【公報種別】特許法第１７条の２の規定による補正の掲載
【部門区分】第６部門第３区分
【発行日】平成21年6月18日(2009.6.18)

【公表番号】特表2007-506199(P2007-506199A)
【公表日】平成19年3月15日(2007.3.15)
【年通号数】公開・登録公報2007-010
【出願番号】特願2006-527169(P2006-527169)
【国際特許分類】
Ｇ０６Ｆ　9/50　(2006.01)
Ｇ０６Ｆ　9/46　(2006.01)
【ＦＩ】
Ｇ０６Ｆ　9/46　４６２Ａ
Ｇ０６Ｆ　9/46　４１０

【誤訳訂正書】
【提出日】平成21年4月20日(2009.4.20)
【誤訳訂正１】
【訂正対象書類名】特許請求の範囲
【訂正対象項目名】全文
【訂正方法】変更
【訂正の内容】
【特許請求の範囲】
【請求項１】
メインスレッドを持つコードのコンパイル中に、データ処理システムのプロセッサで実行可能な、前記メインスレッドからコピーされた命令を含む１以上のスレッドを生成する生成段階と、
前記１以上のスレッドに関するディペンデンシーグラフに従って、前記１以上のスレッドの中からボトムアップの順にカレントスレッドを選択することにより、前記１以上のスレッドの中から最も下部の順を有するカレントスレッドを選択する選択段階と、
前記メインスレッドからコピーされた命令に基づいて、前記カレントスレッドから生成された１以上の子スレッドに割り当てられた、前記カレントスレッドと前記１以上の子スレッドとの間で共有可能なリソースを含むリソースを決定する決定段階と、
前記カレントスレッドと当該カレントスレッドの１以上の子スレッドとの間のリソースコンフリクトを避けるべく、前記カレントスレッドの１以上の子スレッドに割り当てられたリソースに考慮して、前記共有可能なリソースを含むリソースを前記カレントスレッドに割り当てる割り当て段階と
を備える方法。
【請求項２】
コードのコンパイル中に、データ処理システムのプロセッサで実行可能な複数のスレッドを生成する生成段階と、
前記複数のスレッドのディペンデンシーグラフ上をボトムアップの順で横断してカレントスレッドを選択することにより、子スレッドが全て選択された後に当該子スレッドの親スレッドをカレントスレッドとして選択する選択段階と、
前記カレントスレッドから生成された１以上の子スレッドに割り当てられたリソースを決定する決定段階と、
前記カレントスレッドと当該カレントスレッドの一つ以上の子スレッドとの間のリソースコンフリクトを避けるために、前記カレントスレッドの一つ以上の子スレッドに割り当てられたリソースに考慮して、前記カレントスレッドにリソースを割り当てる割り当て段階と

を備え、
前記割り当てられたリソースは、前記複数のスレッドのそれぞれによって使用された、前記プロセッサの少なくとも一つのハードウェアレジスタおよび物理メモリを含み、
前記コンパイルするコンパイラが前記カレントスレッドと前記カレントスレッドの１以上の子スレッドとの間のリソースコンフリクトを避けるために、前記１以上の子スレッドに割り当てられたリソースは、前記コンパイラによって管理されるデータ構造中に記録される
方法。
【請求項３】
前記リソースが、前記１以上のスレッドのそれぞれによって使用される少なくとも一つのハードウェアレジスタおよびメモリを含む
請求項１に記載の方法。
【請求項４】
前記１以上の子スレッドに割り当てられたリソースが、前記カレントスレッドによってアクセス可能なデータ構造中に記録される
請求項１または３に記載の方法。
【請求項５】
前記カレントスレッドに割り当てられた前記リソースに関してのデータ構造中のリソース情報をアップデートする段階
を更に備え、
前記データ構造が前記カレントスレッドの親スレッドによってアクセス可能である
請求項１から４のいずれかに記載の方法。
【請求項６】
前記１以上のスレッドのそれぞれが実行されるまで、ボトムアップの順に、前記選択段階、前記決定段階、および前記割り当て段階を繰り返す段階
を更に備える請求項１から５のいずれかに記載の方法。
【請求項７】
前記１以上のスレッドのそれぞれが実行された後に、前記１以上のスレッドの親スレッドであるメインスレッドにリソースを割り当てる段階
を更に備え、
前記メインスレッドにリソースを割り当てる段階は、前記メインスレッドのリソースを、前記１以上のスレッドに割り当てられたリソースを考慮して割り当てる
請求項６に記載の方法。
【請求項８】
前記割り当て段階に優先して、前記データ処理システム内にリソースが残っているか否か決定する段階と、
前記カレントスレッドの少なくとも一つの子スレッドを消去する段階と
を更に備え、
前記割り当て段階は、前記少なくとも一つの消去された子スレッドに関連付けられる前記リソースを用いて前記カレントスレッドにリソースを割り当てる
請求項１から７のいずれかに記載の方法。
【請求項９】
前記生成段階は、前記メインスレッドのデータ用のデータをプリフェッチする前記１以上のスレッドを生成する
請求項１、３から８のいずれかに記載の方法。
【請求項１０】
プログラムであって、コンピュータに、
メインスレッドを持つコードのコンパイル中に、データ処理システムのプロセッサで実行可能な、前記メインスレッドからコピーされた命令を含む１以上のスレッドを生成する生成段階と、

