(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3617851号
(P3617851)
(45)発行日　平成17年2月9日(2005.2.9)　(24)登録日　平成16年11月19日(2004.11.19)

(51)Int.Cl.[7]　F I
G06F 9/46　G06F 9/46　340B
G06T 1/20　G06T 1/20　C

請求項の数 1　(全 24 頁)

(21)出願番号　特願平6-125267
(22)出願日　平成6年6月7日(1994.6.7)
(65)公開番号　特開平7-105020
(43)公開日　平成7年4月21日(1995.4.21)
審査請求日　平成13年5月25日(2001.5.25)
(31)優先権主張番号　076678
(32)優先日　平成5年6月15日(1993.6.15)
(33)優先権主張国　米国(US)

(73)特許権者　596170170
ゼロックス　コーポレイション
ＸＥＲＯＸ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国　コネチカット州　スタンフォード、ロング・リッジ・ロード　８００
(74)代理人　100079049
弁理士　中島　淳
(74)代理人　100084995
弁理士　加藤　和詳
(72)発明者　デニス　エル．ヴェナブル
アメリカ合衆国　ニューヨーク州　１４６２３　ロチェスター　アカデミー　ドライヴ　５５

最終頁に続く

(54)【発明の名称】パイプライン化データ処理方法

(57)【特許請求の範囲】
【請求項１】
シングル・タスク環境においてデータ・アイテムをパイプライン処理するパイプライン化処理方法であって、該データ・アイテムはヘッダ及び少なくとも１つのデータ・セットを有し、
プロセッサ上のホストアプリケーションを実行するステップと、
リンクされたタスクのパイプラインを形成するために複数のデータ処理タスクを生成するステップであって、各データ処理タスクはライブラリ関数を起動することにより生成され、各データ処理タスクはデータ処理を実行するために前記プロセッサを制御し、各データ処理タスクは、
前記データ処理タスクと、パイプラインの少なくとも１つの他のデータ処理タスク及び前記ホストアプリケーションとの間でデータ・アイテムのヘッダの受け渡しを行うためのヘッダ・チャンネルと、
少なくとも１つのデータ処理チャンネルと、
前記データ処理タスクをパイプラインの少なくとも１つの他のデータ処理タスク及び前記ホストアプリケーションにリンクさせるための少なくとも１つのリンクと、
を有する、複数のデータ処理タスクを生成するステップと、
前記データ処理タスクに前記データ・アイテムの少なくとも１つのデータ・セットを前記データ・アイテムのヘッダに基づき処理させるために前記ホストアプリケーションを用いてパイプラインを呼び出すステップと、

を有するパイプライン化処理方法。
【発明の詳細な説明】
【０００１】
【産業上の利用分野】
本発明は、多重処理環境をエミュレートすることによって単一アプリケーション環境において画像ヘッダ及び画像データを有する画像処理に係り、特に、所定のオブジェクト又はデータストラクチュアのライブラリを用いたオブジェクト指向システムに関しており、当該オブジェクト指向システムは、ユニックス（登録商標）のようなパイプラインデータストラクチュアをエミュレートするストリーム指向のデータ処理ストラクチュアを生成するため、ホストアプリケーションとデータソース（源）の間でリンクされ得る。
【０００２】
【従来の技術】
真の多重処理環境でのデータ処理パイプラインの使用は周知である。公知の多重処理環境の例は、単一プロセッサが真の多重処理を支援することができる多重プロセッサシステム及び高水準（レベル）システムの両方を含む。ユニックス（登録商標）オペレーティングシステムはこのような多重処理システムにおいてしばしば使用される。
【０００３】
このような多重処理環境において、データ処理パイプラインは、画像処理、データベース処理、又はスプレッドシート（表計算）処理と対応するデータブロックのような大容量の高水準で構造化されたデータブロックを処理するのに極めて有用である。このようなデータブロックにおいて、種々のデータ処理動作は、データブロックの各データ要素上で実行されなければならない。さらに、種々のデータ処理動作は特定の順序で実行される。
【０００４】
多重処理環境において、データ処理パイプラインは、データブロックを処理するために非常に効率的な方法を提供する。これらのデータ処理パイプラインにおいて、各個別のデータ処理動作はパイプラインの一部分として定義される。各部分は、それが隣接する複数の部分（上流部分及び下流部分）の内の一つ又は両方へリンクされる。従って、データ処理動作は、データソースとホストアプリケーションの間でリンクされたパイプラインの部分同士の連鎖を形成する。多数の独立プロセッサを有するコンピュータにおいては、各パイプライン部分がこれらのプロセッサの内の一つと対応している。この場合、各プロセッサは独立して動作し、且つコンピュータオペレーティングシステムはプロセッサとメモリアロケーション（割り当て）間でデータの流れを制御する。これは、データを効率的に処理するが、プロセッサとメモリを制御するために必要なオーバーヘッドは、相当な割合でシステム資源を消費する。
【０００５】
同様に、多数の異なる独立した処理動作又は処理（プロセス）を同時に実行することができる単一プロセッサを有するコンピュータにおいて、各パイプライン部分は独立型実行処理の内の一つに対応している。この場合、オペレーティングシステムは、各処理のランタイム、各処理とメモリ間のデータのフロー、及びメモリアロケーションを割り当てる。この場合において、コンピュータを制御するために必要なオーバーヘッドは、各処理とそのデータが、プロセッサが実行される度にそのプロセッサへ、そしてそのプロセッサから、スワップ（交換）されなければならないので、より一層大きな割合でシステム資源を消費することになる。さらに、これらの処理はオペレーティングシステムを介して通信するので、それが可能であっても、ダイナミック（動的）にパイプラインを変えるのは困難である。
【０００６】
一般に、パイプラインに対するデータのソース（源）は、例えば、１）金融情報を提供するスプレッドシート、２）データベース情報を提供するデータベースファイル内の記録、及び３）オリジナル文書、コンピュータ作成画像、他から従来の画像スキャナによって生成された画像データであってもよい。これに対して、ホストアプリケーションは、処理さ

れた金融データからパイチャート、バーチャート、他を生成するためのグラフィックスプログラムか、処理されたデータベースデータを使用する棚卸し（在庫管理）、会計、又は併合プログラムか、又は処理された画像データから画像を形成するための画像形成装置であってもよい。

【０００７】

データの特別なソース又は最終的なホストアプリケーションにかかわらず、データ処理パイプラインの第１の又は最上流の部分は、一般に、パイプラインのためのデータソースである。或いは、この第１のパイプラインのためのデータソースは第２のパイプラインであってもよい。この場合、第２のパイプラインの特別なブランチング（分岐化）又は「ファンアウト（ｆａｎ－ｏｕｔ　）」パイプライン部分が、第１のパイプラインと、第２のパイプラインの下流部分の両方へデータを供給するために使用され得る。両ケースにおいて、第１のパイプライン部分は、データのソースからパイプラインに対するデータ要素を得て、且つこのデータ要素を直ぐ下流の又は第２のパイプライン部分に使用可能とする。第２のパイプライン部分は第１のパイプライン部分からデータ要素をシーケンシャル（順次）に受け取り、このデータ要素を処理し、且つそれを直ぐ次の下流又は第３のパイプライン部分へ渡す。同時に、真の多重処理環境において、第１のパイプライン部分はデータのソースから次のデータ要素を得て、且つそれを第２のパイプライン部分へ出力すると共に、データパイプラインの第２又は下流の隣接部分が第１のパイプライン部分から受け取られたデータ要素を処理し、且つそれをデータ処理パイプラインの第３の部分へ出力する。従って、各データ要素がパイプライン部分の内の一つによって処理されると、そのデータ要素はそれがホストアプリケーションへ出力されるまで次の下流部分へ出力される。

【０００８】

このように、データは、一つの処理動作又はプロセッサをパイプラインの各部分と対応させることによって、多重処理環境において効率的且つ迅速に処理され得る。これは、パイプラインが、最も効率の悪いパイプライン部分及びオーバーヘッドによって生じる非効率性によってのみ限定されるにすぎないデータスループット（処理能力）でデータブロックを処理することが可能であることを確実とする。

【０００９】

これに対して、単一処理環境においては、全ての方法が非常に非効率的であるにもかかわらず、データを処理するために様々な方法が利用可能である。例えば、各データ処理動作は、任意の他のデータ処理動作がデータブロックの任意のデータ要素へ適用されるよりも前に、データブロックの全てのデータ要素へ適用される。即ち、データブロックの全ての要素は、次のデータ処理動作が前もって処理されたデータ要素のいづれかに実行される前に処理されなければならない。従って、単一タスキングデータ処理動作の効率は、各データ処理動作の効率に比例する。

【００１０】

従って、データ要素は多重処理環境でのパイプラインの或るデータ処理部分から連続的に移動するので、データブロックのメモリアロケーション（割り当て）を管理する為に求められる必要オーバーヘッドは、各データ要素が何度もメモリから読み取られ、また何度もメモリへ書き込まれなければならない単一処理環境と比べると小さい。即ち、必要なメモリを管理するために要求されるオーバーヘッドは単一処理環境においては絶対に不可欠である。

【００１１】

【発明が解決しようとする課題】

従って、本発明は、単一処理プロセッサを用いて多重処理画像処理パイプラインをエミュレートすることによって画像を処理するための制御システムを提供する。

【００１２】

【課題を解決するための手段】

本発明の一つの方法においては画像処理動作のライブラリが提供されている。このライブラリにおける各画像処理動作、又は関数は、オブジェクト指向システム内のクラス又はサ

ブクラスである。パイプラインの各部分は具体化（インスタンシェート）された関数又はタスクであり、且つタスクの状態を自己定義するために一つ又はそれより多くの画像処理関数と充分なデータストラクチュアを有する。動作上、ホストアプリケーションがスキャナ又は画像データファイルのようなデータのソースからの処理された画像入力を必要とする時、画像処理パイプラインがそのソースからホストアプリケーションへ形成される。画像処理パイプラインは、画像処理ライブラリにおける関数のうちの一つを呼び出し且つタスクを形成するためにその呼び出された関数を初期化することによって形成される。このタスクはパイプラインの第１の又は最上流の部分となる。第１のパイプラインソースから処理されるべき画像のデータ要素を得る。データ要素は、好ましくは、画像のラスタ走査の単一走査ラインである。このソースは、好ましくは、スキャナ、ファクシミリ装置、遠隔コンピュータ、直列又は並列データ信号を出力するセンサ他、又はＲＯＭ、ＲＡＭの一部、若しくはディスクドライブ内のディスクに記憶されたデータのブロックである。このデータ要素はまた、第１のパイプライン部分それ自体によって直接発生されることができる。この最後のケースにおいて、データ要素は、初期化された変数の値や第１のパイプライン部分の状態などから得られることが可能である。最初に初期化される時、ホストアプリケーションの逆方向又は上流のリンクが第１のパイプライン部分へセットされる。

