Real-time Vision on a Distributed Parallel System

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

花田, 武彦
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：情報処理学会論文誌. 43 (sig11), pp.1-10, 2002-12. Information Processing Society of Japan
バージョン：

分散並列計算機による実時間ビジョン

有田大作[†] 花田武彦[††] 谷口倫一郎[†]

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としてRPVが提案されている．これは，実時間並列画像処理に必要な実時間データ転送機構，同期機構，例外処理機構を提供することにより，アプリケーションを容易に構築することを目指したものである．しかし，RPVの問題点としてレイテンシが大きくなってしまうことがあげられた．そこで我々は，この問題点を解決するために，ストリームデータ転送方式をRPVに導入することを提案する．これは，1フレーム分よりも小さくデータを分割し，それを単位にデータ転送やデータ処理を行う方式である．本論文では，まずRPVの概要を述べ，その問題点をあげる．次にストリームデータ転送方式について述べ，さらにこれを利用したアプリケーションによる性能評価を行い，有効性を示す．

Real-time Vision on a Distributed Parallel System

Daisaku Arita,[†] Takehiko Hanada[††] and Rin-ichiro Taniguchi[†]

RPV, which is a programming environment for real-time parallel computer vision on a PC cluster, has been proposed. It provides functions of real-time data transfer, synchronization mechanism and error handling to make it easy to construct real-time parallel computer vision applications. The problem of RPV is that the latency becomes long because of frame-based pipeline structure of the system. To make the latency shorter keeping high throughput, we have introduced stream data transfer mechanism, in which one frame data is divided into small packets and data transfer and data processing are applied to each packet. First in this paper, we discuss RPV and its problem. Secondly, stream data transfer mechanism is described and experimental results using a large-scaled vision application are shown to evaluate the stream data transfer.

1. はじめに

近年，コンピュータビジョンの分野では，複数のカメラから得られた多視点画像から複雑な対象や環境を理解する研究がさかんに行われるようになってきている．また，画像情報から仮想空間を構築しヒューマンインタフェースに利用する研究もさかんに行われており，実時間で動画像を処理する要求が高まってきている．これらを実現する場合には，複数のカメラから得られる大量の動画像情報を高速に処理する必要がある．複数のカメラを接続するためのI/O能力，および，大量の動画像情報を高速に処理する計算能力の点から考えると，1台の高速計算機を利用する方法では限界がある．そこで計算機をネットワークで接続した分散並列計算機を利用することが考えられる．分散並列計算機は，計算機の台数を増やすことにより，容易にI/O能力と計算能力を高めることが可能であるため，多視点動画像の実時間処理に適している．そこで我々はPCを利用した分散並列計算機（PCクラスタ）上で実時間ビジョンシステムを実現する方式について研究を進めている．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．

これまでも，ワークステーションやPCによる分散並列計算機が利用されてきており，このための並列プログラミング環境としてPVM[1)]やMPI[2)]が提案されてきている．また，静止画像処理に特化し，システム開発を容易にする並列画像処理向けプログラミング環境も提案されてきている[3),4)]．しかし，後者の並列画像処理向けプログラミング環境は静止画像処理のためのものであり動画像処理を行うことはできない．また，PVMやMPIでは以下で述べる実時間性を保証する機能がないため，実時間ビジョンシステムを開発するためにはそれらの機能を作成しなければならず，開発

† 九州大学大学院システム情報科学研究院
Department of Intelligent Systems, Kyushu University
†† 九州大学大学院システム情報科学府
Department of Intelligent Systems, Kyushu University

者にとって大きな負担となってしまう．

実時間ビジョンシステムでは，次々と獲得される動画像データを決められた時間内に処理し，結果を出力することが要求される．これを分散並列計算機上に実現するためには，各通信経路における 1 フレーム分のデータ転送，および，各ノード PC における 1 フレーム分のデータ処理が 1 フレーム時間内（NTSC のカメラを利用している場合は 1/30 秒内）に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，
- 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例外処理機構，

が必要である．また，ビジョンシステムを開発，維持するうえでは，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．我々はこれらの問題を解決するために，実時間ビジョンシステムを 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）を構築してきた[5]．しかし，従来の RPV の問題点としてシステム全体のレイテンシが大きくなってしまうことがあげられた．インタラクティブなアプリケーションでは，レイテンシが問題になることが多く，その削減は重要な問題となっている．そこで本論文では，レイテンシを削減するために新たに導入したストリームデータ転送方式について述べる．さらに，ストリームデータ転送方式を利用して構築した大規模な実時間ビジョンシステムの実例による性能評価も行う．

2. RPV の概要

ストリームデータ転送方式について説明する前に，従来の RPV について説明する．なお，従来の RPV が提供していたデータ転送方式のことをフレーム同期データ転送方式と呼ぶことにする．

