2D-03　予測実行サイクル数に基づいた大域的命令スケジューリング

服部 直也,　荒木 拓也,　坂井 修一,　田中 英彦
{hato,araki,sakai,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学 工学部 *

1　はじめに

近年のプロセッサはスーパースカラやVLIWなどの技術により、複数の命令を同時に実行できる能力を持つ。この能力を生かすために、コンパイラはコード中から並列に実行可能な命令を探し、並列に実行できる順に並べる。この操作をスケジューリングと呼ぶ。スケジューリングはベーシックブロック内の局所スケジューリングと、ベーシックブロックを越えた命令移動を行う大域的スケジューリングに大別される。非数値計算系のプログラムは、ベーシックブロック内の並列性に乏しいため、大域的スケジューリングが重要である。

命令移動には、移動先の機能ユニットと(レジスタリネーミングをする場合)レジスタを消費するというコストがある。そのため、無意味な命令移動は避けなければならない。

従来の研究では、機能ユニットが十分に用意された仮想的なマシンを仮定しているため、投機移動のコストへの配慮が不足している。

本手法では予測実行サイクル数を見積もり、サイクル数が減少する場合に限り、命令移動を行う。

2　関連研究

大域的スケジューリングとしては、古典的なものとしてトレーススケジューリングとパーコレーションスケジューリングが知られている。

トレーススケジューリングは、各分岐について分岐確率の高い側を選んで1つの流れ(トレース)を形成し、このトレース内の命令が最適に並ぶようにスケジューリングする。実行確率が低い側は補償コードが入るため、一般的に遅くなる。そのため分岐確率に偏りが少ないアプリケーションの場合、トレース外実行の速度低下が問題となる。

パーコレーションスケジューリングは、隣り合うブロック間で基本的な4つの操作を繰り返し行い、命令を可能な限り上方へ移動する。命令が集まることによって並列実行性を高めることを狙うが、移動の際に価値判断が伴わないため、実行サイクル数が減少する保証はない。

近年のスケジューリングでは、サイクル数が増加しないことを保証している。小松らのスケジューリング[2]は、実行確率の高いトレースから順にクリティカルパスが縮む移動を選択する。しかし、クリティカルパスでサイクル数を見積もっているため、機能ユニットの数が十分になければ、この見積もりはよい指標ではない。

また、セレクティブスケジューリング[1]は、各サイクル毎に、空きスロットに移動できる命令を下位ブロック全ての中から探し出し、見つかった命令の中で最も実行確率の高い命令を移動する。しかし確率が低くとも、上流にある命令を移動すれば以降の移動を促すことを考慮していない。

3　本研究のスケジューリング方法

「並列実行命令数を増やすこと」は「サイクル数が減少すること」は必ずしも一致しない。図1を例に取る。命令移動によって並列実行命令数を増やしたとしても、移動元の命令がのbの位置にある命令だとすれば、サイクル数の減少は望めない。サイクル数減少の鍵となるのは、移動先ブロックの並列度の向上ではなく、移動元ブロックのサイクル数が減少するか否かである。本手法では実行頻度の高いブロックに注目し、そのブロック内の命令を上流ブロックに上げることで、サイクル数減少を狙う。以下にスケジューリングの手順を説明する。

スケジューリングの前段階として、ループを認識する。認識したループはハイアラキカルリダクションにより、複数レジスタを読み書きし複数の分岐先を持つブロックとして扱う(図2)。そのため関数全体をスケジューリング対象とすることができ、ループを跨いだ移動が可能である。また、各ブロックの実行サイクル数を得るために、全ブロックに局所スケジューリングを行っておく。

大域的命令移動は実行確率の高いトレースに沿って行う。これは命令の依存関係の流れとブロックの実行確率の両方を生かすためである。1つ目のトレースは、関数の入口から出口まで実行頻度の高いブロックを辿る。2つ目以降は、どのトレースにも所属しないブロックを優先して、実行頻度の高いブロックを辿る。例えば図3のような分岐確率をもつブロック群の場合、図中の数字の順にスケジューリングを行う。

