(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第6319880号
(P6319880)

(45)発行日　平成30年5月9日(2018.5.9)　(24)登録日　平成30年4月13日(2018.4.13)

(51)Int.Cl.　F I

G06F 8/41	(2018.01)	G06F	9/44	322F
G06F 9/50	(2006.01)	G06F	9/46	465E

請求項の数 9　(全 21 頁)

(21)出願番号　特願2014-6009(P2014-6009)
(22)出願日　平成26年1月16日(2014.1.16)
(65)公開番号　特開2014-160453(P2014-160453A)
(43)公開日　平成26年9月4日(2014.9.4)
審査請求日　平成28年12月6日(2016.12.6)
(31)優先権主張番号　特願2013-10194(P2013-10194)
(32)優先日　平成25年1月23日(2013.1.23)
(33)優先権主張国　日本国(JP)

(73)特許権者　899000068
学校法人早稲田大学
東京都新宿区戸塚町１丁目１０４番地
(74)代理人　110000800
特許業務法人創成国際特許事務所
(72)発明者　笠原　博徳
東京都新宿区大久保３－４－１　早稲田大学理工学術院基幹理工学部情報理工学科内
(72)発明者　木村　啓二
東京都新宿区大久保３－４－１　早稲田大学理工学術院基幹理工学部情報理工学科内
(72)発明者　林　明宏
東京都新宿区大久保３－４－１　早稲田大学理工学術院基幹理工学部情報理工学科内

最終頁に続く

(54)【発明の名称】並列性の抽出方法及びプログラムの作成方法

(57)【特許請求の範囲】
【請求項１】
元プログラムの並列性をコンピュータによって抽出する方法であって、
前記方法は、
前記元プログラムを、複数のマクロタスクに分割する処理と、
前記複数のマクロタスクの最早実行可能条件を解析する処理と、
前記最早実行可能条件の解析結果に基づいて、条件分岐の同一の分岐方向に制御依存する複数のマクロタスクである複数の後続マクロタスクの相互の間で並列実行可能な前記条件分岐を抽出する処理と、
抽出された前記条件分岐を複製することにより前記条件分岐をそれぞれ含む複数のマクロタスクである複数の先行マクロタスクを生成する処理と、を含むことを特徴とする並列性の抽出方法。
【請求項２】
請求項１に記載の並列性の抽出方法であって、
前記条件分岐として、前記複数の後続マクロタスクの相互の間でデータ依存がない条件分岐を抽出することを特徴とする並列性の抽出方法。
【請求項３】
請求項１又は２に記載の並列性の抽出方法であって、
前記元プログラムから分割された前記複数のマクロタスクを並列に実行するための複数のパスのうち、実行時間が最も長いパスに含まれる条件分岐を複製することを特徴とする

並列性の抽出方法。
【請求項４】
請求項１～３のうちいずれか１項記載の並列性の抽出方法であって、
前記元プログラムから分割された前記複数のマクロタスクを並列に実行するための複数のパスのうち、実行時間が最も長いパスに含まれる条件分岐を複製することによって生成された複数の先行マクロタスクに繋がるパスの中で実行時間が最も長いパスに含まれる条件分岐をさらに複製することを特徴とする並列性の抽出方法。
【請求項５】
請求項１～４のうちいずれか１項記載の並列性の抽出方法であって、
前記複数の後続マクロタスクのうちの第１後続マクロタスクを前記複数の先行マクロタスクのうちの１つの第１先行マクロタスクに後続させ、
前記複数の後続マクロタスクのうちの前記第１後続マクロタスクと異なる第２後続マクロタスクを前記複数の先行マクロタスクのうちの前記第１先行マクロタスクとは異なる第２先行マクロタスクに後続させることを特徴とする並列性の抽出方法。
【請求項６】
請求項１～５のうちいずれか１項記載の並列性の抽出方法であって、
前記複数の後続マクロタスクとは異なる一のマクロタスクにデータ依存し、かつ、前記複数の後続マクロタスクに含まれる第３後続マクロタスクと、前記一のマクロタスクにデータ依存せず、かつ、前記複数の後続マクロタスクに含まれる第４後続マクロタスクとが存在する場合、前記第４後続マクロタスクを、前記一のマクロタスク及び前記第３後続マクロタスクが含まれるパスとは異なるパスに含まれるように、前記複数の先行マクロタスクに前記複数の後続マクロタスクを後続させることを特徴する並列性の抽出方法。
【請求項７】
請求項１～６のうちいずれか１項記載の並列性の抽出方法であって、
前記条件分岐は、実行時の不確定性を有することを特徴とする並列性の抽出方法。
【請求項８】
請求項１～７のうちいずれか１項記載の並列性の抽出方法であって、
前記条件分岐は、ループ制御文とは異なることを特徴とする並列性の抽出方法。
【請求項９】
元プログラムからコンピュータによって複数のプロセッサコアを含むマルチコアプロセッサ上で実行可能なプログラムを作成する方法であって、
請求項１～８のうちいずれか１項記載の並列性の抽出方法を用いて生成された複数の条件分岐をそれぞれ含む複数の先行マクロタスクを、前記複数のプロセッサコアの各々が実行するように割り当て、
前記複数の条件分岐のそれぞれを含む先行マクロタスクを実行するプロセッサコアが、それぞれ、前記複数の後続マクロタスクの各々を実行するように割り当てることによって、前記実行可能なプログラムを作成する方法。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、逐次処理の元プログラムに潜在する並列性を抽出する方法、及びその並列性の抽出方法を用いて複数のプロセッサコアによって構成されるマルチコアプロセッサ上で実行される並列化プログラムを作成する方法に関する。
【背景技術】
【０００２】
複数のプロセッサコアを集積したマルチコアプロセッサが、各マイクロプロセッサメーカによって次々に発表されている。スーパーコンピュータ、サーバ、デスクトップコンピュータ及びＰＣサーバ分野の他、情報家電及び装置組み込みの分野（例えば、携帯電話機、ゲーム機、カーナビゲーションシステム、デジタルテレビ受像機、ＨＤＤ／ＤＶＤレコーダ・プレーヤ等）においても、マイクロプロセッサのマルチコア化の動きが見られる。

【０００３】
また、より安全、快適、省エネを実現する次世代自動車の開発のため、エンジン制御のようなリアルタイム制御系、人認識・他車認識のような外界認識、運転に必要な情報の提示、音楽・映像等を提示する情報系、制御系と情報系を統合して、制御する統合制御系、それぞれの高度化が重要となっている。
【０００４】
これらの制御系、情報系、および統合制御系の高度化のためには、プロセッサの高能力化が重要となる。例えば、安全、快適、省エネな自動車開発のために重要なエンジン制御系を高度化するためには、制御アルゴリズムの高度化、新制御機能の実現など計算負荷の増大を避けられない。
【先行技術文献】
【特許文献】
【０００５】
【特許文献１】特開２００１－１７５６１９号公報
【非特許文献】
【０００６】
【非特許文献１】Seo, K.,etal, Coordinated implementation and processing of a unified chassis control algorithm with multi-central processing unit, J AUT01346, Vol. 224 (2009)
【非特許文献２】Seo, K.,etal: An Investigation into Multi-Core Architecturesto Improve a Proccssing Performancc of the Unified Chassis Control Algorithms, S AE Int.J.Passeng.Cars-Elect1on.Electr.Syst., Vol.3, pp.53-62 (2010)
【非特許文献３】OSEK/VDX-Portal:http://portal.osek-vdx.org/
【非特許文献４】Kasahara, H ., etal: Automatic Coarse Grain Task Parallel Processing on SMP using Open MP, Proc. of The 13th International Workshop on Languages and Compilers for Parallel Computing(LCPC2000) (2000)
【非特許文献５】Y. Yuyama, et al., A 45nm 37.3GOPS/W Heterogeneous Multi-Core SoC, ISSCC2010
【非特許文献６】H. Kasahara,etal, OSCAR FORTRAN COMPILER, 1991International Logic Programming Symposium, Workshop on Compilation of (simbolic) Languages for parallel Computers, Oct.31-Nov.1, 1991, San Diego, U.S.A.
【発明の概要】
【発明が解決しようとする課題】
【０００７】
このようなエンジン制御系の高度化の要求に伴う計算負荷の増大という問題を解決するためには、エンジン制御に用いられるプロセッサの高能力化が必須となる。
【０００８】
従来は、プロセッサの高能力化のためにプロセッサの動作周波数を上げることが必要である。しかし、消費電力は動作周波数の三乗に比例して増大するために消費電力の大幅な増加を招くこと、また、自動車における過酷な動作環境下でも安定動作を保証する必要があることなどのため、自動車などの装置に適用することは困難である。このため１チップ上に複数の低動作周波数プロセッサコアを集積し、低周波数化・低電圧化したプロセッサコアを並列動作させることにより、処理の高速化と低消費電力化とを同時に実現可能なマルチコアプロセッサへの移行が求められている。
【０００９】
このような要求に対し、自動車業界におけるマルチコアを利用した技術が提案されている。例えば、非特許文献１及び非特許文献２に記載されるように、統合シャシ制御（ＵＣＣ）アルゴリズムを利用したマルチコアアーキテクチャでは、マルチコアプロセッサを用いた電子制御ユニットが提案されている。このマルチコアプロセッサは三つのプロセッサコアで構成され、電子制御ユニットを三つの機能に分割し、その各々を前記三つのプロセ

