

(19)日本国特許庁(JP)

(12)公表特許公報(A)

(11)公表番号

特表2023-538811

(P2023-538811A)

(43)公表日 令和5年9月12日(2023.9.12)

(51)国際特許分類

G 0 6 F 9/44 (2018.01)

G 0 6 F 8/41 (2018.01)

F I

G 0 6 F 9/44

G 0 6 F 8/41

テーマコード (参考)

5 B 0 8 1

5 B 3 7 6

審査請求 未請求 予備審査請求 未請求 (全41頁)

(21)出願番号 特願2023-503140(P2023-503140)
(86)(22)出願日 令和3年5月3日(2021.5.3)
(85)翻訳文提出日 令和5年3月2日(2023.3.2)
(86)国際出願番号 PCT/US2021/030388
(87)国際公開番号 WO2022/046192
(87)国際公開日 令和4年3月3日(2022.3.3)
(31)優先権主張番号 17/008,270
(32)優先日 令和2年8月31日(2020.8.31)
(33)優先権主張国・地域又は機関
米国(US)
(81)指定国・地域 AP(BW,GH,GM,KE,LR,LS,MW,MZ,NA
,RW,SD,SL,ST,SZ,TZ,UG,ZM,ZW),EA(
AM,AZ,BY,KG,KZ,RU,TJ,TM),EP(AL,A
T,BE,BG,CH,CY,CZ,DE,DK,EE,ES,FI,FR
,GB,GR,HR,HU,IE,IS,IT,LT,LU,LV,MC,
最終頁に続く

(71)出願人 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 9 8 0 5
2 レッドモンド ワン マイクロソフト
ウェイ
(74)代理人 100079108
弁理士 稲葉 良幸
(74)代理人 100109346
弁理士 大貫 敏史
(74)代理人 100117189
弁理士 江口 昭彦
(74)代理人 100134120
弁理士 内藤 和彦
(74)代理人 100108213

最終頁に続く

(54)【発明の名称】 フォーリンコードのエミュレーションをサポートするためのネイティブエミュレーション
互換可能アプリケーションバイナリインタフェース

(57)【要約】

関数は、ネイティブの第一の命令セットアーキテクチャ (I S A) の第一のアプリケーションバイナリインタフェース (A B I) 及び第二の A B I に向けてコンパイルされる。第二の A B I は、フォーリンの第二の I S A の第三の A B I により想定されるサイズを超えないコンテキストデータを定義し、第一の I S A のレジスタのうち第二の I S A のレジスタにマッピングされるサブセットを使用する。第二の A B I によるレジスタのサブセットの使用により、第一及び第二の A B I の両方を使用したコンパイル時に、幾つかの関数を折り畳むことができる。関数の第一及び第二のコンパイルバージョンは、コンパイルバージョンがマッチするか否かに基づいて、折り畳み可能又は不能として識別される。第一及び第二のコンパイルバージョンは、それらが折り畳み不能である場合は両方ともバイナリファイルへと送信され、これらが折り畳み可能である場合は、第一又は第二のコンパイルバージョンの一方のみがバイナリファイルに送信される。

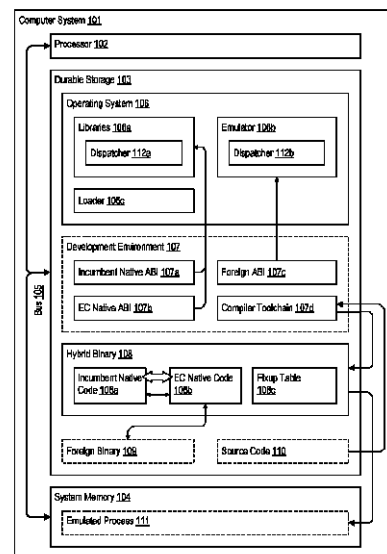


FIG. 1

【特許請求の範囲】

【請求項 1】

ソースコードを、ネイティブの第一の命令セットアーキテクチャ（ISA）をターゲットとする一方で、フォーリンの第二の ISA にコンパイルされたコードと動作可能なバイナリファイルにコンパイルする、プロセッサを含むコンピュータシステムで実行される方法であって、前記方法は、

（i）前記第一の ISA に対応する第一のアプリケーションバイナリインタフェース（ABI）、（ii）前記第一の ISA に対応する第二の ABI、及び（iii）前記第二の ISA に対応する第三の ABI を識別することであって、前記第二の ABI は、

前記第三の ABI により使用される第二のコンテキストデータフォーマットの第二のサイズを超えない第一のサイズを有する第一のコンテキストデータフォーマットを定義し、

前記第一の ISA の複数のレジスタのうちの、前記第一の ABI により使用されるレジスタのサブセットの使用を定義し、前記レジスタのサブセットは前記第三の ABI により使用される前記第二の ISA のレジスタセットにマッピングされ、前記第二の ABI による前記レジスタのサブセットの使用の結果、前記第一の ABI と前記第二の ABI の各々を使ってコンパイルされる際に関数の少なくとも一部が折り畳み可能となる、識別することと、

前記ソースコードで定義される関数について、少なくとも、

前記第一の ABI をターゲットとする前記関数の第一のコンパイルバージョンを生成することであって、前記第一の ISA の第一の命令セットを生成することを含む、生成することと、

前記第二の ABI をターゲットとする前記関数の第二のコンパイルバージョンを生成することであって、前記第一の ISA の第二の命令セットを生成することを含む、生成することと、

前記関数の前記第一のコンパイルバージョンと前記関数の前記第二のコンパイルバージョンが前記バイナリファイルの中で折り畳み可能であるか否かを、少なくとも前記第一の命令セットと前記第二の命令セットがマッチするか否かを特定することに基づいて特定することと、

少なくとも前記関数の前記第一のコンパイルバージョンと前記関数の前記第二のコンパイルバージョンが前記バイナリファイルの中で折り畳み可能であるか否かを特定することに基づいて、

前記関数の前記第一のコンパイルバージョンと前記関数の前記第二のコンパイルバージョンが前記バイナリファイル内で折り畳めないとき特定された場合に、前記関数の前記第一のコンパイルバージョンと前記関数の前記第二のコンパイルバージョンの両方を前記バイナリファイルの中に送信すること、又は、

前記関数の前記第一のコンパイルバージョンと前記関数の前記第二のコンパイルバージョンが前記バイナリファイルの中で折り畳み可能であると特定された場合に、前記関数の前記第一のコンパイルバージョン又は前記関数の前記第二のコンパイルバージョンの一方のみを前記バイナリファイルの中に送信すること

のうちの一方を行うことと、

を実行することと、

を含む方法。

【請求項 2】

前記関数の前記第一のコンパイルバージョンと前記関数の前記第二のコンパイルバージョンが前記バイナリファイルの中で折り畳み可能ではないことを特定することと、その結果、前記関数の前記第一のコンパイルバージョンと前記関数の前記第二のコンパイルバージョンの両方を前記バイナリファイルの中に送信することと、を含む、請求項 1 に記載の方法。

【請求項 3】

前記関数の前記第一のコンパイルバージョンと前記関数の前記第二のコンパイルバージョンが前記バイナリファイルの中で折り畳み可能であることを特定することと、その結果、前記関数の前記第一のコンパイルバージョン又は前記関数の前記第二のコンパイルバージョンの一方のみを前記バイナリファイルの中に送信することと、を含む、請求項 1 に記載の方法。

【請求項 4】

前記第二の A B I は、前記バイナリファイルが (i) 前記第一の A B I を実装するが、前記第二の A B I を実装しない第一のコンピュータシステムと、 (i i) 前記第二の A B I を実装する第二のコンピュータシステムの両方でネイティブとして実行できるようにする、請求項 1 に記載の方法。

10

【請求項 5】

前記バイナリファイルの中にエントリサンクを送信することをさらに含み、前記エントリサンクは、前記第三の A B I を前記第二の A B I に適応させる、前記第一の I S A の中のコードを含む、請求項 1 に記載の方法。

【請求項 6】

前記バイナリファイルの中に 1 つ又は複数のエグジットサンクを送信することをさらに含み、各エグジットサンクは、前記第二の A B I を前記第三の A B I に適応させる、前記第一の I S A の中のコードを含む、請求項 1 に記載の方法。

【請求項 7】

前記第三の A B I をターゲットとする前記関数の第三のコンパイルバージョンを生成することであって、前記第二の I S A の中で第三の命令セットを生成することを含む、生成することと、

20

前記関数の前記第三のコンパイルバージョンを前記バイナリファイルの中に送信することと、

をさらに含み、請求項 1 に記載の方法。

【請求項 8】

前記第二の A B I による前記レジスタのサブセットの使用により、前記第二の A B I をターゲットとしてコンパイルされる関数の少なくとも一部が、エントリサンクを用いずに前記第三の A B I から呼び出されることが可能となる、請求項 1 に記載の方法。

【請求項 9】

前記第一の A B I をターゲットとする前記関数の前記第一のコンパイルバージョンを生成することは、前記第一の I S A のソースコード定義を使用することを含み、前記第二の A B I をターゲットとする前記関数の前記第二のコンパイルバージョンを生成することは、前記第二の I S A のソースコード定義を使用することを含む、請求項 1 に記載の方法。

30

【発明の詳細な説明】

【背景技術】

【0001】

背景

[001] コンピューティングにおいて、各プロセッサ、又は中央処理ユニット (C P U) はある命令セットアーキテクチャ (I S A : instruction set architecture) を実装するが、I S A はこれらの命令をそのプロセッサ上でネイティブとして実行できるようにするためにマシンコード命令が従わなければならないシンタックスとルールを記述したものである。現在の I S A の例は、INTEL CORPORATION の設計による I A - 3 2 I S A 、ADVANCED MICRO DEVICES, INC. の設計による x 8 6 - 6 4 I S A 、ARM LIMITED の設計による A A r c h 6 4 I S A 、INTERNATIONAL BUSINESS MACHINES CORPORATION の設計による P O W E R I S A 、及びカリフォルニア州立大学バークレイ校の設計による R I S C - V I S A であるが、その他多くの I S A が存在する。それに加えて、コンピューティングにおいて、アプリケーションバイナリインタフェース (A B I : application binary interface) は同じ I S A をターゲットとするマシンコード命令を含む 2 つのバイナリプログラムコンポーネント間のインタフェースであ

40

50

る。ABIは多くの場合、オペレーティングシステム（OS）によって定義され、サードパーティソフトウェアコンポーネントが、同じABIをターゲットとする他のライブラリ／コンポーネントをターゲットとし、したがってそれとインタフェースできるようにする。例えば、MICROSOFT WINDOWSは、ソフトウェアがx86-64 ISAを実装するプロセッサを含むWINDOWSデバイスをターゲットとすることができるようにする「Windows-X64」ABIを提供し、また、ソフトウェアがAArch64 ISAを実装するプロセッサを含むWINDOWSデバイスをターゲットとすることができるようにする“ARM64”ABIを提供する。

【0002】

【002】 一般に、ソフトウェアコンポーネントは、そのコンポーネントがコンパイルされるABIを（例えば、OSを介して）提供し、また、そのコンポーネントを定義するマシンコード命令とマッチするISAを有するプロセッサを含むデバイスにとって「ネイティブ」である。それゆえ、あるコンピューティングデバイスにより提供されたABIとマッチしないソフトウェアコンポーネントは全て、そのデバイスにとって「フォーリン（foreign）」と考えることができる。例えば、x86-64のためのLinux ABIをターゲットとするアプリケーションは、x86-64 ISAをターゲットとするが、Windows-X64 ABIしか提供しないプロセッサを有するコンピューティングデバイス上ではネイティブとして実行できない。同様に、Windows-X64 ABIをターゲットとするアプリケーションは、x64-64 ISAをターゲットとするが、x86-64のためのLinux ABIしか提供しないプロセッサを有するコンピューティングデバイス上ではネイティブとして実行できない。

【0003】

【003】 それに加えて、デバイスのプロセッサのISAとマッチしないソフトウェアコンポーネントは全て、そのデバイスにとって「フォーリン」であると考えることができる。例えば、x86-64 ISAをターゲットとするアプリケーションは（ターゲットとされているABIにかかわらず）、AArch64プロセッサを有する何れのコンピューティングデバイス上でもネイティブとして実行できない。同様に、AArch64 ISAをターゲットとするアプリケーションは（ターゲットとされているABIにかかわらず）、x86-64プロセッサを有する何れのコンピューティングデバイス上でもネイティブとして実行できない。

【0004】

【004】 WINDOWS等の幾つかの環境では、かなりのパフォーマンス上のペナルティを伴うことが多いものの、フォーリンソフトウェアをソフトウェアベースのISAエミュレーションを介して実行することが可能である。それゆえ、例えば、ARM64 ABIをターゲットとするアプリケーションを、x86-84プロセッサを有するデバイス上でAArch64エミュレータを介してエミュレートすることが可能であり得、及び／又はWindows-X64 ABIをターゲットとするアプリケーションを、AArch64プロセッサを有するデバイス上でx86-64エミュレータを介してエミュレートすることが可能であり得る。

【0005】

【005】 あるソフトウェアアプリケーションが複数のABI上で利用可能であっても、このアプリケーションの分散と使用が問題となり得る。例えば、平均的なコンピュータのユーザは、自分のコンピューティングデバイスによってサポートされるISA及び／又はABIを知らず、したがって、手持ちのコンピュータ上にプログラムのどのバージョンをインストールすべきかを、情報をもとに決定を下すことができず、混乱やフラストレーションを招く。それに加えて、異なるABI／ISAをターゲットとするために同じアプリケーションの複数のバージョンをディストリビュートし、サポートすることは、開発者にとって面倒である。これらの問題に対処しようとする1つの単純な解決策は、「ネイティブ」及び「フォーリン」コードストリームの両方を格納した「ファット」バイナリとして知られるものの生成及びOSサポートに対応することである。それゆえ、例えば、ソフト

10

20

30

40

50

ウェアベンドはあるアプリケーションのために1つのファットバイナリを生成し、ディストリビュートし得るが、これにはバイナリがAArch64ベースのデバイスにロードされたときに使用されるAArch64コードストリームと、バイナリがx86-64ベースのデバイスにロードされたときに使用されるx86-64コードストリームが含まれる。ファットバイナリによって異なるプロセッサISAの複雑さはエンドユーザからほとんど見えなくなるものの、これはあるバイナリに含まれるコードの量を基本的に二倍にする（すなわち、コードサイズは100%増大する）ことによって達成され、その半分は実際にそのデバイスにより使用されない。限定的な量の保存スペースしか持たないことの多いポータブル機器が一般的である時代では、ファットバイナリは貴重な保存リソースを素早く使い果たし、ソフトウェアの障害やユーザの不満足につながる。

10

【0006】

【006】 今日入手可能なデバイスの幅広く多様なエコシステムを考慮すると、ユーザはそのコンピューティングデバイス上でフォーリンソフトウェアを実行できることが多くの場合に望まれる。そのため、コンピュータメカ及び／又はOSベンドは、フォーリンソフトウェアの実行を比較的シームレスな体験とするように動機付けされ得る。幾つかの解決策により、フォーリンソフトウェアはエミュレーションを介して実行できるようになり、その際、フォーリンソフトウェアはネイティブOSライブラリとインタフェースできるようになった。しかしながら、これらの解決策は、これらのアプリケーションが依然としてネイティブとしてではなくエミュレーションにより実行されるという点で、限定的である。

20

【発明の概要】

【課題を解決するための手段】

【0007】

概要

【007】 本明細書に記載の少なくとも幾つかの実施形態によれば、ネイティブバイナリ（例えば、アプリケーション、モジュール、ライブラリ等）の生成及び／又は実行が容易になり、これはエミュレートされたフォーリンコードと密接に相互運用するように構成され、そのフォーリンコードに高レベルの相互運用性及び互換性を提供する「エミュレーション互換」（EC:emulation compatible）コードを含む。例えば、本明細書の実施形態により、アプリケーション開発者はアプリケーションのターゲットをオペレーティングシステム（OS）が提供するネイティブのECアプリケーションバイナリインタフェース（ABI）として、コンピューティングデバイスでネイティブとして実行するアプリケーションを製作し、分散させることができる。

30

【0008】

【008】 ある実施形態において、EC ABIは、フォーリンABIの多くの挙動を示し、それによってEC ABIをターゲットとするネイティブコードはフォーリンコードを（エミュレータを介して）呼び出し、それと相互運用することが可能となる。それゆえ、EC ABIにより、これらのネイティブとして実行されるアプリケーションは、エミュレートされたフォーリンコード、例えばそのアプリケーションの実行コンテキストの中にロードされるレガシライブラリ及びプラグイン等と効率的且つシームレス（例えば、エンドユーザの視点から見て）に相互運用できる。

40

【0009】

【009】 実施形態において、EC ABIはエミュレートされたフォーリンコード内の例外の巻き戻し及び／又はスレッド一時停止をサポートするネイティブコードのためのコンテキスト及び呼び出し規約（CCs）を提供する。例えば、実施形態において、EC ABIは、フォーリンABIと互換性のあるコンテキストデータ及びCCsを使用し、他方で既存の（incumbent）ネイティブABIにより想定されるコンテキストデータ及びCCsとの互換性は保たれる。

【0010】

【010】 実施形態において、EC ABIはフォーリンABIと互換性があるものの、

50

これは既存のネイティブ A B I との十分な互換性を保ち、E C A B I 向けにコンパイルされたコードの多くは既存のネイティブ A B I に向けてコンパイルされたコードと同じである。これによって、既存のネイティブ A B I (バイナリが既存のネイティブ A B I を認識するが、E C A B I は認識しないレガシシステムにより実行されることを可能にする) と E C A B I (E C A B I を認識する教化システム上でエミュレートされたフォーリンコードとの相互運用性の改善を可能にする) の両方をサポートするハイブリッドバイナリの中でそのコードを「折り畳み」やすくなる。

【0011】

【011】 特に、本明細書の実施形態において、おそらくは、E C A B I 向けにコンパイルされる関数には3つのエントリポイントがある。第一に、関数は、それが他の E C A B I 関数により呼び出された場合に、その「実際の」アドレスにより開始できる。第二に、関数は、関数がフォーリン A B I により呼び出された場合、「エントリサンク (entry thunk)」を介して入ることができる。第三に、関数は、関数がフォーリン A B I の下で実行されるコードによりランタイムにおいて変更されている場合、「早送りシーケンス」を介して入ることができる。関数のために採用される実際のエントリポイントに関係なく、E C A B I 向けにコンパイルされたコードとフォーリン A B I 向けにコンパイルされたコードが相互作用するとき、実施形態によって、関数は、A B I の境界をまたぐ際に一貫したメモリアドレス参照を有するようにすることができる。

