グラフィクスとCAD 66−11
(1993. 12. 16)

超並列処理記述言語Vによる画像処理アルゴリズムの記述

谷口 倫一郎　日下部 茂　雨宮 真人
{rin, kusakabe, amamiya}@is.kyushu-u.ac.jp
九州大学総合理工学研究科

〒816 春日市春日公園6−1

あらまし

本稿では，超並列処理記述言語Vを用いた並列画像処理アルゴリズムの記述法について述べる．Vは実装する並列計算機のアーキテクチャを極力意識させないように設計したものであり，様々な並列アルゴリズムを容易に記述するための枠組みを提供することを目的としている．Vは関数型をベースにした言語であるが，関数型だけでは記述性が十分でないと考え，並列オブジェクト指向の導入を試みている．ここでは，いくつかの例題記述を通して言語の概要を示すと共に，Vの並列画像処理アルゴリズム記述への有効性を示す．

和文キーワード　並列画像処理　超並列処理　並列プログラミング言語

A Massively Parallel Programming Language V and Its Application to Description of Image Processing Algorithms

Rin-ichiro Taniguchi, Shigeru Kusakabe and Makoto Amamiya
{rin, kusakabe, amamiya}@is.kyushu-u.ac.jp
Graduate School of Engineering Sciences, Kyushu University

6-1, Kasuga-koen, Kasuga, Fukuoka 816 JAPAN

Abstract

In this paper, we describe a massively parallel programming language V and its application to description of parallel image processing algorithms. V is designed so as to be independent from target machine architectures as much as possible, and to provide a framework for programmers to describe a wide variety of parallel algorithms easily. V is based on the functional language framework and augmented by introducing the idea of the parallel object-oriented system. Here, we show the outline of the language by using several examples and its effectiveness for the description of parallel image processing algorithms.

英文 key words　Parallel Image Processing　Massively Parallel Processing　Parallel Programming Language

1．はじめに

画像処理やコンピュータビジョンにおける問題の１つは，情報量の多さに起因する計算速度の点にあることは従来より指摘されている点であるが，最近は画像の解像度が高くなってきており，またその処理も複雑になってきているため，画像の高速な処理に対する要求がますます高まって来ている．このような背景から近年，様々な高速画像処理システムが開発されてきている[1]．これらのほとんどは，並列処理技術を導入することによって高速化を達成しているが，更に高速化，高機能化を進めるためには以下のような問題点を解決する必要がある．

(1)適用可能性の高いアーキテクチャ

多くの画像処理システム（特に商用のシステム）は画像の低レベル処理を高速に実行することを目的としていたため，畳み込み演算（主にエッジ検出や平滑化等）を始めとする局所処理を高速化するラインスキャン型のパイプラインプロセッサがほとんどであった．しかし，画像理解などの高機能の処理を高速に実現するためには，そのような構成のアーキテクチャでは機能的に不十分であり，大局的処理や（準）記号的処理が実現できるようなアーキテクチャが必要である．

(2)並列性の高いアーキテクチャ

パイプラインプロセッサをベースとした画像処理プロセッサは局所処理のみを実現していたため，比較的小規模のハードウェアで高速な処理を実現することが出来ていた．しかし，(1)の目標を実現するためには，比較的高機能のプロセッサを多数用いた高並列のシステムが必要になってくる．

このような観点から，我々は画像理解のための超並列処理アーキテクチャに関する研究を進めており，データフローをベースとした画像処理用超並列プロセッサAMPとそのプログラミング言語Valid-Aを開発している[2,3]．超並列処理においてはハードウェアアーキテクチャも重要であるが，多数のプロセッサをいかに効率良く動作させるかという点でソフトウェア，特にプログラミング言語が重要な役割を担ってくることは明らかである．そこで現在我々は，Valid-Aやそのベースとなったデータフローマシン用関数型言語Valid[4]の開発で得られた知見を基に，新たな超並列処理記述言語Ｖの開発を始めている[5]．Valid-A等が，その実行対象マシンのアーキテクチャをある程度意識していたのに対し，Ｖは汎用性を高めるため，極力実マシンを意識しないでプログラムを記述できるようにするという考えに基づいて設計している．Ｖ自体は画像処理専用の言語ではないが，その重要なアプリケーションとして画像情報処理を念頭に置いている．

