(12) 特許協力条約に基づいて公開された国際出願

(19) 世界知的所有権機関
国際事務局

(43) 国際公開日
2021年8月12日(12.08.2021)

WIPO | PCT

(10) 国際公開番号
WO 2021/156955 A1

(51) 国際特許分類:
G06F 8/41 (2018.01)

(21) 国際出願番号： PCT/JP2020/004202

(22) 国際出願日： 2020年2月4日(04.02.2020)

(25) 国際出願の言語： 日本語

(26) 国際公開の言語： 日本語

(71) 出願人: 日本電信電話株式会社 (**NIPPON TELEGRAPH AND TELEPHONE CORPORATION**) [JP/JP]; 〒1008116 東京都千代田区大手町一丁目５番１号 Tokyo (JP).

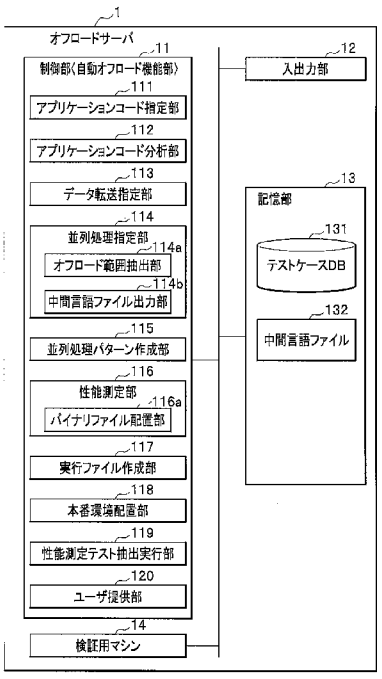
(72) 発明者: 山登 庸次 (**YAMATO, Yoji**); 〒1808585 東京都武蔵野市緑町３丁目９－１１ ＮＴＴ知的財産センタ内 Tokyo (JP).

(74) 代理人: 特許業務法人磯野国際特許商標事務所 (**ISONO INTERNATIONAL PATENT OFFICE, P.C.**); 〒1050001 東京都港区虎ノ門一丁目１番１８号 ヒューリック虎ノ門ビル Tokyo (JP).

(81) 指定国(表示のない限り、全ての種類の国内保護が可能): 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, JO, JP, KE, KG, KH,

(54) **Title:** OFFLOAD SERVER, OFFLOAD CONTROL METHOD, AND OFFLOAD PROGRAM

(54) 発明の名称： オフロードサーバ、オフロード制御方法およびオフロードプログラム



1 Offload server
11 Control unit (automatic offload function unit)
12 Input/output unit
13 Storage unit
14 Verification machine
111 Application code specification unit
112 Application code analysis unit
113 Data transfer specification unit
114 Parallel processing specification unit
114a Offload range extraction unit
114b Intermediate language file output unit
115 Parallel processing pattern creation unit
116 Performance measurement unit
116a Binary file allocation unit
117 Executable file creation unit
118 Actual environment allocation unit
119 Performance measurement test extraction execution unit
120 User provision unit
131 Test case database
132 Intermediate language files

(57) **Abstract:** An offload server (1) is provided with: an application code analysis unit (112) which analyzes the source code of an application; a data transfer specification unit (113) which, on the basis of the code analysis results, specifies that data for variables be collectively transferred before the start and after the end of a GPU process if the variables need to be transferred between a CPU and a GPU but are not mutually referenced or updated by CPU and GPU processes and the results of the GPU process are only returned to the CPU; and a parallel processing specification unit (114) which identifies

WO 2021/156955 A1

[続葉有]

KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, 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.

(84) 指定国(表示のない限り、全ての種類の広域保護が可能): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), ユーラシア (AM, AZ, BY, KG, KZ, RU, TJ, TM), ヨーロッパ (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

添付公開書類:

一 国際調査報告(条約第21条(3))

loop statements for the application, and specifies and compiles a GPU parallel processing specification statement for each identified loop statement.

(57) 要約:オフロードサーバ(1)は、アプリケーションのソースコードを分析するアプリケーションコード分析部(112)と、コード分析の結果をもとに、CPUとGPU間の転送が必要な変数の中で、CPU処理とGPU処理とが相互に参照または更新がされず、GPU処理した結果をCPUに返すだけの変数については、GPU処理の開始前と終了後に一括化してデータ転送する指定を行うデータ転送指定部(113)と、アプリケーションのループ文を特定し、特定した各ループ文に対して、GPUにおける並列処理指定文を指定してコンパイルする並列処理指定部(114)と、を備える。

明　細　書

発明の名称 ：

オフロードサーバ、オフロード制御方法およびオフロードプログラム

技術分野

[0001] 本発明は、機能処理をＧＰＵ（Graphics Processing Unit）等に自動オフロードするオフロードサーバ、オフロード制御方法およびオフロードプログラムに関する。

背景技術

[0002] ＣＰＵ（Central Processing Unit）以外のヘテロな計算リソースを用いることが増えている。例えば、ＧＰＵ（アクセラレータ)を強化したサーバで画像処理を行ったり、ＦＰＧＡ（アクセラレータ)で信号処理をアクセラレートすることが始まっている。ＦＰＧＡは、製造後に設計者等が構成を設定できるプログラム可能なゲートアレイであり、ＰＬＤ（Programmable Logic Device）の一種である。Amazon Web Services（AWS）（登録商標）では、ＧＰＵインスタンス、ＦＰＧＡインスタンスが提供されており、オンデマンドにそれらリソースを使うこともできる。Microsoft（登録商標）は、ＦＰＧＡを用いて検索を効率化している。

[0003] ＯｐｅｎＩｏＴ（Internet of Things）環境では、サービス連携技術等を用いて、多彩なアプリケーションの創出が期待されるが、更に進歩したハードウェアを生かすことで、動作アプリケーションの高性能化が期待できる。しかし、そのためには、動作させるハードウェアに合わせたプログラミングや設定が必要である。例えば、ＣＵＤＡ（Compute Unified Device Architecture）、 OpenＣＬ（Open Computing Language）といった多くの技術知識が求められ、ハードルは高い。OpenＣＬは、あらゆる計算資源（ＣＰＵやＧＰＵに限らない）を特定のハードに縛られず統一的に扱えるオープンなＡＰＩ（Application Programming Interface）である。

[0004] ＧＰＵやＦＰＧＡをユーザのＩｏＴアプリケーションで容易に利用できる

ようにするため下記が求められる。すなわち、動作させる画像処理、暗号処理等の汎用アプリケーションをOpenＩｏＴ環境にデプロイする際に、OpenＩｏＴのプラットフォームがアプリケーションロジックを分析し、ＧＰＵ、ＦＰＧＡに自動で処理をオフロードすることが望まれる。

[0005] ＧＰＵの計算能力を画像処理以外にも使うＧＰＧＰＵ（General Purpose GPU）のための開発環境ＣＵＤＡが発展している。ＣＵＤＡは、ＧＰＧＰＵ向けの開発環境である。また、ＧＰＵ、ＦＰＧＡ、メニーコアＣＰＵ等のヘテロハードウェアを統一的に扱うための標準規格としてOpenＣＬも登場している。

[0006] ＣＵＤＡやOpenＣＬでは、Ｃ言語の拡張によるプログラミングを行う。ただし、ＧＰＵ等のデバイスとＣＰＵの間のメモリコピー、解放等を記述する必要があり、記述の難度は高い。実際に、ＣＵＤＡやOpenＣＬを使いこなせる技術者は数多くはいない。

[0007] 簡易にＧＰＧＰＵを行うため、ディレクティブベースで、ループ文等の並列処理すべき個所を指定し、ディレクティブに従いコンパイラがデバイス向けコードに変換する技術がある。技術仕様としてOpenACC（Open Accelerator）等、コンパイラとしてＰＧＩコンパイラ（登録商標）等がある。例えば、OpenACCを使った例では、ユーザはC/C++/Fortran言語で書かれたコードに、OpenACCディレクティブで並列処理させる等を指定する。ＰＧＩコンパイラは、コードの並列可能性をチェックして、ＧＰＵ用、ＣＰＵ用実行バイナリを生成し、実行モジュール化する。IBM JDK（登録商標）は、Java（登録商標）のlambda形式に従った並列処理指定を、ＧＰＵにオフロードする機能をサポートしている。これらの技術を用いることで、ＧＰＵメモリへのデータ割り当て等を、プログラマは意識する必要がない。

このように、OpenＣＬ、ＣＵＤＡ、OpenACC等の技術により、ＧＰＵやＦＰＧＡへのオフロード処理が可能になっている。

[0008] しかし、オフロード処理自体は行えるようになっても、適切なオフロードには課題が多い。例えば、Intelコンパイラ（登録商標）のように自動並列化

機能を持つコンパイラがある。自動並列化する際は、プログラム上のfor文（繰り返し文）等の並列処理部を抽出する。ところが、ＧＰＵを用いて並列に動作させる場合は、ＣＰＵ-ＧＰＵメモリ間のデータやり取りオーバヘッドのため、性能が出ないことも多い。ＧＰＵを用いて高速化する際は、スキル保持者が、OpenＣＬやＣＵＤＡでのチューニングや、ＰＧＩコンパイラ等で適切な並列処理部を探索することが必要になっている。

このため、スキルが無いユーザがＧＰＵを使ってアプリケーションを高性能化することは難しいし、自動並列化技術を使う場合も、for文を並列するかしないかの試行錯誤チューニング等、利用開始までに多くの時間がかかっている。

[0009] 並列処理箇所の試行錯誤を自動化する取り組みとして、非特許文献１，２が挙げられる。非特許文献１，２は、ＧＰＵオフロードに適したループ文を、進化的計算手法を用いて検証環境での性能測定を繰り返すことで、適切に抽出し、ネストループ文内の変数をできるだけ上位のループでＣＰＵ-ＧＰＵ転送を一括化することで自動での高速化を行っている。

先行技術文献

非特許文献

[0010] 非特許文献1：Y. Yamato, T. Demizu, H. Noguchi and M. Kataoka, “Automatic GPU Offloading Technology for Open IoT Environment, ”IEEE Internet of Things Journal, Sep. 2018.

非特許文献2：Y. Yamato, “Study of parallel processing area extraction and data transfer number reduction for automatic GPU offloading of IoT applications,” Journal of Intelligent Information Systems, Springer, DOI: 10.1007/s10844-019-00575-8, Aug. 2019.

発明の概要

発明が解決しようとする課題

[0011] 非特許文献１，２では、ＣＰＵ向けの汎用的コードから、ＧＰＵオフロー

ドに向けて、適切な並列処理領域を自動抽出し、並列処理可能なループ文群に対してＧＡ（Genetic Algorithm：遺伝的アルゴリズム）を用いて、より適切な並列処理領域を探索することで、ＧＰＵへの自動オフロードを実現している。しかし、OpenACCを用いた自動高速化は、ＣＵＤＡを用いた手動高速化に比べて、性能改善が十分でないアプリケーションが多いことが言える。特許文献１，２の技術は、OpenACCを用いた自動高速化を前提としており、ＣＵＤＡを使った手動高速化に比べて性能改善が物足りないことが課題として挙げられる。

[0012] このような点に鑑みて本発明がなされたのであり、ＣＰＵ-ＧＰＵ間の転送を削減して、オフロードのさらなる高速化を図ることを課題とする。

課題を解決するための手段

[0013] 前記した課題を解決するため、アプリケーションの特定処理をＧＰＵにオフロードするオフロードサーバであって、アプリケーションのソースコードを分析するアプリケーションコード分析部と、コード分析の結果をもとに、ＣＰＵと前記ＧＰＵ間の転送が必要な変数の中で、ＣＰＵ処理とＧＰＵ処理とが相互に参照または更新がされず、前記ＧＰＵ処理した結果を前記ＣＰＵに返すだけの変数については、前記ＧＰＵ処理の開始前と終了後に一括化してデータ転送する指定を行うデータ転送指定部と、前記アプリケーションのループ文を特定し、特定した各前記ループ文に対して、前記ＧＰＵにおける並列処理指定文を指定してコンパイルする並列処理指定部と、コンパイルエラーが出るループ文に対して、オフロード対象外とするとともに、コンパイルエラーが出ないループ文に対して、並列処理するかしないかの指定を行う並列処理パターンを作成する並列処理パターン作成部と、前記並列処理パターンの前記アプリケーションをコンパイルして、アクセラレータ検証用装置に配置し、前記ＧＰＵにオフロードした際の性能測定用処理を実行する性能測定部と、性能測定結果をもとに、複数の前記並列処理パターンから最高処理性能の並列処理パターンを選択し、最高処理性能の前記並列処理パターンをコンパイルして実行ファイルを作成する実行ファイル作成部と、を備える

オフロードサーバとした。

発明の効果

[0014] 本発明によれば、ＣＰＵ－ＧＰＵ間の転送を削減して、オフロードのさらなる高速化を図ることができる。

図面の簡単な説明

[0015] [図1]<data copyやpresentを使用しないケース>と、<data copyやpresentを使用するケース>と、<data copyやpresentを使用し、かつ一時領域をデータの格納場所として使用するケース>と、の各々において、ＣＰＵ-ＧＰＵ間のパラメータおよびデータ転送を説明する図である。

[図2]本発明の実施形態に係るオフロードサーバを含む環境適応ソフトウェアシステムを示す図である。

[図3]上記実施形態に係るオフロードサーバの構成例を示す機能ブロック図である。

[図4]上記実施形態に係るオフロードサーバのＧＡを用いた自動オフロード処理を示す図である。

[図5]上記実施形態に係るオフロードサーバのSimple GAによる制御部（自動オフロード機能部）の探索イメージを示す図である。

[図6]比較例の通常ＣＰＵプログラムの例を示す図である。

[図7]比較例の単純ＣＰＵプログラムを利用してＣＰＵからＧＰＵへデータ転送する場合のループ文の例を示す図である。

[図8]上記実施形態に係るオフロードサーバのネスト一体化した場合のＣＰＵからＧＰＵへデータ転送する場合のループ文の例を示す図である。

[図9]上記実施形態に係るオフロードサーバの転送一体化した場合のＣＰＵからＧＰＵへデータ転送する場合のループ文の例を示す図である。

[図10]上記実施形態に係るオフロードサーバの転送一体化し、かつ一時領域を利用した場合のＣＰＵからＧＰＵへデータ転送する場合のループ文の例を示す図である。

[図11A]上記実施形態に係るオフロードサーバの実装の動作概要を説明するフ

ローチャートである。

[図11B]上記実施形態に係るオフロードサーバの実装の動作概要を説明するフローチャートである。

[図12]本発明の実施形態に係るオフロードサーバの機能を実現するコンピュータの一例を示すハードウェア構成図である。

発明を実施するための形態

[0016] 以下、図面を参照して本発明を実施するための形態（以下、「本実施形態」という）におけるオフロードサーバについて説明する。

（背景説明）

オフロードしたいアプリケーションは、多様である。また、映像処理のための画像分析、センサデータを分析するための機械学習処理等、計算量、時間が多いアプリケーションでは、ループ文による繰り返し処理が長時間を占めている。そこで、ループ文をＧＰＵに自動でオフロードすることで、高速化することがターゲットとして考えられる。

[0017] まず、ループ文をＧＰＵに自動でオフロードする場合の基本的な課題として、下記がある。すなわち、コンパイラが「このループ文はＧＰＵで並列処理できない」という制限を見つけることは可能であっても、「このループ文はＧＰＵの並列処理に適している」という適合性を見つけることは難しいのが現状である。また、ループ文をＧＰＵに自動でオフロードする場合、一般的にループ回数が多い等の計算密度が高いループの方が適しているとされている。しかし、実際にどの程度の性能改善になるかは、実測してみないと予測は困難である。そのため、ループ文をＧＰＵにオフロードするという指示を手動で行い、性能測定を試行錯誤することが行われている。

[0018] 非特許文献１は、上記を踏まえ、ＧＰＵにオフロードする適切なループ文を発見することを、ＧＡ（Genetic Algorithm：遺伝的アルゴリズム）で自動的に行うことを提案している。すなわち、非特許文献１では、並列化を想定していない汎用プログラムから、最初に並列可能ループ文のチェックを行い、次に並列可能ループ文群に対して、ＧＰＵ実行の際を１、ＣＰＵ実行の際

を０と値を置いて遺伝子化し、検証環境で性能検証試行を反復し適切な領域を探索している。並列可能ループ文に絞った上で、遺伝子の部分の形で、高速化可能な並列処理パターンを保持し組み換えていくことで、取り得る膨大な並列処理パターンから、効率的に高速化可能なパターンを探索している。

[0019] 非特許文献１では、ネストループ文の中で利用される変数について、ループ文をＧＰＵにオフロードする際に、ＣＰＵ-ＧＰＵ間の転送がされている。しかし、ネストの下位でＣＰＵ-ＧＰＵ転送が行われると下位のループの度に転送が行われ効率的でない。

非特許文献２は、上位でＣＰＵ-ＧＰＵ転送が行われても問題ない変数については、上位でまとめて転送を行うことを提案している。処理時間がかかるループ回数の多いループは、ネストであることが多いため、転送回数削減による高速化に一定の効果を示す手法である。

[0020] 非特許文献１および非特許文献２では、実際に、ループ文が１００を超える中規模なアプリケーションでも自動高速化を確認している。実用性を意識した際には、より高速化するが求められている。

[0021] （基本的な考え方）

本発明の基本的な考え方について説明する。

OpenACCを用いた自動高速化は、ＣＵＤＡを用いた手動高速化に比べて、性能改善が十分でないアプリケーションが多いことが挙げられる。ＣＵＤＡにおける高速化手法では、逐次処理の並列化を大前提として、ＣＰＵ-ＧＰＵデータ転送削減、複数メモリの適切な使い分け（共有メモリ、コンスタントメモリ、テクスチャメモリ、ローカルメモリ、グローバルメモリ）がある。また、ＣＵＤＡにおける高速化手法では、コアレスアクセス、Warp内分岐の抑制、Warp同時マルチスレッドによる高occupancy化、ストリームによるタスク並列化、スレッド数に適した並列化粒度チューニング等がある。この中で、転送速度についてみると、高速化は、ＧＰＵ内でのメモリ効率化よりも、ＣＰＵ-ＧＰＵ転送の削減の方が効果が大きい。このため、ネストループ変数以外でも削減できる点について述べる。

[0022] 〈ＣＰＵ-ＧＰＵ転送の削減〉

ＣＰＵ-ＧＰＵ転送の削減のため、ネストループの変数をできるだけ上位で転送することに加え、本発明は（１）「多数の変数転送タイミングの一括化」、（２）「コンパイラが自動転送してしまう転送を削減」する。

ＧＰＵに処理をオフロードするため、ＣＰＵ-ＧＰＵ転送は必ず発生する。本発明は、変数転送タイミングの一括化や不要な転送を削減することで、転送数を減らして高速化を実現する。

[0023] （１）「多数の変数転送タイミングの一括化」について

転送の削減にあたり、ネスト単位だけでなく、ＧＰＵに転送するタイミングがまとめられる変数については一括化して転送する。例えば、ＧＰＵの処理結果をＣＰＵで加工してＧＰＵで再度処理させるなどの変数でなければ、複数のループ文で使われるＣＰＵで定義された変数を、ＧＰＵ処理が始まる前に一括してＧＰＵに送り、全ＧＰＵ処理が終わってからＣＰＵに戻すなどの対応も可能である。

[0024] コード分析時にループおよび変数の参照関係を把握するため、その結果から複数ファイルで定義された変数について、ＧＰＵ処理とＣＰＵ処理が入れ子にならず、ＣＰＵ処理とＧＰＵ処理が分けられる変数については、一括化して転送する指定をOpenACCのdata copy文（OpenACCのdata copy）を用いて指定する。併せて、一括化して転送され、そのタイミングで転送が不要な変数はOpenACCのdata present文（OpenACCのdata present）を用いて明示する。なお、present文とは、既にＧＰＵに変数があることを明示する節である。

