2P−9

画像処理用超並列プロセッサAMPのプログラミング言語Valid-Aについて

山元規靖　堀田正利　谷口倫一郎　雨宮真人

九州大学総合理工学研究科

1. まえがき

本稿では，画像処理用自律型非同期超並列プロセッサAMP[1]のプログラミング言語 Valid-A について述べる．AMP では多数の PE を格子状の通信ネットワークで結合し，各 PE をデータ駆動制御を用いて非同期に動作させる．これにより，柔軟で高能率な並列処理を実現しようというものであるが，そのプログラミング言語である Valid-A は，この特性を損なわず，分かりやすく柔軟なプログラミングが可能であるように設計されるものである．

2. Valid-A 言語仕様

Valid-A は，データフローマシン用の記号処理向け関数型言語 Valid[2]を基としており，これに AMP 上での画像処理のために，関数配列の概念や PE 間通信，ピラミッド構造の高能率なマッピングなどを可能にする構文が加えられている．また，オリジナルの Valid では記号処理のため，リストや構造体データの操作のための演算が用意されていたが，Valid-A では画像処理を基本に考えているので算術・論理演算のみをサポートする．以下では，Valid に準ずる構文は省き，Valid-A で新しく加えられた構文について述べる．

2.1 定義

値，型，関数，マクロの諸定義は Valid に準ずる．

(1) 物理 PE 数の定義

```
define PE_size 100:100;
```

物理 PE の数は必ず定義しなければならない．

(2) 小規模配列の定義

Valid-A では，関数が参照する小規模の配列データをサポートしており，コンパイラは対応する物理 PE のオペランドメモリーに割り当てる．この配列データは，物理 PE 内の複数の論理 PE で使われるために，基本的には read-only のルックアップテーブルとして用いることを想定しているが，更新も可能である．

定義：Aname = array(size [, 初期値])

参照：aread(Aname,i)　(省略形 Aname[i])

更新：awrite(Aname,i,y)　(signal を返す)

上記の定義において，Aname は配列データの名前であり，size はその配列の大きさである．初期値として all(x) と書くと配列の要素が全て x で，省略すると零で初期化される．参照では配列 Aname の i 番目の要素が参照され，更新では i 番目の要素に値 y が入り，signal が返される．この配列データの更新については，旧いデータに対する読み出しが終了したことを確認した後に書き込みを行う．

(3) 関数配列の定義

Valid-A では，複数の論理 PE で実行される関数を記述するために，関数の配列表現を用いる．この際下記のように if 文により各 PE に異なる関数を定義することが可能である．

```
array_of_function fname[256][256](x,y:integer)
return(integer);
map fname[i][j] on PE[i/8][j/8];
for each function[i][j] { .....
if (i>M  &  i<N) fname = function definition;
..... }
```

上の例において，3行目の map...; は関数配列（論理 PE）の物理 PE 上へのマッピングであり，省略するとコンパイラが自動的に割り当てる．この例では，一つの物理 PE に 8×8 の論理 PE が割り当てられる．

(4) ピラミッド構造の定義

多重解像度を利用する画像処理の構造であるピラミッド構造[3]において，AMP では下位の層とそれに対応する上位の層が同じ物理 PE 内にあれば，層間の通信が効率よく行われる．Valid-A では層間の対応関係を示すために mapped from という記法を用い，例えば，下位の 2×2 画素が上位 1 画素に対応する 3 層のピラミッド構造は次のように記述する．

```
array of function level0[256][256](x,y);
map level0[i][j] on PE[i/8][j/8];
array of function level1[128][128]
            mapped from level0[2][2];
array of function level2[64][64]
            mapped from level1[2][2];
```

2.2 式

式は，全て Valid に準ずる．ただし，再帰式は末尾再帰型のみ認められ，コンパイラによりループに展開される．

2.3 PE 間通信

(1) 送信・受信

送　信：send($data_1$,...,$data_m$,DEST.LP)

受　信：receive(LP)

送受信：send_and_wait($data_1$,...,$data_m$,DEST.LP)

送信は，DEST という論理 PE にある受取口 LP に $data_1$ から $data_m$ までを転送するもので，受信構文により LP という受取口からデータを受け取る．送受信は，データを転送した後それに対する返信を受け取るものである．

(2) リモート変数

DEST.variable

リモート変数については次節で詳しく述べる．

Programing language Valid-A for AMP,Autonomous Multi-processor for Image processing

Noriyasu YAMAMOTO, Masatoshi HOTTA, Rin-ichiro TANIGUCHI, and Makoto AMAMIYA

Graduate School of Engineering Sciences, Kyushu University

3. リモート変数

ある論理 PE が他の論理 PE にあるデータを互いに参照する際，Valid-A では send 命令，receive 命令といった通信プリミティブを用いて陽に記述する．しかし，実際には下記の Program 1 の例のように，send 命令や receive 命令の羅列になり，通信ポート名を誤って用いるとデッドロックに陥る恐れがある．そこで，このような場合，プログラマーが通信ポート名を考慮せず簡単に記述できるようにリモート変数という概念を導入した．リモート変数は**参照するデータがある論理 PE 名 (DEST) . 参照する変数名 (variable)** という構文で表され，参照するデータを値として持つ read-only の変数として扱われる．実際には，コンパイラにより通信ポートが管理された send 命令と receive 命令のペアに変換される．Program 2 は Program 1 をリモート変数を用いて記述したものであるが，最終的には Program 1 と同じコードとなる．

