

(19) 日本国特許庁(JP)

(12) 公開特許公報(A)

(11) 特許出願公開番号

特開2012-108576

(P2012-108576A)

(43) 公開日 平成24年6月7日(2012.6.7)

(51) Int.Cl.		F I			テーマコード (参考)
G 0 6 F	9/48	(2006.01)	G 0 6 F	9/46	4 5 2 H
G 0 6 F	9/50	(2006.01)	G 0 6 F	9/46	4 6 5 Z

審査請求 未請求 請求項の数 14 O L (全 17 頁)

(21) 出願番号	特願2010-254662 (P2010-254662)	(71) 出願人	000003207
(22) 出願日	平成22年11月15日 (2010.11.15)		トヨタ自動車株式会社
		(74) 代理人	100070150
			弁理士 伊東 忠彦
		(72) 発明者	安藤 栄祐
			愛知県豊田市トヨタ町1番地 トヨタ自動車株式会社内

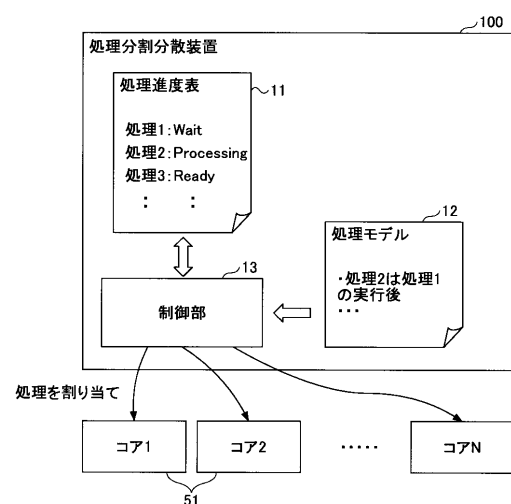
(54) 【発明の名称】 マルチコアプロセッサ、処理実行方法、プログラム

(57) 【要約】

【課題】 既存のプログラムの変更によるコスト増を抑制し、既存のプログラムをマルチコアで実行しても適切に負荷分散できるマルチコアプロセッサを提供すること。

【解決手段】 複数のコア51と、アプリケーション22が記憶された記憶手段57と、を有するマルチコアプロセッサ200であって、アプリケーションを複数の処理に区分した際の処理間の依存関係を登録した処理依存関係情報12と、各処理の進捗状況が登録された進捗状況登録テーブル11と、進捗状況が実行待ちの処理と処理依存関係情報において依存関係がある処理の実行完了が前記進捗状況登録テーブルに登録されている場合、実行待ちの処理の前記進捗状況を実行可能に更新する進捗状況更新手段31と、進捗状況に実行可能と設定された処理を空きコアに割り当てる処理割り当て手段32と、を有する。

【選択図】 図2



【特許請求の範囲】**【請求項 1】**

複数のコアと、アプリケーションが記憶された記憶手段と、を有するマルチコアプロセッサであって、

前記アプリケーションを複数の処理に区分した際の処理間の依存関係を登録した処理依存関係情報と、

各処理の進捗状況が登録された進捗状況登録テーブルと、

前記進捗状況が実行待ちの処理と前記処理依存関係情報において依存関係がある処理の実行完了が前記進捗状況登録テーブルに登録されている場合、実行待ちの処理の前記進捗状況を実行可能に更新する進捗状況更新手段と、

10

前記進捗状況に実行可能と設定された処理を空きコアに割り当てる処理割り当て手段と、を有するマルチコアプロセッサ。

【請求項 2】

各処理の処理終了アドレスが登録された処理終了アドレス情報と、

コアが前記処理終了アドレスの命令を実行した場合に、処理の終了を検出する終了判定手段と、を有し、

前記進捗状況更新手段は、前記終了判定手段が終了を検出した処理の前記進捗状況を実行完了に更新する、請求項 1 記載のマルチコアプロセッサ。

【請求項 3】

各処理の処理開始アドレスが登録された処理開始アドレス情報、を有し、

20

前記処理割り当て手段は、処理開始アドレスを割り当て対象のコアのプログラムカウンタに設定することで、処理を空きコアに割り当てる、

請求項 1 又は 2 記載のマルチコアプロセッサ。

【請求項 4】

前記進捗状況登録テーブルには、複数のコアで並行に実行可能であり、かつ、前記アプリケーションの起動直後に実行される処理にエントリ情報が登録されている、

請求項 1 ～ 3 いずれか 1 項記載のマルチコアプロセッサ。

【請求項 5】

前記進捗状況更新手段は、全ての処理の前記進捗状況が実行完了に設定された場合、前記エントリ情報が登録された処理の前記進捗状況を実行可能に更新する、

30

請求項 4 記載のマルチコアプロセッサ。

【請求項 6】

前記進捗状況更新手段は、全ての処理の前記進捗状況が実行完了に設定される前であって、前記進捗状況が実行中又は実行完了の処理のみになった場合、

前記エントリ情報が登録された処理の前記進捗状況を実行可能に更新する、

請求項 4 記載のマルチコアプロセッサ。

【請求項 7】

前記依存関係は、後の処理が前の処理のデータを使用するデータ依存、又は、後の処理が前の処理の処理結果を前提とする処理依存、である、

請求項 1 ～ 6 いずれか 1 項記載のマルチコアプロセッサ。

40

【請求項 8】

前記処理割り当て手段は、複数の処理を異なるコアに割り当て、各コアが時間的に並行にそれぞれ処理を実行する、

請求項 7 記載のマルチコアプロセッサ。

【請求項 9】

前記進捗状況更新手段は、前記処理割り当て手段により処理が割り当てられたコアが該処理を実行している間に、該処理の前記進捗状況を更新する、

請求項 8 記載のマルチコアプロセッサ。

【請求項 10】

前記進捗状況更新手段は、1 つの処理の実行完了毎に、該処理を実行していたコアに割

50

り当てられることで実現され、該処理の前記進捗状況を更新する、
請求項 8 記載のマルチコアプロセッサ。

【請求項 1 1】

前記進捗状況は、コア毎に、実行待ち、実行可能、実行中、実行完了の順に遷移する、
請求項 1 ～ 1 0 いずれか 1 項記載のマルチコアプロセッサ。

【請求項 1 2】

前記記憶手段にはアプリケーションの他にプログラムが記憶されており、
前記アプリケーションと前記プログラムの優先順位に応じて、前記アプリケーションと
前記プログラムを動的に前記コアに割り当てるスケジュール手段を有する、
請求項 1 ～ 1 1 いずれか 1 項記載のマルチコアプロセッサ。

10

【請求項 1 3】

複数のコアと、アプリケーションが記憶された記憶手段と、
前記アプリケーションを複数の処理に区分した際の処理間の依存関係を登録した処理依
存関係情報と、

各処理の進捗状況が登録された進捗状況登録テーブルと、を有するマルチコアプロセッ
サの処理実行方法であって、

進捗状況更新手段が、前記進捗状況が実行待ちの処理と前記処理依存関係情報において
依存関係がある処理の実行完了が前記進捗状況登録テーブルに登録されている場合、実行
待ちの処理の前記進捗状況を実行可能に更新するステップと、

