(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2014-164664
(P2014-164664A)
(43)公開日　平成26年9月8日(2014.9.8)

| (51)Int.Cl. | | F I | | | テーマコード（参考） |
|---|---|---|---|---|---|
| G06F 9/48 | (2006.01) | G06F | 9/46 | 457 | 5B045 |
| G06F 15/167 | (2006.01) | G06F | 15/167 | 610A | |

審査請求 未請求　請求項の数 10　ＯＬ　（全 22 頁）

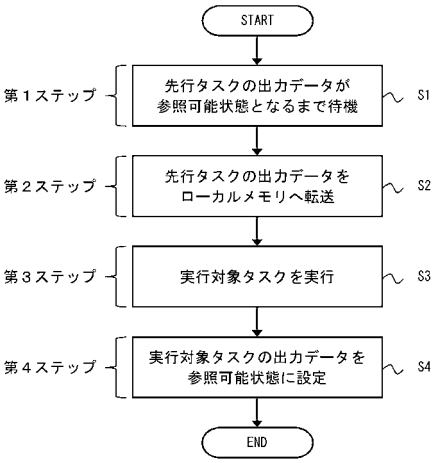
(21)出願番号　特願2013-37130(P2013-37130)
(22)出願日　平成25年2月27日(2013.2.27)

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号
(74)代理人　100103894
弁理士　家入　健
(72)発明者　久村　孝寛
東京都港区芝五丁目７番１号　日本電気株式会社内

Ｆターム(参考)　5B045 BB12　BB28　BB32　DD01　GG11

(54)【発明の名称】タスク並列処理方法、装置及びプログラム

(57)【要約】
【課題】プロセッサ間で共有するデータへのアクセス時間を短縮すること。
【解決手段】タスク並列処理方法は、第１プロセッサに割り当てられた第１タスクの先行タスク（第２タスク）が第１プロセッサ以外に割り当てられている場合に、第２タスクの出力データが第１プロセッサにより参照可能となるまで、第１タスクの実行を待機し、第２タスクの出力データが第１プロセッサにより参照可能となった後、出力データの格納元から出力データを取得して第１プロセッサが有する第１ローカルメモリへ格納し、第１ローカルメモリに格納された出力データを参照して、第１プロセッサが第１タスクを実行し、第１タスクの出力データを第１ローカルメモリに格納し、第１タスクの後続タスク（第３タスク）が第１プロセッサ以外のプロセッサに割り当てられている場合に、第１タスクの出力データを第１プロセッサ以外のプロセッサから参照可能な状態とする。
【選択図】図３



【特許請求の範囲】
【請求項１】
第１プロセッサに割り当てられた第１タスクにおける先行タスクである第２タスクが当該第１プロセッサ以外に割り当てられている場合に、当該第２タスクによる出力データが当該第１プロセッサにより参照可能となるまで、当該第１タスクの実行を待機し、
前記第２タスクによる出力データが前記第１プロセッサにより参照可能となった後、当該出力データの格納元から当該出力データを取得して前記第１プロセッサが有する第１ローカルメモリへ格納し、
前記第１ローカルメモリに格納された出力データを参照して、当該第１プロセッサが前記第１タスクを実行し、当該第１タスクによる出力データを当該第１ローカルメモリに格納し、
前記第１タスクにおける後続タスクである第３タスクが当該第１プロセッサ以外のプロセッサに割り当てられている場合に、前記第１タスクによる出力データを当該第１プロセッサ以外のプロセッサから参照可能な状態とする
タスク並列処理方法。
【請求項２】
前記第１プロセッサにおいて前記第１タスクの後に実行される第４タスクにおける先行タスクが前記第２タスクである場合、前記第２タスクの出力データの格納元から当該出力データを取得せずに、当該第１タスクの実行前に前記第１ローカルメモリに格納された前記第２タスクの出力データを参照して、当該第１プロセッサが当該第４タスクを実行する
請求項１に記載のタスク並列処理方法。
【請求項３】
前記第２タスクによる出力データの格納元は、共有メモリであり、
前記第１タスクの実行後、前記第１プロセッサが前記第１タスクによる出力データを前記第１ローカルメモリから読み出して、前記共有メモリへ格納する
請求項１又は２に記載のタスク並列処理方法。
【請求項４】
前記第２タスクによる出力データの格納元は、当該第２タスクを実行する第２プロセッサが有する第２ローカルメモリであり、
前記第２プロセッサが前記第２タスクの実行後に前記第２タスクの出力データを前記第２プロセッサ以外に対して参照可能とし、
前記第１プロセッサが前記第１タスクの実行後に前記第１タスクの出力データを前記第１プロセッサ以外に対して参照可能する
請求項１又は２に記載のタスク並列処理方法。
【請求項５】
前記第１タスク、前記第２タスク及び前記第３タスクは、タスクフローグラフとして表現可能なプログラムとして実装されていることを特徴とする請求項１乃至４のいずれか１項に記載のタスク並列処理方法。
【請求項６】
ローカルメモリを有する複数のプロセッサを備え、
前記複数のプロセッサのうち第１プロセッサは、
第１ローカルメモリを有し、
当該第１プロセッサに割り当てられた第１タスクにおける先行タスクである第２タスクが当該第１プロセッサ以外に割り当てられている場合に、当該第２タスクによる出力データが参照可能となるまで、当該第１タスクの実行を待機し、
前記第２タスクによる出力データが参照可能となった後、当該出力データの格納元から当該出力データを取得して前記第１ローカルメモリへ格納し、
前記第１ローカルメモリに格納された出力データを参照して、前記第１タスクを実行し、当該第１タスクによる出力データを当該第１ローカルメモリに格納し、
前記第１タスクにおける後続タスクである第３タスクが当該第１プロセッサ以外のプロ

セッサに割り当てられている場合に、前記第１タスクによる出力データを当該第１プロセッサ以外のプロセッサから参照可能な状態とする
タスク並列処理装置。
【請求項７】
前記第１プロセッサは、
当該第１プロセッサにおいて前記第１タスクの後に実行される第４タスクにおける先行タスクが前記第２タスクである場合、前記第２タスクの出力データの格納元から当該出力データを取得せずに、当該第１タスクの実行前に前記第１ローカルメモリに格納された前記第２タスクの出力データを参照して、当該第１プロセッサが当該第４タスクを実行する
請求項６に記載のタスク並列処理装置。
【請求項８】
前記第２タスクによる出力データの格納元は、共有メモリであり、
前記第１プロセッサは、
前記第１タスクの実行後、前記第１プロセッサが前記第１タスクによる出力データを前記第１ローカルメモリから読み出して、前記共有メモリへ格納する
請求項６又は７に記載のタスク並列処理装置。
【請求項９】
前記第２タスクによる出力データの格納元は、当該第２タスクを実行する第２プロセッサが有する第２ローカルメモリであり、
前記第２プロセッサは、前記第２タスクの実行後に前記第２タスクの出力データを前記第２プロセッサ以外に対して参照可能とし、
前記第１プロセッサは、前記第１タスクの実行後に前記第１タスクの出力データを前記第１プロセッサ以外に対して参照可能する
請求項６又は７に記載のタスク並列処理装置。
【請求項１０】
第１プロセッサに割り当てられた第１タスクにおける先行タスクである第２タスクが当該第１プロセッサ以外に割り当てられている場合に、当該第２タスクによる出力データが当該第１プロセッサにより参照可能となるまで、当該第１タスクの実行を待機する処理と、
前記第２タスクによる出力データが前記第１プロセッサにより参照可能となった後、当該出力データの格納元から当該出力データを取得して前記第１プロセッサが有する第１ローカルメモリへ格納する処理と、
前記第１ローカルメモリに格納された出力データを参照して、当該第１プロセッサが前記第１タスクを実行し、当該第１タスクによる出力データを当該第１ローカルメモリに格納する処理と、
前記第１タスクにおける後続タスクである第３タスクが当該第１プロセッサ以外のプロセッサに割り当てられている場合に、前記第１タスクによる出力データを当該第１プロセッサ以外のプロセッサから参照可能な状態とする処理と、
をコンピュータに実行させるタスク並列処理プログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、タスク並列処理方法、装置及びプログラムに関し、特に、有線回線及び無線回線を用いて受信した信号を処理するタスク並列処理方法、装置及びプログラムに関する。
【背景技術】
【０００２】
複数のプロセッサが協調して並列処理を行うためには、複数のプロセッサは何らかの手段でデータを共有する必要がある。例えば、ひとつのＬＳＩ（Ｌａｒｇｅ　Ｓｃａｌｅ　Ｉｎｔｅｇｒａｔｉｏｎ）に複数のプロセッサが搭載されたマルチコアプロセッサにおい

て、一般的なデータ共有手段は、複数のプロセッサからアクセス可能な共有メモリを用いることである。

【０００３】

複数のプロセッサからアクセス可能な共有メモリは、通常、アクセス調停のために、アクセスに時間がかかる。そこで、各プロセッサに高速なメモリを配置するのが一般的である。各プロセッサの高速なメモリの実現手段としては、ローカルメモリ又はキャッシュ（メモリ）のいずれかが挙げられる。

【０００４】

キャッシュは、共有メモリの部分的なコピー（以下、「部分的コピー」という。）を保存しておく一時記憶手段であり、キャッシュの動作はキャッシュ自身によって自動的に制御される。一方、ローカルメモリは単なる一時記憶手段である。

【０００５】

マルチコアプロセッサにおいて、各プロセッサがキャッシュをもつ場合には、複数のキャッシュのデータの整合性（コヒーレント）を保つ必要がある。コヒーレントを維持する手段としては、キャッシュ自身にそのような機能を追加するハードウェア的な方法と、ソフトウェア的な方法と、二つが考えられる。一般的にはハードウェア的な方法が使われることが多い。ハードウェア的な方法はキャッシュの回路規模を大きくし、キャッシュの消費電力を増やすというデメリットがある。

【０００６】

一方、ソフトウェア的な方法は、そのデメリットを回避することができるものの、各プロセッサは共有データにアクセスする際に所定のソフトウェア的手順を順守する必要がある。この所定のソフトウェア的手順は、マルチコアプロセッサ上で動作するソフトウェア全体の構成にも影響を与えるものである。ソフトウェア的な方法のひとつの例が特許文献５に報告されている。

【０００７】

キャッシュは、共有メモリの部分的コピーを自動的に管理するので、非常に便利な一時記憶手段である。しかしながら、プロセッサがデータをキャッシュにリード要求してから、そのデータをキャッシュがプロセッサへ渡すまでにかかる時間は、そのデータがキャッシュに存在するか否かで、大きく変わる。つまり、キャッシュのデータ読み出しにかかる時間はばらつく。

【０００８】

そのため、キャッシュのデータ読み出し時間のばらつきを回避するために、ローカルメモリを積極的に使う、という分野がある。特に、リアルタイム性を重視するような分野で、ローカルメモリをもつマルチコアプロセッサが使われる。

【０００９】

ローカルメモリはキャッシュと異なり、ローカルメモリのデータ読み出しにかかる時間は短く一定である。但し、ローカルメモリの容量は限られているため、マルチコアプロセッサで動作するソフトウェアを設計する開発者が、ローカルメモリにどのようなデータを配置するかを厳密に設計する必要がある。