【００１３】

第２のパイプライン部分は、第１のパイプライン部分によって得られたデータ要素を処理するために必要とされる。従って、ホストアプリケーションはライブラリにおける関数の内の一つを初期化することによって他のタスクを生成する。この第２のパイプライン部分が作成されると、この部分は、自動的に第１のパイプライン部分へリンクされる。さらに、ホストアプリケーションのリンクは第２のパイプライン部分へセットされる。他の画像処理動作が全く要求されない場合、ホストアプリケーションの逆方向リンクは、処理されたデータを要求するホストアプリケーションの部分と第２のパイプライン部分の間に残る。

【００１４】

或いは、パイプラインが追加の部分を要求するならば、ホストアプリケーションが関数ライブラリに対して追加の呼出しを行ない、より多くのタスク又はパイプライン部分を作り出す。新しく作成されたタスクは、データ処理パイプラインの第３の、第４の、と順に続くパイプライン部分となる。各追加のパイプライン部分が作成されると、各追加のパイプライン部分と直ぐ上流のパイプライン部分間で逆方向リンクが形成される。再び、パイプラインの最後の部分が作成された後、ホストアプリケーションの逆方向リンクはパイプラインの最後の部分へセットされたままである。このように、各パイプライン部分へのアクセスは、オペレーティングシステムよりもむしろ直接下流のパイプライン部分（又は最後のパイプラインに対するホストアプリケーション）によって制御され且つ保持される。従って、メモリを保持するために必要なオーバーヘッドは最小であり、且つタスクの実行を計画するために必要なオーバーヘッドは存在していない。それゆえ、本発明の制御システムは、多重処理環境からの効率的な処理の利点を単一処理環境からの最小オーバーヘッドの利点と結び付けると共に、これらの環境の欠点を回避したのである。

【００１５】

さらに、これによって、単一処理アプリケーションは、単一タスキングプロセッサ、単一プロセッサ、自動時分割を用いて多重タスキングシステム、又は真の多重タスキングシステムにおいて実行され得るだけでなく、上記の利点を維持することができる。即ち、オペレーティングシステムよりもホストアプリケーションが、パイプラインを作成し、保持し、制御し、且つ終了するので、パイプラインストラクチュアは、処理のタイプ又はホストアプリケーションを実行することとは無関係である。従って、ホストアプリケーションは、単一プロセッサ多重タスキングシステムにおいて実行される多数の独立したアプリケーションの内の一つとなることができる。同時に、このホストは、パイプラインをスケジュールし、このパイプラインに対するメモリを割り当て、保持し、且つ割り当て解除したりする為に、オペレーティングシステムを使用せずに、パイプラインシステムを用いること

によって、スキャナからパイプラインを実行することができる。このようにして、オペレーティングシステムを用いてパイプラインを実行する際に通常は付きまとう欠点が回避され得る。

【００１６】

画像処理を始めるにあたって、ホストアプリケーションはパイプラインの最後の部分から画像のヘッダを要求する。パイプラインの各部分はまた、それがリンクされる上流のパイプライン部分からヘッダを要求する。これは、上流のパイプライン部分を呼び出すために下流のパイプライン部分から手順呼出しを行なうことによって実行される。これは、下流のパイプライン部分をスタックへプッシュ（付け足すこと）することに類似している。ヘッダが各上流のパイプライン部分から要求されると、プロセッサは下流のパイプライン部分の処理を停止し且つ上流パイプライン部分の処理を開始する。一旦第１のパイプライン部分に達すると、第１のパイプライン部分はヘッダを入手して（又は生成して）、且つそのヘッダをデータ要素として第２のパイプライン部分へリターンする。一旦第１のパイプライン部分がヘッダをリターンすると、ＣＰＵは第１のパイプライン部分の実行を停止し、且つ第２のパイプライン部分の実行を開始する。即ち、下流のパイプライン部分は上流のパイプライン部分を呼び出した手順呼出しからリターンされる。これは、下流のパイプライン部分をスタックからポップオフすることに類似している。一度、第２のパイプライン部分がヘッダを処理すると、ヘッダは第３のパイプライン部分へリターンされる。

【００１７】

次いで、第３のパイプラインは、そのデータ処理動作を、第２のパイプライン部分から受け取られた処理されたヘッダ上で実行し、この処理されたヘッダを、第４のパイプライン部分へリターンし、且つ動作を停止する。このヘッダは、各パイプライン部分において同じように処理される。最後のパイプライン部分がヘッダを処理した後で、この最後のパイプライン部分は、完全に処理されたヘッダをホストアプリケーションへリターンする。次いで、ホストアプリケーションは処理された走査ラインを、この最後のパイプライン部分から要求する。上記に概説された処理は、ホストアプリケーションからの走査ラインの要求ごとに繰り返される。ホストが全体の画像を入力しなかった場合、当該ホストは他の走査ラインを要求する。この要求は、再び、上流から第１のパイプライン部分まで移動し、且つ処理された走査ラインは下流からホストアプリケーションまで移動する。このように、パイプラインの下で処理された走査ラインは、効率の高い低オーバーヘッド方式で行なわれる。さらに、画像処理パイプラインの効率はパイプラインの最低の効率のリンクのみに依存する。

【００１８】

本発明の一つの態様は、シングル・タスク環境においてデータ・アイテムをパイプライン処理するパイプライン化処理方法であって、該データ・アイテムはヘッダ及び少なくとも１つのデータ・セットを有し、プロセッサ上のホストアプリケーションを実行するステップと、リンクされたタスクのパイプラインを形成するために複数のデータ処理タスクを生成するステップであって、各データ処理タスクはライブラリ関数を起動することにより生成され、各データ処理タスクはデータ処理を実行するために前記プロセッサを制御し、各データ処理タスクは、前記データ処理タスクと、パイプラインの少なくとも１つの他のデータ処理タスク及び前記ホストアプリケーションとの間でデータ・アイテムのヘッダの受け渡しを行うためのヘッダ・チャンネルと、少なくとも１つのデータ処理チャンネルと、前記データ処理タスクをパイプラインの少なくとも１つの他のデータ処理タスク及び前記ホストアプリケーションにリンクさせるための少なくとも１つのリンクとを有する、複数のデータ処理タスクを生成するステップと、前記データ処理タスクに前記データ・アイテムの少なくとも１つのデータ・セットを前記データ・アイテムのヘッダに基づき処理させるために前記ホストアプリケーションを用いてパイプラインを呼び出すステップとを有するパイプライン化処理方法である。

【００１９】

【実施例】

図１に示されているように、コンピュータ１００は単一処理コントローラ（ＣＰＵ－中央処理装置）２０を有する。コンピュータ１００は入力／出力（Ｉ／Ｏ）インターフェース２２、ＲＯＭ２４、及びＲＡＭ２６も有する。ＣＰＵ２０は制御信号を制御ライン２８を介してＩ／Ｏインターフェース２２、ＲＯＭ２４、及びＲＡＭ２６へ送り、且つデータバス３０を通してＩ／Ｏインターフェース、ＲＯＭ２４、ＲＡＭ２６へデータを送ったり、Ｉ／Ｏインターフェース、ＲＯＭ２４、ＲＡＭ２６からデータを受け取ったりする。タスクライブラリ１０、データソース１２、ディスプレイ１４、キーボード１６、マウス１８、及びディスクドライブ１９は、Ｉ／Ｏインターフェース２２を介してＣＰＵ２０と接続されている。マウス１８は、マウス、トラックボール、ライトペン、タッチスクリーン、タッチパッド、又はその他同種のもののようなあらゆる第２のポインティング装置を示す。ディスクドライブ１９は、ハードドライブ、ハードカード、フロッピーディスク及びフロッピーディスクドライブ、ＣＤ－ＲＯＭ及びＣＤ－ＲＡＭディスクドライブ、フラッシュメモリ、又はその他同種のもののようなあらゆる非揮発性の記憶装置を示す。

【００２０】

ＣＰＵ２０は単一処理プロセッサである。即ち、ＣＰＵは、多くても一回に一つのアプリケーションのためにデータをアクティブに処理することが可能であり、且つ前もって作動し且つ一回で単一のデータ処理動作のみを実行することが可能である。Ｉ／Ｏインターフェース２２は、ＣＰＵ２０を、直列データポート（図示されてない）及び並列データポート（図示されてない）のいずれかを介して直列又は並列のデータ入力ソース又はデータ出力ソースへ接続する。データのこれらのソースは、当然、キーボード１６及びデータソース１２だけでなく、ディスクドライブ１９及びマウス１８も含む。上記のように、データソース１２は、スキャナ、ファクシミリ装置又はその他同種のもの、遠隔コンピュータ、センサ又はその他同種のものを含む。Ｉ／Ｏインターフェース２２は、ディスプレイ１４を駆動し且つキーボード１６、マウス１８、及びディスクドライブ１９からの信号を入力するために必要なハードウェアも有する。

【００２１】

ＲＯＭ２４は、ＣＰＵ２０に対する全ての基本的なオペレーティングプログラムを記憶し、且つブートストラップ又はその他同種のものを含む。ＲＡＭ２６は、非常に多くのランダムアクセス可能なメモリ位置を備える。ＲＡＭ２６はブロック２６ａ、２６ｂ、２６ｃ、２６ｄ、２６ｅ、及び２６ｆのような割り当て可能なメモリのブロックへ分割されることができる。割り当て可能なメモリ２６ａ乃至２６ｆのブロックは、ホストアプリケーションと、タスクを形成するデータ処理データストラクチュア（即ちデータ処理パイプラインの部分）の一つ又はそれより多くの具体例を記憶する。ディスプレイ１４は目に見える情報を出力するが、キーボード１６は情報をコンピュータ１００へ入力するために使用される。

【００２２】

関数ライブラリ１０は、ＣＰＵ２０によって呼び出されるデータ処理関数を形成する初期化されないデータ処理ストラクチュア又はオブジェクトのライブラリを記憶する。関数ライブラリ１０は、Ｉ／Ｏインターフェース２２を介してアクセスされる独立した要素として示されているが、関数ライブラリ１０がＲＯＭ２４内又はディスクドライブ１９を介してアクセスされる不揮発性メモリ装置上で格納される得ることが理解されよう。さらに、ディスクドライブ１９内の非揮発性メモリ装置又は関数ライブラリ１０へまだ格納されてなかった新しく書き込まれたデータ処理オブジェクト又は関数はＲＡＭ２６内に格納され得る。

【００２３】

同様に、データソース１２はコンピュータ１００の外部にあるのが示されており、且つＩ／Ｏインターフェース２２を介してアクセスされる。しかしながら、データソース１２は、ＲＯＭ２４若しくはＲＡＭ２６内、ディスクドライブ１９内の非揮発性メモリ装置上、又はＣＰＵ２０のレジスタカウンタ若しくは内部メモリ内に格納されるデータであることができる。データソース１２は、キーボード１６を介してデータ入力を有することができ

