(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2015-18379
(P2015-18379A)
(43)公開日　平成27年1月29日(2015.1.29)

(51)Int.Cl.			F I			テーマコード（参考）
G06F	9/48	(2006.01)	G06F	9/46	452A	5B013
G06F	9/38	(2006.01)	G06F	9/38	370C	

審査請求 未請求　請求項の数 7　ＯＬ　（全 25 頁）

(21)出願番号　特願2013-144661 (P2013-144661)
(22)出願日　平成25年7月10日 (2013.7.10)

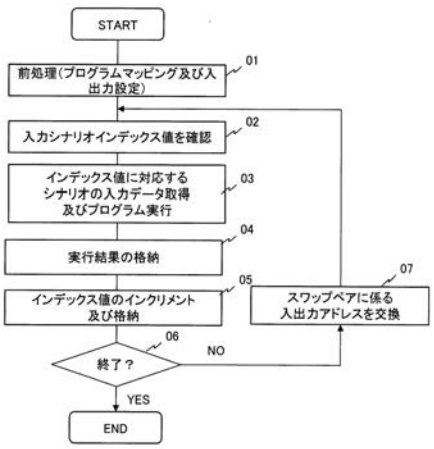
（出願人による申告）平成２４年度、独立行政法人科学技術振興機構、戦略的創造研究推進事業（個人型研究（さきがけ））に関する委託研究、産業技術力強化法第１９条の適用を受ける特許出願

(71)出願人　504171134
国立大学法人　筑波大学
茨城県つくば市天王台一丁目１番１
(74)代理人　100100549
弁理士　川口　嘉之
(74)代理人　100123319
弁理士　関根　武彦
(74)代理人　100105407
弁理士　高田　大輔
(72)発明者　山際　伸一
茨城県つくば市天王台一丁目１番１　国立大学法人筑波大学内

Ｆターム(参考) 5B013 DD03

(54)【発明の名称】プログラム、及び情報処理装置

(57)【要約】
【課題】プログラム毎に生じるデータのやりとりを回避して処理の時間短縮を図ることを可能とする技術を提供する。
【解決手段】プロセッサが周辺バスを介してデータとともに記憶装置に設定し、当該記憶装置に接続されたアクセラレータによって実行されるプログラムであって、プログラムは、複数の部品プログラムを含み、データは、少なくとも、前記部品プログラムの実行順序を示すシナリオデータと、前記複数の部品プログラムのうち最初に実行される部品プログラムに対する入力データと、シナリオインデックスの初期値とを含み、アクセラレータに、シナリオインデックスの初期値に従って実行すべき部品プログラムを特定する処理と、特定した部品プログラムに従った演算処理と、シナリオインデックスの値をインクリメントする処理と、インクリメントされたシナリオインデックスの値に応じて次の部品プログラムを特定する処理と、を少なくとも実行させる。
【選択図】図４



【特許請求の範囲】
【請求項１】
プロセッサが周辺バスを介してデータとともに記憶装置に設定し、当該記憶装置に接続されたアクセラレータによって実行されるプログラムであって、
前記プログラムは、複数の部品プログラムを含む統合プログラムであり、前記データは、少なくとも、前記部品プログラムの実行順序を示すシナリオデータと、前記複数の部品プログラムのうち最初に実行される部品プログラムに対する入力データと、シナリオインデックスの初期値とを含み、
前記アクセラレータに、前記シナリオインデックスの初期値に従って実行すべき部品プログラムを特定する処理と、特定した部品プログラムに従った演算処理と、前記シナリオインデックスの値をインクリメントする処理と、インクリメントされたシナリオインデックスの値に応じて次の部品プログラムを特定する処理と、
を少なくとも実行させるプログラム。
【請求項２】
前記プログラムは、或る部品プログラムの出力データが当該或る部品プログラムの次に実行される部品プログラムの入力データとなるときに、当該出力データと当該入力データとをペアとする定義を含み、
前記アクセラレータに、前記或る部品プログラムの実行終了時に、前記出力データの記憶アドレスと、前記出力データとペアをなす前記入力データの記憶アドレスとを交換する処理をさらに実行させる、
請求項１に記載のプログラム。
【請求項３】
前記シナリオインデックスの初期値が記憶される第１インデックスアドレスと、インクリメントされたシナリオインデックスの値が記憶される第２インデックスアドレスとをペアとする定義をさらに含み、
前記アクセラレータに、部品プログラムの実行が終了するごとに、前記第１インデックスアドレスと前記第２インデックスアドレスとを交換する処理をさらに実行させる、
請求項１又は２に記載のプログラム。
【請求項４】
プロセッサと、
前記プロセッサと周辺バスを介して接続される記憶装置と、
前記記憶装置に接続され、前記プロセッサによって前記記憶装置に設定されたプログラム及びデータを用いて所定の処理を行うアクセラレータとを含み、
前記プログラムは、前記処理にて実行される複数の演算を行うための複数の部品プログラムを含む統合プログラムであり、前記データは、少なくとも、前記部品プログラムの実行順序を示すシナリオデータと、前記複数の部品プログラムのうち最初に実行される部品プログラムに対する入力データと、シナリオインデックスの初期値とを含み、
前記アクセラレータは、前記シナリオインデックスの初期値に従って実行すべき部品プログラムを特定する処理と、特定した部品プログラムに従った演算処理と、前記シナリオインデックスの値をインクリメントする処理と、インクリメントされたシナリオインデックスの値に応じて次の部品プログラムを特定する処理と、を少なくとも実行する
情報処理装置。
【請求項５】
前記プログラムは、或る部品プログラムの出力データが当該或る部品プログラムの次に実行される部品プログラムの入力データとなるときに、当該出力データと当該入力データとをペアとする定義を含み、
前記アクセラレータは、前記或る部品プログラムの実行終了時に、前記出力データの記憶アドレスと、前記出力データとペアをなす前記入力データの記憶アドレスとを交換する処理をさらに実行する、
請求項４に記載の情報処理装置。