ッサコアに割り当てることにより機能を分散している。この方法は、いわゆるＡＭＰ（Asymmetric Multicore Processing）と言われており、機能分散を実現することによってスループットを向上することが可能であるが、レイテンシの削減が困難である。また、機能毎の負荷バランスが均等でないとマルチコア資源を最大限に活用することができないという問題がある。

【００１０】

本発明では、自動車制御系の主要機能であるエンジン制御に対し、後述するように並列処理を適用し、レイテンシの削減すなわち高速化する方法を提案する。このようなマルチコア上で、エンジン制御等の計算を、従来の一つのプロセッサコア上での処理より高速に行うためには、計算を分割し、計算負荷を複数のプロセッサに適切に割り当てて、計算を実行する方法、すなわちＳＭＰ（Symmetric Multicore Processing）が重要となる。このようなＳＭＰという計算実行方法は一般的に並列処理と呼ばれ、この並列処理のためのプログラムを並列化プログラムと称する。そして、１プロセッサ上で動作する逐次処理の元プログラムから並列化プログラムを作成あるいは生成することをプログラムの並列化と称する。

【００１１】

しかし、このプログラムの並列化を人手で行うことには、開発期間の増大、それに伴うソフトウェア開発費の増大、さらには並列化プログラムの信頼性の低下など、大きな問題があった。この問題を解決するために、並列化プログラムを逐次処理のプログラムから自動生成するためのソフトウェアに関する研究が行われてきた。

【００１２】

従来の並列化の方法として、発明者等が研究開発を行ってきたマルチグレイン並列処理を行うコンパイラ、通称ＯＳＣＡＲコンパイラが知られている。以下にＯＳＣＡＲコンパイラの一般的な機能について説明する。コンパイラとは、一般的にはコンパイル対象のソースプログラムをいわゆる計算機が実行可能なプログラム（機械語のプログラム等）に変換するプログラムを指すが、以下に説明する並列化コンパイラ（ＯＳＣＡＲコンパイラ）は逐次処理プログラムのソースコードから並列化プログラムのソースコード（場合によってはオブジェクトコード）を生成する機能を持つ。

【００１３】

まず、ＯＳＣＡＲコンパイラの特徴であるマルチグレイン並列処理とは、元プログラムに存在する、ループやサブルーチン等の粗粒度タスク間の並列性を利用する粗粒度タスク並列処理、ループイタレーションレベルの並列性を利用する中粒度並列処理、基本ブロック内部のステートメントレベルの並列性を利用する近細粒度並列処理を階層的に組み合わせてプログラム全域にわたって行う並列処理である。その詳細は非特許文献４に詳述されているが、以下に図１７のフローチャートを用いてその概要を説明する。

【００１４】

粗粒度並列処理では、コンパイラは、図１７のフローチャートにおける１９０１の「軸解析・構文解析」のステップ、ステップ１９０２の「マクロタスク生成」のステップを経て、ソースとなる元プログラムを疑似代入文ブロック（ＢＰＡ）、繰り返しブロック（ＲＢ）、サブルーチンブロック（ＳＢ）の三種類の粗粒度タスク（以下、「マクロタスク（ＭＴ）」と称する。）に分割する。ＭＴ生成後、コンパイラは、ステップ１９０３の「制御フロー解析」およびステップ１９０４の「データ依存解析」を経て、ＢＰＡ、ＲＢ、ＳＢ、等のＭＴ間の制御フロー及びデータ依存関係を表現したマクロフローグラフ（以下、「ＭＦＧ」と略称）を生成する（ステップ１９０５）。さらに、ステップ１９０６の「データアクセス範囲解析」およびステップ１９０７の「最早実行可能条件解析」を経て、ＭＦＧからＭＴ間の並列性を抽出した結果をマクロタスクグラフ（以下、「ＭＴＧ」と略称）として表現する（ステップ１９０８）。その後コンパイラは、ステップ１９０７の「各種のリストラクチャリング」において、タスク融合（非特許文献６を参照）、キャッシュメモリの最適化、ローカルメモリ管理などのための各種のリストラクチャリングを行う。このリストラクチャリングのステップの処理の後、ステップ１９０２に処理を戻し、ステ

ップ１９０２から１９０７までのステップにおいてプログラムの再解析を繰り返すが、再解析の必要がなくなれば、ステップ１９１０の「タスクスケジューリング」に進み、そのステップにおいてＭＴＧ上のＭＴを、少なくとも一つのプロセッサコア（ＰＥ）をグルーピングしたプロセッサグループ（ＰＧ）に割り当てるスケジューリング処理を行い、ステップ１９１１の「並列化プログラムの生成」において、前記スケジューリング処理の情報も含む並列化プログラムを生成する。

【００１５】

以上のようなコンパイラの並列化プログラム生成のための処理の過程で生成される前記のＭＦＧの例を図１８Ａに示し、前記のＭＴＧの例を図１８Ｂに示す。ＭＦＧにおいてノードはＭＴを表し、実線エッジはデータ依存を、点線エッジは制御フローを表す。また、ノード内の小円は条件分岐を表す。ＭＴＧにおけるノードもＭＦＧと同様にＭＴを表し、ノード内の小円はＭＴ内の条件分岐を表す。また、実線エッジはデータ依存を表し、点線エッジは拡張された制御依存を表す。拡張された制御依存とは、通常の制御依存だけではなく、データ依存と制御依存を複合的に満足させるため、先行ノードが実行されることを確定する条件分岐を含む。また、エッジを束ねるアークには２つの意味があり、実線アークはアークによって束ねられたエッジがＡＮＤ関係にあることを、点線アークは束ねられたエッジがＯＲ関係にあることを表す。ＭＴＧにおいてエッジの矢印は省略したが、下向きである。また、矢印を持つエッジはオリジナルの制御フローを表す。

【００１６】

粗粒度タスク並列処理では、各階層で生成されたマクロタスクはＰＧに割り当てられて実行される。どのＰＧにマクロタスクを割り当てるかを決定するスケジューリング方法として、ダイナミックスケジューリングとスタティックスケジューリングとがあり、そのいずれを選択するかは、並列化コンパイラがＭＴＧの形状、実行時非決定性などに基づいて決定する。

【００１７】

条件分岐などの実行時の不確定性が存在する場合には、ダイナミックスケジューリングが適しており、ダイナミックスケジューリングによって実行時にマクロタスクをＰＧに割り当てる。ダイナミックスケジューリングルーチンは、マクロタスクの終了又は分岐方向の決定に応じて、マクロタスク実行管理テーブルを操作し、各マクロタスクの最早実行可能条件を探索する。マクロタスクが実行可能であればレディキューにマクロタスクが投入される。レディキュー内のマクロタスクはその優先順位に従ってソートされ、レディキューの先頭のマクロタスクがアイドル状態のプロセッサコアに割り当てられる。また、ダイナミックスケジューリングコード生成時には、一つの専用のプロセッサがスケジューリングを行う集中スケジューリング方法と、スケジューリング機能を各プロセッサに分散した分散スケジューリング方法とを、使用するプロセッサ台数及びシステムの同期オーバーヘッドを考慮して、使い分けることができる。

【００１８】

一方、スタティックスケジューリングは、ＭＴＧがデータ依存エッジのみを持つ場合に適しており、自動並列化コンパイラがコンパイル時にマクロタスクのＰＧへの割り当てを決める方法である。スタティックスケジューリングでは、実行時スケジューリングオーバーヘッドを低減し、データ転送及び同期のオーバーヘッドを低減することが可能である。

【００１９】

並列化プログラムは、ＡＰＩを用いて、並列化Ｃあるいは Ｆｏｒｔｒａｎ７７によって ｓｏｕｒｃｅ－ｔｏ－ｓｏｕｒｃｅで生成することも可能である。この場合には、様々なプラットフォームにおいて実行可能な形にするため、例えば、ＯＳＣＡＲ　ＡＰＩ標準解釈系を用いて、ＡＰＩ部分をランタイムライブラリコールに変換した後、各プロセッサ用のコードを逐次コンパイラでコンパイルし、バイナリコードを生成してもよい。

【００２０】

以上、図１７のフローチャートを用いて説明した従来のＯＳＣＡＲコンパイラに、後述するような逐次処理のエンジン制御プログラムを元プログラムとして入力し、出力として

得られる並列化プログラムを後述するマルチコアプロセッサＲＰ－Ｘ（非特許文献５を参照）上で実行させてもコア数に見合った速度向上を実現することができなかった。そのため、発明者はその原因を詳細に分析した結果、元プログラムの特性に従来のＯＳＣＡＲコンパイラによる並列性の抽出方法が適応していないことが判明した。

【００２１】

その理由は以下のとおりである。前述したエンジン制御プログラムには条件分岐などが多いために、従来のＯＳＣＡＲコンパイラでは、条件分岐以降のマクロタスクの計算処理を複数個のプロセッサコアを含むマルチコアプロセッサ上で効率的に動作させるための並列化をスタティックスケジューリングによって行うことが困難であり、並列化プログラムの実行時にプロセッサコアの割り当てを行うダイナミックスケジューリングによらざるを得なかった。そのために、並列化プログラム実行時に分岐毎に別のプロセッサコアに処理を割り当てた場合、コア間で処理の同期のための制御オーバーヘッドが増加し、並列化による実行速度の向上が低かったことが判明した。