前記１以上のスレッドに関するディペンデンシーグラフに従って、前記１以上のスレッドの中からボトムアップの順にカレントスレッドを選択することにより、前記１以上のスレッドの中から最も下部の順を有するカレントスレッドを選択する選択段階と、
前記メインスレッドからコピーされた命令に基づいて、前記カレントスレッドから生成された１以上の子スレッドに割り当てられた、前記カレントスレッドと前記１以上の子スレッドとの間で共有可能なリソースを含むリソースを決定する決定段階と、
前記カレントスレッドと当該カレントスレッドの１以上の子スレッドとの間のリソースコンフリクトを避けるべく、前記カレントスレッドの１以上の子スレッドに割り当てられたリソースに考慮して、前記共有可能なリソースを含むリソースを前記カレントスレッドに割り当てる割り当て段階とを有する方法を実行させるプログラム。
【請求項１１】
プログラムであって、コンピュータに、
コードのコンパイル中に、データ処理システムのプロセッサで実行可能な複数のスレッドを生成する生成段階と、
前記複数のスレッドのディペンデンシーグラフ上をボトムアップの順で横断してカレントスレッドを選択することにより、子スレッドが全て選択された後に当該子スレッドの親スレッドをカレントスレッドとして選択する選択段階と、
前記カレントスレッドから生成された１以上の子スレッドに割り当てられたリソースを決定する決定段階と、
前記カレントスレッドと当該カレントスレッドの一つ以上の子スレッドとの間のリソースコンフリクトを避けるために、前記カレントスレッドの一つ以上の子スレッドに割り当てられたリソースに考慮して、前記カレントスレッドにリソースを割り当てる割り当て段階と
を有する方法を実行させ、
前記割り当てられたリソースは、前記複数のスレッドのそれぞれによって使用された、前記プロセッサの少なくとも一つのハードウェアレジスタおよび物理メモリを含み、
前記コンパイルするコンパイラが前記カレントスレッドと前記カレントスレッドの１以上の子スレッドとの間のリソースコンフリクトを避けるために、前記１以上の子スレッドに割り当てられたリソースは、前記コンパイラによって管理されるデータ構造中に記録される、プログラム。
【請求項１２】
前記リソースが、前記１以上のスレッドのそれぞれによって使用される少なくとも一つのハードウェアレジスタおよびメモリを含む
請求項１０に記載のプログラム。
【請求項１３】
前記１以上の子スレッドに割り当てられたリソースが、前記カレントスレッドによってアクセス可能なデータ構造中に記録される
請求項１０または１２に記載のプログラム。
【請求項１４】
前記方法が、
前記カレントスレッドに割り当てられた前記リソースに関してのデータ構造中のリソース情報をアップデートする段階
を更に備え、
前記データ構造が前記カレントスレッドの親スレッドによってアクセス可能である
請求項１０から１３のいずれかに記載のプログラム。
【請求項１５】
前記方法が、
前記１以上のスレッドのそれぞれが実行されるまで、ボトムアップの順に、前記選択段階、前記決定段階、および前記割り当て段階を繰り返す段階
を更に備える請求項１０から１４のいずれかに記載のプログラム。

