(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4783005号
(P4783005)
(45)発行日　平成23年9月28日(2011.9.28)　(24)登録日　平成23年7月15日(2011.7.15)

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

請求項の数 28 (全 34 頁)

(21)出願番号　特願2004-341236(P2004-341236)
(22)出願日　平成16年11月25日(2004.11.25)
(65)公開番号　特開2006-154971(P2006-154971A)
(43)公開日　平成18年6月15日(2006.6.15)
審査請求日　平成19年11月22日(2007.11.22)

(73)特許権者　000005821
パナソニック株式会社
大阪府門真市大字門真１００６番地
(74)代理人　100090446
弁理士　中島　司朗
(72)発明者　畑野　文博
大阪府門真市大字門真１００６番地　松下電器産業株式会社内
(72)発明者　田中　旭
大阪府門真市大字門真１００６番地　松下電器産業株式会社内

審査官　坂庭　剛史

最終頁に続く

(54)【発明の名称】プログラム変換装置、プログラム変換実行装置およびプログラム変換方法、プログラム変換実行方法。

(57)【特許請求の範囲】
【請求項１】
条件分岐を含むソースプログラムを変換して、２以上の命令を並列して実行できるコンピュータを対象とする目的プログラムを生成するプログラム変換装置であって、
前記ソースプログラムにおいて、条件分岐を跨ぐ一区間についての複数の実行経路のうちの一つの実行経路を指定する実行経路指定手段と、
前記区間にある全ての命令群を基に、その命令群に相当する第一コード列を生成する第一コード列生成手段と、
前記実行経路指定手段によって指定される実行経路上の命令群だけに相当する第二コード列を生成し、当該生成において条件分岐命令については、他の実行経路への分岐条件が成立しない場合に前記区間における当該条件分岐命令以降の命令を続行し、他の実行経路への分岐条件が成立する場合に前記区間における当該条件分岐以降の命令の実行を中止するコードを当該条件分岐命令に相当するコードとして生成する第二コード列生成手段と、
前記ソースプログラムの前記区間に後続する部分の命令群を基に、その命令群に相当する第三コード列を生成する第三コード列生成手段と、
前記第一コード列と、前記第二コード列とを、前記コンピュータに並列実行させるように、かつ、前記第二コード列において他の実行経路への分岐条件が成立しない場合には前記第二コード列に継続して前記第三コード列を実行させ、前記第二コード列において他の実行経路への分岐条件が成立する場合には第一コード列に継続して第三コード列を実行させるように編成した目的プログラムを生成する目的プログラム生成手段とを備える

ことを特徴とするプログラム変換装置。
【請求項２】
前記目的プログラム生成手段は、
前記コンピュータにおいて前記第一コード列の終了が前記第二コード列の終了よりも早い場合には、前記第二コード列を実行している前記コンピュータのプロセッサエレメントに第二コード列の実行を停止させるコードを前記第一コード列の後に含んで編成された目的プログラムを生成する
ことを特徴とする請求項１記載のプログラム変換装置。
【請求項３】
前記プログラム変換装置は更に、
前記ソースプログラムを実行形式に変換した実行プログラムをコンピュータに実行させることで、前記区間において実行された頻度が高い順に、当該頻度が第一位の実行経路を当該コンピュータから取得する実行経路取得手段を備え、
前記実行経路指定手段は、前記取得手段により取得された前記第一位の実行経路を指定する
ことを特徴とする請求項１記載のプログラム変換装置。
【請求項４】
前記プログラム変換装置は更に、
前記コンピュータが並列実行可能な命令数ｍを取得する命令上限取得手段を備え、
前記実行経路取得手段は更に、
前記区間における実行頻度が第２位以下の実行経路を取得し、
前記実行経路指定手段は、
前記実行経路を前記命令数ｍに基づき、前記実行経路取得手段によって取得された、第一位から第ｎ（ｎ＝ｍ－１）位までの実行経路を指定し、
前記第二コード列生成手段は、前記実行経路指定手段によって指定された第一位から第ｎ位までの実行経路を、実行経路ごとに合計ｎ個のコード列に変換し、
前記目的プログラム生成手段は、
前記第一コード列と前記第二コード列生成手段により生成された前記ｎ個のコード列を並列実行させるようにコードを編成した目的プログラムを生成する
ことを特徴とする請求項３記載のプログラム変換装置。
【請求項５】
前記第二コード列生成手段は更に、
前記第二コード列生成手段により生成された第一位から第ｎ位のｎ個のコードのうち、他の実行経路への条件分岐が発生しなかったコード列以外のコード列を停止させる停止コードを含んで生成する
ことを特徴とする請求項４記載のプログラム変換装置。
【請求項６】
前記プログラム変換装置は更に、
前記コンピュータのメモリの形態が、前記コンピュータの全てのプロセッサエレメントが一つのメモリを共有するメモリ共有型であるか、前記コンピュータの全てのプロセッサエレメントが固有のメモリを有するメモリ分散型かであるかのいずれの形態のメモリを使用しているかの情報を取得するメモリ情報取得手段を備え、
前記目的プログラム生成手段は、前記メモリ情報取得手段により取得したメモリ情報に基づき、メモリ共有型である場合に、前記第一コード列と前記第二コード列において利用される前記ソースプログラム中の元となる変数がそれぞれ独立した変数として扱うコードになっている目的プログラムを生成する
ことを特徴とする請求項１記載のプログラム変換装置。
【請求項７】
前記目的プログラム生成手段は、
前記停止コードによって停止させられたスレッドをプロセッサエレメントが消去せずに

保持しておくコードを含んで目的プログラムを生成する
ことを特徴とする請求項５記載のプログラム変換装置。
【請求項８】
前記プログラム変換装置は更に、
前記目的プログラムを前記コンピュータに適合するように機械語に変換する機械語変換手段を備える
ことを特徴とする請求項１記載のプログラム変換装置。
【請求項９】
条件分岐を含むソースプログラムを実行形式である実行形式プログラムに変換して、かつ、２以上の命令を並列して実行できるプログラム変換実行装置であって、
前記ソースプログラムにおいて、条件分岐を跨ぐ一区間についての複数の実行経路のうちの一つの実行経路を指定する実行経路指定手段と、
前記区間にある全ての条件分岐を含んだ命令群を基に、その命令群に相当する第一コード列を生成する第一コード列生成手段と、
前記第一コード列を含む第一プログラムを実行する実行手段と、
前記実行手段が前記第一プログラムを実行することにより得られた前記区間における実行経路のうち実行頻度が高い順に、当該実行頻度が第一位の実行経路を取得する取得手段と、
前記取得手段によって取得された実行経路を前記実行経路指定手段によって指定し、当該実行経路上の命令群に相当する第二コード列を生成し、当該生成において条件分岐命令については、他の実行経路への分岐条件が成立しない場合に前記区間における当該条件分岐命令以降の命令を続行し、他の実行経路への分岐条件が成立する場合に前記区間における当該条件分岐以降の命令の実行を中止するコードを当該条件分岐命令に相当するコードとして生成する第二コード列生成手段と、
前記ソースプログラムの前記区間に後続する部分の命令群を基に、その命令群に相当する第三コード列を生成する第三コード列生成手段と、
前記第一コード列と、前記第二コード列とを、並列実行するように、かつ、前記第二コード列において他の実行経路への条件分岐が発生しない場合には前記第二コード列に継続して前記第三コード列を実行し、前記第二コード列において他の実行経路への条件分岐が発生する場合には前記第一コード列に継続して前記第三コード列を実行するように編成した目的プログラムを生成する目的プログラム生成手段とを備え、
前記実行手段は前記第一プログラムを実行する代わりに前記目的プログラムを実行する
ことを特徴とするプログラム変換実行装置。
【請求項１０】
前記目的プログラム生成手段は、
前記コンピュータにおいて前記第一コード列の終了が前記第二コード列の終了よりも早い場合には、前記第二コード列を実行している前記コンピュータのプロセッサエレメントに第二コード列の実行を停止させるコードを前記第一コードの後に含んで編成されたプログラムを生成する
ことを特徴とする請求項９記載のプログラム変換実行装置。
【請求項１１】
前記プログラム変換実行装置は更に、
当該プログラム変換実行装置が並列実行可能な命令数ｍを取得する命令上限取得手段を備え、
前記実行経路取得手段は更に、
前記区間における実行頻度が第２位以下の実行経路を取得し、
前記実行経路指定手段は、
前記実行経路を前記命令数ｍに基づき、前記実行経路取得手段によって取得された、第一位から第ｎ（ｎ＝ｍ－１）位までの実行経路を指定し、
前記第二コード生成手段は、前記実行経路指定手段によって指定された第一位から第ｎ

位までの実行経路を、実行経路ごとに合計ｎ個のコード列に変換し、
前記目的プログラム生成手段は、
前記第一コード列と前記第二コード列生成手段により生成された前記ｎ個のコード列を並列実行させるようにコードを編成した目的プログラムを生成する
ことを特徴とする請求項１０記載のプログラム変換実行装置。
【請求項１２】
前記第二コード列生成手段は更に、
前記第二コード列生成手段により生成された第一位から第ｎ位のｎ個のコード列のうち、他の実行経路への条件分岐が発生しなかったコード列以外のコード列を停止させる停止コードを含んで生成する
ことを特徴とする請求項１１記載のプログラム変換実行装置。
【請求項１３】
前記目的プログラム生成手段は、
自機のメモリの形態が、全てのプロセッサエレメントが一つのメモリを共有するメモリ共有型である場合に、前記第一コード列と前記第二コード列において利用される前記ソースプログラム中の元となる変数がそれぞれ独立した変数として扱うコードになっている目的プログラムを生成する
ことを特徴とする請求項９記載のプログラム変換実行装置。
【請求項１４】
前記目的プログラム生成手段は、
前記停止コードによって停止させられたスレッドをプロセッサエレメントが消去せずに保持しておくコードを含んで目的プログラムを生成する
ことを特徴とする請求項１２記載のプログラム変換実行装置。
【請求項１５】
プログラム変換装置が実行する、条件分岐を含むソースプログラムを変換して２以上の命令を並列して実行できるコンピュータを対象とする目的プログラムを生成するプログラム変換方法であって、
前記ソースプログラムにおいて、条件分岐を跨ぐ一区間についての複数の実行経路のうちの一つの実行経路を指定する実行経路指定ステップと、
前記区間にある全ての命令群を基に、その命令群に相当する第一コード列を生成する第一コード列生成ステップと、
前記実行経路指定ステップにおいて指定される実行経路上の命令群だけに相当する第二コード列を生成し、当該生成において条件分岐命令については、他の実行経路への分岐条件が成立しない場合に前記区間における当該条件分岐命令以降の命令を続行し、他の実行経路への分岐条件が成立する場合に前記区間における当該条件分岐以降の命令の実行を中止するコードを当該条件分岐命令に相当するコードとして生成する第二コード列生成ステップと、
前記ソースプログラムの前記区間に後続する部分の命令群を基に、その命令群に相当する第三コード列を生成する第三コード列生成ステップと、
前記第一コード列と、前記第二コード列とを、前記コンピュータに並列実行させるように、かつ、前記第二コード列において他の実行経路への分岐条件が成立しない場合には前記第二コード列に継続して前記第三コード列を実行させ、前記第二コード列において他の実行経路への分岐条件が成立する場合には第一コード列に継続して第三コード列を実行させるように編成した目的プログラムを生成する目的プログラム生成ステップとを備える
ことを特徴とするプログラム変換方法。
【請求項１６】
前記目的プログラム生成ステップでは、
前記コンピュータにおいて前記第一コード列の終了が前記第二コード列の終了よりも早い場合には、前記第二コード列を実行している前記コンピュータのプロセッサエレメントに第二コード列の実行を停止させるコードを前記第一コード列の後に含んで編成された目

的プログラムを生成する
ことを特徴とする請求項１５記載のプログラム変換方法。
【請求項１７】
前記プログラム変換方法は更に、
前記ソースプログラムを実行形式に変換した実行プログラムをコンピュータに実行させることで、前記区間において実行された頻度が高い順に、当該頻度が第一位の実行経路を当該コンピュータから取得する実行経路取得ステップを備え、
前記実行経路指定ステップは、前記実行経路取得ステップにおいて取得された前記第一位の実行経路を指定する
ことを特徴とする請求項１５記載のプログラム変換方法。
【請求項１８】
前記プログラム変換方法は更に、
前記コンピュータが並列実行可能な命令数ｍを取得する命令上限取得ステップを備え、
前記実行経路取得ステップは更に、
前記区間における実行頻度が第２位以下の実行経路を取得し、
前記実行経路指定ステップは、
前記実行経路を前記命令数ｍに基づき、前記実行経路取得ステップにおいて取得された、第一位から第ｎ（ｎ＝ｍ－１）位までの実行経路を指定し、
前記第二コード列生成ステップは、前記実行経路指定ステップにおいて指定された第一位から第ｎ位までの実行経路を、実行経路ごとに合計ｎ個のコード列に変換し、
前記目的プログラム生成ステップは、
前記第一コード列と前記第二コード列生成ステップにおいて生成された前記ｎ個のコード列を並列実行させるようにコードを編成した目的プログラムを生成する
ことを特徴とする請求項１７記載のプログラム変換方法。
【請求項１９】
前記第二コード列生成ステップは更に、
前記第二コード列生成ステップにより生成された第一位から第ｎ位のｎ個のコードのうち、他の実行経路への条件分岐が発生しなかったコード列以外のコード列を停止させる停止コードを含んで生成する
ことを特徴とする請求項１８記載のプログラム変換方法。
【請求項２０】
前記プログラム変換方法は更に、
前記コンピュータのメモリの形態が、前記コンピュータの全てのプロセッサエレメントが一つのメモリを共有するメモリ共有型であるか、前記コンピュータの全てのプロセッサエレメントが固有のメモリを有するメモリ分散型かであるかのいずれの形態のメモリを使用しているかの情報を取得するメモリ情報取得ステップを備え、
前記目的プログラム生成ステップは、前記メモリ情報取得ステップにおいて取得されたメモリ情報に基づき、メモリ共有型である場合に、前記第一コード列と前記第二コード列において利用される前記ソースプログラム中の元となる変数がそれぞれ独立した変数として扱うコードになっている目的プログラムを生成する
ことを特徴とする請求項１５記載のプログラム変換方法。
【請求項２１】
前記目的プログラム生成ステップは、
前記停止コードによって停止させられたスレッドをプロセッサエレメントが消去せずに保持しておくコードを含んで目的プログラムを生成する
ことを特徴とする請求項１９記載のプログラム変換方法。
【請求項２２】
前記プログラム変換方法は更に、
前記目的プログラムを前記コンピュータに適合するように機械語に変換する機械語変換ステップを備える