る。上記のように、最後には、第１のパイプライン部分は、いくつかの外部ソースからデータ要素を得るのではなく、データ要素それ自体を発生することができる。

【００２４】

図１６に示されているように、本発明のコントローラ２０は、ホストアプリケーション手段５０、ライブラリ手段１０、メモリ管理手段５６、最後のタスク手段６０、少なくとも一つの中間タスク手段６２、及び第１のタスク手段６４を備える。コントローラはデータソース手段６６を含んでいてもよいが、含む必要もない。この場合において、第１のタスク手段６４もデータソース手段６６の関数を実行する。複数のリンク７０は、リンク７０によってホストアプリケーション手段に接続されたデータ処理パイプライン８０内の第１のタスク手段６４、少なくとも一つの中間手段６２、及び最後のタスク手段６０を接続させる。当然、簡単なパイプライン８０が、第１のタスク手段６４と最後のタスク手段６０だけを含んでもよいことは理解されるべきである。最後に、データ入力リンク７２はデータソース手段６６を第１のタスク手段６４と接続させる。

【００２５】

ホストアプリケーション手段５０は、データ処理パイプライン８０から受け取られたデータをさらに処理するためのデータ処理手段５８を含む。まず、コントローラ２０は、ホストアプリケーション手段５０、ライブラリ手段１０、及びメモリ管理手段５６を含むにすぎない。ホストアプリケーション手段５０は当該ホストアプリケーション５０内に含まれた命令によってコントローラ２０を操作する。コントローラ２０は、単一タスキングコントローラ、単一ＣＰＵ、多重タスキングコントローラ又は真の多重ＣＰＵであり得る。いづれの場合においても、ホストアプリケーション手段５０自体がコントローラ２０内で単一タスキング環境を生成する。

【００２６】

ホストアプリケーション手段５０は、それがデータソース６６から処理されたデータを必要とすると決定する時、当該ホストアプリケーション手段５０は、データ処理パイプライン８０を作成し且つ呼び出す。従って、ホストアプリケーション手段５０は初期化手段５２及び呼出し手段５４を含む。この初期化手段５２は、関数を取り且つタスクを生成する為にその関数を初期化するためにライブラリ手段１０へアクセスする。即ち、ライブラリ手段１０は多数のデータ処理関数を記憶する。データ処理関数は、ライブラリにおいて関数が初期化されなかった時、オブジェクト指向システム内のオブジェクトのクラス又はサブクラスである。

【００２７】

初期化手段５２は、初期化されない関数の内の一つを得るためにライブラリ手段１０へアクセスする。次いで、初期化手段５２は、得られた関数を初期化して、具体化された関数又はタスクをコントローラ２０内に生成する。具体化された関数又はタスクは、ホストアプリケーション手段５０又は他の前もって生成されたタスクへリンクされる。タスクがホストアプリケーション手段へリンクされている場合、それは最後のタスク６０である。タスクが他のタスクへリンクされ、他のタスクもそのタスクへリンクされている場合、それは中間タスク６２である。タスクが他のタスクへリンクされ且つデータソース手段６６からデータを受け取るか、又はそれ自体がデータを発生する場合、それは第１のタスク６４である。

【００２８】

タスクを生成するために関数を初期化するにあたって、初期化手段５２は、タスクを記憶するためにメモリ２６のメモリブロック２６ａ乃至２６ｆの内の一つを割り当てるため、コントローラのメモリ管理手段５６と共に動作する。所望されるタスクの全てが初期化手段５２によって初期化された後で、呼出し手段５４が最後のタスク手段６０からデータを要求することによってパイプライン８０を呼び出す。呼出し手段５４からの要求は、パイプライン８０の上流方向へ第１のタスク手段６４へ向けて流れる（リップルする）。その要求が第１のタスク手段６４に達した時、第１のタスク手段はデータソース６６からデータを得るか又はそれ自体がデータを発生する。次いで、第１のタスキング手段６４はデー

タを下流方向で中間タスク手段６２へリターンする。中間タスク手段６２はそのデータを処理し且つそれを最後のタスキング手段６０へリターンする。最後のタスク手段６０はそのデータを処理し且つそれをホストアプリケーション手段へリターンする。次いで、より多くのデータがホストアプリケーション手段によって必要とされる場合、呼出し手段５４は再びパイプライン８０を呼び出す。

【００２９】

一旦ホストアプリケーション５０が全ての所望されるデータを有すると、呼出し手段５４は、パイプライン８０からエラーコードを要求することによって、当該パイプライン８０から発生するいづれのエラーをも閉じ込め（シャットダウン）且つクリーンアップ（浄化）するために再びパイプライン８０を呼び出す。前と同じようにこの要求は上流へリップルする。各タスキング手段は、それがエラーコードを発生したかを決定し、それらのエラーコードをリターンする。これらのエラーコードが一旦リターンされると、初期化手段５２は、パイプライン８０のタスク手段６０乃至６４に割り当てられたメモリを解放するため、メモリ管理手段５６へ信号を送る。

【００３０】

図１７は、汎用化されたタスク手段６８を示し、これはタスク手段６０乃至６４のいづれかを表す。この汎用化されたタスク手段６８は、リンク手段８２、少なくとも一つのチャネル手段８４（８４ａ、８４ｂ、８４ｃ）、データ処理手段８６、データ分析手段８８、タスク状態手段９０、エラーコード発生手段９２、外部ポート手段９４、及び再構成手段９６を備える。実際のタスク手段６０乃至６４が、一つより多くの要素を欠いた、これらの要素の異なる組み合わせを有してもよいことは当然理解されよう。さらに、汎用化されたタスク手段６８は、パイプライン８０のためにデータを受け取ったり発生したりするためのデータ受取り及び／又は発生手段９８も含む。

【００３１】

リンク手段８２は、順方向リンクが使用される時、汎用化されたタスク手段６８を次の又は上流のタスク手段６８とリンクするか、又は逆方向リンクが使用される時、汎用化されたタスク手段６８を前の又は下流のタスク手段６８とリンクする。両リンクとも一度に使用されることができる。汎用化されたタスク手段６８は、チャネル手段８４を介して次のタスク手段６８からリターンされたデータを受け取る。そのデータは、次いで、それがチャネル８４を介して前のタスク手段６８へリターンされる前にデータ処理手段８６を用いて処理される。

【００３２】

タスク状態手段９０はタスク手段６８の状態を保持する。タスク状態手段９０はまたタスク手段６８の全ての他の手段を制御する。

【００３３】

データ分析手段８８は次のタスク手段６８から受け取られたデータがデータ処理手段８６に適しているかを決定する。データが適していない場合、データ分析手段が二つの異なる可能な応答を持っている。第１のタイプの応答においては、簡単化されたタスク手段６８は、データの代わりに、エラーコード発生手段９２によって発生されたエラーコードを前のタスク手段へリターンするだけである。ホストアプリケーション手段５０がデータの代わりにエラーコードを受け取った時、アプリケーション手段５０は、そのエラーの性質（及び原因）を決定するためにエラー処理システムを使用する。次いで、ホストアプリケーション手段５０は、グレースフルに出るか又はパイプライン８０を再構成且つ／又は再初期化するとによってエラーから回復するように試みることができる。

【００３４】

第２のタイプの応答において、複雑なタスク手段６８は、エラーから回復するためにパイプライン８０をダイナミックに再構成しようとする再構成手段９６を含む。この再構成手段９６は、追加のタスク手段６８をダイナミックに初期化し且つそれらをパイプライン８０へ挿入するとによってこれを行なう。再構成手段は外部ポート手段９４を介して他のタスキング手段６８へアクセスすることによってこれを行なう。この外部ポート手段はタス

キング手段６８への様々なアクセスモードを許容する。これらのモードはタスク状態手段９０を検査及び／又は変更し、エラーコード発生手段９２を検査し、即ちリンク手段８２を変更したりすることも含まれる。

【００３５】

たとえタスク手段６８が洗練された再構成手段９６有していたとしても、回復不可能又はハードエラーはまだ発生する可能性がある。この場合、タスク手段はエラーコードを発生するため、再びエラーコード発生手段９２を使用する。

【００３６】

最後に、第２の実施例において、リンク手段８２は、タスク手段６８自体でなくて、チャネル手段８４をリンクする。リンク手段８２はチャネルを次の又は前のタスク手段６８へ直接リンクする。従って、タスク手段６８が多数のチャネル手段８４ａ、８４ｂ、及び８４ｃを有する場合、各チャネル手段８４ａ乃至８４ｃは異なる次の又は前のタスク手段６８へリンクされ得る。一方、第１の実施例において、リンク手段８２はタスク手段６８をリンクしたので、タスク手段６８のすべてのチャネル手段８４ａ乃至８４ｃは同じ次のタスク手段６８とリンクされることになる。

【００３７】

図２に示されているように、各初期化関数又はタスクはデータソース１２とホストアプリケーション１２０の間で実行されるパイプライン４０の部分４０ａ、４０ｂ、及び４０ｃとしてビジュアル化され得る。ホストアプリケーションはＣＰＵ２０によって現在実行されている（且つＲＡＭ２６に格納される）アプリケーションである。パイプライン４０の各部分４０ａ、４０ｂ、及び４０ｃは、一つ又はそれより多くのデータ処理チャネル４２と、パイプライン部分の状態を定義するデータストラクチュア４４、及びゼロ、一つ又はそれより多くの外部ポート４６を備える。各パイプライン部分４０ｂ及び４０ｃ並びにホストアプリケーション１２０のそれぞれが、パイプライン部分４０ｂ及び４０ｃ又はホストアプリケーション１２０を直ぐ上流のパイプライン部分４０ａ、４０ｂ、及び４０ｃのそれぞれへリンクする逆方向リンク４８を有する。或いは、各パイプライン部分４０ａ、４０ｂ、及び４０ｃは、直ぐ下流のパイプライン部分４０ｂ及び４０ｃ又はホストアプリケーション１２０への順方向リンク４８’を有する。最後には、逆方向リンク４８と順方向リンク４８’の両方が提供されてもよい。

【００３８】

制御システムの好ましい実施例において、パイプラインは「知的な」パイプライン部分を有する。これらの知的なパイプライン部分は、それがホストコンピュータから操作される時、ホストアプリケーションによるいかなる割り込みも必要とせずに、パイプラインをダイナミックに再構成することが可能である。例えば、パイプライン４０ｂは、パイプライン部分４０ａによってリターンされたデータ要素を処理することが可能である。パイプライン４０ｂがデータ要素を処理することが不可能な場合、パイプライン部分４０ｂは、関数ライブラリ１０をダイナミックに呼び出し、且つ新たなパイプライン部分４０ｄを作成する。パイプライン部分４０ｄはパイプライン部分４０ａにリンクされる。次いで、パイプライン部分４０ｂはそれ自体のリンクを変えて、パイプライン部分４０ａよりもむしろパイプライン部分４０ｄを指し示す。

【００３９】

或いは、パイプライン部分４０ｂがそれがデータを処理することが不可能であることを決定した場合（例え、それがそのパイプラインを再構成しようとする場合でも）、当該パイプライン部分４０ｂはハードエラーを示すエラーコードをリターンする。ホストアプリケーションにおけるエラー処理システムは、ライブラリからエラーの性質を決定する。一つの実施例においては、ホストアプリケーションはそれがエラーから回復するためにパイプラインを再生することができるかを決定する。再生できる場合、ホストアプリケーションはパイプラインを再初期化する。しかしながら、ホストアプリケーションがエラーから回復することが不可能である場合又はこの能力を備えていない場合、ホストアプリケーションはグレースフルに出る。