【0012】

【012】 1つ又は複数の実施形態は、ソースコードを、ネイティブの第一の命令セットアーキテクチャ (I S A) をターゲットとする一方で、フォーリンの第二の I S A にコンパイルされたコードと互換性のあるバイナリファイルにコンパイルする方法、システム、及びコンピュータプログラム製品に関する。これらの実施形態において、コンピュータシステムは、(i) 第一の I S A に対応する第一の A B I、(i i) 第一の I S A に対応する第二の A B I、及び (i i i) 第二の I S A に対応する第三の A B I を識別する。第二の A B I は、(i) 第三の A B I により使用される第二のコンテキストデータフォーマットの第二のサイズを超えない第一のサイズを有する第一のコンテキストデータフォーマットを定義し、(i i) 第一の I S A の複数のレジスタのうちの、第一の A B I により使用されるレジスタのサブセットの使用を定義し、レジスタのサブセットは第三の A B I により使用される第二の I S A のレジスタセットにマッピングされ、第二の A B I によるレジスタのサブセットの使用の結果、第一の A B I と第二の A B I の各々を使ってコンパイルされる際に関数の少なくとも一部が折り畳み可能となる。ソースコードで定義される関数の場合、コンピュータシステムは第一の A B I をターゲットとする関数の第一のコンパイルバージョンを生成し、これには第一の I S A の第一の命令セットを生成することが含まれ、また、第二の A B I をターゲットとする関数の第二のコンパイルバージョンを生成し、これには第一の I S A の第二の命令セットを生成することが含まれる。コンピュータシステムは、関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンがバイナリファイルの中で折り畳み可能であるか否かを、少なくとも第一の命令セットと第二の命令セットがマッチするか否かを特定することに基づいて特定する。少なくとも関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンがバイナリファイルの中で折り畳み可能であるか否かを特定することに基づいて、コンピュータシステムは、(i) 関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンの両方をバイナリファイルの中に送るか (すなわち、関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンが、バイナリファイル内で折り畳めないと特定された場合)、又は (i i) 関数の第一のコンパイルバージョン又は関数の第二のコンパイルバージョンの一方のみをバイナリファイルの中に送る (すなわち、関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンがバイナリファイルの中で折り畳めると特定された場合) の何れかを行う。

【0013】

【013】 1つ又は複数の実施形態はまた、ハイブリッドバイナリイメージを生成する方

法、システム、及びコンピュータプログラム製品にも関し、このハイブリッドバイナリイメージは、ネイティブ A B I 及び互換性 A B I のどちらに基づいても実行可能である。これらの実施形態において、少なくともネイティブ A B I に対応する第一のマシントイプを識別することに基づいて、コンピュータシステムは第一のマシントイプをハイブリッドバイナリイメージのマシントイプフィールドに送信する。マシントイプフィールドは、ハイブリッドバイナリイメージがネイティブ A B I に基づいて実行されるネイティブプロセスによりロードされたときに利用されるように構成される。少なくとも折り畳み不能な第一の関数を識別することに基づいて、コンピュータシステムはまた、ハイブリッドバイナリイメージの中に、(i) ネイティブ A B I に基づいて実行可能な第一の関数の第一のコンパイルバージョンと、(i i) 互換性 A B I に基づいて実行可能な第一の関数の第二のコンパイルバージョンの両方を送信する。少なくとも折り畳み可能な第二の関数を識別することに基づいて、コンピュータシステムはまた、ハイブリッドバイナリイメージの中に、ネイティブ A B I と互換性 A B I の両方に基づいて実行可能な第二の関数のコンパイルバージョンを送信する。第二の関数のコンパイルバージョンは、ハイブリッドバイナリイメージがネイティブプロセスによりロードされたときに第一の関数の第一のコンパイルバージョンを呼び出すように構成される。コンピュータシステムはまた、ハイブリッドバイナリイメージの中に、ハイブリッドバイナリイメージが互換性 A B I に基づいて実行される互換性プロセスによりロードされたときに利用されるように構成されたフィックスアップテーブルを送信する。フィックスアップテーブルは、ハイブリッドバイナリイメージからロードされたメモリへの複数の変換を定義し、これにはマシントイプフィールドを、互換性 A B I に対応する第二のマシントイプを含むように調整する第一の変換と、第二の関数のコンパイルバージョンを、第一の関数の第一のコンパイルバージョンの代わりに第一の関数の第二のコンパイルバージョンを呼び出すように構成する第二の変換が含まれる。

10

20

30

40

【 0 0 1 4 】

【 0 1 4 】 1 つ又は複数の実施形態はまた、互換性 A B I に基づいて実行されるプロセスによりハイブリッドバイナリイメージを使用する方法、システム、及びコンピュータプログラム製品にも関し、ハイブリッドバイナリイメージは、ネイティブ A B I と互換性 A B I の両方に基づいて実行可能である。これらの実施形態において、ハイブリッドバイナリイメージのロード中、コンピュータシステムは、ハイブリッドバイナリイメージのマシントイプフィールド内に保存された第一のマシントイプが、プロセスがそれに基づいて実行中の互換性 A B I に対応する第二のマシントイプとミスマッチすることを特定する。第一のマシントイプが第二のマシントイプとミスマッチすると特定したことに基づいて、コンピュータシステムは、バイナリイメージ内で、ハイブリッドバイナリイメージからロードされたメモリへの複数の変換を定義するフィックスアップテーブルを配置する。コンピュータシステムはすると、ハイブリッドバイナリイメージからロードされたメモリへの複数の変換のうちの少なくとも一部を適用し、これには少なくとも、マシントイプフィールドを、互換性 A B I に対応する第二のマシントイプを含むように調整する第一の変換と、互換性 A B I に基づいて実行可能な、折り畳まれていない関数の第二のコンパイルバージョンを呼び出す代わりに、ネイティブ A B I に基づいて実行可能な、折り畳まれていない関数の第一のコンパイルバージョンを呼び出すコールサイトを変更する第二の変換を適用することが含まれる。

【 0 0 1 5 】

【 0 1 5 】 1 つ又は複数の実施形態はまた、(i) ネイティブ I S A に対応し、第一の C C を有するネイティブ A B I をターゲットとするネイティブコードと、(i i) フォーリン I S A に対応し、第二の C C を有するフォーリン A B I をターゲットとするフォーリンコードの両方の実行をサポートするプロセスの中でコールを処理する際に共通の参照メモリアドレスを使用する方法、システム、及びコンピュータプログラム製品にも関する。これらの実施形態において、少なくとも呼び出し先関数のための参照メモリアドレスをターゲットとするコールを識別することに基づいて、コンピュータシステムは、呼び出し先関数がネイティブ A B I に対応するかフォーリン A B I に対応するかを特定する。呼び出し

50

先関数は、(i) 参照メモリアドレスがネイティブコードを記憶する第一のメモリレンジの中に含まれることを示すルックアップ構造、又は(ii) 参照メモリアドレスで識別されている早送りシーケンスの少なくとも一方に基づいてネイティブ A B I に対応すると特定される。呼び出し先関数は、参照メモリアドレスがネイティブコードを記憶しない第二のメモリレンジ内に含まれることを示すルックアップ構造に基づいて、フォーリン A B I に対応すると特定される。少なくともこの特定に基づいて、コンピュータシステムは、呼び出し先関数の実行を開始し、これは、(A) 呼び出し元関数がフォーリン A B I に対応し、呼び出し先関数がフォーリン A B I に対応すると特定されたときに、エミュレータ内で参照メモリアドレスを使って呼び出し先関数を直接呼び出すこと、(B) 呼び出し元関数がフォーリン A B I に対応し、呼び出し先関数がネイティブ A B I に対応すると特定されたときに、(i) 第二の C C を第一の C C に適応させ、その後、(ii) 参照メモリアドレスを使って呼び出し先関数を直接呼び出すエントリサンクを呼び出すこと、(C) 呼び出し元関数がネイティブ A B I に対応し、呼び出し先関数がフォーリン A B I に対応すると特定されたとき、(i) ネイティブ A B I の第一の C C をフォーリン A B I の第二の C C に適合させ、その後(ii) エミュレータに、参照メモリアドレスを使って呼び出し先関数を直接呼び出させるエグジットサンクを呼び出すこと、又は(D) 呼び出し元関数がネイティブ A B I に対応し、呼び出し先関数がネイティブ A B I に対応すると特定されたとき、参照メモリアドレスを使って呼び出し先関数を直接呼び出すこと、のうちの1つを実行することに基づく。

10

【0016】

20

【016】この概要は、以下の詳細な説明の中でさらに説明されるコンセプトの選択を簡略化した形態で紹介するために提供されている。この概要は、特許請求される主題の重要な特徴又は不可欠の特徴を明示することも、特許請求される主題の範囲を特定するのを支援するために使用されることも意図されない。

【0017】

図面の簡単な説明

【017】本発明の上述の、及びその他の利点と特徴を取得できる方法を説明するために、上で簡単に記した本発明を、添付の図面に描かれたその特定の実施形態に関してより具体的に説明する。これらの図面は本発明の典型的な実施形態を図示しているにすぎず、したがって、その範囲を限定するものとはみなされないことを理解した上で、本発明を下記のような添付の図面を使用して、さらに具体的に詳しく記述し、説明する。

30

【図面の簡単な説明】

【0018】

【図1】【018】エミュレートされたフォーリンコードと密接に相互運用するように構成され、フォーリンコードに高レベルの相互運用性と互換性を提供する、エミュレーション互換(E C)コードを含むネイティブバイナリの生成及び／又は実行を容易にする例示的なコンピュータシステムを示す。

【図2】【019】フォーリン A B I をミラーする一方で、既存のネイティブ A B I と互換性を保持するコンテキスト及び呼び出し規約(C C s)を定義する E C ネイティブアプリケーションバイナリインタフェース(A B I)の例を示す。

40

【図3A】【020】デュアルアーキテクチャハイブリッドバイナリの例を示す。

【図3B】【021】インポート及び／又はエクスポートテーブルのウィンドウ化されたビューの例を示す。

【図4A】【022】既存 A B I をターゲットとしてコンパイルされた「既存」ネイティブ関数の呼び出し挙動のほか、E C A B I をターゲットとしてコンパイルされた「E C」ネイティブ関数の呼び出し挙動を示す。

【図4B】【023】同じコードは既存 A B I と E C A B I の両方にソースコード関数をコンパイルしたことによるために折り畳まれている「E C」ネイティブ関数に関する呼び出し挙動を示す。

【図5】【024】ネイティブの第一の命令セットアーキテクチャ(I S A)をターゲットと

50

し、他方でフォーリンの第二の I S A にコンパイルされたコードと動作可能であるバイナリファイルにソースコードをコンパイルする例示的な方法のフローチャートを示す。

【図 6】[025]ネイティブ A B I と互換性 A B I の両方に基づいて実行可能なハイブリッドバイナリイメージを生成する例示的な方法のフローチャートを示す。

【図 7】[026]互換性 A B I に基づいて実行可能なプロセスによりハイブリッドバイナリイメージを実行する例示的な方法のフローチャートを示し、ハイブリッドバイナリイメージはネイティブ A B I と互換性 A B I の両方に基づいて実行可能である。

【図 8】[027] (i) ネイティブ I S A に対応し、第一の C C を有するネイティブアプリケーションバイナリインタフェース (A B I) をターゲットとするネイティブコード (i i) と、フォーリン I S A に対応し、第二の C C を有するフォーリン A B I をターゲットとするフォーリンコードの両方の実行をサポートするプロセスにおいてコールを処理するときに共通の参照メモリアドレスを使用する例示的な方法のフローチャートを示す。

【発明を実施するための形態】

【0019】

詳細な説明

[028] 図 1 は、エミュレートされたフォーリンコードと密接に相互運用するように構成され、フォーリンコードに高レベルの相互運用性と互換性を提供する「エミュレーション互換」(E C) コードを含むネイティブバイナリ (例えば、アプリケーション、モジュール、ライブラリ、等) の生成及び／又は実行を容易にする例示的なコンピュータシステムを示す。コンピュータシステム 101 はさらに、E C コードを含むものの、これらのバイナリに含まれる E C 機能を認識しない「レガシ」コンピューティングデバイスとも依然として互換可能なネイティブバイナリの生成及び／又は実行も容易にする。

【0020】

[029] 一般に、コンピュータシステム 101 は E C ネイティブアプリケーションバイナリインタフェース (A B I) に基づいて動作し、これはフォーリン A B I の多くの挙動を示し、それによって E C ネイティブ A B I をターゲットとするネイティブコードが (エミュレータを介して) フォーリンコード、例えばレガシプラグイン、ライブラリ等と呼ば出し、それと相互運用することを可能とする。実施形態において、E C ネイティブ A B I は、エミュレートされたフォーリンコード内で例外の巻き戻し及び／又はスレッド一時停止をサポートするネイティブコードのためのコンテキスト及び呼び出し規約 (C C s) を提供する。例えば、実施形態において、E C ネイティブ A B I は、フォーリン A B I と互換性のあるコンテキストデータ及び C C s を使用し、他方で既存のネイティブ A B I により想定されるコンテキストデータ及び C C s との互換性を保持する。そのようにする中で、E C ネイティブ A B I 向けにコンパイルされたコードの多くが既存のネイティブ A B I 向けにコンパイルされたコードと同じであり、それによって既存のネイティブ A B I (これによって、バイナリは既存のネイティブ A B I を認識するが、E C ネイティブ A B I は認識しないレガシシステムによって実行可能となる) 及び E C ネイティブ A B I (これによって、E C ネイティブ A B I を認識する教化システム上でエミュレートされたフォーリンコードとのより高い相互運用性が実現される) の両方をサポートするハイブリッドバイナリの中でそのコードを「折り畳む」ことが可能となる。それに加えて、E C ネイティブ A B I 向けにコンパイルされたコードとフォーリン A B I 向けにコンパイルされたコードが相互作用するとき、実施形態では、A B I の境界をまたぐ時に関数が一貫したメモリアドレスリファレンスを有することができ、それによって互換性が向上する。

【0021】

[030] 実施形態において、コンピュータシステム 101 は、特定用途又は汎用コンピュータハードウェア、例えば通信バス 105 を使って通信可能に連結されるプロセッサ 102 (又は複数のプロセッサ)、持続性ストレージ 103、及びシステムメモリ 104 を含むか、これを利用する。コンピュータシステム 101 において、プロセッサ 102 は何れの入手可能なプロセッサ命令セットアーキテクチャ (I S A) を実装してもよく、これは例えば x 8 6 - 6 4、A A r c h 6 4、P O W E R、R I S C - V 等であり、それを本

10

20

30

40

50

明細書においてはプロセッサ102の「ネイティブ」ISAと呼ぶ。プロセッサ102のネイティブISAとマッチしないISAは全て、本明細書中では「フォーリン」ISAと呼ぶ。一般に、プロセッサ102のISAはプロセッサ102の多くのハードウェア面、例えばプロセッサ102により実行可能なマシンコード命令のシンタックス、プロセッサ102がこれらのマシンコード命令による使用のために露出させるレジスタセット、プロセッサにより使用されるメモリモデル、その他を定義する。それゆえ、例えば、プロセッサ102がAArch64 ISAを実装する場合、それはx86-64 ISAを実装した場合とは異なるマシンコード命令セット（例えば、利用可能な命令、命令フォーマット等のうちの1つ又は複数を含む）を実行し、異なるレジスタセット（例えば、レジスタ名、レジスタのサイズ、レジスタの数等のうちの1つ又は複数を含む）を露出させることになる。

10

【0022】

【031】 持続性ストレージ103は実行可能なソフトウェアコンポーネントを表すコンピュータ実行可能命令及び／又はデータ構造を記憶し、それに対応して、このソフトウェアがプロセッサ102上で実行されている間に、これらのコンピュータ実行可能命令及び／又はデータ構造の1つ又は複数の部分はシステムメモリ104にロードされる。例えば、持続性ストレージ103は、オペレーティングシステム106、開発環境107、ハイブリッドバイナリ108、フォーリンバイナリ109、及びソースコード110に対応するコンピュータ実行可能命令及び／又はデータ構造を記憶できるように示されている。

【0023】

【032】 システムメモリ104は、様々なデータを記憶でき、これは持続性ストレージ103からロードし、プロセッサ102により保存し、及び／又はネットワークデバイス（図示せず）等、他の何れかの場所から入手できる。一般に、コンピュータシステム101は、持続性ストレージ103上に記憶された1つ又は複数のバイナリイメージ（例えば、ハイブリッドバイナリ108、フォーリンバイナリ109等）により定義されるメモリページをシステムメモリ104内にロードして、これらのメモリページ（システムメモリ104にロードされた状態）でプロセッサ102を使って動作することによって動作する。これには、これらのメモリページ内に記憶されたマシンコード命令を実行して、これらのメモリページ内に記憶されたデータに基づいて動作することが含まれる。

20

【0024】

【033】 オペレーティングシステム106（以下、OS 106と呼ぶ）はライブラリ106aを含み、これは例えば、プロセッサ102のネイティブISAを使用する1つ又は複数のABIをターゲットとするアプリケーションバイナリの実行をサポートするライブラリである。開発環境107から出ている矢印により示されるように、幾つかの実施形態において、ライブラリ106aは既存のネイティブABI 107a（以下、既存のABI 107aと呼ぶ）及びECネイティブABI 107b（以下、EC ABI 107bと呼ぶ）のサポートを含み、これらについては後でより詳しく説明する。OS 106はまた、バイナリイメージをシステムメモリ104にロードするためのローダ106cも含み、これは既存のABI 107aとEC ABI 107bの両方を認識する。それゆえ、ライブラリ106aとローダ106cを有することに基づいて、OS 106は、既存のABI 107a又はEC ABI 107bの一方又は両方をターゲットとするコードを含むネイティブバイナリ（例えば、アプリケーション、モジュール、ライブラリ等）の実行をサポートする。

30

40

【0025】

【034】 実施形態において、OS 106はまた、エミュレータ106bも含み、これは開発環境107から出ている矢印により示されるように、エミュレーションを介してフォーリンABI 107c（例えば、フォーリンISAに基づく）をターゲットとするバイナリの実行をサポートする。実施形態において、エミュレータ106bは、EC ABI 107bを提供するライブラリを含むライブラリ106aと相互運用するように構成される。それゆえ、エミュレータ106bを有することに基づいて、OS 106は、フ

