自動並列化コンパイラの開発

工学部・情報工学科 小畑正貴

1. はじめに

多くの計算資源を必要とする大規模な数値計算では常により優れた計算機を必要としているため，従来の単一のCPUを用いる計算機より，さらに大きな計算能力を提供できる並列計算機に対しての期待は大きい。だが，一般に並列計算機向きのプログラムを記述することは一般の研究者には困難であるため，従来の一台の計算機のみを対象として記述されたプログラムを自動的に並列化する自動並列化コンパイラの実現が求められている。

本研究では，並列性が一番わかりやすい形で現れることが経験的にわかっているループの解析を行ない並列化を行なう自動並列化コンパイラの開発と，その評価を行なった．この並列化コンパイラはC言語で書かれたプログラムを対象とし，出力ファイルは実行時にデータ分割などの通信処理を受け持つ専用の通信ライブラリを使用するC言語のプログラムである。また専用ライブラリ内では標準的なメッセージ通信ライブラリであるMPI（Message-Passing Interface）を使用している。

本研究のコンパイラの特徴は，実行時にデータの配置を決定する動的なデータ管理機構を用いる点であり，コンパイル時にはタスクの分割を優先して並列化を行い，データ分散の詳細までは決定しない。これにより低レベルの通信関数は隠蔽され，並列化後のプログラムには直接含まれない．データの分散管理のために必要な通信は，実行時にライブラリによって動的に行われる。

また，このようなデータ管理機構を設けることによって，入力コードと出力コードの対応が容易に判別でき，実行時に詳細な内部情報を簡単に得られるなど，並列後プログラムのデバッグを容易にしている。

実験環境として，Intel社のParagon（分散メモリ型並列計算機）を選んだが，本研究で開発したコンパイラの出力はC言語のプログラムであり，並列化のためのライブラリとしてMPIを用いているため，C言語とMPIが動作する環境では問題無く動作する。

2. システムの概要

本研究で開発した自動並列化コンパイラは図1の構造であり，図1の中で太い線で示されているのがプログラムの変換されていく流れである。ライブラリ内のデータ配置解決モジュールが通信を受け持っていることからわかるように，本研究で開発した自動並列化コンパイラは直