ことを特徴とする請求項１５記載のプログラム変換方法。
【請求項２３】
プログラム変換実行装置が実行する、条件分岐を含むソースプログラムを実行形式である実行形式プログラムに変換して、かつ、２以上の命令を並列して実行できるプログラム変換実行方法であって、
前記ソースプログラムにおいて、条件分岐を跨ぐ一区間についての複数の実行経路のうちの一つの実行経路を指定する実行経路指定ステップと、
前記区間にある全ての条件分岐を含んだ命令群を基に、その命令群に相当する第一コード列を生成する第一コード列生成ステップと、
前記第一コード列を含む第一プログラムを実行する実行ステップと、
前記実行ステップが前記第一プログラムを実行することにより得られた前記区間における実行経路のうち実行頻度が高い順に、当該実行頻度が第一位の実行経路を取得する取得ステップと、
前記取得ステップによって取得された実行経路を前記実行経路指定ステップによって指定し、当該実行経路上の命令群に相当する第二コード列を生成し、当該生成において条件分岐命令については、他の実行経路への分岐条件が成立しない場合に前記区間における当該条件分岐命令以降の命令を続行し、他の実行経路への分岐条件が成立する場合に前記区間における当該条件分岐以降の命令の実行を中止するコードを当該条件分岐命令に相当するコードとして生成する第二コード列生成ステップと、
前記ソースプログラムの前記区間に後続する部分の命令群を基に、その命令群に相当する第三コード列を生成する第三コード列生成ステップと、
前記第一コード列と、前記第二コード列とを、並列実行するように、かつ、前記第二コード列において他の実行経路への条件分岐が発生しない場合には前記第二コード列に継続して前記第三コード列を実行し、前記第二コード列において他の実行経路への条件分岐が発生する場合には前記第一コード列に継続して前記第三コード列を実行するように編成した目的プログラムを生成する目的プログラム生成ステップとを備え、
前記実行ステップは前記第一プログラムを実行する代わりに前記目的プログラムを実行する
ことを特徴とするプログラム変換実行方法。
【請求項２４】
前記目的プログラム生成ステップは、
前記コンピュータにおいて前記第一コード列の終了が前記第二コード列の終了よりも早い場合には、前記第二コード列を実行している前記コンピュータのプロセッサエレメントに第二コード列の実行を停止させるコードを前記第一コードの後に含んで編成されたプログラムを生成する
ことを特徴とする請求項２３記載のプログラム変換実行方法。
【請求項２５】
前記プログラム変換実行方法は更に、
当該プログラム変換実行方法が並列実行可能な命令数ｍを取得する命令上限取得ステップを備え、
前記実行経路取得ステップは更に、
前記区間における実行頻度が第２位以下の実行経路を取得し、
前記実行経路指定ステップは、
前記実行経路を前記命令数ｍに基づき、前記実行経路取得ステップにおいて取得された、第一位から第ｎ（ｎ＝ｍ－１）位までの実行経路を指定し、
前記第二コード生成ステップは、前記実行経路指定ステップにおいて指定された第一位から第ｎ位までの実行経路を、実行経路ごとに合計ｎ個のコード列に変換し、
前記目的プログラム生成ステップは、
前記第一コード列と前記第二コード列生成ステップにおいて生成された前記ｎ個のコード列を並列実行させるようにコードを編成した目的プログラムを生成する

ことを特徴とする請求項２４記載のプログラム変換実行方法。
【請求項２６】
前記第二コード列生成ステップは更に、
前記第二コード列生成ステップにより生成された第一位から第ｎ位のｎ個のコード列のうち、他の実行経路への条件分岐が発生しなかったコード列以外のコード列を停止させる停止コードを含んで生成する
ことを特徴とする請求項２５記載のプログラム変換実行方法。
【請求項２７】
前記目的プログラム生成ステップは、
自機のメモリの形態が、全てのプロセッサエレメントが一つのメモリを共有するメモリ共有型である場合に、前記第一コード列と前記第二コード列において利用される前記ソースプログラム中の元となる変数がそれぞれ独立した変数として扱うコードになっている目的プログラムを生成する
ことを特徴とする請求項２３記載のプログラム変換実行方法。
【請求項２８】
前記目的プログラム生成ステップは、
前記停止コードによって停止させられたスレッドをプロセッサエレメントが消去せずに保持しておくコードを含んで目的プログラムを生成する
ことを特徴とする請求項２７記載のプログラム変換実行方法。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、コンパイラによるプログラムの最適化に関し、特に、プログラム内の実行経路とその実行頻度に着目した最適化に関する。
【背景技術】
【０００２】
ソースプログラムを実行形式に変換するコンパイラ装置において、コンパイラ装置によって生成された実行プログラムがターゲットのハードウェア上において実行されて、その実行結果がより早く出るように種々の工夫がなされてきた。
コンパイラ装置によって生成された実行プログラムの結果が速く出るようにコンパイラ装置は命令のスケジューリングを行うが、そのスケジューリング方法の一つにプログラム中の命令を並べ替えて命令の並列度を上げて実行速度を向上させる広域スケジューリング法があり、その広域スケジューリング法の一つにトレーススケジューリング法というものがある。
【０００３】
プログラムにおいて条件分岐を含まずに連続的に処理される命令郡を基本ブロックと呼び、従来においてもこの基本ブロック内における命令の実行順序の変更を行って、命令の並列度を上げて、実行プログラムの実行時間が短縮されてきた。なお、基本ブロックの最後には条件分岐命令を含んでも良い。
トレーススケジューリング法は、この基本ブロックを拡張するべく、条件分岐命令を跨いで一つの基本ブロックから条件分岐命令によって派生する複数の基本ブロックの一つを当該条件分岐命令が存在しないかのように結合、拡張し、拡張された基本ブロック内においてその命令を並べ替えるスケジューリング方法である。元の基本ブロックを拡張した形になるので命令のスケジューリングの自由度が上がり、更にプログラムの実行時間は短縮されることになる。ただし、拡張した基本ブロックの実行経路がプログラムにおいて実際に実行されない場合に備え、値の整合性を保つべく保証用のコードが必要となる。プログラムにおいて、この基本ブロックによる拡張が行われ最適化が施された実行経路を通る場合には実行結果はソースプログラムをそのままコーディングした実行プログラムよりも早く実行結果を得ることができる。その技術を利用した技術が特許文献１に記されている。
なお、基本ブロックの拡張は、基本的にプログラムにおいて実行頻度の高い実行経路上の

基本ブロックに対して適用される。

【０００４】

例えば図２０（ａ）にあるように、元のソースプログラムの一部分が同図のように分岐するプログラムがあったとする。この図２０（ａ）のフローグラフにおいて、基本ブロックＡ２００１、Ｂ２００２、Ｃ２００３を通る実行経路の実行頻度が高いものとする。それに基づき、このプログラムにトレーススケジューリングを適用すると例えば図２０（ｂ）のように、例えば、基本ブロックＡ２００１と基本ブロックＢ２００２を入れ替えることで実行速度が早くなり、基本ブロックＢ２０１２、Ａ２０１１、Ｃ２０１３を含む実行経路２０１０をこのプログラムが通る場合にはプログラム全体の実行時間は短縮される。

【特許文献１】特開平１１－９６００５号公報

【発明の開示】

【発明が解決しようとする課題】

【０００５】

ところで前述したようにトレーススケジューリング法においては基本ブロック内の命令順序を入れ替えたりするので、このフローグラフにおいて他の実行経路を通った場合に実行結果の整合をとる為に保証コードを生成しなければならなくなる。

例えば、図２０（ｂ）においては、ブロックＡ’２０１８がそれに相当する。図２０（ｂ）では、ブロックＢ２０１２から、図２０（ａ）と同じようにそのままブロックＤ２００４に分岐させるとブロックＡ２００１の演算がなされていないことになるので、ブロックＡ２００１の命令に相当する保証コードとしてブロックＡ’２０１８をつけて、図２０（ｂ）においてブロックＢ２０１２、Ａ’２０１８、Ｄ２０１４、Ｅ２０１５を通る経路が実行経路である場合の値の整合性を保つ。

【０００６】

本発明においては、上記のような保証コードを用いずに、特定の実行経路において基本ブロックの拡張を行って最適化を施したプログラムを生成するプログラム変換装置であるところのコンパイラ装置を提供することを目的とする。

【課題を解決するための手段】

【０００７】

上記課題を解決するため、本発明に係るプログラム変換装置であるところのコンパイラ装置は、条件分岐を含むソースプログラムを変換して、２以上の命令を並列して実行できるコンピュータを対象とする目的プログラムを生成するプログラム変換装置であって、前記ソースプログラムにおいて、条件分岐を跨ぐ一区間についての複数の実行経路のうちの一つの実行経路を指定する実行経路指定手段と、前記区間にある全ての命令群を基に、その命令群に相当する第一コード列を生成する第一コード列生成手段と、前記実行経路指定手段によって指定される実行経路上の命令群だけに相当する第二コード列を生成し、当該生成において条件分岐命令については、他の実行経路への分岐条件が成立しない場合に前記区間における当該条件分岐命令以降の命令を続行し、他の実行経路への分岐条件が成立する場合に前記区間における当該条件分岐以降の命令の実行を中止するコードを当該条件分岐命令に相当するコードとして生成する第二コード列生成手段と、前記ソースプログラムの前記区間に後続する部分の命令群を基に、その命令群に相当する第三コード列を生成する第三コード列生成手段と、前記第一コード列と、前記第二コード列とを、前記コンピュータに並列実行させるように、かつ、前記第二コード列において他の実行経路への分岐条件が成立しない場合には前記第二コード列に継続して前記第三コード列を実行させ、前記第二コード列において他の実行経路への分岐条件が成立する場合には第一コード列に継続して第三コード列を実行させるように編成した目的プログラムを生成する目的プログラム生成手段とを備えることを特徴とする。

【０００８】

ここで、相当するとは、ソースプログラムの命令の内容と略同一の内容であることを言う。但し、前記コンピュータのメモリの形態などによって、アクセスするレジスタなどはさまざまに変化する。また、実行経路とは、連続的に実行される命令群のことであり、プ

ログラム中で条件分岐命令の条件によって実行される命令が枝分かれして変化する場合には、その枝の一つのみを指す。また、生成される目的プログラムは、中間コードの場合もあるし、前記コンピュータで実行できるような実行形式プログラムである場合もある。中間コードとはソースプログラムから実行プログラムに変換する際に、コンパイラ装置が扱いやすいように生成されるコードで、その内容はソースプログラムに相当する。

【発明の効果】

【０００９】

これにより、本発明に係るコンパイラ装置によって生成される実行プログラムは、元のソースプログラムを前記コンピュータが実行できるようにそのまま最適化を施さずに変換した前記第一コード列を含むプログラムを前記コンピュータが有する第一プロセッサエレメントに実行させ、特定の実行経路、即ち、実行経路指定手段によって指定された実行経路に関しては最適化を施して前記コンピュータが有する第二プロセッサエレメントに実行させるので、特定の実行経路以外の経路を通る際に値の整合性を保つために必要とされる上記のような保証コードを用いずに、特定の実行経路に関して最適化を施したプログラムを生成することができる。

【００１０】

また、前記区間において実行経路が前記実行経路指定手段によって指定された実行経路を通る場合に第二コードの終了が早くなり、それに合わせて第三コードの実行開始が早くなり、結果、プログラム全体の実行時間は短縮される。

また、第一プロセッサエレメントが元のソースプログラムに相当するプログラムを実行するので値の整合性もとれる。

【００１１】

また、前記プログラム変換装置において、前記目的プログラム生成手段は、前記コンピュータにおいて前記第一コード列の終了が前記第二コード列の終了よりも早い場合には、前記第二コード列を実行している前記コンピュータのプロセッサエレメントに第二コード列の実行を停止させるコードを前記第一コード列の後に含んで編成された目的プログラムを生成することとしてよい。

【００１２】

これにより、第一コードの方が実行結果が早く出た場合に、第二コードを実行しているプロセッサエレメントに、第二コードを停止させ、その後、当該プロセッサエレメントに別の処理を割り振るプログラム編成にすれば、資源の有効活用を行える。

また、前記プログラム変換装置は更に、前記ソースプログラムを実行形式に変換した実行プログラムをコンピュータに実行させることで、前記区間において実行された頻度が高い順に、当該頻度が第一位の実行経路を当該コンピュータから取得する実行経路取得手段を備え、前記実行経路指定手段は、前記取得手段により取得された前記第一位の実行経路を指定することとしてよい。

【００１３】

これにより、これによりコンパイラ装置は、実行頻度の高い実行経路を最適化することができるので、この実行頻度の高い実行経路をプログラムが通る場合に、このコンパイラ装置によって生成されるプログラムの結果が出るタイミングが早くなる。

また、前記プログラム変換装置は更に、前記コンピュータが並列実行可能な命令数ｍを取得する命令上限取得手段を備え、前記実行経路取得手段は更に、前記区間における実行頻度が第２位以下の実行経路を取得し、前記実行経路指定手段は、前記実行経路を前記命令数ｍに基づき、前記実行経路取得手段によって取得された、第一位から第ｎ（ｎ＝ｍ－１）位までの実行経路を指定し、前記第二コード列生成手段は、前記実行経路指定手段によって指定された第一位から第ｎ位までの実行経路を、実行経路ごとに合計ｎ個のコード列に変換し、前記目的プログラム生成手段は、前記第一コード列と前記第二コード列生成手段により生成された前記ｎ個のコード列を並列実行させるようにコードを編成した目的プログラムを生成することとしてよい。

【００１４】

これにより複数の実行頻度の高い実行経路をそれぞれ別のスレッドとして立ち上げて実行することができ、プログラム全体のターゲットハードウェア上における実行時間を短縮できるプログラムを生成できる。

