(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2012-510661
(P2012-510661A)
(43)公表日　平成24年5月10日(2012.5.10)

(51)Int.Cl.			F I			テーマコード（参考）
G06F	9/45	(2006.01)	G06F	9/44	320F	5B057
G06T	1/20	(2006.01)	G06F	9/44	322F	5B081
			G06T	1/20	B	

審査請求 未請求　予備審査請求 未請求　（全 17 頁）

(21)出願番号　特願2011-538103 (P2011-538103)
(86)(22)出願日　平成21年12月1日 (2009.12.1)
(85)翻訳文提出日　平成23年5月31日 (2011.5.31)
(86)国際出願番号　PCT/IN2009/000697
(87)国際公開番号　WO2010/064260
(87)国際公開日　平成22年6月10日 (2010.6.10)
(31)優先権主張番号　2513/MUM/2008
(32)優先日　平成20年12月1日 (2008.12.1)
(33)優先権主張国　インド(IN)

(71)出願人　510077901
ケーピーアイティ　クミンズ　インフォシステムズ　リミテッド
インド，　４１１０５７　ヒンジャワディ　プネ，　エムアイディシィ，　ラジーブ　ガンディ　インフォテック　パーク　フェーズ　１，　３５　アンド　３６
(74)代理人　100107364
弁理士　斉藤　達也
(72)発明者　ヴィナイ　ゴヴィンド　ヴァイディヤ
インド，　４１１　０５７　プネ，　ヒンジャワディ，　エムアイディシィ，　フェーズ　１，　３５　アンド　３６　ラジーブ　ガンディ　インフォテック　パーク

最終頁に続く

(54)【発明の名称】逐次コンピュータプログラムコードを並列処理する方法及びシステム

(57)【要約】
逐次コンピュータプログラムコードの並列処理を行う方法及びシステムを提供する。一実施形態において自動並列処理システムは、逐次コンピュータプログラムコードの構造を解析して、逐次コンピュータプログラムコードにＳＰＩを挿入する位置を識別する構文解析手段と、逐次コンピュータプログラムコードの各ラインと逐次コンピュータプログラムコードの各関数の実行に必要な時間との依存関係を判定するため、コールグラフを作成することによって逐次コンピュータプログラムコードをプロファイリングするプロファイリング手段と、逐次コンピュータプログラムコードを解析及びプロファイリングすることによって得た情報により逐次コンピュータプログラムコードの並列処理可能性を判定する解析手段と、並列処理可能性が判定されると、逐次コンピュータプログラムコードにＳＰＩを挿入し、並列コンピュータプログラムコードを得て、これを実行するために並列計算環境にさらに出力するコード生成手段とを含む。また、並列処理を行う方法も示す。
【選択図】図１

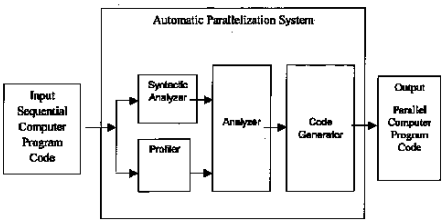


FIG. 1

【特許請求の範囲】
【請求項１】
実行速度を高めるために逐次コンピュータプログラムコードを並列処理する方法であって、
前記逐次コンピュータプログラムコードを自動並列処理システムに入力するステップと、
前記逐次コンピュータプログラムコードの構造を解析して、機能性に影響を与えることなく前記逐次コンピュータコードに特別プログラム命令（ＳＰＩ）を挿入する位置を識別するステップと、
前記逐次コンピュータプログラムコードの各ラインと前記逐次コンピュータプログラムコードの各関数の実行に必要な時間との依存関係を判定するため、関数又はモジュールのコールグラフ、時間測定、及び時間表を作成することによって、前記逐次コンピュータプログラムコードをプロファイリングするステップと、
前記逐次コンピュータプログラムコードを解析及びプロファイリングすることによって得た情報により前記逐次コンピュータプログラムコードの並列処理可能性を判定するステップと、
並列処理可能性が判定されると、前記逐次コンピュータプログラムコードにＳＰＩを挿入して、並列実行可能なコンピュータプログラムコードを得るステップと、
前記並列実行可能なコンピュータプログラムコードを実行するために並列計算環境に送るステップと、
を含む方法。
【請求項２】
前記逐次コンピュータプログラムコードが、逐次実行されるソースコードの形態の関数を複数含む、請求項１に記載の方法。
【請求項３】
前記逐次コンピュータコードを実行する実際のタイムスケールにおいて、異なる関数又はモジュールを実行するシーケンスをグラフで示した、前記逐次コンピュータプログラムコードの変数対時間、変数対ライン番号、及び関数対時間のグラフを、コールグラフが含む、請求項１に記載の方法。
【請求項４】
実行に必要と識別した時間をコールグラフに重ねることによって、前記逐次コンピュータプログラムコードの各関数の無駄時間を判定する、請求項３に記載の方法。
【請求項５】
前記逐次コンピュータプログラムコードを解析及びプロファイリングすることによって得た情報を、その後の検索のために前記自動並列処理システムのメモリに記憶する、請求項１に記載の方法。
【請求項６】
前記並列コンピュータプログラムコードを用いることで、逐次コンピュータプログラムコードの実行パターンによって実行を開始する実際の時間より前に、前記コンピュータプログラムコードを実行できる、請求項１に記載の方法。
【請求項７】
前記方法が、逐次コンピュータプログラムコード内の特定の関数若しくはモジュール、又はその一部の実行を開始するため、絶対最短待機時間（ＡＭＷＴ）を評価する反復プロセスを提供する、請求項１に記載の方法。
【請求項８】
それらに限定されないが、前記逐次コンピュータプログラムコードが、制御フロー文及びループ文等の文をさらに含む、請求項２に記載の方法。
【請求項９】
それらに限定されないが、if-else文、スイッチケース文等の前記逐次コンピュータプログラムコードにおける制御フロー文を並列処理することが、

