(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4339907号
(P4339907)
(45)発行日　平成21年10月7日(2009.10.7)　(24)登録日　平成21年7月10日(2009.7.10)

(51)Int.Cl.　F I
G06F 9/45 (2006.01)　G06F 9/44 322G

請求項の数 13 (全 28 頁)

(21)出願番号　特願2007-275886(P2007-275886)
(22)出願日　平成19年10月24日(2007.10.24)
(65)公開番号　特開2009-104422(P2009-104422A)
(43)公開日　平成21年5月14日(2009.5.14)
審査請求日　平成19年10月24日(2007.10.24)

（出願人による申告）平成１９年度、文部科学省、低電力デバイス・回路技術・論理方式の研究開発　委託事業、産業技術力強化法第１９条の適用を受ける特許出願

(73)特許権者　000005108
株式会社日立製作所
東京都千代田区丸の内一丁目６番６号
(74)代理人　100080001
弁理士　筒井　大和
(72)発明者　高山　恒一
東京都国分寺市東恋ヶ窪一丁目２８０番地
株式会社日立製作所　中央研究所内
(72)発明者　助川　直伸
東京都国分寺市東恋ヶ窪一丁目２８０番地
株式会社日立製作所　中央研究所内

審査官　坂庭　剛史

最終頁に続く

(54)【発明の名称】マルチプロセッサ向け最適コード生成方法及びコンパイル装置

(57)【特許請求の範囲】
【請求項１】
主記憶またはキャッシュメモリを共有する複数のプロセッサから構成される計算機に対して、前記主記憶から前記複数のプロセッサへのデータ転送量を削減しながら前記複数のプロセッサの実行効率を高めることを目的として、ソースコードから前記複数のプロセッサが処理をする最適な並列コードをコンピュータシステムを用いて生成する方法であって、
前記コンピュータシステムは、
前記ソースコードの処理内容を解析して、前記複数のプロセッサの演算量および演算順序依存関係と、前記キャッシュメモリのデータの再利用性と、前記主記憶または前記キャッシュメモリと前記複数のプロセッサとの間のロードデータ量およびストアデータ量とを分析する第１ステップと、
前記計算機の性能としてユーザによって入力された、前記複数のプロセッサの数量、前記主記憶のアクセス時間または前記キャッシュメモリのアクセス時間、レジスタ数、前記キャッシュメモリの容量、前記複数のプロセッサ間の同期の取得方法と同期にかかる時間を保持する第２ステップと、
前記ソースコードの処理内容を前記複数のプロセッサに分割し、この分割されたソースコードを前記複数のプロセッサで実行した場合の実行サイクル時間を前記第１ステップおよび前記第２ステップに基づいて見積りながら、前記実行サイクル時間が最短となる並列コードを生成する第３ステップとを実行することを特徴とする最適化コードの生成方法。

【請求項２】
請求項１記載の最適化コードの生成方法において、
前記複数のプロセッサのそれぞれは、ＳＩＭＤ型またはベクトル型プロセッサであることを特徴とする最適化コードの生成方法。
【請求項３】
請求項１または２記載の最適化コードの生成方法において、
前記第３ステップは、前記ソースコード内に含まれる複数の処理に対して、前記キャッシュメモリのデータの再利用性を高めると共に前記複数のプロセッサから前記主記憶に向けたアクセス回数を少なくする第１方式の分割を行うことを特徴とする最適化コードの生成方法。
【請求項４】
請求項３記載の最適化コードの生成方法において、
前記第３ステップは、更に、前記ソースコード内に含まれる複数の処理に対して、前記複数のプロセッサによる演算処理量を均等にする第２方式の分割を行うことを特徴とする最適化コードの生成方法。
【請求項５】
請求項４記載の最適化コードの生成方法において、
前記第３ステップは、前記第１方式の分割が行われた第１ソースコードに対して更に前記第２方式の分割を行うことで第２ソースコードを生成し、前記第１ソースコードと前記第２ソースコードの前記実行サイクル時間を比較して、いずれか実行サイクル時間が短くなる方のソースコードに対応した並列コードを生成することを特徴とする最適化コードの生成方法。
【請求項６】
請求項４記載の最適化コードの生成方法において、
前記コンピュータシステムは、更に、ユーザによって予め指定された、前記第１方式の分割を行うか前記第２方式の分割を行うかの選択を保持する第４ステップを実行することを特徴とする最適化コードの生成方法。
【請求項７】
請求項６記載の最適化コードの生成方法において、
前記第４ステップは、前記ユーザが前記ソースコード内に、並列化対象範囲と、前記並列化対象範囲に対して前記第１方式の分割を適用するか前記第２方式の分割を適用するかを記載することで実現されることを特徴とする最適化コードの生成方法。
【請求項８】
請求項６記載の最適化コードの生成方法において、
前記第４ステップは、前記ユーザがコンパイラコマンドライン上で、ファイル名及び行数を含む並列化対象範囲と、前記並列化対象範囲に対して前記第１方式の分割を適用するか前記第２方式の分割を適用するかを指定することで実現されることを特徴とする最適化コードの生成方法。
【請求項９】
請求項４記載の最適化コードの生成方法において、
前記第１ステップによって、前記ソースコードに含まれる演算ループのループ制御変数が前記計算機上で並列コードを実行するときまで未確定となることが判明した場合、
前記第３ステップは、前記ループ制御変数の大きさを変更しながら、前記第１方式の分割を適用した場合の前記実行サイクル時間と前記第２方式の分割を適用した場合の前記実行サイクル時間を見積り、前記ループ制御変数の大きさ毎に、前記第１方式の分割か前記第２方式の分割かいずれか前記実行サイクル時間が短い方の方式を選択できるように、前記ループ制御変数の大きさに応じて適用する方式を切り替えるための条件分岐を並列コードに埋め込むことを特徴とする最適化コードの生成方法。
【請求項１０】
コンピュータシステムによって実現され、主記憶またはキャッシュメモリを共有する複

数のプロセッサから構成される計算機に対して、前記主記憶から前記複数のプロセッサへのデータ転送量を削減しながら前記複数のプロセッサの実行効率を高めることを目的として、ソースコードから前記複数のプロセッサが処理をする最適な並列コードを生成するコンパイル装置であって、

前記ソースコードの処理内容を解析して、前記複数のプロセッサの演算量および演算順序依存関係と、前記キャッシュメモリのデータの再利用性と、前記主記憶または前記キャッシュメモリと前記複数のプロセッサとの間のロードデータ量およびストアデータ量とを分析する第１機能と、

前記計算機の性能としてユーザによって入力された、前記複数のプロセッサの数量、前記主記憶のアクセス時間または前記キャッシュメモリのアクセス時間、レジスタ数、前記キャッシュメモリの容量、前記複数のプロセッサ間の同期の取得方法と同期にかかる時間を保持する第２機能と、

前記ソースコードの処理内容を前記複数のプロセッサに分割し、この分割されたソースコードを前記複数のプロセッサで実行した場合の実行サイクル時間を前記第１機能および前記第２機能に基づいて見積りながら、前記実行サイクル時間が最短となる並列コードを生成する第３機能とを有することを特徴とするコンパイル装置。

【請求項１１】

請求項１０記載のコンパイル装置において、

前記第３機能は、前記ソースコード内に含まれる複数の処理に対して、前記キャッシュメモリのデータの再利用性を高めると共に前記複数のプロセッサから前記主記憶に向けたアクセス回数を少なくする第１方式の分割を行うことを特徴とするコンパイル装置。

【請求項１２】

請求項１１記載のコンパイル装置において、

前記第３機能は、更に、前記ソースコード内に含まれる複数の処理に対して、前記複数のプロセッサによる演算処理量を均等にする第２方式の分割を行うことを特徴とするコンパイル装置。

【請求項１３】

請求項１２記載のコンパイル装置において、

前記第３機能は、前記第１方式の分割が行われた第１ソースコードに対して更に前記第２方式の分割を行うことで第２ソースコードを生成し、前記第１ソースコードと前記第２ソースコードの前記実行サイクル時間を比較して、いずれか実行サイクル時間が短くなる方のソースコードに対応した並列コードを生成することを特徴とするコンパイル装置。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、マルチプロセッサ向け最適コード生成方法及びコンパイル装置に関し、特に、複数のプロセッサを効率良く動作させるため、並列処理を行うコードを生成する方法、ならびにコンパイル装置に関する。

【背景技術】

【０００２】

半導体製造技術の発展と共にトランジスタの高集積化が可能となりプロセッサは高い演算性能を実現した。しかし、プロセッサの高い動作周波数やリーク電流により消費電力が多くなる弊害も出ている。この弊害を避ける方法として、例えば非特許文献２に記載されているように、複数のＳＩＭＤ(Single Instruction Multi Data)型プロセッサまたは複数のベクトル型プロセッサを主記憶やキャッシュメモリに共有結合する方法がある。これにより、動作周波数を高くせずに消費電力を抑えながら、演算器を多く並べることによって高い演算性能を実現している。

【０００３】

また、特許文献１には、ループ制御変数の要素を分割する要素並列化と共に、ループ内の命令列を分割するセクション並列化を用いて並列計算機用のオブジェクトコードを生成

するコンパイル処理方法が記載されている。特許文献２には、ループ繰り返しにまたがるデータ依存が存在する多重ループをマルチプロセッサでパイプライン的に並列実行可能にするコンパイル方法が示されている。この方法は、多重ループの前後にバリア同期を発行するループを生成し、分割したループの直後にバリア同期を発行する文を生成するものである。非特許文献１および非特許文献３には、コンパイラの具体的な実装方法などが記載されている。

【特許文献１】特開２００６－２６８０７０号公報

【特許文献２】特開２０００－２０４８２号公報

【非特許文献１】Hans Zima and Barbara Chapman共著、村岡洋一訳、「スーパーコンパイラ」、オーム社、１９９５年

【非特許文献２】C. Scott Ananian, Krste Asanovic, Bradley C. Kuszmaul, Charles E. Leiserson, and Sean Lie、「Cache Refill/Access Decoupling for Vector Machines」、37th International Symposium on Microarchitecture (MICRO-37)、Portland, Oregon、２００４年１２月

【非特許文献３】A.V.エイホ, R.セシィ, J.D.ウルマン著、「コンパイラ」、サイエンス社、１９９０年

【発明の開示】

【発明が解決しようとする課題】

【０００４】

前述したように、複数のＳＩＭＤ型プロセッサや複数のベクトル型プロセッサ等の適用に伴い、プロセッサの処理性能は高まっているが、主記憶からのデータ供給性能はプロセッサの処理性能に見合った向上が図られていない。そのため、主記憶からのデータ転送時間が長くなり、その間にプロセッサ内の処理が進まなくなって、複数のプロセッサの性能を引き出すことができない問題が生じている。これは一般的にメモリウォール問題として知られている。

【０００５】

また、特許文献１等に記載されているように、演算ループを複数のプロセッサに処理を分割して実行する際、演算ループの制御変数を分割してプロセッサに割付けて各プロセッサで演算実行することが行われている。この時の演算を実行する計算機システムの構成図の例を図１に示し、演算の並列化の例を図２に示す。図１の計算機は、キャッシュメモリ１０７を共有する４つのプロセッサ（ＣＰＵ）１０６の構成である。図２では、プログラム２０１が演算ループの制御変数により４分割され、この４分割されたプログラム２０２が各プロセッサに割り当てられている。各プロセッサは、同じ配列名の違う要素を使って同時に計算を実行するため、４つのプロセッサはキャッシュメモリを共有しているが、１つのプロセッサが計算に使えるキャッシュメモリの容量は１／４に減ってしまう。そのため、１つのプロセッサが受け持つキャッシュメモリの容量が少なくなり、再利用できるデータ量が減ってしまう問題がある。

