動的情報を採取・利用する並列プログラミング環境

岩沢 京子

東京農工大学 工学部 電子情報工学科

静的な解析情報と動的な解析情報を組み合わせて用いる並列プログラミング環境を提案する．静的な解析で確定できないデーターフロー情報は，確定条件を解析し，プログラム中条件の成否に関わる部分を検出する．この部分を実行させることにより，各プログラム変換に必要な情報を収集する．静的な情報と動的な情報を組み合わせるためにデータベースと中心とする環境が必要となる．自動並列化，並列プログラムの最適化が最終的な目的であるが，ここでは，デバッグや並列化変換方式の研究に利用できると考える，並列プログラムの逐次プログラムへの変換を報告する．

An Environment for Parallel Programming Collecting and Using the Dynamic Information

Kyoko Iwasawa

Department of Computer Science
Tokyo University of Agriculture and Technology

The Parallel programming environment, which uses the combined statically analyzed information and dynamically analyzed information, was proposed. To the undecided data flow information analyzed statically, the requirements for deciding data flow is analyzed. And the parts of user programs concerned with this requirements is detected. To collect the necessary information, only these detected parts should be executed. In order to combine the dynamic information and the static information, the core of this environment becomes the database. Though the final purpose is the automatically parallelization and the optimization of parallel programs, serializing conversion of parallel programs, which will be used for debugging and studying parallelizing methods was reported,.

1．はじめに

ハードウェアの低価格化や軽量化により，多数台のプロセッサをつなげたさまざまなモデルの並列計算機が提案されている．これらの計算機をより有効に使うためには，並列プログラミング環境の充実が必要である[1),2),3)]．静的な解析には自ずと限界があり，動的な解析だけに頼るとそのデータ量は爆発する．そこで，静的解析と動的解析を組み合わせることが重要となる．

各種データがドライバを介して有機的に接続しているデータベースを中心とした並列プログラミング環境を提案する．必要な動的情報だけを採取し，また，一度採取した情報や解析結果はデータベースとして管理する．データベースでは，ターゲットとする計算機の特性情報や個別のユーザプログラムの静的な解析情報，実行時に得た情報や実行させたときのデータなど情報，これらを関係付けて保持する．

様々な形態の並列計算機に対して，並列計算機の予備知識のないユーザがプログラミングやデバッグが可能な環境を提供するために，システムには次のような機能を持たせる．
静的に決定しない制御フロー情報やデータフロー情報は，実際にプログラムを実行させて情報を採取する．この情報を用いて各種プログラム変換を行う．ただし，動的な情報を用いるわけだから，データを採取した時点から，ソースプログラムも実行に必要なデータも更新されていないことを確認する必要がある．また，一度解析した情報で可能なものは再利用する．そして，これらの解析情報は，デバッグやプログラム変換（最適化・並列化）に用いるので，常にソースプログラムと対応づけた形で管理する．逆に，ソースプログラムの変更に伴う解析情報の更新も必要な処理である．

2．構成

具体的に目指しているのは，次のような機能を実現することである．

（1）静的に決定しない制御フロー情報やデータフロー情報の確定条件の生成
（2）プログラム実行により上記条件の成否の情報を収集
（3）実行環境の変更に伴う動的情報の無効化
（4）マシンに合わせた最適化（並列化も含む）の組み合わせ
（5）インクリメンタルな解析や変換
（6）新たなプログラム変換の登録
（7）プログラム変換の効果の見積り
（8）プログラムの実行により性能情報収集機能

そこで，データベースを中心に，ユーザが入力した状態から希望する状態となるまでを解析を変換を繰り返しながらデータベースをデータが流れていく．図1に概要を示す．

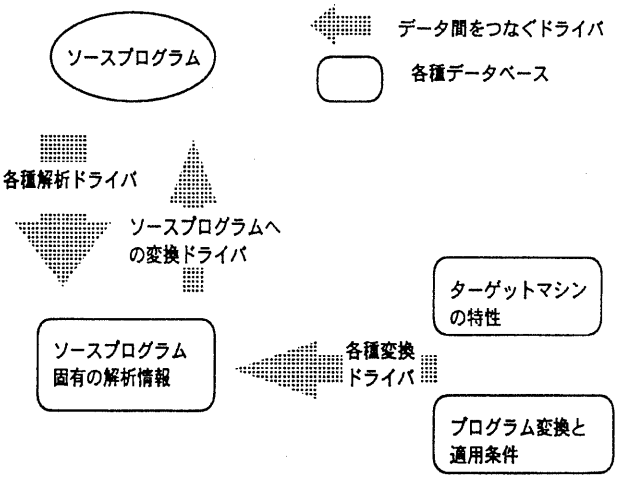

図1　構成

ユーザが入力するソースプログラムをフローグラフを兼ねた中間語に変換する．これが，ソースプログラム固有情報のキーとなり，実行可能なソースプログラムへも変換可能な形式である．これから，データ基本情報解析解析ドライバがデータの定義／使用

