PRIMEPOWER向けコンパイラバックエンドの最適化方式

5F-7

瀧 康太郎　松山 学　原口 正寿　中平 直司

富士通株式会社

1. はじめに

当社では，UNIXサーバPRIMEPOWER[1]の開発を，高速化かつ高信頼性の追求を目的として進めてきた．最新のPRIMEPOWERには，高速化のためにハードウェアレベルで数々の機能が追加されており，性能を最大限に引き出すコンパイラが必要となる．

我々は，最新のPRIMEPOWERにおける高速化を目的に，FORTRAN及びC言語向けコンパイラの最適化開発を進めてきた．本論文では，PRIMEPOWER向けに開発した最適化パスにおける種々の最適化手法について述べ，最後にその効果について触れる．

2. 最適化の方針

今回バックエンドの開発にあたって，PRIMEPOWERが持つレジスタ，演算器，キャッシュメモリといったハード資源を最大限に活用することを大きな目標とした．このためにプリフェッチを生成する技術の開発を行い，さらに，レジスタ割り付け，命令スケジューラといったハード依存最適化部分に対して改良を加えた．

また，最新のアルゴリズムを取り込み，かつ各種最適化間の連係を図るために，中間言語の見直し，及び最適化パスの見直しを実施した．

3. 各種最適化手法

各種最適化パスの構成を図1に示す．

3.1. プリフェッチの生成

新しいPRIMEPOWERでは，ハードウェアによるプリフェッチ機構により，プリフェッチ命令を生成しなくてもある程度の性能向上が見込めるが，さらにコンパイラがデータフロー解析や制御フロー解析などの結果を活かしたプリフェッチを生成することで，より性能を向上させることができる．したがって，ハードウェアによるプリフェッチの効果を阻害せず，かつそれを補完する形で，コンパイラによるプリフェッチ命令を生成する．

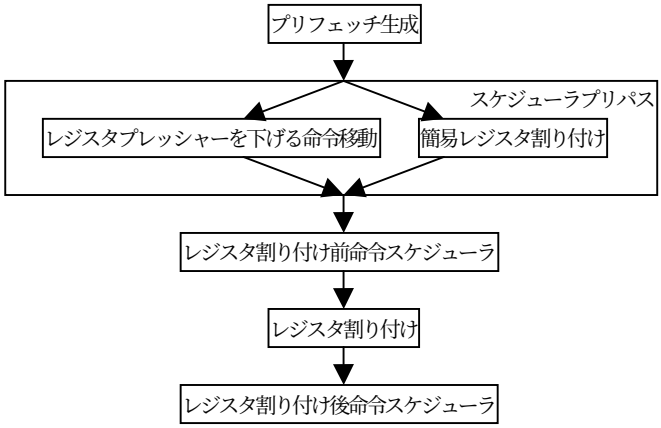

図1　ハード依存最適化パスの構成

また，1次キャッシュと2次キャッシュへのプリフェッチをコンパイラで指定することができる．この機能を有効活用するため，オプションによってプリフェッチ命令の生成を細やかに制御することを可能にした．さらに，プリフェッチ指示行を提供し，より細かい単位でプリフェッチ命令を生成することを可能にした．

3.2. レジスタ割り付け部と命令スケジューラ部の連係

命令スケジューラとレジスタ割り付けは相互に干渉する．例えばレジスタ割り付けを単純に行うと，命令間の出力依存及び逆依存が増加することによって，命令の並列度を高めることができない．この点について，命令スケジューラをレジスタ割り付けの前後でそれぞれ1回ずつ動作させることで解決した．

ただし，レジスタ割り付け前の命令スケジューラで並列度を過度に高めてしまうと，演算の途中結果を保持する変数が多く必要になる．そのため割り付けるレジスタが不足し，スピルコードの増加を招く．この点については，命令の個数によって移動範囲を限定し，後述の3.3.や3.4.に挙げる工夫を加えることで，並列度が過剰に高くなることを抑止している．

