PCクラスタを利用した実時間並列画像処理環境RPV

有田, 大作
九州大学システム情報科学研究院知能システム学部門

濱田, 義雄
九州大学システム情報科学研究院知能システム学部門

米元, 聡
九州大学システム情報科学研究院知能システム学部門

谷口, 倫一郎
九州大学システム情報科学研究院知能システム学部門

出版情報：電子情報通信学会論文誌. J84-D-II (6), pp.965-975, 2001-06. 電子情報通信学会
バージョン：
権利関係：

PCクラスタを利用した実時間並列画像処理環境RPV

有田　大作[†]　濱田　義雄[††]　米元　聡[†]　谷口倫一郎[†]

RPV: A Real-time Parallel Image Processing Environment on PC-Cluster

Daisaku ARITA[†], Yoshio HAMADA[††], Satoshi YONEMOTO[†], and Rin-ichiro TANIGUCHI[†]

あらまし　本論文では，分散並列計算機の1種であるPCクラスタを利用し，実時間並列画像処理アプリケーションを構築するためのプログラミング環境であるRPVを提案する．RPVを利用することによりアプリケーションプログラマはノードPC間のデータフロー情報と各ノードPCで実行されるデータ処理タスクのみを記述すればよく，まずこの記述法について述べる．また，このためには，データ転送機構，同期機構，エラー処理機構をRPV側で用意する必要があるので，次にこれらの実装方法についても述べる．最後に実験によってRPVの性能の評価を行う．

キーワード　動画像処理，実時間処理，並列画像処理，PCクラスタ，分散協調視覚

1. まえがき

1.1 背景と目的

近年，コンピュータビジョンの分野では，複雑な対象や環境を理解するために，複数のカメラから得られた多視点情報を利用するという研究が盛んに行われている．複数のカメラを用いる場合の問題点，特に，実時間の画像処理を行おうとする場合の問題点として，1台の計算機に接続できるカメラ台数の制限が挙げられる．これは，計算機のI/O能力の限界や接続可能な装置数の制限といった物理的な制約によるものであり，多数のカメラを利用しようとする場合大きな問題となってくる．また，視点数の増加に伴って必要とされる計算能力も大きくなってくる．これに対して，計算機とカメラを対にした観測ステーションをネットワークで接続し，分散計算システムとしてコンピュータビジョンシステムを構築しようという研究が進められている[1]．このような分散計算システムでは，カメラの台数に特に制約はなく，必要に応じて観測ステーションを増加させるだけで視点数を増やすことができ，観測ステーションの台数に見合っただけの計算能力が得られるという大きな利点がある．このような背景から，我々は，汎用的なPCをネットワーク接続したPCクラスタ上での実時間画像処理に関する研究を行っている．汎用のPCを利用することにより，コストパフォーマンスが高く，柔軟性に富んだシステムを構築することが可能である．

従来より，数値計算やデータベースの世界では，ネットワーク接続された汎用のワークステーションやPCを分散型並列計算機として利用することが行われてきており，並列プログラムの記述性や移植性を高めるために，標準的なプログラミング環境も提案されている[2],[3]．また，画像処理に関しても静止画を対象とした分散処理システムもいくつか開発されている[4]〜[6]．しかし，実時間画像処理では，フレーム単位で次々と到着する画像データを滞りなく処理し，結果を出力することが要求されるため，その分散並列計算機上での実現は容易ではない．すなわち，実時間画像処理を分散並列計算機上で実現するためには，各通信経路における1フレーム分のデータ転送，及び，各ノードPCにおける1フレーム分のデータ処理が1フレーム時間（通常よく使われるカメラでは1/30秒）で終了することを保証しなければならない．このためには

- 高速かつ低負荷なデータ転送機構
- 1フレーム時間内にデータ転送とデータ処理が終了していることを保証する同期機構

[†] 九州大学大学院システム情報科学研究院，春日市
Faculty of Information Science and Electrical Engineering, Kyushu Univ., 6–1 Kasuga-koen, Kasuga-shi, 816–8580 Japan
[††] 富士通株式会社システム本部，千葉市
System Engineering Group, Fujitsu Limited, 1–9–3 Nakase, Mihama-ku, Chiba-shi, 261–8588 Japan

● データ転送やデータ処理に遅れが生じたときに正常な状態に復帰するためのエラー処理機構

が必要である．また，応用システムを開発，維持する上では，これらの実時間処理機構の複雑なプログラミングも問題となってくる．本論文では，これらの機能を提供することにより実時間並列画像処理を PC クラスタ上で容易に開発できるプログラミング環境 RPV（Real-time Parallel Vision）を提案し，その概要，実時間処理機能の実現方法，性能評価について述べる．

1.2 他研究との比較

本研究のように複数のカメラを複数の計算機に接続して動画像処理を行う研究としては以下のものが挙げられる．

● Kanade らの 3D Dome [7]

これは 49 台の同期のとれたカラーカメラと 17 台の PC を利用し，動画像を撮影するものである．このシステムは複数視点の動画像を記録するためのものであり，オンラインで実時間動画像処理を行うものではない．

● Davis らの Keck Laboratory [8]

これは 64 台の同期のとれたカメラ（48 台の濃淡カメラと 16 台のカラーカメラ）と 17 台の PC を利用し，オンラインで動画像処理を行うものである．しかし，このシステムは高速ネットワークを利用しておらず，実時間動画像処理を目指すまで至っていない．

● 亀田らの研究 [9]

これは 4 台のカラーカメラと 9 台のワークステーションを利用し，オンラインで動画像処理を行うものである．このシステムは高速ネットワークの一種である ATM を利用しており，動画像を高速転送しながら処理を行っているが，同期機構やエラー処理機構を実現していないので実時間性を保証できない．また，カメラ間の同期を行っていないため，撮影時刻のずれが生じ，アプリケーションによっては十分な精度が出ないことが考えられる．