【０００６】

ここで、計算機の演算性能を高めるために非特許文献２に示されるようなＳＩＭＤ型プロセッサまたはベクトル型プロセッサを用いる場合を想定する。ＳＩＭＤ型プロセッサまたはベクトル型プロセッサは演算器が多数並んでいるため、各プロセッサの実行効率を高めるためには、１つのプロセッサで処理する演算ループ長を一定以上に長く保つことが必要である。しかし、演算ループのループ長を延ばすことによりキャッシュメモリに登録するデータ量が増えるため、キャッシュメモリのローカルデータアクセスが損なわれ、データの再利用性が低下することが問題となる。

【０００７】

本発明は、このようなことを鑑みてなされたものであり、本発明の前記ならびにその他の目的と新規な特徴は、本明細書の記述および添付図面から明らかになるであろう。

【課題を解決するための手段】

【０００８】

本願において開示される発明のうち代表的なものの概要を簡単に説明すれば、次の通りである。

【０００９】

本発明の一実施の形態の最適化コードの生成方法は、主記憶またはキャッシュメモリを共有する複数のプロセッサから構成される計算機に対して、主記憶からプロセッサへのデータ転送量を削減しながらプロセッサの実行効率を高めることを目的として、ソースコードから各プロセッサが処理をする最適な並列コードを生成する方法となっている。具体的には、コンピュータシステムは、ソースコードに含まれる複数の処理を複数のプロセッサに分割すると共に、この分割されたソースコードを解析し、プロセッサの演算量および演算順序依存関係や、キャッシュメモリのデータの再利用性や、主記憶またはキャッシュメモリに対するロードデータ量およびストアデータ量などの分析を行う。そして、コンピュータシステムは、予めユーザによって定義された計算機の性能（例えば主記憶のアクセス時間、キャッシュメモリのアクセス時間、キャッシュメモリの容量等）を用いて、前記分割されたソースコードの実行サイクル時間を見積りながら、実行サイクル時間が最短となる並列コードを生成する。このようなコンピュータシステムの処理によって、複数のプロセッサの実行効率を高めるための最適な並列コードが生成可能となる。

【００１０】

なお、ソースコードに含まれる複数の処理を複数のプロセッサに分割する際には、キャッシュメモリのデータの再利用性を高めると共に複数のプロセッサから主記憶に向けたアクセス回数を少なくする第１方式の分割や、あるいは、複数のプロセッサによる演算処理量を均等にする第２方式の分割を行う。また、第１方式の分割が行われたソースコードに対して更に第２方式の分割を行う。そして、このような分割が行われたソースコードに対してそれぞれ実行サイクル時間を見積り、その結果、実行サイクル時間が最短となるソースコードに対応した並列コードを生成する。これによって、主記憶アクセスの観点とプロセッサの処理量均等化の観点と、それらの組合せを総合して、最適な並列コードを生成可能となる。

【００１１】

また、本発明の一実施の形態の最適化コードの生成方法は、予めユーザが、ソースコード内の任意の範囲を決めて、そこに前述した第１方式の分割を適用するか第２方式の分割を適用するかを指定することも可能となっている。これによって、ユーザの知見や経験等を活用しながら、ソースコード内の各範囲毎にその処理内容に応じた最適化を図ることができ、その結果、全体として最適な並列コードを生成することが可能となる。

【発明の効果】

【００１２】

本願において開示される発明のうち代表的なものによって得られる効果を簡単に説明すれば、次の通りである。

【００１３】

本発明の一実施の形態による最適化コードの生成方法を用いると、主記憶またはキャッシュメモリを共有する複数のプロセッサ間で主記憶からのデータ転送量を減らしながら、各プロセッサを効率的に動作させる並列コードを生成可能となる。

【発明を実施するための最良の形態】

【００１４】

以下、本発明の実施の形態を図面に基づいて詳細に説明する。実施の形態を説明するための全図において、同一の部材には原則として同一の符号を付し、その繰り返しの説明は省略する。また、以下の実施の形態においては便宜上その必要があるときは、複数のセクションまたは実施の形態に分割して説明するが、特に明示した場合を除き、それらはお互いに無関係なものではなく、一方は他方の一部または全部の変形例、詳細、補足説明等の関係にある。

【００１５】

また、以下の実施の形態において、要素の数等（個数、数値、量、範囲等を含む）に言

及する場合、特に明示した場合および原理的に明らかに特定の数に限定される場合等を除き、その特定の数に限定されるものではなく、特定の数以上でも以下でも良い。さらに、以下の実施の形態において、その構成要素（要素ステップ等も含む）は、特に明示した場合および原理的に明らかに必須であると考えられる場合等を除き、必ずしも必須のものではないことは言うまでもない。同様に、以下の実施の形態において、構成要素等の形状、位置関係等に言及するときは、特に明示した場合および原理的に明らかにそうでないと考えられる場合等を除き、実質的にその形状等に近似または類似するもの等を含むものとする。このことは、上記数値および範囲についても同様である。

【００１６】

（実施の形態１）

図１は、本発明の実施の形態１によるコード生成方法において、その生成されたコードを用いて制御を実行する計算機システムの一例を示した構成図である。この計算機システムは、コードを実行するためのディスプレイ装置１０１、端末装置１０２、コマンドを入力するキーボード１０３、端末に接続された外部記憶装置１０４、コードを実行する計算ノード群１０５、計算ノード群に接続された外部記憶装置１０９からなる。

【００１７】

１つの計算ノードは、主記憶装置１０８またはキャッシュメモリ１０７を共有する複数のプロセッサ（ＣＰＵ）１０６からできており、１つまたは複数の計算ノードから計算ノード群は構成される。キーボード１０３及び端末装置１０２からコマンドを投入して、生成されたコードを計算ノード群１０５で実行する。本実施の形態では、計算ノード群１０５の複数のプロセッサ１０６を効率的に動作させる並列コードの生成方法を示す。

【００１８】

本実施の形態の並列コード生成方法は、端末装置１０２または計算ノード群１０５または別の計算機システムで実現される。コードを生成する計算機システムの構成図を図３に示す。この計算機システムは、ディスプレイ装置３０１、キーボード３０２、プロセッサ（ＣＰＵ）３０３、主記憶装置３０４、外部記憶装置３０５から構成される。並列コードを生成する際には、キーボード３０２からユーザのコード生成起動のコマンドを入力する。コード生成の状況及びエラーメッセージや終了メッセージはディスプレイ装置３０１に表示される。

【００１９】

外部記憶装置３０５には、ソースコード３１４と最終的に生成された並列コード３１５が格納される。主記憶装置３０４には、構文解析部３０６や、中間コードを生成する中間コードの生成部３０７や、並列コードの生成に伴い解析を行う解析部３０８などが格納される。さらに、主記憶装置３０４には、解析部３０８を実行する際に途中で生成される処理内容テーブル３０９、再利用データテーブル３１０、主記憶データテーブル３１１、並びに各ＣＰＵの処理分割テーブル３１２と、最適化コード生成部３１３とが格納される。並列コード生成処理はプロセッサ（ＣＰＵ）３０３で制御され実行される。

【００２０】

図３でソースコードから並列コードを生成する際の処理の大まかな流れを図４に示す。コードの生成は、ソースコード４０１をコード変換部（コンパイラ）４００に入力することで行われる。コード変換部４００は、まず、構文解析部４０２と中間コードの生成部４０３を用いた処理を行う。構文解析の方法に関しては、例えば非特許文献３に記述されている。中間コードの生成部４０３の処理に関しては図７で説明する。次いで、コード変換部４００は、中間コードに対して解析部４０４を用いて解析を行う。

【００２１】

本実施の形態のコード生成方法は、この解析部４０４による処理内容が主要な特徴となっている。図４の例では、中間コードに対して解析部４０４を用いた解析を行っており、この場合のコード変換部４００は、一般的にコンパイル装置（コンパイラ）と呼ばれているものに該当する。ただし、本実施の形態のコード生成方法では、中間コードに対して解析部４０４を用いた解析を行うのみならず、ソースコード４０１に対して解析部４０４を

用いた解析を行うことも可能となっている。この場合、コード変換部４００は、例えば、ソースコード４０１を解析した後に、各ＣＰＵ毎の処理が明確化された最適化ソースコードを生成することになる。そして、この最適化ソースコードを通常のコンパイル装置でコンパイルすることで最適化コードが得られる。

【００２２】

解析部４０４は、順に、ループ演算切り出し部４０５、依存処理と独立処理解析部４０６、データの再利用性解析部４０７、主記憶データ引用解析部４０８、並列化方法評価部４０９を用いた処理を行う。初めに、ループ演算切り出し部４０５では、演算ループ単位とその前後の処理の主記憶アクセスの分析、及び処理の並列化を行う。その詳細については図１１、図２８で説明する。ループ演算切り出し部４０５の結果を基に、依存処理と独立処理解析部４０６では、演算やファイルからのデータ読込みやプロセッサ間の通信といった処理と、この処理の際にプロセッサで引用するデータとプロセッサからキャッシュメモリまたは主記憶に書き込むデータの順序を求める。データを書き込む際に処理の順序を保障する必要がある場合には、依存処理として、データの種類を処理内容テーブル４１４に記録する。

【００２３】

続いて、データの再利用性解析部４０７では、処理の際に利用するデータが再利用データである場合、再利用するデータの量と再利用する時点までのデータの引用間隔を解析する。このときの詳細手順は図１２、図１６で説明する。再利用データである場合は、再利用データテーブル４１５に記録する。次いで、主記憶データ引用解析部４０８では、処理に使用するデータが主記憶から引用されているのか解析し、主記憶から引用したデータを主記憶データテーブル４１６に登録する。この際の詳細な解析手順は図１２、図１６で説明する。

【００２４】

最後に、並列化方法評価部４０９では、ユーザが入力した並列化コード生成方針４１２、計算機条件入力部４１３、処理内容テーブル４１４、再利用データテーブル４１５、主記憶データテーブル４１６を基に並列時の処理の分割を決める。この際の詳細手順は図１７、図１８、図２０、図２１、図２２で説明する。そして、これによって求めた結果を各ＣＰＵの処理分割テーブル４１７に記録する。このような解析部４０４での処理を経て、コード変換部４００は、並列化方法評価部４０９の結果と各ＣＰＵの処理分割テーブル４１７を基に、最適化コード生成部４１０を用いて最適化コードの生成を行い、その結果、並列コード４１１が生成される。

【００２５】

ユーザが指定する並列化方針に関して図５、図６に示す。図５はコンパイルオプションの例を示す。本実施の形態の並列化方法を適用する際、並列化方針のコマンド５０１を入力し、その後に詳細な指定を続ける。ファイル名、サブルーチン名または関数名、ソースコードで指定されたファイル名の中の行数５０２を入力する。更に並列化の際に各プロセッサで実行する演算量の負荷均等化を優先して並列化コードを生成する場合には「ｃａｌｃｕｌａｔｉｏｎ」、キャッシュメモリの再利用性を高める等をして主記憶へのアクセス時間を最短にすることを優先して並列コードを生成する場合には「ｄａｔａ」として並列化方針５０３を入力する。

【００２６】

また、並列化方針は、コマンドのみならず、ソースコード内でその一部の箇所を指定して与えることも可能となっている。ソースコードに並列化方針の指示を与えたときの例を図６に示す。並列化方針のコマンドは、６０１、６０４で示す。その次の引数６０２、６０５で並列化方針を指定する範囲を行数で示す。最後に適用する並列化方針として、演算量の負荷均等化を優先して並列化コードを作成する場合は「ｃａｌｃｕｌａｔｉｏｎ」、またキャッシュメモリの再利用性を高める等をして主記憶へのアクセス時間を最短にすることを優先して並列化コードを作成する場合には「ｄａｔａ」により指定する（６０３、６０６）。