【請求項６】
前記シナリオインデックスの初期値が記憶される第１インデックスアドレスと、インクリメントされたシナリオインデックスの値が記憶される第２インデックスアドレスとをペアとする定義をさらに含み、
前記アクセラレータは、部品プログラムの実行が終了するごとに、前記第１インデックスアドレスと前記第２インデックスアドレスとを交換する処理をさらに実行する、
請求項４又は５に記載の情報処理装置。
【請求項７】
所定順序で直列に実行される複数のプログラムが統合された統合プログラムの生成処理をコンピュータに実行させるプログラムであって、
（１）プログラム間に、或るプログラムから次のプログラムへの遷移を示す時刻番号を設定するステップと、
（２）時刻番号毎に、或る時刻番号に対応する二つのプログラムの一方からの出力が、他方のプログラムの入力となるときの出力と入力とをスワップペアとして定義するステップと、
（３）最長の処理フローが得られるように時刻番号を合成することによって前記複数のプログラムの実行順を示すシナリオを生成するステップと、
（４）スワップペアをなす入力及び出力を記憶するための記憶容量に基づいて、時刻番号間のスワップペアをなす入力同士及び出力同士の合成を行うステップと、
（５）最初に実行されるプログラムの入力と、いずれかのスワップペアの入力との合成、及び最後に実行されるプログラムの出力と、いずれかのスワップペアの出力との合成を、各入力及び各出力のそれぞれの記憶容量に基づいて試行するステップと、
（６）前記（４）及び（５）の少なくとも一方で合成された入力及び出力と、前記（４）及び（５）で合成されずに残った入力及び出力とがそれぞれ入力及び出力として定義された前記統合プログラムを定義するとともに、前記シナリオに従って実行すべきプログラムの特定に使用されるシナリオインデックスの設定を行うステップと
をコンピュータに実行させるプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、プログラム、及び情報処理装置に関する。
【背景技術】
【０００２】
現在、ネットワークには、様々なセンサ及びデバイスが接続され、これらのセンサ及びデバイスから刻々と出力されるデータがネットワーク上でデータストリームを形成する。データストリームを形成するデータ（ストリームデータと呼ばれる）を滞りなく処理する（リアルタイムに処理する）手法として、ストリームコンピューティングがある。
【０００３】
近年、ストリームコンピューティングを実現するためのアクセラレータ技術が急速に発達している。代表的なアクセラレータの一つに、ＧＰＵ（Graphical Processing Unit）がある。ＧＰＵは、画像表示に必要な演算を高速で行うことを主たる目的として設計されており、制御のための計算を得意としていない。また、通常、ＧＰＵ自体は、通常ＯＳ（Operating System）で行われるようなリソース管理を行っていない。このため、ストリームコンピューティングのためにＧＰＵが適用される場合には、以下のような手法の採用が一般的である。
【０００４】
すなわち、ＧＰＵは、ＣＰＵ（Central Processing Unit）を搭載したコンピュータ（情報処理装置、例えば、パーソナルコンピュータ、ワークステーションのような専用又は汎用のコンピュータ）に備えられた周辺バスに接続される。ＧＰＵが目的の計算を行うに当たっては、ＣＰＵがＧＰＵ用のプログラムのダウンロード（ＧＰＵへのプログラムの提

供：プログラムマッピングともいう）と入出力データに係る設定（Ｉ／Ｏセットアップ）を行う。そして、ＣＰＵがＧＰＵにダウンロードされたプログラムの実行を指示する。ＧＰＵは、指示に従ってプログラムの実行を開始し、目的の計算を行う。このようなハードウェア構成では、ＧＰＵは、ＣＰＵをメインプロセッサとした場合のコプロセッサとして機能する。

【先行技術文献】

【非特許文献】

【０００５】

【非特許文献１】Shinichi Yamagiwa and Leonel Sousa. Modeling and Programming Stream-based Distributed Computing based on the Meta-Pipeline Approach, International Journal of Parallel, Emergent and Distributed Systems, Taylor & Francis, Vol. 24, Issue 4, pp. 311-330, August 2009.

【非特許文献２】Shinichi Yamagiwa and Leonel Sousa, Design and implementation of a tool for modeling and programming deadlock free meta-pipeline applications, 10th Workshop on Advances on Parallel and Distributed Processing Symposium (APDCM/IPDPS), pp.1-8, April. 2008 , IEEE.

【非特許文献３】Shinichi Yamagiwa, Leonel Sousa, Tomas Brandao, "Meta-Pipeline: A new execution mechanism for distributed pipeline processing", 6th International Symposium on Parallel and Distributed Computing (ISPDC 2007), Jun 2007.

【発明の概要】

【発明が解決しようとする課題】

【０００６】

上述したような、ＣＰＵとＧＰＵとが周辺バスで接続されたハードウェア環境において、複数のプログラムがＧＰＵで順次実行されることにより、所望の結果を得る処理フローを考える。この場合、ＣＰＵは、プログラム毎に、ＧＰＵに対するプログラムマッピング（ＧＰＵへのプログラムの提供）と、入出力データの設定を行う。

【０００７】

周辺バスの動作周波数（処理速度）は、ＣＰＵの動作周波数（処理速度）より低いことが少なくない。このため、プログラム毎に実行されるＣＰＵとＧＰＵとの周辺バスを介したやりとりによるオーバヘッドによって、所望の結果を得るまでに時間がかかるという問題があった。オーバヘッドの影響はＧＰＵにおける高速演算により吸収され得るが、そのような環境は、ＧＰＵの本来の性能が発揮される環境と言いがたい。

【０００８】

本発明の態様は、プログラム毎に生じるデータのやりとりを回避して処理の時間短縮を図ることを可能とする技術を提供することを目的とする。

【課題を解決するための手段】

【０００９】

本発明の態様の一つは、プロセッサが周辺バスを介してデータとともに記憶装置に設定し、当該記憶装置に接続されたアクセラレータによって実行されるプログラムであって、

前記プログラムは、複数の部品プログラムを含み、前記データは、少なくとも、前記部品プログラムの実行順序を示すシナリオデータと、前記複数の部品プログラムのうち最初に実行される部品プログラムに対する入力データと、シナリオインデックスの初期値とを含み、

前記アクセラレータに、前記シナリオインデックスの初期値に従って実行すべき部品プログラムを特定する処理と、特定した部品プログラムに従った演算処理と、前記シナリオインデックスの値をインクリメントする処理と、インクリメントされたシナリオインデックスの値に応じて次の部品プログラムを特定する処理と、

を少なくとも実行させるプログラムである。

【００１０】

また、本発明の他の態様は、上記したプログラムを記憶したコンピュータ読み取り可能

な記憶媒体、上記プログラムを実行する情報処理装置、上記したプログラムを用いた情報処理装置のアクセラレータ用プログラムの実行方法としても特定することができる。

【発明の効果】

【００１１】

本発明の態様によれば、プログラム毎に生じるデータのやりとりを回避して処理の時間短縮を図ることが可能となる。

【図面の簡単な説明】

【００１２】

【図１】図１は、ＧＰＵを備える情報処理装置（コンピュータ）の構成例を示す図である。

【図２】図２は、ＧＰＵで実行されるパイプライン処理のフローを模式的に示す図である。

【図３】図３は、図２に示した処理フローが単一のプログラムにパッキングされた状態を模式的に示す図である。

【図４】図４は、ＧＰＵによる、統合プログラムの実行手順の例を示すフローチャートである。

【図５】図５は、或る部品プログラムを用いた繰り返し処理を含む処理パイプラインの処理フロー例を模式的に示す。

【図６】図６は、図５に示した或る部品プログラムの繰り返し処理を再帰的な処理に置換した状態を模式的に示す。

【図７】図７は、或る部品プログラムの後段に位置する他の部品プログラムをパッキングした状態を模式的に示す。

【図８】図８は、統合プログラムのシナリオ生成方法手順の一例を示す図である。

【図９】図９は、ＧＰＵ用のプログラムによって記述された統合プログラムの一例を示す。

【図１０】図１０は、フローモデル（flow-model）と呼ばれる、パイプライン処理のモデルを定義したモデル定義情報と、所定の記述言語で記述されたパイプライン処理で行われる演算を指示するプログラム（カーネルプログラム）とが所定の記述言語で記述されたファイルの例を示す。

