計算機アーキテクチャ 79 − 9
（１９８９．１１．２１）

反復法を対象とする並行プログラムの再構成システム

木村 哲郎　　朴 泰佑　　天野 英晴

慶應義塾大学理工学部

並行プロセス記述言語で記述されたプロセスをマルチプロセッサ上で実行する場合、一般的にプロセスの粒度が小さくなると、プロセス間交信などのオーバヘッドが増大し、効率的に計算を行うことが難しくなる。筆者らが提案するPRIMEシステムは、ユーザが記述した並行プログラム中のプロセスの粒度を評価し、複数の小粒度プロセスを融合することによってプロセスの粒度を増大させるシステムである。融合においては融合されるプロセスの実行順序を静的にスケジュールする必要があり、これにはかなり大がかりな実行前解析が必要となる。PRIMEシステムでは、問題の範囲を反復法による計算に絞ることにより、実行前解析の負荷を軽減している。本稿ではPRIMEシステムおよび専用言語であるNCC/iの概要と実装について述べる。

Concurrent Program Reconstructing System for Iterative Method

Tetsuro Kimura , Taisuke Boku , Hideharu Amano

Faculty of Science and Technology
Keio University

3 - 14 - 1 Hiyoshi, Kohoku-ku, Yokohama 223, Japan

In this paper, a concurrent program reconstructing system for iterative method, named PRIME is proposed. In the calculation with concurrent processes on multiprocessors, the overhead caused by the control of fine grain processes degrades the system performance. One of the good solutions for this problem is to reconstruct the program merging fine grain processes into more coarse one. However, this processing requires a large amount of analysis of the program. To analyze the program easily, the special language named NCC/i is provided. In this language, the method for solving problems is confined within the iterative method. The PRIME system analyzes programs described in NCC/i, and reconstructs it into new one with more coarse grain processes.

1 はじめに

本大学で開発された $(SM)^2$-II[1] の目的の一つは、電子回路解析、電力潮流などの一般疎行列を係数とする一次方程式の求解に帰結されるような問題を反復法を用いて高速に解くことである。このシステムでは、ユーザはNC モデル [2] と呼ばれる一種の並行プロセスモデルを用いてプログラミングを行う。$(SM)^2$-II はNC モデルの持つ交信の性質に適した交信機構を備えているため、頻繁に起こるプロセッサ間交信に対しても効率良く処理を進めることができる。

NC モデルでは問題の持つ並列度と同数のプロセスを生成させて計算を進めるので、大規模な問題を解く場合 PU(Processing Unit) 数を大幅に上回るプロセスが作られる。この場合、1 つの PU には多数のプロセスが割り当てられソフトウェアによる管理の下で処理が進む。このとき、計算量に対して交信の割合が非常に大きい（これをプロセスの粒度が小さいという）と交信に伴うオーバヘッドやコンテキスト・スイッチ等が増大し、処理速度の低下を招く。

この問題に対して我々は 2 つのアプローチを行っている。一つは、プロセスの管理機構のハードウェア化によるオーバヘッドの削減である [4]。もう一つは、生成されるプロセスの粒度が大きくなるように、ユーザの記述した並行プログラムの再構成を行う方法である。PRIME は特に対象を反復法にを利用したプログラムに絞り、この様な再構成を行うシステムである。本稿では、この PRIME システムの概要について述べる。

2 問題の記述

2.1 NC モデル

$(SM)^2$-II における問題記述の例として一般疎行列を係数とする大規模な連立一次方程式を考えよう。このような問題では係数行列の要素数が少ないため、フィルインによる計算量の増加を誘発する消去法は不利であるため、反復法を用いる。

反復法の計算では行列・ベクトル積の計算が主となる。これを並列処理する場合、利用する並列性のレベルとして

- 行列の行毎の内積計算を単位とする並列処理
- 各演算を単位とする並列処理

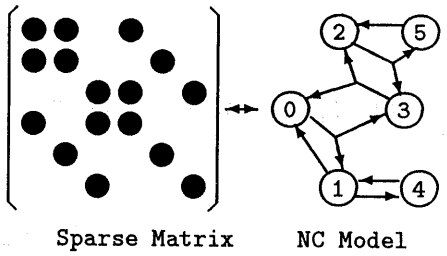

図 1: 反復法の NC モデル (● 行列の非ゼロ要素)

