(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6469083号
(P6469083)
(45)発行日　平成31年2月13日(2019.2.13)　(24)登録日　平成31年1月25日(2019.1.25)

(51)Int.Cl.　F I
G06F 9/48 (2006.01)　G06F 9/48 370

請求項の数 29 (全 24 頁)

(21)出願番号　特願2016-510749(P2016-510749)
(86)(22)出願日　平成26年4月23日(2014.4.23)
(65)公表番号　特表2016-520911(P2016-520911A)
(43)公表日　平成28年7月14日(2016.7.14)
(86)国際出願番号　PCT/US2014/035094
(87)国際公開番号　WO2014/176310
(87)国際公開日　平成26年10月30日(2014.10.30)
審査請求日　平成29年4月14日(2017.4.14)
(31)優先権主張番号　61/815,052
(32)優先日　平成25年4月23日(2013.4.23)
(33)優先権主張国　米国(US)

(73)特許権者　509123208
アビニシオ　テクノロジー　エルエルシー
アメリカ合衆国　０２４２１　マサチューセッツ州　レキシントン　スプリング　ストリート　２０１
(74)代理人　100107984
弁理士　廣田　雅紀
(74)代理人　100102255
弁理士　小澤　誠次
(74)代理人　100096482
弁理士　東海　裕作
(74)代理人　100188352
弁理士　松田　一弘
(74)代理人　100131093
弁理士　堀内　真

最終頁に続く

(54)【発明の名称】コンピューティングシステムによって実行されるタスクの制御

(57)【特許請求の範囲】
【請求項１】
コンピューティングシステムによって実行されるタスクを制御するための方法であって、
複数のタスクの間の少なくとも部分的な順序を規定する順序情報を受信するステップと、
前記順序情報に少なくとも部分的に基づいて前記タスクの少なくとも一部を実行するための命令を少なくとも１つのプロセッサを用いて生成するステップとを含み、前記生成するステップが、
第１のタスクに対応する第１のサブルーチンを実行するための命令を記憶することであって、前記第１のサブルーチンが、第２のタスクに対応する少なくとも第２のサブルーチンの実行を制御する第１の制御セクションを含み、前記第１の制御セクションが、前記第２のタスクに関連する状態情報を変更し、変更された状態情報に基づいて前記第２のサブルーチンの実行を開始すべきか否かを判定するように構成された関数を含む、記憶すること、並びに
前記第２のサブルーチンを実行するための命令を記憶することであって、前記第２のサブルーチンが、前記第２のタスクを実行するためのタスクセクション、及び第３のタスクに対応する第３のサブルーチンの実行を制御する第２の制御セクションを含む、記憶することを含む、方法。
【請求項２】

前記順序情報が、それぞれのタスクを表すノードのペアの間の有向辺を含む制御フローグラフを含み、上流のノードから下流のノードへの有向辺が、前記部分的な順序で、前記上流のノードによって表される前記タスクが前記下流のノードによって表される前記タスクに先行することを示す請求項１に記載の方法。

【請求項３】

前記制御フローグラフが、前記第１のタスクを表す第１のノードと前記第２のタスクを表す第２のノードとの間の有向辺、及び前記第２のノードと前記第３のタスクを表す第３のノードとの間の有向辺を含む請求項２に記載の方法。

【請求項４】

前記関数が、前記第２のタスクに関連するカウンタをデクリメント又はインクリメントし、前記カウンタの値に基づいて前記第２のサブルーチンの実行を開始すべきか否かを判定するように構成される請求項１～３のいずれかに記載の方法。

【請求項５】

前記関数が、前記カウンタをアトミックにデクリメント又はインクリメントし、前記カウンタの値を読むアトミックな操作を実行するように構成される請求項４に記載の方法。

【請求項６】

前記変更された状態情報が、前記第２のタスクを特定する引数を用いた前記関数の前の呼び出しの履歴を含む請求項１～５のいずれかに記載の方法。

【請求項７】

前記関数が、複数の異なる関数のうちの１つであり、前記状態情報が、前記第２のタスクを特定する引数を用いた前記複数の異なる関数のいずれかの前の呼び出しの履歴を捕捉する請求項６に記載の方法。

【請求項８】

前記第２の制御セクションが、前記タスクを実行するための前記タスクセクションが呼び出されるか否かを判定する論理を含む請求項１～７のいずれかに記載の方法。

【請求項９】

前記論理が、前記第２のタスクに関連するフラグの値に基づいて、前記タスクを実行するための前記タスクセクションが呼び出されるか否かを判定する請求項８に記載の方法。

【請求項１０】

前記第１の制御セクションが、少なくとも、前記第２のタスクに対応する前記第２のサブルーチン及び第４のタスクに対応する第４のサブルーチンの実行を制御する請求項１～９のいずれかに記載の方法。

【請求項１１】

前記順序情報が、前記部分的な順序で前記第１のタスクが前記第２のタスクに先行することを示し、前記部分的な順序で前記第１のタスクが前記第４のタスクに先行することを示し、前記部分的な順序で互いに対する前記第２のタスク及び前記第４のタスクの順序を制約しない請求項１０に記載の方法。

【請求項１２】

前記第１の制御セクションが、前記第２のサブルーチンを実行するための新しいプロセスをスポーニングすべきか否かを判定する第１の関数と、第１のサブルーチンを実行した同じプロセスを用いて前記第４のサブルーチンの実行を開始する第２の関数とを含む請求項１０又は１１に記載の方法。

【請求項１３】

タスクを制御するための、コンピュータ可読記憶媒体に記憶されるコンピュータプログラムであって、コンピューティングシステムに請求項１～１２のいずれかに記載の方法を実行させる命令を含む、前記コンピュータプログラム。

【請求項１４】

タスクを制御するためのコンピューティングシステムであって、

順序情報を受信するように構成された入力デバイス又はポートと、

請求項１～１２のいずれかに記載の方法を実行するように構成された少なくとも１つの

プロセッサとを含む、前記コンピューティングシステム。
【請求項１５】
タスクを制御するためのコンピューティングシステムであって、
複数のタスクの間の少なくとも部分的な順序を規定する順序情報を受信するための手段と、
前記順序情報に少なくとも部分的に基づいて前記タスクの少なくとも一部を実行するための命令を生成するための手段とを含み、前記生成することが、
第１のタスクに対応する第１のサブルーチンを実行するための命令を記憶することであって、前記第１のサブルーチンが、第２のタスクに対応する少なくとも第２のサブルーチンの実行を制御する第１の制御セクションを含み、前記第１の制御セクションが、前記第２のタスクに関連する状態情報を変更し、変更された状態情報に基づいて前記第２のサブルーチンの実行を開始すべきか否かを判定するように構成された関数を含む、記憶すること、並びに
前記第２のサブルーチンを実行するための命令を記憶することであって、前記第２のサブルーチンが、前記第２のタスクを実行するためのタスクセクション、及び第３のタスクに対応する第３のサブルーチンの実行を制御する第２の制御セクションを含む、記憶することを含む、コンピューティングシステム。
【請求項１６】
タスクを実行するための方法であって、
複数のタスクを実行するための命令を少なくとも１つのメモリに記憶するステップであって、前記命令が、前記タスクの少なくとも一部のそれぞれに関して、前記タスクを実行するためのタスクセクション、及び後続のタスクのサブルーチンの実行を制御する制御セクションを含むそれぞれのサブルーチンを含む、記憶するステップと、
記憶されたサブルーチンの少なくとも一部を少なくとも１つのプロセッサによって実行するステップとを含み、前記実行するステップが、
第１のタスクに関する第１のサブルーチンを実行するための第１のプロセスをスポーニングすることであって、前記第１のサブルーチンが、第１のタスクセクション及び第１の制御セクションを含む、スポーニングすること、並びに
前記第１のタスクセクションが返った後に、前記第１のサブルーチンとは異なる第２のサブルーチンを実行するための第２のプロセスをスポーニングすべきか否かを判定する前記第１の制御セクションに含まれる少なくとも１つの関数を呼び出すことを含む、方法。
【請求項１７】
前記関数が、前記第２のサブルーチンに関連するカウンタをデクリメントし、前記カウンタの値に基づいて前記第２のサブルーチンの実行を開始すべきか否かを判定するように構成される請求項１６に記載の方法。
【請求項１８】
前記関数が、前記第２のサブルーチンに対応する第２のタスクを特定する引数を用いて呼び出されるときに、前記第２のタスクを特定する引数を用いた前記関数の前の呼び出しの履歴を捕捉する状態情報に基づいて前記第２のサブルーチンの実行を開始すべきか否かを判定するように構成される請求項１６又は１７に記載の方法。
【請求項１９】
前記関数が、複数の異なる関数のうちの１つであり、前記状態情報が、前記第２のタスクを特定する引数を用いた前記複数の異なる関数のいずれかの前の呼び出しの履歴を捕捉する請求項１８に記載の方法。
【請求項２０】
前記関数が、前記第１のプロセスにおいて前記第２のサブルーチンの実行を開始し、前記第２のサブルーチンの実行時間が所定の閾値を超えることに応じて、前記第２のサブルーチンの実行を続けるための前記第２のプロセスをスポーニングするように構成される請求項１６に記載の方法。