【図１１】図１１は、フローモデルを実行形式に変換するプログラム（実行支援プログラム）によって変換された統合プログラムの実行形式の例を示す。

【発明を実施するための形態】

【００１３】

以下、図面を参照して本発明の実施形態について説明する。実施形態の構成及び設定は例示であり、本発明は実施形態の構成及び設定に限定されない。

【００１４】

＜情報処理装置の構成＞

図１は、ＧＰＵを備える情報処理装置（コンピュータ）の構成例を示す図である。情報処理装置１０として、例えば、パーソナルコンピュータ（ＰＣ），ワークステーションのような専用又は汎用のコンピュータを適用することができる。

【００１５】

図１において、情報処理装置１０は、バスＢを介して相互に接続された、ＣＰＵ１１と、主記憶装置１２と、補助記憶装置１３と、入力装置１４と、出力装置１５と、通信インタフェース回路（通信Ｉ／Ｆ）１６とを備える。情報処理装置１０は、さらに、バスＢを介して接続されたビデオＲＡＭ（ＶＲＡＭ）１７と、ＶＲＡＭ１７に接続されたＧＰＵ１８とを備える。

【００１６】

主記憶装置１２は、ＣＰＵ１１の作業領域として使用されるメインメモリとして機能する。メインメモリは、例えば、ＲＡＭ（Random Access Memory）及びＲＯＭ（Read Only Memory）によって形成される。

【００１７】
補助記憶装置１３は、制御装置に相当するＣＰＵ１１によって実行される、各種のプログラム，及び各プログラムの実行時に使用されるデータを記憶する。補助記憶装置１３は、例えば、不揮発性記録媒体であり、例えば、ハードディスク，フラッシュメモリ，ＥＥＰＲＯＭ（Electrically Erasable Programmable Read-Only Memory），ＳＳＤ（Solid State Drive）の少なくとも１つを用いて形成することができる。主記憶装置１２及び補助記憶装置１３のそれぞれは、記憶装置，記録媒体の一例である。
【００１８】
入力装置１４は、キーボード，マウスやタッチパネルのようなポインティングデバイスを含み、情報（データ）の入力に使用される。出力装置１５は、例えば、ディスプレイ装置であり、情報を画面に表示する。通信Ｉ／Ｆ１６は、ネットワークとの通信処理を司る。
【００１９】
バスＢは、周辺バスの一例であり、例えば、ＰＣＩ (Peripheral Component Interconnect)バスや PCI expressを適用することができる。もっとも、主記憶装置１２とＶＲＡＭ１７とは、ＡＧＰ（Accelerated Graphics Port）のような、グラフィック・アクセラレータ（ＧＰＵ）用の専用バスを介して接続されても良い。
【００２０】
ＶＲＡＭ１７は、ＧＰＵ１８によって実行されるプログラム，実行に際して使用されるデータ，ＧＰＵ１８によるプログラムの実行結果としてのデータを記憶する。ＧＰＵ１８は、ＶＲＡＭ１７に記憶されたプログラムを実行することによって、所定の演算を行い、その結果をＶＲＡＭ１７に書き込む。ＶＲＡＭ１７は、記憶装置、記憶媒体の一例である。なお、記憶媒体には、ＣＤやＤＶＤ、ブルーレイディスクのような可搬性を有するディスク記録媒体も含まれる。
【００２１】
ＧＰＵ１８は、アクセラレータの一例であり、ＣＰＵ１１をメインプロセッサとしたときのコプロセッサとして機能する。すなわち、ＧＰＵ１８は、ＣＰＵ１１の制御下で、例えば、所定の複数の演算からなるパイプライン処理を行う。
【００２２】
ＣＰＵ１１は、主記憶装置１２に記憶されたＧＰＵ１８向けのプログラムを実行することにより、ＧＰＵ１８で実行されるプログラム（カーネルプログラムと呼ばれる）のコンパイルを行い、ＶＲＡＭ１７及びＧＰＵ１８の内部メモリの少なくとも一方にＧＰＵ１８が実行可能な形式で記憶（マッピング）する。これが、ＧＰＵ用のプログラムのダウンロードに相当する。また、ＣＰＵ１１は、ＶＲＡＭ１７にカーネルプログラムに応じた入出力設定（入力及び出力に係るアドレス（ポート）設定）を行う。ＣＰＵ１１は、カーネルプログラムの実行によって最終的に得られるＧＰＵ１８の出力データ（ＶＲＡＭ１７に記憶される）を、パイプライン処理の結果（複数の演算の最終結果）として得ることができる。
【００２３】
ＣＰＵ１１は、例えば、通信Ｉ／Ｆ１６で受信されるストリームデータに関して、ＧＰＵ１８を用いたリアルタイム処理を行い、ＧＰＵ１８の演算結果を出力装置１５で表示したり、通信Ｉ／Ｆ１６からネットワークへ送信したりする。これによって、情報処理装置１０は、ストリームコンピューティングを実施することができる。
【００２４】
なお、図１に示したハードウェア構成において、ＶＲＡＭ１７とＧＰＵ１８とがセットにされたグラフィックカードと呼ばれるデバイスが適用されても良い。また、ＶＲＡＭ１７の代わりに、主記憶装置１２の記憶領域の一部をＶＲＡＭ領域として使用する構成が採用されてもよい。
【００２５】
また、アクセラレータはＧＰＵに限られず、ＧＰＵ１８の代わりに、プログラムに従っ

て動作するデバイスが適用されても良い。例えば、デバイスは、ＤＳＰ（Digital Signal Processor）や、ＦＰＧＡ（Field Programmable Gate Array）を含むプログラマブルロジックデバイス（ＰＬＤ）を適用することができる。

【００２６】

＜ＧＰＵで実行される処理＞

次に、ＧＰＵ１８で実行される処理について説明する。図２は、ＧＰＵ１８で実行されるパイプライン処理のフローを模式的に示す図である。図２に示す処理パイプラインの例では、それぞれ異なる演算を行う３つのプログラムＡ，Ｂ及びＣが、プログラムＡ→プログラムＢ→プログラムＣの順でＧＰＵ１８により実行される。プログラムＡ，Ｂ及びＣのそれぞれの入力は、データａ，ｂ，ｃ及びｐとして定義され、プログラムＡ，Ｂ及びプログラムＣのそれぞれの出力は、データｄ，ｅ，ｆ及びｑとして定義される。

【００２７】

プログラムＡの出力データｄ，ｅ及びｆのそれぞれは、プログラムＢの入力データａ，ｂ，ｃとなり、プログラムＢの出力データｄ，ｅ及びｆのそれぞれは、プログラムＣの入力データａ，ｂ，ｃとなる。データｐは、各プログラムＡ，Ｂ，Ｃにおける入力のみのデータであり、データｑは、各プログラムＡ，Ｂ，Ｃにおける出力のみのデータである。

【００２８】

図２に示す処理フローを単純に実行しようとすると、各プログラムＡ，Ｂ，Ｃの実行毎に、ＣＰＵ１１によるＶＲＡＭ１７へのプログラムマッピング及び入出力セッティングが実行されてしまい、オーバヘッドが大きくなる。このため、実施形態では、プログラムＡ，Ｂ及びＣのパッキング（合成）が行われたＧＰＵ用のプログラムが適用される。

