分散画像処理環境 VIOS III の開発

山本伸一 中田浩作 松尾啓志 岩田彰
名古屋工業大学電気情報工学科
〒 466 名古屋市昭和区御器所町
matsuo@elcom.nitech.ac.jp

あらまし　我々は従来よりネットワークで結合された複数の計算機を用いて画像処理を行う分散画像処理環境 VIOS を提案してきた。本発表では、(1) 柔軟な並列処理記述能力を備えた新しい並列画像処理言語 VPE-p の開発 (2) ネットワークを介してアクセスする必要のある大域変数へのアクセスに対して、ユーザが明示的にバッファの統合方法を明記するプログラマブル大域変数バッファの導入 の２点の拡張を行った、VIOS III の提案を行うとともに、並列画像認識アルゴリズムなどを実装することにより、新しい VPE-p の記述能力および処理能力の評価を行う。

キーワード　分散処理、分散処理記述言語、画像処理、マルチスレッド

A Distributed Image Processing Environment VIOS III

Shinichi Yamamoto, Kosaku Nakada, Hiroshi Matsuo, Akira Iwata
Dept. of Electrical and Computer Eng.,Nagoya Institute of Technology,
Gokiso-cho, Showa-ku, Nagoya, 466, Japan

Abstract We proposed a network computer software environment for parallel image processing, VIOS. In this report, the third version, VIOS III is proposed. In VIOS III, a new parallel processing language VPE-p which has flexible syntax for describing parallel algorithms has been developed. And the new programmable buffer for accessing global variable through local area network is also proposed. The description ability for parallel image processing algorithms and processing performance are investigated by several image processing and recognition algorithms.

Keyword　distributed processing, distributed processing description language, image processing, multi thread

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、数時間にも及ぶ計算時間を必要とするアルゴリズムも開発されている。一方、画像処理やパターン認識の分野では、画素ごと、もしくは小領域を単位として並列実行可能なアルゴリズムも少なくない。そこで我々は、従来よりネットワークで結合した複数の CPU を利用することにより、大規模な画像処理を実行する分散画像処理環境 VIOS を提案し、VIOS I,VIOS II の開発を行ってきた [1][2][3]。一方、近年のワークステーションの高速化、低価格化、および転送速度 100Mbit/sec を超えるネットワークの普及、ネットワークで結合された複数のワークステーションを用いた分散処理や、共有バス方式のマルチＣＰＵシステムなどが一般的となってきた。しかしこれらのシステムを用いるためには、ネットワークプログラミング、排他制御などシステムプログラミングに関する知識が必要となる。そこでネットワーク分散環境を構築するためのライブラリとして、テネシー大学などの PVM[5]、アルグローネ国立研究所の MPI[6] などがある。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。

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[7], オスロ大学の XITE[8]、AVS[9] など画像処理環境が開発されている。しかしこれらは、特定の処理だけ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できない。

VIOS は、並列プログラミングの知識があまりないユーザにも容易に並列処理が記述できる画像処理用並列記述言語 VPE-p を有する汎用並列画像処理環境である。しかし VIOS II で開発した並列画像処理言語（VPE-p) は、画像処理の中でも、画素、行、列単位の処理並列性に注目して並列処理を記述する手法を導入したため、並列処理記述能力が低いという欠点があった。そこで本発表では、(1) 柔軟な並列処理記述能力を備えた新しい並列画像処理言語 VPE-p の開発 (2) ネットワークを介してアクセスする必要のある大域変数へのアクセスに対して、ユーザが明示的にバッファの統合方法を明記するプログラマブル大域変数バッファの導入 の２点の拡張を行った VIOS III の提案を行うとともに、並列画像認識アルゴリズムなどを実装することにより、新しい VPE-p の記述能力および処理能力の評価を行う。

2 分散画像処理 VIOS

2.1 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、図 1 に示すように、ネットワーク上の複数のワークステーションに各プロセスが分散して存在する。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。エディタ機能を有し、処理プログラム (VPE-p) を作成することができる。