また、前記プログラム変換装置において前記第二コード列生成手段は更に、前記第二コード列生成手段により生成された第一位から第ｎ位のｎ個のコードのうち、他の実行経路への条件分岐が発生しなかったコード列以外のコード列を停止させる停止コードを含んで生成することとしてよい。

【００１５】

これにより、他のプロセッサエレメントで実行しているスレッドを停止することができ、自スレッドが、処理を終えていて条件分岐が自スレッドの実行経路を通った場合に、そのことを他のプロセッサエレメントに知らせ、他のプロセッサエレメントで実行されているスレッドを停止することができるプログラムを生成できる。

また、前記プログラム変換装置は更に、前記コンピュータのメモリの形態が、前記コンピュータの全てのプロセッサエレメントが一つのメモリを共有するメモリ共有型であるか、前記コンピュータの全てのプロセッサエレメントが固有のメモリを有するメモリ分散型かであるかのいずれの形態のメモリを使用しているかの情報を取得するメモリ情報取得手段を備え、前記目的プログラム生成手段は、前記メモリ情報取得手段により取得したメモリ情報に基づき、メモリ共有型である場合に、前記第一コード列と前記第二コード列において利用される前記ソースプログラム中の元となる変数がそれぞれ独立した変数として扱うコードになっている目的プログラムを生成することとしてよい。

【００１６】

ここで独立した変数として扱うとは、第一コード列と第二コード列でソースプログラム中の同一の変数を参照する場合に、その値を異なるレジスタに格納して演算を行うことをいう。

これにより、メモリ共有型のコンピュータにおいてプログラムの演算結果を保証することができるようになる。

【００１７】

また、前記プログラム変換装置において、前記目的プログラム生成手段は、前記停止コードによって停止させられたスレッドをプロセッサエレメントが消去せずに保持しておくコードを含んで目的プログラムを生成することとしてもよい。

これにより、生成されるスレッドが扱う演算データのみが異なる場合に、スレッドを保持し残しているので、演算に必要なデータのみをプロセッサエレメントに渡せばよく、逐次プロセッサエレメントにスレッドの内容と扱うデータの両方を渡すという非効率性を省け、また、目的プログラムの実行時間の短縮にもつながる。

【００１８】

また、前記プログラム変換装置は更に、前記目的プログラムを前記コンピュータに適合するように機械語に変換する機械語変換手段を備えることとしてよい。

これにより、目的プログラムが中間コードであった場合に対象とするコンピュータの機械語に合わせた実行プログラムを生成できる。

また、条件分岐を含むソースプログラムを実行形式である実行形式プログラムに変換して、かつ、２以上の命令を並列して実行できるプログラム変換実行装置であって、前記ソースプログラムにおいて、条件分岐を跨ぐ一区間についての複数の実行経路のうちの一つの実行経路を指定する実行経路指定手段と、前記区間にある全ての条件分岐を含んだ命令群を基に、その命令群に相当する第一コード列を生成する第一コード列生成手段と、前記第一コード列を含む第一プログラムを実行する実行手段と、前記実行手段が前記第一プログラムを実行することにより得られた前記区間における実行経路のうち実行頻度が高い順に、当該実行頻度が第一位の実行経路を取得する取得手段と、前記取得手段によって取得された実行経路を前記実行経路指定手段によって指定し、当該実行経路上の命令群に相当する第二コード列を生成し、当該生成において条件分岐命令については、他の実行経路への分岐条件が成立しない場合に前記区間における当該条件分岐命令以降の命令を続行し、

他の実行経路への分岐条件が成立する場合に前記区間における当該条件分岐以降の命令の実行を中止するコードを当該条件分岐命令に相当するコードとして生成する第二コード列生成手段と、前記ソースプログラムの前記区間に後続する部分の命令群を基に、その命令群に相当する第三コード列を生成する第三コード列生成手段と、前記第一コード列と、前記第二コード列とを、並列実行するように、かつ、前記第二コード列において他の実行経路への条件分岐が発生しない場合には前記第二コード列に継続して前記第三コード列を実行し、前記第二コード列において他の実行経路への条件分岐が発生する場合には前記第一コード列に継続して前記第三コード列を実行するように編成した目的プログラムを生成する目的プログラム生成手段とを備え、前記実行手段は前記第一プログラムを実行する代わりに前記目的プログラムを実行することとしてよい。

【００１９】

これにより、プログラムを生成しながら実行できるプログラム変換実行装置において、実行頻度の高い実行経路を通る場合に、プログラムの実行時間が短縮される。

また、従来において、保証コードはフローグラフが複雑になるほどに、保証コードの内容も複雑化する。プログラムを逐次解釈実行するインタプリタにおいて部分的なコードの実行性能を上げるために所謂ジャストインタイムコンパイル、つまり動的コンパイル技術が用いられるコンパイラ装置においては、この保証コードの生成は時間のロスになることがあるが、本発明においては保証コードを生成しないので、そういった問題もなくなる。

【００２０】

また、前記プログラム変換実行装置において、前記目的プログラム生成手段は、前記コンピュータにおいて前記第一コード列の終了が前記第二コード列の終了よりも早い場合には、前記第二コード列を実行している前記コンピュータのプロセッサエレメントに第二コード列の実行を停止させるコードを前記第一コードの後に含んで編成されたプログラムを生成することとしてよい。

【００２１】

これにより、第一コードの方が実行結果が早く出た場合に、第二コードを実行しているプロセッサエレメントに、第二コードを停止させ、その後、別の処理を割り振ってやれば、資源の有効活用になる。

また、前記プログラム変換実行装置は更に、当該プログラム変換実行装置が並列実行可能な命令数ｍを取得する命令上限取得手段を備え、前記実行経路取得手段は更に、前記区間における実行頻度が第２位以下の実行経路を取得し、前記実行経路指定手段は、前記実行経路を前記命令数ｍに基づき、前記実行経路取得手段によって取得された、第一位から第ｎ（ｎ＝ｍ－１）位までの実行経路を指定し、前記第二コード生成手段は、前記実行経路指定手段によって指定された第一位から第ｎ位までの実行経路を、実行経路ごとに合計ｎ個のコード列に変換し、前記目的プログラム生成手段は、前記第一コード列と前記第二コード列生成手段により生成された前記ｎ個のコード列を並列実行させるようにコードを編成した目的プログラムを生成することとしてよい。

【００２２】

これにより、複数の実行頻度の高い実行経路をそれぞれ別のスレッドとして立ち上げて実行することができ、プログラム全体のターゲットハードウェア上における実行時間を短縮できる。

また、前記プログラム変換実行装置において、前記第二コード列生成手段は更に、前記第二コード列生成手段により生成された第一位から第ｎ位のｎ個のコード列のうち、他の実行経路への条件分岐が発生しなかったコード列以外のコード列を停止させる停止コードを含んで生成することとしてよい。

【００２３】

これにより、自スレッドが実行される条件が成立している場合に、他のプロセッサエレメントが実行している他スレッドを停止させ、その後のプログラム上の処理を行うことで資源の有効活用が可能となる。

また、前記プログラム変換実行装置において前記目的プログラム生成手段は、自機のメ

モリの形態が、全てのプロセッサエレメントが一つのメモリを共有するメモリ共有型である場合に、前記第一コード列と前記第二コード列において利用される前記ソースプログラム中の元となる変数がそれぞれ独立した変数として扱うコードになっている目的プログラムを生成することとしてよい。

【００２４】

これにより、このプログラム変換実行装置が、メモリ共有型であっても、メモリ分散型であっても、それに対応してプログラム中の値を格納するレジスタ割り振りを考慮したプログラムを生成できる。

また、前記プログラム変換実行装置において、前記目的プログラム生成手段は、前記停止コードによって停止させられたスレッドをプロセッサエレメントが消去せずに保持しておくコードを含んで目的プログラムを生成することとしてもよい。

【００２５】

これにより、生成されるスレッドが扱う演算データのみが異なる場合に、スレッドを保持し残しているので、演算に必要なデータのみをプロセッサエレメントに渡せばよく、逐次プロセッサエレメントにスレッドの内容と扱うデータの両方を渡すという非効率性を省け、また、目的プログラムの実行時間の短縮にもつながる。

また、条件分岐を含むソースプログラムを変換して、２以上の命令を並列して実行できるコンピュータを対象とする目的プログラムを生成するプログラム変換方法であって、前記ソースプログラムにおいて、条件分岐を跨ぐ一区間についての複数の実行経路のうちの一つの実行経路を指定する実行経路指定ステップと、前記区間にある全ての命令群を基に、その命令群に相当する第一コード列を生成する第一コード列生成ステップと、前記実行経路指定ステップにおいて指定される実行経路上の命令群だけに相当する第二コード列を生成し、当該生成において条件分岐命令については、他の実行経路への分岐条件が成立しない場合に前記区間における当該条件分岐命令以降の命令を続行し、他の実行経路への分岐条件が成立する場合に前記区間における当該条件分岐以降の命令の実行を中止するコードを当該条件分岐命令に相当するコードとして生成する第二コード列生成ステップと、前記ソースプログラムの前記区間に後続する部分の命令群を基に、その命令群に相当する第三コード列を生成する第三コード列生成ステップと、前記第一コード列と、前記第二コード列とを、前記コンピュータに並列実行させるように、かつ、前記第二コード列において他の実行経路への分岐条件が成立しない場合には前記第二コード列に継続して前記第三コード列を実行させ、前記第二コード列において他の実行経路への分岐条件が成立する場合には第一コード列に継続して第三コード列を実行させるように編成した目的プログラムを生成する目的プログラム生成ステップとを備えることとしてもよい。

【００２６】

この方法により、前記第一コードと特定の実行経路に関して最適化が施された前記第二コードを並列実行させることができる目的プログラムを生成することができる。

また、前記プログラム生成方法において、前記目的プログラム生成ステップでは、前記コンピュータにおいて前記第一コード列の終了が前記第二コード列の終了よりも早い場合には、前記第二コード列を実行している前記コンピュータのプロセッサエレメントに第二コード列の実行を停止させるコードを前記第一コード列の後に含んで編成された目的プログラムを生成することとしてもよい。

【００２７】

この方法より、第一コードの方の実行結果が早く出た場合に第二コードを実行しているプロセッサエレメントに、第二コードを停止させる目的プログラムを生成することができる。

また、前記プログラム変換方法は更に、前記ソースプログラムを実行形式に変換した実行プログラムをコンピュータに実行させることで、前記区間において実行された頻度が高い順に、当該頻度が第一位の実行経路を当該コンピュータから取得する実行経路取得ステップを備え、前記実行経路指定ステップは、前記取得手段により取得された前記第一位の実行経路を指定することとしてもよい。

【００２８】
この方法により、実行頻度の最も高い実行経路を最適化して、この実行頻度の高い実行経路の内容を並列実行させる目的プログラムを生成することができる。
また、前記プログラム変換方法は更に、前記コンピュータが並列実行可能な命令数ｍを取得する命令上限取得ステップを備え、前記実行経路取得ステップは更に、前記区間における実行頻度が第２位以下の実行経路を取得し、前記実行経路指定ステップは、前記実行経路を前記命令数ｍに基づき、前記実行経路取得手段によって取得された、第一位から第ｎ（ｎ＝ｍ－１）位までの実行経路を指定し、前記第二コード列生成ステップは、前記実行経路指定手段によって指定された第一位から第ｎ位までの実行経路を、実行経路ごとに合計ｎ個のコード列に変換し、前記目的プログラム生成ステップは、前記第一コード列と前記第二コード列生成手段により生成された前記ｎ個のコード列を並列実行させるようにコードを編成した目的プログラムを生成することとしてもよい。
【００２９】
この方法により、複数の実行頻度の高い実行経路に関して最適化し、この複数の実行頻度の高い実行経路の内容を並列実行させる目的プログラムを生成することができる。
また、前記プログラム変換方法において、前記第二コード列生成ステップは更に、前記第二コード列生成ステップにより生成された第一位から第ｎ位のｎ個のコードのうち、他の実行経路への条件分岐が発生しなかったコード列以外のコード列を停止させる停止コードを含んで生成することとしてもよい。
【００３０】
この方法により、自スレッドが、処理を終えていて条件分岐により自スレッドの実行経路を通った場合に、他のプロセッサエレメントで実行されているスレッドを停止することができるプログラムを生成することができる。
また、前記プログラム変換方法は更に、前記コンピュータのメモリの形態が、前記コンピュータの全てのプロセッサエレメントが一つのメモリを共有するメモリ共有型であるか、前記コンピュータの全てのプロセッサエレメントが固有のメモリを有するメモリ分散型かであるかのいずれの形態のメモリを使用しているかの情報を取得するメモリ情報取得ステップを備え、前記目的プログラム生成手段は、前記メモリ情報取得手段により取得したメモリ情報に基づき、メモリ共有型である場合に、前記第一コード列と前記第二コード列において利用される前記ソースプログラム中の元となる変数がそれぞれ独立した変数として扱うコードになっている目的プログラムを生成することとしてもよい。
【００３１】
この方法により、メモリ共有型のコンピュータにおいて演算結果を保証できる目的プログラムを生成できるようになる。
また、前記プログラム変換方法において、前記目的プログラム生成ステップは、前記停止コードによって停止させられたスレッドをプロセッサエレメントが消去せずに保持しておくコードを含んで目的プログラムを生成することとしてもよい。
【００３２】
この方法により、スレッドを消去せず、保持しておき再利用することが可能な目的プログラムを生成することができる。
また、前記プログラム変換方法は更に、前記目的プログラムを前記コンピュータに適合するように機械語に変換する機械語変換ステップを備えることとしてもよい。
この方法により、目的プログラムが中間コードであった場合に、対象とするコンピュータの機械語にあわせた実行プログラムを生成することができる。
【００３３】
また、条件分岐を含むソースプログラムを実行形式である実行形式プログラムに変換して、かつ、２以上の命令を並列して実行できるプログラム変換実行方法であって、前記ソースプログラムにおいて、条件分岐を跨ぐ一区間についての複数の実行経路のうちの一つの実行経路を指定する実行経路指定ステップと、前記区間にある全ての条件分岐を含んだ命令群を基に、その命令群に相当する第一コード列を生成する第一コード列生成ステップ