前記制御フロー文におけるデータ変数を識別するステップと、
前記制御フロー文内の前記データ変数が最後に更新されたライン番号及び実行時間を識別するステップと、
前記識別したライン番号をその後の検索のためにメモリに記憶するステップと、
前記記憶したデータを用いてライン及び時間の依存関係を示すグラフをプロットし、前記データ変数を更新するステップと、
前記ライン及び時間の依存関係の統計を使用して、モジュール／関数全体又はその一部を識別して異なるプロセッサに送り、逐次コンピュータプログラムコードの並列処理を行うステップと、
プロセッサによる要求があった際に前記メモリから呼び出すために変数名又はアドレスを使用して、前記制御文の内部並列処理を行うステップとを含む、請求項８に記載の方法。

【請求項１０】
前記逐次コンピュータプログラムコードにおけるループを並列処理する方法は、
前記ループ文中のデータ変数を識別するステップと、
前記文同士の依存関係をチェックするステップと、
前記逐次コンピュータプログラムコード内の前記データ変数が最後に更新されたライン番号を識別するステップと、
前記識別したライン番号をメモリに記憶するステップと、
前記記憶したデータを使用して、前記データ変数の更新に関するライン依存関係を示すグラフをプロットするステップと、
前記ライン依存関係の統計を使用して、コードの特定のセグメントを識別して異なるプロセッサに送り、逐次コンピュータプログラムコードの並列処理を行うステップとを含む、請求項８に記載の方法。

【請求項１１】
複数のコードを並列処理する代替的方法が、逐次コンピュータプログラムコードをクラスタインデックスによってクラスタに分別することによるものである、請求項１に記載の方法。

【請求項１２】
データ依存関係、オリジナルコードによる実行の時間表、及び実行の時間測定を含む複数のパラメータから、前記クラスタインデックスを得る、請求項１１に記載の方法。

【請求項１３】
逐次コンピュータプログラムコードの並列処理を行う自動並列処理システムであって、
前記逐次コンピュータプログラムコードの構造を解析して、前記逐次コンピュータプログラムコードに特別プログラム命令（ＳＰＩ)を挿入する位置を識別する構文解析手段と、
前記逐次コンピュータプログラムコードの各ラインと前記逐次コンピュータプログラムコードの各関数の実行に必要な時間との依存関係を判定するため、コールグラフを作成することによって、前記逐次コンピュータプログラムコードをプロファイリングするプロファイリング手段と、
前記逐次コンピュータプログラムコードを解析及びプロファイリングすることによって得た情報により前記逐次コンピュータプログラムコードの並列処理可能性を判定する解析手段と、
並列処理可能性が判定されると、前記逐次コンピュータプログラムコードにＳＰＩを挿入し、並列実行可能なコンピュータプログラムコードを得て、これを実行するために並列計算環境にさらに出力するコード生成手段とを含む、システム。

【請求項１４】
それらに限定されないが、前記並列計算環境が、近接配置及び／又は遠隔配置されたマルチコアプロセッサ、対称型プロセッサ、非対称型プロセッサ等の複数のプロセッサを含む、請求項１３に記載のシステム。

【請求項１５】
前記自動並列処理システムが、前記構文解析手段及び前記プロファイリング手段の得た情報を、その後の検索のために記憶するメモリをさらに含む、請求項１３に記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は逐次コンピュータプログラムの実行に関し、より詳細には、逐次コンピュータプログラムの実行を高速化するために逐次コンピュータプログラムコードを並列処理する方法及びシステムに関する。
【背景技術】
【０００２】
ここ数十年で、コンピュータシステムと様々な領域においてそれを適応する上での有用性が開発されてきた。ソフトウェア開発、より詳細にはこのようなコンピュータプログラムの書込みに使用するコード体系における進歩のペースを速くするには、これらコンピュータプログラムの実行に使用するハードウェアの処理能力を高めることが必要であった。それらの目的を達成するために、２つの面で重要な取り組みがなされた。１つはより高速であると共に特定のタスクを行うプロセッサの開発におけるものであり、もう１つは利用可能なプロセッサにおいてより速く実行するためのコンピュータコードを再構築する分野におけるものである。
【０００３】
コンピュータプログラムのより高速な処理を可能にする方法である並列計算は、最近の当該技術分野における進化の中心であった。並列計算により、複数のプロセッサを使用すること、そして最近では多数の処理要素を備えたプロセッサを使用することへの道が開かれた。これに関連して、プロセッサクラスタ並びにグリッドの概念は特筆に値するものであり、マルチプロセッサコンピュータは同一機械内で多数の処理要素を有する。
【０００４】
並列計算の概念は有利であるように思えると共に、ソフトウェア開発者の間で受け入れられ人気が高まったが、当分野の中心となるこの概念の欠点は、既存のプログラミング方法が逐次的であるために、コードの並列処理には直接的に適さないことである。このようなプログラムコードを並列処理用に書き換えることは一般に面倒である。さらに、コードの並列処理可能な部分の識別とマッピング、そして遠隔であるかどうかは別として様々なプロセッサ要素に対して合理的なスケジューリングを行うこと、そしてまたかかる処理中の通信に関しては、今でもコンピュータプログラムの実際の実行を最適に高速化する際の大きな障害となっている。
【０００５】
マルチコアプロセッサの使用は、高クロックサイクル、熱生成、及び電源入力の必要なく高い処理能力を達成することで、複数のプロセッサを使用することよりも勝っている。しかし、マルチコアプロセッサの人気が高まりに伴い、それに関連したツールの必要性も高まっている。アプリケーションの開発者には、マルチコアプロセッサのプログラミングを容易化することのできるツールが必要である。一方、並列ハードウェアを最適に利用するべく、既存のコンピュータプログラムを並列処理するツールと方法に対する必要性も高まっている。
【０００６】
この点で様々な従来技術は、逐次コンピュータプログラムコードの並列実行を、次のような様々な手段により試みてきた：制御及びデータフロー解析を実行して並列可能な状況を判定すること、クラス特定抽出に基づいて並列可能な状況を判定すること、専用の多重処理ユニットを使用すること、入力ソースコードを中間的言語に変換し、その後タスク依存を実行すること、及び有向非巡回グラフ（ＤＡＧ）とpost-wait制御構造を使用すること等である。また別の方法は、データパターンとプロファイリング情報にアクセスして、