【請求項２１】
前記関数が、前記第１のプロセスに関連付けられたスタックフレームを前記第２のプロセスに与えるように構成される請求項２０に記載の方法。
【請求項２２】
前記関数が、前記第１のプロセスが前記第２のプロセスと同時に実行され続けることを可能にするために前記第２のプロセスをスポーニングした後に返すように構成される請求項２０又は２１に記載の方法。
【請求項２３】
前記第１の制御セクションが、前記第１のタスクセクションが呼び出されるか否かを判定する論理を含む請求項１６に記載の方法。
【請求項２４】
前記論理が、前記第１のタスクに関連するフラグの値に基づいて、前記第１のタスクセクションが呼び出されるか否かを判定する請求項２３に記載の方法。
【請求項２５】
前記第２のサブルーチンが、第２のタスクに対応し、前記第１の制御セクションが、前記第１のタスク及び前記第２のタスクを含む複数のタスクの間の少なくとも部分的な順序を規定する順序情報に少なくとも部分的に基づく請求項１６に記載の方法。
【請求項２６】
前記順序情報が、それぞれのタスクを表すノードのペアの間の有向辺を含む制御フローグラフを含み、上流のノードから下流のノードへの有向辺が、部分的な順序で、前記上流のノードによって表される前記タスクが前記下流のノードによって表される前記タスクに先行することを示す請求項２５に記載の方法。
【請求項２７】
タスクを実行するための、コンピュータ可読記憶媒体に記憶されるコンピュータプログラムであって、コンピューティングシステムに請求項１６～２６のいずれかに記載の方法を実行させる命令を含む、前記コンピュータプログラム。
【請求項２８】
タスクを実行するためのコンピューティングシステムであって、
複数のタスクを実行するための命令を記憶する少なくとも１つのメモリと、
請求項１６～２６のいずれかに記載の方法を実行するように構成された少なくとも１つのプロセッサとを含む、前記コンピューティングシステム。
【請求項２９】
タスクを実行するためのコンピューティングシステムであって、
複数のタスクを実行するための命令を記憶するための手段であって、前記命令が、前記タスクの少なくとも一部のそれぞれに関して、前記タスクを実行するためのタスクセクション、及び後続のタスクのサブルーチンの実行を制御する制御セクションを含むそれぞれのサブルーチンを含む、手段と、
記憶されたサブルーチンの少なくとも一部を実行するための手段とを含み、前記実行することが、
第１のタスクに関する第１のサブルーチンを実行するための第１のプロセスをスポーニングすることであって、前記第１のサブルーチンが、第１のタスクセクション及び第１の制御セクションを含む、スポーニングすること、並びに
前記第１のタスクセクションが返った後に、前記第１のサブルーチンとは異なる第２のサブルーチンを実行するための第２のプロセスをスポーニングすべきか否かを判定する前記第１の制御セクションに含まれる少なくとも１つの関数を呼び出すことを含む、コンピューティングシステム。

【発明の詳細な説明】
【技術分野】

【０００１】
関連出願の相互参照
本出願は、２０１３年４月２３日に出願した米国特許出願第６１／８１５，０５２号の優先権を主張するものである。
【０００２】
この説明は、コンピューティングシステムによって実行されるタスクの制御に関する。
【背景技術】
【０００３】
コンピューティングシステムによって実行されるタスクを制御するための一部の技術においては、個々のタスクはプロセス又はスレッドによって実行され、プロセス又はスレッドは、そのタスクのためにスポーニング（spawn）され、そのタスクが完了された後に終了する。コンピューティングシステムのオペレーティングシステム、又はオペレーティングシステムの機能を使用するその他の集中型の制御エンティティが、異なるタスクをスケジューリングし、又は異なるタスクの間の通信を管理するために使用される可能性がある。その他の下流のタスク（例えば、タスクＢ）が始まる前に完了しなければならない特定の上流のタスク（例えば、タスクＡ）を示すことによってタスクの部分的な順序を定義するために、制御フローグラフが使用される可能性がある。制御フローグラフに従ってタスクを実行するための新しいプロセスのスポーニングを管理する制御プロセスが、存在する可能性がある。制御プロセスは、タスクＡを実行するためのプロセスＡをスポーニングした後、プロセスＡが終了したというオペレーティングシステムによる通知を待つ。プロセスＡが終了した後、オペレーティングシステムが、制御プロセスに通知し、そして、制御プロセスが、タスクＢを実行するためのプロセスＢをスポーニングする。
【発明の概要】
【課題を解決するための手段】
【０００４】
一態様においては、概して、コンピューティングシステムによって実行されるタスクを制御するための方法が、複数のタスクの間の少なくとも部分的な順序を規定する順序情報を受信するステップと、順序情報に少なくとも部分的に基づいてタスクの少なくとも一部を実行するための命令を少なくとも１つのプロセッサを用いて生成するステップとを含む。生成するステップは、第１のタスクに対応する第１のサブルーチンを実行するための命令を記憶することであって、第１のサブルーチンが、第２のタスクに対応する少なくとも第２のサブルーチンの実行を制御する第１の制御セクション（control section）を含み、第１の制御セクションが、第２のタスクに関連する状態情報を変更し、変更された状態情報に基づいて第２のサブルーチンの実行を開始すべきか否かを判定するように構成された関数を含む、記憶すること、並びに第２のサブルーチンを実行するための命令を記憶することであって、第２のサブルーチンが、第２のタスクを実行するためのタスクセクション（task section）、及び第３のタスクに対応する第３のサブルーチンの実行を制御する第２の制御セクションを含む、記憶することを含む。
【０００５】
態様は、以下の特徴のうちの１又は２以上を含み得る。
【０００６】
順序情報は、それぞれのタスクを表すノードのペアの間の有向辺を含む制御フローグラフを含み、上流のノードから下流のノードへの有向辺が、部分的な順序で上流のノードによって表されるタスクが下流のノードによって表されるタスクに先行することを示す。
【０００７】
制御フローグラフは、第１のタスクを表す第１のノードと第２のタスクを表す第２のノードとの間の有向辺、及び第２のノードと第３のタスクを表す第３のノードとの間の有向辺を含む。
【０００８】
関数は、第２のタスクに関連するカウンタをデクリメント又はインクリメントし、カウ

ンタの値に基づいて第２のサブルーチンの実行を開始すべきか否かを判定するように構成される。

【０００９】

関数は、カウンタをアトミックにデクリメント又はインクリメントし、カウンタの値を読むアトミックな操作を実行するように構成される。

【００１０】

変更された状態情報は、第２のタスクを特定する引数を用いた関数の前の呼び出しの履歴を含む。

【００１１】

関数は、複数の異なる関数のうちの１つであり、状態情報は、第２のタスクを特定する引数を用いた複数の異なる関数のいずれかの前の呼び出しの履歴を捕捉する。

【００１２】

第２の制御セクションは、タスクを実行するためのタスクセクションが呼び出されるか否かを判定する論理を含む。

【００１３】

論理は、第２のタスクに関連するフラグの値に基づいて、タスクを実行するためのタスクセクションが呼び出されるか否かを判定する。

【００１４】

第１の制御セクションは、少なくとも、第２のタスクに対応する第２のサブルーチン及び第４のタスクに対応する第４のサブルーチンの実行を制御する。

【００１５】

順序情報は、部分的な順序で第１のタスクが第２のタスクに先行することを示し、部分的な順序で第１のタスクが第４のタスクに先行することを示し、部分的な順序で互いに対する第２のタスク及び第４のタスクの順序を制約しない。

【００１６】

第１の制御セクションは、第２のサブルーチンを実行するための新しいプロセスをスポーニングすべきか否かを判定する第１の関数と、第１のサブルーチンを実行した同じプロセスを用いて第４のサブルーチンの実行を開始する第２の関数とを含む。

【００１７】

別の態様においては、概して、コンピュータプログラムが、タスクを制御するために、コンピュータ可読記憶媒体に記憶される。コンピュータプログラムは、コンピューティングシステムに、複数のタスクの間の少なくとも部分的な順序を規定する順序情報を受信させ、順序情報に少なくとも部分的に基づいてタスクの少なくとも一部を実行するための命令を生成するための命令を含む。生成することは、第１のタスクに対応する第１のサブルーチンを実行するための命令を記憶することであって、第１のサブルーチンが、第２のタスクに対応する少なくとも第２のサブルーチンの実行を制御する第１の制御セクションを含み、第１の制御セクションが、第２のタスクに関連する状態情報を変更し、変更された状態情報に基づいて第２のサブルーチンの実行を開始すべきか否かを判定するように構成された関数を含む、記憶すること、並びに第２のサブルーチンを実行するための命令を記憶することであって、第２のサブルーチンが、第２のタスクを実行するためのタスクセクション、及び第３のタスクに対応する第３のサブルーチンの実行を制御する第２の制御セクションを含む、記憶することを含む。

【００１８】

別の態様においては、概して、タスクを制御するためのコンピューティングシステムが、複数のタスクの間の少なくとも部分的な順序を規定する順序情報を受信するように構成された入力デバイス又はポートと、順序情報に少なくとも部分的に基づいてタスクの少なくとも一部を実行するための命令を生成するように構成された少なくとも１つのプロセッサとを含む。生成することは、第１のタスクに対応する第１のサブルーチンを実行するための命令を記憶することであって、第１のサブルーチンが、第２のタスクに対応する少なくとも第２のサブルーチンの実行を制御する第１の制御セクションを含み、第１の制御セ

