(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5893038号
(P5893038)
(45)発行日　平成28年3月23日(2016.3.23)　(24)登録日　平成28年3月4日(2016.3.4)

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

請求項の数 10 (全 22 頁)

(21)出願番号　特願2013-531616(P2013-531616)
(86)(22)出願日　平成23年9月9日(2011.9.9)
(65)公表番号　特表2013-539130(P2013-539130A)
(43)公表日　平成25年10月17日(2013.10.17)
(86)国際出願番号　PCT/US2011/051023
(87)国際公開番号　WO2012/047447
(87)国際公開日　平成24年4月12日(2012.4.12)
審査請求日　平成26年8月12日(2014.8.12)
(31)優先権主張番号　12/892,291
(32)優先日　平成22年9月28日(2010.9.28)
(33)優先権主張国　米国(US)

(73)特許権者　314015767
マイクロソフト　テクノロジー　ライセンシング，エルエルシー
アメリカ合衆国　ワシントン州　９８０５２　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　100107766
弁理士　伊東　忠重
(74)代理人　100070150
弁理士　伊東　忠彦
(74)代理人　100091214
弁理士　大貫　進介

最終頁に続く

(54)【発明の名称】ユーザ定義型のコンパイル時境界検査

(57)【特許請求の範囲】
【請求項１】
少なくとも１つのプロセッサによって実行されると、該プロセッサにユーザ定義型のコンパイル時境界検査の処理を実行させるためのプログラムであって、ソースコードから中間言語コードを通じて実行可能コードとなるプログラムのコンパイル中に実行される前記処理は、
前記ソースコードにおいて、ユーザ定義の境界検査の注釈で注釈付けられるメモリアクセスメソッドを注釈メンバとして有するユーザ定義クラスを特定するステップであって、前記ユーザ定義クラスは、さらに前記メモリアクセスメソッドにおける境界検査を生成するための境界情報を提供することを示すように注釈付けられる境界提供メンバを別の注釈メンバとして有し、前記ユーザ定義の境界検査の注釈は、前記ユーザ定義クラス内のフィールドがメモリアクセスメソッドの境界を包含することを示す、境界包含のフィールドの注釈と、前記ユーザ定義クラス内の境界取得メソッドがメモリアクセスメソッドの境界を返すことを示す、境界取得メソッドの注釈とのうちの少なくとも１つを備え、前記ユーザ定義クラスは構成型として多要素配列型を使用せずに定義される、ユーザ定義クラスを特定するステップと、
前記ユーザ定義の境界検査の注釈の境界検査表現を前記中間言語コードに挿入するステップと、
前記実行可能コードで発生する可能性のある重複した境界検査を減らす最適化を適用するステップと

を備える、プログラム。
【請求項２】
前記処理は、境界検査表現に応答して境界検査コードを前記中間言語コードに挿入するステップをさらに備え、
前記適用するステップは、前記最適化を境界検査表現に適用するのではなく、前記最適化を、挿入された境界検査コードに適用する、
請求項１に記載のプログラム。
【請求項３】
前記適用するステップは、前記最適化を前記中間言語コード内の境界検査コードに適用するのではなく、前記最適化を境界検査表現に適用する、請求項１に記載のプログラム。
【請求項４】
前記特定するステップは、
明示的に割り当てられるメモリへのアクセスの注釈で注釈付けられたメモリアクセスメソッドを特定することと、
内蔵型のシステム定義の境界検査を補足するように、ユーザ定義の境界検査を示す注釈を特定することと、
ガーベジコレクタ管理型のシステム定義の境界検査を補足するように、ユーザ定義の境界検査を示す注釈を特定することと、
のうちの少なくとも１つを含む、請求項１に記載のプログラム。
【請求項５】
ユーザ定義型のコンパイル時境界検査を管理するためにコンピュータシステムが実行する方法であって、
コンピュータプログラムのソースコードを取得するステップと、
前記ソースコードにおいてユーザ定義のデータ型を指定するステップであって、前記ユーザ定義のデータ型は、構成型としていずれの多要素配列型も使用せずに定義されている、ステップと、
前記ユーザ定義のデータ型によって定義されるメモリアクセスメソッドを見つけるステップと、
明示的に割り当てられるメモリへのアクセスの注釈で前記メモリアクセスメソッドに注釈を付けるステップと、
　前記ユーザ定義のデータ型によって定義されるフィールドが前記メモリアクセスメソッドについての境界を包含することを示す、フィールド包含境界の注釈、及び
　前記ユーザ定義のデータ型によって定義される境界取得メソッドが前記メモリアクセスメソッドについての境界を返すことを示す、境界取得メソッドの注釈、
のうちの少なくとも１つによって前記ソースコードに注釈を付けるステップと、
を備える、方法。
【請求項６】
論理プロセッサと、
前記論理プロセッサと動作可能に通信するメモリと、
前記メモリに常駐し、ユーザ定義型を有するソースコードであって、前記ユーザ定義型は、ユーザ定義の境界検査の注釈で注釈付けられるメモリアクセスメソッドを有し、前記ユーザ定義型は、少なくとも１つの境界指定子を有する、ソースコードと、
前記メモリに常駐し、前記ユーザ定義の境界検査の注釈の境界検査表現を、中間言語コードに挿入するように構成されたコンパイラと、
前記メモリに常駐し、重複した境界検査を減らすために最適化を中間言語コードに適用するように構成されたオプティマイザと、
を備える、コンピュータシステム。
【請求項７】
前記注釈付けられるソースコードは、デバイスドライバのソースコードを備え、前記ユーザ定義型は、メモリマップドバッファに対応する、請求項６に記載のシステム。

【請求項８】
前記ソースコードは、ガーベジコレクトされたデータ型を備え、前記ユーザ定義型は、明示的に割り当てられるメモリに対応する、請求項６に記載のシステム。
【請求項９】
前記ユーザ定義型は、多要素配列型ではない、構成型として定義される、請求項６に記載のシステム。
【請求項１０】
前記境界指定子は、
ユーザ定義のデータ型におけるフィールドが、前記メモリアクセスメソッドについての境界を包含することを示す、境界を包含するフィールドの注釈、および
ユーザ定義のデータ型における境界取得メソッドが、前記メモリアクセスメソッドについての境界を返すことを示す、境界取得メソッドの注釈、
のうちの少なくとも１つを備える、請求項６に記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、コンピュータプログラミングに関する。
【背景技術】
【０００２】
変数がある指定された境界内にあるかどうかを検出する境界検査（bounds checking）は、コンピュータプログラムにおいて行うことができる。例えば、ある値が配列中のインデックスとして使用されて、その値が配列の境界内にあるかどうかを判定される前に、その値を検査することができる。この種の境界検査は、インデックス検査または範囲検査と呼ばれることもある。誤った境界の検査は、例外信号などのランタイムエラーを生成する場合もある。配列または他の変数の境界外にアクセスしようとすると、プログラミングエラーを示すことが多い。しかし、境界検査によってプログラムの実行時間が増えるため、境界変数を使用する度、その前に実行される境界検査が必ずしも行われるわけではない。
【０００３】
コンパイラは、必要ないと判断された境界検査を自動的になくすこともある。例として、プログラムコードが配列内のある位置からある値を読み取り、その後別の値（または同じ値）をその同じ位置に戻して記憶すると仮定する。最適化しない場合、このコードは、配列の位置がその配列から読み取られる時に行う第１の境界検査と、その配列の同じ位置が書き込まれる時に行う第２の境界検査とを含むかもしれない。しかし、コンパイラまたは他の最適化ツールは、その配列のサイズが変わらないこと、およびその配列の同じ位置が読み取られて、その後書き込まれることを判定した後に、第２の境界検査を自動的になくすことができる。より一般的には、冗長な境界検査を自動的に減らすまたはなくすためのさまざまな最適化がコンパイラおよび他のツールに使用されている。
【発明の概要】
【０００４】
配列アクセスに対して自動化される境界検査は、それが実際の（または認識された）影響をプログラム性能に及ぼすために、必ずしも使用されるわけではない。しかし、自動境界検査は、それにもかかわらず、配列により安全にアクセスするために選択して使用する開発者にとって容易に利用可能である。しかし、あるいは単純な配列とは別に、開発者がより複雑な構造を有するカスタムデータ型を使用する場合、アクセス違反の検査は、境界検査コードを手動で挿入することを伴うこともある。残念ながら、開発者が境界検査を明示的に書き込む場合、境界検査コードの用途をコンパイラに明らかにすることができない。従ってコンパイラは、コンパイラに冗長な境界検査を解除するようにさせるであろう情報に欠ける。
【０００５】
本明細書で説明するいくつかの実施形態は、ユーザ定義型(user-defined types)の自動

コンパイル時境界検査(automatic compile-time bounds checking)を提供し、一部は、明示的に割り当てられるメモリに安全にアクセスするためにソースコード内のユーザ定義クラス(user-defined class)を特定することによって提供する。ユーザ定義クラスは、ユーザ定義の境界検査注釈(user-defined-bounds check annotation)、例えば、コードがメモリマップドバッファ(memory-mapped buffer)または他の明示的に割り当てられるメモリにアクセスすることをコンパイラに示す注釈を用いて、開発者によって注釈付けられたメモリアクセスコードメンバ(memory-accessing code member)を有する。ユーザ定義クラスは、メモリアクセスコード上で境界検査を生成するための境界情報を提供することをコンパイラに示すように、開発者によって注釈付けられる境界提供メンバ(bound-providing member)も有する。境界提供メンバは、例えば、境界を包含する整数フィールドであってよいし、または呼び出されると境界を返すメソッドであってもよい。