処理割り当て手段が、前記進捗状況に実行可能と設定された処理を空きコアに割り当て
るステップと、を有する処理実行方法。

20

【請求項 1 4】

複数のコアと、アプリケーションが記憶された記憶手段と、
前記アプリケーションを複数の処理に区分した際の処理間の依存関係を登録した処理依
存関係情報と、

各処理の進捗状況が登録された進捗状況登録テーブルと、を有するマルチコアプロセッ
サに、

前記進捗状況が実行待ちの処理と前記処理依存関係情報において依存関係がある処理の
実行完了が前記進捗状況登録テーブルに登録されている場合、実行待ちの処理の前記進
捗状況を実行可能に更新する進捗状況更新ステップと、

30

前記進捗状況に実行可能と設定された処理を空きコアに割り当てる処理割り当てステッ
プと、を実行させるプログラム。

【発明の詳細な説明】

【技術分野】

【0001】

本発明は、複数のコアが搭載されたマルチコアプロセッサに関し、特に、コアに処理を
割り当てることで負荷分散できるマルチコアプロセッサに関する。

【背景技術】

【0002】

車両には数多くの電子制御装置が搭載されており、各電子制御装置が各種のプログラム
を実行して様々な車載装置を制御している。電子制御装置は CPU を有しているが、電子
制御装置においても CPU のマルチコア化を図ることで処理効率を向上させようという試
みがある。

40

【0003】

マルチコアの CPU の処理効率の向上は、複数のタスクを各コアが並行に実行すること
を基本的な考え方にする。しかしながら、タスクには処理内容や重要度などに違いがあり
全てのタスクを公平にコアに割り当てることはできないため、効率よくタスクをコアに割
り当てる技術が考案されている（例えば、特許文献 1 参照。）。特許文献 1 には、複数個
のプロセッサユニットにタスク割当を行なう際、プログラムの実行時間や実行タスク間の
依存性を考慮するマルチプロセッサシステムが開示されている。

50

【0004】

また、タスク単位でなくタスクが分割されたスレッド単位でコアに割り当てる技術も考案されている（例えば、特許文献2参照。）。特許文献2には、サブルーチンやスレッドにタグを付け、スレッドの実行特徴を連続して習得してスレッド性能データを生成し、スレッド特性データに基づきコアにスレッドを割り当てるマルチプロセッサシステムが提供されている。

【先行技術文献】

【特許文献】

【0005】

【特許文献1】特開2009-069921号公報

10

【特許文献2】特開2009-151774号公報

【発明の概要】

【発明が解決しようとする課題】

【0006】

ところで、車両ではすでに電子制御装置が多くプログラムを実行しており、この既存のプログラムを変更することなく、マルチコアに適用することでコスト増を抑制することが検討されている。

【0007】

図1は、従来のシングルコア用のプログラムを模式的に説明する図の一例である。このプログラムの特にメインタスクはシングルコア（図のCPU）用が開発されているため、最初から最後まで逐次実行される。また、CPUに割込みが生じると、CPUは例えば割込みタスクを実行する。

20

【0008】

図1（a）のようなプログラムを、図1（b）のようにマルチコアの各コアに割り当てることを考えると、例えばOS（Operating System）が、タスクの起床毎にメインタスクと割込みタスクを各コアに割り当てることになる。したがって、図1（a）と図1（b）を比較すれば、コア1の処理負荷は割込みタスクを実行する必要がない分だけ低減されたことになる。

【0009】

しかしながら、一般に、メインタスクと割込みタスクではメインタスクの方が処理負荷が大きく、メインタスクと割込みタスクを分散しただけでは、マルチコアによる処理負荷の低減・処理の高効率化の恩恵を十分には受けられない。

30

【0010】

そこで、特許文献2に開示されているように、タスクをスレッドなどのより小さい粒度に分割して、各コアに割り当てることが考えられる。しかし、シングルコア用のプログラム（この場合はメインタスク）を分割することは容易ではなく、特許文献2にもスレッドへの分割方法は開示されていない。コンパイルされたプログラムには、例えばC言語の関数やサブルーチンに対応するラベルが記述されているので、ラベル毎にスレッドに分割することが考えられる。しかし、ラベル毎にスレッド化した場合、1つのスレッドで処理が完結しているという保証がなく、スレッド単位の取り扱いが不具合を引き起こすおそれがある。また、スレッドをあまり小さい粒度にするとコアへの割り当てのオーバーヘッドが目立つようになるため、ラベル毎に一律にスレッド化することは適切でない。

40

【0011】

また、処理単位の適切な粒度にスレッド化できたとしても、スレッド間の依存関係を考慮せずに、OS等が単にスレッドを空いたコアに割り当てるだけでは、プログラムが正常に動作する保証がない。

【0012】

本発明は、上記課題に鑑み、既存のプログラムの変更によるコスト増を抑制し、既存のプログラムをマルチコアで実行しても適切に負荷分散できるマルチコアプロセッサを提供することを目的とする。

50

【課題を解決するための手段】

【0013】

本発明は、複数のコアと、アプリケーションが記憶された記憶手段と、を有するマルチコアプロセッサであって、前記アプリケーションを複数の処理に区分した際の処理間の依存関係を登録した処理依存関係情報と、各処理の進捗状況が登録された進捗状況登録テーブルと、前記進捗状況が実行待ちの処理と前記処理依存関係情報において依存関係がある処理の実行完了が前記進捗状況登録テーブルに登録されている場合、実行待ちの処理の前記進捗状況を実行可能に更新する進捗状況更新手段と、前記進捗状況に実行可能と設定された処理を空きコアに割り当てる処理割り当て手段と、を有することを特徴とする。

【発明の効果】

10

【0014】

既存のプログラムの変更によるコスト増を抑制し、既存のプログラムをマルチコアで実行しても適切に負荷分散できるマルチコアプロセッサを提供することができる。

【図面の簡単な説明】

【0015】

【図1】従来のシングルコア用のプログラムを模式的に説明する図の一例である。

【図2】マルチコアプロセッサの概略的な特徴部を示す図の一例である。

【図3】マルチコアプロセッサのハードウェア構成図の一例である。

【図4】マルチコアプロセッサの実装例を示す図である。

【図5】処理分割分散装置の機能ブロック図の一例である。

20

【図6】処理進捗表の一例を示す図である。

【図7】処理モデルの一例を示す図である。

【図8】処理進捗表更新部が処理進捗表を更新する手順を示すフローチャート図の一例である。

【図9】処理割り当て部がコアに処理1～5を割り当てる手順を示すフローチャート図の一例である。

【図10】S140, S150の処理のフローチャート図の一例である。

【図11】コア1, 2が処理1～5を実行するタイムチャート図の一例である。

【図12】コア1, 2に対する処理1～5と割込みタスクの割り当ての関係を示す図の一例である。

30

【図13】処理分割分散装置の機能ブロック図の一例である。

【図14】コア1, 2が処理1～5を実行するタイムチャート図の一例である。

【発明を実施するための形態】

【0016】

以下、本発明を実施するための形態について図面を参照しながら説明する。

図2は、本実施形態のマルチコアプロセッサの概略的な特徴部を示す図の一例である。まず、マルチコアプロセッサには新たに処理分割分散装置100が配置される。処理分割分散装置100は処理進捗表11と処理モデル12を有する。