これに対し，本研究では同期のとれたカメラ，高速ネットワークを利用し，データ転送機構，同期機構，エラー処理機構を実現しており，実時間並列動画像処理環境を構築している．

2. RPV の概要

2.1 システム構成

本研究で用いる PC クラスタシステム（図 1）は，14 台のノード PC からなっている．そのうち，6 台

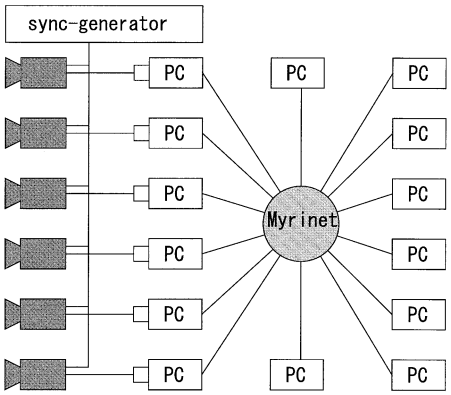


図 1 PC クラスタシステム
Fig. 1 PC-cluster system.

に CCD カメラが接続されており，すべての CCD カメラは同期信号発生装置により同期がとられている．各ノード PC は CPU を 2 個搭載している AT 互換機であり，OS には Linux を用いている．また，各ノード PC はギガビット LAN の一種である Myrinet によって相互に結合されており，Myrinet 上の通信には RWCP によって開発された PM 通信ライブラリ [10] を利用している．Myrinet 上の PM 通信ライブラリの特徴として，

- 7.5 マイクロ秒の低レイテンシ
- 118 メガバイト/秒の高スループット
- 全 2 重通信をサポート
- ゼロコピー通信をサポート
- 割込み受信をサポート

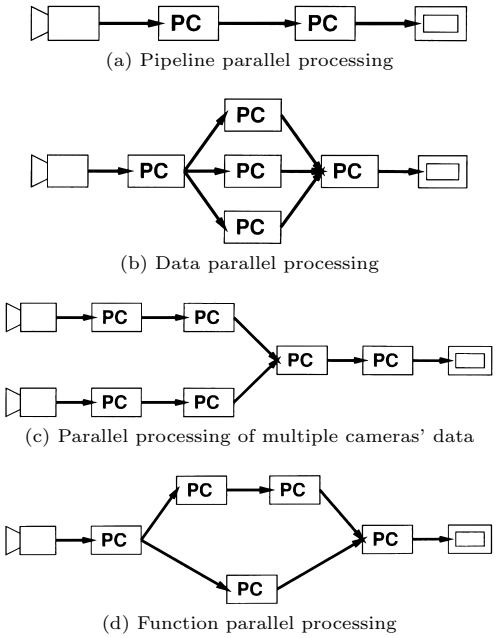
といった点が挙げられる．このデータからもわかるように，非圧縮フルサイズカラー画像（640 × 480 画素/フレーム，4 バイト/画素）を 30 フレーム/秒で転送するための条件，すなわち毎秒 $640 \times 480 \times 4 \times 30 = 36864000 \simeq 35$ メガバイト以上のスループット，を十分満たしている．また，ゼロコピー通信と割込み受信のサポートにより，受信側のノード PC は CPU に負荷をかけることなくデータの受信を行うことが可能である．

2.2 並列処理方式

RPV では，以下の並列処理方式を実現している．

● パイプライン並列処理

図 2 (a) に示すように，処理の段階ごとに 1 台の計算機を割り当てる並列処理であり，実時間画像処理では，一つのフレームに対する処理を各計算機で順々に



(a) Pipeline parallel processing

(b) Data parallel processing

(c) Parallel processing of multiple cameras' data

(d) Function parallel processing

図 2　並列処理方式
Fig. 2　Parallel processing schemes.

行うという方式をよく用いる．この場合，ある時刻では，各計算機は異なったフレームに対する処理を並列に行っていることになる．したがって，計算機台数を増やすと，並列に処理するフレーム数が増えるため，データ処理能力は向上するが，あるフレームが入力されて，その結果が出力されるまでに必要な時間は，計算機台数分のフレーム数だけ遅れることになりレイテンシは増大してしまう．

- データ並列処理

図 2 (b) に示すように，データを分割し複数の計算機で同時に処理する並列処理であり，実時間画像処理では以下の二つの方式が考えられる．

[空間方向データ並列処理]
各フレームのデータを分割して各計算機で処理を行う方式．画像を分割して並列に処理を行うなど，画像処理ではよく利用される並列処理である．また，フレームデータを分割するのではなく，複数のカメラによって同時に獲得されたデータに対して複数の計算機で同時に処理する方式も空間方向データ並列処理の一つである（図 2 (c) 参照）．

[時間方向データ並列処理]
フレームごとに異なる計算機で処理を行う方式．視体積交差法のような分割後の処理の負荷に偏りが生ずるデータ処理タスクや，画像領域分割のようにフレームデータの分割が難しいデータ処理タスクを並列化するときに利用できる．

- 機能並列処理

図 2 (d) に示すように，同じフレームのデータに複数の計算機で同時に異なる処理を行う並列処理であり，原理的には，空間方向データ並列処理と同様の効果がある．ただし，同時に異なる処理を行わなければならないので，処理の内容によってはこの並列処理を実現できるとは限らない．また，計算機間で処理量が異なると，処理量の最も多い計算機の処理時間に，全体の処理時間が拘束され，計算機台数分の処理性能の向上が得られなくなってしまう．

