ソフトウェア基礎論　37－1
プログラミング言語　27－1
（1990. 12. 13）

並列処理言語ADETRANの実装

若谷　彰良　　森康浩　　佐々木真司　　岡本理
（松下電器産業(株)半導体研究センター）

あらまし　並列処理言語ADETRANは、並列計算機ADENA上での高級言語であり、数値シミュレーションに適した構文を備えている。

ADENAでは、プロセッサ間データ転送の最適化により、データ転送オーバヘッドを極力抑えることが可能である。また、プロセッサ内の演算は、１次元配列データに対するものとなり、通常の最適化に加え、ループアンローリング法の最適化により高速化できる。

本稿では、ADETRANの概要、最適化方法を述べた後、特殊構文であるpour/pad文、及びfile配列の実装方針について述べる。

The implementation of a parallel programming language : ADETRAN

Akiyoshi Wakatani　Yasuhiro Mori　Shinji Sasaki　Tadashi Okamoto
Semiconductor Research Center
Matsushita Electric Industrial Co., Ltd.
3-15 Yagumo-nakamachi, Moriguchi, Osaka, 570, JAPAN
wakatani @ vdrl. src. mei. co. jp

Abstract　ADETRAN is a parallel language on the parallel computer ADENA with some extensions, suited for numerical calculation.

A data transfer time can be extremely reduced with some compiler optimization techniques. Furthermore, new loop un-rolling technique is proposed by taking advantage of characteristic that data types are only scalar and one-dimensional array within a processor of ADENA.

In this paper, we describe the overview of ADETRAN, some compiler optimization techniques and the implementation method of special constructs : pour/pad statement and file array data type.

1. はじめに

近年の半導体デバイスの進歩により、計算機の高速化がますます進んできている。それにともない、応用プログラムの規模も膨大なものとなってきている。

これを代表するのが数値シミュレーションである。その多くはベクトル型計算機によって解かれており、そのためのプログラムは、ベクトル化可能な形のものが用いられる。

しかし、ベクトル型計算機の性能も、そのデバイスの制約等により、高速化の限界が見えはじめており、速度向上のためにマルチプロセッシング技術の導入が行なわれている。

そこで我々はベクトル型計算機のとっている「少数の高価なプロセッサ」を用いた構成とは異なる「多数の安価なプロセッサ」を用いた並列計算機 "ADENA" を京都大学工学部（野木達夫助教授）と共同開発した。[1), 2)]

従来、並列計算機上のプログラミング環境として、"ユーザアプリケーションの記述性が高いこと"と"アーキテクチャの特性を生かすこと"の２つの要求を融合させた専用言語(C＊, MPC)を用いたものがある。[3), 4)]

一方、既存の言語（FORTRAN等）から並列性を抽出する処理系を用いるものや[5)]、既存言語に若干の拡張を行ない、特定のアーキテクチャを目的としない汎用並列言語(C-LINDA等）をめざすものがある。[6)]

後者は、既存のソフトウェアの互換性及び将来へのソフトウェアの継承性の観点からみると重要な方式であるが、現時点でのその並列抽出効率及び演算実行時の性能を考えると実用には向いていない。従ってADENAでは前者の方法、すなわちアーキテクチャに適した専用言語を用いるプログラミング環境を開発した。

ADENAは、数値シミュレーションを主なアプリケーションと想定した並列アーキテクチャである。プログラミング言語の条件は次の様に考える。

1　FORTRANタイプの構文
2　ADENAのアーキテクチャを自然に取り込む
3　数値シミュレーションに適応した構文を持つ

１は、現在のベクトル計算機のユーザは、FORTRANでプログラミングしており、これらのユーザが比較的容易にADENAを使用するためには必要である。

２は、コンパイラの実装を容易にし、かつ、実行時のオーバーヘッドを削減できる反面、若干ではあるが、プログラマにアーキテクチャを意識したプログラミングをすることを要求する。これは、ADENAが高速計算を第一目的におく場合不可避な条件である。具体的には、プロセッサ内のデータ構造を示すスラッシュを用いた配列表現、並列実行を表現するpdo文、プロセッサ間のデータ転送を表現するpass文等がこれにあたる。

３は、従来のFORTRANにはないが、アプリケーションを考えた場合、採り入れる必要のある構文である。

具体的には、行列計算用のpad文及びpour文、粒子シミュレーション用file配列及びその配列のための実行文である。

我々は上記の要件を満たす言語 "ADETRAN" をADENA上に実装した。本稿では、ADETRANの概要、応用例、実装態様について述べる。

2. ADENAシステム

ADENAシステムは、256個のプロセッサエレメントとそれらをつなぐネットワーク(HXネット：hyper-crossnet　従来ADENAネットワークと呼んでいた)とからなるADENAと、汎用ワークステーションをホスト計算機とするバックエンド型計算機により構成される。[7)]

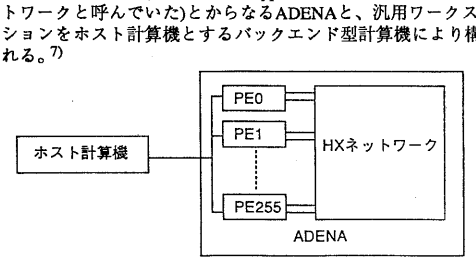


図1 ADENAシステム

プロセッサエレメント（以下プロセッサと呼ぶ）は、ピーク性能20MFLOPSの浮動小数点プロセッサEPU (Element Processing Unit)[8)] とデータ転送用コントローラTCU (Transfer Control Unit)[9)] 及び命令メモリLIM(Local Instruction Memory : 24ビット×64Kワード)とデータメモリLDM(Local Data Memory:72ビット×256Kワード、ECC8ビットを含む)からなる。

