(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4756553号
(P4756553)
(45)発行日　平成23年8月24日(2011.8.24)　(24)登録日　平成23年6月10日(2011.6.10)

(51)Int.Cl.　F I
G06F 9/50 (2006.01)　G06F 9/46 465D

請求項の数 8 (全 17 頁)

(21)出願番号　特願2006-335130 (P2006-335130)
(22)出願日　平成18年12月12日 (2006.12.12)
(65)公開番号　特開2008-146503 (P2008-146503A)
(43)公開日　平成20年6月26日 (2008.6.26)
審査請求日　平成21年11月27日 (2009.11.27)

(73)特許権者　310021766
株式会社ソニー・コンピュータエンタテインメント
東京都港区港南１丁目７番１号
(74)代理人　100105924
弁理士　森下　賢樹
(74)代理人　100109047
弁理士　村田　雄祐
(74)代理人　100109081
弁理士　三木　友由
(74)代理人　100134256
弁理士　青木　武司

最終頁に続く

(54)【発明の名称】分散処理方法、オペレーティングシステムおよびマルチプロセッサシステム

(57)【特許請求の範囲】
【請求項１】
制御用のメインプロセッサ、複数の演算用のサブプロセッサ、およびメモリインタフェースがバスで接続されたマルチコアプロセッサと、共有メモリとを含み、前記メインプロセッサと複数の前記サブプロセッサが前記メモリインタフェースを介して前記共有メモリにアクセス可能なマルチプロセッサシステムにおける分散処理方法であって、
各サブプロセッサの計算資源を時分割して複数のタスクに割り当てることにより、複数のタスクが並列に実行されるマルチタスク環境において、タスクの実行結果を別のタスクに与えることにより、負荷の異なる複数のタスクからなる特定処理を実行するためのパイプライン処理系を構築し、当該パイプライン処理系を複数動作させ、前記共有メモリにコンテキストが退避されて実行可能状態にあるタスクをいずれのタスクも実行していないサブプロセッサに割り当てて実行させることにより、複数のパイプライン処理系で実行される前記特定処理の複数のタスクの内、処理時間が所定の閾値よりも長い各パイプライン処理系の高負荷タスクが異なるサブプロセッサに割り当てられて実行され、その結果、各サブプロセッサの負荷が均一化されることを特徴とする分散処理方法。
【請求項２】
前記サブプロセッサの数を前記特定処理を構成する前記高負荷タスクの数で除算して得られる値を超えない整数値の数だけ前記パイプライン処理系を動作させることを特徴とする請求項１に記載の分散処理方法。
【請求項３】

前記パイプライン処理系で実行される前記特定処理のタスク間でやりとりされるデータの入出力関係を記述した設定ファイルをもとにタスク間の入出力チャネルを構築し、前記入出力チャネルを介したタスク間のストリーム通信を実行することを特徴とする請求項１または２に記載の分散処理方法。

【請求項４】

前記設定ファイルに記述されたタスクの入出力経路に新たなタスクを直列または並列に挿入することにより、前記設定ファイルを動的に変更する手順をさらに含むことを特徴とする請求項３に記載の分散処理方法。

【請求項５】

前記パイプライン処理系で実行される前記特定処理の各タスクは、当該タスクを割り当てられたサブプロセッサによって互いに実行され、各サブプロセッサは、割り当てられたタスクの入力チャネルから入力を受け取って当該タスクを処理し、そのタスクの出力チャネルに実行結果を出力することを特徴とする請求項３に記載の分散処理方法。

【請求項６】

制御用のメインプロセッサ、複数の演算用のサブプロセッサ、およびメモリインタフェースがバスで接続されたマルチコアプロセッサと、共有メモリとを含み、前記メインプロセッサと複数の前記サブプロセッサが前記メモリインタフェースを介して前記共有メモリにアクセス可能なマルチプロセッサシステム上で動作するオペレーティングシステムであって、

各サブプロセッサの計算資源を時分割して複数のタスクに割り当てることにより、複数のタスクが並列に実行されるマルチタスク環境において、タスクの実行結果を別のタスクに与えることにより、負荷の異なる複数のタスクからなる特定処理を実行するためのパイプライン処理系を構築し、当該パイプライン処理系を複数動作させる機能と、

前記共有メモリにコンテキストが退避されて実行可能状態にあるタスクをいずれのタスクも実行していないサブプロセッサに割り当てて実行させることにより、複数のパイプライン処理系で実行される前記特定処理の複数のタスクの内、処理時間が所定の閾値よりも長い各パイプライン処理系の高負荷タスクを異なるサブプロセッサに割り当てて実行させ、その結果、各サブプロセッサの負荷を均一化させる機能とを前記マルチプロセッサシステムに実現させることを特徴とするオペレーティングシステム。

【請求項７】

制御用のメインプロセッサ、それぞれがローカルメモリをもつ複数の演算用のサブプロセッサ、およびメモリインタフェースがバスで接続されたマルチコアプロセッサと、共有メモリとを含み、前記メインプロセッサと複数の前記サブプロセッサが前記メモリインタフェースを介して前記共有メモリにアクセス可能なマルチプロセッサシステムであって、

前記複数の演算用のサブプロセッサ上で動作するオペレーティングシステムは、

各サブプロセッサの計算資源を時分割して複数のタスクに割り当てることにより、複数のタスクが並列に実行されるマルチタスク環境において、タスクの実行結果を別のタスクに与えることにより、負荷の異なる複数のタスクからなる特定処理を実行するためのパイプライン処理系を構築し、当該パイプライン処理系を複数動作させる機能と、

前記共有メモリにコンテキストが退避されて実行可能状態にあるタスクをいずれのタスクも実行していないサブプロセッサの前記ローカルメモリにロードして実行させることにより、複数のパイプライン処理系で実行される前記特定処理の複数のタスクの内、処理時間が所定の閾値よりも長い各パイプライン処理系の高負荷タスクを異なるサブプロセッサに割り当てて実行させ、その結果、各サブプロセッサの負荷を均一化させる機能とを含むことを特徴とするマルチプロセッサシステム。

【請求項８】

前記サブプロセッサに割り当てられた各タスクは、前記メインプロセッサを介在させることなく、通信チャネルを介して互いにデータをやりとりしながら実行されることを特徴とする請求項７に記載のマルチプロセッサシステム。

【発明の詳細な説明】

【技術分野】
【０００１】
この発明は、複数のプロセッサを含むマルチプロセッサシステムにおける分散処理技術に関する。
【背景技術】
【０００２】
ＣＰＵ（Central Processing Unit）の処理速度の高速化のために、ＣＰＵの動作周波数を上げる工夫がなされてきた。動作周波数を上げるためにＲＩＳＣ（Reduced Instruction Set Computer）アーキテクチャが採用され、ＣＰＵの各処理ステージの処理速度を均一化して並列度を高めるために深いパイプラインが形成されてきた。また、基板上の配線幅や配線間隔を小さくしてより多くの論理をチップ内に組み込んで高密度化を図るとともに、チップの消費電力を抑える工夫もなされてきた。
【０００３】
しかしながら、配線が細かくなったことでリーク電流が増加し、消費電力が下がらなくなってきた。また、消費電力を下げることができないため、ＣＰＵの動作周波数を上げることができず、従来のように動作周波数を上げることでＣＰＵの高速化を図るのは難しくなってきている。
【０００４】
また、ＣＰＵ単体の高速化とは別に、複数のＣＰＵをもつマルチプロセッサシステムによって処理の高速化が図られている。特にゲームのように複数のプロセスが動作するアプリケーションの高速化にはマルチプロセッサシステムが適している。
【０００５】
図１（ａ）、（ｂ）はＣＰＵの高速化の一手法であるパイプライン処理の原理を説明する図である。図１（ａ）に示すように、ＣＰＵでは、一つの命令（Instruction）が、フェッチ（ＩＦ;Instruction Fetch）、デコード（ＩＤ;Instruction Decode）、レジスタフェッチ（ＲＦ；Register Fetch）、実行（ＥＸＥ;Execution）、メモリアクセス（ＭＥＭ；Memory Access）、書き戻し（ＷＢ；Write Back）といった６つのステージを経て処理される。通常は、一つ前の命令の処理ステージがすべて完了しなければ次の命令の処理を開始することができないが、パイプライン処理では、各ステージの処理機構を独立して動作させることにより、流れ作業のように、一つ前の命令について全処理ステージの実行が完了する前に次の命令を処理し始めることができる。これにより、複数の命令からなるプログラム全体の処理時間の短縮を図ることができる。
【０００６】
図１（ｂ）は、６個の命令１～６がパイプライン処理される様子を示している。各ステージが１クロックサイクルで処理されるとすると、命令１は第１サイクルでＩＦステージに投入され、以降、１サイクル毎に後続のステージに結果が渡されて各ステージの処理が実行される。命令２は、命令１についてＩＦステージの処理が終わった後、第２サイクルでＩＦステージに投入され、以降、同様に１サイクル毎に後続の各ステージに送られ、処理される。同様に命令３～６は、それぞれ第３～第６サイクルでＩＦステージに投入されて各ステージの処理がなされる。第６サイクルにおいては、６つのステージが並列に動作して各命令を処理することになるから、ＣＰＵのスループットが最大化される。
【０００７】
特許文献１には、プログラムの処理時間を短縮することのできる命令パイプライン処理方法が開示されている。
【特許文献１】特開２０００－１７２５０２号公報
【発明の開示】
【発明が解決しようとする課題】
【０００８】
マルチプロセッサシステムにおいてプログラムを並列に実行させるためには、プログラムを並列に動作する複数のタスクに分解し、タスクをプロセッサに割り当てることにより

