(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号
特許第7425685号
(P7425685)

(45)発行日　令和6年1月31日(2024.1.31)

(24)登録日　令和6年1月23日(2024.1.23)

(51)国際特許分類

Ｇ０６Ｆ　15/167 (2006.01)	Ｇ０６Ｆ	15/167	６１０Ｂ
Ｇ０６Ｆ　12/00 (2006.01)	Ｇ０６Ｆ	12/00	５６０Ｄ
Ｇ０６Ｆ　12/02 (2006.01)	Ｇ０６Ｆ	12/02	５１０Ｍ
	Ｇ０６Ｆ	12/00	５７２Ａ

請求項の数　9　（全22頁）

(21)出願番号　特願2020-117161(P2020-117161)
(22)出願日　令和2年7月7日(2020.7.7)
(65)公開番号　特開2022-14679(P2022-14679A)
(43)公開日　令和4年1月20日(2022.1.20)
審査請求日　令和5年1月27日(2023.1.27)

(73)特許権者　509186579
日立Ａｓｔｅｍｏ株式会社
茨城県ひたちなか市高場２５２０番地
(74)代理人　110001829
弁理士法人開知
(72)発明者　堀口 辰也
東京都千代田区丸の内一丁目６番６号
株式会社日立製作所内
(72)発明者　石郷岡 祐
東京都千代田区丸の内一丁目６番６号
株式会社日立製作所内
(72)発明者　大塚 敏史
東京都千代田区丸の内一丁目６番６号
株式会社日立製作所内
(72)発明者　芹沢 一

最終頁に続く

(54)【発明の名称】　電子制御装置

(57)【特許請求の範囲】

【請求項１】

異なる周期で動作する複数の処理を実行する複数のコアを含むプロセッサと、

前記複数の処理を実行する前記複数のコアごとにそれぞれアクセス可能な複数のメモリ領域を含むメモリと、を備え、

データ書込みを行う先行処理とデータ読出しを行う後続処理とがそれぞれアクセス可能な前記メモリ領域を前記処理の進捗に応じて変更する電子制御装置であって、

前記プロセッサは、

最新の前記処理の結果が書き込まれておらず、かつ読出し中でない書込み可能な前記メモリ領域を探索し、書込み可能な前記メモリ領域が存在しない場合、新たな前記メモリ領域を確保し、新たな前記メモリ領域を前記処理の結果の書き込み先とすることで、前記先行処理から前記後続処理へ前記処理の結果を受け渡すこと、を特徴とする電子制御装置。

【請求項２】

請求項１に記載の電子制御装置であって、

前記複数の処理のそれぞれの開始時刻および終了時刻においてタイマ割込みが入ること、を特徴とする電子制御装置。

【請求項３】

請求項２に記載の電子制御装置であって、

前記メモリ領域は３つの異なるメモリ領域から構成されること、を特徴とする電子制御装置。

【請求項４】
請求項２に記載の電子制御装置であって、
前記メモリ領域は２つの異なるメモリ領域から構成され、
前記プロセッサは、
書込み可能な前記メモリ領域が存在しない場合、新たな前記メモリ領域を確保すること、を特徴とする電子制御装置。
【請求項５】
請求項４に記載の電子制御装置であって、
前記メモリ領域として３つ以上のメモリアドレスを有している条件において、
前記プロセッサは、
１つのメモリアドレスが最新の前記処理の結果の書込み先ではなく、かつ読出し中でもない場合において、前記１つのメモリアドレスを解放すること、を特徴とする電子制御装置。
【請求項６】
請求項１に記載の電子制御装置であって、
前記メモリ領域の管理処理は、
前記プロセッサが専用のミドルウェアを実行することにより実現されること、を特徴とする電子制御装置。
【請求項７】
請求項６に記載の電子制御装置であって、
前記ミドルウェアを実行する前記プロセッサは、
前記複数の処理のそれぞれの終了時刻において、処理未完了を検出した場合に強制的に当該処理を停止させる未完了処理を行うこと、を特徴とする電子制御装置。
【請求項８】
請求項６に記載の電子制御装置であって、
前記ミドルウェアを実行する前記プロセッサは、
前記複数の処理のそれぞれの終了時刻において、処理未完了を検出した場合において、当該処理の演算結果を用いて演算を行う後続処理の開始時刻を、設計時に規定される猶予時間分だけ遅らせることにより、後続処理において最新の演算処理結果を用いて演算を継続すること、を特徴とする電子制御装置。
【請求項９】
請求項１に記載の電子制御装置であって、
前記メモリ領域ごとの占有状況と前記処理の結果の最新値の有無を格納するテーブルを備え、
前記プロセッサは、
前記テーブルのすべての前記メモリ領域について、前記占有状況が専有を示し、または前記最新値の有無が最新値を格納していることを示す場合、書込み可能な前記メモリ領域が存在しないと判定すること、を特徴とする電子制御装置。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、電子制御装置に関する。
【背景技術】
【０００２】
近年、産業用電子制御装置(ECU: Electronic Control Unit)を支える組込みシステムの開発においては、制御性能の向上、厳格化する規制への対応、装置の自律化に伴う制御演算処理増大等による、搭載アプリケーションの複雑化・高負荷化が進んでいる。これに対応する形で、ECUに搭載されるプロセッサにおいてもマルチコア/メニーコア化によるハードウェアの高性能化が進んでいる。
【０００３】

一方で、これら高性能化したハードウェア上に搭載されるソフトウェア開発においては、前述のマルチコア化、メニーコア化に対応したソフトウェア開発が求められ、設計、開発、検証の複雑さが増大する。この原因としてはマルチコアプロセッサの開発において複数のコアが同時に共有資源であるバスやメモリ、キャッシュにアクセスすることから、シングルコアプロセッサ開発と異なりソフトウェア挙動が実行時のプロセッサ状態に応じて容易に変化し得ることが挙げられる。

【０００４】

また、このような課題を解決して製品化を完了した後も、プロセッサ毎に共有資源に関するパラメタ(バス幅、メモリサイズ、キャッシュサイズ)が異なることから、あるプロセッサ上での開発結果を別プロセッサへと移植することも単純ではなく、同様に設計、開発、検証の工数が増大してしまう。

【０００５】

このようなプロセッサ構成の複雑化とそれに伴う工数増に対する方策として、車載ソフトウェアアーキテクチャに関する標準規格であるAUTOSARにおいてタイミング設計に関して時刻同期型のタイミング設計方式であるLET(Logical Execution Time)パラダイムが定義されている。同方式は上流工程でメモリアクセスタイミングの決定を中心とするタイミング設計を行い、このタイミングを各処理が実行時に守ることを特徴としており、これにより各処理の開始/終了時刻におけるメモリ状態が一定となることから、プロセッサ状態の組合せによる開発、検証パターン増を防ぐことができ、工数低減に寄与することができる。

【０００６】

また、異なるプロセッサ間での移植においても、あるプロセッサに対するタイミング設計結果は、別のプロセッサにおいても性能上タイミングを満たすことができる限りにおいて、そのまま移植することが可能となる。

【０００７】

一方で、同パラダイムの構成を実現するためには、データコピーが多発する。これは、同パラダイムが複数タスク間でのデータ共有を行う共有メモリ領域と、各処理が実行時に占有するローカルメモリ領域との間において、処理の開始/終了時にデータコピーを行うような構成を取ることによる。このため大規模なデータを処理間でやり取りするような処理においては、従来設計方式に対しデータコピーのための時間オーバヘッドが大となることから、アプリケーション処理の効率低下が発生する要因となる。

【０００８】

先行特許文献１では、このようなマルチコアプロセッサにおけるコア間通信オーバヘッドを低減する方式として、バッファを用いた解決法が示されている。

【先行技術文献】

【特許文献】

【０００９】

【文献】国際公開第2010/119932号

【発明の概要】

【発明が解決しようとする課題】

【００１０】

一方で、同方式においては、通信機器を前提とした固定的なパケットサイズを単位とする個数管理に対する実装方式が定義されているが、産業用制御装置においては処理間でのデータ通信毎に、その通信量が異なることが特徴である。このように異なるメモリサイズでの処理に対しても、高速なデータ受渡方式が必要である。また、産業用電子制御装置においては、一定の制御周期に沿って動作するアクチュエータを駆動する観点から、各処理、もしくは一連の複数処理が複数の異なる周期で動作することが想定されため、同データ受渡し方式においてはデータ入出力関係のある複数の処理が異なる周期で動作するケースへの対応が求められる。

【００１１】

本発明の目的は、時刻同期型設計におけるアプリケーション間におけるデータ受渡の効率を高めることが可能な電子制御装置を提供することにある。

【課題を解決するための手段】

【００１２】

上記目的を達成するために、本発明は、異なる周期で動作する複数の処理を実行する複数のコアを含むプロセッサと、前記複数の処理を実行する前記複数のコアごとにそれぞれアクセス可能な複数のメモリ領域を含むメモリと、を備え、データ書込みを行う先行処理とデータ読出しを行う後続処理とがそれぞれアクセス可能な前記メモリ領域を前記処理の進捗に応じて変更する電子制御装置であって、前記プロセッサは、最新の前記処理の結果が書き込まれておらず、かつ読出し中でない書込み可能な前記メモリ領域を探索し、書込み可能な前記メモリ領域が存在しない場合、新たな前記メモリ領域を確保し、新たな前記メモリ領域を前記処理の結果の書き込み先とすることで、前記先行処理から前記後続処理へ前記処理の結果を受け渡す。