の基本情報を作成する．さらに，データ依存解析ドライバがデータ依存情報，データ領域解析ドライバがデータ領域情報を作成するが，依存情報や領域情報を作るとき，静的には確定させることができないことがある．このような場合は，その成立条件も作成する．

また，プログラムの実行性能の推定に必要なターゲットとするハードウェアの特性を保持する．

一方，並列化を含めた各種プログラム変換とその適用条件のデータがある．適用条件には，ソースプログラムの文脈を守るための条件と変換によって実行性能を落とさないための条件の２種類がある．デバッグのために，プログラム変換を施すときなどは，性能条件を守る必要なないと考える．まず，各種変換ドライバは，変換可否を判定する．判定に必要な解析情報がないときは，解析ドライバによるインクリメンタルな解析によりデータフロー情報を更新する．ここで，静的に確定できないときはその成立条件を挿入したソースプログラムを実行させ，出力結果から判定する．変換可能と判断した場合は，解析情報を更新しつつ中間語を変換するが，更新不可能と判断すると関連する解析情報をすべて削除して無効にする．

中間語から逆変換したソースプログラム，中間語，各種解析情報などは常に表示可能である．

3．データベース

本システムの中心となるデータベースには次のような情報を持たせる．大きく分けて各ソースプログラムごとのデータベースと，各ターゲットマシン毎のデータベースと，一般的な共通データベースとがある．ソースプログラムごとのデータは主に解析情報であり，静的なデータと動的なデータがある．一般的な共通データには，様々なプログラム変換とそのその適用条件がある．

3．1　各ソースプログラム固有の情報

個々のユーザプログラムに固有の情報である．これには，静的な情報と動的な情報があり，制御フローやデータフローの解析情報や実行比率に関する情報がある．言語としては，Fortran77にFortran90の配列記述を加えたもの，これら対応する仕様の範囲では，Ｃでもかまわない．現状では，レコード型や実行時にメモリを確保／解放してポインタによりアクセスするデータについては，解析の対象としていない．今後の課題の一つである．

3．1．1　静的な解析情報

プログラム固有の解析情報のデータとそれらのつなぐ解析ドライバとの関係を図２に示す．

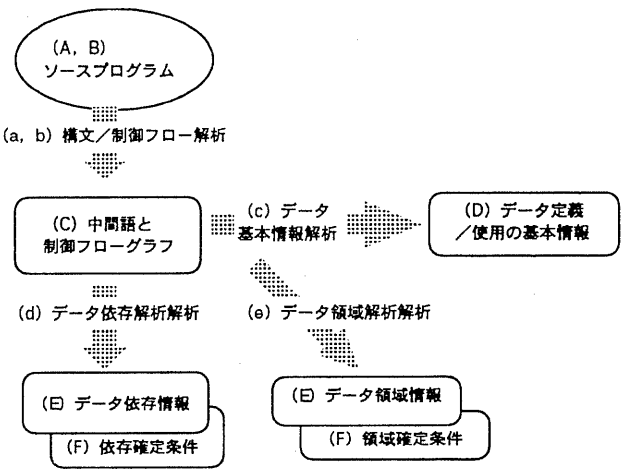

図２　解析情報と解析ドライバの構成

(A) プログラムのスタイル

システムに対するユーザの入力はソースプログラムとそれに対するマクロな変換指示である．ソースプログラムが次のいずれの型であるかをユーザが指定する．場合によってはこれらのいずれかに変換することを指定する．

・逐次型プログラミング
・ＳＰＭＰ(SingleProgrammingMultiProcessors)型
(単一メモリ空間／プロセッサごとのメモリ空間)
・ＭＰＭＰ(MultiProgrammingMultiProcessors)型