ブロックに優先順位を付けたのと同じように、命令にも優先順位を付ける。移動により直接サイクル数が減少するのは、図1の 命令a のように下詰めに命令を配置

---

* "Global instruction scheduling based on estimated execution cycle"
Naoya Hattori, Takuya Araki, Shuichi Sakai, Hidehiko Tanaka
University of Tokyo, School of Engineering,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan

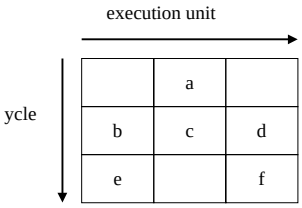

図 1: あるベーシックブロックの下詰め資源予約表

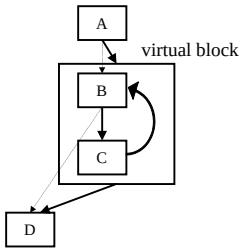

図 2: ハイアラキカルリダクション

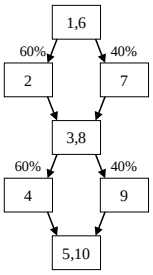

図 3: ブロックのスケジューリング順序

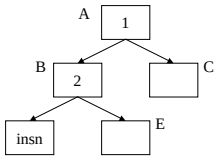

図 4: 移動先は最上流から試す

した場合に最上段に位置する命令である。このような命令を最優先する。直接サイクル数が減少しない命令であっても、移動によってスロットが空き、また早い時期に計算結果が得られるため下流のブロックからの命令移動を可能にすることがある。そのため、直接サイクル数が減少しない命令も移動対象とする。

優先順位が高い命令から順に、その命令がどこまで移動できるか限界を調べる。実際に移動する移動先はこの限界点から下方向に探す。例えば図 4のようなブロック群中で、命令 insn が A,B のどちらにも移動できる場合、A → B の順に移動を試していく。移動先を判断する際には、命令移動の前後のサイクル数見積りを用いる。見積もりには、補償コードによる影響も含める。移動前と比べてサイクル数が減少していれば、その移動は有効であるため移動先として認め、探索を終了する。サイクル数が増加した場合には、その移動は無効なので次のブロックへの移動を評価する。サイクル数が変化しなかった場合は下流からの移動を促すことを期待し、移動先として認める。サイクル数の見積もりは、プロファイルデータを基に

$\sum$(各ベーシックブロックの実行に必要なサイクル数 × 各ベーシックブロックの実行回数)

を用いる。そのため、サイクル数が増加したブロックがある場合でも、総合的にサイクル数が減少していれば移動を行う。

4 評価

評価は Sun Ultra 1(UltraSPARC I 166MHz を搭載) の上ユーザータイムを測定した。スケジューリングされるプログラムとしては SPECint95 の compress 、入力には bigtest.in を用いた。実装がポストスケジューラであるため、egcs ver 1.1 release でアセンブラ出力したソースをスケジューリングした。尚、アセンブルの際に、delay slot への最適化とスケジューリングのオプションは外している。

表 1: compress の測定結果

scheduling condition	user time (speed-up)	estimated cycle
no scheduling	335(1.000)	39,594M
local scheduling	326(1.028)	35,689M
percolation like without renaming	321(1.044)	34,377M
percolation like with renaming	313(1.070)	32,526M
this method	308(1.088)	32,289M

結果を表 1に示す。本手法との比較対象として、パーコレーションスケジューリングのように、サイクル数が増加する移動を許可するタイプのスケジューリングも行った。この比較により、数 % ではあるが本手法が無条件に移動を許可する方式より優れていることが示された。

5 まとめ

本論文では、大域的な命令移動に伴うコストを削減するために、見積もりサイクル数が減少する移動のみ行う手法を提案し、その効果を確認した。

今後は、他の SPECint95 のプログラムに対して、また別のプロセッサに対して評価を行っていきたい。

参考文献

[1] Soo mook moon and Kemal Ebicioğlu. Parallelizing nonnumerical code with selective scheduling and software pipelining. *ACM Transaction on Programming Languages and Systems*, Vol. 19, No. 6, pp. 853–898, november 1997.

[2] 小松秀昭, 古関聰, 深澤良彰. 命令レベルアーキテクチャのための大域的コードスケジューリング技法. 情報処理学会論文誌, Vol. 37, No. 6, pp. 1149–1161, 1996.