【００２９】

図３は、図２に示した処理フローが単一のプログラムにパッキングされた状態を模式的に示す図である。或るプログラムの出力データが次のプログラムの入力となる関係を持つとき、当該出力データと入力データとがペアとして定義される。図３に示す例では、データａとデータｄ，データｂとデータｅ，データｃとデータｆとのそれぞれはペアとして定義される。このような入力データと出力データとの組を“スワップペア（swap pair）”と呼ぶ。

【００３０】

さらに、プログラムＡ，Ｂ，及びＣは、単一のカーネルプログラムの形式（統合プログラム（packed program）と称する）に集約される。統合プログラムの入力は、データａ，ｂ及びｃと、シナリオデータ（scenario）と、入力シナリオインデックス（scenario_index_in）とを含む。一方、統合プログラムの出力は、データｄ，ｅ及びｆと、出力シナリオインデックス（scenario_index_out）とを含む。

【００３１】

シナリオデータ（シナリオ）は、統合プログラムに含まれるプログラムＡ，Ｂ及びＣ（それぞれを部品（コンポーネント）プログラムと称することもある）の動作手順（実行順（Ａ→Ｂ→Ｃ））を指定するデータであり、プログラムＡ，Ｂ及びＣのいずれが実行されるかは、入力シナリオインデックスの値に応じて決定される。

【００３２】

例えば、入力シナリオインデックスの値が初期値（例えば“０”）であるときにプログラムＡが実行され、入力シナリオインデックスの値が“１”であるときに、次のプログラムＢが実行され、入力シナリオインデックスの値が“２”であるときに次のプログラムＣが実行される。シナリオデータは、インデックス値がインクリメントされる毎に、次の部品プログラムが実行対象として特定されるための情報形式を有する。

【００３３】

入力シナリオインデックスは、次に実行する部品プログラムを指定するための値である。入力シナリオインデックスの値は、当該インデックス値で指定された部品プログラムの実行開始前、実行中、実行終了後のいずれかにおいてインクリメントされ、インクリメントされた値が出力シナリオインデックスとして、部品プログラムの出力の一つとして出力

される。入力シナリオインデックスと出力シナリオインデックスとは、スワップペアの１つとして定義することができる。

【００３４】

なお、図３において、統合プログラムに対する入力データＡｐ，Ｂｐ及びＣｐのそれぞれは、プログラムＡ，Ｂ，Ｃのそれぞれに対する入力データであり、出力データＡｑ，Ｂｑ，Ｃｑのそれぞれは、プログラムＡ，Ｂ，Ｃからの出力データである。これらは、隣接する（接続関係にある）部品プログラム間で、スワップペアを形成しないデータである。

【００３５】

図４は、ＧＰＵ１８による、上記した統合プログラムの実行手順の例を示すフローチャートである。図４に示す０１では、前処理が実行される。前処理として、ＣＰＵ１１がＧＰＵ１８側の記憶領域（ＶＲＡＭ１７及びＧＰＵ１８の内部メモリの少なくとも一方）に対する統合プログラムのマッピング及び入出力設定を行う。但し、以下の説明では、ＧＰＵ１８側の記憶領域としてＶＲＡＭ１７のみが使用される例について説明する。これによって、ＶＲＡＭ１７には、統合プログラムがＧＰＵ１８で実行可能な形式で記憶される。また、統合プログラムに対する初期入力データとして、プログラムＡに対する入力データＡ，Ｂ，Ｃ及びＡｐ，プログラムＢに対するデータＢｐ，プログラムＣに対する入力データＣｐ，シナリオデータ，及び入力シナリオインデックスの初期値を示す数値（例えば“０”）が、ＶＲＡＭ１７の所定アドレスにそれぞれ記憶される。ここで、データａ，ｂ及びｃのそれぞれは、所定の入力アドレス“ＡＤＤＲ１”，“ＡＤＤＲ２”，“ＡＤＤＲ３”に格納されており、入力シナリオインデックス値“０”は、所定のアドレス“ＡＤＤＲ４”に格納されるものとする。

【００３６】

その後、ＧＰＵ１８がＣＰＵ１１から発行された統合プログラムの実行指示を受け取ると、ＧＰＵ１８は、シナリオデータと入力シナリオインデックスの初期値とをＶＲＡＭ１７から読み出して、入力シナリオインデックス値に対応するプログラム、すなわち実行すべき部品プログラムを特定する（０２）。ここで、ＧＰＵ１８は、入力シナリオインデックス値“０”（初期値）に従って、最初の部品プログラム、すなわちプログラムＡの実行を開始する。

【００３７】

このとき、ＧＰＵ１８は、プログラムＡ中で指定された入力データのアドレス（入力アドレス）を示す入力ポインタ値に従って、ＶＲＡＭ１７からプログラムＡ用のデータａ，ｂ，ｃ及びデータＡｐを読み出し、プログラムＡに従った演算を行う（０３）。

【００３８】

ＧＰＵ１８は、演算結果として得られたデータｄ，ｅ，ｆ及びＡｑを、出力ポインタで指定されるＶＲＡＭ１７上のアドレス（出力アドレス）にそれぞれ記憶する（０４）。このとき、データｄ，ｅ，ｆのそれぞれは、所定の出力アドレス“ＡＤＤＲ５”，“ＡＤＤＲ６”，“ＡＤＤＲ７”に格納される。

【００３９】

また、ＧＰＵ１８は、入力シナリオインデックスの値をインクリメントし、出力シナリオインデックス“１”として、出力ポインタで指定されたアドレス“ＡＤＤＲ８”に記憶する（０５）。なお、０４の処理と０５の処理の順序は逆でも良い。

【００４０】

次に、ＧＰＵ１８は、シナリオ（プログラムＡ，Ｂ，Ｃ）の全ての実行が終了したか否かを判定する（０６）。シナリオが終了していれば（０６；ＹＥＳ）、図４に示す処理が終了する。これに対し、シナリオが終了していなければ（０６；ＮＯ）、ＧＰＵ１８は、スワップペアとして定義されたデータに関する入力アドレスと出力アドレスとの交換（スワップ）を行う（０７）。入出力アドレスのスワップは、例えば、入力ポインタと出力ポインタとの交換（スワップ）によって行われる。これによって、出力アドレス“ＡＤＤＲ５”～“ＡＤＤＲ８”のそれぞれが次の部品プログラムに関する入力アドレスとなり、入力アドレス“ＡＤＤＲ１”～“ＡＤＤＲ４”のそれぞれが次の部品プログラムに関する出