50

フォーリンABI 107cをターゲットとするコードを含むフォーリンバイナリ（例えば、アプリケーション、モジュール、ライブラリ等）の実行をサポートする。

【0026】

【035】 本明細書で使用されるかぎり、「エミュレーション」という用語は翻訳及び／又は解釈を含むことができる。例えば、AOT (ahead-of-time) 翻訳を用いると、フォーリンISA命令はネイティブISA命令に翻訳されて、ストレージ（例えば、持続性ストレージ103）に持続的に保存され、これらの翻訳された命令はその後、ランタイムに必要な時に呼び出される。一般に、AOT翻訳はプログラムの実行が要求される前、例えばアプリケーションがインストールされる際に行われる。他の例では、JIT (just-in-time) 翻訳の場合、フォーリンISA命令は、フォーリンISA命令が要求されたとき（例えば、ユーザがプログラムを実行するとき）にネイティブISA命令に翻訳される。JIT翻訳では、翻訳されたネイティブコードは、それがフォーリンコードから翻訳されると即時に実行される。実施形態において、JIT翻訳は断片的に行われるが、これは実行のためにより多くのフォーリンコードが「発見される」からである。実施形態において、フォーリンISAコードの同じブロックのJIT翻訳は一度のみ行われ、そのため、コードが数回実行されても翻訳コストは1回のみ発生する。また別の例において、解釈では、フォーリンISA命令は実行が必要であるときに読み取られ、インタプリタにより同等の機能が実行されるが、それに対応するネイティブISAコードは生成されない。フォーリンコードは解釈では翻訳されないため、同じフォーリンISA関数が2回実行された場合、解釈コストは2回発生する。

10

20

【0027】

【036】 開発環境107は、少なくともEC ABI 107bをターゲットとするが、多くの場合、既存のABI 107aもターゲットとするバイナリ（すなわち、デュアルアーキテクチャ又は「ハイブリッド」バイナリイメージとして）の作成をサポートする。実施形態において、開発環境107は、さらにフォーリンABI 107cをターゲットとする（そして、したがってフォーリンABI 107cがネイティブであるコンピュータシステム上でも直接実行できる）バイナリの作成もサポートする。実施形態において、既存のABI 107aは、プロセッサ102のネイティブISAをターゲットとする「レガシ」ABIであり、フォーリンABI 107cは何れかのフォーリンISAをターゲットとするABIである。EC ABI 107bは、それに対して、プロセッサ102のネイティブABIであるが、フォーリンABI 107cをミラーする、又は少なくともそれと幾つかの属性を共有するコンテキストデータ及びCCsを定義する。それゆえ、実施形態において、既存のABI 107aは完全に「ネイティブ」のデータ命令及び挙動を定義するとみなすことができ、EC ABI 107bは、同じくネイティブではあるものの、フォーリンABI 107cと少なくとも幾つかのデータ構造属性及び挙動を共有するとみなすことができる。一例において、既存のABI 107aは、AArch64 ISAをターゲットとするARM64 ABIであり、フォーリンABI 107cはx86-64 ISAをターゲットとするWindows-X64 ABIであり、EC ABI 107bはAArch64 ISAをターゲットとするが、Windows-X64 ABIを少なくとも部分的にミラーするコンテキストデータ及びCCsを含む。実施形態において、EC ABI 107bは、フォーリンABI 107cをミラーし、又はそこにマッピングされる十分なコンテキストデータとCCsを定義して、エミュレートされたフォーリンコード及びECネイティブコードがごく低レベルで相互作用できるようにするだけでなく、既存のABI 107aと共通のもの（例えば、利用可能なレジスタセット）を十分に有して、ソース関数の少なくとも一部を既存のABI 107aとEC ABI 107bの各々にコンパイルし、同じコンパイルされた関数本体が得られるようにする。コンパイルされた関数本体が同じであると、これらは「折り畳み可能」と考えることができ、それによって実際には一方の関数本体だけが結果として得られるバイナリに含められ、その1つの関数本体が既存のABI 107aをターゲットとするコードストリームとEC ABI 107bをターゲットとするコードストリームの両

30

40

50

方により（例えば、ポインタエイリアシングを介して）使用される。

【0028】

【037】 EC ABI 107bの実施形態をさらに説明するために、図2は、ECネイティブABIの例200を示しており、これはフォーリンABIをミラーするコンテキスト及びCCsを定義し、他方で既存のネイティブABIとの互換性も保持する。まず、図2は図1の既存のABI 107a、EC ABI 107b、及びフォーリンABI 107cの各々の表現を示す。図のように、既存のABI 107aは9つの利用可能なレジスタセット201aを定義し、これはレジスタN:A~N:Iを含み、これらは例200において、全てネイティブISAの中で利用可能なレジスタである。それに対して、フォーリンABI 107cは5つの利用可能なレジスタセット201cを定義し、これはレジスタF:V~F:Zを含み、これらは例200では、全てがフォーリンISAの中で利用可能なレジスタであると仮定される。類推として、AArch64 ISA（例えば、「ネイティブ」ISA）は31の汎用64ビットレジスタを定義し、x86-64 ISA（例えば、「フォーリン」ISA）は16の汎用64ビットレジスタを定義する。EC ABI 107bは、レジスタN:A、N:C、N:D、N:E、及びN:Gを含む5つの利用可能なレジスタセット201bを定義することによって既存のABI 107aとフォーリンABI 107cを相互に接続する。それゆえ、ネイティブISAは9つの利用可能なレジスタを有しているが、例200では、EC ABI 107bはこれらのレジスタのうち5つのみを使用することに制約され、これはフォーリンISAで利用可能なレジスタの数に対応する数である。当業者であればわかるように、EC ABI 107bをターゲットとするネイティブコードはネイティブISAにより定義される必ずしも全てのレジスタを使用するとはかぎらないかもしれないが、そのネイティブコードは依然としてネイティブISA上で実行可能である。それゆえ、EC ABI 107bは利用可能なレジスタ201bの使用を、既存のABI 107aと互換性があるように定義でき、それによってEC ABI 107bをターゲットとするネイティブコードはまた、既存のABI 107aのみをサポートするシステムでも実行可能である。

【0029】

【038】 実施形態において、利用可能なレジスタセット201bは、既存のABI 107a向けにコンパイルされたコードが最も一般的に使用するレジスタを使うために選択される。それゆえ、例200では利用可能なレジスタセット201bは利用可能なレジスタセット201aの一部を使用するものの、コンパイルされたコードは、利用可能なレジスタセット201aをターゲットとしていても、実際には利用可能なレジスタセット201bから選択されたレジスタのみを使用する場合が多いかもしれない。これらの状況では、既存のABI 107a及びEC ABI 107bの各々に向けてコンパイルされたコードは同じであり、結果として得られるバイナリの中で折り畳むことができる。特に、例200において、EC ABI 107bの可能なレジスタセット201bは既存のABI 107aの利用可能なレジスタセット201aの一部を含んでいるが、代替的な例は、利用可能なレジスタセット201bを、利用可能なレジスタセット201aの全てを使用するように定義し得る（例えば、フォーリンISAがネイティブISAとマッチする複数のレジスタを有する場合）。

【0030】

【039】 それに加えて、図2内の矢印は、EC ABI 107bの利用可能なレジスタ201bとフォーリンABI 107cの利用可能なレジスタ201cとの間にマッピングが定義されていることを示している。それゆえ、例えば、EC ABI 107bはレジスタN:Aを、フォーリンABI 107cによるレジスタF:Vの使用をミラーするような方法で使用する、EC ABI 107bはレジスタN:Cを、フォーリンABI 107cによるレジスタF:Wの使用をミラーするような方法で使用する、等々である。一例において、EC ABI 107bは利用可能なレジスタ201bの使用を、フォーリンABI 107cにより使用されるCCsをミラーするような方法で定義する。例えば、EC ABI 107bはフォーリンABI 107cの挙動を実現するために、

関数にパラメータを渡すために利用可能なレジスタの数を限定し得る（すなわち、既存の A B I 1 0 7 a と比較して）。類推として、X 6 4 A B I はパラメータを渡すのに 4 つのレジスタのみを使用するため、E C A B I 1 0 7 b は 4 つのレジスタだけを、パラメータを渡すために使用できるようにし得るが、既存の A R M 6 4 A B I はパラメータを渡すことを目的として追加のレジスタを使用するように定義される。それに加えて、E C A B I 1 0 7 b は、（既存の A B I 1 0 7 a の場合のように）その目的のためのレジスタを使用するのではなく、フォーリン A B I 1 0 7 c によるスタック使用をミラーし得る（例えば、戻りアドレスを保存するため、関数からの値を返すため、等）。実施形態において、レジスタ間のマッピングはレジスタ間の揮発性のマッピングを含む。それゆえ、例えば、既存の A B I 1 0 7 a では不揮発性と考えられるネイティブ I S A 内のレジスタは、それがフォーリン A B I 1 0 7 c により揮発性と考えられるレジスタにマッピングされた場合、E C A B I 1 0 7 b により揮発性と考えられ得る。

10

【0031】

【040】 図 2 はまた、既存の A B I 1 0 7 a はコンテキストデータ 2 0 2 a を定義し、E C A B I 1 0 7 b はコンテキストデータ 2 0 2 b を定義し、フォーリン A B I 1 0 7 c はコンテキストデータ 2 0 2 c を定義することを示している。実施形態において、コンテキストデータは、レジスタやフラック等、プロセッサのコンテキストのスナップショットを保存するように A B I により定義されるフォーマットである。このコンテキストデータは様々な目的に、例えばアプリケーションが自己の状態を観察するため（例えば、コピー防止スキームの一環として、ウイルス又はマルウェア検出の一環として、等）、例外後のスタックを巻き戻す例外巻き戻しルーチンのため、又はスレッド一時停止／再開を容易にするために有益である。コンテキストデータ 2 0 2 a、コンテキストデータ 2 0 2 b、及びコンテキストデータ 2 0 2 c の各々の視覚的なサイズにより説明されるように、実施形態において、E C A B I 1 0 7 b はフォーリン A B I 1 0 7 c により使用されるコンテキストデータ 2 0 2 c のサイズを超えないコンテキストデータ 2 0 2 b のフォーマット（例えば、利用可能なレジスタ 2 0 1 b の選択及び配置）を定義する。しかしながら、既存の A B I 1 0 7 a はこのサイズを超えるコンテキストデータ 2 0 2 a を定義し得る。コンテキストデータ 2 0 2 b をコンテキストデータ 2 0 2 c のサイズを超えないように定義することにより、E C A B I 1 0 7 b とフォーリン A B I 1 0 7 c との間の相互運用が可能となる。例えば、フォーリン A B I 1 0 7 c に基づいて実行されるアプリケーションがメモリバッファをコンテキストデータ 2 0 2 c の保存用に割り当てて、そのメモリバッファがコンテキストデータ 2 0 2 b のフォーマットを使用する E C A B I 1 0 7 b により満たされた場合（例えば、例外処理の一環として）、E C A B I 1 0 7 b により書かれたコンテキストデータ 2 0 2 b は割り当てられたメモリバッファを超過しない。

20

30

【0032】

【041】 幾つかの実施形態において、コンテキストデータ 2 0 2 b は、コンテキストデータ 2 0 2 a とコンテキストデータ 2 0 2 c の混合であるフォーマットを有するように定義される。例えば、コンテキストデータ 2 0 2 a のフォーマットは左下がりの斜線で視覚的に表され、コンテキストデータ 2 0 2 c のフォーマットは右下がりの斜線で視覚的に表されている。すると、コンテキストデータ 2 0 2 b のフォーマットは右下がり及び左下がりの斜線の両方で視覚的に示され、フォーマットの少なくとも部分的な混合が示される。例えば、コンテキストデータ 2 0 2 b はネイティブ I S A の配置を定義しているものの、それは、フォーリン I S A が（例えば、利用可能なレジスタ 2 0 1 b と利用可能なレジスタ 2 0 1 c との間に示されるマッピングに基づいて）想定する方法でこれらのレジスタを配置し得る。

40

【0033】

【042】 特に、E C A B I 1 0 7 b （利用可能なレジスタ 2 0 1 b、コンテキストデータ 2 0 2 b、又は利用可能なレジスタ 2 0 1 b と利用可能なレジスタ 2 0 1 c との間のマッピングのうちの 1 つ又は複数を含む）を定義する際、E C A B I 1 0 7 b を既

50

存のABI 107aにより密接に似たものとして定義することと、EC ABI 107bをフォーリンABI 107cにより密接に似たものとして定義することの間にトレードオフがあり得る。例えば、EC ABI 107bが既存のABI 107aにより密接に似ているほど、EC ABI 107bにコンパイルされたコードがネイティブABI 107aにコンパイルされたコードと共に「折り畳む」ことができる可能性が高くなるが、それによって、EC ABI 107bとフォーリンABI 107cとの間の移行時にABI翻訳（後述の、エントリ及び／又はイグジットサックを介する）を利用することが必要となる可能性もより高くなる。逆に、EC ABI 107bがフォーリンABI 107cにより密接に似ているほど、EC ABI 107bの中のネイティブコードとフォーリンABIの中のフォーリンコードがABI翻訳（サック）を使用せず

10

に相互作用できる可能性が高くなるが、それによってEC ABI 107bにコンパイルされたコードをネイティブABI 107aにコンパイルされたコードと「折り畳む」ことができなくなる可能性もより高くなる。

【0034】

[043] 図1に戻ると、幾つかの実施形態において、開発環境107は2つ以上のABIをターゲットとする「デュアルアーキテクチャ」ハイブリッドバイナリの作成をサポートする。例えば、様々な矢印により示されているように、図1は、コンパイラツールチェーン107dがソースコード110を実行して、既存のABI 107a（すなわち、既存のネイティブコード108a）とEC ABI 107b（すなわち、ECネイティブコード108b）の両方をターゲットとするコードを含むハイブリッドバイナリ108を生成することを示している。それゆえ、ネイティブコードしか含まなくても、ハイブリッドバイナリ108は既存のネイティブ挙動（なわち、既存のネイティブコード108a）とフォーリン挙動（すなわち、ECネイティブコード108b）の両方をターゲットとしているとみなすことができる。図示されていないが、幾つかの実施形態において、ハイブリッドバイナリはまた、フォーリンABI 107c等、1つ又は複数のその他のABIをターゲットとするコードも含む。

20

【0035】

[044] 図1において、既存のネイティブコード108aとECネイティブコード108bとの間に太線の矢印があり、これは、既存のネイティブコード108aとECネイティブコード108bが少なくとも部分的に相互に「折り畳まれる」（すなわち、1つの命令セットが既存のネイティブコード108aとECネイティブコード108bの両方により使用されるように）ことを示している。前述のように、EC ABI 107bが少なくとも幾つかのフォーリン挙動をターゲットとしていても、それは既存のABI 107aと十分な共通点を有し、その結果、ソース関数の少なくとも一部が既存のABI 107aとEC ABI 107bの各々へとコンパイルされて、同じコンパイルされた関数本体が得られる。実施形態において、関数を既存のABI 107aとEC ABI 107bの各々にコンパイルすることにより同じコンパイルされたコードが得られたとき、コンパイラツールチェーン107dはこの関数をハイブリッドバイナリ108の中で「折り畳み」、この関数の1つのコンパイルされたバージョンのみを既存のネイティブコード108aとECネイティブコード108bの両方のために送信する。例えば、コンパイラツールチェーン107dがその関数に関してコンパイルコードのネイティブ「バージョン」とコンパイルコードのECネイティブ「バージョン」の両方を生成しても、コンパイラツールチェーン107dは、これらの「バージョン」がマッチする（すなわち、同じネイティブコードを有する）場合に、これらのバージョンのうちの一方のみをハイブリッドバイナリ108に送信する。テスト中、本発明者らは、コードの折り畳みによって、デュアルABI（すなわち、既存のABI 107aとEC ABI 107b）をこのようにサポートすることによって、バイナリサイズは1つのABIをターゲットとしたバイナリと比較して10～30%しか増大しないことを認めた。これは、コードを折り畳む能力のない、従来のファットバイナリではバイナリサイズが100%近く増大することとは著しく対照的である。

30

40

50

【0036】

[045] 実施形態において、ソースコード110をEC ABI 107bをターゲットとしてコンパイルするとき、コンパイラツールチェーン107dは、コンパイラツールチェーン107dがあたかもフォーリンABI 107cのISAをターゲットとしているかのように（コンパイラツールチェーン107dはネイティブコードを生成するが）、ソースコード定義、例えばプリプロセッサディレクティブに追従する。これは、EC ABI 107bがフォーリンABI 107cの挙動を示すからである。例えば、以下の関数はターゲットABIに応じて異なるロジックパスをたどる：

```
int function1 (int x) {
# if defined (__ARM64__)
return x + 10;
# else
return x - 2;
# endif}
```

10

特に、プリプロセッサディレクティブは、ARM64 ISAをターゲットとするときには10の値を‘x’に加えるべきであり、ARM64 ISAをターゲットとしないときには2の値を‘x’から差し引くべきであることを定義する。実施形態において、この関数をARM64 ISAを使用する既存のABI 107aをターゲットとするようにコンパイルする際、コンパイラツールチェーン107dは‘x’に10の値を加える命令（すなわち、ARM64ロジックパス）を生成し、逆に、この関数をEC ABI 107bをターゲットとするようにコンパイルする際には、コンパイラツールチェーン107dは2の値を‘x’から差し引く命令（すなわち、非ARM64ロジックパス）を生成する。

20

【0037】

[046] 実施形態において、ハイブリッドバイナリ108はコンパイラツールチェーン107dにより、レガシローダ（すなわち、既存のABI 107aは認識するが、EC ABI 107bは認識しない）によってネイティブにパースされ、利用されるが、教化ローダ（すなわち、EC ABI 107bを認識する）によって使用可能であるように構成される。実施形態において、ハイブリッドバイナリ108は、既存のABI 107aに基づいて実行するために、レガシローダにより想定されるレイアウト／フォーマット、すなわち「デフォルト」を使用する。しかしながら、ハイブリッドバイナリ108はまた、ハイブリッドバイナリ108がEC ABI 107bによっても実行できるようにするフィックスアップテーブル108c等の追加の情報も含む。