2.3　プログラミングの概要

実時間並列画像処理アプリケーションを構築するとき，アプリケーションによって異なるのは，ノード PC 間のデータフローと各ノード PC におけるデータ処理タスクのみである．これら以外の機能，具体的には

- データ転送機構
- 同期機構
- エラー処理機構

についてはすべてのアプリケーションで共通に利用できる．これらの機能を実現するためには，ハードウェア，OS，ネットワークについての多くの知識が必要であり，プログラミングは容易ではない．そこで，RPV ではこれらの機能を C++のクラスライブラリとして提供することで，アプリケーションを容易に記述できるようにしている．これにより，アプリケーションプログラマはデータフロー情報とデータ処理タスクを記述するだけでよいことになる．

2.3.1　ノード PC 間のデータフロー情報の記述

図 3 で示したクラス RPV_Connection は，ノード PC 間のデータフロー情報をもつクラスである．各ノード PC は，このクラスのもつ情報をもとに，ノード PC 間のデータの送受信を行う．通常このクラスは，ノード PC 間のデータフロー情報を記述したファイル（以下コネクションファイル）によって初期化される．このコネクションファイルには PC クラスタ全体のデータフローを記述する．このようにデータフロー情報は一つのファイルにまとめて記述されるため，データフロー情報の管理が容易であるだけでなく，プログラムの修正，再コンパイルを行うことなく，利用するノード PC と各ノード PC で行う処理を変更することがで

```
class RPV_Connection{
    int myPC_no;              自ノード PC 番号
    char* keyword;            処理内容のキーワード
    int input_PC_num;         受信する同期データの数
    int* input_PC;            受信同期データの転送元ノード PC 番号
    int* input_data_size;     受信する同期データのサイズ
    int input_frame_num;      同時に使用する受信同期データのフレーム数
    int input_buf_num;        受信同期データ用バッファの数
    int output_PC_num;        送信データの数
    int* output_PC;           送信データの転送先ノード PC 番号
    int* output_data_size;    送信データのサイズ
    int output_buf_num;       送信バッファの数
    int ainput_PC_num;        受信非同期データの数
    int* ainput_PC;           受信非同期データの転送元ノード PC 番号
    int* ainput_data_size;    受信非同期データのサイズ
    int ainput_data_num;      同時に使用する受信非同期データのフレーム数
    int connect_PC_num;       使用するノード PC の数
    int* connect_PC;          使用するノード PC 番号列
};
```

図 3　クラス RPV_Connection
Fig. 3　Class RPV_Connection.

きる．コネクションファイルは，1 行に 1 台のノード PC のデータフロー情報を記述し，各行には以下の項目が記述される．

PCno　PC クラスタ内のノード PC に付けられた番号
keyword　ノード PC で行うデータ処理を指定する文字列
i_PC　受信同期データの転送元ノード PC 番号
i_size　受信同期データのサイズ
i_num　同時に使用する受信同期データの数
o_PC　送信データの転送先ノード PC 番号
o_size　送信データサイズ
ai_PC　受信非同期データの転送元ノード PC 番号
ai_size　受信非同期データのサイズ
ai_num　同時に使用する受信非同期データの数

2.3.2 各ノード PC でのデータ処理アルゴリズムの記述

RPV におけるデータ処理は図 4 に示す流れで行われる．まず，初期化 1 でコネクションファイルの読込みが行われ，クラス RPV_Connection が初期化される．次に，変数 keyword によって分岐を行い，ノード PC ごとに引数の異なる関数 RPV_Invoke を起動する．このようにするのは，プログラムの修正，再コンパイルを行うことなくコネクションファイルを修正するだけで，各データ処理タスクを実行するノード PC を変更できるようにするためである．図 5 に示すのが関数 RPV_Invoke の仕様であり，関数 RPV_Invoke の引き数によってそのノード PC で実行されるデータ処理タスクが指示される．また，第 2 引き数の sync_mode で，同期機構に関する選択を行う（**3.4**参照）．

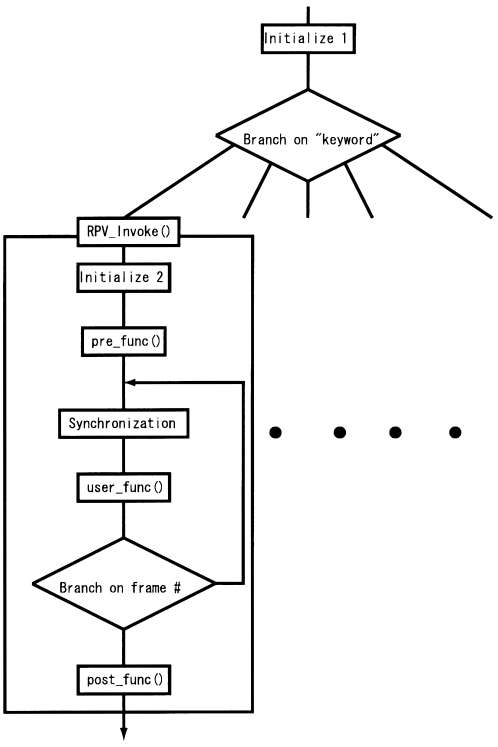


図 4　処理の流れ
Fig. 4　Processing flow.

```
void RPV_Invoke(
    RPV_Connection* connect,                          データフロー情報
    struct RPV_FSM sync_mode,                         同期モード
    int frame_num,                                    処理するフレーム数
    void* (*pre_func)(void*),                         ループ前に実行される前処理関数
    void* pre_func_arg,                               pre_func の引数
    void* (*user_func)(RPV_Input*, RPV_Output*,
                       RPV_Ainput*, void*),           ループ中に実行される関数
    void* user_func_arg,                              user_func の引数
    void* (*post_func)(void*),                        ループを出た後に実行される後処理関数
    void* post_func_arg                               post_func の引数
);
```

図 5　関数 RPV_Invoke
Fig. 5　Function RPV_Invoke.