— Program 1 [エッジ検出] ————

```
Edge[i][j](x) =
{let !=send(x,Edge[i-1][j].N),
     !=send(x,Edge[i-1][j+1].NE),
     !=send(x,Edge[i][j+1].E),
     !=send(x,Edge[i+1][j+1].SE),
     !=send(x,Edge[i+1][j].S),
     !=send(x,Edge[i+1][j-1].SW),
     !=send(x,Edge[i][j-1].W),
     !=send(x,Edge[i-1][j-1].NW),
     center=x+receive(N)+receive(S),
     left =receive(E)+receive(NE)+receive(SE),
     right=receive(W)+receive(NW)+receive(SW),
 in if (center-right) > (center-left)
    then center-left  else  center-right    }
```

— Program 2 [リモート変数使用例] ————

```
Edge[i][j](x) =
{let center= x+Edge[i-1][j].x+Edge[i+1][j].x,
     left  = Edge[i-1][j-1].x+Edge[i][j-1].x
           +Edge[i+1][j-1].x),
     right = Edge[i-1][j+1].x+Edge[i][j+1].x
           +Edge[i+1][j+1].x,
 in if (center-right) > (center-left)
    then center-left  else  center-right    }
```

4. プログラミング例

すでに Program 2 においてリモート変数を用いたエッジ検出の例を示したが，実際の画像処理に際しては，Program 3 のようにメインとなる関数が，画像の読み込み，平滑化，二値化，細線化といった処理を行う関数を呼び出して一連の画像処理を行うことができる．

Program 4 は，全画素に一回の細線化処理を施す関数 thinning の例を示している．前半は Comp と neighbor のマクロ定義を行っている．Comp は，ある画素の周囲 8 方向の画素の値を 8 ビットのビットパターンとしてその示す値を返すもので，画素の周囲の状況を示す値となる．関数本体には，この値に対応して画素が消去可能かどうかのデータを持つ配列 Mask が定義されている．neighbor は，リモート変数を用いて，周囲の画素と x の値をやり取りするものである．関数本体では，最初にマクロ neighbor を用いて周囲の画素の値を取り込む．そして，マクロ Comp と配列 Mask のデータを用いて，消去可能な画素は値を 0 として消去し一回の細線化処理を行う．

— Program 3 [メインプログラム] ————

```
function main(signal)
 return(signal) =
{let x0=read_image(device),          画像の読み込み
     x1=for (y,i) init (x0,0) do     繰り返し平滑化
         if (i=4) then return(y)        (4回)
         else recur(smooth(y),i+1),
     h =histo(x1),                    ヒストグラム
     t =threshold_value(h),           しきい値の決定
     x2=binarize(x1,t),               二値化
     x3=eliminate(x2,10),             小領域の削除
     x4=thin(x3)                      細線化
 in display(x4)   }                   結果の表示
```

— Program 4 [細線化] ————

```
array of function thinning[256][256](pix)
return(integer)
for each function[i][j]{
  /* Macro definition */
macro Comp(x1,x2,x3,x4,x5,x6,x7,x8)
 return(integer) =
  x1<<7|x2<<6|x3<<5|x4<<4|x5<<3|x6<<2|x7<<1|x8,
macro neighbor(x)
 return(y1,y2,y3,y4,y5,y6,y7,y8) =
  {let x1 = thinning[i][j-1].x,
       ...................
       x8 = thinning[i-1][j-1].x
   in (x1,x2,x3,x4,x5,x6,x7,x8)  },
  /* Main body of thinning program */
if (i>0 & i<255 & j>0 & j<255) then
thinning[i][j](pix) =
  let Mask = array(256,0,1,0,1,.....),
      (x1,x2,x3,x4,x5,x6,x7,x8) = neighbor(x),
      y = Mask[Comp(x1,x2,x3,x4,x5,x6,x7,x8)],
      (z1,z2,z3,z4,z5,z6,z7,z8) = neighbor(y),
  in if (y=0) then 0
     else Mask[Comp(z8,z7,z6,z5,z4,z3,z2,z1)]  }
```

5. むすび

本稿では，画像の処理と理解のための自律型非同期超並列プロセッサ AMP でのプログラミング言語 Valid-A について述べ，そのプログラミング例を示した．

今後は，この言語仕様に基づいた Valid-A のコンパイラの作成と，Valid-A で記述されたプログラムによる AMP の性能評価のシミュレーションが必要である．また，市販の共有メモリ型マルチプロセッサシステムで Valid-A のプログラムを実行するためのコンパイラ等の開発も進めている．

参考文献

[1] 谷口，雨宮：“画像処理と理解のための自律型非同期超並列プロセッサ AMP”，「画像理解の高度化と高速化」シンポジウム講演論文集，pp.53–58(1989)．

[2] 長谷川，雨宮：“データフローマシーン用関数型高級言語 Valid”，電子情報通信学会論文誌 D，Vol.j71-D，No.8，pp.1532–1539(1988)．

[3] 谷口，河口：“画像処理のためのデータ構造”，別冊 O puls E 画像処理アルゴリズムの最新動向，pp.73–83(1986)．