【００１０】

ローカルメモリにどのようなデータを配置するかは、マルチコアプロセッサでどのように並列処理を行うかという問題と密接にかかわっている。そのため、ソフトウェア開発者が、並列処理とデータ配置とを合わせて検討してきた。

【００１１】

従来から、複数のプロセッサがアクセスする可能性があるデータを共有メモリに配置し、ひとつのプロセッサだけがアクセスするデータをローカルメモリに配置する、という手法が一般的に使われている（例えば、図１１）。この手法はデータ配置を固定的に決めるものである。この方法は非常にシンプルで考えやすいが、デメリットがある。それは、共有メモリに配置されたデータへのアクセスには時間がかかる、というデメリットである。

【００１２】

このデメリットを回避するために、共有メモリの部分的コピーをローカルメモリに置くという手法が考えられる。ところが、この場合には、部分的コピーの管理が非常に煩雑になる、という別の問題が発生する。この手法を実行するには、マルチコアプロセッサでの並列処理と、各プロセッサのローカルメモリ上の部分的コピーの管理と、マルチプロセッサ間における部分的コピーの整合性維持と、を合わせて考える必要がある。部分的コピーの整合性維持とは、複数プロセッサのローカルメモリに同じ部分的コピーを置いて、それらをそれぞれ異なる値で書き換えることが無いようにすること、である。

【００１３】

つまり、並列処理において、ローカルメモリ上に部分的コピーを置くためには、ローカルメモリ上の部分的コピーの管理と整合性維持と並列処理とを統合的に扱う必要がある。しかしながら、これらを統合的に扱うことは、プログラマにとって大きな負担であり、非常に工数がかかるため、現実的ではなかった。

【００１４】

尚、関連技術として、以下の特許文献１～４がある。特許文献１には、処理性能を落とさずに消費電力を低減し、また、実時間処理を要求するプログラムを実行するに際しても、時間制約を遵守しつつ、電力を低減することを目的とするマルチプロセッサシステム及びコンパイラに関する技術が開示されている。

【００１５】

特許文献２には、メモリへデータを効率よく配置するためのメモリ管理方法に関する技術が開示されている。特に、特許文献１にかかるプロセッサは、メモリの記憶領域を複数の異なるサイズのブロックに分割し、タスクの実行時に使用されるデータに適合するサイズのブロックを選択し、選択されたブロックに、タスクの実行時に使用されるデータを格納するものである。

【００１６】

特許文献３には、共有メモリと複数のプロセッサとを有し、各プロセッサがローカルメモリを有する処理システムに関する技術が開示されている。特許文献３にかかるプロセッサは、共有メモリからプログラムの実行等に関連するローカルメモリへデータをコピーする。

【００１７】

特許文献４には、プロセッサ間排他制御機能を有しない複数個のプロセッサと共有メモリとを装備したマルチプロセッサシステムに関する技術が開示されている。特許文献４にかかるマルチプロセッサシステムでは、あるプロセッサが他系プロセッサからの処理要求を受けると、送信バッファ内のデータに基づいた処理を行い、その後、処理要求元のプロセッサに処理終了を通知する。

【先行技術文献】

【特許文献】

【００１８】

【特許文献１】特開２００６－２９３７６８号公報
【特許文献２】特開２００８－２１７１３４号公報
【特許文献３】特開２００６－２２１６３８号公報
【特許文献４】特開２０００－２３５５５３号公報
【特許文献５】国際公開第２００９／０５７７６２号

【発明の概要】

【発明が解決しようとする課題】

【００１９】

以上説明したように、共有メモリとローカルメモリとを備えるマルチコア・プロセッサ上での並列処理において、「複数のプロセッサからアクセスされる可能性がある共有データを共有メモリに配置する」という従来から一般的に使われている共有データ配置方法は、「共有メモリに配置されたデータへのアクセスには時間がかかる」、という課題を持っている。この課題は、共有データの数が多い場合に並列処理全体の処理時間が増加する、

という課題につながる。その理由は、共有メモリに格納された変数等の共有データに対しては、タスクの実行中に複数回のアクセスが発生し得るからである。尚、特許文献１乃至５にはこれらを解決する手段が開示されていない。

【００２０】

本発明は、上述した問題点を考慮してなされたものであり、プロセッサ間で共有するデータへのアクセス時間を短縮するためのタスク並列処理方法、装置及びプログラムを提供することを目的とする。

【課題を解決するための手段】

【００２１】

本発明の第１の態様にかかるタスク並列処理方法は、

第１プロセッサに割り当てられた第１タスクにおける先行タスクである第２タスクが当該第１プロセッサ以外に割り当てられている場合に、当該第２タスクによる出力データが当該第１プロセッサにより参照可能となるまで、当該第１タスクの実行を待機し、

前記第２タスクによる出力データが前記第１プロセッサにより参照可能となった後、当該出力データの格納元から当該出力データを取得して前記第１プロセッサが有する第１ローカルメモリへ格納し、

前記第１ローカルメモリに格納された出力データを参照して、当該第１プロセッサが前記第１タスクを実行し、当該第１タスクによる出力データを当該第１ローカルメモリに格納し、

前記第１タスクにおける後続タスクである第３タスクが当該第１プロセッサ以外のプロセッサに割り当てられている場合に、前記第１タスクによる出力データを当該第１プロセッサ以外のプロセッサから参照可能な状態とする。

【００２２】

本発明の第２の態様にかかるタスク並列処理装置は、

ローカルメモリを有する複数のプロセッサを備え、

前記複数のプロセッサのうち第１プロセッサは、

第１ローカルメモリを有し、

当該第１プロセッサに割り当てられた第１タスクにおける先行タスクである第２タスクが当該第１プロセッサ以外に割り当てられている場合に、当該第２タスクによる出力データが参照可能となるまで、当該第１タスクの実行を待機し、

前記第２タスクによる出力データが参照可能となった後、当該出力データの格納元から当該出力データを取得して前記第１ローカルメモリへ格納し、

前記第１ローカルメモリに格納された出力データを参照して、前記第１タスクを実行し、当該第１タスクによる出力データを当該第１ローカルメモリに格納し、

前記第１タスクにおける後続タスクである第３タスクが当該第１プロセッサ以外のプロセッサに割り当てられている場合に、前記第１タスクによる出力データを当該第１プロセッサ以外のプロセッサから参照可能な状態とする。

【００２３】

本発明の第３の態様にかかるタスク並列処理プログラムは、

第１プロセッサに割り当てられた第１タスクにおける先行タスクである第２タスクが当該第１プロセッサ以外に割り当てられている場合に、当該第２タスクによる出力データが当該第１プロセッサにより参照可能となるまで、当該第１タスクの実行を待機する処理と、

前記第２タスクによる出力データが前記第１プロセッサにより参照可能となった後、当該出力データの格納元から当該出力データを取得して前記第１プロセッサが有する第１ローカルメモリへ格納する処理と、

前記第１ローカルメモリに格納された出力データを参照して、当該第１プロセッサが前記第１タスクを実行し、当該第１タスクによる出力データを当該第１ローカルメモリに格納する処理と、

前記第１タスクにおける後続タスクである第３タスクが当該第１プロセッサ以外のプロ

セッサに割り当てられている場合に、前記第１タスクによる出力データを当該第１プロセッサ以外のプロセッサから参照可能な状態とする処理と、
をコンピュータに実行させる。
【発明の効果】
【００２４】
本発明により、プロセッサ間で共有するデータへのアクセス時間を短縮するためのタスク並列処理方法、装置及びプログラムを提供することができる。
【図面の簡単な説明】
【００２５】
【図１】本発明の実施形態にかかるマルチコアプロセッサの構成及びデータ配置の例を示すブロック図である。
【図２】本発明の実施形態で対象とするタスクフローグラフの例を示す図である。
【図３】本発明の実施形態にかかるタスク並列処理方法の処理の流れを示すフローチャートである。
【図４】本発明の実施形態にかかるプロセッサの処理の流れを示すフローチャートである。
【図５】本発明の実施形態の具体例１にかかるタスク割当てとデータ配置の例を示す図である。
【図６】本発明の実施形態の具体例１にかかるタスク処理の流れを示すフローチャートである。
【図７】本発明の実施形態の具体例１にかかるタスク処理の流れを示すフローチャートである。
【図８】本発明の実施形態の具体例２にかかるタスク割当てとデータ配置の例を示す図である。
【図９】本発明の実施形態の具体例２にかかるタスク処理の流れを示すフローチャートである。
【図１０】本発明の実施形態の具体例２にかかるタスク処理の流れを示すフローチャートである。
【図１１】関連技術にかかるマルチコアプロセッサにおけるデータ配置の例を示すブロック図である。
【発明を実施するための形態】
【００２６】
以下では、本発明を適用した具体的な実施の形態について、図面を参照しながら詳細に説明する。各図面において、同一要素には同一の符号が付されており、説明の明確化のため、必要に応じて重複説明は省略する。
【００２７】
まず、本発明の実施形態の説明をするにあたり、本実施形態が処理対象とする並列処理モデル及びトポロジカルソートに関して説明する。
【００２８】
＜本発明が対象とする並列処理モデル＞
本発明の模範的な実施形態が扱う並列処理について説明する。本発明の模範的な実施形態は、非循環型有向グラフ（ＤＡＧ：ｄｉｒｅｃｔｅｄ　ａｃｙｃｌｉｃ　ｇｒａｐｈ）に属するタスクフローグラフで表現可能な並列処理を扱う。以降では、特に限定しない場合、「タスクフローグラフ」は非循環型有向グラフに属するものとする。
【００２９】
タスクフローグラフは、並列処理を構成する計算処理（タスク）のデータ依存関係を表すグラフ構造である。並列処理を構成する計算処理をタスクと呼ぶことにする。タスクフローグラフにおいて、ノードはタスクを表し、ノード間のエッジはタスク間のデータ依存関係を表す。タスクフローグラフの例を図２に示す。図２には６個のタスクＴ１～Ｔ６が存在する。タスクフローグラフを使うことにより、対象となる並列処理を構成するタスク

のデータ依存関係を簡単に表現することができる。ここでいう「データ依存関係」とは、実行対象のタスクが使用するデータに関してデータを供給するタスクの役割を果たすデータ供給タスクと、当該データを使用するタスクの役割を果たすデータ使用タスクとの間に存在する実行順序（先行、後続）等の依存関係を示すものである。

【００３０】

例えば、タスクＴ１の計算結果（データＡ）をタスクＴ２が使用する場合を想定する。この場合には、データＡに関して、タスクＴ１はデータ供給タスクであり、タスクＴ２はデータ使用タスクである。そして、このような場合に、データ供給タスクであるタスクＴ１からデータ使用タスクであるタスクＴ２に対してデータ依存関係が存在すると、言える。そして、タスクＴ１からタスクＴ２に対してデータ依存関係が存在するなら、タスクＴ２はタスクＴ１の後で実行されなければならない。この場合、タスクＴ２はタスクＴ１における先行タスクであり、タスクＴ１はタスクＴ２における後続タスクである。

