ＶＬＩＷコンパイラにおけるスピルコード最適化

1J－10

遠藤 浩太郎，境 隆二
（株）東芝　情報・通信システム技術研究所

はじめに

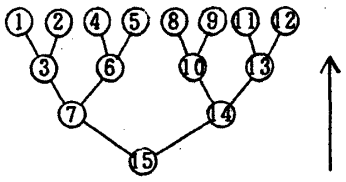
ＶＬＩＷ計算機では、コンパイラの最適化処理において、命令をスケジュールして並列度の高いオブジェクトコードを生成し、プロセッサの性能を引き出している。命令を効果的にスケジュールするためには、変数をできるだけ仮想レジスタに割付ける必要がある。仮想レジスタは最終的に物理レジスタに割り当てる操作が必要であるが、このとき物理レジスタ数には制限があるので、割り当てられない仮想レジスタはメモリ上に配置される。これに伴う命令をスピルコードと呼ぶ。

仮想レジスタを物理レジスタに割り当てるレジスタ割当を命令スケジュール前に行うか、スケジュール後に行うかは、オブジェクトコードの並列度とレジスタスピルの発生度に大きく影響する。スケジュール前にレジスタ割当を行うと、データの逆依存（レジスタ定義の前にあるレジスタ参照への依存）の原因となり、命令の並列性が十分に得られない。逆に命令スケジュールの後に行うと、命令のスケジューリングによって物理レジスタが不足する現象が起きやすくなり、その場合、スピルコードが実行サイクル数を増大させてしまう。

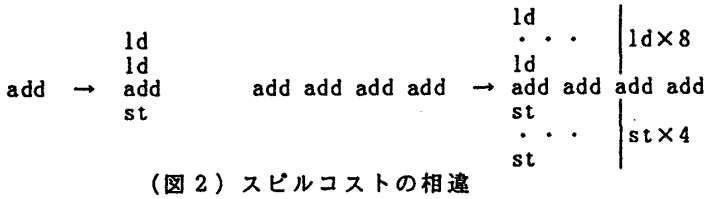
このように実効命令サイクル数を最小にし、並列度を引き上げようとする命令並列化スケジューリングと、使用するレジスタ数を最小にすることは相反する関係にある。ＶＬＩＷ計算機ではコンパイラでの命令並列化が必須であること、スピルコードが必要になる場合は稀であることから、我々は並列化スケジューリングを優先し、スケジューリング後にレジスタ割当を行う方式を選択した。本稿では、並列化された命令列に対するスピルコード生成戦略とその最適化について述べる。

命令並列化と物理レジスタ使用数のトレードオフ

コンパイラは命令並列化において、命令の依存関係をあらわすＤＡＧを作成する。このとき、ＤＡＧの高さと命令数の比によって、スケジュール戦略を切り換える。具体的には、ターゲットマシンの並列度とレジスタ数によって決定されるしきい値を設定し、”命令数／ＤＡＧの高さ＜しきい値”のときは、実行パスが最小になるようにスケジュールするが、この比がしきい値を越える場合は、ＤＡＧを深さ優先探索し、その帰りがけ順に命令をスケジュールする方法をとる（図１）。



（図１）ＤＡＧの深さ優先探索



（図２）スピルコストの相違

スピルコストの相違

ＶＬＩＷ計算機では、同時にできるメモリアクセスは１ないし２に制限され、相対的にメモリアクセスのコストが高くなる。同じ理由で、スピルコードもコストが高くなる。例えば、並列度４で同時にロード、ストア命令が１つずつ指定できるＶＬＩＷ計算機と、通常のシーケンシャルなＲＩＳＣ計算機を比較すると、すべてのレジスタがスピルした最悪の場合には、図２に示すように、通常のＲＩＳＣの場合には３倍のコストがかかるのに対して、ＶＬＩＷでは１２倍のコストがかかる。

A Spill Code Optimization in VLIW Compiler
Kotaro Endo, Ryuji Sakai
Information & Comunication Systems Laboratory, TOSHIBA Corporation