・処理進捗表11には、プログラムのメインタスクから抽出された複数の処理のステータス（Wait、Ready、Processing、Done）が登録されている。

40

・処理モデル12には、処理間の依存関係が登録されている。

【0017】

制御部13は、処理進捗表11のステータスを監視し、Waitの処理があれば処理モデル12を参照して、Waitの処理が実行可能な状態になったか否かを判定する。制御部13は、実行可能になった処理のステータスをReadyに更新することで、コア1又はコア2が該処理を実行することができる。

【0018】

処理進捗表11には、既存のプログラムのメインタスクから適切な粒度の処理が抽出され、処理の先頭アドレス（後述の処理開始アドレス）と終了アドレス（後述の処理終了アドレス）が登録されている。制御部はこれらのアドレスを使用して処理を切り替えること

50

で、既存のプログラムについてはほとんど変更することなく、マルチコアに負荷分散することが可能となる。このため、既存のプログラムを分割してマルチタスク用に再開発するのに比べて、小さい労力でマルチコア化に対応することができる。

【0019】

〔構成〕

図3は、マルチコアプロセッサ200のハードウェア構成図の一例を示す。マルチコアプロセッサ200は、マルチレイヤーバス61を介して接続されたCPU50、DMAC54、SDRAM55、I/Oブリッジ56及びROM57を有する。

【0020】

CPU50は2つの集積回路を有し、各集積回路が、コア51、INT52、及び、RAMを53有する（これらを区別する場合、コア1、コア2のように番号を付す）。コア1, 2は、それぞれ、IFU (Instruction Fetch Unit)、DEC (DECoder)、RF (Register Fetch)、REG (REGister)、LSU (Load Store Unit)、SH (Shifter)、ALU、MUL及びFPU等の演算回路を有し、パイプライン制御により1クロックで1命令を実行することができる。

【0021】

RAM1はコア1に専用のメモリ（一次キャッシュ）であり、RAM2はコア2に専用のメモリ（一次キャッシュ）である。INT1とINT2は、コア間割り込みを制御すると共に、コア1からコア2又はコア2からコア1に、パラメータ等の送受信を制御する。また、INT1, INT2は、RAM1、RAM2、又は、SDRAM55を共有の通信用メモリにしてパラメータ等を送受信することもできる。INT1とINT2を使用したコア間のパラメータ通信はハードウェア的な実装であり、この他、OSが提供するコア間通信のAPIを利用してもコア1とコア2は通信することができる。

【0022】

DMAC54は、コア1及びコア2から要求を受けてプログラム60をSDRAM55、RAM1、RAM2に読み込んだり、周辺機器からのメモリアクセス要求を調停すると共に、I/Oブリッジ56からSDRAM55へ、又は、SDRAM55からI/Oブリッジ56へ、コア1、コア2を介することなくデータを転送する。SDRAM55は、コア1及びコア2がプログラム60を実行している間、RAM1又はRAM2にミスヒットするとプログラム60又はデータを提供するためのメモリ（二次キャッシュ）である。SDRAM55の少なくとも一部はコア1とコア2により共有される。

【0023】

ROM57にはプログラム60及びタスク制御プログラム70が記憶されている。プログラム60は、少なくともメインタスクと割り込みタスクが含まれている。タスク制御プログラム70は、処理分割分散装置100を制御するソフトウェアであるが、処理分割分散装置100がハード的に実装される場合には不要となる場合がある。なお、ROM57には、OSやデバイスドライバ（プラットフォーム）も記憶されている。

【0024】

I/Oブリッジ56は、マルチレイヤーバス61とI/Oバス62の間で相互に周波数や電圧値などを変換する。I/Oバス62には種々のI/O58が接続されている。I/O58はプロセッサ100と外部の機器を接続するインタフェースであり、例えば、CANコントローラ、各種のアクチュエータ、センサ、スイッチ等が接続される。

【0025】

ここで、マルチコアの態様として、それぞれのコアに決まったタスクを割り当てるAMP (Asymmetric Multi-processing)、例えばOSがそれぞれのコアにタスクを動的に割り当てるSMP (Symmetric Multi-processing)、又は、一部のコアにタスクを固定的に割り当て、残りのコアにタスクを動的に割り当てるBMP (Bound Multi-processing) が知られている。本実施形態では、処理分割分散装置100が複数のコアに処理を動的に割り当てるのでSMPのマルチコアプロセッサ200とすることが自然である。だが、いくつかのコア51にだけ処理を割り当てると決めてそのコアの中だけで処理分割分散

装置 100 が処理を動的に割り当てることもできる。したがって、SMP、AMP 又は B
MP のいずれかの実装態様でなければならないということはない。

【0026】

なお、SMP のマルチコアプロセッサ 200 の場合、複数の処理の処理結果をコア間で共有できるように、不図示のスヌープコントローラ (SNC) が RAM1 と RAM2 のデータの更新を監視して、更新データを RAM1 と RAM2 間で通信する。これにより RAM1 と RAM2 のキャッシュのコヒーレンスを維持することができる。また、コア間で 1 つの処理を再割り当てすることも容易になる。

【0027】

図 4 は、マルチコアプロセッサ 200 の実装例を示す図である。アプリケーション 22
は、メインタスクや割り込みタスクのプログラム 60、及び、別の既存のプログラムのメインタスク (区別するため他のタスクという) である。なお、別の既存のプログラムは、処理に分割されずに実行されるものとする。また、割り込みタスクは 2 つ以上あってもよい。

【0028】

PF/OS 21 はプラットフォームとオペレーティングシステムを意味するが両者が独立して存在してもそうでなくてもよく、図示する PF/OS 21 の意義は処理分割分散装置 100 と略一体になって、コア 51 にアプリケーション 22 の処理を割り当てる点にある。処理分割分散装置 100 は複数の処理を各コア 1 ~ N に割り当てるが、この機能は OS のスケジューリングと同系統の機能となる。一方、メインタスクは複数の処理に分割されたので、OS によるスケジューリングは困難になっている。

【0029】

メインタスクを変更せずに OS によるスケジューリングを可能とするには、メインタスクが複数の処理に分割されていることを OS から遮蔽することが好ましい。このため、処理分割分散装置 100 は、PF/OS 21 がいずれかのコア 51 にメインタスクを割り当てるというディスパッチ処理をコア 51 の代わりに受け取り、各処理の実行を制御する。このように、処理分割分散装置 100 は PF/OS 21 に含まれる形態で実装されることがある。なお、上記のように処理分割分散装置 100 をハード的な IC で実装することもできるが、この場合は OS のメインタスクに対するディスパッチ処理をコア 51 の代わりに、処理分割分散装置 100 が取得して、各処理の実行の制御を開始する。

【0030】

図 5 は、処理分割分散装置 100 の機能ブロック図の一例を示す。処理分割分散装置 100 をソフト的に実装する場合、図 5 の処理分割分散装置 100 はいずれかのコア 51 により実行される。また、図 5 では処理分割分散装置 100 と PF/OS 21 を一体に図示した。