【００２２】

このため、本発明では、条件分岐を含むプログラムに潜在する並列性を抽出して、マルチコアプロセッサ上で実行するのに適する並列化プログラムを生成するための効果的な並列性の抽出方法を提案する。

【課題を解決するための手段】

【００２３】

本願において開示される発明の代表的な一例を示せば以下の通りである。すなわち、元プログラムの並列性をコンピュータによって抽出する方法であって、前記方法は、前記元プログラムを、複数のマクロタスクに分割する処理と、前記複数のマクロタスクの最早実行可能条件を解析する処理と、前記最早実行可能条件の解析結果に基づいて、条件分岐の同一の分岐方向に制御依存する複数のマクロタスクである複数の後続マクロタスクの相互の間で並列実行可能な前記条件分岐を抽出する処理と、抽出された前記条件分岐を複製することにより前記条件分岐をそれぞれ含む複数のマクロタスクである複数の先行マクロタスクを生成する処理と、を含む。

【発明の効果】

【００２４】

本発明の代表的な実施の形態によれば、条件分岐を含む元プログラムでも、適切に並列化をすることができる。また、制御プログラムのようにリアルタイム性が必要なプログラムにおいても、適切に並列化をすることができる。

【図面の簡単な説明】

【００２５】

【図１Ａ】本発明の並列性の抽出方法の実施形態を用いた並列化コンパイラの処理フローである。

【図１Ｂ】本発明の並列性の抽出方法の実施形態を用いた並列化コンパイラの処理ステップ２９０９の詳細フローである。

【図２】本発明の実施形態の並列化コンパイラを実行する計算機のブロック図である。

【図３】元プログラムであるエンジン制御プログラムの実行プロファイル結果を説明する図である。

【図４】Ｔａｓｋ１５ｍａｉｎのＭＴＧである。

【図５】Ｔａｓｋ１５ｍａｉｎ内の各マクロタスクの実行コスト（単位：クロック）を説明する図である。

【図６】本発明の実施形態の条件分岐内の並列性を説明する図である。

【図７】本発明の実施形態の条件分岐の複製を説明する図である。

【図８】本発明の実施形態のインライン展開・条件分岐の複製後のＭＦＧである。

【図９】本発明の実施形態のタスク融合を行った後のＭＴＧである。

【図１０】本発明の実施例に用いたＴａｓｋ１５ｍａｉｎを元プログラムとして生成した

並列化プログラムの性能評価に用いた組込用マルチコアプロセッサの構成を説明する図である。
【図１１】本発明の実施形態の実行プロファイル結果である。
【図１２】本発明の実施形態のシナリオ１の速度向上率を説明する図である。
【図１３】本発明の実施形態のシナリオ２の速度向上率を説明する図である。
【図１４】本発明の実施形態のシナリオ３の速度向上率を説明する図である。
【図１５】本発明の実施形態の説明のために挙げた条件分岐複製前のＭＴＧの例である。
【図１６】本発明の実施形態の説明のために挙げた条件分岐複製後のＭＴＧの例である。
【図１７】従来の並列化コンパイラの処理フローである。
【図１８Ａ】ＭＦＧの例を説明する図である。
【図１８Ｂ】ＭＴＧの例を説明する図である。
【発明を実施するための形態】
【００２６】
以下、本発明の実施形態について図面を用いて説明する。
【００２７】
本実施形態では、元プログラムとして、ループ処理がなく、条件分岐や代入文などの基本ブロックが連続していて従来の並列化方法では並列化が困難なエンジン制御プログラムにおける並列性の抽出方法およびその抽出方法を用いた並列化方法について説明する。なお、本発明は、本実施形態で説明するエンジン制御プログラムだけでなく、条件分岐や代入文などの基本ブロックが連続しているなどの特徴を有する他の（制御）プログラムの並列化にも適用することができる。
【００２８】
［コンパイル処理］
図１Ａは、本発明の実施形態の並列性の抽出方法を用いた並列化コンパイラ（以下「本発明の実施形態のコンパイラ」と略称）が実行する処理のフローチャートである。
【００２９】
ここでは、条件分岐の多い逐次処理の元プログラムから高性能の並列化プログラムを生成する本発明の実施形態のコンパイラが、計算機上で実行する並列化プログラム生成までの処理、なかんずく並列性の抽出処理について説明する。図１Ａに示す並列化コンパイラは、従来のＯＳＣＡＲコンパイラに比して、ステップ１９０７のリストラクチャリング機能に以下に説明する並列性の抽出のための条件分岐の複製を追加したことに特徴がある。従って、以下の説明において、ステップ１９０１から１９０８までおよびステップ１９１０、１９１１における処理は従来の並列化コンパイラに実装されている。また、本発明の並列性の抽出方法を用いた並列化コンパイラ（以下「本発明のコンパイラ」と略称）のステップ２９０９を「条件分岐の複製を含むリストラクチャリング」と称することとするが、ステップ２９０９は、従来の並列化コンパイラのステップ１９０７の「各種のリストラクチャリング」に本発明の特徴である条件分岐の複製の機能を追加したものである。以下にその具体的な内容および効果について説明する。
【００３０】
まず、並列化コンパイラは、ソースプログラムの字句を解析し、プログラムの構文を解析する（ステップ１９０１）。なお、本実施形態では、コンパイラがプログラムを解析しているが、簡単なプログラム等の場合はプログラマ（人）がプログラムを解析してもよい。また、ソースプログラムの作成時に、コンパイラに必要なプログラムの情報を作成してもよい。
【００３１】
次に、並列化コンパイラは、構文の解析結果に基づいて、プログラムの階層的マクロタスクによる表現を生成する（ステップ１９０２）。ここでは、粗粒度タスク（マクロタスク）を生成する。なお、生成されたマクロタスクがループイタレーションレベルの並列処理や逐次処理が可能な場合、ローカルメモリ等のメモリサイズを考慮して、異なる複数のマクロタスクに分割する（ループ整合分割）。

【００３２】
その後、生成されたマクロタスク間の制御依存関係（制御フロー）を解析し（ステップ１９０３）、マクロタスク間のデータ依存を解析し（ステップ１９０４）、その結果に基づきＭＦＧを生成する（ステップ１９０５）。次に、そのＭＦＧを基にして各マクロタスクによってアクセスされるデータの範囲を解析する（ステップ１９０６）。
【００３３】
その後、以上に述べたプログラムの解析結果を使用して、プログラムが最も早く実行できる最早実行条件を解析し（ステップ１９０７）、最早実行条件の解析結果を使用して、並列処理区間やマクロタスクを割り当てるプロセッサ数を決定し、ＭＴＧを生成する（ステップ１９０８）。
【００３４】
その後、同一の条件で実行が決定される複数のマクロタスクのうち、前記実行される複数のマクロタスクの相互の間で並列実行可能な（例えば、データ依存がない）条件分岐を抽出し、抽出された条件分岐を複製する（ステップ２９０９の「条件分岐の複製を含むリストラクチャリング」）。複製された条件分岐の中に、さらに複製すべき条件分岐があれば、複製後のプログラムを再度解析するためにステップ１９０２に処理を戻し、ステップ１９０２から１９０８までを実行した上で、ステップ２９０９を実行する。なお、ステップ２９０９は、条件分岐の複製及び他のリストラクチャリングがなくなるまで繰り返し実行される。
【００３５】
その後、各マクロタスクの実行順序を決定するタスクスケジューリングを実行する（１９１０）。ここで実行されるタスクスケジューリングは、メモリ管理・タスクスケジューリング、データ転送スケジューリング、及び低消費電力スケジューリングを含む。
【００３６】
メモリ管理・タスクスケジューリングとは、マルチコアプロセッサ中の各プロセッサコアのローカルメモリを介した効率的なデータの授受を実行するためのスケジューリングである。データ転送スケジューリングとは、プレロードやポストストアのように、各プロセッサコアにおけるデータ転送、及びマルチコアプロセッサ全体におけるデータ転送の最適化を実現するためのスケジューリングである。低消費電力スケジューリングとは、マルチコアプロセッサ中の汎用プロセッサ（ＣＰＵ）、アクセラレータ（ＡＣＣ）、データ転送ユニット（ＤＴＵ）などが待ち状態になる場合に、待ち時間に応じてクロック周波数を低周波数に変化させたり、電源を遮断させたりする電力制御を実現するためのスケジューリングである。
【００３７】
その後、ステップ１９１０におけるタスクスケジューリングに基づいて、マルチコアプロセッサにおいて実行可能な並列化プログラムを生成する（ステップ１９１１）。この並列化プログラムは、マルチコアプロセッサの構成に基づいて生成されるため、汎用プロセッサ（ＣＰＵ）用のマクロタスク、ＡＣＣ用のマクロタスク、ＤＴＵ用のマクロタスクの命令文を含む。ここで生成される並列化プログラムでは、互いに依存のある汎用プロセッサ（ＣＰＵ）用のマクロタスク、ＡＣＣ用のマクロタスク、及びＤＴＵ用のマクロタスクの命令文には、フラグセット文、フラグチェック文が挿入される。これらＡＣＣ用のマクロタスク、ＤＴＵ用のマクロタスクの命令文、及びフラグ領域は分散共有メモリ（ＵＲＡＭ）、ローカルメモリ（ＩＬＭ、ＤＬＭ）、オンチップ集中共有メモリ、オフチップ集中共有メモリのうちの少なくとも一つのメモリに配置される。
【００３８】
図２は、本発明の実施形態のコンパイラの処理を実行する計算機１００の物理的な構成を示すブロック図である。
【００３９】
計算機１００は、プロセッサ（ＣＰＵ）１０１、メモリ１０２、不揮発性の記憶装置（ＨＤＤ）１０３、及び通信インターフェース１０６を有する計算機である。

