命令スケジューリングアルゴリズム

3Q－7

竹内陽一郎，境隆二

(株) 東芝　情報処理・機器技術研究所

1　はじめに

パイプライン演算器や命令並列実行ハードウェアを有効に利用するためのコード最適化技術として、命令スケジュールアルゴリズムの重要性が高まってきている。

現在、広く使用されているリストスケジュールアルゴリズムは、実用的な計算コストで、性能的にはほぼ最適な解を与えるが、変数の生存範囲を大きくしすぎるなど実用的な面で、まだ課題が大きい。

本論文では、このような問題を解消する、新しい命令スケジュール手法として、2フェーズリストスケジュールを提案する。以下、リストスケジュールの問題点を示し、これが新手法でどう解決されるかを中心に説明を進める。

2　リストスケジュールアルゴリズム

最初にリストスケジュールアルゴリズムについて説明する。

（1）命令の依存関係を解析し、命令をノードとして先行関係をエッジで表すグラフ（以下ＤＡＧと呼ぶ）を作成する。

（2）ＤＡＧの各エッジには、先行ノードの命令が実行されてから、後続ノードが実行可能になるまでの遅延時間を、各ノードには、その命令をスケジュールする際のウェイト（通常ＤＡＧ上でのそのノードの高さ）を付加する。

（3）先行制約のないノードをＤＡＧからとりだし、最短実行可能時刻を先行ノードがスケジュールされた時刻と、エッジに付加された遅延時間から求め、この値およびウェイトで優先順位づけされたリスト（スケジュールリスト）に追加する。

（4）スケジュールリストから順に命令をとりだし、これがスケジュール可能（現在時刻≧最短実行可能時刻）であれば、スケジュールする。スケジュール可能な命令が何もなければ、時刻をカウントアップして、リトライする。

（5）以下全命令がスケジュールされるまで、（3）から繰り返す。

3　リストスケジュールの問題

(i) 変数（レジスタ）使用量の増大

2のステップ（4）によって、リストスケジュールでは、そのウェイトにかかわらず、実行可能な命令があればそのサイクルにスケジュールする。この性質によって、リストスケジュールアルゴリズムは、同時実行できる命令の数が有限の場合に対して、ほぼ最適な解を与える。

しかし、基本ブロックが大きくなった場合、ステップ（4）によって、実行可能な命令が、不必要にブロック上部に固まってしまう結果、変数の生存範囲が大きくなり、レジスタアロケーション時に、必要となるレジスタの数が不必要に大きくなりすぎる傾向がある。基本ブロックのサイズが大きくなると、最悪の場合レジスタスピルが多発する。

この問題が実用上一番大きな問題である。実行性能を犠牲にせず、この問題にリストスケジュールの枠内で対処することは極めて困難だといえる。

(ii) 遅延スロットへの命令スケジュール

遅延スロットを持つ命令に対して、依存関係をＤＡＧ上に表現しようとすると、遅延サイクルが負のエッジを導入せざるをえない。（図2のＤＡＧ参照）これを、リストスケジュールの枠組みに取り込むと、先行ノードがスケジュールされた後、ｎサイクル遡って後続命令を再スケジュールする必要がある。しかし、すでにスケジュールが終わったサイクルは、低優先度の命令ですでに埋められている可能性が高く、この方法ではうまくいかない。ｎサイクル分ｕｎｄｏして再スケジュールすることも考えられるが、再スケジュールの際、先行ノードがすべてスケジュールされることを保証しなければならないため、極めて複雑な処理となる。

(iii) 逆ｔｒｅｅ形状の依存関係

リストスケジュールでは、ｔｒｅｅ状の依存関係に対しては、最適になるが、逆ｔｒｅｅ状の依存関係に対しては、後続ノード数を、ウェイトの評価にうまく組み入れない限り、最適にはならない。（図1参照）また、ウェイトの評価にうまく取り込んだとしても、計算コストが非常に高くなり、実用的ではなくなる。