【0031】

マルチタスクをサポートする OS では、タスクの優先順位と状態に応じて、実行可能なタスクから順に実行キュー 23 に登録する。タスクの状態には、例えば、実行可能、実行中、実行待ちの状態がある。OS にとってタスクの優先順位は予め既知である。OS は、実行キュー 23の中から最も優先順位の高いタスクをコア 51 に割り当てる。

【0032】

また、プリエンプション (実行中のタスクの中断と再開) が可能な OS の場合、OS は実行キュー 23の中で最も優先順位の高いタスクと、各コア 51 が実行中のタスクの優先順位を比較して、実行キュー 23で最も優先順位の高いタスクが、各コア 51 が実行中のいずれかのタスクの優先順位よりも高くなると、そのコア 51 に実行キュー 23で最も優先順位の高いタスクを割り当てる。

【0033】

ここで、コア 51 が実行しているタスクを切り替える必要が生じるのは、タスク自身がタスク切り替えを引き起こすサービスコールを OS に要求した場合や、割り込みが起こり割り込み処理を実行していたコア 51 がサービスコールを OS に要求した場合などである。前者の場合は、自身を起床待ちにする場合や、優先度の高いタスクを呼び出す場合に生

10

20

30

40

50

じる。

【0034】

なお、メインタスクは既存のプログラムから変更がないので、メインタスクの各処理がサービスコールをOSに要求することはない。また、メインタスクの優先順位が最も高いのでメインタスクの各処理を実行しているコア51の全てが、例えば割込みタスクに切り替えられることは生じない。

【0035】

本実施形態では、メインタスクの優先順位が最も高いが、割込みが生じ割込みタスクが起床された場合は、割込みタスクの優先順位の方が高くなるものとする。

【0036】

ところで、OSから見てメインタスクとその他のタスクは優先順位の違いはあってもそれぞれは1つのタスクなので、発明の課題で説明したように、メインタスクに割り当てられるコア51は1つである。しかし、マルチコアプロセッサ200にメインタスク以外のタスクが同時に生じられる数よりも2～3個多いコアを用意しておけば、メインタスクに複数のコア51が割り当てられているのとみなすことができる。すなわちSMPでは、処理分割分散装置100がメインタスクの複数の処理を空いているコア（メインタスク以外のタスクを実行していないコア）51に割り当て並行に実行できる。

【0037】

また、マルチコアプロセッサ200の実装をAMPとした場合、メインタスクに専用の複数のコア51が割り当てられるので、この場合も処理分割分散装置100がメインタスクの複数の処理を空いているコア51に割り当て並行に実行できる。しかし、実行可能なコア51の数を可変にできるという点（スケーラビリティ）ではSMPの方がより好ましい。

【0038】

サービスコールが発生したり、プリエンプションが発生すると、OSはディスパッチ処理を開始する。本実施例では、OSがメインタスクを呼び出すディスパッチ処理を、制御部13が受け取る。

【0039】

また、OSがメインタスク以外のタスクを呼び出すディスパッチ処理を実行した場合は、従来と同様に、制御部13を介することなくコア51が実行するタスクの切り替えが行われる。制御部13を介してタスクの切り替えが行われてもよい。

【0040】

OSは、タスクをコア51に割り当てる際、まず、割り当て対象のコア51が実行しているタスクのコンテキスト（タスクの実行に必要な、汎用レジスタ、プログラムカウンタ、スタックポインタなど）を保存し、該コア51に割り当てるタスクのコンテキストを割り当て対象のコア51に設定する。

【0041】

一方、いずれかのコア51にメインタスクが割り当てられた場合、メインタスクが実行中になる。実行中の間、制御部13は、処理進捗表11の更新、各コア51への処理の割り当てを繰り返す。

【0042】

図6は処理進捗表11の一例を示す図である。処理進捗表11には、「エントリ」、「処理識別番号（1～5）」、「処理開始アドレス」、「処理終了アドレス」、及び、「ステータス」が登録される。「エントリ」にフラグ「○」が設定された処理は、メインタスクにおいて処理の順番が最初の処理であることを示す。「エントリ」をなくし、処理割り当て部32が処理進捗表11の最上位の処理からコア51に割り当てるとしてもよいが、この場合、処理1と並行に実行可能な処理（図では処理2）があっても、しばらくは実行されない状態が発生してしまう。また、後述するように「エントリ」があることで、メインタスクの全ての処理1～5が完了した後に、メインタスクを再開するための最初の処理が明らかになる。

10

20

30

40

50

【0043】

「処理識別番号」は処理分割分散装置100が処理を識別するための情報である。メインタスクから処理1～5をどのように分割するかは後述する。

【0044】

「処理開始アドレス」は処理1～5の先頭アドレスであり、「処理終了アドレス」は処理1～5の終了アドレスである。これらのアドレスは、ROM57に記憶された状態のプログラム60のアドレスである。図では処理終了アドレスは処理開始アドレスの直前になっているが、分岐先で終了する場合などはこのようにならない場合もあるし、また、条件分岐した場合に対応するため複数の処理終了アドレスが登録される場合もある。

【0045】

「ステータス」は、各処理の状態(Ready、Processing、Wait、Done)を表す。Readyは処理に必要なデータや事象が全て揃っていて、コア51が割り当てられればいつでも実行可能な状態を、Processingはコア51が実行している状態を、Waitは他の処理による演算結果、入出力終了などの事象の終了を待っている状態を、Doneは処理が完了した状態を、それぞれ意味する。なお、処理のステータスは、OSが監視するタスクの状態とは別のものである。

【0046】

図5の処理割り当て部32は、処理進捗表11のステータスが“Ready”の処理を空いているコア51に割り当てる。具体的には、処理進捗表11に登録された処理開始アドレスを、空いているコア51のPC(プログラムカウンタ)に設定する。割り当て対象の空いているコア51がそれまで別の処理を実行していた場合、処理割り当て部32はコンテキストを保存したり、複数の処理の処理結果を統合するなど、必要な処理を行う。

【0047】

また、終了判定部33は、処理進捗表11のステータスが“Processing”の処理を実行しているコア51のPCを監視して、処理終了アドレスに到達したか否かを判定する。これは、メインタスクは既存のプログラムから変更がないため、各コア51が処理の終了を判定することが困難なためである。このため、外部の終了判定部33が終了を判定する。

【0048】

処理進捗表更新部31は、処理進捗表11のステータスを操作する手段であり、特に、Waitの処理をReadyに更新する処理を行う。

図7は、処理モデル12の一例を示す。処理モデル12は並行に実行可能な処理、及び、処理間の依存関係を示す情報を含む。図7では、処理1～3は並列に処理が可能であることを示し、処理4は処理1及び処理2に対し依存関係を有し、処理5は処理3及び処理4に依存関係を有することをそれぞれ示している。依存関係とは、例えば、処理4は処理1及び2のいずれも完了しないと実行できないことを言う。

【0049】

なお、図6の処理進捗表11のステータスは、メインタスクが2つのコア51を使用している状態で、処理1と処理2が並行に実行され、その後、処理1が先に終了されたことで、処理3がReadyになった状態を示す。