関数 RPV_Invoke の動作は以下のとおりである．

（1） 初期設定として，モジュールの起動（**3. 1**参照），通信路の初期化を行う．

（2） RPV_Invoke の引き数に従って，指定された関数を起動しながら処理終了まで動作する．具体的には以下の動作を行う．

（a） 前処理関数 pre_func を実行する．

（b） 処理開始割込み信号を待つ．

（c） ユーザ関数 user_func を実行し，1 フレーム分のデータを処理する．

（d） frame_num フレーム分のデータを処理が済んでいなければ，(2b）へ戻る．

（e） 後処理関数 post_func を実行し，処理を終了する．

（3） すべてのノード PC で処理が終了したことを確認し，RPV 全体の処理を終了する．

ユーザ関数 user_func の引き数は，三つの入出力データへのポインタと一つのアプリケーションプログラマが自由に利用できる引き数からなる．受信同期データ引き数と受信非同期データ引き数には，システムによって自動的に必要なデータが渡され，ユーザ関数はそれらのデータを読み込むことができる．また，送信データ引き数には，ユーザ関数から書込み可能であり，書込みが終了すると自動的にそれらのデータは送信される．このようにユーザ関数は，データの送受信や同期について考慮する必要がなく，1 枚の画像を入力とし，それを処理し，結果を出力する静止画像処理と同じ形式で記述可能である．そのため，実時間動画像処理ということをほとんど意識することなく，容易にユーザ関数をプログラミングできる．

3. RPV の実装

本章では，RPV におけるデータ転送機構，同期機構，エラー処理機構の実現方法について述べる．

3. 1 実装の基本方式

並列動画像処理を効率的に実行するために，本システムの各ノード PC では，図 6 に示す四つのモジュールが，UNIX プロセスとして並行動作している．これにより，データの受信，処理，送信を並行に行うことができ，外部からのデータの受信にも即座に対応できるようになっている．また，データの送受信のためのバッファは，複数のモジュールからアクセスできるように共有メモリ上に実現されている．以下，各モジュールの動作について説明する．

- データ受信モジュール（DRM）

データ受信に関する情報のやり取りを行うモジュール．

－ 転送元ノード PC からの各フレーム分のデータ転送が終了した知らせを受け取り，データが書き込まれた受信バッファの領域を書込み不可（読込み可能）にする．

－ 処理が済んでデータ書込み可能になった受信バッファの領域をチェックし，転送元ノード PC に伝える．

- データ処理モジュール（DPM）

実際の画像処理を行うモジュール．

－ 最初に関数 pre_func を，最後に関数 post_func を呼び出す．

－ 処理開始の割込み信号を受けとると，受信バッファの読込み可能領域からデータを読込んで画像処理を行う（実際は関数 user_func を呼び出す）．処理結果は送信バッファの書込み可能領域に書き込み，その領

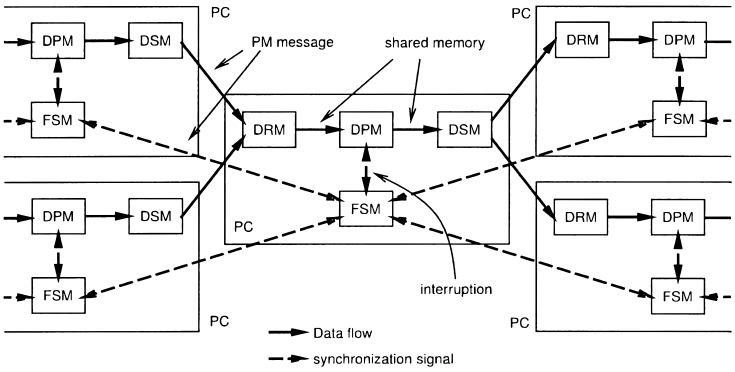


図 6　各ノード PC のモジュール構成
Fig. 6　Modules in each PC.

域を書込み不可とする．

– 処理が終了すると，当該の受信バッファ領域を書込み可能にする．

– 処理結果の書込みが終了すると，書込みの終了を通知する割込み信号を DSM に送る．

• データ送信モジュール（DSM）

データ送信に関する情報のやり取りを行うモジュール．

– 処理モジュールからの各フレーム分のデータ書込みが終了した知らせを割込み信号として受け取り，そのデータを転送先ノード PC に送信する．

– 送信が終了したデータの格納されていた領域を書込み可能とする．

• フレーム同期モジュール（FSM）

ノード PC 間の同期をとるためのモジュール．

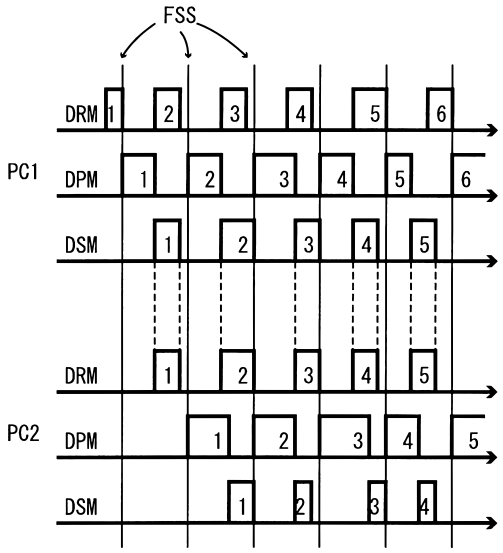
– FSM 同士の通信によって，**3. 3**で述べるフレーム同期信号（Frame Synchronization Signal：FSS）をデータの流れにそって上流から下流へ向けて送受信する．