がある。一般に、扱う行列の規模が大きいため行単位の並列性がプロセッサ数に対して十分大きくなり、演算単位での並列性を活かさなくても十分な並列性が得られる。

この場合、並列実行される各プロセスは行列中の 1 行と、ベクトルの内積を計算する。各プロセスによって計算された内積値が新しいベクトルの要素となる。ここで、疎行列性を有効に利用するために、行列中の非零要素に対応する乗算だけを行うとすると、各プロセスは自分の受け持つ行の非零要素に対応するベクトル要素だけを知っていればよい。

このような計算形態では、各プロセスは行列の非零要素に対応するプロセスとだけ交信すればよい (図 1)。反復法では行列そのものは計算中変化しないので、交信相手のプロセスの組合せは計算中静的である。また、1 つのプロセスの計算結果は複数のプロセスに利用される場合が多い。また、プロセスは計算中に動的に生成・消滅することはない。

NC モデル (Node and Connecting-Line Model) は以上の特徴を活かした計算モデルである。

- 計算を実行する各実体をノードと呼ぶ。ノードは計算の始めから終わりまで静的に存在し、動的な生成・消滅は行われない。
- ノード間はコネクティング・ラインと呼ばれる一対多の単方向交信路で結合されている。コネクティング・ラインに対して送出されたデータはそれに結合されている全てのノードに同時に送られる。これをマルチキャストという。
- ノードとコネクティング・ラインの結合は静的である。
- コネクティング・ラインと受信ノードの結合点にはサイズ 1 のバッファが存在する。

- ノード内では複数のコネクティング・ラインからのデータを待ち、それらを到着順に処理することができる（非決定的受信）。

2.2 並行記述言語 NCC

ユーザが実際にプログラムを記述するときは、C言語を NC モデルに基づいて拡張した並行記述言語 NCC(Node-oriented Concurrent C) [5] を用いる。

NCC のプログラムは、生成するプロセス (NC モデルのノードに対応) をその基本属性と共に宣言するプロセス宣言部と、プロセスのインプリメンテーションを記述するプロセス記述部、プロセス間の交信路 (NC モデルのコネクティング・ラインに対応) を手続き的に記述するネットワーク記述部の 3 部から構成される。

ここでは NCC プログラムの例として、ヤコビ法により連立方程式を解くプログラムの一部を示す (図 2)。実際のプログラムでは、この他にネットワーク記述部が加わる。NCC ではプログラムの記述性を高めるために、プロセスやポートを配列で宣言することができる。

3 再構成の必要性

$(SM)^2$-II システムでは、プロセスは各 PU に静的にマッピングされる。ユーザが宣言するプロセスの数が PU 数に比べて非常に大きい場合、1 つの PU に多数のプロセスが割り当てられることになる。現在 $(SM)^2$-II システムでは、1 つの PU に割り当てられた複数のプロセスの管理や通信要求へのサービスを DIPROS [3] と呼ばれるシステム・ソフトウェアが行っている。

一方、NC モデルに基づき NCC で反復法により問題を解くプログラムを記述した場合、プロセスの粒度は小さくなる傾向にある。1 台の PU に多数の小粒度プロセスが割り当てられた場合、頻繁に起こる交信要求に対するサービスとそれに伴うプロセス切替え等をソフトウェアで行うために、システムの有効稼働率が著しく低下してしまう。

ここで多数の小粒度プロセスが 1 台の PU に割り当てられている状態を考える。交信の局所性を考慮してマッピングを行えば、頻繁に起こる交信要求の多くは同じ PU に割り当てられたプロセスに対するものとなる。この場合、データの交換は相手プロセスの変数領域への読み書きだけですんでしまうが、実際には DIPROS を介すため、オーバヘッドが生じる。

```
#define  SIZE     10
#define  INITVAL  0

process{
        inport   in[SIZE], init ;
        outport  out ;      /* プロセス宣言部 */
        main     row_p() ;
}row[n:SIZE](n) ;

row_p(n)
int n ;                  /* プログラム記述部 */
{
    double  a[SIZE], b, x ;
    int i, itr, elsize;

    ireceive(init, &itr) ;
    receive(init, &b) ;
    ireceive(init, &elsize) ;
    for(i = 0 ; i < elsize ; i++)
        receive(init, &a[i]) ;
    x = INITVAL ;
    while( itr -- ){
        send(out, x) ;
        t = 0 ;
        doeach{      /* 非決定的受信構文 */
            for(i = 0 ; i < elsize ; i++){
                entry(in[i],&temp)
                    t += a[i] * temp ;
            }
        }
        x = (b - t)/a[n] ;
    }
}
```