、並列に実行させる必要がある。プログラムを並列に動作するタスクに分解する作業は自動化が難しく、プログラマの手作業によるしかなく、並列処理のプログラミングはたいへん手間と時間がかかる。また、パイプライン処理は各ステージの処理時間が均一であることが前提となるため、処理時間の異なる複数のタスクからなるプログラムの処理にパイプライン処理の原理をそのまま適用することはできない。

【０００９】

本発明はこうした課題に鑑みてなされたものであり、その目的は、マルチプロセッサシステムにおいて分散処理を効率的に実行するための技術を提供することにある。

【課題を解決するための手段】

【００１０】

上記課題を解決するために、本発明のある態様は、複数のプロセッサを含むマルチプロセッサシステムにおける分散処理方法である。この方法では、各プロセッサの計算資源を時分割して複数のタスクに割り当てることにより、複数のタスクが並列に実行されるマルチタスク環境において、タスクの実行結果を別のタスクに与えることにより、負荷の異なる複数のタスクからなる特定処理を実行するためのパイプライン処理系を構築し、当該パイプライン処理系を複数動作させ、メインメモリにコンテキストが退避されて実行可能状態にあるタスクをいずれのタスクも実行していないプロセッサに割り当てて実行させることにより、複数のパイプライン処理系で実行される前記特定処理の複数のタスクの内、処理時間が所定の閾値よりも長い高負荷タスクが異なるプロセッサに割り当てられて実行される。

【００１１】

本発明の別の態様は、オペレーティングシステムである。このオペレーティングシステムは、複数のプロセッサを含むマルチプロセッサシステム上で動作するオペレーティングシステムであって、各プロセッサの計算資源を時分割して複数のタスクに割り当てることにより、複数のタスクが並列に実行されるマルチタスク環境において、タスクの実行結果を別のタスクに与えることにより、負荷の異なる複数のタスクからなる特定処理を実行するためのパイプライン処理系を構築し、当該パイプライン処理系を複数動作させる機能と、メインメモリにコンテキストが退避されて実行可能状態にあるタスクをいずれのタスクも実行していないプロセッサに割り当てて実行させる機能とを前記マルチプロセッサシステムに実現させる。

【００１２】

本発明のさらに別の態様は、マルチプロセッサシステムである。このマルチプロセッサシステムは、制御用のメインプロセッサと、それぞれがローカルメモリをもつ複数の演算用のサブプロセッサと、共有メモリとを含む。前記複数の演算用のサブプロセッサ上で動作するオペレーティングシステムは、各サブプロセッサの計算資源を時分割して複数のタスクに割り当てることにより、複数のタスクが並列に実行されるマルチタスク環境において、タスクの実行結果を別のタスクに与えることにより、負荷の異なる複数のタスクからなる特定処理を実行するためのパイプライン処理系を構築し、当該パイプライン処理系を複数動作させる機能と、前記共有メモリにコンテキストが退避されて実行可能状態にあるタスクをいずれのタスクも実行していないサブプロセッサの前記ローカルメモリにロードして実行させる機能とを含む。

【００１３】

なお、以上の構成要素の任意の組合せ、本発明の表現を方法、装置、システム、コンピュータプログラム、データ構造、記録媒体などの間で変換したものもまた、本発明の態様として有効である。

【発明の効果】

【００１４】

本発明によれば、マルチプロセッサシステムにおいて分散処理を効率的に実行することができる。

【発明を実施するための最良の形態】

【００１５】
実施の形態１
図２は、実施の形態１に係るマルチプロセッサシステム１００の構成図である。マルチコアプロセッサ１１０は、複数のプロセッサを一つのパッケージに集積したものであり、一つのプロセッサユニット（ＰＵ）１０、複数（ここでは８個）のサブプロセッサユニット（ＳＰＵ）２０ａ～２０ｈ、メモリインタフェース４０、Ｉ／Ｏインタフェース５０がリングバス３０で接続された構成である。ＰＵ１０およびＳＰＵ２０ａ～２０ｈは、メモリインタフェース４０を介してメインメモリ１２０にアクセスすることができる。Ｉ／Ｏインタフェース５０は、外部デバイスとのインタフェースである。
【００１６】
ＰＵ１０はマルチプロセッサシステム１００全体を制御するためのメインプロセッサであり、キャッシュメモリ１２をもつ。ＳＰＵ２０ａ～２０ｈは演算用のプロセッサであり、内部にローカルメモリ２２ａ～２２ｈをもつ。ＳＰＵ２０ａ～２０ｈは、ＤＭＡ（Direct Memory Access）コントローラ（ＤＭＡＣ）２４ａ～２４ｈをもち、ＤＭＡＣ２４ａ～２４ｈは、メインメモリ１２０とローカルメモリ２２ａ～２２ｈの間でデータをＤＭＡ転送する。以下、ＳＰＵ２０ａ～２０ｈを総称するときは添え字ａ～ｈを省略して単にＳＰＵ２０と表記する。また、８個のＳＰＵ２０ａ～２０ｈをＳＰＵ１～ＳＰＵ８と表記することもある。
【００１７】
各ＳＰＵ２０上では分散オペレーティングシステムが動作する。分散オペレーティングシステムにおけるカーネルプログラムが複数のＳＰＵ２０上で協調して動作し、各ＳＰＵ２０上で動作するユーザプログラムを時分割で切り替えながら実行する。プログラムの時分割切り替えにはＰＵ１０が介在しないため、非常に小さいオーバーヘッドでプログラムの切り替えが可能である。
【００１８】
ＳＰＵ２０上で動作するカーネルプログラムは、ＳＰＵ２０の計算資源、具体的にはＣＰＵタイムを時分割して複数のタスクに割り当てることにより、ＳＰＵ２０上で仮想的に複数のタスクが並列に実行されるマルチタスク環境を実現する。一つのユーザプログラムを複数のＳＰＵ２０で実行することも可能であり、複数のＳＰＵ２０は、ＰＵ１０を介さずに互いに直接通信することができる。
【００１９】
ユーザプログラムは複数のタスクに分割され、メインメモリ１２０に保持される。各ＳＰＵ２０は、メインメモリ１２０に保持された実行可能状態にあるタスクをローカルメモリ２２にＤＭＡ転送し、そのタスクを実行する。各ＳＰＵ２０は、時分割されたＣＰＵタイムをタスクの実行に割り当て、タスクを実行し、割り当てられたＣＰＵタイムを消費すると、タスクをローカルメモリ２２からメインメモリ１２０にＤＭＡ転送して退避する。タスクのコンテキストの退避処理はカーネルプログラムが自動的に実行するため、プログラマは意識する必要はない。
【００２０】
タスク間でデータのやりとりをするために、タスクを実行するＳＰＵ２０は互いに通信する。ＳＰＵ２０間の通信帯域は非常に広帯域であるため、一つのプログラムをタスクに分割して複数のＳＰＵ２０で分散処理してもレーテンシが問題となることはなく、むしろ複数のＳＰＵ２０で並列にタスクを処理することによりスループットが向上して、プログラムの処理時間が短くなる。また、タスクはメインメモリ１２０とローカルメモリ２２の間をＤＭＡ転送され、プロセッサは転送に関わらないため、プロセッサに負荷がかからない。
【００２１】
複数のＳＰＵ２０上で実現されるマルチタスク環境において、タスクの実行結果を別のタスクの入力として与えることにより、複数のタスクからなる特定処理を実行するためのパイプライン処理系が構築される。

