MPI 環境を対象とした自動並列化コンパイラの研究

橋 井 邦 夫*・小 畑 正 貴**

*岡山理科大学大学院工学研究科修士課程情報工学専攻

**岡山理科大学工学部情報工学科

(1998年10月5日 受理)

1 はじめに

より大きな計算能力を得るために分散メモリ型の並列計算機に大きな期待が寄せられているが，一般に分散メモリ型の並列計算機に対して，従来のプログラム言語と通信ライブラリを用いたプログラミングを行なうと，独立した複数の資源 (CPU 他) 上で一つの仕事をするための処理でプログラムがどうしても繁雑になる。このため，現在の手法では高速なプログラムを書くために，ある程度の技術が必要になってしまう。

以上の理由より逐次計算機向きに記述されたプログラムに対して，できるだけ変更を加えずに並列計算機上で効率の良い実行結果を得られるようにする環境の実現が求められている。

本研究では，逐次計算機向きに記述されたプログラムを自動で並列化するコンパイラのプロトタイプの開発とその評価[1]を，一般的な数値解析のプログラムで行なう。

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て，出力ファイルは専用の通信ライブラリを使用するC言語のプログラムの形である。これによって，ユーザーは今までの逐次のプログラムをそのまま並列計算機上で最大限の効率を発揮できる形式に変換でき，対象となる並列計算機上の通常のコンパイラで再コンパイルすることで最大限の実行効率を得ることができるようになる可能性がある。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)[2),3)]を使用している。

MPI を用いることによって現在の評価に用いている Paragon のみならず，他の様々な並列計算機でも効率良く実行できる可搬性のある並列プログラムを生成することを目標としている。

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部分の解析をし，並列化を自動で行なうC言語を対象とした並列化コンパイラを作成し Paragon (Intel 社。分散メモリ型並列計算機) 上で評価を行なう。

本稿では，まず2章では本研究で開発中の並列化コンパイラの概要についてで，これはプロトタイプの結果を踏まえて修正したものである。3章ではプロトタイプの並列化コンパイラによる一般的な数値解析のプログラムの並列化の効果の有効性の評価，4章でまとめと今後の課題について述べる。

2 並列化コンパイラの概要

2.1 並列化コンパイラの目的

本研究で開発中の並列化コンパイラは，以下のような戦略に基づいている。

(1) 並列化が容易で効率的に実行できることがわかっているループ部分を並列化する。

(2) 並列化が容易ではなく，並列化を行なっても効率の向上が得られにくい場合には並列化を断念する。

(3) 並列化コンパイラはユーザーから並列化に関する情報をコンパイルオプションを除き全く求めない。

(4) 出力結果を実行環境に依存しないようにする。

以上よりユーザーが気をつけることは

(1) プログラムを書く上で並列化に適したアルゴリズムを採用する。

(2) ユーザーは並列化しやすいようにプログラムを書く。

つまりユーザーは現在のベクトル化機能のあるコンパイラと同じようにコンパイラを使用すればよいようにすることを目的にしている。

2.2 並列化コンパイラの構造

本研究で開発された並列化コンパイラは，複数の構成要素に分解できる。

(1) C言語から中間言語へのコンパイラ。

(2) 中間言語の並列化コンパイラ。

(3) 中間言語からC言語への逆コンパイラ。

以下にそれぞれの機能の説明と，用語の説明を行なう。

- 中間言語

  アセンブリ言語に近い形式で，解析の簡略化のために使われる。レジスタは無くスタックを中心とした形式で実装されている。C言語の要素を全て分解できるようにしている。並列化処理後のC言語への再変換もできるように構成している。

- 中間言語の並列化コンパイラ

  中間言語を読み込み，並列化された中間言語を生成する。ループの並列化を主な処理内容にしている。

  細かく説明すると，関数のインライン展開，依存関係の調査による定型を持つ命令の抽出，出力の依存関係の調査とそれを用いたループの分割，転送命令の前後関係から生成される依存関係を解決する命令の挿入，の順番で並列化処理を行ない，また，並列化された中間言語を読み込み，挿入された通信命令の内で余分な命令は削除と，データの分割の最適化とそれに伴う命令の挿入を行なう。

- 並列化ライブラリ

  並列化のための情報を得るための関数と，幾つかの入力出力関数を，無駄な動作をしないように並列化，もしくは，選択動作できるようにしたもの，並列化コンパイラ

で自動的に使用される。また並列化コンパイラ内での変換の簡略化のために用いる。出力ファイルがC言語であることを利用して，主にマクロ命令として定義された命令で構成され，所有ノードや実行ノードの判断を行なっている。

