2007－ARC－174 (1)
2007／8／1

逆 Dualflow アーキテクチャ

一林宏憲[†] 入江英嗣[††]
五島正裕[†] 坂井修一[†]

Out-of-order スーパースカラ・プロセッサにおいて，レジスタ・リネーミングは命令間の偽の依存を解消するために行われる．しかし，レジスタ・リネーミングに用いる RAM である RMT(Register Map Table) は，ポート数が非常に多く遅延が大きい．また，アクセス頻度が高く消費電力も大きい．このため，レジスタ・リネーミングは高価な処理である．本研究では，レジスタ・リネーミングを省略する手法として逆 dualflow アーキテクチャを提案する．逆 dualflow アーキテクチャでは，命令のオペランドをオペランドのプロデューサへの変位に動的に変換してトレース・キャッシュに保存・再利用することにより，レジスタ・リネーミングを省略する．その代わり，トレース・キャッシュミス率が増加し IPC が低下してしまうが，トレース・キャッシュの構成を工夫することにより現時点で 2.6%の IPC 低下にとどめている．

Anti-Dualflow Architecture

HIRONORI ICHIBAYASHI,[†] HIDETSUGU IRIE,[††] MASAHIRO GOSHIMA[†]
and SHUICHI SAKAI [†]

In out-of-order superscalar processor, register renaming is employed in order to eliminate false dependence. RMT (Register Map Table), the RAM used in register renaming, is, however, heavily multi-ported and thus suffers from high latency. Additionally it is accessed so frequently that it consumes much energy. So, register renaming is very costly. In this paper, we propose a method to eliminate register renaming — anti-dualflow architecture. In anti-dualflow architecture, each operand of an instruction is dynamically converted to the displacement to the producer of the operand, and converted instructions are stored in trace cache and reused. The cost is increase in trace cache miss rate, but some improvement on trace cache structure keeps IPC decrease as low as 2.6%.

1. はじめに

Out-of-order スーパースカラ・プロセッサでは，命令間の偽の依存を解消するために，**レジスタ・リネーミング**が行われている．しかし，レジスタ・リネーミングは高価な処理である．

レジスタ・リネーミングでは，論理レジスタと物理レジスタとの「現在の」マッピングを保持する **RMT (Register Map Table)** を読み出す．RMT は 1 命令ごとに最大 4 回のアクセスが必要であり，アクセス頻度が非常に高い．RMT のサイクル当たりの平均アクセス回数は，同じくアクセス頻度の高い要素であるレジスタ・ファイルと比べても 2.4 倍にもなる．このため，RMT の消費電力は大きくなってしまう．

また，RMT は通常 RAM で実装されるが，典型的な 4-way スーパースカラ・プロセッサを仮定すると，RMT のポート数は 16 にもなる．このような多ポートの RAM の遅延は配線遅延に支配されており，LSI の微細化，動作周波数の向上とともに，レジスタ・リネーミングに必要なサイクル数が増加する傾向にある．やや極端な例であるが，Pentium 4 プロセッサでは，レジスタ・リネーミングに 3 サイクルが割り当てられている[1)]．

Dualflow アーキテクチャ

レジスタ・リネーミングを行わない命令セット・アーキテクチャとして，五島らの **dualflow アーキテクチャ**[2)~4)] がある．Dualflow アーキテクチャでは，通常の制御駆動型の命令セット・アーキテクチャにあるようなレジスタを定義しない．代わりに，データを生産する命令—**プロデューサ**と，データを消費する命令—**コンシューマ**を指定することで，明示的にデータを受け渡す．

Dualflow アーキテクチャは，元々は命令スケジューリング・ロジックを簡略化，高速化するために提案さ

† 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† 科学技術振興機構
Japan Science and Technology Agency

れた．しかし，プロデューサがコンシューマを明示的に指定するので，レジスタ・リネーミングが不要になるというメリットもある．Dualflow アーキテクチャの命令列は，ある意味,「レジスタ・リネーミング済み」ということができる．