【請求項１６】
前記方法は、
前記１以上のスレッドのそれぞれが実行された後に、前記１以上のスレッドの親スレッドであるメインスレッドにリソースを割り当てる段階
を更に備える請求項１５に記載のプログラム。
【請求項１７】
前記方法が、
前記割り当て段階に優先して、前記データ処理システム内にリソースが残っているか否か決定する段階と、
前記カレントスレッドの少なくとも１の子スレッドを消去する段階と
を更に備え、
前記割り当て段階は、前記少なくとも一つの消去された子スレッドに関連付けられる前記リソースを用いて前記カレントスレッドにリソースを割り当てる
請求項１０から１６のいずれかに記載のプログラム。
【請求項１８】
前記生成段階は、前記メインスレッドのデータ用のデータをプリフェッチする前記１以上のスレッドを生成する
請求項１０、１２から１７のいずれかに記載のプログラム。
【請求項１９】
マルチスレッディングオペレーションの実行ができるプロセッサと、前記プロセッサに結合するメモリを備え、前記メモリから前記プロセッサによって実行される処理により、前記プロセッサは、
メインスレッドを持つコードのコンパイル中に、前記プロセッサで実行可能な、前記メインスレッドからコピーされた命令を含む１以上のスレッドを生成し、
前記１以上のスレッドに関するディペンデンシーグラフに従って、前記１以上のスレッドの中からボトムアップの順にカレントスレッドを選択することにより、前記１以上のスレッドの中から最も下部の順を有するカレントスレッドを選択し、
前記メインスレッドからコピーされた命令に基づいて、前記カレントスレッドから生成された１以上の子スレッドに割り当てられた、前記カレントスレッドと前記１以上の子スレッドとの間で共有可能なリソースを含むリソースを決定し、
前記カレントスレッドと当該カレントスレッドの１以上の子スレッドとの間のリソースコンフリクトを避けるべく、前記カレントスレッドの１以上の子スレッドに割り当てられたリソースに考慮して、前記共有可能なリソースを含むリソースを前記カレントスレッドにリソースを割り当てるデータ処理システム。
【請求項２０】
マルチスレッディングオペレーションの実行ができるプロセッサと、前記プロセッサに結合するメモリを備え、前記メモリから前記プロセッサによって実行される処理により、前記プロセッサは、
コードのコンパイル中に、前記プロセッサで実行可能な複数のスレッドを生成し、
前記複数のスレッドのディペンデンシーグラフ上をボトムアップの順で横断してカレントスレッドを選択することにより、子スレッドが全て選択された後に当該子スレッドの親スレッドをカレントスレッドとして選択し、
前記カレントスレッドから生成された１以上の子スレッドに割り当てられたリソースを決定し、
前記カレントスレッドと当該カレントスレッドの一つ以上の子スレッドとの間のリソースコンフリクトを避けるために、前記カレントスレッドの一つ以上の子スレッドに割り当てられたリソースに考慮して、前記カレントスレッドにリソースを割り当てる割り当て、
前記割り当てられたリソースは、前記複数のスレッドのそれぞれによって使用された、前記プロセッサの少なくとも一つのハードウェアレジスタおよび物理メモリを含み、
前記コンパイルするコンパイラが前記カレントスレッドと前記カレントスレッドの１以