最初に接触したデータを制御するコードを生成するべく、同等のデータアクセスパターンを生成するというものである。

【０００７】

しかし、既存の従来技術における方法には次のような欠点がある。これらの方法の大半のものは人的な介入が不可欠であり、これらの方法はループの並列化、即ち反復実行するコードの部分のみに集中している。さらに、逐次コードを変更すると実際の機能性を抑制するおそれがあると共に、かかる方法はアプリケーションの限られた範囲にしか適用できない。さらに、入力ソースコードを並列処理用の中間言語に変換することは面倒なプロセスであり、これを行う方法はほぼ存在しない。

【０００８】

これらの問題を処理することに関連した情報は、以下の特許文献１から特許文献８に記載されている。しかし、それらの文献には、それぞれ上述の欠点が１つ以上存在する。

【先行技術文献】

【特許文献】

【０００９】

【特許文献１】米国特許第６２５３３７１号
【特許文献２】米国特許第６２４３８６３号
【特許文献３】独国特許番号１０２００５０５０５６１８６号
【特許文献４】米国特許第２００１０００３１８７号
【特許文献５】米国特許第２００７０２３４３２６号
【特許文献６】米国特許第２００７０２２６６８６号
【特許文献７】米国特許第６６２２３０１号
【特許文献８】米国特許第６７４２０８３号

【発明の概要】

【発明が解決しようとする課題】

【００１０】

そのため、旧式コード、既存の逐次コード、及び逐次的に書かれた新規コードを、並列実行用のコードに自動的に変換して、並列プロセッサにおいてより高速な実行を達成する必要がある。従って、本発明はコンピュータプログラムコードの効果的な並列処理を可能にすることにより、コンピュータプログラムの実行時間を減少するように作用する方法及びシステムを示すものである。

【００１１】

開示する発明の幾つかの態様の基本的な理解を促すため、発明の概要を以下に示す。以下の概要は詳細に関わる要約ではなく、鍵となる要素／重要な要素を特定するものでもなければ、発明の範囲を明確に示すことを意図したものでもない。その唯一の目的は、以下に記載のより詳細な説明の前置きとして、発明の概念を単純に示すことである。

【００１２】

本発明の主要な目的は、逐次コンピュータプログラムコードの並列計算を可能にする方法及びシステムを示すことである。

【課題を解決するための手段】

【００１３】

本発明の一態様では、逐次コンピュータプログラムコードの実行を高速化するために並列処理を行う方法を示す。該方法は、逐次コンピュータプログラムコードを自動並列処理システムに入力するステップと；逐次コンピュータプログラムコードの構造を解析して、機能性に影響を与えることなく逐次コンピュータコードに特別プログラム命令（ＳＰＩ）を挿入する位置を識別するステップと；逐次コンピュータプログラムコードの各ラインと逐次コンピュータプログラムコードの各関数又はモジュールの実行に必要な時間との依存関係を判定するため、関数又はモジュールのコールグラフ、時間測定、及び時間表を作成することによって、逐次コンピュータプログラムコードをプロファイリングするステップと；逐次コンピュータプログラムコードを解析及びプロファイリングすることによって得

た情報により逐次コンピュータプログラムコードの並列処理可能性を判定するステップと；並列処理可能性が判定されると、逐次コンピュータプログラムコードにＳＰＩを挿入して、並列実行可能なコンピュータプログラムコードを得るステップと；並列実行可能なコンピュータプログラムコードを実行するために並列計算環境に送るステップと；を含む。

【００１４】

本発明の別の態様では、逐次コンピュータプログラムコードの並列処理を行う自動並列処理システムを示す。該システムは、逐次コンピュータプログラムコードの構造を解析して、逐次コンピュータプログラムコードに特別プログラム命令（ＳＰＩ）を挿入する位置を識別する構文解析手段と；逐次コンピュータプログラムコードの各ラインと逐次コンピュータプログラムコードの各関数又はモジュールの実行に必要な時間との依存関係を判定するため、関数又はモジュールのコールグラフ、時間測定、及び時間表を作成することによって、逐次コンピュータプログラムコードをプロファイリングするプロファイリング手段と；逐次コンピュータプログラムコードを解析及びプロファイリングすることによって得た情報により逐次コンピュータプログラムコードの並列処理可能性を判定する解析手段と；並列処理可能性が判定されると、逐次コンピュータプログラムコードにＳＰＩを挿入し、並列実行可能なコンピュータプログラムコードを得て、これを実行するために並列計算環境にさらに出力するコード生成手段と；を含む。

【００１５】

従って、本発明は、逐次コンピュータプログラムコード内の特定の関数又は部分の実行を開始するために絶対最短待機時間（ＡＭＷＴ）を評価する反復プロセスを提供する。

【００１６】

本発明では逐次コンピュータプログラムコード中の関数又はモジュールをそれらの呼び出し時間より前に実行することができるため、典型的な逐次実行と比較して逐次コンピュータプログラムコードの実行速度が増し、有利である。