（単一メモリ空間／プロセッサごとのメモリ空間）

（B）ユーザファイル（手続きや実行に必要なデータ）の関係・更新情報

動的な解析データを用いるためにはデータが変更されたとき，それを用いた解析情報を無効にしなければならない．そのために，実行に必要なデータファイルの更新情報を保持する．さらに，手続き間解析に備えて，ファイルと手続きの関係（一般には必ずしも1対1ではない）やファイルの更新情報を保持する．

（C）構文解析した結果の中間語と制御フローグラフ

ソースプログラムを解析した結果は，基本ブロックをノードとして制御フローも表す中間語（辞書を含む）で表現する．これは，ソースプログラムに逆変換可能な情報を保持している．

（D）データ定義／使用の基本情報

フローグラフに対応させて，変数や配列の定義と使用に関する基本情報を基本ブロックごとに保持する．

（E）データ依存グラフ，領域情報，

フローグラフとデータ定義の基本情報から，データ依存関係と配列に関する領域情報を解析して保持する．主として，領域情報は変換の判定に，依存関係はプログラム変換に用いる．

（F）依存および領域情報の確定条件

データ依存関係や領域が，静的には制御が決定しない，配列の添え字の動きが解析できないなどの理由により，確定できない場合，確定するための条件，目的とする変換が可能となる条件をソースプログラムに挿入可能な状態で保持する．

（G）実行性能の推定値

静的に見積もった実行性能値．

3．1．2　動的な解析情報

実際にプログラムを実行させて次の様な情報を収集する．図3に示す．

・分岐の成立条件
・依存及び領域情報確定情報の成否
・ループ繰り返し回数，実行比率
・並列処理のイベント情報
・動的情報を収拾したときのデータファイル／ソースファイルの更新情報
・測定した性能

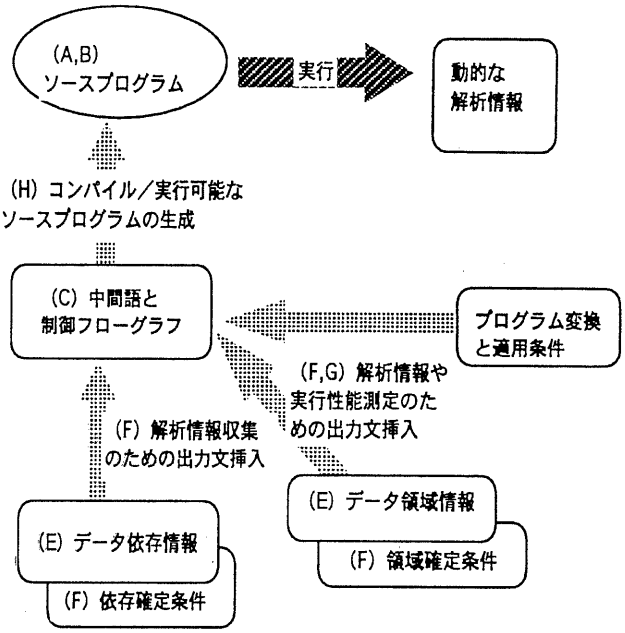

図3　動的情報の収集

3．2　ターゲットマシンの特性

ターゲットとととする並列計算機の特徴を保持する．

・メモリ共有（物理共有，物理分散）／メモリ分散
・同期コスト，や通信コストである．マシンによってはデータ量の関数となる．
・並列実行に必要な組み込み関数の機能と引き数の意味

3．3　プログラム変換と適用条件

システムが用意するプログラム変換とその適用条件である．適用条件には，ソースプログラムの文脈を保持するための条件と性能上の条件（実行時間を劣化させない条件）の2種類がある．前者は一般的で，ユーザプログラムやターゲットマシンに依らず共通の情報である．後者は，ターゲットとする計算