【００３１】

さらに、本発明の模範的な実施形態が扱う並列処理では、タスクフローグラフの各ノードには、トポロジカルソートにもとづいた順序番号が付与されているものとする。トポロジカルソートについては後述する。

【００３２】

加えて、本発明の模範的な実施形態が扱う並列処理では、タスクフローグラフで表現可能な計算処理を複数のプロセッサ又は複数のコア（以下、プロセッサとコアとを含めて「プロセッサ」という）で並列に計算するものとし、タスクフローグラフの各タスクをどのプロセッサが実行するかが予め決定されているものとする。どのプロセッサがどのタスクを実行するかという情報は、タスクフローグラフの付属情報とする。一般的には、プロセッサの数よりもタスクの数が多いので、ひとつのプロセッサが複数のタスクを実行することになる。前述のとおり、タスクにはトポロジカルオーダにもとづいた順序番号が付与されているので、本発明の模範的な実施形態は、この順序番号をタスクの実行順序とみなして、各プロセッサに、順序番号が小さいタスクから順番にタスクを実行させる。本実施形態は、このようなタスクフローグラフで表現可能な計算処理の並列処理を扱う。

【００３３】

＜トポロジカルソートについて＞

続いて、トポロジカルソートについて説明する。トポロジカルソートとは、非循環有向グラフの各ノードを順序付けして、どのノードもその出力エッジの先のノードよりも前にくるように並べることである。トポロジカルソートの典型的な利用例は、タスクのスケジューリングや、コンパイラにおける命令スケジューリングである。つまり、データ依存関係をもつタスクセットを非循環有向グラフで表現し、その非循環有向グラフをトポロジカルソートすることによって、タスクを実行すべき順序がわかることになる。トポロジカルソートで得られる順序は、ひとつだけとは限らず、複数の正しい順序が存在しうる。

【００３４】

＜一般的なデータ配置方法の課題＞

続いて、本発明の実施形態の説明の前に、関連技術にかかる一般的なデータ配置方法の課題をまとめておく。及びローカルメモリ１１～１４を有するＣＰＵ２１～２４と、共有メモリ３０とを備えたマルチコアプロセッサ９００における、一般的なデータ配置方法の例を図１１に示す。図１１に示す関連技術にかかる方法は、データを非共有か共有かで分類し、共有データ９０１を共有メモリ３０に配置し、非共有データ９１１、９２１、９３１及び９４１をローカルメモリ１１、１２，１３及び１４にそれぞれ配置する。ここで、共有メモリ３０とローカルメモリ１１～１４に配置されるデータには重複がない。つまり、共有データ９０１は複数のプロセッサが使用するデータであり、非共有データ９１１はＣＰＵ２１のみが使用するデータであり、非共有データ９２１はＣＰＵ２２のみが使用するデータであり、非共有データ９３１はＣＰＵ２３のみが使用するデータであり、非共有データ９４１はＣＰＵ２４のみが使用するデータである。

【００３５】

共有メモリは、ローカルメモリに比べて、一般的にはデータ読み書きに時間がかかる。したがって、共有メモリに多くの共有データを配置することは、共有データを使用する並列処理全体の計算時間を増加させることになる。さらに、関連技術にかかる方法では、二つ以上のプロセッサが使用する共有データは全て共有メモリに配置されることになり、並列動作させる処理の数を増やすためにプロセッサ数を増やすと、共有メモリにアクセスするプロセッサの数が増えるので、共有メモリへのアクセス回数が増えることになる。つまり、関連技術にかかる方法は、アクセス時間が大きい共有メモリに対してプロセッサが何度もアクセスする、という課題がある。

【００３６】

＜本発明の模範的な実施形態＞

本発明の模範的な実施形態について説明する。本実施形態は、ローカルメモリを備える複数のプロセッサで構成されたマルチコア・プロセッサ上での並列処理を提供する並列情報処理方法および装置である。図１は、本発明の実施形態にかかるマルチコアプロセッサ１００の構成及びデータ配置の例を示すブロック図である。マルチコアプロセッサ１００は、ローカルメモリ１１～１４と、ＣＰＵ２１～２４と、共有メモリ３０とを備える。各プロセッサ２１～２４のローカルメモリ１１～１４には、各プロセッサが割り当てられたタスクが参照するデータ（割当タスク参照用データ１１１、１２１、１３１及び１４１）を配置するとともに、全てのプロセッサからアクセス可能な共有メモリ３０に、複数プロセッサで共有すべきデータ（マルチタスク参照用データ３０１）を配置する。

【００３７】

本実施形態は、タスクフローグラフとして表現可能なプログラムの任意の第１タスクの実行に関して、第１タスクを実行する前に、タスクフローグラフにおいて第１タスクへのエッジをもち、なおかつ第１タスクを実行する第１プロセッサ以外が実行する第２タスク群の出力データが参照可能状態となることを待ち（第１ステップ）、第１ステップの後で、第２タスク群の出力データを第１プロセッサのローカルメモリにコピーし（第２ステップ）、第１タスクを実行してその計算結果を第１プロセッサのローカルメモリに格納し（第３ステップ）、タスクフローグラフにおいて第１タスクからのエッジをもちなおかつ第１プロセッサ以外が実行する第３タスク群が存在する場合に、第１タスクの出力データを第３タスク群が参照可能な状態とする（第４ステップ）、ことを特徴とする、並列情報処理装置、及び方法である。尚、第１タスクに加え、第２タスク及び第３タスクも、タスクフローグラフとして表現可能なプログラムとして実装されているものとする。

【００３８】

言い換えると、本実施形態は、第１プロセッサに割り当てられた第１タスクにおける先行タスクである第２タスクが当該第１プロセッサ以外に割り当てられている場合に、当該第２タスクによる出力データが当該第１プロセッサにより参照可能となるまで、当該第１タスクの実行を待機し、第２タスクによる出力データが第１プロセッサにより参照可能となった後、当該出力データの格納元から当該出力データを取得して第１プロセッサが有する第１ローカルメモリへ格納し、第１ローカルメモリに格納された出力データを参照して、当該第１プロセッサが第１タスクを実行し、当該第１タスクによる出力データを当該第１ローカルメモリに格納し、第１タスクにおける後続タスクである第３タスクが当該第１プロセッサ以外のプロセッサに割り当てられている場合に、第１タスクによる出力データを当該第１プロセッサ以外のプロセッサから参照可能な状態とする、タスク並列処理方法、方法及びプログラムである。

【００３９】

マルチコアプロセッサ１００は、タスクフローグラフとして表現可能なプログラムの任意のタスクを第１タスクとして、図３に示すように、上記の第１ステップ（Ｓ１）から第４ステップ（Ｓ４）を行う。

【００４０】

第１ステップは、第１タスクを実行可能な状態になるまで待ち合せるという処理である。ここで、第２タスク群は、第１プロセッサ以外で実行され、なおかつ第１タスクへ入力

データを供給するタスク群である。そして、第２タスク群の出力データを第１タスクが参照可能な状態であることが確認されると、第１タスクを実行可能な状態となる。もし第２タスク群が存在しないならば、出力データを参照すべき第２タスク群は存在しないので、待ち合わせは不要となり、第１タスクは実行可能な状態となる。例えば、タスクフローグラフにおいて第１タスクへのエッジをもつタスクが全て第１プロセッサ（第１タスクを実行するプロセッサ）で実行されるなら、第２タスク群は存在しないことになる。また、第２タスク群は、１個以上のタスクであればよい。

【００４１】

上述のことは、次のように言い換えることもできる。すなわち、第１タスクを実行するプロセッサとして第１プロセッサが予め割り当てられているものとする。そして、第２タスクを実行するプロセッサとして第１プロセッサ以外のプロセッサが予め割り当てられているものとする。また、第２タスクは第１タスクにおける先行タスクであるものとする。この場合、第１プロセッサは、第２タスクによる出力データが当該第１プロセッサにより参照可能となるまで、当該第１タスクの実行を待機する（Ｓ１）。

【００４２】

第２ステップは、第１タスクへの入力データとなる第２タスク群の出力データを、第１プロセッサのローカルメモリへコピーする処理である。第２タスク群の出力データは第１プロセッサから参照可能な場所に格納されているものの、第１プロセッサにとってはそれらの場所よりもローカルメモリのほうがアクセス時間が早い。したがって、第２タスク群の出力データを第１プロセッサのローカルメモリにコピーすることで、第１タスクを実行する第１プロセッサは第１タスクの入力データに高速にアクセスすることが可能になる。第２ステップにおいて、第２タスク群の出力データが格納される場所は、共有メモリであってもよいし、他プロセッサからアクセス可能な各プロセッサのローカルメモリであってもよい。もし第２タスク群が存在しないならば、コピーすべき出力データは存在しないので、コピー処理は不要となる。

【００４３】

上述のことは、次のように言い換えることもできる。すなわち、第１プロセッサは、第２タスクによる出力データが自身により参照可能となった後、当該出力データの格納元から当該出力データを取得して自身が有する第１ローカルメモリへ格納する（Ｓ２）。

【００４４】

また、コピーすべき出力データが、第１タスクを実行するプロセッサのローカルメモリに存在することが明らかである場合にも、コピー処理は不要である。例えば、或るプロセッサに割り当てられた複数の第１タスクが同じタスクを第２タスク群としてもつ場合には、第２ステップにおいてコピーすべき出力データが同じなので、最初の第２ステップでコピーしておけば、他の第２ステップでは、コピーすべき出力データがローカルメモリに存在することは明らかなので、最初の第２ステップ以降ではコピー処理は不要となる。

【００４５】

上述のことは、次のように言い換えることもできる。すなわち、第１タスクの後に実行される第４タスクも第１プロセッサに割り当てられているものとする。そして、第４タスクにおける先行タスクが第２タスクであるものとする。この場合、第１プロセッサは、第２タスクの出力データの格納元から当該出力データを取得せずに、第１タスクの実行前に第１ローカルメモリに格納された第２タスクの出力データを参照して、第４タスクを実行する。

【００４６】

第３ステップは、第１タスクを実行する処理である。第１ステップで実行可能な状態であることを確認し、第２ステップで入力データを準備したので、第１タスクの実行準備が整う。第３ステップにおいて、第１プロセッサは、第１プロセッサのローカルメモリに存在する入力データを使用して第１タスクを実行し、第１タスクの出力データは第１プロセッサのローカルメモリに格納される。

【００４７】

上述のことは、次のように言い換えることもできる。すなわち、第１プロセッサは、第１ローカルメモリに格納された出力データを参照して、第１タスクを実行し（Ｓ３）、第１タスクによる出力データを第１ローカルメモリに格納する。