図 2: NCC によるヤコビ法の記述

もしこのような関係にある複数のプロセスを実行前に一つのプロセスにまとめることができれば、そのプロセス間交信はプロセス内の変数代入、参照に置き換えることができ、それに伴うプロセス切替えなどのオーバヘッドも減少する。

そこで、同一 PU 上の複数の小粒度プロセスを 1 つのプロセスに融合するように、ユーザの記述したプログラムを再構成するシステムを考える。これにより、上述したようなオーバヘッドを減らすことができ、その結果計算効率を上げることができる。

4 反復法のための新言語 NCC/i

前節では、ユーザが記述したプログラムを低オーバヘッドで実行するために、再構成する事の必要性について述べた。しかし、一般にユーザが自由に記述した NCC のプログラムを再構成する事は困難である。そこで本節ではまず、再構成が容易なプログラムの構造について述べ、つづいて再構成を行うための解析を容易

にする、反復法を対象とした新たな言語 NCC/i(NCC for Iterative method) について述べる。

4.1 完全反復型プログラム

ここではプログラムを再構成し、小粒度プロセスを一つのプロセスに融合する方法について考える。複数の互いに交信し合うプロセスを一つに融合することは、プロセス間交信を変数への代入と参照に置き換え、各プロセスが実行する手続きを静的にスケジュールし、一つのプロセスのプログラムとしてまとめあげる事により実現される。

この融合の過程で重要なのは、交信の代用としての変数への代入と参照の順序関係の保証である。これを保証するには、融合されるプロセス間で行われる通信の send 文と receive 文の対応関係が実行前の解析時に明らかでなければならない。しかも両者が常に対応してなければ、融合は簡単には行えない。ところが、一般的に交信文が制御構造の中にある場合、交信文の対応関係を実行前に正確に把握することは困難である。

そこで、まずプロセスが融合し易いプログラムのクラスを次のように定義する。

- 交信し合う任意の 2 つのプロセス間でその交信文を含む制御構造が一致している (これを共通制御構造と呼ぶ) 。
- 共通制御構造での制御の流れを決める制御変数の変化が同一である。

以上のような構造を持つプログラムをここでは完全反復型プログラムと呼ぶことにする。

完全反復型のプログラムでは図3のように、ネスト構造のループも許されるが、それぞれのループの反復回数が等しくなければならない。

$(SM)^2$-II が対象としている反復法を用いたプログラムは、基本的に完全反復型に属する。すなわち、各プロセスは関係するプロセスからデータを受け、演算を行った後に新たな自分の値を他のプロセスに送るという処理を、各反復毎に同期をとりながら実行する。そして全てプロセスの反復回数は必ず同じである。

しかし、NCC で反復法のプログラムを記述した場合、たとえそのプログラムが完全反復型に属していたとしても、実行前解析でそれを判定することは非常に困難である。なぜなら、各プロセスの変数は独立なので、たとえそれらのプロセスの制御構造が完全に等しい場合でも、制御変数が異なるため、異なった動作をする可能性があるからである。

```
/* Process 1 */                    /* Process 2 */

for(i=0;i<MAX;i++){                for(i=0;i<MAX;i++){
    ...                                ...
    send(a,x) ;          →             receive(a, &x) ;
    ...                                ...
    for(j=0;j<MAX2;j++){               for(j=0;j<MAX2;j++){
        ...                                ...
        send(b, y) ;     →                 receive(b, &y) ;
        ...                                ...
        receive(c, &z) ; ←                 send(c, z) ;
        ...                                ...
    }                                  }
}                                  }
```