30

【0038】

[047] 図3Aは、「デュアルアーキテクチャ」ハイブリッドバイナリの例300aを示す。より詳細には、図3Aは、図1のハイブリッドバイナリ108のより詳しい表現を図示し、相互に関係する2つのコードストリーム、すなわちコードストリーム301aとコードストリーム301bを示している。実施形態において、コードストリーム301aは既存のネイティブコード108aに対応し、コードストリーム103bはECネイティブコード108bに対応する。それゆえ、ハイブリッドバイナリ108は2つの異なるネイティブABI（すなわち、既存のABI 07aとEC ABI 107b）を、それらの異なる挙動（すなわち、完全ネイティブ挙動とフォーリン挙動）と共にターゲットとする。コードストリーム301aとコードストリーム301bとの間の矢印により示されているように、幾つかの関数間に折り畳みがある。特に、コードストリーム301aとコードストリーム301bの各々は、実線の枠として描かれた「折り畳まれていない」関数を含み、これらの関数の異なるコンパイルバージョンがコードストリーム301aとコードストリーム301bの各々の中に存在することを示している。しかしながら、コードストリーム301aとコードストリーム301bの各々はまた、破線を有し、矢印により結ばれる枠として描かれた「折り畳まれた」関数も含んでおり、これは、各関数の1つのバージョンのみがコードストリーム301aとコードストリーム301bの両方により使用されるために（例えば、エイリアシングを介する）存在することを示している。図3Aは

40

50

、ハイブリッドバイナリ 108 が、フォーリンコードストリーム等の追加のコードストリーム 301c を含み得ることを示している。しかしながら、より典型的には、実施形態において、ハイブリッドバイナリ 108 はフォーリンコードストリーム 302 を含むフォーリンバイナリ 109（例えば、レガシプラグイン／ライブラリに対応する）と相互作用する。

【0039】

【048】 前述のように、実施形態において、ハイブリッドバイナリ 108 は、既存の ABI 107a によりネイティブで実行されるように構成される。例えば、ハイブリッドバイナリ 108 の中のマシン識別子 303 のフィールドは、既存の ABI 107a により想定されるマシンタイプを識別し、エントリポイント 304 はコードストリーム 301a へのエントリポイントを明示し、インポート及び／又はエクスポートテーブル（入力／エクスポートテーブル 305）は、既存の ABI 107a に関係する関数（及びその場所）を露出させる「ネイティブ」ビューを提供する。それに加えて、折り畳まれていない関数を呼び出す「折り畳まれた」関数の何れも、コードストリーム 301b 中の「EC」バージョンではなく、コードストリーム 301a の関数の「既存」バージョンを呼び出すように構成される。そのため、レガシローダは、コードストリーム 301a を実行するために、他の何れの互換バイナリの場合とも同様に、ハイブリッドバイナリ 108 のみをロードすればよい。

【0040】

【049】 教化ローダ（例えば、ローダ 106c）によるローディングを容易にするために、図 1 及び 3A は、ハイブリッドバイナリ 108 がフィックスアップテーブル 108c を含むことを示している。実施形態において、フィックスアップテーブル 108c は、ハイブリッドバイナリ 108 がエミュレートされたプロセス 111（例えば、EC ネイティブコード 108b をプロセッサ 102 でネイティブとして実行するほか、フォーリンバイナリ 109 等のフォーリンコードをエミュレータ 106b を介して実行できる互換性プロセス）にロードされるときにローダ 106c がハイブリッドバイナリ 108 からロードされたメモリに適用すべき 1 つ又は複数のメモリ変換を明示する。

【0041】

【050】 実施形態において、ローダ 106c は、ハイブリッドバイナリ 108 内のマシン識別子 303 が EC ABI 107b にとって不適正であると認識したところで、フィックスアップテーブル 108c を配置する。すると、ローダ 106c はフィックスアップテーブル 108c 内で明示された 1 つ又は複数の変換を、システムメモリ 104 中のハイブリッドバイナリ 108 からロードされたメモリページを含む部分に適用して、コードストリーム 301a ではなくコードストリーム 301b を実行する。実施形態において、フィックスアップテーブル 108c の中で明示された各変換は、メモリ位置（例えば、相対アドレスによる、絶対アドレスによる、その他）を、そのメモリ位置で適用される変換（例えば、そのメモリ位置で 1 つ又は複数のバイトを置き換えて、そのメモリ位置で数値演算を適用する）と共に識別する。実施形態において、フィックスアップテーブル 108c は、マシン識別子 303 を、そこにハイブリッドバイナリ 108 がロードされたプロセス（例えば、エミュレートされたプロセス 111）の ISA とマッチするように調整するための 1 つ又は複数のフィックスアップを含む。実施形態において、これはフォーリン ABI 107c に対応するフォーリン ISA である。実施形態において、フィックスアップテーブル 108c は、エントリポイント 304 を、コードストリーム 301b へのエントリポイントを明示するように調整するための 1 つ又は複数のフィックスアップを含む。実施形態において、フィックスアップテーブル 108c は、折り畳まれた関数に、コードストリーム 301a の関数の既存バージョンではなく、コードストリーム 301b 内の折り畳まれていない関数の EC バージョンを呼び出させるための 1 つ又は複数のフィックスアップを含む。実施形態において、フィックスアップテーブル 108c は、インポート／エクスポートテーブル 305 に、EC ABI 107b に関係する関数（及びその位置）を露出させる「互換性」ビューを提供させるための 1 つ又は複数のフィックスアップ

を含む。

【0042】

[051] 幾つかの実施形態において、折り畳まれた関数に、コードストリーム301a内の関数の既存バージョンではなく、コードストリーム301b内の折り畳まれていない関数のECバージョンを呼び出させることは、折り畳まれた関数内のコール自体をパッチすることを含む。しかしながら、他の実施形態において、折り畳まれた関数は、ディスパッチテーブル306を介して折り畳まれていない関数を間接的に呼び出すように構成される。これらの実施形態において、フィックスアップテーブル108cはこのディスパッチテーブル306への1つ又は複数のフィックスアップを含み、これは、コードストリーム301a内の関数の既存バージョンへのアドレス又はオフセットをコードストリーム301b内の関数のECバージョンへのアドレス又はオフセットに置き換える。実施形態において、ディスパッチテーブル306の使用には、コードストリーム301aではなくコードストリーム301bを実行するために、メモリフィックスアップを適用する必要のあるメモリページの数进行限定することにより効率を提供できる。

【0043】

[052] 幾つかの実施形態において、インポート／エクスポートテーブル305に、ECABI107bに関係する関数（及びその位置）を露出させる「互換性」ビューを提供させることは、インポート／エクスポートテーブル305への参照をパッチすることにより、インポート／エクスポートテーブル305の異なる部分（ウィンドウ）を露出させることを含む。この概念を図解するために、図3Bは、インポート／エクスポートテーブルのウィンドウ化されたビューの例300bを示す。特に、図3Bは複数の関数とそれらに対応するメモリ位置（例えば、アドレス又はオフセット）をマッピングするインポート／エクスポートテーブル305を示す。例300bにおいて、インポート／エクスポートテーブル305は、コードストリーム301aのみに適用可能な（例えば、折り畳まれていない関数として）関数（すなわち、A、B、及びC）を含む第一の影付き部分／ゾーン、コードストリーム301aとコードストリーム301bの両方に適用可能な（例えば、折り畳まれた関数として）関数（すなわち、D及びE）を含む影付きのない部分／ゾーン、及びコードストリーム301bのみに適用可能な（例えば、折り畳まれていない関数として）関数（すなわち、A'、B'及びF）を含む第二の影付き部分／ゾーンを含む。括弧は、インポート／エクスポートテーブル305のネイティブビュー307aとインポート／エクスポートテーブル305の互換性ビュー307bを示す。実施形態において、ネイティブビューは、関数A、B、及びC（コードストリーム301aのみに適用可能）のほか、関数D及びE（両方のコードストリームに適用可能）を含む、インポート／エクスポートテーブル305の第一のエントリのベースリファレンス及びサイズ／カウント（例えば、5）として明示される。実施形態において、互換性ビューは、関数D及びE（両方のコードストリームに適用可能）のほか、関数A'、B'、及びF（コードストリーム301bのみに適用可能）を含む、インポート／エクスポートテーブル305の第四のエントリのベースリファレンス及びサイズ／カウント（例えば5）として明示される。それゆえ、実施形態において、インポート／エクスポートテーブル305のベースリファレンスをパッチしてインポート／エクスポートテーブル305の異なるウィンドウを露出させることは、ベースリファレンス又はサイズ／カウントのうちの一方又は両方をパッチすることを含む。

【0044】

[053] 前述のように、ECABI107bは、例えばECABI107bに基づいてネイティブとして実行されるアプリケーションのコンテキストの中のレガシプラグイン／ライブラリを（エミュレーションを介して）実行するための、フォーリンABI107cとの相互運用性を促進する。これは、ECネイティブコード108b内の関数に、フォーリンコード（フォーリンバイナリ109の中のコードストリーム302）内の関数を呼び出させることができるようにすること、及びフォーリンコード内の関数に、ECネイティブコード108b内の関数を呼び出させることができるようにすることを含む。

フォーリンコードはエミュレータ106bに基づいて実行されるため、これらのクロスABI関数コールの各々について、エミュレータ106bに入る、又はそこから出る移行がある。実施形態において、ハイブリッドバイナリ108は、フォーリンABI 107cからEC ABI 107bに移行するため（すなわち、フォーリンバイナリ109内のコードがECネイティブコード108b内の関数を呼び出すとき）のエントリサンクと、EC ABI 107bからフォーリンABI 107cに移行するため（すなわち、ECネイティブコード108b内のコードがフォーリンバイナリ109内の関数を呼び出すとき）のエグジットサンクを含み得る。幾つかの実施形態において、フォーリンコードにより呼び出すことのできるECネイティブコード108b内の各関数は、1つのエントリサンク（大きさはゼロであり得る）を含み、異なるフォーリン関数への各コールのためのエグジットサンクを含む。実施形態において、サンクはEC ABI 107bとフォーリンABI 107cとの違いに、例えばEC ABI 107bのCCをフォーリンABI 107cのCCに（又はその逆に）適応させることによって適応する。

10

20

30

40

50

【0045】

【054】 実施形態において、各エントリサンクにより、フォーリン関数によりECネイティブ関数に渡されるパラメータがECネイティブ関数により実行されるのに適切な位置にあることが確実となる。これは、例えば、1つのレジスタから他のレジスタへと値を移動させること、スタック位置からレジスタへと値を移動させること、レジスタからスタック位置へと値を移動させること等を含み得る。幾つかの実施形態において、EC ABI 107bは、フォーリン関数によりECネイティブ関数に渡されるパラメータが、ECネイティブ関数により実行されるのに適切な位置にすでにあるような方法で定義され得る。これらの状況では、エントリサンクはサイズがゼロであり得る（それゆえ、何もしないか、省かれる）。幾つかの実施例において、これは特に、ECネイティブ関数に渡されるパラメータの数が閾値より低い場合に当てはまる。例えば、本発明者らは、AArch64ネイティブISAとx86_64フォーリンISAを用いた場合、EC ABI 107b（AArch64とx86_64レジスタとの間のマッピングを含む）を、4つ以下のパラメータがフォーリン関数からECネイティブ関数に渡された場合にサイズがゼロのエントリサンクを有することが可能となるような方法で定義できることに気付いており、これらのパラメータの全部が基本的な整数である。

【0046】

【055】 実施形態において、エグジットサンクは、ECネイティブ関数によるフォーリン関数の各コールについて利用され、それゆえ、1つのECネイティブ関数はゼロ以上のエグジットサンクを有し得る（ECネイティブ関数がいくつの異なるフォーリン関数を呼び出すかによる）。実施形態において、各エグジットサンクは、（i）ECネイティブ関数に戻るために関数アドレスを適切な位置（例えば、スタック又はレジスタ）に保存すること、（ii）ECネイティブ関数から呼び出されたフォーリン関数に渡されるパラメータがフォーリン関数による実行のために適当な位置にあることを確実にする、又は（iii）エミュレータ106bにおいてフォーリン関数の実行を開始することのうちの1つ又は複数を実行する。

【0047】

【056】 図4A及び4Bは、ネイティブ関数同士間、及びネイティブ関数とフォーリン関数との間のコールの例400a及び400bを示す。特に、図4Aは、既存のABI 107aをターゲットとしてコンパイルされた既存のネイティブ関数403の呼び出し挙動と、EC ABI 107bをターゲットとしてコンパイルされたECネイティブ関数404の呼び出し挙動を示す。実施形態において、既存のネイティブ関数403とECネイティブ関数404は、同じソースコード関数の異なるコンパイルバージョンであるが、結果として得られるコンパイルコード間のミスマッチにより、一緒に折り畳まれていない。実施形態において、このミスマッチは、既存のネイティブ関数403が既存のABI 107aに基づいて利用可能な（しかしながら、EC ABI 107bでは利用できない）1つ又は複数のレジスタを使用するのに対し、ECネイティブ関数403はフォーリ

ンABI 107cにマッピングされる、より少ないレジスタセットのみを使用する（図3Aに関して説明した）ことから生じ得る。或いは、このミスマッチは、条件付きのコードコンパイルにより、既存のABI 107a及びEC ABI 107bをターゲットとするときにコードロジックの分岐が余儀なくされることから生じ得る。例えば、前述のように、実施形態において、コンパイラツールチェーン107dは、同じ関数を既存のABI 107a及びEC ABI 107bの各々をターゲットとするようにコンパイルするときに、異なるプリプロセッサディレクティブ（及び、それゆえ異なるロジックパス）に従う。特に、コンパイラツールチェーン107dは、既存のABI 107aをターゲットとするときには「ネイティブISA」ロジックパスをとるが、EC ABI 107bをターゲットとするときには「フォーリンISA」ロジックパスをとる。

10

【0048】

【057】 図4Aにおいて、パス#1は、既存のネイティブ関数403がネイティブCC（図中、「ネイティブCC」）を使って呼び出されることを示している。さらに、パス#1は、既存のネイティブ関数403が「ネイティブCC」使ってコールできることを示している。パス#2は他方で、ECネイティブ関数404がフォーリンCC（図中、「フォーリンCC」）を使って、フォーリンCCとネイティブCCとの差に適応し、その後、ECネイティブ関数404を起動させるエントリサンク401を介して間接的に呼び出されることを示している。さらに、パス#2は、ECネイティブ関数404が、ネイティブCCとフォーリンCCとの間の差に適応し、その後、フォーリン関数を起動させるエグジットサンク402を介して間接的に呼び出されることを示す。パス#3は、フォーリン関数405（例えば、コードストリーム301c及び／又はコードストリーム302内）がフォーリンCCを使って（例えば、エグジットサンク402又はその他のフォーリン関数によって）呼び出されることを示す。

20

【0049】

【058】 図4Bは、ソースコード関数を既存のABI 107aとEC ABI 107bの両方にコンパイルしたことから同じコードが得られたことにより折り畳まれているECネイティブ関数406の呼び出し挙動を示す。図4Bにおいて、パス#1は、ECネイティブ関数406がネイティブCCを使って呼び出し可能であることを示し、パス#2は、ECネイティブ関数406はまた、エントリサンク401を介して、フォーリンCCを使って間接的に呼び出せることも示している。それに加えて、図4Bは、「ECネイティブ関数404」がネイティブCCを使って、エグジットサンク402を介して他のネイティブ関数又はフォーリン関数の何れにもコールできることを示している。パス#3は、フォーリン関数405がフォーリンCCを使って（例えば、エグジットサンク402又は他のフォーリン関数により）呼び出されることを示している。

30

【0050】

【059】 実施形態において、コンピュータシステム101は、関数コールサイトがネイティブ及びフォーリンコードを（すなわち、ABI 107bとフォーリンABI 107cとの間で）ブリッジングするときにも、呼び出し先関数の真のメモリアドレスを一貫して呼び出し／参照することができるようになる。特に、コールサイトで真のメモリアドレスを使用することは、ネイティブ及びフォーリンコードをブリッジングするときに問題となり、それは、ABI移行をブリッジングするためにエントリ及びエグジットサンクを使用することによる。これは、2つのネイティブ関数の中のコールサイトが実際には異なるアドレスを使って同じフォーリン関数を呼び出し得る、ネイティブ及びフォーリンコードのブリッジングを行うための先行技術の解決策とは対照的である。これは、これらの先行技術の解決策において、コールサイトは実際にはフォーリン関数の真のアドレスではなく、サンクのアドレスを呼び出しているからである。この挙動は、ネイティブコード内のプログラムロジックがアドレス比較に依存する場合（例えば、第一のネイティブ関数から得られたフォーリン関数への第一のポインタと第二のネイティブ関数から得られたフォーリン関数への第二のポインタを比較する）、ポインタがネイティブコードからフォーリンコードに渡される場合（フォーリンコードにとっては無効／使用不能であるとき）、又は

40

50

ポインタがフォーリンコードからネイティブコードに渡される場合（それがネイティブコードにとって無効／使用不能であるとき）、互換性の問題を生じさせる可能性がある。本明細書に記載されている解決策を使用すると、第一のフォーリン関数の（例えば、フォーリンバイナリ109内）のメモリアドレスを取得する（又は、「採用する」）際に、実施形態では、ECネイティブコード108b内の第一のネイティブ関数とECネイティブコード108b内の第二のネイティブ関数の両方が、第一のフォーリン関数のための同じメモリアドレスを取得することが確実となり、それは第一のフォーリン関数が始まる真のメモリアドレスである。それに加えて、実施形態では第二のフォーリン関数（例えば、フォーリンバイナリ109内）が第一のフォーリン関数のためのその同じ真のメモリアドレスを取得することも確実となる。実施形態ではまた、ネイティブ関数のアドレスが、そのアドレスがフォーリン関数により採用されるか、他のネイティブ関数により採用されるかに関係なく、同じであることも確実となる。

10

【0051】

【060】 実施形態において、一貫したメモリアドレス参照は、ハイブリッドバイナリ108内のECルックアップ構造307（例えば、コンパイラツールチェーン107dによりハイブリッドバイナリ108の中に送信される）と、ディスパッチャ112（すなわち、ライブラリ106a内のディスパッチャ112a及びエミュレータ106b内のディスパッチャ112b）によりイネーブルされる。実施形態において、ECルックアップ構造307は、ハイブリッドバイナリ108により定義されるメモリイメージのメモリアドレスのうちECネイティブコード108bを含む範囲を特定するために使用可能なあらゆる種類の構造である。実施形態において、ECルックアップ構造307はビットマップであり、これはメモリの対応する範囲（例えば、メモリページ）がECネイティブコード108bを含むか否かを示すために1ビット値を有する。しかしながら、ECルックアップ構造307は、ハッシュテーブル又はバイナリツリー等、代替的な種類のデータ構造を含むことができる。

