

6G-8

ロード命令の先行実行とその評価

小沢年弘[†] 西崎慎一郎^{††} 木村康則[†][†] 株) 富士通研究所 ^{††} 株) 富士通ソーシャルサイエンスラボラトリ

1 はじめに

近年、プロセッサの実行速度向上に比べて、メモリ系の速度向上が低い傾向が続いている。このため、実行時間に占めるメモリ系のオーバーヘッドの割合が増加しており、高速実行のためにメモリ系のオーバーヘッドを減らすことが、より重要になっている [1]。

メモリ系のオーバーヘッドを減少させる方法の一つとして、ロード命令の先行実行がある [2]。この方法では、ハードウェアにロード命令の完了を待たずに他の命令の実行ができる non-blocking load を組み込む。そして、ロード命令をそのデータを使用する命令よりも十分早く発行するように命令コードをスケジューリングする。これにより、たとえキャッシュ・ミスが起きてもデータがロードされるまでの間、他の命令の実行が行なわれ、メモリ系のオーバーヘッドを隠すことができる。

この方法においては、ロード命令をそのデータを使用する命令（データ使用命令と呼ぶ。）よりどのくらい早くスケジューリングできるか、つまり、これら2つの命令の距離をどのくらい離すことができるかで、効果が決定される。

我々は、プログラムの意味を変えない範囲で、ロード命令とデータ使用命令とを指定の距離だけ離すように命令スケジューリングを行なうコンパイラを作成した。

このコンパイラにより、整数系プログラムにおけるロード命令とデータ使用命令との距離を測定し、この方法のメモリ系オーバーヘッド削減の効果を評価した。

2 命令スケジューリング

我々の命令スケジューラは、GNU C コンパイラに組み込んで使用され、レジスタ割り付け前と後で呼び出される。

2.1 命令スケジューラの構成

命令スケジューラは、意味的にベシック・ブロック内のスケジューリングと、どのベシック・ブロックのスケジューリングを行なうかという順番を制御する部分に分けられる。

ベシック・ブロック内スケジューリングでは、ベシック・ブロックの最後の命令から逆順にリスト・スケジューリングを行なう。この時、データ使用命令から十分離れていないロード命令は、たとえその後続命令がすべてスケジューリングされても、スケジューリングの候補に入れないでスケジューリングを続けて行く。最後まで十分離せなかったロード命令とそれが（再帰的に）依存している命令は、先行ベシック・ブロックにコピーされて、先行ベシック・ブロックでスケジューリングされる。この処理を行なうために、これらの命令は、ブロック間移動リストというリストに登録される。

ただし、十分離せなかったロード命令でも、それを移動することによってプログラムの意味が変わってしまう場合には、ブロック間移動リストに登録することはしない。先行ベシック・ブロックに移動できる命令の条件は、次節で述べる。

ベシック・ブロック内のスケジューリングは、すべての後続ベシック・ブロックのスケジューリングが完了すると起動される。このとき、後続ベシック・ブロックのブロック間移動リストに登録されている命令は、このベシック・ブロックにコピーされる。

ただし、ベシック・ブロックの接続関係がループ状で、あるベシック・ブロックがバックワード・エッジを持つ場合には、バックワード・エッジにつながっている後続ベシック・ブロックのスケジューリングの完了を待たずに、そのスケジューリングを行なう。そして、完了を待たなかったベシック・ブロックのスケジューリングが完了すると、再び、もとのベシック・ブロックのスケジューリングを行なう。その結果、ブロック間移動リストに命令が新たに登録されたならば、その先行ベシック・ブロックのスケジューリングも再度行なわれる。この再スケジューリングは、ブロック間移動リストに新たに登録される命令がなくなるまでより先行するベシック・ブロックで行なわれる。

2.2 先行ベシック・ブロックへの移動条件

ある命令を先行ベシック・ブロックに移動しても意味が変わらないための条件は、想定するハードウェアに依存する。ここでは、一般の RISC タイプ CPU に、次の機能を付加した2種類のハードウェアを想定した。