【００４０】
この様な回復不可能エラーは、例えば、画像を処理するにあたって発生し得る。ヘッダが１０００の走査ラインが画像内にあるが、走査ライン画像データ要素が第９９８回目の呼び出しでも第１のパイプライン部分へリターンされなかったことを示した場合、システムはハードエラーへ出会う。単に処理できるデータがないために、パイプラインがそれ自体で再構成しようとする能力に関わらず、このエラーは回復することができない。この場合において、第１のパイプライン部分は”ＥＲＲ” エラーコードをリターンする。ホストアプリケーションは、次いで、エラーコードに対する理由を決定するためにエラー処理システムを使用する。このエラーの性質を決定した後、ホストアプリケーションは、システムのいかなる再構成又は再初期化も的を得てないので、グレースフルに出る。

【００４１】
パイプラインが再構成される場合、パイプライン部分４０ｂはパイプライン部分４０ｄからデータを要求する。パイプライン部分４０ｄは、データ要素を再リターンするパイプライン部分４０ａからデータを要求する。しかしながら、ここで、パイプライン部分４０ａは、データ要素を、パイプライン部分４０ｂではなくパイプライン部分４０ｄへリターンする。あるいは、関数ライブラリ１０を呼び出す時、パイプライン部分はデータ要素を提供することができ、これによってパイプライン部分４０ｄはこの部分が作成されるや否や、データ要素を有することができる。従ってパイプライン部分４０ｂがパイプライン４０ｄからデータを要求する時、パイプライン部分４０ｄはすぐに処理を行ない、且つそのデータ要素をリターンすることができる。

【００４２】
さらに、順方向リンク４８’が提供される場合、知的なパイプライン部分は、それらのデータストラクチュアにおいて、データ要素がこのパイプライン部分によって処理されるのを必要とするかを決定するための手段を含むことができる。必要としない場合、パイプライン部分は、それ自体をパイプラインラインから取り去るため、ホストアプリケーションに全く割り込まれることなく、パイプラインをダイナミックに再構成することができる。順方向リンクは必要であり、これによって、パイプラインがどの下流のパイプライン部分を変えるかを知り、これによってパイプライン部分が適切な上流のパイプライン部分とリンクされる。

【００４３】
外部ポート４６は、パイプライン部分の現在状態を検査するため、パイプライン部分のデータ構造４４へデータを書き込むため、そのパイプライン部分の現在状態を変えるか若しくはダイナミックに部分を追加するためにパイプライン部分（又は複数のパイプライン部分）の連結部分４８をダイナミックに変えるため、又はパイプライン４０から複数の部分をデリート（削除）するために使用される。一般に、外部ポート４６は、データ要素を処理するにすぎないデータチャネル４２から独立して、パイプラインを制御するために使用される。要するに、データチャネル４２は、上流のパイプライン部分から受け取られたデータを処理し、且つこの処理されたデータを下流のパイプライン部分へ出力するためだけに使用される。外部ポート４６は、データチャネル４２の動作に影響を与えずにパイプライン部分のデータストラクチュア４４へのアクセスを可能とする。

【００４４】
従って、他の実施例においては、例えば、パイプライン４０が作成された後で、パイプライン部分４０ａとパイプライン部分４０ｂの間にパイプライン部分４０ｄを追加することが必要となった場合、これは外部ポート４６を用いて達成される。この実施例において、パイプライン部分４０ｃは、そのデータストラクチュアの一部として、チャネル内でデータをテストするためのストラクチュアを有する。データの形式が、例えば、パイプライン部分４０ｃの適切な動作に必要とされる形式と互換性を持たないが、データが、例えば、パイプライン部分４０ｂの前に変更されなければならない場合、パイプライン部分４０ｃは、新しいパイプライン部分４０ｄをダイナミックに初期化し且つそれをパイプライン部分４０ａとパイプライン部分４０ｂの間に挿入する。このパイプライン部分４０ｃは、パイ

プライン部分４０ｂにおける外部ポート４６を介して、パイプライン部分４０ｂのデータストラクチュア４４にアクセスすることによってこれを行なう。パイプライン部分４０ｃがパイプライン部分４０ｂのリンク４８を変え、これによりパイプライン部分４０ａからのリンク４８が新しいパイプライン部分４０ｄへ変えられる。同様に、新しいパイプライン部分４０ｄのデータストラクチュア４４は、このストラクチュアが適切に初期化され且つパイプライン部分４０ａへリンクされることを確実とするためにアクセスされる。

【００４５】

動作上、図７に示されているアプリケーションのようなホストアプリケーションがデータをデータソース１２から要求する時、ホストアプリケーションは、最初、関数ライブラリ１０から関数を呼出し且つ初期化することによってパイプライン４０を形成する。図７に示されている実施例において、変数　”ｉ”　、”　ｎ”　、”ａｄｄ＿＿ｖａｌｕｅ”、及び”ｍｕｌ＿＿ｖａｌｕｅ”がホストアプリケーションの５行目で初期化される。６行目では、第１のタスク４０ａが「ヌル」タスクとして定義されている。第１のタスクは、データソース１２からデータ要素を得るために接続されることができると共に、「ヌル」コマンドを用いて第１のタスク４０ａがデータソース１２として作用するのを示すことが可能である。図７の７行目において、パイプライン４０におけるタスクの数は、図８に示されるホストアプリケーションを開始したコマンドラインによって決定される。一般に、ホストアプリケーションを実行すると、パイプライン４０のレングス（長さ）及び構成素（ｃｏｎｓｔｉｔｕｅｎｔｓ）はダイナミックに決定される。或いは、パイプライン４０のレングス及び構成素はホストアプリケーションにおいて明白に説明される。最後に、パイプライン４０の初期のレングス及び構成素がダイナミックに決定された時又は明白に説明された時のいづれであっても、パイプライン４０はデータソース１２から受け取られたデータによってそれ自体をダイナミックに再構成することができる。

【００４６】

図７に示されている実施例において、７行目に示されているように、パイプライン４０のレングスはプログラムを開始するコマンドラインにおいて明白に説明される。この場合において、パイプラインのレングスは図８が示すように５である。図７の８行目と９行目に示されているように、各関数が呼び出され且つ初期化され、且つデータソースからホストアプリケーションへ拡張される。この場合、図７の６行目に示されているように、第１のタスク４０ａもデータソースのように動作する。さらに、９行目に示されているように、パイプライン４０における各タスクは同じ関数の異なる具体化（インスタンシェーション）又はタスクである。図７の９行目と図５のの１７行目に示されているように、この例における関数”Ｍａｔｈ　Ｔａｓｋ”　が初期化されると、そのデータストラクチュアは、各々が現在値”　ｉ”　にセットされる二つのチャネルデータバッファと、パイプライン内の上流タスクを指し示すポインタを含む。さらに、パイプライン内に上流タスクが全くない場合、ポインタは「ヌル」にセットされる。

【００４７】

関数ライブラリ１０における関数がオブジェクト指向言語においてはオブジェクトとして定義されるので、図７に示されている実施例において関数が初期化される度に、同じ関数の異なる具体化が事実上行なわれる。これによって、単一パイプライン４０はあらゆる単一関数の複数のコピーを有していてもよい。従って、関数の各コピーはライブラリ内に格納されたベース関数の異なる具体化であり且つパイプライン４０の異なるタスク又は部分として動作する。これにより、関数ライブラリ１０における関数は、それらがあらゆる単一パイプラインにおいて一回以上使用されることができるので、再帰的である。

【００４８】

８行目及び９行目においてパイプライン内の各タスクが作成された後、初期化されたタスクのチャネル（この場合は二つ）が１０行目と１１行目で実行される。一旦、図６に示されたホストアプリケーションが、１０行目と１１行目で実行された複数のチャネルを介して、タスクから最後のデータを受け取ると、図８に示された出力は、図７の１２行目と１３行目に説明されているようにプリントされる。

【００４９】
図７のホストアプリケーションにおいては、上記に示されているように、８行目及び９行目において、パイプラインは、多数の関数”ＭａｔｈＴａｓｋ”の独立した具体化を有する。図５及び図６は、図７に示されているように、ホストアプリケーションによって形成されるパイプラインのために必要な各具体化を形成するため、図７の８行目及び９行目においてホストアプリケーションによって呼び出されるライブラリオブジェクト”ＭａｔｈＴａｓｋ”を示す。
【００５０】
図５に示されている”ＭａｔｈＴａｓｋ”関数の１乃至６行目において、主要データストラクチュアが定義される。１乃至６行目に示されているように、データストラクチュアは、二つのチャネル即ちチャネル１及びチャネル２並びに逆方向又は上流リンクを含む。次に、７行目乃至１２行目において、第１及び第２のチャネルのそれぞれに対するこの関数に特定のデータストラクチュア、”ａｖａｌ”及び”ｍｖａｌ”が定義される。
【００５１】
次に、１７行目で、チャネル１に対するデータ要求、”ａｖａｌｕｅ”及びチャネル２に対するデータ要求、”ｍｖａｌｕｅ”、並びに上流リンク「リンク」で、タスク”ＭａｔｈＴａｓｋ”に対して具体化手順が定義される。１８乃至２０行目において、チャネル１及びチャネル２のそれぞれに対する変数”ａｖａｌｕｅ”及び”ｍｖａｌｕｅ”並びにリンクが定義される。最後に、２１行目において、関数”ＭａｔｈＴａｓｋ”のタスクを作成し且つＲＡＭ２６のメモリブロックを割り当てるための手順が定義される。
【００５２】
次に、図６の２８乃至３７行目に示されているように、チャネル１に対応するデータ処理手順とデータストラクチュアが定義される。チャネル１のデータ処理手順は、タスクの直ぐ上流のタスクから番号を得る。さらに、図７の６行目に示されているように、パイプラインにおける第１のタスクは「ヌル」タスクである。第１のタスクに対する「ヌル」状態から生じる結果は、”ｖａｌ” の初期値をゼロに定義する３２行目で定義される。３３行目では、逆方向リンクが存在している場合、３４行目で値”ｖａｌ” はチャネル１の上流タスクの出力へセットされる。次いで、３５行目では、値”ｖａｌ” は、現在パイプライン部分のチャネル１の加数値”ａｖａｌ”をその値へ追加することによって更新される。各パイプライン部分に対するチャネル１の加数値”ａｖａｌ”は、２５行目で、図７の９行目でセットされたパイプライン部分の状態パラメータ”ａｖａｌｕｅ”へセットされた。最後に、チャネル１のデータ処理手順は、新しい値”ｖａｌ” を直ぐ下流のパイプライン部分へリターンすることによって終了する。
【００５３】
同様に、図６の３８乃至４７行目において、チャネル２のデータ処理手順が定義される。唯一の差は、図５の２２行目で初期化されたチャネル１の処理手順よりもむしろ図５の２３行目で初期化されたチャネル２の手順を使用する３９行目で発生する。さらに、４２行目では、値”ｖａｌ” は第１のパイプライン部分に対する「ヌル」データソースを補償するため、１にプリセットされる。４５行目では、新しい値”ｖａｌ” は、図７の９行目においてセットされた、図５の２６行目で状態パラメータ”ｍｖａｌｕｅ”にセットされたチャネル２の被乗数によって乗算される古い値”ｖａｌ” に等しい。
【００５４】
動作上、図７の８行目と９行目に示されているホストアプリケーションは指示された回数だけ関数”ＭａｔｈＴａｓｋ”を呼び出す。関数”ＭａｔｈＴａｓｋ”の第１の具体化はデータソースに接続されないヌルタスクとしてセットアップされる。再び、８行目と９行目をループすることによって、関数”ＭａｔｈＴａｓｋ”の第２の具体化がタスク”ＭａｔｈＴａｓｋ”の先の具体化との逆方向リンクによって形成される。その後、図７の８行目と９行目を介して各ループごとに、関数”ＭａｔｈＴａｓｋ”のさらなる具体化が形成される。図７に示されているホストアプリケーションの８行目と９行目を介して最後のループが一旦完了すると、ホストアプリケーションからのタスク”ＭａｔｈＴａｓｋ”の最