【０００６】

ユーザ定義の境界検査は、プログラミング言語を有していないところに境界検査を提供できるし、または、例えば、内蔵配列型(built-in array type)またはガーベジコレクタ管理型(garbage-collector-managed type)をラップすることによって既存の境界検査を補足することができる。境界検査は、配列、およびレイアウトがコンパイラによって制御される他の型にとどまらず拡張可能である。どの多要素配列型(multi-element array type)も構成型として使用せずに、ユーザ定義クラスを定義できる。

【０００７】

ユーザ定義の境界検査注釈の境界検査表現は、コンパイラによって中間言語コードで挿入される。最適化は、その後、実行可能コードで発生するであろう重複した境界検査を自動的に減らす。最適化は、中間言語の境界検査表現および挿入された境界検査コード、またはその両方に適用されてよい。

【０００８】

所与の例は、例示にすぎない。この発明の概要は、特許請求される主題の主要な特徴または不可欠な特徴を明らかにすることを意図せず、特許請求される主題の範囲を限定するのに使用されることも意図しない。むしろ、この発明の概要は、以下の発明を実施するための形態でさらに説明される、いくつかの概念を簡易な形式で紹介するために与えられる。本革新は、特許請求の範囲で定義され、この発明の概要が特許請求の範囲と相反する範囲において、特許請求の範囲に従わなければならない。

【図面の簡単な説明】

【０００９】

添付の図面を参照してより詳細に説明する。これらの図面は、選択された態様を例示しているにすぎず、従って発明の対象範囲を完全に決めるものではない。

【図１】複数のネットワークノード上に存在し得るオペレーティング環境において少なくとも１つのプロセッサ、少なくとも１つのメモリ、少なくとも１つのプログラムソースコード、および他の項目を有するコンピュータシステムを示し、さらに構成された記憶媒体の実施形態を示すブロック図である。

【図２】任意に複雑なユーザ定義のデータ型のコンパイル時境界検査をアーキテクチャの一例において示すブロック図である。

【図３】いくつかの処理のステップおよび構成された記憶媒体の実施形態を示すフローチャートである。

【発明を実施するための形態】

【００１０】

概要

今日の管理コードシステム、例えば、Ｍｉｃｒｏｓｏｆｔ（登録商標）．ＮＥＴやＪａｖａ（登録商標）環境（それぞれ、マイクロソフト社、オラクルアメリカ社の登録商標である）のような管理コードシステムは、管理コードプログラムによって使用されるすべてのメモリが割り当てられ、そして自動的に管理されることを前提としていることが多い。しかし、システムプログラミングにおいて、特にデバイスドライバプログラミングにおい

て、この前提が成り立たないこともある。いくつかの事例において、デバイスドライバによって使用されるメモリは、デバイスのバッファが特定のアドレスセットにメモリマップされる時などに、物理的デバイスによって位置が固定される。他の事例において、メモリは、データの不必要な複製を回避するように、または一定の制限時間内にバッファを再使用するように明示的に管理される。

【００１１】

Ｃ＃などのいくつかの言語において、プログラムが明示的に割り当てられるメモリを使用する場合、プログラマは、ｕｎｓａｆｅポインタを管理されていないデータ構造に利用することが多い。ポインタにアクセスするまたはポインタ演算を行う時にエラーを回避するように細心の注意が必要となる。ｕｎｓａｆｅポインタは、誤ったポインタ演算がメモリ破損またはプログラムクラッシュにつながる恐れがあるので、管理コードの正確性の利点を否定することがある。そのようなエラーは、特にガーベジコレクションが機能している中でデバックするのが困難になる。プログラマがカスタムデータ構造を書き込み、且つ安全のために境界検査を実行したいと望む場合、プログラマは、ソースコードの行に境界検査を明示的に書き込むこともある。そのようなコードの用途は、コンパイラによってそのコードの他の行の用途と区別できない。この事例において、コンパイラは、コンパイラに明示的な境界検査を解除するようにさせる情報に欠ける。

【００１２】

本明細書で説明するいくつかの実施形態は、管理コードが明示的に割り当てられるメモリに安全且つ効率的にアクセスするようにさせる。管理コードは、従ってシステムプログラミングに効率的に使用されることができる。本明細書で説明するように、コンパイル時に最適化される境界検査を用いて管理コードをシステムプログラミングに使用することによって、プログラマは、管理コードで効率的なデバイスドライバを書き込み、そしてオペレーティングシステムクラッシュの原因になることが多いデバイスドライバの共通のエラーをなくすことができる。

【００１３】

本明細書で説明するいくつかの実施形態は、より広い文脈で見られてもよい。例えば、メモリアクセス、変数境界、ソースコードの注釈、およびコンパイルなどの概念は、特定の実施形態と関連することもある。しかし、それでは、発明の独占権が、本明細書では抽象的な考えに求められる広い文脈の利用から得られない。発明の独占権は、抽象的な考えではない。むしろ、本開示は、特定の実施形態を適切に提供することに重点を置いている。例えば、他の媒体、システム、およびメモリアクセスを伴うメソッド、境界検査、コンパイル、および／またはソース注釈は、本発明の範囲外である。従って、曖昧さおよびそれに付随する証明問題も、本開示の適切な理解によって回避される。

【００１４】

図面に例示したような模範的な実施形態を言及するにあたり、特定の用語を本明細書では同じ説明に使用する。しかし、本開示の従来技術（複数）およびそれを占有している当業者が思い付くであろう、本明細書で例示した機能の改変およびさらなる変更、および本明細書で例示した発明の原理の付加的な適用を特許請求の範囲の範囲内と見なさなければならない。

【００１５】

用語の意味は、本開示で明確にされ、従ってこのような明確化に十分注意しながら特許請求の範囲を読まなければならない。特定の例が与えられているが、従来技術（複数）の当業者は、他の例も、使用される用語の意味の範囲内であり、且つ１または複数の特許請求の範囲の範囲内であってよいことを理解するであろう。用語は、ここでは一般的な用法、特定の産業の用法、または特定のディクショナリまたはディクショナリのセットの用法を有するものと必ずしも同じ意味ではない。用語の意味の幅を示すのに役立つように、さまざまな言い回しを用いて参照番号を使用してよい。参照番号を所与の文章から省略するのは、図面の内容が文字によって論じられていないことを必ずしも意味するわけではない。発明者は、自身で辞書編集する自分の権利を主張して行使する。用語は、ここでは発明

を実施するための形態および／またはアプリケーションファイル内のどこかで明示的または暗示的のいずれかで定義されてもよい。

【００１６】

本明細書で使用される際、「コンピュータシステム」は、例えば、１または複数のサーバ、マザーボード、処理ノード、パーソナルコンピュータ（携帯用またはそれ以外のもの）、パーソナルデジタルアシスタント、セルまたはモバイル電話、および／または少なくとも一部は命令によって制御される１または複数のプロセッサを提供するデバイス（複数）を含んでよい。その命令は、メモリおよび／または専用回路内のソフトウェアの形式であってよい。特に、ワークステーションまたはラップトップコンピュータ上で実行する多くの実施形態を思い付くかもしれないが、他の実施形態を計算デバイス上で実行でき、そのような任意のデバイスの１または複数は、所与の実施形態の一部であってよい。

【００１７】

「マルチスレッド」コンピュータシステムは、複数の実行スレッドをサポートするコンピュータシステムである。用語「スレッド」は、同期する能力があるまたは同期に従う任意のコードを含むことを理解しておく必要があり、例えば、「タスク」、「処理」、または「コルーチン」などの、他の呼び名も知っておいた方がよい。スレッドは、並行して、逐次に、または並行実行（例えば、多重処理）と逐次実行（例えば、時間スライス）とを組み合わせて実行できる。マルチスレッド環境は、さまざまな構成で設計されている。実行スレッドは、並行して実行してよいし、または実行スレッドは、並行実行に組織されてもよいが、実際には順番に逐次に実行する。マルチスレッディングは、例えば、多重処理環境において異なるスレッドを異なるコア上で実行することによって、異なるスレッドを単一のプロセッサコア上で時間スライスすることによって、または時間スライスとマルチプロセッサスレッディングとをいくつか組み合わせることによって実装されてもよい。スレッドの文脈切り替えを、例えば、カーネルのスレッドスケジューラによって、ユーザ空間信号によって、またはユーザ空間の動作とカーネルの動作との組み合わせによって開始できる。スレッドは、例えば、順番に共有データに動作できるし、または各スレッドは、自身のデータに動作できる。

【００１８】

「論理プロセッサ」または「プロセッサ」は、単一の独立したハードウェアスレッド処理ユニットである。例えば、１コア当たり２スレッドを実行するハイパースレッドクアドコアのチップは、８つの論理プロセッサを有する。プロセッサは、汎用であってよいし、またはグラフィック処理、信号処理、浮動小数点演算、暗号化、Ｉ／Ｏ処理などの、特定の使用に合わせてもよい。