【００２２】
パイプライン処理系で実行される特定処理のタスク間でデータストリームをやりとりするための入出力チャネルが構築され、入出力チャネルを介したタスク間のストリーム通信が実現される。各タスクは、当該タスクを割り当てられたプロセッサによって互いに非同期に実行され、各プロセッサは、割り当てられたタスクの入力チャネルから入力を受け取って当該タスクを処理し、そのタスクの出力チャネルに実行結果を出力する。
【００２３】
図３は、タスクがスケジュールされてプロセッサに割り当てられる様子を説明する図である。ここでは４つのＳＰＵ１～ＳＰＵ４にタスクが割り当てられる場合を示す。特定の処理が複数のタスクに分割され、これらのタスクはカーネルに実装されたタスクスケジューラによりスケジュールされ、いずれかのＳＰＵ２０に割り当てられて実行される。
【００２４】
いずれかのＳＰＵ２０によって実行されているタスクは、実行中状態（Ｒｕｎｎｉｎｇ状態）にあるという。ＳＰＵ２０が実行中状態にあるタスクを解放すると、そのタスクのコンテキストはメインメモリ１２０に退避される。メインメモリ１２０にコンテキストが退避されたタスクは、実行可能状態（Ｒｅａｄｙ状態）または待ち状態（Ｗａｉｔｉｎｇ）にある。
【００２５】
実行可能状態とは、タスクの入力チャネルに入力値が与えられており、タスクがいつでもＳＰＵ２０によって実行可能な状態にあることである。実行可能状態にあるタスクを割り当て可能なＳＰＵ２０が確保されると、そのタスクは実行中状態に遷移する。
【００２６】
待ち状態とは、タスクの実行に必要な条件が満たされていないため、タスクはまだ実行可能ではなく、待機している状態である。たとえば、タスクの入力チャネルにまだ入力値が与えられていない場合は、入力値が与えられるまではタスクは待ち状態にある。待ち状態にあるタスクの実行に必要な条件が満たされると、そのタスクは実行可能状態に遷移する。
【００２７】
図３では、ＳＰＵ１、ＳＰＵ２およびＳＰＵ４のローカルメモリ２２ａ、２２ｂ、２２ｄにはタスクがロードされて実行されている。ＳＰＵ３にはタスクが割り当てられていない。メインメモリ１２０の待ち行列には、待ち状態のタスクが１つ、実行可能なタスクが２つがキューイングされている。タスクスケジューラは、メインメモリ１２０に退避されている最初の実行可能なタスクをＳＰＵ３に割り当てる。割り当てられたタスクはＳＰＵ３のＤＭＡＣ２４ｃによりローカルメモリ２２ｃにＤＭＡ転送される。
【００２８】
メインメモリ１２０に退避されたタスクは待ち行列にキューイングされ、ラウンドロビン方式などによりスケジュールされ、タスクが割り当てられていないアイドル状態のＳＰＵ２０に割り当てられる。
【００２９】
図４（ａ）、（ｂ）は、特定処理を複数のタスクに分割してパイプライン処理する様子を説明する図である。マルチコアプロセッサ１１０内のＳＰＵが４個（ＳＰＵ１～ＳＰＵ４）であるとする。特定処理は４つのタスクＡ、Ｂ、Ｃ、Ｄに分割され、それぞれのタスクの処理時間Ｔは共通であるとする。
【００３０】
図４（ａ）には、６つの特定処理（Ｐ１～Ｐ６）がパイプライン処理される様子が示されている。各処理Ｐ１～Ｐ６を構成する４つのタスクＡ～Ｄには、ジョブ番号の添え字をつけている。処理Ｐ１について、時刻Ｔ０においてタスクＡ１が実行され、タスクＡ１の処理結果はタスクＢ１に渡され、時刻Ｔ１においてタスクＢ１が実行される。以降、同様にして処理結果を引き継ぎながら、時刻Ｔ２においてタスクＣ１が実行され、時刻Ｔ３においてタスクＤ１が実行される。

【００３１】
処理Ｐ２については、時刻Ｔ０においてジョブ１のタスクＡ１が実行された後、時刻Ｔ１においてタスクＡ２が実行され、以降、時刻Ｔ２、Ｔ３、Ｔ４において、タスクＢ２、Ｃ２、Ｄ２が順次実行される。
【００３２】
以下、同様に、処理Ｐ３～Ｐ６について、時刻Ｔ２～Ｔ５においてタスクＡ３～Ａ６の実行が開始され、以降、処理時間Ｔ毎に後続のタスクが順次実行される。
【００３３】
図４（ｂ）は、時刻Ｔ３、Ｔ４、Ｔ５における各ＳＰＵのタスク割り当て状況を示す。タスクＡはＳＰＵ１、タスクＢはＳＰＵ２、タスクＣはＳＰＵ３、タスクＤはＳＰＵ４にそれぞれ割り当てられたとする。その割り当てのもとで処理Ｐ１～Ｐ６が図４（ａ）に示すようにパイプライン処理されたとする。
【００３４】
時刻Ｔ３では、ＳＰＵ１にはタスクＡ４、ＳＰＵ２にはタスクＢ３、ＳＰＵ３にはタスクＣ２、ＳＰＵ４にはタスクＤ１がそれぞれ割り当てられる。
時刻Ｔ４では、ＳＰＵ１にはタスクＡ５、ＳＰＵ２にはタスクＢ４、ＳＰＵ３にはタスクＣ３、ＳＰＵ４にはタスクＤ２がそれぞれ割り当てられる。
時刻Ｔ５では、ＳＰＵ１にはタスクＡ６、ＳＰＵ２にはタスクＢ５、ＳＰＵ３にはタスクＣ４、ＳＰＵ４にはタスクＤ３がそれぞれ割り当てられる。
このように、時刻Ｔ３～Ｔ５の間、４つのＳＰＵ１～ＳＰＵ４には一つずつタスクが割り当てられているから、負荷の比率は１：１：１：１である。
【００３５】
図４（ａ）、（ｂ）の例では、各タスクの処理時間が同じであったが、これは特殊な場合であり、一般には特定処理は負荷の異なる複数のタスクに分割される。
【００３６】
図５（ａ）、（ｂ）は、負荷の異なる複数のタスクから構成される特定処理がパイプライン処理される様子を説明する図である。特定処理は４つのタスクＡ、Ｂ、Ｃ、Ｄに分割される。タスクＣの処理時間は、タスクＡ、Ｂ、Ｄの処理時間Ｔの３倍であるとする。
【００３７】
図５（ａ）には、８つの特定処理Ｐ１～Ｐ８がパイプライン処理される様子が示されている。各処理は処理時間Ｔだけずらして投入される。タスクＡはＳＰＵ１、タスクＢはＳＰＵ２、タスクＣはＳＰＵ３、タスクＤはＳＰＵ４にそれぞれ割り当てられたとする。タスクＣの処理時間が他のタスクよりも長いため、スループットが最大化される時刻Ｔ５～Ｔ７において、特定のＳＰＵにタスクＣが複数個割り当てられることになる。
【００３８】
図５（ｂ）は、時刻Ｔ５、Ｔ６、Ｔ７における各ＳＰＵのタスク割り当て状況を示す。
時刻Ｔ５では、ＳＰＵ１にはタスクＡ６、ＳＰＵ２にはタスクＢ５、ＳＰＵ３にはタスクＣ４、Ｃ３、Ｃ２、ＳＰＵ４にはタスクＤ１がそれぞれ割り当てられる。
時刻Ｔ６では、ＳＰＵ１にはタスクＡ７、ＳＰＵ２にはタスクＢ６、ＳＰＵ３にはタスクＣ５、Ｃ４、Ｃ３、ＳＰＵ４にはタスクＤ２がそれぞれ割り当てられる。
時刻Ｔ７では、ＳＰＵ１にはタスクＡ８、ＳＰＵ２にはタスクＢ７、ＳＰＵ３にはタスクＣ６、Ｃ５、Ｃ４、ＳＰＵ４にはタスクＤ３がそれぞれ割り当てられる。
このように、時刻Ｔ５～Ｔ７の間、ＳＰＵ１、ＳＰＵ２、ＳＰＵ４には一つずつタスクが割り当てられているが、ＳＰＵ３には３つのタスクが割り当てられている。したがって、ＳＰＵ１～ＳＰＵ４の負荷の比率は１：１：３：１になる。
【００３９】
このように、特定処理を構成する各タスクの処理時間が異なる場合に、パイプライン処理を実行すると、高負荷のタスクが割り当てられたＳＰＵに負荷が集中してしまい、マルチコアプロセッサ１１０の演算処理のスループットを上げることができなくなる。そこで本実施の形態では、特定処理を実行するパイプライン処理系を複数動作させ、各パイプラ