クションが、第２のタスクに関連する状態情報を変更し、変更された状態情報に基づいて第２のサブルーチンの実行を開始すべきか否かを判定するように構成された関数を含む、記憶すること、並びに第２のサブルーチンを実行するための命令を記憶することであって、第２のサブルーチンが、第２のタスクを実行するためのタスクセクション、及び第３のタスクに対応する第３のサブルーチンの実行を制御する第２の制御セクションを含む、記憶することを含む。

【００１９】

別の態様においては、概して、タスクを制御するためのコンピューティングシステムが、複数のタスクの間の少なくとも部分的な順序を規定する順序情報を受信するための手段と、順序情報に少なくとも部分的に基づいてタスクの少なくとも一部を実行するための命令を生成するための手段とを含む。生成することは、第１のタスクに対応する第１のサブルーチンを実行するための命令を記憶することであって、第１のサブルーチンが、第２のタスクに対応する少なくとも第２のサブルーチンの実行を制御する第１の制御セクションを含み、第１の制御セクションが、第２のタスクに関連する状態情報を変更し、変更された状態情報に基づいて第２のサブルーチンの実行を開始すべきか否かを判定するように構成された関数を含む、記憶すること、並びに第２のサブルーチンを実行するための命令を記憶することであって、第２のサブルーチンが、第２のタスクを実行するためのタスクセクション、及び第３のタスクに対応する第３のサブルーチンの実行を制御する第２の制御セクションを含む、記憶することを含む。

【００２０】

別の態様においては、概して、タスクを実行するための方法が、複数のタスクを実行するための命令を少なくとも１つのメモリに記憶するステップであって、命令が、タスクの少なくとも一部のそれぞれに関して、そのタスクを実行するためのタスクセクション、及び後続のタスクのサブルーチンの実行を制御する制御セクションを含むそれぞれのサブルーチンを含む、記憶するステップと、記憶されたサブルーチンの少なくとも一部を少なくとも１つのプロセッサによって実行するステップとを含む。実行することは、第１のタスクに関する第１のサブルーチンを実行するための第１のプロセスをスポーニングすることであって、第１のサブルーチンが、第１のタスクセクション及び第１の制御セクションを含む、スポーニングすること、並びに第１のタスクセクションが返った後に、第２のサブルーチンを実行するための第２のプロセスをスポーニングすべきか否かを判定する第１の制御セクションに含まれる少なくとも１つの関数を呼び出すことを含む。

【００２１】

態様は、以下の特徴のうちの１又は２以上を含み得る。

【００２２】

関数は、第２のサブルーチンに関連するカウンタをデクリメントし、カウンタの値に基づいて第２のサブルーチンの実行を開始すべきか否かを判定するように構成される。

【００２３】

関数は、第２のサブルーチンに対応する第２のタスクを特定する引数を用いて呼び出されるときに、第２のタスクを特定する引数を用いた関数の前の呼び出しの履歴を捕捉する状態情報に基づいて第２のサブルーチンの実行を開始すべきか否かを判定するように構成される。

【００２４】

関数は、複数の異なる関数のうちの１つであり、状態情報は、第２のタスクを特定する引数を用いた複数の異なる関数のいずれかの前の呼び出しの履歴を捕捉する。

【００２５】

関数は、第１のプロセスにおいて第２のサブルーチンの実行を開始し、第２のサブルーチンの実行時間が所定の閾値を超えることに応じて、第２のサブルーチンの実行を続けるための第２のプロセスをスポーニングするように構成される。

【００２６】

関数は、第１のプロセスに関連付けられたスタックフレームを第２のプロセスに与える

ように構成される。
【００２７】
関数は、第１のプロセスが第２のプロセスと同時に実行され続けることを可能にするために第２のプロセスをスポーニングした後に返るように構成される。
【００２８】
第１の制御セクションは、第１のタスクセクションが呼び出されるか否かを判定する論理を含む。
【００２９】
論理は、第１のタスクに関連するフラグの値に基づいて、第１のタスクセクションが呼び出されるか否かを判定する。
【００３０】
第２のサブルーチンは、第２のタスクに対応し、第１の制御セクションは、第１のタスク及び第２のタスクを含む複数のタスクの間の少なくとも部分的な順序を規定する順序情報に少なくとも部分的に基づく。
【００３１】
順序情報は、それぞれのタスクを表すノードのペアの間の有向辺を含む制御フローグラフを含み、上流のノードから下流のノードへの有向辺が、部分的な順序で上流のノードによって表されるタスクが下流のノードによって表されるタスクに先行することを示す。
【００３２】
別の態様においては、概して、コンピュータプログラムが、タスクを実行するために、コンピュータ可読記憶媒体に記憶される。コンピュータプログラムは、コンピューティングシステムに、複数のタスクを実行するための命令を記憶させ、命令が、タスクの少なくとも一部のそれぞれに関して、そのタスクを実行するためのタスクセクション、及び後続のタスクのサブルーチンの実行を制御する制御セクションを含むそれぞれのサブルーチンを含み、記憶されたサブルーチンの少なくとも一部を実行させる命令を含む。実行することは、第１のタスクに関する第１のサブルーチンを実行するための第１のプロセスをスポーニングすることであって、第１のサブルーチンが、第１のタスクセクション及び第１の制御セクションを含む、スポーニングすること、並びに第１のタスクセクションが返った後に、第２のサブルーチンを実行するための第２のプロセスをスポーニングすべきか否かを判定する第１の制御セクションに含まれる少なくとも１つの関数を呼び出すことを含む。
【００３３】
別の態様においては、概して、タスクを実行するためのコンピューティングシステムが、複数のタスクを実行するための命令を記憶する少なくとも１つのメモリであって、命令が、タスクの少なくとも一部のそれぞれに関して、そのタスクを実行するためのタスクセクション、及び後続のタスクのサブルーチンの実行を制御する制御セクションを含むそれぞれのサブルーチンを含む、少なくとも１つのメモリと、記憶されたサブルーチンの少なくとも一部を実行するように構成された少なくとも１つのプロセッサとを含む。実行することは、第１のタスクに関する第１のサブルーチンを実行するための第１のプロセスをスポーニングすることであって、第１のサブルーチンが、第１のタスクセクション及び第１の制御セクションを含む、スポーニングすること、並びに第１のタスクセクションが返った後に、第２のサブルーチンを実行するための第２のプロセスをスポーニングすべきか否かを判定する第１の制御セクションに含まれる少なくとも１つの関数を呼び出すことを含む。
【００３４】
別の態様においては、概して、タスクを実行するためのコンピューティングシステムが、複数のタスクを実行するための命令を記憶するための手段であって、命令が、タスクの少なくとも一部のそれぞれに関して、そのタスクを実行するためのタスクセクション、及び後続のタスクのサブルーチンの実行を制御する制御セクションを含むそれぞれのサブルーチンを含む、手段と、記憶されたサブルーチンの少なくとも一部を実行するための手段

とを含む。実行することは、第１のタスクに関する第１のサブルーチンを実行するための第１のプロセスをスポーニングすることであって、第１のサブルーチンが、第１のタスクセクション及び第１の制御セクションを含む、スポーニングすること、並びに第１のタスクセクションが返った後に、第２のサブルーチンを実行するための第２のプロセスをスポーニングすべきか否かを判定する第１の制御セクションに含まれる少なくとも１つの関数を呼び出すことを含む。

【００３５】

態様は、以下の利点のうちの１又は２以上を含む可能性がある。

【００３６】

タスクがコンピューティングシステムによって実行されるとき、タスクを実行するための新しいプロセスをスポーニングすることに関連し、タスクのプロセスと、スケジューラ、又はタスクの依存関係及び順序を維持するためのその他の中心的なプロセスとの間を行ったり来たり切り替えることに関連する処理時間のコストが存在する。本明細書において説明される技術は、計算効率の良い方法で、新しいプロセスが選択的にスポーニングされること、又は実行されているプロセスが選択的に再利用されることを可能にする。コンパイラは、タスクを実行するためのサブルーチンに追加される比較的少ない量のコードに基づく分散型のスケジューリングメカニズムによって、集中型のスケジューラにのみ頼る必要性を避けることができる。タスクの完了が、自動的に、同時実行及び条件付き論理を可能にする方法で、制御フローグラフなどの入力の制約に従ってコンピューティングシステムがその他のタスクを実行することを引き起こす。タスクに関連するコンパイラにより生成されたコードが、カウンタ及びフラグに記憶された状態情報に基づいてその他のタスクを実行すべきか否かを判定するための関数をランタイムで呼び出す。したがって、コンパイラによって生成されたコードが、タスクのサブルーチンの呼び出しをランタイムで制御する状態機械を効率的に実装している。スケジューラへの及びスケジューラからの切り替えの余計なオーバーヘッドなしに、コンピューティングシステムは、細かい粒度の潜在的に同時のタスクをより効率的に実行することができる。