しかし，Dualflow アーキテクチャには，命令の配置に関して強い制約がある．Dualflow アーキテクチャでは,「n 命令後」というふうにコンシューマを指定する．そのため，たとえば，分岐命令を超えてデータを受け渡すには，taken 側と untaken 側でコンシューマの位置を揃えなければならない．適切な位置に有用な命令を配置することができない場合には，転送命令や nop 命令などの本来無用な命令を挿入する必要がある．これらの無用命令のため，通常の制御駆動型の命令セット・アーキテクチャに比べて，命令数が増加してしまっていた．

そこで本研究では:

- 命令セット・アーキテクチャは通常の制御駆動型のものとして，dualflow アーキテクチャを内部表現—マイクロアーキテクチャとして利用する．制御駆動型命令セット・アーキテクチャを動的に dualflow アーキテクチャに変換する．
- 変換後の命令は，**トレース・キャッシュ**に格納する．

また，データの授受を指定する方向を逆にする，すなわち，後続の命令が，どの命令の実行結果をソース・オペランドとして使用するかを指定する．

トレース・キャッシュからは，dualflow 形式の命令がフェッチされる．前述したように，dualflow 形式の命令は「レジスタ・リネーミング済み」であるので，負荷の高い処理であるレジスタ・リネーミングを省略することができる．

以下，2 章で，レジスタ・リネーミングの負荷について述べる．次いで，3 章において，提案手法について説明する．4 章で，提案手法の評価を行う．最後に，5 章でまとめと今後の課題を述べる．

2. レジスタ・リネーミング

レジスタ・リネーミングでは，偽の依存を解消するため，1 つの命令の実行結果に対し，専用の物理レジスタを 1 つ割り当てる．論理レジスタ番号から物理レジスタ番号への変換は，論理レジスタ番号と物理レジスタ番号との現在のマッピングを保持する **RMT (Register Map Table)** を読み出すことで行う．RMT は，通常 RAM で実装される．この RAM を論理レジスタ番号でアドレッシングして読み出すことで論理レジスタ番号と物理レジスタ番号の対応を得る．

1 つの命令をリネームするには，デスティネーション論理レジスタに割り当てられていた物理レジスタを解放するための読み出し，デスティネーション論理レジスタに新たに割り当てられた物理レジスタ番号の書き込み，2 つのソース論理レジスタに割り当てられている物理レジスタ番号の読み出しと，合計 4 回のアクセスが RMT に対して可能である必要がある．典型的な 4-way スーパースカラ・プロセッサを仮定すると，RMT のポートは 16 個必要である．このような多ポートの RAM の遅延は配線遅延に支配されるので，プロセッサの動作周波数を高めるにつれて，レジスタ・リネーミングに必要なサイクル数は増加してしまう．また，フロントエンドの幅を増加させることは，RMT のポート数がさらに増加することにつながるため，非常に困難である．

さらに，RMT のアクセス頻度は，同じく概念的にはオペランド 1 つごとに 1 回アクセスを行うレジスタ・ファイルと比べても非常に高い．これは，物理レジスタ解放のための読み出しが必要であること，ソース・オペランドの多くはフォワーディングにより供給されること，投機ミスのためリネームは行われたが実行はされない命令が存在することが原因である．我々の行ったシミュレーションによると，ソース・オペランドの約 60%はフォワーディングにより供給され，RMT のサイクル当たり平均アクセス回数はレジスタ・ファイルの約 2.4 倍であった．

このように，RMT は多ポート，高遅延，高アクセス頻度のため，レジスタ・リネーミングを省略することによりパイプライン段数を削減でき，またフロントエンドの幅を増加させたり，消費電力を削減したりできると考えられる．

3. 逆 Dualflow アーキテクチャ

2 章で，論理レジスタ番号から参照すべき物理レジスタを特定するために引く表である RMT の負荷が高いことを述べた．逆 dualflow アーキテクチャは，レジスタ・リネーミングそのものを省略することを目的とする．

逆 dualflow アーキテクチャの基本的な考え方は次の通りである:

- 命令の実行結果を，サイクリックなバッファ（以下，**物理レジスタ・ファイル**と呼ぶ）に，命令の出現した順番と同じ順番で格納する
- 物理レジスタ・ファイルとは別に，論理レジスタの値を保持する**論理レジスタ・ファイル**を用意し，命令の実行結果を in-order に格納する
- 初回の実行時に命令を変換し，ソース・オペラン

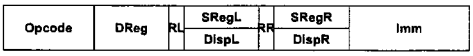

図 1 dualflow 形式

ドを物理レジスタ・ファイル上の変位で指定するようにする．ただし，ソース・オペランドとして用いる値を生産した命令が一定以上離れていて，リタイアしていることが，すなわち論理レジスタ・ファイルに実行結果が格納されていることが保証できる場合には論理レジスタ番号で指定する

- 変換した命令をトレース・キャッシュにキャッシュ・再利用する

このようにすることで，レジスタ・リネーミングを省略してもオペランドへのアクセスを可能にする．

以下，逆 dualflow アーキテクチャについて詳細に述べる．

3.1 命令の内部表現

逆 dualflow アーキテクチャでは，命令のソース・オペランドを，論理レジスタ番号で指定する通常の形式から「n 命令前の命令の実行結果」として物理レジスタ・ファイル上の変位で指定する形式に内部的に変換する．ただし，WS**(命令ウィンドウサイズ)** 命令以上前であるため，または後に 3.5 節で述べる経路情報の不足によってリタイアを待つため，リタイア済みであることが保証できる命令の実行結果を使用する場合には，論理レジスタ番号で指定する．

以下，この形式を dualflow 形式と呼ぶ．**図 1** に，dualflow 形式の例を示す．各フィールドの意味は次の通りである:

Opcode *Opcode* は，命令のオペ・コードを示す

DReg *DReg* は，デスティネーション論理レジスタ番号を示す

RL(R) DispL(R)/SRegL(R) *DispL*(R) / *SRegL*(R) フィールドは，*RL*(R) フィールドが 0 のとき *DispL*(R) 命令前の命令の実行結果が，*RL*(R) フィールドが 1 のとき *SRegL*(R) 番の論理レジスタがこの命令の左 (右) ソース・オペランドとして用いられることを示す．

Imm *Imm* は，即値を示す

3.2 Dualflow 形式の命令の実行

Dualflow 形式の命令の実行方法を，**図 2** を用いて説明する．$[-n]$ は,「n 命令前の命令の実行結果」を表す．

まず，各命令に対し，物理レジスタ・ファイルのエントリを命令の出現した順番と同じ順番で割り当てる．順番に割り当てることにより，読み出すべき物理レジスタは，命令に割り当てられた物理レジスタのインデックスに変位を加算するだけで決定することができる．物理レジスタ・ファイルのエントリには，対応する命令のデスティネーション論理レジスタ番号を付随させておく．命令の実行結果は，実行の終わったものから out-of-order に物理レジスタ・ファイルに格納する．その後，最も古い命令に対応する物理レジスタのエントリから順番に，すなわち in-order に実行結果を論理レジスタ・ファイルにコピーする．

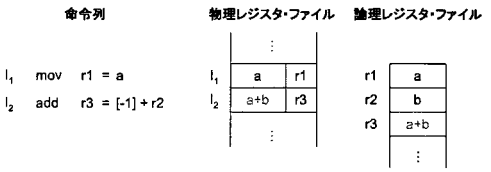

図 2 dualflow 形式の命令の実行

物理レジスタ・ファイルは，命令ウィンドウ中の最も古い命令が，変位でオペランドを指定する範囲である $(WS-1)$ 命令前の命令の実行結果を利用できるようにするため，$(2 \times WS)$ エントリとする．