2.1 システム構成

本研究で用いる PC クラスタシステムは，24 台のノード PC からなっている．さらに 9 台の IEEE 1394 ディジタルカメラ[6]が接続されており，すべての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（PentiumIII 700 MHz と PentiumIII 450 MHz のものがある）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 はスイッチ型ギガビット LAN の 1 つである Myrinet によって相互に結合されおり，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[7]を利用している．PM のスループットは 100 メガバイト/秒以上であり，非圧縮フルサイズカラー画像（640 × 480 画素/フレーム，1 画素 4 バイト）を 30 フレーム/秒で転送することが可能である（$640 \times 480 \times 4 \times 30 = 36864000 \simeq 35\,\mathrm{MB}$）．

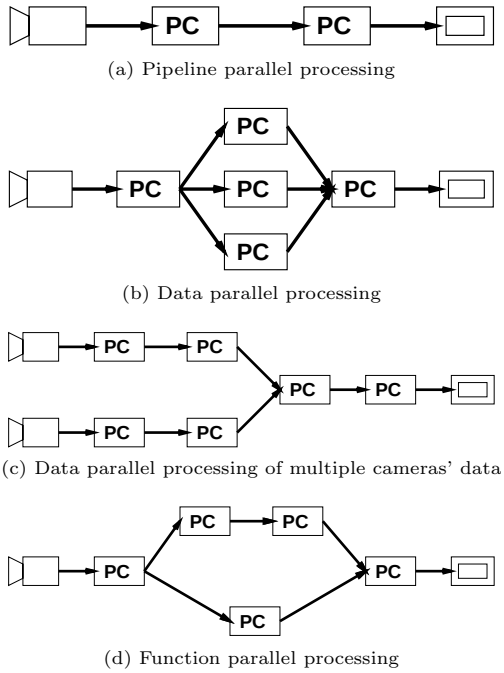


図 1 並列処理方式
Fig. 1 Parallel processing schemes.

2.2 並列処理方式

RPV では，**図 1** に示すような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 1 (a)）RPV による並列処理の基本形
- データ並列処理
  - データ分割によるデータ並列（図 1 (b)）パイプラインの中で特に処理に時間がかかる部分をデータを分割することによって並列処理するもの
  - 複数カメラによるデータ並列（図 1 (c)）複数のカメラから得られたデータに対して並列に処理し，パイプラインの途中で統合するもの
- 機能並列処理（図 1 (d)）パイプラインの中で特に処理に時間がかかる部分を処理の内容を分割することによって並列処理するもの

2.3 モジュール構成

並列動画像処理を効率的に実行するために，RPV における各ノード PC では，**図 2** に示す 4 つのモジュールが，Linux スレッドとして並行動作している．これ

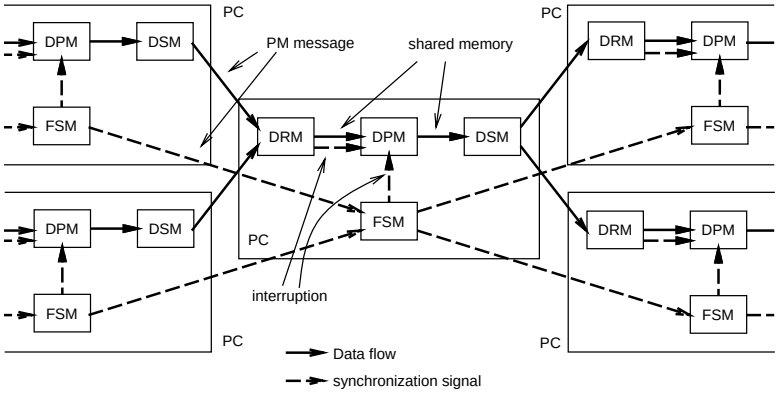


図 2 各ノード PC のモジュール構成
Fig. 2 Modules in each PC.

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，Linux スレッドを利用しているため，モジュール間のデータ送受信はメモリへのアクセスによって実現され，特別な負荷は生じない．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とりを行うモジュール．転送元ノード PC からのデータを受信し，受信バッファへ書き込む．データ転送では，PM における最大メッセージ長（約 8 KB）の制限もあり，それよりもデータサイズが大きい場合は複数のメッセージに分割して転送することになるので，DRM ではこれを順序どおりに受信バッファに格納していく．また，データを受信するたびに，DPM にデータ受信を通知する．

データ処理モジュール（DPM） アプリケーションプログラマが記述した画像処理を行うモジュール．フレーム同期データ転送方式においては，受信バッファのデータを読み込み，処理し，その出力結果を送信バッファへ書き込むフレームデータ処理関数（ユーザが記述）を 1 フレームに 1 回実行する．また，システムの動作開始時に 1 度だけ前処理関数（ユーザが記述）を実行する．さらに，動作終了時に 1 度だけ後処理関数（ユーザが記述）を実行する．

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とりを行うモジュール．送信バッファに書き込まれたデータを必要ならばメッセージ単位に分割して転送先ノード PC へ送信する．

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 どうしで通信を行い，2.4 節で述べるフレーム同期信号（Frame Synchronization Signal: FSS）を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 へ向けて送受信する．具体的には，上流ノード PC から FSS を受信したら即座に下流ノード PC へ FSS を送信する．複数の上流ノード PC から FSS を受信する場合は，すべての上流ノード PC からの FSS が揃ったときに下流ノード PC へ FSS を送信する．また，後述の同期機構のために，FSS を送信するのと同時に DPM にも通知する．