【発明の効果】

【００１３】

本発明によれば、時刻同期型設計におけるアプリケーション間におけるデータ受渡の効率を高めることができる。上記した以外の課題、構成及び効果は、以下の実施形態の説明により明らかにされる。

【図面の簡単な説明】

【００１４】

【図１】本発明の第１の実施例における、マルチコアプロセッサ１の構成を示す図である。

【図２】本発明の第１の実施例における、時刻同期型のタイミング設計における処理の様子を示す図である。

【図３】本発明の第１の実施例における、共有メモリ上に複数の処理領域を定義することで低時間オーバヘッドなデータ授受方式を実現する例を示す図である。

【図４】本発明の第１の実施例における、周期の異なる複数の処理A/Bの周期間で共有メモリ上に複数の処理領域を定義することで低時間オーバヘッドなデータ授受方式を示す図である。

【図５】本発明の第１の実施例における、共有メモリ上のデータ管理方法の例を示す図である。

【図６】本発明の第１の実施例における、時刻T_1における処理状態を示す図である。

【図７】本発明の第１の実施例における、時刻T_1におけるメモリ管理状態を示す図である。

【図８】本発明の第１の実施例における、時刻T_2における処理状態を示す図である。

【図９】本発明の第１の実施例における、時刻T_2におけるメモリ管理状態を示す図である。

【図１０】本発明の第１の実施例における、データ授受に使うメモリ領域が不足する場合の例を示す図である。

【図１１】本発明の第１の実施例における、データ授受に使うメモリ領域が不足する場合におけるメモリ管理表追加方式を示す図である。

【図１２】本発明の第１の実施例における、ミドルウェア動作を示すフロー図である。

【図１３】本発明の第２の実施例における、共有メモリ上に複数の処理領域を定義することで低時間オーバヘッドなデータ授受方式を実現する例を示す図である。

【図１４】本発明の第２の実施例における、共有メモリ上に複数の処理領域を定義することで低時間オーバヘッドなデータ授受方式を実現する例の別例として、処理Bの周期が処理Aの周期の２倍以上長い際の例を示す図である。

【図１５】本発明の第２の実施例における、共有メモリ上に複数の処理領域を定義することで低時間オーバヘッドなデータ授受方式を実現する例の別例として、処理Bの周期が処理Aの周期より短い例を示す図である。

【図１６】本発明の第３の実施例における、共有メモリ上に複数の処理領域を定義するこ

とで低時間オーバヘッドなデータ授受方式を実現する例を示す図である。
【図１７】本発明の第３の実施例における、予備領域の追加および削除によりメモリ効率化を実現する際の例を示す図である。
【図１８】本発明の第３の実施例における、時刻T_3直前におけるメモリ管理表追加方式を示す図である。
【図１９】本発明の第３の実施例における、時刻T_4直前におけるメモリ管理表追加方式を示す図である。
【図２０】本発明の第３の実施例における、ミドルウェア動作を示すフロー図である。
【図２１】本発明の第４の実施例における、実行時間制約違反時の処理挙動を示すフロー図である。
【図２２】本発明の第４の実施例における、実行時間制約違反時において処理を継続するための方式を示す図である。
【図２３】本発明の第４の実施例における、実行時間制約違反時におけるミドルウェア動作を示すフロー図である。
【発明を実施するための形態】
【００１５】
（実施例１）
以下、本発明に係る第１の実施例について図面を用いて説明する。
【００１６】
図１に、本発明が対象とする、車両制御向け電子制御装置に搭載されるマルチコアプロセッサ１、および周辺機器の構成を示す。このマルチコアプロセッサ１は複数のCPUコア２を有するマルチコアプロセッサである。このようなマルチコアプロセッサ１においては、各CPUコア２がそれぞれ有するレベル１キャッシュ(L1キャッシュ)２１と、複数コアにより共有されるレベル2キャッシュ(L2キャッシュ)２２を有する多段キャッシュ構成が主に用いられ、各コア上での命令実行効率を高める工夫が為されている。各CPUコアは内部バス３を通じてプロセッサ外部にある外部メモリ４や、電子制御装置内外の各種センサ５に接続される。なお、簡単のためマルチコアプロセッサ１内部の詳細構成(各種機構や、詳細なバス構造)は省略する。
【００１７】
このようなマルチコアプロセッサ１上において、複数のコア上で同時にアプリケーション処理を行う場合、複数のCPUコア２が共有する外部メモリ４上のデータに対するアクセス競合、内部バス３にかかる負荷や複数プロセスからのメモリ利用に伴うL2キャッシュ２２のキャッシュヒット率変化により、各アプリケーションの処理効率が変化する。
【００１８】
ここでの処理効率は、アプリケーション処理完了までに要する時間により測定される性能指標である。このため、事前に規定された各アプリケーションの実行順序だけでなく、各アプリケーションの動作に応じて動的に変化するプロセッサの内部状態変化(L2キャッシュ２２状況、内部バス３負荷状況)に応じる形で、アプリケーション処理の状況や順序、メモリアクセスタイミングが変化する。
【００１９】
さらに、割込み処理など、動作タイミングを設計時に確定させることが難しいアプリケーションも存在することから、マルチコアプロセッサ上におけるアプリケーション設計、開発、デバッグ、および検証は、取り得るプロセッサ状態を鑑みてそれぞれ為される必要があり、これら各作業の複雑化と工数増大が懸念される。
【００２０】
また、あるマルチコアプロセッサ１上にて設計・検証されたアプリケーションを、別のマルチコアプロセッサに移植する場合は、CPUコア２の動作速度や数量、バス３の動作速度およびバス幅、キャッシュの構成(L1キャッシュ２１、L2キャッシュ２２、場合によっては更に下位のL3キャッシュなどのキャッシュ階層構造)やそのサイズといったマルチコアプロセッサ１の構成要素変化が発生するため、アプリケーション処理効率やそれに伴う

実行順序、メモリアクセスタイミングが変化する。

【００２１】

そのため、あるプロセッサ１を対象とした設計結果を他のプロセッサ上に移植した際には、必ずしも同等の挙動を示すとは限らず、同様に設計、開発、デバッグ、検証の工程が必要となってしまう。

【００２２】

このようなマルチコアプロセッサ１の利用、および他マルチコアプロセッサへの移植を意識した設計方式として、各アプリケーションの処理開始および完了タイミングを規定する時刻同期型の設計方式が知られている。同設計方式について、図２を用いて説明する。

【００２３】

同設計方式においては、各アプリケーション毎に定められた周期の最初に、全ての処理からアクセス可能な共有メモリ領域から、自身のみがアクセス可能なローカルメモリ領域に演算に必要となる値を読み込み(図中Copy-inと記載した部分)、処理区間の最後に共有メモリに演算結果を書き出す(図中Copy-out)ように構成され、演算はこの読み込みから書き出しまでの間の任意のタイミングで実行される。

【００２４】

同設計方式では、このようにローカルメモリ空間上でアプリケーション処理を実行することにより他アプリケーション処理とのメモリ競合を避けると同時に、処理区間の最後における共有メモリへの書き出し順序を設計時に定義することで、CPUコアやバスの動作速度、キャッシュサイズ、コア数変更に伴うアプリケーション実行コア変化などの挙動変化要因に対しても、設計時に定められた処理時間内に処理が完了する限りにおいて、プロセッサ状態に依らずに共有メモリ状態を一致させられることが期待される。

【００２５】

これにより、プロセッサ状態に起因するアプリケーション処理の挙動変化を抑制すると同時に、また異なるプロセッサ間における高い移植性を得ることができる方式となる。

【００２６】

このような方式を実現する方法の1つとして、前述のようなデータコピーを用いた共有メモリ、ローカルメモリ構成が知られているが、同データコピー方式では特にアプリケーション処理が利用するメモリサイズが大となる場合において、データコピーに要する時間オーバヘッドが大きくなる課題がある。

【００２７】

そこで、図３に示すような２つのバッファを用いた時刻同期型タイミング設計実現手法が知られている。なお、同図における処理区間と周期は、それぞれ時刻同期型タイミング設計方式における処理開始(共有メモリ領域からローカルメモリ領域へデータを読出し)から処理完了(演算完了時にデータを逆方向に書込み)までの時間の幅、およびこのような処理区間を繰り返す際の単位を示すものである。

【００２８】

また図面記載の都合からSwap処理の後に処理Aおよび処理Bがアクセスするメモリ領域が指し示されるまでに時間を要す記載となっているが、実際にはSwap処理の終了時点において各処理がアクセスするメモリ領域は決定されている。

【００２９】

同手法においては、データ依存関係を有する２つの処理A、B(A、Bが互いに前周期の相手の演算結果を用いて、次周期に演算を行う)に対し、共有メモリ上に処理Aが書込みアクセスを行うメモリ領域と処理Bが読出しアクセスを行うメモリ領域との2つのメモリ領域から構成される。ここで、A_i、B_iはそれぞれ処理A、 Bのi周期めの処理の処理を表す記号とする。また、上記のように記載される各処理中においては、A、 B以外の図示しない処理によりプリエンプトされ、再開していてもよい。

