# 並列化コンパイラによるタスク並列とデータ並列の統合処理

3E－6

菅原健一　首藤一幸　浜中征志郎　村岡洋一

早稲田大学 理工学部 情報学科

## 1　はじめに

分散メモリ型並列計算機をターゲットとした、自動並列化コンパイラは一般的にデータ並列性を引き出すことによって高速化をはかっている。しかしデータ並列処理における並列化効率は、プロセッサ台数を増やすほど低下する性質があり、利用可能なプロセッサ数が飛躍的に増大している現在、従来のデータ並列のみの手法では高速化に限界がある。この他にプログラム中の比較的独立した計算単位をタスクとして分割し、タスク間の並列性も利用することによって、プログラム中に存在する並列性をさらに引き出すことが可能である。

現在我々は分散メモリ型並列計算機をターゲットとした、タスク並列とデータ並列を同時に引き出す並列化コンパイラを実装中である。本稿では、本コンパイラの概要について述べる。

## 2　対象とするプログラム

本コンパイラは FORTRAN77 に HPF の基本的なデータ並列のためのディレクティブ (PROCESSORS,DISTRIBUTE,ALIGN,INDEPENDENT) を加えたものを入力とする。タスク並列はコンパイラ側が自動的に解析する。そのためユーザにはプログラムに特別な記述を要求しない。

ユーザ定義関数に関してはインタープロシージャ解析が未実装なため、コンパイラ側で必要に応じてインライン展開する。

本コンパイラではタスクを、基本ブロックの集合で、且つお互いの関係がデータ依存のみで表現できる計算単位と定義する。

## 3　内部構成の概要

現在実装中の並列化コンパイラの内部構成の概要を述べる。大きく分けて以下のようなモジュールから成り立っている。内部処理の順に説明する。

### 3.1　前処理

FORTRAN プログラムを本コンパイラが扱う中間表現に変換し、依存解析を行なう。そして、最大限に並列性を引き出すために、スカラリネーミング、スカラエクスパンションなどのプログラム変換を行なう。

### 3.2　タスクグラフ生成

プログラムをタスクに分割し、依存解析の情報をもとにタスクグラフを生成する。タスクはコンパイラが自動的にプログラムを分割することによって生成する。現在は最外側ループ、連続した代入文、制御の分岐から合流までを一つのタスクとして分割している。ユーザはプログラム中に適当なディレクティブを挿入することでタスク指定することも可能である。

タスクグラフのエッジはデータ依存関係を表している。タスクグラフの各ノードは基本的にデータ並列処理することを前提としているので、エッジは同時に再分散が必要な配列変数を表している。

またこの時点でタスク処理コスト、通信コストの見積りも行なう。

#### タスク並列性

タスク間はすべてデータ依存関係で表現されているので、タスク間の並列性は、先行制約関係を求めることで検出できる。

#### コストの見積り

タスクの処理コストは、本コンパイラが扱う内部表現の複雑度を数値化したもので表現する。ループのコストはそのイタレーション数が決定していれば、ボディのコストの総和にイタレーション数を乗じた値と定め、イタレーション数が決定していない場合はユーザによってあらかじめ設定された値を子階層のコストの総和に乗じることにする。条件分岐のコストは、考えられる分岐先の処理コストの中の最大値に条件文のコストを加えたものとする。

データ並列化した場合のコストは、プロセッサ数、並列実行部分の占める割合、タスク内のループ情報をもとに Amdahl の法則から算出する。

タスク間の通信 (データの再分散) コストは、送受信側のプロセッサ数、配列の分散タイプから算出する。タスク処理に必要なデータはすべて通信を必要とすると想定している。

---

Integrating Task and Data Parallelism
Kenichi SUGAHARA, Kazuyuki SHUDOH, Seishiro HAMANAKA,and Yoichi MURAOKA
School of Science and Engineering, WASEDA University