イン処理で実行される特定処理の高負荷タスクを異なるＳＰＵに割り当てることで負荷分散を図る。

【００４０】

図６（ａ）は、１つのパイプライン処理系をマルチプロセッサシステム１００で動作させた場合のタスクのプロセッサへの割り当てを説明する図である。タスクＡ、Ｂ、Ｃ、Ｄを順次実行するパイプライン処理系１５０を４つのＳＰＵ１～ＳＰＵ４で実行する場合、たとえば、タスクＡはＳＰＵ１に、タスクＢはＳＰＵ２に、タスクＣはＳＰＵ３に、タスクＤはＳＰＵ４にそれぞれ割り当てられる。したがって、タスクＣの処理時間が長い場合、ＳＰＵ３に負荷が集中する。

【００４１】

図６（ｂ）は、複数のパイプライン処理系をマルチプロセッサシステム１００で動作させた場合のタスクのプロセッサへの割り当てを説明する図である。４つのパイプライン処理系＃１～＃４（符号１５１～１５４）を並列に実行させるとする。

【００４２】

４つのパイプライン処理系＃１～＃４のそれぞれのタスクＡ～Ｄのすべてがタスクスケジューリングの対象となる。パイプライン処理系＃１の高負荷タスクＣがＳＰＵ３に割り当てられたとする。このとき、ＳＰＵ３は高負荷タスクＣの実行に時間がかかり、他のタスクのためにアイドル状態になるまでに時間がかかる。したがって、パイプライン処理系＃２の高負荷タスクＣは同じＳＰＵ３に割り当てられることはなく、別のＳＰＵ、たとえばＳＰＵ４に割り当てられる。

【００４３】

同様に、高負荷タスクＣを割り当てられたＳＰＵ４はアイドル状態になるまでに時間を要するため、パイプライン処理系＃３の高負荷タスクＣはＳＰＵ３、ＳＰＵ４には割り当てることができず、それ以外のＳＰＵ、たとえばＳＰＵ１に割り当てられる。さらに、パイプライン処理系＃４の高負荷タスクＣは、既に高負荷タスクＣを割り当てられたＳＰＵ３、ＳＰＵ４、ＳＰＵ１を避けて、ＳＰＵ２に割り当てられる。このように、４つのパイプライン処理系＃１～＃４を動作させれば、各パイプライン処理系の高負荷タスクＣを異なるＳＰＵに割り当てることができるようになり、特定のＳＰＵに負荷が集中するのを避けることができる。

【００４４】

なお、マルチプロセッサシステム１００において、タスクのＳＰＵ２０への割り当ては、ＰＵ１０が行うのではなく、各ＳＰＵ２０が自律的にメインメモリ１２０の待ち行列から実行可能なタスクを取り出すことで行われる。したがって、図６（ｂ）の４つのパイプライン処理系＃１～＃４のそれぞれの高負荷タスクＣのＳＰＵ１～ＳＰＵ４への割り当ては事前に決まっているのではない。ＳＰＵ１～ＳＰＵ４がそれぞれ、待ち行列から実行可能なタスクを取り出して実行することで、結果的に、４つの高負荷タスクＣが４つのＳＰＵ１～ＳＰＵ４に分散されて割り当てられることになる。

【００４５】

図７（ａ）～（ｅ）は、複数のパイプライン処理系を実行するマルチプロセッサシステム１００において、各プロセッサのタスク割り当て状況と全プロセッサの負荷比率を説明する図である。

【００４６】

図７（ａ）は、図６（ｂ）のパイプライン処理系＃１について、時刻Ｔ５、Ｔ６、Ｔ７における各ＳＰＵのタスク割り当て状況を示す。これは図４（ｂ）のタスク割り当て状況と同じである。図７（ｂ）は、パイプライン処理系＃２について、時刻Ｔ５、Ｔ６、Ｔ７における各ＳＰＵのタスク割り当て状況を示す。パイプライン処理系＃２ではタスクＣはＳＰＵ４に割り当てられるから、ＳＰＵ４にタスクＣが３つ割り当てられることになる。同様に、図７（ｃ）に示すように、パイプライン処理系＃３についてはＳＰＵ１にタスクＣが３つ割り当てられ、図７（ｄ）に示すように、パイプライン処理系＃４についてはＳＰＵ２にタスクＣが３つ割り当てられる。

【００４７】
図７（ｅ）は、ＳＰＵ１～ＳＰＵ４の負荷比率を説明する図である。図７（ａ）～図７（ｄ）に示したパイプライン処理系＃１～＃４のタスク割り当てにより、高負荷タスクＣはＳＰＵ１～ＳＰＵ４に分散され、負荷が均一化されるから、ＳＰＵ１～ＳＰＵ４の負荷比率は１：１：１：１になる。
【００４８】
以上述べたように、特定処理を実行するためのパイプライン処理系を複数動作させれば、各パイプライン処理系の高負荷タスクが異なるＳＰＵに分散されて割り当てられることになり、マルチプロセッサシステム１００のスループットを向上させることができる。
【００４９】
図１（ａ）、（ｂ）で説明したように、ＣＰＵは、命令のフェッチ、デコードなどの専用の回路をもち、これらの専用回路を並列に動作させてパイプライン処理を実行する。パイプライン処理の各ステージの処理時間を等しくするために、これらの専用回路における処理時間は等しくなるように（たとえば、１クロックサイクルで実行が完了するように）設計されている。このように、一般的なＣＰＵにおけるパイプライン処理では、命令を実行時間が同じサイクル数で実行できるステージに細分化し、並列処理を行うことでスループットの向上を図っている。パイプライン処理を実現するためには、各ステージの処理サイクル数が揃うように機能を分割する必要がある。
【００５０】
それに対して、マルチプロセッサシステム１００の複数のＳＰＵ２０は、汎用のプロセッサであるため、どんな処理でも実行可能である。したがって、特定処理を複数の異なるタスクに分割してパイプライン処理するとき、各ＳＰＵ２０にはどのタスクを割り当ててもかまわない。したがって、特定処理を実行するためのパイプライン処理系を複数動作させて、各パイプライン処理系の高負荷タスクをアイドル状態にある異なる複数のＳＰＵ２０に分散させて割り当てることが可能である。それにより、特定のＳＰＵ２０に負荷が集中する事態を避けることができ、全体のスループットを上げることができる。
【００５１】
ＳＰＵ２０に割り当てられるタスクの負荷を均一にする必要はないから、特定処理が負荷の異なるタスクに分割されるという一般的な状況に対応することができ、並列プログラミングが容易になる。また、タスクの数とタスクを割り当てるプロセッサの数は独立しているため、プログラマは、物理的なプロセッサの構成を気にする必要もない。
【００５２】
特定処理を構成する高負荷タスクが２つあり、この特定処理を実行するためパイプライン処理系を４つ動作させるとすると、高負荷タスクが全体で８個あることになる。高負荷タスクを割り当てるプロセッサ数も８個あると、各プロセッサに高負荷タスクを１つずつ割り当てることができるため、好都合である。一般にはプロセッサの数はシステムによって決まっているから、特定処理を構成する高負荷タスクの数が与えられると、プロセッサの数を特定処理の高負荷タスクの数で除算して得られる値を超えない整数値の数だけパイプライン処理系を動作させればよい。
【００５３】
実施の形態２
実施の形態２は、実施の形態１のマルチプロセッサシステム１００におけるパイプライン処理をビデオやオーディオからメタ情報を抽出する処理に応用したものである。
【００５４】
図８は、実施の形態２に係るメタ情報処理装置２００の機能構成図である。同図は機能に着目したブロック図を描いており、これらの機能ブロックはハードウェアのみ、ソフトウェアのみ、またはそれらの組合せによっていろいろな形で実現することができる。すなわち、これらの機能構成ブロックの少なくとも一部は、図２で説明したマルチプロセッサシステム１００のハードウェア構成により実現され、ハードウェア構成で実現されない機能ブロックは、メインメモリ１２０にロードされたプログラムをＰＵ１０またはＳＰＵ２

０が実行することにより実現される。

【００５５】

メタ情報処理装置２００は、ビデオやオーディオなどを含むコンテンツからメタ情報を抽出する装置であり、メタ情報抽出ブロック２１０と、ジョブデータベース２２０と、メタ情報データベース２３０とを有する。