【００１９】

「マルチプロセッサ」コンピュータシステムは、複数の論理プロセッサを有するコンピュータシステムである。マルチプロセッサ環境は、さまざまな構成で発生する。所与の構成において、すべてのプロセッサが機能的に同等であることもあれば、一方別の構成において、いくつかのプロセッサが異なるハードウェア能力、異なるソフトウェア割り当て、またはその両方を有することによって他のプロセッサと異なることもある。構成に応じて、プロセッサを単一のバスに互いに強固に結合してもよいし、緩く結合してもよい。ある構成において、プロセッサは、中央メモリを共有し、ある構成において、プロセッサは、それぞれが自身のローカルメモリを有し、そしてある構成において、共有メモリとローカルメモリとの両方が存在する。

【００２０】

「カーネル」は、オペレーティングシステム、ハイパーバイザ、仮想マシン、および同様のハードウェアインタフェースソフトウェアを含む。

【００２１】

「コード」は、プロセッサの命令、データ（定数、変数、およびデータ構造を含む）、または命令とデータとの両方を意味する。

【００２２】

「プログラム」は、本明細書では広範に使用され、アプリケーション、カーネル、ドライバ、割り込みハンドラ、ライブラリ、およびプログラマ（開発者とも呼ばれる）によって書き込まれる他のコードを含む。

【００２３】

「自動的に」は、自動化していないものとは対照的に、自動化の使用（例えば、本明細書で説明する特定の演算を行うソフトウェアによって構成される汎用計算ハードウェアの使用）を意味する。特に、「自動的に」実行されるステップは、手書きまたは誰かの考えによって実行されない。それらのステップは、マシンを用いて実行される。しかし、「自動的に」は、必ずしも「即時に」を意味するわけではない。

【００２４】

この文書全体を通じて、任意の複数「（ｓ）」は、示された機能のうちの１または複数が存在することを意味する。例えば、「注釈（複数）」は、「１または複数の注釈」または同等に「少なくとも１つの注釈」を意味する。

【００２５】

この文書全体を通じて、明示的に別段の定めをした場合を除き、処理中のどのステップの基準(reference)も、受益者によって直接実行され、および／または中間機構および／または中間エンティティを通じて受益者によって間接的に実行されてもよく、なおもステップの範囲内であると仮定する。即ち、受益者によってステップを直接実行することは、直接の実行が明示的に定められた用件でない限り要求されない。例えば、「送信する」、「送る」、「通信する」、「適用する」、「挿入する」、「注釈付ける」、「表す」、「指定する」などの、受益者による動作を伴うステップ、あるいは送り先を対象とするステップは、別の受益者によって転送する、複製する、アップロードする、ダウンロードする、符号化する、復号する、圧縮する、展開する、暗号化する、解読するなどの、中間動作を伴う場合もあり、なおも受益者によって直接実行されると理解してよい。

【００２６】

データまたは命令の参照が行われる時はいつも、それらの項目は、コンピュータ可読メモリを構成し、それよって、例えば、単に紙上に存在し、誰かの考えによるものとは対照的に、または有線の一過性信号とは対照的に、そのメモリを特定の項目に変換することが理解されよう。

【００２７】

オペレーティング環境

図１に関して、実施形態のオペレーティング環境１００は、コンピュータシステム１０２を含む。コンピュータシステム１０２は、マルチプロセッサコンピュータシステムであってもよいし、そうでなくてもよい。オペレーティング環境は、所与のコンピュータシステム内に１または複数のマシンを含むことができ、そのコンピュータシステムは、クラスタ、クライアントサーバネットワーク、および／またはピアツーピアネットワークであってもよい。

【００２８】

人間ユーザ１０４は、ディスプレイ、キーボード、および他の周辺機器１０６を使用することによってコンピュータシステム１０２と対話できる。システムの管理者、開発者、技術者、およびエンドユーザは、それぞれ特定のタイプのユーザ１０４である。１または複数の人間の代わりに動作する自動化されるエージェントもユーザ１０４にしてよい。いくつかの実施形態において、記憶デバイスおよび／またはネットワークデバイスを周辺機器と見なしてよい。図１に示していない他のコンピュータシステムは、コンピュータシステム１０２と対話できるし、または例えば、ネットワークインタフェース機器経由で１または複数の接続を使用して、別のシステムの実施形態と対話できる。

【００２９】

コンピュータシステム１０２は、少なくとも１つの論理プロセッサ１１０を含む。コンピュータシステム１０２は、他の適したシステムのように、１または複数のコンピュータ可読の持続性記憶媒体１１２も含む。媒体１１２は、異なる物理的タイプであってよい。

媒体１１２は、揮発性メモリ、不揮発性メモリ、位置が固定されている媒体、取り外し可能な媒体、磁気媒体、光媒体、および／または他のタイプの持続性媒体（単に信号を伝搬する有線などの一過性媒体とは対照的である）であってよい。特に、ＣＤ、ＤＶＤ、メモリスティックなどの、構成媒体（configured medium）１１４、または他の取り外し可能な不揮発性メモリ媒体は、挿入あるいはインストールされた時にコンピュータシステムの機能的部分になり、そのコンテンツがプロセッサ１１０によって使用されるようアクセス可能にすることができる。取り外し可能な構成媒体１１４は、コンピュータ可読記憶媒体１１２の一例である。コンピュータ可読記憶媒体１１２のその他の例は、内蔵ＲＡＭ、ＲＯＭ、ハードディスク、およびユーザ１０４によって容易に取り外すことができない他の記憶デバイスを含む。

【００３０】

構成媒体１１４は、プロセッサ１１０によって実行可能である命令１１６によって構成される。「実行可能」は、本明細書では、例えば、マシンコード、解釈可能コード、および仮想マシン上で実行するコードを含む広い意味で使用される。構成媒体１１４は、データ１１８によっても構成され、そのデータは、作成され、変更され、参照され、および／あるいは命令１１６の実行によって使用される。命令１１６およびデータ１１８は、それらが常駐する構成媒体１１４を構成する。そのメモリが所与のコンピュータシステムの機能部分である場合、命令１１６およびデータ１１８は、そのコンピュータシステムも構成する。いくつかの実施形態において、データ１１８の部分は、製品特性、在庫、物理的測定、設定、画像、読み取り、ターゲット、ボリュームなどの、現実世界の項目表現である。そのようなデータはまた、本明細書で説明する柔軟性のあるコンパイル時最適化境界検査(compile-time optimized bounds checking)によって変換され、例えば、挿入する、適用する、指定する、注釈付ける、表す、結合する、展開する、実行する、変更する、表示する、作成する、読み込む、および／または他の動作によって変換される。

【００３１】

プログラム１２０（例えば、ソースコード１２２、中間言語コード１２４、および実行可能コード１２６を用いる）、デバッガ、コンパイラおよび他の展開ツール１３６、他のソフトウェア、および図に示した他の項目は、１または複数の媒体１１２内に部分的または完全に常駐でき、それによってそのような媒体を構成する。中間言語コード１２４は、中間表現と呼ばれることもある。プログラム１２０は、例えば、内蔵型１２８およびガーベジコレクタ管理型１３０を含むことができる。多くの展開構成において、配列型１３２は、内蔵型と管理型との両方である。プロセッサ（複数）１１０に加え、オペレーティング環境は、例えば、ディスプレイ、メモリマップドデバイス１３４、バス、電源、およびアクセラレータなどの、他のハードウェアを含んでよい。

【００３２】

所与のオペレーティング環境１００は、調整されたソフトウェア開発ツールのセットを開発者に提供する、統合開発環境（ＩＤＥ）１３８を含むことができる。特に、いくつかの実施形態に適したオペレーティング環境のうちのいくつかは、プログラム開発をサポートするように構成されたＭｉｃｒｏｓｏｆｔ（登録商標）Ｖｉｓｕａｌ　Ｓｔｕｄｉｏ（登録商標）開発環境（マイクロソフト社の登録商標である）を含むまたはその作成に役立つ。いくつかの適したオペレーティング環境は、Ｊａｖａ（登録商標）環境（オラクルアメリカ社の登録商標である）を含み、いくつかのオペレーティング環境は、Ｃ＋＋またはＣ＃（「Ｃシャープ」）などの、言語を利用する環境を含む。しかし、本明細書の教示は、各種のプログラミング言語、プログラミングモデル、およびプログラムに適用可能である。

【００３３】

図１の概略図に１または複数の項目を示し、それらの項目が必ずしも例示されたオペレーティング環境の一部というわけではないことを強調しているが、本明細書で論じるようなオペレーティング環境の項目と相互運用できる。いずれの図またはいずれの実施形態においても、概略図にない項目が必然的に必要ということではない。