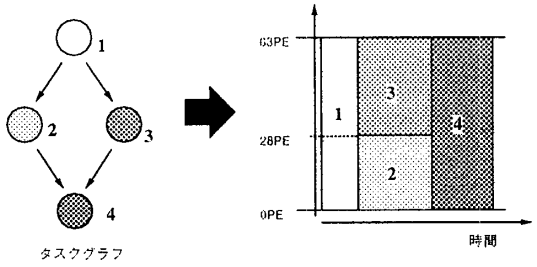


図 1: スケジューリングの例

### 3.3 スケジューリング

実行時間が最小になるようにタスクをプロセッサへ割り当てる (具体的な操作は図1参照)。実行時のオーバヘッドを考え、スケジューリングはコンパイル時に行なう。タスクの実行順序、タスクへのプロセッサ数の割り当て、並列計算機上へのマッピングを同時に行なうアルゴリズムは現時点では実装されておらず、プロセッサ数の割り当て、マッピングに関してはユーザからの情報を必要としている。

### 3.4 タスクグラフのノードのデータ並列化

タスクグラフの各ノードに対して、スケジューリング部で与えられたプロセッサ数の条件下で、Owner Computes Rule に基づきデータ並列化を行なう。単なる Owner Computes Rule による並列化では、生成されるコード内に無駄な部分が多く、効率が良くない。そこで同時に以下のような実行時のオーバヘッド、通信のレイテンシを削減するための最適化も行なう。

- 自プロセッサへの通信の削除
- ループの実行範囲の調整
- 恒真となる IF 文の削除
- SEND/RECV を含むループの分割

**タスク内での配列の分散形式**

コンパイラ側で自動的に再生したタスクに関しては、プログラムの先頭で宣言されている分散形式を継承する。一方、ユーザ指定によって生成されたタスク内に関しては、プログラム中にタスク指定ディレクティブと共に配列分散指定ディレクティブがあれば、その情報に従い、無い場合はプログラムの先頭で宣言されている分散形式を継承する。

### 3.5 コード生成

タスクグラフからターゲットマシン用のCコードを生成する。生成したコード中の通信関数は移植性の良さの点から MPI をベースとしている。プロセッサのグループ化には、MPI のプロセスグループの概念を利用している。

**データの再分散**

あるタスクの中で処理される配列は、プロセッサグループ内に分散されて保持されており、そのタスクで使用された配列が別のタスクで使用される場合は、タスクの処理の終了時と開始時にデータの再分散が必要となる。データの再分散は、専用のライブラリ関数(groupsend,grouprecv)を用意し、コンパイラが生成するコードにその関数を組み込むことで解決する。関数には、送信元及び送信先のプロセッサグループ構成、配列の分散形式を引数として与える。これらの関数はプロセッサグループ内すべてで実行される。

groupsend では、送信元タスクの各々のプロセッサが、送信先タスクのプロセッサに対して送信が必要な配列の要素を計算し、その要素を 1 メッセージにパックして送信する。これは point-to-point 通信で実現されている。そこでの通信のスケジューリングは Ramaswamy[1] の方法を採用している。一方、grouprecv はちょうど groupsend の逆の手続きを行なう。

**タスクの実行形式**

データ並列性を持つタスクはプロセッサグループ内で、完全に独立したデータ並列プログラムとして動作する。一般的なタスク処理の開始から終了までは以下のようになっている。

1. 各プロセッサのグループ内でのランクの取得
2. grouprecv によるデータ収集
3. データ並列処理
4. groupsend によるデータ分散

## 4 まとめ/今後の予定

以上、タスク並列とデータ並列を同時に引き出す並列化コンパイラの概要について述べた。現在、本コンパイラは構想のサブセットであり、スケジューリング、データの再分散ライブラリの一部 (すべての分散形式に対応していない) が未実装である。今後早急に未実装部分を解決し、評価を行なっていく予定である。

## 参考文献

[1] Shankar Ramaswamy, "Simultaneous Exploitation of Task and Data Parallelism in Regular Scientific Applications", Ph.D thesis,Univ. of Illinois at Urbana-Champaign,1996

[2] I.Foster, B.Abalani, M.Xu and A.Choudhary, "A Compilation System That Tntegrates High Performance Fortran and Fortran M", proc. of 1994 Scalable High Performace Computing Conf.