【００３０】

このような構成をとると、i番目の処理区間の開始時に、処理A_iおよびB_iはそれぞれ自身が直前に用いたバッファ領域と反対側のバッファ領域を指すことで、処理A_iはB_{i-1}の結果

であるRes(B_{i-1})を用いて自身の計算を行い、処理A_iの演算結果であるRes(A_{i-1})を同領域に書込む動作を行うことができる。Bも同様に、処理Aの前周期における処理結果Res(A_{i-1})を読み込み、自身の演算結果Res(B_i)を書込むことで、演算処理が進んでいく。なお、本図おいて初回に実行される処理A_1およびB_1は、データの入力なしで演算を開始するものとする。

【００３１】

ここで、本動作における各メモリ領域の状態に着目すると、i番めの周期に処理Aにより書込まれた演算結果を、次のi+1番めの周期にBが、i番めの周期に処理Bにより書込まれた演算結果を、次のi+1番めの周期にAが、それぞれ参照して演算を継続するため、データをコピーすることなく演算を継続することが可能となる。

【００３２】

また、このメモリ領域は、データ依存かつ順序依存関係を有する２つの処理A、Bの間でのみ保有され、かつ常に互いに異なるバッファ領域を有することから、処理A、Bの処理区間中はそれぞれの処理のみがアクセス可能なローカルメモリ領域と見做すことができる。

【００３３】

以上から本構成を用いることで、時刻同期型タイミング設計方式における演算構造を実現しつつ、データコピーに要する時間オーバヘッドを削減することが可能となる。

【００３４】

一方で、処理Aと処理Bの周期は必ずしも同一であるとは限らない。例えば自動運転処理を想定すると、自車周辺状況をマッピングする処理が存在するが、同処理の入力は例えば50ms毎に動作するレーザレンジセンサや数ms毎に動作する自車速度センサに連動して処理が行われ、マッピング結果が更新される。

【００３５】

また、パワートレイン分野に目を向けると、例えばエンジン処理のようなケースにおいてはエンジン回転数に同期した処理が発生するため、動作中に制御周期が変化するケースもある。このように異なる周期で動作する処理間でデータ授受が発生する場合は、前述のようにメモリ領域を固定し、同時に更新するような形での方式実現は困難となる。

【００３６】

特に制御周期変化に対応するためには、制御装置の状態に合わせた動的なメモリ領域の管理が必要となる。図４に、処理Aと処理Bの周期が異なる場合の、本動作の例を示す。なお、図面記載の簡単のため、本例では処理区間と周期とが一致する場合について、スワップタイミング記載を省略して説明するが、両者が一致しない場合においても同様の方式により演算処理が実現される。また処理A、Bの周期によっても、メモリアクセスのタイミングや必要となるバッファ領域数が異なる点に注意が必要である。

【００３７】

同図においては、破線で示したAの周期(本例では処理区間に等しい)と、一点鎖線で示したBの周期とが異なっている例を示しており、A、 Bは互いに極力相手の最新処理結果を用いて処理を進める例を示す。すなわち、処理A、 Bが互いに、自身の周期が開始される以前に終了している相手の処理を検出し、当該メモリ領域を読み出す構成を取る。

【００３８】

本例では、処理Aの周期が処理Bよりも長い例を示しているが、これにより、処理AおよびBの開始タイミングによっては、ある処理A_iの結果Res(A_i)が、2つ以上の複数の処理Res(B_j)、Res(B_k)、…の演算過程において読み出されるケースが発生する。同図においては、処理Aの周期は処理Bの周期の2倍以下となっていることから、ある処理A_iの結果Res(A_i)は、最大2つの処理Bの演算に用いられることとなる。

【００３９】

上記の例では、ある処理A_iの結果Res(A_i)が書き込まれたメモリ領域は、処理B_jが演算結果Res(B_j)を演算する過程で破壊されてしまうことから、処理B_kが演算結果Res(B_j)を演算するメモリ領域を別途生成する必要があるため、本実施例では処理A_iの演算終了後に、新たに別のメモリ領域に演算結果をコピーする必要がある。演算結果をコピーする箇所数

は、処理Aと処理Bの周期により定まる。

【００４０】

この領域数は、

(処理Aの周期)/(処理Bの周期)

の小数点以下を切り上げた数を最小値とする値が取られることとなる。

【００４１】

図４に示したコピー処理$Cp(Res(A_i))$を行うことで、例えば図中処理A_2の結果$Res(A_2)$は、処理B_4および処理B_5に用いることが可能となる。なお、本図では処理A_2の演算結果を自身の保有するメモリ領域2に書込む際、同時に別メモリ領域4へ同時に書込む例を想定した。このため、$Res(A_2)$の領域2への書込みが開始される処理A_2の後半部分からコピー処理$Cp(Res(A_2))$が動作開始する例を示しているが、例えばメモリ領域2を処理A_2が確保すると同時に、コピー先のメモリ領域4も確保する構成を取っても構わない。

【００４２】

このようなメモリ管理構成をとることで、先述の同じ周期の処理A、B間において2つのメモリ領域を用いたデータ授受を行った場合と同様、処理Aと処理B間でのデータコピーオーバヘッドは削減される。これは、またデータコピー処理$Cp(Res(A_i))$も処理A_i実行中に行われることから、例えば処理A_2の後続処理B_5の開始時には、既にデータがメモリ領域上に備わっているためである。

【００４３】

以上のメモリ管理構成を実現する手段として、専用のミドルウェアを定義する方法が考えられる。ミドルウェアを定義することで、アプリケーション開発者は細かなメモリ管理を意識せずアプリケーションの挙動に集中することができ、また産業用制御装置におけるOSやハードウェアといった変化要素からもアプリケーションを解放することができ、時刻同期型タイミング設計の目的であった異なるハードウェア(プロセッサ)間での移植性向上と最も整合する。

【００４４】

同方式をミドルウェアを用いて実現する場合の実装方法の例としては、ミドルウェアが当該メモリ領域を管理し、アプリケーション側にはメモリ領域を示すポインタとして渡す方法などが想定される。本実施例においては、この方式に基づき説明する。

【００４５】

上記のミドルウェアには、各処理における処理区間毎に更新される共有メモリ上のデータを管理し各処理の起動時に適切なメモリ領域を通知すること、および各処理間でデータ授受に使うメモリ領域が不足した際に共有メモリ上に新たなメモリ領域を作成することでデータ授受を維持すること、の2つの機能が求められる。それぞれについて、以下説明する。

【００４６】

まず、共有メモリ上のデータ管理方法の例を、図５に示す。

【００４７】

本方式における必要事項は、ある処理が演算中のメモリ領域に、他の処理がアクセスすることを防ぐこと、および各処理が開始される際に適切な入力情報が書込まれているメモリ領域を渡すこと、の2点である。ここでの適切な入力情報とは、当該処理の開始時刻までに完了している、最新の相手処理の演算結果である。

【００４８】

前者は例えばメモリ上のデータ管理にはメモリ領域の占有状況として、後者はメモリ領域に格納された最新演算結果の位置として、それぞれ管理される。本例では、複数の処理が相互にデータを授受する構成であるため、最新演算結果は双方向に定義される。

【００４９】

図６および図７を用いて、本実施例における処理A_2開始時(時刻T_1)におけるミドルウェア挙動を説明する。時刻T_1においては、メモリ領域3が処理B_2により占有されている。また、処理A_1および処理B_1が、当該時刻の開始時刻までに完了した最新の処理となるため、

処理A_1に関してはメモリ領域１および４が、処理B_1に関してはメモリ領域２が、それぞれ最新処理結果であることが示される。本例では前述のように、処理Aの周期が処理Bより長いことから、処理Aは複数個所に演算結果を保存している。

【００５０】

時刻T_1にて起動される処理A_2に対し、ミドルウェアは図７の管理テーブルにおける最新処理結果の列のうち、処理Bから処理Aの項目を検索する。結果、領域２に最新処理結果(処理B_1の演算結果)が記載されていることから、当該領域が処理A_2の利用すべき領域と分かる。そこで、管理テーブルにおける処理Aによる占有状況欄のメモリ領域２部分にチェック（図７の点線の丸）を入れ、当該メモリ領域のアドレスを処理Aに渡すことで処理A_2を起動する。

【００５１】

次に、時刻T_2にて処理B_2が終了した際の挙動を図８および図９を用いて説明する。図８に示すように、同時刻において処理A_2は継続していること、および当該時刻での最新演算結果が処理B_2の結果となることから、処理B_2が占有していたメモリ領域３が最新の演算結果を保持している領域と分かる。

【００５２】

そこで、処理Bによる占有状況テーブルからメモリ領域３のチェックを外し、代わりに処理B→処理Aへの最新値テーブルの領域３の箇所にチェックを入れる。これにより、メモリ管理テーブルが図９のように更新される。このように管理テーブルが更新されることで、時刻T_2直後に別の処理Aが起動された場合には、メモリ領域３を用いることで処理Bの最新処理結果を用いた演算を行うことが可能となる。