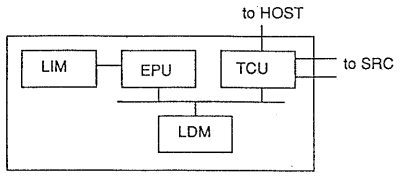


図2 プロセッサエレメント（PE）

EPUは内部に64ビット浮動小数点加減算器及び乗算器を備え、64ビット浮動小数点レジスタを32本、32ビット整数レジスタを32本持つ。さらに、プロセッサを仮想化するための多重度制御機構を持つ。

TCUは、1) ホスト計算機とのデータ転送と、2) SRC (Send Receive Controller)と、LDMとのデータ転送を行なう。SRCとLDMとのデータ転送は、HXネットを使うデータ転送の時に使用される。TCUとEPUは、LDMを共有メモリとして相互にアクセスする。

実行時には、EPUの演算実行と並行してTCUのデータ転送が可能である。特に、EPUが演算結果を配列データとしてLDMに格納し、そのデータをTCUがデータ転送する場合、EPUのLDMのデータの格納と同時にTCUがデータを転送するモード(Sスキームモード)を備えている。

プロセッサ間は、HXネットで接続されている。このネットワークは、バースト転送の制御を行なうSRCとクロスポイントのFIFOメモリであるBMU(Buffer Memory Unit)よりなる。HXネットの論理的な接続は、各プロセッサのプロセッサ番号を/j, k/ (j, k=0～15)とし、BMUのidを/i, j, k/ (i, j, k=0～15)とする時、

PE/j, k/->BMU/i, j, k/->PE/k, i/

となっている。この接続は、双方向になっており、

PE/i, j/->BMU/j, k, i/->PE/k, i/

と見ることもできる。

3. ADETRANの概要

ADETRANは、数値シミュレーションを主アプリケーションとするFORTRANライクな高級言語であり、次の構文を付加したものである。すなわち、並列動作を記述する構文としてpdo文、プロセッサ間のデータ転送のための構文としてpass文、プロセッサ間の同期のための構文としてifall/ifany文である。また、任意長データ構造を表すfile配列、２次元配列用のマルチキャストを行なうpad/pour文を用意している。[10), 11)]

3.1 基本データ構造

ADETRANでは、配列データを操作することを基本とするが、その配列のインデックスにスラッシュを挿入し、担当プロセッサを明示する。例えば、３次元配列aはFORTRANでは、

a(i, j, k)

の様に表すが、ADETRANでは、

a(i, /j, k/) , a(i/, j, /k) もしくはa(/i, j/, k)

の様に表す。a(i, /j, k/)はx方向配列と言いプロセッサ/j, k/が持つ配列aのi番目のデータを表し、a(i/, j, /k)はy方向配列と言いプロセッサ/k, i/が持つ配列aのj番目のデータを表し、a(/i, j/, k)はz方向配列と言いプロセッサ/i, j/が持つ配列aのk番目のデータを表す。

このように、3次元配列は、そのうち2次元をプロセッサアレイにマッピングし、残りの1次元を各プロセッサ内で扱う。同じく2次元配列は、1次元をプロセッサアレイにマッピングし、残りの1次元を演算で扱う。

3.2 region文

ADENAでは、プロセッサの物理的個数を越えた並列動作記述が可能であるが、その場合の最大プロセッサ空間のサイズを示すのがregion文である。つまり、

```
region(21, 32, 43)
```

と記述した場合、x方向配列の場合、最大/32, 43/のプロセッサ空間をとることを表す。

3.3 pdo文

基本データを操作するものとしてpdo文がある。

```
pdo j, k=1, N
do 10 i=1, N
    a(i, /j, k/)=i*1.0+b(i-1, /j, k/)
10  continue
pend
```

スラッシュで囲まれたインデックスがプロセッサ番号に相当するので、各プロセッサが並列かつ独立に動作するためには、囲まれたインデックスは揃っていなければならない。この例では各プロセッサがdoループ10を並列に実行する。同様に、

```
pdo k, i=1, N
do 20 j=1, N
    a(i/, j, /k)=a(i/, j, /k)+j*1.0
20  continue
pend
```

と記述できる。これは、先ほどの例とは異なり、/k, i/がプロセッサ番号を与え、doループ20を並列に実行する。

3.4 pass文

上記2つのpdo文を連続して記述した場合、先のpdo文で定義された配列aは、プロセッサ/j, k/で定義されるが、次のpdo文で参照される配列aは、プロセッサ/k, i/である。従って、配列データaをプロセッサ/j, k/からプロセッサ/k, i/へ転送する必要がある。これを記述するのがpass文である。すなわち、

```
pass i, j, k=1, N
    a(i/, j, /k)=a(i, /j, k/)
pend
```

である。

送り元のプロセッサは/j, k/であり、送り先は、/k, i/である。これは、HXネットワークの構成である。

```
PE/j, k/->BMU/i, j, k/->PE/k, i/
```

と同じ接続である。つまり、HXネットワークは、3次元配列のpass文に対して、効率良く転送できるネットワークであるといえる。

3.5 ifall/ifany文

並列に動作するプロセッサ間でその演算結果にしたがって、制御フローの変更を行ないたい場合がある。例えば、収束するまで繰り返し演算を行なうプログラムなどで収束判定を行なう場合である。その場合、各プロセッサにおいて論理型配列に収束したか否かの情報を代入し、それの論理積もしくは論理和で制御を変える。これを記述するのがifall文／ifany文である。例えば、論理型配列okに収束情報を代入し、

```
ifall (ok (1, /j, k/), j, k=1, N) goto 1000
```

は、全てのプロセッサの論理値okが正であればラベル1000へジャンプする。また、

```
ifany (ok (1, /j, k/), j, k=1, N) goto 2000
```