と、前記第一コード列を含む第一プログラムを実行する実行ステップと、前記実行ステップが前記第一プログラムを実行することにより得られた前記区間における実行経路のうち実行頻度が高い順に、当該実行頻度が第一位の実行経路を取得する取得ステップと、前記取得ステップによって取得された実行経路を前記実行経路指定ステップによって指定し、当該実行経路上の命令群に相当する第二コード列を生成し、当該生成において条件分岐命令については、他の実行経路への分岐条件が成立しない場合に前記区間における当該条件分岐命令以降の命令を続行し、他の実行経路への分岐条件が成立する場合に前記区間における当該条件分岐以降の命令の実行を中止するコードを当該条件分岐命令に相当するコードとして生成する第二コード列生成ステップと、前記ソースプログラムの前記区間に後続する部分の命令群を基に、その命令群に相当する第三コード列を生成する第三コード列生成ステップと、前記第一コード列と、前記第二コード列とを、並列実行するように、かつ、前記第二コード列において他の実行経路への条件分岐が発生しない場合には前記第二コード列に継続して前記第三コード列を実行し、前記第二コード列において他の実行経路への条件分岐が発生する場合には前記第一コード列に継続して前記第三コード列を実行するように編成した目的プログラムを生成する目的プログラム生成ステップとを備え、前記実行手段は前記第一プログラムを実行する代わりに前記目的プログラムを実行することとしてもよい。

【００３４】

この方法により、プログラムを生成しながら、実行頻度の高い実行経路の内容を並列実行させながら実行できる目的プログラムを生成することができる。

また、前記目的プログラム生成ステップは、前記コンピュータにおいて前記第一コード列の終了が前記第二コード列の終了よりも早い場合には、前記第二コード列を実行している前記コンピュータのプロセッサエレメントに第二コード列の実行を停止させるコードを前記第一コードの後に含んで編成されたプログラムを生成することとしてもよい。

【００３５】

この方法により、第一コードの実行結果が早く出た場合に、第二コードを実行しているプロセッサエレメントに第二コードを停止させるプログラムを生成できる。

また、前記プログラム変換実行方法は更に、当該プログラム変換実行方法が並列実行可能な命令数ｍを取得する命令上限取得ステップを備え、前記実行経路取得ステップは更に、

前記区間における実行頻度が第２位以下の実行経路を取得し、前記実行経路指定ステップは、前記実行経路を前記命令数ｍに基づき、前記実行経路取得手段によって取得された、第一位から第ｎ（ｎ＝ｍ－１）位までの実行経路を指定し、前記第二コード生成ステップは、前記実行経路指定手段によって指定された第一位から第ｎ位までの実行経路を、実行経路ごとに合計ｎ個のコード列に変換し、前記目的プログラム生成ステップは、前記第一コード列と前記第二コード列生成手段により生成された前記ｎ個のコード列を並列実行させるようにコードを編成した目的プログラムを生成することとしてもよい。

【００３６】

この方法により、複数の実行頻度の高い実行経路をそれぞれ別のスレッドとして立ち上げて実行する目的プログラムを生成することができる。

また、前記プログラム実行変換方法において、前記第二コード列生成ステップは更に、

前記第二コード列生成ステップにより生成された第一位から第ｎ位のｎ個のコード列のうち、他の実行経路への条件分岐が発生しなかったコード列以外のコード列を停止させる停止コードを含んで生成することとしてもよい。

【００３７】

この方法により、一つのスレッドが実行される条件が成立している場合に、他のスレッドを停止する目的プログラムを生成することができる。

また、前記プログラム実行変換方法において、前記目的プログラム生成ステップは、自機のメモリの形態が、全てのプロセッサエレメントが一つのメモリを共有するメモリ共有型である場合に、前記第一コード列と前記第二コード列において利用される前記ソースプ

ログラム中の元となる変数がそれぞれ独立した変数として扱うコードになっている目的プログラムを生成することとしてもよい。

【００３８】

この方法により、メモリ共有型、メモリ分散型に対応した目的プログラムを生成することができる。

また、前記プログラム変換実行変換方法であって、前記目的プログラム生成ステップは、前記停止コードによって停止させられたスレッドをプロセッサエレメントが消去せずに保持しておくコードを含んで目的プログラムを生成することとしてもよい。

【００３９】

この方法により、スレッドを消去せず、保持しておき再利用することが可能な目的プログラムを生成することができる。

【発明を実施するための最良の形態】

【００４０】

以下、本発明に係るプログラム変換装置であるところのコンパイラ装置の実施の形態を図面を用いながら説明していく。

＜第一の実施形態＞

第一の実施形態におけるコンパイラ装置は、メモリ共有型のコンピュータを対象とする実行プログラムを生成する。

＜概要＞

本発明の概要を図２および図３を用いて説明する。

【００４１】

本発明に係るコンパイラ装置において、ソースプログラムの一部分が図２におけるフローグラフのような分岐の形態を持つソースプログラムであったとし、これを本発明のコンパイラ装置によって実行形式に変換するとする。

なお、ブロックＩ２００、Ｊ２０２、Ｋ２０３、Ｌ２０６、Ｑ２０４、Ｓ２０５、Ｔ２０８、Ｕ２０７、Ｘ２０１はそれぞれ基本ブロックである。基本ブロックはその途中に分岐を含まない命令列のことである。但し、基本ブロックの最後には分岐があっても良い。また、このコンパイラ装置によって生成される実行プログラムは、２以上の処理を同時に実行できるコンピュータを対象としている。

【００４２】

図２のフローグラフにおいて実行経路は、ブロックＩ２００→Ｊ２０２→Ｑ２０４を通る実行経路、ブロックＩ２００→Ｊ２０２→Ｋ２０３→Ｓ２０５→Ｔ２０８を通る実行経路、ブロックＩ２００→Ｘ２０１を通る実行経路、ブロックＩ２００→Ｊ２０２→Ｋ２０３→Ｓ２０５→Ｕ２０７を通る実行経路、ブロックＩ２００→Ｊ２０２→Ｋ２０３→Ｌ２０６を通る実行経路、の５つがあり、それぞれこの順で実行頻度が高いものとする。

【００４３】

そこで、このうち実行頻度の高いものを連続的に実行される命令群にして、実行形式プログラムを生成し、それぞれと、元のソースプログラムをそのまま実行形式に変換したものとを、別々のプロセッサエレメントに並列実行させるプログラムを作成する。つまり、図３のように、まず、ソースプログラムをそのまま実行コードにしたスレッド３００の実行を第一のプロセッサエレメントに実行させ、以下順に実行頻度一位の実行経路のスレッド３０１を第二のプロセッサエレメントが、実行頻度第二位の実行経路のスレッド３０２を第三のプロセッサエレメントが実行し、以下プロセッサエレメントの数の許す限り、また生成可能なスレッドの数の許す限りスレッドを立ち上げ、各プロセッサに実行させるような編成のプログラムを生成する。また、各スレッドにおいて、そのスレッドが成立する条件が整った場合には、自スレッド以外のスレッドを停止させ、自身のスレッドの演算結果によって得られた値を反映させるコミット処理を行う。

【００４４】

これにより保証コードを必要とせず、並列実行しているスレッドの中には元となるプログラムをそのまま実行形式に移したスレッド３００も実行されるので値の整合性について

の問題ない。また、生成された実行経路３０１～３０３のいずれかを通って本プログラムが実行される場合には、実行結果のでるタイミングはスレッド３００だけを実行するよりも早くなりプログラム全体の実行時間も短縮できる。

＜構成＞

本発明に係るコンパイラ装置１００の構成を図１のブロック図を用いて説明する。本発明に係るコンパイラ装置１００は、解析部１０１、実行経路指定部１０２、最適化部１０３、コード変換部１０４からなる。

【００４５】

コンパイラ装置１００は、ＭＰＵ（Micro Processing Unit）、ＲＯＭ(Read Only Memory)、ＲＡＭ（Random Access Memory）、ハードディスク装置を含んで実現されるコンピュータシステムである。ハードディスク装置又は、ＲＯＭに搭載されるコンピュータプログラムにより、本コンパイラ装置は目的とする実行プログラムを作成する。また、ＲＡＭを用いて各部間のデータの受け渡しなどが行われる。

【００４６】

解析部１０１は、ソースプログラムの分岐や実行内容を解析し、ソースプログラムに記入されている「分岐」や「繰り返し」などの情報を取得する機能を有し、解析によって得られた情報である解析情報１０５を実行経路指定部１０２に渡す。

実行経路指定部１０２は、解析部１０１からの実行経路の識別子等を含む解析情報１０５と、実行プログラムに変換するソースプログラム上の実行経路の実行頻度に関する情報１４０を取得し、その情報を基に、取得した実行経路のうち実行頻度の高い実行経路を指定し、その内容を最適化部１０３に送信する機能を有する。

【００４７】

最適化部１０３は、基本的に入力されたソースプログラム１１０の命令の実行順序や実行プログラムの生成のための最適化を行う機能を有し、解析部１０１、及び実行経路指定部１０２から取得した情報を基に、実行頻度の高い実行経路から他の実行経路への分岐が発生しないようにして、命令の実行順序に関する最適化を施す機能を有する。

コード変換部１０４は、最適化部１０３によって最適化が施されたコードをそれぞれのプロセッサエレメントに割り振った実際に実行する実行プログラム１２０をターゲットハードウェア１３０に適合するように生成する機能を有する。生成された実行プログラム１２０は、ターゲットハードウェア１３０に渡される。

【００４８】

ターゲットハードウェア１３０に渡された実行プログラム１２０は、ターゲットハードウェア１３０上において実行される。そして実行プログラム１２０を実行することによって実行プログラム１２０の実行経路に関する情報が実行経路の実行頻度の情報１４０として実行経路１０２に送信される。ここで実行経路に関する情報とは、分岐によって派生する複数の実行経路のうち実際にどの経路を通ったのかを示す情報である。なお、ターゲットハードウェア１３０は、複数のプロセッサエレメントを有するので同時に２以上の処理を実行できる。また、ターゲットハードウェア１３０のメモリ形態にはメモリ共有型とメモリ分散型の２通りが考えられ、第一の実施形態においてはメモリ共有型として説明する。

【００４９】

ここで簡単にメモリ共有型とメモリ分散型とについて説明しておく。

メモリ共有型は、図４（ａ）にあるように、複数のプロセッサエレメント４００～４０２が一つのメモリ４０３に接続されている。それぞれのプロセッサエレメント４００～４０２は、メモリ４０３から必要なデータをロードし、それぞれのレジスタに格納して演算を行い、演算後、その結果に基づきメモリ４０３に格納されているデータを更新する形態をとっている。

【００５０】

メモリ分散型は、図４（ｂ）にあるように、複数のプロセッサエレメントそれぞれにメモリが接続されている形態のことでプロセッサエレメント４１０はメモリ４１３に、プロ

セッサエレメント４１１はメモリ４１４に、プロセッサエレメント４１２はメモリ４１５に接続されている。また、各プロセッサエレメントで行われた演算結果は実行されたプログラムに基づき、その値が各メモリに反映されるように設定されている。例えばプロセッサエレメント４１０で演算結果が出たときにはメモリ４１３だけでなく、メモリ４１４及びメモリ４１５に格納されているデータも更新される。

【００５１】

なお、どちらの場合もプロセッサエレメントを３個として図示し説明したが、その数はいくつであっても良い。

＜データ＞

コンパイラ装置１００に入力されるデータには、実行経路の実行頻度の情報１４０と、ターゲットハードウェア１３０のハードウェアの仕様と、ソースプログラム１１０とがある。以下それらのデータに関する説明を行う。

【００５２】

コンパイラ装置１００に入力される実行経路の実行頻度に関する情報１４０は、解析部１０１によって解析され作成された実行経路の識別子と、その実行経路の識別子に対応して、実際にターゲットハードウェア１３０若しくはその他の実行プログラムを実行できるハードウェア上において実行されることでその実行経路が何回実行されたかの回数とで構成された情報である。その回数が最も多いものを実行頻度一位としており、以下順に実行頻度二位、三位・・・となっており、この情報はまずターゲットハードウェア１３０のＲＡＭに記憶され、その後にコンパイラ装置１００に渡され、コンパイラ装置１００のＲＡＭに記憶される。

【００５３】

また、ターゲットハードウェア１３０の仕様の情報に関しては、メモリの形態に関してはメモリ共有型の場合は０、メモリ分散型の場合には１の２値で管理されており、それがメモリ情報としてコンパイラ装置１００にターゲットハードウェア１３０から入力されコンパイラ装置１００のＲＡＭに記憶される。また、同時実行可能な命令数の情報は、ターゲットハードウェア１３０の有するプロセッサエレメントの数に関する情報であり、その情報もコンパイラ装置１００に入力され、ＲＡＭに記憶される。

【００５４】

ソースプログラム１１０は、図５（ａ）に示すように記述されているプログラムである。

本実施形態においては、ソースプログラム１１０の一例として、図５（ａ）に示す部分ソースプログラム５１０をコンパイラ装置が変換するものとして説明する。以下、入力される部分ソースプログラム５１０の内容、及びそれを元に本コンパイラ装置によって作成されるコードの説明を行っていく。

【００５５】

まず図５（ａ）のソースプログラムの内容について説明する。以降の図６～図１０のコードはこのソースプログラムの内容、若しくは内容の一部を実行するために変換されたコードである。

図５（ａ）は、ソースプログラムのある部分の抜粋の一例であり、この区間におけるプログラムはソースプログラムの全体の中において何度も使用されるものとする。この部分ソースプログラム５１０を、フローグラフの形式に書き換えると図５（ｂ）のように表される。この部分ソースプログラム５１０の内容を図５（ｂ）のフローグラフを用いて説明する。

【００５６】

まず、命令ブロック５００においてｘにａとｂの値を加算した値を格納し、分岐ブロック５０５においてｘが０以上であるかどうかを判定する。

命令ブロック５００において得られたｘが０以下（分岐ブロック５０５のｎｏ）ならば、ブロック５０４に進み、ｘに命令ブロック５００において得られたｘの値をマイナス値にして格納する。命令ブロック５００において得られたｘが０以上（分岐ブロック５０５

