ハイパフォーマンス
コンピューティング 68－5
（1997. 10. 17）

MPIを対象とする自動並列化コンパイラ ～プロトタイプ開発とその評価～

橋井邦夫† 小畑正貴†

大規模な数値計算では，並列化による高速化が期待されている．しかし，並列計算機向きのプログラムの記述は，並列可能部分の認識や通信関数の導入などの困難があり，ユーザには使いにくく，逐次言語で記述されたプログラムを自動的に並列化するコンパイラの実現が求められている．

本研究では，並列化を自動で行なうコンパイラのプロトタイプを開発し，一般的な数値解析のプログラムで評価を行なった．

コンパイルの対象となるプログラムはC言語で普通に書かれたプログラムであり，並列化コンパイラに対しての並列化に関する特殊な命令を含んでいない．また，出力ファイルには，並列計算機でのメッセージ通信の標準ライブラリであるMPI(Message-Passing Interface)を用いた専用ライブラリで並列化を行なう．

An automatic parallelizing compiler for MPI environment ～development of prototype and it's evaluation～

KUNIO HASHII† and MASAKI KOHATA†

In large scale numerical analysis, high speed computation by parallel processing is expected. But there are many difficulties in description of parallel programs, such as recognizing parallelizable part and insertion of communication function. So it is necessary to develop a parallelizing compiler which translates sequential programs to parallel programs for distributed memory machine.

We developed a prototype of the automatic parallelize compiler. and evaluated by executing using general programs of numerical analysis.

The source program is a regular C language program, and the parallelized program includes MPI(Message-Passing Interface) library.

1. はじめに

次世代の高性能計算機として並列計算機に大きな期待が寄せられている．しかし，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，データの分散やループの分割，通信命令を挿入などの逐次のプログラムに必要無い処理が必要になってしまい，プログラムがどうしても繁雑になる．高速なプログラムを書くためには，ある程度の技術が必要になってしまい，プログラミング自体も簡単ではない．よって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コンパイルでき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)[1] [2] を使用している．

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．

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

† 岡山理科大学工学部
Faculty of Engineering, Okayama University of Science

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した並列化コンパイラの概要について, 3 章では一般的な数値解析のプログラムの並列化による効果と有効性の評価, 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

本研究で開発した並列化コンパイラは, 以下のような戦略に基づいて開発されている.

(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) ループリストラクチャのような, ユーザーの書いたプログラムに対しての変更はなるべく控える.
(4) 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.
(5) 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) 中間言語の並列化トランスレーター
(3) 並列化された中間言語の通信命令のオプティマイザ
(4) 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.

- 中間言語の並列化トランスレーター

  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, の三段階に分かれ, それぞれが独立したプログラムである.

  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.

- 並列化された中間言語の通信命令のオプティマイザ

  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の削除と, 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. これによって最終的なコンパイル結果が得られる.

- 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.

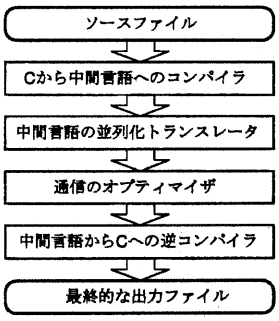

図 1 並列化コンパイラの動作順序

2.3 並列化コンパイラの仕様要求

本研究で開発した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

2.4 並列化コンパイラの動作

本研究で開発した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で処理を行なう.

(1) 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.

(2) 並列化ループの決定
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. このために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.

(3) 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. また転送時に同時に演算を行なう必要がある場合には特別な命令を付加して効率を向上している.

(4) 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見て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.

(5) 並列化に関する後処理
本研究で開発した並列化コンパイラでは二重の並列化には対応していない. このために並列化されていないループの中では並列化されたループを含む関数は存在できるが, 並列化されたループの中では並列化されたループを含む関数は存在することはできない. この矛盾を解決するために, 並列化ループを含む関数に対して, 新たに並列化していない関数を生成しプログラムに追加しなくてはいけない場合があり, これを検出して処理する.