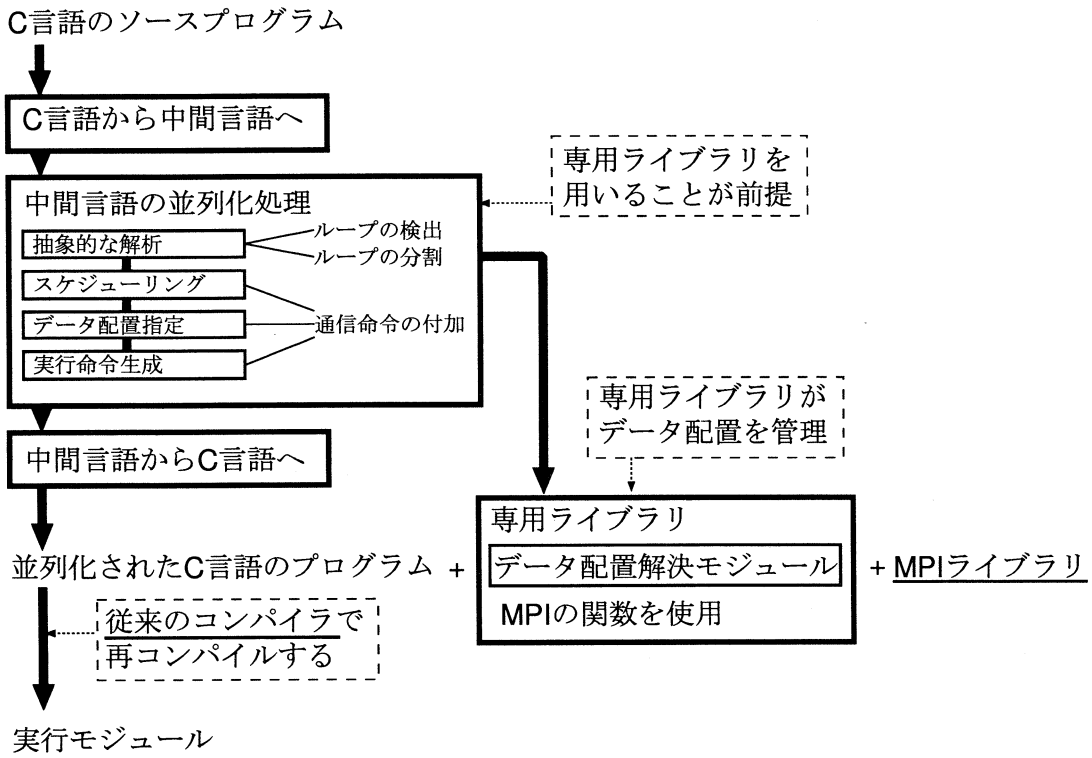

図 1　コンパイラの構成

にMPIの通信命令を埋め込んだコードを生成せず，データ通信については実行時の判断によって動的に通信方式が決定される。

以下に各プログラムとそれに含まれるモジュールについて説明する。

- C言語から中間言語へコンパイルするプログラム
  以後の処理を容易にするためにC言語を一度簡単な形式に変換する。
- 中間言語の並列化処理を行なうプログラム
  並列化された中間言語の命令列を生成する。並列化の判断と専用ライブラリの付加はここで行なう。
  - 抽象的な解析モジュール
    ループの反復回数や，変数の依存解析，データの操作範囲の解析，並列化できるループかどうかの判定，など並列化のために必要な情報の抽出を行なう。
  - スケジューリングモジュール
    抽象的な解析モジュールからの情報を得てどのループを並列化するかの判定を行なう。またデータの分散の方向や分散のための命令の選定などを行なう。
  - データ配置指定モジュール
    スケジューリングモジュールから得られた命令を通信の量や分布を考えた最適化を

したり，プログラム中からのさらに詳細な情報の抜き出しを行なう。

- 実行命令生成モジュール
  データ配置指定モジュールから得られた具体的な並列化のための情報に従って新規の並列化されたプログラムを生成する。ただし直に転送命令を書き込むことはしない。

- 中間言語からC言語へ変換するプログラム
  通常のコンパイラで出力コードをコンパイルできるように中間言語をC言語に逆変換する。
- 専用ライブラリ
  プログラム中のデータの流れを管理する。実行中の各時点でデータがどのような配置であるべきかをコンパイラが示し，ライブラリは自動的にその形式にデータの配置を変換する機能を提供する。
  - データ配置解決モジュール
    配列の情報を管理し，実行時のデータの領域の確保，分割操作並びに，他のノードとの通信を受け持つ。

3. 並列化の手法

3.1 ループ並列化

コンパイラはループの並列化の判断を図1の「抽象的な解析」の段階で行ない，これは以下の2段階の処理で構成されている。

(1) 後に行なう抽象的な解析を容易に行なうためのデータフロー的な解析とそれを用いたプログラムの組変え，並列化できるループの候補の選定。
(2) 前段階で得られた情報を利用しての，そのループの並列化可能かの判断を行なう知識ベースの解析と，その間違いを防ぐための後処理。

両段階は，帰納変数，開始値，増分，反復回数からなるループのパラメータの解析に成功したループの中から，並列化の処理を行なえないループを削っていくことによって最終的な並列化できるループを選び出す。

データフロー的な解析では主にポインタの再利用などによってプログラムが並列化するには複雑になりすぎてないかを判断しループの並列化可能かの判断を行なう。また再帰的に自己への代入を繰り返しているような集合演算に置き換えれる命令の検出を行なう。

3.2 データ管理

多くのコンパイラは，直接プログラム中にMPIなどの通信命令を配置することでデータの管理を行なってきた。たとえばPARADIGMではプログラム内に直接に転送命令を書き込む。

本コンパイラでは変換後のプログラムに低レベルの通信まで含めないので

(1) 変換後のプログラムが理解しやすく，人手による最適化も可能。
(2) 計算機の通信性能に依存する部分を隠蔽できる。

といった利点がある。

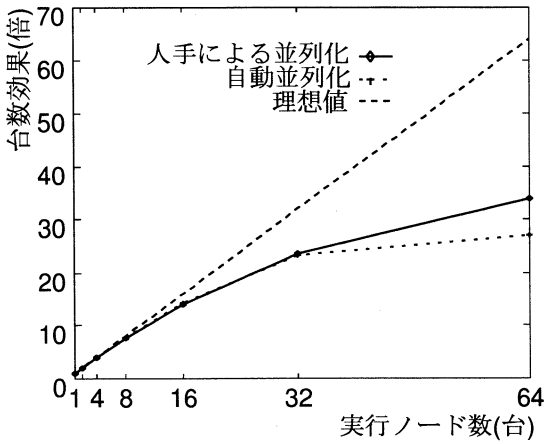

図 2　人手の並列化との比較 (台数効果)