力アドレスとなる。
【００４１】
続いて、処理が０２に戻り、ＧＰＵ１８は、インデックス値の入力ポインタに従って、アドレス“ＡＤＤＲ８”から読み出したインデックス値“１”に基づき、シナリオ上の次の部品プログラムに該当するプログラムＢを特定する。ＧＰＵ１８は、入力アドレスから入力データを得て、プログラムＢの実行を開始する（０３）。このとき、ＧＰＵ１８は、入力ポインタに従ってアドレス“ＡＤＤＲ４”～“ＡＤＤＲ７”から読み出されるプログラムＡの出力データｄ，ｅ，ｆを、入力データａ，ｂ，ｃとして扱う。
【００４２】
ＧＰＵ１８は、プログラムＢの実行によって得られた演算結果としての出力データｄ，ｅ，ｆのそれぞれを、出力ポインタに従って、アドレス“ＡＤＤＲ１”，“ＡＤＤＲ２”，“ＡＤＤＲ３”に格納する（０４）。なお、プログラムＢの実行に際して、データＢｐも読み出され、プログラムＢの実行結果として得られた出力データＢｑは、所定の出力アドレスに格納される。
【００４３】
ＧＰＵ１８は、アドレス“ＡＤＤＲ８”から読み出したインデックス値“１”をインクリメントしたインデックス値“２”を得て、出力ポインタで指定されたアドレス“ＡＤＤＲ４”に格納する（０５）。
【００４４】
続いて、ＧＰＵ１８は、シナリオ未終了との判断（０６；ＮＯ）を経て、スワップペアとして定義されたデータに関する入力ポインタと出力ポインタとの交換（スワップ）を行う（０７）。これによって、出力アドレス“ＡＤＤＲ１”～“ＡＤＤＲ４”のそれぞれが入力アドレスとなり、入力アドレス“ＡＤＤＲ５”～“ＡＤＤＲ８”のそれぞれが出力アドレスとなる。
【００４５】
ＧＰＵ１８は、インデックス値の入力ポインタに従って、アドレス“ＡＤＤＲ４”から読み出したインデックス値“２”を確認し（０２）、インデックス値“２”に応じたプログラムＣの実行を開始する（０３）。ＧＰＵ１８は、入力ポインタに従って入力データを取得する。このとき、ＧＰＵ１８は、アドレス“ＡＤＤＲ１”～“ＡＤＤＲ３”から読み出されるプログラムＢの出力データｄ，ｅ，ｆを、入力データａ，ｂ，ｃとして扱う。
【００４６】
ＧＰＵ１８は、プログラムＣの実行によって得られた演算結果としての出力データｄ，ｅ，ｆのそれぞれを、出力ポインタに従って、アドレス“ＡＤＤＲ５”，“ＡＤＤＲ６”，“ＡＤＤＲ７”に格納する（０４）。なお、プログラムＣの実行に際して、データＣｐも読み出され、プログラムＣの実行結果として得られた出力データＣｑも、所定の出力アドレスに格納される。
【００４７】
ＧＰＵ１８は、プログラムＣの実行が終了すると、シナリオ終了と判定し（０６；ＹＥＳ）、図４の処理（統合プログラムの実行）を終了する。ＶＲＡＭ１７上の出力アドレスに記憶されたプログラムＣの出力データｄ，ｅ，ｆ及びデータＡｑ，Ｂｑ及びＣｑは、ＧＰＵ１８によるパイプライン処理の結果として、ＣＰＵ１１に渡される。
【００４８】
なお、上記した処理における、スワップペアを形成するデータの数（３個）は例示であり、プログラムの実行に際して使用されるデータの数に応じて変動する。また、入力データＡｐ，Ｂｐ，Ｃｐ及び出力データＡｑ，Ｂｑ，Ｃｑも例示である。すなわち、入力データＡｐ，Ｂｐ，Ｃｐの一部又は全部に相当するデータがない場合もあれば、出力データＡｑ，Ｂｑ，Ｃｑの一部又は全部に相当するデータがない場合もある。また、プログラムの内容に応じて、２以上のスワップペアを形成しないデータが出力される場合もあり得る。
【００４９】
図４に示したＧＰＵ１８の処理によれば、統合プログラムに含まれた複数のＧＰＵ用プ

ログラムの実行が、シナリオと、シナリオインデックス値とを用いて制御される。これによって、プログラムＡに後続するプログラムＢ及びＣの実行に際して、ＣＰＵ１１とＧＰＵ１８とのやりとり（プログラムマッピング及び入出力データセッティング）が回避される。このため、ＣＰＵ１１へのプログラムＡ及びＢの実行結果の返却や、ＣＰＵ１１によるプログラムＢ及びＣのマッピング及び入出力セッティングに係るオーバヘッドを削減することができる。よって、ＣＰＵ１１がＧＰＵ１８を用いた複数のプログラムの実行結果を得るための時間を短縮することができる。また、オーバヘッドの削減によって、ＧＰＵ１８による高速演算の効果を十分に発揮させることができる。

【００５０】

また、統合プログラムでは、スワップペアが定義され、次の部品プログラムの実行開始に先立って、スワップペアに係る出力アドレスと入力アドレスとの交換が行われる。これによって、ＣＰＵ１１とＧＰＵ１８とのやりとり（すなわち、主記憶装置１２に記憶された入力データをＶＲＡＭ１７にコピーする処理）が回避されるので、オーバヘッドの削減を図ることができる。

【００５１】

さらに、入出力アドレスの交換が実行されることで、ＣＰＵ１１による前処理での入出力セッティングにおいて、２番目以降の部品プログラムに関して、スワップペアに係る入出力設定を省略することができる。これによって、前処理の処理量が削減される。統合プログラムの作成にあたっては、スワップペアを定義することで、複数のプログラムを容易にパッキングすることができる。

【００５２】

また、シナリオインデックス値の制御は、カウンタによって行うこともできる。但し、上述したように、シナリオインデックス値を入力シナリオインデックスと出力シナリオインデックスとのスワップペアとすることで、入出力アドレスの入れ替えにより、次のインデックス値を与えることができる。これによって、カウンタを省略できるという利点がある。統合プログラム作成の観点からは、カウンタ生成の手順を省略できる。

【００５３】

なお、図３では、スワップペアを定義し得る複数の部品プログラムＡ～Ｃがパッキングされた統合プログラムの例を示した。但し、スワップペアを定義できることは、統合プログラム作成に当たっての条件とされない。例えば、図３の統合プログラムからデータａ～ｆが省略された場合のような、スワップペアのない複数の部品プログラムがパッキングされた統合プログラムがＧＰＵ１８で実行される場合においても、前処理で入力データＡｐ，Ｂｐ，Ｃｐを設定することで、ＧＰＵ１８の処理結果である出力データＡｑ，Ｂｑ，Ｃｑを周辺バスを介したデータ送受信なく得ることができる。すなわち、上述したオーバヘッド削減の効果を得ることができる。

【００５４】

＜他のパッキング手法＞

図５から図７は、図２及び図３に示したケースと異なる複数のプログラムのパッキング手法を模式的に示す図である。図５は、部品プログラム（カーネルプログラム）Ｘ及びＹを用いた処理パイプラインの処理フロー例を示す。図５に示す例では、カーネルプログラムＸを用いた処理が３回繰り返された結果がカーネルプログラムＹに入力される。そして、カーネルプログラムＹから出力されるデータが、処理パイプラインの最終結果として出力される。

【００５５】

図５のカーネルプログラムＸの繰り返し処理に着目すると、２回目、３回目のカーネルプログラムＸに対する入力は、前段のカーネルプログラムＸの出力である。従って、図６に示すように、２回目及び３回目のカーネルプログラムＸの処理は、初回のカーネルプログラムＸにおける再帰的処理に置換可能である。当該処理は、カーネルプログラムＸの入力と出力とをスワップペアとして、初回のカーネルプログラムＸの出力データのアドレス（出力アドレス）を、入力アドレスと交換することで、実現することができる。