2.3 並列化コンパイラの仕様要求

本研究で開発中の並列化コンパイラは以下のルールにしたがっている。

(1) 並列化部分はループ部分だけである。

(2) 並列化のネストはしない。

(3) ループの構造をできるだけそのまま利用する。

(4) ソースに加える変更をできるだけ少なくする。

(5) 関数は main 関数内に inline 展開される。

2.4 並列化コンパイラの動作

本研究で開発中の並列化コンパイラでは中間言語に対して並列化に関する処理を行なう，具体的には以下の順序（図1参照）で処理を行なう。

- ループの反復回数の検出

  C言語では文法上，ループの反復回数は明示されないし，その終了条件は自由に決めることができる。このため，並列化コンパイラではループ分割するためにループの反復回数を検出しなくてはならない。

- ループ並列化可能かどうかの判定

  ループの中には並列化して効果を得ることができないものが多い。このため依存解析を行なって，並列化効果の無いであろうループは並列化しないようにしている。

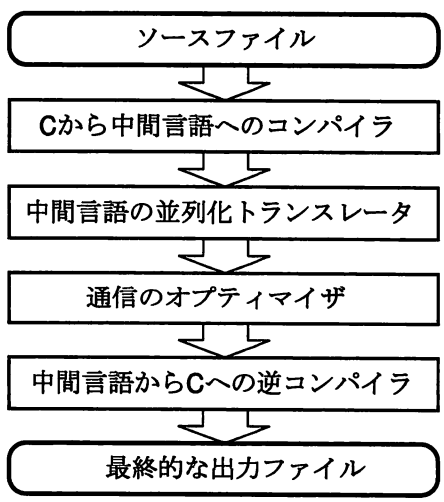

図1 並列化コンパイラの動作順序

• データ分割から見たループ分割の最適化

並列化可能なループが複数ある時，それぞれの並列化の組合せで生じるデータ転送の負荷とその性格について調査して最善の構成を選択する。

• 転送命令の付加

並列化ループ内で生成されるデータを他のループへ転送するための命令を挿入する。またこのための前処理と後処理の命令もそれぞれ挿入する。並列化ライブラリでは多数の小さな通信データをまとめて一つの長い通信データとして扱うことによって，通信命令の起動回数を減らして効率化を行なっている。このためにまとめるためのメモリを確保する必要がある。また転送時に演算を行なう必要がある場合いは特別な命令を付加して効率を向上している。

• 転送命令の最適化

転送命令の付加の際に用いている手法は単純であり，この時点でプログラムの前後関係を解析し最適な処理しているわけではない。このため無駄な転送が生じるので，この削除を行なう。

• 並列化に関する後処理

本研究で関発した並列化コンパイラでは前もって inline 展開されている関数しか並列化しない，またこの inline 展開によって関数自体の大きさが膨れ上がるので，得られたプログラム列からの関数の生成によるプログラムの圧縮が必要になる。

2.5 並列化の例

現段階のコンパイラによって図2のプログラムを並列化すると，図3のプログラムへと変換される。

図3内の PM_ParaLibStartと，PM_ParaLibEnd は並列環境の初期化と終了処理のために挿入される。

図2の18行目から始まるループには依存関係は存在しないのでこれを並列化する。これによりライブラリ内のマクロ関数の PM_PTLoopS（図3の32行目）に置き換えられる。

またデータを分散して所有するために分散対象の配列定義（図2の4行目以降）を PM_DimArray（図3の8，9，10行目）に置き換え，使用する前に PM_DivideDatal と PM_DivideData 2 で，領域を確保する。

3 数値処理プログラムによる評価

3.1 評価目的

本研究で開発した並列化コンパイラのプロトタイプを用いて一般的な数値解析のプログラム[4),5),6)]を実際に自動並列化したプログラムと並列化を行なわないプログラムをそれぞれ実行し，並列化によって得られた効率を算出した。

分散メモリ型並列計算機である Paragon（Intel 社）上で評価を行なっているが最終

的には MPI の動作する全ての計算機で効率良く実行できるようにすることを目標としている。

一般的な数値解析のプログラムとして，行列積，Gauss-Seidel 法，共役傾斜法，クラウト法のプログラムを選んだ。ただし，これらのプログラムには並列化コンパイラがまだ対応していない命令（プリプロセッサ命令，typedef, struct, union 他）を置き換える処理を行ない，時間計測のために MPI の命令を付加している。

なお今回の評価で用いたプロトタイプでは，現在開発中のコンパイラに搭載する inline 展開，データ分割から見たループ分割の最適化，転送命令の最適化の機能が搭載されていないが，ライブラリ内の動的な分散情報の再利用機構は動作している。