【００４８】

第４ステップは、第１タスクの出力データを入力データとして使用する第３タスク群のために、第１プロセッサのローカルメモリに格納されている第１タスクの出力データが参照可能な状態にする処理である。この処理には、第１タスクの出力データが第１プロセッサ以外から参照可能で無い場合に第１タスクの出力データを第１プロセッサ以外が参照可能な場所（例えば共有メモリ）にコピーする処理と、第１タスクの実行が完了しその出力データが参照可能な状態であることを記録する処理と、を含む。このコピー処理は、第２ステップのコピー処理と対（つい）になる処理である。もし第２ステップのコピー元が共有メモリであるならば、第４ステップのコピー先も共有メモリとする。もし第２ステップのコピー元が或るプロセッサのローカルメモリであるならば、各プロセッサは互いのローカルメモリをアクセス可能なので、第４ステップのコピー処理は不要である。この一連の処理によって、第１タスクの出力データが参照可能であることを第３タスク群の実行担当のプロセッサが知ることができる。もし第３タスク群が存在しないならば、第１プロセッサ以外は第１タスクの出力データを使わないため、第４ステップのコピー処理は不要であり、第１タスクの出力データが参照可能である否かを知るべきタスクが存在しないため、第１タスクの出力データの参照可能状態を記録する必要もない。

【００４９】

上述のことは、次のように言い換えることもできる。すなわち、第１プロセッサは、第１タスクにおける後続タスクである第３タスクが当該第１プロセッサ以外のプロセッサに割り当てられている場合に、第１タスクによる出力データを当該第１プロセッサ以外のプロセッサから参照可能な状態とするように設定する（Ｓ４）。

【００５０】

続いて、上述の第１ステップから第４ステップを、各プロセッサの視点で説明する。図４は、本発明の実施形態にかかるプロセッサの処理の流れを示すフローチャートである。まず、各プロセッサは、自身に割り当てられたタスクを順序番号の小さいものから順に選択する（Ｓ２１）。そして、各プロセッサは、選択したタスクについて上述した第１ステップから第４ステップの順序で実行する（Ｓ２２）。その後、自身に割り当てられた全てのタスクの処理が終了したか否かを判定する（Ｓ２３）。そのため、各プロセッサは、自身に実行が割り当てられているタスクの数だけ、第１ステップから第４ステップを繰り返す。

【００５１】

さらに、各プロセッサは、トポロジカルオーダにもとづいた順序番号にしたがってタスクを順番に実行する。この実行順序は、タスクフローグラフのデータ依存関係にもとづいて予め決められた実行順序なので、ひとつのプロセッサに割り当てられたタスク同士でタスクの実行完了を確認することは不要である。また、ひとつのプロセッサに割り当てられたタスクの出力データは同じローカルメモリに格納されるため、ひとつのプロセッサに割り当てられたタスク同士で共有メモリを介して出力データを受け渡しすることも不要である。したがって、上述のように、第２タスク群が存在しない場合には、第１ステップの待合せが不要となるとともに、第２ステップのコピー処理が不要となる。さらに、第３タスク群が存在しない場合には、第４ステップのコピー処理及び実行完了記録処理が不要となる。

【００５２】

＜本発明の実施形態の具体例１＞

実施形態の具体例１について説明する。具体例１は、ローカルメモリを備えた複数のプロセッサと、プロセッサ間のデータ共有のための共有メモリと、を含むマルチコア・プロセッサ上での並列処理を提供する並列情報処理装置、及び方法である。

【００５３】

具体例１は、タスクフローグラフとして表現可能なプログラムの任意のタスクを第１タスクとして、前述の第１ステップから第４ステップを行う。具体例１では、共有メモリを使ってプロセッサ間でデータを共有する。したがって、第２ステップのコピー元は共有メモリとなり、第４ステップのコピー先も共有メモリとなる。

【００５４】

第１ステップにおいて、具体例１は、第１タスクを実行可能な状態（第２タスク群の出力データが参照可能な状態）になるまで待合わせる。続いて、第２ステップにおいて、具体例１は、第１タスクへの入力データとなる第２タスク群の出力データを、共有メモリから第１プロセッサのローカルメモリへコピーする。続いて、第３ステップにおいて、具体例１は、第１タスクを実行し、その出力データをローカルメモリに格納する。続いて、第４ステップにおいて、具体例１は、第１タスクの出力データを入力データとして使用する第３タスク群が存在するならば、第１プロセッサのローカルメモリに格納されている第１タスクの出力データを共有メモリへコピーするとともに、第１タスクの出力データが参照可能な状態であることを共有メモリに記録する。

【００５５】

続いて、上述の第１ステップから第４ステップを、各プロセッサの視点で説明する。前述のとおり、本発明の模範的な実施形態は、各プロセッサは順序番号が小さいタスクから順番に実行するものとしている。つまり、具体例１の各プロセッサは、自身に実行が割り当てられているタスクの数だけ、第１ステップから第４ステップを繰り返す。

【００５６】

さらに、各プロセッサは、トポロジカルオーダにもとづいた順序番号にしたがってタスクを順番に実行する。この実行順序は、タスクフローグラフのデータ依存関係に基づいて予め決められた実行順序なので、ひとつのプロセッサに割り当てられたタスク同士でタスクの実行完了を確認することは不要である。また、ひとつのプロセッサに割り当てられたタスクの出力データは同じローカルメモリに格納されるため、ひとつのプロセッサに割り当てられたタスク同士で共有メモリを介して出力データを受け渡しすることは不要である。したがって、上述のように、第２タスク群が存在しない場合には、第１ステップの待合せが不要となるとともに、第２ステップのコピー処理が不要となる。さらに、第３タスク群が存在しない場合には、第４ステップのコピー処理および出力データ参照可能状態記録処理が不要となる。

【００５７】

＜具体例１の動作例＞

続いて、具体例１の動作例について図面を使って説明する。まず、図２は、具体例１が対象とするタスクフローグラフの例を示す図である。つまり、６個のタスクＴ１～Ｔ６が図２のようなデータ依存関係をもつものとする。具体的には、タスクＴ１は、先行タスクがなく、タスクＴ２及びＴ３を後続タスクとする。そのため、タスクＴ１は、外部データを入力とし、データＡを出力する。タスクＴ２は、タスクＴ１を先行タスクとし、タスクＴ４を後続タスクとする。そのため、タスクＴ２は、タスクＴ１により出力されたデータＡを入力とし、データＢを出力する。タスクＴ３は、タスクＴ１を先行タスクとし、タスクＴ４及びＴ５を後続タスクとする。そのため、タスクＴ３は、タスクＴ１により出力されたデータＡを入力とし、データＢを出力する。タスクＴ４は、タスクＴ２及びＴ３を先行タスクとし、タスクＴ６を後続タスクとする。そのため、タスクＴ４は、タスクＴ２により出力されたデータＢ及びタスクＴ３により出力されたデータＣを入力とし、データＤを出力する。タスクＴ５は、タスクＴ３を先行タスクとし、タスクＴ６を後続タスクとする。そのため、タスクＴ５は、タスクＴ３により出力されたデータＣを入力とし、データＥを出力する。タスクＴ６は、タスクＴ４及びＴ５を先行タスクとし、後続タスクがないものとする。そのため、タスクＴ６は、タスクＴ４により出力されたデータＤ及びタスクＴ５により出力されたデータＥを入力とし、データＦを出力する。

【００５８】

図５は、本発明の実施形態の具体例１にかかるマルチコアプロセッサ１００のタスク割

当てとデータ配置の例を示す図である。図５では、具体例１が備えるプロセッサの数を４として説明するが、プロセッサ数はこれに限定されない。図２の６個のタスクＴ１～Ｔ６を具体例１の４個のプロセッサに図５のように予め割当てておくものとする。つまり、タスクＴ１及びＴ２をＣＰＵ２１に、タスクＴ３及びＴ４をＣＰＵ２２に、タスクＴ５をＣＰＵ２３に、タスクＴ６をＣＰＵ２４に、それぞれ割り当てるものとする。このように割当てを決めたことにより、本発明の実施形態にかかる基本的な考え方「プロセッサのローカルメモリにタスクの入出力データを配置する」に基づいて、各タスクの入出力データの配置は図５のように決まることになる。つまり、ＣＰＵ２１のローカルメモリ１１にはタスクＴ１とタスクＴ２の入出力データである外部データ５１１、データＡ５１２及びデータＢ５１３を、ＣＰＵ２２のローカルメモリ１２にはタスクＴ３及びＴ４の入出力データであるデータＡ５２１、データＢ５２２、データＣ５２３及びデータＤ５２４を、ＣＰＵ２３のローカルメモリ１３にはタスクＴ５の入出力データであるデータＣ５３１及びデータＥ５３２を、ＣＰＵ２４のローカルメモリ１４にはタスクＴ６の入出力データであるデータＤ５４１、データＥ５４２及びデータＦ５４３を、それぞれ配置することになる。

【００５９】

そして、具体例１では、共有メモリを用いてプロセッサ間でデータを共有するので、相異なるプロセッサの間で実行されるタスクの間で受け渡すべきデータは、図５のように共有メモリ３０に配置されることになる。つまり、共有メモリ３０には、外部データ３１１、データＡ３１２、データＢ３１３、データＣ３１４、データＤ３１５、データＥ３１６及びデータＦ３１７が格納される。尚、外部データ５１１と外部データ３１１とは同一内容であり、データＡ５１２、データＡ５２１及びデータＡ３１２は同一内容であり、データＢ５１３、データＢ５２２及びデータＢ３１３は同一内容であり、データＣ５２３、データＣ５３１及びデータＣ３１４は同一内容であり、データＤ５２４、データＤ５４１及びデータＤ３１５は同一内容であり、データＥ５３２、データＥ５４２及びデータＥ３１６は同一内容であり、データＦ５４３及びデータＦ３１７は同一内容であるものとする。

【００６０】

次に、具体例１にかかるタスク処理（タスクＴ１からタスクＴ６を実行する様子）の流れを示すフローチャートを図６及び図７に示す。図６及び図７において、具体例１の４個のプロセッサ２１～２４は、トポロジカルオーダにもとづいた順序番号にしたがってタスクを順番に実行する。各タスクの実行は前述の第１ステップから第４ステップに基づいて行われる。タスクＴ１からタスクＴ６のタスクフローグラフ（図２）によると、どのタスクの出力データも使用しないで計算を行うのはタスクＴ１であるので、タスクＴ１だけが第１ステップの待合せが不要で、すぐに実行できる状態にあることがわかる。したがって、図６及び図７では、ＣＰＵ２１によるタスクＴ１の処理から動作が開始する。

【００６１】

ＣＰＵ２１は、まずタスクＴ１の第１ステップを行うが、上述の通りタスクＴ１については第１ステップの待合せが不要となるので、ＣＰＵ２１は、すぐに、タスクＴ１の第２ステップに移る。次に、ＣＰＵ２１は、タスクＴ１の第２ステップにおいて、タスクＴ１の入力データ（外部データ３１１）を共有メモリ３０からローカルメモリ１１へ（外部データ５１１として）コピーする（Ｓ３１０）。次に、ＣＰＵ２１は、タスクＴ１の第３ステップにおいて、タスクＴ１を実行し（Ｓ３１１）、タスクＴ１の出力データ（データＡ５１２）をローカルメモリ１１に書き込む。次に、ＣＰＵ２１は、タスクＴ１の第４ステップにおいて、前述の第３タスク群に相当するタスク（ＣＰＵ２２が実行するタスクＴ３）が存在するため、タスクＴ１の出力データ（データＡ５１２）をローカルメモリ１１から共有メモリ３０へ（データＡ３１２として）コピーする（Ｓ３１２）とともに、タスクＴ１の出力データが参照可能な状態になったというフラグ（完了フラグ）をセットする（Ｓ３１３）。完了フラグは共有メモリ３０上に置くものとする。ここまでで、タスクＴ１の第１ステップから第４ステップが終了する。