後の一つへの逆方向リンクはセットされたままとなる。図４は、このようにして初期化された３－セクションパイプラインを示す。

【００５５】

次いで、１０行目と１１行目において、ホストアプリケーションは、関数”ＭａｔｈＴａｓｋ”の最後の具体化のチャネル１とチャネル２からデータを要求する。最後のタスクは回転し且つその直ぐ上流のタスクからデータを要求する。これは、その”ＭａｔｈＴａｓｋ”の第１の具体化がチャネル１とチャネル２を通してのデータに対する要求を受け取るまで続く。図３はこのタイプのデータフローの例を示す。ホストアプリケーションは図７の１０行目及び１１行目でパイプラインを呼び出した後、最後のパイプライン部分はそれがリンクされるパイプライン部分の図６の３４行目と４４行目で各チャネルを呼び出す。次いで、最後のパイプライン部分は、各々の呼び出されたパイプライン部分が、その部分がリンクされる上流のパイプライン部分を呼び出す。第１のパイプライン部分に達すると、この第１のパイプライン部分はデータソース１２からデータ要素を受け取るか、又はこの場合、データ要素を自己発生する。第１のパイプライン部分は、次いで、上記のようにそれを呼び出したパイプライン部分へそのデータ要素をリターンする。

【００５６】

関数”ＭａｔｈＴａｓｋ”の第１の具体化は、次いで、チャネル１とチャネル２に対して定義されたデータ処理手順を実行する。第１の具体化は、チャネル１に対しては０＋１に等しい値１及びチャネル２に対しては１×１に等しい値１を関数”ＭａｔｈＴａｓｋ”の第２又は具体化へリターンする。関数”ＭａｔｈＴａｓｋ”の各具体化は、引き続いて、第１の数をチャネル１で保持された値へ加算し、次いでチャネル２で保持された数を第２の数によって乗算する。図５及び図６に示されている例において、第１と第２の数は、図７の９行目でパイプラインが作成された時に初期化された加数と被乗数である。しかしながら、数はダイナミックに発生されることもできた。従って、チャネル１及びチャネル２において処理されたデータは、ホストアプリケーションへ向けて下流へリップルする。これによって、関数”ＭａｔｈＴａｓｋ”の最下流の具体化は、要求されたデータをホストアプリケーションへ提供する。

【００５７】

ＣＰＵ２０が図７に示されているホストアプリケーションを実行する時、それはホストアプリケーションをＲＡＭ２６のメモリブロック２６ａに格納する。ホストアプリケーションは、事実上、割り当て可能なメモリブロック２６ａ内に格納されたデータストラクチュアである。データストラクチュア自体は指定されたメモリの位置にわたるテンプレートを有しており、このテンプレートはＣＰＵ２０の制御動作のフロー（流れ）を定義する。割り当てられたメモリブロック２６ａの第１の部分において、種々のメモリ位置が定義され且つ／又は変数として割り当てられる。データブロック２６ａはまたホストアプリケーションの状態を定義する種々のメモリ位置、及びホストアプリケーションによって要求されたデータ又はホストアプリケーションによって処理されたデータを記憶するためのバッファとして動作する種々のメモリ位置を含む。

【００５８】

同様に、ホストアプリケーションが、例えば、関数”ＭａｔｈＴａｓｋ”の具体化を初期化するために関数ライブラリ１０を呼び出す時、他のデータブロック２６ｂがそのタスク又はパイプライン部分へ割り当てられる。ホストアプリケーションのように、初期化されたパイプライン部分は、実際に、割り当てられたメモリブロック２６ｂに対するテンプレートを備える割り当てられたメモリブロック２６ｂ内のデータストラクチュアである。割り当てられたメモリブロックは、処理されるデータ、データのタイプを定義するための種々のメモリ位置、このタスクがそのデータを受け取る場所を示す手順ポインタ、そのタスクの状態を定義するデータを記憶する種々のメモリ位置、及び関数”ＭａｔｈＴａｓｋ”の直ぐ上流の具体化から受け取られた処理されたデータを記憶するための一つ又はそれより多くのデータバッファを記憶するためのバッファを有する。

【００５９】

割り当てられたデータブロック２６ｂはチャネル１とチャネル２に対するデータ処理動作を定義するデータストラクチュアも含む。それとは別に、割り当てられたデータブロック２６ｂ内のデータストラクチュアは、データ処理を実行するための、又は関数ライブラリ１０における他の関数へのポインタルックアップテーブルとしての、ルックアップ（検索）テーブルを含む。一般に、各タスクは、それがデータブロック２６ａ乃至２６ｆの内の一つへ初期化され且つ割り当てられると、データ入力、データ出力、タスク状態、及びそのタスクによって実行される種々のデータ処理手順に充分なバッファを自動的に割り当てる。

【００６０】

図９乃至図１１は、この方法を実施する制御ルーチンに対して汎用化されたフローチャートを示す。図９に示されているように、スタートしてから、ステップＳ１０において、ホストアプリケーションは単一処理ＣＰＵ２０によって実行される。ステップＳ２０においては、コントローラが、一つのデータ処理パイプラインがコンピュータ１００の現在動作に必要とされるか否かをチェックする。必要とされない場合、制御ルーチンはステップＳ１０へ戻る。

【００６１】

しかしながら、パイプラインが必要とされる場合、制御ルーチンはステップＳ３０へ進み、このステップにおいて、関数ライブラリから関数を呼び出す。次いで、ステップＳ４０において、呼び出された関数は第１のタスク又は第１のパイプラインを形成するために初期化される。次いで、ステップＳ５０においては、コントローラは追加のパイプライン部分のいづれかが必要とされるか否かをチェックする。必要とされる場合、制御ルーチンは、ライブラリから次の関数を呼び出すためにステップＳ３０へ戻る。従って、制御ルーチンは、追加のタスクが要求されなくなるまで、ステップＳ３０乃至Ｓ５０をループする。この時点で制御ルーチンはステップＳ６０へ進む。

【００６２】

ステップＳ６０において、パイプラインが呼び出される。ステップＳ７０において、パイプラインは処理されたデータ要素をホストアプリケーションへリターンする。次いで、ステップＳ８０において、コントローラは追加のデータが必要か否かを決定する。追加のデータが必要な場合、制御ルーチンはパイプラインを再び呼び出すためにステップＳ６０へリターンする。

【００６３】

しかしながら、追加データが必要とされない場合、制御ルーチンはステップＳ９０へ進み、種々のパイプライン部分へ割り当てられたメモリを割り当て解除することによって初期化解除される。ステップＳ１００において、ホストアプリケーションの次のコマンドが実行される。次いで、ステップＳ１１０において、コントローラはホストアプリケーションの追加のパイプラインが実行される必要があるかを決定する。実行される必要がある場合、制御ルーチンはステップＳ３０へリターンする。

【００６４】

しかしながら、この時点でパイプラインが必要とされない場合、制御ルーチンはステップＳ１２０へ進み、コントローラがホストアプリケーションが完了したか否かを決定する。完了した場合、制御ルーチンはストップする。そうでない場合（完了していない場合）、制御ルーチンはステップＳ１００へ戻る。

【００６５】

図１０は初期化ステップＳ４０に対するフローチャートを示す。ステップＳ４０で初期化ルーチンへエンターした後、制御ルーチンは、タスクに対する主要データストラクチュアが作成されるステップＳ２００へ進む。次いで、ステップＳ２１０において、このタスクに対して特定であるデータストラクチュアの部分が作成される。次いで、ステップＳ２２０においては、チャネルデータストラクチュアが作成され、且つステップＳ２３０においては、逆方向リンクが作成される。順方向リンクが必要とされる場合、そのデータストラクチュアは定義されるが、順方向リンクがリンクする下流タスクがまだ作成されていない

ので、それはパイプラインが呼び出されるまで終了されない。リンクはリンクされたパイプライン部分を呼び出すための手順ポインタを有する。次いで、ステップＳ２４０において、制御ルーチンはステップＳ４０へ戻る。

【００６６】

図１１はステップＳ６０の呼出しルーチンに対するフローチャートを示す。ステップＳ６０からスタートして、制御ルーチンはホストアプリケーションが最後のパイプライン部分からデータを要求するステップＳ３００へ進む。次いで、ステップＳ３１０において、現在のパイプライン部分は次の上流パイプライン部分からデータを要求する。ステップＳ３２０においては、コントローラは次のパイプライン部分が最初パイプライン部分であるか否かを決定する。そうでない場合、制御ルーチンはステップＳ３１０へ戻る。

【００６７】

しかしながら、次のパイプライン部分が最初パイプライン部分である場合、制御ルーチンはステップＳ３３０へ進む。ステップＳ３３０において、コントローラは第１のパイプライン部分が「ヌル」部分であるか否かを決定する。そうでない場合、制御ルーチンは最初パイプライン部分が外部ソースからデータ要素を得るステップＳ３４０へ進む。第１のパイプライン部分が「ヌル」部分である場合、制御ルーチンは、第１のパイプライン部分がデータ要素を自己発生するステップＳ３５０へ進む。

【００６８】

両ケースにおいて、ステップＳ３４０又はステップＳ３５０のいづれかを経て、制御ルーチンはステップＳ３６０へ進み、ここで第１のパイプライン部分はデータを第２即ち次のパイプライン部分へ下流へリターンする。ステップＳ３７０において、次のパイプライン部分はデータを処理する。次いでステップＳ３８０において、次のパイプライン部分が最後のパイプライン部分である場合、処理されたデータはステップＳ３９０でホストアプリケーションへリターンされる。しかしながら、次のパイプライン部分が最後のパイプライン部分でない場合、データは新しい次のパイプライン部分へリターンされ、且つステップＳ３６０へのループバックにより示されるように、データ要素が処理される。この「ルーピング」は、最後のパイプライン部分へ至り、且つステップＳ３９０でデータがホストアプリケーションへリターンされるまで行なわれる。次いで制御ルーチンはステップＳ６０へ戻る。