【００１７】

本明細書で開示するシステム及び装置は、様々な態様を達成する任意の手段で実行可能である。添付の図面及び以下の詳細な記載から、その他の特徴が明らかとなろう。

【００１８】

幾つかの実施形態を例として示すが、実施形態は添付の図面に限定されない。また図面では、同じ要素については同じ参照番号を付して示している。

【００１９】

添付の図面及び以下の詳細な記載から、実施形態のその他の特徴が明らかとなろう。

【図面の簡単な説明】

【００２０】

【図１】本発明の原理による自動並列処理システムを示すブロック図である。

【図２】逐次コンピュータプログラムコード中の関数又はモジュールの呼び出しを示す図である。

【図３】本発明の図３の好適な実施形態による時間前実行論理を示すグラフである。

【図４】典型的なコンピュータプログラムにおける関数又はモジュールを説明するために入力及び出力を有するチップを例示した図である。

【発明を実施するための形態】

【００２１】

添付の図面を参照して、本発明の実施形態を説明する。ただし、開示する実施形態は本発明の単なる一例であり、本発明は様々な形態で実施できることを理解されたい。以下の記載と図面は本発明を制限するものとらえるべきではなく、また特許請求の範囲の基礎として、そして本発明の構築及び／又は使用を当業者に教示するための基礎として、特定的な詳細を複数示す。ただし、場合によっては、本発明の詳細を不必要に不明瞭にしないように、周知又は従来の技術に関する詳細については記載しない。

【００２２】

本発明はコンピュータプログラムコードを並列処理する方法とシステムを示す。逐次コ

ンピュータプログラムコードは、逐次実行用のソースコードの形態の関数又はモジュールを複数含む。一実施形態では、アプリケーションを少なくとも一度実行することによりオフラインで逐次コンピュータプログラムを解析し、解析及びプロファイリングのプロセスを通して並列可能なコードを判定する。さらに、コード又はその一部を事前に実行するための特別目的命令（ＳＰＩ）を挿入することにより、逐次ソースコードを修正して新規コードを生成することで、利用する並列処理ハードウェアを効率的に使用することができる共に、アプリケーションの実行に必要な総時間が短くなる。「コンピュータプログラムコード」、「ソースコード」、「コード」、「逐次コンピュータプログラムコード」といった用語は、特定しない限り本明細書を通して同じ意味で用いることとする。任意の並列計算システム及びコンピュータプログラムコードに本発明を適用できることは、当業者には明らかであろう。

【００２３】

図１は、本発明の原理による自動並列処理システムを示すブロック図である。詳細には、自動並列処理システムは図１に示すように連結した構文解析手段、プロファイリング手段、解析手段、及びコード生成手段を含む。

【００２４】

別の実施形態では、構文解析手段は逐次コンピュータプログラムコードの構造を解析して、逐次コンピュータプログラムコードにおけるＳＰＩの挿入位置を識別する。プロファイリング手段は、関数又はモジュールのコールグラフ、時間測定、及び時間表を作成することにより、逐次コンピュータプログラムコードを解析し、逐次コンピュータプログラムコードの各ラインと逐次コンピュータプログラムコードの各関数の実行に必要な時間との依存関係を判定する。さらに、解析手段は逐次コンピュータプログラムコードの解析とプロファイリングによって得た情報から逐次コンピュータプログラムコードの並列処理可能性を判定し、並列処理可能性が判定されると、コード生成手段が逐次コンピュータプログラムコードにＳＰＩを挿入して並列コンピュータプログラムコードを得て、これを実行するために並列計算環境にさらに出力する。さらに、自動並列処理システムは並列計算環境に関連したメモリを含み、構文解析手段とプロファイリング手段の得た情報をその後の検索のために記憶する。

【００２５】

並列計算環境は複数のプロセッサを含み、プロセッサとしては例えば近接配置及び／又は遠隔配置されたマルチコアプロセッサ、対称型プロセッサ、非対称型プロセッサ等があるが、それらに限定されないことを理解されたい。

【００２６】

動作上、逐次コンピュータプログラムコードの並列処理を行う方法は、自動並列処理システムに逐次コンピュータプログラムコードを入力することを含む。さらに、入力された逐次コンピュータプログラムコードの構造を構文解析手段が解析し、機能性に影響を与えることなく逐次コンピュータコードに特定プログラム命令（ＳＰＩ）を挿入する位置を識別する。同時に、プロファイリング手段が逐次コンピュータプログラムコードにおいて関数又はモジュールのコールグラフ、時間測定、及び時間表を生成し、逐次コンピュータプログラムコードの各ラインと逐次コンピュータプログラムコードの各関数を実行するのに必要な時間との依存関係を判定する。

【００２７】

さらに、解析手段は、図２で示すように逐次コンピュータプログラムコードにおける関数又はモジュールを読み出すことにより、逐次コンピュータプログラムコードを解析及びプロファイリングすることによって得た情報により逐次コンピュータプログラムコードの並列可能性を判定する。並列可能性が判定されると、コード生成手段が逐次コンピュータプログラムコードにＳＰＩを挿入して並列コンピュータプログラムコードを得て、これを実行するために並列計算環境に送る。

【００２８】

一例示的な実施形態では、プロファイリングは逐次コンピュータプログラムコードにお

けるデータ変数の実行又は定義と更新に必要な時間を判定するため、変数対時間の３Ｄコールグラフをプロットすることを含む。このグラフをレンダリングするために、逐次コンピュータプログラムコードを少なくとも１回オフラインで実行し、変数の値を更新された時間と関連づける。レンダリングする別のグラフは変数対ライン番号のグラフであるが、これは逐次コンピュータプログラムコードのライン番号に基づいた変数の依存をチェックするものである。さらに、解析段階は関数対時間のグラフを含むが、これは個々の関数にかかる時間を表したものである。なお、この解析を通して並列処理の対象とする関数が判定される。時間の大半を使う関数又はモジュールは並列処理に最も適している。さらに、得た情報はその後の検索のために共有メモリ等の自動並列処理システムのメモリに記憶される。

【００２９】

コールグラフを作成してＳＰＩを挿入する目的は、オリジナルの逐次コンピュータプログラムコードの実行パターンによって実行を開始する実際の時間より前にコンピュータプログラムコードを実行し始めるためである。別の好適な実施形態は、必要に応じて冗長コンピュータプログラムコードを実行することを含む（例えばif-else文やスイッチケース文があるが、それらに限定されない）。これらにより、データの依存関係及びデータ変数の更新の統計データの記録の解析に基づいて、逐次実行時間（ＳＥＴ）より前に非逐次プログラムコードを実行することになる。さらに、依存関係とタイミング情報を自動的に検出して、並列実行に備えて並列処理できる可能性のあるコードに対するモジュールのタイミングを識別し解析する。

