仮想スケジューリングに基づくレジスタ割り付け方式

5D－5

本川　敬子　　十山　圭介
（株）日立製作所システム開発研究所

１．はじめに

最適化コンパイラにおいてレジスタ割り付けと命令スケジューリングは相互に干渉する。例えばレジスタ割り付けで使用レジスタ数を最小化するために同じレジスタを再利用すれば、同じレジスタの使用-再定義の組が命令スケジューリングに制約を与えることになる。この干渉を緩和するような２つの最適化の組み合わせ方式としては、DAGを対象にレジスタ割り付けを行うもの[1]や、対象とするマシンや中間語の限定により木を辿りながら２つの最適化を同時に行うもの[2]などがある。

本稿では、木構造の中間語を辿りながら、命令実行順序を想定しつつレジスタを割り付ける方法を提案する。本方式は様々なアーキテクチャや中間木を対象とすることが可能であり、使用レジスタ数を増やし過ぎることなく命令スケジューリングへの悪影響をなくすことができる。

２．仮想スケジューリングに基づくレジスタ割り付け方式

２．１　前提と定義

本方式ではロード結果を直後の演算で参照する場合に１サイクルの遅延を生じるスカラプロセッサをターゲットとし、木構造の中間語を対象としてレジスタ割り付けを行う。

本方式で使用する用語を定義する。

・遅延エッジ

中間語のあるエッジに対して、エッジの子ノードがロードノード、親ノードが子のロード結果を使用するノードであるとき、そのエッジを「遅延エッジ」と呼ぶ。遅延エッジの両端のノードはロード・ユースの関係にあるので、この２つのノードに対する命令が連続して実行されるとき、遅延が発生する。

・潜在遅延数

ノードｎの潜在遅延数とは「ルートからｎまでを結ぶ経路上の遅延エッジの数」である。潜在遅延数はノードｎ以降、親の発行する命令列で発生する可能性のある遅延数を表している。図１の中間語を例に各ノードの潜在遅延数を示す。

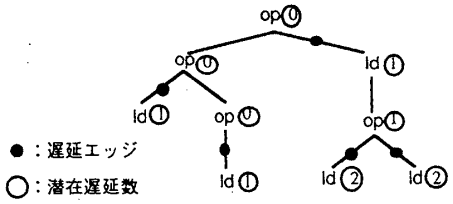

図１　潜在遅延数の例

・命令リストと命令エントリ

想定する命令実行順を保持するデータ構造として「命令リスト」を使用する。命令リストの要素を「命令エントリ」と呼び、次の３種類がある。

・(ld r1)　　：r1へのロード。
・(D r1)　　：r1のロード・ユースによる遅延。（遅延エントリ）
・(op r1[r2])：演算。割り付けレジスタはr1、演算後にr2を解放。

２．２　レジスタ割り付け方式

本レジスタ割り付けは基本ブロックを処理単位とする。基本ブロックの中間語の木を辿り、帰りがけ順にノードを処理する。各ノードの処理ではレジスタの選択および命令リストの更新を行う。本レジスタ割り付けは次の２つの方針に基づいて遅延を回避する。

［方針１］遅延が発生したら、なるべく早く他の命令を挿入して遅延を埋める

［方針２］潜在遅延数が正のとき、その祖先ノードに対応する命令列を割り付け済みノードの命令列の間に埋め込むことにより、遅延発生を防ぐ

各ノードの処理について以下に説明する。

（１）レジスタの選択

子ノードを持つノードでは、なるべく子と同じレジスタを割り付けるようにする。子ノードを持たないノードの割り付けレジスタの選択について以下に述べる。レジスタ選択は命令リストの状態（仮想スケジューリングの結果）に基づいて行う。命令リストは処理中の基本ブロックで既に割り付けが終了したノードに対応する命令エントリを要素とし、想定した実行順序に従ってリストされている。命令リスト上の遅延エントリの有無により次のようにレジスタ選択を行う。

・命令リストに遅延エントリがある場合

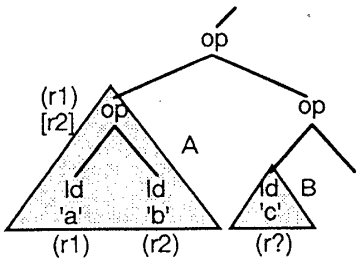

図２　方針１に対応する例

方針１に基づきレジスタを選択する。図２の中間語では部分木Ａの割り付け後にノードＢの割り付けを行う。Ａの割り付け後の命令リストは次のとおりである。

(ld r2) - (ld r1) - (D r1) - (op r1[r2])

Ａのルートで解放したレジスタr2をＢに割り付けたとする

Register Allocation using simulated Instruction Scheduling
Keiko MOTOKAWA, Keisuke TOYAMA
Systems Development Laboratory, Hitachi, Ltd.

