社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－ARC－158 (3)
2004／5／14

フロントエンド実行

小西将人[†] 五島正裕[††] 中島康彦[††]
森眞一郎[††] 富田眞治[††]

値予測は，データ依存による先行制約を緩和する手法として盛んに研究されてきたが，現状では十分な性能向上が得られているとは言い難い．それに対して本稿では，フロントエンド実行と呼ぶ手法を提案する．スーパースカラ・プロセッサの命令パイプラインの，命令ウィンドウより上流をフロントエンド，命令ウィンドウおよびその下流をバックエンドと呼ぶ．したがって通常のスーパースカラ・プロセッサでは，命令の実行ステージはバックエンドにある．フロントエンド実行とは，バックエンドに加えてフロントエンドにも演算器を配し，実行可能な命令をフロントエンドにおいても実行することである．シミュレーションによる性能評価の結果，フロントエンド実行を実装したプロセッサは 16.3% の高速化を達成できることが分かった．

Frontend Execution

Masahito Konishi,[†] Masahiro Goshima,[††] Yasuhiko Nakashima,[††]
Shin-ichiro Mori[††] and Shinji Tomita[††]

Value prediction can relax the restriction on the order of instruction execution exceeding data-flow dependence. But it has not achieved sufficient performance. This paper describes a new scheme called frontend execution, which has a similar effect to value prediction. The instruction pipeline of a superscalar processor can be divided into two parts: the frontend is upper part than the instruction window, and the backend is the instruction window and the lower part of it. Thus conventional superscalar processor has the execution stage in its backend. Frontend execution is to execute ready instructions by function units posed in the frontend as well as the backend. Evaluation result shows a processor with frontend execution is about 16.3 times faster than a normal processor.

1. はじめに

値予測[1]は，データ依存による先行制約を緩和する手法として盛んに研究されてきた．しかし現状では，十分な性能向上が得られているとは言い難い．

それに対して本稿では，**フロントエンド実行** (frontend execution) と呼ぶ手法を提案する．スーパースカラ・プロセッサの命令パイプラインの，命令ウィンドウより上流を**フロントエンド**，命令ウィンドウおよびその下流を**バックエンド**と呼ぶ．したがって通常のスーパースカラ・プロセッサでは，命令の実行ステージはバックエンドにある．フロントエンド実行とは，バックエンドに加えてフロントエンドにも演算器を配し，実行可能な命令をフロントエンドにおいても実行することである．

フロントエンド実行は，値予測と同様の効果を持つ．その上フロントエンド実行では，値予測が持つ欠点が，以下のように解消される：

(1) 値予測には，予測ミスのペナルティがある．一方フロントエンド実行は，投機ではないので，ペナルティがない．

(2) 値予測では，予測ヒットする命令も，ヒット / ミスの確認のために実行しなければならず，実行される命令の総数が減るわけではない．
一方フロントエンドで実行された命令は，バックエンドで再び実行する必要がなく，バックエンドで実行される命令の数は削減される．

本稿では，フロントエンド実行について述べる．以下，2 章でフロントエンド実行について概説した後，3 章でフロントエンド実行の方法について詳しく述べる．4 章で評価結果を示す．

2. 値予測とフロントエンド実行

本稿では，命令レベルの値予測を代替，あるいは，補完する技術として，フロントエンド実行を提案する．スーパースカラ・プロセッサの命令パイプラインの，命令ウィンドウより上流を**フロントエンド**，命令ウィンドウおよびその下流を**バックエンド**と呼ぶ．したがって通常のスーパースカラ・プロセッサでは，命令の実行ステージ

