論　文

分散画像処理環境 VIOS-III

松尾　啓志[†]　　川脇　智英[†]

Distributed Image Processing Enviroment VIOS-III

Hiroshi MATSUO[†] and Tomohide KAWAWAKI[†]

あらまし
本論文では，ネットワーククラスタ環境，または共有メモリ型マルチプロセッサ環境を用いて大規模な画像処理を行う分散画像処理環境 VIOS を提案する．特にクラスタ環境では，膨大なネットワークレイテンシが問題となる．そこで VIOS では，並列画像処理記述言語 VPE-P に，(1) 依存型リダクション，(2) 並列画像処理ワーキングセットとアクセスポリシ，(3) 画像処理エンジン IPU の実装にメタスレッドを導入し，ネットワークレイテンシの低減，隠蔽を図った．さらに，複数の画像処理，認識アルゴリズムを実装することにより，VPE-P の記述能力，PVM や C 言語で記述した場合との実行時間を比較することにより VIOS の処理能力の評価を行った．

キーワード　分散処理，ネットワーククラスタ，リダクション，ネットワークレイテンシ隠蔽

1. はじめに

近年，画像処理やパターン認識アルゴリズムが大規模化しており，高速なワークステーションを用いても，アルゴリズムそのものの実行や，種々のパラメータの試行錯誤的な決定のための計算量が膨大なアルゴリズムも少なくない．

また計算機の低価格化およびネットワーク化に伴い，2 台から 4 台の比較的少数の CPU から構成される共有メモリ型マルチプロセッサ環境 (以下 SMP) や，ネットワークで結合された複数の計算機群（以下クラスタ）を容易に実現することが可能となり，この SMP や，クラスタを対象とする数々の研究が行われている．

クラスタ環境を用いる環境として一般的な方法は，ライブラリを用いる方法である．オークジッジ国立研究所と，エモリー大学の異機種分散計算プロジェクトで開発された PVM [1] や，MPI フォーラムにより規格化されたメッセージパッシング API 仕様 MPI [2] と，この API に準拠したアーゴン国立研究所が開発したライブラリ MPICH [3] などが一般的に用いられている．このライブラリを利用することにより，比較的容易に分散処理を実現可能であるため，多くの利用例がある．しかしこれらは比較的低水準のライブラリとして実装されているため，利用にはやはり分散処理の知識を必要とする．

さらに大規模な分散環境構築環境（グローバルコンピューティング，GRID コンピューティング）も，多数提案されている [4] [5]．Ninf [6] は，JAVA を用いた分散トランザクションシステムである．これらグローバルコンピューティングは，完全に処理が並列化可能である場合には非常に有効なモデルである．しかし通信遅延が膨大であるため，一般的な並列計算への適用は困難である．

また SMP のための並列プログラミングのプログラミングモデルとして OpenMP [7] が提案されている．OpenMP は基本となる言語（Fortran,C,C++) に対する指示文を基本としたプログラミングモデルである．この OpenMP は，従来のマルチスレッドを基本とするプログラムに対して，移植性が高く，かつ従来の逐次プログラムからの移行も考慮されている．PC クラスタに対応した OpenMP 環境として，新情報処理開発機構により Omni OpenMP [8] が 開発されている．

専用言語も多数開発されている．High Performance Fortran(以下 HPF) [9] は，従来の FORTRAN 言語に，最小限の指示文を付加することにより分散メモリ並列システムで容易に並列記述を可能とする言語で

[†] 名古屋工業大学電気情報工学科，愛知県
Nagoya Institute of Technology, Gokiso, Showa-ku, Nagoya, Aichi, 466-8555 Japan

ある．また京都大学で開発されたNCX[10]は，仮想プロセッサーとその結合を示すフィールドを定義することにより，従来のC言語からの互換性を考慮したデータ並列記述言語である．

一方画像処理の分野においても，ニューメキシコ大学のKhoros[11], オスロ大学のXITE[12]，AVS[13]など画像処理環境が開発されている．しかしこれらは，特定の処理単位をネットワーク上の他の計算機で実行する機能は含まれているものの，画像処理の持つデータ並列性を考慮したプログラムを作成することは考慮されていない．

本論文で提案するVIOS IIIは，

（1） PVMやMPIなどのメッセージパッシングライブラリによる並列，分散プログラミングとは異なり，並列，分散プログラミングの知識が少ないユーザにも容易に処理を記述可能とする並列画像処理記述言語VPE-Pおよびその統合処理環境の提供

（2） Khoros,XITE,AVSのように画像処理間の並行性のみを抽出し，並行実行するのではなく，画像処理内の並列性も記述可能とする並列画像処理記述能力の向上

（3） 並列実行単位が大量に発生する画像処理，画像認識アルゴリズムの特性を考慮し，スレッドを用いた実装によるネットワークレイテンシの隠蔽と，メタスレッドによるスケジューリングオーバヘッドの削減

（4） 画像処理，画像認識アルゴリズムの特性を考慮した，依存型リダクション，ネットワーク間の変数を擬似的に参照する補間実行の提案

を特徴とする並列画像処理環境環境である．

なおVIOSは今までに，複数の画像処理間をRPC(Remote Procedure Call)で結合し，画像処理間の並行実行性を自動検出することにより並行処理を行う画像処理環境VIOS I[14]，画像処理エンジンを自己拡張可能なインタプリタ言語による仮想計算機として実装し，さらに，VIOS Iでは記述不可能であった画像処理内の並列性も考慮した並列実行単位を生成可能なVIOS II[15]と開発を行ってきたが，本論文では，並列画像処理記述能力の大幅な拡張，従来のインタプリタ言語による仮想計算機型画像処理エンジンに加えて，CPUのネイティブコードによる画像処理モジュール実行などの拡張を行い，平成12年6月に公開したVIOS III REV 2(以下VIOS)について示す．

