スケジューリング時におけるレジスタ不足の解消方式に関する研究

4U－5

志水 浩司† 李 鼎超‡ 石井 直宏†

†名古屋工業大学知能情報システム学科 ‡名古屋工業大学情報処理教育センター

1 はじめに

逐次実行の計算機において、レジスタ不足時にスピルコードを使用することは、プログラムの遅延の原因となるため極力避けられてきた。しかし、並列実行可能な計算機においては、スピルコードを他の命令と並列に実行し、プログラムに遅延を起こさないようにすることが可能である。本稿では、プログラム内の命令の実行タイミングの解析情報を基に変数の生存区間を予測する方法について述べる。その予測情報を基に、スケジューリング時にレジスタ不足が生じた時にレジスタ再利用およびレジスタスピルによるプログラムへの影響を動的に計算しレジスタ不足を解消する。これにより、コンパイラが選択的にスピルコードを使用し、プログラムのスケジューリング長を可能な限り縮める事ができる。

2 計算機及びプログラムモデル

本稿で対象とする計算機はVLIWアーキテクチャをベースとする。レジスタは全ての機能ユニットからアクセスでき、固定少数点数と浮動少数点数を共に扱うことのできる共有レジスタを持つ。レジスタは変数を定義する命令の実行開始から占有され、変数を最後に使用する命令の終了時に解放される。

プログラムモデルは、通常のコンパイラによって作成された中間プログラムを対象とする。その一連のプログラムにおける基本ブロックに対して本研究におけるコードスケジューリング及びレジスタ割当を行い、VLIW命令コードを完成させる。その際、プログラムの並列性をDAGグラフで表現する。また、$\mu(I_i)$は命令I_iの実行時間を表し、$\nu(I_i)$は命令I_iが実行可能な機能ユニットの種類を表す。

最適化処理に必要な命令の実行タイミングの予測値を計算する方式として[1]の方法を採用する。これは、機能ユニットによる資源制約を考慮した命令の実行タイミングの解析手法である。この手法により、ある命令I_iの実行が可能となる最も早い実行開始時刻$\tau_{es}(I_i)$および、I_iがプログラム全体の実行に遅延を起こさずに実行できる最も遅い実行開始時刻$\tau_{ls}(I_i)$が求められる。

"An approach for code scheduling in presence of register constraints" Kouji Shimizu, Dingchao Li, Naohiro Ishii
Nagoya Institute of Technology
Gokiso-cho, Syowa-ku, Nagoya 466-8555, Japan

3 本アルゴリズムの説明

CP法などのリストスケジューリング法によりスケジューリングする命令I_iを選択する。その際I_iに割り当て可能なレジスタが存在する場合は同時にレジスタを割り当て、存在しない場合は以降で述べるレジスタ不足の解消法を用いて確保したレジスタを命令に予約しておき、後にレジスタが使用可能になった時にスケジューリングを行う。

本稿では、レジスタ不足の解消のために次の2つの方法を考慮する。1つ目の方法は、他の命令によって現在占有されているレジスタについて、レジスタを参照する命令が終了するまで命令I_iの実行を待機させるという方法である。このときの命令の実行待機により必要なレジスタを確保する方法をレジスタ再利用と呼ぶ。

I_iに割り当てるためのレジスタが不足して、レジスタ不足の解消のためにレジスタ再利用を行なうとする。レジスタ再利用を行なうレジスタをrとし、現在のrの値を定義した命令をI_jとする。図1のように、I_jの直接後続ノードの全てから、I_iへ新たなアークを加えることで再利用を行なう。ただし、I_iの後続ノードにI_jの直接後続ノードのいずれかが含まれる場合、rについて再利用を行なわないことにする。

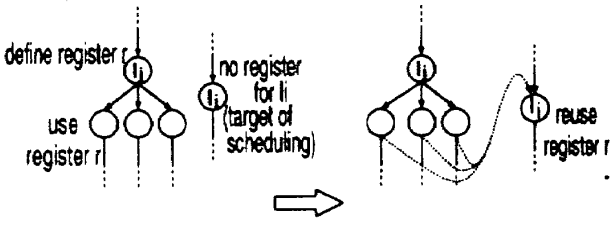

図1: レジスタ再利用方式

2つ目の方法は、他の命令によって占有されているレジスタの値をスピルコードを用いてメモリに退避することによって必要なレジスタを確保するという方法である。メモリに退避された値は、その値を再び使用する命令よりも前のタイミングでロード命令によって復帰されなければならない。

スピルコードの挿入によって確保されるレジスタをrとし、現在のrの値を定義した命令をI_jとする。図2のように、スピルコードの挿入を行なう。適切な順序で命令が実行されるよう、I_jおよび、ス

ケジューリング済の I_j の直接後続命令からストア命令に新たなアークを加え、ストア命令から I_i にアークを加え、ストア命令からロード命令へアークを加え、ロード命令から I_j の未スケジューリングの直接後続命令にアークを加える。