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を用いて一般的な数値解析のプログラム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.

3.2 行列積

評価には文献 [4] 内のプログラムを, 本研究で開発した並列化コンパイラで自動並列化した. 現在並列化コンパイラが対応していない typedef 命令などは人手によって修正している.

評価には図 3 のプログラムを本研究で開発した並列化コンパイラで並列化した.

このプログラムを並列化を行なわずに動作させるには MPI の命令を使っているので, MPI の初期化のために関数の最初と最後に並列化ライブラリの命令を加えるようにしなくてはならない. また P_idcount にはプログラムの実行されるノード数が代入される.

このプログラムを並列化した場合の台数効果は図 2. に示す.

並列化コンパイラで行なうデータ転送命令の最適化によって無駄な転送が無くなり良い結果が得られている.

ノード数が増えて現れるグラフの階段状の変化は同期の時間待ちによるものであると考えられる.

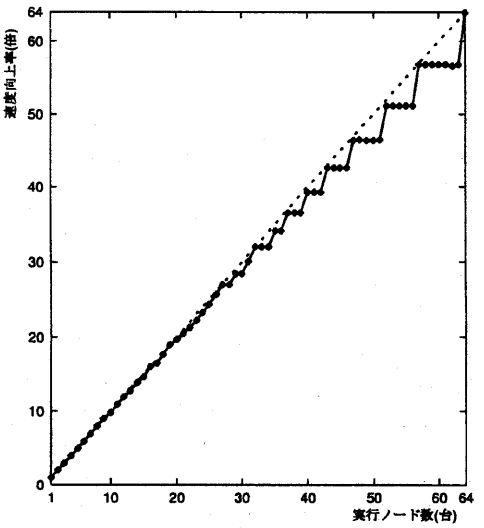

図 2 行列積の実行

3.3 共役傾斜法

評価には文献 [5] 内のプログラムを, C言語に書き換え, 本研究で開発した並列化コンパイラで自動並列化した. 現在並列化コンパイラが対応していない typedef 命令などは人手によって修正している. 実際に計測に用いたプログラムの概要は図 5 に示す.

収束条件を無効にして反復回数を固定にしている.

このプログラムを並列化した場合の台数効果は図 4. に示す.

プロセッサ数が増加すると 1 プロセッサ当たりの演算量の減少によって並列化効率が低下すると考えら

```
#include <stdio.h>
#include <math.h>
#include "paralib.h"
int a[512][512],b[512][512],c[512][512];
void main( int argc, char *argv[] )
{
    int i,j,k,l,ii;
    int m;
    int t;
    double s;
    int max;
    double st,et;
    max=512;
    MPI_Barrier(MPI_COMM_WORLD);
    st=MPI_Wtime();
    for(i=0;i<max;i++){
        for(j=0;j<max;j++){
            s=0.0;
            for(k=0;k<max;k=k+1){
                s=s+a[i][k]*b[k][j];
            }
            c[i][j]=s;
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
    et=MPI_Wtime();
    if(P_myid==0)
     printf("%d\t%f\n",P_idcount,et-st);
}
```

図 3　行列積のプログラム

れる.

また並列化コンパイラで行なうデータ転送命令の最適化によって無駄な転送が無くなり良い結果が得られている.

共役傾斜法での実行効率が高いものになっているのはデータ転送が必要になる部分が限られているからである.

これを図 5 を用いて説明する.

25 行目のループでは全く依存関係は生じないのでそのまま並列化することができる. そして内部で生成される r[i] は 43 行目,48 行目,53 行目,60 行目で参照, または代入されるがこの際のループの分割は 25 行目と同様に考えることができる. このために並列化コンパイラは r[i] に関しての転送命令は挿入しない.