† 現 大阪工業大学
†† 京都大学

はバックエンドにある．フロントエンド実行とは，バックエンドに加えてフロントエンドにも演算器を配し，実行可能な命令をフロントエンドにおいても実行することである．

2.1 フロントエンド実行可能性

リザベーション・ステーションを用いるスーパースカラ・プロセッサでは，レジスタ読み出しはフロントエンドにおいて行われる．このときに読み出されたソース・オペランドのすべてが利用可能であるわけではない．未実行の命令の実行結果をソース・オペランドとする場合には，レジスタが読み出されたオペランド・データは無効であり，実行結果がフォワードされてはじめて利用可能になる．

逆に言えば，レジスタ読み出しを終えた時点で実行に必要なソース・オペランドが揃った命令は，この時点で既に実行可能である．そのため，もしレジスタ読み出しステージの直下に演算器を用意してやれば，これらの命令をそこで実行することができる．

2.2 値予測とフロントエンド実行の効果

図 1 に，命令 I_p と，それに依存する命令 I_c の実行の様子を示す．最上段は，通常のプロセッサにおける実行の様子を示す．I_p と I_c は，同時にフェッチ，デコード，ディスパッチされているが，データ依存により，I_c は I_p の実行レイテンシ，1 サイクルだけ遅れて実行されることになる．

値予測によって I_p の実行結果の値を予測されたとしよう．その場合 I_c は，予測された値を用いて，命令ウィンドウにディスパッチされた直後から実行可能になる．図 1 では，I_c は，データ依存による先行制約を破って，I_p の実行レイテンシだけ早く実行することができる．I_c からは，I_p の実効的な実行レイテンシが 0 サイクルになったように見える．

フロントエンド実行にも，同様の効果がある：同様に，命令 I_p と，それに依存する命令 I_c があり，I_p がフロントエンド実行されたとしよう．その場合 I_c は，値予測の場合と同様に，フロントエンド実行された結果を用いて，命令ウィンドウにディスパッチされた直後から実行可能になる．したがってフロントエンド実行では，I_p と I_c はデータ依存による先行制約を破ってはいないが，値予測と全く同じ効果が得られることになる．すなわち I_c は，I_p の実行レイテンシだけ早く実行することができ；I_c からは，I_p の実効的な実行レイテンシが 0 サイクルになったように見える．

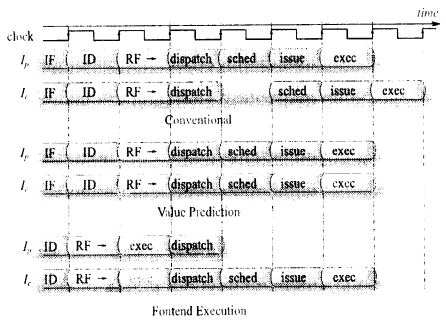

図 1 パイプライン

2.3 値予測とフロントエンド実行の違い

値予測には，以下のような問題点がある：

1. **命令の総数** 予測ヒットする命令であっても，予測の確認のために実行しなければならず，実行される命令の総数が減るわけではない．
2. **予測ミス** 予測ミス時には再実行が必要となるから，実行される命令の総数は実際には増加する．
3. **ハーウェア・コスト** 大容量の予測表を必要とするため，比較的ハードウェア・コスト が高い

フロントエンド実行では，上述した値予測の問題点が以下のように解決，緩和される：

1. **命令の総数** フロントエンドで実行された命令はバックエンドで再び実行する必要がなく，バックエンドで実行される命令数が削減される．
2. **予測ミス** 投機ではないので，予測ミスが生じない．
3. **ハードウェア・コスト** フロントエンド実行のためには，命令フェッチ，ディスパッチ幅と同じだけの演算器を必要とする．これらの演算器は，現在のスーパースカラ・プロセッサにとって，大きなハードウェア ではない．また，フロントエンド実行ステージのため，パイプライン段数は 1〜2 段増加することになるが，それによる性能低下は，分岐予測の高いヒット率によって補償することができる．