【００５６】

メタ情報抽出ブロック２１０は、ジョブデータベース２２０からジョブを取り出し、コンテンツに含まれるビデオ、オーディオ、テキストからそれぞれメタ情報を抽出し、抽出したメタ情報をメタ情報データベース２３０に登録する。メタ情報抽出ブロック２１０によるメタ情報抽出処理は、複数のタスクに分解される。メタ情報抽出処理を構成するタスクをマルチプロセッサシステム１００の複数のＳＰＵ２０に割り当て、パイプライン処理によって実行する。

【００５７】

メタ情報抽出ブロック２１０の各機能構成を説明する。リクエスト処理部２４０は、ジョブデータベース２２０からジョブを取り出し、ファイル転送部２４２に与える。ファイル転送部２４２はジョブによって指定されたファイルをデータベースから読み出し、ファイル解析・分割部２４４に与える。ファイル解析・分割部２４４は、ファイルを解析し、オーディオデータをオーディオデコーダ２５０に、テキストデータをテキスト変換部２６４に、画像データを画像デコーダ２５２に与える。

【００５８】

オーディオデコーダ２５０は、オーディオデータを復号し、１２音解析部２６０に与え得る。画像デコーダ２５２は、画像データを復号し、画像解析部２６２に与える。

【００５９】

１２音解析部２６０は、復号されたオーディオデータを１２音解析技術により周波数分析して特徴量を抽出する。１２音解析部２６０は、抽出した特徴量にもとづいてオーディオのメタデータを生成し、データベース登録部２７０に与える。

【００６０】

テキスト変換部２６４は、テキストデータからキーワードなどを抽出することでテキストのメタデータを生成し、データベース登録部２７０に与える。

【００６１】

画像解析部２６２は、復号された画像データを画像処理して特徴量を抽出し、抽出した特徴量にもとづいてメタデータを生成する。生成された画像のメタデータはデータベース登録部２７０に与えられる。

【００６２】

データベース登録部２７０は、オーディオのメタデータ、テキストのメタデータ、画像のメタデータをコンテンツのメタ情報としてメタ情報データベース２３０に登録する。データベース登録部２７０による登録が終わると、結果通知部２７２は、ジョブデータベース２２０にジョブの完了を通知する。

【００６３】

メタ情報抽出ブロック２１０の各機能構成がタスクに対応し、各機能構成の入出力がそのままタスクの入出力チャネルに対応する。

【００６４】

図９は、メタ情報抽出パイプライン処理系３００を１つだけ動作させてメタ情報抽出処理を実行した場合の各タスクのプロセッサへの割り当てを説明する図である。このパイプライン処理系３００で実行されるメタ情報抽出処理は、リクエスト処理部２４０、ファイル転送部２４２、ファイル解析・分割部２４４、オーディオデコーダ２５０、１２音解析部２６０、データベース登録部２７０、および結果通知部２７２の７つのタスクからなり、それぞれのタスクは、ＳＰＵ１～ＳＰＵ７に割り当てられ、ＳＰＵ８はアイドル状態である。１２音解析部２６０がもっとも負荷の高いタスクであるため、このタスクを割り当てられたＳＰＵ５は稼働率が１００％になる。オーディオデコーダ２５０は次に負荷の高

いタスクであり、このタスクを割り当てられたＳＰＵ４は稼働率が５０％となる。それ以外の処理時間が短いタスクを割り当てられたＳＰＵの稼働率は極めて低いものとなっている。

【００６５】

図１０（ａ）～（ｄ）は、４つのメタ情報抽出パイプライン処理系＃１～＃４（符号３００ａ～３００ｄ）を動作させてメタ情報抽出処理を実行した場合の高負荷タスクのプロセッサへの割り当てを説明する図である。各パイプライン処理系＃１～＃４において、１２音解析部２６０ａ～２６０ｄと画像解析部２６２ａ～画像解析部２６２ｄが高負荷タスクであり、合計すると８個の高負荷タスクがある。

【００６６】

」

８つのＳＰＵ１～ＳＰＵ８は、処理時間の短いタスクを実行していることもあるが、そのタスクの実行が終わると、アイドル状態になる。８個の高負荷タスクは、アイドル状態のＳＰＵに割り当てられていくことになり、いったん高負荷タスクが割り当てられたＳＰＵは再びアイドル状態になるまで時間がかかる。結果的に、８個のＳＰＵに均等に８個の高負荷タスクが割り当てられることになる。

【００６７】

同図に示すように、パイプライン処理系＃１の１２音解析部２６０ａはＳＰＵ１に、画像解析部２６２ａはＳＰＵ２にそれぞれ割り当てられる。パイプライン処理系＃２の１２音解析部２６０ｂはＳＰＵ３に、画像解析部２６２ｂはＳＰＵ４にそれぞれ割り当てられる。パイプライン処理系＃３の１２音解析部２６０ｃはＳＰＵ５に、画像解析部２６２ｃはＳＰＵ６にそれぞれ割り当てられる。パイプライン処理系＃４の１２音解析部２６０ｄはＳＰＵ７に、画像解析部２６２ｄはＳＰＵ８にそれぞれ割り当てられる。これにより、ＳＰＵ１～ＳＰＵ８はすべて稼働率１００％になり、マルチプロセッサシステム１００の処理スループットが最大化する。

【００６８】

図１１（ａ）～（ｃ）は、メタ情報抽出パイプライン処理系のバリエーションを説明する図である。図１１（ａ）のパイプライン処理系Ａ（符号３００）は、オーディオのメタデータを抽出するタスクで構成される。これに対して、図１１（ｂ）のパイプライン処理系Ｂ（符号３１０）は、オーディオ解析と画像解析を行い、オーディオのメタデータと画像のメタデータを抽出するタスクで構成される。図１１（ａ）と比べると、画像デコーダ２５２と画像解析部２６２のタスクが、オーディオデコーダ２５０と１２音解析部２６０のタスクに対して並列に追加されている。

【００６９】

このようなタスクの追加は、実行ファイルを変更することなく、タスクの接続関係を記述した設定ファイルを変更するだけで行うことができ、設定ファイルの変更により動的にパイプライン処理系のタスクを自由自在に変更できる。この設定ファイルはたとえば、ＸＭＬ（Extensible Markup Language）により記述される。設定ファイルは、タスク間でやりとりされるデータの入出力関係を記述したものであり、これをもとにタスク間の入出力チャネルが構築され、入出力チャネルを介してタスク間のストリーム通信が実行される。

【００７０】

たとえば、タスクＡの出力をタスクＢの入力に接続するには、「タスクＡの出力チャネルをタスクＢの入力チャネルに接続する」という記述を設定ファイルに設ければよい。図１１（ａ）のパイプライン処理系Ａの例で言えば、オーディオデコーダ２５０の出力チャネルを１２音解析部２６０の入力チャネルに接続するという記述を設けることになる。

【００７１】

図１１（ｂ）のパイプライン処理系Ｂを構築するには、図１１（ａ）のパイプライン処理系Ａの設定ファイルを一部変更するだけでよい。ファイル解析・分割部２４４の出力チャネルをオーディオデコーダ２５０の入力チャネルだけでなく、画像デコーダ２５２の入力チャネルにもつなげる。そして、画像デコーダ２５２の出力チャネルを画像解析部２６

２の入力チャネルにつなげる。さらに、画像解析部２６２の出力チャネルをデータベース登録部２７０の入力チャネルにつなげる。以上の設定ファイルの変更により、図１１（ａ）の処理系Ａが図１１（ｂ）の処理系Ｂに変更される。

【００７２】

図１１（ｃ）は、図１１（ａ）の１２音解析部２６０を１２音低レベル解析部２６０ａと１２音高レベル解析部２６０ｂにタスク分解したパイプライン処理系Ｃを示す。１２音低レベル解析部２６０ａは、人間の聴覚能力に応じた低レベルの特徴を解析するものであり、一例としてビート検出、コード進行検出、楽曲構造解析、キー（調）検出などを行い、ビート、コード、曲構造などの低レベルの特徴量を抽出する。１２音高レベル解析部２６０ｂは、人間の知的な理解能力に応じた高レベルの特徴を解析するものであり、一例として統計解析、ジャンル判別、楽器音検出、ムード検出、音質検出などを行い、ジャンル、楽器音、ムード、音質などの高レベルの特徴量を抽出する。

【００７３】

１２音低レベル解析部２６０ａと１２音高レベル解析部２６０ｂは両方とも高負荷タスクであるが、このように２つのタスクに分解すれば、別々のＳＰＵに割り当てられることになるから、負荷の分散を図ることができる。図１１（ｃ）のパイプライン処理系Ｃを構築するには、図１１（ａ）のパイプライン処理系Ａの設定ファイルを一部変更する。図１１（ａ）のオーディオデコーダ２５０から１２音解析部２６０を経てデータベース登録部２７０に至るデータストリームの経路において、１２音解析部２６０を削除し、１２音低レベル解析部２６０ａと１２音高レベル解析部２６０ｂを直列につなぎ直す。