【００６２】

続いて、ＣＰＵ２１は、タスクＴ１の処理が終わると、順序番号による実行順序にもと

づいて、タスクＴ２の処理を行う。図２のタスクフローグラフによると、タスクＴ２はタスクＴ１の出力データだけを使用するので、タスクＴ２を前述の第１タスクとみなす場合に、前述の第２タスク群に相当するタスクが存在しない。第２タスク群は存在せず、タスクＴ１は既にＣＰＵ２１によって実行完了しているため、タスクＴ２の第１ステップは不要となり、ＣＰＵ２１は、すぐに、次のステップに移る。次に、ＣＰＵ２１は、タスクＴ２の第２ステップにおいて、第２タスク群に相当するタスクが存在しないので、コピーは不要となり、次のステップに移る。次に、ＣＰＵ２１は、タスクＴ２の第３ステップにおいて、タスクＴ２を実行し（Ｓ３１４）、タスクＴ２の出力データ（データＢ５１３）をローカルメモリ１１に書き込む。次に、ＣＰＵ２１は、タスクＴ２の第４ステップにおいて、前述の第３タスク群に相当するタスク（ＣＰＵ２２が実行するタスクＴ４）が存在するため、タスクＴ２の出力データ（データＢ５１３）をローカルメモリ１１から共有メモリ３０へ（データＢ３１３として）コピーする（Ｓ３１５）とともに、タスクＴ２の出力データが参照可能な状態になったというフラグ（完了フラグ）をセットする（Ｓ３１６）。ここまでで、タスクＴ２の第１ステップから４が終了する。

【００６３】

続いて、ＣＰＵ２２によるタスクＴ３の処理について説明する。ＣＰＵ２１がタスクＴ１とタスクＴ２を実行すると、タスクＴ１及びＴ２の出力データが参照可能な状態になる。図２のタスクフローグラフによると、それによって、ＣＰＵ２２は、タスクＴ３と（タスクＴ３の実行終了後）タスクＴ４を実行可能な状態となる。図２のタスクフローグラフによると、タスクＴ３を前述の第１タスクとみなす場合に、前述の第２タスク群に相当するタスクはタスクＴ１である。ＣＰＵ２２は、タスクＴ３の第１ステップにおいて、タスクＴ１の出力データが参照可能な状態になったというフラグ（完了フラグ）を参照して、タスクＴ３を実行可能な状態（タスクＴ１の出力データが参照可能な状態）になるまで待合わせる（Ｓ３２０）。そして、ＣＰＵ２１によるタスクＴ１の処理（第１ステップから第４ステップ）が完了した後は、タスクＴ１の出力データの参照可能フラグ（完了フラグ）がセットされている（Ｓ３１３）ので、ＣＰＵ２２はタスクＴ１の完了フラグを検出して（Ｓ３２１）待合せを終了し、次のステップに移る。次に、ＣＰＵ２２は、タスクＴ３の第２ステップにおいて、第２タスク群に相当するタスク（タスクＴ１）が存在するため、タスクＴ１の出力データ（データＡ３１２）を共有メモリ３０からＣＰＵ２２のローカルメモリ１２へ（データＡ５２１として）コピーする（Ｓ３２２）。次に、ＣＰＵ２２は、タスクＴ３の第３ステップにおいて、タスクＴ３を実行し（Ｓ３２３）、タスクＴ３の出力データ（データＣ５２３）をローカルメモリ１２に書き込む。次に、ＣＰＵ２２は、タスクＴ３の第４ステップにおいて、前述の第３タスク群に相当するタスク（ＣＰＵ２３が実行するタスクＴ５）が存在するため、タスクＴ３の出力データ（データＣ５２３）をローカルメモリ１２から共有メモリ３０へ（データＣ３１４として）コピーする（Ｓ３２４）とともに、タスクＴ３の出力データが参照可能な状態になったというフラグ（完了フラグ）をセットする（Ｓ３２５）。ここまでで、タスクＴ３の第１ステップから４が終了する。

【００６４】

続いて、ＣＰＵ２２によるタスクＴ４の処理について説明する。ＣＰＵ２２は、タスクＴ３の処理が終わると、順序番号による実行順序にもとづいて、タスクＴ４の処理を行う。図２のタスクフローグラフによると、タスクＴ４を前述の第１タスクとみなす場合に、前述の第２タスク群に相当するタスクはタスクＴ２である。ＣＰＵ２２は、タスクＴ４の第１ステップにおいて、タスクＴ２の出力データが参照可能な状態になったというフラグ（完了フラグ）を参照して、タスクＴ４を実行可能な状態（タスクＴ２の出力データが参照可能な状態）になるまで待合わせる。ここでは、ＣＰＵ２１によるタスクＴ２の処理（第１ステップから第４ステップ）が完了した後、タスクＴ２出力データの参照可能フラグ（完了フラグ）がセットされている（Ｓ３１６）ので、ＣＰＵ２２はタスクＴ２の完了フラグを検出して（Ｓ３２６）待合せを終了し、次のステップに移る。次に、ＣＰＵ２２は、タスクＴ４の第２ステップにおいて、第２タスク群に相当するタスク（タスクＴ２）が

存在するため、タスクＴ２の出力データ（データＢ３１３）を共有メモリ３０からＣＰＵ２２のローカルメモリ１２へ（データＢ５２２として）コピーする（Ｓ３２７）。次に、ＣＰＵ２２は、タスクＴ４の第３ステップにおいて、タスクＴ４を実行し（Ｓ３２８）、タスクＴ４の出力データ（データＤ５２４）をローカルメモリ１２に書き込む。次に、ＣＰＵ２２は、タスクＴ４の第４ステップにおいて、前述の第３タスク群に相当するタスク（ＣＰＵ２４が実行するタスクＴ６）が存在するため、タスクＴ４の出力データ（データＤ５２４）をローカルメモリ１２から共有メモリ３０へ（データＤ３１５として）コピーする（Ｓ３２９）とともに、タスクＴ４の出力データが参照可能な状態になったというフラグ（完了フラグ）をセットする（Ｓ３３０）。ここまでで、タスクＴ４の第１ステップから４が終了する。

【００６５】

続いて、ＣＰＵ２３によるタスクＴ５の処理について説明する。ＣＰＵ２２がタスクＴ３を実行すると、タスクＴ３の出力データが参照可能な状態になる。図２のタスクフローグラフによると、それによって、ＣＰＵ２３は、タスクＴ５を実行可能な状態となる。図２のタスクフローグラフによると、タスクＴ５を前述の第１タスクとみなす場合に、前述の第２タスク群に相当するタスクはタスクＴ３である。ＣＰＵ２３は、タスクＴ５の第１ステップにおいて、タスクＴ３の出力データが参照可能な状態になったというフラグ（完了フラグ）を参照して、タスクＴ５を実行可能な状態（タスクＴ３の出力データが参照可能な状態）になるまで待合わせる（Ｓ３４０）。そして、ＣＰＵ２２によるタスクＴ３の処理（第１ステップから第４ステップ）が完了した後は、タスクＴ３の出力データの参照可能フラグ（完了フラグ）がセットされている（Ｓ３２５）ので、ＣＰＵ２３はタスクＴ３の完了フラグを検出して（Ｓ３４１）待合せを終了し、次のステップに移る。次に、ＣＰＵ２３は、タスクＴ５の第２ステップにおいて、第２タスク群に相当するタスク（タスクＴ３）が存在するため、タスクＴ３の出力データ（データＣ３１４）を共有メモリ３０からＣＰＵ２３のローカルメモリ１３へ（データＣ５３１として）コピーする（Ｓ３４２）。次に、ＣＰＵ２３は、タスクＴ５の第３ステップにおいて、タスクＴ５を実行し（Ｓ３４３）、タスクＴ５の出力データ（データＥ５３２）をローカルメモリ１３に書き込む。次に、ＣＰＵ２３は、タスクＴ５の第４ステップにおいて、前述の第３タスク群に相当するタスク（ＣＰＵ２４が実行するタスクＴ６）が存在するため、タスクＴ５の出力データ（データＥ５３２）をローカルメモリ１３から共有メモリ３０へ（データＥ３１６として）コピーする（Ｓ３４４）とともに、タスクＴ５の出力データが参照可能な状態になったというフラグ（完了フラグ）をセットする（Ｓ３４５）。ここまでで、タスクＴ５の第１ステップから４が終了する。

【００６６】

続いて、ＣＰＵ２４によるタスクＴ６の処理について説明する。ＣＰＵ２２がタスクＴ４を、ＣＰＵ２３がタスクＴ５を、それぞれ実行すると、タスクＴ４及びタスクＴ５の出力データが参照可能な状態になる。図２のタスクフローグラフによると、それによって、ＣＰＵ２４は、タスクＴ６を実行可能な状態となる。図２のタスクフローグラフによると、タスクＴ６を前述の第１タスクとみなす場合に、前述の第２タスク群に相当するタスクはタスクＴ４及びタスクＴ５である。ＣＰＵ２４は、タスクＴ６の第１ステップにおいて、タスクＴ３の出力データが参照可能な状態になったというフラグ（完了フラグ）を参照して、タスクＴ６を実行可能な状態（タスクＴ３の出力データが参照可能な状態）になるまで待合わせる（Ｓ３５０）。そして、ＣＰＵ２２によるタスクＴ４の処理（第１ステップから第４ステップ）及びＣＰＵ２３によるタスクＴ５の処理（第１ステップから第４ステップ）が完了した後は、タスクＴ４の出力データの参照可能フラグ（完了フラグ）及びタスクＴ５出力データ参照可能フラグ（完了フラグ）がセットされている（Ｓ３３０及びＳ３４５）ので、ＣＰＵ２４はタスクＴ４及びＴ５の完了フラグを検出して（Ｓ３５１及びＳ３５２）待合せを終了し、次のステップに移る。次に、ＣＰＵ２４は、タスクＴ６の第２ステップにおいて、第２タスク群に相当するタスク（タスクＴ４及びタスクＴ５）が存在するため、タスクＴ４及びタスクＴ５の出力データ（データＤ３１５及びデータＥ３

１６）を共有メモリ３０からＣＰＵ２４のローカルメモリ１４へ（データＤ５４１及びデータＥ５４２として）コピーする（Ｓ３５３及びＳ３５４）。次に、ＣＰＵ２４は、タスクＴ６の第３ステップにおいて、タスクＴ６を実行し（Ｓ３５５）、タスクＴ６の出力データ（データＦ５４３）をローカルメモリ１４に書き込む。次に、ＣＰＵ２４は、タスクＴ６の第４ステップにおいて、前述の第３タスク群に相当するタスクは存在しないものの、タスクＴ６の出力データは図２のタスクフローグラフ全体の出力データとみなせるので、タスクＴ６の出力データ（データＦ５４３）をローカルメモリ１４から共有メモリ３０へ（データＦ３１７として）コピーする（Ｓ３５６）とともに、タスクＴ６の出力データが参照可能な状態になったというフラグ（完了フラグ）をセットする（Ｓ３５７）。ここまでで、タスクＴ６の第１ステップから４が終了する。

