5Q-04

ループ本体のサイズを変えないソフトウェアパイプライン化の一手法について

竹原 功二　奥田 真也　大山口 通夫　太田 義勝
三重大学 工学部 情報工学科

1 はじめに

最新のコンピュータアーキテクチャにおいては、並列実行度の高い命令コードを生成するコンパイル手法が重要であり、その中でも非常に重要な手法としてソフトウェア・パイプライン化法 (SP 法) がある。

これまでの研究成果は、次の 2 つに大別することができる。まず、一つ目のアプローチは、得られるコードの実行サイクル数が最適となるスケジューリングを行うものである [1]。しかし、新しく得られるループ本体の命令数は元のループ本体の命令数と同じであるとは限らず、場合によっては指数関数的に大きくなってしまうことがあり、キャッシュ容量を越えるなどで効果が低減する場合がある。また新しいループの発見に多大な時間を要することもあるという問題点がある。

もう一つのアプローチは新しいループ本体の命令数の増加を許さないもの (モジュロースケジューリング [2]) である。しかし、ループ本体の命令数を変えずに最適なループ本体のスケジューリングを求めることは、NP 困難であることが知られており [1] 従って、発見的手法を用いたスケジューリング法がいくつか提案されてきた [2][3]。これらは多項式時間内では効率よくスケジューリングを完了するが、並列実行度を最大限引き出すことができるとは限らない。

URPR 法 [3] では、始めにループ本体をグリーディにスケジューリングし、次にループ運搬依存の始点と終点の差 +1 の最大のものを遅延幅とし、次のイタレーションをこの遅延幅ずつ下げていくことで新しいループを構成する。武市らの手法 [4] では、グリーディスケジューリングから始めて、ループ運搬依存の終点の命令の実行を遅らせることにより、より小さな遅延幅を求めるように [3] の手法を改善している。しかし [3] の手法より悪くなる場合がないという保証がなされていないなど、いくつかの問題点が残されていた。

そこで、本稿では [3] の手法を改善した新しいモジュロースケジューリング法を提案する。本手法は遅延幅を決定するループ運搬依存に対し、その終点を下げることでより小さい遅延幅を求め、並列実行度の高いループ本体を再構成する。これは [4] の制限を緩めており、アルゴリズムの有効範囲が拡張されている。また、ループ運搬依存間の関係を表す、ループ運搬依存グラフを新たに導入することによりアルゴリズムの実行効率を向上させている。

2 準備

ループの依存関係を、命令を頂点、ループ内の依存関係を辺とする非循環有向グラフ (DAG)$G = (V, E)$ とループ間の依存関係 (ループ運搬依存)E' で表す。

$V' = \{u, v \mid [u, v] \in E'\}$

$Des'(v) = \{w \in V' \mid w \in Des_G(v)\}$

ここで $Des_G(v)$ は G における v の子孫の集合、L_G は G の高さ、$dep(v)$ はそれぞれ頂点 v の DAG における深さを表す。

2.1 ループ運搬依存グラフ

ループ運搬依存グラフとは、ループ運搬依存を頂点とし、そのループ運搬依存の終点から他のループ運搬依存の始点へのパスがある時、そのクリティカルパス長の重みをラベル付けした有向辺を 2 頂点間の辺とするグラフである。

A New Modulo Scheduling Algorithm for Software Pipelining
Koji Takehara, Shinya Okuda, Michio Oyamaguchi, Yoshikatsu Ohta
Department of Information Engineering, Faculty of Engineering, Mie University
1515 Kamihama-cho, Tsu-shi, Mie-ken, 514-8507 Japan

2.2 前処理

本手法で対象とするループは、ループ分割によりループ運搬依存が影響を与えない頂点を削除し、ループアンローリングにより次のイタレーションのみにループ運搬依存があるようにしたものとする。

また、ループ運搬依存グラフにおいてサイクルに属していないループ運搬依存を除去し、最大の重みの辺を選びその終点から出る最大の重みを持つ辺を順次辿ることで適当なサイクルを見つけ、そのサイクルから新しいループ本体のサイクル数の下界を求める。

