動いているお茶の水1号 － 同期ビットを使用したコンパイル技法 －

6K－7

稲垣 達氏　松本 尚　平木 敬

東京大学理学部情報科学科

1 はじめに

汎用細粒度並列計算機お茶の水1号[4]は、大域的な同期機構であるElastic Barrierに加えメモリ上の同期ビットを持つ。同期ビットはメモリアクセス単位の同期を通信と同時に行い、フロー依存によって生じる先行関係の効率的実現に有効である。また、バリア型の同期と異なり、制御に余分な先行関係を付加しない。お茶の水1号ではElastic Barrierを使用した細粒度並列処理を行なう最適化コンパイラOP.1[2]が稼働中であるが、OP.1は現在同期ビットを使用するコードを生成しない。本報告では、メモリ上の同期ビットを通信と同期に使用するコード生成手法について述べる。

2 スカラデータの通信

この節ではマルチプロセッサ上の静的タスクスケジューリングを用いた細粒度並列処理において、スカラデータを同期ビットを用いてプロセッサ間で通信する場合のコード生成について述べる。元来フロー依存であるため、バリアで同期をとる場合に比べ、余計な先行関係を導入せず、動的なオーバーヘッド吸収能力が高い。また同期の挿入による命令オーバーヘッドを生じない。

同期ビットは再利用のため極性による管理[3]が行なわれる。これは同期ビットの状態(0/1)のいずれをfull(同期成立)と見做すかという極性情報を動的に切替える方法である。この場合に問題になるのは、同期ビットの極性管理の単位である。同期ビットを用いたロード／ストアでは極性によって異なる値のアドレスを生成しなければならない。コードを極性によらずに再実行可能にするためには、この極性情報は静的なコード内の即値ではなく、動的なデータとして与える必要がある。極性情報は頻繁に使われるので、ある程度長い実行単位を通して同じ値を利用できるのが望ましい。そこで、トレーススケジューリング[1]で用いられるトレースを極性管理の単位とする。各トレースは極性情報をベースアドレスの形で持つ。極性情報自体は、同じベースを使用する全てのメモリアクセスの間で資源管理しなければならないので、各トレース間を移動するときには全体で同期する必要がある。この同期にはElastic Barrierを用いる。

ループ、条件分岐、条件脱出の三つの構造から生成される単純なトレースを考える。ループボディのトレースを再実行可能にするためには、トレース内で同期ビットによる通信が行なわれる領域の極性が、トレースを出た時点で一貫していなければいけない。条件分岐・条件脱出で、実行確率の高い方のトレース(前後の基本ブロックと同一のトレース)を親のトレース、実行確率の低い方のトレースを子供のトレースとする。子供のトレースは親のトレースの一部の実行をキャンセルしてしまうので、子供のトレースで親のトレースのキャンセルされる部分での極性を一貫させなければならない。図1の四角で基本ブロックを表すと、影を付けたブロックが、子供のトレースによって実行をキャンセルされる部分である。

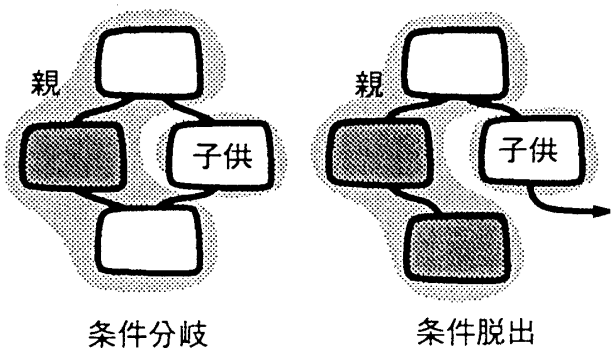

図 1: 実行がキャンセルされる部分

以上ではタスクスケジューリングでの順方向のエッジについて述べた。逆方向のエッジに関しては、上記の方法ではトレース間で同期をとっているため、通信に同期ビットを使う必要はない。

3 配列データの通信

A Compilation Technique using Memory Based Synchronization Bits

Tatsushi INAGAKI, Takashi MATSUMOTO, Kei HIRAKI
Department of Information Science, Faculty of Science, the University of Tokyo