【００７４】

このようにパイプライン処理系の新たにタスクを追加したり、タスクの構成を変更する際、一般にはタスクの負荷を考慮してパイプライン処理系を適切に設計する必要がある。しかし、本実施の形態では、複数のパイプライン処理系を動作させてタスクをプロセッサに割り当てる構成であり、負荷が自動的に均一化されて複数のプロセッサに分散される。そのため、プログラマはタスクの負荷に注意を払うことなく、単に設定ファイルを変更するだけで簡単にパイプライン処理系を設計し直すことができ、プログラミングに柔軟性をもたせることができる。

【００７５】

図１２は、４つのメタ情報抽出パイプライン処理系＃１～＃４が互いに独立にデータベースにアクセスしながら並列に動作する構成を説明する図である。マルチプロセッサシステム１００の複数のＳＰＵ２０は、ＰＵ１０の介在なしに、互いに独立に自律的に動作し、必要に応じてタスク間通信により協調動作する。ＰＵ１０が集中管理すると、ＰＵ１０に負荷が集中してボトルネックになるからである。

【００７６】

図１２に示すように、各パイプライン処理系＃１～＃４のリクエスト処理部２４０ａ～２４０ｄは互いに独立にジョブデータベース２２０をポーリングし、未処理のジョブがジョブデータベース２２０に登録されると、ＳＰＵ２０はその未処理ジョブを開始する。

【００７７】

各パイプライン処理系＃１～＃４のデータベース登録部２７０ａ～２７０ｄは互いに独立にメタ情報データベース２３０にメタデータを登録する。また、結果通知部２７２ａ～２７２ｄも、互いに独立にジョブデータベース２２０にジョブの終了を通知する。

【００７８】

ジョブの投入とジョブの実行結果の登録は同期させる必要がないため、制御用のＰＵ１０が介在して同期を取ったり、排他制御をすることはない。そのため、メタ情報抽出処理を複数のタスクに分割し、複数のＳＰＵ２０がタスクを自律的に実行するという複雑な処理系を構成しても、制御用のＰＵ１０に負荷が集中してボトルネックとなる事態を避けることができる。

【００７９】

以上、本発明を実施の形態をもとに説明した。実施の形態は例示であり、それらの各構

成要素や各処理プロセスの組合せにいろいろな変形例が可能なこと、またそうした変形例も本発明の範囲にあることは当業者に理解されるところである。

【図面の簡単な説明】

【００８０】

【図１】図１（ａ）、（ｂ）はＣＰＵの高速化の一手法であるパイプライン処理の原理を説明する図である。

【図２】実施の形態１に係るマルチプロセッサシステムの構成図である。

【図３】タスクがスケジュールされてプロセッサに割り当てられる仕組みを説明する図である。

【図４】図４（ａ）、（ｂ）は、特定処理を複数のタスクに分割してパイプライン処理する様子を説明する図である。

【図５】図５（ａ）、（ｂ）は、負荷の異なる複数のタスクから構成される特定処理がパイプライン処理される様子を説明する図である。

【図６】図６（ａ）は、１つのパイプライン処理系をマルチプロセッサシステムで動作させた場合のタスクのプロセッサへの割り当てを説明する図であり、図６（ｂ）は、複数のパイプライン処理系をマルチプロセッサシステムで動作させた場合のタスクのプロセッサへの割り当てを説明する図である。

【図７】図７（ａ）～（ｅ）は、複数のパイプライン処理系を実行するマルチプロセッサシステムにおいて、各プロセッサのタスク割り当て状況と全プロセッサの負荷比率を説明する図である。

【図８】実施の形態２に係るメタ情報処理装置の機能構成図である。

【図９】メタ情報抽出パイプライン処理系を１つだけ動作させてメタ情報抽出処理を実行した場合の各タスクのプロセッサへの割り当てを説明する図である。

【図１０】図１０（ａ）～（ｄ）は、４つのメタ情報抽出パイプライン処理系＃１～＃４（符号３００ａ～３００ｄ）を動作させてメタ情報抽出処理を実行した場合の高負荷タスクのプロセッサへの割り当てを説明する図である。

【図１１】図１１（ａ）～（ｃ）は、メタ情報抽出パイプライン処理系のバリエーションを説明する図である。

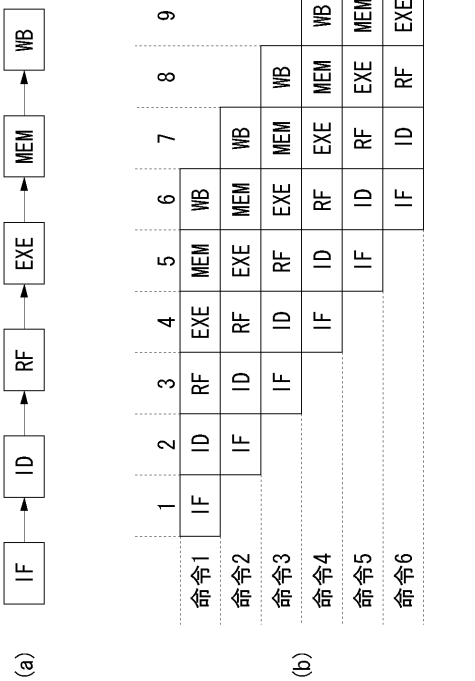
【図１２】４つのメタ情報抽出パイプライン処理系が互いに独立にデータベースにアクセスしながら並列に動作する構成を説明する図である。

【符号の説明】

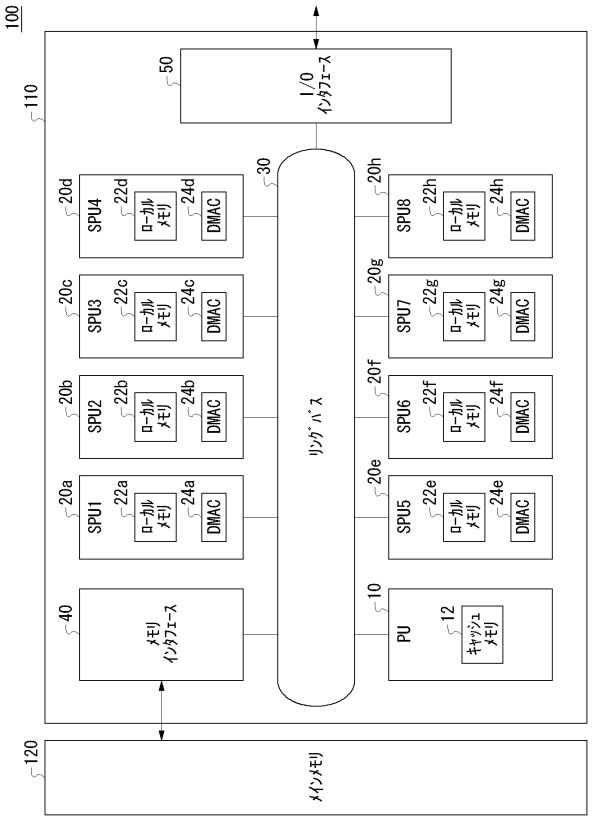
【００８１】

１　プロセッサユニット、　２０　サブプロセッサユニット、　２２　ローカルメモリ、　２４　ＤＭＡコントローラ、　３０　リングバス、　４０　メモリインタフェース、　５０　Ｉ／Ｏインタフェース、　１００　マルチプロセッサシステム、　１１０　マルチコアプロセッサ、　１２０　メインメモリ、　１５０　パイプライン処理系、　２００　メタ情報処理装置、　２１０　メタ情報抽出ブロック、　２２０　ジョブデータベース、　２３０　メタ情報データベース、　２４０　リクエスト処理部、　２４２　ファイル転送部、　２４４　ファイル解析・分割部、　２５０　オーディオデコーダ、　２５２　画像デコーダ、　２６０　１２音解析部、　２６２　画像解析部、　２６４　テキスト変換部、　２７０　データベース登録部、　２７２　結果通知部、　３００　メタ情報抽出パイプライン処理系。