Preload and its Evaluation

T. Ozawa[†] S. Nishizaki^{††} Y. Kimura[†][†]FUJITSU LABORATORIES LTD.^{††}Fujitsu Social Science Laboratory Ltd.

1015, Kamikodanaka Nakahara-ku, Kawasaki 211, Japan

表 1: ロード命令とデータ使用命令との平均静的距離

program	GNU C	TRAP	NON-TRAP
espresso	2.7	4.0	8.2
li	1.9	3.5	5.3
eqntott	2.1	4.4	12.0
compress	2.1	8.2	11.7
sc	2.1	5.6	7.4
gcc	2.2	6.1	14.6

- non-blocking load のみの付加を想定する。以下、TRAP と呼ぶ。
- non-blocking load と、trap を無視する機能の付加を想定する。以下、NON-TRAP と呼ぶ。

両者に共通する移動可能のための条件は、以下のものである。

- 移動することによって、他のベーシック・ブロック中で生きているレジスタの内容を変更しない。
- 多方向分岐する先行ベーシック・ブロックに移動する場合は、移動する命令中にサブルーチン・コール、ストア命令がない。

TRAP の場合には、さらに次の条件が必要になる。

- 多方向分岐する先行ベーシック・ブロックに移動する場合は、移動するロード命令が、illegal memory access を起こさない。この条件を満たすロード命令で、コンパイル時に判定できるものには、次のようなものがある。
 - そのロード・アドレスがすべての分岐先で参照されている。
 - frame pointer からのオフセットで参照されている。
 - 大域の変数へのアクセスである。

また、ループの外から中に命令を移動した場合には、実行命令数が大幅に増加し、実行速度が低下してしまう可能性が高いので、ループの中への命令の移動は行わないようにしている。

3 ロード命令先行実行の評価

以上のようなスケジューリングによって、ロード命令とそのデータを使用する命令との距離をどのくらい離すことができるのかについて、TRAP, NON-TRAP の 2 つの場合について測定した。測定には、SPECint92 ベンチマーク・プログラムを用いた。また、ベースとなるマシン・アーキテクチャは SPARC である。

表 2: TRAP において距離を広げられない理由

program	first	loop	reg	call	store	trap
espresso	10.9	12.2	1.6	29.2	7.5	33.6
li	24.0	7.0	7.4	26.8	3.2	28.1
eqntott	9.8	11.2	6.9	18.6	7.1	33.7
compress	13.9	9.0	9.4	23.3	11.3	23.7
sc	7.8	10.9	12.6	20.2	4.6	18.1
gcc	10.2	3.5	10.4	17.9	5.8	39.2

GNU C コンパイラ SPARC 用、NON-TRAP 用、TRAP 用、それぞれの場合におけるロード命令とそのデータを使用する命令との平均静的距離を、表 1 に示す。NON-TRAP、TRAP においては、距離を 2 0 まで広げようとしてコンパイルした。静的距離は、コンパイラの出力コード上での距離であり、ロード命令が発行されてからのサイクル数である。例えば、マルチサイクル命令があると、それ一つで、そのサイクル数だけの距離になる。

TRAP において距離を広げられない原因の割合を表 2 に示す。first とは、ロード命令が関数の先頭になってそれ以上移動できないため、loop とは、移動するとループの中に入ってしまうため、reg とは、他のベーシック・ブロック中のレジスタを破壊してしまうため、call とは、サブルーチン・コール命令を移動しなければならないため、store とは、ストア命令を移動しなければならないため、trap とは、illegal memory access を引き起こす可能性があるため移動できないことを示す。

TRAP の場合で数サイクル、NON-TRAP の場合で 1 0 サイクル程度、距離を離すことができた。これにより、キャッシュ・ミスはある程度隠すことが期待できる。さらに距離を広げるためには、関数間解析や memory disambiguation の機能強化、ロード・アドレスの予測の機能が重要であることが分かる。

4 今後の課題

実行時間をシミュレータで測定する。また、実行命令数の増加を防ぐためには、キャッシュ・ミスを引き起こす可能性の高いロード命令を識別し、それらだけを早く実行するようにスケジューリングを行なう必要がある。

参考文献

- [1] W. Y. Chen, S. A. Mahlke, P.P. Chang, and W.W. Hwu. Data access microarchitectures for superscalar processors with compiler-assisted data prefetching. In Proceedings of Microcomputing 24, 1991.
- [2] T. C. Mewry, and M. S. Lam. Design and Evaluation of a Compiler Algorithm for Prefetching. In Proceedings of 5th ASPLOS, 1991.