本論文では2.で画像処理の持つ特徴（大規模並列性，並行性など）について示す．さらに3.で，VIOSの概要，並列画像処理記述言語VPE-Pによるプログラミングモデルについて，4.で画像処理エンジンIPUの実装方法などについて示す．さらに5.で，VIOSの記述能力およびPVMとの性能比較について示す．

2. 画像処理における分散，並列処理の特徴

2.1 大規模並列性

画像処理アルゴリズムは，比較的並列度が高いアルゴリズムが多い．たとえばラプラシアンフィルタを代表とするマスク演算処理などは画素単位での並列性がある．また画像理解システムにおいてもボトムアップ解析による対象認識過程は，(1) 画像処理レベル，(2) 画像解析レベル，(3) 認識レベル，の3階層から構成でき，かつ各階層で実行されるアルゴリズムデータレベル並列性を有することが多い[16]．

従って，スレッドなどの並列実行方式を用いて，これら実行単位を並列化することは自然である．しかし，例えば512x512画素の画像で，画素単位の並列性を有するアルゴリズムを実装する場合，26万ものスレッドを生成，実行する必要がある．このような大規模なスレッドの場合，生成および実行のオーバヘッドは無視できないどころか，現実的ではない．東京大学の田浦らは，細粒度マルチスレッドを一般的なCPUで効率的に実行する手法としてStack Threadsを開発した[17]．この方式は，スレッド実行を関数に置き換える方式であり，非常に効率よく実行可能であるが，比較的低レベルな実装であるためユーザの負担は依然大きい。また本システムで対象としている画像処理は、データ各部の計算量がある程度推定可能な処理であるため、さらに効率の良い手法を導入することが可能であると考える．

2.2 並 行 性

画像処理理解アルゴリズムは，入力として画像（もしくはパラメータ），出力が画像（もしくはパラメータ）である画像処理単位（以下モジュール）の集合体として構成される場合が多い．たとえば，医用画像処理で良く用いられる心臓の機能DSA(digital subtraction angiography)[18]は，拡張期と収縮期の造影剤注入前後の画像の差分画像を拡張期，収縮期で，それぞれ撮影し，フィルタ処理を行い，両者の差分を算出する処理である．つまり拡張期画像の差分処理と収縮期画像の差分処理，フィルタ処理は並行して行うことが可能である．Khoros[11]やAVS[13]など画像処理環境では，モジュールの実行が入力パラメータが

揃った段階で始まるイベント駆動型の実行手法を採用すると同時に，モジュールを実行する計算機を，ネットワーク上の他の計算機に明示的に指定し，並行実行を実現している．

2.3 補 間 実 行

他の計算機の持つ情報の獲得に，ネットワークを介する必要のある分散処理の場合，ネットワークを介したデータの獲得は，致命的な速度低下の原因となる．一方，画像処理の中には，ある処理を実行する際に，必ずしもその処理に必要なすべての情報（画素情報）が揃っていなくても，ローカルの計算機が持つ情報からの補間，もしくは定数への置き換えなどによって求めた情報を用いて処理を行った場合でも，目的の処理に必要かつ十分な処理も存在する．例えば，最終的な処理がハフ変換やジオメトリックハッシングのような投票アルゴリズムの場合，前処理の段階では，多少の誤差の混入より，実行速度低下要因の排除を優先させるほうが有効である．

2.4 実行単位間通信

画素や小領域を演算単位とするデータ並列演算においても，すべての演算単位を同時に実行するのではなく，ある演算単位からの計算結果を受け取った後，演算を実行する場合や，ある演算単位が終了した後にしか実行できない演算が存在する．例えば，松山らの分類 [16] ではスキャン型演算に属する線形計画法がその一例である．このようなアルゴリズムの実行には，実行単位間の通信の実装が必要となる．つまり，ある実行単位は，他の実行単位からの演算結果もしくは演算終了情報の到着を確認した後，実行可能となる．

3. 画像処理分散画像処理 VIOS と並列画像処理記述言語 VPE-P

3.1 VIOS のシステム構成

VIOS は，VPE・IPU・OM の 3 種類のプロセスから成り立ち，各プロセスが，ネットワーク上の複数のワークステーション，もしくは SMP 環境の各 CPU に，分散して存在する．この 3 種類のプロセスが，互いに通信しながら処理を行う分散処理環境である．

- VPE（Visual Programming Editor）ユーザとのインタフェースプロセス．エディタ機能やデバック環境などを有する統合開発環境である．
- IPU（Image Processing Unit）VIOS において，実際の画像処理を行う画像処理エンジンプロセス．ユーザ側からは，並列実行単位 (画像処理ワーキングセット) 毎に仮想的な計算資源を割り当てる仮想計算資源 (Virtual Image Processor:VIP) として，また実装上は，スレッド単位に実装した計算資源 (Image Processor:IP) として実装される．
- OM（Object Manager）画像データ，IPU の管理，および処理のスケジューリングを行うプロセス．