また p[i] については 37 行目で参照されるがこの時の添字 j は並列化対象のループである 34 行目の添字とは違うので,37 行目ではどのような参照の仕方をするのかを本研究での並列化コンパイラでは知ることはできないので, この並列化ループの実行までに全ノードに全ての p の要素を転送してループの実行準備を整えなくてはならない. このためにこの場所に全体への転送命令を挿入する.

やがて p は 60 行目で使用されるがこの後にはループの反復が待っているので, 並列化コンパイラは遡って p の参照があるのかを調べる. ここで, 再び 34 行目で用いるので, 全ての p の要素を転送してループの実行準備を整えなくてはならない. このためにこの場所に全体への転送命令を挿入する.

同様に a,x,y について調べるとこれらも r と同じく転送命令は必要無い.

よって通信命令は集合演算 (MPI_Allreduce) と配列 p の転送命令を挿入する必要があるのみである.

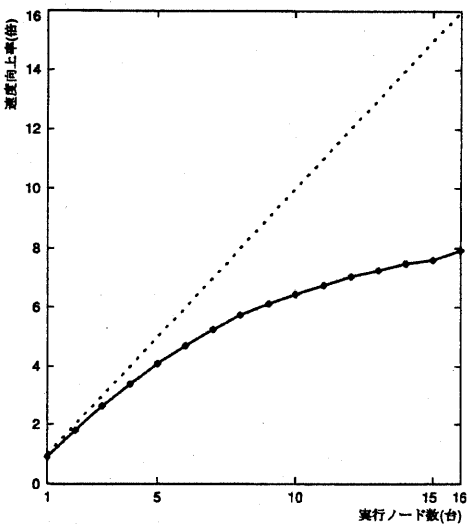

図 4　共役傾斜法の実行

3.4 Gauss-Seidel 法

評価には文献 [3] 内のプログラムを, 本研究で開発した並列化コンパイラで自動並列化した. 現在並列化コンパイラが対応していない typedef 命令などは人手によって修正している.

実際に計測に用いたプログラムの概要は図 7 に示す.

配列は 1024x1024 の大きさで収束条件は無効にし, 反復回数を固定 (1 回のみ) にしている.

このプログラムを並列化した場合の台数効果は図 6. に示す.

プロセッサ数が増加すると 1 プロセッサ当たりの演算量の減少によって並列化効率が低下すると考えられる. 図 6 で示す Gauss-Seidel 法のプログラムを本研究で開発した並列化コンパイラで並列化した場合に通信が挿入されるのは, データの差分を取っている 22 行目と 23 行目である. この集合演算は行列の 1 行毎に 1 列分だけ行なわれるので, 1024x1024 の行列でも 1024 個分の和しか計算しない. このためにループの分散によって得られる演算時間の短縮と通信を伴う集合演算での通信量の台数分の増大の関係は微妙になってしまう. このために Paragon のメッシュ型のネットワーク構成中での各ノードの配置によって計算時間が上下していると考えられる. Gauss-Seidel 法では集合演算を含むループを列分繰り返すことになるので, グラフが極端な変化をしたと考えられる。