【００５３】

次に、各処理間でデータ授受に使うメモリ領域が不足した際に共有メモリ上に新たなメモリ領域を作成することでデータ授受を維持する方法について、図10および図11を用いて説明する。

【００５４】

図１０に示した時刻T_0は、処理Aにおいて演算結果の書込みが開始されたタイミングである。前述のように、処理Aは処理Bに対して長い周期と定義されており、処理Aから処理Bへのデータ授受については複数のメモリ領域を備える必要がある。そこで、処理Aは複数のメモリ領域を確保する必要があるが、同タイミングにおいて確保されているメモリ領域は領域１のみであるため、ミドルウェアを通じて新たな書込み先を確保する必要がある。

【００５５】

そこで、ミドルウェアは図１１の管理テーブルを参照するが、確保されているメモリ領域１およびメモリ領域３は処理A_1、処理B_2にそれぞれ占有されており、またメモリ領域２には処理Bの最新処理結果が格納されていることから、処理A_1が新たに確保することができるメモリ領域が存在しない。そこで、新たに共有メモリの一部(処理A_1の書込みサイズに相当)を処理A-処理B間のデータ授受用メモリ領域として確保および管理テーブル上の占有状態をAと更新し、メモリ領域４を作成する。このメモリ領域４を処理Aに渡すことで、処理A_1の終了時には処理Aの最新値が両メモリ領域に書込まれた状態となる。

【００５６】

以上の構成によりミドルウェアを用いた上述のメモリ管理方式が提供され、先述の同じ周期の処理A、B間において2つのメモリ領域を用いたデータ授受を行った場合と同様、処理Aと処理B間でのデータコピーオーバヘッドは削減される。

【００５７】

また、産業用電子制御装置においては、一定の制御周期に沿って動作するアクチュエータを駆動する観点から、各処理、もしくは一連の複数処理に対する実行時間制約(リアルタイム制約)が存在する。これに違反した場合、アクチュエータ制御指令値が出力されない、もしくはセンサ入力値に対して適切でない指令値が出力される可能性があるため、アクチュエータに対応した未完了時処理が必要となる。例えば、冗長系構成を用いた別系統を利用した制御継続方式や、縮退制御方式などが挙げられる。

【００５８】
本構成においては、アプリケーション処理の終了予定時刻が設計時に規定されていることから、このタイミングまでに処理が完了しないケースに対して前述の未完了処理が実行される。未完了処理実施の要否をミドルウェアが判別するためには、例えばミドルウェアが処理を起動する際にフラグをセットし、処理の完了時にこのフラグを解除することで、処理終了時刻においてミドルウェアがチェックするような実装が考えられる。
【００５９】
以上の構成を実現するミドルウェア挙動について、図１２のフローチャートを用いて説明する。ミドルウェアは前述のように各処理の起動時刻に合わせる形で動作するため、例えば処理開始時刻のタイマ割込みにより起動される。換言すれば、複数の処理のそれぞれの開始時刻においてタイマ割込みが入る。その後、管理テーブルから最新処理結果格納先を探索し(S100)、当該領域を確保するため管理テーブルの占有状況箇所を更新する(S101)。
【００６０】
そして、当該メモリ領域を処理に渡して処理を起動することで、アプリケーション処理が開始される(S102)。ここで、アプリケーション処理の周期の長短により、複数のメモリ領域へデータを書込む必要があり、かつそのような書込み先領域が確保されていない場合には新たなメモリ領域を確保し(S103)、管理表にアドレスを追加(S104)、当該書込み先の状態を占有中に変更した上で(S105)、処理に渡す(S106)。
【００６１】
書込み先領域がミドルウェアにより既に確保され管理テーブルに記載されている場合は、S103およびS104をスキップし、S105が実施される。そして、アプリケーション処理終了時刻には開始時と同様にタイマ割込みが入る。換言すれば、複数の処理のそれぞれの終了時刻においてタイマ割込みが入る。アプリケーション処理終了時においては、メモリ管理テーブルの最新値状態を更新し(S107)、メモリ占有状態を解除する(S108)。以上により、1つのアプリケーション処理に対するミドルウェア処理が完結する。
【００６２】
換言すれば、本実施例の電子制御装置は、異なる周期で動作する複数の処理を実行する複数のコア（CPUコア２）を含むプロセッサ（マルチコアプロセッサ１）と、複数の処理を実行する前記複数のコア（CPUコア２）ごとにそれぞれアクセス可能な複数のメモリ領域を含むメモリ（共有メモリ）と、を備え、データ書込みを行う先行処理とデータ読出しを行う後続処理とがそれぞれアクセス可能な前記メモリ領域を処理の進捗に応じて変更する。プロセッサ（マルチコアプロセッサ１）は、最新の処理の結果が書き込まれておらず、かつ読出し中でない書込み可能なメモリ領域を探索し、書込み可能なメモリ領域が存在しない場合、新たなメモリ領域を確保し、新たなメモリ領域を処理の結果の書き込み先とすることで、先行処理から後続処理へ処理の結果を受け渡す。
【００６３】
図５に示すように、電子制御装置は、メモリ領域ごとの占有状況と処理の結果の最新値の有無を格納するテーブルを備える。プロセッサは、このテーブルのすべてのメモリ領域について、占有状況が専有を示し、または最新値の有無が最新値を格納していることを示す場合、書込み可能な前記メモリ領域が存在しないと判定する。
【００６４】
以上の構成により、マルチコアCPUにおける設計、開発、検証工数を低減する時刻同期型タイミング設計方式を低時間オーバヘッドに実現し、かつ異なるハードウェア(プロセッサ)間での移植性向上を実現する方式が提供される。
【００６５】
（実施例２）
以下、本発明に係る第２の実施例について図面を用いて説明する。想定するCPU構成(図１)や適用する時刻同期型タイミング設計方式(図２)は実施例１に準拠するが、本実施例では複数の処理Aおよび処理Bの間に一方向のデータ依存関係があるケース、すなわち処理A

の実行結果を利用して処理Bが実行されるが、逆は存在しないケースにおける本発明の適用事例を、主に実施例１との差分に着目して説明する。

【００６６】

本実施例においては、処理B実行に必要な処理Aの演算結果のみを共有メモリ上に展開することで、例えば処理Aへの入力データといった処理B実行に不要なデータを共有メモリ上に複数保持する必要がなく、処理間のメモリ利用効率の向上および、これに伴う制御装置内におけるメモリ利用効率向上、最終的には制御装置に搭載するメモリサイズ低減による低コスト化が期待される。

【００６７】

図１３に、本発明が例題とする処理構成を示す。本実施例では、メモリ領域は３つの異なるメモリ領域（領域１～３）から構成される。前述のように、本実施例においては、処理Aの演算結果を処理Bが読込む構成となっていることから、図中の共有メモリ領域記載においては、処理Aは自身の演算結果の書込みのみ、処理Bは処理Aの演算結果の読込みのみ、をそれぞれ記載している。

【００６８】

処理Aが自身の演算に用いる読込データや、処理Bが自身の演算結果を書込む領域については、図中の共有メモリ領域に書込む構成をとることも、それ以外のデータ依存関係を有する処理との間で同様の共有メモリ構成をとることも、可能である。

【００６９】

このような共有メモリの利用方式においては、処理A、Bの2つの処理間でデータ共有する場合については、処理Aの書込み先が2箇所(処理Aの周期が処理Bの周期より短い場合)と、処理Bの読出し先が1ヶ所の合計3箇所以上あれば、実施例１に記載のデータ受渡し(1つの処理が書込みもしくは読出しのいずれかのみを担当する)が可能となる。

【００７０】

なお、本実施例のような構成については、図１４に示したように、処理Aの周期が処理Bの周期に対して更に小さくなる場合についても、同様のメモリ構成をとることが可能である。このように、処理Bの周期が、処理Aの周期の2倍以上となったケースについては、実施例１に示したメモリ管理表において、最新値ではないメモリ領域に処理結果が上書きされることで、共有メモリ領域の更なる追加が不要となる。

【００７１】

例えば、処理A_1の演算結果Res(A_1)が処理Bにより読出される前に処理A_2が演算完了するため、処理B_2には処理A_2の演算結果Res(A_2)が読出されればよい。そのため、同図に示したように、処理A_3の演算結果Res(A_3)は、メモリ領域2にある処理A_1の演算結果Res(A_1)を上書きすることが許容される。

【００７２】

逆に、処理Aの周期が処理Bの周期より長いケースにおいても、処理Bが自身が読出すメモリ領域に自身の演算結果を上書きしない限りにおいて、図１５に示すように同様の構成をとることが可能となる。

【００７３】

以上の構成、すなわちデータ受け渡しが発生する2つの処理間において、1つの処理が書込みのみ、もう1つの処理は読出しのみ、を担当する場合については、本実施例のように3つの共有メモリ領域のみを用いることで、実施例1の事例に比べ、必要メモリ量の削減およびミドルウェア(もしくは、実装形態によりOS、各アプリケーション)におけるメモリ領域の追加判定処理が不要となり、実装の効率化が図られる。