次章では，フロントエンド実行の具体的な機構について述べる．

3. フロントエンド実行機構

スーパースカラ・プロセッサの命令パイプラインの，命令ウィンドウより上流をフロントエンド，命令ウィンドウおよびその下流をバックエンドと呼ぶ．したがって通常のスーパースカラ・プロセッサでは，命令の実行ステージはバックエンドにある．フロントエンド実行とは，バックエンドに加えてフロントエンドにも演算器を配し，実行可能な命令をフロントエンドにおいても実行することである．

3.1 ベース・モデル

Out-of-order スーパースカラ・プロセッサの構成方式は，リザベーション・ステーションとリオーダ・バッファを併用する方式と，物理レジスタに対するリネーミングによる方式に大別することができる．以降では，前者を **RS+RB 方式**，後者を**物理レジスタ方式**と呼ぶことにする．フロントエンド実行は，RS+RB 方式を基にするのが都合がよい．

レジスタ読み出しを命令パイプラインのどこで行うかに注目すると，2つの方式は以下のように説明できる：

物理レジスタ方式 物理レジスタ方式は，物理レジスタの読み出しをバックエンドで行う．そのため物理レジスタ方式では，フロントエンドとバックエンドを命令ウィンドウでほぼ完全に分離 (decouple) することができ，チップ上のレイアウトの点で有利である．しかし，物理レジスタ読み出しをバックエンドで行うため，命令が発行されてから実際に実行されるまでのレイテンシ，発行レイテンシが長くなる．

RS+RB 方式 RS+RB 方式では，レジスタ読み出しはフロントエンドで行われる．レジスタ・ファイルは，フロントエンドで読み出され，バックエンドでライトバックされることになる．また，実行ステージからリザベーション・ステーションへ，実行結果のフォワーディング・パスを設ける必要があり，データ・パスが複雑になる．これらの理由のため，レイアウト上の制約が比較的厳しい．

しかし，レジスタ読み出しをフロントエンドで行うので，物理レジスタ方式に比べて，その分だけ発行レイテンシを短くすることができる．発行レイテンシは，物理レジスタ方式の半分程度になる．

発行レイテンシはキャッシュ・ヒット / ミス予測を始めとする実行レイテンシ予測のミス・ペナルティとなる．この発行レイテンシの差は，配線遅延が問題となる近い将来のプロセッサでは，2割程度の IPC の差を生じることが分かっている[2),3)]．

フロントエンド実行を行うためには，RS+RB 方式を基にするのが都合がよい．RS+RB 方式では，フロントエンドでレジスタ読み出しを行うため，その直下に演算器を配置すれば，自然にフロントエンド実行を行うことができる．

3.2 フロントエンド実行ステージ

フロントエンド実行のステージは，レジスタ読み出しと，ディスパッチの間に設ける．その分，1〜2 サイクル，分岐予測ミス・ペナルティが増加するが，その影響は軽微である．最近では，ベースとなるパイプライン段数がそもそも非常に深いため，1〜2 サイクル程度の増加は相対的に小さくなる．また，分岐予測ミス・ペナルティの増加は，分岐予測ヒット率の高さによって補償することができる．

フロントエンド実行ステージに配する演算器は，整数ユニットを基本とする．整数ユニットをフロントエンド・パイプラインの幅だけ並べればよい．この場合，通常のバックエンドの実行ユニットのようなオペランド・バイパス・ネットワークは不要であり，演算器自体のハードウェア・コストはほとんど無視できる．

3.3 カスケード

フロントエンド実行ステージを2ステージ分用意し，1段目の演算器とカスケード接続することが考えられる．

命令フェッチ幅が大きくなると，フェッチ・グループ内に互いに依存する命令が含まれる可能性が高くなる．カスケードは，このような場合に，フロントエンド実行可能な命令の数を増加させる効果がある．図1の例では，I_c は I_p に依存するため，フロントエンド実行できていないが，2段目を追加することによりフロントエンド実行可能になる．