と、レジスタ再利用による制約のためBの命令でAの遅延を埋められない。Bに別のレジスタ(r3)を割り付ければ、命令リストを

(ld r2) - (ld r1) - (ld r3) - (op r1[r2])

のように更新でき、遅延を埋めることができる。

遅延エントリがある場合のレジスタ選択は、遅延エントリより前のエントリで解放されたレジスタがあればそれを選択し、なければ新たなレジスタを選択するものである。

・命令リストに遅延エントリがない場合

方針2に基づき次の条件をみたすレジスタrを選択する。

（rを解放する命令エントリより後ろのエントリ数）
≧（潜在遅延数）　・・・（＊）

なければ新たなレジスタを選択する。

（2）命令リストの更新

割り付けレジスタrを選択したら、新たな命令エントリを作成し命令リストに挿入する。挿入範囲は子供のノードの命令エントリより後ろかつ、rが解放されたエントリより後ろである。この範囲内で次の基準に従い命令エントリを挿入する。

・遅延エントリがある場合

本命令エントリがオペランドとして遅延エントリのレジスタr´を使用しないならば、その遅延エントリを本命令エントリで置換する。この置換は方針1に対応する。

・遅延エントリがない場合

（潜在遅延数）＞0ならば、挿入範囲の先頭に挿入する。但し先頭への挿入により遅延が生じる場合は1つ後ろに挿入する。これは方針2に対応し，
なるべく前方にエントリを挿入し、すでに命令リストにあるエントリの間にこれから生成するエントリを挟むことにより、遅延発生を防ぐものである。

（潜在遅延数）＝0ならば、挿入範囲の最後に挿入する。これは条件（＊）を満たすレジスタ数をなるべく多くするためである。このとき必要ならば遅延エントリも作成して挿入する。

3．レジスタ割り付けの例

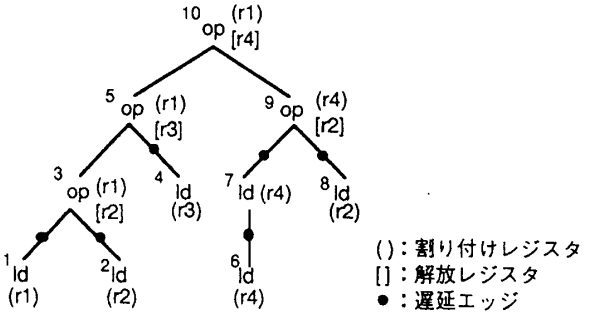

図3　本方式によるレジスタ割り付けの例

図3の中間語を例にレジスタ割り付けの過程を示す。以下、各命令エントリには先頭にノード番号を付けて中間語との対応を示す。ノード1、2、3での処理は、

1：r1を割り付け。2：r2を割り付け。潜在遅延＞0なので先頭に挿入。3：1の後ろに追加するので遅延発生。

となり、この結果命令リストは次の状態となる。

(2 ld r2) - (1 ld r1) - (D r1) - (3 op r1[r2])

以下、各ノードについて上記手順に従い処理を行う。ノード4では方針1に基づき新たなレジスタを選択し、ノード6では方針2に基づき条件（＊）により新たなレジスタを選択する。またノード8では条件（＊）を満たすr2を再利用する。処理過程の命令リストの状態を以下に示す。

(2 ld r2) - (1 ld r1) - (4 ld r3) - (3 op r1[r2]) - (5 op r1[r3])

(6 ld r4) - (2 ld r2) - (7 ld r4) - (1 ld r1) - (4 ld r3) - (3 op r1[r2]) - (5 op r1[r3])

(6 ld r4) - (2 ld r2) - (7 ld r4) - (1 ld r1) - (4 ld r3) - (3 op r1[r2]) - (8 ld r2) - (5 op r1[r3]) - (9 op r4[r2]) - (10 op r1[r4])

4．本方式による効果

表1は、SPECベンチマークespressoの上位3関数について、本方式によるロード・ユースインタロックの減少を調べたものである。従来方式は使用レジスタ数が最小となるように割り付けを行う。回避可能なインタロック数とは本方式の適用によるものである。これらのインタロックの回避にはたいていの場合レジスタ数を1つ増やすだけでよい。その他の回避できないインタロックは基本ブロックが小さいことなどが原因である。本方式により、従来方式に比べてSPECベンチマークのespressoで約3％、liで約1％実行性能を向上できた。

表1　インタロック数の状況

関数名	静的命令数	従来方式でのインタロック数	回避可能なインタロック数
massive_count	293	46	6
essen_parts	183	23	13
elim_lowering	133	22	7

5．おわりに

本稿では1サイクルのロード・ユースインタロックを対象としてレジスタ割り付け方式を説明したが、潜在遅延数の設定によって様々なインタロックに対応するよう、容易に拡張可能である。また大域レジスタ割り付けされた変数がある場合にも、本方式を適用できる。

本稿では基本ブロック単位の命令スケジューリングの効果を上げることを目的として、基本ブロックを処理単位とする局所割り付け方式を提案した。今後は基本ブロックを越えた大域スケジューリングや、並列実行可能なアーキテクチャ向けのスケジューリングと、レジスタ割り付けの組み合わせについて検討が必要である。

参考文献

[1] J. R. Goodman and W.-C. Hsu：Code Scheduling and Register Allocation in Large Basic Blocks. ICS '88
[2] T. A. Proebsting and C. N. Fischer：Linear-time, Optimal Code Scheduling for Delayed-Load Architectures. PLDI '91