[0025] （２）「コンパイラが自動転送してしまう転送の削減」について

コンパイラが自動転送する場合の転送の削減について述べる。

図１は、〈data copyやpresentを使用しないケース〉と、〈data copyやpresentを使用するケース〉と、〈data copyやpresentを使用し、かつ一時領域をデータの格納場所として使用するケース〉と、の各々において、ＣＰＵ-ＧＰＵ間のパラメータおよびデータ転送を説明する図である。

図１中の一方方向の矢印（⇒）は、ＣＰＵからＧＰＵ、またはＧＰＵから

ＣＰＵへのデータ転送を示し、図２中の双方向の矢印（⇔）は、ＣＰＵ-ＧＰＵ間の双方向のデータ転送を示している。

[0026] 図１の<data copyやpresentを使用しないケース>は、OpenACCのコンパイラとして著名なＰＧＩコンパイラのケースである。すなわち、パラメータ領域確保およびパラメータ初期化については、ＣＰＵがＧＰＵに対しパラメータデータ転送を行うとともに、ＧＰＵが初期化データを受信する。

[0027] また、ループ開始通知送信およびループ終了通知受信については、ＣＰＵからＧＰＵ、またはＧＰＵからＣＰＵ、ＣＰＵからＧＰＵへの双方向パラメータデータ転送を行う。すなわち、ＣＰＵはＧＰＵに対しループ開始通知送信を行うとともに、ＧＰＵからループ終了通知を受信する。これにより、ＧＰＵは、ループ単位でホスト（ここではＣＰＵ）と同期する。

[0028] このように、OpenACCのdata copyやpresent節を用いずに、単にループを#pragma acc kenerls節でＧＰＵ処理を指定している場合は、ループ単位でＣＰＵとＧＰＵの間でループ内変数の同期が行われる。

[0029] 図１の<data copyやpresentを使用するケース>は、非特許文献２におけるケースである。すなわち、図１の<data copyやpresentを使用するケース>は、パラメータ領域確保およびパラメータ初期化に加え、データ領域開始通知送信についても、ＣＰＵがＧＰＵに対しパラメータデータ転送を行うとともに、ＧＰＵが初期化データを受信する。

また、ループ開始通知送信については、ＣＰＵ-ＧＰＵ間の同期転送である。ＧＰＵからみると、ループ構成によって自動同期が行われる。そして、ＣＰＵはＧＰＵからループ終了通知を受信し、これを受けてＧＰＵに対しデータ領域終了を通知する。ＧＰＵは、ＣＰＵに対して最終結果をホスト（ここではＣＰＵ）に送信してホストと同期する。

[0030] 非特許文献２では、data copyやpresentをOpenACCで指定した場合でも、コンパイラで変数がＣＰＵ-ＧＰＵで自動転送される場合がある。コンパイラは基本的に安全側に処理を行うため、グローバル変数かローカル変数であるか、初期化はどこでされるか、ループ含む他関数から取得されるものか、参照

されるだけか、ループ内で更新されるものか、等複数の条件によって、コンパイラ依存で転送が不要であっても転送が発生する。

[0031] 図１の<data copyやpresentを使用し、かつ一時領域をデータの格納場所として使用するケース>は、本発明のケースである。

本発明は、OpenACCの指示では意図しないが性能を劣化する転送を削減するため、一時領域を作成し一時領域でパラメータを初期化して、ＣＰＵ-ＧＰＵ転送に用いることで、不要なＣＰＵ-ＧＰＵ転送を遮断する。

すなわち、本発明は、図１の<data copyやpresentを使用し、かつ一時領域をデータの格納場所として使用するケース>に示すように、ＧＰＵで、一時領域を作成し、一時領域でパラメータを作成する。この一時領域でパラメータを作成することがいままでなかった点である。

そして、図１の<data copyやpresentを使用し、かつ一時領域をデータの格納場所として使用するケース>に示すように、ＣＰＵはＧＰＵからデータ領域開始通知を受信し、ＣＰＵに対して最終結果をホスト（ここではＣＰＵ）に送信してホストと同期する。

[0032] （実施形態）

次に、本発明を実施するための形態（以下、「本実施形態」と称する。）における、オフロードサーバ１等について説明する。

図２は、本発明の基本的考え方を適用した、本実施形態に係るオフロードサーバ１を含む環境適応ソフトウェアシステムを示す図である。

本実施形態に係る環境適応ソフトウェアシステムは、従来の環境適応ソフトウェアの構成に加え、オフロードサーバ１を含むことを特徴とする。オフロードサーバ１は、アプリケーションの特定処理をアクセラレータにオフロードするオフロードサーバである。また、オフロードサーバ１は、クラウドレイヤ２、ネットワークレイヤ３、デバイスレイヤ４の３層に位置する各装置と通信可能に接続される。クラウドレイヤ２にはデータセンタ３０が、ネットワークレイヤ３にはネットワークエッジ２０が、デバイスレイヤ４にはゲートウェイ１０が、それぞれ配設される。

[0033] そこで、本実施形態に係るオフロードサーバ１を含む環境適応ソフトウェアシステムでは、デバイスレイヤ、ネットワークレイヤ、クラウドレイヤのそれぞれのレイヤにおいて、機能配置や処理オフロードを適切に行うことによる効率化を実現する。主に、機能を３レイヤの適切な場所に配置し処理させる機能配置効率化と、画像分析等の機能処理をＧＰＵやＦＰＧＡ（Field Programmable Gate Array）等のヘテロハードウェアにオフロードすることでの効率化を図る。クラウドレイヤでは、ＧＰＵやＦＰＧＡ等のヘテロジニアスなＨＷ（ハードウェア）（以下、「ヘテロデバイス」と称する。）を備えたサーバが増えてきている。例えば、Microsoft（登録商標）社のBing検索においても、ＦＰＧＡが利用されている。このように、ヘテロデバイスを活用し、例えば、行列計算等をＧＰＵにオフロードしたり、ＦＦＴ（Fast Fourier Transform）計算等の特定処理をＦＰＧＡにオフロードしたりすることで、高性能化を実現している。

[0034] 以下、本実施形態に係るオフロードサーバ１が、環境適応ソフトウェアシステムにおけるユーザ向けサービス利用のバックグラウンドで実行するオフロード処理を行う際の構成例について説明する。

[0035] 図３は、本発明の実施形態に係るオフロードサーバ１の構成例を示す機能ブロック図である。

オフロードサーバ１は、アプリケーションの特定処理をアクセラレータに自動的にオフロードする装置である。

図３に示すように、オフロードサーバ１は、制御部１１と、入出力部１２と、記憶部１３と、検証用マシン１４（Verification machine）（アクセラレータ検証用装置）と、を含んで構成される。

[0036] 入出力部１２は、各機器等との間で情報の送受信を行うための通信インタフェースと、タッチパネルやキーボード等の入力装置や、モニタ等の出力装置との間で情報の送受信を行うための入出力インタフェースとから構成される。

[0037] 記憶部１３は、ハードディスクやフラッシュメモリ、ＲＡＭ（Random Acce

ss Memory）等により構成される。

この記憶部１３には、テストケースＤＢ（Test case database）１３１が記憶されるとともに、制御部１１の各機能を実行させるためのプログラム（オフロードプログラム）や、制御部１１の処理に必要な情報（例えば、中間言語ファイル(Intermediate file)１３２）が一時的に記憶される。

[0038] テストケースＤＢ１３１には、性能試験項目が格納される。テストケースＤＢ１３１は、高速化するアプリケーションの性能を測定するような試験を行うための情報が格納される。例えば、画像分析処理の深層学習アプリケーションであれば、サンプルの画像とそれを実行する試験項目である。

検証用マシン１４は、環境適応ソフトウェアの検証用環境として、ＣＰＵ（Central Processing Unit）、ＧＰＵ、ＦＰＧＡ(アクセラレータ)を備える。

[0039] 制御部１１は、オフロードサーバ１全体の制御を司る自動オフロード機能部（Automatic Offloading function）である。制御部１１は、例えば、記憶部１３に格納されたプログラム（オフロードプログラム）を不図示のＣＰＵが、ＲＡＭに展開し実行することにより実現される。

[0040] 制御部１１は、アプリケーションコード指定部（Specify application code）１１１と、アプリケーションコード分析部（Analyze application code）１１２と、データ転送指定部１１３と、並列処理指定部１１４と、並列処理パターン作成部１１５と、性能測定部１１６と、実行ファイル作成部１１７と、本番環境配置部（Deploy final binary files to production environment）１１８と、性能測定テスト抽出実行部（Extract performance test cases and run automatically）１１９と、ユーザ提供部（Provide price and performance to a user to judge）１２０と、を備える。

[0041] <アプリケーションコード指定部１１１>

アプリケーションコード指定部１１１は、入力されたアプリケーションコードの指定を行う。具体的には、アプリケーションコード指定部１１１は、ユーザに提供しているサービスの処理機能（画像分析等）を特定する。

[0042] 〈アプリケーションコード分析部１１２〉

アプリケーションコード分析部１１２は、処理機能のソースコードを分析し、ループ文やＦＦＴライブラリ呼び出し等の構造を把握する。

[0043] 〈データ転送指定部１１３〉

データ転送指定部１１３は、コード分析の結果をもとに、ＣＰＵとＧＰＵ間の転送が必要な変数の中で、ＣＰＵ処理とＧＰＵ処理とが相互に参照または更新がされず、ＧＰＵ処理した結果をＣＰＵに返すだけの変数については、ＧＰＵ処理の開始前と終了後に一括化してデータ転送する指定を行う。

ここで、ＣＰＵとＧＰＵ間の転送が必要な変数は、コード分析の結果から複数ファイルまたは複数ループで定義された変数である。

[0044] データ転送指定部１１３は、ＧＰＵ処理の開始前と終了後に一括化してデータ転送する指定を、OpenACCのdata copyを用いて指定する。

[0045] データ転送指定部１１３は、ＧＰＵで処理すべき変数が、既にＧＰＵ側に一括転送されている場合に、転送不要である指示句を追加する。

[0046] データ転送指定部１１３は、ＧＰＵ処理の始まる前に一括化して転送され、かつループ文処理のタイミングで転送が不要な変数についてはOpenACCのdata presentを用いて転送不要であることを明示する。

[0047] データ転送指定部１１３は、ＣＰＵとＧＰＵ間のデータ転送時に、ＧＰＵ側で一時領域を作成し（#pragma acc declare create）、データを一時領域に格納後、当該一時領域を同期（#pragma acc update）することで変数転送を指示する。

[0048] データ転送指定部１１３は、コード分析の結果をもとに、ループ文へのＧＰＵ処理を、OpenACCの、kernels指示句、parallel loop指示句、およびparallel loop vector指示句からなる群より選択される少なくとも一つを用いて指定する。

[0049] OpenACCのkernels指示句は、single loopおよびtightly nested loopに用いる。

OpenACCのparallel loop指示句は、non-tightly nested loopに用いる。

OpenACCのparallel loop vector指示句は、parallelizeはできないがvectorizeはできるループに用いる。

[0050] 〈並列処理指定部１１４〉

並列処理指定部１１４は、アプリケーションのループ文（繰り返し文）を特定し、各繰り返し文に対して、ＧＰＵにおける処理をOpenACCの指示句で指定してコンパイルする。

並列処理指定部１１４は、オフロード範囲抽出部（Extract offloadable area）１１４ａと、中間言語ファイル出力部（Output intermediate file）１１４ｂと、を備える。

[0051] オフロード範囲抽出部１１４ａは、ループ文等、ＧＰＵオフロード可能な処理を特定し、オフロード処理に応じた中間言語を抽出する。

[0052] 中間言語ファイル出力部１１４ｂは、抽出した中間言語ファイル１３２を出力する。中間言語抽出は、一度で終わりでなく、適切なオフロード領域探索のため、実行を試行して最適化するため反復される。

[0053] 〈並列処理パターン作成部１１５〉

並列処理パターン作成部１１５は、コンパイルエラーが出るループ文（繰り返し文）に対して、オフロード対象外とするとともに、コンパイルエラーが出ない繰り返し文に対して、並列処理するかしないかの指定を行う並列処理パターンを作成する。

[0054] 〈性能測定部１１６〉

性能測定部１１６は、並列処理パターンのアプリケーションをコンパイルして、検証用マシン１４に配置し、ＧＰＵにオフロードした際の性能測定用処理を実行する。

性能測定部１１６は、バイナリファイル配置部（Deploy binary files）１１６ａを備える。バイナリファイル配置部１１６ａは、ＧＰＵを備えた検証用マシン１４に、中間言語から導かれる実行ファイルをデプロイ(配置)する。

[0055] 性能測定部１１６は、配置したバイナリファイルを実行し、オフロードし

た際の性能を測定するとともに、性能測定結果を、オフロード範囲抽出部１１４ａに戻す。この場合、オフロード範囲抽出部１１４ａは、別の並列処理パターン抽出を行い、中間言語ファイル出力部１１４ｂは、抽出された中間言語をもとに、性能測定を試行する（後記図３の符号ｅ参照）。

[0056] 〈実行ファイル作成部１１７〉

実行ファイル作成部１１７は、所定回数繰り返された、性能測定結果をもとに、複数の並列処理パターンから最高処理性能の並列処理パターンを選択し、最高処理性能の並列処理パターンをコンパイルして実行ファイルを作成する。

[0057] 〈本番環境配置部１１８〉

本番環境配置部１１８は、作成した実行ファイルを、ユーザ向けの本番環境に配置する（「最終バイナリファイルの本番環境への配置」）。本番環境配置部１１８は、最終的なオフロード領域を指定したパターンを決定し、ユーザ向けの本番環境にデプロイする。

[0058] 〈性能測定テスト抽出実行部１１９〉

性能測定テスト抽出実行部１１９は、実行ファイル配置後、テストケースＤＢ１３１から性能試験項目を抽出し、性能試験を実行する（「最終バイナリファイルの本番環境への配置」）。

性能測定テスト抽出実行部１１９は、実行ファイル配置後、ユーザに性能を示すため、性能試験項目をテストケースＤＢ１３１から抽出し、抽出した性能試験を自動実行する。

[0059] 〈ユーザ提供部１２０〉

ユーザ提供部１２０は、性能試験結果を踏まえた、価格・性能等の情報をユーザに提示する（「価格・性能等の情報のユーザへの提供」）。テストケースＤＢ１３１には、アプリケーションの性能を測定する試験を自動で行うためのデータが格納されている。ユーザ提供部１２０は、テストケースＤＢ１３１の試験データを実行した結果と、システムに用いられるリソース（仮想マシンや、ＦＰＧＡインスタンス、ＧＰＵインスタンス等）の各単価から

決まるシステム全体の価格をユーザに提示する。ユーザは、提示された価格・性能等の情報をもとに、サービスの課金利用開始を判断する。

[0060] ［遺伝的アルゴリズムの適用］

オフロードサーバ１は、オフロードの最適化にＧＡ等の進化計算手法を用いることができる。ＧＡを用いた場合のオフロードサーバ１の構成は下記の通りである。

すなわち、並列処理指定部１１４は、遺伝的アルゴリズムに基づき、コンパイルエラーが出ないループ文（繰り返し文）の数を遺伝子長とする。並列処理パターン作成部１１５は、アクセラレータ処理をする場合を１または０のいずれか一方、しない場合を他方の０または１として、アクセラレータ処理可否を遺伝子パターンにマッピングする。

[0061] 並列処理パターン作成部１１５は、遺伝子の各値を１か０にランダムに作成した指定個体数の遺伝子パターンを準備し、性能測定部１１６は、各個体に応じて、ＧＰＵにおける並列処理指定文を指定したアプリケーションコードをコンパイルして、検証用マシン１４に配置する。性能測定部１１６は、検証用マシン１４において性能測定用処理を実行する。

[0062] ここで、性能測定部１１６は、途中世代で、以前と同じ並列処理パターンの遺伝子が生じた場合は、当該並列処理パターンに該当するアプリケーションコードのコンパイル、および、性能測定はせずに、性能測定値としては同じ値を使う。

また、性能測定部１１６は、コンパイルエラーが生じるアプリケーションコード、および、性能測定が所定時間で終了しないアプリケーションコードについては、タイムアウトの扱いとして、性能測定値を所定の時間（長時間）に設定する。

[0063] 実行ファイル作成部１１７は、全個体に対して、性能測定を行い、処理時間の短い個体ほど適合度が高くなるように評価する。実行ファイル作成部１１７は、全個体から、適合度が高いものを性能の高い個体として選択し、選択された個体に対して、交叉、突然変異の処理を行い、次世代の個体を作成

する。上記選択は、適合度の比に応じて確率的に選ぶルーレット選択等の方法がある。実行ファイル作成部１１７は、指定世代数の処理終了後、最高性能の並列処理パターンを解として選択する。

[0064] 以下、上述のように構成されたオフロードサーバ１の自動オフロード動作について説明する。

［自動オフロード動作］

本実施形態のオフロードサーバ１は、環境適応ソフトウェアの要素技術としてユーザアプリケーションロジックのＧＰＵ自動オフロードに適用した例である。

図４は、オフロードサーバ１のＧＡを用いた自動オフロード処理を示す図である。

図４に示すように、オフロードサーバ１は、環境適応ソフトウェアの要素技術に適用される。オフロードサーバ１は、制御部（自動オフロード機能部）１１と、テストケースＤＢ１３１と、中間言語ファイル１３２と、検証用マシン１４と、を有している。

オフロードサーバ１は、ユーザが利用するアプリケーションコード（Application code）１３０を取得する。

[0065] オフロードサーバ１は、機能処理をＣＰＵ-ＧＰＵを有する装置１５２、ＣＰＵ-ＦＰＧＡを有する装置１５３のアクセラレータに自動オフロードする。

[0066] 以下、図４のステップ番号を参照して各部の動作を説明する。

〈ステップＳ１１：Specify application code〉

ステップＳ１１において、アプリケーションコード指定部１１１（図３参照）は、ユーザに提供しているサービスの処理機能（画像分析等）を特定する。具体的には、アプリケーションコード指定部１１１は、入力されたアプリケーションコードの指定を行う。

[0067] 〈ステップＳ１２：Analyze application code〉

ステップＳ１２において、アプリケーションコード分析部１１２（図３参照）は、処理機能のソースコードを分析し、ループ文やＦＦＴライブラリ呼

び出し等の構造を把握する。

[0068] 〈ステップＳ１３：Extract offloadable area〉

ステップＳ１３において、並列処理指定部１１４（図３参照）は、アプリケーションのループ文（繰り返し文）を特定し、各繰り返し文に対して、ＧＰＵ処理をOpenACCで指定してコンパイルする。具体的には、オフロード範囲抽出部１１４ａ（図３参照）は、ループ文等、ＧＰＵにオフロード可能な処理を特定し、オフロード処理に応じた中間言語を抽出する。

[0069] 〈ステップＳ１４：Output intermediate file〉

ステップＳ１４において、中間言語ファイル出力部１１４ｂ（図３参照）は、中間言語ファイル１３２を出力する。中間言語抽出は、一度で終わりでなく、適切なオフロード領域探索のため、実行を試行して最適化するため反復される。

[0070] 〈ステップＳ１５：Compile error〉

ステップＳ１５において、並列処理パターン作成部１１５（図３参照）は、コンパイルエラーが出るループ文に対して、オフロード対象外とするとともに、コンパイルエラーが出ない繰り返し文に対して、並列処理するかしないかの指定を行う並列処理パターンを作成する。

[0071] 〈ステップＳ２１：Deploy binary files〉

ステップＳ２１において、バイナリファイル配置部１１６ａ（図３参照）は、ＧＰＵを備えた検証用マシン１４に、中間言語から導かれる実行ファイルをデプロイする。バイナリファイル配置部１１６ａは、配置したファイルを起動し、想定するテストケースを実行して、オフロードした際の性能を測定する。