3.2 結　　果

逐次プログラムを自動で並列化するコンパイラのプロトタイプを開発し，数値計算プログラムを用いて評価した。

計測された台数効果については，行列式については図 4，共役傾斜法については図 5，Gauss-Seidel 法については図 6，SOR 法については図 7，クラウト法については図 8 に

```
 1:#include <stdio.h>
 2:#include <math.h>
 3:
 4:double a[1024][1024],b[1024],x[1024];
 5:double p[1024],r[1024];
 6:double c[1024];
 7:
 8:void main( int argc, char *argv[] )
 9:{
10:    int i,j,k,l,ii;
11:    int m;
12:    int t;
13:    int max;
14:
15:    max=1024;
16:    e=0.00005;
17:
18:    for(i=0;i<max;i++){
19:        for(j=0;j<max;j++){
20:            a[i][j]=1;
21:        }
22:        a[i][i]=2;
23:        b[i]=max+i;
24:        x[i]=0;
25:    }
26:}
```

図 2　並列化前のプログラム

```
 1:#include "ParaLib.h"
 2:
 3:PM_DimArraySet(3);
 4:
 5:#include <stdio.h>
 6:#include <math.h>
 7:
 8:PM_DimArray(double**,a,0);
 9:PM_DimArray(double*,b,1);
10:PM_DimArray(double*,x,2);
11:
12:double p[1024];
13:double r[1024];
14:double c[1024];
15:
16:PM_PTLoopDim(0);
17:
18:void main(int argc,char** argv)
19:{
20:    int i;
21:    int j;
22:    int P_loop_count_0;
23:    int P_loop_count_1;
24:    int P_loop_CountTmp__0;
25:    PM_ParaLibStart(&argc,&argv);
26:    PM_DivideData2("P_loop_count_1",double,a,1,1024,1024);
27:    PM_DivideData1("P_loop_count_1",double,b,1,1024);
28:    PM_DivideData1("P_loop_count_1",double,x,1,1024);
29:    e=0.00005;
30:    i=0;
31:    P_loop_count_1=1024;
32:    PM_PTLoopS(i,0,+1,P_loop_count_1){
33:        j=0;
34:        P_loop_count_0=1024;
35:        P_loop_CountTmp__0=0;
36:        while(1){
37:            j=(0+(+1*P_loop_CountTmp__0));
38:            if(!((P_loop_CountTmp__0<P_loop_count_0))){
39:                goto TMP2;
40:            }
41:            a[i][j]=1;
42:            P_loop_CountTmp__0=(P_loop_CountTmp__0+1);
43:        }
44:      TMP2:;
45:        a[i][i]=2;
46:        b[i]=1025;
47:        x[i]=0;
48:    }
49:    PM_PTLoopE();
50:  TMP0:;
51:    PM_ParaLibEnd();
52:}
```