【００２７】
図６の例において、並列化方針６０１で指定している処理は、演算ループの並列化を対象としている。一方、並列化方針６０４で指定している処理は、演算だけではなく、ディスクから主記憶へのデータの読込み（read文）と、プロセッサ間のデータの通信（call mpi_isend文）が含まれている。この処理に関して、「ｄａｔａ」により４つのプロセッサに向けて並列化された場合、例えば、プロセッサ０番がディスクから主記憶への読み込み、プロセッサ１番がプロセッサ間の通信処理、プロセッサ２、３番が演算を分けて処理することもできる。
【００２８】
本明細書では、処理ループの制御変数を分けて処理の負荷を均等化する並列化方法を演算負荷均等並列化法と呼ぶことにする。また、ループ構造となっていない処理を分割してプロセッサに割当てる並列化、および前述した図６の例のようにループ構造となっていない処理の分割とループ制御変数による分割を混合して用いる方法を主記憶アクセス削減並列化法と呼ぶことにする。主記憶アクセス削減並列化法は、図５または図６のコンパイルオプションやディレクティブで並列化方針が「ｄａｔａ」で指定されたときにソースコードの解析が実施される。
【００２９】
図７では、図４の構文解析部４０２と中間コードの生成部４０３の処理により、ソースコード７０１から中間コード７０２を生成したときの例を示す。中間コードでは基本ブロックをエッジで結んだグラフで表現される。こういったグラフは制御フローグラフと呼ばれている。Ｂ０からＢ３は、基本ブロックを表す。基本ブロックは複数の文で構成され、各文は実効命令にほぼ対応しているので、「命令」と呼ばれることもある。Ｂ２の「ｒ０＝ｌｏａｄ（ｘ［ｉ］）」は配列型の変数ｘのｉ要素値をロードして変数ｒ０に代入することを表している。本実施の形態では最適化に関係するキャッシュメモリアクセスまたは主記憶アクセスを「ｌｏａｄ」で表している。
【００３０】
図７の中間コード７０２を用いて、４つのプロセッサ上で実行するときに演算を分割して各ＣＰＵで行う中間コードを図８に示す。図７では、ループ演算の制御変数iをＢ１で１にセットしてからＢ２の演算を開始する。Ｂ２の中でｉをカウントアップしていきｉ＝４０１になった時点でＢ２の演算を終了してＢ３に処理を移行する。この演算を均等に４分割して各プロセッサで実行することを図８が表している。図８のＢ２では、ｉ＝１～１００、ｉ＝１０１～２００、ｉ＝２０１～３００、ｉ＝３０１～４００の４分割をすることにより、各プロセッサでの演算量を均等に分割できる。このような分割方法は、演算量の負荷均等化を優先した並列化法（すなわち演算負荷均等並列化法）となる。
【００３１】
演算量の負荷均等化を優先した並列化方法では、各プロセッサが演算するデータが違っているため、図１の計算ノード群１０５の１つの計算ノードで４プロセッサがキャッシュメモリ１０７を共有している場合、１プロセッサが使用する共有キャッシュメモリの容量は全容量の１／４となる。もし、図８で使用するプロセッサがベクトルプロセッサであった場合の中間コードの例を図９に示す。ベクトル長を１００とすると、Ｂ２の１行目の「ｒ０＝ｌｏａｄ（ｘ［ｉ］）」はｖｌｏａｄ命令を使った「ｒ０＝ｖｌｏａｄ（ｘ）」となり、ｉの１要素に対して実行されるのではなく、ｉ＝１～１００までの要素を一度に実行する。そのため、ｒ０は１要素ではなく１～１００までの要素を持つ変数となる。その後に実行される命令である、「ｒ１＝ｖｌｏａｄ（ｙ）」「ｒ２＝ｒ０＋ｒ１」などに関しても全て１００要素に対するデータの読み込みと演算である。
【００３２】
ここで、変数の１要素のデータ量が８Ｂｙｔｅの場合、１変数当たり要素数が１００であるため、１変数のデータ量は８００Ｂｙｔｅである。図１の計算ノード群１０５の１つの計算ノードで実行する場合、再利用データのロードが共有キャッシュメモリから実行できると、主記憶からロードする場合に比べて高速に実行できる。演算Ｂ２の中では、変数

ｘだけが２回利用している。２回目に変数ｘがロードされるまでに他の変数が８種類（ｘ，ｙ，ｕ，ｖ，ａ，ｂ，ｐ，ｑ）で各変数が８００Ｂｙｔｅであるため、１プロセッサ当たりが使うキャッシュメモリは６．４ＫＢｙｔｅとなる。４プロセッサでは、６．４ＫＢｙｔｅの４倍の２５．６ＫＢｙｔｅになる。

【００３３】

図１０に、図９に示した各プロセッサの演算量の負荷均等化を優先した並列化法とは異なる並列化の方法を示す。図１０では、Ｂ２のループ制御変数は分割していないため、各プロセッサのループ実行回数は並列化前と同じである。そのかわり、ループの中にある演算を４つに分割して各プロセッサに割当てる。図９と図１０ではループ内で変数ａ、ｂ、ｃ、ｄを求めている。図９では各プロセッサが分担した範囲で変数を求めているが、図１０では４つに分割した１００１で変数ａ、１００２で変数ｂ、１００３で変数ｃ、１００４で変数ｄを計算している。この場合は、１００１と１００３の初めの演算を実行すれば１００４の計算を実行でき、各変数の１００要素を全てキャッシュメモリに読み込む前に演算を終えることも可能となるため、使用するキャッシュメモリは例えば１５要素分（ｘ［１－２］，ｙ［１－２］，ａ［１］，ｕ［１－２］，ｖ［１－２］，ｐ［１－２］，ｑ［１－２］，ｃ［１］×２）のデータ量である１２０Ｂｙｔｅとなる。したがって、図９に示した演算量の負荷均等化を優先した並列化の場合の２５．６ＫＢｙｔｅに比べて小さくなっている。

【００３４】

図１０のようにキャッシュメモリを利用する容量を削減しながら再利用データの計算を実行する方法は、主記憶へのアクセス時間を最短にする並列化法（主記憶アクセス削減並列化法）となる。ただし、各Ｂ２の行数から１００１と１００３の処理量は少ないが、１００２と１００４の処理量は多くなっており、１００１～１００４の処理量に負荷の均等化が図れていない。そのため、プロセッサの処理性能が高い割りに主記憶からのデータ転送能力が低い場合、プロセッサの処理性能が高い割りにキャッシュメモリの容量が少ない場合に、演算量の均等化が図れなくても主記憶からのデータ転送量を削減できるため、主記憶へのアクセス時間を最短にする並列化法が有効となる。

【００３５】

以上のように、図５のコンパイルオプション５０３、図６のソースコードの指示行６０３と６０６で、Ｃａｌｃｕｌａｔｉｏｎを指示した場合は、図９のような演算負荷均等並列化法による最適化を用いたコードの生成が行われ、ｄａｔａを指示した場合は、図１０のような主記憶アクセス削減並列化法による最適化を用いたコードの生成が行われる。

【００３６】

図１１に、図４のループ演算切り出し部４０５が行う詳細な処理ループの分割手順を示す。図２８は、図１１の補足図である。まず、図２８を用いて、図１１の分割手順が行う処理の概要について説明する。図２８には、一例として多重ループを含むソースコードが示されている。この多重ループにおいて、最内側ループ２８０１の処理１１を調査して演算に依存が無いことが分かれば、ループ２８０１を複数のＣＰＵにより並列化して独立に演算することが可能となる。演算に依存が無い場合とは、例えば、処理１１に含まれる複数の命令文の中で、ある命令文の結果を他の命令文で使用することが無いような場合を意味する。その後、ループ２８０１の外側に位置するループ２８１１の処理を調査する。ここで、処理２１、処理３１、処理４１に演算の依存が無いことが分かれば、ループ２８１１を複数のＣＰＵによって並列化して独立に演算することが可能となる。

【００３７】

続いて、更にループ２８１１の外側に位置するループ２８１２の処理を調査する。ループ２８１２では、処理１２が処理１１に依存のある演算となっている。このように依存がある処理が含まれると、そのループ２８１２に対して単純に主記憶アクセス削減並列化法や演算負荷均等並列化法を適用するのは困難となる。したがって、この場合、複数のＣＰＵで分割できる最大の単位はループ２８１１である。ループ２８１１を分割する際、演算負荷均等並列化法を用いて４ＣＰＵに処理を分割して実行する場合、例えば、ループ２８

１１のループ長３００を７５（＝３００／４）ずつに分けて実行する。一方、主記憶アクセス削減並列化法を用いる場合、例えば、ループ２８１１に含まれるループ２８０１、ループ２８０２、ループ２８０３、ループ２８０４をそれぞれ４つのＣＰＵに分割すればよい。

【００３８】

なお、分割可能な最大ループであるループ２８１１の代わりに、例えば最内側ループ２８０１を対象として、演算負荷均等並列化法などを用いて複数のＣＰＵに分割することも可能である。ただし、この場合は、例えば４つのＣＰＵを用いてループ２８０１を実行した後に、４つのＣＰＵを用いてループ２８０２を実行するといった処理の流れとなり、このループ２８０１とループ２８０２の繋ぎ目で各ＣＰＵ間の同期が必要となる場合がある。このような同期は実行効率を低下させる要因となるため、前述したように、演算の依存性がない範囲で可能な限り最大ループを探索し、この最大ループを対象として各種分割方法の適用を検討することが望ましい。

【００３９】

このように、例えば図２８のループ２８１１のような複数のＣＰＵで分割できる最大の単位を探索するため、図４のループ切り出し部４０５は、図１１の分割手順を行う。図１１においては、まず初めに中間コードの中からループ処理をする部分を特定する（１１０２）。この時点では、特定したループの内側に他のループ処理が含まれていても良い。特定したループの内側に別のループが存在しているのか判断する（１１０３）。内側にループ処理がある場合は内側のループを対象として（１１０４）、再度対象ループの中に別のループ処理が含まれるか判断する（１１０３）。これにより、最内側のループを特定する。

【００４０】

最内側ループの中に含まれる処理の間に演算の依存性を判断し（１１０５）、依存性が無い場合には、最内側ループの１つ外側のループを評価対象とする。そのため、外側にループが存在するのか判断を行う（１１０６）。外側にループが無い場合には、この最内側ループが評価対象処理ループとなる（１１１０）。外側にループがある場合には、そのループを評価対象として、外側処理の間でデータの依存関係を持つ処理が含まれるのか判断する（１１０７）。データ間に依存関係の無い処理であれば、更に外側のループを評価対象とする（１１０６）。このように、評価対象ループの範囲を広げることで、レジスタ上のデータの再利用性、キャッシュメモリ上のデータの再利用性を高め、主記憶へのアクセス時間の短縮を図りやすくする。すなわち、主記憶アクセス削減並列化法の適用が容易となる。

【００４１】

また、最内側ループの処理の依存性の判定（１１０５）で依存がある場合、最内側ループ内の処理で使用するデータの間に依存関係が無く独立に実行できる処理の有無を判定する（１１０８）。独立に実行できる処理が無い場合は、逐次処理（１１０９）として、本実施の形態の並列化評価の対象としない。独立に実行できる処理の有無の判定（１１０８）で、独立に処理ができる場合は、本実施の形態の並列化評価の対象となり、評価対象処理ループ（１１１０）とする。

【００４２】

図１２に、図４の依存処理と独立処理解析部４０６、データ再利用性解析部４０７、主記憶データ引用解析部４０８の詳細な評価手順を示す。なお、データ再利用性解析部４０７、主記憶データ引用解析部４０８の評価は図１６でも行われる。図１２では、図１１の結果として得られた評価対象ループ（１２０１）を基に評価する。評価対象処理ループ内に処理があることを判定して（１２０２）、処理を１つずつ取り出して評価を実行する（１２０３）。

【００４３】

１つの処理の中で引用するデータが前の処理結果に依存するのか判定を行い（１２０４）、依存しない場合（１２０５）は、演算量の算出（１２０６）をした後、独立処理であ

ることを処理内容テーブルに登録する（１２０７）。その上で、使用したデータが他の処理との依存関係が無く、主記憶から読み込むことになるため、主記憶データテーブルに使用したデータを登録する（１２０８）。また、処理の結果を主記憶にストアすることになるが、このデータに関しても主記憶データテーブルに記録する（１２０８）。