[0072] 〈ステップＳ２２：Measure performances〉

ステップＳ２２において、性能測定部１１６（図３参照）は、配置したファイルを実行し、オフロードした際の性能を測定する。

オフロードする領域をより適切にするため、この性能測定結果は、オフロード範囲抽出部１１４ａに戻され、オフロード範囲抽出部１１４ａが、別パ

ターンの抽出を行う。そして、中間言語ファイル出力部１１４ｂは、抽出された中間言語をもとに、性能測定を試行する（図４の符号ｅ参照）。性能測定部１１６は、検証環境での性能測定を繰り返し、最終的にデプロイするコードパターンを決定する。

[0073] 図４の符号ｅに示すように、制御部１１は、各繰返し文に対して、ＧＰＵ処理をOpenACCで指定してコンパイルする。

[0074] 〈ステップＳ２３：Deploy final binary files to production environment〉

ステップＳ２３において、本番環境配置部１１８は、最終的なオフロード領域を指定したパターンを決定し、ユーザ向けの本番環境にデプロイする。

[0075] 〈ステップＳ２４：Extract performance test cases and run automatically〉

ステップＳ２４において、性能測定テスト抽出実行部１１９は、実行ファイル配置後、ユーザに性能を示すため、性能試験項目をテストケースＤＢ１３１から抽出し、抽出した性能試験を自動実行する。

[0076] 〈ステップＳ２５：Provide price and performance to a user to judge〉

ステップＳ２５において、ユーザ提供部１２０は、性能試験結果を踏まえた、価格・性能等の情報をユーザに提示する。ユーザは、提示された価格・性能等の情報をもとに、サービスの課金利用開始を判断する。

[0077] 上記ステップＳ１１～ステップＳ２５は、ユーザのサービス利用のバックグラウンドで行われ、例えば、仮利用の初日の間に行う等を想定している。

[0078] 上記したように、オフロードサーバ１の制御部（自動オフロード機能部）１１は、環境適応ソフトウェアの要素技術に適用した場合、機能処理のオフロードのため、ユーザが利用するアプリケーションのソースコードから、オフロードする領域を抽出して中間言語を出力する（ステップＳ１１～ステップＳ１５）。制御部１１は、中間言語から導かれる実行ファイルを、検証用マシン１４に配置実行し、オフロード効果を検証する（ステップＳ２１～ステップＳ２２）。検証を繰り返し、適切なオフロード領域を定めたのち、制

御部１１は、実際にユーザに提供する本番環境に、実行ファイルをデプロイし、サービスとして提供する（ステップＳ２３～ステップＳ２５）。

[0079] なお、上記では、環境適応に必要な、コード変換、リソース量調整、配置場所調整を一括して行う処理フローを説明したが、これに限らず、行いたい処理だけ切出すことも可能である。例えば、ＧＰＵ向けにコード変換だけ行いたい場合は、上記ステップＳ１１～ステップＳ２１の、環境適応機能や検証環境等必要な部分だけ利用すればよい。

[0080] ［ＧＡを用いたＧＰＵ自動オフロード］

ＧＰＵ自動オフロードは、ＧＰＵに対して、図４のステップＳ１２～ステップＳ２２を繰り返し、最終的にステップＳ２３でデプロイするオフロードコードを得るための処理である。

[0081] ＧＰＵは、一般的にレイテンシーは保証しないが、並列処理によりスループットを高めることに向いたデバイスである。ＩｏＴで動作させるアプリケーションは、多種多様である。ＩｏＴデータの暗号化処理や、カメラ映像分析のための画像処理、大量センサデータ分析のための機械学習処理等が代表的であり、それらは、繰り返し処理が多い。そこで、アプリケーションの繰り返し文をＧＰＵに自動でオフロードすることでの高速化を狙う。

[0082] しかし、従来技術で記載の通り、高速化には適切な並列処理が必要である。特に、ＧＰＵを使う場合は、ＣＰＵとＧＰＵ間のメモリ転送のため、データサイズやループ回数が多くないと性能が出ないことが多い。また、メモリデータ転送のタイミング等により、並列高速化できる個々のループ文（繰り返し文）の組み合わせが、最速とならない場合等がある。例えば、１０個のfor文（繰り返し文）で、１番、５番、１０番の３つがＣＰＵに比べて高速化できる場合に、１番、５番、１０番の３つの組み合わせが最速になるとは限らない等である。

[0083] 適切な並列領域指定のため、ＰＧＩコンパイラを用いて、for文の並列可否を試行錯誤して最適化する試みがある。しかし、試行錯誤には多くの稼働がかかり、サービスとして提供する際に、ユーザの利用開始が遅くなり、コス

トも上がってしまう問題がある。

[0084] そこで、本実施形態では、並列化を想定していない汎用プログラムから、自動で適切なオフロード領域を抽出する。このため、最初に並列可能for文のチェックを行い、次に並列可能for文群に対してＧＡを用いて検証環境で性能検証試行を反復し適切な領域を探索すること、を実現する。並列可能for文に絞った上で、遺伝子の部分の形で、高速化可能な並列処理パターンを保持し組み換えていくことで、取り得る膨大な並列処理パターンから、効率的に高速化可能なパターンを探索できる。

[0085] ［Simple GAによる制御部（自動オフロード機能部）１１の探索イメージ］

図５は、Simple GAによる制御部（自動オフロード機能部）１１の探索イメージを示す図である。図５は、処理の探索イメージと、for文の遺伝子配列マッピングを示す。

ＧＡは、生物の進化過程を模倣した組合せ最適化手法の一つである。ＧＡのフローチャートは、初期化→評価→選択→交叉→突然変異→終了判定となっている。

本実施形態では、ＧＡの中で、処理を単純にしたSimple GAを用いる。Simple GAは、遺伝子は１、０のみとし、ルーレット選択、一点交叉、突然変異は１箇所の遺伝子の値を逆にする等、単純化されたＧＡである。

[0086] 〈初期化〉

初期化では、アプリケーションコードの全for文の並列可否をチェック後、並列可能for文を遺伝子配列にマッピングする。ＧＰＵ処理する場合は１、ＧＰＵ処理しない場合は０とする。遺伝子は、指定の個体数Ｍを準備し、１つのfor文にランダムに１、０の割り当てを行う。

具体的には、制御部（自動オフロード機能部）１１（図２参照）は、ユーザが利用するアプリケーションコード（Application code）１３０（図３参照）を取得し、図５に示すように、アプリケーションコード１３０のコードパターン（Code patterns）１４１からfor文の並列可否をチェックする。図５に示すように、コードパターン１４１から５つのfor文が見つかった場合（

図５の符号ｆ参照）、各for文に対して１桁、ここでは５つのfor文に対し５桁の１または０をランダムに割り当てる。例えば、ＣＰＵで処理する場合０、ＧＰＵに出す場合１とする。ただし、この段階では１または０をランダムに割り当てる。

遺伝子長に該当するコードが５桁であり、５桁の遺伝子長のコードは2^5＝３２パターン、例えば１０００１、１００１０、…となる。なお、図５では、コードパターン１４１中の丸印（〇印）をコードのイメージとして示している。

[0087] 〈評価〉

評価では、デプロイとパフォーマンスの測定（Deploy & performance measurement）を行う（図５の符号ｇ参照）。すなわち、性能測定部１１６（図３参照）は、遺伝子に該当するコードをコンパイルして検証用マシン１４にデプロイして実行する。性能測定部１１６は、ベンチマーク性能測定を行う。性能が良いパターン（並列処理パターン）の遺伝子の適合度を高くする。

[0088] 〈選択〉

選択では、適合度に基づいて、高性能コードパターンを選択（Select high performance code patterns）する（図５の符号ｈ参照）。性能測定部１１６（図３参照）は、適合度に基づいて、高適合度の遺伝子を、指定の個体数を選択する。本実施形態では、適合度に応じたルーレット選択および最高適合度遺伝子のエリート選択を行う。

図５では、選択されたコードパターン（Select code patterns）１４２の中の丸印（〇印）が、３つに減ったことを探索イメージとして示している。

[0089] 〈交叉〉

交叉では、一定の交叉率Ｐｃで、選択された個体間で一部の遺伝子をある一点で交換し、子の個体を作成する。

ルーレット選択された、あるパターン（並列処理パターン）と他のパターンとの遺伝子を交叉させる。一点交叉の位置は任意であり、例えば上記５桁のコードのうち３桁目で交叉させる。

[0090] 〈突然変異〉

突然変異では、一定の突然変異率Ｐｍで、個体の遺伝子の各値を０から１または１から０に変更する。

また、局所解を避けるため、突然変異を導入する。なお、演算量を削減するために突然変異を行わない態様でもよい。

[0091] 〈終了判定〉

図５に示すように、クロスオーバーと突然変異後の次世代コードパターンの生成（Generate next generation code patterns after crossover & mutation）を行う（図５の符号ｉ参照）。

終了判定では、指定の世代数Ｔ回、繰り返しを行った後に処理を終了し、最高適合度の遺伝子を解とする。

例えば、性能測定して、速い３つ１００１０、０１００１、００１０１を選ぶ。この３つをＧＡにより、次の世代は、組み換えをして、例えば新しいパターン（並列処理パターン）１０１０１（一例）を作っていく。このとき、組み換えをしたパターンに、勝手に０を１にするなどの突然変異を入れる。上記を繰り返して、一番早いパターンを見付ける。指定世代（例えば、２０世代）などを決めて、最終世代で残ったパターンを、最後の解とする。

[0092] 〈デプロイ（配置）〉

最高適合度の遺伝子に該当する、最高処理性能の並列処理パターンで、本番環境に改めてデプロイして、ユーザに提供する。

[0093] 〈補足説明〉

ＧＰＵにオフロードできないfor文（ループ文；繰り返し文）が相当数存在する場合について説明する。例えば、for文が２００個あっても、ＧＰＵにオフロードできるものは３０個くらいである。ここでは、エラーになるものを除外し、この３０個について、ＧＡを行う。

[0094] OpenＡＣＣには、ディレクティブ #pragma acc kernelsで指定して、ＧＰＵ向けバイトコードを抽出し、実行によりＧＰＵオフロードを可能とするコンパイラがある。この#pragmaに、for文のコマンドを書くことにより、そのf

or文がＧＰＵで動くか否かを判定することができる。

[0095] 例えばC/C++を使った場合、C/C++のコードを分析し、for文を見付ける。for文を見付けると、OpenＡＣＣで並列処理の文法である #pragma acc kernels を使ってfor文に対して書き込む。詳細には、何も入っていない #pragma acc kernels に、一つ一つfor文を入れてコンパイルして、エラーであれば、そのfor文はそもそも、ＧＰＵ処理できないので、除外する。このようにして、残るfor文を見付ける。そして、エラーが出ないものを、長さ（遺伝子長）とする。エラーのないfor文が５つであれば、遺伝子長は５であり、エラーのないfor文が１０であれば、遺伝子長は１０である。なお、並列処理できないものは、前の処理を次の処理に使うようなデータに依存がある場合である。

以上が準備段階である。次にＧＡ処理を行う。

[0096] for文の数に対応する遺伝子長を有するコードパターンが得られている。始めはランダムに並列処理パターン１００１０、０１００１、００１０１、…を割り当てる。ＧＡ処理を行い、コンパイルする。その時に、オフロードできるfor文であるにもかかわらず、エラーがでることがある。それは、for文が階層になっている（どちらか指定すればＧＰＵ処理できる）場合である。この場合は、エラーとなったfor文は、残してもよい。具体的には、処理時間が多くなった形にして、タイムアウトさせる方法がある。

[0097] 検証用マシン１４でデプロイして、ベンチマーク、例えば画像処理であればその画像処理でベンチマークする、その処理時間が短い程、適応度が高いと評価する。例えば、処理時間の逆数、処理時間１０秒かかるものは１、１００秒かかるものは０．１、１秒のものは１０とする。

適応度が高いものを選択して、例えば１０個のなかから、３～５個を選択して、それを組み替えて新しいコードパターンを作る。このとき、作成途中で、前と同じものができる場合がある。その場合、同じベンチマークを行う必要はないので、前と同じデータを使う。本実施形態では、コードパターンと、その処理時間は記憶部１３に保存しておく。

以上で、Simple GAによる制御部（自動オフロード機能部）１１の探索イメ

ージについて説明した。次に、データ転送の一括処理手法について述べる。

[0098] ［データ転送の一括処理手法］

〈基本的な考え方〉

ＣＰＵ-ＧＰＵ転送の削減のため、ネストループの変数をできるだけ上位で転送することに加え、本発明は、多数の変数転送タイミングを一括化し、さらにコンパイラが自動転送してしまう転送を削減する。

転送の削減にあたり、ネスト単位だけでなく、ＧＰＵに転送するタイミングがまとめられる変数については一括化して転送する。例えば、ＧＰＵの処理結果をＣＰＵで加工してＧＰＵで再度処理させるなどの変数でなければ、複数のループ文で使われるＣＰＵで定義された変数を、ＧＰＵ処理が始まる前に一括してＧＰＵに送り、全ＧＰＵ処理が終わってからＣＰＵに戻すなどの対応も可能である。

[0099] コード分析時にループおよび変数の参照関係を把握するため、その結果から複数ファイルで定義された変数について、ＧＰＵ処理とＣＰＵ処理が入れ子にならず、ＣＰＵ処理とＧＰＵ処理が分けられる変数については、一括化して転送する指定をOpenACCのdata copy文を用いて指定する。

ＧＰＵ処理の始まる前に一括化して転送され、ループ文処理のタイミングで転送が不要な変数はdata presentを用いて転送不要であることを明示する。

ＣＰＵ-ＧＰＵのデータ転送時は、一時領域を作成し（#pragma acc declare create）、データは一時領域に格納後、一時領域を同期（#pragma acc update）することで転送を指示する。

[0100] 〈比較例〉

まず、比較例について述べる。

比較例は、通常ＣＰＵプログラム（図６参照）、単純ＧＰＵ利用（図７参照）、ネスト一括化（非特許文献２）（図８参照）である。なお、以下の記載および図中のループ文の文頭の〈1〉～〈4〉等は、説明の便宜上で付したものである（他図およびその説明においても同様）。

図６に示す通常ＣＰＵプログラムのループ文は、ＣＰＵプログラム側で記述され、

<1> ループ〔for(i=0; i<10; i++)〕{

}

の中に、

<2> ループ〔for(j=0; j<20; j++〕 {

がある。図６の符号ｊは、上記 <2>ループにおける、変数ａ，ｂの設定である。

また、

<3> ループ〔for(k=0; k<30; k++)〕{

}

と、

<4> ループ〔for(l=0; l<40; l++)〕{

}

と、が続く。図６の符号ｋは、上記<3>ループにおける変数ｃ，ｄの設定であり、図６の符号ｌは、上記<4>ループにおける変数ｅ，ｆの設定である。

図６に示す通常ＣＰＵプログラムは、ＣＰＵで実行される（ＧＰＵ利用しない）。

[0101] 図７は、図６に示す通常ＣＰＵプログラムを、単純ＧＰＵ利用して、ＣＰＵからＧＰＵへのデータ転送する場合のループ文を示す図である。データ転送の種類は、ＣＰＵからＧＰＵへのデータ転送、および、ＧＰＵからＣＰＵへのデータ転送がある。以下、ＣＰＵからＧＰＵへのデータ転送を例にとる。

図７に示す単純ＧＰＵ利用のループ文は、ＣＰＵプログラム側で記述され、

<1> ループ〔for(i=0; i<10; i++)〕{

}

の中に、

〈2〉ループ〔for(j=0; j<20; j++〕 {

がある。

さらに、図７の符号ｍに示すように、〈1〉ループ〔for(i=0; i<10; i++)〕{

}の上部に、ＰＧＩコンパイラによるfor文等の並列処理可能処理部を、OpenＡＣＣのディレクティブ #pragma acc kernels（並列処理指定文）で指定している。

図７の符号ｍを含む破線枠囲みに示すように、#pragma acc kernelsによって、ＣＰＵからＧＰＵへデータ転送される。ここでは、このタイミングでａ，ｂが転送されるため１０回転送される。

[0102] また、図７の符号ｎに示すように、〈3〉ループ〔for(k=0; k<30; k++)〕{

}の上部に、ＰＧＩコンパイラによるfor文等の並列処理可能処理部を、OpenＡＣＣのディレクティブ #pragma acc kernelsで指定している。

図７の符号ｎを含む破線枠囲みに示すように、#pragma acc kernelsによって、このタイミングでｃ，ｄが転送される。

[0103] ここで、〈4〉ループ〔for(l=0; l<40; l++)〕{

}の上部には、#pragma acc kernelsを指定しない。このループは、ＧＰＵ処理しても効率が悪いのでＧＰＵ処理しない。

[0104] 図８は、ネスト一括化（非特許文献２）による、ＣＰＵからＧＰＵおよびＧＰＵからＣＰＵへのデータ転送する場合のループ文を示す図である。

図８に示すループ文では、図８の符号ｏに示す位置に、ＣＰＵからＧＰＵへのデータ転送指示行、ここでは変数ａ，ｂのcopyin 節の #pragma acc data copyin(a, ｂ)を挿入する。

上記 #pragma acc data copyin(a, ｂ)は、変数ａの設定、定義を含まない最上位のループ（ここでは、〈1〉ループ〔for(i=0; i<10; i++)〕{

}の上部）に指定される。

図８の符号ｏを含む一点鎖線枠囲みに示すタイミングでａ，ｂが転送されるため１回転送が発生する。

[0105] また、図８に示すループ文では、図８の符号ｐに示す位置に、ＧＰＵからＣＰＵへのデータ転送指示行、ここでは変数ａ，ｂの copyout 節の #pragma acc data copyout(a, ｂ)を挿入する。

上記 #pragma acc data copyout(a, ｂ)は、〈1〉ループ〔for(i=0; i<10; i++)〕{
}の下部に指定される。

[0106] このように、ＣＰＵからＧＰＵへのデータ転送において、変数ａの copyin 節の #pragma acc data copyin(a, ｂ)を、上述した位置に挿入することによりデータ転送を明示的に指示する。これにより、できるだけ上位のループでデータ転送を一括して行うことができ、図７の符号ｍに示す単純ＧＰＵ利用のループ文のようにループ毎に毎回データを転送する非効率な転送を避けることができる。

[0107] 〈実施形態〉

次に、本実施形態について述べる。

《転送不要な変数をdata presentを用いて明示》

本実施形態では、複数ファイルで定義された変数について、ＧＰＵ処理とＣＰＵ処理が入れ子にならず、ＣＰＵ処理とＧＰＵ処理が分けられる変数については、一括化して転送する指定をOpenACCのdata copy文を用いて指定する。併せて、一括化して転送され、そのタイミングで転送が不要な変数はdata presentを用いて明示する。

[0108] 図９は、本実施形態のＣＰＵ-ＧＰＵのデータ転送時の転送一括化によるループ文を示す図である。図９は、比較例の図８のネスト一括化に対応する。

図９に示すループ文では、図９の符号ｏに示す位置に、ＣＰＵからＧＰＵへのデータ転送指示行、ここでは変数ａ，ｂ，ｃ，ｄの copyin 節の #pragma acc datacopyin(a, ｂ, c, d)を挿入する。

上記 #pragma acc data copyin(a, ｂ, ｃ, ｄ)は、変数ａの設定、定義を含まない最上位のループ（ここでは、〈1〉ループ〔for(i=0; i<10; i++)〕{
}の上部）に指定される。

[0109] このように、複数ファイルで定義された変数について、ＧＰＵ処理とＣＰＵ処理が入れ子にならず、ＣＰＵ処理とＧＰＵ処理が分けられる変数については、一括化して転送する指定をOpenACCのdata copy文#pragma acc data copyin(a, ｂ, ｃ, ｄ)を用いて指定する。