【００４０】
プロセッサ１０１は、メモリ１０２に格納されたプログラムを実行する。
【００４１】
メモリ１０２は、例えば、ＤＲＡＭ（Dynamic Random Access Memory）のような高速かつ揮発性の記憶装置であり、オペレーティングシステム（ＯＳ）及びアプリケーションプログラムを格納する。プロセッサ１０１が、オペレーティングシステムを実行することによって、計算機１００の基本機能が実現され、アプリケーションプログラムを実行することによって、並列化コンパイラの機能が実装される。
【００４２】
記憶装置１０３は、例えば、磁気記憶装置、フラッシュメモリ等の大容量かつ不揮発性の記憶装置であり、プロセッサ１０１によって実行されるコンパイラ及び元プログラムを格納する。すなわち、プロセッサ１０１が実行するコンパイラは、記憶装置１０３から読み出されて、メモリ１０２にロードされて、プロセッサ１０１によって実行される。
【００４３】
通信インターフェース１０６は、計算機１００と他の計算機との通信を制御する。
【００４４】
計算機１００は、入力インターフェース１０４及び出力インターフェース１０５を有してもよい。入力インターフェース１０４は、入力機器であるキーボード１０７及びマウス１０８からの入力を受ける。出力インターフェース１０５は、出力装置であるディスプレイ装置１０９に接続されており、ディスプレイ装置１０９に演算結果を表示するための信号を出力する。
【００４５】
計算機１００は、論理的又は物理的に構成された一つ又は複数の計算機上で稼働するシステムである。並列化コンパイラは、一つの計算機上で動作してもよく、複数の物理的計算機資源上に構築された仮想計算機上で動作してもよい。
【００４６】
プロセッサ１０１によって実行されるプログラムは、リムーバブルメディア（ＣＤ－ＲＯＭ、フラッシュメモリなど）又はネットワークを介して各サーバに提供され、非一時的記憶媒体である記憶装置１０３に格納される。このため、各サーバは、リムーバブルメディアを読み込むインターフェースを備えるとよい。
【００４７】
［逐次処理のエンジン制御プログラムの概要］
次に、本実施例における元プログラムを含むエンジン制御プログラムの概要を示す。
【００４８】
本実施例におけるエンジン制御プログラムは、Ｃ言語で記述されており、自動車向けリアルタイムＯＳであるＯＳＥＫ／ＶＤＸ上（例えば、非特許文献３参照）で動作する。
【００４９】
プログラムの動作を以下に示す。
【００５０】
以下の説明において、エンジン制御プログラムにおける「タスク」は自動車制御用プログラムで使われる用語であり、前述した並列化コンパイラに入力する元プログラムに相当する。
【００５１】
前記エンジン制御プログラムが従来の単一コアプロセッサ上で実行される場合は、まず、Ｃ言語のｍａｉｎ関数からＯＳＥＫ／ＶＤＸが提供するＡＰＩであるＳｔａｒｔＯＳ関数をコールすることによってＯＳを起動する。
【００５２】
その後、エントリタスクから周期的に実行されるタスクの実行を予約する。本エンジン制御プログラムでは、タスクはＴａｓｋ２－Ｔａｓｋ３９の合計３８個が定義されており、周期タスクであるＴａｓｋ２２－Ｔａｓｋ３９の実行が予約される。なお、Ｔａｓｋ２

－Ｔａｓｋ２１は周期タスクから間接的に呼び出されるタスクである。
【００５３】
その後、ＯＳ内のタイマにより、周期タスクが周期的に実行される。
【００５４】
図３に実行プロファイルの結果を示す。なお、本結果は実際にエンジン制御プログラムを動作させている車載向けマイコンを用いて、測定した結果である。図３において、横軸はタスク番号、縦軸は各タスクの実行時間がプログラム全体実行時間に占める割合を示している。図３より、Ｔａｓｋ２０、Ｔａｓｋ２１、Ｔａｓｋ１５の順番で処理時間が大きいことがわかる。このうち、Ｔａｓｋ２０、Ｔａｓｋ２１はアイドルタスク（演算処理を行わない時間待ちのためだけに用意されているタスク）であるため、本実施例では実質的に処理量が一番大きいＴａｓｋ１５に着目し、Ｔａｓｋ１５を元プログラムとしてその並列性を抽出し、並列化プログラムを生成する。
【００５５】
［エンジン制御プログラムからの並列性の抽出との並列化手法］
次に、前記エンジン制御プログラム中のＴａｓｋ１５からの並列性の注出方法と並列化方法について以下に説明する。また、生成される並列化プログラムを図１０に示すマルチコアプロセッサＲＰ－Ｘ上で動作させ、性能評価を行うことを前提に本発明のコンパイラによる並列化処理を行う。
【００５６】
まず、Ｔａｓｋ１５の並列性を解析する。この並列性解析における１９０１から１９０８までのコンパイラの処理ステップは、図１７に示す従来のＯＳＣＡＲコンパイラの処理ステップと変わることはない。Ｔａｓｋ１５にはエントリ関数があり、これをＴａｓｋ１５ｍａｉｎと称する。当該エンジン制御プログラムは、Ｔａｓｋ１５計算部本体とテストドライバからなり、テストドライバはシナリオ１、シナリオ２、シナリオ３からなる三つの実行シナリオでＴａｓｋ１５を実行する。以下の説明では、Ｔａｓｋ１５ｍａｉｎを元プログラムとして焦点を当て、その並列性の抽出方法について説明する。
【００５７】
図４に、図１Ａに示した本発明の実施形態のコンパイラが実行する処理のフローチャートにおけるステップ１９０７の「最早実行可能条件解析」を経て生成されたＴａｓｋ１５ｍａｉｎのＭＴＧを示す。図４のＭＴＧにおいて、ｓｂ２やｓｂ４などのように四角で囲んだマクロタスクはサブルーチンブロック、ｂｂ３やｂｂ５などのように二重四角で囲んだマクロタスクはベーシックブロックを示す。図５に各タスクの実行シナリオ別のマクロタスクの実行コスト（時間）を示す。図５における実行コストの単位はクロック（ｃｌｏｃｋ）である。
【００５８】
図５に示すマクロタスクの実行コストは、ＲＰ－Ｘ上で６４８ＭＨｚにて計測したもので、コンパイラによるＴａｓｋ１５ｍａｉｎの並列化処理におけるステップ１９０７の「最早実行可能時間解析」などに必要なデータとしてコンパイラに予め入力される。なお、ＲＰ－Ｘは、組込用マルチコアプロセッサであり、その構成は図１０を用いて後に説明する。
【００５９】
Ｔａｓｋ１５ｍａｉｎのＭＴＧ生成後、コンパイラは図１Ａのステップ２９０９に示す、本発明の並列性抽出方法の特徴である「条件分岐の複製を含むリストラクチャリング」のステップに入る。このステップは、すでに述べたように、図１７に示す従来の並列化コンパイラのステップ１９０７の「各種のリストラクチャリング」の機能に本発明の条件分岐の複製の機能を追加したものである。以下の説明では単に「リストラクチャリング」と呼ぶこともある。ステップ２９０９における処理をさらに細かく展開した処理フローである図１Ｂを用いて詳細を説明する。なお、図１Ｂにおいて、１９０７１から１９０７４までのステップと１９０７６、１９０７７のステップは、図１７に示す従来の並列化コンパイラのステップ１９０７に含まれており、ステップ２９０９５は本発明の特徴である条件

分岐の複製のステップである。

【００６０】

図４より、この階層では、ｓｂ１、ｂｂ３、ｓｂ４をそれぞれ並列に実行可能であるため、並列度は３であるが、図５より、ｓｂ４の実行時間の占める割合が、シナリオ１で９４．３％、シナリオ２で９２．６％と相対的に大きい。このため、これらのシナリオではｓｂ４の内部を並列化することが重要である。シナリオ３では、ｓｂ４が２９％、ｓｂ７が６８．９％となるため、ｓｂ４及びｓｂ７の内部をそれぞれ並列化することが重要である。いずれにしても、ｓｂ４及びｓｂ７を含むパスがクリティカルパスとなる。このようにクリティカルパスを見出す処理が図１Ｂのステップ１９０７１のクリティカルパスの抽出である。

【００６１】

また、プログラムの構造に、大きなループ処理がなく、条件分岐及び代入文が連続する場合、すでに述べたように、従来の並列性の抽出方法を用いる並列化コンパイラではダイナミックスケジューリングに依らざるを得ない。しかし、図５よりわかるように、実行時間が非常に短いため、並列化した際の同期処理オーバーヘッドや、ダイナミックスケジューリングのスケジューリングオーバーヘッドが相対的に大きくなる問題点がある。

