フロントエンド実行によるプリロードの提案

小西将人[†] 福田匡則[††] 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．それに対して我々は，フロントエンド実行と呼ぶ手法を提案している．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．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．本稿では，フロントエンド実行によってロード命令のアドレス計算を行い，その直後からキャッシュ・アクセスを開始するプリロードを提案する．シミュレーションによる性能評価の結果，プリロードにより11.8%の高速化を達成できることが分かった．

A Preload Scheme by Frontend Execution

Masahito Konishi,[†] Masanori Fukuda,[††] 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. We propose 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. Frontend execution is to execute ready instructions by function units posed in the frontend as well as the backend. This paper describes a scheme called Preload, which calculates an address of a load instruction by frontend execution and accesses cache earlier. Evaluation result shows a processor with Preload is about 11.8 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．

それに対して我々は，**フロントエンド実行** (frontend execution) と呼ぶ手法を提案している[2]．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．

本稿ではロード命令のアドレス計算をフロントエンド実行で行い，フロントエンドでキャッシュ・アクセスを開始するプリロードを提案する. ロード命令に依存する後続命令は，ディスパッチ直後から投機的に実行可能とし，成功すればロードレイテンシは 0 に見える．

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機構について詳述する．4 章でプリロード機構について提案し，5 章で評価結果を示す．

2. フロントエンド実行の概要

スーパースカラ・プロセッサの命令パイプラインの，命

† 大阪工業大学
†† 京都大学

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．

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

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．

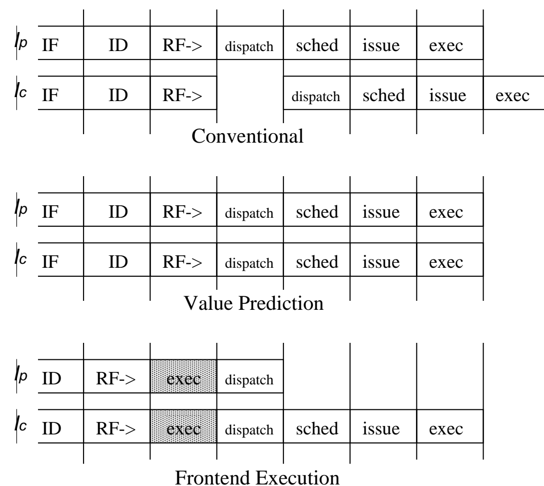

図 1 パイプライン

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，値予測と全く同様の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．

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．

2.4 フロントエンド実行の効果

値予測はデータ依存による先行制約を緩和することにより，クリティカル・パス自体を短縮することができる．

一方で，フロントエンド実行はデータ依存による先行制約を破ってはいないため，クリティカル・パス自体を短縮することはできない．したがって，従来のプロセッサでクリティカル・パス上の命令が back-to-back に実行できている部分には効果がない．

しかし実際には，以下のような要因で back-to-back にクリティカル・パス上の命令を実行できないことがある．フロントエンド実行は，従来のプロセッサで back-to-back に実行できていない部分に対して，実行間隔を縮める効果がある．

- 分岐予測ミス
- 演算器の不足

● スケジューリング能力の不足

図 2 に，フロントエンド実行によってクリティカル・パス上の命令の実行間隔が短縮されている例を示す．濃い色の四角形はクリティカル・パス上の命令を，白い四角形はそれ以外の命令を示している．

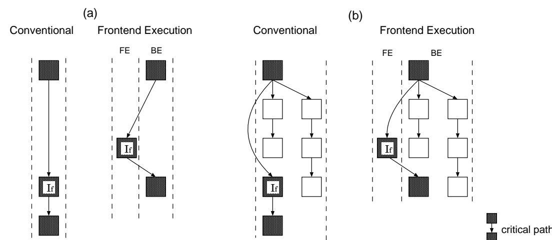

図 2　フロントエンド実行の効果

図 2(a) は分岐予測ミスにより，実行間隔がミス・ペナルティだけ開いている例である．正しいパスの先頭の命令 I_f がフロントエンド実行された場合，示している通り実行間隔は 1サイクル短縮される．