本稿では，特に画像処理の並列プログラミングという点に焦点を当て，我々が開発を進めている超並列記述言語Ｖの概要と，Ｖ言語を用いた画像処理アルゴリズムの記述について述べる．

2．超並列記述言語Ｖ

2.1 Ｖの概要

Ｖ言語はValidをベースにしたものである．逐次処理言語に並列実行のための機構を付加するという方法で設計した言語ではなく，依存関係のないものは全て並行動作することを前提とした言語である．すなわち，計算の進行する順序を規定するのは依存関係だけである．このようなモデルに基づくと，計算の進行に従い，頻繁に同期が必要となるが，データフロー同期機構により同期は自動化されるため，プログラマが明示的に同期を指定する必要はない（ただし，必要な場合は明示的に計算順序を指定することも出来る）．データフロー同期機構では，ある計算に必要な値がまだ求まっていない場合は，その値が決まるまで実行が中断し値が求まった時点で自動的に同期がとられ計算が再開する．

Ｖ言語は単一代入則をとっており変数は手続き型言語の変数とは異なる．手続き型言語では変数は値の格納場所であって計算の進行にしたがって格納されている値は変化するが，Ｖ言語では変数は値の格納場所ではなく，いったん値が求まると後で変わることはない．また，Ｖ言語はブロック構造による静的な可視範囲を持つ言語である．Ｖ言語はコンパイル指向で変数には静的に型が付けられる．

Ｖ言語では並列オブジェクト指向記述が行なえる言語仕様を導入している．通信を行ないながら式を評価していく計算体（agentと呼ぶ）のインスタンスを生成することで，並列に動作するプロセスをつくり出すことができる．それらagentインスタンスの間ではストリームによってデータをやり取りしながら計算を行なう．

ストリームは，データが次々と流れる一種の通信路とみなすことができ，送信側では，受信側とは非同期に通信路に次々にデータを流し，受信側では送信順にデータを読みとって計算を進める．データが到着していなければ読みとり側のプロセスは待たされる．ストリームを通じて通信するプロセス間の同期もデータフロー同期機構によって行なわれる．

ストリームの要素をメッセージとみなせば並列オブジェクト指向のメッセージ交換と考えることも出来る．受信側ではストリームのheadで先頭データを見ることが出来，tailにはストリームの続きが入っている．処理内容がメッセージだけで決定するなら，Validの枠組の中での関数呼び出しで済むが，ここでagentを導入したのは，状態を持つ並行プロセスを導入することが目的である．agentの中で評価する式が，何らかの再帰によってカプセル化された内部状態を保持しながら，入力ストリームを次々に処理していくものであれば，そのagentのインスタンスは履歴依存オブジェクトと見なし得る．計算モデルとしては，実行順を決定するのは依存関係だけであり，実マシンを考慮した明示的な並列展開法やマッピングの記述を言語自体は持っていない．実装に依存した並列実行の指定は別途，注記（annotation）によっておこなう．依存関係に沿った半順序関係を満たす範囲内であれば，計算の順序は結果に影響しない．annotationによって影響を受けるのは実行効率のみである．ここでは，Validに準ずる部分は省き，agentに関連する事項について簡単に述べる．

2.2 Vにおける並列オブジェクト

V言語では，agentテンプレート（定義）からインスタンスを生成することによって，通信機能と計算能力を持つプロセスを作り出すことができる．agentのインスタンス同士は依存関係がない限り全く並列に動作する．agentインスタンスは入力と出力のストリームを持つことができ，agentインスタンス間の通信はストリームを通じて行なう．後述するように，インスタンス間の依存関係が確定している場合は，インスタンス間でストリームを結合し通信することができる．同一agent定義から生成されたインスタンスは共通の動作記述を持つが，各インスタンス毎に固有のコンテクスト，内部状態を持つ．動作記述はValidで認められている範囲内の式，および主に以下のことを行なうために拡張した式を用いて構成する．

・agentインスタンス生成
・agentインスタンス間通信
・ストリーム操作

agentインスタンス生成時には引数を渡すことが可能で，内部コンテクストおよび初期状態の値は，引数の値を用い定義に従って計算される．生成時に渡される引数に依存するよう記述したコンテクストおよび初期状態は，各agentインスタンス毎に固有のものとなる．