【００６７】

具体例１にかかるマルチコアプロセッサ１００は、図２のタスクフローグラフ全体の出力データであるタスクＴ６の出力データが参照可能な状態になったことを確認することによって、図２のタスクフローグラフ全体の処理が完了したと判断する。

【００６８】

＜本発明の実施形態の具体例２＞

実施形態の具体例２について説明する。具体例２は、複数のプロセッサが各々備えるローカルメモリを各プロセッサが相互にアクセス可能なように構成されたマルチコアプロセッサ上での並列処理を提供する並列情報処理装置、及び方法である。

【００６９】

具体例２は、タスクフローグラフとして表現可能なプログラムの任意のタスクを第１タスクとして、前述の第１ステップから第４ステップを行う。具体例２では、共有メモリを使わず、プロセッサが有するローカルメモリを参照することによって、プロセッサ間でデータを共有する。したがって、第２ステップのコピー元は或るプロセッサのローカルメモリとなり、第４ステップのコピー処理は不要となる。

【００７０】

第１ステップにおいて、具体例２は、第１タスクを実行可能な状態（第２タスク群の出力データが参照可能な状態）になるまで待合わせる。続いて、第２ステップにおいて、具体例２は、第１タスクへの入力データとなる第２タスク群の出力データを、第２タスク群を実行したプロセッサのローカルメモリ群から第１プロセッサのローカルメモリへコピーする。続いて、第３ステップにおいて、具体例２は、第１タスクを実行し、その出力データをローカルメモリに格納する。続いて、第４ステップにおいて、具体例２は、第１タスクの出力データを入力データとして使用する第３タスク群が存在するならば、第１タスクの出力データが参照可能な状態であることを第１プロセッサのローカルメモリに記録する。

【００７１】

続いて、上述の第１ステップから第４ステップを、各プロセッサの視点で説明する。前述のとおり、本発明の模範的な実施形態は、各プロセッサは順序番号が小さいタスクから順番に実行するものとしている。つまり、具体例１と同様に、具体例２の各プロセッサは、自身に実行が割り当てられているタスクの数だけ、第１ステップから第４ステップを繰り返す。さらに、具体例２の各プロセッサは、トポロジカルオーダにもとづいた順序番号にしたがってタスクを順番に実行する、という点も具体例２と同様である。したがって、具体例２と同様で、具体例１においても、ひとつのプロセッサに割り当てられたタスクどうしでタスクの実行完了を確認することは不要で、第２タスク群が存在しない場合に、第１ステップの待合せが不要となるとともに、第２ステップのコピー処理は不要で、第３タスク群が存在しない場合に、第４ステップのコピー処理および出力データ参照可能状態記録処理が不要、である。

【００７２】

＜具体例２の動作例＞

続いて、具体例２の動作例について図面を使って説明する。図８は、本発明の実施形態

の具体例２にかかるマルチコアプロセッサ１００ａのタスク割当てとデータ配置の例を示す図である。図８では、具体例２が備えるプロセッサの数を４として説明するが、プロセッサ数はこれに限定されない。具体例１の説明で使用したものと同じ図２のタスクフローグラフを使って具体例２の動作を説明する。図２の６個のタスクＴ１～Ｔ６を具体例２の４個のプロセッサ７１～７４に図８のように予め割当てておくものとする。つまり、タスクＴ１とタスクＴ２をＣＰＵ７１に、タスクＴ３とタスクＴ４をＣＰＵ７２に、タスクＴ５をＣＰＵ７３に、タスクＴ６をＣＰＵ７４に、それぞれ割り当てるものとする。この割り当ても、具体例１と同等である。このように割当てを決めたことにより、本発明の実施形態にかかる基本的な考え方「プロセッサのローカルメモリにタスクの入出力データを配置する」に基づいて、各タスクの入出力データの配置は図８のように決まることになる。つまり、図５におけるローカルメモリ１１～１４がローカルメモリ６１～６４に置き換わったものである。ローカルメモリのデータ配置は具体例１と同等である。そして、具体例２では、各プロセッサのローカルメモリを用いてプロセッサ間でデータを共有する。

【００７３】

次に、具体例２にかかるタスク処理（タスクＴ１からタスクＴ６を実行する様子）の流れを示すフローチャートを図９及び図１０に示す。具体例１の図６及び図７と具体例２の図９及び図１０の違いとしては、大きく２点ある。一点目は、具体例２では、各タスクの第２ステップにおける第２タスク群の出力データのコピー処理が不要であることである。具体的には、図６及び図７のＳ３１０、Ｓ３２２、Ｓ３２７、Ｓ３４２、Ｓ３５３、Ｓ３５４、Ｓ３５６に相当するコピー処理が図９及び図１０には不要である。二点目は、具体例２では、各タスクの第４ステップにおける第３タスク群のためのコピー処理の宛先が第３タスク群を実行するプロセッサのローカルメモリであることである。具体的には、図６及び図７のＳ３１２、Ｓ３１５、Ｓ３２４、Ｓ３２９、Ｓ３４４に相当するコピー処理の宛先が後続タスクが有するローカルメモリであることである。

【００７４】

尚、これとは逆に、各タスクの第４ステップにおける第３タスク群のためのコピー処理を省略する場合には、第２ステップにおける第２タスク群の出力データのコピー処理を実行することでも実現できる。つまり、第２プロセッサが第２タスクの実行後に第２ローカルメモリを第１プロセッサに対して参照可能とし、第１プロセッサが第１タスクの実行後に第１ローカルメモリを第２プロセッサに対して参照可能することでも実現できる。

【００７５】

＜その他の発明の実施の形態＞

尚、本発明にかかる他の実施形態である並列情報処理方法あるいは装置は、次のように表現することもできる。すなわち、タスクフローグラフとして表現可能なプログラムの任意の第１タスクの実行に関して、第１タスクを実行する前に、タスクフローグラフにおいて第１タスクへのエッジをもち、なおかつ第１タスクを実行する第１プロセッサ以外が実行する第２タスク群の出力データが参照可能状態となることを待ち（第１ステップ）、第２タスク群の出力データを第１プロセッサのローカルメモリにコピーし（第２ステップ）、第１タスクを実行してその計算結果を第１プロセッサのローカルメモリに格納し（第３ステップ）、タスクフローグラフにおいて第１タスクからのエッジをもちなおかつ第１プロセッサ以外が実行する第３タスク群が存在する場合に、第１タスクの出力データを第３タスク群が参照可能な状態とする（第４ステップ）、ことを特徴とする。

【００７６】

また、上記の第２ステップにおいて、第２タスク群の出力データが第１プロセッサのローカルメモリに存在することが明確である場合にコピーを省略することが望ましい。さらに、上記の第２ステップのコピー元は共有メモリであり、第４ステップのコピー先も共有メモリであることもできる。または、上記の第２ステップのコピー元は第２タスク群を実行するプロセッサのローカルメモリであり、プロセッサのローカルメモリを相互に参照可能とすることによって第４ステップのコピー処理を省略するとよい。

【００７７】

このように、対象とする並列処理に合わせて、マルチコアプロセッサが備えるローカルメモリに共有データを配置し、それをプロセッサ間で受け渡すことにより、共有メモリへのアクセス回数を減らし、さらに並列処理全体の処理時間を減らす、ことを可能にする。つまり、本発明にかかる実施形態は、共有メモリとローカルメモリとを備えるマルチコアプロセッサと、そのうえで動作するソフトウェアによって、共有メモリへのアクセス回数が少なく、全体の処理時間が少ない、並列情報処理装置、あるいは並列情報処理方法、を提供する。そのため、プロセッサが備えるローカルメモリを活用することにより、共有メモリへのアクセス回数を減らし、メモリアクセスコストを低くし、全体の処理時間を短くすることができる。

【００７８】

尚、本発明にかかる実施形態は、組み込み向けプロセッサ、汎用コンピュータ用プロセッサ、など、さまざまなプロセッサによる並列処理に応用可能である。

【００７９】

さらに、本発明は上述した実施の形態のみに限定されるものではなく、既に述べた本発明の要旨を逸脱しない範囲において種々の変更が可能であることは勿論である。例えば、上述の実施の形態では、本発明をハードウェアの構成として説明したが、本発明は、これに限定されるものではない。本発明は、任意の処理を、ＣＰＵ（Central Processing Unit）にコンピュータプログラムを実行させることにより実現することも可能である。

【００８０】

上述の例において、プログラムは、様々なタイプの非一時的なコンピュータ可読媒体（non-transitory computer readable medium）を用いて格納され、コンピュータに供給することができる。非一時的なコンピュータ可読媒体は、様々なタイプの実体のある記録媒体（tangible storage medium）を含む。非一時的なコンピュータ可読媒体の例は、磁気記録媒体（例えばフレキシブルディスク、磁気テープ、ハードディスクドライブ）、光磁気記録媒体（例えば光磁気ディスク）、ＣＤ－ＲＯＭ（Read Only Memory）、ＣＤ－Ｒ、ＣＤ－Ｒ／Ｗ、ＤＶＤ（Digital Versatile Disc）、ＢＤ(Blu-ray(登録商標) Disc)、半導体メモリ（例えば、マスクＲＯＭ、ＰＲＯＭ（Programmable ROM）、ＥＰＲＯＭ（Erasable PROM）、フラッシュＲＯＭ、ＲＡＭ（Random Access Memory））を含む。また、プログラムは、様々なタイプの一時的なコンピュータ可読媒体（transitory computer readable medium）によってコンピュータに供給されてもよい。一時的なコンピュータ可読媒体の例は、電気信号、光信号、及び電磁波を含む。一時的なコンピュータ可読媒体は、電線及び光ファイバ等の有線通信路、又は無線通信路を介して、プログラムをコンピュータに供給できる。