【００７４】

（実施例３）

以下、本発明に係る第３の実施例について図面を用いて説明する。

【００７５】

図１６に、本発明が例題とする処理構成を示す。本例においては、実施例２と同様に処理Aは書込みのみを、処理Bは読出しのみを行う構成ではあるが、Aの処理区間とAの周期

とが不一致な場合を例として説明する。時刻同期型のタイミング設計方式においては、前述のように必ずしも処理区間と周期とが一致する必要はないため、このような構成をとることが可能となる。

【００７６】

このようなケースにおいては、同図に示したように、処理A_3の演算結果Res(A_3)が読みだされる前に処理A_4が完了し、処理B_4には処理A_4の演算結果Res(A_4)を用いて演算を開始することができる。この場合、実施例１および２とは異なり、共有メモリ領域の利用率が低下し、メモリ効率の悪化が発生する。

【００７７】

そこで、共有メモリ上には(同一周期の場合と同様に)最低限必要な２つのメモリ領域を構成し、３つめのメモリ領域は必要に応じて共有メモリから動的に確保し、不要と判別された場合には解放することで、制御装置全体の処理構成の間で予備領域を融通し、メモリ効率を更に向上することが可能となる。

【００７８】

換言すれば、本実施例では、メモリ領域は２つの異なるメモリ領域から構成され、プロセッサ（マルチコアプロセッサ１）は、書込み可能なメモリ領域が存在しない場合、新たなメモリ領域を確保する。その後、例えば、メモリ領域として３つ以上のメモリアドレスを有している条件において、プロセッサ（マルチコアプロセッサ１）は、１つのメモリアドレスが最新の処理の結果の書込み先ではなく、かつ読出し中でもない場合において、前記１つのメモリアドレスを解放する。

【００７９】

以上の構成を、ミドルウェアによりメモリ管理により実現する方式を示す。本実施例では、メモリ領域の管理処理は、プロセッサ（マルチコアプロセッサ１）が専用のミドルウェアを実行することにより実現される。なお、実施例１と同様、本実施例で説明するミドルウェア処理はOSや各処理内部にて実施されても良い。

【００８０】

予備領域の確保については、実施例１に説明した構成と同様のもので実現可能であるため、本実施例においては予備領域の解放処理について説明する。

【００８１】

前述のように、予備領域が不要となるケースについて図１７から図１９を用いて説明する。図１７における時刻T_3直前(処理A_4の処理区間終了直前)においては、図１８のようにメモリ領域１は処理A_4がメモリ領域２を占有、また処理A_3の結果が最新値となっている。

【００８２】

時刻T_3直後のメモリ状態は図１９のようになっており、メモリ領域２がB占有中かつ最新処理結果が書き込まれており、予備領域は空欄となる。このように処理A_4の完了に伴い、予備領域が占有されておらず、かつ最新値が書き込まれていないケースにおいて、予備領域の解放が可能となる。処理Aの周期が処理Bより長い場合においては、最新値が書き込まれておらず、かつ処理Aが予備領域を占有する必要がない場合に、同領域が解放可能となる。

【００８３】

以上のことから、ミドルウェア処理では、処理の開始時にメモリ解放可能かを判別する構成をとることで、全てのケースに対して適用が可能となる(処理Aの周期が処理Bより短い場合は、次の処理A起動時まではオーバヘッドが継続する)。

【００８４】

図２０に、このようなミドルウェア挙動フローを示す。基本的な構成は実施例１にと同様だが、処理Aおよび処理Bは書込みもしくは読出し処理のいずれかを行うため、この判別機構によりフローが少々異なっている。そのため、処理起動時に、読出し処理か書込み処理化の条件分岐が入る。以降は実施例１のミドルウェア処理を踏襲する。本例において追加となるのは、書込み処理側において書込み可能領域が存在した場合である。この際に、管理メモリ領域数が3つ以上であり、かつ、予備領域が最新値を保持しておらず読出し中

でもない場合に、解放可能となり、メモリ領域を管理表から削除する(S109)。

【００８５】

なお、本例では処理間でのデータ受渡し用に用いる固定的な２つのメモリ領域と、必要に応じて動的に確保/解放する1つの予備領域を用いる例について説明したが、2つのメモリ領域が常に確保されるのであれば、前述の固定的なメモリ領域としての形態は不要である。

【００８６】

すなわち、常に管理テーブル上に確保される2つのメモリ領域と、さらに必要に応じて動的に確保/解放される1つのメモリ領域、の構成でも、本実施例は実現可能である。この場合には、管理メモリ領域の解放条件は、管理アドレスが３つ以上であり、かつ、最新値を保持せず読出し中でもない、場合において、管理表からアドレス削除を行うことが可能となる。この場合には、S109処理が、当該メモリ領域を探索し削除、と変化する。

【００８７】

（実施例４）

以下、本発明に係る第４の実施例について図面を用いて説明する。

【００８８】

産業系電子制御装置はセンサ情報の入力に基づき制御指令値を演算しアクチュエータを制御する構成が取られる。このような構成においては、周期的に動作するアクチュエータを正しく制御するため、一般的にセンサ入力から制御指令値出力までの間に実行時間制約が存在する。

【００８９】

一方で、マルチコア/メニーコアCPUにおける設計においては、前述の複数コア間におけるソフトウェア挙動は実行時のプロセッサ状態に応じて容易に変化し得ることから、例えば実行時間の変動が発生する。このような実行時間変動に対して、設計時におけるマージン設計などによる対処が取られるが、実行時間制約の厳しさや各種資源の競合により必ずしも処理完了が保証されるとは限らない。そのため、ミドルウェアに処理完了の確認と、未完了時の対処が必要となる。

【００９０】

図２１に、処理Aの図中2回目の実行である処理A_2が、実行時間変動の影響により規定の処理区間内に処理を完了できなかった例を示す。

【００９１】

本時刻同期型タイミング設計方式における、処理未完了時の対処としては以下の２つの方式が想定される。１つは、処理A_2を強制的に中断もしくは実行終了させるケースである。対処法としては上位システムへ通知を行い、判断を仰ぐケースや、設計時にこのような違反発生ケースに対する処理が規定され、そちらに移行するケースも考えられるが、制御目的や実行時間制約違反時の影響範囲が小さいケースにおいては処理を継続するケースも想定される。

【００９２】

このような場合、ミドルウェアでは処理A_2を中止し、かつメモリ管理テーブルを更新しないことで、処理A_2の演算結果$Res(A_2)$を用いるはずであった処理B_4および処理B_5が、演算未完了な値を用いないように制御する。これら処理は、処理A_1の演算結果$Res(A_1)$を用いて制御を継続する。

【００９３】

換言すれば、ミドルウェアを実行するプロセッサ（マルチコアプロセッサ１）は、複数の処理のそれぞれの終了時刻において、処理未完了を検出した場合に強制的に当該処理を停止させる未完了処理を行う。

【００９４】

もう1つの方式は、このようなケースにおいて、処理Aに対する処理Bのような、先行処理の結果を用いて処理を行う後続処理の設計時マージンを用いて処理Aの実行時間を延長することで、後続処理Bに最新の処理結果を渡す方式である。

【００９５】
換言すれば、ミドルウェアを実行するプロセッサ（マルチコアプロセッサ１）は、複数の処理のそれぞれの終了時刻において、処理未完了を検出した場合において、当該処理の演算結果を用いて演算を行う後続処理の開始時刻を、設計時に規定される猶予時間分だけ遅らせることにより、後続処理において最新の演算処理結果を用いて演算を継続する。
【００９６】
図２１において、処理A_2の後続処理である処理B_4、およびコア数によっては処理A_2の遅れの影響が自身の処理時間削減に繋がる処理A_3の両者は、共にマージン時間、すなわち処理区間と各処理の実行時間との差分が存在する。この時間は通常、前述のようなマルチコア上における実行時間変動を吸収するために用いられるものであるが、設計時における最悪実行時間が常に発生するわけではないことから、同マージン時間を利用することは可能である。
【００９７】
本実施例においては、例えばマージン時間を最悪実行時間+20%と設計し、そのうち10%をこのような先行処理の遅れに対し適用可能であるとする。このような時間を、遅延吸収時間（図２２）と呼ぶと、このような条件において、処理A_2に遅延が発生した場合、影響を受ける処理(処理B_4、および例えば2コアで本処理を行う場合には処理A_3も含む)のうち、直近の遅延吸収時間終了時刻(同図における時刻T_4)を、ミドルウェアがタスク情報などから取得し、同時刻まで処理A_2を継続する。同時刻において処理A_2が完了した場合には実施例３に記載のメモリ管理と同様の終了処理を実施し、同時刻においても処理A_2が完了しない場合には、前述の1つめの方式と同様の強制終了処理を取る。
【００９８】
図２３に、本実施例におけるミドルウェア挙動に関するフロー図を示す。上述の未完了時処理(S110)を追加することで、産業用制御装置に求められる実行時間制約に関し、特に制約違反の検出および対処方式が提供され、より幅広い産業用制御装置への本方式適用が実現される。
【００９９】
以上の構成により、産業用制御装置に求められる実時間制約に対し、本発明が準拠することが出来る。また上記に示した２つめの対処法においては、実時間制約に違反した場合においても、処理実行上のマージン時間を用いて対処可能となる場合において処理継続もしくは最新の処理結果を用いた処理継続を可能とすることで、産業用制御装置による制御演算継続およびその精度向上が実現される。
【０１００】
以上説明したように、第１～第４の実施例によれば、時刻同期型設計におけるアプリケーション間におけるデータ受渡の効率を高めることができる。
【０１０１】
なお、本発明は上記した実施例に限定されるものではなく、様々な変形例が含まれる。例えば、上述した実施例は本発明を分かりやすく説明するために詳細に説明したものであり、必ずしも説明した全ての構成を備えるものに限定されるものではない。また、ある実施例の構成の一部を他の実施例の構成に置き換えることが可能であり、また、ある実施例の構成に他の実施例の構成を加えることも可能である。また、各実施例の構成の一部について、他の構成の追加・削除・置換をすることが可能である。
【０１０２】
また、上記の各構成、機能等は、それらの一部又は全部を、例えば集積回路で設計する等によりハードウェアで実現してもよい。また、上記の各構成、機能等は、プロセッサがそれぞれの機能を実現するプログラムを解釈し、実行することによりソフトウェアで実現してもよい。各機能を実現するプログラム、テーブル、ファイル等の情報は、メモリや、ハードディスク、SSD（Solid State Drive）等の記録装置、または、ICカード、SDカード、DVD等の記録媒体に置くことができる。
【０１０３】