【００３０】

さらに、入力逐次コードをプロファイリングする間に判定するコールグラフは、全ての関数又はモジュールが明確な「呼び出し順」に従ったものをグラフで示したものであるが、そのシーケンスは、コンピュータプログラムコードに含まれるプログラム論理の部分と、より早い関数又はモジュールの実行を介してデータ変数を更新する必要があるかどうかに関する依存関係において判定される。このデータ変数の依存関係とプログラム論理のプロセスフローを解析することにより、コールチャートを作成する。コールグラフは、対象となる逐次コードを実行する実際のタイムスケールにおいて異なるモジュールを実行するシーケンスをグラフで示す。コールチャートを判定する目的は、逐次コンピュータプログラムコード内の個別のモジュールの実行を始めるのに必要な最短時間を判定することである。さらに、修正コードを一旦生成すれば、並列計算環境での非逐次実行に動作上適応させることができる。

【００３１】

別の実施形態では、コンピュータプログラムコードの性質とデータ依存統計値を厳密に評価することにより、コンピュータプログラムを実行するための絶対最短待機時間（ＡＭＷＴ）を判定する。これには入力された逐次コンピュータソースコードを少なくとも一度実行する必要があり、下流のコンピュータプログラムコードの実行を開始するのに必要な時間の短縮量を識別する毎にデータプロファイリングを行う。時間測定を評価するこの方法を反復して行うことで、下流のコンピュータプログラムコードを実行開始できる最短時間が判定されることになる。

【００３２】

別の実施形態では、特定のデータ変数の値がその後変わっていない逐次コンピュータプログラムコード内の時間及びライン番号の重要な組合せを判定することは、下流の関数又はモジュールの「事前」実行を判定する際に有益であり、これについて説明する。従って、データ変数のそれぞれの臨界点における定常状態値はメモリに記憶され、下流のモジュールが必要とする際にメモリから呼び出されることになる。これにより、逐次実行に対してコード化したモジュールを個別に実行するように論理を展開することができる。

【００３３】

一例示的な実施形態では、対応する関数における真の無駄時間又は真の最も早い時間を判定する反復プロセスが実行され得る。よって、本発明は個別の関数の真の無駄時間を判

定する反復方法を記載する。実行に必要と識別した時間をコールグラフに重ねることによって、逐次コンピュータプログラムコードの各関数の無駄時間が判定される。

【００３４】

例えば、図３に示すように入力チャネル及び出力チャネルを備えたチップと同義のコード中の各関数又はモジュールについて検討する。同様に、それぞれの関数は入力引数と幾つかの出力引数を有する。Ｘ１、Ｘ２、及びＸ３、並びにＫ１、Ｋ２、及びＫ３は関数に入力される変数であり、関数の出力をｙ１、ｙ２、．．ｙｎとする。ｙｉは他の何らかの関数に入力されるものであり、この関数と同様の方法で処理されることができる。関数の使用するこれらの引数と変数を識別する。「学習モード」において、アプリケーションのいくつかの実行中にこれらの変数を観測する。これらの観測によって、関数又はモジュールコードの一部が時間前に実行される。同様の状況は大まかな並列処理の場合にも当てはまるが、ここでアプリケーションのコールグラフについて考察する。各関数は何らかの他の関数（１つ又は複数）を呼び出す。このグラフとプロファイリング情報から、別個のコアで独立した関数全体を時間前に、又は別個のコアで関数コードの一部を実行することができる。関係するステップは、データの依存関係に関するグラフの描画、コールグラフの描画、データ依存関係グラフのコールグラフへの埋め込み、関数の呼び出し前に変数が定常状態に到達してからの時間であるＴｉを見つけるプログラムの実行、そして所与の関数に対する全てのＴｉの最小値を見つけること、である。

【００３５】

Ｆ１を時間ｔ１の逐次処理において呼び出す。
Ｆ１は時間（ｔ－T_{1min}）において実行することができる。
Ｆ２は逐次処理における時間ｔ１＝（ｔ＋Ｆ１がＦ２を呼び出すのに必要な時間）で呼び出される。
しかし、Ｆ２は（ｔ１－T_{2min}）＝ｔ２で実行することができる。

【００３６】

Ｆ２は異なるプロセッサで実行できる。ｔ２に達すると、利用可能なプロセッサを見つけてＦ２を実行させる。なお、プロセッサの利用可能性を見つけるのはスケジューリング手段である。

【００３７】

このように、本発明は逐次コンピュータプログラムコードの最適な並列処理のために必要なプロセッサの最適な数を判定する。かかる実行には、様々なプロセッサにおける各タスク又はその一部をスケジューリングすることが含まれ、費用及び時間の節約になる。

【００３８】

実際のケースでは、コンピュータプログラムコードを並列実行するのにＮ個のプロセッサがない場合がある。例えば、プロセッサが４つしかない場合：
１番目はマスター＝プロセッサ番号０
２番目にＦ１を実行＝プロセッサ番号１
３番目にＦ２を実行=プロセッサ番号２
４番目にＦ３を実行＝プロセッサ番号３、である。

【００３９】

別の関数Ｆ４を実行する状態にありながら、利用可能なプロセッサがない場合には、無駄時間（T_{dead}）を計算する。各関数はそれ自身のインデックス（＝関数における並列処理コードのインデックスセクションの番号）を有する。インデックスは、コールグラフにおけるその関数の近傍にある各関数の実行時間にも依存する。

【００４０】

例えば関数Ｆ１、Ｆ２、及びＦ３の実行にはそれぞれ、α_1、α_2、α_3の時間がかかるとする。Ｆ４を実行できる状態であるが、コア１、２、３はＦ１、Ｆ２、Ｆ３の実行でビジー状態である。Ｆ４はＦ１、Ｆ２、Ｆ３それぞれにおけるα_1、α_2、α_3の最短時間の間、Ｆ４の実行に利用できるコアを得るために待機しなければならない。このα_{min}が無駄時間となる。