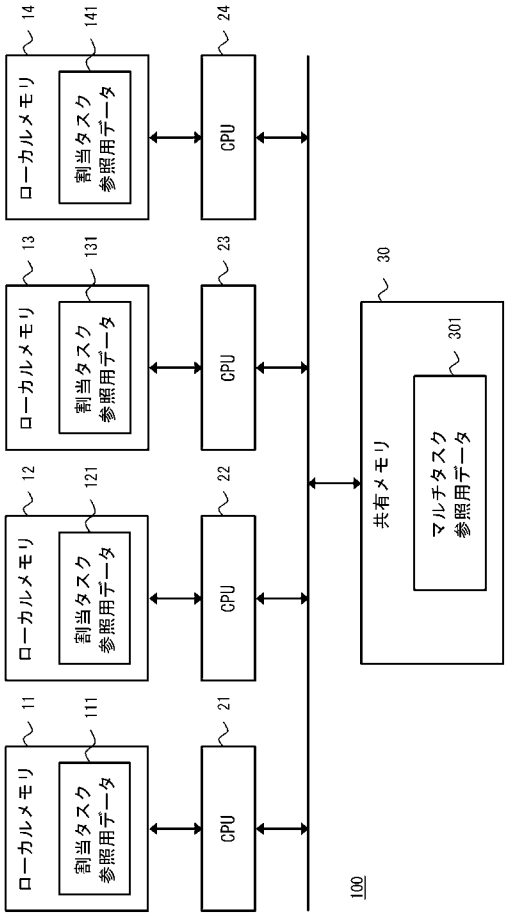
【符号の説明】

【００８１】

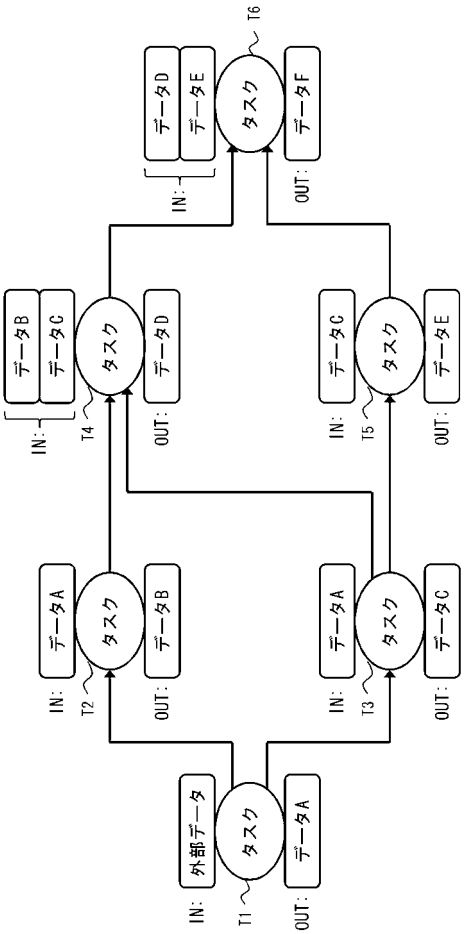
１００　マルチコアプロセッサ
１００ａ　マルチコアプロセッサ
１１　ローカルメモリ
１２　ローカルメモリ
１３　ローカルメモリ
１４　ローカルメモリ
１１１　割当タスク参照用データ
１２１　割当タスク参照用データ
１３１　割当タスク参照用データ
１４１　割当タスク参照用データ
２１　ＣＰＵ
２２　ＣＰＵ
２３　ＣＰＵ
２４　ＣＰＵ
３０　共有メモリ

３０１　マルチタスク参照用データ
３１１　外部データ
３１２　データＡ
３１３　データＢ
３１４　データＣ
３１５　データＤ
３１６　データＥ
３１７　データＦ
５１１　外部データ
５１２　データＡ
５１３　データＢ
５２１　データＡ
５２２　データＢ
５２３　データＣ
５２４　データＤ
５３１　データＣ
５３２　データＥ
５４１　データＤ
５４２　データＥ
５４３　データＦ
６１　ローカルメモリ
６２　ローカルメモリ
６３　ローカルメモリ
６４　ローカルメモリ
７１　ＣＰＵ
７２　ＣＰＵ
７３　ＣＰＵ
７４　ＣＰＵ
９００　マルチコアプロセッサ
９０１　共有データ
９１１　非共有データ
９２１　非共有データ
９３１　非共有データ
９４１　非共有データ
Ｔ１　タスク
Ｔ２　タスク
Ｔ３　タスク
Ｔ４　タスク
Ｔ５　タスク
Ｔ６　タスク

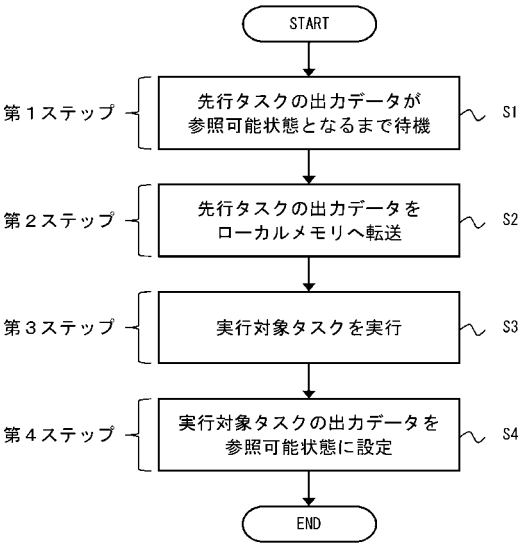
【図１】


14
141
ローカルメモリ
割当タスク参照用データ
24
CPU
13
131
ローカルメモリ
割当タスク参照用データ
23
CPU
30
301
共有メモリ
マルチタスク参照用データ
12
121
ローカルメモリ
割当タスク参照用データ
22
CPU
11
111
ローカルメモリ
割当タスク参照用データ
21
CPU
100

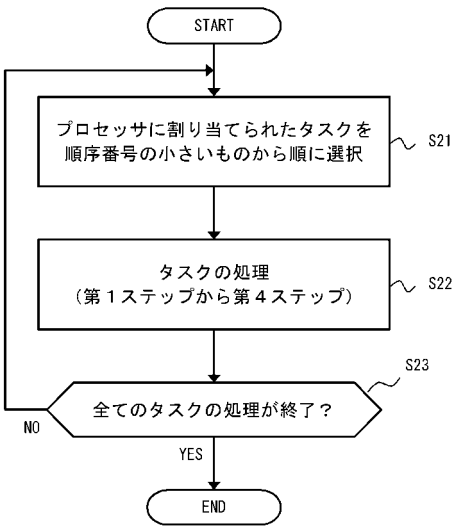
【図２】


T6
データD
データE
タスク
データF
IN:
OUT:
データB
データC
タスク
データD
T4
データC
タスク
データE
T5
データA
タスク
データB
T2
データA
タスク
データC
T3
外部データ
タスク
データA
T1

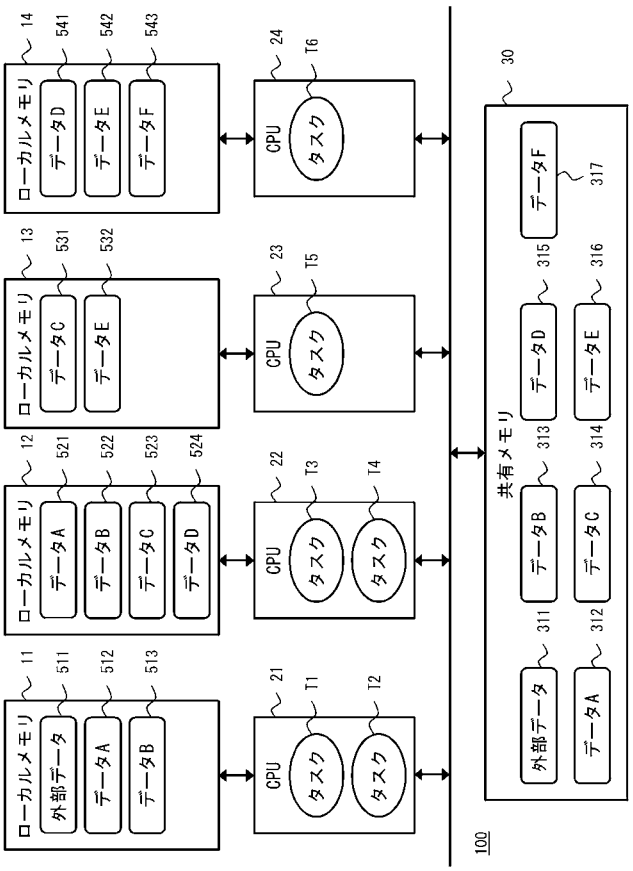
【図３】


START
第１ステップ
先行タスクの出力データが参照可能状態となるまで待機
S1
第２ステップ
先行タスクの出力データをローカルメモリへ転送
S2
第３ステップ
実行対象タスクを実行
S3
第４ステップ
実行対象タスクの出力データを参照可能状態に設定
S4
END

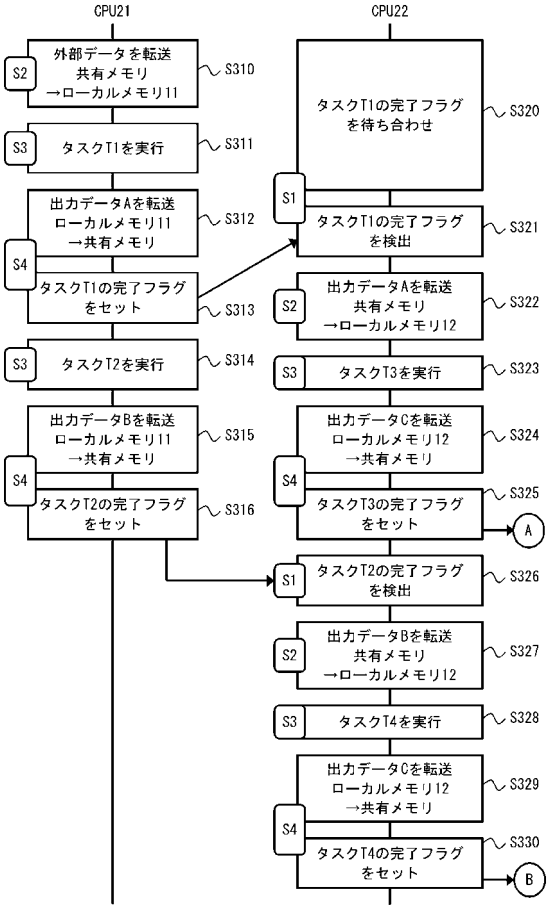
【図４】


START
プロセッサに割り当てられたタスクを順序番号の小さいものから順に選択
S21
タスクの処理
（第１ステップから第４ステップ）
S22
S23
全てのタスクの処理が終了？
NO
YES
END

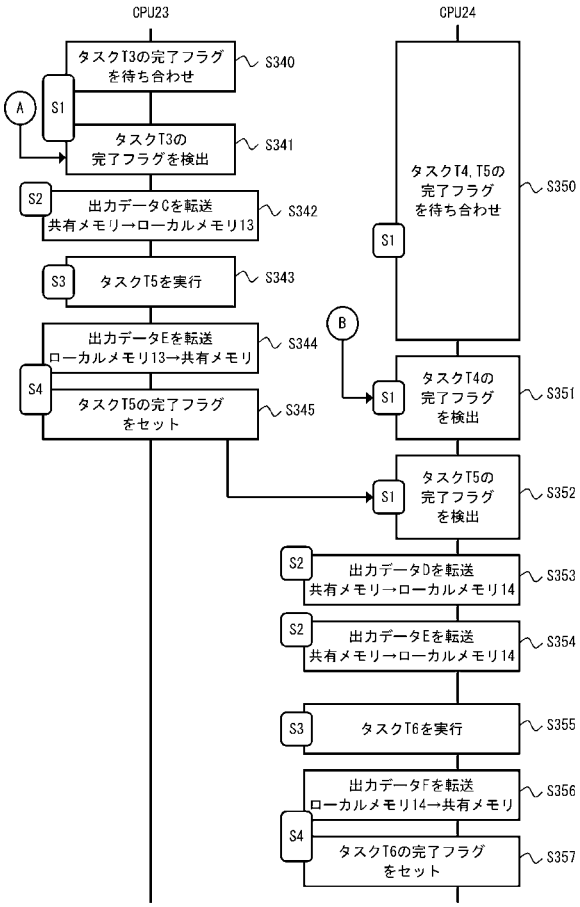
【図５】


ローカルメモリ
データD
データE
データF
CPU
タスク
ローカルメモリ
データC
データE
CPU
タスク
ローカルメモリ
データA
データB
データC
データD
CPU
タスク
タスク
ローカルメモリ
外部データ
データA
データB
CPU
タスク
タスク
共有メモリ
データF
データD
データE
データB
データC
外部データ
データA
100