20

【0052】

【061】 実施形態において、コールが呼び出し元関数から呼び出し先関数に行われているとき、ディスパッチャ112はECルックアップ構造307を使って呼び出し先関数のためのディスティネーションメモリアドレスがECネイティブコード内にあるか否かを特定する。すると、呼び出し元関数がネイティブかフォーリンコードかが本来わかっていることにより、ディスパッチャ112はコールを適切にディスパッチする。実施形態において、ディスパッチャ112は少なくとも、ネイティブの呼び出し元とネイティブの呼び出し先、ネイティブの呼び出し元とフォーリンの呼び出し先、フォーリンの呼び出し元とフォーリンの呼び出し先、及びフォーリンの呼び出し元とネイティブの呼び出し先の4つのシナリオで動作する。

30

【0053】

【062】 第一のシナリオでは、呼び出し元関数はEC ABI 107bに基づいて実行されるネイティブ関数であり、コールはそれゆえライブラリ106a内のディスパッチャ112aにより扱われる。ディスパッチャ112aはECルックアップ構造307を使って、呼び出し先の参照メモリアドレスがECネイティブコード108bに対応するメモリ領域内にあり、したがって呼び出し先関数もまたEC ABI 107bに基づいて実行されるネイティブ関数であることを特定する。この状況では、呼び出し元は呼び出し先関数の真のメモリアドレスを呼び出ししており、サンクは必要なく、したがって、ディスパッチャ112aは参照メモリアドレスを使って呼び出し先関数を直接起動させる。

40

【0054】

【063】 第二のシナリオでは、呼び出し元関数はEC ABI 107bに基づいて実行されるネイティブ関数であり、コールはそれゆえ、ライブラリ106a内のディスパッチャ112aにより扱われる。ディスパッチャ112aはECルックアップ構造307を使って、呼び出し先の参照メモリアドレスがECネイティブコード108bに対応するメモリ領域内になく、したがって呼び出し先関数はフォーリンABI 107cに基づいて

50

実行されるフォーリン関数であることを特定する。図 4 Aを参照すると、これはパス # 2 の状況であり、EC ネイティブ関数 404 はエグジットサンク 402 を介してフォーリン関数を呼び出す。そのため、ディスパッチャ 112 a は参照メモリアドレス（すなわち、呼び出し先関数の）を呼び出すことができず、なぜなら、それがエグジットサンク 402 を迂回するからである。その代わりに、ディスパッチャ 112 a はエグジットサンク 402 への新しい参照アドレスを特定し、この新しい参照メモリアドレスを使ってエグジットサンク 402 を起動する。実施形態において、エグジットサンク 402 は次に、EC ABI 107 b の CC をフォーリン ABI 107 c の CC に適応させ、その後、エミュレータ 106 b を起動する。エミュレータ 106 b は次に、当初の参照メモリアドレスを使って呼び出し先関数を直接呼び出す。

10

【0055】

【064】 実施形態において、エグジットサンクまでの新しい参照アドレスは、呼び出し先関数自体の中に含まれる。これらの実施形態では、コールサイトは 2 つの参照メモリアドレス、すなわち呼び出し先関数の当初の参照メモリアドレスとエグジットサンクまでの新たな参照アドレスに関連付けられる。ハイブリッドバイナリ 108 が既存の ABI 107 a に基づいてロードされる場合、既存の ABI 107 a により使用されるディスパッチャは、呼び出し先関数の当初の参照メモリアドレスを直接使用し、エグジットサンクまでの新しい参照アドレスは無視する。特に、既存の ABI 107 a がエグジットサンクまでの新たな参照アドレスを無視できるのは、これがフォーリンコードと相互作用していないからである。それに対して、ハイブリッドバイナリ 108 が EC ABI 107 b に基づいてロードされる場合は、ディスパッチャ 112 a はフォーリンコードと相互作用するためにエグジットサンクまでの新たな参照アドレスも利用する。特に、コンパイラツールチェーン 107 d により、関数が既存の ABI 107 a と EC ABI 107 b のどちらをターゲットとするかを問わず、当初の参照メモリアドレスと新たな参照アドレスの両方をコンパイルコードに含めることによって折り畳み可能性が向上する。

20

【0056】

【065】 第三のシナリオでは、呼び出し元関数は、エミュレータ 106 b の中でフォーリン ABI 107 c に基づいて実行されるフォーリン関数であり、コールはそれゆえ、エミュレータ 106 b の中のディスパッチャ 112 b により扱われる。ディスパッチャ 112 b は、EC ルックアップ構造 307 を使って、呼び出し先の参照メモリアドレスが EC ネイティブコード 108 b に対応するメモリ領域内になく、したがって呼び出し先関数はフォーリン ABI 107 c に基づいて実行されるフォーリン関数でもあることを特定する。この状況では、呼び出し元は呼び出し先関数の真のメモリアドレスを呼び出しており、サンクは不要であるため、ディスパッチャ 112 b は、エミュレータ 106 b 内で参照メモリアドレスを使って呼び出し先関数を直接起動する。

30

【0057】

【066】 第四のシナリオでは、呼び出し元関数は、エミュレータ 106 b の中でフォーリン ABI 107 c に基づいて実行されるフォーリン関数であり、コールはそれゆえ、エミュレータ 106 b の中でディスパッチャ 112 b により扱われる。ディスパッチャ 112 b は EC ルックアップ構造 307 を使って、呼び出し先の参照メモリアドレスが EC ネイティブコード 108 b に対応するメモリ領域内にあり、呼び出し先関数がしたがって、EC ABI 107 b に基づいて実行されるネイティブ関数であることを特定する。図 4 A を参照すると、これはパス # 2 の状況であり、フォーリン関数は EC ネイティブ関数 404 を、エントリサンク 401 を介して呼び出している。そのため、ディスパッチャ 112 b は参照メモリアドレス（すなわち、呼び出し先関数の）を直接呼び出すことができず、それは、これがエントリサンク 401 を迂回するからである。その代わりに、ディスパッチャ 112 b はエントリサンク 401 までの新たな参照アドレスを設定して、新しい参照メモリアドレスを使ってエントリサンク 401 を起動する。実施形態において、エントリサンク 401 は次に、フォーリン ABI 107 c の CC を EA ABI 107 b の CC に適応させて、当初の参照メモリアドレスを使って呼び出し先関数を起動する。

40

50

【0058】

[067] 特に、第四のシナリオでは、呼び出し元関数はEC ABI 107bを認識して、又はそれと互換的に設計されていないレガシフォーリンコードであり得る。それゆえ、第二のシナリオとは異なり、呼び出し元関数は、エントリサンのアドレスを含むように（例えばコンパイラツールチェーンによって）変更できない。実施形態において、ディスパッチャ112bは、呼び出し先関数の当初の参照メモリアドレスの直前のメモリブロックからエントリサンのための新たな参照アドレスを取得する。実施形態において、エントリサンのための新たな参照アドレスを取得するためのデータは、ハイブリッドバイナリ108の生成中にコンパイラツールチェーン107dによってこのメモリブロックに挿入されている。このメモリブロック内に含まれる特定のデータは変更できるが、いくつかの実施形態では、これは（例えば、ハイブリッドバイナリ108により定義されるメモリイメージの開始からの）メモリオフセットである。他の実施形態において、これはエントリサンまでの直接的なアドレス参照とすることができる。他の実施形態において、ディスパッチャ112bは新たな参照アドレスは、何らかの代替的な方法で、例えばアドレスのデータツリー又は順序付けされた配列から取得する。

10

【0059】

[068] 実施形態において、EC ABI 107bは、コンパイラツールチェーン107d及びディスパッチャ112と共に、「互換性モード」をサポートし、これはネイティブ関数を呼び出すフォーリンコードがネイティブ関数の開始部に挿入されたフォーリンコードをうまく識別し、おそらくはパッチすることができるようにする。これによって、フォーリンコードは、呼び出された関数の開始部を（例えば、ウイルス保護、複製防止等の一環として）逆アセンブルし、及び／又はその関数を呼び出すことに関連して、呼び出された関数の開始部を（例えば、プロファイリングリダイレクションの一環として）パッチすることができる。フォーリンコードは、それがエミュレートされていることを認識し得ず、呼び出された関数の開始部における認識可能なフォーリンコードを発見すると予想し得るため、いくつかの実施形態において、コンパイラツールチェーン107dはハイブリッドバイナリ108の中に、ネイティブ関数のための「早送りシーケンス」（サンの形態）を送信し、その上でフォーリンコードは逆アセンブリ及び／又はパッチングのために動作できる。実施形態において、「早送りシーケンス」は、プレースホルダフォーリンコードを含み、これは参照で終わるか、ネイティブ関数の真のアドレスへとジャンプする。実施形態において、この早送りシーケンスは、ECルックアップ構造307の中でECネイティブコードを含まないと表示されたメモリページに保存される。それゆえ、実施形態において、フォーリン呼び出し元が早送りシーケンスを有するネイティブ関数を呼び出す場合、このコールは当初、フォーリンフォーリンコールとして扱われ（すなわち、前述の第三のシナリオと同様）、それによってエミュレータ106bは早送りシーケンスを実行して、早送りシーケンスの終了時にフォーリンネイティブコールを行う（すなわち、前述の第四のシナリオと同様）。

20

30

【0060】

[069] 特に、この配置によって二重サンキングパフォーマンスの問題を生じさせる可能性があり、この場合、エミュレータ106bが早送りサンクの中の2、3の命令のみをエミュレートするように起動され（例えば、第三のシナリオと同様）、それによって他のサンク、すなわちネイティブ関数へのエントリサンへのコールを開始する（例えば、第四のシナリオと同様）。いくつかの実施形態は、この二重サンキングパフォーマンスの問題を、呼び出し先の参照メモリアドレスがECネイティブコード108bに対応するメモリ領域内にないことを（すなわち、ECルックアップ構造307を使って）特定した後に、追加のチェックを実行するようにディスパッチャ112b（エミュレータ106bの中）を構成することによって回避する。この追加のチェックは、呼び出し先関数の開始時に「のぞいて見て」、早送りシーケンスのシグナチャが存在するか否かを特定することである。存在すれば、ディスパッチャ112bは、ネイティブ関数のアドレスを早送りシーケンスから取得し、ネイティブ関数へのコールを直接開始する（すなわち、前述の第四のシナ

40

50

リオと同様)。この追加のチェックを行うことにより、エミュレータ 106b による早送りサンクの処理が回避される。

【0061】

【070】 以上の導入部を考慮して、以下の議論はここで様々な方法及び方法行為に言及する。方法行為は特定の順序で論じられているか、又はフローチャートの中で特定の順序で行われるように描かれているかもしれないが、いかなる特定の順序も、特に明示されないかぎり、又はある行為が、他の行為がその行為の実行より前に完了していることに依存するために要求されないかぎり、必要ではない。

【0062】

【071】 フォーリンコードのエミュレーションをサポートするためのネイティブエミュレーション互換可能アプリケーションバイナリインタフェース

10

【0063】

【072】 図5は、ソースコードを、ネイティブの第一のISAをターゲットとしながら、フォーリンの第二のISAにコンパイルされたコードと動作可能なバイナリファイルへとコンパイルする例示的な方法500のフローチャートを示す。まず、方法500は第一のISAのための第一及び第二のABIと、第二のISAのための第三のABIを識別する行為501を含み、第二のABIは第三のABIのコンテキストデータサイズを超えないコンテキストデータを定義し、第三のABIにマッピングされるレジスタのサブセットの使用を定義する。実施形態において、行為501は(i)第一のISAに対応する第一のABIと、(ii)第一のISAに対応する第二のABIと、(iii)第二のISAに対応する第三のABIを識別することを含む。ある例において、コンパイラツールチェーン107dは、既存のABI 107a(すなわち、第一のABI)、EC ABI 107b(すなわち、第二のABI)、及びフォーリンABI(すなわち、第三のABI)の各々を識別する。

20

【0064】

【073】 これらの実施形態において、第二のABIは、第三のABIが使用する第二のコンテキストデータの第二のサイズを超えない第一のサイズを有する第一のコンテキストデータフォーマットを定義する。図3Aを参照すると、例えば、コンテキストデータ202bはEC ABI 107bにより、(フォーリンABI 107cにより定義される)コンテキストデータ202cのサイズを超えないサイズを有するように定義される。

30

【0065】

【074】 これらの実施形態において、第二のABIはまた、第一のISAの複数のレジスタの中から第一のABIにより使用されるレジスタのサブセットの使用を定義する。幾つかの実施形態において、レジスタのサブセットは第一のISAの複数のレジスタの一部を含む(例えば、例200と同様)が、他の実施形態では、レジスタのサブセットは第一のISAの複数のレジスタの全部を含む。実施形態において、このレジスタのサブセットは、第三のABIにより使用される第二のISAのレジスタセットにマッピングされる。実施形態において、第二のABIは、第一のABIにより許可された1つ又は複数の他のレジスタを除き、レジスタのサブセットの使用のみを許可する。図3Aを参照すると、EC ABI 107bは利用可能なレジスタセット201bを定義し、これらは既存のABI 107aにより使用される、利用可能なレジスタ201aのサブセットである(すなわち、既存のABI 107aにより使用される利用可能なレジスタ201aのうちの1つ又は複数省く)。それに加えて、利用可能なレジスタ201bは、EC ABI 107bにより、フォーリンABI 107cにより使用される利用可能なレジスタセット201cにマッピングされる。

40

【0066】

【075】 実施形態において、第二のABIによるレジスタのサブセットの使用により、関数の少なくとも一部は、第一のABI及び第二のABIの各々を使ってコンパイルされるときに折り畳み可能となる。例えば、実施形態において、利用可能なレジスタ201bの選択は、利用可能なレジスタ201aを十分に密接にミラーし、それによって少なくと

50

も幾つかのケースにおいて、既存の A B I 1 0 7 a 及 E C A B I 1 0 7 b の各々に向けてコンパイルされた関数によって同じコンパイルコードが得られる。実施形態において、第二の A B I によるレジスタのサブセットの使用により、第二の A B I をターゲットとしてコンパイルされた関数の少なくとも一部を、エントリサックを用いずに第三の A B I により呼び出されることができる。例えば、第一の A B I が A A r c h 6 4 A B I であり、第三の A B I が W i n d o w s - X 6 4 A B I である場合、第二の A B I を 4 つ以下の入力パラメータを受信するネイティブ関数のためのエントリサックを省略できるような方法で定義することが可能となり得る。

【0067】

[076] 方法 5 0 0 はまた、関数をバイナリに送信する行為 5 0 2 も含む。図のように、行為 5 0 2 は、第一の A B I をターゲットとして関数をコンパイルし、第一のコンパイルバージョンを作成する行為 5 0 3 と、第二の A B I をターゲットとして関数をコンパイルし、第二のコンパイルバージョンを作成する行為 5 0 4 を含む。図のように、行為 5 0 3 及び 5 0 4 の間に特定の順序はなく、これらの行為は並行にも（何れかの順序で）逐次的にも実行できると理解されたい。実施形態において、ソースコードで定義された関数について、行為 5 0 3 は、第一の I S A で第二の命令セットを生成することを含めて、第二の A B I をターゲットとする関数の第二のコンパイルバージョンを生成することを含み、行為 5 0 4 は、第一の I S A で第一の命令セットを生成することを含めて、第一の A B I をターゲットとする関数の第一のコンパイルバージョンを生成することを含む。ある例において、コンパイラツールチェーン 1 0 7 d は、ソースコード関数を 2 回、すなわち第一の A B I をターゲットとする間に 1 回と、第二の A B I をターゲットとする間に 1 回コンパイルし、その結果、この関数の 2 つのコンパイルバージョンが得られる。前述のように、実施形態において、コンパイラツールチェーン 1 0 7 d は、既存の A B I 1 0 7 a をターゲットとするときには「ネイティブ I S A」ロジックパスを採用し、E C A B I 1 0 7 b をターゲットとするときには「フォーリン I S A」ロジックパスを採用する。それゆえ、幾つかの実施形態において、行為 5 0 2 において第一の A B I をターゲットとする関数の第一のコンパイルバージョンを生成することは第一の I S A のソースコード定義を使用することを含み、行為 5 0 3 において第二の A B I をターゲットとする関数の第二のコンパイルバージョンを生成することは第二の I S A のソースコード定義を使用することを含む。

【0068】

[077] 行為 5 0 2 はまた、第一及び第二のコンパイルバージョンがマッチし、したがって折り畳み可能であるか否かを特定する行為 5 0 5 も含む。実施形態において、行為 5 0 5 は、関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンがバイナリファイルの中で折り畳み可能か否かを、少なくとも第一の命令セットと第二の命令セットがマッチするか否かを特定することに基づいて特定することを含む。ある例において、コンパイラツールチェーン 1 0 7 d は、行為 5 0 3 及び行為 5 0 4 において生成される関数の 2 つのコンパイルバージョン間の比較を行う。これらがマッチすると、関数の 2 つのコンパイルバージョンは折り畳み可能であり、そうでなければこれらは折り畳み不能である。

【0069】

[078] 行為 5 0 5 の結果に応じて、行為 5 0 2 は、両方のコンパイルバージョンをバイナリに送信する行為 5 0 6（コンパイルされた関数が折り畳みか不能であるとき）又は、一方のコンパイルバージョンのみをバイナリに送信する行為 5 0 7（コンパイルされた関数が折り畳み可能である場合）を含む。実施形態において、行為 5 0 6 は、少なくとも関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンがバイナリファイルの中で折り畳み可能であるか否かを特定することに基づいて、関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンがバイナリファイル内で折り畳み不能であるときに、関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンの両方をバイナリファイルに送信することを含む。ある例において、コンパイラツールチ

チェーン 107d は、コンパイルされた関数が折り畳み不能であるとき、関数の既存バージョン（例えば、既存のネイティブ関数 403）と関数の EC バージョン（例えば、EC ネイティブ関数 404）の両方をハイブリッドバイナリ 108 に送信する。実施形態において、行為 507 は、少なくとも関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンがバイナリファイルの中で折り畳み可能であるか否かを特定することに基づいて、関数の第一のコンパイルバージョンと関数の第二のコンパイルバージョンがバイナリファイルの中で折り畳み可能であると特定されたとき、関数の第一のコンパイルバージョン又は関数の第二のコンパイルバージョンの一方のみをバイナリファイルに送信することを含む。ある例において、コンパイラツールチェーン 107d は、コンパイル関数が折り畳み可能であるとき、関数の 1 つの EC バージョン（例えば、EC ネイティブ関数 406）をハイブリッドバイナリ 108 に送信する。