のｙｅｓ）ならば、命令ブロック５０１に進み、ｙに命令ブロック５００において得られたｘの値からｃを引いた値を格納する。

【００５７】

命令ブロック５０１を実行した後、分岐ブロック５０６においてｙが１０以上であるかどうかを判定する。ｙが１０以上（分岐ブロック５０６のｙｅｓ）ならば、命令ブロック５０２に進み、ｘに命令ブロック５０１で得られたｙから１０引いた値を格納する。ｙが１０以下（分岐ブロック５０６のｎｏ）ならば、命令ブロック５０３に進み、ｘに命令ブロック５００において得られたｘの値に１０足した値を格納する。

【００５８】

以上が部分ソースプログラム５１０の実行内容である。なお、ａ、ｂ、ｃの値は、この部分ソースプログラム５１０の前部分において既に与えられているものとする。また、この部分ソースプログラム５１０中の条件分岐によって派生する３つの実行経路のうち最も実行頻度の高い実行経路が、実行経路５５１であり、その次に実行頻度の高い実行経路が実行経路５５２であるとする。これは予めソースプログラムに最適化を施さずに変換した実行プログラムをターゲットハードウェア１３０上で実行することによって、実行頻度の情報を得ることができる。

【００５９】

図６～図１０に記したコードは、コンパイラ装置１００から出力されるプログラムをアセンブラコードで表記したものであり、図５（ａ）のソースプログラムを元に生成される。図１０のスレッド１０００がメインスレッドで、図７のスレッド７００、図８のスレッド８００、図９のスレッド９００はメインスレッドの中で使用されていて、それぞれのスレッドは、コードには記していないが、ターゲットハードウェア１３０上にて別のプロセッサエレメントで実行される構成になっている。

【００６０】

部分ソースプログラム５１０に最適化を施さずにそのままアセンブラコードに変換すると図６のスレッド６００になる。なお、これらのコードは基本的に上から順に処理されていくものとする。また、各コードの命令の意味に関しては後述する。

スレッド６００の内容を簡単に説明すると、コード６０１、６０９、６１７、６２２、６２７、６３２はラベルコードで、プログラム中の分岐において命令が飛ぶ先の指定に用いられる。

【００６１】

コード６０２～６０８は、図５（ｂ）のフローグラフにおいて、ブロック５００、及びブロック５０５の命令の内容をコード化したものである。

コード６１０～６１６は、図５（ｂ）のフローグラフにおいて、ブロック５０１、及びブロック５０６の命令の内容をコード化したものである。

コード６１８～６２１は、図５（ｂ）のフローグラフにおいて、ブロック５０２の命令の内容をコード化したものである。

【００６２】

コード６２３～６２６は、図５（ｂ）のフローグラフにおいて、ブロック５０３の命令の内容をコード化したものである。

コード６２８～６３１は、図５（ｂ）のフローグラフにおいて、ブロック５０４の命令の内容をコード化したものである。

そして、コード６３３、６３４は、このスレッド６００が終了した際の処理を行うコードである。

【００６３】

本発明に係るコンパイラ装置はスレッド６００以外に、実行頻度の高い実行経路に関して、その実行経路上の命令を実行できるように変換、生成したスレッドが図７～図９に示すコード列である。

図７には、図５の実行頻度一位の実行経路５５１上の命令群をコード化したスレッド７００を記してある。コード７０１、７１３、７１９はそれぞれラベルコードである。コー

ド７０２～７１２は、図５のブロック５００、５０１、５０２を他の実行経路に分岐しないようにコード化した内容になっており、ブロック５０５、５０６がこの実行経路を通るかどうかの２択に変えたコードを含んでいる。

【００６４】

コード７１４、７１５は実行経路５１１を通る場合に他のプロセッサエレメントで実行されているスレッドを停止させるコードである。

コード７１７、７１８は、実行経路５５１が実行されたときにコードを反映させる処理になる。この反映処理は、実行経路５５１の出口で生存していて、かつ実行経路５５１で変更されているデータが対象になる。

コード７２０、７２１はこのスレッド７００の終了処理である。

図８には、実行経路５５２上の命令をアセンブラコードに変換したスレッド８００を記してある。

【００６５】

コード８０１、８１４、８１７、８２０はそれぞれラベルコードである。

またコード８０２～８１３は図５におけるブロック５００、５０１、５０３の命令をコード化した内容になっている。コード８１５、８１６は実行経路５５２を通ることが確定した場合に、他のプロセッサエレメントで実行されているスレッドを停止させ、コード８２１、８２２はスレッド８００終了処理を行っている。また、コード８１８、８１９は、実行経路５５２が実行されたときにコードを反映させる処理になる。

【００６６】

図９には、図５におけるブロック５００、５０４を通る実行経路を通る場合の最適化されたコードであるスレッド９００を記してある。

コード９０１、９０９、９１２、９１４はそれぞれラベルコードである。

またコード９０２～９０８は図５におけるブロック５００、５０４の命令をコード化した内容になっている。コード９１０、９１１はこの実行経路を通ることが確定した場合に、他のプロセッサエレメントで実行されているスレッドを停止させ、コード９１５、９１６はスレッド９００の終了処理を行っている。また、コード９１３は、ブロック５００、５０４が実行されたときにコードを反映させる処理になる。

【００６７】

図７のコード７０２～７１２、図８のコード８０２～８１３、図９の９０２～９０８において同じメモリへの格納処理が生じる場合、各スレッドでの値の保証ができなくなり、プログラム作成者の望む結果が得られないことがあるため、メモリ共有型では別のメモリ領域への格納処理に変えることがある。

【００６８】

図１０には、図６～９のように生成される各スレッドをターゲットハードウェアに並列実行させるための、スレッド制御のためのコード列を示している。このスレッド１０００がメモリ共有型のコンピュータを対象とした場合のメインスレッドということになる。

コード１００１～１００４においては、解析部１０１から得た解析情報と、実行経路の実行頻度の情報を基に、実行頻度の高かった実行経路に関するスレッドを生成している。ここでは、ターゲットハードウェアは、十分なプロセッサエレメントを有するものとして、全ての実行経路のスレッドを立ち上げている。

【００６９】

ラベルコード１００５から実行されるコード１００６～１００８はスレッドの開始を各プロセッサエレメントに行わせるコードである。ラベルコード１００９から実行されるコード１０１０～１０１２は、実行されているスレッドから終了したかどうかの返答を待つコードである。ラベルコード１０１３から実行されるコード１０１４～１０１６は、全スレッドの終了後、それぞれのスレッドを破棄し、プロセッサエレメントを自由にするコードである。

【００７０】

この図１０のメインコードとスレッド６００、スレッド７００、スレッド８００、スレ

ッド９００を含んだ実行プログラムをコンパイラ装置１００は生成する。なお、スレッド６００、スレッド７００、スレッド８００、スレッド９００は並列実行される。

ここから、生成されるプログラムに使用され、図６～図１４及び図２１に用いられているコードの説明を行う。

【００７１】

図６は、ソースプログラムをそのまま、特に最適化を施さずに変換したコード列を示した図であり、図７、図８、図９はそれぞれ順に、ターゲットハードウェア１３０のメモリ形態がメモリ共有型である場合における、部分ソースプログラム５１０の実行経路５５１、５５２、そしてブロック５０１、５０４を通る実行経路に関して最適化したコード列であり、図１２、図１３、図１４はメモリ分散型である場合のコード列である。また、図２１は、ターゲットハードウェアのメモリ形態がメモリ分散型である場合のメインスレッドである。

【００７２】

図１０は、ターゲットハードウェア１３０の並列実行可能な命令数が既知である場合のメモリ共有型のメインスレッドを示したコード列であり、図１１は、未知の場合のメインスレッドを示したコード列である。

なお、以下において番地は、プロセッサ上の命令の番地であり、レジスタの番地であったり、そのレジスタに格納されている値であったりする。

【００７３】

「mov （番地１）,（番地２）」は、（番地１）の値を（番地２）のレジスタに格納するコードである。例えば図６のコード６０２においては、ａが示す番地の値がレジスタＤ１に格納される。

「add （番地１）,（番地２）」は、（番地１）の値と（番地２）の値とを加算し、その結果で得られた値で（番地２）の値を更新するコードである。例えば図６のコード６０４においては、レジスタＤ１に格納されている値と、レジスタＤ０に格納されている値を加算し、計算結果の値でレジスタＤ０の値を更新する。

【００７４】

「sub （番地１）,（番地２）」は、（番地２）の値から（番地１）の値を減算し、その結果で得られた値で（番地２）の値を更新するコードである。例えば図６のコード６１２においては、レジスタＤ０に格納されている値からレジスタＤ１に格納されている値を減算し、その計算結果をレジスタＤ０に格納している。

「cmp （番地１）,（番地２）」は、（番地１）の値と（番地２）とを比較するコードである。例えば図６のコード６０６においては、０とレジスタＤ０に格納されている値とを比較している。

【００７５】

「bge （番地３）」は、直前の比較コード cmp （番地１）,(番地２） の比較において、（番地２）の値が（番地１）の値以上であった場合に、（番地３）に指定されるコードに命令を飛ばすコードである。それ以外の場合には次のコードを続行する。例えば、図６のコード６０７においては、その前のコード６０６の比較を受けてレジスタＤ０に格納されている値が０以上であった場合に、コード６０８を実行させずにコード６０９に飛び、以降のコードを実行することになる。

【００７６】

「blt （番地３）」は、直前の比較コード cmp （番地１）,(番地２） の比較において、（番地２）の値が（番地１）の値よりも小さい場合に、（番地３）に指定されるコードに命令を飛ばすコードである。それ以外の場合には以降のコードを続行する。例えば、図７のコード７０６においては、その前のコード７０５の比較を受けて、レジスタＤ０に格納されている値が０よりも小さい場合に、コード７０７からコード７１９までは実行されずにコード７２０に飛び、以降のコードが実行されることになる。

【００７７】

「jmp （番地１）」は、（番地１）で指定されるコードに命令を飛ばすコードである

。例えば、図６のコード６０８においては、コード６０９以下、コード６２６までを実行させずにコード６２７に飛び、以降のコードを実行することになる。

「not （番地１）」は、（番地１）の値をビット反転した（１の補数）値にして、その値で（番地１）を更新するコードである。例えば、図６のコード６２９においては、レジスタＤ０に格納されている値をビット反転した（１の補数）値にして、レジスタＤ０に格納しなおしている。

【００７８】

「inc （番地１）」は、（番地１）の値に１加算して、その値で（番地１）を更新するコードである。例えば図６のコード６３０においては、レジスタＤ０に格納されている値に１足して、Ｄ０＋１の値をレジスタＤ０に格納しなおしている。

「dec （番地１）」は、（番地１）の値から１減算して、その値で（番地１）を更新するコードである。例えば、図１１のコード１１１３においてはレジスタＤ１に格納されている値から１引いた、Ｄ１－１の値をレジスタＤ１格納しなおしている。

【００７９】

「clr （番地１）」は、（番地１）の値をクリアするコードでその値を０にするコードである。例えば、図６のコード６３３においてはレジスタＤ０の値をクリアし、レジスタＤ０の値を初期化している。

「asl （番地１）,（番地２）」は、ターゲットハードウェアで使用されている命令語長の違いによる番地のずれを防ぐためのコードであり、主にコード間の遷移を行う場合に必要となる。プログラムにおいては各命令の番地は、命令語長の単位で管理されており、例えば、命令語長が８bitであった場合には、命令１の番地が０であった場合に、その次に続く命令２の番地は８になる。命令１の次の命令２に移行したい場合に、単純に命令１の番地に１足しても命令２の番地にならないので命令２は実行されず、番地の整合性が取れなくなる。このコードの実質的内容はというと、命令語長の値を（番地２）の値にかけて、（番地２）のレジスタに格納することがこのコードの内容である。

【００８０】

「ret」は、スレッドからプログラムのメインへの復帰を実行する実行するコードである。

次に、スレッド制御のためのコードの内容について説明する。

「__createthread （番地１）,（番地２）」は、スレッドを生成するコードであり、（番地１）から始まるプロセスを生成する。その実行状態の情報は（番地２）に更新される。例えば、図１０のコード１００２においては、ＬＡＢＥＬ５００－５０１－５０２で始まるスレッド、即ち図７のスレッド７００を生成し、その実行情報はＴＨＲＥＡＤ５００－５０１－５０２に格納される。

【００８１】

「#beginthread （番地）」は、スレッドの開始コードで、（番地）のスレッドの実行開始を促がす。例えば、図１０のコード１００６においては、ＬＡＢＥＬ５００－５０１－５０２で始まるスレッド、即ち図７のコード列で示されるスレッド７００を実行する。

「#endthread」は、スレッドの終了コードで、現在実行しているスレッドを終了状態に設定し、スレッドが終了したことを示す情報を返す。例えば、図７のコード７２０においてはスレッド７００を終了し、終了したことを示す情報をプログラムのメインに返す。

【００８２】

「__deletethread （番地）」は、スレッドの破棄コードで、（番地）から始まるスレッドを破棄する。例えば、図１０もコード１０１４においては、ＬＡＢＥＬ５００－５０１－５０２のスレッド、つまりスレッド９００を破棄する。

「__killthread （番地）」は、他のプロセッサエレメントで実行されているスレッドの強制終了コードで、（番地）から始まるスレッドを停止させる。例えば、図７のコード７１４においては、ＬＡＢＥＬ５００－５０１－５０３で始まるスレッド、即ち、図８のスレッド８００の実行を、実行途中であっても中止させる。

【００８３】

「#waitthread （番地）」は、スレッドの終了を待つコードで、（番地）から始まるスレッドの実行結果の終了を待つ。この終了は上記#endthreadからの情報によって知ることができる。例えば、図１０のコード１０１０においては、ＴＨＲＥＡＤ５００－５０４の終了を待っている。

「#commit （番地１）」は、メインプログラム、若しくはスレッドプロセスで生成した情報（番地１）を、メインプログラムと全てのスレッドプロセスに反映させるコードである。

【００８４】

「__broadcast （番地１）,（番地２）」は、ターゲットハードウェアのメモリ管理方式が分散型である場合に、各プロセッサエレメントに接続されているメモリに実行結果を反映させるコードである。スレッドの実行結果の値（番地１）で、全てのスレッドの（番地２）の値を更新する。