図９の符号ｏを含む一点鎖線枠囲みに示すタイミングでａ，ｂ，ｃ，ｄが転送されるため１回転送が発生する。

[0110] そして、上記#pragma acc data copyin(a, ｂ, ｃ, ｄ)を用いて一括化して転送され、そのタイミングで転送が不要な変数は、図９の符号ｑを含む二点鎖線枠囲みに示すタイミングで既にＧＰＵに変数があることを明示するdata present文#pragma acc data present (a, ｂ)を用いて指定する。

[0111] 上記#pragma acc data copyin(a, ｂ, ｃ, ｄ)を用いて一括化して転送され、そのタイミングで転送が不要な変数は、図９の符号ｐを含む二点鎖線枠囲みに示すタイミングで既にＧＰＵに変数があることを明示するdata present文#pragma acc data present(ｃ, ｄ)を用いて指定する。

<1>、<3>のループがＧＰＵ処理されＧＰＵ処理が終了したタイミングで、ＧＰＵからＣＰＵへのデータ転送指示行、ここでは変数ａ，ｂ，ｃ，ｄの copyout 節の #pragma acc datacopyout(a, ｂ, c, d)を、図９の<3>ループが終了した位置ｐに挿入する。

[0112] 一括化して転送する指定により一括化して転送できる変数は一括転送し、既に転送され転送が不要な変数はdata presentを用いて明示することで、転送を削減して、オフロード手段のさらなる効率化を図ることができる。しかし、OpenACCで転送を指示してもコンパイラによっては、コンパイラが自動判断して転送してしまう場合がある。コンパイラによる自動転送とは、OpenACCの指示と異なり、本来はＣＰＵ－ＧＰＵ間の転送が不要であるにもかかわらずコンパイラ依存で自動転送されてしまう事象のことである。

[0113] 《データの一時領域格納》

図１０は、本実施形態のＣＰＵ-ＧＰＵのデータ転送時の転送一括化によるループ文を示す図である。図１０は、図９のネスト一括化および転送不要な

変数明示に対応する。

図１０に示すループ文では、図１０の符号ｓに示す位置に、ＣＰＵ-ＧＰＵのデータ転送時、一時領域を作成するOpenACCのdeclare create文#pragma acc declare createを指定する。これにより、ＣＰＵ-ＧＰＵのデータ転送時は、一時領域を作成し（#pragma acc declare create）、データは一時領域に格納される。

[0114] また、図１０の符号ｔに示す位置に、一時領域を同期するためのOpenACCのdeclare create文#pragma acc updateを指定することで転送を指示する。

[0115] このように、一時領域を作成し、一時領域でパラメータを初期化して、ＣＰＵ-ＧＰＵ転送に用いることで、不要なＣＰＵ-ＧＰＵ転送を遮断する。OpenACCの指示では意図しないが性能を劣化する転送を削減することができる。

[0116] ［ＧＰＵオフロード処理］

上述したデータ転送の一括処理手法により、オフロードに適切なループ文を抽出し、非効率なデータ転送を避けることができる。

ただし、上記データ転送の一括処理手法を用いても、ＧＰＵオフロードに向いていないプログラムも存在する。効果的なＧＰＵオフロードには、オフロードする処理のループ回数が多いことが必要である。

[0117] そこで、本実施形態では、本格的なオフロード処理探索の前段階として、プロファイリングツールを用いて、ループ回数を調査する。プロファイリングツールを用いると、各行の実行回数を調査できるため、例えば、５０００万回以上のループを持つプログラムをオフロード処理探索の対象とする等、事前に振り分けることができる。以下、具体的に説明する（図４で述べた内容と一部重複する）。

[0118] 本実施形態では、まず、オフロード処理部を探索するアプリケーションを分析し、for，do，while等のループ文を把握する。次に、サンプル処理を実行し、プロファイリングツールを用いて、各ループ文のループ回数を調査し、一定の値以上のループがあるか否かで、オフロード処理部探索を本格的に行うか否かの判定を行う。

[0119] 探索を本格的に行うと決まった場合は、ＧＡの処理に入る（図４参照）。初期化ステップでは、アプリケーションコードの全ループ文の並列可否をチェックした後、並列可能ループ文をＧＰＵ処理する場合は１、しない場合は０として遺伝子配列にマッピングする。遺伝子は、指定の個体数が準備されるが、遺伝子の各値にはランダムに１，０の割り当てをする。

[0120] ここで、遺伝子に該当するコードでは、ＧＰＵ処理すると指定されたループ文内の変数データ参照関係から、データ転送の明示的指示（#pragma acc data copyin/copyout/copy）を追加する。

[0121] 評価ステップでは、遺伝子に該当するコードをコンパイルして検証用マシンにデプロイして実行し、ベンチマーク性能測定を行う。性能が良いパターンの遺伝子の適合度を高くする。遺伝子に該当するコードは、上述のように、並列処理指示行（例えば、図６の符号ｊ参照）とデータ転送指示行（例えば、図６の符号ｌ、図７の符号ｍ参照、図８の符号ｏ参照）が挿入されている。

[0122] 選択ステップでは、適合度に基づいて、高適合度の遺伝子を、指定の個体数分選択する。本実施形態では、適合度に応じたルーレット選択および最高適合度遺伝子のエリート選択を行う。交叉ステップでは、一定の交叉率Ｐｃで、選択された個体間で一部の遺伝子をある一点で交換し、子の個体を作成する。突然変異ステップでは、一定の突然変異率Ｐｍで、個体の遺伝子の各値を０から１または１から０に変更する。

[0123] 突然変異ステップまで終わり、次の世代の遺伝子が指定個体数作成されると、初期化ステップと同様に、データ転送の明示的指示を追加し、評価、選択、交叉、突然変異ステップを繰り返す。

[0124] 最後に、終了判定ステップでは、指定の世代数、繰り返しを行った後に処理を終了し、最高適合度の遺伝子を解とする。最高適合度の遺伝子に該当する、最高性能のコードパターンで、本番環境に改めてデプロイして、ユーザに提供する。

[0125] 以下、オフロードサーバ１の実装を説明する。本実装は、本実施形態の有

効性を確認するためのものである。

［実装］

C/C++アプリケーションを汎用のＰＧＩコンパイラを用いて自動オフロードする実装を説明する。

本実装では、ＧＰＵ自動オフロードの有効性確認が目的であるため、対象アプリケーションはC/C++言語のアプリケーションとし、ＧＰＵ処理自体は、従来のＰＧＩコンパイラを説明に用いる。

[0126] C/C++言語は、ＯＳＳ（Open Source Software）およびproprietaryソフトウェアの開発で、上位の人気を誇り、数多くのアプリケーションがC/C++言語で開発されている。一般ユーザが用いるアプリケーションのオフロードを確認するため、暗号処理や画像処理等のＯＳＳの汎用アプリケーションを利用する。

[0127] ＧＰＵ処理は、ＰＧＩコンパイラにより行う。ＰＧＩコンパイラは、OpenＡＣＣを解釈するC/C++/Fortran向けコンパイラである。本実施形態では、for文等の並列可能処理部を、OpenＡＣＣのディレクティブ #pragma acc kernels（並列処理指定文）で指定する。これにより、ＧＰＵ向けバイトコードを抽出し、その実行によりＧＰＵオフロードを可能としている。さらに、for文内のデータ同士に依存性があり並列処理できない処理やネストのfor文の異なる複数の階層を指定されている場合等の際に、エラーを出す。併せて、#pragma acc data copyin/copyout/copy 等のディレクティブにより、明示的なデータ転送の指示が可能とする。

[0128] 上記 #pragma acc kernels（並列処理指定文）での指定に合わせて、OpenＡＣＣのcopyin 節の#pragma acc data copyout(a[…])の、上述した位置への挿入により、明示的なデータ転送の指示を行う。

[0129] <実装の動作概要>

実装の動作概要を説明する。

実装は、以下の処理を行う。

下記図１１Ａ－Ｂのフローの処理を開始する前に、高速化するC/C++アプリ

ケーションとそれを性能測定するベンチマークツールを準備する。

[0130] 実装では、C/C++アプリケーションの利用依頼があると、まず、C/C++アプリケーションのコードを解析して、for文を発見するとともに、for文内で使われる変数データ等の、プログラム構造を把握する。構文解析には、LLVM/Clangの構文解析ライブラリ等を使用する。

[0131] 実装では、最初に、そのアプリケーションがＧＰＵオフロード効果があるかの見込みを得るため、ベンチマークを実行し、上記構文解析で把握したfor文のループ回数を把握する。ループ回数把握には、GNUカバレッジのgcov等を用いる。プロファイリングツールとしては、「GNUプロファイラ(gprof)」、「GNUカバレッジ(gcov)」が知られている。双方とも各行の実行回数を調査できるため、どちらを用いてもよい。実行回数は、例えば、１０００万回以上のループ回数を持つアプリケーションのみ対象とするようにできるが、この値は変更可能である。

[0132] ＣＰＵ向け汎用アプリケーションは、並列化を想定して実装されているわけではない。そのため、まず、ＧＰＵ処理自体が不可なfor文は排除する必要がある。そこで、各for文一つずつに対して、並列処理の#pragma acc kernels ディレクティブ挿入を試行し、コンパイル時にエラーが出るかの判定を行う。コンパイルエラーに関しては、幾つかの種類がある。for文の中で外部ルーチンが呼ばれている場合、ネストfor文で異なる階層が重複指定されている場合、break等でfor文を途中で抜ける処理がある場合、for文のデータにデータ依存性がある場合等がある。アプリケーションによって、コンパイル時エラーの種類は多彩であり、これ以外の場合もあるが、コンパイルエラーは処理対象外とし、#pragmaディレクティブは挿入しない。

[0133] コンパイルエラーは自動対処が難しく、また対処しても効果が出ないことも多い。外部ルーチンコールの場合は、#pragma acc routineにより回避できる場合があるが、多くの外部コールはライブラリであり、それを含めてＧＰＵ処理してもそのコールがネックとなり性能が出ない。for文一つずつを試行するため、ネストのエラーに関しては、コンパイルエラーは生じない。また

、break等によりで途中で抜ける場合は、並列処理にはループ回数を固定化する必要があり、プログラム改造が必要となる。データ依存が有る場合はそもそも並列処理自体ができない。

[0134] ここで、並列処理してもエラーが出ないループ文の数がａの場合、ａが遺伝子長となる。遺伝子の１は並列処理ディレクティブ有、０は無に対応させ、長さａの遺伝子に、アプリケーションコードをマッピングする。

[0135] 次に、初期値として，指定個体数の遺伝子配列を準備する。遺伝子の各値は、図５で説明したように、０と１をランダムに割当てて作成する。準備された遺伝子配列に応じて、遺伝子の値が１の場合はＧＰＵ処理を指定するディレクティブ ¥#pragma acc kernels, ¥#pragma acc parallel loop, ¥#pragma acc parallel loop vectorをC/C++コードに挿入する。single loop等はparallelにしない理由としては、同じ処理であればkernelsの方が、ＰＧＩコンパイラとしては性能が良いためである。この段階で、ある遺伝子に該当するコードの中で、ＧＰＵで処理させる部分が決まる。

[0136] 並列処理およびデータ転送のディレクティブを挿入されたC/C++コードを、ＧＰＵを備えたマシン上のＰＧＩコンパイラでコンパイルを行う。コンパイルした実行ファイルをデプロイし、ベンチマークツールで性能を測定する。

[0137] 全個体数に対して、ベンチマーク性能測定後、ベンチマーク処理時間に応じて、各遺伝子配列の適合度を設定する。設定された適合度に応じて、残す個体の選択を行う。選択された個体に対して、交叉処理、突然変異処理、そのままコピー処理のＧＡ処理を行い、次世代の個体群を作成する。

[0138] 次世代の個体に対して、ディレクティブ挿入、コンパイル、性能測定、適合度設定、選択、交叉、突然変異処理を行う。ここで、ＧＡ処理の中で、以前と同じパターンの遺伝子が生じた場合は、その個体についてはコンパイル、性能測定をせず、以前と同じ測定値を用いる。

[0139] 指定世代数のＧＡ処理終了後、最高性能の遺伝子配列に該当する、ディレクティブ付きC/C++コードを解とする。

[0140] この中で、個体数、世代数、交叉率、突然変異率、適合度設定、選択方法

は、ＧＡのパラメータであり、別途指定する。提案技術は、上記処理を自動化することで、従来、専門技術者の時間とスキルが必要だった、ＧＰＵオフロードの自動化を可能にする。

[0141] 図１１Ａ－Ｂは、上述した実装の動作概要を説明するフローチャートであり、図１１Ａと図１１Ｂは、結合子で繋がれる。

C/C++向けOpenＡＣＣコンパイラを用いて以下の処理を行う。

[0142] 〈コード解析〉

ステップＳ１０１で、アプリケーションコード分析部１１２（図３参照）は、C/C++アプリのコード解析を行う。

[0143] 〈ループ文特定〉

ステップＳ１０２で、並列処理指定部１１４（図３参照）は、C/C++アプリのループ文、参照関係を特定する。

[0144] 〈ループ文の並列処理可能性〉

ステップＳ１０３で、並列処理指定部１１４は、各ループ文のＧＰＵ処理可能性をチェックする（#pragma acc kernels）。

[0145] 〈ループ文の繰り返し〉

制御部（自動オフロード機能部）１１は、ステップＳ１０４のループ始端とステップＳ１１７のループ終端間で、ステップＳ１０５－Ｓ１１６の処理についてループ文の数だけ繰り返す。

[0146] 〈ループの数の繰り返し（その１）〉

制御部（自動オフロード機能部）１１は、ステップＳ１０５のループ始端とステップＳ１０８のループ終端間で、ステップＳ１０６－Ｓ１０７の処理についてループ文の数だけ繰り返す。

ステップＳ１０６で、並列処理指定部１１４は、各ループ文に対して、OpenACCでＧＰＵ処理（#pragma acc kernels）を指定してコンパイルする。

ステップＳ１０７で、並列処理指定部１１４は、エラー時は、次の指示句でＧＰＵ処理可能性をチェックする（#pragma acc parallel loop）。

[0147] 〈ループの数の繰り返し（その２）〉

制御部（自動オフロード機能部）１１は、ステップＳ１０９のループ始端とステップＳ１１２のループ終端間で、ステップＳ１１０－Ｓ１１１の処理についてループ文の数だけ繰り返す。

ステップＳ１１０で、並列処理指定部１１４は、各ループ文に対して、OpenACCでＧＰＵ処理（#pragma acc parallel loop）を指定してコンパイルする。

ステップＳ１１１で、並列処理指定部１１４は、エラー時は、次の指示句でＧＰＵ処理可能性をチェックする（#pragma acc parallel loop vector）。

[0148] 〈ループの数の繰り返し（その３）〉

制御部（自動オフロード機能部）１１は、ステップＳ１１３のループ始端とステップＳ１１６のループ終端間で、ステップＳ１１４－Ｓ１１５の処理についてループ文の数だけ繰り返す。

ステップＳ１１４で、並列処理指定部１１４は、各ループ文に対して、OpenACCでＧＰＵ処理（#pragma acc parallel loop vector）を指定してコンパイルする。

ステップＳ１１５で、並列処理指定部１１４は、エラー時は、当該ループ文からはＧＰＵ処理指示句を除去する。

[0149] 〈for文の数カウント〉

ステップＳ１１８で、並列処理指定部１１４は、コンパイルエラーが出ないfor文の数をカウントし、遺伝子長とする。

[0150] 〈指定個体数パターン準備〉

次に、初期値として、並列処理指定部１１４は、指定個体数の遺伝子配列を準備する。ここでは、０と１をランダムに割当てて作成する。

ステップＳ１１９で、並列処理指定部１１４は、C/C++アプリコードを、遺伝子にマッピングし、指定個体数パターン準備を行う。

準備された遺伝子配列に応じて、遺伝子の値が１の場合は並列処理を指定するディレクティブをC/C++コードに挿入する（例えば図５の#pragmaディレ

クティブ参照）。

[0151] 制御部（自動オフロード機能部）１１は、ステップＳ１２０のループ始端とステップＳ１２９のループ終端間で、ステップＳ１２１－Ｓ１２８の処理について指定世代数繰り返す。

また、上記指定世代数繰り返しにおいて、さらにステップＳ１２１のループ始端とステップＳ１２６のループ終端間で、ステップＳ１２２－Ｓ１２５の処理について指定個体数繰り返す。すなわち、指定世代数繰り返しの中で、指定個体数の繰り返しが入れ子状態で処理される。

[0152] 〈データ転送指定〉

ステップＳ１２２で、データ転送指定部１１３は、変数参照関係をもとに、明示的指示行（#pragma acc data copy/copyin/copyout/presentおよび#pragam acc declarecreate, #pragma acc update）を用いたデータ転送指定を行う。

[0153] 〈コンパイル〉

ステップＳ１２３で、並列処理パターン作成部１１５（図２参照）は、遺伝子パターンに応じてディレクティブ指定したC/C++コードをＰＧＩコンパイラでコンパイルする。すなわち、並列処理パターン作成部１１５は、作成したC/C++コードを、ＧＰＵを備えた検証用マシン１４上のＰＧＩコンパイラでコンパイルを行う。

ここで、ネストfor文を複数並列指定する場合等でコンパイルエラーとなることがある。この場合は、性能測定時の処理時間がタイムアウトした場合と同様に扱う。

[0154] ステップＳ１２４で、性能測定部１１６（図２参照）は、ＣＰＵ-ＧＰＵ搭載の検証用マシン１４に、実行ファイルをデプロイする。

ステップＳ１２５で、性能測定部１１６は、配置したバイナリファイルを実行し、オフロードした際のベンチマーク性能を測定する。

[0155] ここで、途中世代で、以前と同じパターンの遺伝子については測定せず、同じ値を使う。つまり、ＧＡ処理の中で、以前と同じパターンの遺伝子が生

じた場合は、その個体についてはコンパイルや性能測定をせず、以前と同じ測定値を用いる。

ステップＳ１２７で、実行ファイル作成部１１７（図２参照）は、処理時間が短い個体ほど適合度が高くなるように評価し、性能の高い個体を選択する。

[0156] ステップＳ１２８で、実行ファイル作成部１１７は、選択された個体に対して、交叉、突然変異の処理を行い、次世代の個体を作成する。実行ファイル作成部１１７は、次世代の個体に対して、コンパイル、性能測定、適合度設定、選択、交叉、突然変異処理を行う。

すなわち、全個体に対して、ベンチマーク性能測定後、ベンチマーク処理時間に応じて、各遺伝子配列の適合度を設定する。設定された適合度に応じて、残す個体の選択を行う。実行ファイル作成部１１７は、選択された個体に対して、交叉処理、突然変異処理、そのままコピー処理のＧＡ処理を行い、次世代の個体群を作成する。

[0157] ステップＳ１３０で、実行ファイル作成部１１７は、指定世代数のＧＡ処理終了後、最高性能の遺伝子配列に該当するC/C++コード（最高性能の並列処理パターン）を解とする。

[0158] 〈ＧＡのパラメータ〉

上記、個体数、世代数、交叉率、突然変異率、適合度設定、選択方法は、ＧＡのパラメータである。ＧＡのパラメータは、例えば、以下のように設定してもよい。

実行するSimple GAの、パラメータ、条件は例えば以下のようにできる。

遺伝子長：並列可能ループ文数

個体数Ｍ：遺伝子長以下

世代数Ｔ：遺伝子長以下

適合度：(処理時間)$^{(-1/2)}$

この設定により、ベンチマーク処理時間が短い程、高適合度になる。また、適合度を、処理時間の(-1/2)乗とすることで、処理時間が短い特定の個体

の適合度が高くなり過ぎて、探索範囲が狭くなるのを防ぐことができる。また、性能測定が一定時間で終わらない場合は、タイムアウトさせ、処理時間１０００秒等の時間（長時間）であるとして、適合度を計算する。このタイムアウト時間は、性能測定特性に応じて変更させればよい。