【００４１】
上の状況を解決する方法の１つは、Ｆ１、Ｆ２、Ｆ３、Ｆ４の実行にかかる専用の時間を見つけて、最長の実行時間を見つけることである。Ｆ４の実行には（Ｆ１＋Ｆ２＋Ｆ３）を実行する最長時間がかかる、又はＦ１／Ｆ２／Ｆ３の任意の組み合わせを同時に実行しなければならないと仮定する。
【００４２】
詳細な記載における説明をさらに詳細に述べるために、本発明を実行するアプリケーションを説明することのみを目的とし、いくつかの例を述べることにする。当業者には、以下に示す例は本発明の全てを網羅しているわけではなく、本発明の範囲を制限しないことを理解されたい。
【００４３】
例１：逐次コード化によって事前に行うコードの実行
データ依存関係に関してコードを解析すると、事前実行を開始できるライン番号を判定し得る。以下のような構造を定義し得る：
Execute_on_priority_currentprocess_linestart_lineend_wait_linenumberEnd_nextprocess
【００４４】
ここでライン番号１００が現在実行中であるが、１つのループであってライン１０１から３４９に依存しないライン番号３５０から３８０までの実行を開始する必要がある、と仮定する。
【００４５】
上記の場合、
execute_on_priority_P2_350_380_wait_349_P1
と構築される。Ｐ１は現在のプロセスであり、ここから第２プロセスＰ２にジャンプする（この場合にはＦＯＲループ）。
【００４６】
例２：if-else文に対する並列処理方法
if-else文を見つける場合にはいつでも、コードを解析して実際の実行前に変数の依存関係を見つけるが、これはこのライン番号又は位置以降if-else文内で使用される変数が更新されていないことを意味する。多くの場合、if及びelseブロック内で使用される変数は、実際のif及びelseが始まるかなり前に最終更新された可能性がある。しかしif-elseの実際の実行を決める条件チェックは最新の変数で行われない。本発明では、if及びelse両方を実行して、条件文が準備できていれば、if又はelseのいずれかの結果を、条件文の結果に基づいて選択する。以下に示すコードセグメントでは、if-else依存関係はライン番号５で終了するため、ライン番号６からのif及びelseの実行を開始することができる。ライン番号７以降にのみif_condが見られるため、コアの１つでif部分（ライン９）の実行を開始し、他のものでelse部分（ライン１１）の実行を開始する。メインコアはライン６－７の実行を続ける。ライン７が実行され、if_condの値がわかれば、関連するコアからの結果が採用され、他の結果は破棄される。
【００４７】
```
1. module main start
2. variables if_var, else_var,a,b, if_cond
3. if_var = prime()
4. else_var_prime1()
5. comment: No dependency after this line
6. Factorial(a,b)
7. If_cond = mod3()
8. If(if_cond) then
9. comment: if_var used here
10. else
```

11. comment: else_var used here
12. end if
13. module main end

【００４８】

例３：スイッチケース文に対する並列処理方法

同様の方法をスイッチケース分に対して使用できる。スイッチケース文の依存関係が終了する位置において、異なるケースの実行を異なるコアに対してスケジューリングすることができる。ケース文に対する条件変数がわかれば、関連する結果を保持することができるため、その他の結果は破棄すべきである。

【００４９】

要約すると、if-else文、スイッチケース文等（ただしそれらに限定されない）の逐次コンピュータプログラムコードにおける制御フロー文を並列処理する方法は、制御フロー文におけるデータ変数を識別するステップと、制御フロー文内の前記データ変数が最後に更新されたライン番号及び時間を識別するステップと、識別したライン番号を電子レポジトリに記憶するステップと、前記記憶したデータを用いてライン及び時間の依存関係を示すグラフをプロットし、前記データ変数を更新するステップと、ライン及び時間の依存関係の統計を使用して、関数全体又は関数の一部を識別して異なるプロセッサに送り、逐次コンピュータプログラムコードの並列処理を行うステップと、プロセッサによる要求があった際に前記メモリから呼び出すために変数名又はアドレスを使用して、前記制御文の内部並列処理を行うステップと、を含む。

【００５０】

例４：ループに対する並列処理方法

時間前にループを実行するために、メインコアが現行コードを実行している間に、別のプロセッサコアでループの終了部分で使用される変数の依存関係をチェックし、ループ全体を実行する。例えば、ループがライン番号１００において変数を使用し、その変数がライン番号３０で最後に更新されたものであれば、ライン番号３１から先は事前にループを実行することができる。よって、ライン番号３１から９９を１つのプロセッサで実行し、それと同時にライン１００において始まるループを別のプロセッサで実行する。

【００５１】

要約すれば、逐次コンピュータプログラムコードにおけるループを並列処理する方法は、ループ文中のデータ変数を識別するステップと、文同士の依存関係をチェックするステップと、逐次コンピュータプログラムコード内の前記データ変数が最後に更新されたライン番号を識別するステップと、識別したライン番号をメモリに記憶するステップと、前記記憶したデータを使用して、前記データ変数の更新に関するライン依存関係を示すグラフをプロットするステップと、前記ライン依存関係の統計を使用して、コードの特定のセグメントを識別して異なるプロセッサに送り、逐次コンピュータプログラムコードの並列処理を行うステップと、を含む。

【００５２】

例５：反復プロセス

前掲の例のうちの一つで述べたモジュールＦ３について検討する。T_{3min}は、変数の依存関係がない間のＦ３の実行時間とする。モジュールＦ１及びモジュールＦ２がプロセッサ及び利用可能性に基づいてスケジューリングされるとT_{3min}がT_{3min}’に変わる。ここで$T_{3min} < T_{3min}$’である。

【００５３】

これで第２の反復とT_{1min}’、T_{2min}’、T_{3min}’等を計算する必要が始まる。これはＮ回反復され、全てのタイミングが無駄時間になると停止する。