「__getparallelnum （番地）」は、ターゲットハードウェアが同時実行可能なスレッドの数を（番地）に返すコードで、ターゲットハードウェアの並列実行可能なプロセッサエレメントの数を取得するために必要なコードで、特にコンパイル時にターゲットハードウェアの並列実行可能なプロセッサエレメントの数が分からない場合に必要となる。

＜動作＞

本コンパイラ装置によって生成される実行プログラムの生成における本コンパイラ装置の動作を実行プログラムの生成手順に沿ってフローチャートを用いながら説明する。

【００８５】

まず、コンパイラ装置１００に入力された、ソースプログラム１１０は、解析部１０１によって、その中の分岐や繰り返しに関する情報を取得し、それを元にどのような実行経路があるかを検出し、後に実行経路を特定できるように実行経路を識別子化する。

一度ソースプログラム１１０は、最適化部１０３、コード変換部１０４を通じて、特別な最適化を施さずに実行プログラムに変換されて、ターゲットハードウェア１３０上において実際に実行されて実行経路の実行頻度に関する情報を得る。この実行経路の実行頻度の取得方法に関して、図１５のフローチャートを用いて説明する。

【００８６】

部分ソースプログラム５１０の中の実行経路の実行頻度を計測するために、最適化部１０３は、ソースプログラムをそのまま最適化処理などを施さずに、プロファイリング用コードを組み込んで実行コードを作成し、作成された実行コードはコード変換部１０４によってターゲットハードウェア１３０上において実行できるような実行プログラムに変換、生成される（ステップＳ１５００）。ここでプロファイリング用コードはソースプログラム上で条件分岐があった場合にその分岐においてどちらの分岐に進んだかを検出するためのコードであり、識別子化した実行経路に関して、その実行経路を一回通るたびに１カウント加算するコードである。このプロファイリングコードを挿入すると実行速度は遅くなるので、最終的に生成される実行プログラムには当然このプロファイリングコードは組み込まれない。

【００８７】

その後に作成された当該実行プログラムをターゲットハードウェア１３０上で実行し、実行経路の実行頻度を計測する（ステップＳ１５０２）。解析部１０１によって作成されている実行経路の識別子に、その実行経路が実行された回数を加算していき、その情報をターゲットハードウェアのメモリに記憶させ、これを実行経路の実行頻度の情報１４０とする。そして取得した実行経路の実行頻度の情報１４０はコンパイラ装置１００の実行経路指定部１０２に渡され、それを基に、実際の目的とする実行プログラムは作成される。

【００８８】

ここで、実行経路の実行頻度に関する情報１４０をコンパイラ装置１４０に渡す際に、ターゲットハードウェア１３０のハードウェアの仕様に関する情報も渡す。このターゲットハードウェア１３０のハードウェアの仕様には、ターゲットハードウェア１３０の並列実行可能なプロセッサエレメントの数と、ターゲットハードウェア１３０のメモリ形態に

関する情報がある。これらの情報は元からターゲットハードウェア１３０のＲＯＭに記憶されており、それがコンパイラ装置１００に送信される。

【００８９】

その後に、実際の目的とする実行プログラムの生成を行う。その生成手順に関して図１９のフローチャートを用いて説明する。

まずコンパイラ装置１００は、大本のソースプログラムをそのまま実行形式に出来るコードに変換した第一コードを作成する（ステップＳ１９０１）。そして、実行経路指定部１０２は、ターゲットハードウェア１３０から取得した実行経路の実行頻度に関する情報１４０に基づき、その実行頻度の高かった、即ち実行回数の多かった優先実行経路を実行頻度の高い順に抽出し（Ｓ１９０５）、それとターゲットハードウェア１３０の並列実行可能なプロセッサエレメントの数により、優先実行経路上の命令を最適化した第二コードを生成する（Ｓ１９０７）。この第二コードはターゲットハードウェア１３０の並列実行可能なプロセッサエレメントの数より１少ない数まで生成されて良く、実行経路によって内容を変えて生成されて良く、実行頻度の回数の多かった実行経路の順に、それぞれの実行経路上の命令に対応するスレッドを生成して、その実行経路上の命令が最適化される。例えば、ターゲットハードウェア１３０の並列実行可能なプロセッサエレメントの数が４であった場合には、実行頻度第一位から第三位までの実行経路のスレッドを生成する。第一コードには複数の第二コードを制御するコードも含まれている。

【００９０】

そして、生成された第一コードと第二コードを並列実行させる編成にしたコードをコード変換部１０４がターゲットハードウェア１３０上において実行できるように実行プログラムを生成する（Ｓ１９０９）。

この動作を具体的に図５（ａ）の部分ソースプログラム５１０を実行プログラムに変換するとして、その過程において生成されるコード等を用い説明する。

【００９１】

まず、コンパイラ装置１００には、図５（ａ）にある部分ソースプログラム５１０を含むソースプログラムが入力される。解析部１０１は、部分ソースプログラム５１０を解析し、その実行経路が、図５（ｂ）のフローグラフにおけるブロック５００、５０１、５０２を通る経路、ブロック５００、５０１、５０３を通る経路、ブロック５００、５０４を通る経路の３つの経路があることを解析し、それぞれの実行経路を識別子化する。最適化部１０３は、最適化を施さずに、コード変換部１０４は、部分ソースプログラム５５１をそのままアセンブラコードにしたスレッド６００のコード列を生成し、これにプロファイリングコードを挿入した実行プログラムを生成する。当該実行プログラムをターゲットハードウェア１３０が実行し、その実行によってカウンティングされた実行経路の実行頻度の情報を、例えば、実行経路５００－５０１－５０２：２４回、実行経路５００－５０１－５０３：１５回、実行経路５００－５０４：３回という情報として、コンパイラ装置１００に渡される。また、ターゲットハードウェア１３０のハードウェアの仕様に関する情報も渡す。ここではターゲットハードウェアのプロセッサエレメントの数は例えば４としたら並列実行可能なプロセッサエレメントの数として４を、そしてメモリの形態がメモリ共有型であるのでメモリ情報として０をコンパイラ装置１００に渡す。

【００９２】

コンパイラ装置１００の実行経路指定部１０２は、実行経路の実行頻度の情報１４０を受け取り、最適化部１０３、コード変換部１０４はこれに基づき、メインスレッド１０００を生成する。ターゲットハードウェア１３０の並列実行可能なプロセッサエレメントの数が４であるので、並列実行できるスレッドはメインスレッドを含めて４になり、メインスレッド内において、４つのスレッド６００、７００、８００、９００が生成される。それぞれのスレッド６００、７００、８００、９００はターゲットハードウェア１３０上の別のプロセッサエレメントで実行されるように編成されたコードを最終的に生成し、コード変換部１０４は、ターゲットハードウェア１３０が実行できるように実行プログラム１２０を生成する。

＜第二の実施形態＞

第二の実施形態においては、ターゲットハードウェアのメモリ形態がメモリ分散型であった場合について、主に、第一の実施形態と異なる点を説明する。

【００９３】

その主な違いは、プロセッサエレメントそれぞれにメモリが接続され、プロセッサエレメントはそれぞれのメモリの値を使用するため、メモリ共有型のような値のメモリアクセス競合による性能の低下の恐れがなくなることにある。

その違いを示すために図１２～図１４及び図２１のコード列を用意した。図１２のスレッド１２００の実行内容は、図７のスレッド７００に、図１３のスレッド１３００の実行内容は、図８のスレッド８００に、図１４のスレッド１４００の実行内容は、図９のスレッド９００に、相当する。図２１は、メモリ分散型の場合のメインスレッドである。

【００９４】

ターゲットハードウェア１３０のメモリの形態がメモリ共有型であった場合には、図７～図９におけるコード７０２、８０２、９０２のように逐一ａの値をレジスタに格納しなおさなければならなかったが、メモリ分散型ではそのようにする必要はなく、例えば、図２１にあるメモリ分散型の場合のメインスレッドのように、メインスレッドにおいて各プロセッサエレメントの各レジスタにブロードキャストすることで、処理を省略することができる。そのためのコードが図２１におけるコード２１０４～２１０７である。

【００９５】

コード２１０５では、コード２１０１～２１０３で生成されたスレッドにａの値を各プロセッサエレメントのメモリのレジスタＤ０に格納するように各スレッドを実行しているプロセッサエレメントに通達する。

コード２１０６では、コード２１０１～２１０３で生成されたスレッドにｂの値を各プロセッサエレメントのメモリのレジスタＤ１に格納するように各スレッドを実行しているプロセッサエレメントに通達する。

コード２１０７では、コード２１０１～２１０３で生成されたスレッドにｃの値を各プロセッサエレメントのメモリのレジスタＤ２に格納するように各スレッドを実行しているプロセッサエレメントに通達する。

【００９６】

また、各スレッドで実行された実行結果が、そのスレッドの実行条件が成立した場合に、メインスレッドが走っているプロセッサエレメントに接続されているメモリに、その実行結果を反映させる必要があり、それが「__commit」ではじまるコードになる。例えば図１２においては、コード１２１５やコード１２１６がそれにあたる。これにより、スレッドの実行結果が反映されるようになる。

【００９７】

ターゲットハードウェアのメモリ形態がメモリ分散型である場合、スレッド２１００、１２００、１３００、１４００を含んで編成された実行プログラムが生成される。これにより、ターゲットハードウェア１３０のメモリ形態がメモリ分散型であっても実行プログラムは、値の整合がとれ、正常に実行される。

メモリ分散型のハードウェアを対象とした場合の実行プログラムの実行手順について、主にスレッドの制御に関する部分を、図１７のフローチャートを用いて説明する。

【００９８】

まず、他のプロセッサエレメントにおいて実行されるスレッドを生成する（ステップＳ１７００）。つまり、スレッド１２００、１３００、１４００を生成する。それぞれにおけるこの前部分において得られたデータを各スレッドを実行するプロセッサエレメントのメモリに送信し、記憶させる（ステップＳ１７０１）。その後に、各スレッドを実行し（ステップＳ１７０２）、スレッドが終了した後（ステップＳ１７０３）に、そのスレッドの成立条件が成立している場合（ステップＳ１７０４）に、プログラムのメインへの値の反映を行う（ステップＳ１７０５）。そして、その後に自スレッドの破棄を行う（ステップＳ１７０５）。

＜第三の実施形態＞

第一、及び第二の実施形態においては、ターゲットハードウェアの並列実行可能な処理の数がコンパイラ装置には既知の物として説明してきたが、ターゲットハードウェアが並列実行可能なプロセッサエレメントの数が分からない場合もある。つまり、実行経路の実行頻度に関する情報、及びターゲットハードウェアのメモリ形態が予めコンパイラ装置に与えられており、いきなり実行プログラムをターゲットハードウェアに実行させたい場合などである。この場合メインプログラムの中に、当該プロセッサエレメントの数を取得するコードを組み込み、それと生成されるスレッドの数との整合を採るためのコードも組み込む必要が出てくる。そのために必要なコード列を図１１に示してあり、その実行内容を説明する。なお、ここでは、ソースプログラムは図６にあるものであり、生成されるスレッドは図７～９の３つであるものとして説明する。

【００９９】

ターゲットハードウェアのプロセッサエレメントの数を取得し、コンパイラによって生成されるスレッドの数との整合をとるコードがラベル１１０５から始まるコード１１０６～１１１８に記されている。

まず、コンパイラによって生成されるスレッドの数ｍを取得し、その数ｍをレジスタＤ０に格納する（コード１１０６）。次にターゲットハードウェアの並列実行可能なプロセッサエレメントの数ｎを取得し、その値をレジスタＤ１に格納する（コード１１０７）。そしてレジスタＤ０に格納されたｍとレジスタＤ１に格納されたｎの値を比較し（コード１１０８）、ｎ≧ｍならばラベルコード１１１１に飛び（コード１１０９）、ｎ＜ｍならばラベルコード１１１３に飛ぶ（コード１１１０）。

【０１００】

ｎ≧ｍの場合には、特に問題はなく、ｍの値をレジスタＤ１に格納する（コード１１１２）。

ｎ＜ｍの場合には、生成されたスレッドの数ｍの方が並列実行可能な命令数ｎを上回っているため、すべてのスレッドを実行できない。

そこで、まず、レジスタＤ１に格納されている値ｎから１引いた数をＤ１に格納しなおす（コード１１１４）。このｎ－１の数が必要とする実行可能なスレッドの数である。一つ余るプロセッサエレメントは、元のプログラムをそのままコードにした図６のコードを実行する。

【０１０１】

次に命令の番地計算を行うために、ｎ－１の値に命令語長、例えば8bitなら８をかけ（コード１１１５）、Ｐ＿ＰＯＩＮＴＥＲの番地をレジスタＤ２に格納する（コード１１１６）。レジスタＤ２に格納された値からレジスタＤ１に格納された値を引いて、算出された値でレジスタＤ２を更新する（コード１１１７）。そして、レジスタＤ２に格納されている番地の値に命令を飛ばす（コード１１１８）。このＤ２に格納されている値によって以下のどのスレッドから開始するのかを決定する。例えば、ターゲットハードウェアの並列実行可能数が２であった場合には、コード１１２２から開始する。並列実行可能数が３の場合には、コード１１２１から開始する。コード１１２０～コード１１２２に関しては下から順に実行頻度の高かった実行経路を実行するスレッド開始のコードになっている。

【０１０２】

このスレッド１１００をメインスレッドにすることにより、ターゲットハードウェアの並列実行可能数を得ていない場合であってもこのコンパイラ装置は、実行プログラムを生成できる。なお、コード１１２４以降のコードは、全てを図示していないが、図１０におけるコード１０１２以降のコードと同様の構成とする。

ターゲットハードウェアの性能が分からない場合に、その性能を取得する必要があり、その流れを図１６のフローチャートに簡単に示しておいた。

【０１０３】

まず、コンパイラ装置１００の最適化部１０３がターゲットハードウェア１３０に関し

て、同時並列実行可能な処理数が未知であるか、既知であるかを判定する（ステップＳ１６０１）。これは、ターゲットハードウェア１３０、その仕様に関する情報を得ているか、いないかで判断する。未知である場合には、この第三の実施形態において説明した図１１のコードを実行プログラムの中に組み込む。そしてターゲットハードウェア１３０のメモリ形態がメモリ共有型か、メモリ分散型であるかの情報を得て（ステップＳ１６０３）それを元に実行プログラムを作成する。