選択：ルーレット選択

ただし、世代での最高適合度遺伝子は交叉も突然変異もせず次世代に保存するエリート保存も合わせて行う。

交叉率Ｐｃ：０．９

突然変異率Ｐｍ：０．０５

[0159] 〈コストパフォーマンス〉

自動オフロード機能のコストパフォーマンスについて述べる。

NVIDIA Tesla等の、ＧＰＵボードのハードウェアの価格だけを見ると、ＧＰＵを搭載したマシンの価格は、通常のＣＰＵのみのマシンの約２倍となる。しかし、一般にデータセンタ等のコストでは、ハードウェアやシステム開発のコストが１／３以下であり、電気代や保守・運用体制等の運用費が１／３超であり、サービスオーダ等のその他費用が１／３程度である。本実施形態では、暗号処理や画像処理等動作させるアプリケーションで時間がかかる処理を２倍以上高性能化できる。このため、サーバハードウェア価格自体は２倍となっても、コスト効果が十分に期待できる。

[0160] 本実施形態では、gcov，gprof等を用いて、ループが多く実行時間がかかっているアプリケーションを事前に特定して、オフロード試行をする。これにより、効率的に高速化できるアプリケーションを見つけることができる。

[0161] 〈本番サービス利用開始までの時間〉

本番サービス利用開始までの時間について述べる。

コンパイルから性能測定1回は３分程度とすると、２０の個体数、２０の世代数のＧＡで最大２０時間程度解探索にかかるが、以前と同じ遺伝子パターンのコンパイル、測定は省略されるため、８時間以下で終了する。多くのクラウドやホスティング、ネットワークサービスではサービス利用開始に半日

程度かかるのが実情である。本実施形態では、例えば半日以内の自動オフロードが可能である。このため、半日以内の自動オフロードであれば、最初は試し利用ができるとすれば、ユーザ満足度を十分に高めることが期待できる。

[0162] より短時間でオフロード部分を探索するためには、複数の検証用マシンにより個体数分並列で性能測定することが考えられる。アプリケーションに応じて、タイムアウト時間を調整することも短時間化に繋がる。例えば、オフロード処理がＣＰＵでの実行時間の２倍かかる場合はタイムアウトとする等である。また、個体数、世代数が多い方が、高性能な解を発見できる可能性が高まる。しかし、各パラメータを最大にする場合、個体数×世代数だけコンパイル、および性能ベンチマークを行う必要がある。このため、本番サービス利用開始までの時間がかかる。本実施形態では、ＧＡとしては少ない個体数、世代数で行っているが、交叉率Ｐｃを０．９と高い値にして広範囲を探索することで、ある程度の性能の解を早く発見するようにしている。

[0163] ［指示句の拡大］

本実施形態では、適用できるアプリケーション増加のため、指示句の拡大を行う。具体的には、ＧＰＵ処理を指定する指示句として、kernels指示句に加えて，parallel loop指示句、parallel loop vector指示句にも拡大する。

OpenACC標準では、kernelsは、single loopやtightly nested loopに使われる。また、parallel loopは、non-tightly nested loopも含めたループに使われる。parallel loop vectorは、parallelizeはできないがvectorizeはできるループに使われる。ここで、tightly nested loopとは、ネストループにて、例えば、ｉとjをインクリメントする二つのループが入れ子になっている時、下位のループでｉとｊを使った処理がされ、上位ではされないような単純なループである。また、ＰＧＩコンパイラ等の実装においては、kernelsは、並列化の判断はコンパイラが行い、parallelは並列化の判断はプログラマが行うという違いがある。なお、非特許文献２では、単純なループを対象にしており、non-tightly nested loopやparallelizeできないループ等kerne

lsではエラーになるループ文は対象外であったため、適用範囲が狭かった。

[0164] そこで、本実施形態では、single、tightly nested loopにはkernelsを使い、non-tightly nested loopにはparallel loopを使う。また、parallelizeできないがvectorizeできるループにはparallel loop vectorを使う。

ここで、parallel指示句にすることで、結果がkernelsの場合より信頼度が下がる懸念がある。しかし、最終的なオフロードプログラムに対して、サンプルテストを行い、ＣＰＵとの結果差分をチェックしその結果をユーザに見せて、ユーザに確認してもらうことを想定している。そもそも、ＣＰＵとＧＰＵではハードが異なるため，有効数字桁数や丸め誤差の違い等があり、kernelsだけでもＣＰＵとの結果差分のチェックは必要である。

[0165] ［ハードウェア構成］

本実施形態に係るオフロードサーバ１は、例えば図１２に示すような構成のコンピュータ９００によって実現される。

図１２は、オフロードサーバ１の機能を実現するコンピュータ９００の一例を示すハードウェア構成図である。

コンピュータ９００は、ＣＰＵ９１０、ＲＡＭ９２０、ＲＯＭ９３０、ＨＤＤ９４０、通信インタフェース（Ｉ／Ｆ：Interface）９５０、入出力インタフェース（Ｉ／Ｆ）９６０、およびメディアインタフェース（Ｉ／Ｆ）９７０を有する。

[0166] ＣＰＵ９１０は、ＲＯＭ９３０またはＨＤＤ９４０に格納されたプログラムに基づいて動作し、各部の制御を行う。ＲＯＭ９３０は、コンピュータ９００の起動時にＣＰＵ９１０によって実行されるブートプログラムや、コンピュータ９００のハードウェアに依存するプログラム等を格納する。

[0167] ＨＤＤ９４０は、ＣＰＵ９１０によって実行されるプログラム、および、かかるプログラムによって使用されるデータ等を格納する。通信インタフェース９５０は、通信網８０を介して他の機器からデータを受信してＣＰＵ９１０へ送り、ＣＰＵ９１０が生成したデータを通信網８０を介して他の機器へ送信する。

[0168] ＣＰＵ９１０は、入出力インタフェース９６０を介して、ディスプレイやプリンタ等の出力装置、および、キーボードやマウス等の入力装置を制御する。ＣＰＵ９１０は、入出力インタフェース９６０を介して、入力装置からデータを取得する。また、ＣＰＵ９１０は、生成したデータを入出力インタフェース９６０を介して出力装置へ出力する。

[0169] メディアインタフェース９７０は、記録媒体９８０に格納されたプログラムまたはデータを読み取り、ＲＡＭ９２０を介してＣＰＵ９１０に提供する。ＣＰＵ９１０は、かかるプログラムを、メディアインタフェース９７０を介して記録媒体９８０からＲＡＭ９２０上にロードし、ロードしたプログラムを実行する。記録媒体９８０は、例えばＤＶＤ（Digital Versatile Disc）、ＰＤ（Phasechangerewritable Disk）等の光学記録媒体、ＭＯ（Magneto Optical disk）等の光磁気記録媒体、テープ媒体、磁気記録媒体、または半導体メモリ等である。

[0170] 例えば、コンピュータ９００が本実施形態に係るオフロードサーバ１として機能する場合、コンピュータ９００のＣＰＵ９１０は、ＲＡＭ９２０上にロードされたプログラムを実行することにより、オフロードサーバ１の各部の機能を実現する。また、ＨＤＤ９４０には、オフロードサーバ１の各部内のデータが格納される。コンピュータ９００のＣＰＵ９１０は、これらのプログラムを記録媒体９８０から読み取って実行するが、他の例として、他の装置から通信網８０を介してこれらのプログラムを取得してもよい。

[0171] ［効果］

以上説明したように、本実施形態に係るオフロードサーバ１は、アプリケーションのソースコードを分析するアプリケーションコード分析部１１２と、コード分析の結果をもとに、ＣＰＵとＧＰＵ間の転送が必要な変数の中で、ＣＰＵ処理とＧＰＵ処理とが相互に参照または更新がされず、ＧＰＵ処理した結果をＣＰＵに返すだけの変数については、ＧＰＵ処理の開始前と終了後に一括化してデータ転送する指定を行うデータ転送指定部１１３と、アプリケーションのループ文を特定し、特定した各ループ文に対して、ＧＰＵに

おける並列処理指定文を指定してコンパイルする並列処理指定部１１４と、コンパイルエラーが出るループ文に対して、オフロード対象外とするとともに、コンパイルエラーが出ないループ文に対して、並列処理するかしないかの指定を行う並列処理パターンを作成する並列処理パターン作成部１１５と、並列処理パターンのアプリケーションをコンパイルして、アクセラレータ検証用装置に配置し、アクセラレータにオフロードした際の性能測定用処理を実行する性能測定部１１６と、性能測定結果をもとに、複数の並列処理パターンから最高処理性能の並列処理パターンを選択し、最高処理性能の並列処理パターンをコンパイルして実行ファイルを作成する実行ファイル作成部１１７と、を備える。

[0172] このようにすることにより、プログラムに分散して存在するＧＰＵへの指示内容（data copy等）を個別にＧＰＵに転送するのではなく、一括転送できる変数をまとめて一括で転送・指示を行うことで、ＣＰＵ-ＧＰＵ間の転送を削減して、オフロードのさらなる高速化を図ることができる。

[0173] オフロードサーバ１は、データ転送指定部１１３が、ＧＰＵ処理の開始前と終了後に一括化してデータ転送する指定を、OpenACCのdata copyを用いて指定する。

[0174] このようにすることにより、一括転送できる変数をまとめて一括で転送・指示を行うことで、ＣＰＵ-ＧＰＵ間の転送を削減して、オフロードのさらなる高速化を図ることができる。

[0175] オフロードサーバ１は、データ転送指定部１１３が、ＧＰＵで処理すべき変数が、既にＧＰＵ側に一括転送されている場合に、転送不要である指示句を追加する。

[0176] このようにすることにより、本来はＣＰＵ－ＧＰＵ間の転送が不要であるにもかかわらず転送されてしまう事象ついて、転送不要である指示句を追加指定することで、転送を防止することができる。その結果、ＣＰＵ-ＧＰＵ間の不要な転送を防止して、オフロードのさらなる高速化を図ることができる。

[0177] オフロードサーバ１は、データ転送指定部１１３が、ＧＰＵ処理の始まる前に一括化して転送され、かつループ文処理のタイミングで転送が不要な変数についてはOpenACCのdata presentを用いて転送不要であることを明示する。

[0178] このようにすることにより、OpenACCのdata presentを用いて、ＣＰＵ-ＧＰＵの不要な転送を防止して、オフロードのさらなる高速化を図ることができる。

[0179] オフロードサーバ１は、データ転送指定部１１３が、ＣＰＵとＧＰＵ間のデータ転送時に、ＧＰＵ側で一時領域を作成し（#pragma acc declare create）、データを一時領域に格納後、当該一時領域を同期（#pragma acc update）することで変数転送を指示する。

[0180] このようにすることにより、一時領域を作成して、どのタイミングでＣＰＵ－ＧＰＵ間転送をするかを指定することで、コンパイラ依存で行われる自動転送を防止して、オフロードのさらなる高速化を図ることができる。

[0181] オフロードサーバ１において、並列処理指定部１１４は、遺伝的アルゴリズムに基づき、コンパイルエラーが出ないループ文の数を遺伝子長とし、並列処理パターン作成部１１５は、ＧＰＵ処理をする場合を１または０のいずれか一方、しない場合を他方の０または１として、アクセラレータ処理可否を遺伝子パターンにマッピングし、遺伝子の各値を１か０にランダムに作成した指定個体数の遺伝子パターンを準備し、性能測定部１１６は、各個体に応じて、ＧＰＵにおける並列処理指定文を指定したアプリケーションコードをコンパイルして、アクセラレータ検証用装置１４に配置し、アクセラレータ検証用装置において性能測定用処理を実行し、実行ファイル作成部１１７は、各個体に対して、性能測定を行い、処理時間の短い個体ほど適合度が高くなるように評価し、各個体から、適合度が所定値より高いものを性能の高い個体として選択し、選択された個体に対して、交叉、突然変異の処理を行い、次世代の個体を作成し、指定世代数の処理終了後、最高性能の並列処理パターンを解として選択する。

[0182] このようにすることにより、最初に並列可能なループ文のチェックを行い、次に並列可能繰り返し文群に対してＧＡを用いて検証環境で性能検証試行を反復し適切な領域を探索する。並列可能なループ文（例えばfor文）に絞った上で、遺伝子の部分の形で、高速化可能な並列処理パターンを保持し組み換えていくことで、取り得る膨大な並列処理パターンから、効率的に高速化可能なパターンを探索できる。

[0183] 本発明は、コンピュータを、上記オフロードサーバとして機能させるためのオフロードプログラムとした。

[0184] このようにすることにより、一般的なコンピュータを用いて、上記オフロードサーバ１の各機能を実現させることができる。

[0185] また、上記実施形態において説明した各処理のうち、自動的に行われるものとして説明した処理の全部又は一部を手作業で行うこともでき、あるいは、手作業で行われるものとして説明した処理の全部又は一部を公知の方法で自動的に行うこともできる。この他、上述文書中や図面中に示した処理手順、制御手順、具体的名称、各種のデータやパラメータを含む情報については、特記する場合を除いて任意に変更することができる。

また、図示した各装置の各構成要素は機能概念的なものであり、必ずしも物理的に図示の如く構成されていることを要しない。すなわち、各装置の分散・統合の具体的形態は図示のものに限られず、その全部又は一部を、各種の負荷や使用状況などに応じて、任意の単位で機能的又は物理的に分散・統合して構成することができる。

[0186] また、上記の各構成、機能、処理部、処理手段等は、それらの一部又は全部を、例えば集積回路で設計する等によりハードウェアで実現してもよい。また、上記の各構成、機能等は、プロセッサがそれぞれの機能を実現するプログラムを解釈し、実行するためのソフトウェアで実現してもよい。各機能を実現するプログラム、テーブル、ファイル等の情報は、メモリや、ハードディスク、ＳＳＤ（Solid State Drive）等の記録装置、又は、ＩＣ（Integrated Circuit）カード、ＳＤ（Secure Digital）カード、光ディスク等の記

録媒体に保持することができる。

[0187] また、本実施形態では、組合せ最適化問題を、限られた最適化期間中に解を発見できるようにするため、遺伝的アルゴリズム（ＧＡ）の手法を用いているが、最適化の手法はどのようなものでもよい。例えば、local search（局所探索法）、Dynamic Programming（動的計画法）、これらの組み合わせでもよい。

[0188] また、本実施形態では、C/C++向けOpenＡＣＣコンパイラを用いているが、ＧＰＵ処理をオフロードできるものであればどのようなものでもよい。例えば、Java lambda（登録商標） GPU処理、IBM Java 9 SDK（登録商標）でもよい。なお、並列処理指定文は、これらの開発環境に依存する。

例えば、Java（登録商標）では、Java 8よりlambda形式での並列処理記述が可能である。IBM（登録商標）は、lambda形式の並列処理記述を、ＧＰＵにオフロードするJITコンパイラを提供している。Javaでは、これらを用いて、ループ処理をlambda形式にするか否かのチューニングをＧＡで行うことで、同様のオフロードが可能である。

[0189] また、本実施形態では、繰り返し文（ループ文）として、for文を例示したが、for文以外のwhile文やdo-while文も含まれる。ただし、ループの継続条件等を指定するfor文がより適している。

符号の説明

[0190] １　オフロードサーバ

１１　制御部

１２　入出力部

１３　記憶部

１４　検証用マシン（アクセラレータ検証用装置）

１５　OpenｌｏＴリソース

１１１　アプリケーションコード指定部

１１２　アプリケーションコード分析部

１１３　データ転送指定部

１１４　並列処理指定部

１１４ａ　オフロード範囲抽出部

１１４ｂ　中間言語ファイル出力部

１１５　並列処理パターン作成部

１１６　性能測定部

１１６ａ　バイナリファイル配置部

１１７　実行ファイル作成部

１１８　本番環境配置部

１１９　性能測定テスト抽出実行部

１２０　ユーザ提供部

１３０　アプリケーションコード

１３１　テストケースＤＢ

１３２　中間言語ファイル

１５１　各種デバイス

１５２　ＣＰＵ-ＧＰＵを有する装置

１５３　ＣＰＵ-ＦＰＧＡを有する装置

１５４　ＣＰＵを有する装置

請求の範囲

[請求項1] アプリケーションの特定処理をＧＰＵ（Graphics Processing Unit）にオフロードするオフロードサーバであって、

アプリケーションのソースコードを分析するアプリケーションコード分析部と、

コード分析の結果をもとに、ＣＰＵ（Central Processing Unit）と前記ＧＰＵ間の転送が必要な変数の中で、ＣＰＵ処理とＧＰＵ処理とが相互に参照または更新がされず、前記ＧＰＵ処理した結果を前記ＣＰＵに返すだけの変数については、前記ＧＰＵ処理の開始前と終了後に一括化してデータ転送する指定を行うデータ転送指定部と、

前記アプリケーションのループ文を特定し、特定した各前記ループ文に対して、前記ＧＰＵにおける並列処理指定文を指定してコンパイルする並列処理指定部と、

コンパイルエラーが出るループ文に対して、オフロード対象外とするとともに、コンパイルエラーが出ないループ文に対して、並列処理するかしないかの指定を行う並列処理パターンを作成する並列処理パターン作成部と、

前記並列処理パターンの前記アプリケーションをコンパイルして、アクセラレータ検証用装置に配置し、前記ＧＰＵにオフロードした際の性能測定用処理を実行する性能測定部と、

性能測定結果をもとに、複数の前記並列処理パターンから最高処理性能の並列処理パターンを選択し、最高処理性能の前記並列処理パターンをコンパイルして実行ファイルを作成する実行ファイル作成部と、

を備えることを特徴とするオフロードサーバ。

[請求項2] 前記データ転送指定部は、前記ＧＰＵ処理の開始前と終了後に一括化してデータ転送する指定を、OpenACC（Open Accelerator）のdata copyを用いて指定する

ことを特徴とする請求項１に記載のオフロードサーバ。

[請求項3] 前記データ転送指定部は、前記ＧＰＵで処理すべき変数が、既に前記ＧＰＵ側に一括転送されている場合に、転送不要である指示句を追加する

ことを特徴とする請求項１または請求項２に記載のオフロードサーバ。

[請求項4] 前記データ転送指定部は、前記ＧＰＵ処理の始まる前に一括化して転送され、かつループ文処理のタイミングで転送が不要な変数についてはOpenACCのdata presentを用いて転送不要であることを明示する

ことを特徴とする請求項１に記載のオフロードサーバ。

[請求項5] 前記データ転送指定部は、前記ＣＰＵと前記ＧＰＵ間のデータ転送時に、ＧＰＵ側で一時領域を作成し、データを前記一時領域に格納後、当該一時領域を同期することで変数転送を指示する

ことを特徴とする請求項１に記載のオフロードサーバ。

[請求項6] 前記並列処理指定部は、遺伝的アルゴリズムに基づき、

コンパイルエラーが出ないループ文の数を遺伝子長とし、

前記並列処理パターン作成部は、ＧＰＵ処理をする場合を１または０のいずれか一方、しない場合を他方の０または１として、ＧＰＵ処理可否を遺伝子パターンにマッピングし、

前記遺伝子の各値を１か０にランダムに作成した指定個体数の前記遺伝子パターンを準備し、

前記性能測定部は、各個体に応じて、前記ＧＰＵにおける並列処理指定文を指定したアプリケーションコードをコンパイルして、前記アクセラレータ検証用装置に配置し、

前記アクセラレータ検証用装置において性能測定用処理を実行し、

前記実行ファイル作成部は、各個体に対して、性能測定を行い、処理時間の短い個体ほど適合度が高くなるように評価し、

各個体から、前記適合度が所定値より高いものを性能の高い個体と

して選択し、

選択された個体に対して、交叉、突然変異の処理を行い、次世代の個体を作成し、

指定世代数の処理終了後、最高性能の前記並列処理パターンを解として選択する

ことを特徴とする請求項１に記載のオフロードサーバ。

[請求項7] アプリケーションの特定処理をＧＰＵ（Graphics Processing Unit）にオフロードするオフロードサーバのオフロード制御方法であって、