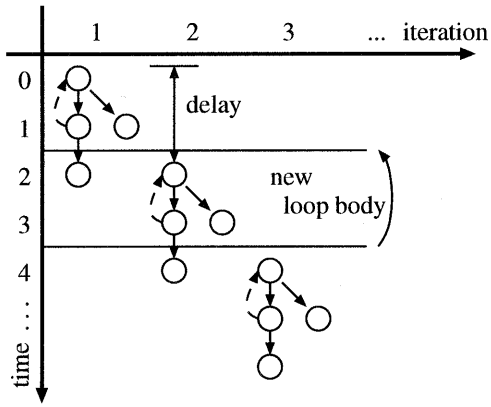

図1　SP 法の例

図1に SP 法の例を示す。図1において実線はループ内依存、破線はループ運搬依存を表す。

3 アルゴリズム

3.1 ループ運搬依存を考慮した DAG のスケジューリング

DAG とループ運搬依存からスケジューリングに使う $down(v)$、$delay(e)$ を求める。

0. **for each** $e = [u, v] \in E'$ **do**
   $delay(e) := dep(u) - dep(v) + 1$;
   $des\_del\_max(e) := Max\{delay(e') \mid$
   e'は v の子孫から出るループ運搬依存 $\}$
   ループ運搬依存グラフから $delay$ の下界 low_bound を計算する。
   $down\_max := Min\{|V'|(\, Max\{delay(e)\}$
   $- low\_bound)\,, |V'| \, low\_bound + L_G\}$
   for each $v \in V'$ **do**　$down(v) := 0$;
1. E' を $delay$ の大きい順(等しいときは des_del_max の小さい順)にソートして、$list\_del$ とする。先頭の辺 $e = [u, v]$ を取り出す。
2. **if** $(down(v) = down\_max$
   or $delay(v) = des\_del\_max(v)$
   or $delay(v) = low\_bound)$
   then *Stop*;
3. $t := down(v)$;
4. **for each** $w \in Des'(v)$ **do**
   if $down(w) = t$ **then** Modify(w);
5. 1 へ戻る

Modify(w);

$down(w) := down(w) + 1$;
for each $e = [u, w] \in E'$ **do**
$delay(e) := delay(e) - 1$;
for each $e = [w, u] \in E'$ **do**
$delay(e) := delay(e) + 1$;

3.2 新しいループの構成

3.1 のアルゴリズム終了時の最大 $delay$ を d とする。元の DAG をグリーディにスケジュールしたものから各頂点をその頂点の $down$ の値だけ遅らせたものを各イタレーションのスケジュールとし、それを d ずつ遅らせてスケジュールしたものから新しいループを構成する。本アルゴリズムは多項式時間で計算を行う。

4 おわりに

本稿では、文献 [3,4] で提案されたモジュロースケジューリング法を改善した新しい SP 法を提案した。

本手法の特長はループ運搬依存グラフを導入して新しいループ本体のサイクル数の下界をあらかじめ計算するなどの前処理を行うことにより、実行効率を向上させていること、及び文献 [4] の本質的な部分を抽出して、より洗練されたアルゴリズムに改善したことである。

参考文献

[1] A.Aiken, A.Nicolau, "Optimal Loop Optimization", Proceedings of the SIGPLAN'88 Conference on Programming Language Design and Implementation, pp.308-317, June, 1988.

[2] B.R.Rau, "Iterative Modulo Scheduling: An Algorithm for Software Pipelining Loops", MICRO27, PP.63-74, November, 1994

[3] B.Su, S.Ding, J.Xia, "URPR-An Extension of URCR for Software Pipelining", Proceedings of the 19th Microprogramming Workshop, pp.104-108, October, 1986.

[4] 武市雅俊, 大山口通夫, 太田義勝, "レベル付き依存グラフを用いた効率のよいソフトウェア・パイプライン化法について", 京大数解研講究録, No.1148, pp.17-22, April, 2000