3.3 Dualflow 形式への変換

Dualflow 形式への変換は，レジスタ・リネーミングと同様に行うことができる．まず，命令にフェッチされた順の通し番号を振る．論理レジスタ番号とそのレジスタに最後に書き込んだ命令の番号の対応表を用意し，命令がフェッチされるたびに更新する．対応表からソース・オペランド・レジスタに対応する命令の番号を読み出し，変換対象の命令の番号との差を取れば，何命令前の命令の実行結果をソース・オペランドとして用いればよいかが分かる．

ここで，同時に多数の命令を変換する場合，結局，この対応表の負荷が RMT と同様に高くなってしまう．しかし，変換は初回の実行時のみ行われ，以後はキャッシュした dualflow 形式の命令を再利用できるので，同時に変換できる命令が少なくても性能への影響は少ないと考えられる．今回は，1 サイクルに 1 命令を変換できるとした．

3.4 変換の経路依存性

逆 dualflow アーキテクチャでは，命令のソース・オペランドを「n 命令前」と変位で表現する形式に変換する．しかし，ソース・オペランド・レジスタに最後に書き込んだ命令の位置は，変換中の命令までに実行した命令 (以下，**経路**と呼ぶ) によって変化する．**図 3** に例を示す．分岐命令 bne の成否により，r1 に最後に書き込んだ命令が変わり，変換結果も異なってしまう．このため，変換結果は経路によって区別してキャッシュする必要がある．ただし，前述したように WS 命令以上前の命令の実行結果は論理レジスタ番号で参照で

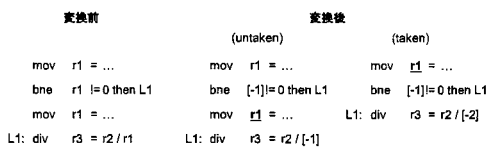

図 3 変換の経路依存性

きるので，経路の長さは $(WS-1)$ 命令でよい．

また，経路によって変換結果が異なるのは，キャッシュのエントリ内の分岐でも同様である．よって，実行順に命令をキャッシュするトレース・キャッシュを用いるのがよい．

3.5 経路の表現

3.4 節で述べたように，変換結果のトレースは経路によって区別してキャッシュする必要がある．経路は直前に実行した $(WS-1)$ 個の命令であるから，単純には直前の $(WS-1)$ 命令のアドレスで表現することができる．しかし，この表現ではサイズが膨大になってしまうので，コンパクトな表現が必要である．

今，ある命令の次の命令について次のことが言える:

- 分岐でない命令の次の命令は一意に定まる
- 条件分岐の次の命令は，分岐の成否によって一意に定まる
- 間接分岐の次の命令は，間接分岐のターゲットによって一意に定まる

よって，経路の先頭のアドレスと，経路内の条件分岐の成否，間接分岐のターゲットにより，経路を表現することができる．また，これに経路長を合わせることで，この経路の直後の命令，つまりトレースの先頭アドレスも一意に定まる．

条件分岐の成否は 1 ビットで表せるから，経路内の条件分岐の成否を保持するには $(WS-1)$ ビットあればよい．経路情報の生成を簡単にするため，条件分岐でない命令に当たる部分は 0 として，n ビット目に n 命令前の分岐の成否を保持する．これにより，m 命令フェッチしたときには，このビット列を m ビットだけシフトし，フェッチした m 命令分の分岐成否を付け加えればよくなる．

また，経路に含めることのできる間接分岐の数を N 個に制限する．制限することによって $(WS-1)$ 命令前までの間接分岐ターゲットを全て保持することができなくなる場合，経路情報として，$(N+1)$ 個前の間接分岐のターゲット（H とする）を経路の先頭とし，N 個前までの間接分岐ターゲットのみを保持することにする．このような経路情報の不足が起こった場合，変換時には，H より前の命令の実行結果は論理レジスタ番号で参照するようにする．トレース・キャッシュのフェッチ時には，H より前の命令がリタイアし，実行結果が論理レジスタ・ファイルに格納されるまで，フェッチをストールさせる．

経路情報として用いることのできる間接分岐の数を減らすと，フェッチのストールする頻度が高くなるが，経路情報の無駄は少なくなる．間接分岐の数を増やすと，フェッチのストールする頻度は低くなるものの，経路情報の無駄は多くなってしまう．