図 2(b) は，演算器の不足とスケジューリングの不足により，クリティカル・パス上にない命令によって，クリティカル・パスの実行が妨げられている場合である．演算器は 2 個であり，I_f がスケジューリングにより選ばれなかったために，実行間隔が開いている．I_f がフロントエンド実行された場合，示しているように実行間隔を縮めることができる．

3. フロントエンド実行機構

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．

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：

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．

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．

一方，物理レジスタでは，レジスタ読み出しステージをフロントエンドにも設ける必要があり，レジスタの読み出しポートは増加する．

3.2 フロントエンド実行ステージ

図 3 にフロントエンド実行機構の模式図を示す．図は RS+RB 方式を基本としたものであり，フロントエンドのレジスタ読み出し結果はリザベーション・ステーションに書かれる．

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．フロントエンド実行では，それだけ 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 フロントエンド実行データの書き戻し

フロントエンド実行で得られた結果を，後続の命令が利用するために，バックエンドの書き戻し処理と同様の処理が必要である．書き戻しはディスパッチと並列

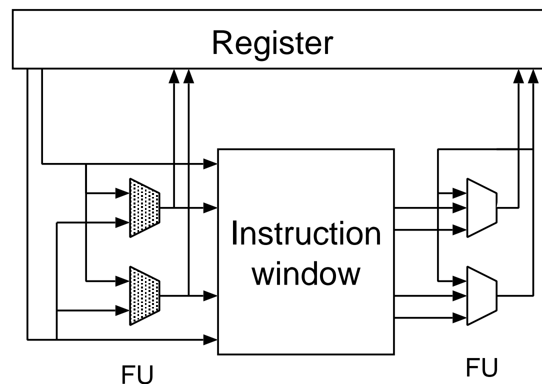

図 3　フロントエンド実行機構

に行う．

以降では，I_p のフロントエンド実行で得られた結果を，後続の命令 I_c が利用するとしよう．

物理レジスタの場合には，フロントエンド実行の結果をレジスタに書き戻すだけで良い．この結果を利用する I_c はバックエンドでレジスタからソース・オペランドを読み出す．この読み出しは，当然 I_c のディスパッチ・ステージの後であり，必ず I_p のフロントエンド実行の結果をレジスタに書き込んだ後に行われる．したがって物理レジスタの場合には，フロントエンド実行結果のレジスタ書き込みのみで，後続命令にデータを渡すことができる．

一方 RS+RB 方式の場合には，フロントエンドでレジスタ読み出しを行う．したがって I_c が I_p と同じフェッチ・グループに属する場合，I_p のフロントエンド実行結果のレジスタ書き込みは I_c のレジスタ読み出しに，に間に合わない．このとき I_c に対応するリザベーション・ステーション・エントリに，I_p のフロントエンド実行の結果を書き込む必要がある．これには通常のフォワーディングと同様の処理が必要である．

ハザードの回避のためには，以上で十分であり，フロントエンドの演算器からバックエンドの演算器へのバイパスは必要ない．I_p の結果は，I_c のディスパッチ・ステージ以前，すなわち I_c のバックエンドの実行ステージの 1サイクル以上前に得られるから，フロントエンドの演算器からバックエンドの演算器にバイパスを設ける必要はない．

4. プリロード機構

本稿では,フロントエンド実行によってメモリ・アクセス命令のアドレス計算を行い，このアドレスを利用してプリロードを行うことを提案する．

4.1 ロードのフロントエンド実行

ロードのフロントエンド実行を行うために，我々はフロントエンドに 1サイクルでアクセスできる小容量のキャッシュを用意する方法を提案している[2]．フロントエンド実行のために 2ステージ用意し，1 段目でアドレス計算を行い，2 段目でそのキャッシュにアクセスするのである．

図 4 上段にその実行の様子を示す，I_p はロード命令で，I_c はこれに依存する命令である．フロントエンド実行によって I_p のアドレス計算がなされた場合，次のステージでキャッシュにアクセスしている．1サイクルでロード・データが得られた場合，I_c は通常のフロントエンド実行の場合と同様ディスパッチ直後から実行可能となる．

ロード命令は，キャッシュのヒット/ミスによって動的に実行レイテンシが変化する．したがって，その実行ステージに固定サイクルを割り当てると，1サイクルでロード・データが得られる場合しかフロントエンド実行できない．従来の提案では，キャッシュ・ミスによりデータが 1サイクルで得られない場合には，フロントエンド実行は行われなかったものとし，ロード命令は通常通りバックエンドで実行していた．