は、一つ以上のプロセッサの論理値okが正であればラベル2000へジャンプする。

3.6 サブルーチン

サブルーチンの階層を下図に示す。

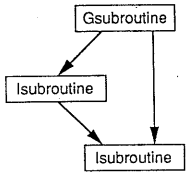


図3　サブルーチン階層

FORTRANから実際に呼ばれるのは、gsubroutineと呼ばれるサブルーチンである。ADETRANにおいては、この他lsubroutineがある。lsubroutineは、pdo文内から呼ばれるサブルーチンで、引数には、1次元配列もしくはスカラ変数がおかれる。

gsubroutine内で記述できるのはregion文、各種宣言文、pdo文、pass文、ifall/ifany文、goto文などである。pdo文の中から外へ、もしくは外から中へはジャンプできない。またホスト／ADENA間のデータ転送を表わすpass文を越えるジャンプは許していない。

3.7 pad文／pour文

以上の構文を使うことによってほとんどの計算は記述可能であるが、一部の行列計算を記述するため若干の構文を追加する。

ガウス消去を用いて逆行列計算を行なう時に、各プロセッサに分散しているデータを集め一つの配列にして全てのプロセッサに再分配する場合（ピボットの選択）がある。これを表すのがpour文である。

また、あるプロセッサの持っているデータを全プロセッサに分配する必要がある場合（ピボット行の分配）に用いられるのがpad文である。これらを図示すると下記のようになる。

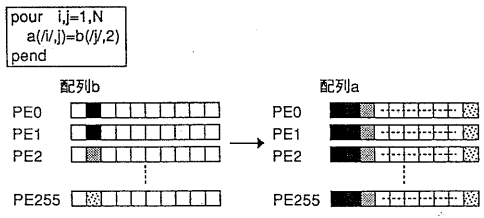


図4-a　pour文

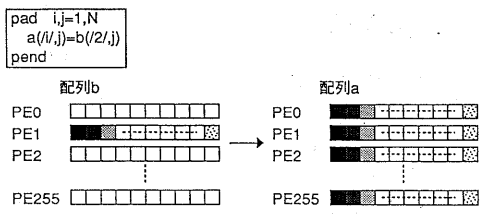


図4-b　pad文

3.8 file配列

粒子プラズマモデルを用いたプラズマシミュレーションは、次の2つのフェーズを繰り返して、計算を進める。

1　プラズマ粒子の位置より電場を計算する。
2　電場を用いて各粒子にかかる力を求め、粒子の位置を進める。

これを並列計算する場合、空間メッシュの各格子点を並列化の単位とし、その格子点に影響を与えるプラズマ粒子をその格子点を担当するプロセッサに持たせる様にする。従って、各格子点の電場を計算し、求まった電場を補間することにより、各粒子にかかる力は求まる。それを用いて次のステップでの粒子の位置が求まり、新たにその粒子を担当するプロセッサが決まる。よって、そのプロセッサへ粒子情報を転送することになる。

これらを実現するデータ構造がfile配列である。file配列は順アクセスのみ可能な任意長のレコードへのポインタの配列である。つまり複数情報からなる粒子情報をレコードとして持ち、そのレコードは、各点に対して任意個持つことができる。
例を示す。

```
file/pp(//,)/ii, jj, uu, vv
```

上記の例は、整数フィールドiiとjj、実数フィールドuuとvvを持つレコードを任意個蓄えられる配列を表す。

これを扱う構文として4つの実行文と転送文がある。実行文は、file配列を再書き込み可能とするためのrewrite文、再読み出しするためのreset文、書き込みを行なうput文、読み出しを行なうget文がある。またfile配列を転送するためのpush文である。

file配列は粒子シミュレーションを行なうために設けられた構文群であるが、粒子シミュレーション以外にも、疎行列のリストベクトル等にも使える。

4. 応用例

ADETRANで記述した代表的なプログラム例を示す。

4.1 Splitting Up

ポワソン方程式の解法に対し、反復解法の一つであるCG法を用いたアルゴリズムを示す。解くべき方程式を

$$Ax=b$$

とおく場合、反復法は下のようになる。

＜アルゴリズム＞

x_0: 任意の出発値
$r_0=b-Ax_0$
$\boxed{r'_0=M^{-1}r_0}$
$p_0=r'_0$
$k=0$
while $r_k \neq 0$ do
  begin

$$a_k=\frac{(r_k, r'_k)}{(p_k, Ap_k)}$$

$x_{k+1}=x_k+a_kp_k$
$r_{k+1}=r_k-a_kAp_k$
$\boxed{r'_{k+1}=M^{-1}r_{k+1}}$

$$\beta_k=\frac{(r_{k+1}, r'_{k+1})}{(r_k, r'_k)}$$

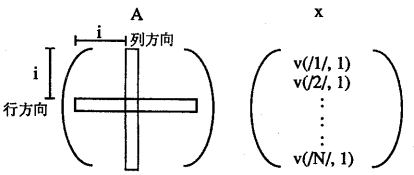
$p_{k+1}=r'_{k+1}+\beta_k p_k$
$k=k+1$
  end
$x=x_{k+1}$

これは加速するために行列Mを用いている。ADENA向きの加速法として空間方向に応じた作用素の因子分解型のMを用いたSplitting Up法があり、それに対するコードを下に示す。

```
      gsubroutine solv
      region (32,32,32)
      real a(*,/*,*/),b(*,/*,*/),c(*,/*,*/),r(*,/*,*/),t(*,/*,*/)
                :
      pdo j,k=1,32
            call trid(t(,/j,k/),a(,/j,k/),b(,/j,k/),c(,/j,k/),
     +       r(,/j,k/),imax)
      pend
      pass i,j,k=1,32
            r(i/,j,/k)=t(i,/j,k/)
      pend
      pdo k,i=1,32
            call trid(t(i/,,/k),a(i/,,/k),b(i/,,/k),c(i/,,/k),
     +       r(i/,,/k),jmax)
      pend
                :
      return
      end
c
      lsubroutine trid(t,a,b,c,r,imax)
      real t(1),a(1),b(1),c(1),r(1),ti(1)
      real tl(10000),tm(10000)
c
      tl(imax) = 1/a(imax)
      tm(imax) = tl(imax)*r(imax)
      do 13 i = imax-1,1,-1
        tl(i) = 1/(a(i)-c(i)*tl(i+1)*b(i+1))
 13     tm(i) = tl(i)*( r(i)-c(i)*tm(i+1))
      t(1) = tm(1)
      do 14 i = 2,imax
 14     t(i) = -b(i)*tl(i)*t(i-1)+tm(i)
      return
      end
```