- 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。すでに存在する画像データに画像処理モジュールで記述された処理を施すことにより、新しい画像データを生成する。このプロセスをネットワーク上に複数配置することにより分散処理を実現する。

- 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。VPE から送られてくる命令を解析し、適切な IPU へ処理を振り分けることにより処理の分散化を行う。

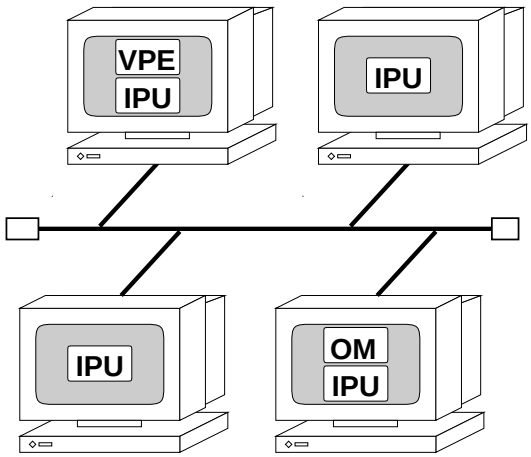

図 1: VIOS のシステム構成

2.2 VIOS I

VIOS I では、画像処理アルゴリズムの実行過程で、（中間）画像から（中間）画像を生成する画像間演算を処理の基本単位（以下画像処理モジュール）と

し、並列実行単位とした。例えば、入力画像を、ラプラシアンフィルタリングを行い、中央値フィルタリングの後、二値化するような処理の場合は、おのおの３つの処理をモジュールとして定義した。VPE において、視覚的にプログラミングされたプログラムを解析し、画像処理モジュール間で互いにその処理が独立なものを、別の IPU で実行することにより分散処理を行った。

さらに VIOS I では、負荷予測を元にした分散スケジューリングを提案した。これは、IPU から定期的に送られてくる負荷情報と OM 内部での負荷推定を元に瞬時的な負荷を予測し、処理のスケジューリングを行うものである。

2.3 VIOS II

VIOS II では、画像処理モジュール内の並列性に注目し、モジュール内の並列性に合わせて、画像を (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つの情報を持つ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）と呼ぶ最小単位まで分割することにより分散処理を行う手法を提案した (図 2参照)。

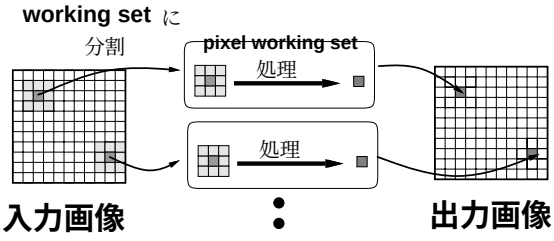

図 2: 並列画像処理ワーキングセット単位の処理

画像処理の記述方法は、C 言語に似た文法を持つ並列画像処理記述言語 VPE-p によって行う。VPE-p での並列性の記述は、ワーキングセット単位で行う。従って並列実行が可能なワーキングセット (pixel, row, column, box) では、１モジュールが複数のワーキングセットを有することとなる。

ワーキングセットは、以下の 4 種類が現在実装されている。なお実行時には、ワーキングセットの実行は、同じ IPU に割り当てられたワーキングセット群を単位に行う。

- **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 1 列ごとに独立で、必要な入力画像の範囲が注目画素行（注目画素列）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。

2.4 VIOS II の問題点

ワーキングセットの導入により、明らかなデータ並列性がある画像処理演算については簡単に処理を記述することが可能となった。しかし記述できる処理は、松山らが分類した基本演算パターン [4] で分類した場合、画素すべてに対して同一な演算を行う一括型演算と、１画像と単一要素データが入力となる分散型演算のみであり、その他の分類である集約型演算、スキャン型演算、展開型演算は記述不可能であった。