3.6 逆 dualflow アーキテクチャの効果

逆 dualflow アーキテクチャにより，レジスタ・リネーミングを省略することができる．レジスタ・リネーミングを省略することにより:

- レジスタ・リネーミングに必要なハードウェア・電力 (特に RMT) が削減できる
- レジスタ・リネーミング・ステージのぶん，分岐予測ミス・ペナルティが減少する
- フェッチした後フロントエンドで行う処理に命令間の依存がなくなるため，フロントエンドの幅を増やせる可能性がある

ただし，トレースを経路で区別することによりトレースの種類が増加するため，トレース・キャッシュ・ミス率が増加してしまう．

3.7 世代別トレース・キャッシュ

逆 dualflow アーキテクチャでは，関数 call・return 時，ループ開始・終了時，強偏向な分岐が逆に分岐したときなど，出現頻度の低い経路を通るときに使用頻度の低いトレースが多数生成されてしまう．使用頻度の低いトレースによって使用頻度の高いトレースがリプレースされてしまうことを防ぐことで，トレース・キャッシュ・ミス率を低減できると考えられる．

そこで，次のような方法を考える．まず，新たに生成されたトレース用のトレース・キャッシュ(**子トレース・キャッシュ** と呼ぶ) と，一定回数以上使用されたトレース用のトレース・キャッシュ(**親トレース・キャッシュ** と呼ぶ) を用意する．新たに生成されたトレースは，まず子トレース・キャッシュに格納する．子トレース・キャッシュ中のトレースに，そのトレースが使用された回数を保持する飽和カウンタを設け，このカウンタをトレースがフェッチされたときにインクリメントする．このカウンタが一定の閾値 (**昇格閾値** と呼ぶ) 以上になったとき，当該トレースを親トレース・キャッシュに移動させ，子トレース・キャッシュでは無効にする．トレースのフェッチ時には，両方のトレース・キャッシュを平行にアクセスする．

このようなトレース・キャッシュの構成を**世代別トレース・キャッシュ**と呼ぶことにする．世代別トレース・キャッシュを用いることで，通常の構成においても

表 1　ベース・モデルの主要なパラメータ

フェッチ幅	4
発行幅	4
命令ウィンドウ	32 エントリ (集中)
L1 命令キャッシュ	32KB, 4 ウェイ, 3 サイクル, 16B ライン
L1 データ・キャッシュ	16KB, 4 ウェイ, 3 サイクル, 32B ライン
L2 データ・キャッシュ	1MB, 4 ウェイ, 15 サイクル, 32B ライン
メイン・メモリ	200 サイクル
演算器	IntALU × 4, IntMUL × 2, IntDIV × 2, FPADD/MUL × 2, FPDIV × 1
分岐予測	BTB: 8K エントリ, gshare: 32K エントリ PHT, 10 ビットグローバル分岐履歴 RAS: 16 エントリ
トレース・キャッシュ	2K エントリ, 4 ウェイ, エントリ・サイズ: 16B (4 命令), トレース中の分岐数: 1

キャッシュ・ミス率を低減できる．しかし，逆 dualflow アーキテクチャでは，出現頻度の低い経路を通るとき，続いて生成される数個のトレースの経路情報として出現頻度の低い経路が使用されるため，使用頻度の低いトレースがより多く生成されてしまう．このため，逆 dualflow アーキテクチャにおいてより効果的であると考えられる．

4. 評　　価

4.1 評価環境

シミュレーションを行い，逆 dualflow アーキテクチャの評価を行った．シミュレーションには本研究室で開発したシミュレータ「鬼斬」を用いた．ベンチマークには，SPEC CINT2000 のプログラムから，gzip, vpr, gcc, crafty, parser, eon, gap, vortex, bzip2, twolf の 10 本を用いた．入力セットには train を用い，最初の 1G 命令をスキップし直後の 100M 命令を実行した．