【００５６】
さらに、カーネルプログラムＹの入力に着目すると、カーネルプログラムＹの入力は、カーネルプログラムＸによる３回目の処理結果（出力データｘ，ｙ）である。このため、カーネルプログラムＸの３回目の出力とカーネルプログラムＹの入力をスワップペアとして定義し、カーネルプログラムＸの処理からカーネルプログラムＹの処理へ遷移する際に、出力アドレスと入力アドレスとの交換によって、出力データｘ，ｙがカーネルプログラムＹに入力されるようにする。このようにして、４つの演算ステップが単一の統合プログラムとしてパッキングされる。
【００５７】
なお、上記したように処理フローが或る部品プログラム（図５のカーネルプログラムＸ）の繰り返し処理を含む場合には、シナリオに、或る部品プログラムによる処理の繰り返し回数が定義される。この場合、繰り返し回数とインクリメントの回数とが比較され、インクリメントの回数が繰り返し回数を上回ったときに、次の部品プログラムによる処理に遷移する。
【００５８】
＜統合プログラムのシナリオ生成方法＞
次に、統合プログラムのシナリオ生成方法の一例について説明する。複数のＧＰＵ用プログラム（カーネルプログラム）を１つのプログラムにまとめる（統合プログラムを生成する）ために、以下の手順で、スワップペアを生成し、さらに、シナリオを生成する。
【００５９】
図８は統合プログラムの生成方法の手順の一例を示す図である。図８には、例として、複数のカーネルプログラムとしてのプログラム１，２，３が、プログラム１→プログラム２→プログラム３の順で実行される処理フローが示されている。プログラム１～３のそれぞれは、２つの入力と２つの出力を有し、それぞれを特定する番号（識別子）が設定される。
【００６０】
すなわち、プログラム１への２つの入力には、“１．ａ”と“１．ｂ”とが設定されており、プログラム１からの２つの出力には、“１．ｃ”と“１．ｄ”とが設定されている。同様に、プログラム２への２つの入力には、“２．ａ”と“２．ｂ”とが設定されており、プログラム２からの２つの出力には、“２．ｃ”と“２．ｄ”とが設定されている。また、プログラム３への２つの入力には、“３．ａ”と“３．ｂ”とが設定されており、プログラム３からの２つの出力には、“３．ｃ”と“３．ｄ”とが設定されている。当該処理フローを例に、生成手順を説明する。
【００６１】
＜＜ステップ（１）＞＞
プログラム間に、或るプログラムから次のプログラムへの遷移を示す時刻番号を設定する。すなわち、複数のカーネルプログラムの実行順に従って、入出力（Ｉ／Ｏ）の接続に時刻番号を振る。図８の例を用いて説明すると、プログラム１とプログラム２の接続に関しては時刻番号“Time A”が設定され、その次の時刻である、プログラム２とプログラム３の接続に関しては時刻番号“Time B”が設定される。
【００６２】
＜＜ステップ（２）＞＞
次に、それぞれの時刻に関して、スワップペアが定義される。つまり、各時刻において、プログラム間で接続されている入出力（Ｉ／Ｏ）組が、すべて列挙される。すなわち、時刻番号“Time A”及び“Time B”のそれぞれに関して、例えば、“ペア名＝[出力,入力],バッファサイズ”の書式で、スワップペアが定義される。ペアをなす出力のバッファサイズと入力のバッファサイズとは、上記した入出力アドレスの交換を考慮して同じとされる。具体的には、以下のようなスワップペアの定義がなされる。
Time A：Pair 1=[1.c, 2.a], N1 及びPair 2=[1.d, 2.b], N2
Time B：Pair 1=[1.c, 2.a], N3 及びPair 2=[1.d, 2.b], N4

【００６３】
ここに、バッファは、プログラムへの入力データ、及びプログラムからの出力データを格納する記憶領域を指し、バッファサイズは、バッファの記憶容量を示す。バッファは、ＶＲＡＭ１７上に形成される。もっとも、バッファは、ＧＰＵ１８の内部メモリ上に形成されることもあり得る。
【００６４】
＜＜ステップ（３）＞＞
次に、すべての入出力（Ｉ／Ｏ）のペアが入出力を得られるための最長の実行手順をシナリオとする。具体的には、時刻番号“Time A”では、プログラム１からプログラム２へ遷移し、時刻番号“Time B”では、プログラム２からプログラム３へ遷移している。これより、共通のプログラム２で時刻番号“Time A=[1→2]”と時刻番号“Time B=[2→3]”とを結ぶと、最長のフロー（１→２→３）が得られ、当該フローがシナリオとして決定される。このように、ステップ（３）では、最長の処理フローが得られるように、時刻番号を合成することによって統合プログラムのシナリオが生成される。
【００６５】
＜＜ステップ（４）＞＞
次に、スワップペアをなす入力及び出力を記憶するための記憶容量（バッファサイズ）に基づいて、時刻番号間のスワップペアをなす入力同士及び出力同士の合成を行う。すなわち、ステップ（１）で定義した時刻に対応するスワップペアのそれぞれに対して、その次の時刻のペアの大きいか等しいバッファサイズを持つものとグループ化し、新たなペア（合成ペアと呼ぶ）を作成する。バッファサイズが小さいと、入出力アドレスの交換（スワップ）によってデータが格納できなくなるからである。このとき、合成ペアのバッファサイズとして、合成ペアをなす２つのスワップペアのバッファサイズのうち、大きいバッファサイズが採用される。一度、合成したペア（合成ペアをなすスワップペア）は、再度、他のスワップペアとの合成に使われない。残った入力のペア（1.aと1.b）及び出力のペア（3.c及び3.d）はそのままとし、次のステップへ進む。
【００６６】
具体的には、ステップ（４）では、TimeAとTimeBのステップ（１）で定義したそれぞれの時刻のスワップペアを減らすため、上記したバッファの大きさに基づくステップ（４）のルールに従い、隣接する時刻間でスワップペアを合成する。例えば、バッファサイズＮ１，Ｎ２，Ｎ３及びＮ４の大小関係が“Ｎ１＞Ｎ３”であり、“Ｎ２＝Ｎ４”であり、“Ｎ１＜Ｎ４”であると仮定する。すなわち、“Ｎ３＜Ｎ１＜Ｎ４＝Ｎ２”であると仮定する。
【００６７】
この場合、スワップペア“Pair 1”と“Pair 3”との集合が新たなスワップペア“Pair 1’=Pair 1 ∪ Pair 3 = [c’, a’], N1”として定義され、スワップペア“Pair 2”と“Pair 4”との集合が新たなスワップペア“Pair 2’=Pair 2 ∪ Pair 4 = [d’, b’], N2”として定義される。
【００６８】
＜＜ステップ（５）＞＞
次に、最初に実行されるプログラムの入力と、いずれかのスワップペアの入力との合成、及び最後に実行されるプログラムの出力と、いずれかのスワップペアの出力との合成を、各入力及び各出力のそれぞれの記憶容量（バッファサイズ）に基づいて試行する。すなわち、次に合成ペアの全てと残った入力ペア及び出力ペアとを新たな入出力（Ｉ／Ｏ）として、統合プログラムを定義する。このとき、ステップ（３）で作成したシナリオにおける、最初のプログラム（プログラム１）の入力と、合成ペアとが新たなペアを形成する場合において、入力側のバッファサイズが合成ペアのバッファサイズと等しいか小さい場合には、当該入力は合成ペアの入力と合成される。処理フロー中の最後に位置するプログラムに関しても、その出力が合成ペアと等しいか小さい場合には、当該出力は合成ペアの出力と合成される。このとき、合成は１つの入力に関して１回のみ可能である。