2.4 時刻管理

RPV ではカメラの時刻管理のために同期信号発生器を利用している．これにより，すべてのカメラのシャッタータイミングを合わせることができる．また，同期信号発生器からの信号の送出開始によってすべてのカメラが同時に動作し始めるので，各 PC の動作開始タイミングを合わせることもでき，同時に撮影された画像データに対し同じフレーム番号を付与することができる．これにより，どのフレームとどのフレームが同時刻に撮影されたものであるかを PC 間で容易に同定することができる．

カメラで撮影された画像はノード PC へと転送される．カメラから画像データを受信したノード PC は，1 フレーム分の画像データを受信し終わるとただちに FSS を次段のノード PC へと送信する．これにより，1 フレーム時間に 1 個の FSS が生成され，下流のすべてのノード PC に転送されることになる．さらに，各ノード PC が FSS に合わせて動作することにより，システム全体が同期して動作することになる．

2.5 同期機構と例外処理機構

FSM が FSS 受信タイミングを DPM に通知し，DPM はそれにあわせて各フレームに対する処理を開始する．これにより，DPM では 1 フレームにつき 1 回フレームデータ処理関数を実行することができる．

さらに，FSS 受信時に，次に処理するべきフレームデータが到着しているかどうか，および，前のフレームデータに対する処理が終了しているかどうかをチェックすることにより，遅れ状態になっているかどうかを検出することができる．

このような同期機構により遅れ状態を検出した場合，実時間性を保つために例外処理を実行して，遅れ状態から回復しなければならない．例外処理による DPM の振舞いはアプリケーションプログラマが記述できるのでさまざまなものが考えられるが，代表的なものとして以下の例外処理を RPV の標準として用意している．

不完全データ型 処理遅れを検出した場合は，そのフレームのデータ処理を中断し，アプリケーションプログラマが指定したデータ（たとえば，途中結果，前フレームの結果，デフォルトデータなど）を送信する．また，転送遅れを検出した場合は，そのフレームのデータの処理そのデータが到着するのを待つ．ただし，処理を終了するべき時刻になってもデータが到着しない場合は，処理遅れの場合と同様の例外処理を行う．これにより，すべてのフレームのデータに対する処理結果を決められた時間内に出力することができるので，実時間で画像処理を行っていることになる．

データ落ち型 処理遅れを検出した場合でも，そのフレームのデータ処理を継続し，それが終了するとその時点での最新のフレームデータに対する処理を開始し，それ以外のフレームデータは破棄する．また，転送遅れを検出した場合はそのフレームデータを破棄する．これにより，処理結果が出力されるのが遅くなることがあり，その影響で破棄されるフレームデータも存在するが，得られる結果はすべて完全に処理されたものとなる．結果の出力が遅れることがあるので，厳密には実時間処理とはいえないが，実際の動画像処理では利用しやすい方式である．

完全保持型 処理遅れを検出した場合でも，そのフレームのデータ処理を継続し，それが終了すると次のフレームのデータに対して処理を行う．また，転送遅れを検出した場合はそのデータが到着するのを待つ．これにより，すべてのフレームのデータを完全に処理することが可能となるが，遅れ状態から回復するための処理をまったく行わないので，実時間性は保証されない．また，受信バッファや送信バッファがあふれる可能性もある（このようなときには，上流ノードのデータ処理やデータ転送が止まるようになっており，最終的にはカメラからの画像獲得のところでバッファに入りきれないフレームデータが破棄されるようになっている）．これは動画像データを先に獲得しておきオフラインで実験を行うときに有効である．

3. ストリームデータ転送方式

3.1 フレーム同期データ転送方式の問題点

2.2 節で述べたように，RPV はパイプライン並列処理を基本としている．このためパイプラインの段数（パイプラインに並べられたモジュールの数）を増やすことによって，スループットを向上させることが可能となる．しかし，フレーム同期データ転送方式では，時系列画像をフレーム単位で処理していたために，パイプラインの段数に比例してレイテンシ（あるフレームのデータに対する第 1 段目のノード PC での処理が開始されてから，そのフレームのデータに対する最終段のノード PC での処理が終了するまでの時間）が大きくなってしまう．具体的には，すべての PC が 1 フレーム時間ごとに到着する FSS に同期して動作していたので，データ転送 1 段につき 1 フレーム時間，データ処理 1 段につき 1 フレーム時間のレイテンシが加えられる（**図 3** (a) 参照）．したがって，N 台の PC によるパイプライン並列処理を行うとそのレイテンシは，1 フレーム時間を T とすると

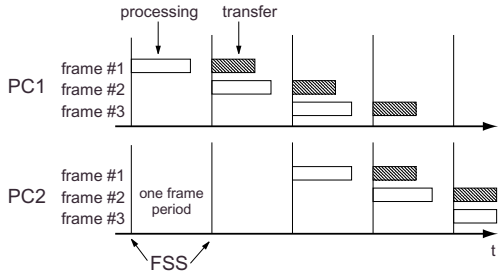
$$L_f = (2N-2)T + \tau_{2N-1} \quad (1)$$

