定した PEE (10%-PEE,4%-PEE,1%-PEE,0.1%-PEE,0.01%-PEE) を評価する。

図 10 にフェッチした命令数を示す。ここでは、逐次実行における命令数に対して何倍の命令をフェッチしたかという指標で示している。枝刈りをおこなう閾値を低くするにつれてフェッチする命令数が増加するが、Eager Execution と比較し、うまく枝刈りがおこなわれていることがわかる。

図 11 には、逐次実行と比較した場合の、PEE モデルにおける性能向上率を示した。一点鎖線は Eegar Execution の性能向上を示しているが (図 6 における中間の線と同じ)、JMPL 命令のアドレス予測のヒット率が 73% と低いために性能向上の上限が 10 程度に制限されてしまっている。このために、閾値を 1% から 0.1% さらに 0.01% と低くしてもそれほど性能向上に変化は見られない。フェッチ段数 60 のとき、閾値を 10% から 1% にすることで得られる性能向上率の差分が 1.5、1% から 0.1% にすることで得られる性能向上率の差分が 0.92、0.1% から 00.1% にすることで得られる性能向上の差分が 0.51 だった。

表 1 演算器の利用率 (%)
Table 1 ratio of ALU usage

フェッチ段数	10	20	30	40	50	60
10%-PE	46.9	33.1	28.6	26.6	25.5	25.3
4%-PEE	32.2	20.8	16.5	14.6	13.7	13.4
1%-PEE	23.2	10.0	6.30	5.10	4.58	4.42
0.1%-PEE	17.6	4.20	1.80	1.12	.854	.811
0.01%-PEE	14.8	2.57	.614	.240	.146	.117

表 2 パック内の平均命令数 (x1000)
Table 2 instructions in a pack

フェッチ段数	10	20	30	40	50	60
10%-PEE	.092	.178	.223	.250	.270	.279
4%-PEE	.146	.378	.486	.575	.636	.655
1%-PEE	.222	.845	1.52	1.99	2.27	2.38
0.1%-PEE	.396	2.73	7.37	12.5	16.7	18.7
0.01%-PEE	.520	7.72	29.3	67.3	110	147

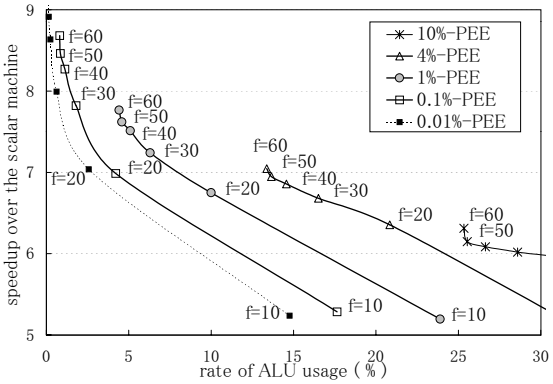

図 12 演算器利用率と性能向上率の関係
Fig. 12 relation between ALU usage ratio and speedup

演算器の利用率を**表 1** に示す。演算器の利用率は、フェッチした命令数 (図 10) の逆数として計算できる。フェッチした命令数を実行したパックの数で割ることにより得られるパック内の平均命令数を表 2に示す。

性能向上率と違い演算器利用率の変化は大きい。フェッチ段数の増加と、閾値の低下により、演算器の利用率は大きく低下する。閾値 1%, フェッチ段数 20 段で利用率が 10% となり、閾値 0.1%, フェッチ段数 50 段で利用率が 1% を下回る。

図 12 に、演算器利用率と性能向上率の関係を示す。同じ閾値の PEE は曲線で結んである。また、フェッチ段数を f として各標本点に付けた。図 12 では右上にいくほど、演算器利用率が高く、性能向上率が高いことになり理想的な構成を示していることになる。

今回の評価では 1%-PEE でフェッチ段数を 20 〜 30

とした設定が性能向上率と演算器利用率のバランスの良い点といえる。1%-PEE のフェッチ段数 30 のとき (演算器の利用率 6.3%, 性能向上率 7.23) を基準に考える。10%-PEE,4%-PEE では同様の性能向上に達することはできない。0.1%-PEE ではフェッチ段数を 30 にすることで性能向上率は 0.58 上昇するが、演算器の利用率は訳 1/3 に低下する。0.01%-PEE においてもフェッチ段数を 30 にすることで性能向上率は 0.75 上昇するが、演算器の利用率は訳 1/10 に低下する。これらの設定は、得られる性能向上率と比較し演算器利用率の低下が著しい。