例えば、入力画像から最大値を発見する演算は、明らかに画像を部分的に分割した単位で並列性が存在するが、VIOS II の VPE-p では、ブロック単位での実行ができるワーキングセット box は存在するものの、言語構造の中に集約型演算の枠組みが記述不可能であったため、このような処理は並列実行不可能であった。

2.5 VIOS III での拡張

VIOS III では、画像処理アルゴリズムが有する様々な並列性の記述を可能とするために並列画像処理記述用言語 VPE-p に以下のような拡張を行った。

- モジュール内に逐次処理記述部を導入

  逐次的な処理を含む画像処理演算をモジュール内部に記述できるように、モジュール内部をワーキングセットごとの記述を行う複数の並列処理記述部と、ワーキングセットによる分割を前提としない処理の記述を行う複数の逐次処理記述部に別けて記述が行えるようにする。この拡張を box ワーキングセットと組み合わせることにより、並列記述能力は飛躍的に向上するものと考える。

- ネットワーク透過型大域変数の導入

  ネットワーク上に分散配置されている複数のワーキングセット間での共有可能な変数として、モジュール内で大域的にアクセスできる大域変数を定義する。このことにより、並列処理記述部の間や、逐次処理部と並列処理部の間で

値を受け渡すことが可能となる。さらに、ワーキングセットを超えた画素データのアクセスも可能とする。

- 非同期実行

  ネットワーク分散処理の様に、計算資源が並列レベルに対して多く配置できる場合、非同期実行を導入することにより、より並列性が増し、結果として処理速度が向上する。MPI では、非同期実行を導入することにより、並列記述能力の向上が可能である。VIOS III では、VPE-p のすべての記述部に非同期実行を導入することは、制御の複雑性を増すと考え、モジュール間の処理関係を記述するメインフロー記述部において、変数の依存関係を解析することにより、モジュールの非同期実行を可能とする。

- プログラマブル大域変数バッファ

  画像処理にはハフ変換やジオメトリックハッシングなどの確率的手法のように、変数のアクセスに必ずしも、厳密な排他制御を必要としない場合や、ある処理単位の終りに、適切な規則に基づいて各 IPU 内に設けたバッファを統合することにより大域変数へのアクセスを IPU 内の局所変数へのアクセスに置き換え可能な場合も少なくない。そこで VIOS III では、分散画像処理環境に適したプログラマブル大域変数バッファを提案する。

3 分散画像処理記述言語 VPE-p

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定義部からなる。

3.1 画像処理モジュール内並列記述の拡張

モジュール定義部は、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。

モジュール宣言部では以下のように入力画像データ名、出力画像データ名、その他の変数名の順に並べて記述する。

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

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* [cache *n*]]

入出力変数が Image 型[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、その幅を指定する。

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。

図 3は、512x512 画素からなる画像を 32x32 画素の box ワーキングセットに分けて、最大値検出を行う画像処理モジュールの記述例である。

```
module  Max(a:input, x:output)
int a on box[32][32];
int x;
{
  int max[16][16];
  concurrent
  {
    max[hotx(a)/32][hoty(a)/32]=a[0][0];
    for (int i = 1; i < 32; i++)
      for (int j = 0; j < 32; j++)
        if (max[@x][@y] < a[i][j])
          max[@x][@y] = a[i][j];
  }
  {
  x = max[0][0];
  for (int i = 0; i < 16; i++)
    for (int j = 0; j < 16; j++)
      x = (x > max[i][j]) ? x:max[i][j];
  }
}
```

図 3: 処理モジュール定義例 (最大値検出)

逐次処理部において宣言された変数は、大域変数として扱われる。この例では、max が大域変数である。

concurrent ブロックは、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。conncurrent 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。concurrent ブロックの 1 行目における a[0][0] は、各 box の最も左上の値となる。なおワーキングセットに分割する前の原画像

[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、過去の処理の履歴など) を持つ

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、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、並列処理部単位に同期がとられる。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。

この例は、concurrent ブロック内において各 box ワーキングセット内ごとの最大値を大域変数 max に代入し、すべてのワーキングセットの最大値の中から、原画像の最大値を求める処理例である。