【００６９】

本発明のデータ処理パイプラインの第１の実施例において、前もって定義された逆方向リンクを置換するか又はこの逆方向リンクに付け加えられる順方向リンクが提供される。この場合、タスクライブラリのタスクの内の一つの現在最下流具体化から逆方向リンクが定義された後で、直ぐ上流のタスクから最下流タスクへの順方向リンクが定義される。或いは、順方向リンクのみが提供されなければならない。しかしながら、この場合において、パイプライン部分は、ホストアプリケーションから上流へ、存在していれば、データのデータソースへ向かって初期化され、次いで下流へのデータソースからホストアプリケーションへ呼び出される。

【００７０】

さらなる実施例において、各タスクが二つ又はそれより多くのデータ処理チャネルを有する時、単一の順方向リンク又は単一の逆方向リンクをタスク間へ提供するよりも、これらのチャネル自体が複数の順方向及び／又は逆方向リンクによって共にリンクされる。この場合において、各チャネルは、タスクライブラリからのタスクの具体化の内のいづれか一つのどれか他の対応チャネルと逆方向又は順方向リンクされることができる。従って、第１のパイプライン部分のチャネルの内の一つは第２のパイプライン部分の対応チャネルとリンクされるが、第１のパイプライン部分の他のチャネルは第３のパイプライン部分の対応チャネルとリンクされる。

【００７１】

本発明のまた他の実施例において、特別な「ファンイン（ｆａｎ－ｉｎ）」及び「ファンアウト（ｆａｎ－ｏｕｔ　）」のパイプライン部分が提供される。これらのパイプライン

部分において、二つ又はそれより多くの上流のパイプライン部分がファンイン分岐パイプライン部分を用いて単一の下流のパイプライン部分とリンクされる。同様に、単一上流パイプライン部分がファンアウト分岐パイプライン部分を用いて二つ又はそれより多くの下流パイプライン部分と接続される。

【００７２】

図１２乃至図１５に示されているように、このシステムのアプリケーションにおいて、システムは、走査ライン×走査ラインを基本にして画像を非常に効率的に処理するストリーム指向のユニックス（登録商標）パイプラインをエミュレートするために作成される。ユニックス（登録商標）パイプラインは、保持されなければならないモジュラー性と容易なメンテナンスを含むなど多数の利点を有する。しかしながら、ユニックス（登録商標）パイプラインは、多重処理プロセッサを必要としたり、携帯性の欠如、ダイナミックな再構成のそれ自体の能力の欠如、及びタスクスケジューリング及びデータフロー処理におけるシステムオーバーヘッドの必要性などを含む多くの欠点も有している。

【００７３】

本発明の画像処理パイプラインにおいては、画像は、二つの主要部分、ヘッダと画像本体に分割される。ヘッダは、そこに表示されるカラースペース、配向、走査ラインのレングスのようなその寸法、走査ラインの数、インタレースファクタ、ピッチその他のような画像の状態を定義する。画像本体は、ラスタ出力スキャナ形式等で、各走査ラインがパイプラインに対する一つのデータ要素である走査ラインデータを備える。本発明を用いた画像処理パイプラインにおいて、同じ画像ストラクチュアはデータブロックとして使用される。

【００７４】

従って、パイプライン部分はそれぞれ少なくとも三つのチャネルを有する。第１のチャネルはヘッダ情報を処理するために使用される。第２のチャネルは、走査ライン×走査ラインベースでデータブロックとして画像本体を処理するために使用される。一つ又はそれより多くの第２のチャネルタイプのチャネルがパイプラインによって提供される画像処理のタイプに従って提供される。第３のチャネルは、要求されたクリーンアップ動作を実行し、且つ一度画像が完全に処理されると、パイプライン部分を備えるデータストラクチュアへ割り当てられるメモリを解除するために使用される。本質的に、画像処理はこの第３のチャネルが呼び出される時まで完了するので、チャネルを介したデータフローイングのみがエラー処理情報となる。実際、第３のチャネルは、各上流のパイプライン部分に、画像処理から発生したあらゆるエラーコードをリターンさせ、且つこの情報が一度リターンされると、受け取る下流パイプライン部分に、上流のパイプライン部分をダイナミックにデリートさせる。さらに、あらゆる必要な外部ポート手順ポインタはこの画像処理パイプラインのパイプライン部分のデータストラクチュア内に含まれる。

【００７５】

即ち、図１２乃至図１５に示された画像処理システムにおいて、図１７の汎用化されたタスク手段６８は、ヘッダチャネルとして第１のチャネル手段８４ａと、画像処理チャネルとして第２のチャネル手段８４ｂと、エンドチャネルとして第３のチャネル手段８４ｃを備える。

【００７６】

上記のように、ホストアプリケーションが処理された画像データが必要とされると決定した時、画像処理パイプラインが作成される。パイプラインは関数ライブラリ１０から種々のライブラリ関数を初期化することによって作成される。これらの関数は、画像のカラーを反転する「反転（ｉｎｖｅｒｔ）」、白黒画像をカラー化する「カラー化（ｃｏｌｏｒｉｚｅ）」、例えば、画像内のエッジをソフトにするために重畳フィルタをかけるフィルタ、画像本体を拡大する「拡大（ｅｎｌａｒｇｅ　）」、及び画像本体を縮小する「縮小（ｒｅｄｕｃｅ）」を含む。これは例示を目的としており、全てを網羅してはいない。例えば、「反転（ｉｎｖｅｒｔ）」は黒を白へ、又はその逆へ変換することができる。さらに、反転は、カラー画素を、例えば、赤からシアンへ、緑をマゼンタへ、及び青を黄へ変

換することもできる。

【００７７】

初期のパイプラインは例えば、３色のフォトタイオードスキャナ、及びホストアプリケーションに接続された種々の他のパイプライン部分からの画像を生成する第１のパイプライン部分を提供することによって形成される。ホストアプリケーションは、例えば、４色のタンデムプリンタで印刷するために走査された画像を準備する。次いで、パイプラインは、スキャナからヘッダを入力するため、ホストアプリケーションによって呼び出される。ヘッダに対する要求は、第１の（最新の）パイプライン部分までパイプラインのチャネル１まで移動する。第１のパイプライン部分は、スキャナからヘッダを得て、且つそれをチャネル１の下流へリターンする。

【００７８】

各パイプライン部分がヘッダを受け取ると、各パイプライン部分は、それが画像本体を正確に処理することができるかを確実とするためにこのヘッダをチェックし、このパイプライン部分が画像本体に対して行なうことを反映するためにヘッダを変更し、次いでそれを下流へリターンする。しかしながら、パイプライン部分が画像が適切に処理されることが不可能であると決定した場合、このパイプライン部分は、その欠点を補正するため、そのパイプラインをダイナミックに再構成する。

【００７９】

例えば、上記のように、スキャナから受け取られた画像は３色画像である。しかしながら、例えば、パイプラインが、３色スキャナと３色プリンタが、画像を作成するにあたって同じカラースペースを使用すると仮定して、元々、セットアップされた場合、このスキャナとプリンタは実際には異なるカラースペースを使用するので、エラーが生じる。この場合、可能性のあるエラーはヘッダを処理する時に発見される。エラーを検出したこのパイプライン部分４０ｘは、そのパイプラインの直ぐ上流に新しいパイプライン部分４０ｙを初期化し且つ追加することにより自動的にパイプラインを再構成する。同時に、このパイプライン部分４０ｘは、新しいパイプライン部分４０ｙの位置を反映するためにリンクを変更する。次いで、ヘッダは再実行される。ここで、パイプライン部分４０ｙがそれが画像のカラースペースを変えることを反映するためにヘッダを変更した後、パイプライン部分４０ｘは、それが画像本体を正確に処理することができると決定する。

【００８０】

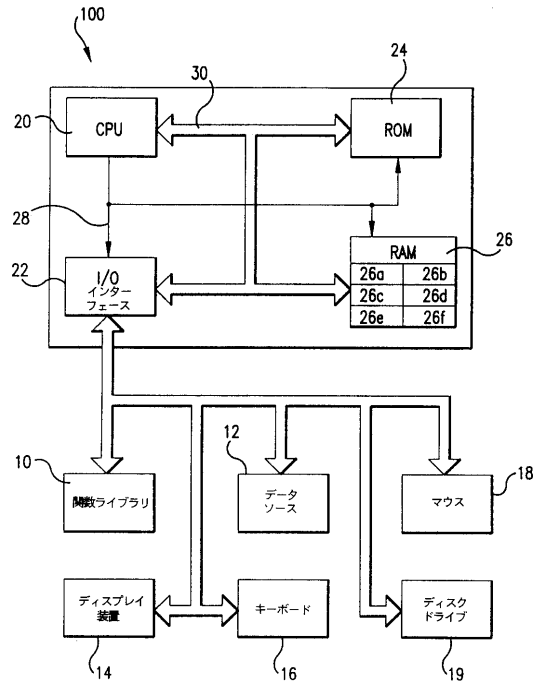
一度、ヘッダが完全に処理され且つホストアプリケーションへリターンされると、ホストアプリケーションは画像本体を入力するためにパイプラインを繰り返し呼び出す。このヘッダから、ホストアプリケーションはいくつの走査ラインが画像本体内にあるかも知っており、これによって、画像本体全体を入力し且つ処理するために呼出しサイクルがどれだけ必要かをも知ることになる。パイプラインが繰り返し呼び出されると、画像本体の一つの走査ラインが得られ、処理され且つ各呼出しごとにデータ要素としてホストアプリケーションへリターンされる。他のすべての点において、パイプラインは概して上記のように動作する。一旦、全体の画像本体が得られ、処理され、且つホストアプリケーションへリターンされると、ホストアプリケーションはパイプラインを停止（シャットダウン）するために第３のチャネルを呼び出す。図２は、このパイプラインの実行を示すフローチャートを示す。

【００８１】

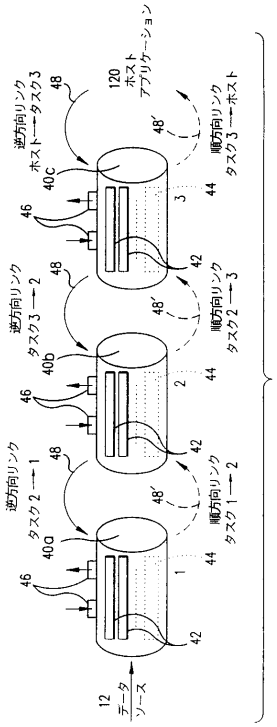
図１４に示されているように、この画像処理パイプラインの他の実施例において、第２と第３のチャネルだけが初期化され且つ呼び出される。この場合、第１の又はヘッダチャネルよりもむしろ、ヘッダを分析するためには外部ポート又は外部手順が使用される。この外部手順は、初期化の間、ヘッダを受け取り、処理し、且つ出力するために使用される。即ち、第１のパイプライン分が初期化される時、ヘッダを得る。次いで、各下流のパイプライン部分が初期化されると、ヘッダは外部ポートを用いて外部手順によってそのパイプラインへ出力される。図１５はこの実施例のパイプラインの実行を示すフローチャートを示す。