7. ま と め

多数の演算器の接続形態を動的に変更し、プログラムの最適な実行を目指す多数演算器方式の性能向上率と演算器利用率を検討した。フェッチ・ステージでは、枝刈りをおこなう Eager Execution(Pruning Eager Execution) を採用した。枝刈りのための閾値を 10%,4%,1%,0.1%,0.01% に設定した PEE を評価した結果 (図 10) より、Eager Execution と比較し、うまく枝刈りがおこなわれていることを確認した。

演算器利用率 (表 1) と性能向上率 (図 11) の関係を示す図 12 のバランスより、1%-PEE でフェッチ段数 30 の時、演算器の利用率 6.30% で逐次実行に対して 7.2 倍の性能向上が得られることがわかった。このとき、平均して、約 96 の有効命令を処理するために 1520 命令のフェッチが必要となることを示した。

オラクル・マシンによる解析から汎用プログラムにおいても数十から数百倍という性能向上の可能性があることが示されている。しかし、これらの並列性を利用するためには、少なくとも複数のプログラム・カウンタを用いプログラムの広範囲から命令をフェッチすることが必要となる。数十という性能向上率を達成するために、複数命令流の利用は今後の課題である。

謝辞 本研究の一部は文部省科学研究費 (一般研究 (B) 課題番号 07458052「大規模データパスプロセッサの研究」) による。

参 考 文 献

1) Butler, M., Yeh, T.-Y. and Patt, Y.: Single Instruction Stream Parallelism Is Greater than Two, *In Proceedings of the 18th Annual International Symposium on Computer Architecture*, pp. 276–286 (1991).
2) D.Smith, M., Johnson, M. and A.Horowitz, M.: Limits on Multiple Instruction Issue, *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 290–302 (1989).
3) Hara, T., Ando, H., Nakanishi, C. and Nakaya, M.: Performance Comparison of ILP Machines with Cycle Time Evaluation, *In Proceedings of the 23th Annual International Symposium on Computer Architecture*, pp. 213–224 (1996).
4) Lam, M.S. and Wilson, R.P.: Limits of Control Flow on Parallelism, *In Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 46–57 (1992).
5) Melvin, S.: Exploiting Fine-Grainded Parallelism Through a Combination of Hardware and Software Techniques, *In Proceedings of the 18th Annual International Symposium on Computer Architecture*, pp. 289–296 (1991).
6) Olukotun, K., A.Nayfeh, B., Hammond, L., Wilson, K. and Chang, K.: The Case for a Single-Chip Multiprocessor, *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 2–11 (1996).
7) P.Jouppi, N. and david W. Wall: Availabl Instruction-Leval Parallelism for Super Scalar and Superpipelined Machines, *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 272–282 (1989).
8) SPARC International, I.: *The SPARC Architecture Manula Version 8*, Prentice-Hall (1992).
9) S.Sohi, G., E.Breach, S. and T.N.Vijaykumar: Multiscalar Processors, *In Proceedings of the 22th Annual International Symposium on Computer Architecture*, pp. 414–425 (1995).
10) Tullsen, D. M., Egtgers, S. J. and M.Levy, H.: Simultaneous Multithreading : Maximizing On-Chip Parallelism, *In Proceedings of the 22th Annual International Symposium on Computer Architecture*, pp. 392–403 (1995).
11) Uht, A. K. and Sindagi, V.: Disjoint Eager Execution: An Optimal form of Speculative Execution, *In Proceedings of MICRO-28*, pp. 313–325 (1995).
12) Yeh, T.-Y. and Patt, Y. N.: Alternative Implementations of Two-Level Adaptive Branch Prediction, *In Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 124–134 (1992).
13) 田中英彦: ここいらで、計算機アーキテクチャを再考しよう, 情報処理学会研究報告 94-ARC-108, pp. 33–40 (1994).
14) 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, 情報処理学会研究報告 97-ARC-124, pp. 13–18 (1997).