【００３４】
システム
図２は、いくつかの実施形態の使用に適しているアーキテクチャを示す。ユーザ定義クラス２０２などのユーザ定義型２０４は、開発者の境界検査の意思を、コンパイラに境界検査だけでなく冗長な境界検査の解除も提供するようにさせる方法でコンパイラ２２４に伝達する注釈２０６を有する。注釈は、例えば、ユーザ定義型のメモリアクセス(memory-accessing)２０８コードおよび境界提供(boundary-providing)２１０コードを特定できる。メモリアクセスコードを、例えば、インライン化ステートメント(inline statements)および／または個別メソッド(distinct methods)にできる。境界提供コードは、境界２１８を包含するフィールド２１２および／または呼び出されると境界２１８を返すメソッド２１４であってよい。
【００３５】
配列インデックス検査は、非常に有用であるに違いないが、本明細書の重点は、他の種類の境界検査、即ち、単なる配列ではないユーザ定義構造(user-defined structures)の境界検査に置かれる。よく知られた配列型１３２とは違って、クラス２０２または他のユーザ定義型２０４は、内蔵型ではなく、それゆえコンパイラ２２４によって制御されないデータレイアウト２１６を有することができる。
【００３６】
いくつかの実施形態において、コンパイラ２２４は、注釈２０６に応答して中間言語コード１２４で境界検査表現２２０を挿入する。境界検査コード２２２は、その後、境界検査表現２２０に応答して実行可能コード１２６内に置かれる。中間言語コードのよく知られた表記法は、境界検査表現２２０によって遵守され、生成された境界検査コード２２２は、よく知られた条件付きジャンプ命令などを含むことができる。しかし、このようなよく知られた表記法および命令の文脈は、本用途では、単なる配列ではなく、且ついくつかの実施形態においてその配列を構成型としても使用していない、ユーザ定義型である。
【００３７】
いくつかの実施形態において、オプティマイザ２２６は、実行可能コード１２６で発生するであろう冗長な境界検査を解除するために、最適化（複数）２２８を境界検査表現２２０、境界検査コード２２２、またはその両方に適用する。オプティマイザ２２６は、コンパイラ２２４に統合されてもよいし、またはコンパイラ２２４によってまたは開発者によって呼び出される別個のツールであってもよいが、実施形態によって異なる。配列境界検査に使用される最適化を、代わりにユーザ定義型に適応させて適用することができる。例えば、注釈２０６の利点が型２０４に与えられれば、オプティマイザは、そのユーザ定義型２０４の変数に対するループ内のすべてのアクセスは、その変数が許容可能なメモリアドレス境界内であると判定でき、従ってオプティマイザは、ループを実行する結果として発生するであろう複数の境界検査を解除することができる。
【００３８】
図２で提案したように、ユーザ定義型および最適化されたコンパイル時境界検査は、デバイスドライバ２３０コードを開発する際のプログラム１２０として特に有用である。管理コードをシステムプログラミングに使用することができ、そしてデバイスドライバを、明示的に割り当てられないメモリのガーベジコレクションを提供するシステムで実行するＩＤＥ１３８を使用して開発することができる。明示的に割り当てられるメモリ、それゆえガーベジコレクトされないメモリは、境界検査を犠牲にせずに、且つ極めて非効率な境界検査を強いることをせずに開発者のコードによって管理されることができる。例えば、クラス２０２を、注釈付けられたバッファアクセス２０８メソッドまたはバッファを読み取る／書き込むメソッドを用いて、デバイス１３４に明示的に割り当てられるメモリマップドバッファ２３２を含むように定義することができる。バッファのサイズを、動的に判定し、その後ｂｕｆｆｅｒＢｏｕｎｄフィールド２１２またはｇｅｔＢｕｆｆｅｒＢｏｕｎｄ（）メソッド２１４などの、注釈２０６および境界提供２１０機構を経て境界検査コードに提供できる。

【００３９】
図１に関して、いくつかの実施形態は、本明細書で説明するように最適化されたコンパイル時境界検査のサポートによってコード１２２、コード１２４、コード１２６を変換するために回路、ファームウェア、および／またはソフトウェアによって構成される論理プロセッサ１１０およびメモリ媒体１１２をコンピュータシステム１０２に提供する。そのメモリは、論理プロセッサと動作可能に通信する。メモリに常駐するソースコード１２２は、ユーザ定義型２０４を有する。ユーザ定義型は、ユーザ定義の境界検査注釈２０６で注釈付けられる、メモリアクセス２０８メソッドを有する。ユーザ定義型は、境界提供２１０フィールドまたはメソッドなどの、少なくとも１つの境界指定子も有する。メモリに常駐するコンパイラ２２４は、ユーザ定義の境界検査注釈による中間言語コード１２４の境界検査表現２２０に挿入するように構成される。メモリに常駐するオプティマイザ２２６は、重複した境界検査を減らすために、最適化２２８を中間言語コードに適用するように構成される。

【００４０】
いくつかの実施形態において、注釈付けられたソースコードは、デバイスドライバ２３０のソースコード１２２を含み、そしてユーザ定義型２０４は、メモリマップドバッファ２３２に対応する。メモリマップドバッファは、単なる一例であり、いくつかの実施形態において、注釈付けられたデバイスドライバコードは、明示的に割り当てられた他のメモリ１１２にアクセスする。

【００４１】
いくつかの実施形態において、注釈付けられたソースコードは、ガーベジコレクトされたデータ型１３０を含み、そしてユーザ定義型は、明示的に割り当てられるメモリに対応する。いくつかの実施形態において、ユーザ定義型２０４は、コンパイラ２２４によって制御されないデータレイアウト２１６を有する。いくつかの実施形態において、ユーザ定義型２０４は、どの多要素配列型も構成型としないように定義される。他の実施形態において、ユーザ定義型２０４は、１または複数の配列を構成型として有するが、配列よりも複雑である。いくつかの実施形態において、ユーザ定義型２０４は、配列型をラップし、そして補足的な境界検査、例えば、配列に割り当てられた空間内にあるだけでなく、更新された要素を保持している空間内にあるか、または指定された値のセットを保持するように開発者によって意図された配列の下位部分内にある、配列アクセスの検査を提供する。

【００４２】
いくつかの実施形態において、境界指定子（例えば、境界提供２１０機構）は、以下の、ユーザ定義のデータ型２０４のフィールド２１２がメモリアクセス２０８メソッドの境界２１８を包含することを示したフィールド包含境界注釈(field-contains-bound annotation)２０６と、ユーザ定義のデータ型２０４の境界取得メソッド(bound-getting method)２１４がメモリアクセス２０８メソッドの境界２１８を返すことを示した境界ゲッタメソッド注釈(bound-getter-method annotation)２０６とのうちの少なくとも１つを含む。

【００４３】
いくつかの実施形態において、システム１０２は、メモリに常駐する中間言語コード１２４を含み、そしてそのコード１２４は、ユーザ定義の境界検査注釈２０６の境界検査表現２２０で注釈付けられる。いくつかの実施形態において、コンパイラ２２４は、ユーザ定義の境界検査注釈２０６だけでなく、内蔵型１２８にも境界検査コード２２２を挿入するように構成される。いくつかの実施形態において、ユーザ定義の境界検査注釈２０６は、補足的な境界検査を示し、その中でユーザ定義型２０４は、任意のユーザ定義の境界検査注釈２０６が存在しているかどうかにかかわらず、コンパイラ２２４が境界検査を終了するように構成された内蔵型１２８をラップする。

【００４４】
いくつかの実施形態において、人間ユーザのＩ／Ｏデバイス（スクリーン、キーボード、マウス、タブレット、マイクロフォン、スピーカ、動きセンサなど）などの周辺機器１０６は、１または複数のプロセッサ１１０およびメモリと動作可能に通信する際に存在す

る。しかし、ある実施形態は、どの人間ユーザ１０４もその実施形態と直接対話しないように、システムに深く組み込まれることもある。ソフトウェアの処理はユーザ１０４が行ってもよい。

【００４５】

いくつかの実施形態において、システムは、ネットワークによって接続される複数のコンピュータを含む。ネットワークインタフェース機器は、例えば、パケット交換ネットワークインタフェースカード、無線トランシーバ、または電話ネットワークインタフェースなどの、コンポーネントを使用して、ネットワーク１０８へのアクセスを提供することができ、それらのコンポーネントは、コンピュータシステムに存在する。しかし、ある実施形態は、直接メモリアクセス、取り外し可能な不揮発性媒体、または他の情報記憶－読み出しおよび／または送信アプローチを通じて通信することもできるし、またはコンピュータシステムの実施形態は、他のコンピュータシステムと通信せずに動作できる。

【００４６】

処理

図３は、いくつかの処理の実施形態をフローチャート３００で例示している。図面に示した処理は、いくつかの実施形態で自動的に、例えば、ユーザ入力をほとんどまたはまったく必要としないスクリプトの制御下でコンパイラ２２４およびオプティマイザ２２６によって、またはユーザ定義型２０４をユーザ供給仕様(user-supplied specifications)から生成する自動化されたソースコード１２２ジェネレータによって実行されることができる。処理を、一部では自動的に、および表示がなければ一部では手動で実行することもできる。所与の実施形態において、例示された０または１以上の処理のステップを繰り返してもよく、恐らく動作する異なるパラメータまたはデータを用いるであろう。実施形態のステップは、図３に並べられた上から下への順序とは異なる順序で行われてもよい。ステップは、順次に、部分的に重なる方法で、または完全に並行して実行されてもよい。フローチャート３００が処理中に実行されるステップを示すためにトラバースする順序は、ある処理の実行から別の処理の実行までさまざまになり得る。フローチャートのトラバース順序も、ある処理の実施形態から別の処理の実施形態までさまざまになり得る。ステップは、実行される処理が動作可能で、且つ少なくとも１つの特許請求の範囲に従うという条件で、省略され、組み合わされ、名前を変えられ、再グループ化され、あるいは例示されたフローから離れてもよい。