3.2 遠隔データへのアクセス

従来の VIOS では、ワーキングセット外へのアクセスを行うことは不可能であった。しかし、この制限は明らかに多様な分散画像処理を記述する上での重大な欠点となった。そこで、VIOS III 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を透過的に行えるように VPE-p の拡張を行った。

しかし、(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われると同時に、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画素値に厳密さを必要としない場合も少なくない。

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として、次に示す４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 (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情報から、画素値を補完する (comlete)
- 指定した定数を参照結果とする (const *n*)

なお VPE-p 上では、以下のように記述する。

#vios boundary 画像データ名 *access mode*
(get,complete,const 定数のいずれか。指定が無い場合はネットワークを介してアクセスすることはしない)

3.3 非同期実行可能 main 文

画像処理は、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。VIOS II では、こういった処理をモジュールの形で記述し、メインフローにおいてモジュールを組み合わせることにより画像処理全体を構成する。つまり、モジュールは逐次的な組み合せで実行されることになる。

しかし、逐次的に処理されるモジュール間で依存関係のないものについては、同時に実行が可能である。VIOS III ではこの点に着目し、メインフローの変数の依存関係を実行時に解析することによりモジュールを非同期に実行することを可能にした。

図 4は、メインフローの記述例である。load はファイルから画像を読み込むモジュールで、VIOS にあらかじめ用意されているものである。ここで、PreSyori() は、画像に固有な値を計算するモジュール、Syori は、入力画像から新しい画像を作り出すモジュールであるとする。

この例では、load モジュールは同時に実行することが可能である。また、2 つの PreSyori は、load モジュールの終了を待つ必要がある。次の if 文では、a と b の値を必要とするので、PreSyori モジュールの終了を待つ必要がある。つまり、load モジュールが終了するまでの間、このプログラムはブロックする。load モジュールが終了すると、それに対応した PreSyori モジュールが起動される。そして、2 つの PreSyori モジュールの実行が終わるまで if 文でブロックする。2 つの PreSyori モジュールの実行が終了すると if 文の条件式が評価され、対応する Syori モジュールが起動される。

```
main()
{
  image A,B,C;
  int a,b;

  A = load("foo.obj");
  B = load("bar.obj");
  C = new("hoo.obj");

  PreSyori(A,a);
  PreSyori(B,b);

  if (a > b)
    Syori(A,C);
  else
    Syori(B,C);
}
```

図 4: メインフロー記述例

4 プログラマブル大域変数バッファを用いた遠隔データへの書き込み

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 III 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。

図 5は、濃度ヒストグラムの計算にバッファ統合処理を用いた場合のデータの流れである。各 IPU ごとに大域変数 hist のバッファを持たせ、バッファに対して書き込みを行う。field の実行がすべて終わった時点で、各バッファを足し合わせることによりヒストグラムの計算が終了する。

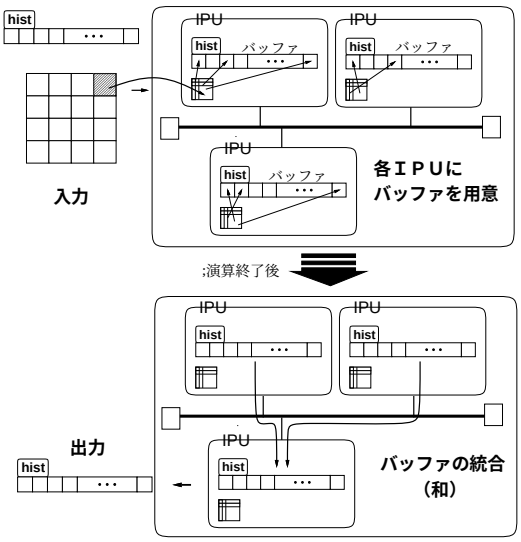

図 5: プログラマブル大域変数バッファを用いた濃度ヒストグラムの計算

バッファの統合手法としては、現在、最大値、最小値、和を実装している。

5 VIOS III の実装と性能評価

5.1 IPU の実装

VIOS II では、IPU を画像処理モジュールの実行が可能で、動的に画像処理モジュールの追加ができる仮想計算機として実装し、IPU を動作させるための言語として、PostScript に似た構文を持つ IPU-p を導入した。