[通信]

agentインスタンスは，お互いの間でストリームを通じてデータをやり取りしながら計算を進める．送信側ではストリームに次々にデータを流し，受信側ではストリームからデータを読みとって計算を進める．インスタンス間の同期は，参照しているデータが到着していなければ読みとり側は待たされ，データが到着すると待ちが解かれ処理が進むデータフロー同期（メッセージ駆動）方式によって行なわれる．ストリームの要素をメッセージとみなせばメッセージ交換による並列オブジェクト指向と考えることが出来る．ストリーム通信において受信側インスタンスの内側から見ると単に入力のストリームが見えるだけだが，送信側から見ると大別して以下の2通りの通信形態がある．

sendタイプ：通信のたびに（その時点で通信可能なインスタンスの中から）agentインスタンス名を指定してその入力ストリームにデータを送る．受信側では入力ストリームに到着順にマージされる．尚，ここではこのタイプのものは取り扱わない．

stream結合タイプ：agentインスタンス間で生産者／消費者などの依存関係が固定的に決定している場合，agentインスタンスの出力ストリームを他のagentインスタンスの入力ストリームに結合して通信することが出来る．出力側でその出力ストリームに値を送れば結合先では入力ストリームの要素として読み出すことが出来る．この結合により，対象問題の持つ通信のトポロジーを明示的に反映した記述をおこなうことができる．

[内部状態]

agentインスタンス内の処理内容がメッセージだけで決定可能であるなら，Validの枠組内の関数呼び出しで済む．agentを導入したのは，並行に動作し内部状態を持ったプロセスと，それらの間の自由度の高い通信を容易に記述できるようにすることが大きな目的である．定義に値を与えて計算実体を動的に生成するという点で，agentインスタンスと関数インスタンスは共通しているが，agentインスタンスは内部状態を持つという点で異なっている．

単一代入則を遵守してagent内で履歴依存の内部状態を保持するために，何らかの再帰的記述で，明示的に状態値をフィードバックする必要がある．内部状態の更新は，一般に，インスタンスのその時点の内部コンテクス

トにおいて，状態値と入力ストリームから読み出したデータを用いて，定義中に記述された式を評価することによって行う．

[agent結合]

先に述べたように，agentインスタンスの出力ストリームを他のagentインスタンスの入力ストリームに結合し，各agentインスタンス間の通信（構造）トポロジーを表すことが出来る．

[agent集合体]

V言語では同一定義を持つagentインスタンスの集合体の記述ができる．また，並列処理においては“データ（オブジェクト）の論理的構造＋通信のトポロジー＋物理的マッピング”が重要であるので，インスタンス集合に論理的な構造を持たせることができるようにする．規則的な構造を持つ集合の要素インスタンス間をストリーム結合すれば，固定的通信のトポロジーを実現し，論理的な構造の上で同期を取ったりするなどデータ並列的な処理が行なえる．

2.3 並列オブジェクト記述の構文

[agent定義]

agentは以下のように定義する．

```
agent <agent定義名>(<生成時仮引数部>)
<ストリームインターフェース部>
= <式>;
```

生成時仮引数部は，agentインスタンス生成時にインスタンス毎に固有の値を渡すためのもので，その変数名はインスタンス内で局所的に有効である．ストリームインターフェース部は

```
{ channel <ストリーム引数要素1>;
          <ストリーム引数要素2>;...}
[{ export <ストリーム引数要素1>;
          <ストリーム引数要素2>;...} ]
```

のように入力ストリームの変数を指定するchannel変数部と，出力ストリームの変数を指定するexport部からなる．V言語はコンパイル指向であるためストリーム要素も型の指定がなされる．

本体の式には正しい式であれば任意の式が可能だが，歴依存オブジェクトを記述するためには

・インスタンス内の環境，状態値を初期化
・履歴依存の状態値を保持
・ストリームを着順に処理

を行なうため，生成時仮引数を通して渡された値によりインスタンス内の環境，状態値を初期化する式と，状態値とストリームを再帰的に処理する再帰式を含む式を書く．新しいagentインスタンスの生成や，出力ストリームに対する操作（メッセージ送信）なども行なわれる．

[agentインスタンス生成]

agentのインスタンスを生成し名前をつけるには次のような記法を用いる．

```
create(<agent定義名>[,式の並び])
```

createはagentインスタンス生成のための既定義関数で，式の並びとagent定義の生成時仮引数は数，型とも一致している必要がある．agentインスタンスは変数名を指定して名前をつけることができる．また，channel変数宣言における変数名並びの順番は意味を持つ．疑似変数chiによってagentが持つ複数のストリームのうち，（channel変数宣言の左から）i番目のストリームを特定できる．