4.2 疎行列の積演算

正方疎行列Aと一次元ベクトルxとの積を考える。



疎行列なので行方向iの内容は、

nni(/i/, 1)　非ゼロ要素の総数
nnl(/i/, 20)　非ゼロ要素の位置
aav(/i/, 20)　非ゼロ要素の値
※非ゼロ要素は20以下と仮定

を用いて表す。

また、行列計算を行なうために、非ゼロ要素に対応したv(/j/, 1)を格納する必要がある。それを

```
nnv(/i/,20)
```

とおく。また、列方向iの非ゼロ要素位置は、自データを転送すべき転送先であり、それを

anni(/i/, 1)　転送先総数
annl(/i/, 20)　転送先位置
※対称行列では、nniとanni及びnnlとannlは一致する。

とおく。以上の配列とfile配列を用いて記述する。

```
C
C       matrix a * vector x
C
        dimension pp(/*/,1),qq(/*/,1)
        file /pp,qq/ v,iloc,idst,isrc
        :
        pdo i=1,N
                rewrite ($pp(/i/,1))
                do 10 j=1,anni(/i/,1)
                        $pp(/i/,1).v    = v(/i/,1)
                        $pp(/i/,1).idst = annl(/i/,j)
                        $pp(/i/,1).isrc = i
                        $pp(/i/,1).iloc = l
10                      put ($pp(/i/,1))
        pend

        push(qq(/idst/,iloc)=pp(/i/,j),i=1,N,j=1,1)

        pdo i=1,N
                reset ($qq(/i/,1))
                do 20 j=1,nni(/i/,1)
                        if(eof($qq(/i/,1)) call error
                        call jsrch(nnl(//,),$qq(/i/,j).isrc,jj)
                        nnv(/i/,jj)=$qq(/i/,1).v
10                      get ($qq(/i/,1))
        pend

        pdo i=1,N
                v(/i/,1) = 0.0
                do 30 j=1,nni(/i/,1)
30                      v(/i/,1)=v(/i/,1) + aav(/i/,j)*nnv(/i/,j)
        pend
```

5. 実装方針

5.1 概要

ADETRANをADENAに実装するために、ホスト計算機上でコンパイル・リンクできるクロス環境を構成した。ほとんどの部分をC言語で作成し、構文解析部にはYACCを使用した。[12]

ADETRANのロードモジュールは、実行前にホスト計算機よりADENAへダウンロードされる。

実行時には、ADETRANのgsubroutineがFORTRANから呼ばれ、ADENA内でそのサブルーチンボディが実行される。

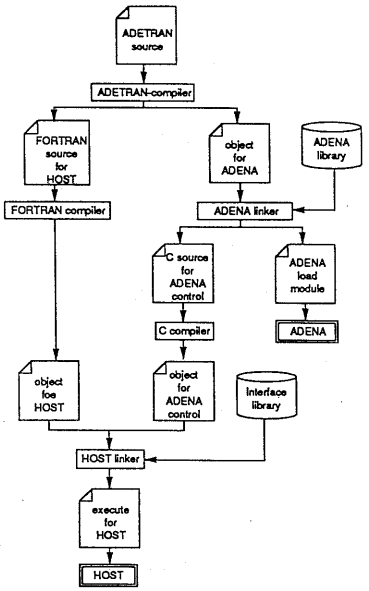


図5　コンパイルフロー

gsubroutineは、前述の通りpdo文、pass文等から構成されるが、ホスト計算機上のFORTRANから見た場合は、

1　ADENA上での演算実行
2　ホスト計算機からADENAへのデータ転送
3　ADENAからホスト計算機へのデータ転送

の3種類の構文に分けられる。つまり連続するpdo文、ifall/ifany文、goto文、ADENA内でのデータ転送を表すpass文等は、1つの構文にまとめ、ホストからは、その1つの連続した実行文に対して起動をかけるだけである。

つまり、図6に示すgsubroutineは、

```
gsubroutine  sub(u,v)
parameter(N=32)
region(N,N,N)
  :
pass  i,j,k=1,N
  u(i,/j,k/)=u(i,j,k)
pend
pdo  j,k=1,N
  :
pend
pdo  j,k=1,N
  :
pend
pass  i,j,k=1,N
  v(i,j,k)=v(i,/j,k/)
pend
return
end
```

図6　プログラム例

```
sub_(u,v)
int  *u,*v;
{
  snd(....);
  fpustarta(...);
  while(gethalta(...))
    ;
  rcv(....);
}
```

図7　変換後Cプログラム

図7に示すCプログラムに変換され、FORTRANのcall文で呼ばれることになる。

コンパイル方法は、下記の3パスからなる。

1　上位構造及び下部構造それぞれの中間言語を作成する（構文解析）
2　上位構造の組換え、及び、下部構造内の最適化を行なう（最適化）
3　コード生成する（コード生成）

中間コードにおける上位構造とは、gsubroutineを構成するpass文やpdo文を表す木構造である。また、下部構造とは、pdo文内部の実行文を表す4つ組コードである。例を下に挙げる。

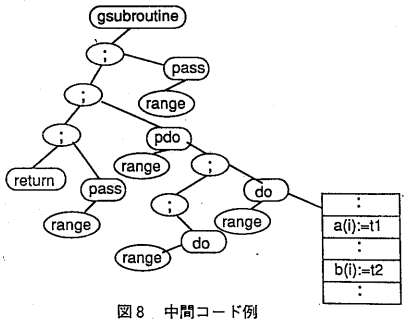


図8　中間コード例

上位構造は、第2パスにおいて、passの最適化を行なうために組換えを行なう。また、下部構造は、その内容、スカラ変数及び1次元配列を扱う通常のFORTRANの構文になっているので、ループ最適化などの従来の最適化処理、及び、5.3で扱うループアンローリング等の最適化処理を行なう。その後、第3パスでジャンプの飛び先アドレスの計算を行なった後、オブジェクトコードの生成を行なう。