【００４４】

一方、処理で使用するデータが前の処理結果に依存する判定（１２０４）で依存がある（１２１０）の場合は、演算量の算出（１２１１）をした後、依存処理を処理内容テーブルに登録する（１２１２）。次いで、前の処理結果を引用するデータを再利用データテーブルに登録する（１２１３）。ここで登録される再利用データは、例えば図７のソースコード７０１の例では、ａ（ｉ）とｃ（ｉ）に該当する。更に、前の処理に依存しない引用データまたは書き出しデータを主記憶データテーブルに登録する（１２０８）。また、再利用データテーブルに登録した引用データが主記憶データテーブルのストアの項目にある場合は、主記憶データテーブルのストア記録を削除する。

【００４５】

この手順により１つの処理に関する依存処理と独立処理解析が終了する。この後、評価対象処理に後続の処理があるか判断して（１２０２）、後続の処理がある場合には、次の１つの処理の解析（１２０３）へと続ける。１２０２で後続の処理が無くなった段階で、処理の解析を終了する（１２０９）。

【００４６】

図１２の依存処理と独立処理の解析に使用した処理内容テーブルの例を図１３に示す。解析した処理毎に識別番号として処理番号１３０１をつける。処理の項目１３０２として、演算の種類、プロセッサ間の通信関数名、ファイルの入出力を記述する。この処理項目の内容でデータの依存性の有無を１３０３に記述する。また、処理を実行する際に使用する変数名１３０４、依存のある変数名１３０５、処理を実行する際の演算量［Ｆｌｏｐ］１３０６、並列化方法として演算負荷均等並列化法を優先するか主記憶アクセス削減並列化法を優先するのかを１３０７に記述する。表の各項目１３０１～１３０７の内容は、図１２の１２０５、１２０６、１２０７、１２１０、１２１１、１２１２の中で求める。

【００４７】

図１２の依存処理と独立処理の解析に使用した再利用データテーブルの例を図１４に示す。このテーブルでは、図１３の処理内容テーブルで割付けた処理番号１３０１毎に区別して処理番号１４０１を管理する。同じ処理番号１４０１の中に複数の読み込みデータと書込みデータがあり区別する必要があるため、補助番号１４０２で区別する。このほかに再利用データテーブルには、再利用変数名１４０３、変数のデータ長［Ｂｙｔｅ］１４０４、要素数１４０５、各要素の間隔を示すストライド幅［Ｂｙｔｅ］１４０６、一度使ってから再利用するまでのデータ間隔を示す再利用間隔［Ｂｙｔｅ］１４０７を記載する。

【００４８】

図１２の依存処理と独立処理の解析に使用した主記憶データテーブルの例を図１５に示す。このテーブルでは、図１３の処理内容テーブルで割り付けた処理番号１３０１毎に区別して処理番号１５０１を管理する。同じ処理番号１５０１の中に複数の読み込みデータと書込みデータがあり区別する必要があるため、補助番号１５０２で区別する。主記憶データテーブルにはこの他、主記憶引用変数名１５０３、変数のデータ長［Ｂｙｔｅ］１５０４、変数の要素数１５０５、各要素の間隔を示すストライド幅［Ｂｙｔｅ］１５０６、読み込みデータと書き出しデータの区別１５０７を記載する。

【００４９】

図１２では処理間に依存がある際に、処理間のデータの再利用性を解析した。ただし、処理間に依存関係が無く、引用するデータが同じである場合にも再利用データ（即ち、例えば図７のソースコード７０１の例では、ｄ（ｉ）の演算時に用いているｘ（ｉ）に該当）となるので、図１６の手順で再利用データの調査を行う。図１６において、まず、評価対象となる処理ループを入力する（１６０１）。評価対象の処理に後続処理があるのか判断をして（１６０２）、無ければ再利用データの解析処理を終了する（１６０３）。１６

０２の判断で後続の処理がある場合、後続処理を１つ取り出して評価対象とする。評価対象の引用データがそれ以前の処理で利用したデータであるのか判断をする（１６０４）。

【００５０】

引用データが以前の処理で使われていない場合、主記憶データテーブルに登録していないときは、登録して（１６０５）、後続の処理の評価（１６０２）に移る。引用データが以前の処理に使われている場合、再利用する引用データが再利用データテーブルに登録済みか判断する（１６０６）。登録済みであれば後続処理の評価（１６０２）に戻る。登録されていない場合、主記憶データテーブルに記録されているのか確認する（１６０７）。主記憶データテーブルに記録されていなければ、再利用データテーブルへ登録して（１６０９）、後続の処理の評価（１６０２）に戻る。主記憶データテーブルへの確認（１６０７）で登録されていれば、主記憶データテーブルの登録を削除してから（１６０８）、再利用データテーブルへ登録して（１６０９）、後続の処理の評価（１６０２）に戻る。この操作を、評価対象の処理について実行して、図１４の再利用データテーブル及び図１５の主記憶データテーブルを作成する。

【００５１】

図４において、前述したような手順で処理内容テーブル４１４、再利用データテーブル４１５、主記憶データテーブル４１６が作成されると、その後、並列化方法評価部４０９による処理が実行される。図１７に、並列化方法評価部の詳細な評価手順を示す。図１７に示すように、並列化方法評価部４０９は、まず、ユーザによって並列化コード生成方針（１７０２）が入力され、その後、評価対象がループ処理だけしか含まないか判断する（１７０３）。ループ処理しか含まないときは、図１８で後述するようなキャッシュメモリデータの再利用性が無い場合のコード生成解析（１７０４ａ）を行い、この結果によって主記憶アクセス削減並列化法の解析を行うか演算負荷均等並列化法の解析を行うかを選択する（１７０４ｂ）。一方、ループ処理以外の処理を含むときは、キャッシュメモリ上のデータの再利用だけではなく、処理の分割も考慮する必要があるため、１７０４ａ，ｂの処理を実施せずに１７０５の処理に移行する。

【００５２】

次に、１７０３において評価対象にループ処理以外の処理が含まれる場合、または１７０４ｂにおいてキャッシュメモリデータの再利用性がある場合は、１７０２で入力された並列化コードの生成方針が「ｄａｔａ」であるのか判定する（１７０５）。「ｄａｔａ」である場合は、主記憶アクセス削減並列化法の解析（１７０６）を実施する。その後、演算負荷均等並列化法の解析（１７０７）と、実行サイクル数が最短となるコード生成の解析（１７０８）を実施する。一方、１７０５の並列化方針の判定で「ｄａｔａ」では無かった場合、または１７０４ｂでキャッシュメモリデータの再利用性が無いと判定された場合は、演算負荷均等並列化法の解析（１７０７）を行い、演算負荷均等並列化コードの生成（１７０９）に処理を移す。なお、１７０８および１７０９の処理は、図４の最適化コード生成部４１０の処理に相当する。

【００５３】

以下では、データ再利用性の無い場合のコードの生成解析（１７０４ａ）を図１８で、主記憶アクセス削減並列化法の解析（１７０６）を図２０で、演算負荷均等並列化法の解析（１７０７）を図２１で、実行サイクル数が最短となるコード生成の解析（１７０８）を図２２で説明する。

【００５４】

図１８に、評価対象処理ループを基に、データ再利用性の無い場合のコード解析の手順を示す。評価対象処理ループの入力（１８０１）の後、計算機条件の入力（１８０２）、再利用データテーブルの入力（１８０３）を行う。この計算機条件入力（１８０２）で使用する入力内容の例を図１９に示す。テーブルの項目としては、レジスタ数、プロセッサ数、独立キャッシュ容量、共有キャッシュ容量、キャッシュメモリからのロードコスト、主記憶からのロードコスト、主記憶からのストアコスト、同期取得コスト、他がある。

【００５５】

次いで、図１８において、再利用データテーブルに記載されている変数の再利用間隔（図１４の１４０７）とキャッシュメモリの容量を比較する（１８０４）。全ての再利用データの再利用間隔とキャッシュメモリの容量の比較（１８０５）の結果、キャッシュメモリ容量よりも再利用間隔が大きい場合、キャッシュメモリ上のデータの再利用性がないため、図１０で述べたようなキャッシュデータの再利用性を高めるような処理の分割（セクション分割）を検討する必要がある。そこで、各ＣＰＵの処理分割テーブルの生成（１８０６）を行って、主記憶アクセス削減並列化法を選択する（１８０７）。各ＣＰＵの処理分割テーブルは、各プロセッサが実行するループ制御変数の範囲と処理内容が記録され、並列コード生成の基になる。実際のコード生成の処理に関しては、従来技術を使用する。従来技術として例えば、非特許文献１、特許文献２に記載された技術が知られている。

【００５６】

全ての再利用データの再利用間隔とキャッシュメモリの容量の比較（１８０５）において、キャッシュメモリ容量の方が大きい場合、再利用データテーブルの変数量と再利用間隔をレジスタ数と比較して（１８０８）、変数量がレジスタ数よりも小さい場合、再利用データをレジスタ割当てるようにブロック化することが可能である（１８０９）。このときは、キャッシュメモリ上のデータの再利用ではなく、レジスタ上のデータの再利用となり、主記憶アクセス削減並列化法を適用してもキャッシュメモリのデータの再利用性が変化しない。そこで、各ＣＰＵの処理分割テーブルの作成を行い（１８１０）、演算負荷均等並列化法の選択を行う（１８１１）。

【００５７】

一方、再利用データテーブルの変数量と再利用間隔がレジスタ容量よりも大きいときは、主記憶アクセス削減並列化法の適用の評価をするため、図１３に示した処理内容テーブルの入力（１８１２）を行い、処理内容の評価をする。処理数と使用するプロセッサ数を比較して処理数が多い場合は、各処理を適宜複数のプロセッサに振り分けていく主記憶アクセス削減並列化法の適用効果が見込めるため、主記憶アクセス削減並列化法を選択する（１８１７）。また、依存処理、独立処理数がプロセッサ数よりも少ない場合でも、制御変数分割と処理分割の併用を判断して（１８１４）、併用できる場合は主記憶アクセス削減並列化法を選択する（１８１７）。制御変数分割と処理分割の併用を判断して（１８１４）、併用ができない場合、図１０のように各プロセッサに処理内容を振り分ける並列化方法が適用できないことになる。そこで、各ＣＰＵの処理分割テーブルの生成を行い（１８１５）、演算負荷均等並列化法の選択を行う（１８１６）。

【００５８】

図２０に、図１７の主記憶アクセス削減並列化法の解析（１７０６）の詳細手順を示す。図１８の解析の結果、主記憶アクセス削減コード生成の解析の開始に移る（２００１）。図２０の解析内容を大まかに説明すると、まず、依存のある処理を１プロセッサに割り当て、他の処理を残りのプロセッサに割り当て、処理と主記憶アクセスの負荷が均等になるようにして、実行に要するサイクル数を求める。その後、依存のある処理を割り当てるプロセッサ数を増やしながら、実行に要するサイクル数を求める。この操作を、依存のある処理の数とプロセッサの数の内、小さい方の数だけ繰り返して、処理サイクル数の最小値とそれを実現する処理分割法を求める。

【００５９】

以下、図２０の解析内容の詳細を説明するにあたり、まずは、図２９を用いて図２０の解析内容の具体的なイメージを説明し、その後に図２０の詳細な説明を行う。図２９は、図２０の補足図であり、図２０の処理内で生成される各ＣＰＵの処理分割テーブルの一例を示す図である。処理分割テーブルは、例えば図１３に示した処理内容テーブル内の複数の処理番号にそれぞれ対応する複数の処理を、各ＣＰＵ毎に分割したようなものである。図２９の例では、処理１１、処理１２、処理１３、処理１４に演算の依存があるものとする。最初の段階では、ＣＰＵ０においてこの依存のある処理１１～処理１４を全て実行しており、これらの実行サイクル数を図２０の処理で算出した結果が１５０となっている。