上の子スレッドとの間のリソースコンフリクトを避けるために、前記１以上の子スレッドに割り当てられたリソースは、前記コンパイラによって管理されるデータ構造中に記録されるデータ処理システム。
【請求項２１】
前記処理により、前記プロセッサは更に、前記カレントスレッドに割り当てられた前記リソースに関してのデータ構造中のリソース情報をアップデートし、
前記データ構造は、前記カレントスレッドの親スレッドによってアクセス可能である
請求項１９に記載のデータ処理システム。
【請求項２２】
前記処理により、前記プロセッサは、前記１以上のスレッドのそれぞれが実行されるまで、ボトムアップの順に、前記カレントスレッドの選択と、前記リソースの決定と、前記カレントスレッドへのリソースの割り当てとを繰り返す
請求項１９から２１のいずれかに記載のデータ処理システム。
【請求項２３】
前記処理により、前記プロセッサは更に、前記１以上のスレッドのそれぞれが実行された後に、前記１以上のスレッドの親スレッドであるメインスレッドにリソースを割り当て、
前記プロセッサは、前記メインスレッドにリソースを割り当てる場合に、前記メインスレッドのリソースを、前記１以上のスレッドに割り当てられたリソースを考慮して割り当てる
請求項２２に記載のデータ処理システム。
【請求項２４】
前記処理により、前記プロセッサは更に、
前記カレントスレッドへの前記リソースの割り当てに優先して、前記データ処理システム内にリソースが残っているか否か決定し、
前記カレントスレッドの少なくとも一つの子スレッドを消去し、
前記プロセッサは、前記カレントスレッドにリソースを割り当てる場合に、前記少なくとも一つの消去された子スレッドに関連付けられる前記リソースを用いて前記カレントスレッドに前記リソースを割り当てる
請求項１９から２３のいずれかに記載のデータ処理システム。
【請求項２５】
前記リソースが、前記１以上のスレッドのそれぞれによって使用される少なくとも一つのハードウエアレジスタおよびメモリを含む
請求項１９、２１から２４のいずれかに記載のデータ処理システム。
【請求項２６】
前記プロセッサは、前記１以上のスレッドを生成する場合に、前記メインスレッドのデータ用のデータをプリフェッチする前記１以上のスレッドを生成する
請求項１９、２１から２５のいずれかに記載のデータ処理システム。
【誤訳訂正２】
【訂正対象書類名】明細書
【訂正対象項目名】０００２
【訂正方法】変更
【訂正の内容】
【０００２】
メモリ待ち時間は、現在のプロセッサの高い性能を達成するに際して重大なボトルネックとなっている。アプリケーションのメモリアクセスパターンの予測が困難であり、ワーキングセットがかなり大きくなっているために、今日の多くの大きいアプリケーションはメモリ集約的である。キャッシュデザインの向上およびプリフェッチ技術における新しい開発にもかかわらず、メモリのボトルネック問題は依然として続いている。この問題は、従来のストライドベースのプリフェッチ技術に逆らう傾向にある、ポインタ集中のアプリケ