なお、本発明の実施例は、以下の態様であってもよい。

【０１０４】

複数のコアを含むプロセッサを有する電子制御装置であって、前記プロセッサは、異なる周期で動作する複数の処理を実行し、前記複数の処理を実行する前記複数のコアごとにそれぞれアクセス可能な複数のメモリ領域を備え、データ書込みを行う先行処理と、データ読出しを行う後続処理とがそれぞれアクセス可能な前記メモリ領域を、前記処理の進捗に応じて変更する電子制御装置において、最新の前記処理の結果が書き込まれておらず、かつ読出し中でない書込み可能な前記メモリ領域を探索し、書込み可能な前記メモリ領域が存在しない場合、新たな前記メモリ領域を確保し、新たな前記メモリ領域を前記処理の結果の書き込み先とすることで、前記先行処理から前記後続処理へ前記処理の結果を受け渡すこと、を特徴とする電子制御装置。

【０１０５】

これにより、産業用制御装置に求められる時刻同期型設計におけるアプリケーション間におけるデータ受渡の効率を高めることが可能である。

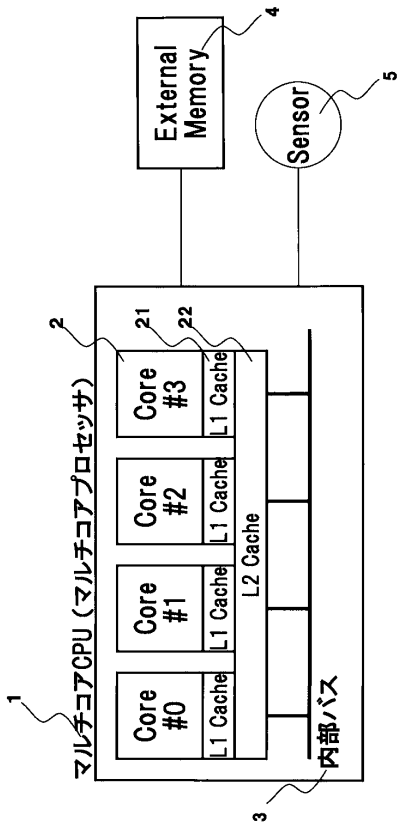
【符号の説明】

【０１０６】

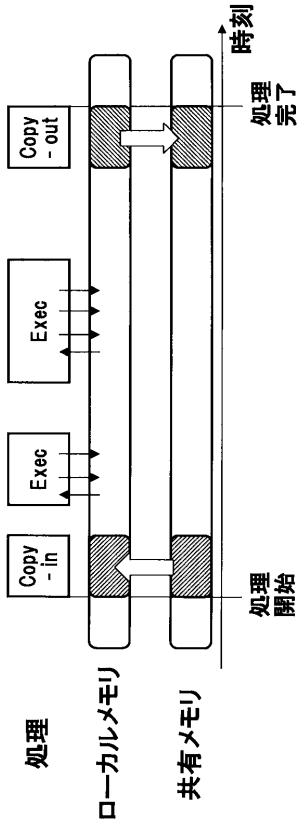
１：マルチコアプロセッサ
２：CPUコア
２１：L1キャッシュ
２２：L2キャッシュ
３：内部バス
４：外部メモリ
５：センサ

【図面】

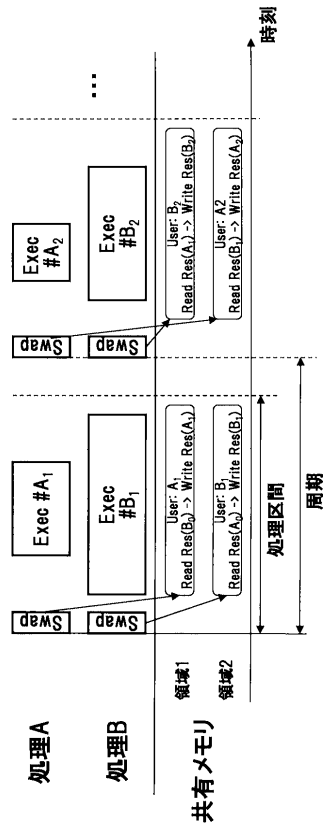
【図１】


External Memory
4
Sensor
5
マルチコアCPU（マルチコアプロセッサ）
1
2
21
22
Core #0
Core #1
Core #2
Core #3
L1 Cache
L2 Cache
内部バス
3

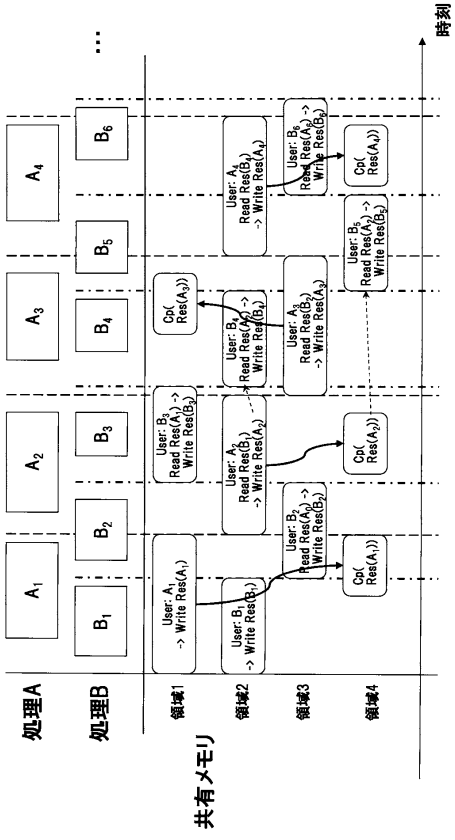
【図２】


時刻
処理完了
Copy - out
Exec
Exec
Copy - in
処理開始
処理
ローカルメモリ
共有メモリ

【図３】


時刻
Exec #A1
Exec #B1
Exec #A2
Exec #B2
Swap
User: A1 Read Res(B0) -> Write Res(A1)
User: B1 Read Res(A0) -> Write Res(B1)
User: B1 Read Res(A1) -> Write Res(B2)
User: A2 Read Res(B1) -> Write Res(A2)
処理区間
周期
処理A
処理B
共有メモリ
領域1
領域2