【００６０】

次に、この実行サイクル数を更に短くするため、ＣＰＵ１に処理１２を分割する。これに伴いＣＰＵ０とＣＰＵ１の間で同期を取る処理が加わる（図２９の「同期（０，１）」）。この場合に算出された実行サイクル数は１３０である。更に実行サイクル数を短くするためＣＰＵ２に処理１３を分担する。これに伴いＣＰＵ０とＣＰＵ２の間で同期を取る処理が加わる（図２９中の「同期（０，２）」）。この場合に算出された実行サイクル数は１２０である。このようにして、実行サイクル数が最小となる（この場合は１２０）となる処理分割方法が決定され、これに基づいて実際のコードが生成される。

【００６１】

図２０において、具体的には、まず、依存のある処理を１プロセッサに割り当て（２００２）、独立な処理を残りのプロセッサに割り当て（２００３）、分割方法を各ＣＰＵの処理分割テーブルに記録する（２００４ａ）。この処理分割テーブルでは、各プロセッサの識別番号毎にそれぞれが実行する処理内容が記述される。続いて、再利用データテーブルと主記憶データテーブルを更新する（２００４ｂ）。すなわち、処理の分割によって、再利用データテーブル内の再利用間隔１４０７が変わることがあり、これに応じて主記憶データテーブルにおける主記憶引用変数名１５０３も変わることがあるため、これらの見直しを行う。その後、処理内容テーブルや処理分割テーブルを参照して、各プロセッサの演算量からそれに要する演算サイクル数を算出する（２００５）。処理の分割方法毎の実行サイクル数を変数として、２００５で算出した演算サイクル数を代入する（２００６）。

【００６２】

次いで、依存処理を実行するときのプロセッサ間の同期回数を算出する（２００７）。依存処理を１プロセッサで実行するときにはプロセッサ間の同期は０回であるが、依存処理を複数のプロセッサに割り当てて実行する時には、処理の順番を保障するため、プロセッサ間の同期が必要になる。同期は依存処理を担当したプロセッサ間でのみ取得すればよいため、同期を取るプロセッサの識別番号を求めて、各ＣＰＵの処理分割テーブルに記録しておく。２００７で求めた同期回数に図１９の計算機条件に記載された同期取得コストをかけて、同期実行サイクル数を算出する（２００８）。ここで求めた同期実行サイクル数を実行サイクル数に足しこむ（２００９）。

【００６３】

続いて、処理で使用するデータのアクセスサイクル数を求める。再利用データテーブルに登録された変数を使った処理があるか判断する（２０１０）。この処理がある場合は、計算機のキャッシュメモリ容量が再利用データテーブルの１つの変数の再利用間隔よりも大きいのか判断して（２０１１）、データ量のカウントをする。キャッシュメモリ容量の方が再利用間隔よりも小さいときは、この変数はキャッシュメモリからデータが溢れてしまい、再利用する時には主記憶からデータを読み込むため、主記憶アクセスデータ量としてカウントアップする（２０１２）。その後、再利用データテーブルに登録された次の変数があるのか判断する（２０１０）。

【００６４】

また、２０１１でキャッシュメモリ容量よりも再利用データテーブルの再利用間隔が小さいと判断した場合は、変数を再利用するときにキャッシュメモリ上にあるため、キャッシュメモリアクセスデータ量としてカウントアップ（２０１３）した上で、２０１０の処理に移り、再利用データテーブルに登録された次の変数の評価を行う。２０１０で次に処理するべき変数がなくなった場合、主記憶アクセスサイクル数とキャッシュメモリアクセスサイクル数の算出をする。

【００６５】

具体的には、まず、主記憶データテーブルのデータ量を主記憶アクセスデータ量に足し込み（２０１４）、主記憶アクセスデータ量と主記憶ロードコストから主記憶アクセスサイクル数を算出して（２０１５）、実行サイクル数に主記憶アクセスサイクル数を足し込む（２０１６）。また、主記憶へのストアデータに関しては、図１５からデータ量が求まっているため、図１９の主記憶ストアコストを使って主記憶アクセスサイクル数が求めら

れるので、このサイクル数を実行サイクル数に足し込む。次に、キャッシュメモリアクセスデータ量とキャッシュロードコストからキャッシュアクセスサイクル数を算出し（２０１７）、キャッシュアクセスサイクル数を実行サイクル数に足し込む（２０１８）。これにより、１つの分割方法に対する実行サイクル数が求まる。

【００６６】

その後、依存処理部分の処理を更に分割した場合の実行サイクル数の算出に移る。まず、依存処理部分の更なる分割が可能であるか判断をする（２０１９）。分割ができない場合は、これまで算出した実行サイクル数で最小となる値を算出して（２０２１）、主記憶アクセス削減並列化法の決定となる（２０２２）。２０１９で依存処理が更に分割できる場合、分割した依存処理を新たに割当てるプロセッサが存在するのかどうか判断する（２０２０）。割当てるプロセッサが存在しない場合は、２０２１の処理に移り、実行サイクル数で最小となる値を算出して、主記憶アクセス削減並列化法の決定となる（２０２２）。２０２０で分割を割当てるプロセッサが存在する場合は、依存のある処理を実行するプロセッサ数を１つ増やして（２０２３）、２００３の処理に移行する。図２０のプロセスを実行することにより、実行サイクル数が最短となる、主記憶アクセス削減並列化法を求める。

【００６７】

図２１に、図１７の演算負荷均等並列化法の解析（１７０７）の手順を示す。図２１において、演算負荷均等化コード生成の解析を開始し（２１０１）、初めに、ループ制御変数分割を適用してプロセッサに処理の割付けを行う（２１０２）。この分割方法を各ＣＰＵの処理分割テーブルに記録する（２１０３ａ）。処理分割テーブルでは、各プロセッサの識別番号毎にそれぞれの処理内容を記述する。続いて、再利用データテーブルと主記憶データテーブルを更新する（２１０３ｂ）。ただし、この２１０３ｂの処理は、通常、ループ制御変数の分割を行っても再利用データテーブル内の再利用間隔１４０７が変わることはないため、場合によっては不要である。

【００６８】

次に、演算サイクル数とプロセッサ間の同期サイクル数をそれぞれ求め合計値を実行サイクル数として算出する。具体的には、まず、処理内容テーブルや処理分割テーブルを参照して各プロセッサの演算量を求め、図１９に記載の演算性能等から演算に要するサイクル数を求める（２１０４）。求めた演算サイクル数を実行サイクル数に代入する（２１０５）。処理を実行するときのプロセッサ間の同期回数を算出する（２１０６）。この同期回数に図１９で登録された同期取得コストをかけて同期実行サイクル数を算出する（２１０７）。求めた同期実行サイクル数を実行サイクル数に足し込む（２１０８）。

【００６９】

続いて、処理で使用するデータのアクセス時間を求める。使用するデータに関して、キャッシュメモリから入出力する場合と主記憶から入出力する場合に分けて、そのアクセス時間を求める。初めに、再利用データテーブルに登録されたデータがあるのか判断する（２１０９）。登録されたデータがある場合には、登録の順番に従い変数１つずつの評価を行う。評価する変数に関して、キャッシュメモリの容量と変数の再利用間隔の比較を行う（２１１０）。キャッシュメモリの容量が再利用間隔よりも小さい場合、この変数は主記憶から入出力されるデータとなり、主記憶アクセスデータ量としてカウントアップする（２１１１）。その後、再利用データテーブルに登録された次の変数があるのか判断する（２１０９）。２１１０でキャッシュメモリの容量が再利用間隔よりも大きい場合、再利用データはキャッシュメモリから入出力されるため、キャッシュメモリアクセスデータ量としてカウントアップする（２１１２）。その後、再利用データテーブルに登録された次の変数があるのか判断する（２１０９）。このループ処理を繰り返すことにより再利用データテーブルに登録された全ての変数を１つずつ解析する。

【００７０】

全ての解析が終わった時点で、主記憶アクセスのサイクル数とキャッシュメモリアクセスのサイクル数を求める。具体的には、２１０９で次に解析すべき登録変数がなくなった

場合に、主記憶データテーブルのデータ量を主記憶アクセスデータ量に足し込む（２１１３）。次に、主記憶アクセスデータ量と主記憶ロードコストから主記憶アクセスサイクル数を算出して（２１１４）、実行サイクル数に主記憶アクセスサイクル数を足し込む（２１１５）。また、主記憶へのストアデータに関しては、図１５からデータ量が求まっているため、図１９の主記憶ストアコストを使って主記憶アクセスサイクル数が求められるので、このサイクル数を実行サイクル数に足し込む。

【００７１】

キャッシュメモリアクセスに関しては、キャッシュメモリアクセスデータ量とキャッシュメモリロードコストからキャッシュアクセスサイクル数を算出して（２１１６）、キャッシュアクセスサイクル数を実行サイクル数に足し込む（２１１７）。これらの手順により、各ＣＰＵの処理分割テーブルと実行サイクル数が求まり、演算負荷均等並列化法の分割法とそれに対応する実行サイクルの決定となる（２１１８）。

【００７２】

図２２に、図１７の実行サイクル数が最短のコード生成の解析（１７０８）の詳細手順を示す。図２２の並列化法による実行サイクル数の評価（２２０１）で、図２０で求めた主記憶アクセス削減並列化法のサイクル数（２２０２）と図２１で求めた演算負荷均等並列化法のサイクル数（２２０３）を入力する。演算負荷均等並列化法のサイクル数と主記憶アクセス削減並列化法のサイクル数を比較して（２２０４）、演算付加均等並列化法のサイクル数が小さい場合は、各ＣＰＵの処理分割テーブルとして、演算付加均等並列化法を適用したときに図２１の２１１８で求まった各ＣＰＵの処理分割テーブルを記録して（２２０５）、演算負荷均等並列化コード生成の処理に移行する（２２０６）。各ＣＰＵの処理分割テーブルでは、実行するプロセッサ毎に識別番号を付けて、各プロセッサが行う処理を記録する。そのため、プロセッサ毎に違う処理をするときには、コード生成の段階で識別番号に応じた条件分岐を埋め込むことにより、制御できる。

【００７３】

一方、２２０４において、主記憶アクセス削減並列化法の実行サイクル数の方が小さい場合は、各ＣＰＵの処理分割テーブルに主記憶アクセス削減並列化法の分割を記録してから（２２０７）、主記憶アクセス削減並列化コードの生成処理に移行する（２２０８）。この場合も、各ＣＰＵの処理分割テーブルでは、実行するプロセッサ毎に識別番号を付けて、各プロセッサが行う処理、及び特定のプロセッサで取得する同期処理を記録する。そのため、プロセッサ毎に違う処理をするときには、コード生成の段階で識別番号に応じた条件分岐を埋め込むことにより、制御できる。

【００７４】

前述した演算負荷均等並列化法の指針に沿った実際のコード生成方法または主記憶アクセス削減並列化法の指針にそった実際のコード生成方法は、特許文献２、非特許文献１、非特許文献３の方法が知られている。

【００７５】

以上、本実施の形態１による最適コード生成方法を用いることで、主記憶アクセス削減の観点と、プロセッサによる演算負荷均等の観点と、それらの組合せを考慮して、実際の実行サイクル時間を最短となる最適な並列コードを生成できる。そして、これによって、特に複数のＳＩＭＤ型またはベクトル型プロセッサの実行効率を高めることが可能となる。

【００７６】

（実施の形態２）

本実施の形態２では、前述した実施の形態１の図１でＣＰＵが共有キャッシュを備えない場合のコード生成方法の一例を示す。図２３は、本発明の実施の形態２によるコード生成方法において、その生成されたコードを用いて制御を実行する計算機システムの一例を示した構成図である。図２３の計算機システムは、図１の計算機システムから共有キャッシュを省いた構成となっており、計算ノード群２３０５の中で各プロセッサ（ＣＰＵ）２３０６に共有メモリ２３０７が接続された構成となっている。