前記オフロードサーバは、

アプリケーションのソースコードを分析するステップと、

コード分析の結果をもとに、ＣＰＵ（Central Processing Unit）と前記ＧＰＵ間の転送が必要な変数の中で、ＣＰＵ処理とＧＰＵ処理とが相互に参照または更新がされず、前記ＧＰＵ処理した結果を前記ＣＰＵに返すだけの変数については、前記ＧＰＵ処理の開始前と終了後に一括化してデータ転送する指定を行うステップと、

前記アプリケーションのループ文を特定し、特定した各前記ループ文に対して、前記ＧＰＵにおける並列処理指定文を指定してコンパイルするステップと、

コンパイルエラーが出るループ文に対して、オフロード対象外とするとともに、コンパイルエラーが出ないループ文に対して、並列処理するかしないかの指定を行う並列処理パターンを作成するステップと、

前記並列処理パターンの前記アプリケーションをコンパイルして、アクセラレータ検証用装置に配置し、前記ＧＰＵにオフロードした際の性能測定用処理を実行するステップと、

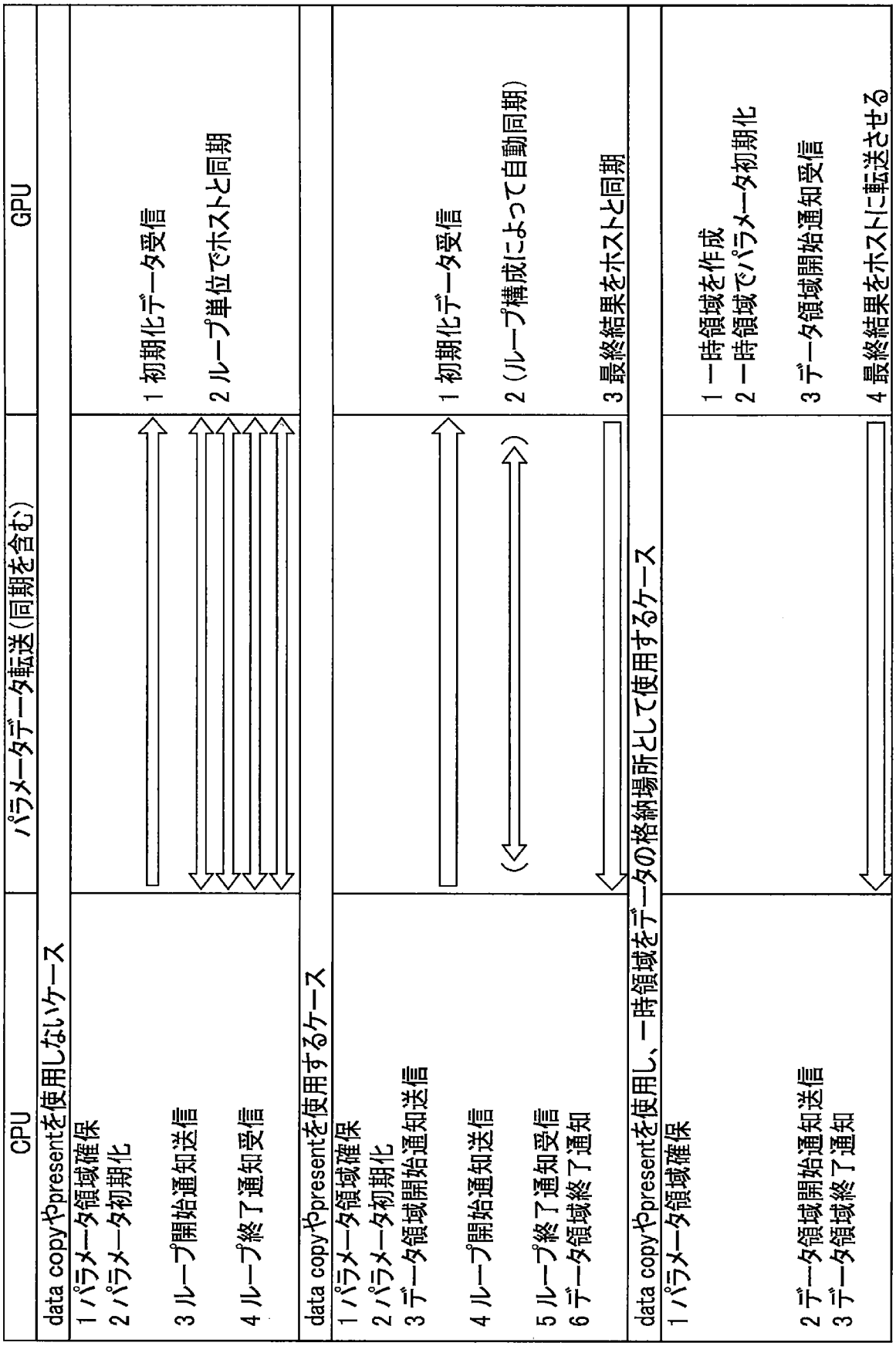
性能測定結果をもとに、複数の前記並列処理パターンから最高処理性能の並列処理パターンを選択し、最高処理性能の前記並列処理パタ

ーンをコンパイルして実行ファイルを作成するステップと、を実行する

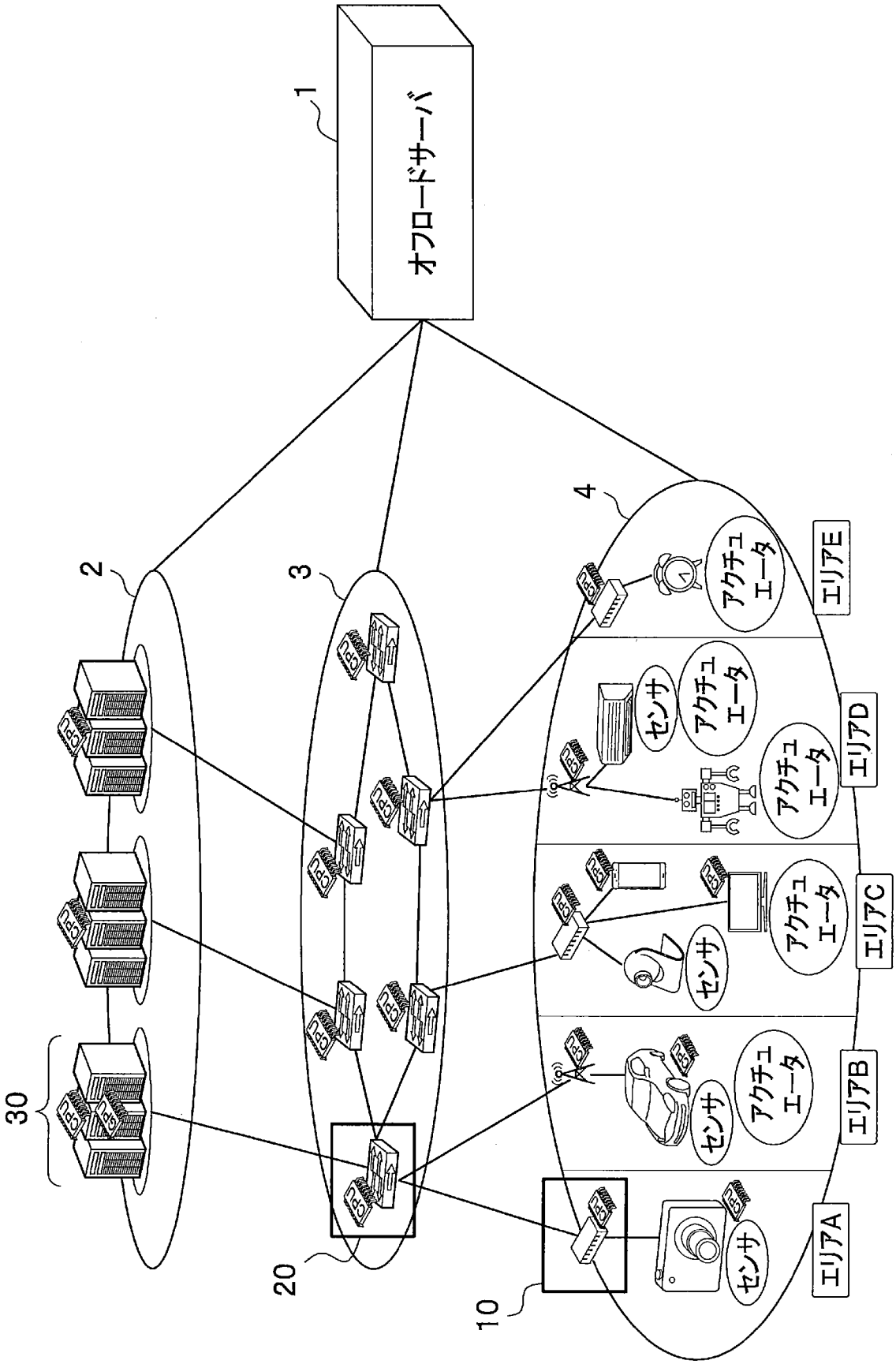
ことを特徴とするオフロード制御方法。

[請求項8] コンピュータを、請求項１乃至請求項６のいずれか１項に記載のオフロードサーバとして機能させるためのオフロードプログラム。

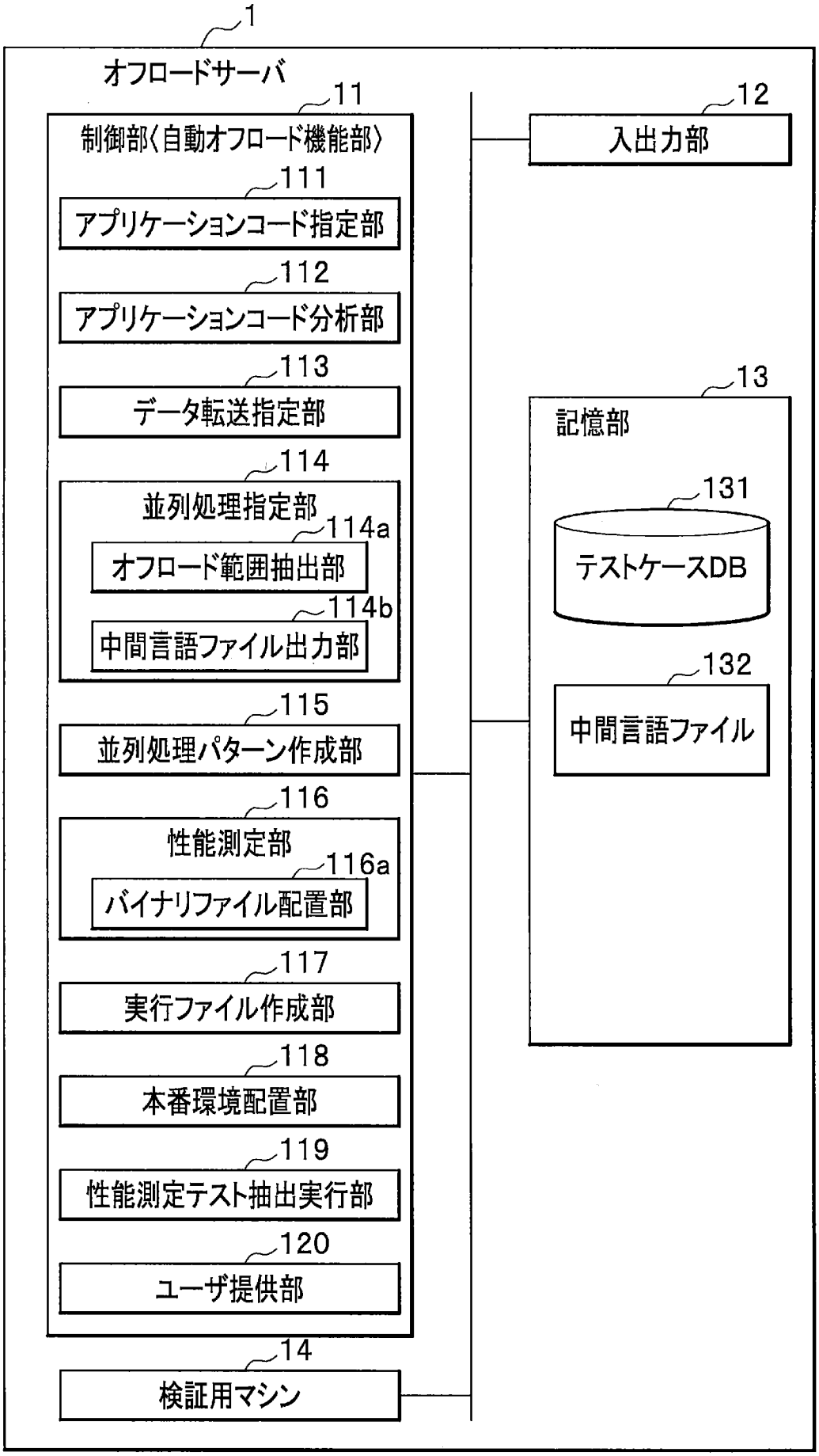
[図1]


CPU
パラメータデータ転送(同期を含む)
GPU
data copyやpresentを使用しないケース
1 パラメータ領域確保
2 パラメータ初期化
3 ループ開始通知送信
4 ループ終了通知受信
1 初期化データ受信
2 ループ単位でホストと同期
data copyやpresentを使用するケース
1 パラメータ領域確保
2 パラメータ初期化
3 データ領域開始通知送信
4 ループ開始通知送信
5 ループ終了通知受信
6 データ領域終了通知
1 初期化データ受信
2 (ループ構成によって自動同期)
3 最終結果をホストと同期
data copyやpresentを使用し、一時領域をデータの格納場所として使用するケース
1 パラメータ領域確保
2 データ領域開始通知送信
3 データ領域終了通知
1 一時領域を作成
2 一時領域でパラメータ初期化
3 データ領域開始通知受信
4 最終結果をホストに転送させる

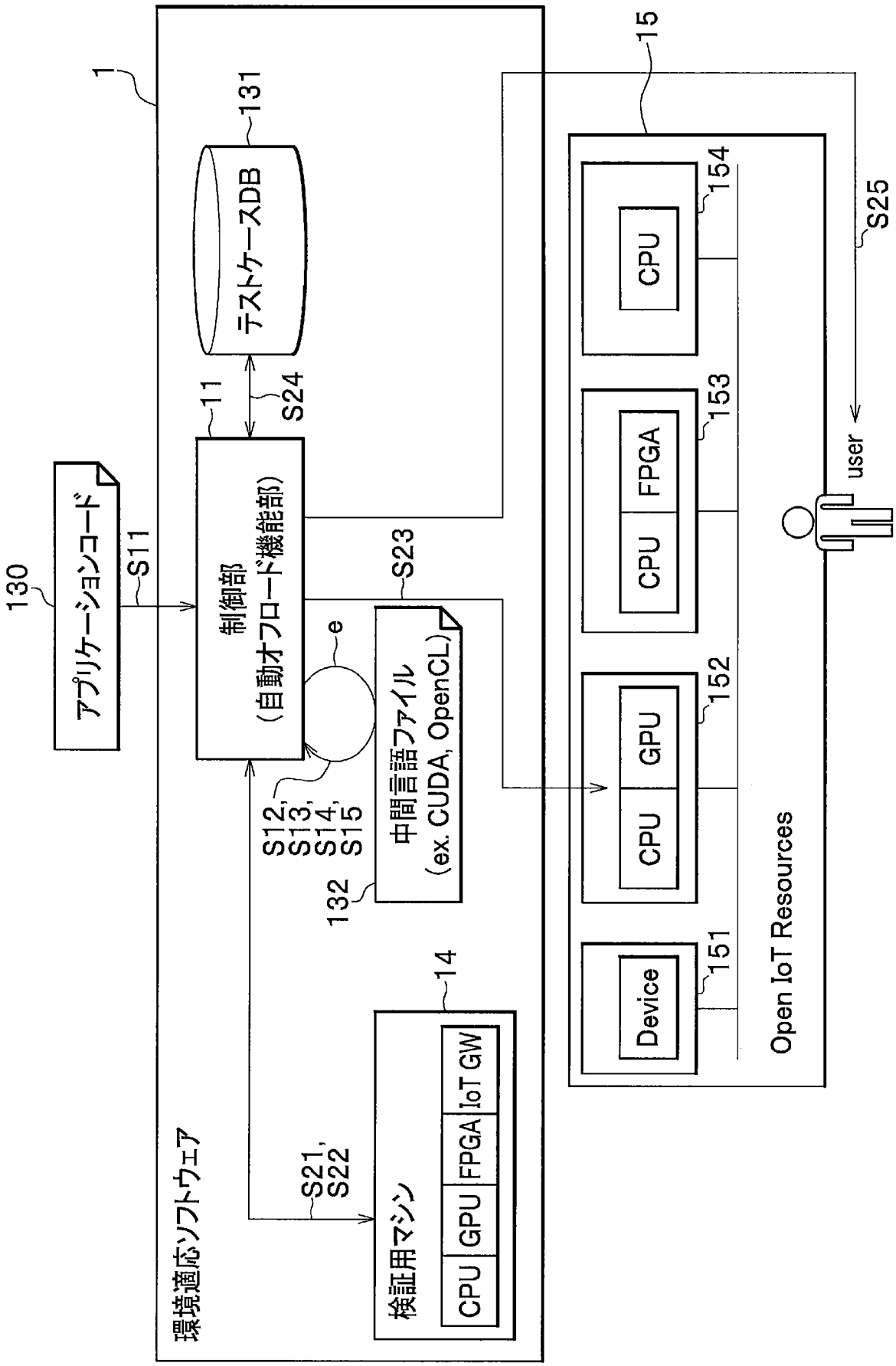
[図2]



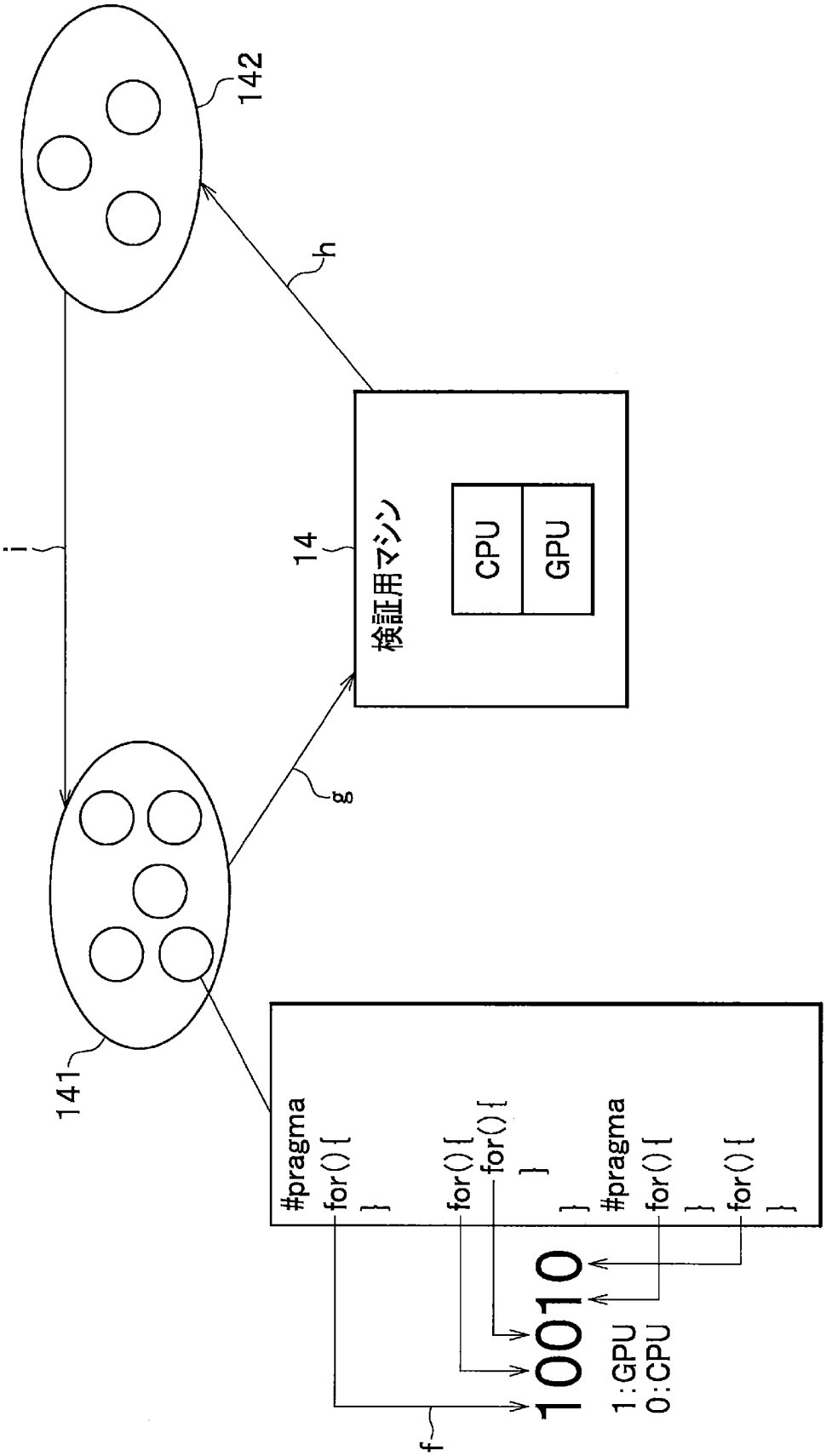
[図3]