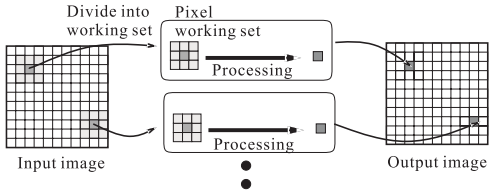

図 1　並列画像処理ワーキングセット単位の処理
Fig. 1　The parallel image processing working set

3.2 並列画像処理ワーキングセット

NCX [10] では，無限個存在する仮想的な計算資源を，フィールドと呼ばれるネットワークの結合状態を用いて表現した．このフィールドの構成要素となる結合形態として mesh, hypercube, binary tree の 3 形態が用意されている．

NCX が，仮想計算資源の結合状態のみを表すフィールドを用いているのに対して，VIOS では，処理の並列性に合わせて，(1) データ並列の依存関係，(2) 処理の中心となる画素集合（以後注目画素），(3) 参照だけを行う画素集合（以下周辺画素）の 3 種類の情報を有する並列画像処理ワーキングセット（以下ワーキングセット）と呼ぶ単位を，無限個存在する仮想計算資源 (Virtual Image Processor :以下 VIP) 上で実行することにより並列処理を実行する（図 1 参照）．これは VIOS で実行する処理対象を画像処理に特化したため，各画素の位置と結合関係が密接な関係を有することになる．従って各画素間の結合関係を明示的に定義する必要がないためである．

しかし 2.1 で示した通り，大規模な並列性を独立に実行できる VIP を用いて実行する方式は，処理のオーバヘッドが膨大となる．この解決策については，4.2 に示す．

ワーキングセットにおける，データ並列性は以下の 4 種類を現在実装している．

- **pixel** 演算処理が 1 画素ごとに独立で，必要な入力画像の範囲が注目画素と注目画素近傍の周辺画素である演算．
- **row, column** 演算処理が列（行）ごとに独立

で，必要な入力画像の範囲が注目画素列（注目画素行）とその近傍の周辺画素である演算．

- **box** 演算処理が 1 ブロックごとに独立で，必要な入力画像の範囲が注目しているブロックとその周辺画素である演算．
- **all** 演算処理が分割実行不可能で，1 度に全体の計算を行う演算．

3.3 アクセスポリシ

VIOS ではワーキングセットを超えた画素データへのアクセスは，ユーザが明示することなしに透過的に行う．つまりワーキングセット外のデータは，ネットワークを通して獲得することが可能である．しかし，ネットワークを介したデータの獲得は以下の 2 つの問題点がある．

（1）透過的なアクセスを実現するためには，ワーキングセット内の画素データに対するアクセス毎に，そのアクセス方法を決定する必要がある．(2）ワーキングセット外の画素へのアクセスは，ネットワークを介して行うと同時に，排他制御も必要なため莫大なアクセスコストが必要となる．

一方，画像処理においては 2.3 で示したように，画素値に厳密さを必要としない場合も少なくない．そこでこの 2 点を考慮して，ワーキングセット外の画素へのアクセスポリシを導入し，次に示す 5 つのポリシを用意し，ユーザがアルゴリズムに応じて明示できるものとした．

- ワーキングセット外の画素へのアクセスをチェックしない（nocheck（デフォルト））
- ネットワークを介して真の値を獲得する（get）
- ワーキングセットが有する隣接画素情報で代用する（near）
- ワーキングセットが有する情報から，画素値を補完する（interpolate）
- 指定した定数を参照結果とする（const n）

なお VPE-P 上では，指示子を用いて以下のように記述する．

> #vios boundary 画像データ名 *access_mode*（*access_mode* は nocheck, get, interpolate, near, const のいずれかを指定。なお const の場合は続いて定数を指定する）

3.4 依存型リダクション

ネットワークを介する大域変数へのアクセスは，各プロセッサ間の通信遅延が膨大なネットワーク分散環境においてはシステムのボトルネックになる．しかし，画像処理，認識アルゴリズムの中には，大域変数を局所変数と見なして実行し適当な処理単位毎にそれぞれの値を統合し処理を進めることができるものが多い．

VIOS では，画像処理が持つこのような特性を利用し，各ワーキングセット群を処理する IPU 毎に，バッファを持たせ，演算に応じてユーザがバッファの統合処理を指定することにより大域変数へのアクセスの高速化を実現した．この処理は HPF [9] で導入されたリダクションと同様である．

なお現在，バッファの統合手法として，最大値，最小値，和を実装している．VPE-P 上では，バッファ統合の制御を，以下のように大域変数への排他制御と同時に記述する． 変数名に続き，その変数へのアクセスに際する排他制御の有無，さらにバッファ統合の手法を記述する．

さらに VPE-P では，このリダクションを拡張した，依存型リダクションを導入する．HPF でのリダクションと異なり，VIOS ではリダクション変数に配列を用いることを可能とした．つまり，配列変数に対して，統合を行った場合，配列要素個々に対して演算子で指定された統合を可能とする．また，テンプレートマッチングのように，リダクション指定された配列要素各々の最大（最小）値ではなく，ある要素の最大（または最小）が必要であり，他の要素は，独立した最大値（最小値）ではなく，ある要素に対応する値が必要な場合がある．

このような，最大値（最小値）の値ではなく，最大値（最小値）を取る配列添字が必要な処理の場合，変数の最大，最小，和，AND 演算,OR 演算などのバッファー統合処理しか定義されていないリダクションを用いることはできず，最大値 (最小値) を，アクセスに膨大なコストが必要な大域変数として宣言し，更新する記述を行う必要がある．