5.2 仮想プロセッサ

gsubroutine内のregion文によって、仮想プロセッサ空間が設定される。つまり、

region (NX, NY, NZ)

の時、x方向仮想プロセッサ空間は、/NY, NZ/であり、y方向及びz方向仮想プロセッサ空間は、/NZ, NX/及び/NX, NY/である。従って、実際のプロセッサ空間（物理プロセッサ空間）との対応は、図9のようになり、各プロセッサを複数回（多重度）使用する。また、メモリも、多重度毎に持つ。それぞれの多重度に対するメモリにアクセスするためにベースポインタを使い、よって、各多重度の実行は同一コードになる。

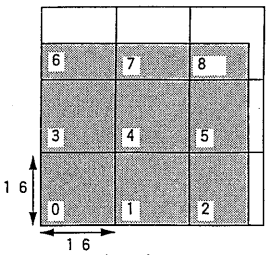


図9　仮想プロセッサ空間
region　(20,40,44)のx方向

pdo文に対する実際の動作は、pdo文のインデックスの範囲（pdoレンジ）と、pdo文内の実際の演算実行文（pdoボディ）によって決まる。例えば、先ほどの仮想プロセッサ空間とpdoレンジの関係は、図１０のようになる。

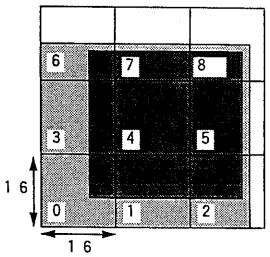


図１０　pdoレンジ
pdo　j=4,38,k=12,42

従って、多重度１の場合、ベースポインタはBで、物理プロセッサ空間で動作するのは、

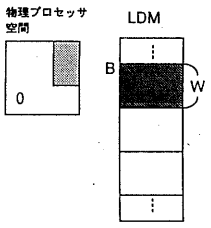


図１１　ベースポインタ１

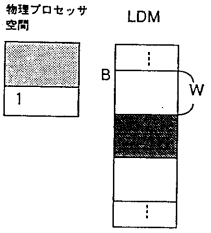


図１２　ベースポインタ２

図１１のようになる。
また多重度２の場合は、ベースポインタはB+Wで、物理プロセッサ空間で動作するのは、図１２のようになる。以下同様の動作となり、この多重度切替えで必要となるのは、

1　メモリのベースポインタの切替え
2　動作プロセッサの選択

となる。
これらは、物理プロセッサ空間全体に関する動作であるが、ホストから制御するのはオーバーヘッドも多く、あまり好ましくない。従って、今回のADENAでは、各プロセッサは、自プロセッサ番号(id)を用いて、分散制御する。

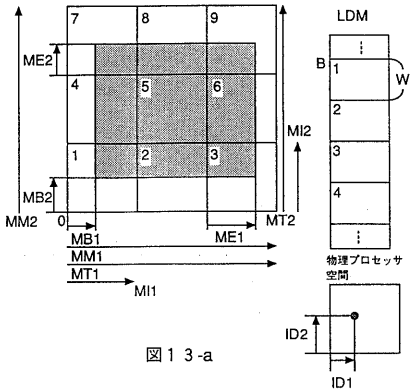


図１３-a

MLTS

```
Base=B;
for  (MC2=0;MC2<MM2;MC2++){
  for  (MC1=0;MC1<MM1;MC1++,Base+=W){
    if(MC2<MI2  ¦¦  MC1<MI1
    ¦¦  MC1>MT1  ¦¦  MC2>MT2)
      ;
    else  if(MC2==MI2  &&  ID2<MB2)
      ;
    else  if(MC1==MI1  &&  ID1<MB1)
      ;
    else  if(MC2==MT2  &&  ID2>ME2)
      ;
    else  if(MC1==MT1  &&  ID1>ME1)
      ;
    else
      execute();
}
```

図１３-b　MLTS

上記のようなルーチン（MLTSルーチン）を通し、pdoボディを実行する。
また、pdoボディに入るところでベースポインタを設定する。

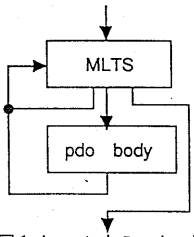


図１４　pdo文のマクロ構造

pdo文に対しては、おおむね上記のようなコードを生成しpdoボディ内の構文に関しては、通常のコンパイラ処理を行なう。
上記のように、仮想プロセッサ動作のための分散制御を行なうことによる。長所として２点挙げることができる。

1　ホストよりの操作が減り、実行時のオーバーヘッドが減る。
2　実際の並列度が上がり、実行速度が速くなる。

１については前述の通りだが、２について若干説明する。
例えば、region文で仮想プロセッサ空間が、

region(32, 32, 32)

と設定してあり、pdo文として、

```
pdo j=16, 31, k=16, 31
    u(1, /j, k/)=1.0
pend
```

を実行するとする。
これを、ホスト制御によって実行すると、

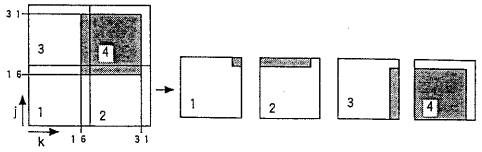


図15　ホスト制御の場合

のように4多重分の実行時間がかかる。つまり、1多重目は、j=16, 16, k=16, 16であり、2多重目は、j=16, 16, k=17, 31であり、3多重目は、j=17, 31, k=16, 16であり、4多重目は、j=17, 31, k=17, 31である。

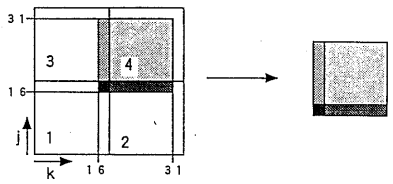


図16　分散制御の場合

しかし、分散制御すると、各プロセッサエレメントは、MLTSルーチンは、4回通るが、pdoボディには1回しか通らない。つまり、実行時間としては、ホスト制御する場合の25%で済むことになる。