1
オフロードサーバ
11
制御部〈自動オフロード機能部〉
111
アプリケーションコード指定部
112
アプリケーションコード分析部
113
データ転送指定部
114
並列処理指定部
114a
オフロード範囲抽出部
114b
中間言語ファイル出力部
115
並列処理パターン作成部
116
性能測定部
116a
バイナリファイル配置部
117
実行ファイル作成部
118
本番環境配置部
119
性能測定テスト抽出実行部
120
ユーザ提供部
14
検証用マシン
12
入出力部
13
記憶部
131
テストケースDB
132
中間言語ファイル

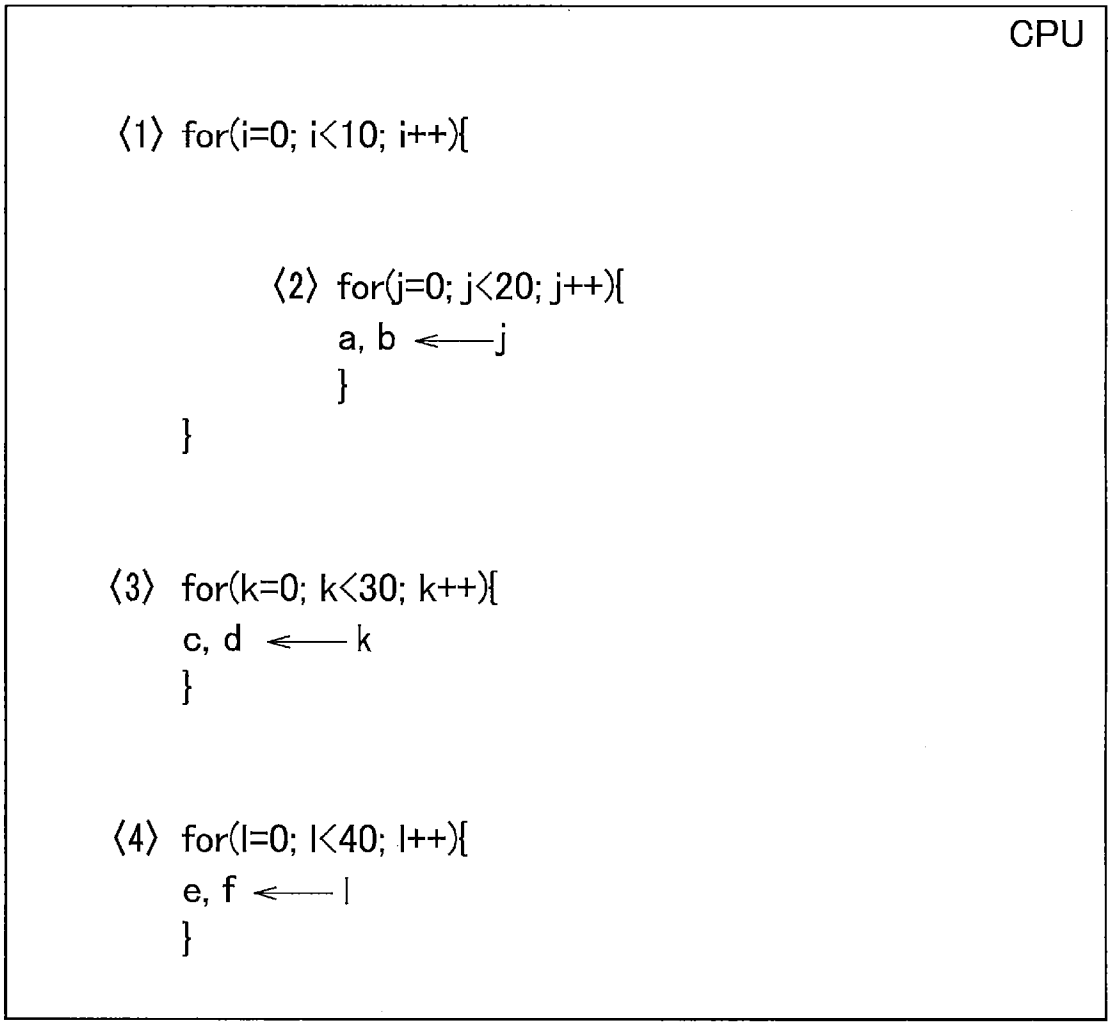
[図4]


1
131
テストケースDB
11
S24
15
154
CPU
S25
153
CPU
FPGA
user
130
アプリケーションコード
S11
制御部
(自動オフロード機能部)
S23
e
中間言語ファイル
(ex. CUDA, OpenCL)
S12,
S13,
S14,
S15
132
152
CPU
GPU
151
Device
Open IoT Resources
14
検証用マシン
CPU
GPU
FPGA
IoT GW
S21,
S22
環境適応ソフトウェア

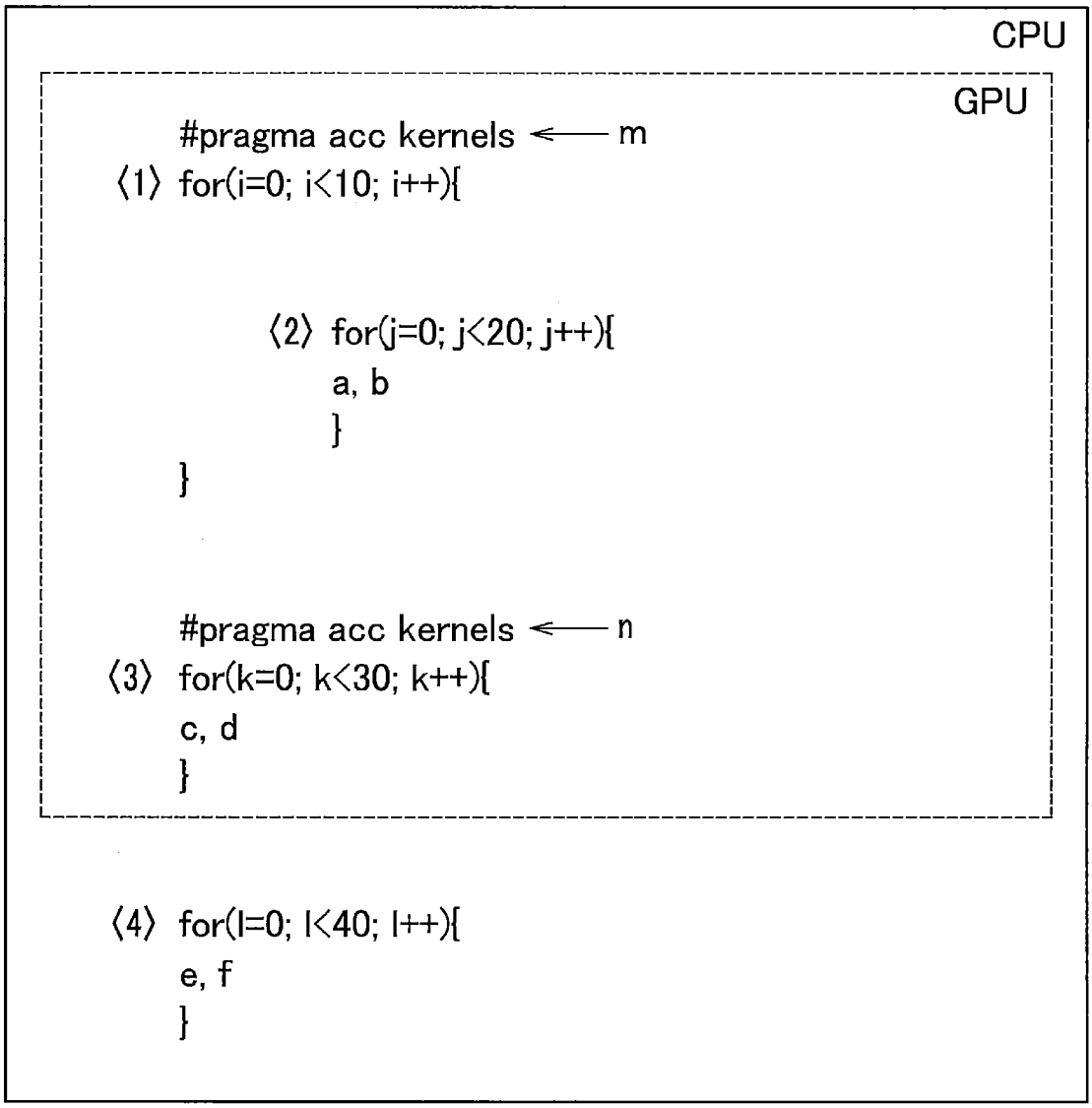
[図5]


142
h
i
14
検証用マシン
CPU
GPU
g
141
#pragma
for(){
}
for(){
for(){
}
}
#pragma
for(){
}
for(){
}
10010
1:GPU
0:CPU
f

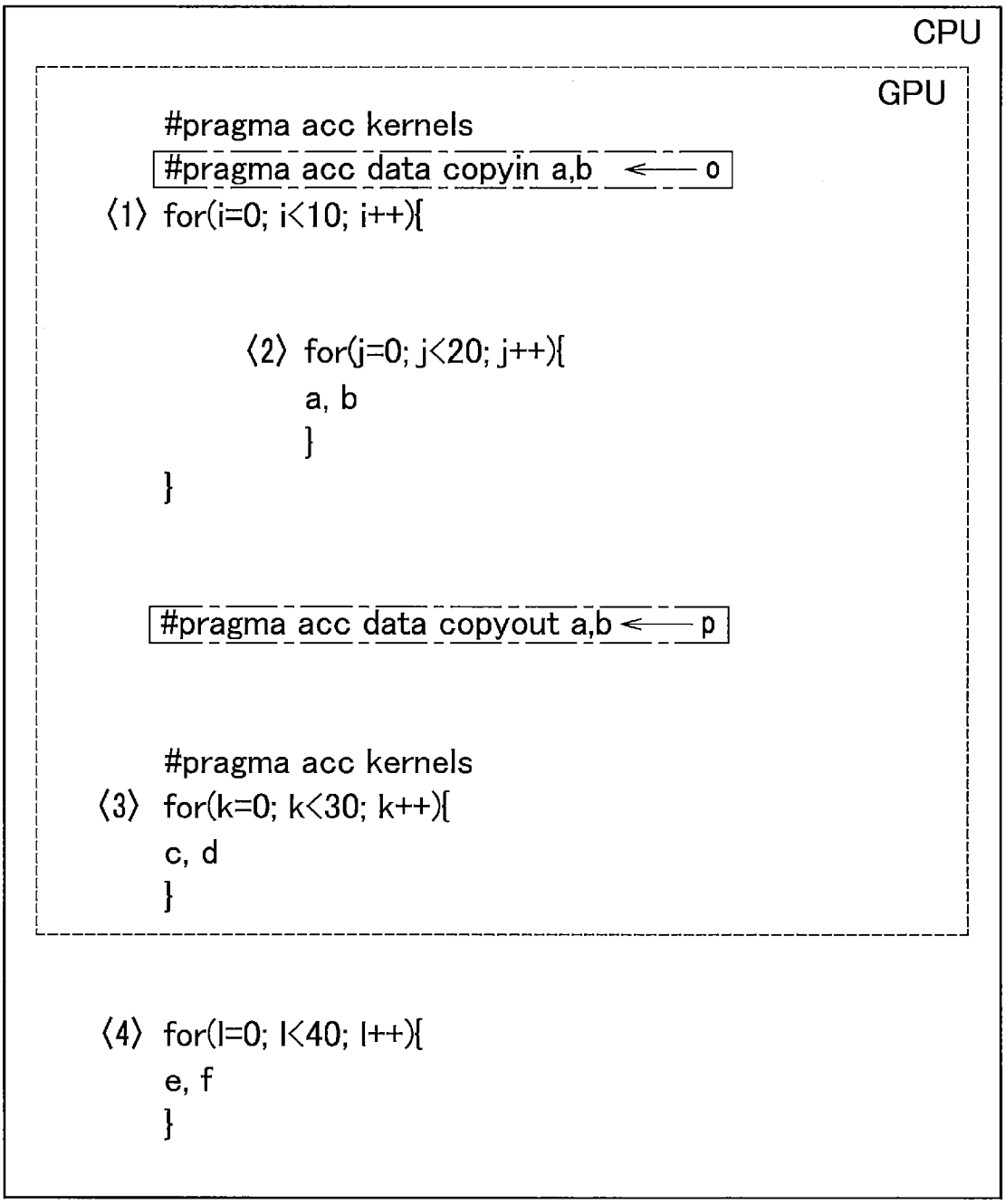
[図6]


CPU
〈1〉 for(i=0; i<10; i++){
〈2〉 for(j=0; j<20; j++){
a, b ←— j
}
}
〈3〉 for(k=0; k<30; k++){
c, d ←— k
}
〈4〉 for(l=0; l<40; l++){
e, f ←— l
}

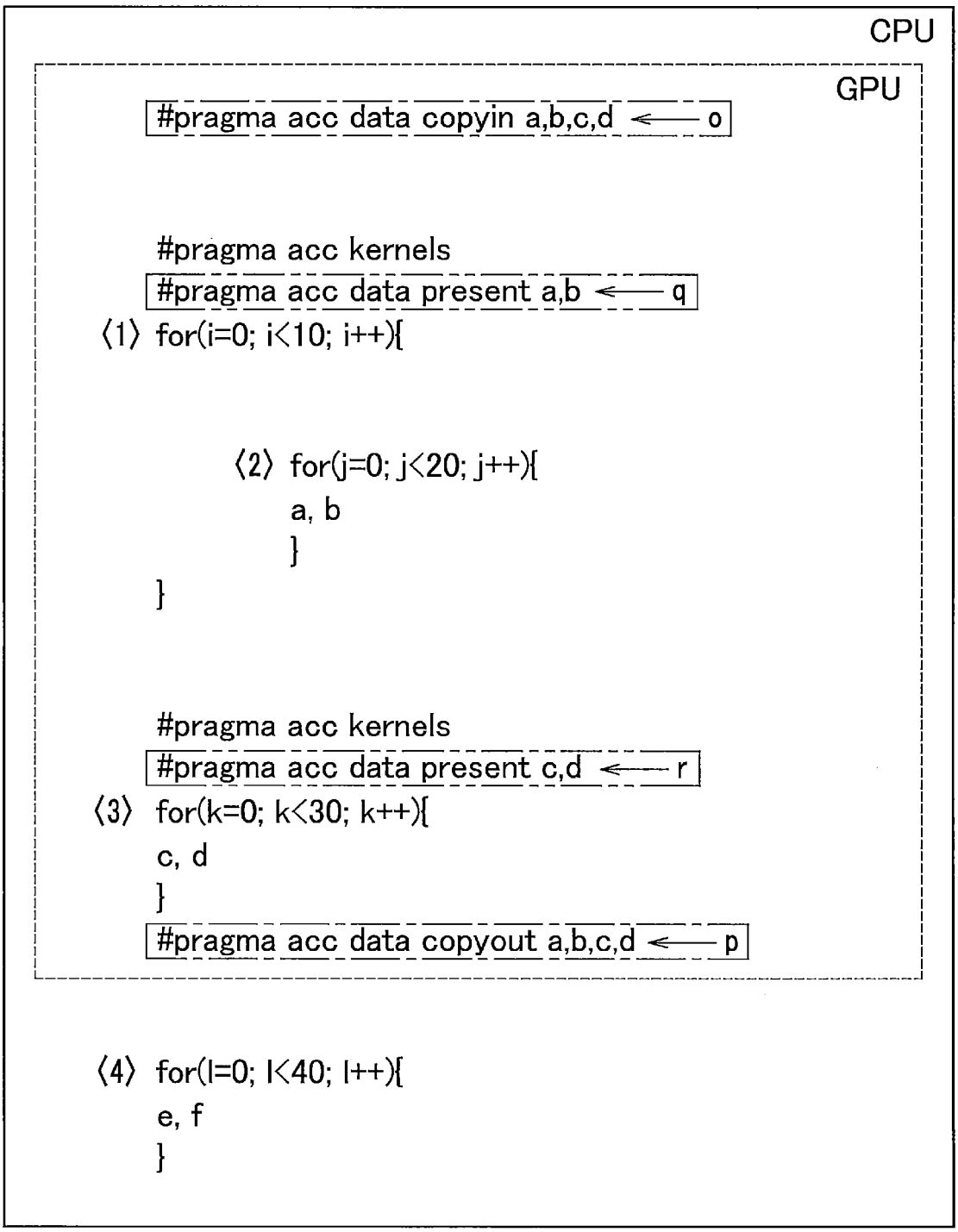
[図7]


CPU
GPU
#pragma acc kernels ← m
〈1〉 for(i=0; i<10; i++){
〈2〉 for(j=0; j<20; j++){
a, b
}
}
#pragma acc kernels ← n
〈3〉 for(k=0; k<30; k++){
c, d
}
〈4〉 for(l=0; l<40; l++){
e, f
}