10

【0070】

【079】 実施形態において、第二の ABI により、バイナリファイルは、(i) 第一の ABI を実装するが、第二の ABI を実装しない第一のコンピュータシステムと、(ii) 第二の ABI を実装する第二のコンピュータシステムの両方でネイティブとして実行できる。例えば、方法 500 を使って生成されたハイブリッドバイナリ 108 は、既存の ABI 107a のみを実装する「レガシ」コンピュータシステム上で、又は EC ABI 107b も実装する現代のコンピュータシステム上でも実行可能である。

【0071】

【080】 幾つかの実施形態において、方法 500 はまた、エントリサックをバイナリファイルに送信することも含み、エントリサックは、第三の ABI を第二の ABI に適応させる第一の ISA 内のコードを含む。ある例において、行為 506 が実行されるとき、コンパイラツールチェーン 107d は、送信された関数（例えば、EC ネイティブ関数 404）の送信された EC バージョンのためのエントリサックを送信し、フォーリンコードがその関数を呼び出すことができるようにする。他の例において、行為 507 が実行されるとき、コンパイラツールチェーン 107d は、EC 関数（例えば、EC ネイティブ関数 406）と考えることのできる 1 つの送信された関数のためのエントリサックを送信し、フォーリンコードがこの関数を呼び出すことができるようにする。

20

【0072】

【081】 幾つかの実施形態において、方法 500 はまた、1 つ又は複数のエグジットサックをバイナリファイルに送信することも含み、各エグジットサックは、第二の ABI を第三の ABI に適応させる第一の ISA のコードを含む。ある例において、行為 506 が実行されるとき、コンパイラツールチェーン 107d は、関数（例えば、EC ネイティブ関数 404）の送信された EC バージョンのための 1 つ又は複数のエグジットサックを送信して、関数がフォーリンコードを呼び出すことができるようにする。他の例において、行為 507 が実行されるとき、コンパイラツールチェーン 107d は、1 つの送信された関数のための 1 つ又は複数のエグジットサックを送信して、関数がフォーリンコードを呼び出すことができるようにする。幾つかの実施形態において、方法 500 は、関数による個々の異なるコールに対して異なるエグジットサックを生成するが、他の実施形態では、方法 500 は各呼び出し先について 1 つのエグジットサックを生成することもできる。

30

40

【0073】

【082】 前述のように、ハイブリッドバイナリは、追加のフォーリンコードストリーム、例えばフォーリンコードを含むコードストリーム 301c を含み得る。それゆえ、方法 500 は、第二の ISA の第三の命令セットを生成することを含めた、第三の ABI をターゲットとする関数の第三のコンパイルバージョンを生成することと、関数の第三のコンパイルバージョンをバイナリファイルに送信することをさらに含むことができる。

【0074】

【083】 方法 500 の幾つかの変形型では、「既存の」コードストリームを省略し得て、送信されたバイナリは、EC ABI 107b をターゲットとするが、既存の ABI 107a をターゲットとしないものとなる。理解されるように、結果として得られるバイ

50

ナリは、フォーリン A B I 107c をターゲットとするコードとの互換性を提供するが、既存 A B I 107a は実装しても、E C A B I 107b を実装しないプログラムとの後方互換性はない。実施形態において、「既存」のコードストリームを省略することは、既存の A B I 107a を使用しないアプリケーションで使用されることだけが意図されるバイナリにおいて有益であり得る。一例として、ネイティブアプリケーションとの併用のためにデプリケート (deprecate) されているが、エミュレートされたアプリケーションとの互換性のために保持される機能性 (例えば、旧式のビデオフォーマットのためのコーデックサポート) を実装するバイナリ。

【0075】

【084】 コードストリームの折り畳みをサポートするハイブリッドバイナリ

10

【0076】

【085】 図5はコードの折り畳みを可能にする E C A B I 107b を使ってコードを生成する方法に焦点を当てたものであったが、ハイブリッドバイナリ108は、レガシシステム (例えば、既存の A B I 107a のみを有する) との後方互換性を実現し、他方で現代のシステム (例えば、E C A B I 107b を有する) がフォーリンコードと相互作用するために追加の特徴を利用できるようにする追加の特徴を含み得る。これらのデュアルアーキテクチャハイブリッドバイナリ108の作成についてさらに説明するために、図6はハイブリッドバイナリイメージを生成する例示的な方法800のフローチャートを示し、このハイブリッドイメージは、ネイティブ A B I 及び互換性 A B I の両方に基づいて実行できる。その後、これらのデュアルアーキテクチャハイブリッドバイナリ108の実行についてさらに説明するために、図7は、互換性 A B I に基づいて実行されるプロセスによりハイブリッドバイナリイメージを実行する例示的な方法700のフローチャートを示しており、このハイブリッドバイナリイメージは、ネイティブ A B I と互換性 A B I の両方に基づいて実行可能である。

20

【0077】

【086】 まずバイナリイメージの作成及び図6に関して、方法600は複数の行為 (すなわち、行為601~604) を含み、これらは、フローチャート中で示されるように、相互に関して何れの順序でも実行できる。図のように、方法600は、ネイティブ A B I のマシンタイプを識別し、ネイティブ A B I により読み出されるマシンタイプフィールドの中に送信する行為601を含む。幾つかの実施形態において、行為601は、少なくともネイティブ A B I に対応する第一のマシンタイプを識別することに基づいて、第一のマシンタイプをハイブリッドバイナリイメージのマシンタイプフィールドの中に送信することを含み、マシンタイプフィールドは、ハイブリッドバイナリイメージが、ネイティブ A B I に基づいて実行されるネイティブプロセスによりロードされる際に利用されるように構成される。ある例において、コンパイラツールチェーン107dは、ハイブリッドバイナリ108の中に、既存の A B I 107a をターゲットとするバイナリをロードするときに、既存の A B I 107a により想定される識別子とマッチするマシン識別子303を送信する。既存の A B I 107a により想定される識別子を送信することにより、既存の A B I 107a は、ハイブリッドバイナリ108が E C A B I 107b との互換性も有していても、ハイブリッドバイナリ108をロードするときに何が想定されるかわかる。

30

40

【0078】

【087】 方法600はまた、折り畳み不能なバージョンを識別し、送信する行為602も含む。図のように、行為602は、(i) ネイティブ A B I に基づいて実行される折り畳み不能な関数の第一のコンパイルバージョンを送信する行為602aと、(ii) 互換性 A B I に基づいて実行される折り畳み不能な関数の第二のコンパイルバージョンを送信する行為602bの両方を含む。図のように、行為602a及び602b間には何れの特定の順序も示されておらず、これらの行為は並列にも、(何れかの順序で) 逐次的にも実行できると理解されたい。幾つかの実施形態において、行為602aは、少なくとも折り畳み不能な第一の関数を識別することに基づいて、ハイブリッドバイナリイメージの中に

50

、ネイティブ A B I に基づいて実行可能な第一の関数の第一のコンパイルバージョンを送信することを含み、他方で、行為 6 0 2 b は、少なくとも折り畳み不能な第一の関数を識別することに基づいて、ハイブリッドバイナリイメージに、互換性 A B I に基づいて実行可能な第一の関数の第二のコンパイルバージョンを送信することを含む。ある例として、少なくとも（例えば、方法 5 0 0 の行為 5 0 3 及び 5 0 4 において）ソースコード関数を異なるコンパイルバージョンにコンパイルすることに基づいて、また、（例えば、方法 5 0 0 の行為 5 0 6 において）これらのコンパイルバージョンが折り畳み不能であると特定したことにも基づいて、コンパイラツールチェーン 1 0 7 d は、（例えば、方法 5 0 0 の行為 4 0 5 の一環として）行為 6 0 2 a 及び行為 6 0 2 b の中でこれらのコンパイルバージョンの両方をハイブリッドバイナリ 1 0 8 に送信する。

10

【0079】

【088】 方法 6 0 0 はまた、ネイティブ A B I 及び互換性 A B I の両方に基づいて実行される折り畳み可能な関数を識別し、送信する行為 6 0 3 も含み、これには、折り畳み可能な関数のコンパイルバージョンを折り畳み不能の関数の第一のコンパイルバージョンへのコールと共に送信することが含まれる。幾つかの実施形態において、行為 6 0 3 は、少なくとも折り畳み可能な第二の関数を識別することに基づいて、ハイブリッドバイナリイメージの中に、ネイティブ A B I と互換性 A B I の両方に基づいて実行可能な第二の関数のコンパイルバージョンを送信することを含む。ある例において、少なくとも（例えば、方法 5 0 0 の行為 5 0 3 及び 5 0 4 において）異なるコンパイルバージョンにソースコード関数をコンパイルすることに基づいて、また、（例えば、方法 5 0 0 の行為 5 0 5 において）これらのコンパイルバージョンが折り畳み可能であると特定したことにも基づいて、コンパイラツールチェーン 1 0 7 d は、行為 6 0 3 の中で（方法 5 0 0 の行為 5 0 7 の一環として）ハイブリッドバイナリ 1 0 8 にこれらのコンパイルバージョンの一方のみを送信する。

20

【0080】

【089】 実施形態において、第二の関数のコンパイルバージョンは、ハイブリッドバイナリイメージがネイティブプロセスによりロードされるときに第一の関数の第一のコンパイルバージョンを呼び出すように構成される。ある例において、コンパイラツールチェーン 1 0 7 d はハイブリッドバイナリ 1 0 8 を、行為 6 0 3 の中で送信されたコードが行為 6 0 2 a で送信されたコードを「デフォルト」で呼び出し、ハイブリッドバイナリ 1 0 8 が既存の A B I 1 0 7 a に基づいてロードされたときに、（行為 6 0 2 b で送信されたコードではなく）行為 6 0 2 a で送信されたコードが実行されるように構成する。

30

【0081】

【090】 方法 6 0 0 はまた、互換性 A B I により利用されるフィックスアップテーブルを送信する行為 6 0 4 も含む。幾つかの実施形態において、行為 6 0 4 は、ハイブリッドバイナリイメージの中に、ハイブリッドバイナリが、互換性 A B I に基づいて実行される互換性プロセス（例えば、エミュレートされたプロセス 1 1 1）によってロードされるときに利用されるように構成されたフィックスアップテーブルを送信することを含み、フィックスアップテーブルは、ハイブリッドバイナリイメージからロードされるメモリへの複数の変換を定義する。ある例において、コンパイラツールチェーン 1 0 7 d は、ハイブリッドバイナリ 1 0 8 の中に、ハイブリッドバイナリ 1 0 8 が E C A B I 1 0 7 b に基づいてロードされたときに適用される変換を含むフィックスアップテーブル 1 0 8 c を送信し、これはハイブリッドバイナリ 1 0 8 の E C 機能が利用されるようにする。

40

【0082】

【091】 実施形態において、フィックスアップテーブル 1 0 8 c の複数の変換は、マシンタイプフィールドを、互換性 A B I に対応する第二のマシンタイプを含むように調整する変換を含む。ある例において、コンパイラツールチェーン 1 0 7 d は、フィックスアップテーブル 1 0 8 c の中に、（システムメモリ 1 0 4 の中で）マシン識別子 3 0 3 からシステムメモリ 1 0 4 にロードされるマシン識別子をフォーリン A B I 1 0 7 c に対応するフォーリン I S A とマッチするマシン識別子に置き換えるメモリ変換を送信する。

50

【0083】

【092】 実施形態において、フィックスアップテーブル108c内の複数の変換は、（折り畳み可能な）第二の関数のコンパイルバージョンを、（折り畳み不能な）第一の関数の第一のコンパイルバージョンではなく（折り畳み不能な）第一の関数の第二のコンパイルバージョンを呼び出すように構成する変換を含む。ある例において、コンパイラツールチェーン107dは、フィックスアップテーブル108cの中に、（折り畳み不能な）第一の関数の第一のコンパイルバージョンを参照する第一のメモリアドレスを、（折り畳み不能な）第一の関数の第二のコンパイルバージョンを参照する第二のメモリアドレスに置き換えるメモリ変換を送信する。第一のメモリアドレスは（折り畳み可能な）第二の関数のコンパイルバージョンの中で直接第二のメモリアドレスに直接置き換えることができるが、実施形態において、（折り畳み可能な）第二の関数のコンパイルバージョンは、（折り畳み不能な）第一の関数の第一のコンパイルバージョンを、（折り畳み不能な）第一の関数の第一のコンパイルバージョンを参照するディスパッチテーブル306を介して呼び出すように構成される。これらの実施形態では、（折り畳み可能な）第二の関数のコンパイルバージョン自体を変更するのではなく、この変換はディスパッチテーブル306を、（折り畳み不能な）第一の関数の第一のコンパイルバージョンの代わりに（折り畳み不能な）第一の関数の第二のコンパイルバージョンを参照するディスパッチテーブル306を変更する。さらに、これらの実施形態において、方法600はまた、ディスパッチテーブルをハイブリッドバイナリイメージの中に送信することを含む。

10

【0084】

【093】 実施形態において、方法600はまた、ハイブリッドバイナリイメージの中に、（折り畳み可能な）第一の関数の第一のコンパイルバージョンを参照するエントリポイント304を送信することを含む。実施形態において、このエントリポイント304は、ハイブリッドバイナリイメージがネイティブプロセスによって（例えば、既存のABI 107aを使って）ロードされるときに利用されるように構成される。実施形態において、フィックスアップテーブル108cの中の複数の変換は、エントリポイントを、その代わりに（折り畳み可能な）第一の関数の第二のコンパイルバージョンを参照するように調整する変換を含む。調整されたこのエントリポイントはそれゆえ、ハイブリッドバイナリイメージが互換性プロセスによって（例えば、EC ABI 107bを使って）ロードされる時に利用される。

20

30

【0085】

【094】 図3A及び3Bに関して述べたように、ハイブリッドバイナリ108は、インポートテーブル又はエクスポートテーブル（インポート／エクスポートテーブル305）の1つ又は複数を含み得て、ハイブリッドバイナリ108は、既存のABI 107aに關係する関数（及びその位置）を露出させる「ネイティブ」ビューを示すように構成される。例えば、図3Bに示されているように、インポート／エクスポートテーブル305は、ネイティブビュー307aを露出させるベースリファレンスのサイズ／カウントを使って参照できる。それゆえ、実施形態において、方法600は、ハイブリッドバイナリイメージの中に、（i）（折り畳まれていない）第一の関数の第一のコンパイルバージョンを使って少なくとも（折り畳まれていない）第一の関数を参照する第一のゾーン、（ii）第二の関数のコンパイルバージョンを使って少なくとも（折り畳まれた）第二の関数を参照する第二のゾーン、及び（iii）（折り畳まれていない）第一の関数の第二のコンパイルバージョンを使って、少なくとも（折り畳まれていない）第一の関数を参照する第三のゾーンを含む1つ又は複数のテーブルを送信することを含む。

40

【0086】

【095】 実施形態において、これらのテーブルの「ネイティブ」ビューを提供するために、方法600は、ハイブリッドバイナリイメージの中に、第一のゾーンと第二のゾーンを含むが第三のゾーンを含まない1つ又は複数のテーブルのネイティブビューを提供する1つ又は複数のテーブルへの参照を送信することを含む。この参照は、ハイブリッドバイナリイメージがネイティブプロセス（例えば、既存のABI 107aに対応する）によ

50

ってロードされるときに利用されるように構成される。これらの実施形態において、フィックスアップテーブル 108c の複数の変換はすると、1つ又は複数のテーブルのへ参照を、第二のゾーンと第三のゾーンを含むが第一のゾーンを含まない1つ又は複数のテーブル (EC ABI 107b が使用するため) の互換性ビューを提供するように調整する変換を含む。それゆえ、実施形態において、1つ又は複数のテーブルへの参照は、オフセットとサイズを明示することにより1つ又は複数のテーブルのネイティブビューを提供し、フィックスアップテーブル 108c の中の変換は、オフセット又はサイズの1つ又は複数を変更することによってこれらのテーブルの互換性ビューを提供するように参照を調整する。

【0087】

[096] 実施形態において、送信されたテーブルはインポートテーブルを含み、それゆえ、方法 600 の実施形態において、1つ又は複数のテーブルは1つ又は複数のインポートテーブルを含み、第一、第二、及び第三のエントリは第一、第二、及び第三の関数インポートを含む。追加的又は代替的な実施形態において、送信されたテーブルは、エクスポートテーブルを含み、それゆえ、方法 600 の実施形態において、1つ若しくは複数のテーブルは、1つ又は複数のエクスポートテーブルを含み、第一、第二、及び第三のエントリは第一、第二、及び第三の関数エクスポートを含む。

【0088】

[097] 次に、バイナリイメージの利用と図 7 に関して、方法 700 は、互換性 ABI を使用するプロセス内でハイブリッドバイナリのローディングを開始する行為 701 を含む。ある例において、ローダ 106c は、EC ABI 107b を実装するライブラリ 106a の1つを利用して、ハイブリッドバイナリ 108 のローディングを開始する。

【0089】

[098] 方法 700 はまた、ハイブリッドバイナリの中に保存されたマシンタイプが互換性 ABI のためのマシンタイプとミスマッチすることを特定する行為 702 も含む。幾つかの実施形態において、行為 702 は、ハイブリッドバイナリイメージのローディング中に、ハイブリッドバイナリイメージのマシンタイプフィールドの中に保存された第一のマシンタイプが、それに基づいてプロセスが実行されている互換性 ABI に対応する第二のマシンタイプとミスマッチすることを特定することを含む。ある例において、ハイブリッドバイナリ 108 をローディングすることの一環として、ローダ 106c はマシン識別子 303 を含むメモリページをシステムメモリ 104 にコピーする。すると、ローダ 106c は、マシン識別子 303 の値が EC ABI 107b (例えば、フォーリン ISA 107c に対応するフォーリン ISA) について想定される値とミスマッチすることを特定する。

【0090】

[099] 方法 700 はまた、ミスマッチに基づいて、フィックスアップテーブルを位置付ける行為 703 も含む。幾つかの実施形態において、行為 703 は、第一のマシンタイプミスが第二のマシンタイプとミスマッチすることを特定したことに基づいて、バイナリイメージの中に、ハイブリッドバイナリイメージからロードされたメモリへの複数の変換を定義するフィックスアップテーブルを位置付けることを含む。ある例において、ローダ 106c は、例えばハイブリッドバイナリ 108 内の所定のアドレス若しくはオフセットを、又はハイブリッドバイナリ 108 からロードされたメモリページによりポピュレートされたシステムメモリ 104 の一部の中の所定のアドレス又はオフセットを参照することによって、ハイブリッドバイナリ 108 の中でフィックスアップテーブル 108c を識別する。

【0091】

[100] 方法 700 はまた、フィックスアップテーブルの中の1つ又は複数の変換をハイブリッドバイナリからロードされたメモリに適用する行為 704 も含む。幾つかの実施形態において、行為 704 は、複数の変換の少なくとも一部をハイブリッドバイナリイメージからロードされたメモリに適用することを含む。ある例において、ローダ 106c は