になる．ここで，$n\ (=1,2,\cdots,N)$ 段目のノード PC におけるデータ処理時間を τ_{2n-1}，n 段目のノード PC から $n+1$ 段目のノード PC へのデータ転送時間を τ_{2n} とする．たとえば 3 段のパイプライン並列処理を NTSC カメラを利用して行った場合，T は 33 ミリ秒であり τ_n の最大値は T なので，そのレイテンシは最大約 0.167 秒になる．このようなレイテンシは，インタラクティブなアプリケーションにとっては大きな問題となる可能性がある．

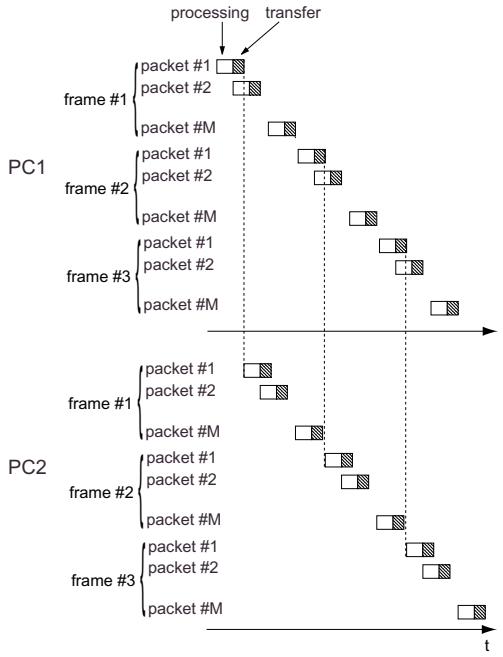
3.2 ストリームデータ転送方式の導入

この問題を解決するために，新たに細粒度データ転送を行うストリームデータ転送方式を導入した．ストリームデータ転送方式は，1 フレーム分のデータを小さなパケット（たとえば画素の集合，ボクセルの集合など）に分割し，パケット単位でデータ転送やデータ処理を行う方式である．これにより各 PC において，到着したパケットから順次処理を行い，処理が終了したパケットから順次送信を行うことで，パイプライン並列処理におけるレイテンシを大幅に削減することが可能となる（図 3 (b) 参照）．

フレーム同期データ転送方式に比べて，ストリーム



(a) Frame synchronization data transfer



(b) Stream data transfer

図 3 2 種類のデータ転送方式
Fig. 3 Two types of data transfer.

データ転送方式の特徴としては以下の点があげられる．

- **スループットを保ったままレイテンシを減少させることができる**

各 PC でのデータ処理量は変わらないので，M を 1 フレームあたりのパケット数とすると，1 段目のノード PC において第 1 パケットに対する処理が開始されてから，最終段のノード PC において処理が終了するまでの時間は

$$\frac{1}{M}\sum_{k=1}^{2N-1}\tau_k \tag{2}$$

となる．さらに，最終段のノード PC において，第 1 パケットに対する処理が終了してから最終パケットに対する処理が終了するまでの時間は，最も時間がかかるデータ転送またはデータ処理によって決定

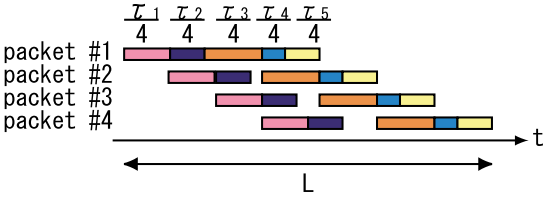


図 4 ストリームデータ転送方式におけるレイテンシ
Fig. 4 Latency on stream data transfer.

され，その時間は

$$\frac{M-1}{M}\max_k \tau_k \tag{3}$$

となる．したがって，全体のレイテンシは

$$L_s = \frac{1}{M}\sum_{k=1}^{2N-1}\tau_k + \frac{M-1}{M}\max_k \tau_k \tag{4}$$

となる．具体例として**図 4** を用いて説明する．これは 1 フレーム分のデータを 4 つのパケットに分割して 3 段のパイプライン並列処理を行ったときのデータ処理とデータ転送の時間を示したものである．横軸は時間の流れを，グラフの各段はそれぞれのパケットを表す．グラフの各段において，左から順に 1 段目のノード PC におけるデータ処理時間（$\tau_1/4$），1 段目から 2 段目のノード PC へのデータ転送時間（$\tau_2/4$），2 段目のノード PC におけるデータ処理時間（$\tau_3/4$），2 段目から 3 段目のノード PC へのデータ転送時間（$\tau_4/4$），3 段目のノード PC におけるデータ転送時間（$\tau_5/4$）を表している．τ_1 から τ_5 のうちで最も長いのは τ_3 である．このとき，1 段目のノード PC における第 1 パケットの処理開始から 3 段目のノード PC における第 1 パケットの処理終了までの時間は

$$\frac{\tau_1}{4}+\frac{\tau_2}{4}+\frac{\tau_3}{4}+\frac{\tau_4}{4}+\frac{\tau_5}{4} = \frac{1}{4}\sum_{k=1}^{5}\tau_k \tag{5}$$

