1 E − 6

分散メモリ型並列計算機のための自動ループ並列化の実現

藤原優史　田村智洋　新名博　岩澤京子
東京農工大学工学部電子情報工学科

1. はじめに

分散メモリ型並列計算機用のプログラムを作るのは，専門の知識を必要としデバッグも簡単ではないので，自動並列化コンパイラが有効と考えた．そこで，本システムでは機能限定 C 言語で記述されたプログラムを汎用的並列ライブラリ MPI を使った汎用性のある並列プログラムに自動変換することを試みた．

2. システムの構成

本システムの入力は C 言語のソースファイル，出力は MPI ライブラリを用いた並列計算機用のソースファイルである．

入力ファイルの構文解析，字句解析を行い，中間語に変換する．その中間語を解析・変換し，その結果 MPI 関数などが中間語に追加され，その中間語から並列プログラムを生成する．

図 1 にシステムの全体図を示す．

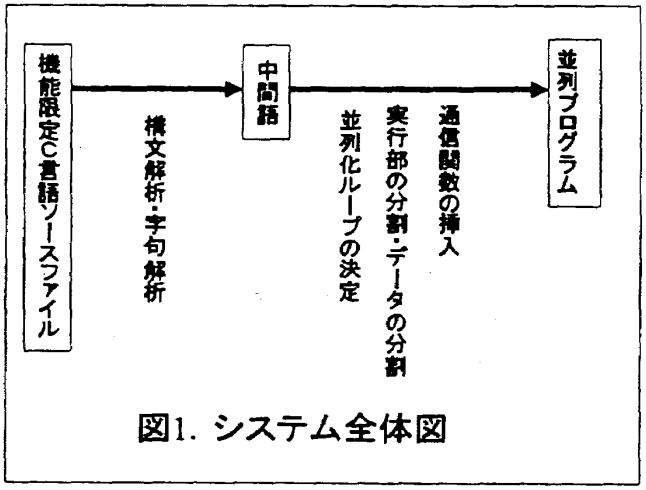

図1. システム全体図

Implementation of Automatic Parallelization symstems of loops for distributed computer
Masashi Fujiwara Chihiro Tamura Hiroshi Niina Kyoko Iwasawa
Tokyo Univ.of Agri. and Tech.

3. 並列化ループの決定と実行部の分割

基本的に多重ループの場合は，外側のループを並列化ループとし，配列を含んでいる場合は，配列の添え字を変化させているループを並列化の対象とする．

実行部の分割は，ループの各繰り返しを巡回的にプロセッサに割り当てる．

4. 各プロセッサへのデータの割当

変数は，各プロセッサが全変数を持ち，各プロセッサが必要に応じて毎回通信を行って値を保証している．

配列は，各プロセッサで定義又は使用する可能性のある要素についてのみ持ち，複数のプロセッサで重複して持つ要素は，その更新があった場合に，その要素を持つプロセッサ全てに通信することにより，最新の値を保証している．

5. 通信関数の位置決定と挿入

通信の位置を決める際には，定義-使用連鎖，使用-定義連鎖，イタレーション間依存情報を利用している．まず定義-使用連鎖情報で並列化ループ内で定義しているデータの使用箇所をリストアップし，その定義が使用を全て支配していなければ，また，配列の場合は添字解析の結果，他のイタレーションに定義が到達すれば，他のイタレーションから定義が到達する可能性があるということなので，通信が必要と判断する．

受信の位置は，リストアップされた使用箇所の中から，実行順序が一番早いものの直前とし，送信の位置は，使用-定義連鎖から得られる到達定義の中から，実行順序が一番遅いものの直後とする．

また，通信関数の位置が，if 文中やループ内など条件分岐の片側のみに存在するように決定された場合は，else 側にも同じ関数を挿入する，ループの外側で通信を行うなどして，並列化ループ内のどの経路を通っても全く同じ通信が行われるようにし，かつ，送信・受信が必ず 1 対 1 で対応するにしている．

6. 実行例

図 2 はシングルプロセッサ用のバブルソートのプログラムで，図 3 はそれを本システムで自動並列化した後の出力で，この図では，通信箇所などを図中に文章で示してある．

7. 実験結果

このシステムで，バブルソートのプログラムを並列化し，120 個のデータ数で cenju-3 で実行させたところ，

・1 プロセッサ：0.79 ミリ秒（並列化なし）

・8 プロセッサ：2.63 ミリ秒

という結果が得られた．

自動生成された並列プログラムは，元のプログラムの意味を保持することは達成できたが，実行速度が並列化しない場合より遅くなるという結果である．

この原因として考えられるのは，ループの実行部よりも，通信とその通信先の判定にかかる時間の方が多くなったためだと考えられる．また，バブルソートが並列化に向いていないアルゴリズムであったことが最も大きな要因であると考えられる．

今後，効率の良い並列プログラムを生成するための最適化の機能を研究開発する予定である．

自動生成したプログラムの正しさの確認と性能測定に NEC C&C 研究所並列処理センターの並列計算機 cenju-3 を使わせていただき，深く感謝いたします．

```
#include <stdio.h>
#define  N  100

void  main(){
    int  a[N];
    int  i,j,t;

    for(i=0;i<N;i=i+1){
        for(j=0;j<N-1;j=j+1){
            if(a[j]>a[j+1]){
                t=a[j];
                a[j]=a[j+1];
                a[j+1]=t;
            }
        }
    }
}
```

図 2. 入力例

```
#include <stdio.h>
#include <mpi.h>
#define  N  100
#define  NPE  8

void  main( ){
  int    pid,npe;
  int    a[N];
  int    i,j,t;;

  /*プログラムの最初で*/
  /*各プロセッサへのデータの割当を行う*/

  for(i=0;i<n;i=i+1){

    /*並列化ループ*/
    for(j=0;j<N-1;j=j+1
      /*a[j]は前のイタレーションに依存しているので*/
      /*ここで前のイタレーションから値を受信する */
      Recv a[j]
      if(a[j]>a[j+1]){
        t=a[j];
        a[j]=a[j+1];
        /*a[j]は前のプロセッサと重複しているので*/
        /*値を送信する*/
        Send a[j]
        a[j+1]=t;
        /*a[j+1]は次のイタレーションのa[j]で使用するので*/
        /*ここで次のイタレーションに値を送信する        */
        Send a[j+1]
      }
      else
        /*送信命令はif文の中にあり行われない可能性が*/
        /*あるのでelse側にも必要                      */
        Send a[j]   Send a[j+1]

      Recv a[j+1]
    }

  }
}
```

図 2. 出力例