IPU を仮想計算機する理由として

- 画像処理には、比較的パラメータの調整など対話的な要素が必要な処理が多い。そこで、処理部をインタープリタとして実装することにより、処理の対話性を向上させる。なおデバックやパラメータの調整の終わったモジュールは、IPU の記述言語である C++へ変換して IPU 内に実装を行う方式とする。

- 現在の汎用ＯＳが提供する情報だけでは、分散スケジューリングを行う上で必要な情報を得ることができない。そこで IPU を実装計算機として実装することにより、例えばモジュール内に設定された部分の実行時間の通知など、分散スケジューリングをより効率よく行える情報を得ることが可能となる。

が挙げられる。

VIOS III では、基本的に IPU を VIOS II と同じ考えかたで実装を行ったが、以下の変更を加えた。

- 従来、IPU で実行されるすべてのモジュールを、インタプリタ言語である IPU-p に変換して実行可能としていたが、C++にコンパイルして、IPU 内に組み込む方式に比べて数十倍実行速度が遅く、大規模な演算が多い画像処理では、たとえデバックのためといえども致命的な遅さであった。そこで VIOS III ではインタプリタとして実行可能なモジュールをメインフロー部分のみとし、他のモジュール部分は、VPE-p から直接 C++に変換し、コンパイルした後ダイナミックロード [11] することとした。この変更によっても、初期の目的である対話性とスケジューリング情報の獲得には問題ないと考える。

- スレッドライブラリを SunOS4.1.x の LWP(Light Weight Process) からスレッドに独立な CPU 資源を割り当てることが可能な Solaris Thread[10] に変更した。この変更により、単にネットワーク分散環境だけでなく、マルチ CPU の計算機で１つの IPU から複数のスレッドを実行することにより、処理速度の向上が期待できる。

IPU 内には、IPU-p を理解し実行する ip(image processor) を用意する。ip は 1thread 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

要時に確保し画像演算処理を実行させることで、1 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（以後 ip1）と、メインフロー実行用 ip(以後 ip2) の 3 種類の ip から構成される。ip0 は、OM からの入力を受付、その入力に応じてモジュールの定義や、他の ip の起動を行う。ip1 では、IPU にあらかじめ定義されたモジュールまたはユーザの定義したモジュールを実行する。ユーザ定義モジュールは、ダイナミックロードライブラリを用いて動的に IPU にロードし、実行する。ip1 は、モジュールの修了時に解放される。ip2 では、インタプリタとして実行されるメインフローの実行を行う。

5.2 大域変数へのアクセス方法とその実装

通常モジュールは、IPU の数に応じてそのワーキングセットのタイプに応じた適当なワーキングセット群単位で、複数の IPU に振り分けられ分散実行される。大域変数は、その複数の IPU のうち、最も処理能力の高い計算機で実行されている IPU に保持する。これは、逐次処理部を実行するメインフローモジュールから大域変数のアクセスを、ネットワークを介さず行えるようにするためである。

なお VPE-p 上では、大域変数への排他制御を以下のように記述する。

```
#vios mutex 変数名 [on | off]
```

5.3 並列処理記述能力

拡張した 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[8] のうち、明らかに処理に並列性があるものについて 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、拡張した VPE-p で記述が可能であることが確かめられた。

- 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)

なお松山らの分類でのスキャン型演算、展開型演算は、今回提案した拡張でも記述不可能である。しかし、各画素に別々の処理を割り当てる展開型演算は、一般的な画像処理アルゴリズムとしては適応例が少なく、またデータ並列を前提とする VPE-p で導入することは困難である。

