多重レジスタリネーミング方式

4G－6

安里 彰　細井 聡　新井 正樹
富士通研究所

1 はじめに

プロセサの高速化手法の一つであるレジスタリネーミング[1]をハードウェアでインプリメントする場合、R_{arc} 個のアーキテクチャレジスタに対し、それより多い R_{ren} 個のレジスタを用意して、それらをマッピングするのが普通である。

一方、コンパイラがプロセサの性能を充分に引き出すようなコードを生成するためには、レジスタの個数が R_{arc} 個では不足に感じることが多い。特に、スーパスカラやVLIWプロセサ等の実行並列度を向上させるためのコードスケジューリングを行なう場合この傾向は顕著になる。

我々は、レジスタリネーミングのために用意された余剰な $R_{ren} - R_{arc}$ 個のレジスタを有効に活用して、コンパイラが使用できるレジスタ数を実質的に増やす方式である、多重レジスタリネーミング方式を本論文で提案し、その効果について議論する。

2 多重レジスタリネーミングの考え方

今、番号 r のレジスタが r' にリネーミングされているとする。この状態で新たに r への代入が起こると、普通のリネーミング方式では新規の番号 r'' を r に割り当て、r' はもはや使用しない。これに対して、r' を無効にせずに、r' に代入された値と r'' に代入された値の双方を、アーキテクチャレジスタ r の値として参照できるようにするのが多重レジスタリネーミングの考え方である。

多重レジスタリネーミングを行なうに当たっては、

- 何を契機に多重リネーミングを行なうのか。
- どう区別して参照するか。
- 何を契機に多重リネーミングを解消するか。

の3点を明確にする必要がある。一例として、1点目については同一レジスタへの書き込みが連続したことを契機とし、2点目にについては参照の順番で区別することにし、3点目については多重にリネーミングされているレジスタへの新たな書き込み命令が現れた時点で解消することにした場合の簡単なプログラム例を図1に示す。

```
命令1   load data1 -> r1
命令2   load data2 -> r1     /* r1を多重リネーミング */
命令3   r1 + 4     -> r2     /* 命令1の結果参照 */
命令4   r1 and 0xff -> r3    /* 命令2の結果参照 */
命令5   load data3 -> r1     /* 多重リネーミング解消 */
命令6   r1 + 4     -> r4     /* 命令5の結果参照 */
命令7   r1 and 0xff -> r5    /* 命令5の結果参照 */
```

図1　プログラム例

3 インプリメント

多重レジスタリネーミングのインプリメントを図2に示す。デコーダによって命令コードから抽出されたデスティネーションレジスタ番号 rd に、フリーリストから出力された番号が rd' として割り当てられる。rd' は、rd が単一リネーミング状態ならリネームテーブル1の rd でアドレスされるエントリに格納され、多重リネーミング状態ならそれ以外のいずれかのリネームテーブルに格納される。

各リネームテーブルはリネーミング前後のレジスタ番号の対応を保持するもので、R_{arc} 個のエントリを持ち、各エントリは R_{ren} を表現できる $\log_2 R_{ren}$ ビットの領域である。

また、ソースレジスタ番号 rs でアドレスされる各リネームテーブルの内容のうち一つが選択されて rs' として読み出される。選択にあたっては、前章で述べた仕様に従ってセレクト信号を生成すればよい。

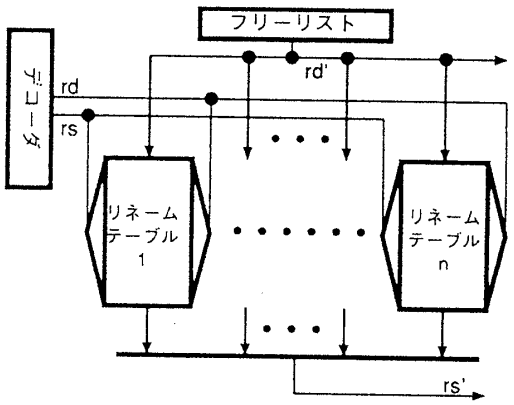

図2　多重リネーミングのインプリメント

Multiple Register-Renaming Method
Akira Asato, Akira Hosoi, Masaki Arai
FUJITSU Laboratories

```
        sub  %l0,%o2,%l0
L246:   sra  %l0,31,%o0
        and  %o0,%i3,%o0
        add  %l0,%o0,%l0
        sll  %l0,2,%l0
        ld   [%o0+%l4],%o0
        cmp  %o0,%l1
        be   L238
        cmp  %o0,0
        ble  L183
        nop
        sub  %l0,%o2,%l0
        sra  %l0,31,%o0
        and  %l0,%o2,%o0
        add  %l0,%o0,%l0
        sll  %l0,2,%o0
        ld   [%o0+%l4],%o0
        cmp  %o0,%l1
        be   L238
        cmp  %o0,0
        bg,a L246
        sub  %l0,%o2,%l0
L183:
```