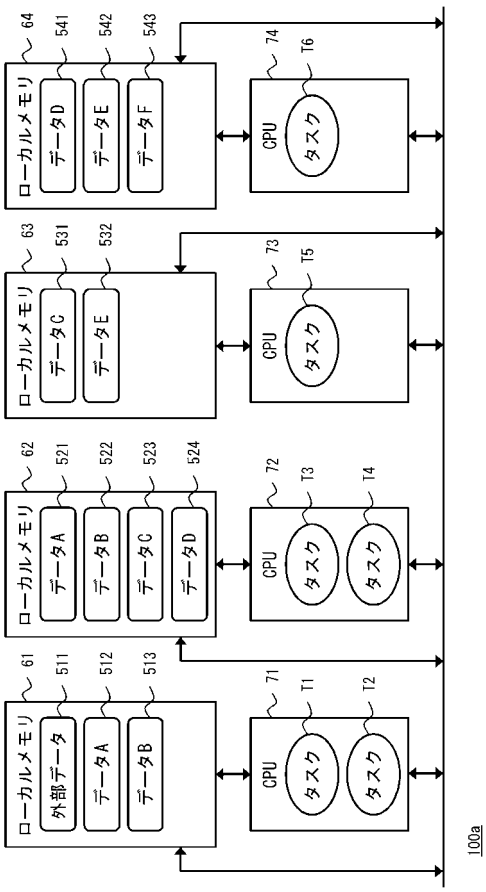
【図６】


CPU21
CPU22
外部データを転送
共有メモリ
→ローカルメモリ11
S310
タスクT1を実行
S311
出力データAを転送
ローカルメモリ11
→共有メモリ
S312
タスクT1の完了フラグ
をセット
S313
タスクT2を実行
S314
出力データBを転送
ローカルメモリ11
→共有メモリ
S315
タスクT2の完了フラグ
をセット
S316
タスクT1の完了フラグ
を待ち合わせ
S320
タスクT1の完了フラグ
を検出
S321
出力データAを転送
共有メモリ
→ローカルメモリ12
S322
タスクT3を実行
S323
出力データCを転送
ローカルメモリ12
→共有メモリ
S324
タスクT3の完了フラグ
をセット
S325
タスクT2の完了フラグ
を検出
S326
出力データBを転送
共有メモリ
→ローカルメモリ12
S327
タスクT4を実行
S328
出力データCを転送
ローカルメモリ12
→共有メモリ
S329
タスクT4の完了フラグ
をセット
S330
A
B

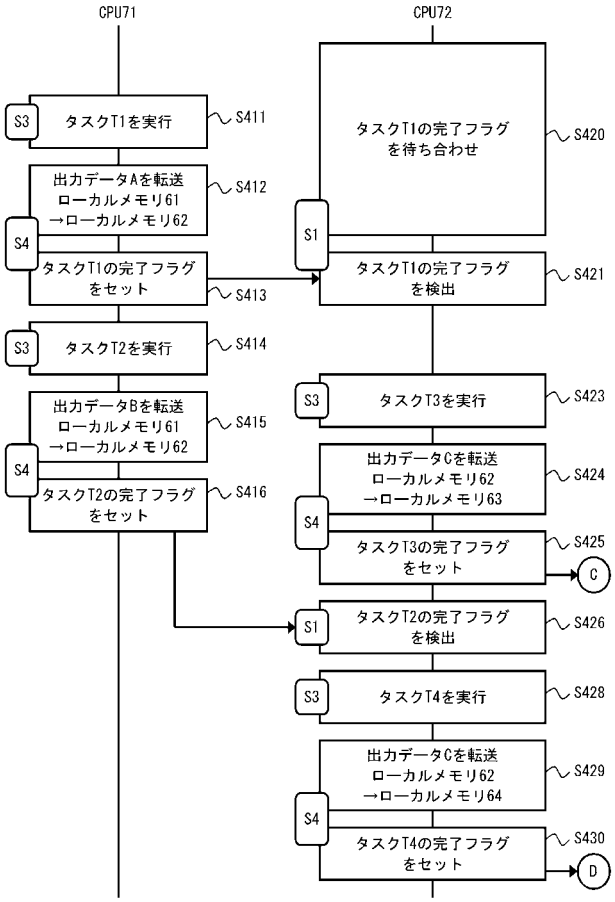
【図７】


CPU23
CPU24
タスクT3の完了フラグ
を待ち合わせ
S340
タスクT3の
完了フラグを検出
S341
出力データCを転送
共有メモリ→ローカルメモリ13
S342
タスクT5を実行
S343
出力データEを転送
ローカルメモリ13→共有メモリ
S344
タスクT5の完了フラグ
をセット
S345
タスクT4,T5の
完了フラグ
を待ち合わせ
S350
タスクT4の
完了フラグ
を検出
S351
タスクT5の
完了フラグ
を検出
S352
出力データDを転送
共有メモリ→ローカルメモリ14
S353
出力データEを転送
共有メモリ→ローカルメモリ14
S354
タスクT6を実行
S355
出力データFを転送
ローカルメモリ14→共有メモリ
S356
タスクT6の完了フラグ
をセット
S357
A
B

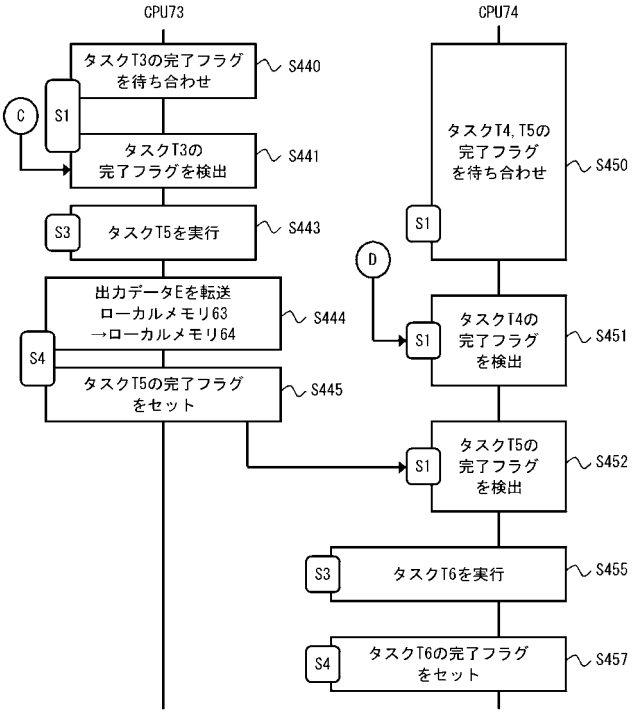
【図８】


ローカルメモリ
データD
データE
データF
CPU
タスク
ローカルメモリ
データC
データE
CPU
タスク
ローカルメモリ
データA
データB
データC
データD
CPU
タスク
タスク
ローカルメモリ
外部データ
データA
データB
CPU
タスク
タスク
100a

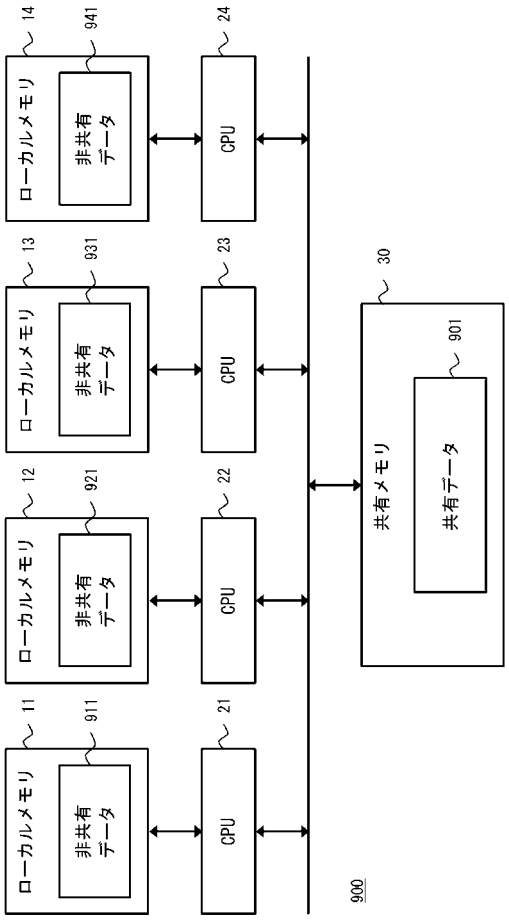
【図９】


CPU71
CPU72
S3
タスクT1を実行
S411
出力データAを転送
ローカルメモリ61
→ローカルメモリ62
S4
S412
タスクT1の完了フラグ
をセット
S413
S3
タスクT2を実行
S414
出力データBを転送
ローカルメモリ61
→ローカルメモリ62
S4
S415
タスクT2の完了フラグ
をセット
S416
タスクT1の完了フラグ
を待ち合わせ
S420
S1
タスクT1の完了フラグ
を検出
S421
S3
タスクT3を実行
S423
出力データCを転送
ローカルメモリ62
→ローカルメモリ63
S4
S424
タスクT3の完了フラグ
をセット
S425
C
S1
タスクT2の完了フラグ
を検出
S426
S3
タスクT4を実行
S428
出力データCを転送
ローカルメモリ62
→ローカルメモリ64
S4
S429
タスクT4の完了フラグ
をセット
S430
D

【図１０】


CPU73
CPU74
タスクT3の完了フラグ
を待ち合わせ
S440
C
S1
タスクT3の
完了フラグを検出
S441
S3
タスクT5を実行
S443
出力データEを転送
ローカルメモリ63
→ローカルメモリ64
S444
S4
タスクT5の完了フラグ
をセット
S445
タスクT4, T5の
完了フラグ
を待ち合わせ
S450
S1
D
タスクT4の
完了フラグ
を検出
S451
S1
タスクT5の
完了フラグ
を検出
S452
S1
S3
タスクT6を実行
S455
S4
タスクT6の完了フラグ
をセット
S457

【図１１】


14
941
ローカルメモリ
非共有
データ
24
CPU
13
931
ローカルメモリ
非共有
データ
23
CPU
12
921
ローカルメモリ
非共有
データ
22
CPU
11
911
ローカルメモリ
非共有
データ
21
CPU
30
共有メモリ
901
共有データ
900