そこで VPE-P では，統合手法が最大値，最小値の場合に限り，配列変数の最初の要素を統合し，その統合結果により配列変数の第 2 要素以降の要素の統合手法を決定する依存型リダクションを導入した．つまりあるワーキングセットが書き込みを行った配列要素が最大値（最小値）として統合された場合，それ以降の要素の値も該当するワーキングセットが設定した値となる．依存型リダクションは，予約語

```
// テンプレートマッチング用モジュール
#define R 64    //テンプレート画像の半径
#vios mutex result
  off min depend_on_first  <= (1)
module match(img,tmp:input, <= (2)
                    result:parameter)
int img on pixel cache R;   <= (3)
int tmp on all;
int result[3];
{
//逐次実行部
  result[0] = MAXINT;
//並列実行部
  parallel{                 <= (4)
    int i, j, cnt, d, dd;
    int x, y;
    x = hotx(img);  <= (5)
    y = hoty(img);
    if(x>R && x<width(img)-R && y>R && y<height(img)-R){
      cnt = 0; d = 0;
      for(i=1;i < R*2-1;i++)
        for(j=1;j < R*2-1;j++){
          cnt++;
          dd = img[i-R][j-R]-tmp[i][j];
          d += abs(dd);
       }
        if(cnt){
          d /= cnt;
          if(d < result[0]{
            result[0] = d; result[1] = x; result[2] = y;
          }
       }
    }
  }
// parallel ブロック終了時にバリア同期 <= (6)
// 以後，逐次実行部，並列実行部とも記述可能
}
```

図 2　テンプレートマッチングモジュール
Fig. 2　Template matching module

depend_on_first　により指定する．図 2 中 **(1)** に利用例を示す．

> #vios mutex 変数名 [on ｜ off] [add ｜ max ｜ min] [depend_on_first]

3.5　ワーキングセット間通信

2.4 で示したように，あるワーキングセットからの処理結果（もしくは，処理終了情報）を獲得したのち，始めてそのワーキングセットの演算処理が実行可能となる演算がある．VIOS では，他のワーキングセットからの情報が必要な場合は，情報が到着するまで実行をウエイトするブロッキング型のワーキングセット間の通信 (comm 命令) を実装した．図 3 中，**(1)** がワーキングセット (my_x-1,my_y) からの通信を受けるコマンドであり，**(2)** がワーキングセット

```
// 通信ストリームを用いて処理結果データを取得
    comm(my_x-1, my_y) >> pre_x;   <= (1)
    comm(my_x, my_y-1) >> pre_y;
    comm(my_x-1, my_y-1) >> pre_xy;
// 2 つの系列の比較を行い、表の該当位置の値を決める
    if(String_A[[my_x]][[0]] == String_B[[0]][[my_y]]){
        value[][] = pre_xy + 1;
        arrow[][] = 0;
    }    else {       省略       }
// このワーキングセットでの処理結果データを必要とする
// ワーキングセットにデータを転送
        comm(my_x+1, my_y) << value[][];  <= (2)
        comm(my_x, my_y+1) << value[][];
        comm(my_x+1, my_y+1) << value[][];
    }
}
```

図 3　DP マッチングのモジュールの通信部分
Fig. 3　Communication part of DP matching program)

(my_x+1,my_y) へ送信する部分である．

3.6　VPE-P での並列プログラミング概略

VPE-P は，各モジュール間の処理の流れを記述するメインフロー記述部と，個々の画像処理アルゴリズムを記述するモジュール記述部からなる．

モジュール記述部は，実際の画像処理の記述を行う部分である．入出力のワーキングセットのタイプを変数名とともに指定し，各ワーキングセットの処理を記述する．

宣言部では以下のように，予約語 input の前に入力，output の前に出力画像変数を指定したのちに，画像以外の変数を parameter として宣言する (図 2 中 **(2)**)．

> module　モジュール名　(a1,…:input, x1,…:output, p1,…:parameter)

モジュール宣言部に続く引数宣言部では，以下のように入出力の型などを指定する．

> データ型　変数名 [on *working_set* 型 [cache *n*]]

入出力変数が Image 型(注 1)のときは，on に続いてワーキングセットタイプを指定する．演算に周辺画素が必要なときには，cache に続いて，その周辺画素幅を指定する (図 2 中 **(3)**)．

(注 1)：VIOS では画像特有のデータ構造として，Image 型を導入した．この型は，本来画像が持つべき種々の情報（各画素の型，大きさ）を持つ

モジュール内部はワーキングセットごとの記述を行う並列処理記述部と，画像分割を行わない処理の記述を行う逐次処理記述部に分けられる．

parallel ブロック (図 2 中 **(4)**) は，モジュール内でのワーキングセット単位での並列処理を記述するブロックであり，モジュール内に複数個配置することができる．parallel ブロック内部では，引数宣言部で指定されたワーキングセットごとの記述を行う．このブロック内の画像の座標は，ワーキングセットの原点からの相対座標で表される． なおワーキングセットに分割する前の原画像の原点からの座標でアクセスするときは，二重かっこ（**[[添字]]**）を用いる．また， 図 2 中 **(5)** の hotx(),hoty() は，原画像におけるワーキングセットの注目画素（複数の注目画素がある場合は左上）の絶対位置を表す組み込み関数である．