【００４７】

テクノロジーの態様を明らかにするのに役立つ例を本明細書で与えるが、本文書内で与えられる例は、考えられる実施形態をすべて説明しているわけではない。実施形態は、本明細書で与えられる特定の実装、配置、表示、機能、アプローチ、またはシナリオに限定されない。所与の実施形態は、例えば、付加的または異なる機能、機構、および／またはデータ構造を含んでもよく、本明細書で与えられた例から逸脱しなければよい。

【００４８】

ユーザ定義型を特定するステップ３０２において、実施形態は、ソースコードのユーザ定義型２０４を特定する。ステップ３０２は、本明細書で説明するようなユーザ定義型２０４を特定するように適応した、例えば、字句アナライザ、パーサおよび／または他の機構を使用して実現できる。具体的には、よく知られたソースコードの注釈を認識するのに使用される機構を適応させて、キーワードによって注釈２０６を認識できる。

【００４９】

境界検査表現を挿入するステップ３０４において、実施形態は、対応する注釈付けられたソースコードのコンパイル中、境界検査表現２２０を中間言語コード１２４に挿入する。ステップ３０４は、本明細書で説明するような境界検査注釈２０６を表現するために適応した、例えば、解析木、抽象構文木、属性、汎用のドープベクトル、および／または他の機構を使用して実現できる。

【００５０】

最適化を適用するステップ３０６において、実施形態は、最適化（複数）２２８を適用

して、冗長な境界検査を減らすまたはなくす。最適化をソースコード、中間コード、および／または実行可能コードに適用して、実行可能コードで発生するであろう重複した境界検査を減らすことができる。ステップ３０６は、境界検査に従うメモリアクセスは、実行中、結果として許可された境界外のメモリアクセスになるであろう値を想定することができないと分析的に判定することによって実現できる。例えば、ポインタが、コードの第１のポイントにおいて境界検査された場合、およびその境界およびポインタ値が、コードの実行の後に第２のポイントにおいて変更されなかった場合、第２のポイントにおける境界検査は必要ない。別の例として、、ポインタが、コードの第１のポイントにおいて境界検査された場合、および所与の指示で変更されたポインタ値が、コードの実行の後に第２のポイントにおいてその指示で変更された境界ほど変更されなかった場合、第２のポイントにおける境界検査は必要ない。さらに別の例として、メモリアクセスが、コードの実行中にどのフロー制御によっても到達不可能な場合、そのメモリアクセスの境界検査は必要ない。

【００５１】

境界検査コードを挿入するステップ３０８において、実施形態は、対応する注釈付けられたソースコードのコンパイラ中に境界検査コード２２２を実行可能コード１２６に挿入する。いくつかの実施形態は、中間言語コード１２４および実行可能コード１２６を別個に、例えば、別個のファイルに保持し、一方他の実施形態は、中間言語コード１２４および実行可能コード１２６を混合する。従って、ステップ３０８は、中間言語コード１２４として作業しているメモリの同じファイルまたは同じブロックに示される実行可能コード１２６に境界検査コード２２２を挿入することも起こり得る。ステップ３０８は、例えば、解析木、抽象構文木、選択命令、スケジューリング命令、レジスタ割り当て、および／または他の機構を使用して、境界検査コード２２２を挿入するように適応することを実現できる。

【００５２】

境界検査を補足するステップ３１０において、実施形態は、例えば、内蔵型の境界検査または単純な配列型の境界検査などの、すでに提供された境界検査を補足する。ステップ３１０は、例えば、境界検査された構成型を有する型２０４を定義することによって、またはそのような型をコンパイルすることによって実現できる。従って、境界検査を補足するステップ３１０は、境界検査表現を挿入するステップ３０４中および／または境界検査コードを挿入するステップ３０８中に、その挿入が以前に提供された境界検査を補足する場合に発生し得る。境界検査を補足するステップ３１０は、以前に示した境界検査にさらに境界検査を追加するように注釈付けられる型２０４を定義する開発者によって実行されることもできる。

【００５３】

特定の型を定義するステップ３１２において、ユーザは、配列型１３２でない型２０４、即ち、配列型を持たない型２０４を構成型として定義する。クラス２０２は、ユーザ定義型２０４の一例であると見なされる。整数変数などの単一値の変数は、配列の特殊な例と見なされない。ステップ３１２の用途として、配列は、少なくとも２つの要素を有する。ステップ３１２によって定義される型の配列の欠如は、よく知られた配列専用の境界検査と比較して、本明細書で説明するようなコンパイル時境界検査による柔軟性の改善を強調する働きをする。よく知られたソースコード編集ツールおよび開発環境１３８を、開発者によってステップ３１２中に定義される型２０４を受け取るのに使用されてもよい。

【００５４】

ソースコードを取得するステップ３１４において、開発者または開発者の代わりに動作する実施形態は、ソースコード１２２を取得する。ステップ３１４は、ファイルシステム、ネットワーク、ＩＤＥ１３８、および／または他のよく知られた機構を使用して実現できる。

【００５５】

型を指定するステップ３１６において、開発者または開発者の代わりに動作する実施形

態は、ソースコード１２２のユーザ定義型２０４（例えば、ユーザ定義クラス２０２であってよい）を指定する。よく知られたソースコード編集ツールおよび開発環境１３８は、開発者によってステップ３１６中に型２０４を指定するのに使用されてもよい。特定の実施形態において、ステップ３１６は、特定の型を定義するステップ３１２および／または境界検査を補足するステップ３１０を含むことができる。

【００５６】

メソッドを見つけるステップ３１８において、開発者または開発者の代わりに動作する実施形態は、ユーザ定義型２０４によって定義される（例えば、指定されるステップ３１６）メモリにアクセスするメソッド３２０を見つける。そのようなメソッド３２０は、一般にメモリアクセスコードの例である。ステップ３１８は、よく知られたソースコード編集ツールおよび開発環境１３８を使用して実現でき、特に、キーワード検索能力で実現できる。

【００５７】

注釈付けるステップ（複数）３２２において、開発者または開発者の代わりに動作する実施形態は、ユーザ定義型２０４のオブジェクトまたは他の変数を保持するために明示的に割り当てられるメモリの境界検査情報をコンパイラ２２４に提供するソースコードを注釈付ける。例えば、メモリアクセスコードは、ユーザ定義の境界検査(user-defined-bounds-check)３２４の注釈２０６で注釈付けられ、それは、注釈２０６を持たない言語環境によってどんな検査が提供されても、そのコードが、明示的に境界検査に割り当てられるおよび／あるいは従うメモリにアクセスする（またはアクセスできる）ことを示す。明示的に割り当てられるメモリにアクセスする（またはアクセスできる）コードは、明示的に割り当てられるメモリへのアクセス(accesses-explicitly-allocated-memory)３２６の注釈２０６によってコンパイラ２２４に特定されることができる。境界提供２１０の注釈２０６は、境界を包含するフィールド(field-contains-bound)３２８の注釈２０６で注釈付けるステップ３２２のフィールド２１２によって、または境界ゲッタメソッド（bound-getter-method）３３０の注釈２０６で注釈付けるステップ３２２のメソッド２１４によってなどで境界を示す、注釈付けるステップ３２２の機構に置かれてもよい。

【００５８】

ラップするステップ３３２において、開発者または開発者の代わりに動作する実施形態は、ユーザ定義型２０４の既存の型をラップする。即ち、ユーザは、既存の型を構造型として有する、型２０４を定義（指定する）し、それによって、必要であれば、既存の型の境界検査を補足するステップ３１０を用いる。ステップ３２２は、よく知られたソースコード編集ツールおよび開発環境１３８を使用して実現できる。

【００５９】

第２の境界を表すステップ３３４において、開発者または開発者の代わりに動作する実施形態は、ユーザ定義型２０４の第２の境界２１８の条件を表す。即ち、ユーザは、異なる境界を指定することによってユーザ定義型の境界検査を補足するステップ３１０を用いる。例えば、型は、割り当てられた全メモリを反映する第１の境界を有し、および割り当てられたメモリの実際の使用を反映する第２の境界も有することができ、例えば、「古い(obsolete)」とマークされたレコードは、たとえメモリに常駐していてレコードを保持するために割り当てられていても、境界外と見なされる。ステップ３２４は、よく知られたソースコード編集ツールおよび開発環境１３８を使用して実現できる。

【００６０】

重複した境界検査を減らすステップ３３６において、実施形態は、例えば、少なくとも１つの冗長な境界検査を見つけて、そして解除するのに成功した、最適化（複数）２２８を適用するステップ３０６によって、重複した境界検査を減らす。

【００６１】

コンパイルするステップ３３８において、実施形態は、注釈付けるステップ３２２のソースコードをコンパイルする。ステップ３３８は、よく知られたコンパイルツール、および本明細書で説明するようなユーザ定義型のコンパイル時に最適化される境界検査(compi

le-time-optimized bounds checking)を提供するように適応された技術を使用して実現できる。

【００６２】

メモリを構成するステップ３４０において、メモリ媒体１１２は、ユーザ定義型２０４と、コンパイラ２２４を最適化する２２６と、および／あるいは本明細書で説明するようなユーザ定義型のコンパイル時最適化境界検査に接続することによって構成される。