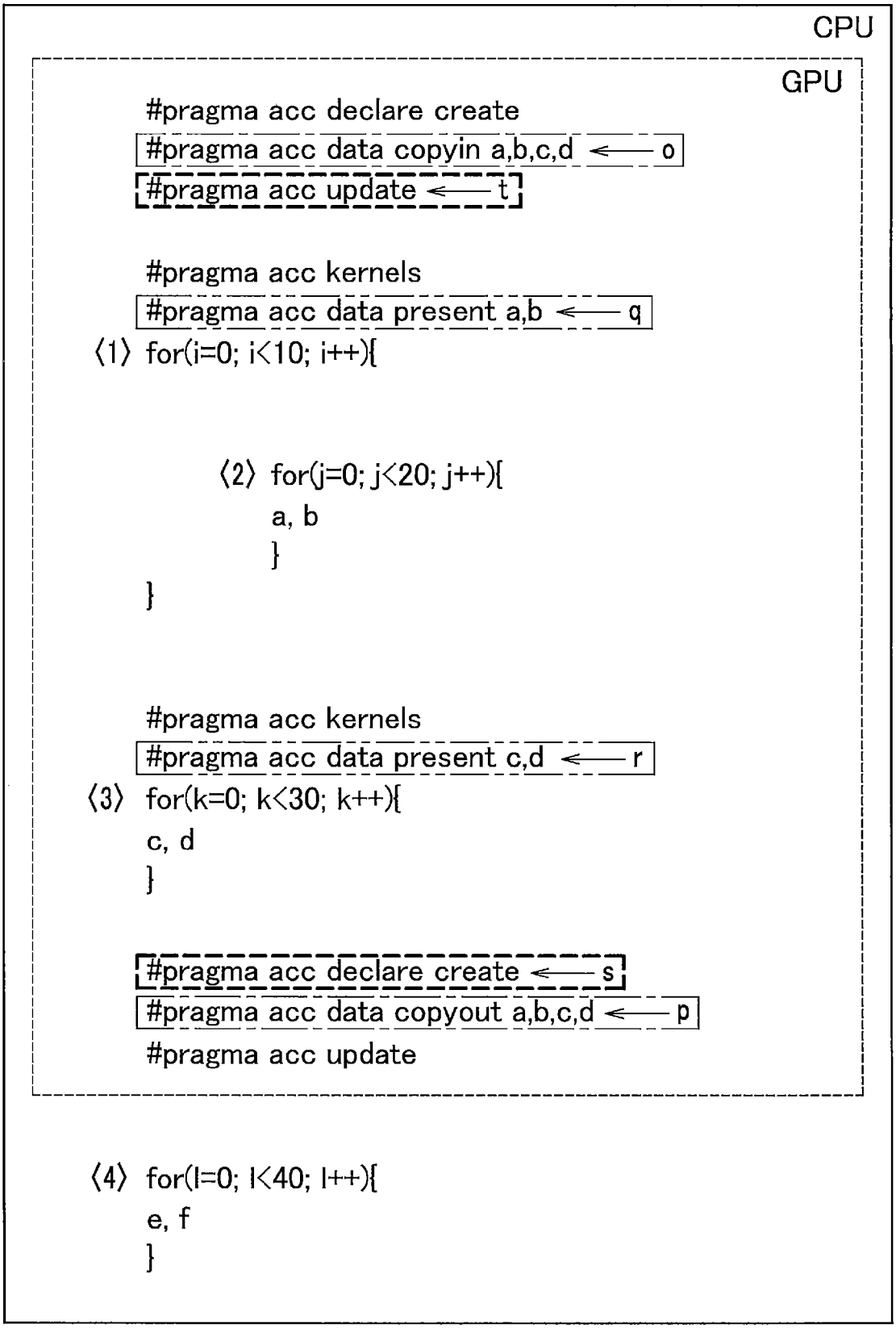
[図8]


CPU
GPU
#pragma acc kernels
#pragma acc data copyin a,b ← o
⟨1⟩ for(i=0; i<10; i++){
⟨2⟩ for(j=0; j<20; j++){
a, b
}
}
#pragma acc data copyout a,b ← p
#pragma acc kernels
⟨3⟩ for(k=0; k<30; k++){
c, d
}
⟨4⟩ for(l=0; l<40; l++){
e, f
}

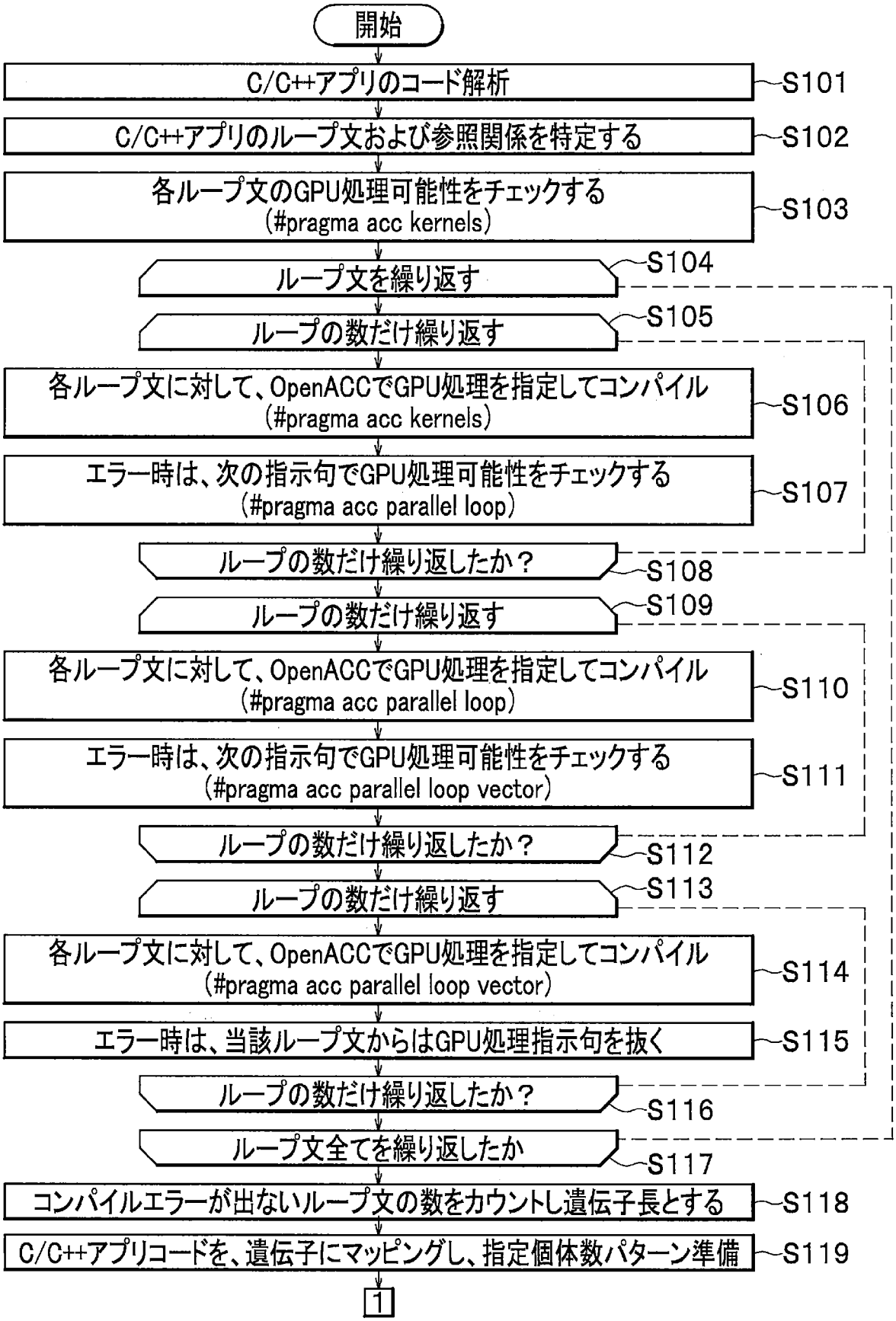
[図9]


CPU
GPU
#pragma acc data copyin a,b,c,d ← o
#pragma acc kernels
#pragma acc data present a,b ← q
⟨1⟩ for(i=0; i<10; i++){
⟨2⟩ for(j=0; j<20; j++){
a, b
}
}
#pragma acc kernels
#pragma acc data present c,d ← r
⟨3⟩ for(k=0; k<30; k++){
c, d
}
#pragma acc data copyout a,b,c,d ← p
⟨4⟩ for(l=0; l<40; l++){
e, f
}

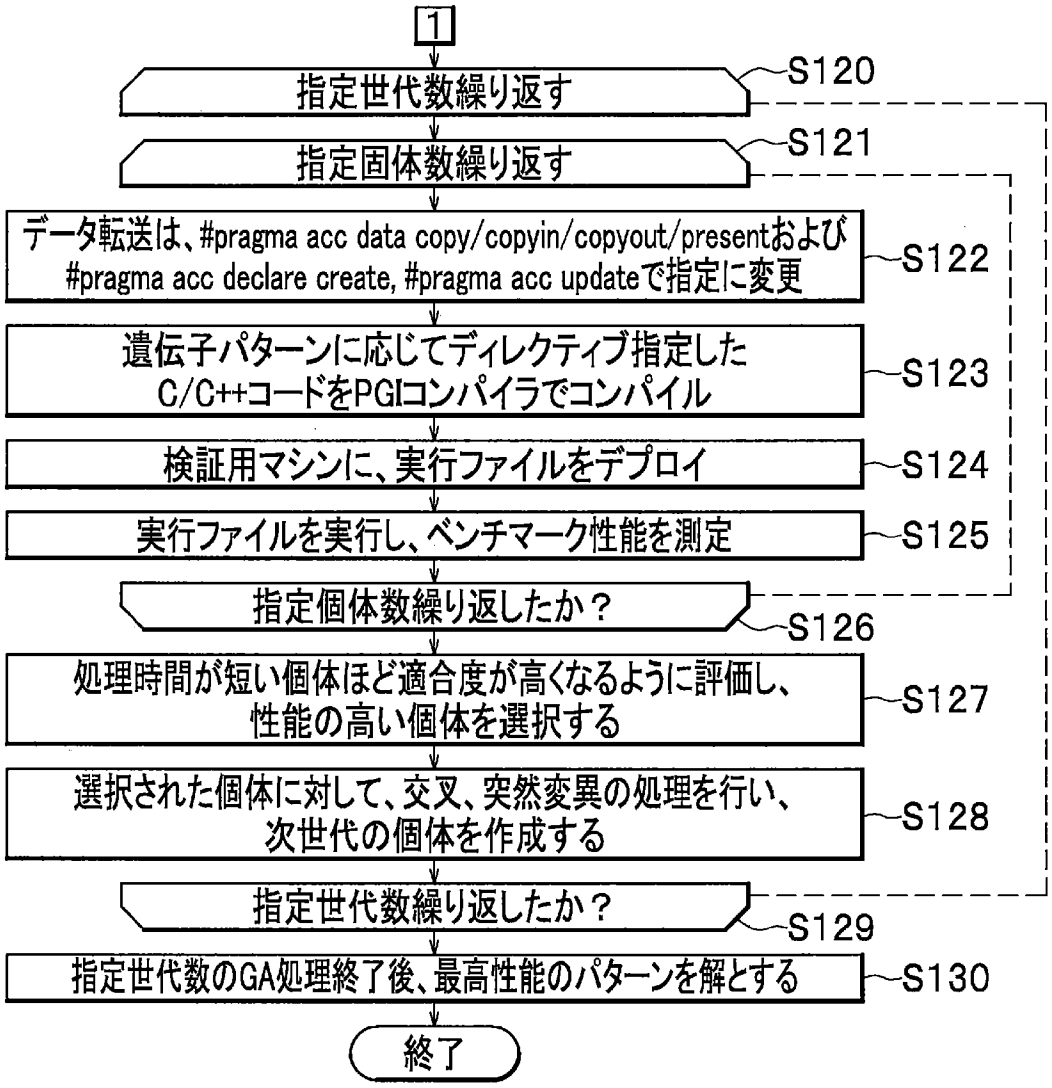
[図10]


CPU
GPU
#pragma acc declare create
#pragma acc data copyin a,b,c,d ← o
#pragma acc update ← t
#pragma acc kernels
#pragma acc data present a,b ← q
⟨1⟩ for(i=0; i<10; i++){
⟨2⟩ for(j=0; j<20; j++){
a, b
}
}
#pragma acc kernels
#pragma acc data present c,d ← r
⟨3⟩ for(k=0; k<30; k++){
c, d
}
#pragma acc declare create ← s
#pragma acc data copyout a,b,c,d ← p
#pragma acc update
⟨4⟩ for(l=0; l<40; l++){
e, f
}

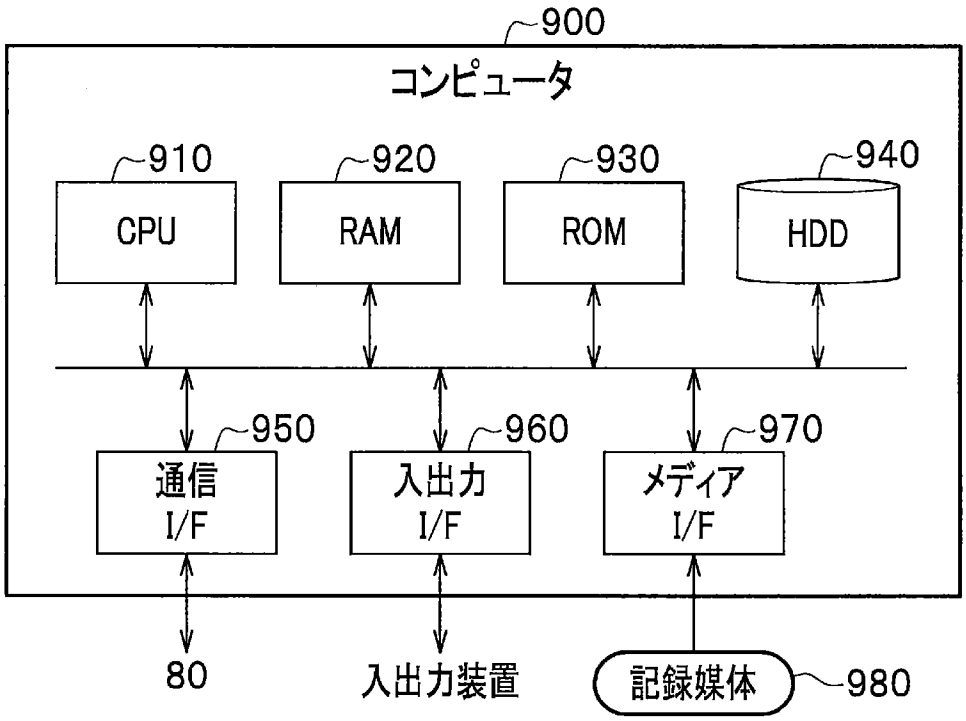
[図11A]


開始
C/C++アプリのコード解析
S101
C/C++アプリのループ文および参照関係を特定する
S102
各ループ文のGPU処理可能性をチェックする
(#pragma acc kernels)
S103
ループ文を繰り返す
S104
ループの数だけ繰り返す
S105
各ループ文に対して、OpenACCでGPU処理を指定してコンパイル
(#pragma acc kernels)
S106
エラー時は、次の指示句でGPU処理可能性をチェックする
(#pragma acc parallel loop)
S107
ループの数だけ繰り返したか？
S108
ループの数だけ繰り返す
S109
各ループ文に対して、OpenACCでGPU処理を指定してコンパイル
(#pragma acc parallel loop)
S110
エラー時は、次の指示句でGPU処理可能性をチェックする
(#pragma acc parallel loop vector)
S111
ループの数だけ繰り返したか？
S112
ループの数だけ繰り返す
S113
各ループ文に対して、OpenACCでGPU処理を指定してコンパイル
(#pragma acc parallel loop vector)
S114
エラー時は、当該ループ文からはGPU処理指示句を抜く
S115
ループの数だけ繰り返したか？
S116
ループ文全てを繰り返したか
S117
コンパイルエラーが出ないループ文の数をカウントし遺伝子長とする
S118
C/C++アプリコードを、遺伝子にマッピングし、指定個体数パターン準備
S119
1

[図11B]


1
指定世代数繰り返す
S120
指定固体数繰り返す
S121
データ転送は、#pragma acc data copy/copyin/copyout/presentおよび
#pragma acc declare create, #pragma acc updateで指定に変更
S122
遺伝子パターンに応じてディレクティブ指定した
C/C++コードをPGIコンパイラでコンパイル
S123
検証用マシンに、実行ファイルをデプロイ
S124
実行ファイルを実行し、ベンチマーク性能を測定
S125
指定個体数繰り返したか？
S126
処理時間が短い個体ほど適合度が高くなるように評価し、
性能の高い個体を選択する
S127
選択された個体に対して、交叉、突然変異の処理を行い、
次世代の個体を作成する
S128
指定世代数繰り返したか？
S129
指定世代数のGA処理終了後、最高性能のパターンを解とする
S130
終了

[図12]


900
コンピュータ
910
CPU
920
RAM
930
ROM
940
HDD
950
通信
I/F
960
入出力
I/F
970
メディア
I/F
80
入出力装置
記録媒体
980

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2020/004202

A. CLASSIFICATION OF SUBJECT MATTER
Int.Cl. G06F8/41(2018.01)i
FI: G06F8/41170

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)
Int.Cl. G06F8/41, G06F9/50

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Published examined utility model applications of Japan 1922-1996
Published unexamined utility model applications of Japan 1971-2020
Registered utility model specifications of Japan 1996-2020
Published registered utility model applications of Japan 1994-2020

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 2019/216127 A1 (NIPPON TELEGRAPH AND TELEPHONE CORPORATION) 14.11.2019 (2019-11-14), entire text	1-8
A	山登庸次 ほか, IoT アプリケーションの GPU オフロード時の並列処理部抽出とデータ転送回数低減手法, 電子情報通信学会技術研究報告, 2018.11.10, vol. 118, KBSE2018-37, SC2018-32(2018-11), pp. 53-58, ISSN: 2432-6380, entire text, (YAMATO, Yoji et al., Parallel processing area extraction and data transfer number reduction for automatic GPU offloading of IoT applications, IEICE Technical Report)	1-8

☐ 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	Date of mailing of the international search report
10.03.2020	24.03.2020
Name and mailing address of the ISA/ Japan Patent Office 3-4-3, Kasumigaseki, Chiyoda-ku, Tokyo 100-8915, Japan	Authorized officer Telephone No.

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/JP2020/004202

WO 2019/216127 A1 14.11.2019 (Family: none)

国際調査報告

国際出願番号
PCT/JP2020/004202

A. 発明の属する分野の分類（国際特許分類（ＩＰＣ））

G06F 8/41(2018.01)i
FI: G06F8/41 170

B. 調査を行った分野

調査を行った最小限資料（国際特許分類（ＩＰＣ））

G06F8/41, G06F9/50

最小限資料以外の資料で調査を行った分野に含まれるもの

日本国実用新案公報	１９２２－１９９６年
日本国公開実用新案公報	１９７１－２０２０年
日本国実用新案登録公報	１９９６－２０２０年
日本国登録実用新案公報	１９９４－２０２０年

国際調査で使用した電子データベース（データベースの名称、調査に使用した用語）

C. 関連すると認められる文献

引用文献のカテゴリー*	引用文献名　及び一部の箇所が関連するときは、その関連する箇所の表示	関連する請求項の番号
A	WO 2019/216127 A1 (日本電信電話株式会社) 14.11.2019 (2019-11-14) 全文	１－８
A	山登庸次　ほか，ＩｏＴアプリケーションのＧＰＵオフロード時の並列処理部抽出とデータ転送回数低減手法，電子情報通信学会技術研究報告，2018.11.10, Vol.118, KBSE2018-37, SC2018-32(2018-11), pp.53-58, ISSN:2432-6380 全文	１－８

☐ Ｃ欄の続きにも文献が列挙されている。

☑ パテントファミリーに関する別紙を参照。

* 引用文献のカテゴリー
“A” 特に関連のある文献ではなく、一般的技術水準を示すもの
“E” 国際出願日前の出願または特許であるが、国際出願日以後に公表されたもの
“L” 優先権主張に疑義を提起する文献又は他の文献の発行日若しくは他の特別な理由を確立するために引用する文献（理由を付す）
“O” 口頭による開示、使用、展示等に言及する文献
“P” 国際出願日前で、かつ優先権の主張の基礎となる出願の日の後に公表された文献
“T” 国際出願日又は優先日後に公表された文献であって出願と抵触するものではなく、発明の原理又は理論の理解のために引用するもの
“X” 特に関連のある文献であって、当該文献のみで発明の新規性又は進歩性がないと考えられるもの
“Y” 特に関連のある文献であって、当該文献と他の１以上の文献との、当業者にとって自明である組合せによって進歩性がないと考えられるもの
“&” 同一パテントファミリー文献

国際調査を完了した日	国際調査報告の発送日
10.03.2020	24.03.2020

名称及びあて先
日本国特許庁(ISA/JP)
〒100-8915
日本国
東京都千代田区霞が関三丁目４番３号

権限のある職員（特許庁審査官）
今城　朋彬 5B 7888

電話番号 03-3581-1101 内線 3545

国際調査報告
パテントファミリーに関する情報

国際出願番号
PCT/JP2020/004202

引用文献	公表日	パテントファミリー文献	公表日
WO 2019/216127 A1	14.11.2019	（ファミリーなし）	