【0050】

処理進捗表11の処理1～5は、この処理モデル12を開発者が抽出する際に区分した一連の(ひとまとまりの)処理となる。処理の抽出について説明する。シングルタスクのプログラムはコードの上から順番に実行される。関数型のプログラムではmain関数から定義された関数j(j:1～m)やマクロi(i:1～n)が順番に呼び出されるようにして実行される。開発者は、関数jやマクロiで依存関係のある箇所を特定する。依存関係には、次のようにデータ依存と処理依存がある。

・データ依存

後の関数jやマクロiがそれよりも前の関数jやマクロiが使用する変数を使用する場合である。例えば、後の関数がそれよりも前の関数から「引数」を受け渡される場合や、後の関数と前の関数が共通のグローバル関数を使うような場合である。より具体的な例では、

10

20

30

40

50

前の関数がクランク角を求める処理であり、後の関数がクランク角を用いて燃料噴射タイミングなどを決定する処理の場合、2つの処理にデータ依存関係があると言える。また、例えば、前の関数が圧縮されたデータを伸張する処理であり、後の関数がデータを表示したり出力する処理の場合、2つの処理にデータ依存関係があると言える。

・ 処理依存

後の関数jやマクロiの実行に、それよりも前の関数jやマクロiが実行した状況が必要となる場合である。例えば、後の関数がアクチュエータを元の状態に戻す処理であり、前提として前の関数がアクチュエータをある状態にする処理である場合、2つの処理に処理依存関係があると言える。また例えば、後の関数が入出力装置にデータを出力する処理であり、前の関数が入出力装置を初期化する処理である場合、2つの処理に処理依存関係があると言える。

10

【0051】

開発者は、このような依存関係を抽出し、メインタスクが使用可能なコア数を考慮して、適当な数の処理に区分する。例えば、メインタスクが使用可能なコア数が少ない（例えば2つしかない）のに、それよりもかなり多い数（例えば、50～100）に処理を区分すると並列に実行可能な処理は増えるが、処理の切り替えのオーバーヘッドが処理効率を下げてしまう。一方、メインタスクが使用可能なコア数が十分にある（例えば10個ある）のに、区分後の処理の数が少ないと（例えば、10以下）、並列に実行可能な処理が少なくなり、マルチコアの有効性を活かせない。このため、開発者はある程度最適化しながら処理を適切な数に区分する。

20

【0052】

〔動作手順〕

図8は、処理進度表更新部31が処理進度表11を更新する手順を示すフローチャート図の一例である。図8の手順は、マルチコアプロセッサ200が起動すると繰り返し実行される。なお、以下では、コア数は2つ（処理分散分割装置100を入れると3つ）であるとして説明する。

【0053】

起動直後の処理進度表更新部31は処理進度表11を初期化する。例えば、RAM53に処理進度表11を生成し初期状態を設定する。「エントリ」「処理識別番号」「処理開始アドレス」「処理終了アドレス」については固定なので、処理進度表更新部31は既知のこれらの情報をRAM53に書き込む。

30

【0054】

また、起動直後の各処理のステータスは、「エントリ」にフラグが設定された処理1, 2が“Ready”でなければならないので（そうしないとどの処理も開始されない）、処理進度表更新部31は処理1, 2のステータスをReadyに更新する。その他の処理3～5のステータスは全てWaitに更新する。なお、処理モデル12により依存関係のないことが分かる処理3のステータスをReadyとしておいてもよい。

【0055】

処理進度表更新部31は、処理進度表11のステータスがWaitの処理があるか否かを判定する（S10）。判定のタイミングは、例えば、クロックサイクル毎でもよいし、終了判定部33が、コア51が実行している処理が完了したと判定したタイミングでもよい。

40

【0056】

ステータスがWaitの処理3, 4, 5がある場合（S10のYes）、処理進度表更新部31はステータスがWaitの処理3, 4, 5について1つずつ処理モデル12に基づき依存関係のある処理が全てDoneか否かを判定する（S20）。

【0057】

処理モデル12によれば、処理3に対し依存関係のある処理はなく、処理4に対し依存関係のある処理は処理1, 2なので、処理進度表更新部31は処理1, 2がDoneとなったか否かを判定し、処理5に対し依存関係のある処理は処理3, 4なので、処理進度表更新部31は処理3, 4がDoneとなったか否かを判定する。ステータスがWaitの処理に関し依

50

存関係のある処理が全てDoneになると、処理進度表更新部 3 1 はWaitであったステータスをReadyに更新する（S 3 0）。

【0 0 5 8】

ステップ S 1 0 に戻り、ステータスがWaitの処理がなくなると（S 1 0 の N o）、処理のステータスはReady、Done、又は、Processingということになるが、全てDoneになった場合に「エントリ」の処理 1、2 から再開する必要がある。

【0 0 5 9】

このため、処理進度表更新部 3 1 は処理進度表 1 1 の全てのステータスがDoneになったか否かを判定する（S 4 0）。全てのステータスがDoneになっていない場合（S 4 0 の N o）、手順は S 1 0 に戻る。

【0 0 6 0】

全てのステータスがDoneになった場合（S 4 0 の Y e s）、処理進度表更新部 3 1 は「エントリ」にフラグが設定された処理 1、2 にReadyを設定する（S 5 0）。メインタスクをシングルコアのプロセッサで実行する場合と同様に、本実施形態のマルチコアプロセッサ 2 0 0 はメインタスクを繰り返し実行することができる。

【0 0 6 1】

なお、ステップ S 4 0 で全てのステータスがDoneになるまで待つのでなく、ステータスが D o n e とProcessingだけになったら、「エントリ」にフラグが設定された処理 1、2 にReadyを設定することもできる。こうすることで、例えば、コア 1 が処理 5 を実行中（Processing）、処理 1、2 のステータスがReadyになるので、いずれか 1 つ以上のコア 2 が空き次第、そのコア 2 が処理 1 又は処理 2 を実行できるようになる。したがって、処理の並列度をさらに向上させることができる。

【0 0 6 2】

図 9 は、処理割り当て部 3 2 がコア 5 1 に処理 1 ～ 5 を割り当てる手順を示すフローチャート図の一例である。図 9 の手順は、マルチコアプロセッサ 2 0 0 が起動すると繰り返し実行される。

【0 0 6 3】

まず、処理割り当て部 3 2 は、「エントリ」にフラグが設定された処理 1、2 のステータスがReadyか否かを判定する（S 1 1 0）。こうすることで、マルチコアプロセッサ 2 0 0 はメインタスクで最初に行うべき処理から実行を開始できる。処理 1、2 のステータスがReadyでない場合（S 1 1 0 の N o）、処理進度表 1 1 の初期化が完了していないとして待機する。

【0 0 6 4】

処理 1、2 のステータスがReadyである場合（S 1 1 0 の Y e s）、処理割り当て部 3 2 は処理進度表 1 1 の処理 1、2 の処理開始アドレスを読み出す（S 1 4 0）。そして、処理割り当て部 3 2 は、空いているコア 1、2 で処理 1、2 を実行する（S 1 5 0）。すなわち、処理 1 の処理開始アドレスをコア 1 の P C に、処理 2 の処理開始アドレスをコア 2 の P C に、それぞれ設定する。

【0 0 6 5】

図 1 0 は、S 1 4 0、S 1 5 0 の処理のフローチャート図の一例を示す。処理割り当て部 3 2 は、空いているコア 1、2 に割り当てた処理 1、2 の識別情報を処理進度表更新部 3 1 に通知するので、処理進度表更新部 3 1 は処理 1、2 のステータスをProcessingに更新する（S 2 1 0）。