【００３７】

本発明のその他の特徴及び利点は、以下の説明及び請求項から明らかになるであろう。

【図面の簡単な説明】

【００３８】

【図１】コンピューティングシステムのブロック図である。

【図２Ａ】制御フローグラフの図である。

【図２Ｂ－２Ｄ】図２Ａの制御フローグラフのノードに関するサブルーチンの実行に関連するプロセスの存続期間の図である。

【図３－４】制御フローグラフの図である。

【発明を実施するための形態】

【００３９】

図１は、タスク制御技術が使用され得るコンピューティングシステム１００の例を示す。システム１００は、タスクの仕様１０４を記憶するための記憶システム１０２と、タスクの仕様を、タスクを実行するためのタスクサブルーチンにコンパイルするためのコンパイラ１０６と、メモリシステム１１０にロードされたタスクサブルーチンを実行するための実行環境１０８とを含む。それぞれのタスクの仕様１０４は、どのタスクが実行されることになるか、及び異なるタスクの間の順序の制約を含む、いつそれらのタスクが実行され得るかに関する制約についての情報を符号化する。タスクの仕様１０４の一部は、実行環境１０８のユーザインターフェース１１４を介してユーザ１１２がインタラクションすることによって構築され得る。実行環境１０８は、例えば、ＵＮＩＸオペレーティングシステムのバージョンなどの好適なオペレーティングシステムの制御下の１又は２以上の汎用コンピュータでホストされる可能性がある。例えば、実行環境１０８は、ローカルの（例えば、対称型マルチプロセッシング（ＳＭＰ，symmetric multi-processing）コンピュータなどのマルチプロセッサシステム）又はローカルに分散された（例えば、クラスタ若

しくは超並列処理（ＭＰＰ，massively parallel processing）システムとして接続された複数のプロセッサ）か、或いは遠隔の又は遠隔に分散された（例えば、ローカルエリアネットワーク（ＬＡＮ，local area network）及び／若しくは広域ネットワーク（ＷＡＮ，wide-area network）を介して接続された複数のプロセッサ）か、或いはこれらの任意の組合せかのいずれかの複数の中央演算処理装置（ＣＰＵ，central processing unit）或いはプロセッサコアを用いるコンピュータシステムの構成を含むマルチノード並列コンピューティング環境を含む可能性がある。記憶システム１０２を提供する記憶デバイスは、実行環境１０８のローカルにあり、例えば、実行環境１０８をホストするコンピュータに接続された記憶媒体（例えば、ハードドライブ）に記憶される可能性があり、又は実行環境１０８の遠隔にあり、例えば、（例えば、クラウドコンピューティングインフラストラクチャによって提供される）リモート接続を介して実行環境１０８をホストするコンピュータと通信するリモートシステムでホストされる可能性がある。

【００４０】

図２Ａは、コンピューティングシステム１００によって実行される１組のタスクに対して課される部分的な順序を定義する制御フローグラフ２００の例を示す。制御フローグラフ２００によって定義される部分的な順序は、記憶されるタスクの仕様１０４で符号化される。一部の実施形態において、ユーザ１１２は、制御フローグラフに含まれるさまざまな種類のノードを選択し、接続されたノードの間の順序の制約を表すリンクによってそれらのノードの一部を接続する。ノードの１つの種類は、図２Ａにおいて角の四角いブロックによって表されるタスクノードである。それぞれのタスクノードは、実行される異なるタスクを表す。（有向リンクの始点の）第１のタスクノードから（有向リンクの終点の）第２のタスクノードに接続された有向リンクは、第２のノードのタスクが開始し得るよりも前に第１のノードのタスクが完了しなければならないという順序の制約を課す。ノードの別の種類は、図２Ａにおいて角の丸いブロックによって表される接合ノードである。制御フローグラフが条件付きの動作（behavior）を含まない場合、接合ノードは、単に、順序の制約を課すように働く。単一の入力リンク及び複数の出力リンクを有する接合ノードは、出力リンクによって接続されるタスクノードのいずれのタスクが開始し得るよりも前に入力リンクによって接続されるタスクノードのタスクが完了しなければならないように順序の制約を課す。複数の入力リンク及び単一の出力リンクを有する接合ノードは、出力リンクによって接続されるタスクノードのタスクが開始し得るよりも前に入力リンクによって接続されるタスクノードのすべてのタスクが完了しなければならないように順序の制約を課す。タスクノードは、複数の入力リンクの終点でもあり、そのタスクノードのタスクが開始し得るよりも前に入力リンクによって接続されるタスクノードのすべてのタスクが完了しなければならないように順序の制約を課す可能性がある。以下でより詳細に説明されるように、条件付きの動作によって、複数の入力リンクを有するタスクノードは、複数の入力を有する接合ノードとは異なる論理的な動作をさらに提供する。

【００４１】

制御フローグラフが構築された後、コンパイラ１０６は、タスク情報と、その制御フローグラフによって表される順序の情報とを符号化するタスクの仕様１０４をコンパイルし、タスクを実行するための命令を生成する。命令は、実行される用意のできている低レベルのマシンコードの形態、又は最終的に実行される低レベルのマシンコードを提供するためにさらにコンパイルされるより高レベルのコードの形態である可能性がある。生成された命令は、それぞれのタスクノードに関するサブルーチン（「タスクサブルーチン」）及びそれぞれの接合ノードに関するサブルーチン（「接合サブルーチン」）を含む。タスクサブルーチンのそれぞれは、対応するタスクを実行するための（タスク本体（task body）とも呼ばれる）タスクセクションを含む。タスクノードは、コンパイラが適切なタスクセクションを生成することができるように実行される対応するタスクの何らかの説明を含む。例えば、一部の実施形態において、タスクノードは、呼び出される特定の関数、実行されるプログラム、又はタスクセクションに含められるその他の実行可能コードを特定する。また、タスクサブルーチンの一部は、制御フローグラフの別のノードに関する後続の

サブルーチンの実行を制御する制御セクションを含み得る。いかなる下流のノードにも接続されないタスクノードに関するタスクサブルーチンは、そのタスクノードの完了後に制御がいかなる後続のタスクにも渡される必要がないので制御セクションを必要としない可能性がある。接合ノードの目的は、制御のフローに対する制約を規定することであるので、接合サブルーチンのそれぞれは、制御セクションをその主要部として含む。

【００４２】

制御セクションに含まれる関数の例は、制御フローグラフのノードに関連する状態情報に基づいて後続のノードに関するサブルーチンを実行するための新しいプロセスをスポーニングすべきか否かを判定する「ｃｈａｉｎ」関数である。ｃｈａｉｎ関数の引数が、その後続のノードを特定する。下の表は、制御フローグラフ２００のノードのそれぞれに関してコンパイラによって書かれるサブルーチンに含まれる関数の例を示し、タスクサブルーチンのタスクセクションが、関数呼び出しＴ＃（）によって表され、サブルーチンの残りが、制御セクションを表すと考えられる。（その他の例において、タスクセクションは、最後の関数が返った後にタスクが完了されるようにして複数の関数呼び出しを含む可能性がある。）接合サブルーチンは、タスクセクションを含まず、したがって、すべて制御セクションによって構成される。この例においては、別々の関数呼び出しは、それらの関数が呼び出される順序でセミコロンによって分けられる。

【００４３】

【表１】

ノード	サブルーチン
タスクノードT1	T1(); chain(J1)
接合ノードJ1	chain (T2); chain(T3)
タスクノードT2	T2(); chain(J2)
タスクノードT3	T3(); chain(J2)
接合ノードJ2	chain(T4)
タスクノードT4	T4()

表1

【００４４】

タスクの仕様１０４がコンパイルされた後、コンピューティングシステム１００は、生成されたサブルーチンを実行環境１０８のメモリシステム１１０にロードする。特定のサブルーチンが呼び出されるとき、プログラムカウンタが、サブルーチンが記憶されるメモリシステム１１０のアドレス空間の一部の始めの対応するアドレスに設定される。

【００４５】

スケジューリングされた時間に、又はユーザ入力若しくは所定のイベントに応じて、コンピューティングシステム１００は、制御フローグラフのルートを表すロードされたサブルーチンのうちの少なくとも１つの実行を開始する。例えば、制御フローグラフ２００に関して、コンピューティングシステム１００は、タスクノードＴ１に関するタスクサブルーチンを実行するためのプロセスをスポーニングする。サブルーチンが実行を開始するとき、プロセスは、最初に、タスクノードＴ１のタスクを実行するためのタスクセクションを呼び出し、それから、タスクセクションが返った（タスクノードＴ１のタスクの完了を示す）後に、プロセスは、サブルーチンの制御セクションのｃｈａｉｎ関数を呼び出す。特定のノードに関するサブルーチンを実行するために新しいプロセスをスポーニングすべ

きか否かを判定するためにｃｈａｉｎ関数によって使用される状態情報は、以下でより詳細に説明されるように、その特定のノードを引数として呼び出された前のｃｈａｉｎ関数の履歴を捕捉する情報である。

【００４６】

この履歴情報は、異なるノードに関連するアクティブ化カウンタに保有され得る。カウンタの値は、例えば、メモリシステム１１０の一部又はその他の作業用記憶域に記憶される可能性がある。第１のプロセスがスポーニングされる前に、各ノードに関するアクティブ化カウンタの値は、そのノードへの入力リンクの数に初期化される。したがって、制御フローグラフ２００に関しては、以下の値に初期化された６つのアクティブ化カウンタが存在する。