【００７７】
このように、複数のプロセッサ間に共有キャッシュを備えていない場合でも、実施の形態１と同様に２種類の並列化法がある。例えば図６の６０６で「ｃａｌｃｕｌａｔｉｏｎ」を選択した場合は、ループ制御変数を分割する演算付加均等並列化法が適用され、「ｄａｔａ」を選択した場合は、通信関数や配列データの読み込みも分割する主記憶アクセス削減並列化法が適用される。実施の形態１との違いは、キャッシュメモリが無いため、キャッシュメモリアクセスサイクル数の算出が不要になることである。この算出に関与する処理は、図４の並列化方法評価部４０９である。図１７に示した並列化方法評価部４０９の手順でキャッシュメモリの評価に関係する部分は、主記憶アクセス削減並列化方法の解析（１７０６）と演算負荷均等並列化法の解析（１７０７）である。

【００７８】
図２４にキャッシュメモリの評価をしない場合の主記憶アクセス削減並列化法の解析手順を示す。図２４の２４０１～２４０９は、分割した演算のサイクル数とプロセッサ間のサイクル数の算出であり、実施の形態１で示したキャッシュメモリの評価を含む図２０の２００１～２００９とほぼ同様である。ただし、キャッシュメモリが存在しないため、図２０の２００４ｂにおける再利用データテーブルおよび主記憶データテーブルの更新処理が不要となっている点が異なっている。

【００７９】
次に、主記憶アクセスのサイクル数の算出を行う。キャッシュメモリを搭載していない計算機システムの場合は、処理に必要な全てのデータを主記憶へのアクセスで得るため、主記憶データテーブルと再利用データテーブルのデータ量の合計を主記憶アクセスデータ量とする（２４１０）。主記憶アクセスデータ量と主記憶ロードコストから主記憶アクセスサイクル数を算出する（２４１１）。この主記憶アクセスサイクル数を実行サイクル数に足し込む（２４１２）。また、主記憶へのストアデータに関しては、図１５からデータ量が求まっており、図１９の主記憶ストアコストを使って主記憶アクセスサイクル数が求められるので、このサイクル数を実行サイクル数に足し込む。これにより、１つの分割法による全体の実行サイクル数が求められる。この後、依存処理部を他のプロセッサに分割して最適な分割方法を求める。このときの手順である２４１３～２４１７は、キャッシュメモリを備えた場合の図２０の２０１９～２０２３と同じである。

【００８０】
図２５にキャッシュメモリを搭載しない計算機システムの演算負荷均等並列化法の解析手順を示す。図２５の２５０１～２５０８は、分割した演算のサイクル数とプロセッサ間のサイクル数の算出であり、実施の形態１に示したキャッシュメモリの評価を含む図２１の２１０１～２１０８とほぼ同様である。ただし、キャッシュメモリが存在しないため、図２１の２１０３ｂにおける再利用データテーブルおよび主記憶データテーブルの更新処理が不要となっている点が異なっている。

【００８１】
次に、主記憶アクセスのサイクル数の算出を行う。キャッシュメモリを搭載していない計算機システムの場合は、処理に必要な全てのデータを主記憶へのアクセスで得るため、主記憶データテーブルと再利用データテーブルのデータ量の合計を主記憶アクセスデータ量とする（２５０９）。主記憶アクセスデータ量と主記憶ロードコストから主記憶アクセスサイクル数を算出する（２５１０）。主記憶アクセスサイクル数を実行サイクル数に足し込む（２５１１）。主記憶へのストアデータに関しては、図１５からデータ量が求まっており、図１９の主記憶ストアコストを使って主記憶アクセスサイクル数が求められるので、このサイクル数を実行サイクル数に足し込む。これにより、演算負荷均等並列化法の分割法と実行サイクル数を決める（２５１２）。この後に実行される、図１７の実行サイクル数が最短のコード生成の解析（１７０８）は、キャッシュメモリを備えている実施の形態１の場合と同様に図２２の手順に従って処理を行う。

【００８２】
以上、本実施の形態２による最適コード生成方法を用いることで、ＣＰＵが共有キャッ

シュを備えない場合においても実行サイクル時間が最短となる最適コードを生成可能となる。

【００８３】

（実施の形態３）

本実施の形態２では、前述した実施の形態１の図７でソースコード７０１のループ制御変数が定数ではなく変数で与えられている場合のコード生成方法の一例を示す。図２６は、本発明の実施の形態３によるコード生成方法において、その評価対象となるソースコードの一例を示す図である。図２６に示すソースコード２６０１は、図７のソースコード７０１において定数となっているループ制御変数ｉ＝１～４００が、変数ｉ＝ｍ～ｎで与えられている。

【００８４】

このような場合は、図１２における依存処理と独立処理の解析及びデータ再利用性の解析の手順と、図１６における利用データの再利用性の解析手順はデータ量を変数のまま解析する。図１４の再利用データテーブルの例と図１５の主記憶データテーブルの例でデータ量、ストライド幅、再利用間隔が未定の場合はループ制御変数に依存した式で表す。図１７の並列化方法の評価の際、ループ制御変数の値により、実行サイクル数が最短となる並列化方法が変わる場合には、ループ制御変数の値による条件分岐を付けて、複数の並列コードを生成する。例えば図２６の例において、ループ制御変数ｍ，ｎの値が実行時に決められるものとし、この場合における生成コードの例を図２７に示す。

【００８５】

図２７に示す生成コードは、コードを計算機上で実行する時に決まるループ制御変数ｍとｎの値によって、並列化法１（主記憶アクセス削減並列化法）２７０１、並列化法２（主記憶アクセス削減並列化法）２７０２、並列化３（演算負荷均等並列化法）を切り替えて実行するものとなっている。そのため、コンパイラ等によるコード生成の際は、従来のように１つの並列コードを生成するのではなく、対応する並列コードを複数生成することになる。そして、計算機システムは、このような複数の並列コードをメモリ上に配置し、実行時に動的に算出されたループ制御変数の値を判断して、その後に実行する並列コードを適宜選択するような動作を行う。

【００８６】

以上、本実施の形態３による最適コード生成方法を用いることで、ループ制御変数の値が動的に変更する場合であっても、実行サイクル時間が最短となる最適コードを生成することが可能となる。

【００８７】

以上、本発明者よりなされた発明を実施の形態に基づき具体的に説明したが、本発明は前記実施の形態に限定されるものではなく、その要旨を逸脱しない範囲で種々変更可能であることは言うまでもない。

【産業上の利用可能性】

【００８８】

本発明によれば、キャッシュメモリまたは主記憶を共有した複数のプロセッサから構成される計算機に対して、各プロセッサの負荷を均等に分割する並列化と主記憶へのアクセスを削減する並列化の中から最も実行時間が短くして実行効率の高い並列化方法のコードを生成できる。並列処理に適したコンパイラまたはコード生成に適用できる。

【図面の簡単な説明】

【００８９】

【図１】本発明の実施の形態１によるコード生成方法において、その生成されたコードを用いて制御を実行する計算機システムの一例を示した構成図である。

【図２】演算の並列化の一例を示す説明図である。

【図３】本発明の実施の形態１によるコード生成方法において、そのコードを生成する計算機システムの一例を示した構成図である。

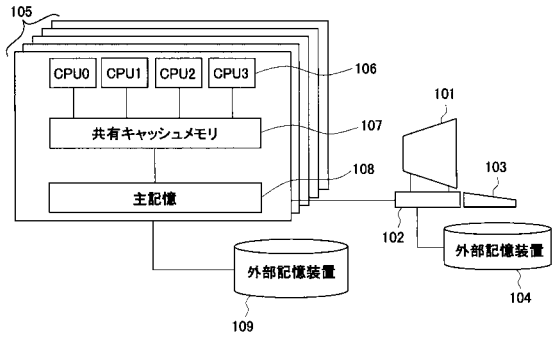
【図４】図３の計算機システムにおいて、その並列化コードを生成する機能の詳細な一例

を示した構成図である。
【図５】コンパイルオプションによって並列化方針の指示を与えた場合の具体例を示す説明図である。
【図６】ソースコードに対して並列化方針の指示を与えた場合の具体例を示す説明図である。
【図７】ソースコードから生成される中間語の具体例を示す説明図である。
【図８】図７の中間語に対して演算並列化を行った場合の具体例を示す説明図である。
【図９】ベクトルプロセッサを対象に、演算負荷均等並列化法を用いて生成された中間語の具体例を示す説明図である。
【図１０】ベクトルプロセッサを対象に、主記憶アクセス削減並列化法を用いて生成された中間語の具体例を示す説明図である。
【図１１】図４におけるループ演算切り出し部の詳細な処理内容の一例を示すフロー図である。
【図１２】図４における依存処理と独立処理解析部、データ再利用性解析部及び主記憶データ引用解析部の詳細な処理内容の一例を示すフロー図である。
【図１３】図４における処理内容テーブルの構成例を示す説明図である。
【図１４】図４における再利用データテーブルの構成例を示す説明図である。
【図１５】図４における主記憶データテーブルの構成例を示す説明図である。
【図１６】図４のデータ再利用性解析部において、図１２に加えて行われる詳細な処理内容の一例を示すフロー図である。
【図１７】図４の並列化方法評価部の詳細な処理内容の一例を示すフロー図である。
【図１８】図１７のフローにおいて、キャッシュメモリ上のデータ再利用性が無い場合のコード生成手順の一例を示すフロー図である。
【図１９】図４における計算機条件入力部の詳細な一例を示す説明図である。
【図２０】図１７における主記憶アクセス削減並列化法の解析を行う際の詳細な処理内容の一例を示すフロー図である。
【図２１】図１７における演算負荷均等並列化法の解析を行う際の詳細な処理内容の一例を示すフロー図である。
【図２２】図１７における実行サイクル数が最短のコード生成の解析を行う際の詳細な処理内容の一例を示すフロー図である。
【図２３】本発明の実施の形態２によるコード生成方法において、その生成されたコードを用いて制御を実行する計算機システムの一例を示した構成図である。
【図２４】図２３の計算機システムを対象として主記憶アクセス削減並列化法の解析を行う際の詳細な処理内容の一例を示すフロー図である。
【図２５】図２３の計算機システムを対象として演算負荷均等並列化法の解析を行う際の詳細な処理内容の一例を示すフロー図である。
【図２６】本発明の実施の形態３によるコード生成方法において、その評価対象となるソースコードの一例を示す図である。
【図２７】本発明の実施の形態３によるコード生成方法において、その生成された並列コードの一例を示す図である。
【図２８】図１１の補足図である。
【図２９】図２０の処理内で生成される各ＣＰＵの処理分割テーブルの一例を示す図である。
【符号の説明】
【００９０】
１０１，３０１，２３０１　ディスプレイ装置
１０２，２３０２　端末装置
１０３，３０２，２３０３　キーボード
１０４，２３０４　外部記憶装置
１０５，２３０５　計算ノード群

１０９，３０５，２３０９　外部記憶装置
１０８，３０４　主記憶装置
１０７　キャッシュメモリ
１０６，３０３，２３０６　プロセッサ（ＣＰＵ）
３０１　ディスプレイ装置
３０２　キーボード
３１４，４０１，７０１，２６０１　ソースコード
３１５，４１１　並列コード
３０６，４０２　構文解析部
３０７，４０３　中間コードの生成部
３０８，４０４　解析部
３０９，４１４　処理内容テーブル
３１０，４１５　再利用データテーブル
３１１，４１６　主記憶データテーブル
３１２，４１７　処理分割テーブル
３１３　最適化コード生成部
４００　コード変換部（コンパイラ）
４０５　ループ演算切り出し部
４０６　依存処理と独立処理解析部
４０７　データの再利用性解析部
４０８　主記憶データ引用解析部
４０９　並列化方法評価部
４１２　並列化コード生成方針
４１３　計算機条件入力部
４１０　最適化コード生成部
７０２　中間コード
２３０７　共有メモリ