図 3　並列化後のプログラム

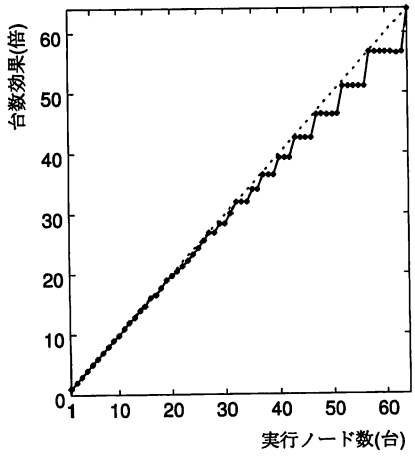

図 4　行列式の台数効果

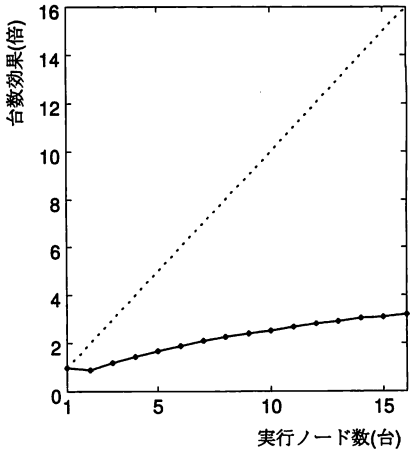

図 7　SOR 法の台数効果

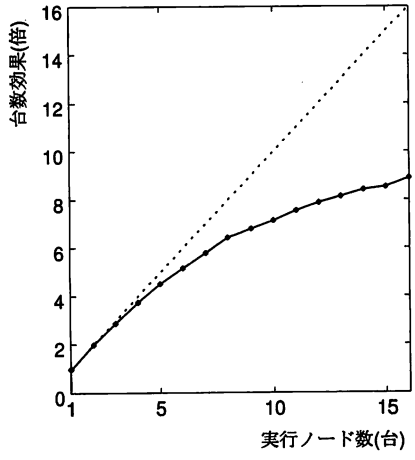

図 5　共役傾斜法の効果

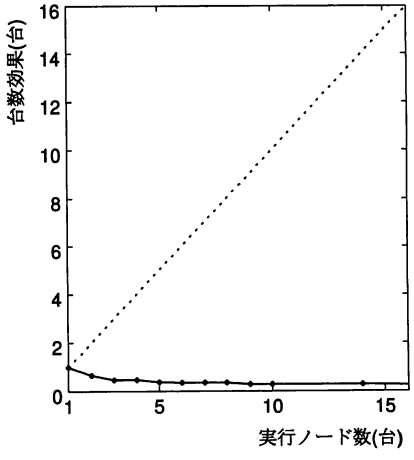

図 8　クラウト法の台数効果

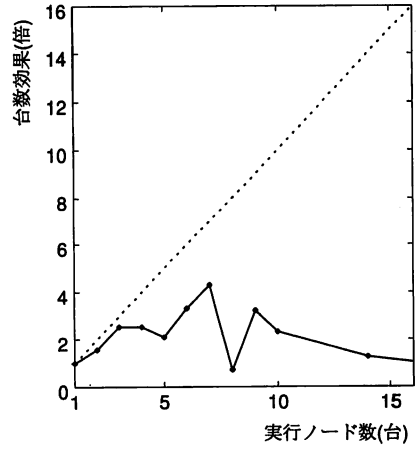

図 6　Gauss-Seidel 法の台数効果

示す。

行列積　　：16台で15.9倍の台数効果を得た
共役傾斜法　　：16台で8.96倍の台数効果を得た
Gauss-Seidel 法：7台で4.29倍までの台数効果を得たが以降は低下
SOR 法　　：16台で3.22倍の台数効果を得た
クラウト法　　：台数効果は得られなかった

4 むすび

主にループ部分を検出し解析し自動で MPI を用いた並列化をし，分散メモリ型並列計算機を対象とする，自動並列化コンパイラのプロトタイプを開発した。

数値計算の評価によって，並列化に向いているような計算ではそれなりの実行効率を得ることができたがデータの転送が演算に比べて多いようなプログラムの並列化においては，ノード数を増やせば増すほど実行効率が低下することがわかった。

データ通信の最適化を施しても，データの転送の大小が大きくプログラムの実行速度に影響することがわかったので，データの転送量を少なくするための新たな機構として，次の開発ではループブロック間のデータ転送を考慮して並列化を行なう機構をコンパイラに搭載することを決定し開発中である。

MPI を用いることによって様々な並列計算機で効率良く実行できる可搬性のある並列プログラムを生成することを目標としているので他の計算機上での実行効率の計測を行ないたい。

参考文献

1) 橋井邦夫，小畑正貴，MPI を対象とする自動並列化コンパイラ，情報処理学会ハイパフォーマンス・コンピューティング研究会**68-5** (1997.10.17)，1997.
2) 青山幸也，RS6000 SP 並列プログラミング虎の巻 MPI 版，日本アイ・ビー・エム株式会社，1996.
3) MPI フォーラム MPI 日本語訳プロジェクト訳，MPI：メッセージ通信インターフェース標準（日本語訳ドラフト），MPI フォーラム，1996.
4) 奥村晴彦，C言語による最新アルゴリズム事典，技術評論社，1991.
5) 戸川隼人，UNIX ワークステーションによる科学技術計算ハンドブック基礎編C言語版，サイエンス社，1992.
6) 水上孝一，市山寿夫，野田松太郎，南原英生，渡辺敏正共著，コンピュータによる数値計算，朝倉出版，1985.

Study on An Automatic Parallelizing Compiler for MPI Environment

Kunio HASHII and Masaki KOHATA
Graduate School of Engineering,
Department of Information & Computer Engineering,
Faculty of Engineering,
Okayama University of Science,
Ridai-cho 1-1, Okayama-shi 700-0005 Japan
(Received October 5, 1998)

For more massively computation, high speed computation by parallel by parallel processing is expected.

In general, as independent resources undertake one task on distributed memory machine, the conventional programming style with sequential language and message passing library is complicated.

Therefore, some degree of programming technique is raquired to obtain the efficiency of parallel processing.

So it is necessary to develop a parallelizing compiler which translates sequential programs to parallel programs for distributed memory machine.

We are developing an automatic parallelize compiler, and evaluated it by executing general numerical analysis programs by using a current version of prototype.