ただし，1段目と2段目の間には，基本的には全対全結合のネットワークが必要となる．通常のバックエンドの実行ユニットのオペランド・バイパス・ネットワークのように，出口と入口を接続する必要はないので，それほど複雑にはならないものの，シンプルさというメリットは損われることになる．

3.4 0次キャッシュ

近年の1次キャッシュ・レイテンシの増加に対応するため，1KB 程度以下の0次キャッシュを用意することが検討されている[4),5)]．4章で評価するモデルには，各ロード / ストア・ユニットに 1KB の0次キャッシュを分散配置している．

この0次キャッシュを，フロントエンド実行の2段目にも用意することが考えられる．フロントエンド実行の1段目でアドレス計算を行い，2段目で0次キャッシュアクセスを行うのである．これによって，定数のロードなどがフロントエンド実行可能になると考えられる．

なお，この場合，フロントエンドでストア命令を実行する意味はほとんどないので，ロードだけを実行する．

4. 評　　価

SimpleScalar ツールセット (ver. 3.0) の sim-outorder シミュレータに対して，フロントエンド実行ユニットと値予測器を実装し，性能の比較を行った．評価には SPEC int95 の 8 つのプログラムを実行した．コンパイラは，gcc (ver. 2.7.2.3) を用いた．最適化オプションは，-O6 -funroll-loops である．

4.1 評価方法

ベース・モデルの主要なパラメタを**表1**に示す．ベース・モデルは RS+RB 方式とし，命令パイプラインの各フェーズに要するサイクル数は以下のように仮定した：フェッチ: 3，デコード: 1，レジスタ読み出し: 1，ディスパッチ: 1，スケジューリング: 1，発行: 1 サイクル．

表1 ベース・モデルの主要なパラメータ

ウェイ数	4
機能ユニット	iALU 4, LD/ST 2, iMUL/DIV 2, fADD 4, fMUL 2, DIV 2
0 次キャッシュ	1K× 2, レイテンシ 1
1 次キャッシュ	64KB, 2 way, レイテンシ 3
2 次キャッシュ	1MB, 8 way, レイテンシ 6