5.3　最適化[13)]

5.3.1　Sスキーム

並列計算機において、その演算効率を下げる要因としてデータ転送時間の存在がある。データ転送は逐次計算においては存在しなかったが、並列計算においては、異なるプロセッサでその計算結果を交換する必要がある。ADENAでは、演算の為にEPU、転送の為にTCUと、異なるプロセッサでそれぞれの動作を行なっている。
実際は、データ転送を行なう場合、

1　EPUが、送信を開始するアドレス、ワード数、受信を開始するアドレス、転送のパターン等のパラメータをTCUにセットする
2　TCUをスタートさせる
3　TCUの転送状態をモニタする

のステップで行なわれる。上記のステップの内、2と3の間は、EPUの動作としては特に規定していない。つまり、EPUは、TCUが転送動作を行なっている間、演算実行を行なうことができる。これにより、データ転送は見かけ上EPUの演算実行の裏に隠れることになり、演算効率は高くなる。
では、どのような場合に上記のような処理が可能であろうか。転送データの種類によって、場合分けを行なう。

1　pass文の前のpdo文で送信すべき配列をモディファイしていない
2　pass文の前のpdo文で送信すべき配列をモディファイする

条件1の場合、pass文とその前のpdo文には、依存関係が無いので、その実行順序は可換である。従って、先程の例のように、TCUを予めスタートしておいて、その間にpdo文を実行し、その後にTCUの終了判定を行なうようにすることができる。
しかし、条件2の場合pass文とpdo文には依存関係があるので、その実行順序を変えることはできない。一般にはpdo文で計算した結果を次のpdo文で使うため、pass文を使用することが多いと考えられる。そこで、条件2の場合でもある一定の条件を満たしている場合は、条件1と同様な転送ができることが望ましい。その条件とは、

pass文の転送範囲とpdo文及びdo文の代入範囲が等しい

である。

ハードウェアとしては、

EPU：LDMへのデータの格納と同時にTCUへシグナルを送る機構
TCU：シグナルを受ける毎に決められた順にデータを1ワードづつ送る機構

を付加する。
上記の条件により、第3の条件ができる。

3　pass文の前のpdo文で送信すべき配列をモディファイするが、それぞれの動作範囲が等しい

この条件3の場合、EPUの実際の演算実行としては、

1　EPUが、送信を開始するアドレス、ワード数、受信を開始するアドレス、転送のパターン等のパラメータをTCUにセットする。ただし、シグナルを受けて動作するモードにする。
2　TCUをスタートさせる。
3　pdo文の処理を行ない、転送に対応するデータへのストアは、シグナル付きにする。
4　TCUの転送状態をモニタする。

となる。つまり、文単位の依存関係をデータ単位の依存関係に細分化することにより、モディファイドデータのバースト転送が可能となる。この転送モードのことを、以後、Sスキームと呼ぶ。これは、代入(Substitute)と送信(Send)が同時に行なわれることを表す。
Sスキームを用いた転送の効果を下に示す。

三重対角ソルバ

```
pdo   j,k=1,64
  call  trid(...)
pend
pass  i,j,k=1,64
  p(i/,j,/k)=p(i,/j,k/)
pend
```

上記のプログラムを58回繰り返す。

三重対角ソルバ計算部分=t0	0.1568sec
データ転送部分=t1	0.0330sec
Sスキーム=t2	0.1643sec

t1=0.0330sec
t2-t0=0.0075sec
となり、約22.7%に減る。

図17　Sスキーム

このようにSスキームによりデータ転送オーバヘッドは減らすことができることがわかる。
次に、条件3について考察する。実際のプログラムにおいて、条件3を満たしている頻度は、必ずしも高くないと考えられる。その要因を挙げ、条件3への変形可能性を検討する。

1　代入する範囲と転送する範囲が異なる
2　転送すべき配列の内、代入していない配列がある
3　条件ifにより代入が静的には不明

まず1の場合であるが、それぞれの範囲が一致するようにソースコードを変形する。

```
pdo   j,k=1,N
  do  10  i=2,N
10  a(i,/j,k/)=b(i-1,/j,k/)
pend
pass  i,j,k=1,N
  a(i/,j,/k)=a(i,/j,k/)
pend
```

→

```
pdo   j,k=1,N
  a(1,/j,k/)$$=a(1,/j,k/)
  do  10  i=2,N
10  a(i,/j,k/)$$=b(i-1,/j,k/)
pend
pass  i,j,k=1,N,$$
  a(i/,j,/k)=a(i,/j,k/)
pend
```

$$は強制的にSスキームにすることを示す

図１８　変形１

次に２の場合であるが、代入文を追加して配列数を合わせる。

```
pdo   j,k=1,N
  do  10  i=1,N
10  a(i,/j,k/)=u(i,/j,k/)
pend
pass  i,j,k=1,N
  a(i/,j,/k)=a(i,/j,k/)
  b(i/,j,/k)=b(i,/j,k/)
pend
```

→

```
pdo   j,k=1,N
  do  10  i=1,N
    a(i,/j,k/)$$=u(i,/j,k/)
10  b(i,/j,k/)$$=b(i,/j,k/)
pend
pass  i,j,k=1,N,$$
  a(i/,j,/k)=a(i,/j,k/)
  b(i/,j,/k)=b(i,/j,k/)
pend
```

図１９　変形２

最後に、３の場合であるが、else節に対応する代入文を付加する。

```
pdo   j,k=1,N
  do  10  i=1,N
    if(x(i,/j,k/).gt.1.0)then
      a(i,/j,k/)=u(i,/j,k/)
    endif
10  continue
pend
pass  i,j,k=1,N
  a(i/,j,/k)=a(i,/j,k/)
pend
```

