5G－8

マルチプロセッサシステムにおける リダクションオペレーション

菅沼俊夫・小松秀昭
日本ＩＢＭ（株）東京基礎研究所

1．はじめに

リダクションとは、配列のあるセクションに対して、加減算、乗算など交換法則および結合法則の成立する演算子に基づいて、単一の結果を求める操作である。マルチプロセッサシステムにおいて、通常このリダクションループは、スカラー変数によるデータ依存関係から並列化する事ができないが、プログラム中に極めて頻繁に現われるループパターンである事から、これを効率よく実行させる事は、全体のパフォーマンス向上のために非常に重要である。

従来リダクションループは、プログラム中のある特定の形を持つ断片をイディオムレコグニションとして認識し、この断片をこれに対応する実行時ライブラリの呼び出しへと変換する事で処理されてきた。すなわち、実行時にこのルーチンが呼び出されると、各プロセッサは、このリダクションの対象となる配列の個々の持ち分のデータに対してまず中間結果を求め、次いでそれらを全プロセッサ間でまとめる事で最終結果を得る事になるが、この時対象配列データを持つ各プロセッサ間で通信の為の同期が発生する事になる。しかしながらＳＰＭＤモデルにおいては、各プロセッサの実行制約状態はプログラム全体の並列性に支配されており、リダクションのための通信に対してもプログラムの並列性を損なわないような通信スケジューリングを行なう事が望ましい。

本稿ではリダクションループに対し、プログラムの並列性を保存する、より効率的なコード生成および通信スケジューリングの手法について報告する。

2．リダクションコードの生成

プログラム中にリダクションループを認識すると、これを従来のように単にランタイムライブラリの呼び出しに置き換えるのではなく、ループの並列化ルーチンと協調しながらコードを生成する事を考える。これによって、リダクションループを含むループに対し、並列化可能なループを最大に抽出する事ができ、また各プロセッサの実行制約ループをプログラム中から取り出す事によって、関連するプロセッサ間における通信において不必要な同期を避けるようなスケジューリングをする事ができる。すなわち、コード生成においては以下に述べるように、対象配列に対するリダクションオペレーションの並列性を最大にするため、リダクションループが並列実行可能なループ内に存在する場合はオペランドのプリフェッチングを、そうでない場合には対象配列をそのデータ分割毎にまとめるループ変換を施す。

2－1．ループ変換

リダクションループが最外ループである場合、あるいは並列実行不可能なループ内に存在する場合は、対象となる配列セクションを持つすべてのプロセッサがリダクションループに突入する。従って全体の並列性に影響を与えることなく、リダクションループの変換を行う事が可能である。すなわち、リダクションの対象配列のうちデータ分割が同一の配列を選択してまとめ、それらのオペランド毎にリダクションループを複製する。複製されたそれぞれのループは、対象配列のデータ分割に従って繰り返しの分割が行われ、並列に実行可能となる。

2－2．オペランドプリフェッチング

リダクションループが並列実行可能なループ内に存在する場合、並列可能な次元を保存しつつリダクションを実行させる事が必要となる。並列ループに対しては通常、左辺のオーナープロセッサが計算主体となる(owner-computes rule)ため、ループ本体の右辺のオペランドに対して、通信解析の結果からプリフェッチ可能である時、そのオペランドをベクトル通信により左辺のオーナープロセッサへ送信するコードが挿入される。しかしながら内側のループがリダクショ

Reduction Operation on Multi-processor Computing Systems.
Toshio Suganuma and Hideaki Komatsu
Tokyo Research Laboratory
IBM Japan, Ltd.

ンである場合、個々のデータではなく各プロセッサの持ち分に対するリダクション結果をまとめて送信する事で通信量を減らす事が可能である。

従って、同じプリフェッチレベルを持ちかつ配列のデータ分割が同一のリダクションオペランドに対して、プリフェッチングのためにテンポラリ配列へ結果を格納する並列実行可能なループを新たに生成する（図１）。並列実行されるループは、このテンポラリ配列にアクセスするように変換される。これに伴って、通信メッセージ長は各通信イベントにおいて、配列分割のブロックサイズから単位メッセージ長へと軽減される。

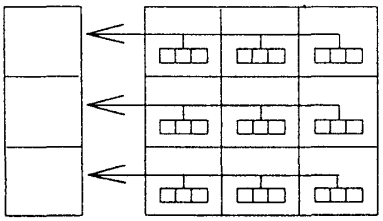

図１：プリフェッチされたオペランドの並列実行の流れ

３．通信スケジューリング

各プロセッサにおいて求められた中間結果をまとめ最終結果を得る為、最終的にランタイムルーチンにより通信が行われる。この時、この通信のスケジューリング手法として実行制約ループを参照し、そのループの並列性と同じ通信スケジューリングを行う事により、全プロセッサ間の同期の必要性をなくし、プログラムの並列性を維持する事を考える。

実行制約ループは次のステップにより求められる。すなわち、

１）リダクションループからスタートし、プログラムをそのコントロールフローをさかのぼってトラバースしループ文をサーチする。

２）パイプライン実行可能なループであり、かつそのループの実行分割がリダクションループと等しいループがあれば、これが実行制約ループとなりそのパイプラインの方向を返す。

３）プログラムの先頭まで到達した場合は、実行制約はないものとみなす。

図２のサンプルプログラムにおいて、配列ａは両次元ともブロックで３分割されているものと仮定する。リダクションの対象配列であるａは直前のループにおいて定義されており、データの依存関係から、図３に示すように両次元においてパイプラインで実行されるループであることがわかる。この場合このループが各プロセッサの制約条件となり、リダクションループに到達する各プロセッサはこの形による実行制約を受けており、従ってこのループと同じ通信パターンを実現することで、プログラム全体の並列実行の流れを保存する事ができる。ランタイムルーチンの中では図４に示すような通信スケジューリングが行われる。

本方式は、ＳＯＲループなど、パイプライン並列のループをくり返し実行しリダクションによってその収束判定を行う場合などにおいて、各プロセッサの実行制約であるパイプライン実行の流れを損なわない通信スケジューリングを行なう事ができ、極めて有効であると思われる。

４．おわりに

プログラム中に極めて頻繁に現われるループパターンであるリダクションを取り上げ、この効率的なコード生成方法および実行時の通信スケジューリングについて考察した。今後は、より多くのベンチマークや応用プログラムを対象にして、その有効性について検討を行いたい。

参考文献

[1] H.Zima, B.Chapman: Supercompilers for Parallel and Vector Computers, ACM Press 1990

[2] M. Wolfe: Optimizing Supercompilers for Supercomputers, MIT Press 1989

```
do i = 1, n
   do j = 1, m
      a(i, j) = a(i-1, j-1)
   enddo
enddo

do i = 1, n
   do j = 1, m
      amax = max(amax, a(i, j))
   enddo
enddo
```

図2：サンプルプログラム

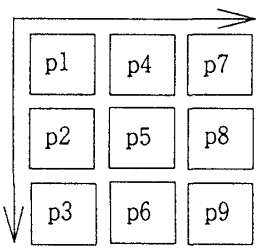

図3：ウエーブフロントの処理の流れ

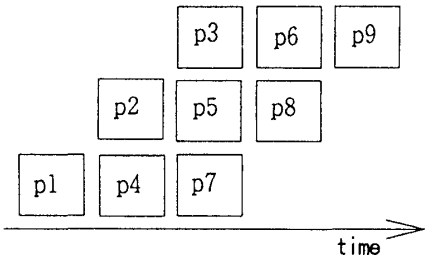

図4：リダクションの通信イベントの流れ