である．また，3 段目のノード PC における，第 1 パケットの処理終了から第 2 パケットの処理終了までの時間は $\tau_3/4$ と等しくなり，これは以降のパケットでも同様である．したがって，3 段目のノード PC における，第 1 パケットの処理終了から第 4 パケットの処理終了までの時間は

$$\frac{3}{4}\tau_3 \tag{6}$$

となる．これらより，全体のレイテンシは

$$L_s = \frac{1}{4}\sum_{n=1}^{5}\tau_n + \frac{3}{4}\tau_3 = \frac{\tau_1}{4} + \frac{\tau_2}{4} + \tau_3 + \frac{\tau_4}{4} + \frac{\tau_5}{4} \quad (7)$$

となり，レイテンシが短縮されていることが分かる．ただし，実際にはパケットの数が多くなるとオーバヘッドが無視できなくなり，データ処理時間およびデータ転送時間が伸び，スループットは低下してしまう．オーバヘッドはパケット数に比例すると考えられ，この比例定数を α とすると，1 フレームあたりのデータ処理時間およびデータ転送時間を τ_k から $\tau_k + M\alpha$ に伸ばさざるをえなくなるので，全体のレイテンシは

$$L = \frac{1}{M}\sum_{k=1}^{2N-1}(\tau_k + M\alpha) + \frac{M-1}{M}\max_k(\tau_k + M\alpha) = \frac{1}{M}\sum_{k=1}^{2N-1}\tau_k + \frac{M-1}{M}\max_k \tau_k + (2N + M - 2)\alpha \quad (8)$$

となる．

- **各ノード PC において，1 パケット分のデータ処理によりそれに対応する 1 パケット分の結果が得られる必要がある**

ストリームデータ転送方式では，パケットに分割されたデータを処理し，即座にその出力結果を送信することによって，データ転送のレイテンシを削減している．これができない処理，たとえば画像全体を走査して初めて結果が得られるような処理では，それに続くデータ転送のレイテンシを削減することができない．

- **各ノード PC において，データ受信のパケットの順番とデータ処理のパケットの順番が同じである必要がある**

ストリームデータ転送方式では，パケットに分割されたデータが受信されると，即座にそのデータに対する処理を開始することによって，データ処理のレイテンシを削減している．これができない処理，たとえば画像中の画素にランダムにアクセスするような処理では，そのデータ処理のレイテンシを削減することができない．

このように，ストリームデータ転送方式によりレイテンシの削減は可能になるが，その必要条件からすべてのアプリケーションをストリームデータ転送方式で実現することはできない．一方，フレーム同期データ転送方式は，画像データはフレーム単位で獲得されるので，どのようなアプリケーションに対しても適用することが可能である．そこで，1 つのアプリケーションの中の適用可能な部分だけをストリームデータ転送方式で実現し，残りの部分をフレーム同期データ転送方式で実現することで，可能な限りレイテンシを削減することも可能である．

3.3 ストリームデータ転送方式のための同期機構

従来の RPV では，1 フレームにつき 1 回のフレームデータ処理関数の実行を保証するために，FSS のタイミングに合わせてフレームデータ処理関数を起動していた．また，従来の RPV では，FSS のタイミングに合わせてデータ転送とデータ処理の遅れを検出していた．しかし，この方法をストリームデータ転送方式でも利用するためには，そのノード PC でのデータ処理のレイテンシ，および後段のノード PC へのデータ転送のレイテンシを考慮して FSS を受信してから送信するまでの時間間隔を設定しなければならず，これを正確に行うことは困難であるという問題がある．この問題を回避するために，必要なデータがそろった段階で FSS に関係なくデータ駆動方式でデータ処理を開始し，FSS は従来どおりのタイミングでノード PC 間で転送され，遅れの検出のみに利用するように RPV の同期機構を改良した．具体的には，以下のような処理を行っている．

データ受信時の処理 DRM はデータパケットを受信するとそのことを DPM に通知する．したがって，DPM はデータの受信状況をつねに監視することができるので，どのような同期処理を行うかを柔軟に設定できる．たとえば，複数のノード PC からデータを受信するノード PC においてすべて PC からのデータがそろった時点で処理を開始するように設定したり（バリア同期），必要な数のデータがそろった時点で処理を開始するように設定したりすることが可能となる．

この機構により，ストリームデータ転送方式による処理を簡単に実現することができる．すなわち，パケット単位でデータが到着したときに，そのパケットに対する処理を開始すればよい．このような細かい設定をアプリケーションプログラマが記述できるようになっている．

なお，フレーム同期データ転送方式においても，FSS によってデータ処理を開始するのではなく，データ駆動方式でデータ処理を開始することにより，フレーム同期データ転送方式をストリームデータ同期転送方式における $M = 1$ の場合と考えることができる