3.5 クラウト法による LU 分解

評価には文献 [4] 内のプログラムを, 本研究で開発した並列化コンパイラで自動並列化した. 現在並列化コンパイラが対応していない typedef 命令などは人手

```
 1: #include <stadio.h>
 2: #include <math.h>
 3: #include "paralib.h"
 4:
 5: double a[1024][1024],b[1024],x[1024];
 6: double p[1024],r[1024],y[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:     double st,et;
15:     double w,rr,pap,alpha,s,e,e1,beta;
16:
17:     PF_ParaLibS(&argc,&argv);
18:
19:     max=1024;
20:     e=0.00005;
21:
22:     MPI_Barrier(MPI_COMM_WORLD);
23:     st=MPI_Wtime();
24:
25:     for(i=0;i<max;i++){
26:         w=0.0;
27:         for(j=0;j<max;j++){
28:             w=w+a[i][j]*x[j];
29:         }
30:         r[i]=b[i]-w;
31:         p[i]=r[i];
32:     }
33:     for(k=0;k<max*2;k++){
34:         for(i=0;i<max;i++){
35:             w=0.0;
36:             for(j=0;j<max;j++){
37:                 w=w+a[i][j]*p[j];
38:             }
39:             y[i]=w;
40:         }
41:         rr=0.0;
42:         pap=0.0;
43:         for(i=0;i<max;i++){
44:             rr=rr+r[i]*r[i];
45:             pap=pap+p[i]*y[i];
46:         }
47:         alpha=rr/pap;
48:         for(i=0;i<max;i++){
49:             x[i]=x[i]+alpha*p[i];
50:             r[i]=r[i]-alpha*y[i];
51:         }
52:         s=0.0;
53:         for(i=0;i<max;i++){
54:             s=s+r[i]*r[i];
55:         }
56:         e1=sqrt(s);
57:         /*計測時はこの部分を削除*/
58:         if(e1<e)break;
59:         beta=s/rr;
60:         for(i=0;i<max;i++){
61:             p[i]=r[i]+beta*p[i];
62:         }
63:     }
64:
65:     MPI_Barrier(MPI_COMM_WORLD);
66:     et=MPI_Wtime();
67:     if(P_myid==0)
68:      printf("%d\t%f\n",P_idcount,et-st);
69:     PF_ParaLibE();
70: }
```

図 5　共役傾斜法のプログラム

によって修正している.

このプログラムを並列化した場合の台数効果は図 8. に示す.

プログラムが長いのでここではその一部を示す.

本研究で評価したプログラムでは再外郭ループは 3 個存在する. このうち最初の二つ

```
1:for(i=0;i<max;i++){
2:     l[i][0]=a[i][0];
```

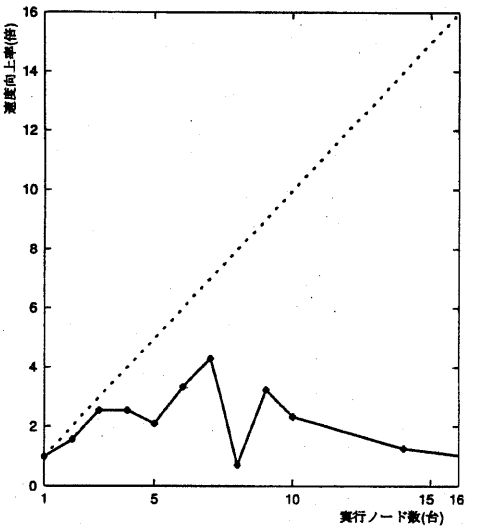

図 6　Gauss-Seidel 法の実行

```
3:     u[i][0]=0.0;
4:}
5:p=a[0][0];
6:u[0][0]=1.0;
7:for(j=1;j<max;j++){
8:     l[0][j]=0.0;
9:     u[0][j]=a[0][j]/p;
10:}
```

では 1 行目のループで生じたデータと, 7 行目で生じたデータは両者とも全ノードに転送されなくてはいけない. 転送データに対してのデータ生成のための演算量は小さくさらに本研究で評価したような小規模の配列での演算では転送時間の方が並列化の効果に比べて小さくなってしまう.

またこのような形式でのデータ転送は, 最適化を行なう段階でも削除することはできない.

次に

```
1:for ( i=1 ; i<max ; i++ ){
2:     …略…
3:     for ( j=i+1 ; j<max ; j++ ){
4:         s=a[j][i];
5:         for ( k=0 ; k<i ; k++ )
6:             s-=l[j][k]*u[k][i];
7:         l[j][i]=s;
8:         u[j][i]=0.0;
9:         s=a[i][j];
10:        for ( k=0 ; k<i ; k++ )
11:            s-=l[i][k]*u[k][j];
12:        u[i][j]=s/p;
13:        l[i][j]=0.0;
14:    }
15:}
```