P1

```
        sub  %l0,%o2,%l0
L246:   sra  %l0,31,%o0
        and  %o0,%i3,%o0
        add  %l0,%o0,%l0
        sll  %l0,2,%l0
        ld   [%o0+%l4],%o0
(1)     sub  %l0,%o2,%i5
        cmp  %o0,%l1
        be   L238
        cmp  %o0,0
(2)     sra  %i5,31,%o0
        ble  L183
        and  %l0,%o2,%o0
        add  %i5,%o0,%l0
        sll  %l0,2,%o0
        ld   [%o0+%l4],%o0
        cmp  %o0,%l1
        be   L238
        cmp  %o0,0
        bg,a L246
        sub  %l0,%o2,%l0
L183:
```

P2

```
        sub  %l0,%o2,%l0
L246:   sra  %l0,31,%o0
        and  %o0,%i3,%o0
        add  %l0,%o0,%l0
        sll  %l0,2,%l0
        sub  %l0,%o2,%i5
(3)     ld   [%o0+%l4],%o0
(4)     sra  %i5,31,%o0
(5)     cmp  %o0,%l1
(6)     mov  %o0,%g0
        be   L238
(7)     cmp  %o0,0
(8)     and  %l0,%o2,%o0
        ble  L183
        nop
        add  %i5,%o0,%l0
        sll  %l0,2,%o0
        ld   [%o0+%l4],%o0
        cmp  %o0,%l1
        be   L238
        cmp  %o0,0
        bg,a L246
        sub  %l0,%o2,%l0
L183:
```

P3

図 3　適応例

表 1　性能向上

P1	1.00
P2	1.00
P3	1.03

4　命令スケジューリングへの適用

多重レジスタリネーミングの効果の一つに、命令スケジューリングに適応して、使用するレジスタを増やさずに命令レベルの並列度を上げることが考えられる。

4.1　適用例

例として、SPECint92/026.compress の一部 (図 3 の P1、SPARC のアセンブラで記述) を用いる。この部分は、静的な命令数のうちの約 1%、動的な命令数のうちの約 25% を占める。このループ内で定義されるレジスタのうち、l0 はループ脱出後も参照されるが、o0 は使われない。P1 のままでは、多重レジスタリネーミングを用いても全く命令を移動することができないので、まず余っているレジスタ i5 を用いてスケジューリングを行ない、P2 のようにする。

次に、P2 に対して 2 章の方式による多重レジスタリネーミングを適用してスケジューリングを行ない、P3 を得る。P3 では、命令 (3)(4) が o0 の多重定義、(5)(7) が (3) の値の参照で、(6)(8) が (4) の値の参照である。値を交互に参照する制約から、(6) のような dummy 命令が必要になる。(8) は同時に多重定義の解消も兼ねている。

4.2　シミュレーション

多重レジスタリネーミングの効果を調べるために、compress および、compress の P1 の部分だけをマニュアルで P2 あるいは P3 のように変更したプログラムを用いて、実行トレースベースによるシミュレーションを行なった。ターゲットアーキテクチャは、4 命令 issue のスーパスカラを想定した。

P1 に対する性能向上を表 1 に示す。多重レジスタリネーミングにより、3% の性能向上が見られた。

5　今後の課題

5.1　その他の効果について

前章で述べた以外の効果として次のようなものが予想できる。

- spill された変数をレジスタに割り当てることができるようになる。
- 基本ブロック内だけで使用するレジスタの数を減らして、global な変数へ割り当てるレジスタ数を増やすことができる。
- 関数全体で使用するレジスタの数が少なくなり、leaf の最適化が適用できるようになる。

これらについての検討は、今後の課題である。

5.2　コンパイラへの組み込みについて

最も問題となるのは、多重レジスタリネーミングが適用できるパターンをどのようにして見つけるかということである。図 1 のように、基本ブロック内だけで見つけることは比較的容易であるが、適用できる場合が少ないであろう。一方、図 3 のように複数の基本ブロック間に跨って適用できるパターンを見つけることは一般には難しい。

1 案として、カラーリングによるレジスタ割り当ての際、従来はライフタイムが重なる変数は別のレジスタに割り当てていたが、逆にライフタイムが重なる変数を同じレジスタに割り当てて、多重レジスタリネーミングの候補とする方法が考えられる。これについても、さらなる検討が必要である。

参考文献

[1] M. Johnson, Superscalar Microprocessor Design, Prentice Hall, 1991.