→

```
pdo   j,k=1,N
  do  10  i=1,N
    if(x(i,/j,k/).gt.1.0)then
      a(i,/j,k/)$$=u(i,/j,k/)
    else
      a(i,/j,k/)$$=a(i,/j,k/)
    endif
10  continue
pend
pass  i,j,k=1,N,$$
  a(i/,j,/k)=a(i,/j,k/)
pend
```

図２０　変形３

以上のように多くの転送はSスキームに変形することができるが、不用な代入文等が発生するので、これらの変形はプログラマの指定で制御できる。

5.3.2　ループアンローリング

pdo文内は、１次元配列を扱う通常のFORTRANプログラムと同じなので、従来のコンパイル手法が用いられる。よって、最適化手法も従来のものが用いられるが、アプリケーションの特性によって次のような考察ができる。すなわち、pdo文内には通常、doループ内では、１次元配列とスカラー変数を用いた計算が多く、しかもそのインデックスが+1や-1だけ違うようなものの計算が多い。

例えば、２階微分の中心差分を計算するには、

```
u(i+1, j)-2*u(i, j)+u(i-1, j)
+u(i, j+1)-2*u(i, j)+u(i, j-1)
```

を求めることになる。このx方向部分だけを通常のコンパイリングした場合を疑似コードで示す。

```
x(i)=u(i-1)-2*u(i)+u(i+1)

load   fr0, (#u+i-1)
load   fr1, (#u+i)
fmul   fr2, fr1, #2.0
load   fr3, (#u+i+1)
fsub   fr4, fr0, fr2
fadd   fr5, fr4, fr3
store  (#x+i), fr5
```

上記のコードが、iに関するdoループ内で実行されている場合、u(i)は、ループを全て実行する間に３回ロードすることになる。つまり、u(L)は、i=L-1の時に、u(i+1)としてロードし、i=Lの時u(i)としてロードし、i=L+1の時u(i-1)としてロードする。

これらのロードの無駄を減らすために、元のコードを３回アンローリングする。それに対し、レジスタをサイクリックに割り当てる。

```
x(i   )=u(i -1)-2*u(i   )+u(i+1)
x(i+1)=u(i   )-2*u(i+1)+u(i+2)
x(i+2)=u(i+1)-2*u(i+2)+u(i+3)

load   fr0, (#u+i-1)
load   fr1, (#u+i)

fmul   fr2, fr1, #2.0
load   fr3, (#u+i+1)
fsub   fr4, fr0, fr2
fadd   fr5, fr4, fr3
store  (#x+i), fr5

fmul   fr2, fr3, #2.0
load   fr0, (#u+i+2)
fsub   fr4, fr1, fr2
fadd   fr5, fr4, fr0
store  (#x+i), fr5

fmul   fr2, fr3, #2.0
load   fr1, (#u+i+3)
fsub   fr4, fr3, fr2
fadd   fr5, fr4, fr1
store  (#x+i), fr5
```

上記のように、従来７命令に対し３命令のロードが必要であったものが、１５命令に対し３命令で済むことになる。約40%の速度上昇が予想できる。方法をまとめると、

1　インデックスのディスプレースメントの範囲を求める
2　ディスプレースメントの範囲でアンローリングする
3　同一配列を同一レジスタに割り当てる

となる。すなわち、従来のループアンローリング手法であるが、そのアンロール回数を上記のように設定することにより、より効果的なものとなる。

5.4　pour文／pad文

pour文／pad文は、前述のように一般行列の逆行列計算で主に用いられるものであり、２次元行列に対するもののみある。現処理系では、両構文共に、pdo及びpassの構文に変換して実行するようにしている。

まず、pour文であるが、これは、各プロセッサ（仮想プロセッサ）の１ワードを連結したものを全プロセッサへ分配するものである。したがって、下記のように全プロセッサ分のコピーを行ない、それを２次元転送する。

```
      [pour]
      pdo  i=1, N
             do  10  j=1, N
10             t1(/i/, j)=b(/i/, 2)
      pend
      pass  i, j=1, N
             t2(i, /j/)=t1(/i/, j)
      pend
      pdo   i=1, N
             do  20  j=1, N
20             a(/i/, j)=t2(j, /i/)
      pend
```

最後のpdo文は、一般ユーザでは使用できない構文であるが、同一プロセッサ内での代入であるので、コンパイラのマクロ出力からのみ許しているものである。

次に、pad文であるが、これはあるプロセッサの配列を全部に分配するものである。まず、ソース側配列を転送する。そして、コピーすべき部分を全配列に対しコピーする。さらに、再度転送する。

```
      [pad]
      pass  i, j=1, N
            t1(i, /j/)=b(/i/, j)
      pend
      pdo   j=1, N
            do  10  i=1, N
10          t1(i, /j/)=t1(2, /i/)
      pend
      pass  i, j=1, N
            a(/i/, j)=t1(i, /j/)
      pend
```

以上のように、他のADETRAN構文を用いる実現法を取っているが、通常のpass文の約2.5～3.0倍の時間がかかる。そこで現在、TCUのコマンド設定を変えることによりpass文と同一位の時間で実行できるように検討中である。

5.5 file配列

file配列は、順アクセスのみ可能な任意長のレコードへのポインタの配列である。図示すると下記のようになる。

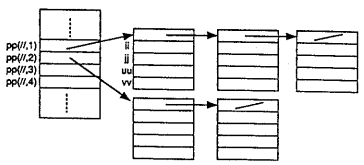


図２１　file配列の概念図

実行は、書き込みを行なうために、書き込み用ポインタをリセットするrewrite文、書き込みを行なうput文、読み出しポインタをリセットするreset文、読み込みを行なうget文で行なう。また、実際の計算は、各配列要素毎にウインドウを設定し、そこで行なう。したがって、実際の配列構造は、

- レコードリストの先頭ポインタ（head）
- ウインドウ上で有効なレコードのポインタ（current）
- 次のレコードのポインタ（next）
- ウインドウ（レコードの内容を保持する：数ワード）