[メッセージ送信・受信]

V言語ではagentインスタンス間でデータの生産者／消費者の関係が明らかでそれらの間で規則的な通信を行なう場合，出力ストリームexport変数を他のインスタンスの入力ストリームchannel変数に結合することによって問題構造を反映した通信トポロジーを明示できる．このように結合されたストリームに対してはexport変数側に値を出力することで結合先のchannel変数に渡される．

```
put(<export変数>,<式>)
```

export変数の結合先に複数の送信先を指定しておくことで，マルチキャストも可能である．

一方，受信であるが，agentインスタンスの本体式では一般には何らかの再帰記述により，状態を保持しながらストリーム要素を次々に再帰的に処理する．agentインスタンスでchannel変数を通じて，処理できるメッセージストリームは受信順であり，それ以外の順序でメッセージを処理したい場合はagent内の局所変数にと

り込んで明示的に処理手順を記述する必要がある．ストリームに対するプリミティブ演算は以下のものがある．

・head(<channel変数>)
・tail(<channel変数>)
・empty(<channel変数>)

headはストリームの先頭を返し，tailは残りを返す（リストデータに対するcar，cdrと同じ）．ストリームは，tail部が確定していなくてもhead部は参照できるようなリスト構造で．channel変数要素への参照はデータが書き込まれるまでサスペンドし，データが書き込まれると待ちが解かれ処理が続けられる．また柔軟な同期の記述を行なうためデータが到着していないことを判断するemptyがある．もしchannel変数が空ならtrue，そうでなければfalseを返す．

agentインスタンスの本体で，ストリームの要素一つずつを取り出して処理する場合は

```
for(m) init (x) -- xはchannel変数
 ...
 ...F(head(m)) -- 先頭要素を読み出して処理
 ...
recur(tail(m)) --先頭を取った残りを次フェーズに
```

といったように先頭要素を読み出しては，残りを次のフェーズに渡せばよい．例えばある条件の時は先頭要素を取らずにそのままにしたいときは，その条件が成り立つ時はrecur(m)とストリームをそのまま次のフェーズに渡せはよい．

[agent集合体]

V言語では同一定義を持つagentインスタンスの集合体であるfieldインスタンスの生成ができる．fieldの定義，生成は以下のように行う．

```
field <field定義名>(<agent定義名>)(<仮引数部>)
        <ストリームインターフェース部>
        (<インデクス指定部>)[on <構造テンプレート>]
= <設定式>

<変数> = organize(<field定義名[, 式の並び])
```

一つのfieldインスタンスは同一のagent定義のagentインスタンスから構成される．オブジェクト並列プログラミングで並列効果がもっとも得られるのは規則的な構造にagentインスタンスの集合を構成できる場合と考え，agent集合の論理的構造を指定して記述できるようにした．構造テンプレートはfield要素のインスタンスの論理的な構造を指定するもので，現在mesh，tree，hypercube等が指定できる．各要素インスタンスにはインデクスが付けられる．インデクス指定部は，<変数>:<式>のようにインデクス変数とサイズを指定する．ここで，変数がインデクス変数，式がサイズになる．meshの場合はこの組みを次元数だけ指定する．

また，単一のagentと同様にfieldインスタンス間の入出力のためのストリームインターフェースを定義することができる．設定式では，各要素インスタンスの初期設定やストリームの結合を記述する．

[ストリームの結合]

```
link(<export変数@インスタンス名>,
     <channel変数@インスタンス名>,
     <channel変数@インスタンス名>,...)
```

ストリーム要素の生産者側をexport変数@インスタンス名に，消費者側をchannel変数@インスタンス名に指定してストリームを結合することができる（export変数とchannel変数は型が一致している必要がある）．通信相手を指定することにより，プログラムの可読性も高まり，実行時にそのトポロジー情報を利用した最適配置などができ，通信効率をあげることが期待できる．

尚，これらの構文に基づいた実際のプログラム例については3.2で示す．

3．V言語による画像処理アルゴリズムの記述

3.1 並列処理からみた画像処理アルゴリズム

画像処理の並列アルゴリズムは，データのアクセスパターン（並列マシンではプロセス間の通信パターンと考えることもできる）や同期の取り方などの点でいくつかのクラスに分類することができる．そして，その各クラスに対応したプログラミング技法が存在する．

