整数系プログラムへの広域命令スケジューリングの適用

6G－6

新井正樹　細井聡　木村康則
（株）富士通研究所

1　はじめに

近年、マイクロプロセッサにおいて命令レベルの並列処理を強化することで高速化を目指したアーキテクチャであるスーパースカラ方式や VLIW 方式が注目を集めている。これらのアーキテクチャをもつプロセッサの性能を最大限に引き出すためには、コンパイラによって命令レベルの並列度を抽出することが不可欠である。命令レベルの並列度を抽出する手法に広域命令スケジューリングがある。広域命令スケジューリングに関する研究は今まで主に浮動小数点系プログラムについて行なわれてきた。我々は整数系プログラムに対して広域命令スケジューリング手法が有効であるかを検証するために、その手法の一つであるパーコレーションスケジューリング [1, 2] を整数系プログラムに適用し、その静的な評価を行なった。

2　パーコレーションスケジューリング（PS）

整数系プログラムでは一般に基本ブロックの長さが小さい。したがって、整数系プログラムに対して命令レベルの並列性を最大限に引き出すためには、基本ブロック内で命令を入れ換える局所命令スケジューリングだけでなく、基本ブロックを越えて命令を入れ換える広域命令スケジューリングを行なう必要がある。広域命令スケジューリング手法の一つにパーコレーションスケジューリング（PS）がある。PS は、制御フローグラフの下流から上流に向けて命令を移動することで、上流の基本ブロックの命令レベルの並列度をあげる手法である。その核はコア変換と呼ばれる 5 つの基本的なプログラム変換の集合からなる。

我々は PS を GNU C コンパイラに実装した。コンパイラのターゲットマシンは SPARC アーキテクチャのスーパースカラプロセッサである。PS とレジスタ割り当ての関係を調べるために、

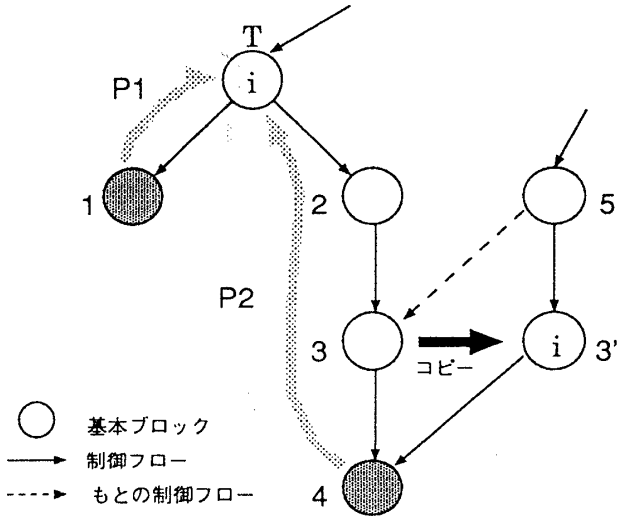

図 1: 命令移動の例

PER1 レジスタ割り当て前にスケジューリング

PER2 レジスタ割り当て後にスケジューリング

の二通りの方法でスケジューリングを行なえるようにコンパイラを作成した。スケジューリング時には、命令を依存の許す限り制御フローグラフの上流に移動し、その過程で作られる命令のコピーの移動も行なう。局所命令スケジューラは GCC のものを使用した。投機的命令移動は分岐確率に偏りが少ない場合には性能の低下を招く可能性がある。そこで作成したコンパイラでは非投機的命令移動のみを行うこととした。

3　静的な評価

作成したコンパイラを使って行なった、整数系ベンチマークプログラムに対するコンパイル時の調査について述べる。各基本ブロックに平均何個の命令を移動できるかを示す基本ブロックあたりの平均命令移動数と、どれだけ離れた基本ブロックから命令を移動できるかを示す平均命令移動距離について調査を行なった。PS とレジスタ割り当ての関係を調べるために、PER1 と PER2 の二通りの方法の比較を行なった。テスト用の整数系ベンチマークプログラムとして SPECint92 を使用した。

Applying Global Instruction Scheduling
to Non Numerical Applications.
Masaki Arai, Akira Hosoi, Yasunori Kimura
FUJITSU LABORATORIES LTD.
1015, Kamikodanaka Nakahara-ku, Kawasaki 211, Japan

表 1: 基本ブロックあたりの平均命令移動数

program	PER1	PER2
008.espresso	0.74	0.81
022.li	0.44	0.32
023.eqntott	0.61	0.52
026.compress	1.0	0.82
072.sc	0.84	0.65
085.gcc	0.97	0.98

表 2: 平均命令移動距離

program	PER1	PER2
008.espresso	1.63	1.23
022.li	1.72	1.13
023.eqntott	1.74	1.21
026.compress	2.10	1.24
072.sc	2.01	1.36
085.gcc	1.50	1.73

3.1 基本ブロックあたりの平均命令移動数

基本ブロックあたりの平均命令移動数を次のように定義する。

$$\text{基本ブロックあたりの平均命令移動数} = \frac{\text{移動した命令の総数}}{\text{スケジューリング後の基本ブロック数}}$$

命令移動の例を図 1に示す。図 1は基本ブロック 1 と 4 にあった命令 i を基本ブロック T へ移動した結果を示す。このとき基本ブロック T へ一つの命令が移動されたと計算する。図 1では、命令を移動したことで、もとの制御フロー 5 → 3 → 4 の意味が変わることを防ぐために基本ブロック 3 のコピー 3' を作り、そこに命令 i を移動してプログラムの意味保存を行なっている。各ベンチマークプログラムに対する基本ブロックあたりの平均命令移動数を表 1 に示す。表 1 では、図 1の基本ブロック 3' に移動した命令のような、命令移動時にプログラムの意味保存のために移動した命令は数えていない。表 1 から各基本ブロックに移動できる命令数は 1 個以下であるということがわかる。

3.2 平均命令移動距離

平均命令移動距離を次のように定義する。

$$\text{平均命令移動距離} = \frac{\sum \text{命令の移動距離}}{\text{移動した命令の総数}}$$

$$\text{命令の移動距離} = max\{\text{命令が移動する時に通ったパスの長さ}\}$$

例えば、図 1では命令 i の移動距離は命令が移動する時に通ったパス P1 と P2 のパスの長さの最大値 3 である。各ベンチマークプログラムに対する平均命令移動距離を表 2 に示す。表 1 と同様に表 2 では、命令移動時にプログラムの意味保存のために移動した命令は考慮していない。表 2 から平均命令移動距離は 2 以下であるということがわかる。

4 結論

コンパイル時の静的な調査から、レジスタ割り当て後にスケジューリングをするよりもレジスタ割り当て前にスケジューリングをする方がより多くの命令をより遠くへ移動できるということがわかった。また、基本ブロックあたりの平均命令移動数は少なく、命令の平均移動距離も小さいということがわかった。このことから、整数系プログラムに対しては投機的な命令移動をしない **PS** による性能の向上は期待できないと考えられる。性能の向上のためには、より多くの命令を移動することが必要であり、そのためには投機的な命令移動を行なう必要がある。

日頃ご指導いただく林部門長、津田部長、服部主管研究員に深く感謝致します。

参考文献

[1] Alexander Aiken and Alexandru Nicolau. *A Development Environment for Horizontal Microcode*, IEEE Transactions on Software Engineering. Vol.14 No.5. May 1988.

[2] Steven Novack and Alexandru Nicolau. *Trailblazing: A Hierarchical Approach to Percolation Scheduling*, Department of Information and Computer Science, University of California, Irvine, CA 92717, Technical Report Number 92-56.