フロントエンド実行

フロントエンド実行については以下4つを評価をした．

F1 1ステージの整数ユニットを持つ．

F2 2ステージ分の整数ユニットをカスケード接続する．

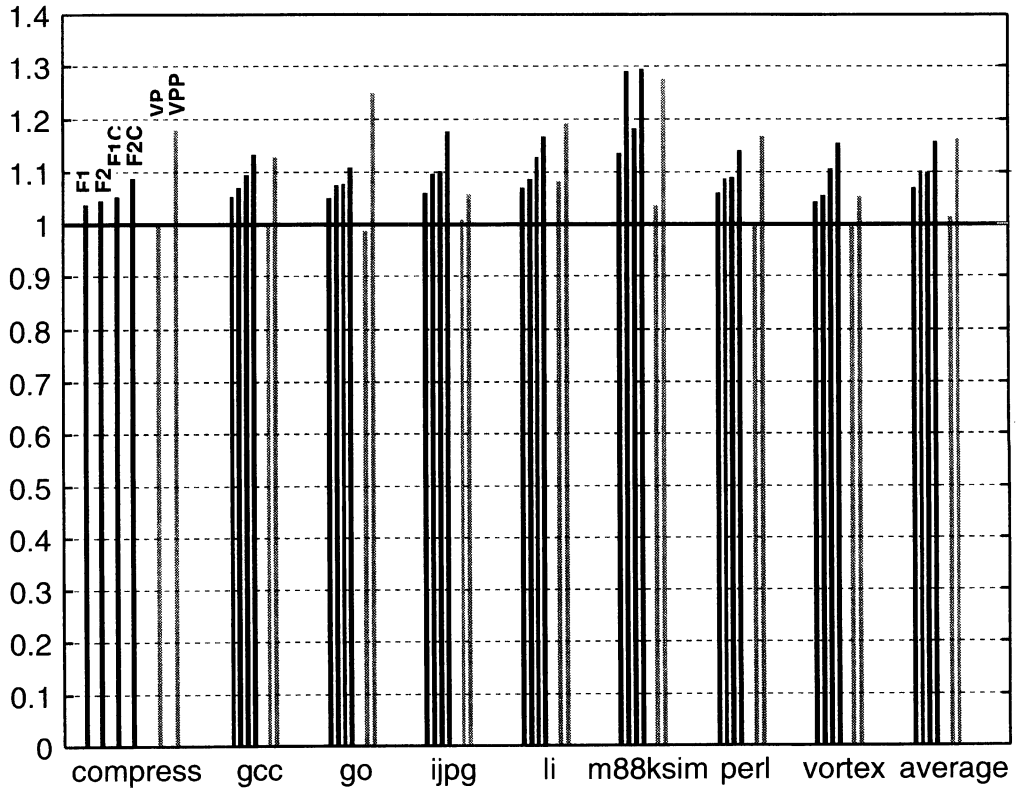

図 2　性能向上率

F1C　**F1** の 2ステージ目にキャシュを持つ．

F2C　**F2** の 2ステージ目にキャシュを持つ．

F1C，**F2C** については，整数ユニットによる実行に加えて，3.4 で述べたようにフロントエンド実行ステージの 1 段目でアドレス計算を行い，2 段目で 0 次キャッシュにアクセスする．今回の評価では，0 次キャッシュでミスした場合，すなわち 1 サイクルでデータが得られなかった場合にはフロントエンド実行されなかったものとし，通常通りバックエンドで実行する．

F1 については 1サイクル，その他のモデルについては 2サイクルだけフロントエンドが増大することになる．

値　予　測

本稿では，前回の実行結果を予測値とする最終値予測器 (Last Value Predictor: **LVP**)[1] を用いた．値履歴テーブル (Value History Table: VHT) は 4Kエントリとし，各エントリは予測値，タグのほか予測値の信頼度を示す 2b 飽和型カウンタを持つ．カウンタは実行結果が前回の値と同じであった場合 1 増加させ，異なる場合には 0 にリセットする．

予測器の参照はディスパッチの直前で，予測器の更新と予測値の検証は実行・ステージの直後で行う．

予測ミス発生時には後続の依存する命令の実行をキャンセルする必要がある．今回の評価では，キャンセルを行う必要のある命令全てを即時に求められるとしキャンセルする．

予測の制御は以下 2 通りの方法で行った：

VP　信頼度カウンタの値が，最大値である 3 のときのみ予測を行う．

VPP　VHT の予測値が実際の実行結果と一致するときのみ，すなわち完全な信頼度カウンタを持つ (VP-Perfect)．

4.2 評価結果

フロントエンド実行と値予測の IPC 向上率を図 **2** に示す．各プログラムごとに 6 本のバーがあり，左 4 つがフロントエンド実行，右 2 つが値予測の効果である．

フロントエンド実行について左から **F1**，**F2**，**F1C**，**F2C** の効果を示している．値予測については左が **VP**，右が **VPP** を示している．

VP とフロントエンド実行と比較すると全てのプログラムで良い性能向上が得られている．平均では **F1**:7.5%，**F2**:10.7%，**F1C**: 10.4%，**F2C**: 16.3% 性能が向上している．**F2C** では，予測ミスの発生しない理想的なモ