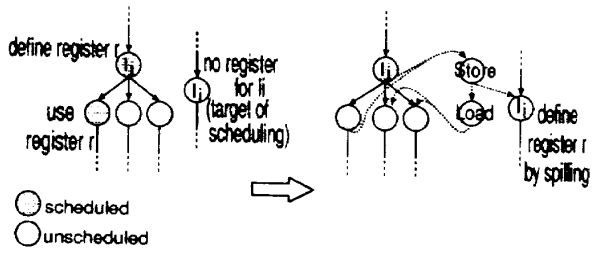

図 2: スピルコードの使用法

以上の2つのレジスタ不足の解消法のどちらを使用するかを、再利用による遅延コスト $Cost_{reuse}$ および、スピルコード使用による遅延コスト $Cost_{spill}$ を以下に述べる方法を用いて見積り、$Cost_{reuse} \leq Cost_{spill}$ となる場合はレジスタ再利用を行い、それ以外の場合はスピルコードを使用する。

レジスタ r を最後に参照する命令の終了タイミングを $\tau_f^R(r)$ とすると、$Cost_{reuse} = \min_{r \in R}\{\tau_f^R(r)\} - \tau_{ls}(I_i)$ となる。

$Cost_{spill}$ はストア命令による遅延コスト $Cost_{spillout}$ と、ロード命令による遅延コスト $Cost_{spillin}$ の和として見積もられる。まず $Cost_{spillout}$ の計算法について述べる。現在、レジスタ r が他の命令によって最後に参照された時刻を $\tau_f^{use}(r)$ とし、Load/Store ユニットの使用が完了した時刻を $\tau_f(LD/ST)$ とすると、ストア命令の実行開始時刻は $\tau_s(Store) = \max\{\tau_f(LD/ST), \tau_f^{use}(r)\}$ となり $Cost_{spillout} = \tau_s(Store) + \mu(Store) - \tau_{ls}(I_i)$ と計算する。

つぎに、$Cost_{spillin}$ の計算方法について述べる。まず、ストア命令の挿入による遅延を考慮し、実行タイミングを再計算し、求められた命令の最早命令実行開始時刻を $\tau_{es}^{spill}(I_i)$ とし、最遅命令実行開始時刻を $\tau_{ls}^{spill}(I_i)$ とする。また、プログラムにおいて使われている変数の生存区間を予測することで必要なレジスタ数を判断する。ある変数 v が命令 I_p で定義され、命令 I_q で最後に使用されるとすると、v の生存区間を、I_q がリロードされた値を必要とする命令である場合 $[\tau_{es}^{spill}(I_p), \tau_{ls}^{spill}(I_q) + \mu(I_q)]$ とし、それ以外の場合 $[\tau_{es}^{spill}(I_p), \tau_{es}^{spill}(I_q) + \mu(I_q)]$ とする。変数の生存区間の情報により、ある区間 $[\theta_1, \theta_2]$ における必要なレジスタ数 $V(\theta_1, \theta_2)$ を求めることができる。図3は、$Cost_{spillin}$ およびロード命令の最適実行時刻 $\tau_s(Load)$ を求めるアルゴリズムである。

```
Procedure _estimate_spillin_cost()
 Begin
  Cost_spillin(r) = ∞
  for τ_f from τ_ls^spill(I_v) to τ_s^spill(I_u) + μ(Load) Do
   τ_s = τ_f − μ(Load)
   I_LD/ST(τ_s, τ_f) = μ(Load)
   for each instruction I_i ∈ I_unscheduled Do
    if η(τ_s, τ_f, I_i) ≠ 0 and ν(I_i) = LD/ST then
     I_LD/ST(τ_s, τ_f) += max{0, min{τ_f − τ_ls^spill(I_i),
          τ_es^spill(I_i) + μ(I_i) − τ_s, τ_f − τ_s, μ(I_i)}}
   δ_Load(τ_s, τ_f) = I_LD/ST(τ_s, τ_f)/m_LD/ST − μ(Load)
   if |V(τ_s, τ_f)| ≥ |R| then
    return Cost_spillin
   else
    if Cost_spillin > δ_Load(τ_s, τ_f)
     Cost_spillin(r) = δ_Load(τ_s, τ_f)
     τ_s(Load) = τ_s
     if δ_Load(τ_s, τ_f) ≤ 0 then
      Cost_spillin(r) = 0
      return Cost_spillin
 End.
```

図 3: $Cost_{spillin}$ 及びロード命令挿入時刻の計算

4 評価結果、まとめ

ここでは、代表的なベンチマークプログラムである、Livemore kernel のプログラムについて本研究の方式による実験を行なった。比較対象として、レジスタ不足の解消方式において再利用を優先して使い、スピリングは再利用が不可能な時にのみ行うという従来の方式による実験も同時に行なった。2つの方式のスケジューリング長を様々なプログラムおよびレジスタ数において比較した。この実験においては、VLIW プロセッサは整数演算ユニットを2個、浮動少数点演算ユニットを2個、Load/Store ユニットを2個持つものとしている。

53 の例による結果によると、本方式の方が従来の方式よりも優れている例が 19 例存在した。逆に、従来の方式の方が優れている例が 1 例存在した。

この論文では、スケジューリング時にレジスタ数が不足した時にコンパイラが効果的にスピルコードを使用するための方式について述べた。実験結果では、従来の方式よりも優れたスケジューリング長でスケジューリングする事ができた。今後の課題としては本方式をより多くのプログラムについて実験し、その有効性を評価する事が挙げられる。

参考文献

[1] 李 鼎超, 有田 隆也, 石井 直宏, 曽和 将容, 情報処理学会論文紙, Vol.34, No.11, pp.2378-2385, Nov. 1993.