ーションの実行の際により悪化する。
【特許文献１】米国特許出願公開第２００３／００３７２９０号明細書
【特許文献２】米国特許出願公開第２００４／０１９４０９４号明細書
【特許文献３】米国特許出願公開第２００５／００７１８４１号明細書
【特許文献４】米国特許出願公開第２００５／００８１２０７号明細書
【特許文献５】米国特許出願公開第２００５／０１６５６７１号明細書
【特許文献６】米国特許第６２３３５９９号明細書
【特許文献７】米国特許第６３６３４１０号明細書
【特許文献８】米国特許第６５６７８３９号明細書
【特許文献９】米国特許第７０３６１２４号明細書
【特許文献１０】米国特許第７３１３７９５号明細書
【特許文献１１】米国特許第７３２８２４２号明細書
【特許文献１２】米国特許第７４１５６９９号明細書
【非特許文献１】Steenkiste, Peter A. et al., "A Simple Interprocedural Register Allocation Algorithm and Its Effectiveness for LISP", ACM Transactions on Programming Languages and Systems, New York, NY, vol. 11, No. 1, Jan. 1989, pp. 1-32.
【非特許文献２】Luk, Chi-Keung, "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors", Proceedings of the 28.sup.th International Symposium on Computer Architecture, (ISCA, Jun. 2001) Goteborg, Sweden, IEEE, pp. 40-51.
【非特許文献３】Kim, Dongkeun et al., "Design and Evaluation of Compiler Algorithms for Pre-Execution" Proceedings of the 10.sup.th International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 2002, San Jose, CA, USA, pp. 159-170.
【誤訳訂正３】
【訂正対象書類名】明細書
【訂正対象項目名】００１４
【訂正方法】変更
【訂正の内容】
【００１４】
ここで留意すべきは、図１はコンピュータシステムの様々な構成要素を図示する一方で、どのような特定のアーキテクチャ、又は、構成要素の相互接続の方法を表すことは意図していないことである。そのような詳細は本発明と密接に結びついていないからである。また、より少ない構成要素又はおそらくより多くの構成要素を有するネットワークコンピュータ、携帯型コンピュータ、携帯電話、そして他のデータ処理システムが、本発明と共に用いられてもよいことが理解されるであろう。図１に示すように、データ処理システムの形態であるコンピュータシステム１００は、マイクロプロセッサ１０３と、ＲＯＭ１０７、揮発性ＲＡＭ１０５、および非揮発性メモリ１０６とを接続するバス１０２を含む。インテル社からのペンティアム（登録商標）プロセッサ、又はモトローラ社からのパワーＰＣプロセッサであってよいマイクロプロセッサ１０３は、図１の例に示すようにキャッシュメモリ１０４に結合されている。バス１０２は、これらの様々な構成要素を互いに相互接続し、また、これらの構成要素１０３、１０７、１０５、および１０６を、マウス、キーボード、モデム、ネットワークインターフェース、プリンタ、および当該技術分野で周知の他のデバイスなどの入出力（Ｉ／Ｏ）デバイス１１０と同様にディスプレイコントローラおよびディスプレイデバイス１０８にも相互接続する。一般的に、入出力デバイス１１０は、入出力コントローラ１０９を介してシステムに接続される。揮発性ＲＡＭ１０５は、典型的には、リフレッシュ又はメモリ内のデータを保持するために継続的に電源供給を必要とする、動的ＲＡＭ（ＤＲＡＭ）として実行される。非揮発性メモリ１０６は、典型的には、磁気ディスク装置、磁気光学ディスク装置、光ディスク装置、あるいはＤＶＤＲＡＭまたはシステムから電源供給がなくなった後でもデータを保持する他の型のメモ

リシステムである。一般的に、非揮発性メモリはまた、必ずしも必要ではないが、ランダムアクセスメモリであってよい。図１では、非揮発性メモリが、データ処理システムの残りの構成要素に直接的に接続しているローカルデバイスである一方で、システムから離れている非揮発性メモリを、モデムやイーサネット（登録商標）インターフェースのようなネットワークインターフェースを介してデータ処理システムに接続されているネットワーク上の記憶装置として、本発明が利用してもよいことを理解されるであろう。バス１０２は、当該技術分野の当業者にとって周知の様々なブリッジ、コントローラ、および／又はアダプタを介してお互いに接続されている、一つ以上のバスを含んでいてよい。一実施形態においては、入出力コントローラ１０９は、ＵＳＢ周辺機器の制御のためのＵＳＢ（ユニバーサルシリアルバス）アダプタ、あるいは入出力デバイス１１０の中に含まれるＰＣＩデバイスの制御のためのＰＣＩコントローラを含んでいる。他の実施形態においては、入出力コントローラ１０９は、ファイアワイヤ（ＦｉｒｅＷｉｒｅ）装置としても知られる、ＩＥＥＥ１３９４装置の制御のためのＩＥＥＥ１３９４コントローラを含む。

【誤訳訂正４】
【訂正対象書類名】明細書
【訂正対象項目名】００１６
【訂正方法】変更
【訂正の内容】
【００１６】
典型的なシステム１００の中で稼動しているオペレーティングシステムは、マイクロソフト社からのウィンドウズ（登録商標）、または、アップルコンピュータ社からのマックＯＳ（登録商標）であってよい。他の例では、オペレーティングシステムは、リナックス（登録商標）、または、ユニックス（登録商標）であってもよい。組み込みの実時間オペレーティングシステムのような、他のオペレーティングシステムを利用してもよい。