– FSS を受け取ると，処理開始を通知する割込み信号を DPM に送る．

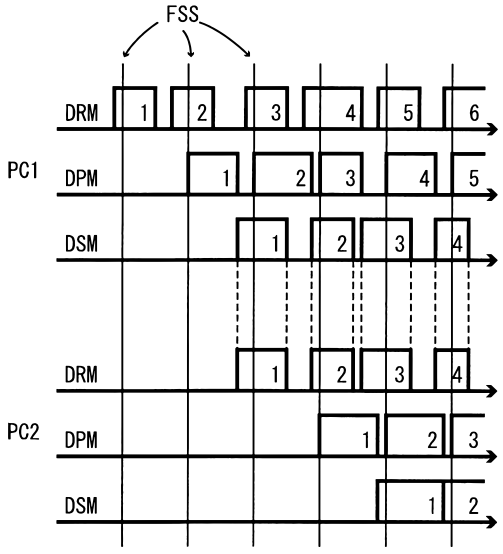
3. 2　ノード PC における処理の並列化

システム全体としての性能を向上させるためには，各ノード PC での性能を向上させる必要がある．そこで，ここでは，ノード PC 内のモジュールの並列化について考察する．**3. 1**で述べたモジュールのうち CPU を多く利用するものは，DPM 以外に DSM がある．これは PM 通信ライブラリではデータ転送の単位が最大 8 キロバイトのため，画像のような大きなデータは分割して送信する必要があり，その制御に CPU が必要なためである．その他のモジュールはほとんど CPU を利用しない．また，Myrinet は全 2 重通信可能であるため，データ受信とデータ送信が競合することもない．したがって，データの転送，処理に直接関与するモジュールである DRM，DPM，DSM での処理は，一般に図 7 (a) のようになる．この図で，上半分と下半分がそれぞれ一つのノード PC の動作を表しており，それぞれのノード PC の中では，上から順に DRM におけるデータ受信時間，DPM におけるデータ処理時間，DSM におけるデータ送信時間を表している．また，上段のノード PC から下段のノード PC へと同期データが転送されている．この図のように，DPM によるデータ処理の終了後，DSM によるデータ送信が行われるという排他的な処理になる．

このとき画像の実時間処理を実現するためには，DPM によるデータ処理時間と DSM によるデータ送信時間の合計が 1 フレーム時間以内となることが必要となる．しかし，1 台のノード PC に 2 個の CPU を搭載すれば，DPM と DSM を並列動作させることが可能となり，図 7 (b) のように，t 番目のフレームのデータ処理と $t-1$ 番目のフレームのデータ送信をオーバラップさせることが可能となる．つまり，1 フレーム時間内で画像データの処理に CPU を多く割り当てられるようになる．このような点を考慮し，本研究の PC クラスタでは各ノード PC に 2 個の CPU を搭載している．ただし，データ処理とデータ送信をオーバラップさせない場合に比べてオーバラップさせる場合は，1 フレーム時間当りの処理可能量が増加する（スループットが増大する）代わりにレイテンシが 2 倍になってしまう．したがって，どちらの方式を採用するかをアプリケーションの計算量やノード PC の台数な



(a) Without overlaping DPM and DSM



(b) With overlaping DPM and DSM

図 7　モジュールのタイムチャート
Fig. 7　Time chart of modules.

どを考慮してアプリケーションプログラマが選択できるようになっている．

3.3 時刻管理

分散並列計算機環境において実時間画像処理を行う場合，時刻の管理が重要である．これは，複数のカメラによって撮影された画像が同一時刻のものでなければならず，また各計算機でのデータの処理と計算機間の同期データの転送を同じ時間間隔で行わなければならないからである．このために，以下のことを行っている．

- すべてのカメラに外部同期信号を与えることによって，すべてのカメラを完全に同期させる．これにより，全く同じ時刻の画像を撮影することができる．
- カメラが接続された各ノード PC で同時刻に画像獲得を開始する．時刻を合わせるために NTP [11] を導入し，ノード PC の内部時計を一致させている．NTP を使うことによって，数ミリ秒以下の精度でノード PC の時刻を合わせることができるので，33 ミリ秒ごとの画像獲得に対しては十分な精度である．また，データにはフレーム番号が付けられており，これを比較することにより，異なるカメラからの画像の獲得タイミングが同じであるかどうかを判断できる．
- カメラが接続されたノード PC は，カメラからの垂直同期信号のタイミングに合わせて，FSS を発生する．FSS は同期データの流れに沿って上流のノード PC から下流のノード PC へと転送される．この FSS によって各ノード PC 間のデータ処理とデータ転送の同期をとることができる（同期機構の詳細は **3.4**を参照）．

3.4 同期機構

本システムでは，実時間でデータを受信，処理，送信することが要求される．これらのデータ転送，処理を滞りなく実行するために，ノード PC 間の同期をとることが必要である．そこで，本システムでは，

- データ転送同期
- データ処理同期

の 2 種類の同期機構を提供し，同期の実現のために FSS を利用している．

データ転送同期は，DPM に同期データが到着しているかどうかをチェックする同期である．この同期により，前段のノード PC からの同期データ転送の遅れや前段までのノード PC における同期データのデータ落ちを検出することができる．図 8 は連続した二つのノード PC のタイムチャートである．DRM が受け取る同期データは，正常時は二つの FSS の間に受信される．そのため，FSS を受信したときに DPM は処理を開始することができる．しかし，(a) でデータ転送の遅れが起こっており，処理開始時に同期データが到着していない．このため，DPM は処理を開始することができず，プログラマによって指定されたエラー処理が実行されることになる．