【００６３】

前述のステップおよびそれらの相互関係は、さまざまな実施形態に関連して以下により詳細に論じる。

【００６４】

いくつかの実施形態は、ユーザデータ型のコンパイル時境界検査の処理を提供する。この処理は、ソースコードから中間言語コードを通じて実行可能コードに流れるプログラムのコンパイル中に実行されるステップを含む。ソースコードのユーザ定義クラス２０２または他の型２０４の特定は、特定するステップ３０２を用いる。ユーザ定義クラスは、例えば、明示的に割り当てられるメモリに安全にアクセスすることを意図する場合もある。いくつかの実施形態において、ユーザ定義クラスは、多要素配列型でない任意の型を構成型として定義されるステップ３１２を用いる。ユーザ定義クラスは、メモリアクセス２０８コードを注釈メンバとして有することができ、そのコードの注釈は、ユーザ定義の境界検査３２４の注釈２０６で注釈付けるステップ３２２を用いる。クラス２０２は、境界提供２１０メンバを注釈メンバとしても有することができ、その提供メンバの注釈は、メモリアクセスコードの境界検査を生成するための境界２１８情報を提供することを示すように注釈付けるステップ３２２を用いる。注釈付けられた型２０４に応答して、ユーザ定義の境界検査の注釈の境界検査表現２２０は、中間言語コードに挿入されるステップ３０４を用いて、場合によっては、最適化２２８は、実行可能コードで発生するであろう重複した境界検査を減らすために適用するステップ３０６を用いる。

【００６５】

いくつかの実施形態において、処理は、境界検査表現に応答して、境界検査コードを中間言語に挿入するステップ３０８を含み、ステップの適用は、最適化を境界検査表現（複数）に適用するよりはむしろ、挿入された境界検査コードに最適化を適用するステップ３０６を用いる。いくつかの実施形態において、ステップの適用は、最適化を境界検査コード２２２に適用するよりはむしろ、最適化を境界検査表現（複数）２２０に適用するステップ３０６を用いる。

【００６６】

いくつかの実施形態において、ステップの特定は、明示的に割り当てられるメモリへのアクセス３２６の注釈で注釈付けられたステップ３２２のメモリにアクセスするメソッド３２０を特定するステップ３０２を用いる。いくつかの実施形態において、ステップの特定は、内蔵型１２８のシステム定義の境界検査(system-defined bounds checking)を補足するステップ３１０を意図する、ユーザ定義の境界検査を示す注釈２０６を特定するステップ３０２を用いる。いくつかの実施形態において、ステップの特定は、ガーベジコレクタ管理型１３０のシステム定義の境界検査を補足するステップ３１０を意図した、ユーザ定義の境界検査を示す注釈２０６を特定するステップ３０２を用いる。

【００６７】

いくつかの実施形態は、ユーザ定義型、即ち、内蔵ではない型のコンパイル時境界検査を管理する処理をプログラム開発者に提供する。その処理は、コンピュータプログラムのソースコード取得するステップ３１４と、（例えば、型２０４を書き込むことによってまたは以前に書き込まれた型を受け取ることによって）そのソースコードのユーザ定義型を指定するステップ３１６とを含む。処理はまた、ユーザ定義のデータ型によって定義される、メモリにアクセスするメソッド３２０を見つけるステップ３１８と、例えば、明示的に割り当てられるメモリへのアクセス３２６または他の注釈２０６でメモリにアクセスするメソッドを注釈付けるステップ３２２とを含む。さらに、処理は、以下の、ユーザ定義

のデータ型２０４によって定義されるフィールド２１２が、メモリにアクセスするメソッドの境界２１８の値を包含することを示した、境界を包含するフィールド３２８の注釈２０６と、ユーザ定義のデータ型によって定義される境界取得メソッド２１４が、メモリにアクセスするメソッドの境界２１８の値を返すことを示した、境界ゲッタメソッド３３０の注釈２０６とのうちの少なくとも１つでソースコードを注釈付けるステップ３２２を含む。

【００６８】

いくつかの実施形態において、開発者は、メモリにアクセスするメソッドの第２の境界を表すステップ３３４の注釈２０６でソースコードを注釈付けるステップ３２２を用いる。いくつかの実施形態において、ユーザ定義型２０４は、内蔵された１２８の配列型１３２をラップするステップ３３２を用いる。いくつかの実施形態において、ユーザ定義型２０４は、内蔵された１２８の管理型１３０をラップするステップ３３２を用いる。

【００６９】

構成媒体

いくつかの実施形態は、構成されたコンピュータ可読記憶媒体１１２を含む。媒体１１２は、ディスク（磁気ディスク、光ディスクその他）、ＲＡＭ、ＥＥＰＲＯＭまたは他のＲＯＭ、および／または、特に持続性のコンピュータ可読媒体（有線または他の伝搬信号媒体とは対照的である）を含む、他の構成可能メモリを含んでもよい。構成される記憶媒体は、特に、ＣＤ、ＤＶＤ、またはフラッシュメモリなどの、取り外し可能な記憶媒体１１４であってもよい。取り外し可能でも取り外し不能でもよく、且つ揮発性でも不揮発性でもよい、汎用メモリは、ユーザ定義型２０４（それらの注釈２０６を含む）、および／または（注釈２０６を処理するように適応した）オプティマイザ２２６などの、項目を使用した実施形態に、取り外し可能な媒体１１４および／またはネットワーク接続などの別のソースから読み取って構成媒体を形成する、データ１１８および命令１１６の形式で構成されることができる。構成媒体１１２は、コンピュータシステムに、本明細書で説明するような注釈とコンパイル時に最適化される柔軟性のある境界検査とを通じて、ソースコードまたは他のコードを変換する処理ステップを実行するようにさせる能力がある。従って図１から図３までは、構成された記憶媒体の実施形態および処理の実施形態、ならびにシステムおよび処理の実施形態を明らかにするのに役立つ。特に、図３に例示した、あるいは本明細書で教示した処理ステップのいずれも、構成媒体の実施形態を形成する記憶媒体を構成するのに役立つように使用できる。

【００７０】

付加的な例

付加的な詳細および設計考察を以下に与える。本明細書の他の例と同様に、説明する機能は、所与の実施形態において、個々におよび／または組み合わせて使用されてもよいし、または全く使用されなくてもよい。

【００７１】

当業者は、実装の詳細は、専用ＡＰＩおよび専用サンプルプログラムなどの、特定のコードに関連する場合もあり、従って、それぞれの実施形態で示す必要がないことを理解するであろう。当業者は、詳細を論じるのに用いるプログラム識別子および他のテクノロジーは、実装時固有のものであり、従ってそれぞれの実施形態に関連する必要がないことも理解するであろう。それらの詳細は、必ずしもここで提示される必要があるわけではないが、そのような詳細は、一部の読み手に文脈を与えて読み易くできるし、および／または本明細書で論じたテクノロジーの多くの考えられる実装のうちのいくつかを明らかにできるため、提供される。

【００７２】

本明細書で説明するいくつかの実施形態は、以下の態様を与える。

【００７３】

第一に、プログラマに、明示的に割り当てられるメモリに安全な形でアクセスするデータ型２０４（例えば、クラス２０２）を定義するようにさせる方法である。プログラマは

、プログラマがデータ型によって定義されるメソッドに置く、注釈２０６のセットを使用することができる。ある種類の注釈２０６は、注釈付けられたメソッドが、明示的に割り当てられるメモリにアクセスし、そして境界検査によって保護されなければならないことを示す。別の種類の注釈２０６は、データ型のフィールド２１２が、メモリアクセスの境界２１８を包含することを示す。３番目の種類の注釈２０６は、データ型のメソッド２１４が、メモリアクセスの境界を返すことを示す。２番目の種類の注釈または３番目の種類の注釈のいずれか（または両方）を、第１の種類の注釈の特定のインスタンスに使用できる。

【００７４】

第二に、コンパイラ２２４は、このような注釈２０６を、その中間表現で、即ち、中間言語コードで表現する。

【００７５】

第三に、中間表現の注釈に基づいて、コンパイル２２４は、明示的に割り当てられるメモリにアクセスするメソッド３２０を呼び出す前に、境界検査を挿入する。

【００７６】

第四に、境界検査を挿入するステップ３０４／３０８の後、コンパイラは、不必要な境界検査を減らす（なくすこともあり得る）ステップ３３６の最適化を行う。このような最適化２２８は、注釈付けられた中間表現を理解し、そして単なる配列よりも複雑である明示的に割り当てられるメモリにアクセスするメソッドを呼び出す前に挿入される境界検査をなくすように配列の最適化を拡張することによって、リテラチャー(literature)によくある最適化によって現在の文脈で使用されるように適応されることができる。コンパイラ２２４または他のオプティマイザ２２６は、境界アクセスまたは境界アクセスを返すメソッドを包含するフィールド、および機能を呼び出す前に挿入される検査を比較して特定し、その後その検査を安全になくすことができるかどうかを（アレイの不必要な境界検査をなくすためのよく知られた技術によって適応されるアプローチを経て）シンボルで判定する。

【００７７】

このようにして、プログラマは、明示的に割り当てられるメモリを比較的安全な形で使用することができる。これによって、明示的に割り当てられるメモリに管理コードで効率的よく、安全にアクセスすることができるようになる。

【００７８】

いくつかの実施形態において、コンパイラ２２４は、単なる配列ではないデータ構造を含む、データ構造に任意に適用することができる属性のセットを提供する。この方法で、これらの実施形態は、一般化され、そして例えば、内蔵言語型のような配列に特有な作業などの、境界検査の初期作業に柔軟性を付加する。これらの実施形態は、プログラマが、境界検査をそのプログラマによって定義される代替データ構造に適用できるようにさせ、特に、コンパイラおよび言語システムがデータレイアウト２１６を制御しないか、またはそこでのデータレイアウト２１６が任意であってよい状況で境界検査ができるようにさせる。