【0 0 6 6】

その後、コア 1、2 は処理 1、2 を実行する（S 2 2 0）。コア 1、2 が処理 1、2 を実行している間、終了判定部 3 3 はコア 1、2 の P C を I N T 経由で監視したり、命令バッファを読み出すことで、コア 1、2 が処理終了アドレスまで処理を実行したことを検出する。終了判定部 3 3 は、コア 1、2 が処理終了アドレスまで処理を実行したことを検出すると、P C や命令バッファに処理を終了する命令（例えば、NOP、E N D 等）を設定するなどして、処理 1、2 を強制的に終了させる。

【0067】

そして、終了判定部33は処理1, 2の識別情報と終了を処理進度表更新部31に通知する。これにより、処理進度表更新部31は処理1, 2のステータスをDoneに更新する(S230)。

【0068】

図11はコア1, 2が処理1~5を実行するタイムチャート図の一例である。初期状態の処理1, 2のステータスはReadyであり、ステップS210の手順により処理1, 2のステータスはProcessingになる。また、この後、処理1, 2が実行中に処理3のステータスはReadyとなる。その後、コア1が処理1の処理終了アドレスまで実行すると、処理1のステータスはDoneとなり、コア2が処理2の処理終了アドレスまで実行すると、処理2のステータスはDoneとなる。

10

【0069】

図9に戻り、処理割り当て部32は、定期的に又は処理進度表11が更新される毎に、ステータスがReadyの処理があるか否かを判定する(S130)。処理1, 2の実行中は処理3のステータスがReadyであるので、処理割り当て部32は空きコアがあるか否かを判定する(S130)。空きコアの有無は、例えば、OSに問い合わせたり、フラグを参照することで判断できる。

【0070】

例えば、処理1, 2の実行中は空きコアがないので手順はステップS120に戻る。そして、コア1が処理1の実行を完了すると、コア1が空きコアとなるので(S130のYes)、処理割り当て部32はコア1に処理3を割り当て、コア1が処理3を実行する(S140、S150)。

20

【0071】

これにより、処理進度表更新部31は処理3のステータスをProcessingに更新する(S210)。ここで、処理割り当て部32は、「エントリ」にフラグが設定された処理以外の処理を割り当てる際、コンテキスト管理を実行する。

【0072】

まず、処理割り当て部32は、それまでコア1が実行していた処理1のコンテキストを保存する。次に、処理割り当て部32は、処理モデル12を参照して、割り当て対象の処理がデータ依存の関係を有する処理を探す。処理3の場合は、データ依存の関係にある処理はないので、処理割り当て部32は何もしない。

30

【0073】

これに対し、例えば処理4が処理1と処理2に対しデータ依存の関係がある場合、処理割り当て部32は、保存してある処理1と処理2のコンテキストを読み出し、予め定められた処理を施して、コンテキストを復帰する。予め定められた処理は、処理1, 2に対する処理4のデータ依存関係、処理3, 4に対する処理5のデータ依存関係によって様々であるが、例えば、処理1, 2のコンテキストのうち処理4が使用するデータが格納されたレジスタのみ取り出す処理、状態を示すフラグを取り出す処理等である。なお、処理4が処理1, 2に対しデータ依存関係があっても、処理1, 2のコンテキストを互いに演算するような処理は処理4内で行われる。

40

【0074】

その後、コア1は処理3を実行するので(S220)、終了判定部33はコア1が処理終了アドレスまで処理を実行したことを検出する。終了判定部33は処理3の識別情報と終了を処理進度表更新部31に通知するので、処理進度表更新部31は処理3のステータスをDoneに更新する(S230)。

【0075】

図11に示すように、コア1が処理3の処理終了アドレスまで実行すると、処理3のステータスはDoneとなる。処理4, 処理5についても同様に割り当てられ実行される。

【0076】

すなわち、処理4のステータスは、処理2がDoneになるとReadyになり、コア2が処理

50

4の実行を完了するとDoneになる。すると、処理5のステータスがReadyになるので、コア1が処理5の実行を開始することができる。そして、コア1が処理5の実行を完了すると、処理1, 2のステータスがReadyになる。上記のように、処理5のステータスがProcessingになった時点で処理1, 2のステータスをReadyとすれば、空いているコア2が処理1を実行することができ、並列度が向上する。

【0077】

〔メインタスクと他のタスクとの競合について〕

コア51の数に対し、OSが割り当てるタスクの数がある程度多いと、メインタスクを実行していたコア51が処理1～5を途中で中断し、別のタスクを実行する必要も生じる。しかしこの場合でも、本実施形態のマルチコアプロセッサ200は有効である。

10

【0078】

図12(a)は、コア1, 2に対する処理1～5と割り込みタスクの割り当ての関係を示す図の一例である。処理分割分散装置100は、OSからの割り当てに関係なく、複数のコア1, 2に処理を分散できるので、2つしかないコア1, 2に処理1, 2を割り当てることができる。コア1が処理1をコア2が処理2を実行している状態で、OSが割り込みタスクを起床した場合、コア1又はコア2(図ではコア2に)に割り込みタスクをディスパッチする。これによりコア2は割り込みタスクを実行するが、コア1は処理1の実行後、処理2には依存関係のない処理3を実行できる。

【0079】

そして、コア2が割り込みタスクを終了したら、OSが処理2のコンテキストをコア2に復帰させるのでコア2は処理2を完了させることができる。なお、コア1が処理3を実行完了した後、処理2の実行が完了しないとコア1は処理4を実行できないが、換言すれば、処理3の後にはコア1が空きコアになり、メインタスク以外のタスクをコア1が実行できるようになる。したがって、メインタスクと他のタスクとのリソースを共有しても、不都合は生じない。

20

【0080】

なお、図12(a)ではコア2が処理2の完了後、コア1が処理4を実行しているが、2つのコア1, 2が空いている場合にどちらのコアが処理2を実行するか(OSがどちらのコアに割り当てるか)は設計できる。

【0081】

図12(b)は、コア1, 2に対する処理1～5と割り込みタスクの割り当ての関係を示す図の別の一例である。コア2が割り込みタスクを実行するところまでは図12(a)と同じであるが、図12(b)ではコア2が実行する割り込みタスクの実行時間が長い。しかしながら、このような場合も、処理分割分散装置100が処理2をコア1に再割り当てすることで、処理を進行させることができる。

30

【0082】

具体的には、処理分割分散装置100がOSの割り込みタスクの割り当てを監視し、OSが退避したコンテキストを取得することで、処理2をコア1に再割り当てすることができる。再割り当てが可能であれば、以降は、処理分割分散装置100が処理進捗表11に基づきReadyになった処理から順番にコア1に割り当てるので、シングルコアがメインタスクを実行する場合と同様に処理1～5を完了させることができる。

40

【0083】

〔処理分割分散装置を専用のコアに割り当てない場合〕

これまでは、固定のコアが処理分割分散装置100の機能を提供するとしたが、タスクを実行するコアが処理分割分散装置100の機能を提供することもできる。

図13は、処理分割分散装置100の機能ブロック図の一例を示す。図13において図5と同一部について説明を省略する。図13では機能ブロックの種類は図5と同じであるが、終了判定部33の機能が異なっている。