コンパイル時点でのデータの配置などの指定は抽象的なものにとどめて，実際にどのようなデータ配置になるのかは実行時にライブラリが決める。これは，従来の通信ライブラリの層が実際の個別の通信を受け持つ一層のみに限られていたのに対し，その上に抽象的な一層を加えて二層の通信ライブラリを構築したことを示している。

4.　性能評価

本研究で開発した自動並列化コンパイラの評価として，従来のCで書かれたプログラムに対して実際に自動並列化を行ないその効果を確かめた。実行にはIntel社製，分散メモリ型並列計算機Paragonを用いた。実行形式の生成のためのコンパイラには，Paragon用のCコンパイラであるicc/Paragon Sun4 Version R5.0.3を用いている。

4.1　人手との比較

本研究では比較のために共役傾斜法の並列化を行なった。図2は並列化による台数効果を示す。並列化する前のプログラムの実行時間は，617秒である．32ノード程度までは人手との台数効果の差は小さいが，64ノードでは大きく差が出ている。これはノード数が増えるとそれに比例してライブラリの処理時間が増えるためと考えられる。ライブラリの処理時間が増えるのは，現状ではMPI_Alltoallvを用いているからである。

比較対象となる人手によって並列化されたプログラムは高速化のためにプログラムの組みかえを行なっている。これによってノード数が一台の状態でも元のプログラムより高速である。並列化後のプログラムにはオーバーヘッドもあることを念頭において考えれば，複数のノードで実行した場合のライブラリの負荷は小さい。

	問題規模	データ型	反復回数
行列積	512x512	double	-
共役傾斜法	1024x1024	double	1024
クラウト法	1024x1024	float	-
Gauss-Seidel 法	1024x1024	double	1024
SOR 法	1024x1024	double	100

（表 1）　評価に用いたプログラム

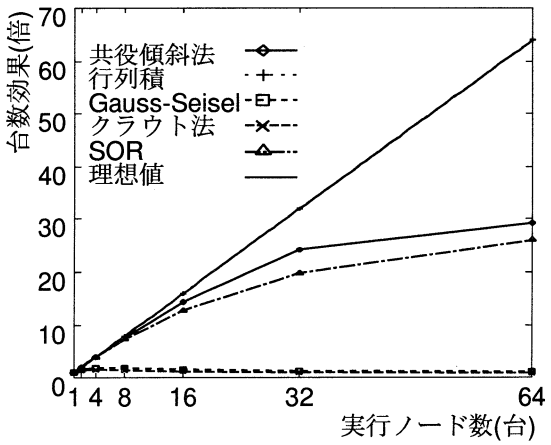

図 3　並列化による速度向上

4.2　コンパイラの評価

評価対象として行列の積を求めるプログラムと，連立一次方程式の解を共役傾斜法と，LU 分解の手法であるクラウト法と，Gauss-Seidel 法と，SOR 法でそれぞれ求めるプログラムを選んだ。条件については表 1 で示す.

図 3 からは，台数効果が三個のグループに別れていることが分かる．理想的なもの（行列積），良好な結果が得られたもの（共役傾斜法，SOR 法），良い効果は得られなかったもの（クラウト法，Gauss-Seidel 法）である。良好な結果が得られたグループは比較的並列化に向いているとされているものである。また，良い効果は得られなかったグループは問題自体が並列化に向いていないとされているものであるが，ノード数が 2〜8 台では，ある程度の効果を得ることができた。

5.　まとめ

本研究で開発したコンパイラは実行時にデータの配置を決定する動的なデータ管理機構を用いるため，コンパイル時にはタスクの分割を優先して並列化を行い，データ分散のための処理はライブラリが動的に行なう。

複数の数値計算のプログラムを並列化して評価した結果，問題によっては人の手による並列化にも匹敵する良好な台数効果を得られた。今のところC言語の文法全てに対応していないなどの低機能なところが多いが，現在の開発段階でもテスト対象にしたプログラムを正常にコンパイルでき，またいくつかのプログラムでは高い効果が得られた。

実行時にプログラムの実行状況に合わせて仕事の分割とデータの分割を行なうことにより，計算機自体の性能やMPIの機能を十分に引き出せるようにすることができるという特徴を持っているが，現状でのライブラリはまだ低機能であり，計算機に適した最適化をライブラリに加えていくことによってさらに効果を向上できると考えられる。