ようになる．これにより，フレーム同期データ転送方式におけるレイテンシ L_f が式 (1) から

$$L'_f = \sum_{k=1}^{2N-1} \tau_k + (2N-2)\alpha \qquad (9)$$

となり，オーバヘッドが十分小さければ，レイテンシを削減することができる．

FSS 受信時の処理 上に述べたデータ受信時のデータ待ち合わせだけでは，データ処理やデータ転送に遅れが生じた場合，それらの終了を待ち続けることしかできず，定められた時間内に何らかの結果を出力しなければならない実時間処理を実現することはできない．そこで，FSS を受信したときにもデータ処理やデータ転送の状況をチェックする．受信した FSS の数とフレームデータ処理関数が処理しているデータのフレーム番号（各フレームデータに添付されている）を比較することにより，データ処理の遅れを検出することが可能である．また同様に，受信した FSS の数と到着しているデータのフレーム番号を比較することにより，データ転送の遅れを検出することも可能である．なお，パイプライン 1 段につき 1 フレーム時間の遅れしか許さないのか，2 フレーム時間の遅れまで許すのかを選択することができる．そして，これらの機能により遅れ状態が検出された場合は 2.5 節で述べた例外処理が実行される．

4. 性 能 評 価

4.1 評価対象アルゴリズムの構成

基礎的な性能評価実験については参考文献 5) で述べているので，本論文ではより大規模なアプリケーションによる評価実験を行った．具体的には，9 台のカメラを人間を取り囲むように配置し，23 台のノード PC を利用して視体積交差法（**図 5** 参照）による 3 次元形状復元を行った（**図 6** 参照）．各ノード PC でのデータ処理は以下のとおりである．なお，この実験では CVC 以降でレイテンシ削減のためストリームデータ転送方式を用いている．

EOS 各視点ごとに背景差分を利用して対象物のシルエットを抽出する．今回はカラー画像に対する背景差分をもとにした対象物シルエットの抽出処理を行っている．生成されるシルエット画像は 1 画素 1 ビットで表現されているので，データサイズが小さく，データ転送時間は非常に短い．さらに，そのパケット順と次段の CVC におけるデータ処理のパケット順が異なる．このため，ストリームデータ転送方式を用いても効果が薄いので，フレーム同期データ転送方式を用いている．したがって，EOS におけるデータ処理のレイテンシはそのデータ処理時間 τ_1 となる．また，EOS から CVC へのデータ転送のレイテンシはそのデータ転送時間 τ_2 となる．

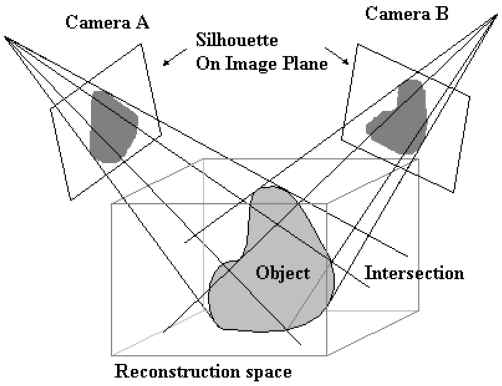


図 5 視体積交差法のアルゴリズム
Fig. 5 Algorithm of visual cone intersection.

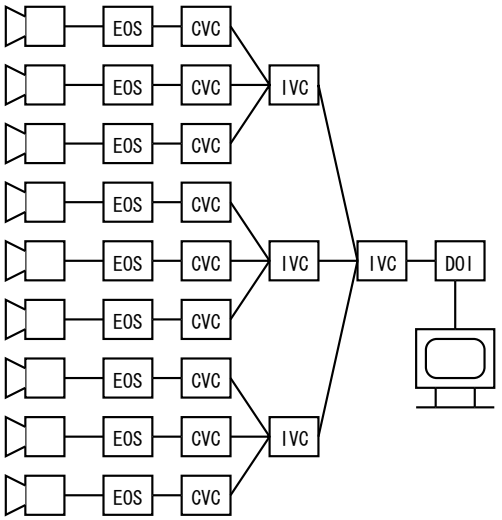


図 6 視体積交差法による実時間 3 次元形状復元システムの構成
Fig. 6 Configuration of real-time 3D shape reconstruction system based on visual cone intersection.

CVC 観測空間全体をボクセルに分割し，各視点ごとに対象物のシルエット画像からボクセル表現された視体積を構築する．具体的には，各ボクセルごとにそのボクセルに対応する画素がシルエットに含まれるかどうかをチェックする．この処理は，シルエット画像へのボクセルの投影と画素の包含判定からなる．ここでは処理時間を短縮するために，事前にボクセルと画素の対応関係を計算し，ルックアップテーブルに保持しておく．

CVC におけるシルエット画像からの視体積構築処

理では，ボクセル順に計算される．また，視体積の送信もボクセル順に行われる．したがって，ボクセルデータを M_v 個のパケットに分割したストリームデータ転送方式を適用することができる．なお，1 フレームあたりのこのデータ処理時間を τ_3 とする．また，1 フレームあたりのボクセルデータの転送時間を τ_4 とする．