【００６２】

そこで、当該プログラムを並列化するために、代入文の並列性や関数間の並列性を利用する粗粒度並列化と、スタティックスケジューリングとを適用することが重要となる。

【００６３】

次に、並列性向上のために行った逐次処理の元プログラムへのリストラクチャリングの次のステップ１９０７２のインライン展開について述べる。

【００６４】

まず、図４のｓｂ４及びｓｂ７が内包する並列性を有効活用するため、ｓｂ４及びｓｂ７の内部関数のＭＴＧ及びＲＰ－Ｘのプロファイル情報に基づいて、並列性があり、かつ相対的に実行コストが大きい関数を選択し、その関数をＴａｓｋ１５ｍａｉｎの階層までインライン展開をしていく。これにより、Ｔａｓｋ１５ｍａｉｎの階層における並列性が向上する。

【００６５】

インライン展開した後のＭＴＧにおいて並列処理時間を短縮するために、ＭＴＧ上の最長の処理時間を示すパス（クリティカルパス）上にある条件分岐を抽出し（ステップ１９０７３）、抽出した条件分岐のうち条件分岐に後続する複数のマクロタスク間の並列性（すなわち、当該マクロタスク間でデータ依存がないこと）を抽出する（ステップ１９０７４）。その抽出した並列性に基づき、条件分岐を複製する（ステップ２９０９５）。

【００６６】

この実施例で取り上げた元プログラムの例では、ｓｂ７は条件分岐内に存在するため、ｓｂ７をインライン展開しても、それらは一つの条件分岐内に収まってしまい、一つのマクロタスクとしてプロセッサに割り当てられる。そこで、条件分岐内の並列性を抽出するために、コンパイラが条件分岐を複製する。例えば、図６に示す条件分岐があり、条件分岐内の３個の関数が並列化可能だとし、条件式（ｃｏｎｄｉｔｉｏｎ）が条件分岐内で変更されない場合を考える。

【００６７】

図６に示す状態では、この条件分岐を一つのプロセッサに割り当てることになり、ｆｕｎｃ１－ｆｕｎｃ３の並列性を活かすことができない。そこで、図７のようにプログラムを書き換える。この処理によって、各条件分岐が合計３つのマクロタスクに複製され、各マクロタスクを別々のプロセッサに割り当てることが可能となる。以上の条件分岐の複製のための一連の処理はステップ２９０９５において行われる。また、この処理の結果、ＭＴＧ上で他のパスがクリティカルパスになることがある。その場合には、処理をステップ１９０２に戻し、ステップ１９０２から１９０８までの処理を経て、ステップ２９０９において新たなクリティカルパスに対して条件分岐の複製を適用する。これらの処理は条件

分岐の複製と後述するタスク融合が必要な限り行う。これにより、条件分岐内の並列性を抽出することが可能となり、Ｔａｓｋ１５ｍａｉｎの階層における並列性を向上することができる。

【００６８】

図８に、インライン展開や条件分岐の複製を行い、並列性を向上した後にステップ１９０５を経て生成されたＴａｓｋ１５ｍａｉｎのＭＦＧを示す。図８のＭＦＧは、ステップ２９０９においてインライン展開を行ったまま条件分岐の複製を行った後にステップ１９０５にて生成されたものなので、粒度の小さなマクロタスクが多数存在する。

【００６９】

そのため、ステップ１９０８において、図８のＭＦＧからＭＴＧを生成した後に、本実施例では、ステップ２９０９の中のステップ１９０７６において実行コストが小さい複数のタスクを一つの粗粒度タスク（マクロタスク）として融合するタスク融合を行った。このタスク融合により生成されたＭＴＧを図９に示す。図９において、タスク融合を行ったブロックはｌｏｏｐと表示される。図９からわかるように、並列性が損なわれない範囲で条件分岐や代入文をタスク融合することによって、一つ一つのマクロタスクを、プログラムの制御オーバーヘッドが相対的に小さくなるような処理コストを持った粒度とすることができる。また、このような処理を経て、データ依存のみの２並列程度の並列性を抽出することができる。これにより、制御フローを全てデータ依存の形に集約することができ、低オーバーヘッドなスタティックスケジューリングの適用が可能となる。なお、実行コストに基づくタスク融合の処理機能（ステップ１９０７６）は、すでに述べたように本発明の特徴ではなく、従来のＯＳＣＡＲコンパイラにも実装されている（非特許文献６を参照）。

【００７０】

以上、条件分岐の複製を含むリストラクチャリングによる並列性の抽出と並列化プログラムの作成を本発明のコンパイラにより実行した場合のフローを説明してきたが、その中で実行された条件分岐の複製について、その具体例を用いて以下に簡単に説明する。

【００７１】

図１５は、条件分岐の複製のステップ２９０９５をより具体的に説明するために用意した条件分岐の複製前のＭＴＧの一例であり、図１６は、図１５のＭＴＧに条件分岐の複製の処理を施した後のＭＴＧである。

【００７２】

図１５に示す条件分岐の複製前のＭＴＧによると、Ｓｂ２（ｆｕｎｃ２）はｓｂ６（ｆｕｎｃ５）及びｓｂ７（ｆｕｎｃ６）にデータ依存するが、ｓｂ５（ｆｕｎｃ４）にデータ依存しない。従って、図１５のＭＴＧに表された条件分岐を複製して２つの条件分岐を生成し、その一方にｓｂ２を含むパスが含まれ、もう一方の条件分岐にはｓｂ２と並列実行可能なｓｂ５が含まれるようにすることができる。

【００７３】

このように、図１６に示す条件分岐の複製後のＭＴＧによると、ｓｂ２（ｆｕｎｃ２）とデータ依存のないｓｂ５（ｆｕｎｃ４）とが分かれ、並列実行可能となるので、その各々を２つのプロセッサコアに割り当てることができる。なお、条件分岐を複製したため、一部のサブブロック番号が変更されている。

分岐複製前　ｓｂ６（ｆｕｎｃ５）　→　分岐複製後　ｓｂ９
分岐複製前　ｓｂ７（ｆｕｎｃ６）　→　分岐複製後　ｓｂ１３
分岐複製前　ｓｂ１０（ｆｕｎｃ３）　→　分岐複製後　ｓｂ１６

【００７４】

［性能評価］

本項では、実施形態の説明のために挙げた自動車エンジン制御プログラムを本発明の並列性の抽出方法を用いて並列化したプログラムを組込用マルチコアプロセッサＲＰ－Ｘ上で実行させ、その並列処理性能を評価する。

【００７５】

まず、これまでの説明で述べた方法で並列化したエンジン制御プログラムの性能評価を行った組込用マルチコアプロセッサＲＰ－Ｘの構成について説明する。

【００７６】

図１０に示すように、ＲＰ－Ｘは、４５ｎｍ　ＬｏｗＰｏｗｅｒテクノロジ、１５コアのマルチコアプロセッサで、汎用プロセッサコアとして動作周波数を６４８ＭＨｚ、３２４ＭＨｚ、１６２ＭＨｚ、８１ＭＨｚと変更して動作するＳＨ－４Ａコアを８個、アクセラレータコアとして３２４ＭＨｚで動作するＦＥ－ＧＡを４個、その他ハードウェアＩＰを搭載している。

【００７７】

各汎用プロセッサコア内メモリは、命令キャッシュ（３２ＫＢ）、データキャッシュ（３２ＫＢ）、ローカルメモリ（ＩＬＭ、ＤＬＭ：１６ＫＢ）、分散共有メモリ（ＵＲＡＭ：６４ＫＢ）及びデータ転送ユニットを有する。また、アクセラレータコアはコントローラなしアクセラレータであり、オンチップバス（ＳＨｗｙ＃１）に接続されている。

【００７８】

現在のエンジン制御系で２コアのマルチコアが検討されているため、本実施例では、汎用プロセッサコアとして２個のＳＨ－４Ａコアを計算資源として用いた。また、汎用コアの動作周波数を６４８ＭＨｚから３２４ＭＨｚ、１６２ＭＨｚ、８１ＭＨｚと変化させた時のバスの動作周波数は３２４ＭＨｚに固定し、性能を評価した。これはバスの動作周波数を固定し、汎用コアの動作周波数を下げることによって、メモリアクセスレイテンシを相対的に低くするためであり、メモリアクセスレイテンシが小さい車載向けマイコンの環境に近付けるためである。

【００７９】

次に、組込用マルチコアプロセッサＲＰ－Ｘ上での並列処理性能評価条件を説明する。

【００８０】

図１１にＲＰ－Ｘ上６４８ＭＨｚ実行時のＴａｓｋ１５ｍａｉｎの実行プロファイル結果を示す。なお、単位はｃｌｏｃｋである。ＲＰ－Ｘ上で性能評価を行うにあたっては、各シナリオ毎にコンパイラに本プロファイル情報を与え、スタティックスケジューリングを行うことで、負荷を分散する。図１１より、各シナリオで実行コストが異なる。これは、各シナリオにおいて処理を高速化するためには、負荷分散が重要なためである。

【００８１】

また通常、グローバル変数はオフチップ共有メモリに配置する。この場合、キャッシュヒットの際には１サイクルでデータを読み出せるが、キャッシュミスの際には、データの読み出しに５５サイクルを必要とする。エンジン制御プログラムのようなマイクロ秒オーダーで動作するプログラムにおいて、このペナルティは非常に大きい。そこで、メモリアクセスレイテンシが小さいローカルメモリにグローバル変数を配置することが重要となる。しかし、全てのグローバル変数をローカルメモリに配置すると、メモリ容量を超えるため、メモリ容量内に収めるために、初期値無しのグローバル変数（約７．５ｋｂｙｔｅ）のみをローカルメモリに配置する。また、プロセッサコア間の同期を担う同期変数を、オフチップ共有メモリに配置すると、メモリアクセスレイテンシのペナルティが大きいため、レイテンシが小さい分散共有メモリに配置する。これにより、高速化が可能となる。