このような観点で分類した場合，基本的な画像処理のクラスとしては以下のようなものが挙げられる．

・点演算
・局所演算
・反復型型局所演算
・頻度分布生成演算

・分割統治型演算
・バタフライ演算
・多重解像度処理
・動画像処理†

これらに加えて，３次元画像処理や画像理解の範疇まで対象を広げると更に多くの種類のアルゴリズムが存在する．ここでは，紙面の都合上，全てについて細かくアルゴリズムの記述法を述べることはできないので，２，３の代表的なもののみを取り上げ，Ｖ言語の特徴を示すことにする．

3.2 アルゴリズムの記述例

ここでは，超並列処理のモデルとして１つの画素を１つのagentに対応させて処理を行う例を考える．すなわち画像が１つのagent集合に対応することになる．例えば近傍処理の場合，agent同士が通信を行うことによって近傍画素の値を獲得し，自画素の値を決定する．

まず簡単な例として，３×３の４-近傍で単純平均による平滑化をM回繰り返す例を考える．ここでは，入力画素の値をagentに対する初期状態と考え，agentが計算を停止したときの最終状態を出力画素の値として受け取るというスタイルを用いている．まず，要素となるagentは以下のように定義できる．このプログラムで，for以下が繰り返しを表現しており，until(count>=M)で繰り返し回数を制限し，この条件が成立すると最終状態vを出力画素としてagentから返すようにしている．

```
agent average(val:integer)
{channel N,S,E,W: integer}
{export out: integer}
={for (v,count,n,s,e,w)
      init (val,0,N,S,E,W)
      until(count>=M) return(v) do
   recur(next,count+1,tail(n),tail(s),tail(e),
         tail(w))
     { where
         next =
          (head(n)+head(s)+head(e)+head(w))/5,
         !=put(out,v)};
```

†上述のような視点で画像処理を分類していく場合，動画像処理を１つのクラスとして考えるのは現実的ではない．本稿では簡単な動画像処理の例も取り上げるが，そこでは単に画像データの系列が入力され，出力が得られるまでのデータの取り扱いの概要を示すに留める．これらの処理の場合も更にデータアクセスパターン（空間的パターンならびに時間的パターン）や同期の取り方等の点から細かく分類する必要がある．

以上のようなagent集合体をmesh構造（図１）に生成するには，fieldの設定式の記述が以下のようになる．

```
field Average(average)(val)(i:n,j:n) on mesh
=
{ foreach (i) in 1..n do
     if i=1 then
      foreach (j) in 1..n do
       if j=1 then
         link(ex1@[1,1],ch1@[n,1],ch2@[1,2],
                        ch3@[2,1],ch4@[1,n])
        elsif j=n then
         link(ex1@[1,n],ch1@[n,n],ch2@[1,1],
                        ch3@[2,n],ch4@[1,n-1])
        else
         link(ex1@[1,j],ch1@[n,j],ch2@[1,j+1],
                        ch3@[2,j],ch4@[1,j-1])
     elsif i=n then
      foreach (j) in 1..n do
       if j=1 then
         link(ex1@[n,1],ch1@[n-1,1],ch2@[n,2],
                        ch3@[1,1],ch4@[n,n])
        elsif j=n then
         link(ex1@[n,n],ch1@[n-1,n],ch2@[n,1],
                        ch3@[1,n],ch4@[n,n-1])
        else
         link(ex1@[n,j],ch1@[n-1,j],ch2@[n,j+1],
                        ch3@[1,j],ch4@[n,j-1])
     else
      foreach (j) in 1..n do
       if j=1 then
         link(ex1@[i,j],ch1@[i-1,1],ch2@[i,2],
                        ch3@[i+1,1],ch4@[i,n])
        elsif j=n then
         link(ex1@[i,n],ch1@[i-1,n],ch2@[i,1],
                        ch3@[i+1,n],ch4@[i,n-1])
        else
         link(ex1@[i,j],ch1@[i-1,j],ch2@[i,j+1],
                        ch3@[i+1,j],ch4@[i,j-1])
}
```