＜第四の実施形態＞

第四の実施形態においては、上記実施の形態と異なり、図１８にある機能ブロック図にあるように、上記実施の形態におけるコンパイラ装置にプログラムを実行できる実行部１８０７を組み込んだプログラム変換実行装置１８００の実施の形態を示す。

【０１０４】

その主な差は、図１８において、プログラム変換実行装置１８００はその内部に、実行プログラム格納部１８０６と実行部１８０７を組み込んだことにあり、これにより、ターゲットハードウェアに予め一度プログラムを実行して実行頻度情報を得るためにハードウェアと接続して実行させる手間をはぶけ、自機によって実行頻度情報を取得でき、かつプログラム実行結果を得ることも可能となる。

【０１０５】

実行プログラム格納部１８０６は、コード変換部１８０５によって生成された実行プログラムを記憶しておく機能を有し、ＲＡＭを含んで構成される。

実行部１８０７は、実行プログラム格納部１８０６から実行プログラムを読み出し、当該実行プログラムを実行する機能を有し、ＭＰＵ、ＲＯＭ、ＲＡＭを含んで構成され、図１におけるターゲットハードウェア１３０と同等の働きをする。なお、このＣＰＵは複数のプロセッサエレメントで構成されている。

【０１０６】

生成されるコードに関しては第一～第三の実施形態におけるものと変わらない。また、第四の実施形態においては、プログラムを変換しながら実行するインタプリタとしても使用できるようになる。

＜補足＞

なお、上記第一の実施形態及び第二の実施形態においては、ターゲットハードウェアは生成されるスレッド全てを実行できるだけの十分な数のプロセッサエレメントを有するものとして説明したが、例えばプロセッサエレメントの数が少なく２個とかの場合には、スレッド６００とスレッド７００だけが並列実行されるようにメインスレッドは構成される。この場合、図１０においてはコード１００３、１００４、１００７、１００８、１０１１、１０１２、１０１５、１０１６は不要になる。

【０１０７】

また、上記実施の形態においては第一コード、つまり概要の図３のスレッド３００の実行速度は、通常、他のスレッドよりも遅いことを想定して、実行プログラムは作成されているが、速い場合も考慮にいれて、スレッド３００の最後に、他のスレッドを停止させるコードを含んでも良い。

また、上記実施の形態においてはターゲットハードウェアが複数のプロセッサエレメントを内包するように記述したが、例えば、一台のパソコンを一つのプロセッサエレメントと見立てて、複数のパソコンをネットワークを介して接続して並列実行する形をとっても良い。

【０１０８】

また、上記実施の形態において一つのスレッドが成立した場合に、他のプロセッサエレメントは実行していたスレッドを停止し、普通はスレッドと演算データを消去し、次に割り振られるスレッドを実行するが、同一のスレッドが何度も実行される場合には、逐次スレッドを割り振ることは非効率的であり、生成される目的プログラムの実行速度の低下を招くこともある。そこで、次に実行するスレッドが割り振られたスレッドと同内容であり、与えられる演算用データ値だけが異なる場合には、当該スレッドは破棄せずに保持して

おき、スレッドを実行するのに必要な演算データだけがメインスレッドからブロードキャストされるようなコードを生成を含んだ目的プログラムを生成することとしても良い。

【産業上の利用可能性】

【０１０９】

本発明に係るコンパイラ装置は、大容量計算を要するプログラムが必要とされる分野においてその計算結果がより早く出るようなプログラムの生成に活用できる。

【図面の簡単な説明】

【０１１０】

【図１】本発明のコンパイラ装置の構成を示したブロック図である。

【図２】本発明の概念を説明するためのフローグラフを示した説明図である。

【図３】本発明の概要を説明するための概要図である。

【図４】プロセッサエレメントとメモリの関係を示した関係図である。

【図５】本発明の内容を説明するために用意したソースプログラムとそのフローグラフである。

【図６】図５のソースプログラムをそのままアセンブラコードに変換したコード列である。

【図７】ターゲットハードウェアがメモリ共有型の場合の実行経路５００－５０１－５０２のコード列である。

【図８】ターゲットハードウェアがメモリ共有型の場合の実行経路５００－５０１－５０３のコード列である。

【図９】ターゲットハードウェアがメモリ共有型の場合の実行経路５００－５０４のコード列である。

【図１０】ターゲットハードウェアがメモリ共有型の場合のスレッド制御コードである。

【図１１】ターゲットハードウェアの並列実行可能なプロセッサエレメントの数が未知の場合の制御コードである。

【図１２】ターゲットハードウェアがメモリ分散型の場合の実行経路５００－５０１－５０２のコード列である。

【図１３】ターゲットハードウェアがメモリ分散型の場合の実行経路５００－５０１－５０３のコード列である。

【図１４】ターゲットハードウェアがメモリ分散型の場合の実行経路５００－５０４のコード列である。

【図１５】実行頻度を検出するための手順を示したフローチャートである。

【図１６】ターゲットハードウェアの性能の違いによるコードの変化を示すフローチャートである。

【図１７】メモリ分散型におけるスレッド生成から値の反映までを示したフローチャートである。

【図１８】プログラム変換実行装置１８００の機能ブロック図である。

【図１９】本発明のプログラム変換装置の動作を示したフローチャートである。

【図２０】従来技術におけるトレーススケジューリングの説明に用いる説明図である。

【図２１】ターゲットハードウェアがメモリ分散型の場合のスレッド制御コードである。

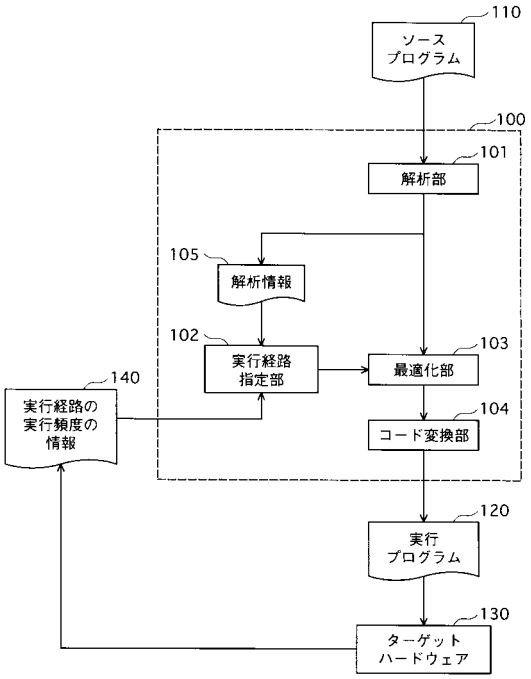
【符号の説明】

【０１１１】

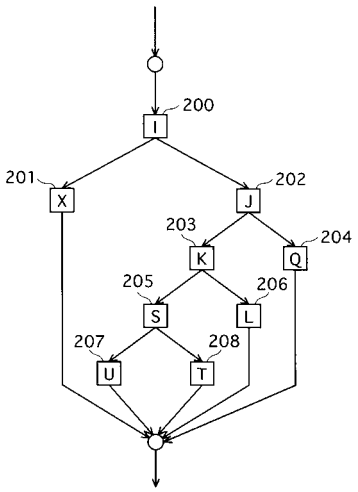
１００　コンパイラ装置
１０１　解析部
１０２　実行経路指定部
１０３　最適化部
１０４　コード変換部
１０５　解析情報
１２０　実行プログラム
１３０　ターゲットハードウェア

１４０　実行経路の実行頻度の情報
４００、４０１、４０２、４１０、４１１、４１２　プロセッサエレメント
４０３、４１３、４１４、４１５　メモリ
５００、５０１、５０２、５０３、５０４　命令ブロック
５０５、５０６　分岐ブロック
５１０　部分ソースプログラム
５１１　実行頻度一位の実行経路
５１２　実行頻度二位の実行経路
１８０６　実行プログラム格納部
１８０７　実行部

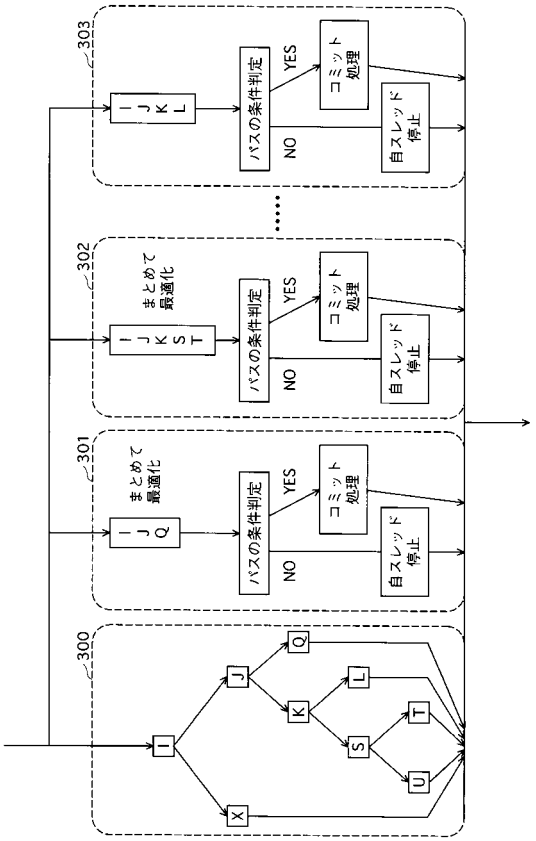
【図１】



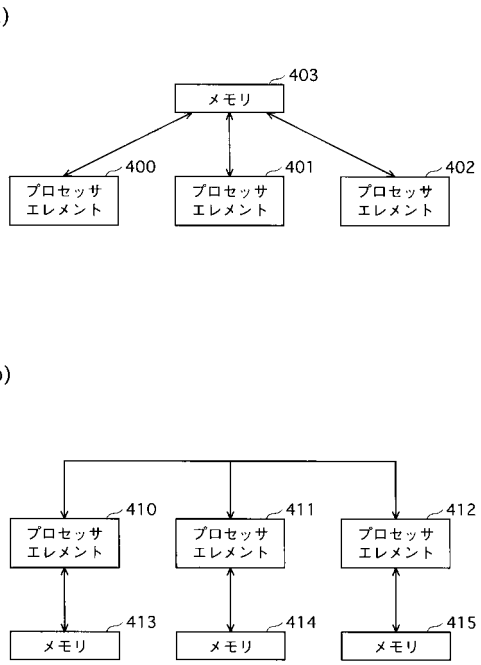
【図２】



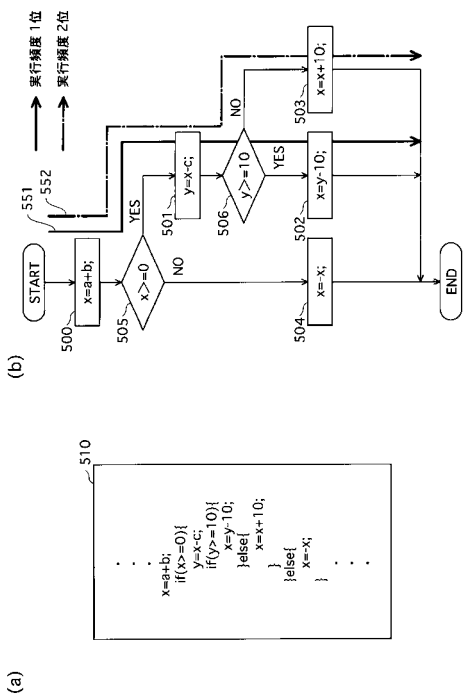
【図３】


303
I J K L
パスの条件判定
YES
NO
コミット処理
自スレッド停止
302
まとめて最適化
I J K S T
301
まとめて最適化
I J Q
300
I
J
K
L
S
T
Q
U
X

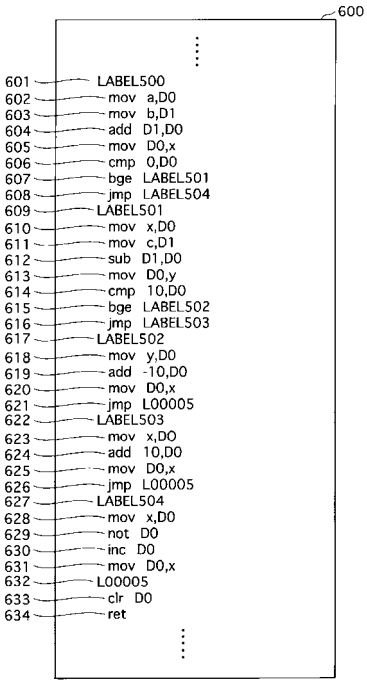
【図４】


(a)
403
メモリ
400
401
402
プロセッサエレメント
(b)
410
411
412
プロセッサエレメント
413
414
415
メモリ

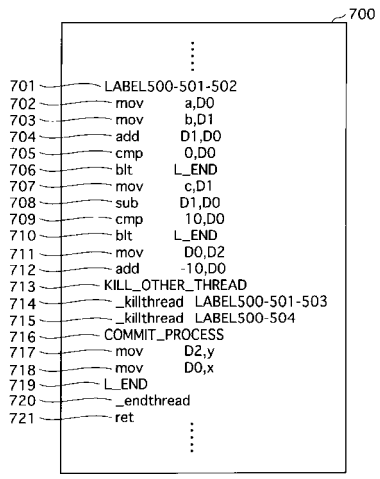
【図５】


実行頻度 1位
実行頻度 2位
551
552
START
x=a+b;
500
x>=0
505
501
y=x-c;
506
y>=10
503
x=x+10;
502
x=y-10;
504
x=-x;
END
(b)
510
x=a+b;
if(x>=0){
y=x-c;
if(y>=10){
x=y-10;
}else{
x=x+10;
}
}else{
x=-x;
}
(a)