機に依存する．

文脈保持のための条件はデータ依存関係の論理式で，性能向上のための条件はターゲットマシンの情報と同じシステム定数と実行負荷の論理式から記述する．

4．データを変換し情報を引き出すドライバ

与えられたデータから，必要なデータをえるための各種解析・変換処理がデータベースから情報を引き出したり，データを異なる形に変換する．

4．1　解析系

(a) 構文解析

ソースプログラム→中間語（辞書を含む）

(b) 制御フロー解析

中間語→フローグラフ

中間語に制御に従う構造を付加する．

(c) データ基本情報解析

中間語，フローグラフ

→基本ブロックごとの定義／使用情報

(d) データ依存解析

フローグラフ，データ基本情報

→データ依存グラフ，依存成立条件

データ基本情報から依存グラフを作るが，確定できないものについては，成立条件を解析する．

(e) 領域解析

フローグラフ，データ基本情報，データの依存関係→領域情報，領域確定条件

データ基本情報や添え字に関する依存グラフから配列の定義／使用に関する領域情報を作るが，確定できないものについては，領域確定条件を解析する．

（f）条件成立に必要な計算の特定

フローグラフ，データ依存関係→中間語

依存成立条件や領域確定条件，または各種変換適用条件など目的に必要な条件に成否を確認するために必要な計算を特定する．実際には，その部分だけを実行させることにより，解析時の不要な演算を避けることができる．

（g）イベント解析に必要な計算の特定

ターゲットマシンの並列関数情報，フローグラフ，データ依存関係→中間語

ターゲットマシンの並列処理を行うためのデータ送受信や同期などの組み込み関数の特定の引き数の決定に必要な計算を特定する．実際には，その部分だけを実行させることにより，通信の不整合などの誤りを検出することができる．

4．2　変換系

4．2．1　マクロな変換

いくつかの基本的な変換の組み合わせから成り立つ．何通りもの変換手段から実行効率を優先させて選択する．次章に並列プログラムの一本化変換の例を示す．プリミティブな変換の組み合わせを変えることにより新しいマクロな変換を定義することができる．この変換は静的／動的解析情報を用いて中間後に対して行う．

（1）メモリ分散型並列化

（2）メモリ共有型並列化

（3）逐次化（並列プログラムの一本化）

4．2．2　基本的な変換

中間語やデータフロー情報と適用条件から，変換可否を判定する．判定に必要な解析情報が無い場合は，適宜解析ドライバを呼び出し，解析させる．

（4）から（7）までは中間語上の変換であり，データフロー情報や変換適用条件から変換可否を判断し，必要に応じて変換処理を施す．そのとき，データフロー情報は可能な限り修復するか，不可能な場合は対象とする部分の情報を削除する．（8）は中間語をソースプログラムに変換するもので，特に判定処理は必要ない．

（4）ループ構造変換

分割・一重化・融合・展開・交換など

（5）最適化

（通信のまとめ，不要な同期通信の削除）

（6）解析情報収集のための出力文挿入

データフロー情報を確定させるための条件文や

変換可否を判定するための条件文とそれらの条件の成否を出力する出力文を中間語に挿入する．

（7）実行負荷（Erapsed-time，CPU-time）測定のための関数／出力文の挿入

一般に手続きやループの実行比率を調べたいとき，並列実行時の負荷バランスを見たいとき，変換による効率向上が確定できないときなど，計時の組み込み関数とその結果を出力する出力文を中間語に挿入する．

（8）コンパイル・実行可能なソースプログラムの生成

中間語→ソースプログラム

コンパイル可能，かつ，可読性のあるソースプログラムを生成する．

4．3　表示系

（1）解析結果の表示

（2）変換結果（中間語／ソースプログラム）の表示

（3）実行負荷に関する測定結果の表示

4．4　学習系

（1）変換効果予測

中間語から得られる演算量や通信／同期処理，ターゲットマシン情報から実行性能を見積もる．精度よりも見積時間が早いことを目指している．

（2）変換効果の評価と適用条件への反映

予測実行性能，変換ソースプログラムの実行結果

→変換適用条件，ターゲットマシン情報

実際にプログラムを実行させて実行性能を測定した場合，見積性能と比較し，修正の必要があると判断した場合は，その変換の適用条件やターゲットマシン情報を更新する．

5．逐次化（並列プログラムの一本化）

このような構成の環境を用意して，最終的に実現させたいのは逐次プログラムの自動分割・自動並列化であり，分散プログラムに対する新しい最適化である．

しかし，効率の良いデータの自動分割や処理の自動並列化の手法は確立されておらず，メモリ分散型の並列計算機に対しては，ユーザが並列計算機の構成や各性能を意識してプロセッサに対応させてプログラムを作成する方が，デバッグの困難さがあるにせよ，一般に実行効率はよい．既存のシステムにおいてもプロセスを逐次に動くようにしてプログラムをデバッグする手法が用意されているものがある．