4.2 プリロード

本来フロントエンド実行によって 1サイクルでデータが得られなくても，これを利用する命令のバックエンドの実行ステージまでにデータが到着すれば良い．フロントエンドからバックエンドの実行ステージまでには，ディスパッチ，スケジューリング，発行といったフェーズが存在するので，この間にデータが到着すれば良い．

本提案では，上記のような実行レイテンシの長いロード命令に対してもフロントエンド実行の効果を得るために，キャッシュ・アクセスにステージを割り当てず，フロントエンド実行されたロードに依存する命令を投機的に実行可能とする．

図 4 下段に提案機構による実行タイミングを示す．

I_p は，フロントエンド実行により参照アドレスが判明した直後からキャッシュ・アクセスを開始する．このとき I_c は I_p の結果が既に利用可能であるものとして，ディスパッチ直後から投機的に実行可能とする．I_c の実行が開始されるより前に I_p のデータが到着していれば，投機は成功であり，I_c は正しい値を用いて実行される．このとき I_c にとって I_p のレイテンシは 0 に見える．

一方，I_p のデータ到着が間に合わなければ，I_c 及びこれに依存する命令は再実行する必要がある．I_p のデータ到着時に，既に実行されている依存する命令をキャンセルする．この再実行機構は，通常の投機的なロード実行の再実行機構と同様に構成できる．

データ・パス

3.3 節で述べたように，通常のフロントエンド実行では，フロントエンド実行の結果はそれ利用する命令の実行ステージの 1サイクル以上前に得られ，新たにパイパスを設ける必要なかった．しかし提案機構では，ロードの実行レイテンシによってはハザードが生じる．

図 4 下段は，I_c の実行ステージの直前にロード・データが得られる場合である．このとき I_c はレジスタもし

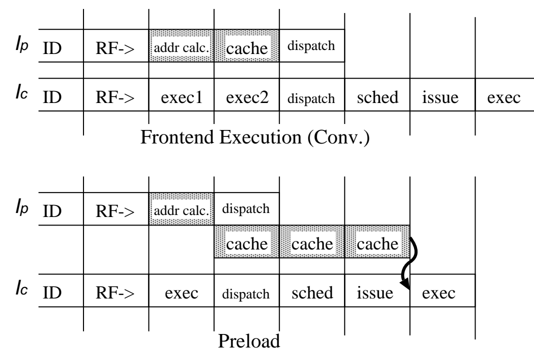

図 4 提案機構のパイプライン

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

くはリザベーション・ステーションを介してデータを得ることはできず，ハザードが生じる．この場合，I_c にデータを受け渡すには，キャッシュから演算器にバイパスが必要である．

このキャッシュはフロントエンド実行で述べた書き戻しに加えてバックエンドの演算器へのバイパスを持つので，従来のキャッシュと同等である．したがって，4.1 のようにフロントエンドにキャッシュを設けることはせず，フロントエンドで求めたアドレスでバックエンドの通常のキャッシュにアクセスする．

5. 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とプリロードを実装し IPC を計測した．評価には SPEC CINT95 の 8 つのプログラムを実行した．

5.1 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

ベース・モデルに対し以下 3 つのモデルを評価・比較した．

FEX　フロントエンドに 1ステージの整数ユニットを持ち，整数演算命令のみフロントエンド実行する．

FEX+C　4.1 節で述べたように，フロントエンドに 0 次キャッシュと同じ大きさのキャッシュを持ちヒット時のみフロントエンド実行する．

PLOAD　4.2 節で述べたように，フロントエンド実行されたロードに依存する命令を投機的に発行する．フロントエンドからバックエンドの実行ステージ開始の間は 3サイクルであるから，1 次キャッシュ・ヒットの場合まで投機は成功となる．

5.2 評価結果

各モデルのベース・モデルからの IPC 向上率を**図 5** に示す．各プログラムごとに 3 本のバーがあり，左から **FEX FEX+C**, **PLOAD** を示している．