【0084】

まず、マルチコアプロセッサ200の起動時に処理進捗表更新部31が処理進捗表11

50

を初期化し、処理割り当て部 3 2 が処理 1, 2 を空きコアに割り当てるのは、上述したとおりである。そして、終了判定部 3 3 は、コア自身が処理 1 ~ 5 の終了を検出できるように、処理の直前に有効フラグと処理終了アドレスをコア 1, 2 に設定する。コア 1, 2 は、有効フラグがオンであることを条件に、PC が処理終了アドレスと一致すると、割込みを発生する。この割込みによって、コア 1, 2 は処理分割分散装置 1 0 0 として起動する。

【0085】

すると、終了判定部 3 3 は、終了した処理の識別番号を処理進捗表更新部 3 1 に通知するので、処理進捗表更新部 3 1 は処理進捗表 1 1 の終了した処理のステータスを Done に更新する。次に、処理進捗表更新部 3 1 は図 8 の手順を実行するので、処理依存のなくなった処理があればステータスに Ready が更新されることになる。

10

【0086】

処理割り当て部 3 2 は図 9, 1 0 の手順を実行するので、ステータスに Ready が更新された処理があれば、空きコアに処理を割り当てることができる。なお、ステータスに Ready が更新された処理がない場合でも、後に別のコアが割込みを発生したいずれかのタイミングで、必ずステータスに Ready が更新される処理が生じる。

【0087】

図 1 4 は、コア 1, 2 が処理 1 ~ 5 を実行するタイムチャート図の一例である。図 1 4 ではメッシュ部が、処理分割分散装置 1 0 0 がコアを占有したことを示し、丸数字は処理識別番号を示す。処理 1, 2 の実行前に、処理分割分散装置 1 0 0 は処理進捗表 1 1 を初期化すると共に処理 1, 2 のステータスを Ready から Processing に更新する。

20

【0088】

コア 1 が処理 1 の処理終了アドレスまで実行すると、割込みによりコア 1 が処理分割分散装置 1 0 0 として起動するので、処理進捗表更新部 3 1 は処理 1 のステータスを Done に、依存関係のない処理 3 のステータスを Ready に更新する。また、処理割り当て部 3 2 が処理 3 をコア 1 に割り当ててすることで、処理進捗表更新部 3 1 が処理 3 のステータスを Processing に更新する。

【0089】

コア 2 が処理 2 の処理終了アドレスまで実行すると、割込みによりコア 2 が処理分割分散装置 1 0 0 として起動するので、処理進捗表更新部 3 1 は処理 2 のステータスを Done に、依存関係のない処理 4 のステータスを Ready に更新する。また、処理割り当て部 3 2 が処理 4 をコア 2 に割り当ててすることで、処理進捗表更新部 3 1 が処理 4 のステータスを Processing に更新する。

30

【0090】

次に、コア 1 が処理 3 の処理終了アドレスまで実行すると、割込みによりコア 1 が処理分割分散装置 1 0 0 として起動するので、処理進捗表更新部 3 1 は処理 3 のステータスを Done に更新する。しかし、処理 3 が終了した時点では処理 4 が終了していないので、処理進捗表更新部 3 1 は処理 5 のステータスを Ready に更新できない。

【0091】

このため、次に、コア 2 が処理 4 の処理終了アドレスまで実行すると、割込みによりコア 2 が処理分割分散装置 1 0 0 として起動するので、処理進捗表更新部 3 1 は処理 4 のステータスを Done に、依存関係のない処理 5 のステータスを Ready に更新する。また、処理割り当て部 3 2 が処理 5 をコア 2 に割り当ててすることで、処理進捗表更新部 3 1 が処理 5 のステータスを Processing に更新する。

40

【0092】

なお、コア 2 が処理 5 の処理終了アドレスまで実行した場合、処理 1, 2 のステータスが Ready から Processing にそれぞれ更新される。

【0093】

したがって、専用のコアに処理分割分散装置 1 0 0 が割り当てられない場合も、処理 1 ~ 5 から処理分割分散装置 1 0 0 の切り替えへのオーバーヘッドが生じるものの、処理分

50

割分散装置 100 がメインタスクの処理を分散させることができる。

【0094】

以上説明したように、本実施形態のマルチコアプロセッサ 200 は、既存のプログラムから適切な粒度の処理と依存関係を抽出することで、既存のプログラムをそのままマルチコアに適用でき、1つのメインタスクをマルチコアに負荷分散することができる。

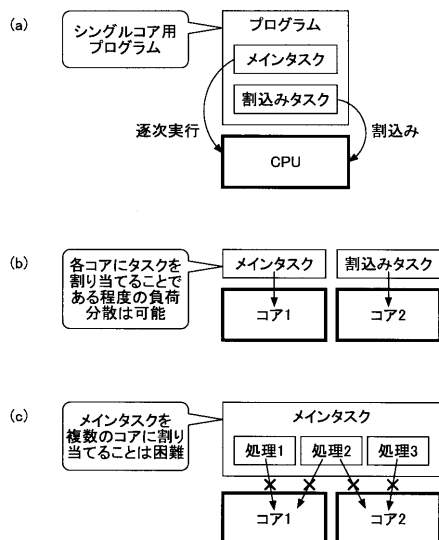
【符号の説明】

【0095】

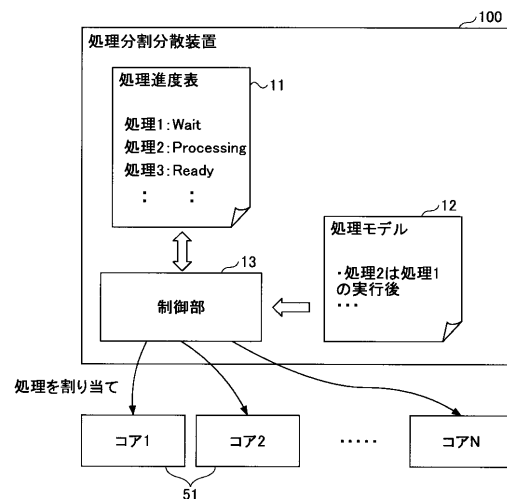
1 1	処理進捗表
1 2	処理モデル
2 1	P F / O S
2 2	アプリケーション
3 1	処理進捗表更新部
3 2	処理割り当て部
3 3	終了判定部
6 0	プログラム
7 0	タスク制御プログラム
1 0 0	処理分割分散装置
2 0 0	マルチコアプロセッサ

