分散環境でのストリーム処理のための並列コード生成

3E－5

浜中 征志郎　首藤 一幸　菅原 健一　村岡 洋一
早稲田大学　理工学部

1. はじめに

分散環境における並列処理の問題点として、通信の性能が低いことが挙げられる。通信のレイテンシが大きい環境での並列処理は、少数の通信であってもそれがネックとなり並列化効率を落とすことがある。

またシミュレーションの可視化、画像処理などの応用にみられるストリーム処理を単純にデータ並列・タスク並列により並列化したのでは、分散環境において効率の良いコードは得られない。

本稿ではストリーム処理を検出し、データ処理の多重化等を行なうことで、分散環境においても効率の良いコードを生成する手法について述べる。

2. 対象とする処理

本稿で扱うストリーム処理とは、以下のものを指すことにする (図1)。

- ストリームを処理する各タスクは前のプロセスまたはファイルなどから次々とデータを入力として受け、何らかの処理を加えて次のプロセスまたはファイルに出力する
- データの流れは一方向で、出力したデータを入力とするようなデータの流れのサイクルはない
- 各タスクは一般にm入力n出力
- 複数のタスクが一定の方向に接続され、全体としてストリームを処理する場合を含む

なおファイルの入出力については並列化することはここでは期待できないものとし、サーバと呼ばれる特定の計算機からのみデータの入出力ができるものとする。

3. 単純な並列化における問題点

上記のようなストリーム処理の各タスクは図2のような形式に一般化することができる。

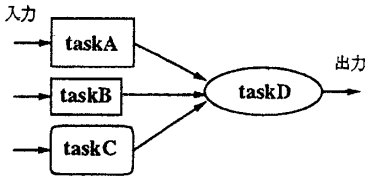

図 1: ストリーム処理

LOOP
　前処理 (B_1)
　// 前のプロセスやファイルからの入力
　input(In_1,...,In_m) (B_1)
　データ処理 (B_2)
　// 次のプロセスやファイルへの出力
　output(Out_1,...,Out_n) (B_3)
　後処理 (B_3)
ENDLOOP

図 2: ストリーム処理を行なうタスクの構造

このようなループにおいては、入力文や出力文に依存があり、更に一般には入出力は分散することはできないため、ループ全体を並列化することはできない。従って「依存距離=1→ 逐次ループ」という単純な並列化の手法ではデータ処理の部分の並列性だけを抽出し並列化することになる。

しかしデータ処理の部分を並列化しても、データの分配・収集には複数の計算機との通信が生じるため、効率の良い処理ができるとは限らない。またもともと並列性が抽出できない場合も考えられる。

4. 並列化の手法

上記の問題点から、本研究では以下の手法によりストリーム処理のための並列コードを生成する。

ストリーム処理はDOACROSS型のループに属するが、データ依存については更に一定の性質を持つ。それは図2のように処理を B_1、B_2、B_3 の3つのブロックに分けた時、B_3 から B_1、B_2 から B_2 へのフロー依存を持たないことである。ループがこのような特徴を持

Parallelized Code Generation for Stream Processing in Distributed Computational Environment
Seishiro HAMANAKA, Kazuyuki SHUDOH, Kenichi SUGAHARA, Yoichi MURAOKA
School of Science and Engineering, WASEDA University

つ時、データ処理(B_2)のタスクの大きさによってはより効率の良い処理が可能である。コンパイラはデータ依存解析を行なって上記の依存関係をストリーム処理を検出した場合、次の処理を行なう。

B_2の部分についてはイタレーション間のデータ依存のある変数を計算機毎にローカルな変数に拡張し、多重化する。これはスカラエクスパンション、アレイエクスパンションと概念的には同じであり、フロー依存が存在しない場合は依存関係を取り除くことができる。これによりデータ処理を複数の計算機でオーバーラップさせて実行できるようになり、B_2に並列性がない場合についても全体としてのスループットの向上を実現する。

さらにB_2が並列性を持つ場合はこれを並列化することにより各データの処理時間の短縮を同時に計ることができる(図3)。ただし並列度を上げると一つのデータが処理される時間は短縮されるが並列化効率が落ちることにもなるので、多重化とのバランスを考慮しなければならない。現時点では多重化のみを行なっている。

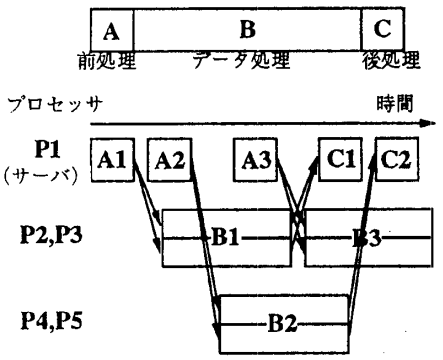

図 3: 並列化と多重化の併合

5. データの分配

多重化された各々のデータはクライアント(データ処理側)に分配されて処理される。現在の実装では完全にスタティックに分配している。サーバは利用できるすべての計算機にローテーション方式でデータを分配/回収していく(データパラレルパイプライン方式)。この方法は利用できる計算機が均質的であり、十分な数の計算機が利用できる場合には、サーバとクライアントとのネゴシエーションを一切必要としないため効率は最善となる。

しかし分散環境においては計算機は均質であることはあまり期待できない。また各計算機の負荷が動的に変化して実行時間の見積りが困難な状況にある。従ってデータを動的に分散/回収する方式が必要となるが、入出力はサーバに限られているため、サーバの負荷を考慮に入れなければならない。そこで本研究では現在以下のような方式を検討中である。

サーバはデータを処理していないクライアントがある間は処理能力の高い順に分配していく(計算機資源については既知のものとする)。サーバはクライアントの計算したデータを先着順に受け取り、バッファリングをして元の順序に正して出力する。サーバはクライアントからのデータの受信により処理の終了を知ることができ、低コストで動的なデータ分配が可能となる。

6. 現在の状況

当研究室で研究されている HAREDAS に必要となる処理を実装中である。現在、ユーザによる指示を元にストリーム処理を行なうコードを生成するコンパイラが実現されている。FORTRAN にストリーム処理の指示(データ処理部を指定するディレクティブによる)を加えたものを入力言語とし、TCP/IP による通信を行なうサーバ/クライアントプログラム(C のソースコード)を生成する。

7. まとめ

本稿では、ストリーム処理を検出し、多重化することで、分散環境においても効率良く処理するコードを生成する方法について述べた。

しかし現在動いている並列化コンパイラは構想の一部であり、まだ多くの問題点を抱えている。実用的なものが完成した段階で評価を行なう予定である。

参考文献

[1] Jaspal Shbhlok and Gary Vondran: Optimal Latency-Throughput Tradeoffs for Data Parallel Pipelines, the 8th ACM SPAA, 1996.

[2] 竹村真一, 仲谷栄伸, 砂原秀樹: 分散環境を利用した並列動画生成, 情報処理学会 SWoPP, Vol.37, pp.1-8, 1991.

[3] Yung-Terng Wang, Robert J. T. Morris: Load Sharing in Distributed Systems, IEEE trans. on Computers, Vol. c-34, No. 3, 1985.