【００４７】

【表２】

ノード	アクティブ化カウンタ値
タスクノードT1	0
接合ノードJ1	1
タスクノードT2	1
タスクノードT3	1
接合ノードJ2	2
タスクノードT4	1

表２

【００４８】

タスクノードＴ１は入力リンクを持たないので、そのタスクノードＴ１のアクティブ化カウンタは、ゼロに初期化される。代替的に、いかなる入力リンクも持たない最初のノードに関しては、そのノードに関連するアクティブ化カウンタが存在する必要がない。入力リンクを介して接続される異なるノードの制御セクションは、下流のリンクされたノードのアクティブ化カウンタをデクリメントし、デクリメントされた値に基づいてアクションを決定する。一部の実施形態においては、カウンタにアクセスする関数が、カウンタをアトミックにデクリメントし、デクリメント操作の前か又は後かのどちらかでカウンタの値を読むアトミックな操作（例えば、アトミックな「ｄｅｃｒｅｍｅｎｔ－ａｎｄ－ｔｅｓｔ」操作）を使用することができる。一部のシステムにおいては、そのような操作が、システムのネイティブの命令によってサポートされる。代替的に、カウンタの値がゼロに達するまでカウンタをデクリメントする代わりに、カウンタが、ゼロで始まる可能性があり、関数が、（例えば、アトミックな「ｉｎｃｒｅｍｅｎｔ－ａｎｄ－ｔｅｓｔ」操作を用いて）カウンタの値がノードへの入力リンクの数に初期化された所定の閾値に達するまでカウンタをインクリメントする可能性がある。

【００４９】

ｃｈａｉｎ関数「ｃｈａｉｎ（Ｎ）」の呼び出しは、ノードＮのアクティブ化カウンタをデクリメントし、デクリメントされた値がゼロである場合、ｃｈａｉｎ関数は、新しくスポーニングされたプロセスによるノードＮのサブルーチンの実行をトリガし、それから返す。デクリメントされた値がゼロよりも大きい場合、ｃｈａｉｎ関数は、新しいサブルーチンの実行をトリガすること又は新しいプロセスのスポーニングすることなしにただ返す。サブルーチンの制御セクションは、表１の接合ノードＪ１に関する接合サブルーチン

と同様に、ｃｈａｉｎ関数の複数の呼び出しを含む可能性がある。制御セクションの最後の関数が返った後、サブルーチンを実行するプロセスは、終了する可能性があり、又は一部の関数呼び出しに関しては（例えば、以下で説明される「ｃｈａｉｎＴｏ」関数呼び出しに関しては）、プロセスは、別のサブルーチンの実行を続ける。新しいプロセスのこの条件付きのスポーニングは、新しいプロセスのスポーニングを管理するためのスケジューラプロセスへの切り替え及び新しいプロセスのスポーニングを管理するためのスケジューラプロセスからの切り替えを必要とせずに、所望の部分的な順序に従って（潜在的に同時に）タスクサブルーチンが実行されることを可能にする。

【００５０】

表１のサブルーチンに関して、タスクサブルーチンＴ１に関するタスクセクションが返った後のｃｈａｉｎ関数の呼び出し「ｃｈａｉｎ（Ｊ１）」は、ノードＪ１に関するアクティブ化カウンタが１から０にデクリメントされる結果をもたらし、ｃｈａｉｎ関数の呼び出し「ｃｈａｉｎ（Ｔ２）」及び「ｃｈａｉｎ（Ｔ３）」を含む接合サブルーチンの実行を引き起こす。これらの呼び出しのそれぞれは、ノードＴ２及びＴ３に関するそれぞれのアクティブ化カウンタが１から０にデクリメントされる結果をもたらし、ノードＴ２及びＴ３に関するタスクサブルーチンの実行を引き起こす。両方のタスクサブルーチンが、ノードＪ２に関するアクティブ化カウンタをデクリメントする「ｃｈａｉｎ（Ｊ２）」を呼び出す制御セクションを含む。ノードＴ２及びＴ３に関するタスク本体のうち最初に終了する方が、ノードＪ２に関するアクティブ化カウンタを２から１にデクリメントするｃｈａｉｎ関数の呼び出しにつながる。２番目に終了するタスクセクションは、ノードＪ２に関するアクティブ化カウンタを１から０にデクリメントするｃｈａｉｎ関数の呼び出しにつながる。したがって、タスクのうちで最後に完了するもののみが、ノードＪ２に関する接合サブルーチンの実行を引き起こし、その実行が、ｃｈａｉｎ関数の最後の呼び出し「ｃｈａｉｎ（Ｔ４）」及びノードＴ４に関するアクティブ化カウンタの１から０へのデクリメントにつながり、そのデクリメントが、ノードＴ４に関するタスクサブルーチンの実行を開始する。ノードＴ４に関するタスクセクションが返った後、制御フローは、ノードＴ４に関するタスクサブルーチンに関する制御セクションが存在しないので完了する。

【００５１】

表１のサブルーチンの例においては、新しいプロセスが、制御フローグラフ２００の各ノードのサブルーチンに関してスポーニングされる。中心的なタスク監視又はスケジューリングプロセスを必要とせずに、新しいプロセスをスポーニングすべきか否かを判定する制御セクションを独自に含む各プロセスのサブルーチンによっていくらかの効率が得られるが、制御セクションに対する特定のコンパイラの最適化によってより一層の効率が得られる可能性がある。例えば、１つのコンパイラの最適化では、第１のサブルーチンの制御セクションにｃｈａｉｎ関数の単一の呼び出しが存在する場合、次のサブルーチン（すなわち、そのｃｈａｉｎ関数の引数）が、第１のサブルーチンを実行している同じプロセス内で（アクティブ化カウンタがゼロに達するときに）実行される可能性がある――新しいプロセスは、スポーニングされる必要がない。これを実現する１つの方法は、コンパイラがノードの最後の出力リンクに関する異なる関数呼び出し（例えば、「ｃｈａｉｎ」関数の代わりに「ｃｈａｉｎＴｏ」関数）を明示的に生成することである。ｃｈａｉｎＴｏ関数は、アクティブ化カウンタがゼロであるときにその関数の引数のサブルーチンを実行するための新しいプロセスをスポーニングする代わりに、その関数の引数のサブルーチンを同じプロセスに実行させることを除いてｃｈａｉｎ関数と同様である。ノードが単一の出力リンクを有する場合、そのノードのコンパイルされたサブルーチンは、ｃｈａｉｎＴｏ関数の単一の呼び出しを有する制御セクションを有する。ノードが複数の出力リンクを有する場合、そのノードのコンパイルされたサブルーチンは、ｃｈａｉｎ関数の１又は２以上の呼び出し及びｃｈａｉｎＴｏ関数の単一の呼び出しを有する制御セクションを有する。これは、独立したプロセスでスポーニングされるサブルーチンの数及びそれらのサブルーチンの関連する開始オーバーヘッドを削減する。表３は、このコンパイラの最適化を用いて制御フローグラフ２００に関して生成されるサブルーチンの例を示す。

【００５２】
【表３】

ノード	サブルーチン
タスクノードT1	T1(); chainTo(J1)
接合ノードJ1	chain(T2); chainTo(T3)
タスクノードT2	T2(); chainTo(J2)
タスクノードT3	T3(); chainTo(J2)
接合ノードJ2	chainTo(T4)
タスクノードT4	T4()

表３

【００５３】
表３のサブルーチンの例においては、第１のプロセスがノードＴ１及びＪ１のサブルーチンを実行し、そして、ノードＴ２のサブルーチンを実行するために新しいプロセスがスポーニングされる一方、第１のプロセスはノードＴ３のサブルーチンの実行を続ける。これら２つのプロセスのうちでそれらのプロセスのそれぞれのタスクセクションから始めに返すものは、接合ノードＪ２のアクティブ化カウンタを（２から１へ）始めにデクリメントするものであり、それからそのプロセスは、終了する。プロセスのタスクセクションから返る第２のプロセスは、接合ノードＪ２のアクティブ化カウンタを１から０にデクリメントし、それから、関数呼び出し（「ｃｈａｉｎＴｏ（Ｔ４）」）である接合ノードＪ２のサブルーチン及び最後にはタスクノードＴ４のサブルーチンを実行することによって継続する。図２Ｂは、ノードＴ３のタスクがノードＴ２のタスクの前に終了する場合に関して、第１の及び第２のプロセスが制御フローグラフ２００の異なるノードに関するサブルーチンを実行するときのそれらのプロセスの存続期間の例を示す。プロセスを表す線に沿った点は、（破線によって点に接続された）異なるノードに関するサブルーチンの実行に対応する。点の間の線分の長さは、必ずしも経過時間に比例せず、単に、異なるサブルーチンが実行される相対的な順序と、新しいプロセスがスポーニングされる時点とを示すように意図されている。
【００５４】
潜在的に効率をさらに向上させることができる修正の別の例は、特定のサブルーチンが同時実行の恩恵を受け得ることを示す閾値が満たされるまで遅らされた新しいプロセスのスポーニングである。異なるプロセスによる複数のサブルーチンの同時実行は、サブルーチンのそれぞれが完了するのにかなりの量の時間がかかる場合に特に恩恵がある。そうではなく、サブルーチンのいずれかがその他のサブルーチンと比べて完了するのに比較的少ない量の時間がかかる場合、そのサブルーチンは、効率の大きな損失なしに別のサブルーチンと逐次的に実行される可能性がある。遅らされたスポーニングのメカニズムは、かなりの量の時間がかかり、一緒に実行され得る複数のタスクが、プロセスを同時に実行することによって実行されることを可能にするが、さらに、より短いタスクのための新しいプロセスのスポーニングを抑制するように試みる。
【００５５】
遅らされたスポーニングを用いるｃｈａｉｎ関数の代替的な実施形態においては、ｃｈａｉｎＴｏ関数に似たｃｈａｉｎ関数が、その関数の引数のサブルーチンの実行を同じプロセスに開始させる。しかし、ｃｈａｉｎＴｏ関数とは異なり、タイマーが、サブルーチ