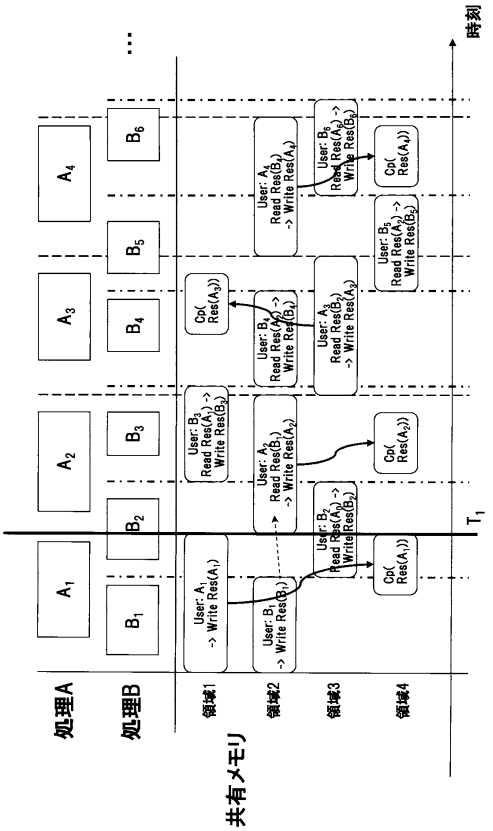
【図４】


時刻
処理A
処理B
共有メモリ
領域1
領域2
領域3
領域4

【図５】

	占有状況		最新値	
	A占有中	B占有中	A→B	B→A
領域1			○	
領域2				○
領域3		○		
領域4			○	
		…		

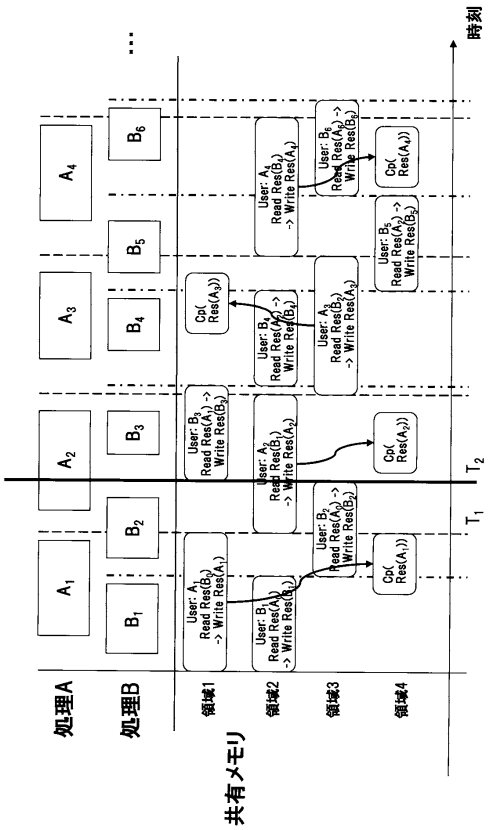
【図６】



【図７】

	占有状況		最新値	
	A占有中	B占有中	A→B	B→A
領域1			○	
領域2	◌			○
領域3		○		
領域4			○	
		…		

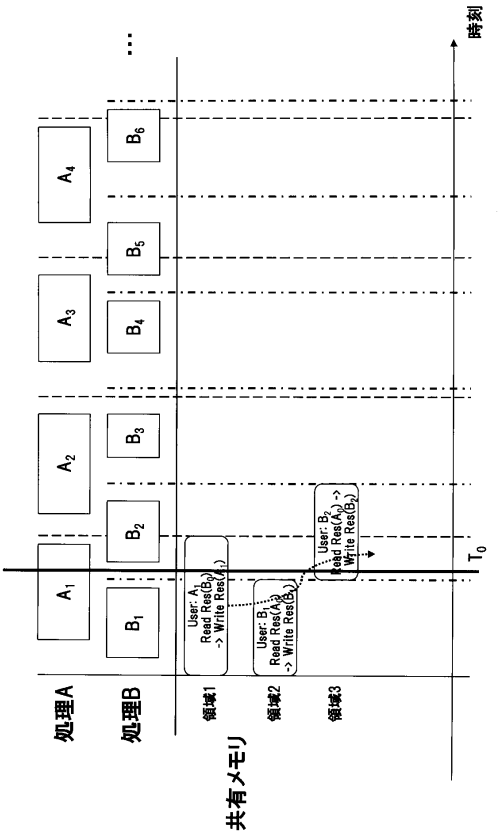
【図８】



【図９】

	占有状況		最新値	
	A占有中	B占有中	A → B	B → A
領域1			○	
領域2	○			
領域3				○
領域4			○	
		⋮		

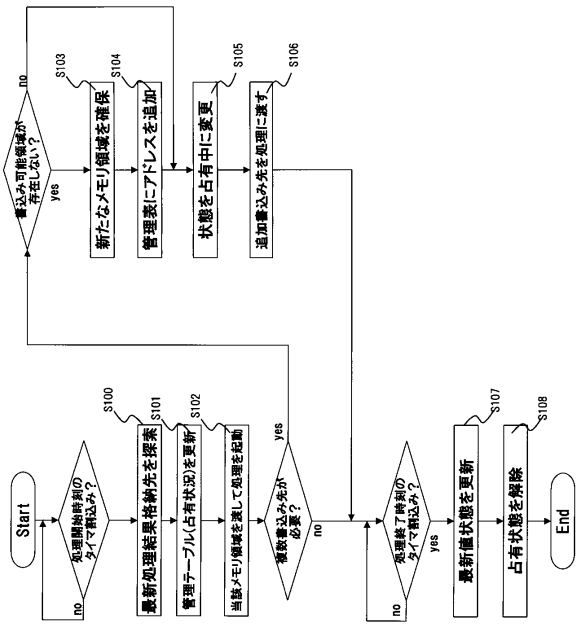
【図１０】



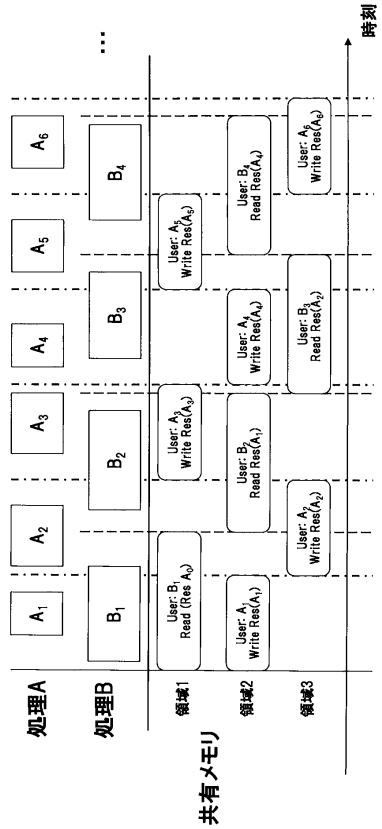
【図１１】

	占有状況		最新値	
	A占有中	B占有中	A → B	B → A
領域1	○		○	
領域2		○		
領域3				○
(作成) → 領域4	◌		◌	
		⋮		