10

20

30

40

50

、フィックスアップテーブル 108 c から得た 1 つ又は複数の変換を、ハイブリッドバイナリ 108 からロードされるメモリページによりポピュレートされたシステムメモリ 104 の 1 つ又は複数の部分に適用する。

【0092】

【101】 ロダ 106 c はフィックスアップテーブル 108 c 内の全ての変換を同時に適用することもできるが、実施形態において、ロダ 106 c はこれらをページごとに、これらのページがハイブリッドバイナリ 108 からロードされるたびに（例えば、ページフォルトハンドリングルーチンの一環として）適用する。そのため、図 7 において、行為 704 は行為 704 に戻る矢印と共に示されており、この行為が、メモリページがハイブリッドバイナリ 108 から漸進的にロードされるたびに繰り返し適用され得ることを示している。それゆえ、実施形態において、ハイブリッドバイナリイメージからロードされるメモリへの変換は、メモリページフォルトを処理することに関連して、ハイブリッドバイナリイメージからロードされるメモリページに適用される。

10

【0093】

【102】 実施形態において、フィックスアップテーブル 108 c から得られた複数の変換は、マシンタイプフィールドを、互換性 ABI に対応する第二のマシンタイプを含むように調整する変換を含む。ある例において、ロダ 106 c は、システムメモリ 104 内のマシン識別子 303 に対応するメモリ位置に適用し、これはメモリ位置を、フォーリン ABI 107 c に対応するフォーリン ISA とマッチする値を記憶するように調整する。

【0094】

20

【103】 実施形態において、フィックスアップテーブル 108 c から得られた複数の変換は、ネイティブ ABI に基づいて実行可能な、折り畳まれていない関数の第一のコンパイルバージョンを呼び出すコールサイトを、その代わりに既存の ABI に基づいて実行可能な、折り畳まれていない関数の第二のコンパイルバージョンを呼び出すように変更する変換を含む。ある例において、ロダ 106 c は、コールサイトに対応するメモリ位置を適用して、折り畳まれていない関数の第一のコンパイルバージョンを参照する第一のメモリアドレスを、折り畳まれていない関数の第二のコンパイルバージョンを参照する第二のメモリアドレスに置き換える。コールサイト自体を変換できるものの、実施形態において、コールサイトは、折り畳まれていない関数の第一のコンパイルバージョンを、折り畳まれていない関数の第一のコンパイルバージョンを参照するディスパッチテーブル 306 を介して呼び出すように構成される。これらの実施形態において、コールサイト自体を変更するのではなく、この変換はディスパッチテーブル 306 を、折り畳まれていない関数の第一のコンパイルバージョンの代わりに折り畳まれていない関数の第二のコンパイルバージョンを参照するように変更する。

30

【0095】

【104】 実施形態において、ハイブリッドバイナリイメージは、折り畳まれていない関数の第一のコンパイルバージョンを参照するエントリポイントを含む。実施形態において、フィックスアップテーブル 108 c から得られ（及び行為 704 において適用され）た複数の変換は、折り畳まれていない関数の第二のコンパイルバージョンを参照するようにエントリポイントを調整する変換を含む。

40

【0096】

【105】 図 6 に関連して述べたように、実施形態において、方法 600 は、例えばインポート及び／又はエクスポートテーブルに対応する 1 つ又は複数のテーブルを、ネイティブビューを提供するこれらのテーブルへの参照と共に送信する。それゆえ、方法 700 に関して、幾つかの実施形態において、ハイブリッドバイナリイメージは 1 つ又は複数のテーブルを含み、これは、（i）折り畳まれていない関数の第一のコンパイルバージョンを使って、少なくとも折り畳まれていない関数を参照する第一のゾーン、（ii）少なくとも折り畳まれた関数を参照する第二のゾーン、及び（iii）折り畳まれていない関数の第二のコンパイルバージョンを使って、少なくとも折り畳まれていない関数を参照する第三のゾーンを含む。これらの実施形態において、ハイブリッドバイナリイメージはまた、

50

第一のゾーンと第二のゾーンを含むが第三のゾーンを含まない1つ又は複数のテーブルのネイティブビューを提供する1つ又は複数のテーブルへの参照も含む。バイナリイメージがEC ABI 107bに基づいてロードされるときに互換性ビューを提供するために、方法700は、1つ又は複数のテーブルへの参照を、第二のゾーンと第三のゾーンを含むが第一のゾーンを含まない1つ又は複数のテーブルの互換性ビューを提供するように調整する変換を適用することを含むことができる。

【0097】

[106] 一貫した参照アドレスを有するデュアルアーキテクチャ関数ポインタ

【0098】

[107] 図8は、(i) ネイティブISAに対応し、第一のCCを有する、ネイティブABIをターゲットとするネイティブコードと、(ii) フォーリンISAに対応し、第二のCCを有する、フォーリンABIをターゲットとするフォーリンコードの両方の実行をサポートするプロセス内でコールを処理する際に共通の参照メモリアドレスを使用する例示的な方法800のフローチャートを示す。

10

【0099】

[108] 図のように、方法800は、参照アドレスを使って呼び出し先関数へのコールを識別する行為801を含む。1つの例において、ディスパッチャ112aは、ECネイティブコード108bのネイティブ関数から参照メモリアドレスへのコールを識別する。他の例では、ディスパッチャ112bは、フォーリンバイナリ109の（及びエミュレータ106bによりエミュレートされている）フォーリン関数から参照メモリアドレスへのコールを識別する。

20

【0100】

[109] 方法800はまた、ルックアップ構造と参照アドレスを使用して、呼び出し先関数がネイティブABIに対応するか、フォーリンABIに対応するかを特定する行為802も含む。実施形態において、動作802は、少なくとも呼び出し先関数の参照メモリアドレスをターゲットとするコールを識別したことに基づいて、呼び出し先関数がネイティブABIに対応するか、フォーリンABIに対応するかを特定することを含む。実施形態において、呼び出し先関数は、ルックアップ構造が参照メモリアドレスはネイティブコードを保存する第一のメモリレンジ内に含まれることを示すことに基づいて、ネイティブABIに対応すると特定され、呼び出し先関数は、少なくともルックアップ構造が参照メモリアドレスはネイティブコードを保存しない第二のメモリレンジ内に含まれることを示していることに基づいて、フォーリンABIに対応すると特定される。ある例において、ディスパッチャ112（呼び出し元がネイティブであるときにはディスパッチャ112a、呼び出し元がフォーリンであるときにはディスパッチャ112bであり得る）は、ECルックアップ構造307に照会して、呼び出し先関数の参照メモリアドレスがECネイティブコード108bに対応するメモリ領域内にあること（そのこの場合、呼び出し先はEC ABI 107bに対応すると特定される）、又は呼び出し先関数の参照メモリアドレスがECネイティブコード108bに対応するメモリ領域内でないこと（この場合、呼び出し先はフォーリンABI 107cに対応すると特定される）を特定する。

30

【0101】

[110] 前述のように、実施形態において、ECルックアップ構造307はビットマップであり、これはメモリの対応するレンジ（例えば、メモリページ）がECネイティブコード108bを含むか否かを示すための1ビット値を使用する。しかしながら、ECルックアップ構造307は、ハッシュテーブル又はバイナリツリー等、代替的な種類のデータ構造を含むことができる。それゆえ、方法800において、ルックアップ構造は、ビットマップ、ハッシュテーブル、又はバイナリツリーのうちの少なくとも1つを含む。

40

【0102】

[111] 前述のように、呼び出し先関数がネイティブABIに対応すると特定される代替的なシナリオがあり得、すなわち、ディスパッチャ112bは呼び出し先関数が早送りシーケンスのシグナチャを含むことを特定する。それゆえ、行為802の幾つかの実施形

50

態において、呼び出し先関数は、(i) ルックアップ構造が、参照メモリアドレスはネイティブコードを記憶する第一のメモリレンジ内に含まれることを示していること、又は(ii) 早送りシーケンスが参照メモリアドレスで識別されたことの一方向に基づいて、ネイティブABIに対応すると特定される。それに加えて、行為802の幾つかの実施形態において、ルックアップ構造が参照メモリアドレスはネイティブコードを記憶しない第二のメモリレンジ内に含まれることを示しているとき、行為802は、早送りシーケンスが参照メモリアドレスにおいて識別可能であるか否かを特定することを含む。これらの実施形態において、ディスパッチャ112bは、早送りシーケンスから新たな参照メモリアドレス(すなわち、ネイティブ関数の真のアドレス)を取得して、新たな参照メモリアドレスをネイティブコールのために使用する。それゆえ、実施形態において、早送りシーケンスが参照メモリアドレスにおいて識別された場合、方法800は参照メモリアドレスを早送りシーケンスから得られた新たな参照メモリアドレスに更新することを含む。

10

【0103】

[112] 方法800はまた、呼び出し先関数の実行を開始する行為803も含む。実施形態において、動作803は、少なくとも特定することに基づいて、呼び出し先関数の実行を開始することを含む。図のように、行為803は、呼び出し元がフォーリンであり、呼び出し先がフォーリンであるとき、エミュレータ内で参照アドレスを直接呼び出す行為803a、呼び出し元がフォーリンで、呼び出し先がネイティブであるとき、エントリサンの位置を特定し、呼び出す行為803b、呼び出し元がネイティブで呼び出し先がフォーリンであるとき、エグジットサンクを呼び出す行為803c、又は呼び出し元がネイティブで、呼び出し先がネイティブのとき、参照アドレスを直接呼び出す行為803dのうちの1つを実行することを含む。

20

【0104】

[113] 実施形態において、行為803aは、呼び出し元関数がフォーリンABIに対応し、呼び出し先関数がフォーリンABIに対応すると特定された場合、エミュレータ内の参照メモリアドレスを使って呼び出し先関数を直接呼び出すことを含む。ある例において、フォーリン関数からのコールに基づいて、ディスパッチャ112bは、呼び出し先もまたフォーリン関数であると特定する。それゆえ、呼び出し元は、呼び出し先関の真のメモリアドレスを呼び出しており、何れのサンクも不要であるため、ディスパッチャ112bはエミュレータ106bの中で参照メモリアドレスを使って呼び出し関数直接起動する。

30

【0105】

[114] 実施形態において、行為803bは、呼び出し元関数がフォーリンABIに対応し、呼び出し先関数がネイティブABIに対応すると特定された場合、(i) 第二のCCを第一のCCに適応させ、その後、(ii) 参照メモリアドレスを使って呼び出し先関数を直接呼び出すエントリサンクを呼び出すことを含む。ある例において、フォーリン関数からのコールに基づいて、ディスパッチャ112bは呼び出し先がネイティブ関数であることを特定する。それゆえ、ディスパッチャ112bは、参照メモリアドレス(すなわち、呼び出し先関数の)を直接呼び出すことができず、これは、それがエントリサンクを迂回するからである。その代わりに、ディスパッチャ112bは、エントリサンクへの新たな参照アドレスの位置を特定し、新しい参照メモリアドレスを使ってエントリサンクを起動する。エントリサンク自体は、フォーリンABI 107cのCCをEC ABI 107bのCCに適応させ、当初の参照メモリアドレスを使って呼び出し先関数を起動する。

40

【0106】

[115] 前述のように、実施形態において、ディスパッチャ112bは、呼び出し先関数の当初の参照メモリアドレスの直前のメモリのブロック内に含まれるデータ(例えば、オフセット又は直接アドレス参照)に基づいて、エントリサンクへの新たな参照アドレスの位置を特定する。それゆえ、実施形態において、方法800はまた、少なくとも(i) 参照メモリアドレスの直前のメモリブロックを読み出すこと、及び(ii) メモリブロックからエントリサンクの位置までのオフセット又はポインタを特定することに基づいて、

50

エントリサンの位置を識別することを含む。

【0107】

[116] 実施形態において、行為803cは、呼び出し元関数がネイティブABIに対応し、呼び出し先関数がフォーリンABIに対応すると特定された場合、(i) ネイティブABIの第一のCCをフォーリンABIの第二のCCに適応させ、その後、(ii) エミュレータに参照メモリアドレスを使って呼び出し先関数を直接呼び出させるエグジットサンクを呼び出すことを含む。ある例において、ネイティブ関数からのコールに基づいて、ディスパッチャ112aは、呼び出し先がフォーリン関数であると特定する。それゆえ、ディスパッチャ112aは、参照メモリアドレス（すなわち、呼び出し先関数の）を直接呼び出すことができず、これは、それがエグジットサンクを迂回するからである。その代わりに、ディスパッチャ112aはエグジットサンクまでの新たな参照アドレスの位置を特定し、新たな参照メモリアドレスを使ってエグジットサンクを起動する。エグジットサンク自体は、EC ABI 107bのCCをフォーリンABI 107cのCCに適応させ、エミュレータ106bを起動する。エミュレータ106b自体は、当初の参照メモリアドレスを使って呼び出し先関数を直接呼び出す。前述のように、実施形態において、エグジットサンクまでの新たな参照アドレスは呼び出し先関数自体に含まれる。それゆえ、方法800の幾つかの実施形態において、エグジットサンクの位置は呼び出し元関数の中に含まれる。

10

【0108】

[117] 実施形態において、行為803dは、呼び出し元関数がネイティブABIに対応し、呼び出し先関数がネイティブABIに対応すると特定された場合、参照メモリアドレスを使って呼び出し先関数を直接呼び出すことを含む。ある例において、ネイティブ関数からのコールに基づいて、ディスパッチャ112aは、呼び出し先もまたネイティブ関数であると特定する。それゆえ、呼び出し元は、呼び出し関数の真のメモリアドレスを呼び出しており、何れのサンクも不要であるため、ディスパッチャ112aは参照メモリアドレスを使って呼び出し先関数を直接起動する。

20

【0109】

[118] 主題は構造的特徴及び／又は方法論的行為について具体的な文言で説明されているが付属の特許請求項で定義される主題は、必ずしも説明された特徴若しくは前述の行為に、又は前述の行為の順序に限定されるわけではないと理解されたい。むしろ、説明された特徴と行為は、特許請求項を実施する例示的な形態として開示されている。

30

【0110】

[119] 本発明の実施形態は、以下により詳しく説明するように、コンピュータハードウェア、例えば1つ又は複数のプロセッサ（例えば、プロセッサ102）とシステムメモリ（例えば、システムメモリ104）を含む特定用途又は汎用コンピュータシステム（例えば、コンピュータシステム101）を含むか、利用し得る。本発明の範囲に含まれる実施形態はまた、コンピュータ実行可能命令及び／又はデータストラクチャを担持又は記憶する物理及びその他のコンピュータ可読媒体も含む。このようなコンピュータ可読媒体は、汎用又は特定用途コンピュータシステムによりアクセス可能な何れの入手可能な媒体とすることもできる。コンピュータ実行可能命令及び／又はデータ構造を記憶するコンピュータ可読媒体は、コンピュータ記憶媒体、（例えば、持続性ストレージ103、システムメモリ104）である。コンピュータ実行可能命令及び／又はデータ構造を担持するコンピュータ可読媒体は伝送媒体である。それゆえ、例えば、これに限定されないが、本発明の実施形態は少なくとも2つの異なる種類のコンピュータ可読媒体、すなわちコンピュータ記憶媒体と伝送媒体を含むことができる。

40

【0111】

[120] コンピュータ記憶媒体は、コンピュータ実行可能命令及び／又はデータ構造を記憶する物理記憶媒体である。物理記憶媒体としては、コンピュータハードウェア、例えばRAM、ROM、EEPROM、ソリッドステートドライブ（SSDs）、フラッシュメモリ、相変化メモリ（PCM）、光ディスクストレージ、磁気ディスクストレージ、若

50

しくはその他の磁気記憶デバイス、又は、汎用若しくは特定用途コンピュータにより、開示されている本発明の機能を実施するためにアクセス及び実行可能なコンピュータ実行可能命令又はデータ構造の形態でプログラムコードを記憶するために使用できる他のあらゆるハードウェア記憶デバイスが含まれる。

【0112】

【121】 伝送媒体は、プログラムコードをコンピュータ実行可能命令又はデータ構造の形態で担持するために使用でき、汎用又は特定用途コンピュータシステムによりアクセス可能なネットワーク及び／又はデータリンクを含むことができる。「ネットワーク」は、コンピュータシステム及び／又はモジュール及び／又はその他の電子機器間の電子データの輸送を可能にする1つ又は複数のデータリンクとして定義される。情報がネットワーク又はその他の通信接続（ハードワイヤード、ワイヤレス、又はハードワイヤード又はワイヤレスの組合せ）上でコンピュータシステムに伝送又は提供されるとき、コンピュータシステムはこの接続で送信媒体とみなし得る。上記の組合せもまた、コンピュータ可読媒体の範囲に含められるべきである。

10

【0113】

【122】 さらに、各種のコンピュータシステムコンポーネントに到達すると、コンピュータ実行可能命令又はデータ構造の形態のプログラムコードは、送信媒体からコンピュータ記憶媒体へと（又はその逆へと）自動的に伝送され得る。例えば、ネットワーク又はデータリンク上で受信されたコンピュータ実行可能命令又はデータ構造は、ネットワークインタフェースモジュール（例えば、“NIC”）内でRAMに格納され、その後、最終的にコンピュータシステムRAMに、及び／又はコンピュータシステムにおけるより揮発性の低いコンピュータ記憶媒体に伝送できる。それゆえ、コンピュータ記憶媒体は、伝送媒体も（又は、さらには主として）利用するコンピュータシステムコンポーネントにも含めることができると理解すべきである。

20

【0114】

【123】 コンピュータ実行可能命令は、例えば、1つ又は複数のプロセッサにおいて実行されると、汎用コンピュータシステム、特定用途コンピュータシステム、又は特定用途処理装置に、特定の機能又は機能群を実行させる命令及びデータを含む。コンピュータ実行可能命令は例えば、バイナリ、アセンブリ言語等の中間フォーマット命令、又はさらにはソースコードであり得る。