ンを実行するのにかかる時間を追跡し、閾値の時間が超えられる場合、ｃｈａｉｎ関数は、サブルーチンの実行を継続するために新しいプロセスをスポーニングする。第１のプロセスは、サブルーチンが完了したかのように継続することができ、第２のプロセスは、第１のプロセスが終了した時点でサブルーチンの実行を引き継ぐことができる。これを実現するために使用され得る１つのメカニズムは、第２のプロセスが第１のプロセスからサブルーチンのスタックフレームを継承することである。実行されているサブルーチンに関するスタックフレームは、特定の命令を指すプログラムカウンタと、サブルーチンの実行に関連するその他の値（例えば、ローカル変数及びレジスタ値）を含む。この例において、Ｔ２に関するタスクサブルーチンのスタックフレームは、プロセスがＴ２に関するタスクサブルーチンの完了後にＪ１に関する接合サブルーチンに返すことを可能にするリターンポインタ（return pointer）を含む。遅らされたスポーニングのタイマーが超えられるとき、新しいプロセスがスポーニングされ、Ｔ２に関するタスクサブルーチンの既存のスタックフレームに関連付けられ、第１のプロセスが、（「ｃｈａｉｎＴｏ（Ｔ３）」を呼び出すために）Ｊ１に関する接合サブルーチンに直ちに戻る。継承されたスタックフレームのリターンポインタは、新しいプロセスがＴ２に関するタスクサブルーチンの完了後にＪ１に関する接合サブルーチンに戻る必要がないので取り除かれる（つまり、ヌルにされる）。したがって、遅らされたスポーニングは、タスクが（構成可能な閾値に比して）迅速である場合に関して、新しいプロセスをスポーニングするオーバーヘッドなしに後続のタスクに関するサブルーチンが実行されることを可能にし、既存のスタックフレームを継承することによって、タスクがより長い場合に関して、新しいプロセスのスポーニングにかかわるオーバーヘッドを削減する。

【００５６】

図２Ｃは、ノードＴ２のタスクが遅らされたスポーニングの閾値よりも長い場合に関して、第１の及び第２のプロセスが制御フローグラフ２００の異なるノードに関するサブルーチンを実行するときのそれらのプロセスの存続期間の例を示す。スポーニングの閾値が達せられるとき、プロセス１が、ノードＴ２のタスクを実行するサブルーチンのスタックフレームを継承し、そのサブルーチンの実行を続けるプロセス２をスポーニングする。この例においては、（プロセス１によって実行される）ノードＴ３のタスクが、（プロセス１によって開始され、プロセス２によって完了される）ノードＴ２のタスクが終了する前に終了する。したがって、この例において、Ｊ２のアクティブ化カウンタを２から１にデクリメントする（そして終了する）のはプロセス１であり、Ｊ２のアクティブ化カウンタを１から０にデクリメントし、プロセス２がタスクノードＴ４のタスクを実行する結果となるのはプロセス２である。この例においては、ノードＴ２のタスク及びノードＴ３のタスクの同時実行が、そのような同時実行が全体的な効率に寄与すると判定された後に可能にされる。

【００５７】

図２Ｄは、ノードＴ２のタスクが遅らされたスポーニングの閾値よりも短い場合に関して、単一のプロセスが制御フローグラフ２００の異なるノードに関するサブルーチンを実行するときのそのプロセスの存続期間の例を示す。この例においては、（プロセス１によって実行される）ノードＴ２のタスクが、（プロセス１によってやはり実行される）ノードＴ３のタスクの前に終了する。したがって、この例においては、プロセス１が、ノードＴ２のタスクを完了した後にＪ２のアクティブ化カウンタを２から１にデクリメントし、プロセス１が、ノードＴ３のタスクを完了した後にＪ２のアクティブ化カウンタを１から０にデクリメントし、その結果、プロセス１が、タスクノードＴ４のタスクを実行する。この例においては、ノードＴ２及びノードＴ３のタスクの同時実行が、ノードＴ２のタスクが迅速に完了され得ると判定された後に第２のプロセスをスポーニングする必要性を避けることによって得られる効率のために放棄される。

【００５８】

制御フローグラフに含まれる可能性があるノードの別の種類は、図３に示される制御フローグラフ３００の円によって表される条件ノードである。条件ノードは、条件ノードの

出力に接続されたタスクノードのタスクが実行されるべきであるか否かを判定するための条件を定義する。ランタイムで、定義された条件が真である場合、制御フローは、その条件ノードを過ぎて下流に進むが、ランタイムで、定義された条件が偽である場合、制御フローは、条件ノードを過ぎて進まない。条件が偽である場合、条件ノードの下流のタスクノードのタスクは、それらのタスクノードに至る制御フローグラフを抜ける（その他の偽の条件ノードによってそれら自体が遮断されない）その他の経路が存在する場合にのみ実行される。

【００５９】

コンパイラは、それぞれの条件ノードに関する「条件サブルーチン」を生成し、さらに、条件ノードによって定義された条件を用いて条件ノードの下流の特定のその他のノードのサブルーチンを修正する。例えば、コンパイラは、制御フローグラフによって定義された制御の流れに従うためにランタイムで適用される「スキップメカニズム」のための命令を生成し得る。スキップメカニズムにおいて、各ノードは、（もしあれば）対応するタスクセクションが実行されるか否かを制御する関連する「スキップフラグ」を有する。スキップフラグが設定される場合、（ノードが「抑制」状態であるようにして）タスクセクションの実行が抑制され、この抑制は、コンパイラによって制御セクションに入れられた適切な制御コードによってその他のタスクに伝播され得る。前の例においては、タスクサブルーチンのタスクセクションが、制御セクションに先行していた。以下の例においては、一部のタスクのサブルーチンの制御セクションが、タスクセクションの前に現れる制御コード（「プロローグ（prologue）」とも呼ばれる）と、タスクセクションの後に現れる制御コード（「エピローグ（epilogue）」とも呼ばれる）とを含む。例えば、このスキップメカニズムを実装するために、コンパイラは、条件付きの命令（例えば、ステートメント（statement）の場合）と、下流のノードを特定する引数を用いて呼び出される「ｓｋｉｐ関数」の呼び出しとをプロローグ（すなわち、タスクセクションの前に実行されるコード）に含める。コンパイラは、ｃｈａｉｎ又はｃｈａｉｎＴｏ関数の呼び出しをエピローグ（すなわち、タスクセクションの後に実行されるコード）に含める。場合によっては、スキップフラグの値によって表される記憶された状態が原因で、プロローグのみが実行される可能性があり、タスクセクションとエピローグとの両方がスキップされる可能性がある。表４は、制御フローグラフ３００に関して生成されるサブルーチンの例を示す。

【００６０】

【表４】

ノード	サブルーチン
タスクノードT1	T1(); chainTo(J1)
接合ノードJ1	chain(C1); chain(C2); chainTo(J3)

条件ノードC1	if (<condition1>)
	chainTo(T2)
	else
	skip(T2)

条件ノードC2	if (<condition2>)
	chainTo(T3)
	else
	skip(T3)

タスクノードT2	if (skip)
	skip(J2)
	else
	T2(); chainTo(J2)

タスクノードT3	if (skip)
	skip(J2)
	else
	T3(); chainTo(J2)

接合ノードJ2	if (skip)
	skip(T4)
	else
	chainTo(T4)

タスクノードT4	if (skip)
	skip(J3)
	else
	T4(); chainTo(J3)

接合ノードJ3	if (skip)
	skip(T5)
	else
	chainTo(T5)

タスクノードT5	if (skip)
	return
	else
	T5()

表４

【００６１】

ｃｈａｉｎ及びｃｈａｉｎＴｏ関数と同様に、ｓｋｉｐ関数「ｓｋｉｐ（Ｎ）」は、そのｓｋｉｐ関数の引数（ノードＮ）のアクティブ化カウンタをデクリメントし、デクリメントされた値が０である場合、対応するサブルーチンを実行する。この例において、ｓｋｉｐ関数は、新しいプロセスをスポーニングすることなく同じプロセスを使用し続けることによってｃｈａｉｎＴｏ関数の動作に従うが、コンパイラは、同様の方法でタスクのスポーニングを制御するためにｃｈａｉｎ及びｃｈａｉｎＴｏ関数がするのと同じように振る舞うｓｋｉｐ関数の２つのバージョンを使用する可能性がある。コンパイラは、サブルーチンが実行されているノードのスキップフラグが設定される（つまり、ブール真値と評価される）場合、タスクセクションを呼び出すことなく下流のノードのｓｋｉｐを呼び出し、スキップフラグがクリアされる（つまり、ブール偽値と評価される）場合、（ノード