【００８２】
【発明の効果】
単一処理プロセッサを用いて多重処理画像処理パイプラインをエミュレートすることにより画像を処理するための制御システムを提供する。
【図面の簡単な説明】
【図１】本発明のシステムのブロック図を示す図である。
【図２】本発明のパイプラインのブロック図を示す図である。
【図３】本発明のパイプライン内のデータの流れを示す図である。
【図４】適例となるデータ処理プログラムのためにリンクされたパイプラインを示す図である。
【図５】サンプルタスクのための適例となるコードリスティングを示す図である。
【図６】サンプルタスクのための適例となるコードリスティングを示す図である。
【図７】パイプラインを形成するためにライブラリからタスクを呼び出す適例となるホストアプリケーションにためのコードリスティングを示す図である。
【図８】図４に示され且つ図５及び６並びに図８にリストされているパイプラインの出力のリスティングを示す図である。
【図９】主要ルーチンのフローチャートを示す図である。
【図１０】機能初期化ルーチンを示す図である。
【図１１】呼出し及び処理ルーチンのフローチャートを示す図である。
【図１２】画像処理パイプラインを示す図である。
【図１３】図１２のパイプラインを形成し且つ実行するためのフローチャートを示す図である。
【図１４】図１２の画像処理パイプラインの第２の実施例を示す図である。
【図１５】図１４の第２の実施例の画像処理パイプラインを形成し且つ実行するためのフローチャートを示す図である。
【図１６】本発明のコントローラを示すブロック図である。
【図１７】本発明の汎用化されたタスクを示すブロック図である。
【符号の説明】
１０　タスクライブラリ
１２　データソース
１３　ディスプレイ
１６　キーボード
１８　マウス
１９　ディスクドライブ
２０　単一処理コントローラ
２２　Ｉ／Ｏインターフェース
２４　ＲＯＭ
２６　ＲＡＭ
１００　コンピュータ

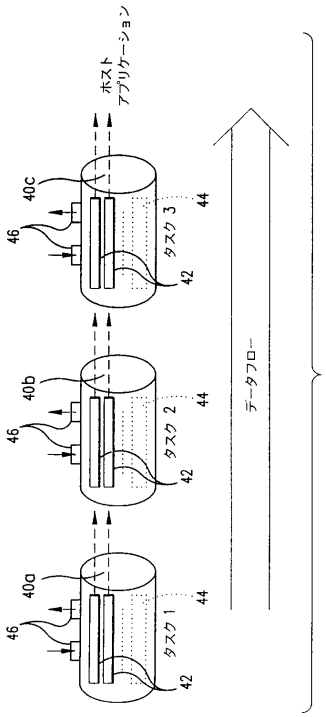
【図１】


100
CPU
ROM
RAM
I/O インターフェース
関数ライブラリ
データソース
マウス
ディスプレイ装置
キーボード
ディスクドライブ

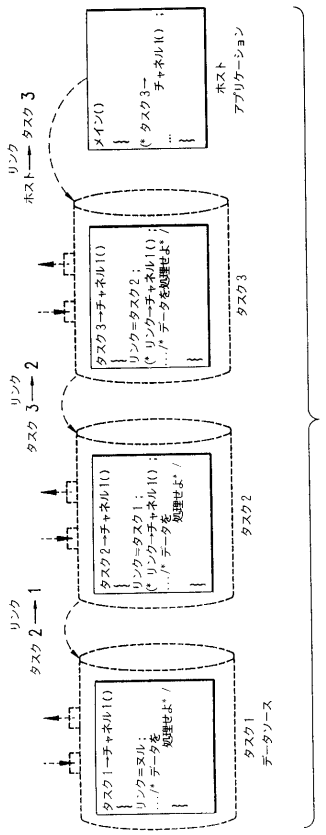
【図２】


ホストアプリケーション
順方向リンク
逆方向リンク
データソース

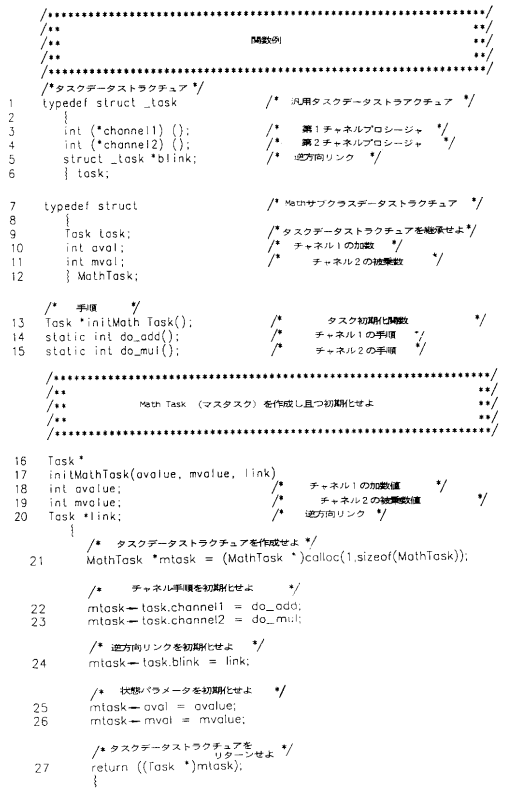
【図３】


ホストアプリケーション
タスク 3
タスク 2
タスク 1
データフロー

【図４】


ホストアプリケーション
リンク ホスト→タスク 3
リンク タスク 3→2
リンク タスク 2→1
タスク1 データソース
タスク2
タスク3

【図５】



```
/*****************************************************************/
/**                                                             **/
/**                        関数例                               **/
/**                                                             **/
/*****************************************************************/
/*タスクデータストラクチュア */
typedef struct _task                    /*  汎用タスクデータストラクチュア  */
  {
  int (*channel1) ();                   /*    第１チャネルプロシージャ   */
  int (*channel2) ();                   /*    第２チャネルプロシージャ   */
  struct _task *blink;                  /*   逆方向リンク   */
  } task;

typedef struct                          /* Mathサブクラスデータストラクチュア  */
  {
  Task task;                            /*タスクデータストラクチュアを継承せよ*/
  int aval;                             /*  チャネル１の加数    */
  int mval;                             /*      チャネル２の被乗数      */
  } MathTask;

/*   手順      */
Task *initMath Task();                  /*      タスク初期化関数              */
static int do_add();                    /*      チャネル１の手順     */
static int do_mul();                    /*      チャネル２の手順     */

/*****************************************************************/
/**                                                             **/
/**          Math Task （マスタスク）を作成し且つ初期化せよ     **/
/**                                                             **/
/*****************************************************************/

Task *
initMathTask(avalue, mvalue, link)
int avalue;                             /*    チャネル１の加数値        */
int mvalue;                             /*      チャネル２の被乗数値            */
Task *link;                             /*   逆方向リンク   */
  {
    /*   タスクデータストラクチュアを作成せよ */
    MathTask *mtask = (MathTask *)calloc(1,sizeof(MathTask));

    /*      チャネル手順を初期化せよ       */
    mtask->task.channel1 = do_add;
    mtask->task.channel2 = do_mul;

    /* 逆方向リンクを初期化せよ    */
    mtask->task.blink = link;

    /*   状態パラメータを初期化せよ    */
    mtask->aval = avalue;
    mtask->mval = mvalue;

    /* タスクデータストラクチュアを
                        リターンせよ */
    return ((Task *)mtask);
    }
```

【図６】

```
/*****************************************************************/
/**                                                             **/
/**          チャネル１処理手順  －  値を入力へ追加せよ         **/
/**                                                             **/
/*****************************************************************/

static int
do_add(mtask)
MathTask *mtask;
  {
  int val = 0;

  /*   リンクがあれば、リンクのチャネル１を呼び出して入力データを得る              */
  if (mtask->task.blink)
    val = (*mtask->task.blink->channel1)(mtask->task.blink);

  /*    データを処理せよ  */
  val = val + mtask->aval;

  /* 処理されたデータをリターン
                        せよ */
  return (val);
  }

/*****************************************************************/
/**                                                             **/
/**          チャネル２処理手順  －  一つの値で入力を乗算せよ   **/
/**                                                             **/
/*****************************************************************/

static int
do_mul(mtask)
MathTask *mtask;
  {
  int val = 1;

  /*       リンクがあれば、リンクのチャネル２を呼び出して入力データを得る          */
  if (mtask->task.blink)
  val = (*mtask->task.blink->channel2)(mtask->task.blink);

    /*  データを処理せよ    */
    val = val * mtask->mval;

    /* 処理されたデータをリターン
                        せよ */
    return (val);
    }

/*****************************************************************/
/**                                                             **/
/**                        例コード終了                         **/
/**                                                             **/
/*****************************************************************/
```

【図７】

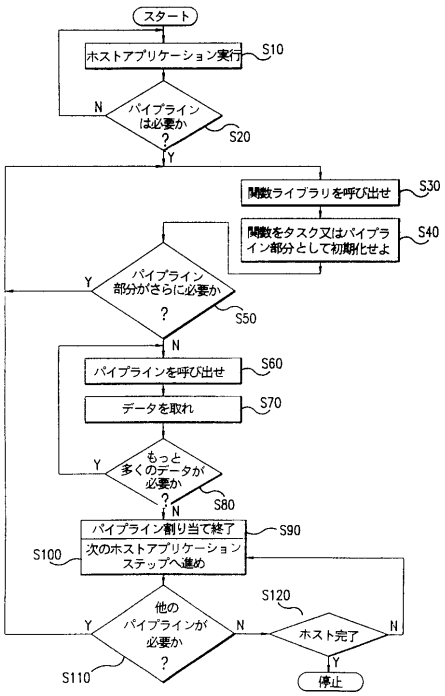
```
/*****************************************************************/
/**                                                             **/
/**      リスティング２）アプリケーション例  －  MathTaskパイプライン  **/
/**                                    を呼出し且つ実行せよ     **/
/**                                                             **/
/*****************************************************************/

#include <stdio.h>

main( argc,argv)
int argc;
char **argv;
  {
  int i,n,add_value,mul_value;
  Task *task = NULL;

  /*       コマンド引数からパイプラインのレングスを読み取れ              */
  n = atoi(argv[1]);

  /*      パイプラインの各タスクを初期化せよ        */
  for (i=1; i<=n; ++i)
    task = initMathTask(i,i,task);

  /*     チャネル１とチャネル２の両方に対してパイプラインを実行せよ       */
  add_value = (*task->channel1)(task);
  mul_value = (*task->channel2)(task);
  printf("   #tasks: %d, channel1: %d, channel2: %d",
        n, add_val, mul_val);
  }

/*****************************************************************/
/**                                                             **/
/**                    アプリケーション例の終了                 **/
/**                                                             **/
/*****************************************************************/
```