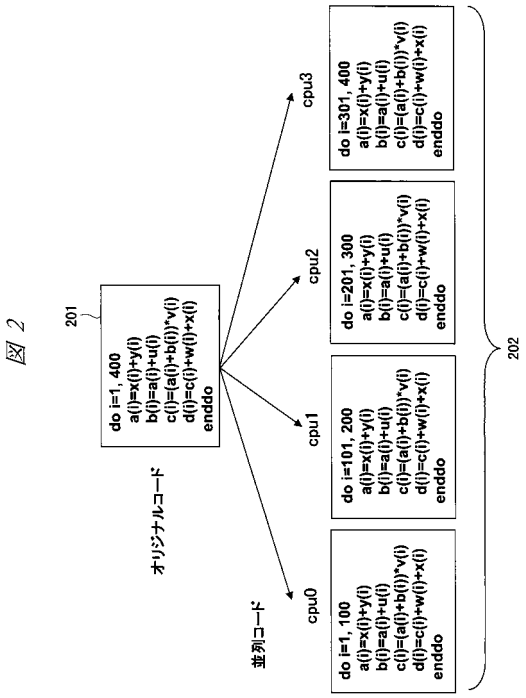
【図１】

図 1



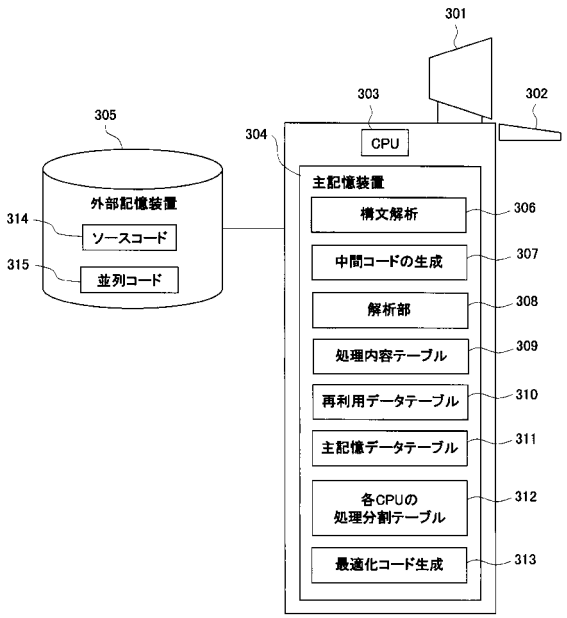
【図２】

図 2



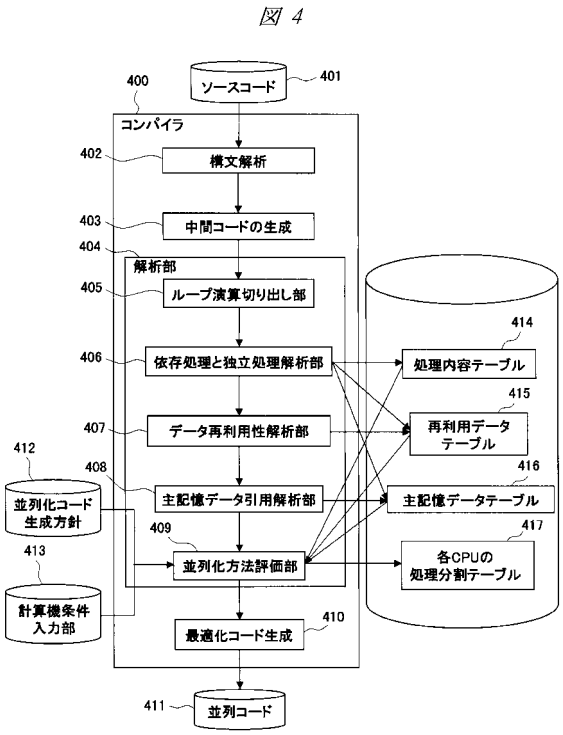
【図３】

図 3



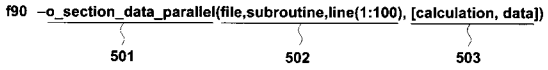
【図４】

図 4



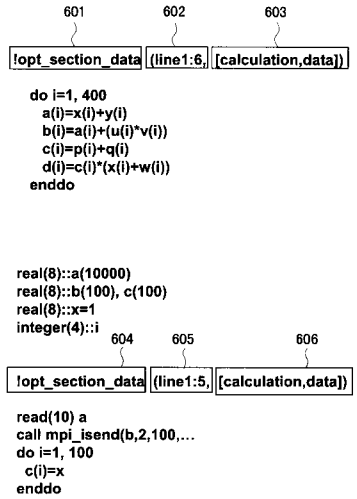
【図５】

図 5



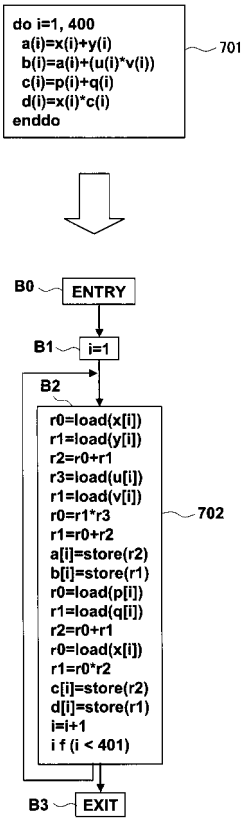
【図６】

図 6



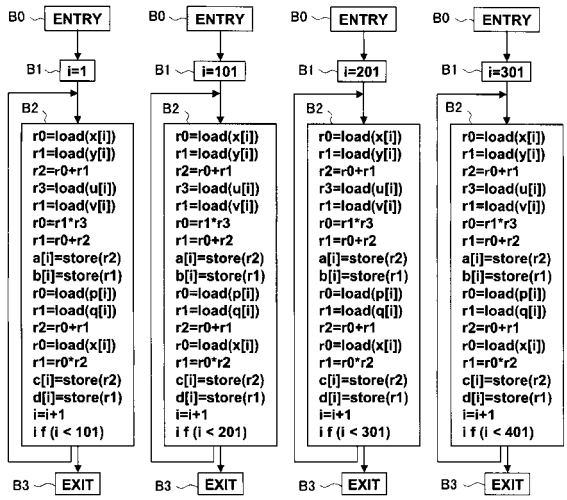
【図７】

図 7



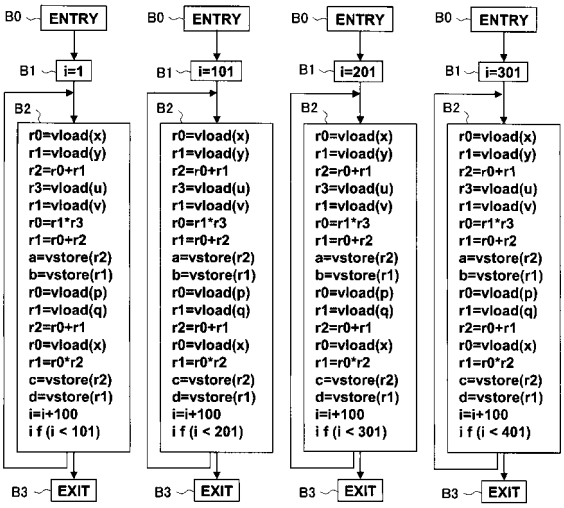
【図８】

図 8



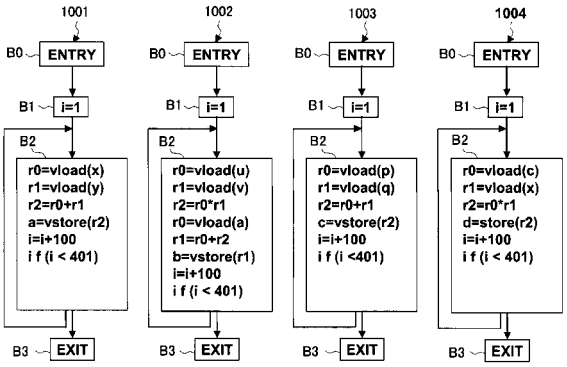
【図９】

図 9



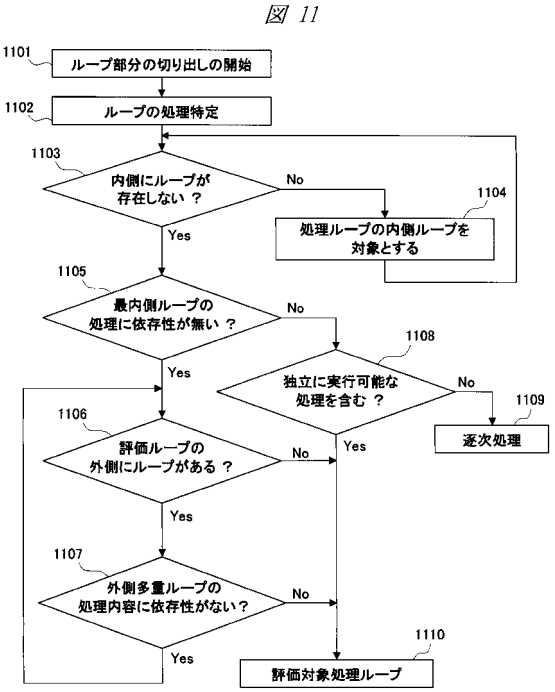
【図１０】

図 10



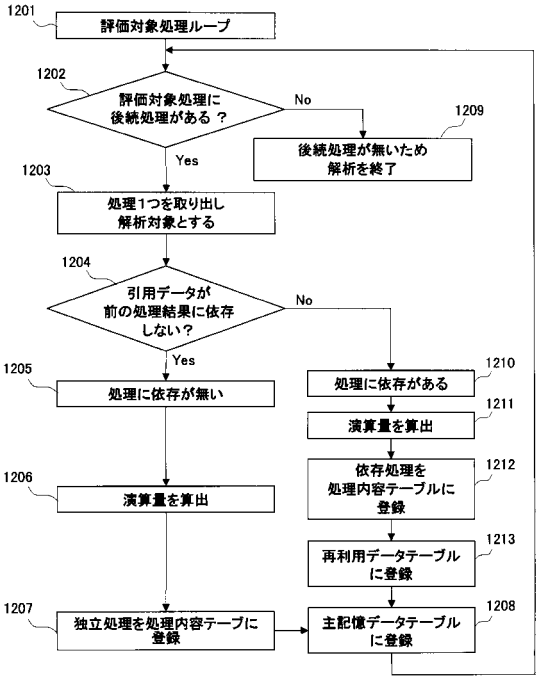
【図１１】

図 11



【図１２】

図 12



【図１３】

図 13

1301 処理番号	1302 処理項目	1303 依存性の有無	1304 利用変数名	1305 依存変数名	1306 演算量 [Flop]	1307 並列化法
1	加算	無し	x[i], y[i], a[i]		400	主記憶優先
2						
⋮						
5	乗算	有り	d[i],c[i],x[i]	c[i]	800	主記憶優先

【図１４】

図 14

1401 処理番号	1402 補助番号	1403 再利用変数名	1404 データ長[Byte]	1405 要素数	1406 ストライド幅[Byte]	1407 再利用間隔[Byte]
1						
2						
⋮						
5	1	c[i]	3200	400	8	3200
5	2	x[i]	3200	400	8	25600

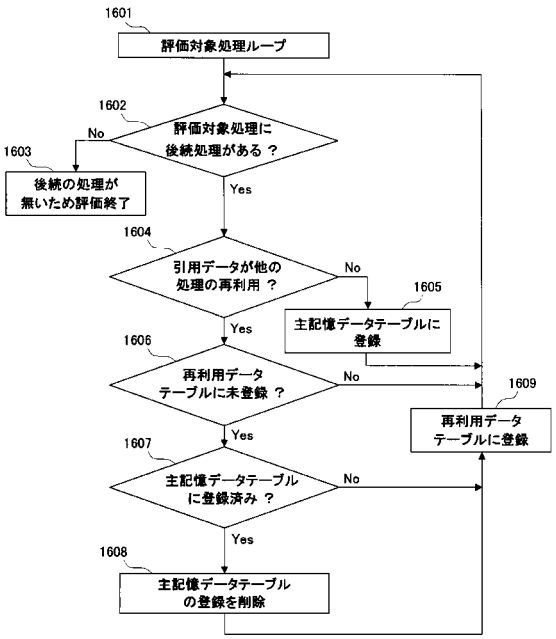
【図１５】

図 15

1501 処理番号	1502 補助番号	1503 主記憶引用変数名	1504 データ長[Byte]	1505 要素数	1506 ストライド幅[Byte]	1507 種類
1	1	x[i]	3200	400	8	load
1	2	y[i]	3200	400	8	load
⋮						