となる。

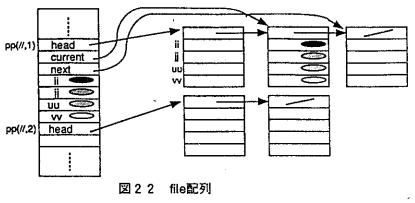


図２２　file配列

以上のようなポインタを用い、各実行文を処理する。

まず、rewrite文は現在つながっているレコードリストをフリーリストに返し、それぞれのポインタを初期化する。

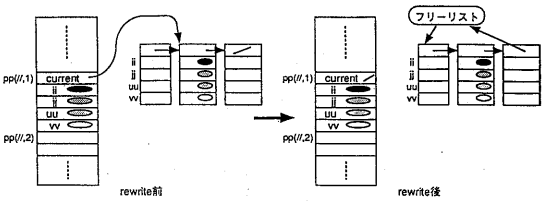


図２３　rewrite文

次にput文は、ウインドウ上の内容レコードリストに書き込み現在のリストの次につなぐ。

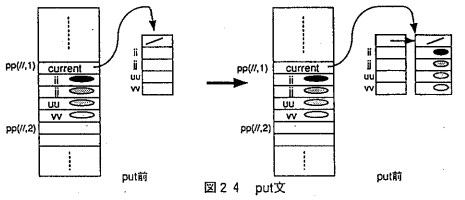


図２４　put文

reset文は、currentをheadにし、その内容をウインドウに取り込む。

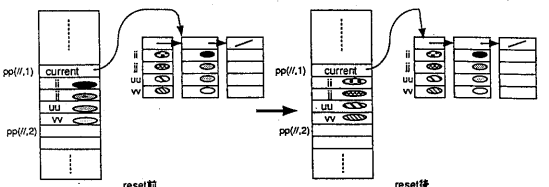


図２５　reset文

最後にget文は、nextの指すレコードをウインドウに取り込む。この時、取り込むレコードがない場合はcurrentをNULLにする。

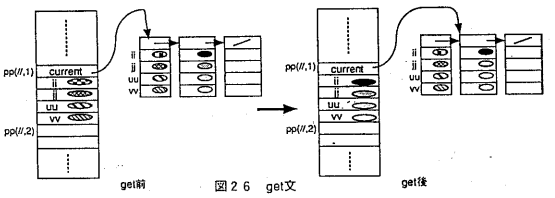


図２６　get文

file配列は、プラズマ粒子シミュレーションにおいてはプラズマ粒子そのものである。従って、場の計算が終わると場の力によって、他のプロセッサの管理する空間へ移動する。これを表現するpush文は次のサブルーチンによって実現される。

```
push(a(/ii/,jj)=b(/i/,j),i,j=1,N)
      push転送は、任意間転送なので２回転送になる。
        PE/i,j/->PE/j,k/->PE/k,m/
  call  rcvt（レシーブの起動）              １回目の転送
  call  push01-pdo
              do
                reset
                while(~e of){
                  call  pusht1（レコード転送）
                  get
                }
  call  psynch（同期）
  call  rcvt                                ２回目の転送
  call  pusht2
  call  psynch
  call  push02-while(~eo f)                 レコードリストにつなぐ
                put
```

図２７　push文

6. おわりに

並列処理言語ADETRANの概要、応用例及び実装について述べた。ADETRANは、数値シミュレーションを行なうユーザが"素直に"並列処置が記述でき、かつADENAにおいて高パフォーマンスを得ることを意図して設計したものであり、その目的は果たしている。

今後は、出力コードのより一層の最適化を図るとともに記述性を高める言語拡張を検討していく予定である。

謝辞

本言語処理系の開発にあたり、終始御助言、御討論いただいた京都大学 野木達夫 助教授、半導体研究センター超LSIデバイス研究所 廉田 主幹技師、および本研究の機会を与えていただいた半導体研究センター超LSIデバイス研究所 間野 所長に感謝致します。

参考文献

[1] T. Nogi : Parallel computation, patterns and waves-qualitative analysis of nonlinear differential equations., P.279-318(1986).

[2] 谷川裕二 他 : 並列計算機ADENA、情報処理学会計算機アーキテクチャ研究会報告、CPSY88-11、P33-40(1989).

[3] P. Christy : Software To Support Massively Parallel Computing on the MasPar MP-1, Proc. IEEE Compcon Spring 1990, (1990)

[4] John R Rose, et al. : C*: An Extended C Language for Data Parallel Programming, Thinking Machines Technical Report PL87-5, (1987)

[5] M. J. Wolfe : Optimizing Supercompilers for Supercomputers, Ph. D thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, Report No. UIUCDCS-R-82-1105, 1982

[6] N. Carriero et al. : How to Write Parallel Programs : A Guide to the Perplexed, ACM Comput. Surv. Vol.21, No.3, P.323-357 (1989)

[7] H. Kadota, et al. : VLSI parallel computer with data transfer network : ADENA., Proc. ICPP, P.319-322(1989).

[8] K. Kaneko, et al. : A VLSI RISC with 20-MFLOPS peak, 64-bit floating-point unit., IEEE, Journal of Solid-State Circuits 24, No. 5, P.1331-1340(1989).

[9] Y. Nakakura, et al. : A versatile data transfer control unit for a parallel processor system., Symposium on VLSI Circuit, P.15-16 May(1989).

[10] 野木達夫 : 並列言語ADETRAN、第一回数値流体力学シンポジウム公演論文、（文部省重点領域研究『数値流体力学』）、P.343-346(1987).

[11] T. Nogi : Parallel programming language ADETRAN. memoirs of the Faculty of Engineering, Kyoto University, 51(4)(1989).

[12] 若谷彰良 他 : 並列計算機用言語ADETRAN処理系の開発 情報処理学会第36回全国大会

[13] 若谷彰良 他 : 並列計算機ADENAに対する最適化コンパイルの一手法、情報処理学会計算機アーキテクチャ研究会報告、89-ARC-79(1989).