がタスクノードである場合）タスクセクションを確かに呼び出し、下流のノードのｃｈａｉｎを呼び出すように条件付きノードの下流のノードに関するサブルーチンを生成する。代替的に、コンパイラは、どのノードが条件付きノードの下流にあるかを判定する必要なしにデフォルトでサブルーチンの制御セクションに条件文を含める可能性がある。特に、スキップフラグを調べる「ｉｆ」文が、コンパイラがその文を含めるべきか否かを判定する必要なしにあらゆるノードのサブルーチンに関してデフォルトで含められる可能性がある（ただし、それは、スキップフラグの不必要な検査を招く可能性がある）。

【００６２】

制御フローグラフに条件ノードが存在する場合、複数の入力を有するノードは、ノードの種類に依存する論理的動作をランタイムで取得する。複数の入力リンク及び単一の出力リンクを有する接合ノードは、出力リンクによって接続される出力ノードがそのノードのサブルーチンを（ｓｋｉｐの呼び出しではなく）ｃｈａｉｎの呼び出しの引数にすべきである場合、入力リンクによって接続される少なくとも１つの入力ノードがそのノードのサブルーチンに（ｓｋｉｐの呼び出しではなく）ｃｈａｉｎの呼び出しを実行させなければならないように論理「ＯＲ」演算に対応する。複数の入力リンク及び単一の出力リンクを有するタスクノードは、そのタスクノードのサブルーチンが（ｓｋｉｐの呼び出しではなく）ｃｈａｉｎの呼び出しの引数であるべきである場合、入力リンクによって接続される入力ノードのすべてがそれらのノードのサブルーチンに（ｓｋｉｐの呼び出しではなく）ｃｈａｉｎの呼び出しを実行させなければならないように論理「ＡＮＤ」演算に対応する。

【００６３】

この論理的動作を確実にするために、ノードに関連するスキップフラグは、所定の規則に従ってランタイムで設定され、クリアされる。スキップフラグの初期値は、制御フローグラフのノードのサブルーチンのいずれかの実行の前に行われる初期化フェーズの間に与えられ、さらに、ノードの種類に依存する。また、コンパイラは、ノードの種類に依存する異なる動作を有するｓｋｉｐ関数及びｃｈａｉｎ関数の異なるバージョンを用いる。ノードＮのスキップフラグを変更するための所定の１組の規則と、コンパイラによって使用される異なるバージョンの関数の動作との一例は、以下の通りである。

・複数入力接合ノード（ＯＲ演算）に関して、スキップフラグは、最初に設定され、ｓｋｉｐ＿ＯＲ（Ｎ）は、スキップフラグを変更せず、ｃｈａｉｎ＿ＯＲ（Ｎ）は、スキップフラグをクリアする

・複数入力タスクノード（ＡＮＤ演算）に関して、スキップフラグは、最初にクリアされ、ｓｋｉｐ＿ＡＮＤ（Ｎ）は、スキップフラグを設定し、ｃｈａｉｎ＿ＡＮＤ（Ｎ）は、スキップフラグを変更しない

・単一入力ノードに関して、スキップフラグは、最初に設定され、ｓｋｉｐ（Ｎ）は、スキップフラグを変更せず、ｃｈａｉｎ（Ｎ）は、スキップフラグをクリアする

ｃｈａｉｎＴｏ関数の動作は、スキップフラグに関してｃｈａｉｎ関数と同じである。単一入力ノードに関して、ＯＲ演算及びＡＮＤ演算の動作は等価であり、（この例のＯＲ演算の動作など）どちらかが使用され得る。この１組の規則に関して、（１又は２以上の）開始ノード（すなわち、いかなる入力リンクもないノード）は、それらのノードのスキップフラグを（そのスキップフラグの初期値がまだクリアされていない場合）クリアする。

【００６４】

制御フローグラフ３００に関して、ノードＣ１に関する条件が真であり、ノードＣ２に関する条件が偽であり、ノードＣ２の条件の検査が完了される前にノードＴ３に関するタスクが終了する、つまり、ノードＴ３に関するサブルーチンが、ノードＪ２のスキップフラグをクリアし、ノードＪ２に関するアクティブ化カウンタを（２から１に）デクリメントする（ｓｋｉｐ論理とは対照的な）ｃｈａｉｎ論理に従い、それから、ノードＴ４に関するサブルーチンが、（スキップフラグを変更しない）ｓｋｉｐ論理に従い、ノードＪ２に関するアクティブ化カウンタを（１から０に）デクリメントし、そのことが、ノードＪ２のスキップフラグがノードＴ３のサブルーチンによってクリアされたのでｃｈａｉｎ（

Ｔ５）を引き起こす場合を考える。

【００６５】

その他の規則も、あり得る。ノードＮのスキップフラグを変更するための所定の１組の規則と、コンパイラによって使用される異なるバージョンの関数の動作との別の例は、以下の通りである。

・接合ノードに関して、スキップフラグは、最初に設定され、ｓｋｉｐ＿Ｊ（Ｎ）は、スキップフラグを変更せず、ｃｈａｉｎ＿Ｊ（Ｎ）は、スキップフラグをクリアする

・タスクノード又は条件付きノードに関して、スキップフラグは、最初にクリアされ、ｓｋｉｐ（Ｎ）は、スキップフラグを設定し、ｃｈａｉｎ（Ｎ）は、スキップフラグを変更しない

この１組の規則に関して、（１又は２以上の）開始ノード（すなわち、いかなる入力リンクもないノード）は、それらのノードのスキップフラグを（そのスキップフラグの初期値がまだクリアされていない場合）やはりクリアする。

【００６６】

コンパイラは、制御フローグラフの分析に基づいてサブルーチンの制御セクションの条件文又はその他の命令のさまざまな最適化を実行していてもよい可能性がある。例えば、制御フローグラフ３００から、タスクノードＴ５のタスクは、最終的にタスクノードＴ５のタスクの実行を引き起こす接合ノードＪ１と接合ノードＪ３との間のリンクが存在するので、条件ノードＣ１及びＣ２の条件が真であるか又は偽であるかにかかわりなくスキップされると判定され得る。したがって、コンパイラは、タスクノードＴ５のスキップフラグの検査を避け、単にタスクノードＴ５のタスクセクションＴ５（）を呼び出すタスクノードＴ５に関するサブルーチンを生成することができる。例えば、スキップされたセクションの後の下流のノードへのすべてのその他の入力が適切に処理される（つまり、下流のノードのカウンタを、スキップされたセクションに関する中間的な呼び出しが含まれていたとした場合にそのカウンタがデクリメントされたであろう回数だけデクリメントする）限り、制御フローグラフのセクション全体が条件付きノードの後でスキップされるべきである場合に関して、中間的なスキップフラグの検査及びｓｋｉｐ関数の呼び出しを省くことによって、その他の最適化がコンパイラによってなされ得る。

【００６７】

図４は、それぞれが条件ノード（それぞれＣ１及びＣ２）に続くタスクノードＴ１及びＴ２からの入力リンクを有する多入力タスクノードＴ３を含む単純な制御フローグラフ４００の例を示す。この例において、タスクノードＴ３は、ノードＴ３のタスクが実行されるためにノードＴ１及びＴ２のタスクが両方とも（スキップされずに）実行されなければならないように論理ＡＮＤ演算に対応する。表５は、制御フローグラフ４００に関して生成されるサブルーチンの例を示す。

【００６８】

【表５】

ノード	サブルーチン
接合ノードJ1	chain(C1); chainTo(C2)

条件ノードC1	if (<condition1>)
	chainTo(T1)
	else
	skip(T1)

条件ノードC2	if (<condition2>)
	chainTo(T2)
	else
	skip(T2)

タスクノードT1	if (skip)
	skip(T3)
	else
	T1(); chainTo(T3)

タスクノードT2	if (skip)
	skip(T3)
	else
	T2(); chainTo(T3)

タスクノードT3	if (skip)
	return
	else
	T3()

表5

【００６９】

一部の実施形態において、接合ノード（又はその他のノード）は、ノードへの入力の特徴に依存するさまざまな種類の論理演算を行うように構成される可能性がある。例えば、ノードは、すべての入力が「必須」入力として指定されるとき、論理ＡＮＤ演算を行い、すべての入力が「任意選択」入力として指定されるとき、論理ＯＲ演算を行うように構成される可能性がある。一部の入力が「必須」と指定され、一部の入力が「任意選択」と指定される場合、１組の所定の規則が、ノードによって実行される論理演算の組合せを解釈するために使用される可能性がある。

【００７０】

上述のタスク制御技術は、好適なソフトウェアを実行するコンピューティングシステムを用いて実装され得る。例えば、ソフトウェアは、それぞれが少なくとも１つのプロセッサ、（揮発性及び／又は不揮発性メモリ及び／又は記憶要素を含む）少なくとも１つのデ