並列処理部と逐次処理部の間では，並列処理部単位にバリア同期が行われる (図 2 中 **(6)**)．つまりすべての並列部で記述されたすべてのワーキングセットの演算が終了するまで，逐次処理部に制御が移ることはない．

4. 画像処理エンジン IPU

4.1 仮想計算機による実装

IPU 内には，スタック形式インタプリタ言語 IPU-P [15] を理解し実行する IP（image processor）が用意される．1 つの IP は 1 スレッドとして実装した仮想的なプロセス資源であり，無限に作成可能であると同時に，複数並列に動作する．この IP を必要時に動的に生成し，画像演算処理を実行させることで，1 つの IPU 内での処理の並列実行を実現する．

IP は，モジュール定義・起動 IP（以後 ip0），処理演算実行 IP（ip1）と，メインフロー実行用 IP（ip2）の 3 種類の IP から構成される．

ip0 は、OM からの入力を受け付け、モジュールの定義など実行する準備をし、ip1 の起動を行う。ip1 ではあらかじめ利用可能なモジュール、またはユーザ定義のモジュールを実行する。ユーザ定義モジュールは、ダイナミックリンクライブラリ形式により動的にロードされ，実行される。また ip1 はモジュール終了時に解放される。 ip2 は特殊な IP であり、システム中 1 つの IPU にのみ存在し，メインフロー記述部のみを実行する IPU-P インタプリタとして動作する．また、メインフロー記述部内のモジュール呼び出しについては、並列実行するために OM へスケジューリングを依頼する。

IPU 内の ip1 の実行数は，システムの性能に大きな影響を及ぼす．適切な実行数は，各 IPU の結合形態（バス結合，ネットワーク結合），IPU 数，処理内容により変化するため，現在は経験的に決定している．

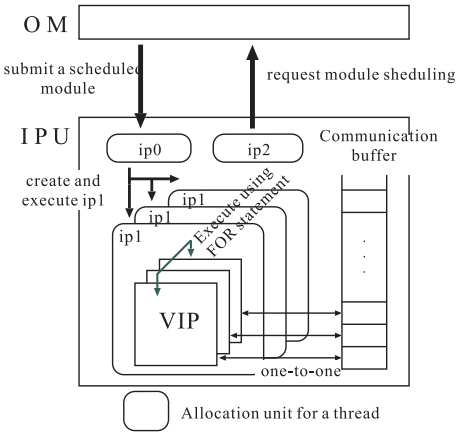

図 4 IPU の実装方法
Fig. 4 The implementation of IPU

4.2 スレッドによる実装とメタスレッド

並列性の高いプログラミングの場合，スレッドによる実装は，リモート処理時のネットワークレイテンシ隠蔽のためにも有効である．VIOS では，IPU 内の ip0,ip1,ip2 を全てスレッドとして実装した．つまり IPU 内に更に細かい演算単位が並列実行されることになり，通信路が確立されるまでの待ち，および同期待ちの間に，他のスレッドが通信を行うことが可能となり，結果として通信レイテンシの隠蔽に寄与することになる [20].

さらに，VIOS では，ユーザから見える仮想計算資源である VIP 単位で並列性を有するため，VIP 単位でスレッドを割り当てる実装も可能である．しかし，2.1 で述べた通り，並列性を有する画像処理の場合は，大量のスレッドが並列実行されることになる．

本システムで想定している並列実行形式は，SIMD 型である．従って、StackTreads [17] のように必ずしも実装上においても細粒度レベルで扱う必要はない。また、処理対象として画像処理に着目しているため、Fleng [19]、Id [21] のようなデータフロー言語で必要となる複雑な解析を行わなくとも，比較的容易に粒度を上げることができる。そこで、本システムでは以下の特徴を持つメタスレッドを導入した．

- ユーザ側から見た仮想的な CPU 資源，VIP を複数組み合わせたメタスレッド単位で ip1 を生成し，OS にスレッド生成を要求する．
- メタスレッド内では，複数の VIP の実行を，FOR 文に展開し，逐次実行することにより細粒度タスクを実行する。
- ワーキングセット間通信である comm 命令を実装するため、任意の VIP 間で非同期に通信を行うことを可能とするため，図 4 に示すように，VIP 各々に独立した通信バッファを確保した。

なお並列実行単位 VIP を，複数まとめるメタスレッドの導入により，comm 命令実行時に，デットロックが生じる可能性がある．現在，デットロックに関する検出，予防処理は行っておらず，メタスレッドの実行順は，DP マッチングに代表されるシストリック型通信を想定してラスタスキャン方向に設定している．(注 2)

4.3 ノンブロッキング実行可能なメインフロー記述部

画像処理は，2.2 で示したように，一般に複数の画像から新しい画像を作り出すといった動作の連続で構成されるものが多い．つまり，モジュールは逐次的な組み合せで実行されると同時に，処理モジュールから生成される画像はすべて新規に生成されるという制限（単一代入）を導入することは，使用メモリの増大にはつながるものの，処理過程が容易に確認できるという利点もある．

そこで，VIOS では メインフロー記述部において，Image 型変数の依存関係のみを，実行時に解析することにより，依存関係のないモジュールの実行がノンブロッキングで行うことを可能とするノンブロッキング実行を採用した．