【００５４】

例６：プログラムコードを実証する擬似コード

以下に示す擬似コードの例は、図４に示すようにマスクを使用したエッジ検出アルゴリズムの一例のコードである。想定実行－事前実行の基礎を、以下の例で説明する。なお、

以下の例は多数の変数を考慮したものではないため、例としては完全なものではない。

【００５５】

```
1. module main start
2. datastructs originalImage, edgeImage, filterImage
3. variables X, Y, I, J, sumX, nColors, p, q, vectorSize
4. variables fileSize, mask, row, col, origcols, origrows
5. variables coeffhighpass, coefflowpass, Filttype, sum
6. q='0'
7. mask =(-1,0,1,-2,0,2,-1,0,1)
8. bmpInput = fileopen(mode, “ in “ )
9. bmpOutput = fileopen(mode, “ out “ )
10. fileSize = getImageInfo()
11. originalImage.cols = getImageInfo()
12. originalImage.rows = getImageInfo()
13. edgeImage.rows = originalImage.rows
14. edgeImage.cols = originalImage.cols
15. loop from row =0 to row <= originalImage.rows-1
16. loop from col =0 to col <= originalImage.cols -1
17. pchar = read each pixel from bmpInput
18. originalImage.data = pChar
19. emd loop
20. end loop
21. nColors = getImageInfo()
22. vectorSize = fileSize -(14+40+4*nColors)
23. copyImageInfo (bmpInput, bmpOutput)
24. loop from Y=0 to Y<= originalImage.rows-1
25. loop from X=0 to X<= originalImage.cols-1
26. sumX =0
27. loop from I=-l to I<=1
28. loop from J=-l to J<=1
29. sumX = sumX + originalImage.data +X+I+
30. (Y+J)*originalImage.cols))*mask[I+1][J+1]
31. end loop
32. end loop
33. if sumX >255 then
34. sumX =255
35. Filttype = low
36. endif
37. if sumX <0 then
38. sumX =0
39. Filttype = high
40. endif
41. edgeImage.data +X+Y*originalImage.cols = 255 - sumX
42. end loop
43. end loop
44. if (Filetpe = low) then
45. filterImage = ApplyHighPassFilter(originalImage)
46. Else
47. filterImage = ApplyLowPassFilter(originalImage)
48. endif
```

```
49. loop from Y=0 to Y<=originalImage.rows-1
50. loop from X=0 to X<=originalImage.cols-1
51. resultImage.data = filterImage.data + EdgeImage.data
52. end loop
53. end loop
54. module main ends
55. function getImageInfo start
56. variables        numberOfChars ,value, dummy, i
57. dummy ='0'
58. loop form i=1 to i<= numberOfChars
59. value = read value from file
60. end loop
61. module getImageInfo end
62. module copyImageInfo
63. variables       numberOfChars ,value, dummy, i
64. dummy ='0'
65. loop from i=0 to i<=50
66. value = read value from file
67. bmpOutput = write 'value'
68. end loop
69. module copyImageInfo end
70. module ApplyHighPassFilter
71. loop from ht =0 to row
72. loop form wd =0 to col
73. sum =0
74. loop from c =0 to coeffhighpass
75. sum = sum + coeffliighpass * originalImage.data
76. end loop
77. filterImage.data = sum
78. end loop
79. end loop
80. funciton ApplyHighPassFilter end
81. module ApplyLowPassFilter
82. loop from ht =0 to row
83. loop form wd =0 to col
84. sum =0
85. loop from c =0 to coefflowpass
86. sum = sum + coefflowpass * originalImage.data
87. end loop
88. filterImage.data = sum
89. end loop
90. end loop
91. funciton ApplyLowPassFilter end
```

【００５６】

事前実行：

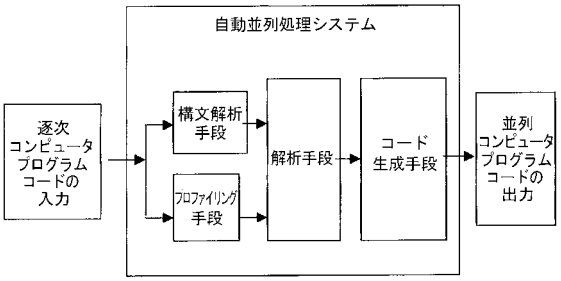
変数‘Original Image rows’について検討すると、上記コードを解析することにより、この変数はライン番号１２で更新されており、その後ライン番号１３、１５、２４、５０で使用されている。ライン番号２４におけるループで使用される変数はいずれも、ライン１３から２３までで更新された変数を使用しない。そのため、上記コードがライン１３から２３を実行している間、ライン２４から３２におけるループを別のプロセッサで並列

実行する。よって、ライン番号２５から３１におけるループの実行開始は、事前実行されることになる。

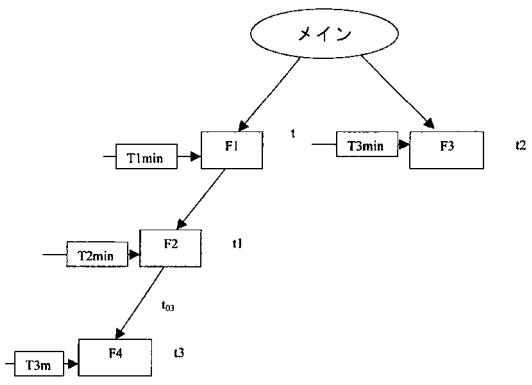
【００５７】

本発明の代表的な例である前掲の特定の実施形態、実施例、及び図面を参照することで、本発明をより容易に理解できる。ただし、それらは例示の目的で示したものであり、本発明はその主旨及び範囲を逸脱することなく、特定的に示した形態とは別の形態で実行可能であることは理解されよう。認識される通り、本発明は様々な他の実施形態が可能であり、そのいくつかの要素と関連する詳細は、すべて本発明の基本的な概念から逸脱することなく、種々の変更が可能である。従って、本明細書の記載は本質的に例示であり、いずれの形態にも限定されない。本明細書に記載したシステムと装置の変更及び変形は、当業者には明らかであろう。それらの変更及び変形は、添付の特許請求の範囲に含まれることを意図している。