30

【0115】

【124】 当業者であればわかるように、本発明は多くの種類のコンピュータシステム構成のネットワークコンピューティング環境で実践でき、これにはパーソナルコンピュータ、デスクトップコンピュータ、ラップトップコンピュータ、メッセージプロセッサ、ハンドヘルドデバイス、マルチプロセッサシステム、マイクロプロセッサベースの、若しくはプログラム可能な民生用電子機器、ネットワークPCs、マイクロコンピュータ、メインフレームコンピュータ、携帯電話、PDAs、タブレット、ページャ、ルータ、スイッチその他が含まれる。本発明はまた、分散型システム環境でも実践され得て、その場合、ローカル及びリモートコンピュータシステムがネットワークを通じて（ハードワイヤードデータリンク、無線データリンク、又はハードワイヤード及びワイヤレスデータリンクの組合せの何れかにより）連結され、どちらもタスクを実行する。そのため、分散型システム環境では、コンピュータシステムは複数のコンピュータシステムを構成要素として含み得る。分散型システム環境では、プログラムモジュールはローカル及びリモート両方のメモリ記憶媒体の中に配置され得る。

40

【0116】

【125】 同じく当業者であればわかるように、本発明はクラウドコンピューティング環境でも実践され得る。クラウドコンピューティング環境は分散型であり得るが、そうでなくてもよい。分散型である場合、クラウドコンピューティング環境は、ある組織の中で内部的に分散されても、及び／又は複数の組織にまたがって所有されるコンポーネントを有していてもよい。本明細書及び以下の特許請求の範囲の中で、「クラウドコンピューティン

50

グ」とは、設定可能なコンピューティングリソース（例えば、ネットワーク、サーバ、ストレージ、アプリケーション、及びサービス）の共有プールにオンデマンドでネットワークアクセスできるモデルと定義される。「クラウドコンピューティング」の定義は、適切に展開されたときにこのようなモデルから取得できる他の様々な利点の何れにも限定されない。

【0117】

【126】 クラウドコンピューティングモデルは、オンデマンドセルフサービス（on-demand self-service）、幅広いネットワークアクセス（broad network access）、リソースの共用（resource pooling）、スピーディな拡張性（rapid elasticity）、計測可能なサービス（measured service）その他、様々な特徴で構成できる。クラウドコンピューティングモデルは、例えば SaaS（Software as a Service）、PaaS（Platform as a Service）、IaaS（Infrastructure as a Service）をはじめとする各種のサービスモデルの形態をとり得る。クラウドコンピューティングモデルはまた、プライベートクラウド、コミュニティクラウド、パブリッククラウド、ハイブリッドクラウド等、異なる展開モデルを使っても展開され得る。

10

【0118】

【127】 クラウドコンピューティング環境等、幾つかの実施形態は、各々が1つ又は複数の仮想マシンを動かすことのできる1つ又は複数のホストを含むシステムを含み得る。動作中、仮想マシンは、あるOSと、おそらく1つ又は複数のその他のアプリケーションもサポートする動作可能なコンピューティングシステムをエミュレートする。幾つかの実施形態において、各ホストは、仮想マシンのビューから抽出される物理リソースを使って仮想マシンのための仮想リソースをエミュレートするハイパバイザを含む。ハイパバイザはまた、仮想マシン間を適正に分離する。それゆえ、何れかの仮想マシンから見ると、ハイパバイザは仮想マシンが物理リソースと相互作用しているような錯覚がもたらされるが、その仮想マシンは物理リソースの見た目（例えば、仮想リソース）と相互作用しているだけである。物理リソースの例としては、処理能力、メモリ、ディスクスペース、ネットワーク帯域幅、メディアドライブ等が含まれる。

20

【0119】

【128】 本発明は、その主旨又は本質的特徴から逸脱することなく、他の特定の形態で実施できる。説明された実施形態は、あらゆる点において制約ではなく例示と考えるものとする。したがって、本発明の範囲は、上記の説明ではなく、付属の特許請求項により示される。特許請求項の等価性の意味と範囲に含まれる全ての変更は、その範囲内に含まれるものとする。付属の特許請求の範囲の中で要素を紹介する際、冠詞（a, an, the, said）は、その要素が1つ又は複数あることを意味することが意図される。「～を含む（comprising, including）」、「～を有する（having）」という用語は、包含的であることが意図され、挙げられた要素以外の他の要素もあり得ることを意味する。「セット」、「サブセット」という用語は、空集合を排除することが意図され、それゆえ、「セット」とは空ではない集合と定義され、「サブセット」とは空ではない小集合と定義される。

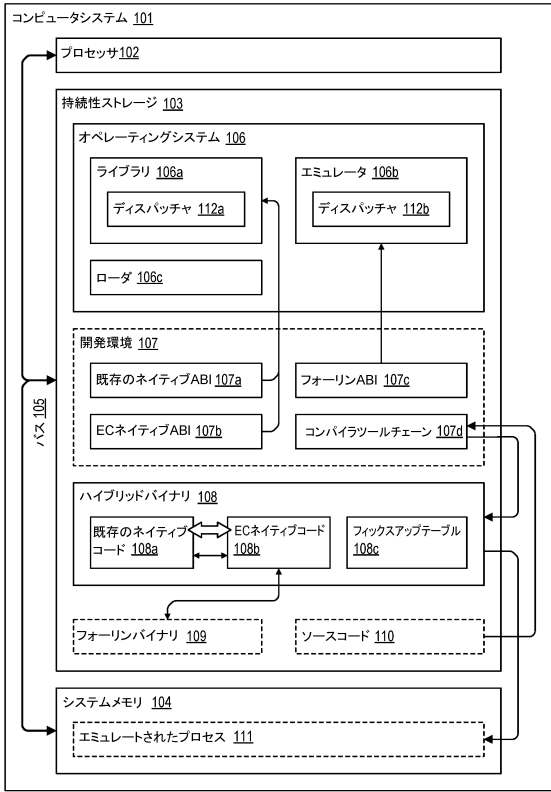
30

40

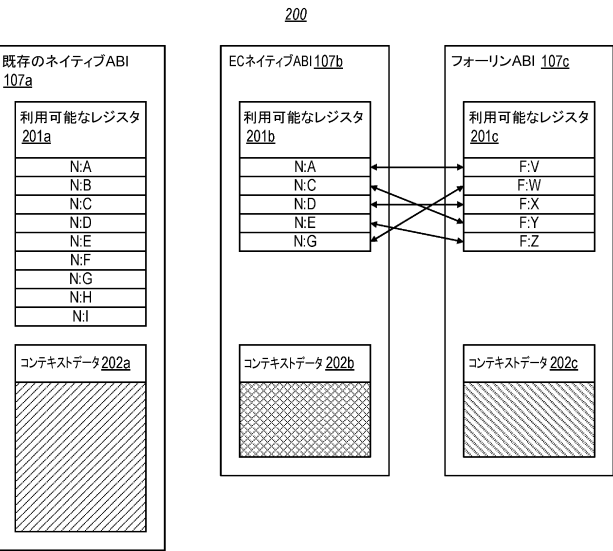
50

【図面】

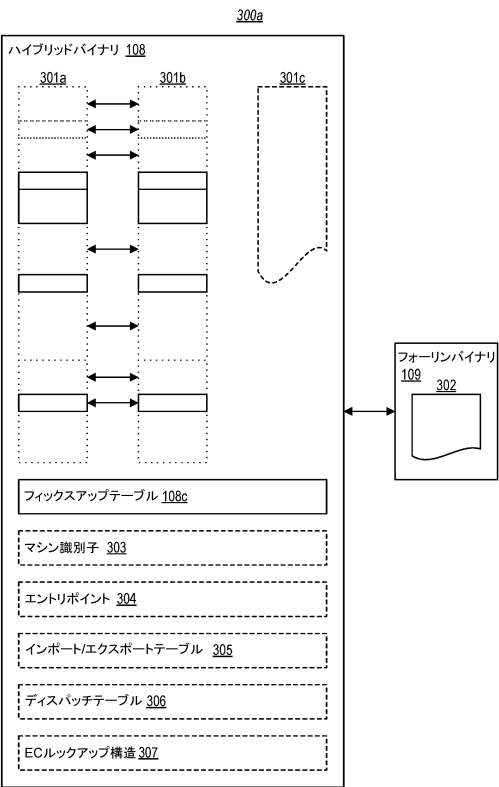
【図 1】



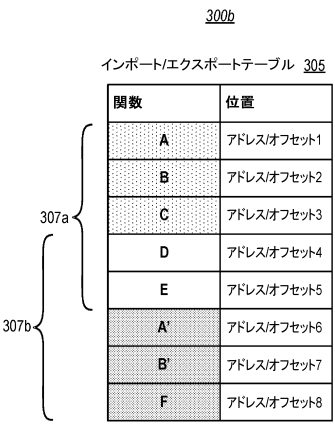
【図 2】



【図 3 A】



【図 3 B】



10

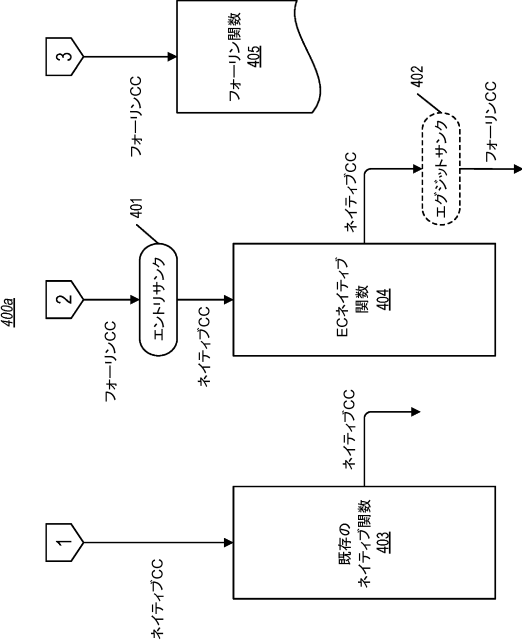
20

30

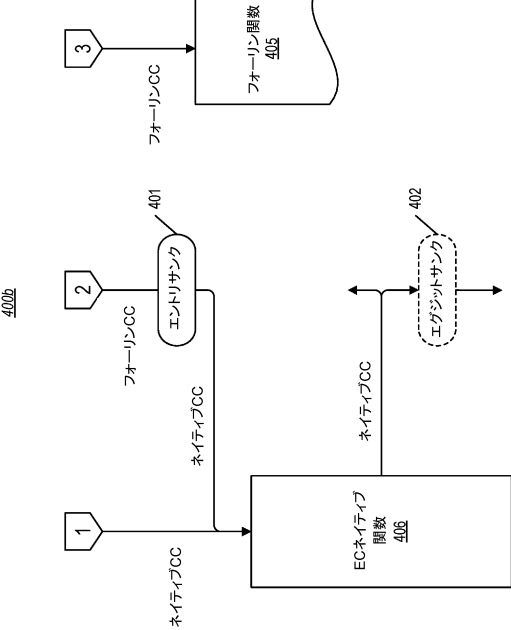
40

50

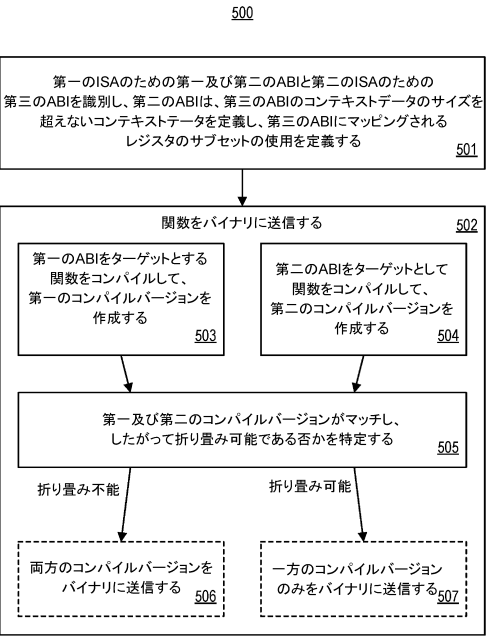
【図 4 A】



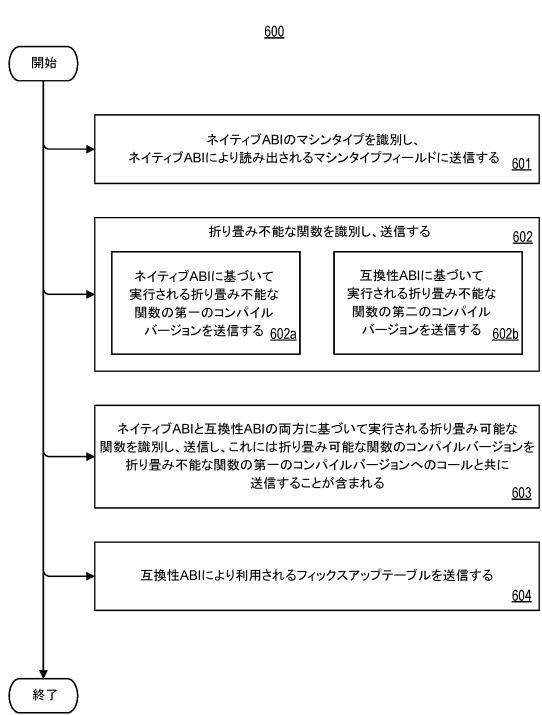
【図 4 B】



【図 5】



【図 6】



10

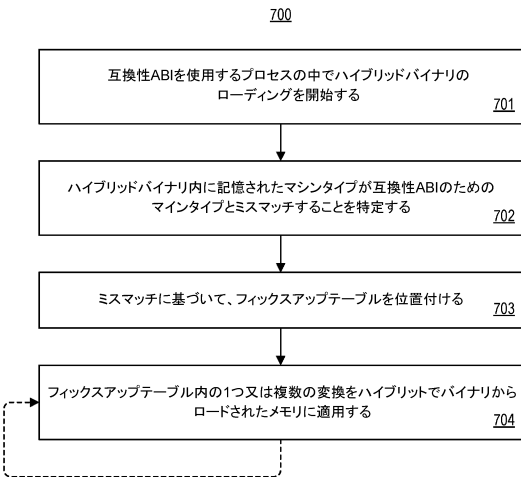
20

30

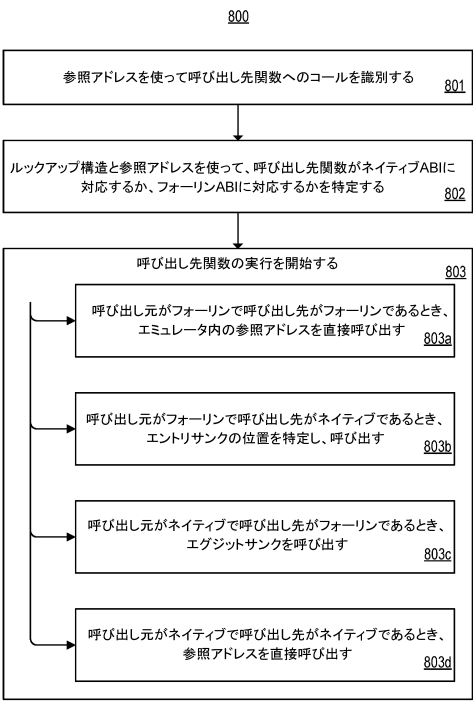
40

50

【 図 7 】



【 図 8 】



10

20

30

40

50

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No

PCT/US2021/030388

A. CLASSIFICATION OF SUBJECT MATTER

INV. G06F9/445

ADD. G06F8/41

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal, COMPENDEX, INSPEC, WPI Data, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2019/265993 A1 (TZEN TEN H [US] ET AL) 29 August 2019 (2019-08-29) paragraph [0117] - paragraph [0131]; claim 1; figures 3B, 3C, 3D, 4A, 4B, 5A, 5B paragraph [0033] - paragraph [0034] paragraph [0060]	1-9
A	----- Anonymous: "Optimize Options (Using the GNU Compiler Collection (GCC))", 27 July 2020 (2020-07-27), XP055819824, Retrieved from the Internet: URL:https://web.archive.org/web/2020072705 4919/https://gcc.gnu.org/onlinedocs/gcc/Op timize-Options.html [retrieved on 2021-06-30] the whole document -----	1-9



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

10 August 2021

Date of mailing of the international search report

19/08/2021

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040,
Fax: (+31-70) 340-3016

Authorized officer

Filimon, Diana

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2021/030388

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2019265993 A1	29-08-2019	CN 108139921 A	08-06-2018
		EP 3356933 A1	08-08-2018
		EP 3637255 A1	15-04-2020
		US 2017097835 A1	06-04-2017
		US 2019265993 A1	29-08-2019
		WO 2017058704 A1	06-04-2017

フロントページの続き

MK,MT,NL,NO,PL,PT,RO,RS,SE,SI,SK,SM,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,KM,ML,MR,N
E,SN,TD,TG),AE,AG,AL,AM,AO,AT,AU,AZ,BA,BB,BG,BH,BN,BR,BW,BY,BZ,CA,CH,CL,CN,CO,CR,CU,
CZ,DE,DJ,DK,DM,DO,DZ,EC,EE,EG,ES,FI,GB,GD,GE,GH,GM,GT,HN,HR,HU,ID,IL,IN,IR,IS,IT,JO,JP,K
E,KG,KH,KN,KP,KR,KW,KZ,LA,LC,LK,LR,LS,LU,LY,MA,MD,ME,MG,MK,MN,MW,MX,MY,MZ,NA,N
G,NI,NO,NZ,OM,PA,PE,PG,PH,PL,PT,QA,RO,RS,RU,RW,SA,SC,SD,SE,SG,SK,SL,ST,SV,SY,TH,TJ,TM,
TN,TR,TT,TZ,UA,UG,US,UZ,VC,VN,WS,ZA,ZM,ZW

(特許庁注：以下のものは登録商標)

1. L i n u x

弁理士 阿部 豊隆

- (72)発明者 セケイラ デ ジュスト テイシェイラ, ペドロ ミゲル
アメリカ合衆国, ワシントン州 98052-6399, レッドモンド, ワン マイクロソフト ウ
エイ, 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
- (72)発明者 ミホカ, ダレク ヨシッブ
アメリカ合衆国, ワシントン州 98052-6399, レッドモンド, ワン マイクロソフト ウ
エイ, 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
- (72)発明者 ベリー, ジョン ロバート
アメリカ合衆国, ワシントン州 98052-6399, レッドモンド, ワン マイクロソフト ウ
エイ, 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
- (72)発明者 ハドリー, ラッセル チャールズ
アメリカ合衆国, ワシントン州 98052-6399, レッドモンド, ワン マイクロソフト ウ
エイ, 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
- (72)発明者 クリアリー, ジェームズ デイビッド
アメリカ合衆国, ワシントン州 98052-6399, レッドモンド, ワン マイクロソフト ウ
エイ, 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
- (72)発明者 ダン, クラレンス シウ イェーン
アメリカ合衆国, ワシントン州 98052-6399, レッドモンド, ワン マイクロソフト ウ
エイ, 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

F ターム (参考) 5B081 CC00
5B376 DA26