プロセッサごとに分散してコーディングされた並列プログラムを，ただ単に順次1台のプロセッサで逐次に動かすのではなく，まったく同じ計算を1台のプロセッサで逐次に実行するテキスト上もただ一つのプログラムを生成することが可能である．一般には，並列プログラムを作成する前に問題を解決するためのアルゴリズムを考える．生成された逐次プログラムがこれと合致するか否かを調べることは並列プログラムのバグを取るのに有効である．

また，効率のよい並列プログラムを逐次に変換する手法を集めれば，その逆変換を行えば，効率のよい

ＳＰＭＰ（ターゲットマシンはnCUBE2）のプログラムは次に示すような手順でプロセッサごとに記述されたプログラムを一本化し逐次プログラムにすることにより，ユーザの意図した処理を行っているか否かの判定を行うことができる．簡単な差分法のプログラムを例に示す．対象とした並列プログラムを次に示す．

（1）並列処理のための組み込み関数を検出と引き数の解析する．

ターゲットマシンの情報よりプロセスidの変数とこの変数の値により処理が別れる部分を検出する．

```
C   parallel programming
    parameter (ni=10,nj=10)
    integer myid,pid,hostid,ndim
    real*8 py(0:ni+1,0:nj+1),q,r
    whoami(myid,pid,hostid,ndim)
    n=2**ndim
          ・
      do 100 i=0,ni+1
      do 100 j=2,nj-1
        py(i,j) = 0.d0          ；初期条件
100   continue
      j=1
      do 110 i=0,ni+1
        py(i,j)=q               ；境界条件
110   continue
       j=nj
      do 120 i=0,ni+1
        py(i,j)=r               ；境界条件
120   contine

      do 500 i=1,ni
      do 500 j=2,nj-1
        if (i.eq.1) then
          if (myid.eq.0) then
            py(0,j) = ((py(1,j)+py(0,j-1)+py(0,j+1))/3.
          else
            nread(py(0,j),8,myid-1,j)
          endif
        endif
      py(i,j) = ((py(i-1,j)+py(i+1,j)+py(i,j-1)+py(i,j+1))/4.
      if (i.eq.ni) then
        if (myid.eq.n) then
          py(ni+1,j) = ((py(ni,j)+py(ni+1,j-1)+py(ni+1,j+1))/3.
        else
          nwrite(py(ni,j),8,myid+1,j)
        endif
      endif
500   continue
          ・
end
```

（2）最外側にプロセスidによる全プロセス数のループを作る．

```
    do 900 myid ←1：n
        ・
        ・
900   continue
```

（3）作成したプロセスidループでループ可変な変数は配列に，配列は次元数を一つ増やす．

```
    real*8 py(0:ni+1,0:nj+1,0:n),q,r

    do 900 myid ←1：n

     py(i,j, myid )

900   continue
```

（4）同期／通信の有る部分とない部分でループ分割を施し，無いループはプロセスidの初期値，増分値を任意に決める．ループ不変な文や式をループの外へ移動する．

```
    do 900 myid=0,,n
    do 100 i=0,ni+1
    do 100 j=2,nj-1
        ・
100   continue
    do 110 i=0,ni+1
        ・
110   continue
    do 120 i=0,ni+1
        ・
120   contine
900     continue

    do 910   myid←0：n
    do 500 i=1,ni
    do 500 j=2,nj-1
        ・
        ・
500   continue
910     continue
```

（5）ネットワークを疑似配列として導入し，データの送受信を疑似配列との代入文に変換する．プロセスidとデータ識別子とデータの長さの3次元の疑似配列とする．

```
nread(py(0,j,myid),8,myid-1,j)
                ↓
 py(0,j,myid) = net(myid,j,8)

nwrite(py(ni,j,myid),8,myid+1,j)
                ↓
net(myid+1,j,8) = py(ni,j,myid)
```

（6）上記の疑似配列に対して，必ず定義が先になるように通信があるプロセスidループの初期値と増分値を決める．必要であればループ分割を行う．

```
    do 910  myid = 0,n
    do 500 i=1,ni
    do 500 j=2,nj-1
        ・
        ・
500   continue
910     continue
```

（7）代入文の伝播により疑似配列を削除する．必要であれば，イベントトレースから伝播可否を判定する．このとき，疑似配列が削除できない場合は，データの送受信に誤りがあり，それがソースに対応して検出することができる．

```
py(0,j,myid) = net(myid,j,8)
            : where 1 ≦ myid ≦ n
          ↓
 py(0,j,myid) = py(ni,j,myid-1)


 net(myid+1,j,8) = py(ni,j,myid)
            :where 0 ≦ myid ≦ n-1
          ↓
 py(0,j,myid+1) = py(ni,j,myid)
```