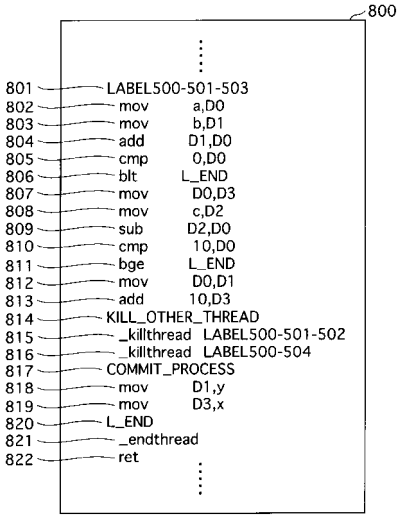
【図６】


600
601 LABEL500
602 mov a,D0
603 mov b,D1
604 add D1,D0
605 mov D0,x
606 cmp 0,D0
607 bge LABEL501
608 jmp LABEL504
609 LABEL501
610 mov x,D0
611 mov c,D1
612 sub D1,D0
613 mov D0,y
614 cmp 10,D0
615 bge LABEL502
616 jmp LABEL503
617 LABEL502
618 mov y,D0
619 add -10,D0
620 mov D0,x
621 jmp L00005
622 LABEL503
623 mov x,D0
624 add 10,D0
625 mov D0,x
626 jmp L00005
627 LABEL504
628 mov x,D0
629 not D0
630 inc D0
631 mov D0,x
632 L00005
633 clr D0
634 ret

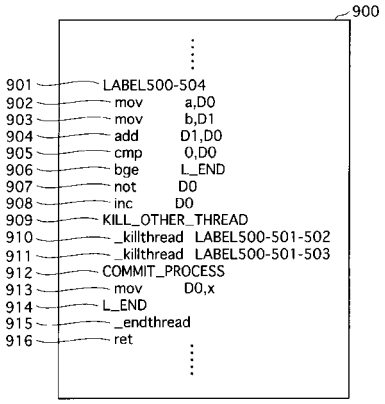
【図７】


700
701 LABEL500-501-502
702 mov a,D0
703 mov b,D1
704 add D1,D0
705 cmp 0,D0
706 blt L_END
707 mov c,D1
708 sub D1,D0
709 cmp 10,D0
710 blt L_END
711 mov D0,D2
712 add -10,D0
713 KILL_OTHER_THREAD
714 _killthread LABEL500-501-503
715 _killthread LABEL500-504
716 COMMIT_PROCESS
717 mov D2,y
718 mov D0,x
719 L_END
720 _endthread
721 ret

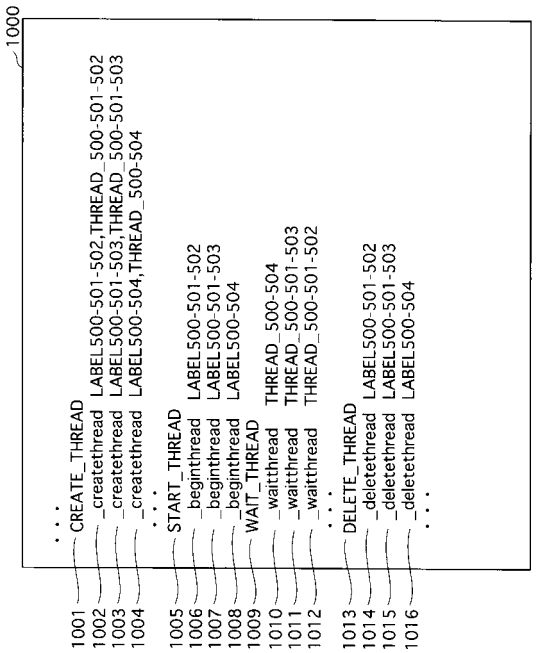
【図８】


800
801 LABEL500-501-503
802 mov a,D0
803 mov b,D1
804 add D1,D0
805 cmp 0,D0
806 blt L_END
807 mov D0,D3
808 mov c,D2
809 sub D2,D0
810 cmp 10,D0
811 bge L_END
812 mov D0,D1
813 add 10,D3
814 KILL_OTHER_THREAD
815 _killthread LABEL500-501-502
816 _killthread LABEL500-504
817 COMMIT_PROCESS
818 mov D1,y
819 mov D3,x
820 L_END
821 _endthread
822 ret

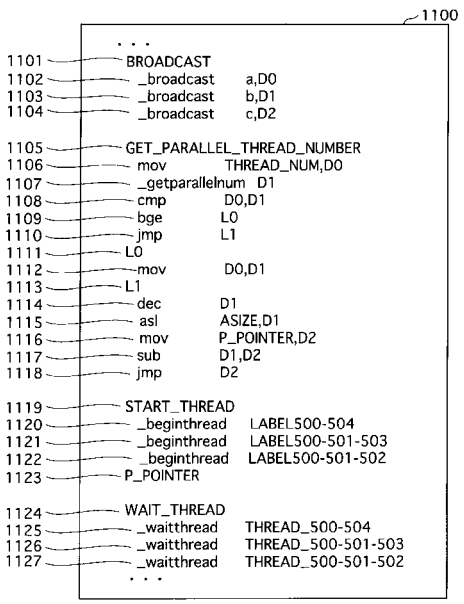
【図９】


900
901 LABEL500-504
902 mov a,D0
903 mov b,D1
904 add D1,D0
905 cmp 0,D0
906 bge L_END
907 not D0
908 inc D0
909 KILL_OTHER_THREAD
910 _killthread LABEL500-501-502
911 _killthread LABEL500-501-503
912 COMMIT_PROCESS
913 mov D0,x
914 L_END
915 _endthread
916 ret

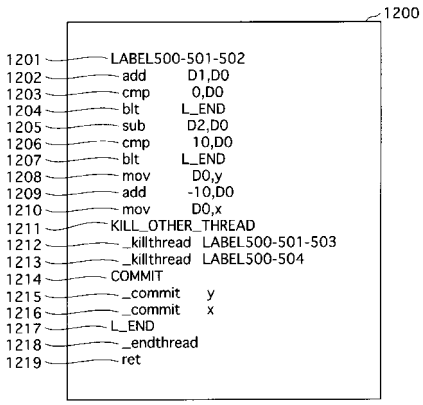
【図１０】


1000
CREATE_THREAD
_createthread LABEL500-501-502,THREAD_500-501-502
_createthread LABEL500-501-503,THREAD_500-501-503
_createthread LABEL500-504,THREAD_500-504
START_THREAD
_beginthread LABEL500-501-502
_beginthread LABEL500-501-503
_beginthread LABEL500-504
WAIT_THREAD
_waitthread THREAD_500-504
_waitthread THREAD_500-501-503
_waitthread THREAD_500-501-502
DELETE_THREAD
_deletethread LABEL500-501-502
_deletethread LABEL500-501-503
_deletethread LABEL500-504
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016

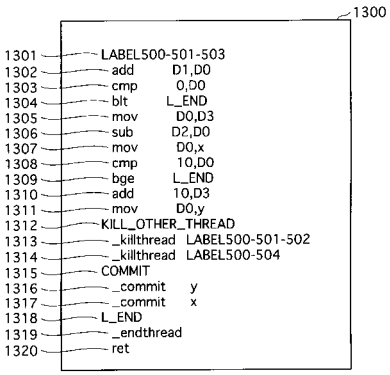
【図１１】



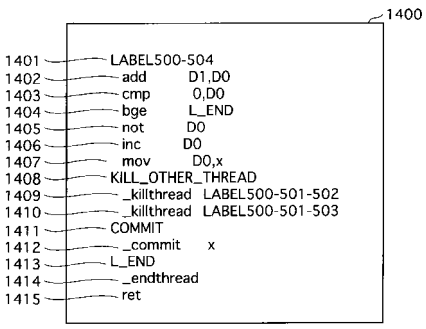
【図１２】



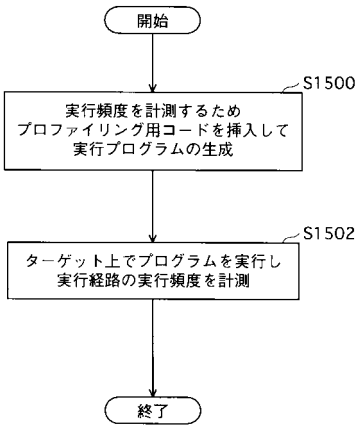
【図１３】



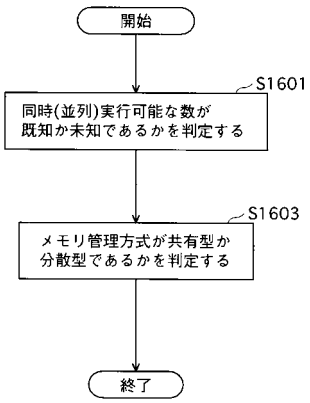
【図１４】



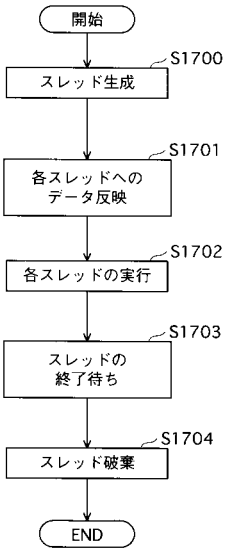
【図１５】



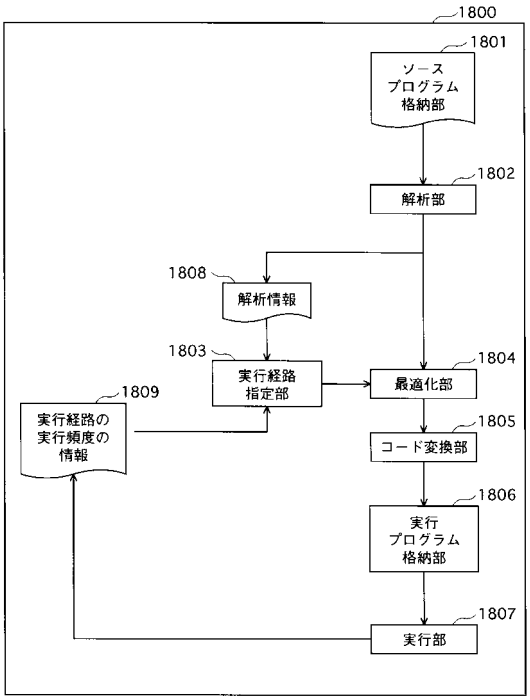
【図１６】



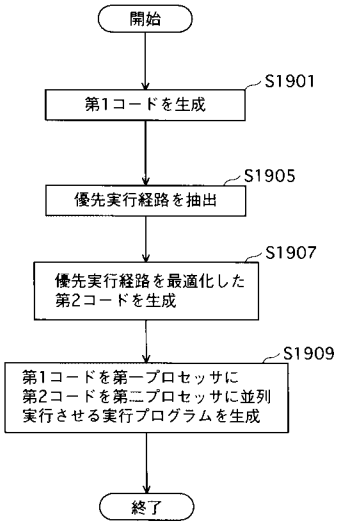
【図１７】



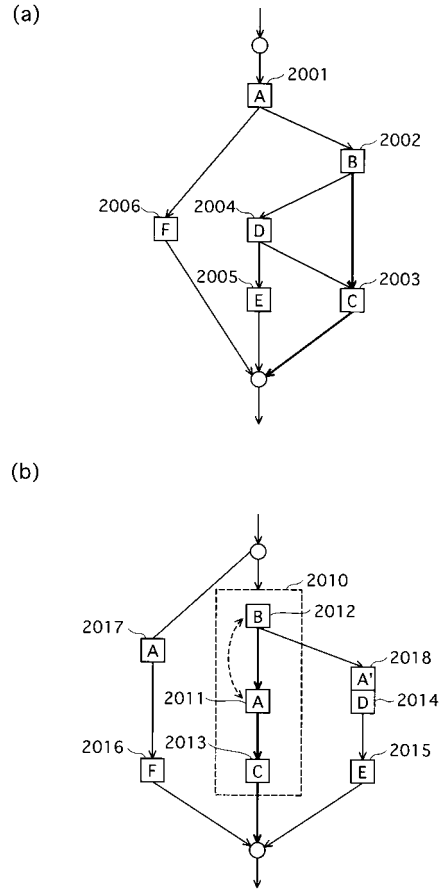
【図１８】



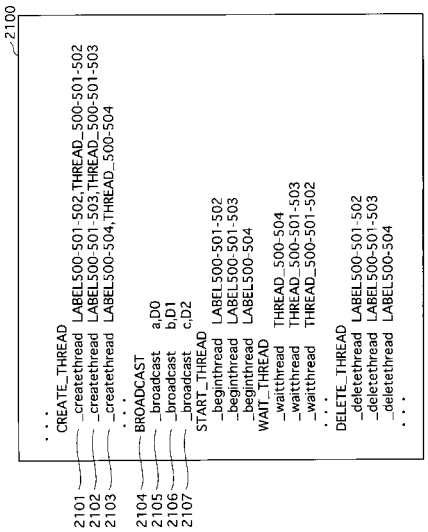
【図１９】


開始
S1901
第1コードを生成
S1905
優先実行経路を抽出
S1907
優先実行経路を最適化した
第2コードを生成
S1909
第1コードを第一プロセッサに
第2コードを第二プロセッサに並列
実行させる実行プログラムを生成
終了

【図２０】


(a)
2001
A
2002
B
2006
F
2004
D
2005
E
2003
C
(b)
2010
2012
2017
A
B
2018
A'
2011
A
D
2014
2016
2013
2015
F
C
E

【図２１】


2100
. . .
CREATE_THREAD
_createthread LABEL500-501-502,THREAD_500-501-502
_createthread LABEL500-501-503,THREAD_500-501-503
_createthread LABEL500-504,THREAD_500-504
. . .
BROADCAST a,D0
_broadcast b,D1
_broadcast c,D2
START_THREAD
_beginthread LABEL500-501-502
_beginthread LABEL500-501-503
_beginthread LABEL500-504
WAIT_THREAD THREAD_500-504
_waitthread THREAD_500-501-503
_waitthread THREAD_500-501-502
. . .
DELETE_THREAD
_deletethread LABEL500-501-502
_deletethread LABEL500-501-503
_deletethread LABEL500-504
. . .
2101
2102
2103
2104
2105
2106
2107

フロントページの続き

(56)参考文献 特開２００１－２８２５４９（ＪＰ，Ａ）
特開平１１－０９６００５（ＪＰ，Ａ）
特開２００３－３２３３０４（ＪＰ，Ａ）
特開平０５－０３５７７２（ＪＰ，Ａ）
特開２０００－１６３２６６（ＪＰ，Ａ）
特開平０７－０３６６８０（ＪＰ，Ａ）
特開平０６－０６００４７（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／４５