図 3: 完全反復型プログラム

もし、完全反復型であることがプログラム中で明示的に記述されていれば、プログラムの再構成は容易に行える。そこで我々は NCC をベースに、これらを明示的に記述する機能を付加した、反復法を対象とした言語 NCC/i を提案する。

4.2 NCC/i

NCC/i のプログラムは NCC の記述方法を基にしているので、NCC と同様にプロセス宣言部、プロセス記述部、ネットワーク記述部から構成される。NCC/i に求められていることは、完全反復型のプログラムを記述する場合に、そうであることを明示する記述方法を提供することである。

そこで、我々は NCC のプロセス記述部にイテレーション・グループという記述方法を導入した。これは、完全反復型であるプロセス群を明示的に記述するためのものである。

共通制御構造を持つ複数のプロセスは、一つのイテレーション・グループとしてまとめられる。記述方法は、par-for,par-while,par-if などの構文 (これを共通制御構文と呼ぶ) を用いて共通な制御の流れを記述し、各プロセス・タイプに固有の手続きはその中で local 構文を用いて記述する (図 4) 。ただし、共通制御構造は、send や receive 等の交信文を含む制御構造を表現する時にのみ用いれば良い。

さらに制御変数の値の違いによるプロセスの振舞いの差を無くすために、共通制御構文では、その制御変数に対する操作を限定する。すなわち、このような変

```
iteration_group{
    inport  common_port ;  /* 共通入力ポート */
    int     cond = 1 ;     /* 共通制御変数   */

    local PROCESS_A {
        プロセスタイプ PROCESS_A に固有の手続き
    }

    local PROCESS_B {
        プロセスタイプ PROCESS_B に固有の手続き
    }

    par_while( cond ){
        local PROCESS_A {
            プロセスタイプ PROCESS_A に固有の手続き
        }

        local PROCESS_B {
            プロセスタイプ PROCESS_B に固有の手続き
        }
        receive( common_port, &cond ) ;
    }
}
```

図 4: NCC/i プログラムの概略 (プロセス記述部)

数 (共通制御変数と呼ぶ) に対する代入は、local 構文では行うことができないとする。さらに、共通制御変数に対しては、共通制御変数および定数だけからなる式のみの代入が許される。これにより共通制御構文内では、各プロセスの制御の流れが等しいことが保証される。これらの制限により、共通制御変数の変化は全プロセスで等しくなり、ユーザの記述したプログラムの実行前の解析が容易となる。ただし共通制御変数の実体は各プロセスが持つ。(なお、各プロセスはローカルな変数を持つことができるが、それらの参照は local 構文内に限られる。)

しかし、このような共通制御変数に対する操作の制限は、プログラムの解析性を向上させるが、その一方で記述の柔軟性を損なう。例えば、反復法のプログラムでは反復の終了は計算中にデータの収束を検出して行うのが一般的である。しかしこの制限により、あるプロセスがデータの収束を検出した時、共通変数に対する操作が行えないために、反復を終了することができないという問題が生じる。

NCC/i では、解析性を落さずにこの問題を解決する方法として、共通入力ポートを提供している。これにより、あるプロセスが動的に反復を終了させようとした時、この共通入力ポートに終了データを流し、共通入力ポートから受けたデータを共通制御変数に代入すればよい。

```
#define    SIZE    8

/* プロセス宣言部 */
process node {
    inport    north, south, east, west ;
    outport   out ;
    double    mydata, east_data, west_data,
              north_data, south_data ;
}nodep[SIZE][SIZE]() ;

process bound {
    outport   out ;
    double    mydata, indata ;
}bound[4]() ;

/* プログラム記述部 */
iteration_group{
    local node{
        mydata = 1.0 ;
    }

    local bound {
        mydata = 0.0 ;
    }

    par_while(1){
        local node{
            send( out , mydata ) ;
            receive( east,  &east_data ) ;
            receive( west,  &west_data ) ;
            receive( north, &north_data ) ;
            receive( south, &south_data ) ;
            mydata = (east_data+south_data
                      +north_data+south_data)/4.0 ;
        }

        local bound {
            send(out, mydata) ;
        }
    }
}
```