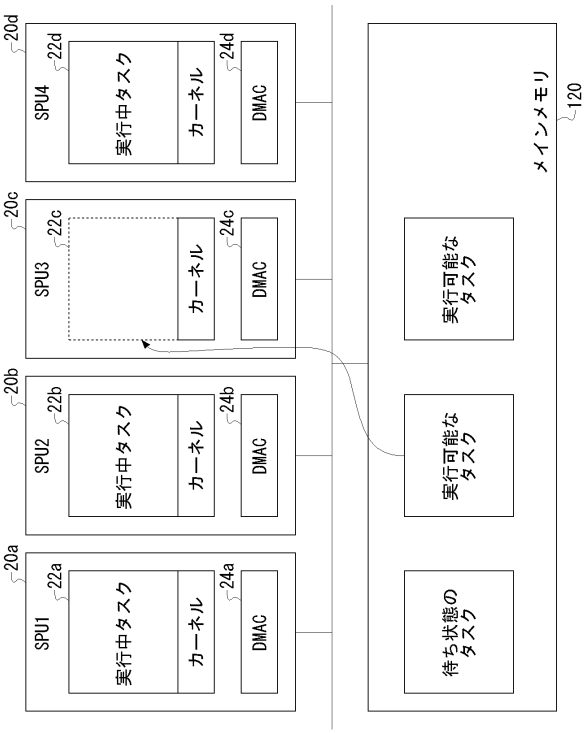
【図１】


(a)
IF
ID
RF
EXE
MEM
WB
(b)
1 2 3 4 5 6 7 8 9 10 11
命令1 IF ID RF EXE MEM WB
命令2 IF ID RF EXE MEM WB
命令3 IF ID RF EXE MEM WB
命令4 IF ID RF EXE MEM WB
命令5 IF ID RF EXE MEM WB
命令6 IF ID RF EXE MEM WB
1サイクル

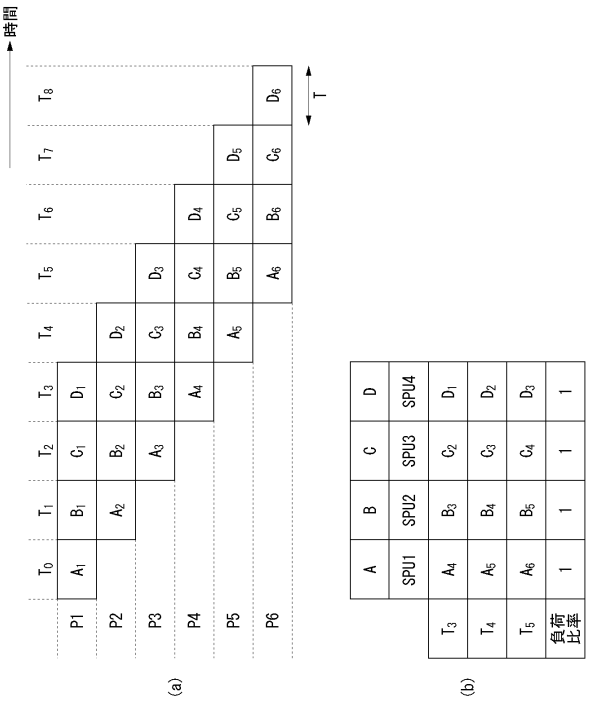
【図２】


100
110
50
I/Oインタフェース
30
リングバス
40
メモリインタフェース
10
PU
12
キャッシュメモリ
120
メインメモリ
SPU1 20a 22a ローカルメモリ 24a DMAC
SPU2 20b 22b ローカルメモリ 24b DMAC
SPU3 20c 22c ローカルメモリ 24c DMAC
SPU4 20d 22d ローカルメモリ 24d DMAC
SPU5 20e 22e ローカルメモリ 24e DMAC
SPU6 20f 22f ローカルメモリ 24f DMAC
SPU7 20g 22g ローカルメモリ 24g DMAC
SPU8 20h 22h ローカルメモリ 24h DMAC

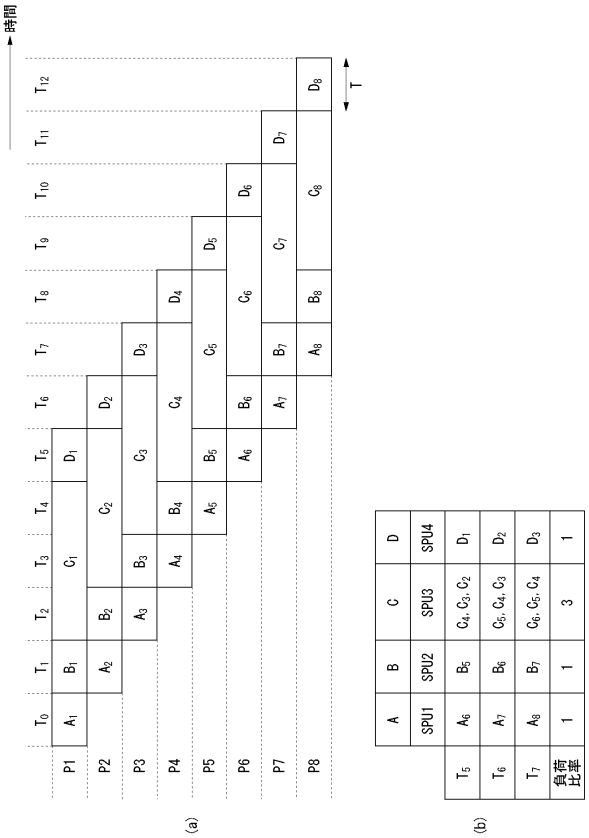
【図３】


SPU1 20a 22a 実行中タスク カーネル 24a DMAC
SPU2 20b 22b 実行中タスク カーネル 24b DMAC
SPU3 20c 22c カーネル 24c DMAC
SPU4 20d 22d 実行中タスク カーネル 24d DMAC
待ち状態のタスク
実行可能なタスク
実行可能なタスク
メインメモリ 120

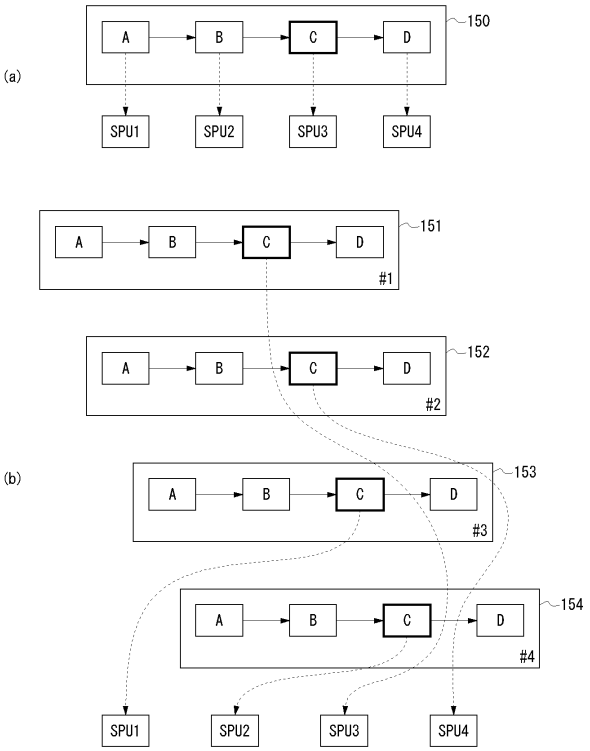
【図４】


時間
T
(a)
P1 P2 P3 P4 P5 P6
(b)
A B C D
SPU1 SPU2 SPU3 SPU4
負荷比率

【図５】



【図６】



【図７】

(a)

#1	A	B	C	D
	SPU1	SPU2	SPU3	SPU4
T_5	A_6	B_5	C_4, C_3, C_2	D_1
T_6	A_7	B_6	C_5, C_4, C_3	D_2
T_7	A_8	B_7	C_6, C_5, C_4	D_3

(b)

#2	A	B	C	D
	SPU1	SPU2	SPU4	SPU3
T_5	A_6	B_5	C_4, C_3, C_2	D_1
T_6	A_7	B_6	C_5, C_4, C_3	D_2
T_7	A_8	B_7	C_6, C_5, C_4	D_3

(c)

#3	A	B	C	D
	SPU3	SPU2	SPU1	SPU4
T_5	A_6	B_5	C_4, C_3, C_2	D_1
T_6	A_7	B_6	C_5, C_4, C_3	D_2
T_7	A_8	B_7	C_6, C_5, C_4	D_3