IVC 複数視点の視体積の交差をとり，ボクセル表現された対象物の 3 次元形状を復元する．具体的には，同じ位置のボクセルどうしの AND 演算を行うことになる．データ転送能力の限界により，すべての視体積の交差を 1 台のノード PC で行うことはできないので，IVC を 2 段階に配置し，交差処理を行っている．

ボクセルデータの受信，ボクセルデータのの交差処理，ボクセルデータの送信はすべてボクセル順に行われ，不整合は起こらない．したがって，すべてストリームデータ転送方式を利用できる．なお，1 フレームあたりのこのデータ処理時間を τ_5 とする．また，1 フレームあたりのボクセルデータの転送時間を τ_6 とする．

DOI 獲得された 3 次元形状を表示する．現在は任意の視点からの対象物のシルエット画像を構築し，X ウインドウを利用してそれを表示している．

3 次元形状を表すボクセルデータの受信とシルエット画像構築処理はどちらもボクセル順に行われるため，ストリームデータ転送方式を利用することができる．なお，1 フレームあたりのこのデータ処理時間を τ_7 とする．さらに，シルエット画像を表示する処理はパケットごとではなく，シルエット画像構築終了後にまとめて行っている．しかし，X サーバの負荷による時間変動の影響を避けるため，今回の実験ではこの時間は全体のレイテンシに含めずに考える．

以上をまとめると，全体のレイテンシは

$$\begin{aligned} L(M_v) = {} & \tau_1 + \tau_2 \\ & + \frac{1}{M_v}(\tau_3 + \tau_4 + 2\tau_5 + 2\tau_6 + \tau_7) \\ & + \frac{M_v - 1}{M_v} \max_{3 \le k \le 7} \tau_k \\ & + (M_v + 8)\alpha \end{aligned} \tag{10}$$

となる．

4.2 実験結果

今回の実験では，入力画像は 320×240 画素，各画素 16 ビットの YUV 画像．シルエット画像は 320×240 画素，各画素 1 ビットの白黒画像．計測空間はボクセル数 $80 \times 80 \times 80 = 512,000$，各ボクセル 1 バイトとした．また，画像の入力速度は 10 fps とし，例外処理としては完全保持型を選択した．なお，データ駆動方式で各処理を開始することから，フレームレートにかかわらずレイテンシは一定である．このような条件のもとで，各処理モジュールのデータ処理時間，処理モジュール間のデータ転送時間，およびシステム全体のレイテンシを測定した．

まず，それぞれのデータ処理およびデータ転送にかかる時間を**表 1** に示す．さらに，M_v を変化させたときの全体のレイテンシとオーバヘッドに関する測定結果を**表 2** に示す．表 2 の中で，$L(M_v)$ が全体のレイテンシ，$L(1) - L(M_v)$ が $M_v = 1$ のとき（すべてフレーム同期データ転送方式を利用したとき）から削減されたレイテンシの大きさ，$\bar{L}(M_v)$ がオーバヘッドがないと仮定したときの全体のレイテンシ，ε（%）は Ł(M_v) に対する $\bar{L}(M_v)$ の誤差である．

これらの結果より，以下のことがいえる．

- ストリームデータ転送方式を利用することにより全体のレイテンシが削減されており，ストリームデータ転送方式の効果が確認できた．具体的には，$M_v = 1$ のときに比べ $M_v = 64$ のときは全体のレ

表 1 データ処理とデータ転送にかかる時間（msec）
Table 1 Data processing time and data transfer time (msec).

τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
25.6	0.3	41.4	15.4	16.6	11.3	9.6

表 2 全体のレイテンシとオーバヘッド
Table 2 Total latency and overhead.

M_v	1	2	4	8	16	32	64	128	256	512
$L(M_v)$(msec)	148.2	111.2	99.3	87.6	85.5	81.2	79.8	80.7	83.2	93.5
$L(1) - L(M_v)$	–	37.0	48.9	60.6	62.7	67.0	68.4	67.5	65.0	54.7
$\bar{L}(M_v)$ (msec)	148.1	107.7	87.5	77.4	72.4	69.8	68.6	67.9	67.6	67.5
ε (%)	0.1	3.1	11.9	11.6	15.4	14.0	14.1	15.8	18.7	27.8
α (msec)	0.01	0.35	0.98	0.64	0.55	0.28	0.16	0.09	0.06	0.05

イテンシが 46.2%削減されている．さらにストリームデータ転送方式を利用している CVC 以降だけで見るとレイテンシが 55.9%削減されている．

- M_v が大きくなるほどオーバヘッドは増加し，$M_v = 64$ のときで全体のレイテンシのうちの 14.1%をオーバヘッドによるレイテンシが占めている．さらに CVC 以降だけで見るとレイテンシのうちの 20.1%を占めている．
- オーバヘッドの比例定数 α は 0.01〜0.98 ミリ秒となり，ばらつきが出た．これは，OS として通常の Linux を利用したことによるタスク切替えの不確実性や時間計測の誤差のため，オーバヘッドがパケット数に比例するという仮定が成り立たなかったためと思われる．より正確な計測には RT-Linux などのリアルタイム OS を用いて行う必要があると考えており，今後の課題としたい．