（8）可能な場合は，（7）の代入文により（3）

の処理で増やした次元を，減らして元に戻す．増やしたプロセスid に関する次元の添え字を０にして表現する．

```
py(i,j,myid) → py(ni* myid +i,j,0) → py(ni* myid +i,j)

py(0,j,myid) → py(ni* myid +0,j,0) → py(ni* myid +0,j)

py(ni,j,myid-1) → py(ni*(myid -1)+ni,j,0)
                                  → py(ni*(myid -1)+ni,j)

py(0,j,myid+1) → py(ni*(myid +1)+ni,j,0)
                                  → py(ni*(myid +1)+ni,j)
```

```
    parameter (ni=10,nj=10, n=プロセス数)
    integer myid,pid,hostid,ndim
   real*8 py(0:(ni+1)*ni,0:nj+1),q,r
          •
    do 910 k=0,n
    do 500 i=1,ni
    do 500 j=2,nj-1
     if (i.eq.1) then
       if (k.eq.0) then
         py(ni*k,j) = ((py(ni*k+1,j)+py(ni*k+0,j-1)
                                    +py(ni*k+0,j+1))/3.
       endif
     endif
     py(ni*k+i,j) = ((py(ni*k+i-1,j)+py(ni*k+i+1,j)
                        +py(ni*k+i,j-1)+py(ni*k+i,j+1))/4.
     if (i.eq.ni) then
       if (k.eq.n) then
         py(ni*k+ni+1,j) = ((py(ni*k+ni,j)
              +py(ni*k+ni+1,j-1)+py(ni*k+ni+1,j+1))/3.
       endif
     endif
500   continue
910   continue
        •
      end
```

（９）ループ分割やループ一重化などの最適化処理を施して，最終的には十分読みやすい逐次のプログラムにする．

```
    parameter (ni=10,nj=10)
    integer myid,pid,hostid,ndim
   real*8 py(0:ni+1,0:nj+1),q,r
          •
      do 100 i=1,ni*n
      do 100 j=2,nj-1
       py(i,j) = 0.d0
100   continue
      j=1
      do 110 i=1,ni*n
       py(i,j)=q
110   continue
     j=nj
      do 120 i=1,ni*n
       py(i,j)=r
120   contine

      i=1
      do 500 j=2,nj-1
       py(i,j) = ((py(i+1,j)+py(i,j-1)+py(i,j+1))/3.
500   continue
      do 510 i=2,ni*n-1
      do 510 j=2,nj-1
       py(i,j) = ((py(i-1,j)+py(i+1,j)+py(i,j-1)+py(i,j+1))/4.
510   continue

      i=ni*n
      do 610 j=2,nj-1
       py(i,j) = ((py(i-1,j)+py(i,j-1)+py(i,j+1))/3.
610   continue
        •
    end
```

今回は，SPMP型のプログラムの一本化方式しか示すことができなかったが，このスタイルのプログラムのデバッグに有効であることが確認できたら，プロセッサごとに全く異なるプログラムの一本化についても検討していきたい．

５．おわりに

データベースを中心とした並列プログラミング環境を提案した．静的な解析情報を利用することにより，動的な情報を有効に採取しプログラム変換の判定に役立たせる．その一例として並列プログラムの一本化を示した．従来，ソースプログラムと対応付けが困難なイベントトレースや，プロセッサの台数を制限して逐次化して行っていた並列プログラムのデバッグに役立たせることができると考える．

まだ，データを中心とした一つの並列プログラミング環境を提案したにすぎない．今後，これらの機能を実装させ，評価を行っていく．また，今回の構想には入れなかったが，手続き間解析や，動的なメモリ割り当てを行うプログラムの最適化についても検討していきたい．

参考文献

[1] D.J.Huson et al. : An Advanced Source-to-Source Vectorizer for S-1 Mark IIa Supercomputer, in Proc. of the 1986 International Conf. on Parallel Processong, IEEE Press (1986).

[2] K.Kennedy et al. : Interactive Parallel Programming Using the ParaScope Editor,IEEE Trans. on Parallel and Distributed Systems, Vol. 2, No. 3, pp. 329-341 (1991).

[3]菊池：並列化支援システム，情報処理学会誌, Vol. 34, No. 9, pp. 1158-1169 (1993).