というループが現れる. 1 行目ループに関しては並列化を既に諦めているとすると, 並列化コンパイラはまず書き込みが存在する領域 l (中間言語の前後関係に依存する) に着目して 7 行目の l[j][i] と後に出現する 6 行目の l[j][k] と 11 行目の l[i][k] と 13 行目の l[i][j] が依存関係にあると判断する (8,13 も同様). このため

```
 1: #include <stdio.h>
 2: #include <math.h>
 3: #include <mpi.h>
 4: #include "paralib.h"
 5:
 6: double a[1024][1024],b[1024],x[1024];
 7: void main( int argc, char *argv[] )
 8: {
 9:     int i,j,max;
10:     double s;
11:     double st,et;
12:     int ok, iter;
13:
14:     max=1024;
15:     PF_ParaLibS(&argc,&argv);
16:     MPI_Barrier(MPI_COMM_WORLD);
17:     st=MPI_Wtime();
18:     /*for(iter=1;iter<=500;iter++)*/{
19:         ok = 1;
20:         for(i=0;i<max;i++){
21:             s = b[i];
22:             for(j=0;j<i;j++)
                  s-=a[i][j]*x[j];
23:             for(j=i+1;j<max;j++)
                  s-=a[i][j]*x[j];
24:             s/=a[i][i];
25:             if(ok&&fabs(x[i]-s)>
                  0.000001*(1+fabs(s))){
26:               ok=0;
27:             }
28:             x[i]=s;
29:         }
30:         /*if (ok) break;*/
31:     }
32:     MPI_Barrier(MPI_COMM_WORLD);
33:     et=MPI_Wtime();
34:     if(P_myid==0)
35:      printf("%d\t%f\n",P_idcount,et-st);
36:     PF_ParaLibE();
37: }
```

図 7 Gauss-Seidel 法のプログラム

に 3 行目のループも並列化できないためにこのループ全体では並列化できるループは 5 行目と 10 行目となる. この場合には通信を伴う集合演算用いられる.

以上のような並列化によって生じる無駄なデータの退避や転送が大きな負荷になってグラフに現れている.

またノード数が増えて現れるグラフの階段状の変化は集合演算での待ち時間がノードの配置で変化するためだと思われる.

3.6 並列化ライブラリの負荷

図内の一台の時は並列化処理をしていないプログラムで実行している. 一台の時の並列化処理を行なっているプログラムとの対比は以下のようになる.

並列化による速度低下 (一台)

	並列化前	並列化後	比率
行列積	112.6 sec	115.0 sec	1.02 倍
共役傾斜法	176.7 sec	195.1 sec	1.10 倍
Jacobi 法	42.1 sec	62.5 sec	1.48 倍
クラウト法	39.3 sec	41.9 sec	1.06 倍

この時間の差は並列化ループの反復数の計算などが行なわれているからであると考えられる.

4. まとめと今後の課題

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.

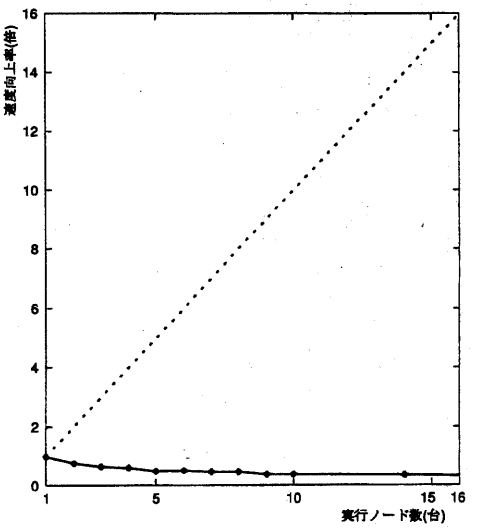

図 8 クラウト法の実行

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や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た現段階で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をコンパイラに搭載することが現段階の大きな課題である.

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いたい.

5. 参考文献

[1] 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

[2] 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

[3] 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

[4] 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

[5] 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