ータ記憶システム、（少なくとも１つの入力デバイス又はポートを用いて入力を受け取るため、及び少なくとも１つの出力デバイス又はポートを用いて出力を与えるための）少なくとも１つのユーザインターフェースを含む（分散、クライアント／サーバ、又はグリッドなどのさまざまなアーキテクチャである可能性がある）１又は２以上のプログラミングされた又はプログラミング可能なコンピューティングシステムで実行される１又は２以上のコンピュータプログラムの手順を含み得る。ソフトウェアは、例えば、データフローグラフの設計、構成、及び実行に関連するサービスを提供するより大きなプログラムの１又は２以上のモジュールを含む可能性がある。プログラムのモジュール（例えば、データフローグラフの要素）は、データリポジトリに記憶されたデータモデルに準拠するデータ構造又はその他の編成されたデータとして実装され得る。

【００７１】

ソフトウェアは、ＣＤ－ＲＯＭ又は（例えば、汎用若しくは専用のコンピューティングシステム若しくはデバイスによって読み取り可能な）その他のコンピュータ可読媒体などの有形の非一時的媒体で提供されるか、或いはそのソフトウェアが実行されるコンピューティングシステムの有形の非一時的媒体にネットワークの通信媒体を介して配信される（例えば、伝播信号で符号化される）可能性がある。処理の一部又はすべては、専用のコンピュータで、又はコプロセッサ若しくはフィールドプログラマブルゲートアレイ（ＦＰＧＡ，field-programmable gate array）若しくは専用の特定用途向け集積回路（ＡＳＩＣ，application-specific integrated circuit）などの専用のハードウェアを用いて実行される可能性がある。処理は、ソフトウェアによって指定された計算の異なる部分が異なる計算要素によって実行される分散された方法で実装される可能性がある。それぞれのそのようなコンピュータプログラムは、本明細書において説明された処理を実行するために記憶デバイスの媒体がコンピュータによって読み取られるときにコンピュータを構成し、動作させるために、汎用又は専用のプログラミング可能なコンピュータによってアクセス可能な記憶デバイスのコンピュータ可読記憶媒体（例えば、ソリッドステートメモリ若しくは媒体、又は磁気式若しくは光学式媒体）に記憶されるか又はダウンロードされることが好ましい。本発明のシステムは、コンピュータプログラムで構成された有形の非一時的媒体として実装されると考えられる可能性もあり、そのように構成された媒体は、本明細書において説明された処理ステップのうちの１又は２以上を実行するために特定の予め定義された方法でコンピュータを動作させる。

【００７２】

本発明のいくつかの実施形態が、説明された。しかしながら、上述の説明は、添付の特許請求の範囲によって定義される本発明の範囲を例示するように意図されており、限定するように意図されていないことを理解されたい。したがって、その他の実施形態も、以下の特許請求の範囲内にある。例えば、本発明の範囲を逸脱することなくさまざまな修正がなされ得る。さらに、上述のステップの一部は、順序に依存しない可能性があり、したがって、説明された順序とは異なる順序で実行され得る。

【図１】

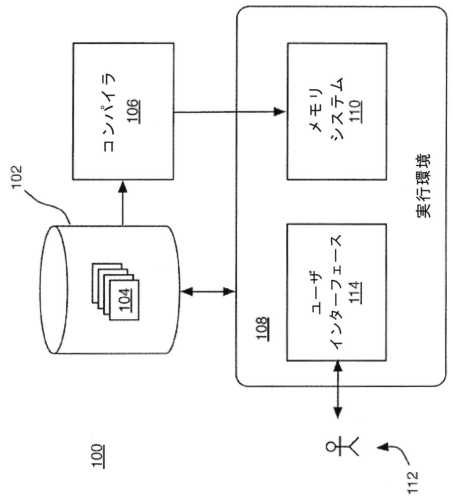

FIG. 1

【図２Ａ】

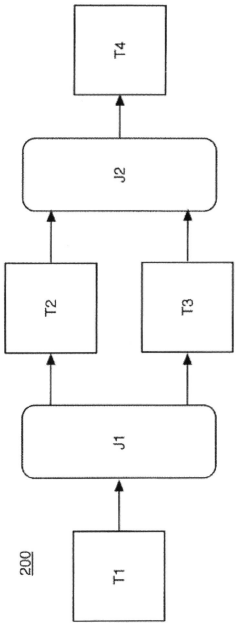

FIG. 2A

【図２Ｂ】

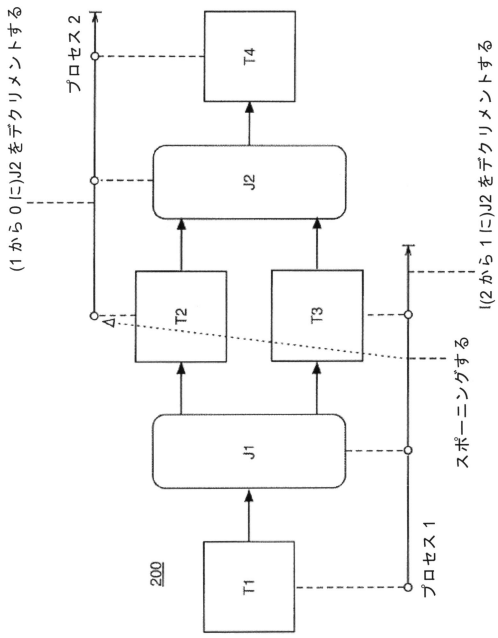

FIG. 2B

【図２Ｃ】

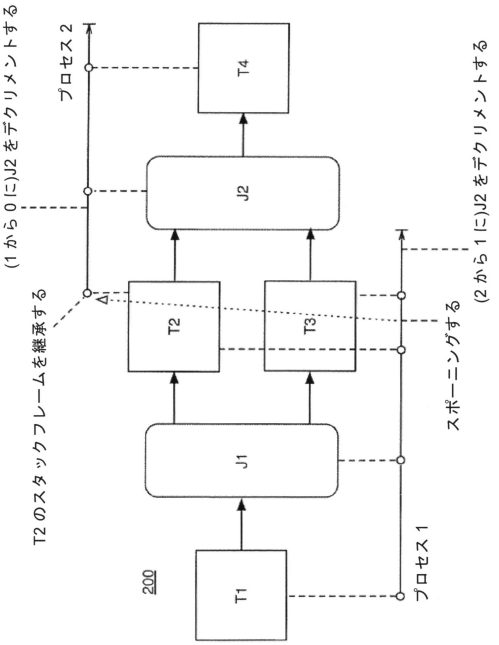

FIG. 2C

【図２Ｄ】

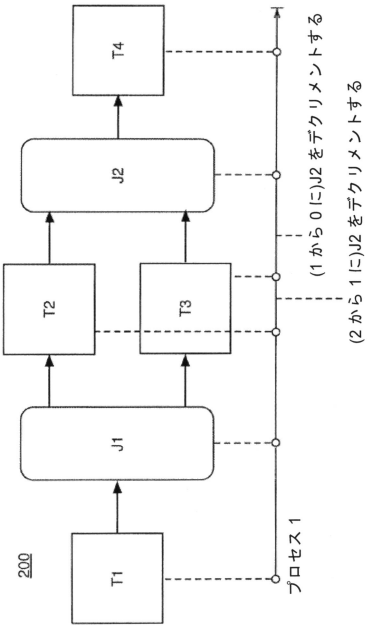

FIG. 2D

【図３】

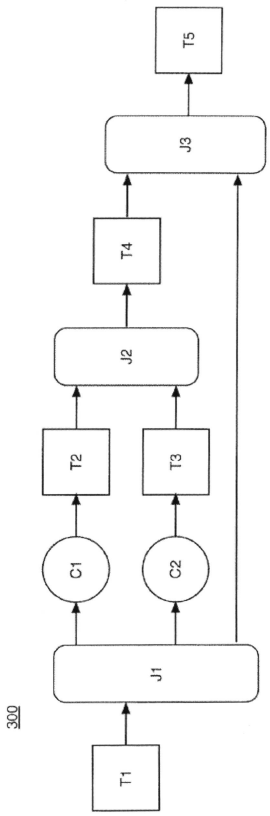

FIG. 3

【図４】

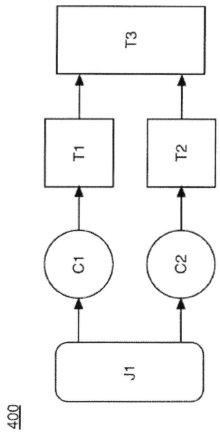

FIG. 4

フロントページの続き

(74)代理人 100150902
弁理士 山内 正子
(74)代理人 100141391
弁理士 園元 修一
(74)代理人 100198074
弁理士 山村 昭裕
(74)代理人 100172797
弁理士 有馬 昌広
(72)発明者 スタンフィル クレイグ ダブリュ.
アメリカ国 マサチューセッツ０１７７３ リンカーン ハックルベリーヒルロード４３

審査官 原 忠

(56)参考文献 特開２０１０－２４４５６３（ＪＰ，Ａ）
特表２０１１－５１７３５２（ＪＰ，Ａ）
特開２０１０－２８６９３１（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ ９／４６ － ９／５４