一方，データ処理同期は，DPM の処理が 1 フレー

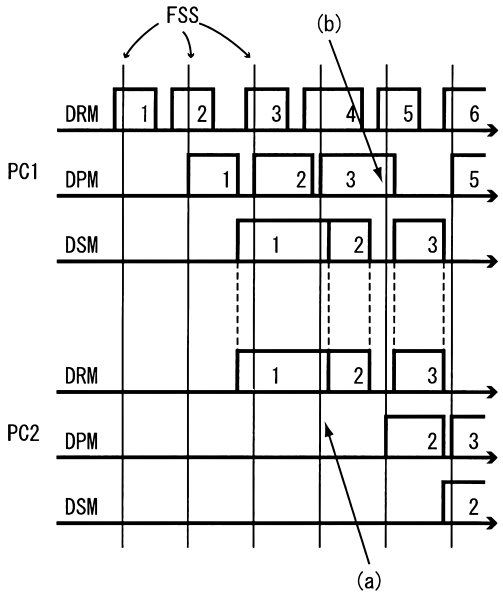


図 8　データ転送同期とデータ処理同期
Fig. 8　Data transfer synchronization and data processing synchronization.

ム時間以内に終了しているかどうかをチェックする同期である．図 8 において，(b) でデータ処理の遅れが起こっており，処理開始時になっても前のフレームのデータ処理が終了していない．このため，そのままでは DPM は処理を開始できず，また，DSM はデータを送信することができない．この場合もプログラマが指定したエラー処理が実行されることになる．

なお，本システムでは，ここで述べた同期機構に基づいた同期データ転送以外に，非同期データ転送もサポートしている．非同期データ転送は，フレームタイミングとは関係なくデータを転送するものであり，データは 1 フレームに何回も転送されても，あるいは全く転送されなくてもよい．通常は，フィードバックや協調処理のための制御信号などに利用される．アプリケーションプログラマは関数 user_func 中の任意のタイミングで最新の非同期データを転送することができる．

3.5　エラー処理機構

2 種類の同期機構におけるエラー処理として，以下の方法が考えられる．

（a）　データ転送の遅れ

（1）　データが到着するのを待って，処理を開始する．

（2）　次の FSS まで処理を行わない．次の処理では最新のデータを処理する．

（b）　データ処理の遅れ

（1）　処理を打ち切り，次のフレームのデータに対する処理を開始する．

（2）　終了するまで処理を継続する．次の処理では最新のデータを処理する．

これらのエラー処理の組合せにより，以下の 3 種類のエラー処理モードを用意し，アプリケーションプログラマはその中から適切なものを選択することができる．なお，(a) の (2) は次のフレームのデータに対する処理を行うために現在のフレームのデータを落とすエラー処理であり，(b) の (1) は次のフレームのデータに対する処理を行うために現在のフレームのデータに対する処理を途中で打ち切るエラー処理であるから，これらのエラー処理の組合せは効果が重複しており，無意味なので考えない．

- データ落ち型（Data missing）

(a) は (2), (b) は (2) の組合せ（図 9 (a) 参照）．完全なデータだけを出力するが，エラーのときはデータ落ちが起こる．この方式は，処理が遅れたときに決められた時間以内に出力が得られないことになるので，厳密には実時間システムとはいえなくなる．しかし，データを落とすことによってすぐに遅れの状態から正常な状態に復帰できるので，実際の動画像処理としては最も利用される方式である．

- 不完全データ転送型（Incomplete data）

(a) は (1), (b) は (1) の組合せ（図 9 (b) 参照）．データ落ちは起こらないが，データ処理が遅れたときは不完全なデータが出力される．この方式のみが厳密な意味での実時間処理となる．ただし，計算機の処理能力と比べて処理量が多くなりすぎると，完全な出力が全く得られなくなってしまうので，プログラマが負荷分散を慎重に行う必要がある．なお，このとき出力されるデータは，以下の中からアプリケーションプログラマが選択する．

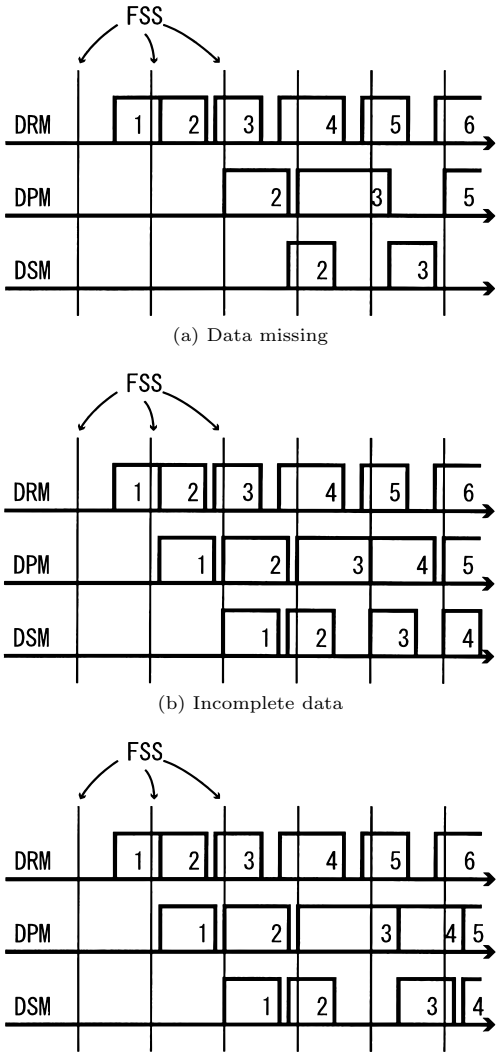
（1）　処理途中のデータ

（2）　前フレームの処理結果

（3）　あらかじめ指定されたデータ

- 完全保持型（Complete queuing）

(a) は (1), (b) は (2) の組合せ（図 9 (c) 参照）．データ落ちも起こらず，完全なデータだけを出力する．しかし，この方式は，一度エラーが起こるとそれ以降の実時間性の保証ができず，更に最悪の場合は受信バッ



(a) Data missing

(b) Incomplete data

(c) Complete queuing

図 9　同期におけるエラー処理方式
Fig. 9　Error recovery modes.