さらに、松山ら実装した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、ジオメトリックハッシングを含む並列認識処理過程 [4] についても記述が可能であることが確かめられた。

```
module Histogram(in:input,
                 histogram:output)
int   in    on box[32][32];
int   histogram[];
#vios mutex histogram off add
{
   concurrent
   {
      for (int x = 0; x < 32; x++)
         for (int y = 0; y < 32; y++)
            histogram[in[x][y]]++;
   }
}
```

図 6: ヒストグラム演算プログラム

5.4 実行性能評価

次に、VIOS III 上で以下の画像処理を実行することにより、プログラマブル大域変数バッファの有効性、及び台数効果性能の評価を行った。

5.4.1 プログラマブル大域変数バッファの評価

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、入力画像からヒストグラムを作成する図 6で示したプログラムを実行することにより、バッファ統合の有効性を検討した。その結果を表 1に示す。なお表中のヒストグラム演算の列は、実際に各 IPU でヒストグラム演算を行うために必要であった実処理時間であり、分散処理コストは、ネットワークを介したヒストグラム配列へのアクセスコストである。表に示す通り、バッファ統合なしの場合は、ヒストグラムを格納する配列をアクセスする毎にネットワークを介したアクセスおよび排他制御が必要となり膨大な分散処理コストが必要となる。一方バッファ統合ありの場合は、ヒストグラム演算終了後の統合処理のみが必要であり、大幅な処理時間の短縮結果が得られた。

	バッファ統合なし	バッファ統合あり
ヒストグラム演算 (秒)	0.6	0.6
分散処理コスト (秒)	15206.4	1.9
合計処理時間	15207.0	2.5
速度比	1.0	6082.0

表 1: バッファ統合手法の有無によるヒストグラム算出アルゴリズムの実行時間

5.4.2 密結合計算機による実行性能の評価

次に VPE-p で記述した松山らの実装した並列認識処理過程 [4] 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 上で実行することにより、IPU の増加による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た。実行結果を表 2 に示す。表に示す通り、8台の CPU を用いて実行した場合、約6倍の速度向上結果を得た。

IPU 数	1	2	4	8
処理時間 (秒)	612.6	329.4	167.0	101.6
速度比	1.00	1.86	3.67	6.03

表 2: 並列認識処理過程の並列処理効果

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 の管理その他処理のスケジューリングを行う OM の３つ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。

並列画像処理記述用言語 VPE-p の拡張を行い、様々な並列性を持つ画像処理アルゴリズムの記述が可能であることを確認した。また、ネットワーク大域変数へのアクセス手法として、プログラマブル大域変数バッファの提案、実装を行い、実験によりシステムの有効性を示した。

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、ネットワークで結合された複数のワークステーション上での実装を仮定しているので、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。

- **分散スケジューリングアルゴリズムの強化**

  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つけ割り振りを最適化するようなスケジューリングを検討中である。

- **画像処理ライブラリの充実化**

  VIOS の処理ライブラリは、基本演算しか用意されていない。そこで汎用的に使えるように最低限必要なライブラリをそろえる。

なお本研究の一部は財団法人中部電力基礎技術研究所の助成により行われた。

参考文献

[1] 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)

[2] H.MATSUO, A.IWATA:“A distributed image processing enviroment VIOS II”, ACCV93, pp.715-718(1993)

[3] 加藤博之、松尾啓志、岩田彰:“分散画像処理環境 VIOS II の検討”, 通学技報, Vol.PRU92-166, pp.79-86(1993)

[4] 松山 隆司:“再起トーラス結合アーキテクチャに基づく並列画像理解システムに関する研究”, 平成７年度科学研究費補助金（一般研究（B））研究成果報告書 (課題番号 05452359)

[5] http://www.epm.ornl.gov/pvm/pvm_home.html

[6] http://www.osc.edu/lam.html#MPI

[7] http://www.khoral.com/khoros/

[8] http://www.ifi.uio.no/~blab/Software/Xite/

[9] 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,pp.193-203(1991/4)

[10] SunSoft:“Solaris 2.5 Multithread Programming Guide”

[11] SunSoft:“Solaris 2.5 Linker and Libraries Guide”