5. VIOS III の実装と性能評価

5.1 並列処理記述能力

VPE-P の並列処理記述能力を調べるため，オスロ大学で開発されている画像処理ライブラリ XITE [12] のうち，明らかに処理に並列性があるもの，さらにはスキャン型演算の代表例について VPE-P での記述を試みた．その結果以下に示すような並列性を持つものについて，VPE-P で記述が可能であることが確かめられた．以下，松山ら [16] の並列画像処理の分類に基づいて示す．

- 一括型演算（微分処理，ノイズ除去，線分検出，Sobel などのマスク処理など）
- 分散型演算（濃度変換，二値化など）
- 集約型演算（濃度ヒストグラム，最大値検出，線分検出など）
- スキャン型演算（DP マッチング，並列スネーク，ヤコビ法による温度分布計算）

展開型演算は，データフロー的にはスキャン型演算と同等であるので，VIOS では記述可能である．ただしワーキングセットごとに関数を定義するため，1 つのモジュール内に，ワーキングセット毎の複数の処理を書く必要がある．

さらに，ソーベルフィルタおよび非極大点の抑制による細線化，大津の閾値決定法による閾値選択，二値化，ハフ変換を含む並列認識処理過程 [16] についても記述が可能であることが確かめられた．

5.2 実行性能評価

次に，VPE-P で記述した並列認識処理過程 [16] を，実行することにより，実行速度向上の評価を行った．なお入力画像は 512x512 画素からなる画像を用い，ハフ変換前の投票対象点は 8259 点（全画素数の約 3%），また 180x1446 からなる投票空間を用いた．

5.2.1 共有メモリ型マルチプロセッサ環境による実行性能の評価

上記処理を，8 CPU がローカルバスで結合されたサンマイクロシステムズ社製の SS-1000（OS:Solaris2.5, Memory:128MB）上で実行することにより，IPU 内の並列度（IPU が同時実行する ip1 数）の増加による実行速度向上の評価を行った．実行結果を表 1 に示す．表に示す通り，並列度が 2 の時で，実行速度が 1.8 倍，4 の場合に 2.9 倍，8 の場合に 4.0 倍の速度向上結果を得た．並列度の向上による実行速度の低下は，画像の分割統合処理，および各モジュールの起動時間によるものである．

さらに C 言語で同様のアルゴリズムを記述した実行速度と比較するための比較実験を行った．C 言語のみで記述した場合と，VPE-P で記述した場合のそれぞれを 1 CPU で実行することにより求めた並列化のオーバヘッドは約 2 割であった．

5.2.2 クラスタ環境化での実行性能の評価

ネットワークで結合された 4 台のワークステーション上で IPU を実行し，さらに OM，VPE を別々のワークステーション上で実行した分散環境上で，5.2.1

(注 2)：次回リリースでは，この点について解決する予定である

と同様のプログラムを実行した結果を表2に示す．なお実験環境は，CPU:PentiumPro（200MHz），メモリ:128M，OS: Solaris2.5.1 の計算機を，100BaseTX のスイッチングハブ（Bay networks Model 28115）で構成されたネットワークで結合した環境を用い，各 IPU に対して 3 つの ip1 を生成した．比較対象として C 言語で記述した場合（1CPU のみ），と C 言語とメッセージパッシングライブラリ PVM 3.3 (リダクションは独自に実装) を用いた．

実験の結果，4 台の計算機を用いた場合で，2.6 倍の速度向上結果を得た．C 言語のみで記述した場合と，VPE-P で記述した場合のそれぞれを 1CPU で実行することにより求めた並列化のオーバヘッドは約 2 割であった．VIOS ではメタスレッドによる実装により，マルチスレッドの実行に際するオーバヘッドは無視できる．従ってこの原因は，画像をメタスレッドに分割するため，画素の位置を相対的にアクセスする必要があることや，OM が行うスケジューリングのオーバヘッドであると考える．

また PVM による実装よりも，VIOS による実装が約 7 割高速となった (使用する計算機が 4 台の場合)．データの転送が大部分を占めるハフ変換処理の速度向上が，PVM の場合計算機 4 台の際にも 1.1 倍と，ほとんど速度向上が得られないためである．これは実装に用いた PVM3.3 がマルチスレッドに対応していないため，大量のデータを転送する際，マルチスレッドを用いて通信開始時や同期時に，他のスレッドが通信できる VIOS と異なり，マルチスレッドに対応していない PVM では通信レイテンシの隠蔽が困難であるためである．

なおさらに，細線化や二値化および最大値検出のように比較的短時間で終わる処理は分割せずに実行することにより，分割処理のオーバヘッドが軽減し，より高い速度向上結果が得られるものと考える．

6. ま と め

VIOS システムは，ユーザとのインタフェースである VPE，画像処理演算を行なう IPU，画像データや IPU の管理，その他の処理のスケジューリングを行う OM の 3 種類のプロセスをネットワーク上に配置した分散画像処理システムである．

本論文では，並列画像処理記述言語 VPE-P による並列画像処理プログラミングが，様々な並列性を持つ画像処理アルゴリズムを記述可能であることを確認した．さらに既存の分散環境である PVM や C 言語との比較を行い，システムの有効性を示した．

本システムは，平成 11 年 4 月より，オープンソースとして VIOS Web PAGE (http://mars.elcom.nitech.ac.jp/vios/) 上で公開中である．記述言語は IPU，OM 部分は C++，VPE 部分は TCL/TK である．実装した OS は Solaris (SPARC,X86) であるが，POSIX thread を含め，汎用的に記述しているため，他の OS への移植は容易である(注 3)，平成 12 年 6 月までに，バグ修正，機能拡張を含め 3 回のリリースを行い，延べ 400 件ダウンロードされた．