【００８２】

メモリ配置による性能差を比較するため、グローバル変数全てを共有メモリに配置した場合と、グローバル変数の一部をローカルメモリに配置し、プロセッサコア間の同期を担う同期変数を分散共有メモリに配置した場合の性能を評価する。

【００８３】

次に、組込用マルチコアＲＰ－Ｘ上での並列処理性能評価結果を説明する。

【００８４】

図１２にシナリオ１を１ＣＰＵ、２ＣＰＵ、メモリ配置を工夫した２ＣＰＵで実行した場合の速度向上率を示す。図１２において、横軸は動作周波数、縦軸は１ＣＰＵ実行時に対する速度向上率である。具体的な実行時間は表１に示すとおりである。８１ＭＨｚで１

．５７倍、１６２ＭＨｚで１．５５倍、３２４ＭＨｚで１．５３倍、６４８ＭＨｚで１．４８倍に速度が向上した。メモリ配置を工夫した場合、８１ＭＨｚで１．６０倍、１６２ＭＨｚで１．７１倍、３２４ＭＨｚで１．６９倍、６４８ＭＨｚで１．６２倍に速度が向上した。メモリ配置を工夫することによって、２％から１１％の性能が改善した。

【００８５】

【表１】

	81MHz	162MHz	324MHz	648MHz
1CPU	356.2us	253.9us	198.5us	174.7us
2CPU	227.0us	163.8us	129.8us	118.3us
2CPU-memory opt	222.4us	148.1us	117.1us	107.7us

シナリオ1を実行した場合の処理時間をまとめた表

【００８６】

図１３にシナリオ２を１ＣＰＵ、２ＣＰＵ、メモリ配置を工夫した２ＣＰＵで実行した場合の速度向上率を示す。図１３において、横軸は動作周波数、縦軸は１ＣＰＵ実行時に対する速度向上率である。具体的な実行時間は表２に示すとおりである。８１ＭＨｚで１．３８倍、１６２ＭＨｚで１．４６倍、３２４ＭＨｚで１．４０倍、６４８ＭＨｚで１．１７倍の速度向上が得られた。メモリ配置を工夫した場合、８１ＭＨｚで１．５４倍、１６２ＭＨｚで１．５８倍、３２４ＭＨｚで１．４５倍、６４８ＭＨｚで１．２５倍に速度が向上した。メモリ配置を工夫することによって、３％から１１％の性能が改善した。

【００８７】

【表２】

	81MHz	162MHz	324MHz	648MHz
1CPU	122.0us	74.4us	49.9us	37.2us
2CPU	88.6us	51.0us	35.6us	31.7us
2CPU-memory opt	79.5us	47.1us	34.5us	29.9us

シナリオ2を実行した場合の処理時間をまとめた表

【００８８】

図１４にシナリオ３を１ＣＰＵ、２ＣＰＵ、メモリ配置を工夫した２ＣＰＵで実行した場合の速度向上率を示す。図１４において、横軸は動作周波数、縦軸は１ＣＰＵ実行時に対する速度向上率である。具体的な実行時間は表３に示す通りである。８１ＭＨｚで１．５１倍、１６２ＭＨｚで１．４６倍、３２４ＭＨｚで１．４１倍、６４８ＭＨｚで１．４２倍の速度向上が得られた。メモリ配置を工夫した場合、８１ＭＨｚで１．５３倍、１６２ＭＨｚで１．５０倍、３２４ＭＨｚで１．４６倍、６４８ＭＨｚで１．４３倍に速度が向上した。メモリ配置を工夫することによって、１％から３％の性能が改善した。

【００８９】

【表３】

	81MHz	162MHz	324MHz	648MHz
1CPU	546.3us	356.2us	255.5us	207.7us
2CPU	360.8us	243.2us	181.1us	146.7us
2CPU-memory opt	356.5us	237.7us	175.1us	145.4us

シナリオ3を実行した場合の処理時間をまとめた表

【００９０】

先に述べた本発明の実施例では、逐次処理の自動車エンジン制御プログラムを並列化してマルチコアプロセッサ上で実行させる場合を例に挙げて、本発明の特徴である並列性の抽出方法とその方法を用いたプログラムの並列化方法を説明した。また、本発明を適用した並列化コンパイラを用いて、逐次処理の元プログラムの並列性を抽出するために行った、相対的に実行コストが大きい関数のインライン展開と条件分岐の複製を含む並列性の抽出、およびスタティックスケジューリング適用のためのタスク融合を行った後、すなわち逐次処理の元プログラムのリストラクチャリングを行った後、並列化プログラムを生成し、組込用マルチコアプロセッサ上で並列処理性能の評価を行った。その結果、マルチコア上で従来並列化に成功した例がない、極めて並列化が困難であった自動車エンジン制御プログラムにおいて、２プロセッサコアを用いた場合、１プロセッサコアを使用した場合と比較して、シナリオ１における１６２ＭＨｚの場合、１．７１倍の性能向上が得られた。

【００９１】

本実施例では、コンパイラによる並列化を利用し、高速化を実現でき、自動車エンジン制御プログラムの自動並列化及び高速化が可能であることが確認できた。このように、本発明の方法を用いることにより、自動車エンジン制御プログラム又は一般的な制御用プログラムのような、条件分岐や代入文の多いプログラムにおいても、並列処理による高速化が可能となる。

【００９２】

以上、本発明を添付の図面を参照して詳細に説明したが、本発明はこのような具体的構成に限定されるものではなく、添付した請求の範囲の趣旨内における様々な変更及び同等の構成を含むものである。

【００９３】

特許請求の範囲に記載した以外の本発明の観点の代表的なものとして、次のものがあげられる。

【００９４】

（１）複数のプロセッサコアを含むマルチコアプロセッサの制御方法であって、

前記複数のプロセッサコアによって実行されるプログラムは、１以上の条件分岐の集合を含み、

前記方法は、

２以上の前記プロセッサコアの各々が、少なくとも一つの前記複数の条件分岐の集合を実行し、

前記２以上のプロセッサコアの各々が、前記実行した条件分岐の集合の同一条件を満たした後に実行されるべき１以上のマクロタスクを実行するように、前記マルチコアプロセッサを制御する方法。