Optimization Infrastructure of the Compiler Back End for PRIMEPOWER.
Kotaro TAKI, Manabu MATSUYAMA, Masatoshi HARAGUCHI, Tadashi NAKAHIRA
Fujitsu Limited

3.3. レジスタプレッシャーを下げる命令移動

命令を移動して全ての変数の生存範囲を狭めて，一旦並列度を低下させ，その後命令スケジューリングを行うことで，並列度の調節を行っている．

まずデータフローグラフを作成し，各命令に重み付けを行う．重みは他の命令との依存関係の個数によって決定される．この重みが高いものほど他と多く干渉している命令であるから，前後の変数の生存範囲を狭めて，干渉を減らす．これを繰り返すことによって同時に生存する変数の最大値を減らし，レジスタプレッシャーを減少させる．

3.4. 簡易レジスタ割り付け

擬似的レジスタ割り付けを行う．この結果を利用して，レジスタ割り付け前の命令スケジューリングを行うことで，並列度を調節する．

簡易レジスタ割り付けは各変数に擬似的にレジスタを割り付けるもので，実際にアセンブリコードとして出力されるレジスタ割り付け結果とは異なる．しかし，レジスタが割り付く程度に命令スケジューリングを行うには十分である．

3.5. 命令スケジューラ

命令スケジューラでは，それぞれの基本ブロックに対してリストスケジューリング[2]を行い，ハード資源の有効利用を図っている．

命令スケジューリングを行うために必要なレイテンシ情報や PRIMEPOWER 特有の命令間依存情報などはハード依存情報としてデータベース化し，アルゴリズムと切り離した．このことによって複数のアーキテクチャにも容易に対応できるよう設計している．

3.6. レジスタ割り付け

レジスタ彩色法[3]により，各関数に対してグローバルなレジスタ割り付けを行う．[3]の手法ではレジスタを割り付けた変数をスタックに蓄えていく．さらに以下の工夫を加えた．

スピルすべき変数は，生存範囲の包含関係を見て，なるべく他の生存範囲を包含している変数を選ぶ．他の生存範囲を多く包含する変数は，干渉が多い変数である．したがって，この変数をスピルすることで干渉を減らし，スピルコードを削減することができる．

また，上記の包含関係が存在していない場合，ある変数をスピルするときに同時に犠牲となってスピルされる変数を選択する．このことによって，レジスタ彩色法による割り付けが早く収束し，翻訳時間を節約できる．

スピルコードを生成する位置については，後述の 3.7. の方法を用いて，最適な位置に生成している．

3.7. スピルコード生成位置の決定法

変数を退避する範囲を考えることで，スピルコードを生成する位置を戦略的に決定する．

この方法では，変数の退避が必要な範囲，もしくは割り当てられるレジスタを解放しなければいけない範囲を，レジスタの数などのハードウェアの制限によって特定して，この範囲外にスピルコードを生成する．位置については，なるべくスピルコードが実行されないように，実行回数の予測または計測結果や，制御解析結果を利用して，コストの低い位置（例えばループの外）に生成する．

4. 効果

新しい最適化パスを使用して，SPEC2000 ベンチマークの Fortran プログラム（全 10 本），及び実コードの Fortran プログラム（全 71 本）を実測した．その結果を図 2 に示す．

	旧コンパイラ	新コンパイラ
SPEC2000	1.00	1.22
実コード	1.00	1.09

図 2　実行性能比（旧版を 1 とした平均値）

上図に示した通り，旧コンパイラの最適化パスを使用した場合と比較して約 10%～20%の性能改善を図ることができた．

5. 今後の課題

現段階では命令スケジューリングが基本ブロック単位で行われているため，広域スケジューリングを行う．さらに，命令スケジューラにおける出力依存や逆依存をコンパイラ側で解消するレジスタリネーミングを行うという案について検討中である．

参考文献

[1]http://primeserver.fujitsu.com/primepower/
[2]コンパイラの構成と最適化，中田 育男，朝倉書店，ISBN 4254121393
[3]Iterated Register Coalescing, George, L. and Appel, A. W., POPL '96, pp-208-218, 1996