本システムは，ソフトウェアのみによって成り立つシステムであり，特別なハードウェアを必要としない．また，近年急速に普及してきた，ネットワーククラスタでの利用を前提に実装しているため，導入が容易であり，計算機資源の有効利用に役立つものと考える．

本システムの今後の課題として，次の点を検討中である．(1) 処理モジュールの実行時間や，プログラムのフローから，処理モジュールに優先順位を付け，実行を最適化するようなスケジューリング手法の開発．(2) VIOS 用分散並列画像処理ライブラリの開発．

文 献

[1] A. Geist et al.,"Parallel Virtual Machine A Users Guide and Tutorial for Networked Parallel Computing," MIT Press, 1994.

[2] C. H. Still, "Portable Parallel Computing Via the MPI1 Message-Passing Standard," Computers in Physics, Vol.8, No.5, pp.533-538, 1994.

[3] William Gropp, Ewing Lusk and Anthony Skjellum,"Using MPI: Portable Parallel Programming with the Message-Passing Interface," MIT Press, 1999.

[4] Shirley Browne, Henri Casanova and Jack Dongarra, "Providing Access to High Performance Computing Technologies," Springer-Verlag's Lectures Notes in Computer Science No.1184, pages 123-134, 1997.

[5] M. Romberg,"The UNICORE Architecture: Seamless Access to Distributed Resources," Proc. of the eights IEEE Int. Symposium on High Performance Distributed Computing(HPDC-8), pp. 287-293, 1998.

[6] 中田秀基, 高木浩光, 松岡 聡, 長嶋雲兵, 佐藤 三久, 関口 智嗣,"Ninf による広域分散並列計算," 情処学論 Vol.39, No.6, pp. 1818-1826, 1998.

[7] Leonardo Dagum and Ramesh Menon,"OpenMP: An Industry-Standard API for Shared-Memory Programming," IEEE Computational Science & Engineering,

(注 3)：LINUX 2.2.x 系の OS はスレッド内でのシグナルの扱いの相違により VIOS の移植は現在困難である

Module name	Description Language				
	C	VPE-P			
	Number of CPU				
	1	1	2	4	8
Sobel filter	0.90	1.45(1.0)	0.87(1.7)	0.44(3.3)	0.28(5.2)
Thinning	0.29	0.54(1.0)	0.33(1.6)	0.24(2.3)	0.17(3.2)
OTSU	0.15	0.61(1.0)	0.42(1.5)	0.32(1.9)	0.27(2.3)
Hough trans.	7.93	10.99(1.0)	5.44(1.8)	3.06(3.3)	2.60(3.9)
local max detect	4.68	4.69(1.0)	2.52(1.9)	1.84(2.6)	1.03(4.6)
max detect	0.12	0.28(1.0)	0.20(1.8)	0.15(2.9)	0.11(4.0)
sum	14.07	17.57(1.0)	9.78(1.8)	6.05(2.9)	4.46(4.0)

表 1　並列認識過程の SparcCenter1000 上での実行結果 (単位: 秒)
Table 1　Performance for pattern recognition on Sparc Center 1000 (unit: sec)

Module name	Description Language						
	C	VIOS III and VPE-P			PVM and C		
	Number of Workstation						
	1	1	2	4	1	2	4
Sobel	0.69	0.78(1.0)	0.47(1.7)	0.29(2.7)	0.73(1.0)	0.38(1.9)	0.20(3.7)
Thinning	0.09	0.15(1.0)	0.21(0.7)	0.29(0.5)	0.14(1.0)	0.08(1.8)	0.06(3.5)
Otsu binarization	0.06	0.42(1.0)	0.56(0.8)	0.67(0.6)	0.10(1.0)	0.06(1.7)	0.05(2.0)
Hough Trans.	6.44	6.97(1.0)	3.60(1.9)	2.10(3.3)	7.44(1.0)	6.22(1.2)	6.58(1.1)
Local Max detect.	2.05	2.86(1.0)	1.53(1.9)	0.82(3.5)	2.21(1.0)	1.42(1.6)	0.81(2.7)
Max detect	0.05	0.13(1.0)	0.15(0.9)	0.26(0.5)	0.06(1.0)	0.04(1.5)	0.03(2.0)
Sum	9.38	11.31(1.0)	6.52(1.7)	4.43(2.6)	10.68(1.0)	8.20(1.3)	7.73(1.4)

表 2　並列認識過程のワークステーションクラスタ上での実行結果 (単位: 秒)
Table 2　Performance for pattern recognition on Workstation Cluster (unit:sec)