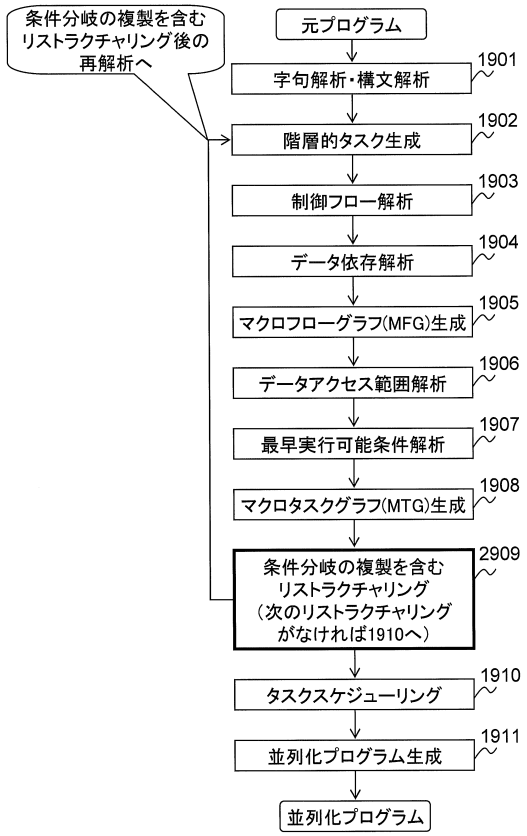
【００９５】

（２）複数のプロセッサコアを含むマルチコアプロセッサ上で実行可能プログラムを元プログラムからコンピュータによって作成する方法であって、

同一の条件で実行が決定される複数のマクロタスクのうち、前記実行される複数のマクロタスクの相互の間でデータ依存がない条件分岐の集合を前記元プログラムから抽出し、

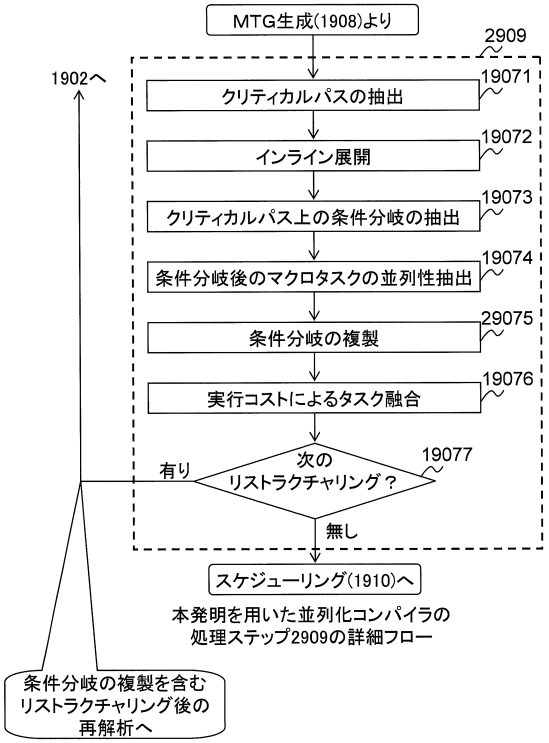
前記抽出された条件分岐の集合を複製し、
前記複製された条件分岐の集合を、２以上の前記プロセッサコアの各々が実行するように割り当てることによって実行可能プログラムを作成する方法。
【００９６】
（３）複数のプロセッサコアを含むマルチコアプロセッサの制御方法であって、
前記複数のプロセッサコアによって実行されるプログラムは、１以上の条件分岐の集合を含み、
前記方法は、
１以上の前記プロセッサコアの各々が、少なくとも一つの前記１以上の条件分岐の集合を実行し、
前記１以上のプロセッサコアの各々が、前記実行した条件分岐の集合の同一条件を満たした後に実行されるべき１以上のマクロタスクを実行するように、前記マルチコアプロセッサを制御する方法。
【００９７】
（４）前記条件分岐の集合は、一つの条件判断の結果による分岐、あるいは一つの条件判断の結果によるさらなる条件分岐を１又は複数含むことを特徴とする(３)に記載の制御方法。
【００９８】
（５）複数のプロセッサコアを含むマルチコアプロセッサ上で実行可能プログラムを元プログラムからコンピュータによって作成する方法であって、
同一の条件で実行が決定される複数のマクロタスクのうち、前記実行される複数のマクロタスクの相互の間でデータ依存がない条件分岐の集合を前記元プログラムから抽出し、
前記抽出された条件分岐の集合を複製し、
前記複製された条件分岐の集合を、１以上の前記プロセッサコアの各々が実行するように割り当てることによって実行可能プログラムを作成する方法。
【００９９】
（６）前記条件分岐の集合は、一つの条件判断の結果による分岐、あるいは一つの条件判断の結果によるさらなる条件分岐を１又は複数含むことを特徴とする（５）に記載のプログラムの作成方法。
【符号の説明】
【０１００】
１００　計算機
１０１　プロセッサ（ＣＰＵ）
１０２　メモリ
１０３　記憶装置（ＨＤＤ）
１０６　通信インターフェース

【図１Ａ】



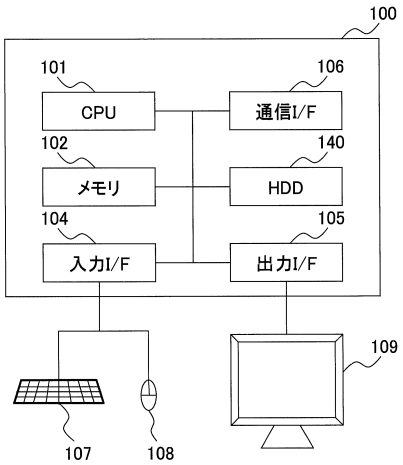
本発明を用いた並列化コンパイラの処理フロー

【図１Ｂ】

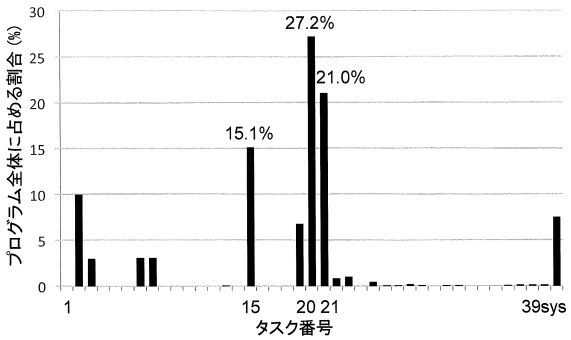


本発明を用いた並列化コンパイラの処理ステップ2909の詳細フロー

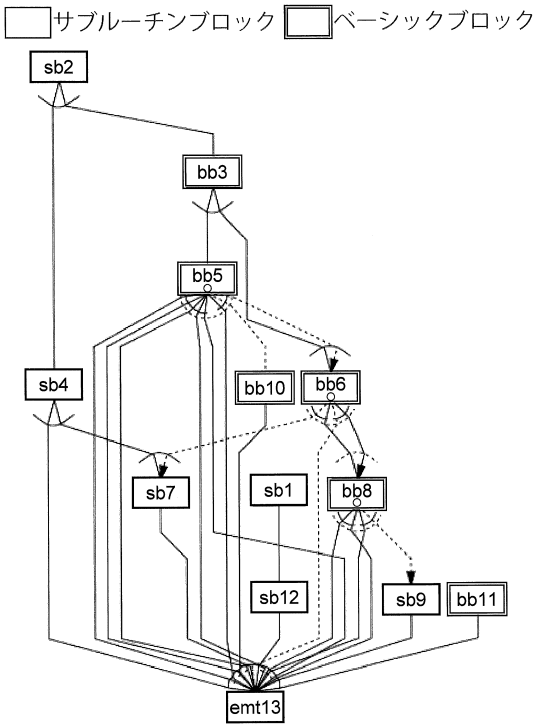
【図２】



【図３】



【図４】



【図５】

Scenario1				
sb1	sb2	sb4	sb12	全体
978	3415	114739	519	121680

Scenario2					
sb1	sb2	sb4	sb9	sb12	全体
42	489	20031	393	38	21641

Scenario3					
sb1	sb2	sb4	sb7	sb12	全体
40	228	37225	88476	455	128364

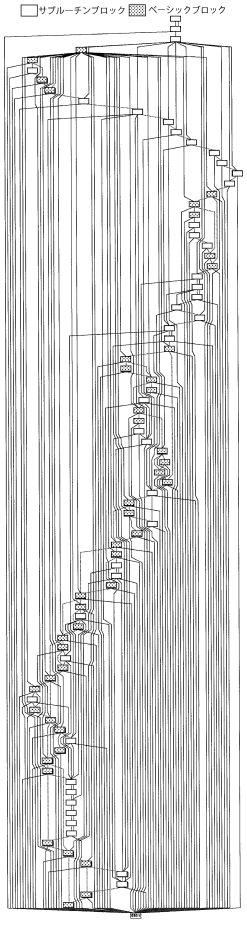
【図６】

```
If (condition){
   func1();
   func2();
   func3();
}
```

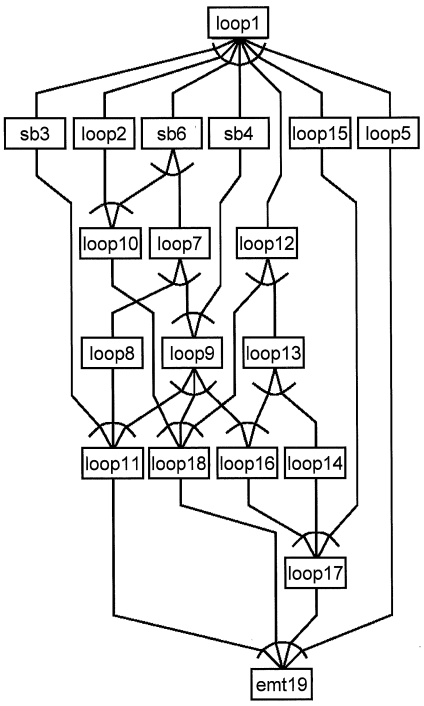
【図７】

```
If (condition){
   func1();
}
If (condition){
   func2();
}
If (condition){
   func3();
}
```

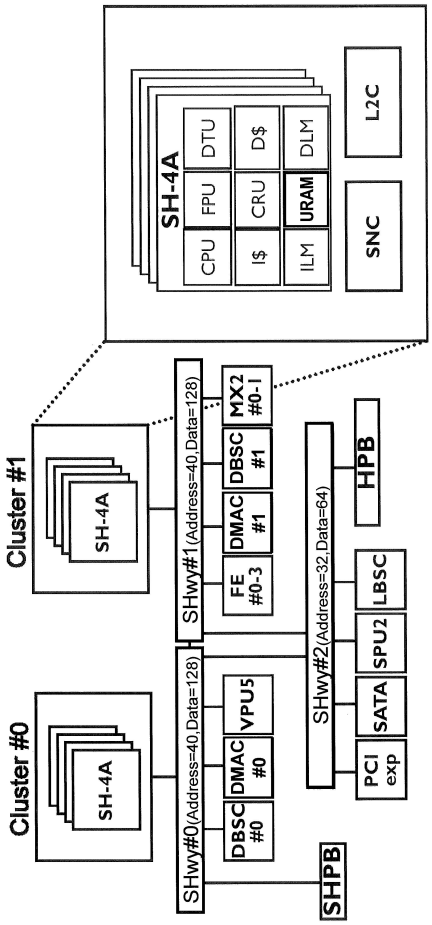
【図８】



【図９】



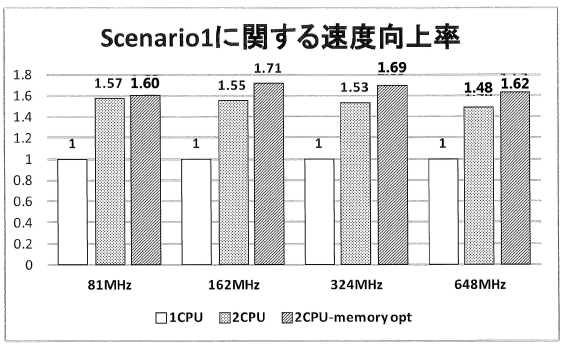
【図１０】



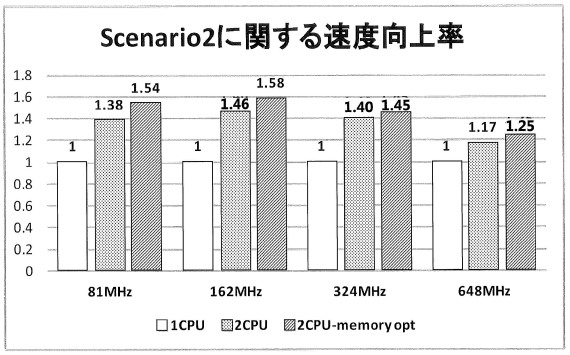
【図１１】

	Scenario1	Scenario2	Scenario3		Scenario1	Scenario2	Scenario3
loop1	6038	987	586	loop10	22476	1485	168
loop2	9665	2518	1558	loop11	38268	4672	18534
Sb3	4204	197	1090	loop12	285	357	2826
Sb4	1720	727	403	loop13	370	129	2459
loop5	7371	7961	4583	loop14	403	127	1685
sb6	5606	3096	11696	loop15	116	129	3739
loop7	12957	2518	2291	loop16	351	129	19811
loop8	4234	323	1393	loop17	426	129	39603
loop9	10326	3742	4641	loop18	380	324	15219