図 5: NCC/i による poisson 方程式の記述

NCC/i のプログラム例として、poisson 方程式の記述例を図 5 に示す。

5 再構成システム PRIME

本節では、NCC/i によって記述されたプログラムを解析し、融合可能なプロセスを融合することによって、できるだけ大粒度のプロセスから成るようにプログラムを再構成するシステム PRIME について述べる。PRIME システムでは、イテレーション・グループによって記述された融合可能なプロセスを融合対象とするため、必ずしも PU 中の全プロセスが 1 プロセスに融合されるとは限らない。このようにして残った複数のプロセスの実行管理は、DIPROS に任される。以下

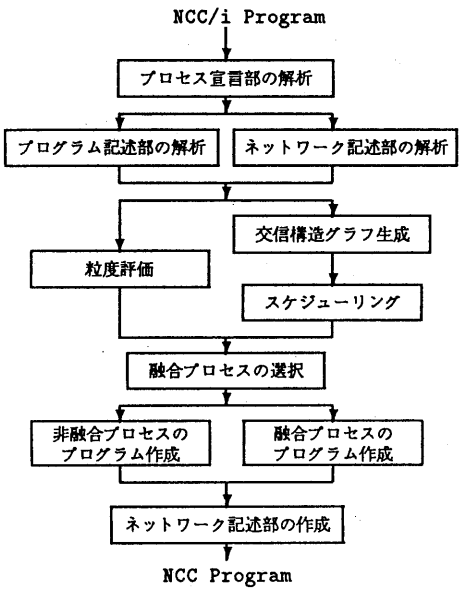

図 6: 再構成システム PRIME の全体構成

PRIME システムの概要について述べる。

5.1 全体構成

PRIME システムは入力として NCC/i のプログラムを与えると、解析・再構成を行い、その結果として NCC のプログラムを生成する。すなわち、PRIME システムは NCC へのトランスレータとして実装される。

PRIME システムの構成を図 6に示す。PRIME システムは入力に対して、プログラムの構文解析等を行い、まず一種のタスクグラフを生成し、それに対してスケジューリングを行い、レベル付けを行う。NCC/i によって記述されているプログラムはプロセス間の依存関係グラフの作成が容易であり、レベルの付けも簡単である。一方、プログラム中で宣言されている各プロセスの粒度を評価し (粒度評価部) 、融合を必要とするプロセスを選び出す。ネットワーク情報とタスクグラフから融合を必要とするプロセスと、融合の組合わせを決定する (融合プロセス選択部) 。

融合する組が決定すると、レベル付けされた依存関係グラフを基に融合が行われ、融合されてできたプロセスのプロセス記述部と、プロセス宣言部が作成される。また、融合に関与しないプロセスは、単純に NCC のプログラムに変換される。さらに、融合により変更

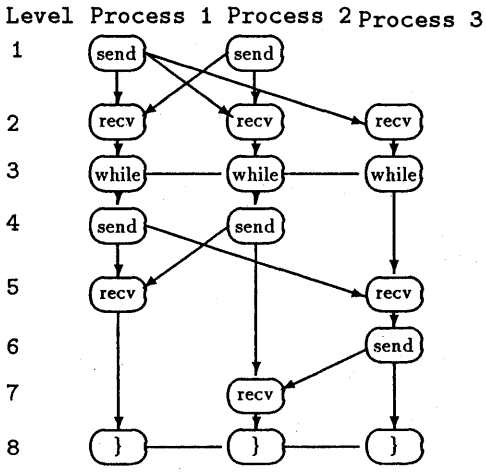

図 7: 交信構造グラフ

された新しいネットワークに対応するネットワーク記述部が作成される。

現在、融合する組を決定するためのアルゴリズムがまだ決定していないために、融合プロセス選択部は未完成である。現状では融合するプロセスの組はユーザが与える様になっている。

以下、プログラムからのグラフ生成、スケジュール、および融合の各過程について述べる。

5.2 グラフ生成

制御の流れと交信による依存関係を示すグラフを交信構造グラフと呼ぶ。ここでは、NCC/i のプログラムから、各プロセスの制御構造と交信文の対応関係および対応する交信文の関係を示す交信構造グラフを生成する過程について述べる。