【００６９】
具体的には、ステップ（５）では、ステップ（３）で定義したシナリオの最初のプログラム１への入力（1.a及び1.b）に関して、ステップ（４）で得られた合成ペアの入力と合成可能か否かがバッファサイズに基づき判定される。同様に、スワップペアに含まれていない、シナリオ中の最後のプログラム３の出力（3.c及び3.d）に関して、合成ペアの出力との合成が可能か否かがバッファサイズに基づき判定される。
【００７０】
入力１．ａ及び１．ｂのバッファサイズがそれぞれＮ５，Ｎ６であり、出力３．ｃ及び３．ｄのバッファサイズがそれぞれＮ７，Ｎ８であると仮定する。さらに、バッファサイズの大小関係が、Ｎ５＝Ｎ１，Ｎ６＞Ｎ２，Ｎ７＝Ｎ３，Ｎ８＞Ｎ４であると仮定する。
【００７１】
このとき、入力１．ａのバッファサイズと合成ペア“Pair 1'”のバッファサイズＮ１とは等しいので、入力１．ａは合成ペア“Pair 1'”の入力“ａ'”と合成される。一方、入力１．ｂのバッファサイズＮ６は、合成ペア“Pair 2'”のバッファサイズＮ２より大きいので、入力１．ｂは合成できず、そのまま統合プログラムの入力の一つとなる。
【００７２】
入力１．ａのバッファサイズと合成ペア“Pair 1'”の入力“ａ'”のバッファサイズＮ１とは等しいので、入力１．ａは合成ペア“Pair 1'”の入力“ａ'”と合成される。一方、入力１．ｂのバッファサイズＮ６は、合成ペア“Pair 2'”の入力“ｂ'”のバッファサイズＮ２より大きい。このため、入力１．ｂは合成できず、そのまま残される。
【００７３】
出力３．ｃのバッファサイズＮ７と合成ペア“Pair 1'”の出力“ｃ'”のバッファサイＮ３とは等しいので、入力３．ｃは出力“ｃ'”と合成される。一方、入力３．ｄのバッファサイズＮ８は、合成ペア“Pair 2'”の出力“ｄ'”のバッファサイズＮ４より大きい。このため、出力３．ｄは合成できず、そのまま残る。
【００７４】
＜＜ステップ（６）＞＞
最後に、最終的に残った入出力と、合成ペアの入出力を部品プログラム（プログラム１，２，３）がパッキングされた統合プログラムの入出力として定義したものを作成する。このとき、シナリオ及び入出力シナリオインデックスも設定される。
【００７５】
具体的には、図８に示すように、統合プログラムの入力として、データａ'と、データ１．ｂと、データｂ'と、シナリオと、入力シナリオインデックスとが定義される。一方、統合プログラムの出力として、データｃ'と、データｄ'と、データ３．ｄと、出力シナリオインデックスとが定義される。入力ａ'と出力ｃ'、入力ｂ'と出力ｄ'とのそれぞれはスワップペアであり、部品プログラムの変更に際して入出力の交換が行われる。
【００７６】
上述したステップ（１）～（６）を実行するプログラム及び各ステップで使用されるデータ（少なくとも、各プログラムの入力及び出力を示す情報（入力、出力の番号）、入力及び出力のバッファサイズを示す情報、プログラム間の接続（出力と入力との接続）を示す情報）は、図１に示した補助記憶装置１３に記憶される。ＣＰＵ１１は、当該プログラムを主記憶装置１２にロードしてステップ（１）～（６）を実行することにより、統合プログラムを自動的に作成することができる。
【００７７】
図９は、ＧＰＵ用のプログラム（例えば、Open CL）によって記述された統合プログラムの一例を示す。図９の一行目は、ＧＰＵ１８が備える複数のプロセッサのうち、プログラムを実行するプロセッサ番号（プロセッサＩＤ）を特定する。ここでは、プロセッサバ番号“０”のプロセッサが指定されている。
【００７８】
次の行は、スイッチ指示を示す。すなわち、シナリオとして、カーネルプログラムＡ，

Ｂ，Ｃを、Ａ→Ｂ→Ｃの順で、シナリオインデックスのインクリメントに応じて、実行対象のカーネルプログラムを切り替えることで実行することを指示する。その次には、異なる演算を行うカーネルプログラムＡ，Ｂ，及びＣが記述されており、例えば、カーネルプログラムＡは、入力データｄ，ｅ，ｆ及びＡｐによって所定の演算を行うことが指示されている。カーネルプログラムの次には、カーネルプログラムＡ，Ｂ，Ｃが同一のプロセッサで実行されることを指定している。そして、最終行は、シナリオインデックス値のインクリメントの指示である。このような統合プログラムが、ＣＰＵ１１のコンパイルによって主記憶装置１２上に展開され、ＶＲＡＭ１７にマッピングされる。

【００７９】

図１０は、フローモデル（flow-model）と呼ばれる、パイプライン処理のモデルを定義したモデル定義情報と、所定の記述言語で記述されたパイプライン処理で行われる演算を指示するプログラム（カーネルプログラム）とが所定の記述言語で記述されたファイルの例を示す。フローモデルのファイルは、補助記憶装置１３に記憶される。ファイル形式として、図１０に示すように、例えばＸＭＬが適用される。

【００８０】

図１０に示すように、フローモデルは、入力データストリーム（例えばａ及びｂ）の定義と、出力データストリーム（例えばｃ）の定義とを含む。さらに、カーネルプログラムの記述部分を含む。当該記述部分に、図９に示したような複数のカーネルプログラムがパッキングされた統合プログラム（単一プログラム）が記述される。さらに、フローモデルは、ターゲット機能の名称の記述部分と、下位レイヤのランタイム（Runtime、例えば、ＧＰＵで実行されるOpenCL）を特定する部分と、スレッドブロック及び１つのスレッドのサイズの定義部分とを含む。

【００８１】

図１１は、“CarSh（カーシュ）”と呼ばれる、上記したフローモデルを実行形式に変換するプログラム（実行支援プログラム）によって変換された統合プログラムの実行形式の例を示す。実行形式において、図１０に示したようなフローモデルのファイルが特定される。さらに、フローモデルに含まれたカーネルプログラム（統合プログラム）の実行に際して使用されるデータを含んだデータファイルが定義される。

【００８２】

さらに、フローモデルにおける入力（Input）及び出力（Output）の後に、スワップペアの定義部分（<SwapPair>）が含まれる。スワップペアとして、入力シナリオインデックスと出力シナリオインデックスの他、上述した出力データと入力データとのスワップペアが定義される。また、スワップペアの定義部分には、シナリオにおける入出力のスワップ回数を定義することもできる。例えば、自然数Ｎ個のカーネルプログラムが統合されている場合には、Ｎ－１回のスワップ回数が指定されることで、すべての出力データのタイミングを合わせることができる。