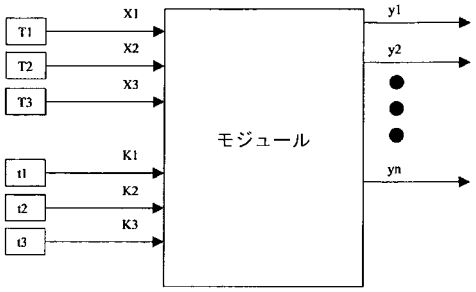
【図１】



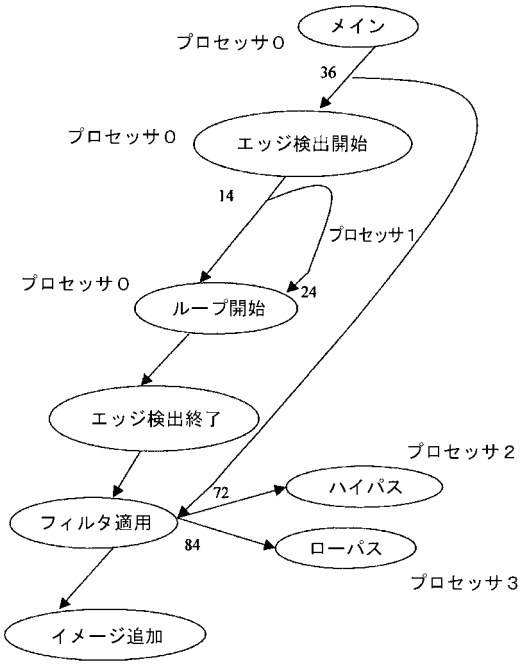
【図２】



【図３】



【図４】


プロセッサ０
メイン
36
プロセッサ０
エッジ検出開始
14
プロセッサ１
プロセッサ０
24
ループ開始
エッジ検出終了
プロセッサ２
ハイパス
72
フィルタ適用
84
ローパス
プロセッサ３
イメージ追加

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IN 09/00697

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 15/16 (2010.01)
USPC - 712/32
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
USPC: 712/32

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC: 712/1, 28, 32, 214, 220, 235, 245; 700/1, 2, 4, 7 (search term limited)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Electronic databases: PubWEST(PGBP, USPT, USOC, EPAB, JPAB); Google Scholar; Freepatentsonline.com
Seach Terms Used: parallelization, multi-threading, sequential, linear, serial, wait, dead, quiescient, timing, dependency, syntactical, analysis, sequence, chronometry, graphing, clustering, call graph etc.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2007/0192766 A1 (PADALIA et al.) 16 August 2007 (16.08.2007) Entire document, especially: para [0022], [0069]-[0071], [0074], [0076], [0087]-[0089], [0093]-[0096]	1-15
Y	US 2007/0011684 A1 (DU et al.) 11 January 2007 (11.01.2007) Entire document, especially: [0004], [0011], [0013], [0016], [0018], [0019], [0022]-[0026], [0036], [0037], [0045], [0049]-[0050], [0053]-[0054], [0057]-[0059] and Fig. 6, 7	1-15
Y	US 2006/0200796 A1 (OTA et al.) 07 September 2006 (07.09.2006) Entire document, especially: [0008], [0049], [0052], [0057], [0081], [0118], [0120], [0124], [0132], [0136]	9-10
A	US 2007/0079281 A1 (LIAO et al.) 05 April 2007 (05.04.2007)	1-15
A	US 7,159,211 B2 (JALAN et al) 02 January 2007 (02.01.2007)	1-15
A	An article entitled "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks", (MICHAEL ISARD et al.,), EuroSys-07, March 21-23, 2007, Lisboa, Portugal, [retried 24 April 2010 (24.04.2010)] Retrieved from the Internet. <URL: http://www.cs.brandeis.edu/~olga/cs228/Reading%20List_files/dryad.pdf>	1-15

☐ Further documents are listed in the continuation of Box C. ☐

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
24 April 2010 (24.04.2010)	29 APR 2010
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-3201	Authorized officer: Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (July 2009)

フロントページの続き

(81)指定国 AP(BW,GH,GM,KE,LS,MW,MZ,NA,SD,SL,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,BG,CH,CY,CZ,DE,DK,EE,ES,FI,FR,GB,GR,HR,HU,IE,IS,IT,LT,LU,LV,MC,MK,MT,NL,NO,PL,PT,RO,SE,SI,SK,SM,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AO,AT,AU,AZ,BA,BB,BG,BH,BR,BW,BY,BZ,CA,CH,CL,CN,CO,CR,CU,CZ,DE,DK,DM,DO,DZ,EC,EE,EG,ES,FI,GB,GD,GE,GH,GM,GT,HN,HR,HU,ID,IL,IN,IS,JP,KE,KG,KM,KN,KP,KR,KZ,LA,LC,LK,LR,LS,LT,LU,LY,MA,MD,ME,MG,MK,MN,MW,MX,MY,MZ,NA,NG,NI,NO,NZ,OM,PE,PG,PH,PL,PT,RO,RS,RU,SC,SD,SE,SG,SK,SL,SM,ST,SV,SY,TJ,TM,TN,TR,TT,TZ,UA,UG,US,UZ,VC,VN,ZA,ZM,ZW

(72)発明者 プリティ ラナディヴ
インド, ４１１ ０５７ プネ, ヒンジャワディ, エムアイディシィ, フェーズ １, ３５ アンド ３６ ラジーブ ガンディ インフォテック パーク
(72)発明者 スダカー サー
インド, ４１１ ０５７ プネ, ヒンジャワディ, エムアイディシィ, フェーズ １, ３５ アンド ３６ ラジーブ ガンディ インフォテック パーク

Ｆターム(参考) 5B057 CA08 CA12 CA16 CB08 CB12 CB16 CC02 CE06 DC16
5B081 CC32 CC64