【００７９】

いくつかの実施形態は、プログラミング言語による配列の実装に使用された「ドープベクトル」の考えから思い付いたものであり、プログラマが、プログラミング言語の実装によって定義されるデータ構造の代わりに、境界検査されるデータ構造を定義できる実施形態に到達する過程でその概念を変更している。よく知られたドープベクトルは、配列要素、配列境界、および可能であれば他の情報を包含するメモリブロックのポインタを包含する。いくつかの実施形態は、プログラムが、よく知られた配列境界検査と本明細書で教示した柔軟性のあるユーザ定義型２０４の境界検査との両方に対してより効率的になるように、配列境界検査をなくす、よく知られた作業に統合されるまたは統合されることができる。

【００８０】

いくつかの実施形態において、注釈２０６は、必要に応じてランタイム時に強制される、ライブラリ動作のハイレベルではない意味のプロパティ(semantic property)の正確性検査を記述する。オプティマイザ２２６は、不必要な検査をなくすように試みる。他の作業では、対照的に、オプティマイザは、ライブラリの意味のプロパティを記述する注釈を使用して、そのライブラリの使用法(usage)を、本明細書で説明するような不必要で安全な検査を減らすステップ３３６を用いずに、最適化する。

【００８１】

いくつかの実施形態は、ジャストインタイム（ＪＩＴ）コンパイル、ガーベジコレクション（ＧＣ）、ランタイムリフレクション他などの、サービスおよび機能を含む比較的大規模なランタイムである、Ｍｉｃｒｏｓｏｆｔ（登録商標）の共通言語ランタイム（ＣＬＲ）を包含したオペレーティング環境１００を有する。一部のＧＣが、型安全性のために提供されることもあるが、いくつかの実施形態は、従来のコンパイル（事前コンパイルと呼ばれることもある）を用いてＣ言語モデルに密接に従う。

【００８２】

いくつかの実施形態において、Ｃ＃は、以下のメソッドの属性に従って使用することができるように、返り値を注釈付けできるようにする。

［ｒｅｔｕｒｎ；ＳｏｍｅＡｔｔｒｉｂｕｔｅ］

ｉｎｔＳｏｍｅＭｅｔｈｏｄ（）｛．．．｝

【００８３】

いくつかの実施形態において、コンパイラ生成およびコンパイラ解除の境界検査(compiler-generated and compiler-removed bounds checks)は、、管理されていないリソース、典型的には、メモリのインデックスプールを有するデータ構造に利用可能であるが、これに限定されない。プログラマは、コンパイラ２２４が、ランタイム時に（例えば、境界違反の例外を送出することによって）配列境界検査と同様にふるまう境界検査と、よく知られたオンデマンドの配列境界検査（ＡＢＣＤ）アプローチまたはＡＢＣＤの代わりとなるよく知られた最適化で適応したアプローチによって解除される境界検査とを生成するように、プログラマのデータ構造を注釈付けることができる。

【００８４】

いくつかの実施形態において、３つのカスタム属性を提供する。

【００８５】

ＢｏｕｎｄｓＣｈｅｃｋｉｎｇ属性は、注釈付けられるメソッド３２０に適用される。それに応じて、コンパイラ２２４は、呼位置(call site)における境界検査をＢｏｕｎｄｓＣｈｅｃｋｉｎｇとマークされたメソッドに挿入する。一実施形態において、コンパイラ２２４は、ＢｏｕｎｄｓＣｈｅｃｋｉｎｇメソッドに、少なくとも１つの引数を有することを要求し、そしてその第１の引数がＩｎｔ３２型であることを要求する。境界検査は、第１の引数がゼロとＢｏｕｎｄとマークされたフィールド（以下を参照）との間であることを検査する。この実施形態において、ＢｏｕｎｄｓＣｈｅｃｋｉｎｇメソッドを有するすべての型は、Ｂｏｕｎｄとマークされたのと同じ１つのＩｎｔ３２フィールドを有する。ＢｏｕｎｄｓＣｈｅｃｋｉｎｇを付加することによる安全検査の解除は、ブレイク変更と見なさなければならない。

【００８６】

Ｂｏｕｎｄ属性は、注釈付けられるフィールド２１２に適用される。一実施形態において、そのフィールドは、Ｉｎｔ３２であり、同じ型２０４のＢｏｕｎｄｓＣｈｅｃｋｉｎｇメソッドによって生成される境界検査によって使用される。

【００８７】

ＢｏｕｎｄＧｅｔｔｅｒ属性は、注釈付けられるメソッド２１４に適用される。一実施形態において、Ｂｏｕｎｄを返すメソッドがインライン化されない場合、そのメソッドは、ＢｏｕｎｄＧｅｔｔｅｒとマークされ、そしてそのメソッドの呼は、そのＢｏｕｎｄへのアクセスとして処理される。

【００８８】

いくつかの実施形態において、コンパイラ２２４は、上記の要件を検査するが、Ｂｏｕｎｄが意味のあるフィールドに唯一適用されること、およびＢｏｕｎｄＧｅｔｔｅｒがＢｏｕｎｄ（またはそのＢｏｕｎｄ未満の値）を返すメソッド（複数）に唯一適用されることを確認するのは、プログラマの責任である。境界検査を解除するための最適化に基づいて適応した配列を有する一実施形態において、その境界検査は、Ｂｏｕｎｄフィールドが変化していない場合、安全でない解除が行われる恐れがある。

【００８９】

いくつかの実施形態は、アプローチに基づく型をとり、そしてユーザが配列に似た形の型を書き込むことを予測し、そしてユーザにその形（レングスメソッドの場所、アクセス機構の場所）をコンパイラに記述するように求める。しかし、構造型は、それぞれの実施形態では必要ない。いくつかの実施形態において、関与する特性は、ユーザコードの位置が、あるユーザ変数の検査を必要とし、そしてコンパイラは、その検査を構築する方法を命令されるという要因を含む。いくつかの実施形態は、実装の選択として、配列検査［０，長）に似た利用可能な検査の形をとる。

【００９０】

いくつかの実施形態は、メソッド３２０に［ＢｏｕｎｄｓＣｈｅｃｋｉｎｇ］を付ける。いくつかの実施形態はまた、以下の例のような、他のメモリアクセス２０８コードでそれらを直接ソースコードに置く。

```
ｖｏｉｄＦｏｏ（ｉｎｔ　ｉ）｛
　ｂｙｔｅ＊ｐ＝．．．
　［ＢｏｕｎｄｓＣｈｅｃｋｉｎｇ］（ｏｒ［ＢｏｕｎｄｓＣｈｅｃｋｉｎｇ（ｉ）］
　．．．＊（ｐ＋ｉ）．．．
｝
```

【００９１】

実際、この種類の注釈は、あるソース言語の標準から外れているかもしれない。それは、ＥＣＭＡ３３５標準の実装である、マイクロソフト中間言語（ＭＳＩＬ）に従っていないが、他の言語に従うことができる。

【００９２】

いくつかの実施形態は、明示的に注釈付けられるメモリの保護に限定されない。例えば、ある実施形態では、ラップするステップ３３２の管理配列がこのようになる。

```
ｃｌａｓｓ　Ｌｉｓｔ｛
　ｉｎｔ［］ａｒｒ＝ｎｅｗ　ｉｎｔ［２０］；
　［Ｂｏｕｎｄ］
　ｉｎｔ　ｃｏｕｎｔ＝０；
　ｖｏｉｄ　Ａあｄｄ（ｉｎｔ　ｉ）｛
　　ａｒｒ［ｃｏｏｕｎｔ］＝ｉ；
　ｃｏｕｎｔ＝ｃｏｕｎｔ＋１；
｝
［ＢｏｕｎｄｓＣｈｅｃｋｉｎｇ］
ｖｏｉｄ　Ｇｅｔ（ｉｎｔ　ｉ）｛
　　ｒｅｔｕｒｎ　ａｒｒ［ｉ］；
　｝
｝
```

【００９３】

この例において、プログラム言語は、ａｒｒで既存の配列境界検査を提供するが、開発者は、ｉが２０未満であることだけでなく、リストに付加された項目数よりも少ないことも確認するためにより強固な補足的な検査も必要とする。両方の検査は、配列境界検査および／または他の最適化が済むと解除の候補となる。

【００９４】

排除

本明細書で説明される柔軟性のあるコンパイル時境界検査と先行アプローチとの違いをさらに明らかにするために、よく知られた配列境界検査の考察を以下に与える。この考察で説明する概念およびテクノロジーは、本明細書で教示された実施形態と互いに使用されないようにするという点において、実際に置き換えられるが、それらは、ここでは保護が求められる実施形態の範囲外にある。