ファがあふれてしまう．

4. 評価実験

本論文では，基礎実験による基本性能の評価と，実アプリケーションによる RPV 全体の評価を行った．これらの実験に用いたノード PC には，CPU として PentiumIII（450 MHz）を 2 個，メモリを 384 メガバイト搭載している．

4.1 基礎実験による評価

まず，画像データの転送能力は表 1 のようになった．

表 1　画像 1 枚当りの転送時間（32 ビット/画素，単位はミリ秒）
Table 1　Data transfer time for one frame. (32 bit/pixel, ms)

画像サイズ	最小時間	平均時間	最大時間
640 × 480	11.8	12.1	12.4
320 × 240	2.9	3.1	3.4

表 2　データ転送量に対するデータ処理能力の変化
Table 2　Change of data processing performance to the amount of data transfer.

画像サイズ	処理画素数	比率
320 × 240	85.7×10^3	1
640 × 240	85.0×10^3	0.992
640 × 480	84.4×10^3	0.985

これは各サイズの画像を同期データ転送によって転送するときの，送信側ノード PC における画像 1 枚当りの送信時間である．この結果より，本システムは，一般的なテレビカメラの解像度で 30 フレーム/秒の画像転送が可能であり，転送時間も安定しているため，実時間画像処理システムとして利用できることがわかる．

次に，データ処理量とデータ転送量がシステム性能に与える影響について調べたのが表 2 である．これは，データ処理とデータ転送をオーバラップさせ，3 種類の大きさの画像に対する平均値フィルタ処理を行い，30 フレーム/秒時における処理可能最大画素数を計測したものである．データ転送量を大きくしたときもデータ処理能力はほとんど減少していない．これは，データ転送による負荷がほとんど隠ぺいされ，オーバラップが有効に機能していることを意味している．

また，この実験において，データ引渡しや同期などに要するシステム時間を計測したところ，0.06 ミリ秒であった．この結果よりシステム時間は極めて短くほとんど負荷をかけていないことがわかる．ただし，処理遅れなどのエラーが生じてエラー処理が実行された場合，表 3 に示す時間がシステム時間に上乗せされる．

最後に，RPV の処理能力を表 4 に示す．これは，640 × 480 の大きさ画像に対して，各画像処理を行った実行時間である．2 値化，エッジ検出の処理は濃淡画像を対象とし，その他の処理は，カラー画像を対象としている．RPV の列は，1〜3 台のノード PC に画像を分割してデータ分割並列方式で処理を行った場合の実行時間を示している．この結果が示すように，1 台の PC で比べると，RPV を用いた場合の方が時間がかかっている．これは，同期によるオーバヘッドやデータ転送を行うプロセスが別に動いていることによ

表 3　エラー処理にかかる時間（単位はミリ秒）
Table 3　Error recovery time. (ms)

エラー処理モード		時間
データ落ち型		0.780
不完全データ型	処理途中のデータ	0.083
	前フレームの処理結果	2.156
	あらかじめ指定されたデータ	1.208
完全保持型	バッファがあふれないとき	0.048
	バッファがあふれたとき	0.830

表 4　画像処理アルゴリズムの実行時間（単位はミリ秒）
Table 4　Image processing time. (ms)

画像処理アルゴリズム	単一CPU	RPV		
		1	2	3
平均値フィルタ（3 × 3）	96.1	107.1	56.2	37.3
背景差分	35.4	48.0	21.6	14.4
テンプレートマッチング（8 × 8）	2289.5	2529.1	1282.2	858.2
2 値化	11.8	12.7	5.9	3.9
エッジ検出	107.9	111.2	52.3	34.6

るメモリアクセスの競合といった理由が考えられる．

また，この結果は，計算量の極めて多いアルゴリズムでは RPV による並列化を用いても実時間（30 フレーム/秒）の画像処理は容易ではないことを示している．しかしながら，CPU が提供しているマルチメディア拡張命令や，階層的アルゴリズム等の高速化戦略を利用することで，様々な応用で実時間画像処理が可能になると考えている．具体的な応用事例については，次節で簡単に述べる．

4.2 応用事例

RPV の性能を測るために，RPV を利用してモーションキャプチャシステムを構築した [12]．このシステムは図 10 のように，6 台のカメラ（1 台の鉛直カメラと 5 台の水平カメラ）と 14 台のノード PC を利用し，4 段階のパイプライン並列処理と 6 視点によるデータ並列処理を併用して，人間の動きをオンラインで獲得するものである．パイプライン並列処理の各段階の処理の内容は以下のとおりである．

- 画像獲得モジュール（ICM）

カメラからの画像を獲得し，画像を縮小（640 × 480 から 320 × 240 へ）する．

- 2 次元画像処理モジュール（2 DPM）

画像中の人物領域のシルエットを抽出し，そこから特徴点を追跡し，その 2 次元座標を求める．

- 3 次元画像処理モジュール（3 DPM）

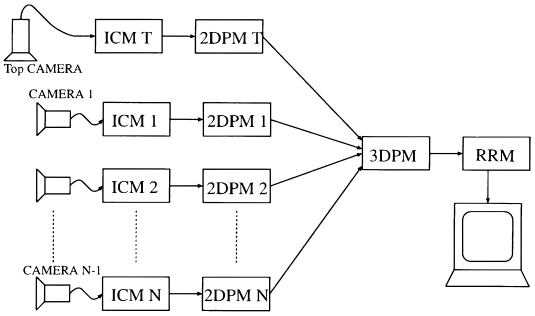


図 10　モーションキャプチャシステムの構成
Fig. 10　Motion capture system.