表 1 にベース・モデルの主要なパラメータを示す．

4.2 経路情報に用いる間接分岐数

3.5 節で述べたように，経路情報に用いることのできる間接分岐の数を多くすれば，経路情報の不足によりフェッチがストールすることは少なくなるが経路情報のサイズは大きくなる．逆に間接分岐の数が少なすぎると，経路情報の不足によりフェッチが頻繁にストールすることになり，IPC が低下してしまう．

そこで，経路情報に用いる間接分岐数の IPC への影響を測定した．**図 4** に測定結果を示す．グラフの横軸がベンチマーク（右端は平均）であり，縦軸はベース・モデルを 1 として正規化した IPC である．ベンチマークごとの 5 本の棒グラフは，左からそれぞれ，ベース・モデル，間接分岐数 0，1，2，3 個の場合の逆 dualflow アーキテクチャに対応する．

間接分岐数が 0 個の場合に比べて 1 個の場合は平均 IPC が 10.5%向上している．これに対し，間接分岐数が 1 個の場合に比べて 2 個の場合には平均 IPC が 0.9%しか向上していない．これは，2 個目の間接分岐ターゲットが格納されないことが多い，もしくは経路に間接分岐を 1 個しか含めない場合に経路情報として表現できなくなる部分の長さが十分に小さいためと考えられる．よって，経路情報としては間接分岐を 1 個含むことができれば十分であると言える．

また，間接分岐を 1 つ含めるとき，逆 dualflow アーキテクチャは，ベース・モデルに比べて IPC が 4.9%低下してしまっている．

4.3 世代別トレース・キャッシュを用いる場合

次に，世代別トレース・キャッシュを用いた場合の IPC を測定した．**図 5**，**図 6** に，世代別トレース・キャッシュをベース・モデル，逆 dualflow アーキテクチャにそれぞれ適用したときの測定結果を示す．グラフの横軸は世代別トレース・キャッシュの構成であり，親トレース・キャッシュ，子トレース・キャッシュのエントリ数が，左はそれぞれ 2K，1K，右はそれぞれ 1K，2K である．縦軸は通常のトレース・キャッシュを用いるベース・モデルを 1 として正規化した IPC である．構成ごとの 10 本の棒グラフは，左からそれぞれ，昇格閾値が 1，2，3，4, 6, 8, 15, 63, 255, 1023 の場合である．

世代別トレース・キャッシュをベース・モデルに適用した場合と逆 dualflow アーキテクチャに適用した場合のいずれも親トレース・キャッシュ2K エントリ，子トレース・キャッシュ1K エントリの場合に最大の IPC の向上が見られた．ベース・モデルに適用した場合，昇格閾値が 3 のとき IPC の向上が最大であり，1.2%向上している．逆 dualflow アーキテクチャに適用した場合，昇格閾値が 2 のとき IPC は 3.6%向上して，世代別トレース・キャッシュを用いないベース・モデルの 98.6%になっている．つまり，世代別トレース・キャッシュを用いた場合には，逆 dualflow アーキテクチャはベース・モデルに対して 2.6% の IPC 低下に抑えられている．

5. おわりに

本研究では，レジスタ・リネーミングを省略する手法として，逆 dualflow アーキテクチャを提案し，性能への影響を評価した．逆 dualflow アーキテクチャでは，命令のオペランドをプロデューサへの変位に動的に変換してトレース・キャッシュにキャッシュ・再利用することにより，レジスタ・リネーミングを省略することができる．レジスタ・リネーミングに用いる RMT へのアクセス頻度が高いため，レジスタ・リネーミン