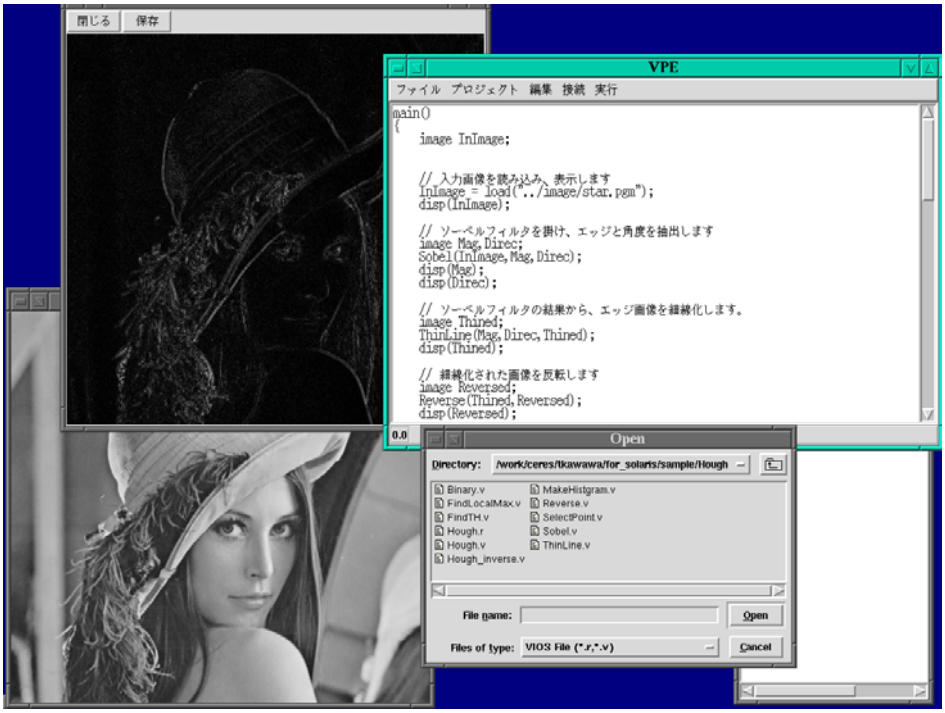

図 5　VIOS の実行時スナップショット
Fig. 5　A snapshot of VIOS

Vol. 5, No. 1, January/March, 1998.
[8] Mitsuhisa Sato, Shigehisa Satoh, Kazuhiro Kusano and Yoshio Tanaka, "Design of OpenMP Compiler for an SMP Cluster," Proc. of 1st European Workshop on OpenMP (EWOMP'99), pp.32-39, Lund, Sweden, Sep., 1999.
[9] Koelbel, D.Loveman, R.Schreiber, G.Steele and M.Zosel," The High Performance Fortran Handbook," MIT Press, 1993.
[10] 湯淺太一，貴島寿郎，小西浩," データ並列計算のための拡張C言語NCX," 信学論, Vol.J78-D-I, No.2, pp.200-209, 1995.
[11] Konstantinedes and Rasure,"The Khoros Software Development Environment for Image and Signal Processing," IEEE Trans. on Image Processing, Vol.3, No.3, pp.243-252, 1994.
[12] Otto Milvang, Tor Lnestad,"An object oriented image display system," Proc. of 11th ICPR, pp.D218-221, 1992.
[13] 吉川慈人ほか," ビジュアルプログラミング技術を使った可視化ツール AVS," 日経 CG, Apr, pp.193-203, 1991.
[14] 松尾啓志，和田錦一，岩田 彰，鈴村宣夫," 分散画像処理環境 VIOS," 信学論, Vol.J75-D-II, No.8, pp.1328-1337, 1992.
[15] H.Matsuo and A.Iwata,"A distributed image processing environment VIOS II, Asian Conference on Computer Vision (ACCV93), pp.715-718, 1993.
[16] 松山 隆司，浅田尚紀，青山正人，朝津英樹," 再帰トーラス結合アーキテクチャにおける並列対象認識のためのデータレベル並列プロセスの構成," 情処学論,Vol.36, No.10, pp.2310-2320, 1995.
[17] 田浦健次郎，米澤明憲," 最小限のコンパイラサポートによる細粒度マルチスレッディング— 効率的なマルチスレッド言語を実装するためのコスト効率の良い方法," IPSJ Vol.41, No.5, pp.1459-1469, 2000.
[18] 松尾啓志、岩田彰、堀場勇夫、鈴村宣夫、高橋睦正,"Digital Subtraction Angiography に適した左心室輪郭自動抽出アルゴリズム," 信学技報,MBE84-50, pp.31-38, 1984.
[19] 荒木 拓也, 田中 英彦, "Committed-Choice 型言語 Fleng における粒度制御法の評価," 情報処理学会論文誌: プログラミング, Vol.40, No.SIG1(PRO2), pp.23-31, 1999.
[20] 岡本一晃，松岡浩司，廣野英雄，横田隆史，坂井修一," 超並列計算機におけるマルチスレッド処理機構と基本性能," 情処学論, Vol.37.No.12, pp.2398-2407, 1996.
[21] Rishiyur S. Nikhil and Arvind, "Id: a language with implicit parallelism," Technical report CSG 305, MIT Laboratory for Computer Science, Cambridge, MA, 1990.

謝辞 本研究の一部は，堀情報科学振興財団，大川情報通信基金，立松財団の補助を受けた．

（平成年月日受付，月日再受付）

松尾 啓志 （正員）

昭 58 名工大・情報卒．昭 60 同大大学院修士課程了．同年松下電器産業（株）入社．平 1 名工大大学院博士課程了．同年名工大・電気情報・助手．平 5 名工大・電気情報・講師，平 8 名工大・電気情報・助教授，現在に至る．分散システム，画像認識，分散協調処理に関する研究に従事．工博，情報処理学会，人工知能学会，IEEE 各会員．

川脇 智英 （学生員）

平成 12 年名工大・電気情報卒．現在同大大学院博士前期課程在学中．分散システムに関する研究に従事．