命令並列化後のスピルコード生成とその最適化

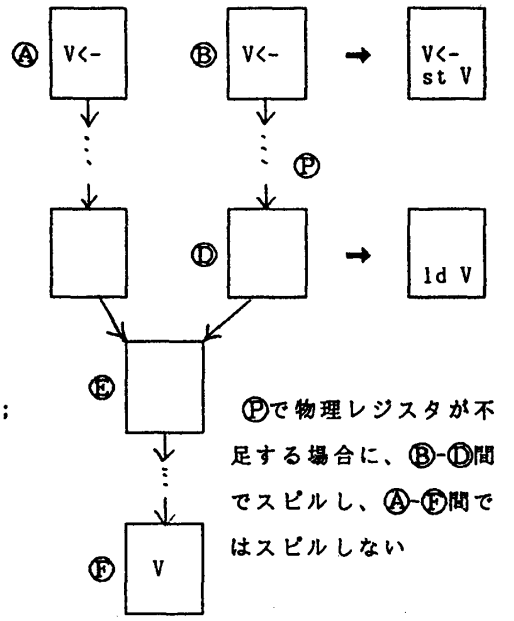
スピルコストの問題に対する局所的なアプローチを示す。レジスタ割当の結果、物理レジスタが不足した場合には、まずスピル用ワークレジスタとして８個の物理レジスタを確保する。次に各基本ブロックの命令列をスキャンし、スピルする仮想レジスタにスピル用ワークレジスタをラウンドロビンで割り当てていく。この方式の効果として、スピルした変数の定義・参照の直後に参照があった場合は、スピル領域からその変数をロードする必要はない。同様にスピル領域へのストアが省略可能である。またスピルコード間のデータ依存関係が緩和されて並列化しやすいスピルコードを生成できる利点もある。すなわち、スピルコードを命令列中にスケジュールすることでスピルコストを低減できる（図３）。

```
① fadd(f15,f14,f16);
② fadd(f8,v3,f9) fmul(f10,f11,v5);
③ fmul(v3,v7,f12);
   ・・・
  // v3,v5,v7がスピルしたとき

                ↓

   ・・・ add(r20,#32,r12)  add(r20,#24,r14);
① fld([r12],f0) fadd(f15,f14,f16) add(r20,#16,r13);
② fld([r13],f2) fadd(f8,f0,f9) fmul(f10,f11,f1);
③ fst(f1,[r14]) fmul(f0,f2,f12);
   ・・・
 // v7(f2)のﾛｰﾄﾞはv5(f1)のｽﾄｱを越えてｽｹｼﾞｭｰﾙされる
```

（図３）並列化後の局所的なスピル生成



（図４）並列化前の大域スピルコード生成

命令並列化前スピルコード生成方式

スピルコストの問題の別の解として、命令スケジュール後の物理レジスタ使用量を見積もり、仮想レジスタのスピルコードをスケジュール前に生成する方式について考察する。この方式でも、物理レジスタへの割当はスケジュール後に行うので、スケジュール後にも見積り不足分のスピルコードが生成される。この方式の目的は、スケジュール前のスピルコードを仮想レジスタによって記述することで、それを他の命令と同等に最適化することである。スピルコード以外の命令と混在してスケジュールすることで、スピルコストを抑えることができる。またスピルコードの大域最適化（図４）が容易になる、遅延分岐スロットでのスピルコード生成などの煩わしさがないといった利点もある。

おわりに

ＶＬＩＷ計算機における命令並列スケジュールとスピルコードの問題点を明らかにし、その解決方法を示した。命令スケジュール後に生成されるスピルコードを最適化することで、物理レジスタ不足時の性能劣化を最小限に抑えることができた。加えてスケジュール前のスピルコード生成を提案した。これについては現在評価中である。ここで示した方式は、スーパスカラやスーパパイプラインといった並列ＲＩＳＣアーキテクチャの最適化コンパイラにおいても適用可能である。

参考文献

Priyadarshan Kolte and Mary Jean Harrold.
"Load/store range analysis for global register allocation"
In Proceedeings of the SIGPLAN '93 Conference on PLDI, pp268-277,1993