【００９５】

境界属性の収束の文脈において、あるアプローチは、ポインタまたはＣ＃配列または配列型のフィールド、パラメータ、または返り値で示すことができる属性を記す。

【００９６】

［ＳｔａｔｉｃＢｏｕｎｄ（ｎ）］、ここでのｎは、ある整数リテラルである。

【００９７】

［ＢｏｕｎｄｅｄＢｙ（ｉｄｅｎｔ）］、ここでのｉｄｅｎｔは、以下のいずれかの識別子である。

【００９８】

即座に構造を包含する同じメンバである整数型のその他のフィールド。

【００９９】

または、同じ手順／メソッドのその他の仮パラメータ。

【０１００】

または、返り値の例において、実際にメソッドにアタッチされる返り値。

【０１０１】

これらの識別子は、引数が｛ｓｔｒｉｎｇ｜ｉｎｔ｝と定義されることが許可される場合、単一の属性に折り畳まれる。第２の属性名を必要としなくてもよい。

【０１０２】

これらの属性を搬送するフィールド／パラメータがポインタである場合、属性のプレゼンスは、そのポインタを通じたインデックス動作を検査しなければならないというコントラクトを搬送する。社会的観点から、バックオフを適合した方が逆のやり方よりも簡単であることに留意されたい。

【０１０３】

パラメータ、フィールド、またはインデックス（ある整数型によって構成されなければならない）として機能する返り値で示すことができる属性。

【０１０４】

［Ｒａｎｇｅ（ｂｅｇｉｎ，ｅｎｄ）］、ここでのｂｅｇｉｎおよびｅｎｄは、整数リテラル、またはｓｔｒｉｎｇで符号化される識別子のいずれであってもよく、そして通常の予測では、ｂｅｇｉｎは、ゼロのリテラル定数になる。

【０１０５】

この属性が、仮パラメータで表示される時、仮パラメータは、コーラ(caller)が実パラメータを範囲検査／解除しなければならないことを示す。

【０１０６】

この属性が、フィールドで表示される時、フィールドは、右側(RHS)の割り当てまたは初期化が範囲検査または同等に解除されなければならないことを示す。

【０１０７】

この属性が、メソッドで表示される時、メソッドは、そのメソッドが値を返す前に検査／解除しなければならないという返り値の用件を示す。

【０１０８】

マングリングおよびラッピングを通じたバブルバージョンに関して、それが望ましいと証明されれば、関与しないコーラ(oblivious caller)と互換するようにダウンロードすることが可能である。

【０１０９】

範囲をより精密に保存できる。

【０１１０】

［Ｒａｎｇｅ（ｉｎｃｌｕｓｉｖｅＢａｓｅ，ＥｘｃｌｕｓｉｖｅＢｏｕｎｄ）］

【０１１１】

排他境界の損得に関して、ある不利点は、ｉｎｔｓでの符号化（例）ＭＡＸＩＮＴが不可能なことである。代替［Ｒａｎｇｅ（ｉｎｃｌｕｓｉｖｅＢａｓｅ，ＩｎｃｌｕｓｉｖｅＢｏｕｎｄ）］は、ユースケース形式：［Ｒａｎｇｅ（０，ｂｏｕｎｄ｜ｄｅｎｔ－１）］にほとんど変換しないであろう。それによって、動作がぎこちなく見え、属性の表現に問題が生じる。

【０１１２】

その問題は、表現可能な最大値を範囲に含めなければならない場合に、異なる属性を使用することによって直接解決することができる。

【０１１３】

［ＡｔＬｅａｓｔ（ｌｏｗｅｒＢｏｕｎｄ）］ｏｒ［ＧｒｅａｔｅｒＴｈａｎＯｒＥｑｕａｌＴｏ（ｌｏｗｅｒＢｏｕｎｄ）］

【０１１４】

どのＣ＃型もその型の結果として範囲境界を元来有するという事実に基づいて、下位のパラメータを解放して、上位境界を暗示的に指定する。

【０１１５】

ＢｏｕｎｄｅｄＢｙおよびＲａｎｇｅ属性を分離することができる。

【０１１６】

属性は、所与のパラメータの位置において「ｉｎｔ関数またはｓｔｒｉｎｇ関数」をとるように指定されてよい。ある属性は、その属性が同じ位置で異なる型をとることができるように、複数のコンストラクタをカスタム属性用に書き込むか、またはこれを名前付きパラメータで行うことができる。

【０１１７】

これで排除の考察を終了する。

【０１１８】

結論

特定の実施形態を、本明細書では、処理として、構成媒体として、またはシステムとして明確に説明しているが、ある種類の実施形態の考察はまた、概して別の種類の実施形態にまで及ぶことが認識されよう。例えば、図３と関連した処理の説明も、構成媒体を説明するのに役立ち、そして他の図面と関連して論じたようなシステムおよび製品の動作を説明するのにも役立つ。それは、一実施形態の制限を、必然的に別の実施形態を制限する意味に解釈するということではない。特に、処理は、システムまたは構成メモリなどの製品を論じている間に提示されたデータ構造および配置に必ずしも限定されるというわけではない。

【０１１９】

図面に示したすべての項目が、どの実施形態にも存在するというわけではない。反対に、１つの実施形態は、図面に明確に示した項目（複数）を包含してよい。いくつかの可能性を、ここでは特定の例による文章および図で説明しているが、実施形態は、そのような例から逸脱してもよい。例えば、１つの例の特定の機能は、省略され、名前を変えられ、異なってグループ化され、繰り返され、ハードウェアおよび／またはソフトウェアに異なってインスタンス生成され、または例のうちの２または３以上に示される機能の組み合わせにされてもよい。いくつかの実施形態において、ある場所で示した機能性は、異なる場所で提供されてもよい。

【０１２０】

参照は、参照番号によって図全体で行われている。所与の参照番号と関連する言い回しにおいて、図面においてまたは文章においての明白な矛盾のいずれも、その番号によって参照された内容の範囲を単に広げたものとして理解されたい。

【０１２１】

本明細書では、「１つの(a)」または「その(the)」などの用語は、示された項目または

ステップのうちの１または複数を包括する。特に、特許請求の範囲において、項目への言及は、概してそのような項目の少なくとも１つが存在することを意味し、そしてステップへの言及は、そのステップのうちの少なくとも１つのインスタンスが実行されることを意味する。

【０１２２】

見出しは、唯一便宜上のものである。所与の論説(topic)についての案内は、見出しがその論説を示す節の外側に見られる。

【０１２３】

出願されるすべての特許請求の範囲は、この明細書部分である。

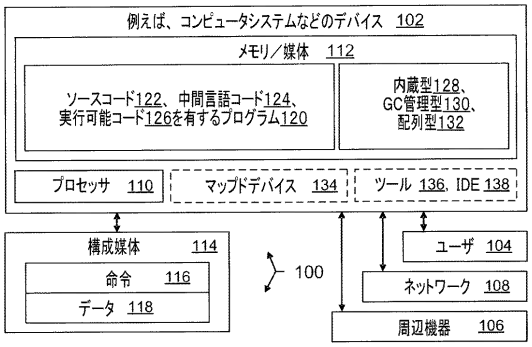
【０１２４】

模範的な実施形態を上記に図示して説明しているが、多くの変更は、本発明の原理および特許請求の範囲に記載した概念から逸脱せずに行うことが可能であり、そしてそのような変更は、すべての抽象的概念を網羅する必要がないことが当業者には明らかであろう。本発明の主題は、構造的特徴および／または手順の動作に特有の用語で説明されているが、添付の特許請求の範囲で定義された本発明の主題は、特許請求の範囲の上記で説明された特定の特徴または動作に必ずしも限定されるわけではないことを理解されたい。所与の定義または例で特定されたすべての手段または態様が、必ずしもすべの実施形態において提示されまたは利用される必要があるわけではない。むしろ、説明した特定の特徴または動作は、特許請求の範囲を実装する場合に考慮される例として開示される。

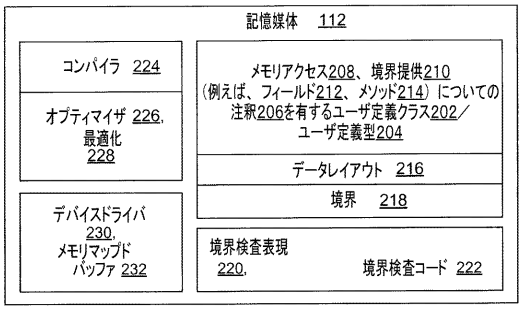
【０１２５】

すべての抽象的考えを網羅する範囲内であるが、特許請求の範囲の等価の意味および範囲内であるすべての変更は、法律の及ぶ限りそれらの範囲内に包含されるものとする。

【図１】



【図２】



【図３】

フロントページの続き

(72)発明者 ダニエル スティーブン ハーベイ
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション エルシーエー－インターナショナル パテンツ内

審査官 坂庭 剛史

(56)参考文献 特開平１０－２７５０８７（ＪＰ，Ａ）
特開２０００－０８１９８３（ＪＰ，Ａ）
Matthew M. Papi et al., “Pluggable Type-checking for Custom Type Qualifiers in Java”, Computer Science and Artificial Intelligence Laboratory Technical Report, 米国, MIT, ２００７年 9月１７日, pp.1-10, MIT-CSAIL-TR-2007-047
酒匂 寛, “Ｅｉｆｆｅｌ－仕様記述能力をもつオブジェクト指向言語－”, 情報処理, 日本, 社団法人情報処理学会, １９９４年 3月１５日, 第３５巻, 第3号（通巻３４９号）, ｐｐ. ２０４－２１４, ISSN 0447-8053

(58)調査した分野(Int.Cl., ＤＢ名)
Ｇ０６Ｆ ９／４５