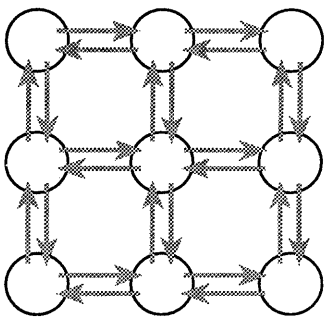

図１ Mesh fieldのイメージ

agentの概念はデータの流れを取り扱う問題を素直に表現することができるため，動画像処理の記述にも極めて有効である．すなわち，画像の時系列をストリームとして表現し，動画像処理をこのストリームを処理するagentとして定式化できる．ここでは，ストリームとして入ってくる画像系列の各画像フレームの雑音を除いた後エッジを検出して，エッジ検出された動画像を出力するという例題を考える．（簡単のため，ここでは状態を利用しない例を用いるが，本質的な動画処理には状態の概念が必要である．）

この例では，前の例と異なり，入力画像，出力画像ともfieldのチャネル（下の例ではinput,output）を通して与えられるものとする．また，雑音除去，エッジ検出のためのagent集合体を用意し，それを連結することによって雑音除去，エッジ検出を連続して行うという形で実現する．

```
-- 雑音除去のagent単体の定義
agent smooth()
{channel C, N, S, W, E, NW, NE, SW, SE: integer}
{export RESULT, OUT: integer}
-- C:自画素（フィールド外から入力する）
-- その他:周囲の８近傍の値（フィールド内の入力）
-- OUT:自画素を近傍に通知するためのポート
-- RESULT:処理結果の値（フィールドの外へ通知する）
= {for (c, n, s, w, e, nw, ne, sw, se)
       init (C, N, S, W, E, NW, NE, SW, SE)
      until(error_occurs()) return(error_code())
    do recur(tail(c),tail(n), .........)
      where
          result = -- 雑音除去後の値
          !=put(result, RESULT),
            -- 結果をフィールドの外へ
          !=put(head(c), OUT)
            --  自画素の値を近傍へ
}

--  エッジ検出のagent単体の定義
agent edge()
{channel C, N, S, W, E, NW, NE, SW, SE: integer}
{export RESULT, OUT: integer}
= {for (c, n, s, w, e, nw, ne, sw, se)
       init (C, N, S, W, E, NW, NE, SW, SE)
      until(error_occurs()) return(error_code())
    do recur(tail(c),tail(n), .........)
      where
          result = -- エッジ強度の値
          !=put(result, RESULT),
            -- 結果をフィールドの外へ
          !=put(head(c), OUT)
            --  自画素の値を近傍へ
}

-- 雑音除去のフィールド定義
field SMOOTH(smooth)
{channel input: integer}
{export output: integer}
  (i:n, j:n) on mesh
= for each (i,j) in ([1..n],[1..n]) do
    link(input[i,j], ch1@[i.j])
    link(ex1@[i.j], output[i.j])
  ...................
  -- Averageの定義（前ページ）と同様


-- エッジ検出のフィールド定義
field EDGE(edge)
{channel input: integer}
{export output: integer}
  (i:n, j:n) on mesh
  ..............

-- fieldの生成と結合
  Smooth = organize(SMOOTH),
     -- 雑音除去フィールドの生成
  Edge = organize(EDGE),
     -- エッジ検出フィールドの生成
  concatenate(output@Camera, input@Smooth),
     -- カメラの出力と雑音除去フィールドを結合
  concatenate(output@Smooth, input@Edge),
     -- 雑音除去とエッジ検出を結合
  concatenate(output@Edge, input@Display)
     -- エッジ検出とディスプレイを結合
```

4．通信パターンの抽象化

画像処理のアルゴリズムに限らず，並列処理のプログラムはプロセス（実行時のイメージではプロセッサ）間の通信とプロス内での演算に大別できる．プロセス間通信の形態は，プロセスの結合トポロジーに依存することが多く，その記述方法も先に示したVによる画像処理アルゴリズムの記述を見ても分かる通りかなり繁雑になってくる．しかし，３．で述べたように並列画像処理アルゴリズムはプロセス間通信のパターンによっていくつかのクラスに分類できると考えられ，その場合あるクラスに属する画像処理アルゴリズムはプロセス内の計算方法が異なるだけであると考えてよい．例えば３×３の近傍処理（各画素で同時に計算できるものを考えており，画面の走査によって逐次的に画素の値が決まるものは除いて考える）は全て通信パターンが同一であり，異なるのは自画素の値を近傍の画素値からどうのように決定するかである．