10

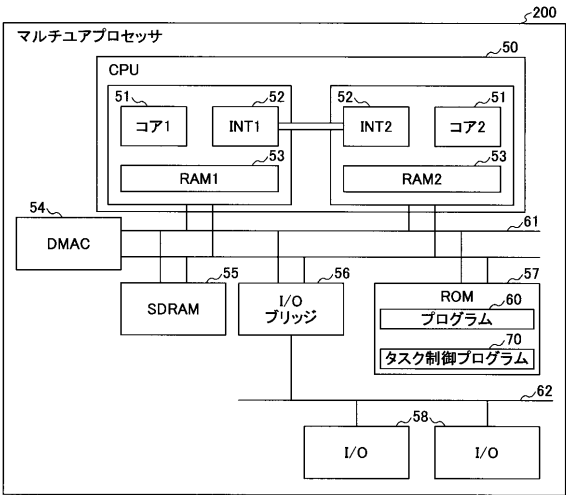
【図 1】



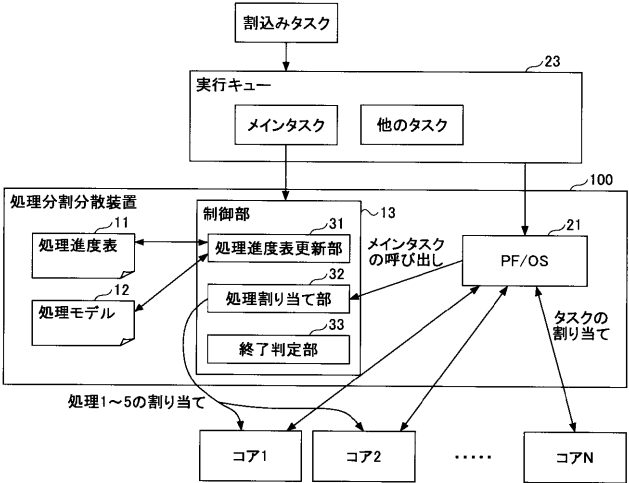
【図 2】



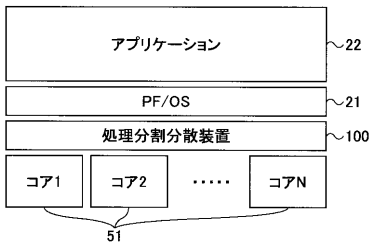
【図 3】



【図 5】



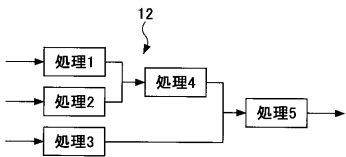
【図 4】



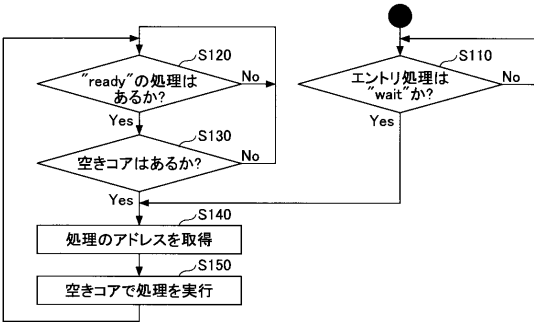
【図 6】

エン트리	処理識別番号	開始アドレス	終了アドレス	ステータス
○	1	0x1234	0x3233	Done
○	2	0x3234	0x3245	Processing
	3	0x3246	0x5677	Ready
	4	0x5678	0x7788	Wait
	5	0x7789	0x8910	Wait

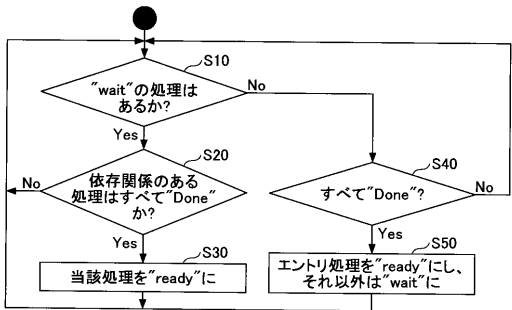
【図 7】



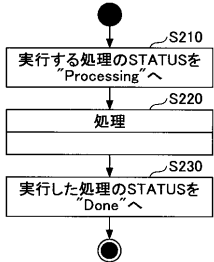
【図 9】



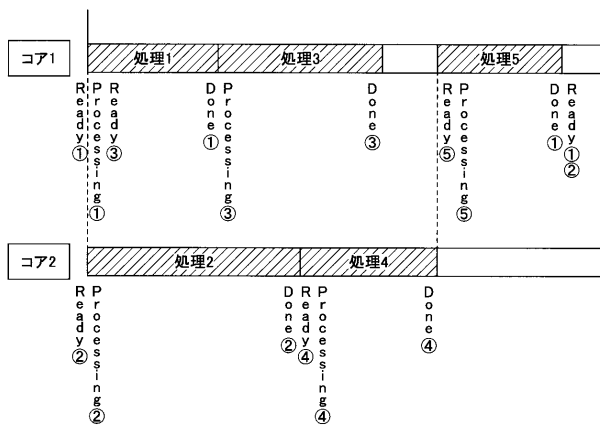
【図 8】



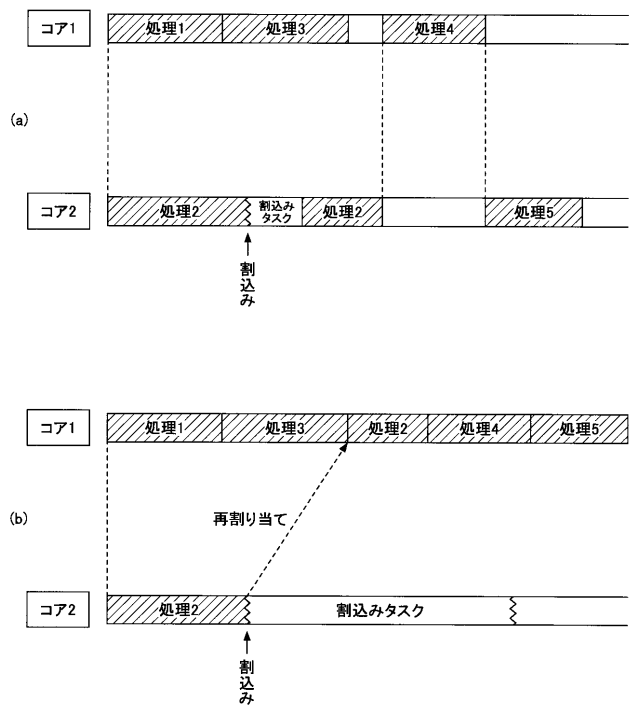
【図 10】



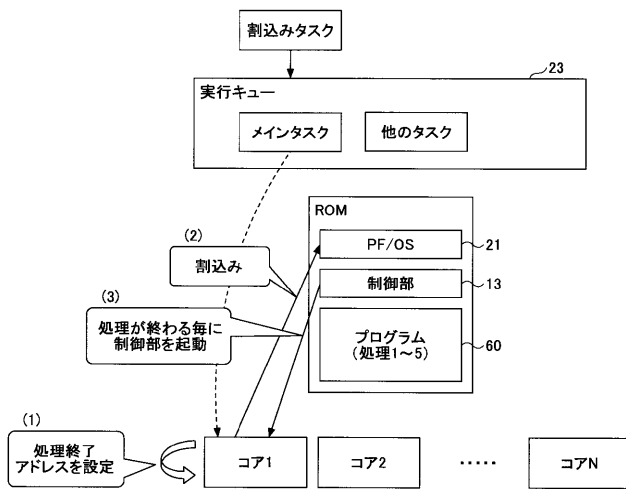
【図 1 1】



【図 1 2】



【図 1 3】



【図 1 4】