5. おわりに

本論文では，分散並列計算機の一種である PC クラスタを用い実時間ビジョンシステムを構築するためのプログラミング環境 RPV について述べた．RPV が実時間並列ビジョンシステムの構築に必要な機能を提供することで，フレーム同期データ転送方式およびストリームデータ転送方式による実時間並列ビジョンシステムを容易に構築することが可能となる．

特に，ストリームデータ転送方式によるレイテンシの削減は実時間画像処理には有効な方式と考えられる．しかし，適用範囲が限られることもあり，万能ではない．オンラインのアプリケーションではシステムのレイテンシが大きな問題になりうるので，レイテンシの削減に関する研究を進める必要がある．この点については，予測に基づいたデータ処理など[8),9)]を積極的に導入する必要がある．

また，現在の同期機構ではパイプライン 1 段につき 1 フレーム時間ないし 2 フレーム時間以上の遅れを検出するようになっている．これにより，スループットを保証，つまり実時間処理を保証することはできる．しかし，パイプライン 1 段につき 1 フレーム時間ないし 2 フレーム時間までのレイテンシを許してしまうことになる．したがって，ストリームデータ転送方式を用いることによりレイテンシを削減することはできたが，短縮されたレイテンシを保証することができないという問題が残っている．これを解決するためには，FSM における FSS 送信のタイミングを保証したいレイテンシの大きさによって自由に変更できるようにすることが考えられる．

その他の今後の課題としては，

- プログラミングインタフェースの洗練，
- 多くのアプリケーション構築を通した詳細な性能評価，
- タスク分割や負荷分散の（半）自動化，

があげられる．

謝辞 本研究は，通信・放送機構による「次世代インテリジェント・マルチメディア情報通信網の基盤技術に関する研究（No.10080）」の補助を受けた．

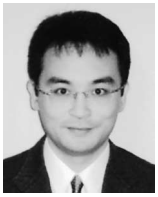
参考文献

1) Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V.: *PVM: Parallel Virtual Machine — A Users' Guide and Tutorial for Networked Parallel Computing*, The MIT Press (1994).
2) Message Passing Interface Forum: MPI: A message-passing interface standard, *International Journal of Supercomputer Applications and High Performance Computing*, Vol.8, No.3, pp.159–416 (1994).
3) Matsuo, H., Nakada, K. and Iwata, A.: A Distributed Image Processing Environment VIOS III and its Performance Evaluation, *Proc. 14th International Conference on Pattern Recognition*, Vol.II, pp.1538–1542 (1998).
4) Taniguchi, R., Makiyama, Y., Tsuruta, N., Yonemoto, S. and Arita, D.: Software platform for parallel image processing and computer vision, *Proc. SPIE '97 Parallel and Distributed Methods for Image Processing*, pp.1–10 (1997).
5) 有田大作，濱田義雄，米元 聡，谷口倫一郎：PC クラスタを利用した実時間並列画像処理環境 RPV，電子情報通信学会論文誌，Vol.J84-D-II, No.6, pp.965–975 (2001).
6) 吉本廣雅，有田大作，谷口倫一郎：1394 カメラを利用した多視点動画像獲得環境，第 6 回画像センシングシンポジウム講演論文集，pp.285–290 (2000).
7) Tezuka, H., Hori, A., Ishikawa, Y. and Sato, M.: PM: An Operating System Coordinated High Performance Communication Library, *High-Performance Computing and Networking*, Sloot, P. and Hertzberger, B. (Eds.), 1225 of Lecture Notes in Computer Science, pp.708–717, Springer-Verlag (1997).
8) Matsuyama, T., Hiura, S., Wada, T., Murase, K. and Yoshioka, A.: Dynamic Memory: Architecture for Real Time Integration of Visual Perception, Camera Action, and Network Communication, *Proc. Computer Vision and Pattern Recognition*, pp.728–735 (2000).

9) Yoshimoto, H., Arita, D. and Taniguchi, R.: Real-time Communication for Distributed Vision Processing based on Imprecise Computation Model, *CD-ROM Proc. Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia* (2002).

(平成 14 年 3 月 29 日受付)
(平成 14 年 9 月 12 日採録)

（担当編集委員　内海 章）



有田　大作（正会員）

平成 4 年京都大学工学部情報工学科卒業．平成 10 年九州大学大学院システム情報科学研究科博士後期課程修了．同年，同（現大学院システム情報科学研究院）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．電子情報通信学会，映像情報メディア学会各会員．

花田　武彦

平成 14 年九州大学工学部電気情報工学科卒業．現在，同大学院システム情報科学府修士課程在学中．実時間並列画像処理の研究に従事．

谷口倫一郎（正会員）

昭和 53 年九州大学工学部情報工学科卒業．昭和 55 年同大学院工学研究科修士課程修了．同年同大学院総合理工学研究科助手．平成元年同助教授．平成 8 年九州大学大学院システム情報科学研究科（現大学院システム情報科学研究院）教授．工学博士．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．本会論文賞（1993），同坂井記念特別賞（1995）を受賞．