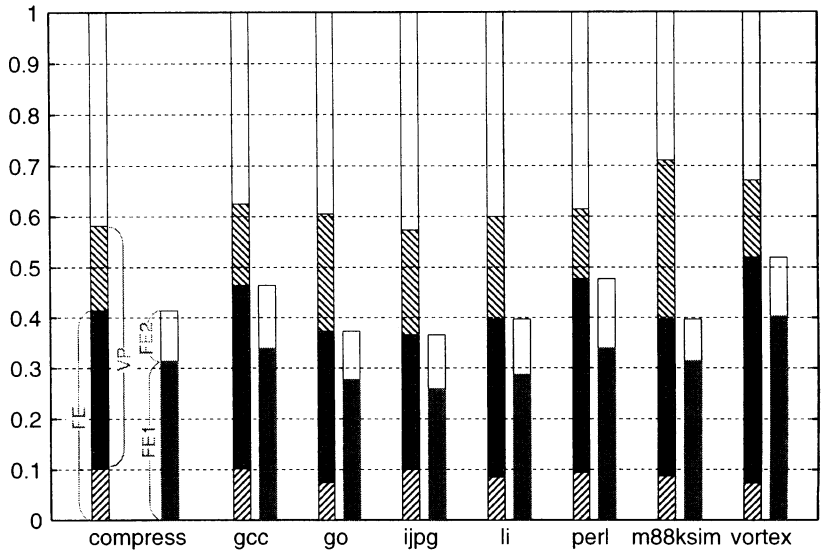

図 3 フロントエンド/値予測できた命令の内訳

デルである **VPP** に匹敵する性能を示している．

命令の内訳

F2C で実行できた命令/**VPP** で値予測が成功した命令の内訳を**図 3** に示す．左側のバーは下から，フロントエンド実行できたが値予測が成功しなかった命令，フロントエンド実行できかつ値予測が成功した命令，フロントエンドで実行できなかったが値予測が成功する命令，どちらもできなかった命令数の割合を示している．

また右側のバーはフロントエンド実行できる命令のうち，フロントエンド実行の 2 ステージのうちどちらのステージで実行できたかを示している．

表から次のようなことが読み取れる．まず全体の 3 割弱から 5 割の命令がフロントエンド実行できていることがわかる．また左側のバーのから値予測できる命令の半分以上はフロントエンドで実行できるような命令であるといえる．

5. おわりに

本章ではフロントエンドに演算器を配置し，実行可能な命令をフロントエンドでも実行するフロントエンド実行と呼ぶ手法を提案した．この手法ではデータ依存による先行制約を破ることなく値予測と同じ効果が得ることができる．その上投機ではないため予測ミス・ペナルティもなく，値予測と比較してハードウェア・コストも比較的小さいという利点を持つ．評価によってフロントエンド実行により平均 16.3% 性能が向上することが確認でき，これは理想的な値予測に匹敵する効果である．

謝辞

本研究の一部は文部省科学研究費補助金，基盤研究 (B)(2) #13480083 による．

参考文献

1) Lipasti, M. H. and Shen, J. P.: Exceeding the Dataflow Limit via Value Prediction, *29th. Int'l Symp. on Microarchitecture (MICRO-29)*, pp. 226–237 (1996).

2) 小西将人, 五島正裕: リザベーション・ステーションと物理レジスタ・ファイルを併用するスーパースカラ・プロセッサの構成方式, 先進的計算基盤システムシンポジウム SACSIS 2003, pp. 191–192 (2003). (ポスター).

3) 小西将人, 西野賢悟, 五島正裕, 中島康彦, 森眞一郎, 富田眞治: 直接依存行列型命令スケジューリングを適用したクラスタ化スーパースケーラ・プロセッサの評価, 情報処理学会研究報告 2002–ARC–149 (SWoPP 2002), pp. 151–156 (2002).

4) Wilson, K. M. and Olukotun, K.: High Bandwidth On-Chip Cache Design, *IEEE Trans. Comp.*, Vol. 50, No. 4, pp. 292–307 (2001).

5) 福田祥貴, 片山清和, 島田俊夫: ライン・バッファ・ヒット/ミス予測を利用した動的命令スケジューリングの高精度化手法, 先進的計算基盤システムシンポジウム SACSIS 2003, pp. 227–234 (2003).