【図８】

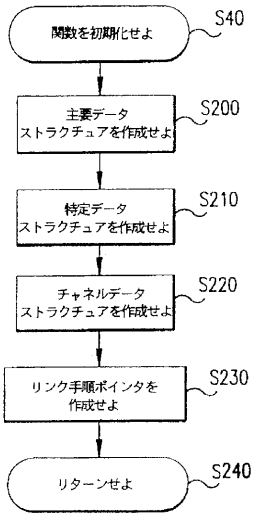
リスティング３）上記例の結果：

```
lpau:213> rrl_example 5
    #tasks: 5, channel1: 15, channel2: 120
```

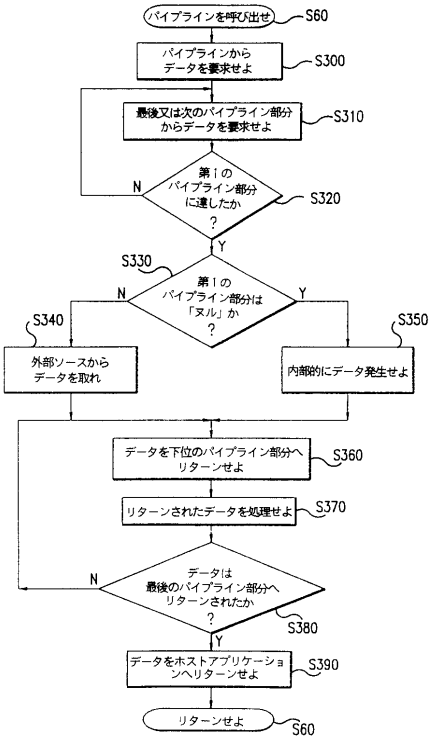
【図９】



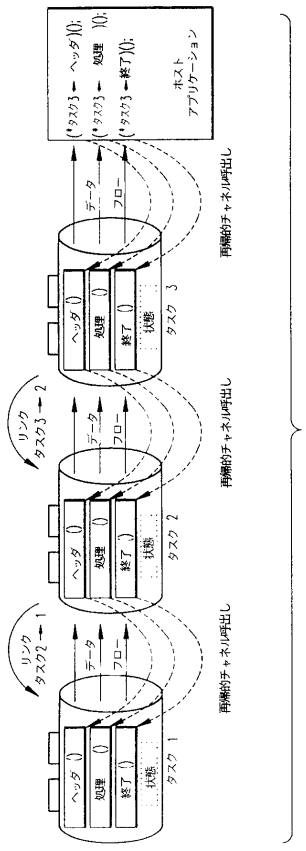
【図１０】



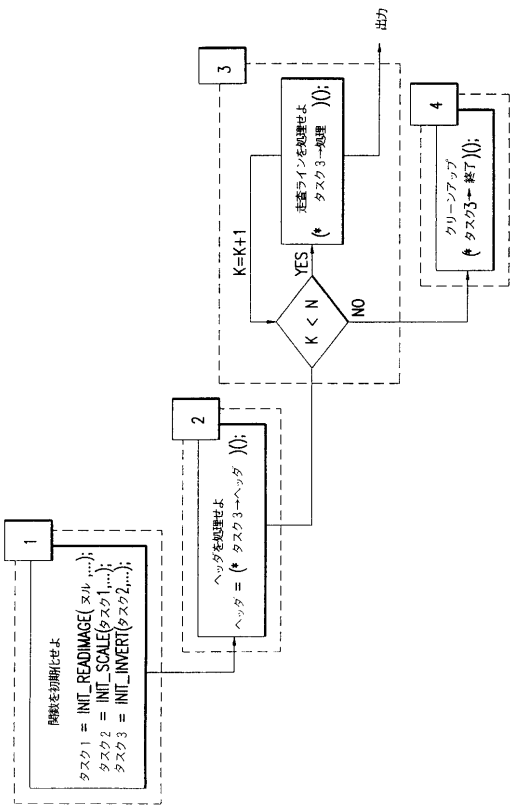
【図１１】



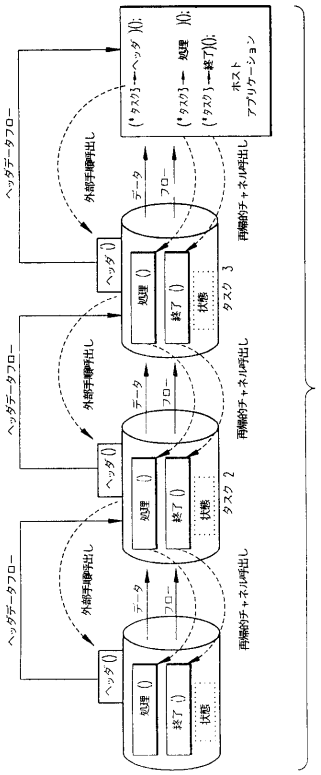
【図１２】



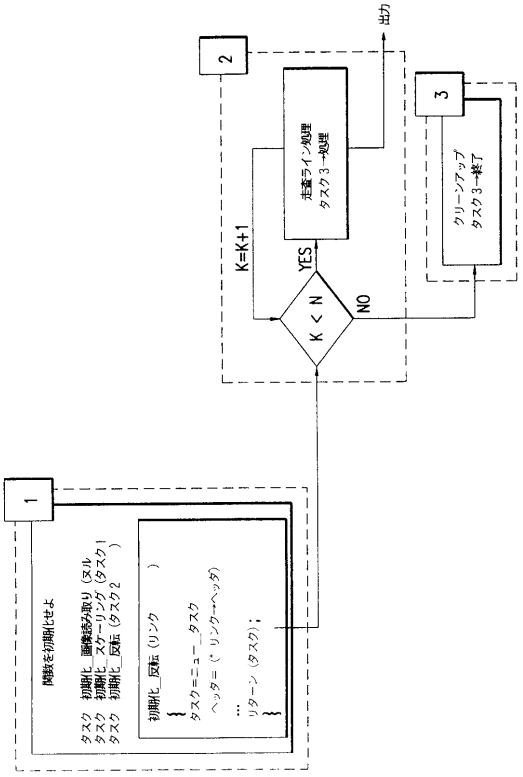
【図１３】



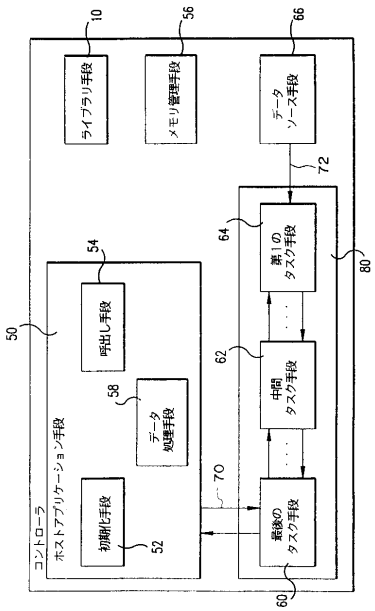
【図１４】



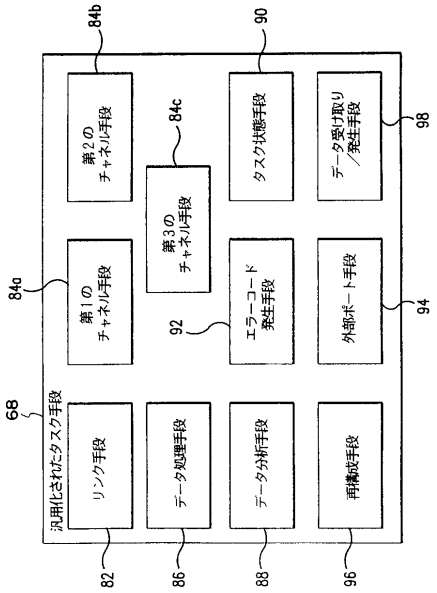
【図１５】



【図１６】



【図１７】


84b
90
84c
98
84a
92
94
68
汎用化されたタスク手段
第2のチャネル手段
第3のチャネル手段
タスク状態手段
データ受け取り／発生手段
第1のチャネル手段
エラーコード発生手段
外部ポート手段
リンク手段
データ処理手段
データ分析手段
再構成手段
82
86
88
96

フロントページの続き

審査官　久保　光宏

(56)参考文献　特開平２－２９７６３２（ＪＰ，Ａ）
特開昭６２－３８０６０（ＪＰ，Ａ）
特開昭６４－９１２６０（ＪＰ，Ａ）
特開昭５９－６０６５１（ＪＰ，Ａ）
特開平６－１８７４３４（ＪＰ，Ａ）
特開平１－２８４９６１（ＪＰ，Ａ）
特開平６－９６２１２（ＪＰ，Ａ）
特開昭５６－１７４３７（ＪＰ，Ａ）
特開平４－１３０９２８（ＪＰ，Ａ）
特開平１－１９３９３５（ＪＰ，Ａ）
「AIX/ESA 使用者の手引き第２版　リリース２　初版 SC88-6708-00」，日本，日本アイ・ビー・エム株式会社，１９９３年　5月，初版，pp.145-152
塩谷修・他，「実用ＵＮＩＸシステムプログラミング（第２版）」，日本，日刊工業新聞社，１９９３年　1月２０日，第２版，pp.177-198，ISBN:4-526-03261-1
Mark G. Sobell，「実用ＵＮＩＸシステム（第２版）」，日本，株式会社工学社，１９８８年１０月　5日，第２版，pp.105-108，ISBN:4-87593-099-2
「SX/AU カーネル説明書　初版 07SP-2510-1」，日本，富士通株式会社，１９８７年　3月，初版，pp.11-12
「FM Rシリーズ通信タスクモニタ(CPMGR)V3.1&DSLINKドライバV1.1使用手引書 87SP-6310-1-0」，日本，富士通株式会社，１９９３年　1月，初版，pp.124
廣田周吾・他，「分散型関数型言語Cmexの構成とその応用について」，情報処理学会研究報告，日本，社団法人情報処理学会，１９９２年　8月２１日，Vol.92，No.67（92-PRG-8），pp.83-90
長尾真・他，「岩波情報科学辞典」，日本，株式会社岩波書店，１９９０年　5月２５日，初版，pp.67，ISBN:4-00-080074-4
梶光雄，「高精細印刷画像標準化の動向」，テレビジョン学会技術報告，日本，社団法人テレビジョン学会，１９９０年１１月２２日，Vol.14，No.69，pp.17-24
インターフェース，日本，ＣＱ出版株式会社，１９８９年　7月　1日，Vol.15，No.7，pp.174-195，ISSN:0387-9569

(58)調査した分野(Int.Cl.[7]，ＤＢ名)
G06F9/46-9/54，
G06F9/38-9/42，
G06F9/455，
G06F13/00，
G06T1/20，
H04L13/00，
JSTファイル(JOIS)，
CSDB(日本国特許庁)，
INSPEC(DIALOG)