この節ではループのイテレーションを分割して並列処理を行なう場合に、逐次化ループで通信される配列データを、同期ビットを用いて通信する場合を考える。

3.1 同期ビットだけを使う場合

2節でも触れたように、極性情報の一貫性を保つには何らかの別の同期が必要である。従って、逐次化ループの全てのイテレーションで同じ極性で同期ができなければならない。次の条件が成り立つ時に、同期ビットだけで逐次化ループの先行関係を守ることができる。

1. イテレーション間の配列データの先行関係がフロー依存のみ。
2. イテレーション空間を $\mathcal{I}$、配列空間を $\mathcal{A}$ とし、イテレーション $i \in \mathcal{I}$ で読み出す配列の領域を $R_i \subset \mathcal{A}$、書き込む領域を $W_i \subset \mathcal{A}$ とすると、

$$W_i \cap W_j = \emptyset \ (i \neq j)$$

$$\bigcup_i R_i - \bigcup_i W_i \text{ が } i \text{ に対してループ不変}$$

これによって、逐次化ループの先行関係を同期ビットだけで保証できる。また、逐次ループの DOACROSS 実行にも適用できる。

3.2 Elastic Barrier と一緒に使う場合

今度は逐次化ループに出力依存と逆依存がある例として、次のような特殊な場合について考える。各イテレーションは $a_r \in R_i$ を読みだし、$a_r \in W_i$ に書き込む値を計算する。以下の制約が成り立つとする。

$$\forall i \quad R_i \subset W_{i-1} \subset W_{i-2} \quad (a < b)$$

逆依存の距離が短いときは同期が頻繁に必要になるので、配列 $\mathcal{A}$ をリネーミングして二つの配列を交互に使うようにする。この時それぞれの配列アクセスの間の依存をグラフにすると図2のようになる。 点線矢印は

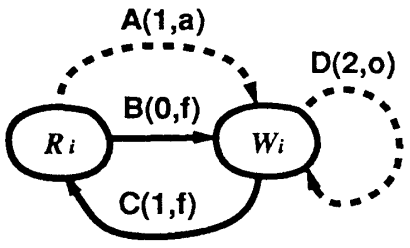

図 2: 配列アクセスの間の依存関係

制御依存 (出力依存と逆依存)、実線矢印はデータ依存 (フロー依存) を表す。括弧内は i の依存距離と依存の種類 (f: フロー依存,a: 逆依存,o: 出力依存) である。依存 B と C はフロー依存であり、イテレーション間に跨る配列 $\mathcal{A}$ の依存は同期ビットで保証できる。依存 A は制御依存で、W_{i+1} は全ての R_i が終了するまで待たなければならないので、ここを Elastic Barrier 同期によって保証する。すると依存 D は $W_{i-1} \rightarrow R_i \rightarrow W_{i+1}$ と推移的に保証される。疑似コードは次のようになる。

```
W0;
preq;
for i = 1 to N do
  Ri;
  preq;
   :
  rreq;
  Wi;
end for;
rreq;
```

同期命令	機能
preq	今回の同期を予告する
rreq	同期条件の成立を待つ

バリア領域が次のバリア領域と重なり、全ての R_i から全ての W_i への同期のオーバーヘッドが緩和されている。

4 おわりに

お茶の水 1 号における同期ビットを使用したコード生成の手法について述べた。今後実機に対するコード生成による評価を行なう予定である。

参考文献

[1] Fisher, J. A., "Trace Scheduling: A Technique for Global Microcode Compaction," *IEEE Trans. on Comput.*, vol. C-30, no. 7, pp. 478–490, July 1981.

[2] 稲垣 達氏, 松本 尚, 平木 敬, "システムの階層的並列性を統一的に扱う最適化コンパイラ," 信学技報 CPSY-94-40, pp. 105–112, July 1994.

[3] Matsumoto, T., T. Tanaka, T. Moriyama, and S. Uzuhara, "MISC: a Mechanism for Integrated Synchronization and Communication using Snoop Caches," in *Proc. of 1991 Int. Conf. on Parallel Processing*, vol. I, pp. 161–170, Aug. 1991.

[4] 戸塚 米太郎, 大津 金光, 中里 学, 秋葉 智弘, 松本 尚, 平木 敬, "汎用細粒度並列計算機: お茶の水 1 号 – 構成と性能評価 –," 並列処理シンポジウム JSPP'94 論文集, pp. 73–80, May 1994.