各視点における特徴点の 2 次元座標から，特徴点の 3 次元座標を求める．シルエットからの特徴点追跡は視点によって成否が左右されるので，6 台のカメラから適切なカメラを選択している．

- 実時間レンダリングモジュール（RRM）

特徴点の 3 次元座標を人物モデルにあてはめ，仮想空間に CG キャラクタを生成する．

以上の処理を 30 フレーム/秒で実行できることを実験によって確認した．

5. むすび

本論文では，実時間並列画像処理アプリケーションを作成するためのプログラミング環境である RPV について述べた．RPV を用いると，ユーザは，データ転送や同期，エラー処理といった分散システム上で実時間処理を行う際の問題に悩まされることなく，データフロー情報とデータ処理アルゴリズムを記述するだけでプログラミングを行うことができる．また，性能評価実験により，RPV は実時間画像処理を行うための十分な性能を有することを示した．

今後の課題としては

- ノード PC とネットワークについての負荷分析を利用した半自動負荷分散ツールの構築
- オーバヘッドの解析によるシステム性能の向上
- RPV プログラミングツールを用いた様々なアプリケーションの開発

が挙げられる．

謝辞　本研究は，日本学術振興会未来開拓学術研究推進事業「分散協調視覚による動的 3 次元状況理解」プロジェクト（JSPS-RFTF 96P00501）の補助を受けて行った．

文　　献

[1] 松山隆司，“分散協調視覚 —視覚・行動・コミュニケーション機能の統合による知能の創発，” 画像の認識・理解シンポジウム，分冊 I, pp.343–352, July 1998.

[2] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, PVM: Parallel virtual machine —A users' guide and tutorial for networked parallel computing, The MIT Press, 1994.

[3] Message Passing Interface Forum, “MPI: A message-passing interface standard,” Int. J. Supercomputer Applications and High Performance Computing, vol.8, no.3, pp.159–416, 1994.

[4] R. Taniguchi, Y. Makiyama, N. Tsuruta, S. Yonemoto, and D. Arita, “Software platform for parallel image processing and computer vision,” SPIE'97, Parallel and Distributed Methods for Image Processing, pp.1–10, July 1997.

[5] H. Matsuo, K. Nakada, and A. Iwata, “A distributed image processing environment VIOS III and it's performance evaluation,” Int. Conf. on Pattern Recognition, vol.II, pp.1538–1542, Aug. 1998.

[6] 松山隆司，浅田尚紀，青山正人，浅津英樹，“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.

[7] T. Kanade, H. Saito, and S. Vedula, “The 3D room: Digitizing time-varying 3D events by synchronized multiple video streams,” Technical Report of Robotics Institute, Carnegie Mellon University, CMU-RI-TR-98-34, Dec. 1998.

[8] L. Davis, E. Borovikov, R. Cutler, D. Harwood, and T. Horprasert, “Multi-perspective analysis of human action,” Int. Workshop on Cooperative Distributed Vision, pp.189–223, Nov. 1999.

[9] 亀田能成，太尾田健男，角所　孝，美濃導彦，“時空間の分割とビデオ画像のパイプライン処理による高速三次元再構成，” 情処学論，vol.40, no.1, pp.13–22, 1999.

[10] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato, “PM: An operating system coordinated high performance communication library,” in High-Performance Computing and Networking, ed. P. Sloot and B. Hertzberger, pp.708–717, Springer-Verlag, 1997.

[11] D.L. Mills, “Internet time synchronization: The network time protocol,” IEEE Trans. Commun., vol.39, no.10, pp.1482–1493, 1991.

[12] 星野竜也，米元　聡，有田大作，谷口倫一郎，“多視点動画像の輪郭線解析を用いた人間の実時間姿勢推定，” 第 5 回知能メカトロニクスワークショップ，pp.95–1001, Aug. 2000.

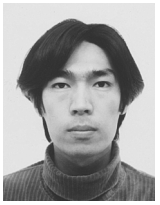
（平成 12 年 8 月 25 日受付，12 月 25 日再受付）

有田　大作　（正員）

平 4 京大・工・情報卒．平 10 九大大学院システム情報学研究科博士後期課程了．同年，同助手．博士（工学）．文書画像処理，画像処理における知識獲得，実時間並列画像処理の研究に従事．情報処理学会，映像情報メディア学会各会員．

濱田　義雄

平 10 九大・工・情報卒．平 12 同大大学院システム情報科学研究科修士課程了．現在，富士通（株）システム本部勤務．実時間並列画像処理の研究に従事．



米元　　聡　（正員）

平 7 九大・工・情報卒．平 12 同大大学院システム情報科学研究科博士後期課程了．同年，同助手．博士（工学）．画像処理，コンピュータビジョン応用に関する研究に従事．情報処理学会会員．

谷口倫一郎　（正員）

昭 53 九大・工・情報卒．昭 55 同大大学院工学研究科修士課程了．同年同大学院総合理工学研究科助手．平 1 同助教授．平 8 九大大学院システム情報科学研究科教授．工博．画像処理，コンピュータビジョン，並列処理に関する研究に従事．平 1 篠原記念奨励賞受賞．