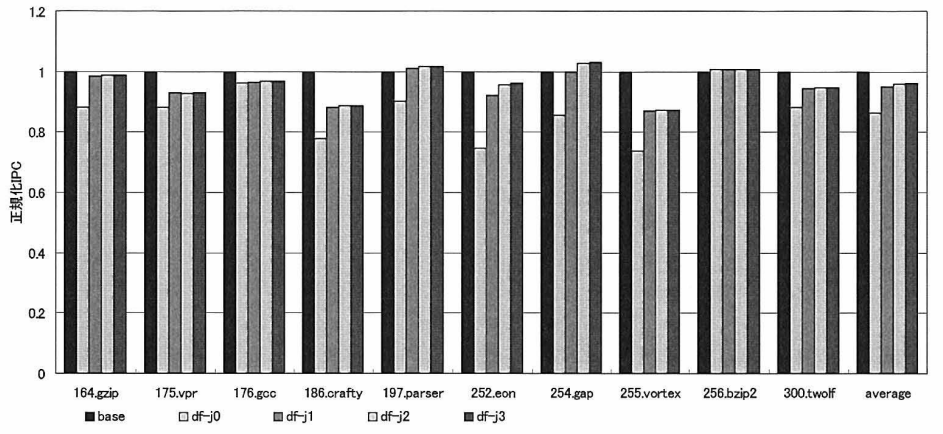

図 4　経路中の間接分岐数と IPC

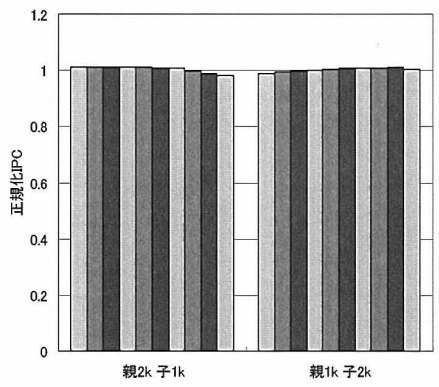

図 5　世代別トレース・キャッシュの構成と IPC (ベース・モデル)

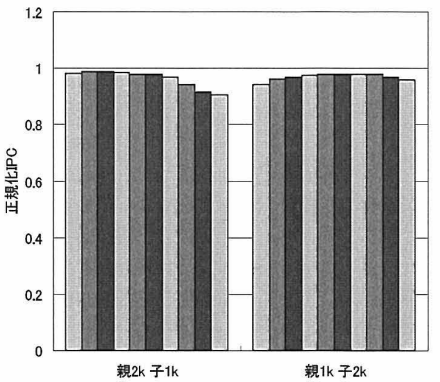

図 6　世代別トレース・キャッシュの構成と IPC (逆 dualflow アーキテクチャ)

グを省略することにより消費電力の低下が見込める．その代わり，変換した命令を経路によって区別して格納する必要が生じるため，トレース・キャッシュ・ミス率が増加してしまい，IPC が低下してしまう．ただし，トレース・キャッシュの構成を工夫することにより，2.6%の IPC 低下にとどめられている．

今後は，トレース・キャッシュのミス率をより低くし，IPC への影響をさらに低くしていきたい．また，逆 dualflow アーキテクチャではレジスタ・リネーミングが不要であるため，フロントエンドで行う処理に命令間での依存がない．これを利用して，トレースの長さとフロントエンドの幅を 2 倍にし，クロックを半分にすることにより，さらなる低消費電力が達成できないか考えている．

謝辞　本研究の一部は，21 世紀 COE「情報技術戦略コア」，及び科学技術振興機構 CREST「ディペンダブル情報処理基盤」による．

参 考 文 献

1) Hinton, G., Sager, D., Upton, M., Boggs, D., Carmean, D., Kyker, A. and Roussel, P.: The Microarchitecture of the Pentium 4 Processor, *Intel Technology Journal Vol.5 Issue 1* (2001).
2) 五島正裕, グェンハイハー, 縣亮慶, 森眞一郎, 富田眞治: Dualflow アーキテクチャの提案, 並列処理シンポジウム JSPP 2000, pp. 197–204 (2000).
3) 五島正裕, グェンハイハー, 縣亮慶, 中島康彦, 森眞一郎, 北村俊明, 富田眞治: Dualflow アーキテクチャの命令発行機構, 情報処理学会論文誌, Vol. 42, No. 4, pp. 652–662 (2001).
4) 五島正裕: Out-of-Order ILP プロセッサにおける命令スケジューリングの高速化の研究, 博士論文, 京都大学 (2004).