NCC/i のプログラムでは、複数のプロセスの振舞いを一つのイテレーション・グループという枠組みの中で同時に行う。そこで、まずイテレーション・グループ中の処理を各プロセス別に分離する作業を行う。各プロセス・タイを節とする構造をこの構造をそのプロ各プロセス毎に共通制御構造と交信文をノードとする実行順序グラフを作る。と receive 文の参照関係をグラフに加える。

以上の作業により交信構造グラフ (図 7) が作成できる。

5.3 レベル付けによるスケジューリング

一つに融合される複数のプロセスの持つ手続きを静的にスケジュールするために、交信構造グラフに対するレベル付けを行なう。レベルの付けはプロセスを融合することを念頭に置き、全てのプロセス間交信が変数代入に置き換えられることを仮定して行う (この変数を交信用変数と呼ぶ) 。この時、交信用変数に対する2つの依存関係 (フロー依存関係と反依存関係) を考慮する必要がある。また、交信構造グラフ中の同一の共通制御構造に対しては、全プロセスで同じレベルを付ける。以下にこのレベル割当のアルゴリズムを示す。

1. 各プロセスに初期レベルとして 0 を与える。
2. 各プロセスのルート・ノードを、そのプロセスのカレント・ノードとする。
3. 各プロセスのカレント・ノードへのレベル付けは、ノードの種類により以下の条件が成り立つまで待たされる。

   send ノード (最初の send ノードを除く) そのコネクティング・ラインに対する一つ前の send ノードに対応する全ての receive ノードのレベルが決定している (それらの中の最高値を Lr_{max} とする) 。

   receive ノード 対応する send ノードのレベルが決定している。

   制御構文ノード ('}' ノードを含む) 他の全てのプロセスのカレント・ノードが、その制御構文ノードと同じである。

4. 上記の条件を満たしたカレント・ノードには、以下のレベルが与えられる。ただし、親ノードのレベルを Lp とする。

   send ノード 最初の send ノード:$Lp+1$
   それ以外: $\max(Lp, Lr_{max})+1$

   receive ノード $\max(Lp, Ls)+1$
   (対応する send ノードのレベルを Ls とする)

   制御構文ノード $\max(Lp_1, Lp_2, \ldots, Lp_n)+1$
   (Lp_i は i 番目のプロセスの対応する制御構文ノードの親ノードのレベル)

5. カレント・ノードを一つ先に進め 3. 以降を繰り返す。

5.4 プロセス融合

ここでは、一つに融合されるプロセスの組が決定しているものとして、それらを一つのプロセスに融合し、そのプログラムを生成する過程について述べる。

融合される複数のプロセスのそれぞれの処理を、一つのプロセスとして再構成する場合、

1. 依存関係の保持
2. 変数名、ポート名などの重複回避
3. 交信文の変換

```
start0()
{
    /* process 0 の変数*/
    double      proc0_mydata ;
    double      proc0_east_data, proc0_west_data ;
    double      proc0_north_data, proc0_south_data ;

    /* process 1 の変数 */
    double      proc1_mydata ;
    double      proc1_east_data, proc1_west_data ;
    double      proc1_north_data, proc1_south_data ;

    /* 交信用変数 */
    double      channel0, channel1, channel66 ;

    proc0_mydata = 1.0 ;
    proc1_mydata = 1.0 ;

    while (1) {
        /* send 文の変換 (case B) */
        send(proc0_out, proc0_mydata) ;
        channel0 = (proc0_mydata) ;
        send(proc1_out, proc1_mydata) ;
        channel1 = (proc1_mydata) ;

        /* receive 文の変換 (case B) */
        *(&proc0_east_data) = channel1 ;
        *(&proc1_west_data) = channel0 ;

        /* receive 文の変換 (case D) */
        receive(proc1_east, &proc1_east_data) ;
        receive(proc0_west, &proc0_west_data) ;
        receive(proc0_south, &proc0_south_data) ;
        receive(proc1_south, &proc1_south_data) ;

        /* receive 文の変換 (case C) */
        receive(channel66_inport, &proc0_north_data) ;
        channel66 = *(&proc0_north_data) ;
        *(&proc1_north_data) = channel66 ;

        proc0_mydata = (proc0_east_data + proc0_south_data
                + proc0_north_data + proc0_south_data) / 4.0 ;
        proc1_mydata = (proc1_east_data + proc1_south_data
                + proc1_north_data + proc1_south_data) / 4.0 ;
    }
}
```