まず整数演算命令のみフロントエンド実行を行う **FEX** は，平均で 7.4% の性能向上が得られている．

さらに，フロントエンドにキャッシュを設けて，ヒットした場合にはロード命令のフロントエンド実行を行う **FEX+C** では，平均で 8.5% の性能向上が得られている．**PLOAD** は一部を除いて，**FEX+C** より高い性能を示し 5〜15%程度，平均では 11.8%の性能向上を得られた．

6. おわりに

本章ではフロントエンド実行によってロード命令のアドレス計算を行い，その直後からキャッシュ・アクセスを開始するプリロードを提案した．この手法では，プリロードされた命令に依存する命令を投機的に実行可能とすることで，投機に成功した場合ロード・レイテンシは 0 とみなすことができる．評価により，プリロードにより平均 11.8% の性能向上を得られることがわかった．

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, 中島康彦, 森眞一郎, 富田眞治: フロントエンド実行, 先進的計算基盤システムシンポジウム SACSIS 2004 (2004). （ポスター）.

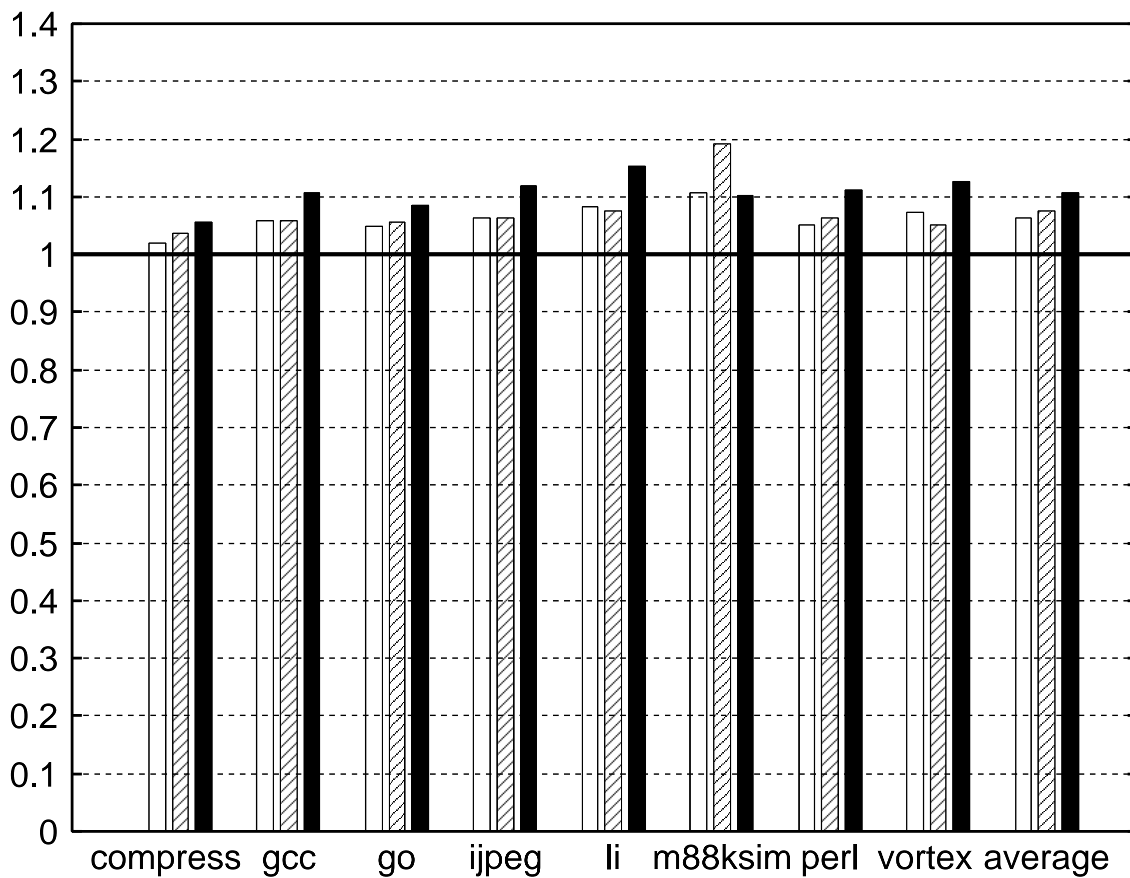

図 5　性能向上率