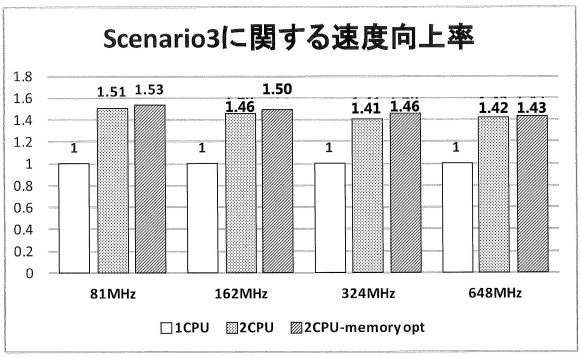
【図１２】



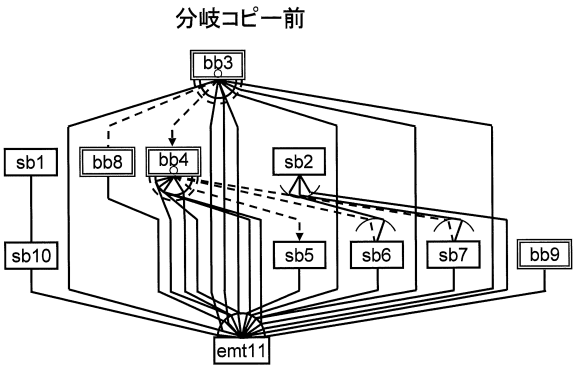
【図１３】



【図１４】

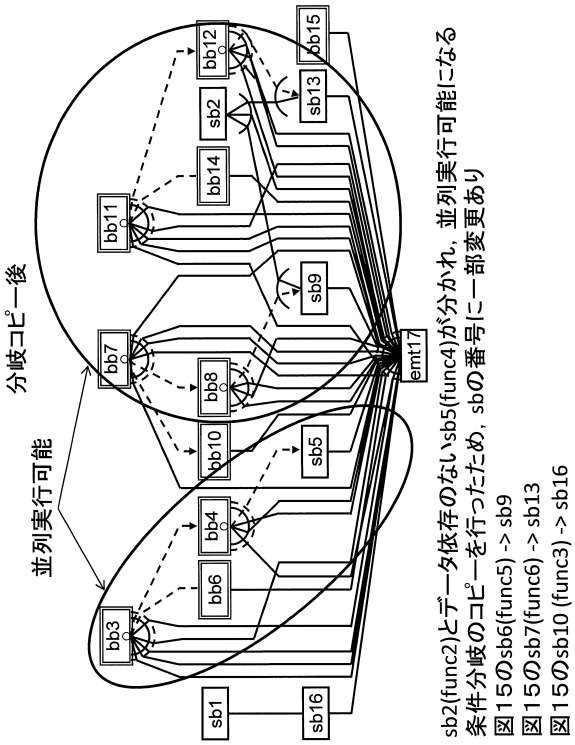


【図１５】

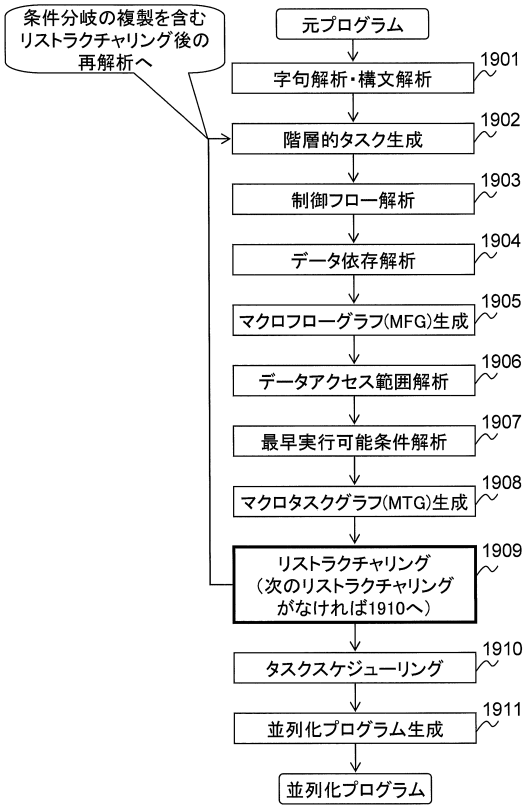


sb2(func2)はsb6(func5)とsb7(func6)にデータ依存があるが，sb5(func4)にデータ依存はない

【図１６】

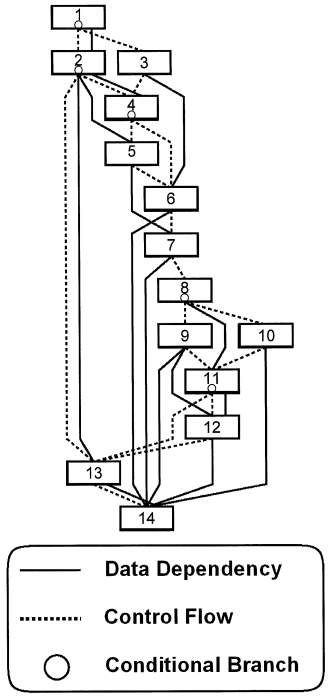


【図１７】



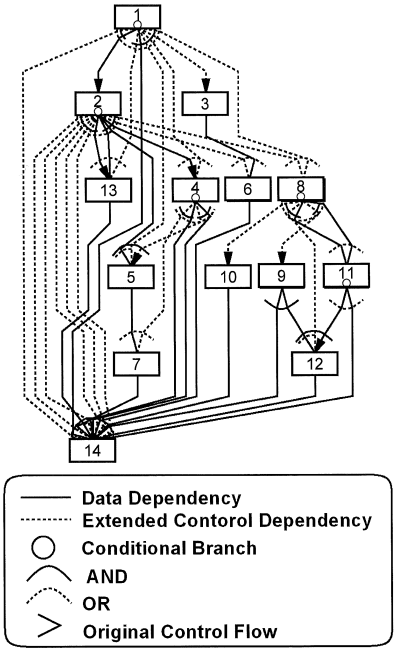
従来の並列化コンパイラの処理フロー

【図１８Ａ】



(a) Macro Flow Graph (MFG)

【図１８Ｂ】



(b) Macro Task Graph (MTG)

フロントページの続き

(72)発明者　見神　広紀
東京都新宿区大久保３－４－１　早稲田大学理工学術院基幹理工学部情報理工学科内
(72)発明者　金羽木　洋平
東京都新宿区大久保３－４－１　早稲田大学理工学術院基幹理工学部情報理工学科内
(72)発明者　梅田　弾
東京都新宿区大久保３－４－１　早稲田大学理工学術院基幹理工学部情報理工学科内
(72)発明者　沢田　光男
愛知県豊田市トヨタ町１番地　トヨタ自動車株式会社内

審査官　坂庭　剛史

(56)参考文献　国際公開第２０１０／０４７１７４（WO，Ａ１）
特開２００２－１１６９１６（ＪＰ，Ａ）
特開２００６－１５４９７１（ＪＰ，Ａ）
合田憲人、岡本雅巳、吉田明正、本多弘樹、笠原博徳，マクロデータフロー処理におけるマクロタスク分割・融合手法，電子情報通信学会技術研究報告，日本，社団法人電子情報通信学会，１９９１年　7月16日，Ｖｏｌ．９１，Ｎｏ．１３０，ｐ．２０５－２１２（CPSY91-30）
岩澤京子、黒澤 隆、菊池純男，並列化支援システムによるＦｏｒｔｒａｎ ＤＯループの並列化方法，情報処理学会論文誌，日本，社団法人情報処理学会，１９９５年　8月１５日，Ｖｏｌ．３６，Ｎｏ．８，ｐ．１９９５－２００６，ISSN0387-5806
梅田 弾、金羽木洋平、見神広紀、林 明宏、谷 充弘、森 裕司、木村啓二、笠原博徳，エンジン基本制御ソフトウェアモデルのマルチコア上での並列処理，情報処理学会研究報告 ２０１２（平成２４）年度▲3▼［ＣＤ－ＲＯＭ］，日本，一般社団法人情報処理学会，２０１２年１０月１５日，Vol.2012-ARC-201，No.22，ｐ．１－７，ISSN1884-0930