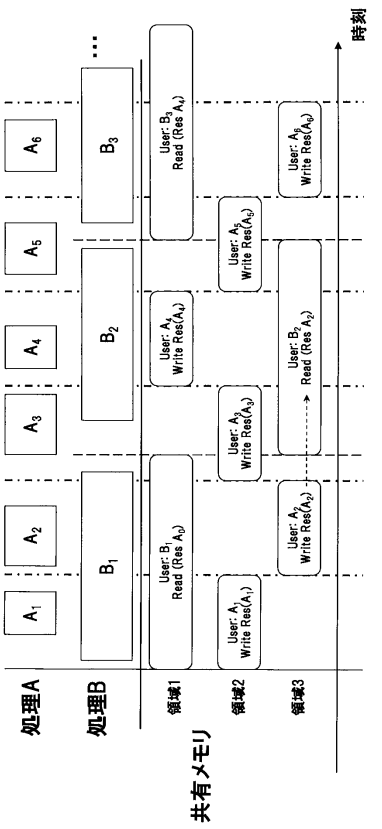
【図１２】



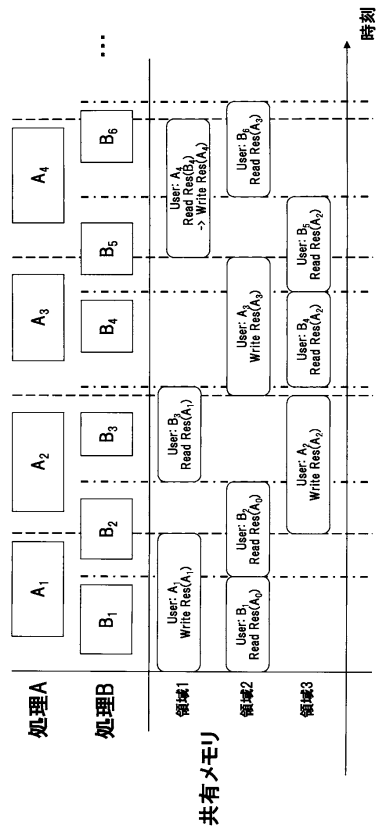
【図１３】



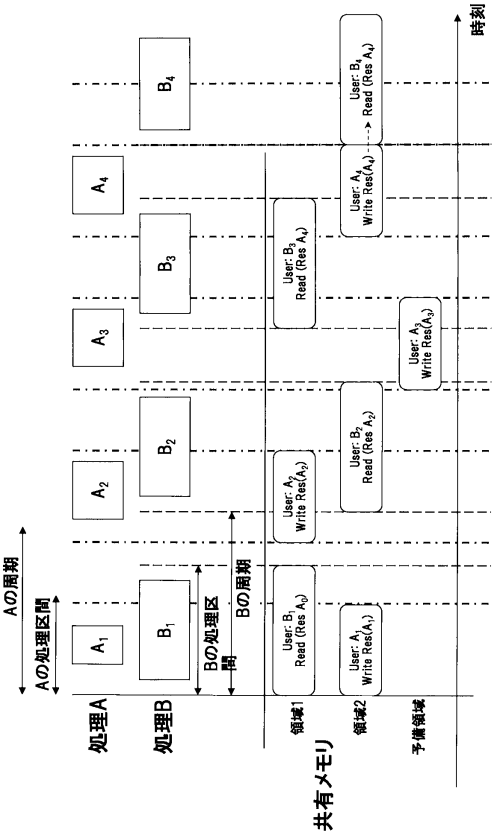
【図１４】



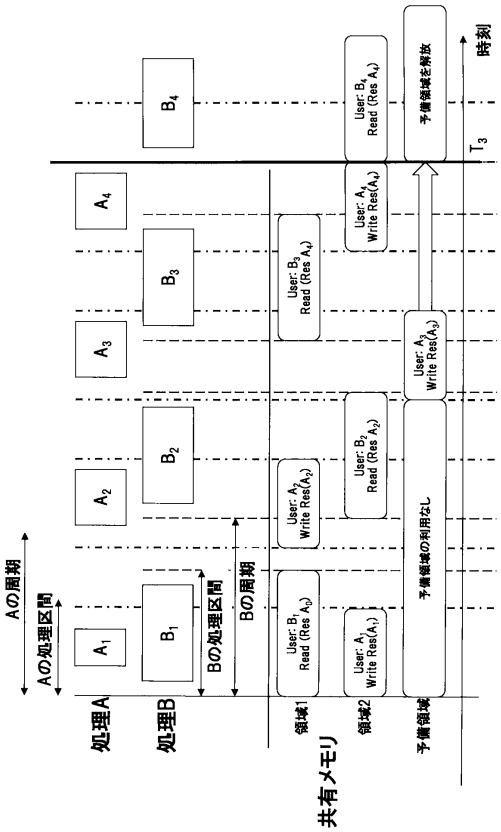
【図１５】



【図１６】



【図１７】



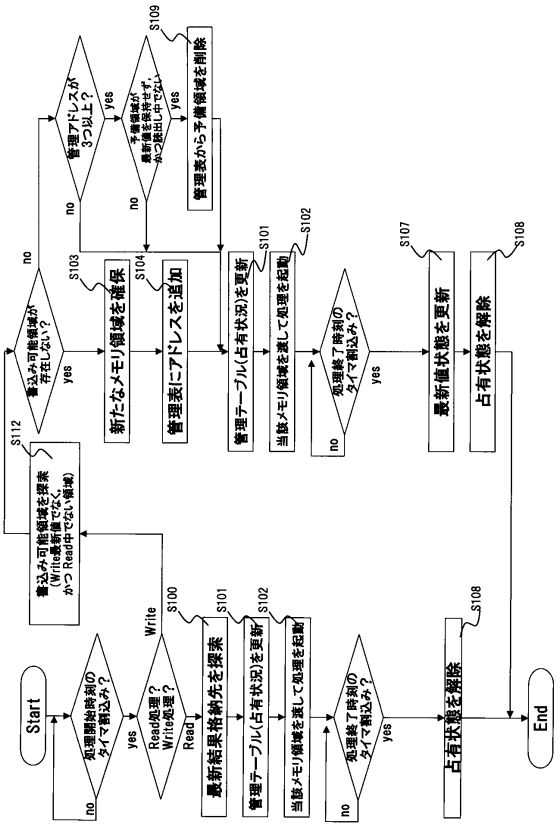
【図１８】

	占有状況		最新値	
	A占有中	B占有中	A→B	B→A
領域1				
領域2	○			
予備領域			○	
		…		

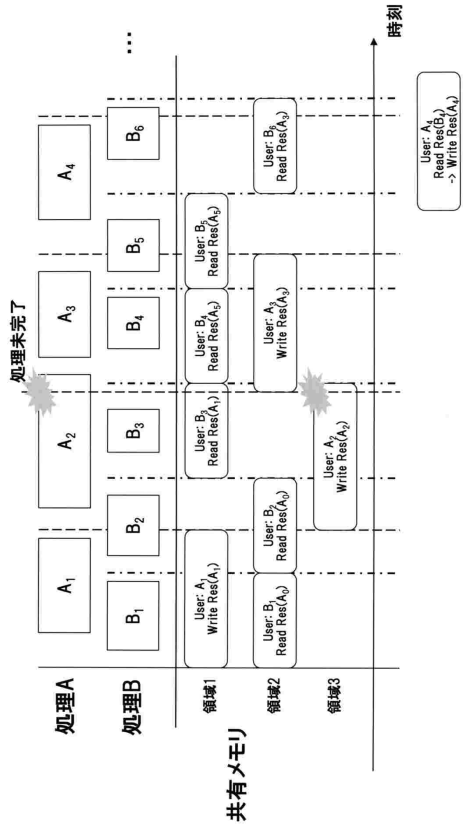
【図１９】

	占有状況		最新値	
	A占有中	B占有中	A→B	B→A
領域1				
領域2		○	○	
予備領域				
		…		

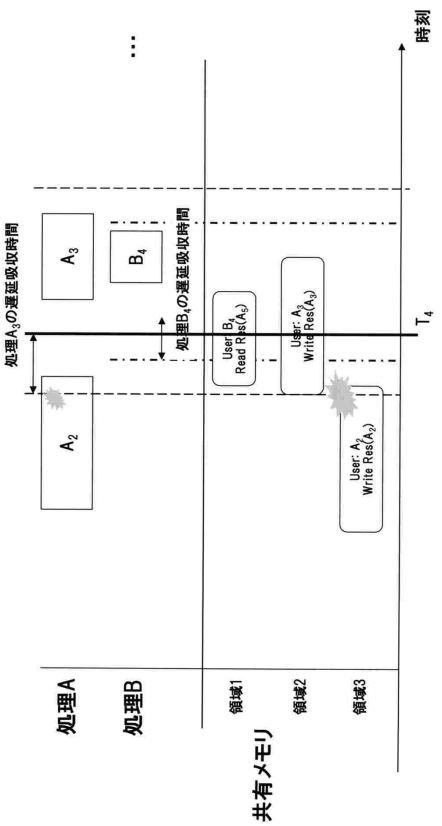
【図２０】



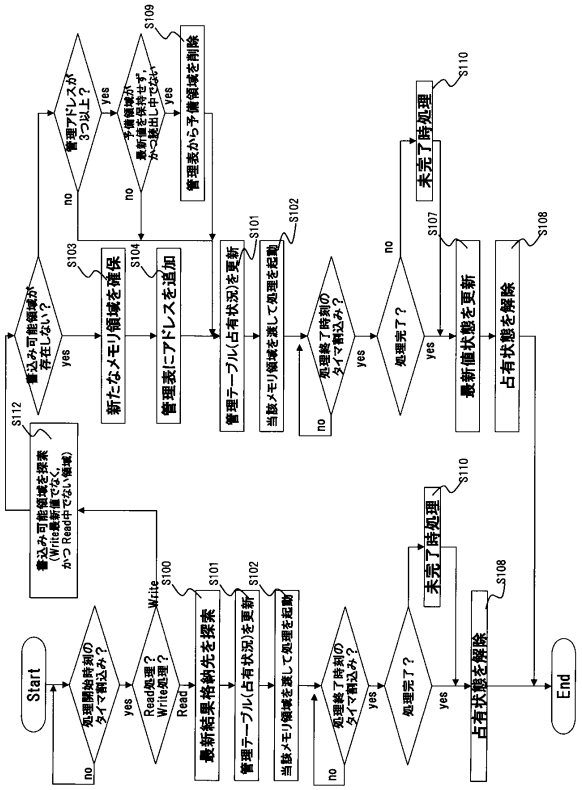
【図２１】


処理未完了
処理A
処理B
領域1
領域2
領域3
共有メモリ
時刻
User: A1 Write Res(A1)
User: B1 Read Res(A1)
User: B2 Read Res(A2)
User: A2 Write Res(A2)
User: B3 Read Res(A3)
User: B4 Read Res(A3)
User: A3 Write Res(A3)
User: B5 Read Res(A5)
User: B6 Read Res(A5)
User: A4 Read Res(B3) -> Write Res(A4)

【図２２】


処理A3の遅延吸収時間
処理B4の遅延吸収時間
T4
処理A
処理B
領域1
領域2
領域3
共有メモリ
時刻
User: B4 Read Res(A3)
User: A3 Write Res(A3)
User: A2 Write Res(A2)

【図２３】


Start
End
処理開始時刻のタイマ割込み？
Read処理？ Write処理？
Write
Read
書込み可能領域を探索 (Write最新値でなく、かつRead中でない領域)
最新結果格納先を探索
管理テーブル(占有状況)を更新
当該メモリ領域を渡して処理を起動
処理終了時刻のタイマ割込み？
処理完了？
未完了時処理
占有状態を解除
書込み可能領域が存在しない？
新たなメモリ領域を確保
管理表にアドレスを追加
管理アドレスが3つ以上？
予備領域が最新値を保持せず、かつ読出し中でない
管理表から予備領域を削除
最新値状態を更新
S100
S101
S102
S103
S104
S107
S108
S109
S110
S112
yes
no

フロントページの続き

茨城県ひたちなか市高場２５２０番地　日立オートモティブシステムズ株式会社内
(72)発明者　村上 隆
茨城県ひたちなか市高場２５２０番地　日立オートモティブシステムズ株式会社内
審査官　田中 幸雄
(56)参考文献　特開２０１０－２４４０９６（ＪＰ，Ａ）
国際公開第２０１０／１１９９３２（WO，Ａ１）
(58)調査した分野　(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１５／１６７
Ｇ０６Ｆ　１２／００
Ｇ０６Ｆ　１２／０２