1J－9

RISC命令セットプログラムのVLIW命令セットプログラムへの変換*

柴田 礼†
飯塚 肇‡
成蹊大学工学研究科情報処理専攻§

1 はじめに

計算機の性能に対する要求は日々高まるばかりであり、それに応えるべく、計算機の技術は多様な進歩を続けている。RISCプロセッサはパイプライン化によって、1サイクルあたり1命令に近い性能を実現し、さらにスーパースケーラ化によって、1サイクルあたり2～3命令を実現したが、動的な並列性の検出は、ハードウェアを複雑にし、クロックサイクルの低下、コストの増大、開発サイクルの遅れなどの原因となりうる。それに対しVLIW(Very Long Instuction Word)は、比較的シンプルなハードウェアで、高い並列性を実現できるが、従来のアーキテクチャとのバイナリ互換がないことが、大きな問題とされている。そこでRISC命令セットプログラムをVLIW命令セットプログラムに変換するトランスレータを設計し、その効果を検証したものである。

2 VLIWとガード機構 [2]

VLIWは、一つの命令中に複数の小命令を持つもので、複数の演算処理を同時に行える。通常は基本ブロック中の命令をいくつか一つのVLIW命令に詰め込むが、条件分岐が多く存在する場合などには、命令を十分に詰め込むことが出来ずに、命令中に大量のNOPを持つことになる。そのため、違う条件に依存する小命令を一つにまとめるために、小命令ごとに実行条件(ガード)を付け、条件によって実行をキャンセルするものがガード機構である。また、すでに提案されたガード機構は、小命令ごとに複数のコンディションレジスタとその状態を指定することが出来るが、これによって、命令を表現するのに必要なビット数が大きくなっている。本研究では、小命令ごとに、一つのコンディションレジスタとその状態を指定するものとした。さらに、それぞれのコンディションレジスタに真と偽のビットを別に持たせ、コンディションセット命令が実行されると、その演算結果によって、真または偽のビットが1になる。コンディションレジスタをセットする命令そのものがキャンセルされた場合、コンディションレジスタは真と偽が0となり、そのコンディションレジスタに依存する命令は、すべて実行されなくなる。それによってツリー状の分岐のサブツリー部分をまとめてキャンセルすることが出来るので、小命令ごとに複数のコンディションレジスタを持たなくても、ツリー状の分岐を効率よく処理できる。

3 変換の概略

今回設計したトランスレータは、DLX[1]用gccによって生成されたコードを、ガード付きのDLX命令を小命令として持つVLIW命令に変換する。アセンブリシンボリックコード間での変換を行うが、変換手法としては、バイナリ間の変換、あるいはコンパイルの一部でも、同じ手法が用いることが出来る。DLXは32個のレジスタを持つが、本研究では、レジスタのリネーミングを可能にするために、レジスタの数を32個以上に設定できる

*Transration from RISC instruction set program to VLIW instruction set program

†Tadashi Shibata
‡Hajime Iizuka
§Seikei University 3-3-1 Kichijoji-kitamachi, Musashino, Tokyo 180, Japan

ようにした。

また、実用的な変換は、変換されたコードの実行効率が高くなければならない。言い換えれば、最適化を含む変換が要求される。具体的な変換手法は以下に述べるが、プロファイラの必要性が高いと思われるループアンローリング等は、用いなかった。しかし、プロファイラを用いた変換と組み合わせた場合の相性を確かめるため、ソースレベルでループアンローリングを施しコンパイルしたコードも、測定対象に加えた。

4 レジスタのリダイレクトを用いた命令の移動

逆依存をさけて命令を移動させるために、レジスタのリダイレクトを行う。例えば、

r1 = r2 + r3; r2 = r3 + r4;
→ r5 = r3 + r4; r1 = r2 + r3; r2 = r5;

という置き換えることが出来る。また、リダイレクトのために生じた命令 r2 = r5; の後に r2 への書き込みがあるならば、r2 = r5; を後に実行されるように移動させていけば、不要な命令となり、以下のような完全なレジスタのリネーミングとなる。

```
r1 = r2 + r3;          r5 = r3 + r4;
r2 = r3 + r4;          r1 = r2 + r3;
r4 = r2 + r4;    →     r2 = r5;
r2 = r1 + r3;          r4 = r2 + r4;
                       r2 = r1 + r3;

r5 = r3 + r4;          r5 = r3 + r4;
r1 = r2 + r3;          r1 = r2 + r3;
r4 = r5 + r4;    →     r4 = r5 + r4;
r2 = r5;               r2 = r1 + r3;
r2 = r1 + r3;
```

しかし、VLIW のスケジューリングにおいては、ある程度の NOP が生じることはやむを得ず、そのような NOP の代わりにリダイレクト命令を置くことで、レジスタの浪費を抑えることが出来る。

5 測定結果

いくつかのプログラムを、C 言語で記述し、作成したトランスレータにかけた。コンパイラは DLX 用 gcc で、最適化オプション-O を付けてコンパイルし、初期化部分などを除いたループ部分のみ計測した。例を挙げると

配列中の最大値を求めるプログラム:

DLX:16 命令 → VLIW:18 命令→ 最適化された VLIW:13 命令 (2 命令同時実行) → 最適化された VLIW:12 命令 (4 命令同時実行)

4 回分ループアンローリングされた配列中の最大値を求めるプログラム:

DLX:46 命令 → VLIW:51 命令→ 最適化された VLIW:33 命令 (2 命令同時実行) → 最適化された VLIW:22 命令 (4 命令同時実行)

特に、ループアンローリングとの組み合わせ時に、NOP の割合が少なくなっている。それに対し、ある値をリストから探し出すプログラムなどは、効率が悪く、ループアンローリングと組み合わせた場合でも、あまり改善しなかった。これは、線形リスト上の探索が、配列の処理よりも、長いクリティカルパスを持つためと思われる。特に、リスト上の処理が、値の探索のような単純なものの場合、クリティカルパスがそのまま処理の長さを決定することが多く、並列化のメリットが現れない。

6 結論

今回作成したのはトランスレータのみであり、また、仮にシミュレータなどを用いても、RISC と VLIW のクロックサイクル等はインプリメンテーション依存なので実行速度に関する議論をするのは難しい。しかし、VLIW において NOP を減らすことは、実行速度を改善する効果が期待され、実用的自動変換の可能性を示せたと思われる。

参考文献

[1] John L. Hennessy, David A. Patterson: *Computer Architecture A Quantitative Approach*, Morgan Kaufmann Publishers (1990).

[2] 小松, 古関, 鈴木, 深沢: 拡張 *VLIW* プロセッサ *GIFT* における命令レベル並列処理機構, 情報処理学会論文誌, 34, 12, pp.2599-2611 (1993).