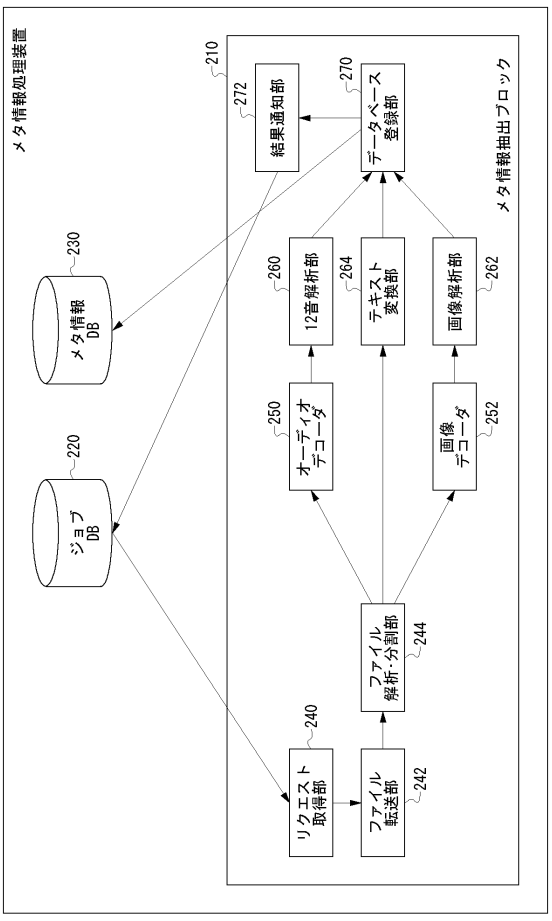
(d)

#4	A	B	C	D
	SPU1	SPU3	SPU2	SPU4
T_5	A_6	B_5	C_4, C_3, C_2	D_1
T_6	A_7	B_6	C_5, C_4, C_3	D_2
T_7	A_8	B_7	C_6, C_5, C_4	D_3

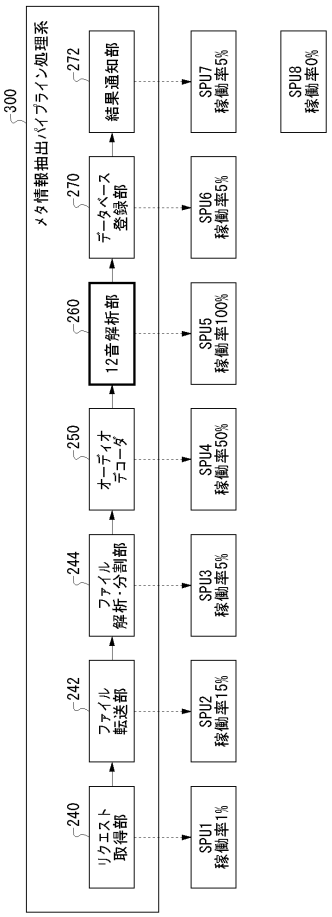
(e)

	SPU1	SPU2	SPU3	SPU4
負荷比率	1	1	1	1

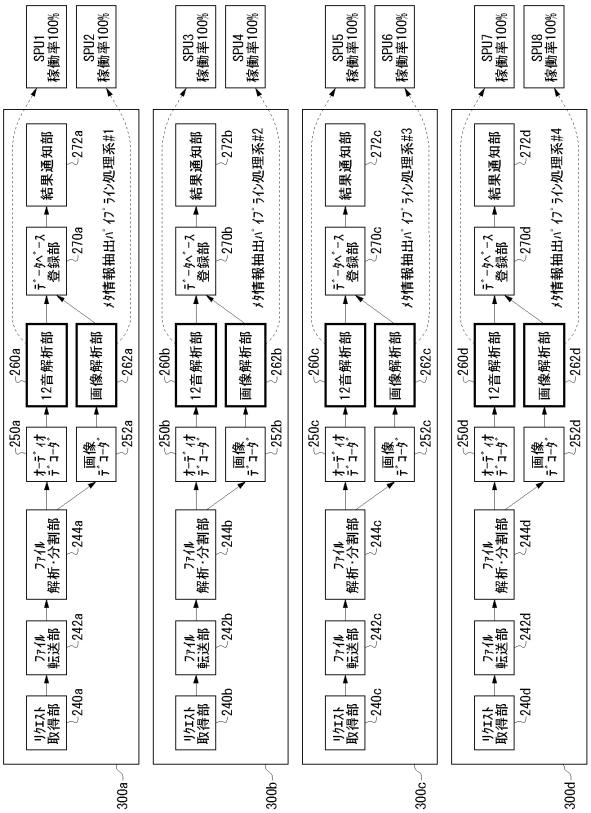
【図８】



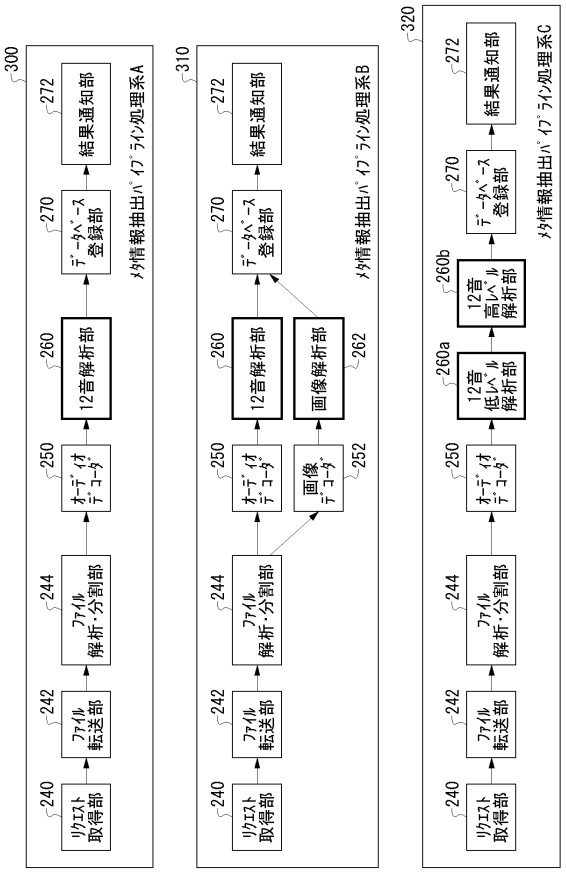
【図９】



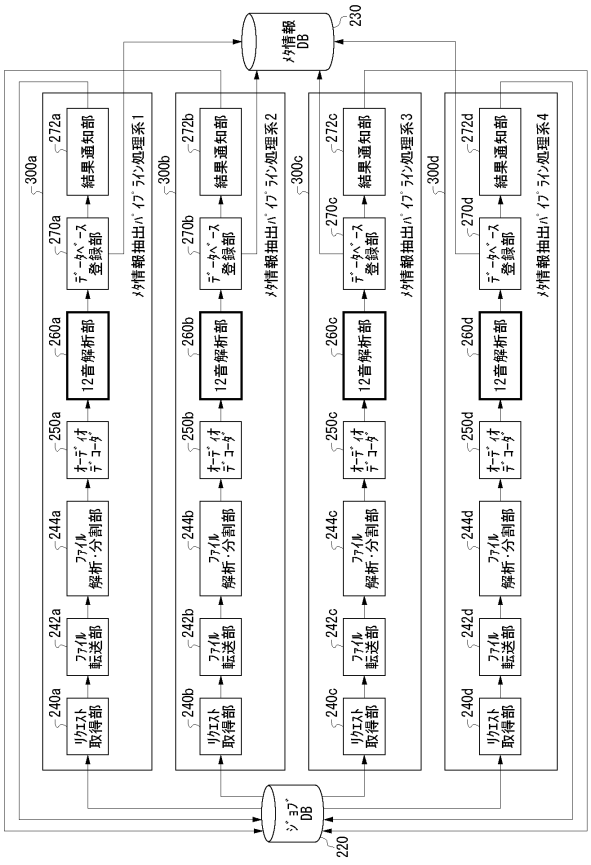
【図１０】



【図１１】



【図１２】



フロントページの続き

(72)発明者　斉藤　勝
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内
(72)発明者　赤羽　誠
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内
(72)発明者　鈴木　章
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内
(72)発明者　池田　望
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内
(72)発明者　高橋　良和
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内
(72)発明者　森　晴
東京都港区南青山２丁目６番２１号　株式会社ソニー・コンピュータエンタテインメント内

審査官　北元　健太

(56)参考文献　特開平７－８４９６７（ＪＰ，Ａ）
特開２００４－１９９６７４（ＪＰ，Ａ）
特開平１０－１１６３３８（ＪＰ，Ａ）
特開２００６－１０６８５９（ＪＰ，Ａ）
特開２０００－３５３０９９（ＪＰ，Ａ）
特開平６－１２３９２（ＪＰ，Ａ）
特表２００４－５０９３８６（ＪＰ，Ａ）
特開２００６－９９５７９（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／４６　－　９／５４