An Algorithm for Instruction Scheduling

Yoichiro TAKEUCHI , Ryuji SAKAI

TOSHIBA CORPORATION

4　2フェーズリストスケジュール

以上のような、リストスケジュールの問題を解決するため、以下に示すアルゴリズムを導入する。

（1）前向きリストスケジュールフェーズ（ＦＬフェーズ）

2のステップ（1）から（5）で示される、リストスケジュールアルゴリズムで一度トップダウンに命令をスケジュールする。（空きサイクルは厳密にｎｏｐをいれる。）

（2）後向きコード圧縮フェーズ（ＢＣフェーズ）

ＤＡＧを逆向きにする。

リストスケジュールと同じ方法で、逆向きのＤＡＧをボトムアップにたどり、選択された命令に対し、逆向きＤＡＧの先行関係を満たす範囲内で、可能な限り後方に命令を移動する。ただし、リストから命令をとりだす優先順位は、

(i) 逆向きＤＡＧ上での高さが高いもの

(ii) 同じ高さのとき、ＦＬフェーズでのスケジュール位置が後方にあるもの

というルールできめる。

(ii) の条件は、命令が移動した際に生じる穴が、後続の命令でなるべく有効利用できるようにすることを狙っている。

このアルゴリズムの適用例を図1に示す。

従来のＦＬフェーズにＢＣフェーズを付加することで、以下のようなコードの改善が得られる。

（a）ＦＬフェーズで得たクリティカルパス長を越えない範囲内で変数の生存範囲を大幅に短縮する。

（b）ＦＬフェーズでは最適になりにくい、逆ｔｒｅｅ依存関係を含むケースの実行サイクルを改善する。（図1参照）

（c）特に意識しなくとも、遅延スロットへの命令再配置が、有効に行なわれる。（図2参照）

ＢＣフェーズのアルゴリズムは、基本的にＦＬフェーズと同じであるため、計算コストは、ｏ（ｎｌｏｇ（ｎ））ですむ。従って、実用的には全く問題ない。（空きフィールドの検索および情報のアップデートは2分探索木を利用することにより、ｏ（ｌｏｇ（ｎ）で可能。）

5　評価と今後の課題

現在、2フェーズリストスケジュールアルゴリズムに基づく、命令スケジューラを、試作評価中である。ループ展開などで生じる大規模な基本ブロックに対して、非常にうまくコード生成できることを確認している。また、遅延スロットや、複雑なリソース競合による依存関係をとりいれた場合、従来の方法にくらべ、かなり性能が改善されるという結果を得ている。

今後の課題としては、ＢＣフェーズでの命令選択のヒューリスティックルールが妥当かどうかの検討と、これを変えた場合にスケジュール結果がどう変化するかについての調査が必要であろう。

参考文献

(1) J.R.Ellis Bulldog: A Compiler for VLIW Architectures
The MIT Press, 1986.

(2) H.S.Warren,Jr. Instruction schedulinng
for the IBM RISC System/6000 processor
IBM J. RES. DEVELOP. VOL,34 No.1 Jan 1990

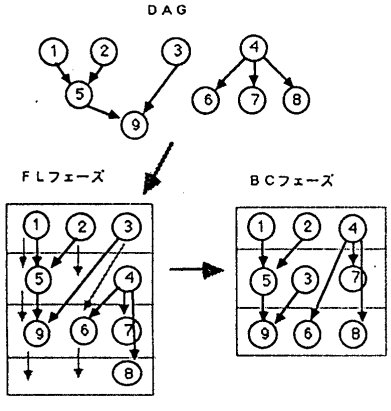

図1　2フェーズリストスケジュール実施例

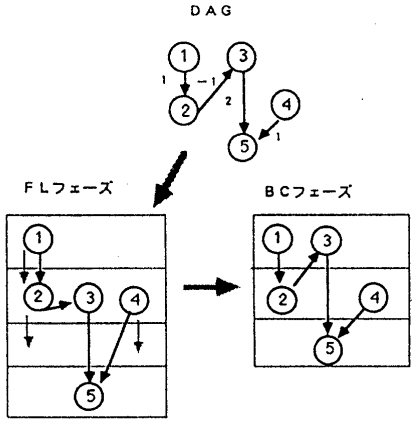

図2　遅延スロットへのスケジュール