このような観点から考えると，画像処理アプリケーションプログラマに対して，プロセス内演算の部分だけを定義すれば自動的に並列画像処理プログラムが生成できるような仕掛けを提供することができれば，プログラム記述の効率化，プログラムの誤り防止，可読性の向上といった点で極めて有効であると考えられる．また，システムプログラマ側で通信パターン記述を提供するため，最適な通信プログラムが提供されるはずであり，プログラムの実行性能を向上させるという点でも大きな意義がある．もちろん，考えられる全ての通信パターンを予め準備しておくことは難しいので，必要な通信パターンを定義できるような枠組みは必要がある．

Vでは，このような問題を高階関数を用いることによって解決することができる．以下の例では，Mask3x3が通信パターンを抽象化したものと考え，各画素での計算を定義する関数（この例では近傍画素の単純平均による平滑化）を引数とし，結果として近傍処理を行う並列画像処理プログラム，すなわちagent集合体（field）を返す．Mask3x3の戻り値（この例ではsmooth_image）に画像を与えると平滑化された画像が生成される．

```
function smooth(c, n, s, w, e, nw, ne, sw,
                se: integer)
         return(integer)
= (c + n + s + w + e + nw + ne + sw + se)/9

smooth_image: field = Mask3x3(smooth)

--
function smooth_real(c, n, s, w, e, nw, ne,
                     sw, se: real)
         return(real)
= (c + n + s + w + e + nw + ne + sw + se)/9
```

ここで述べた高階関数を利用する方法は，厳密な意味で通信パターン（通信トポロジー）を抽象化しているわけではない．実は，ストリームを通じて通信されるデータタイプまで規定することになっている．例えば，上の例で実数データ（real型）に対する平滑化の関数smooth_realは，smoothとは型が異なり（実数の引数を取り，実数の結果を返す），Mask3x3の引数として渡すことができない．従って，整数型の計算，実数型の計算というような区別をつける必要があり，プログラマは自分の行う計算のタイプにより適当な通信パターン関数を選択する必要がある．しかし，この問題は多重定義(overloading)を導入することによりユーザに対しては見かけ上１つの通信パターン関数を提供しているように見せることができるため，それほど大きな問題にはならないと考える．

５．おわりに

本稿では超並列処理記述言語Vの概要とそれを用いた並列画像処理アルゴリズムの記述について述べた．Vは特定の並列マシンアーキテクチャを意識しない汎用の言語を目指しているが，実際に実行効率の高いコードをコンパイラが生成するためにannotationをどのように付けるかという点については，今後更に検討が必要である．また，並列画像処理アルゴリズムの記述という観点から見た場合，アプリケーションプログラマが容易にアルゴリズムを記述するためにどのような基本的な関数群（特に通信パターンの抽象化のための関数）を用意すべきかをより詳細に検討する必要がある．今後，これらの点に加えて，商用並列マシン（AP-1000，CM-5等）用のコンパイラの開発やそれを用いた性能評価などを順次行う予定である．

謝辞

本研究は文部省科学研究費補助金（重点領域研究：04235104，一般研究(C)：05680300）及び久留米鳥栖技術振興センターの援助を受けた．

参考文献

[1]谷口：コンピュータビジョンのための並列処理，情報処理九州シンポジウムヒューマン・インターフェースとビジュアリゼーション，pp.53-70, 1991.

[2]山元 谷口，雨宮: 画像処理用超並列プロセッサAMPのプログラミングと性能評価について，情報処理学会論文誌，Vol.32, No.7, pp.933-940, 1991.

[3]R. Taniguchi, N. Yamamoto, N. Tsuruta and M. Amamiya: Valid-A: Parallel Functional Language for Image Processing -A programming language of AMP (Autonomus Multi-Processor)-, Proc. of 7th Scandinavian Conf. on Image Analysis, pp.1178-1187, 1991.

[4]長谷川，雨宮：データフローマシン用関数型言語Valid，電子情報通信学会論文誌，Vol.J71-D，No.8，pp.1532-1539, 1988.

[5]九州大学情報認識研究室：超並列Ｖ言語，九州大学情報認識研究室テクニカルレポート，LIU-TR-0002, 1993.