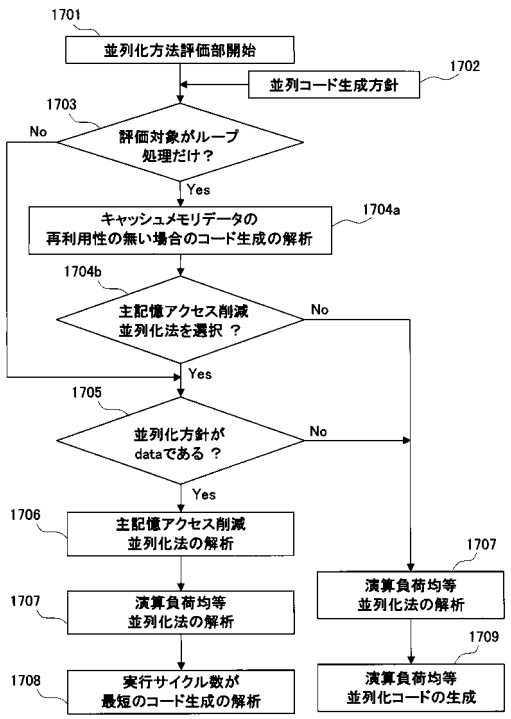
【図１６】

図 16



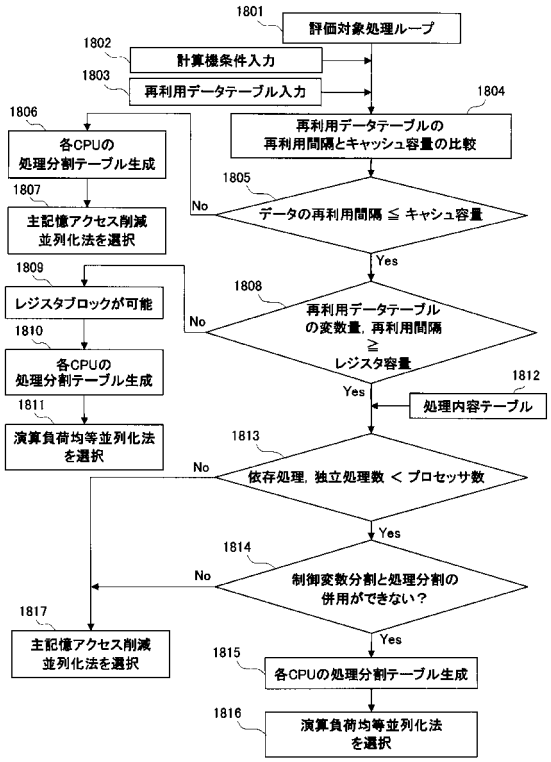
【図１７】

図 17



【図１８】

図 18



【図１９】

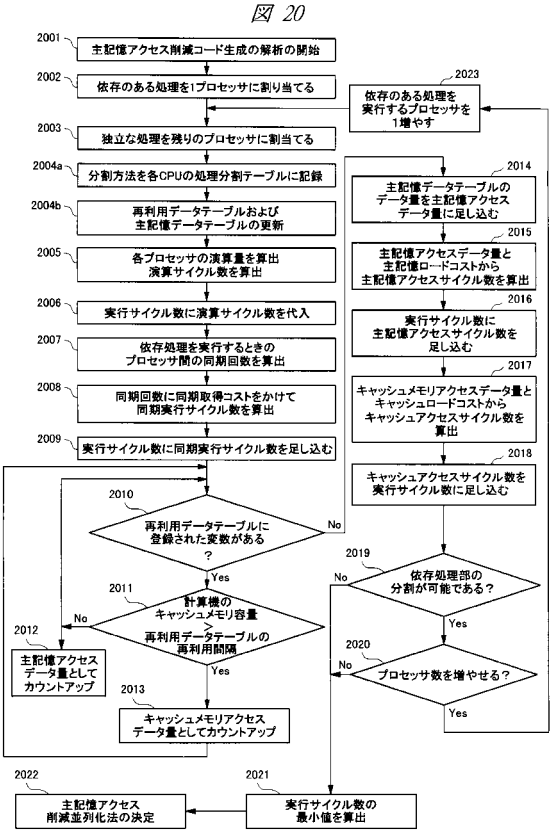
図 19

1901

No.	項目	数値
1	レジスタ数	128
2	プロセッサ数	4
3	独立キャッシュ容量[Byte]	32000
4	共有キャッシュ容量[Byte]	1000000
5	キャッシュロードコスト[cycle/Byte]	6
6	主記憶ロードコスト[cycle/Byte]	100
7	主記憶ストアコスト[cycle/Byte]	120
8	同期取得1コスト[cycle]	200
9	同期取得2コスト[cycle]	150
10	演算性能[flop/cycle]	4
11	⋮	

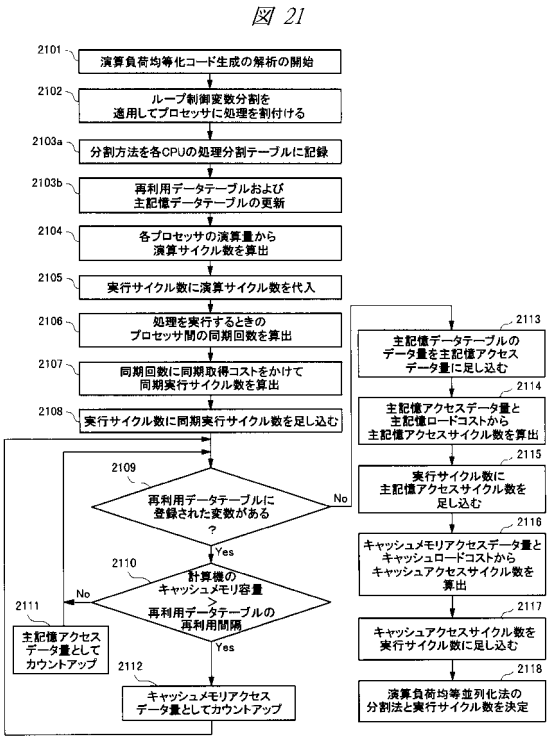
【図２０】

図 20



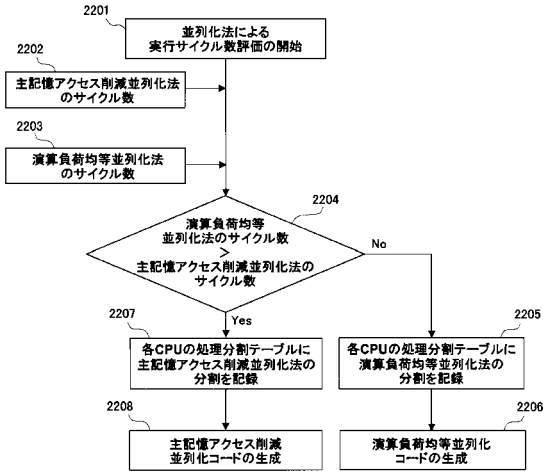
【図２１】

図 21



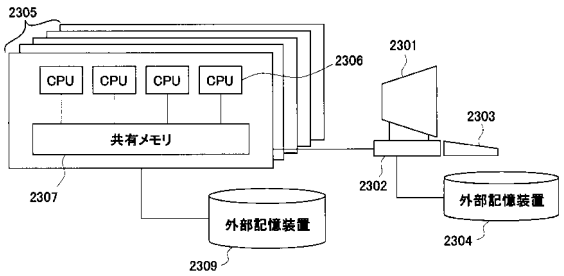
【図２２】

図 22



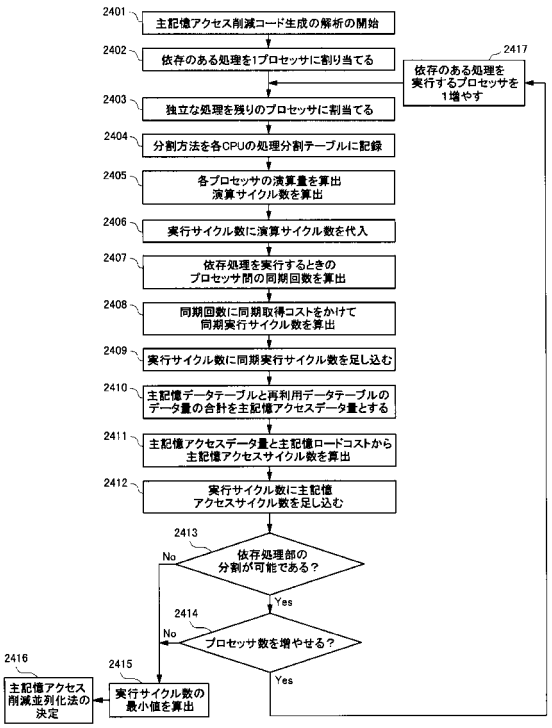
【図２３】

図 23



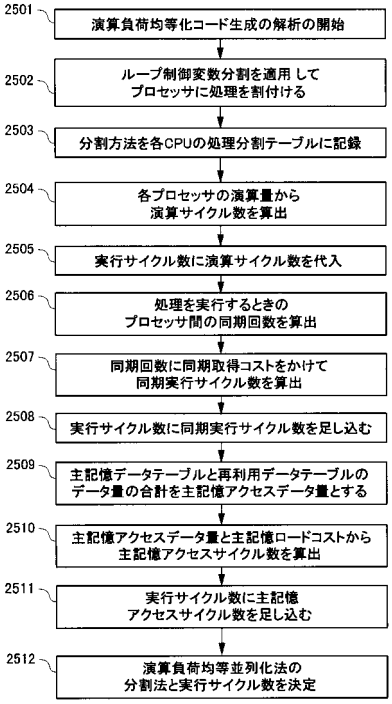
【図２４】

図 24



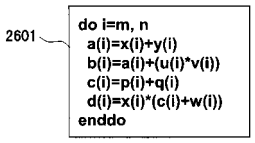
【図２５】

図 25



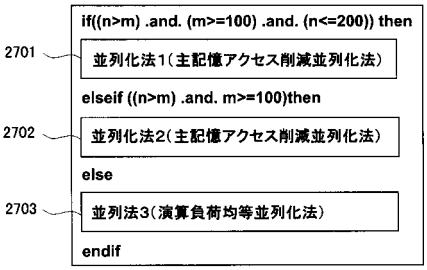
【図２６】

図 26



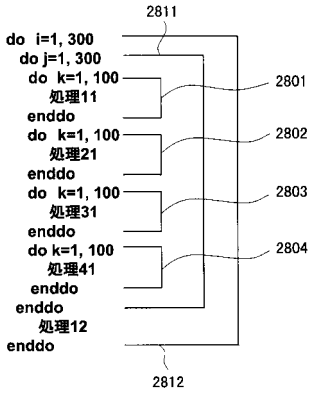
【図２７】

図 27



【図２８】

図 28



【図２９】

図 29

実行サイクル数	CPU0	CPU1	CPU2	CPU3
150	処理11 処理12 処理13 処理14	処理21	処理31	処理41
130	処理11 処理13 処理14	処理21 同期(0, 1) 処理12	処理31	処理41
120	処理11 処理14	処理21 同期(0,1) 処理12	処理31 同期(0, 2) 処理13	処理41

フロントページの続き

(56)参考文献 特開平２－１７６９３８（ＪＰ，Ａ）
特開平４－２９３１５０（ＪＰ，Ａ）
特開平５－２６５７６９（ＪＰ，Ａ）
特開平１０－１０５４１２（ＪＰ，Ａ）
特開平１１－１３４１９９（ＪＰ，Ａ）
特開２００４－３０３１１３（ＪＰ，Ａ）
特開２００６－２５９８２１（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／４５
Ｇ０６Ｆ　１２／０８