図 8: 2 プロセスを融合した例

の3点に注意する必要がある。

1: 交信構造グラフに対するレベル付けは、ノード間の依存関係から生じる各ノードの実行順序を明確にしている。よって、融合するプロセスのノードを、レベルの大小関係を保ちながら融合することによって依存関係を保つことができる。ノード間に存在する実際の処理は、融合されたプロセス中の対応するノードの後ろに挿入される。

2: 融合される複数のプロセス間の変数名やポート名の重複を防ぐために、各プロセスの局所識別子を変更する。

3 融合における交信文の変換には以下の 4 通りがある。

(case A) コネクティング・ラインにつながる send プロセスと全ての receive プロセスが融合される場合。
⇒ 交信用変数を設け、send はその変数に対する代入、receive はその変数の参照に置き換える。

(case B) コネクティング・ラインにつながる send プロセスと、複数の receive プロセスのうちの一部が融合される場合。
⇒ 交信用変数を設け、send は send 文 (融合されない receive プロセスへの送信のため) とその変数に対する代入、receive はその変数の参照に置き換える。

(case C) コネクティング・ラインにつながる複数の receive プロセスが一つに融合される場合。
⇒ 交信用変数を設け、最初に行う receive は receive 文とその変数に対する代入に、残りの receive はその変数の参照に変換する。

(case D) コネクティング・ラインにつながるプロセスが全く融合されない場合 ⇒ 無変換

このようにして融合されたプロセスのプログラムが作成される。また、プロセス宣言部において、融合されたプロセスの宣言を、融合後の新たなプロセスのそれに置き換える。さらに、ネットワーク記述部でも、融合によるネットワークの形状の変化に対応して、ネットワークを記述し直す。

以上の処理により、NCC/i のプログラムの再構成が行え、結果として粒度の大きいプロセスから成る NCC のプログラムに変換される。図8に、融合されたプログラムの例を示す。これは、図5のプログラム中の2つのプロセスを融合して作られたものである。

6 おわりに

現在 PRIME システムのプロトタイプの実装が完了したところであり、今後さまざまなプログラムを作成し評価を行っていく予定である。

また現段階では、融合を必要とするプロセスを選択する粒度評価部と、融合するプロセスの組合せを決定する融合プロセス選択部が未完成である。融合プロセスの選択はプロセスの PU へのマッピングと非常に深い関連があるため、そのアルゴリズムはプロセス間交信の量、PU 間の負荷分散、プロセスの融合 (大粒度化) のしやすさなどを目安として、慎重に決定しなければならない。今後はこのようなアルゴリズムの開発と PRIME システムの未完成部分の作成を行っていく予定である。

謝辞

本システムの設計にあたり、スケジューリング等に関して貴重な助言を頂いた慶應義塾大学電気工学科の凌暁萍女史に感謝致します。また、貴重な御意見を頂いた慶應義塾大学の相磯秀夫教授ならびに安西祐一郎教授に感謝致します。

参考文献

[1] H.Amano, et al., *$(SM)^2$-II: The new version of Sparse Matrix Solving Machine* , Proc. of the 12th International Symposium on Computer Architecture, Jun. 1985

[2] T.Kudoh, et al., *NDL : A language for solving scientific problems on MIMD machines* , Proc. of the 1st International Conference on Super Computing Systems, Dec. 1985

[3] T.Boku, et al., *DIPROS : a distributed processing system for NDL on $(SM)^2$-II* , Proc. of the 20th Hawaii International Conference on System Sciences, Jan. 1987

[4] T.Boku, et al., *IMPULSE : A high performance processing unit for multiprocessors for scientific calculation* , The 15th Annual International Symposium on Computer Architecture, May. 1988

[5] 朴 泰佑 他、マルチプロセッサのための科学技術計算用並行記述言語 NCC 、電子情報通信学会論文誌、Vol.J72-D-I,No.10, 1989-10