【００８３】

ＣＰＵ１１は、実行支援プログラムの実行時に、例えば入力装置１４から入力されるフローモデルの指定情報を受け付け、指定情報に応じたフローモデルのファイルを補助記憶装置１３から読みだすとともに、フローモデルファイルに応じたデータファイルを補助記憶装置１３から読みだす。

【００８４】

続いてＣＰＵ１１は、フローモデルのファイルを用いて、図１１に示したような統合プログラムの実行形式を生成し、カーネルプログラムのコンパイル、ＶＲＡＭ１７へのマッピング（ダウンロード）、データファイルを用いたＶＲＡＭ１７への入出力セッティングを行う。マッピング及び入出力セッティングが終了すると、ＣＰＵ１１は、カーネルプログラム（統合プログラム）の実行をＧＰＵ１８に指示する。

【００８５】

ＧＰＵ１８は、カーネルプログラム（統合プログラム）の実行によって、Ｎ回、ＣＰＵ１１とのデータ転送を行うことなく、複数のカーネルプログラムを実行することができる

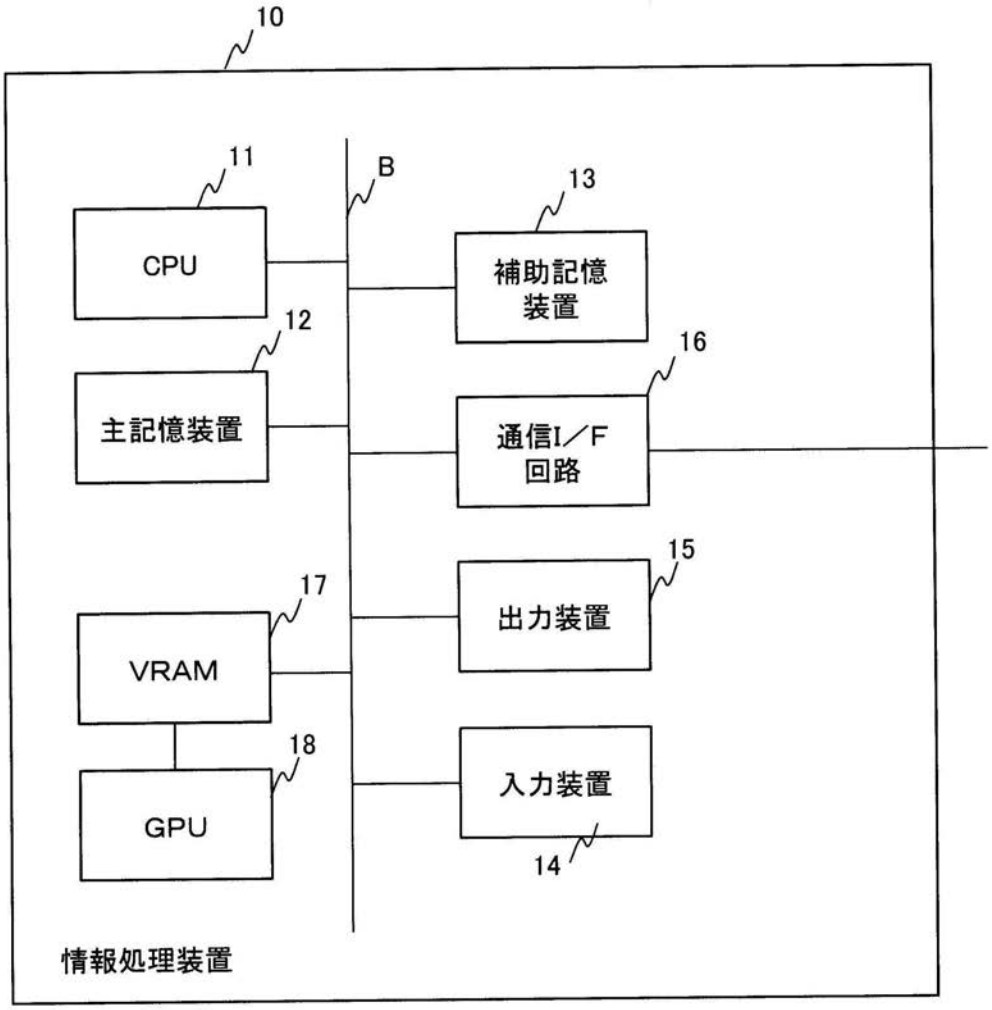
。したがって、統合プログラムの最終的な出力を得るまでの時間の短縮化を図り、ＧＰＵ１８の性能を十分に引き出すことができる。

【符号の説明】

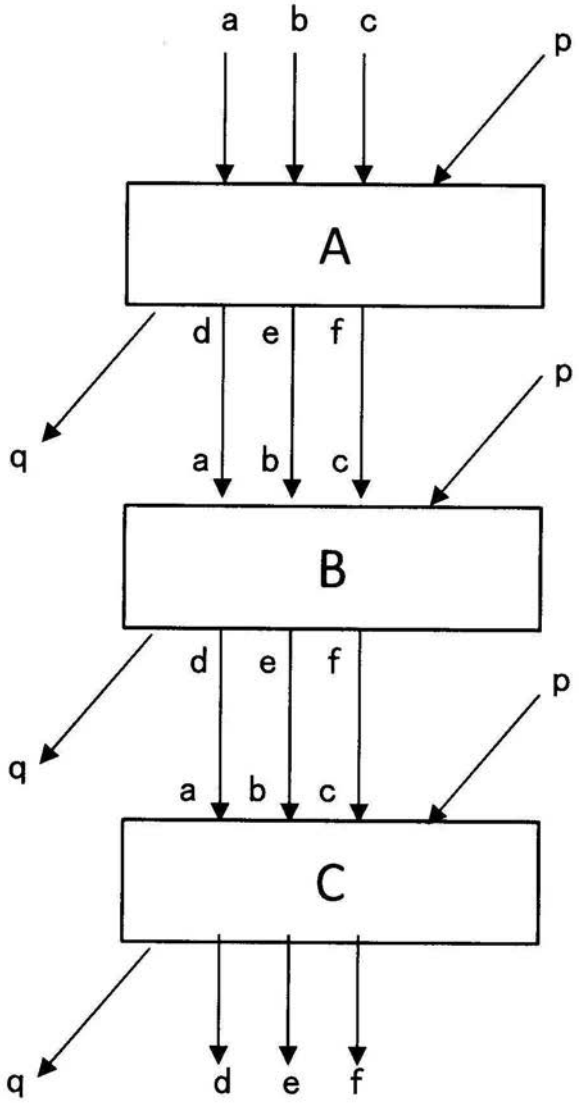
【００８６】

１０・・・情報処理装置（コンピュータ）
１１・・・ＣＰＵ（プロセッサ）
１２・・・主記憶装置
１３・・・補助記憶装置
１４・・・入力装置
１５・・・出力装置
１６・・・通信Ｉ／Ｆ
１７・・・ＶＲＡＭ（記憶装置）
１８・・・ＧＰＵ（アクセラレータ）

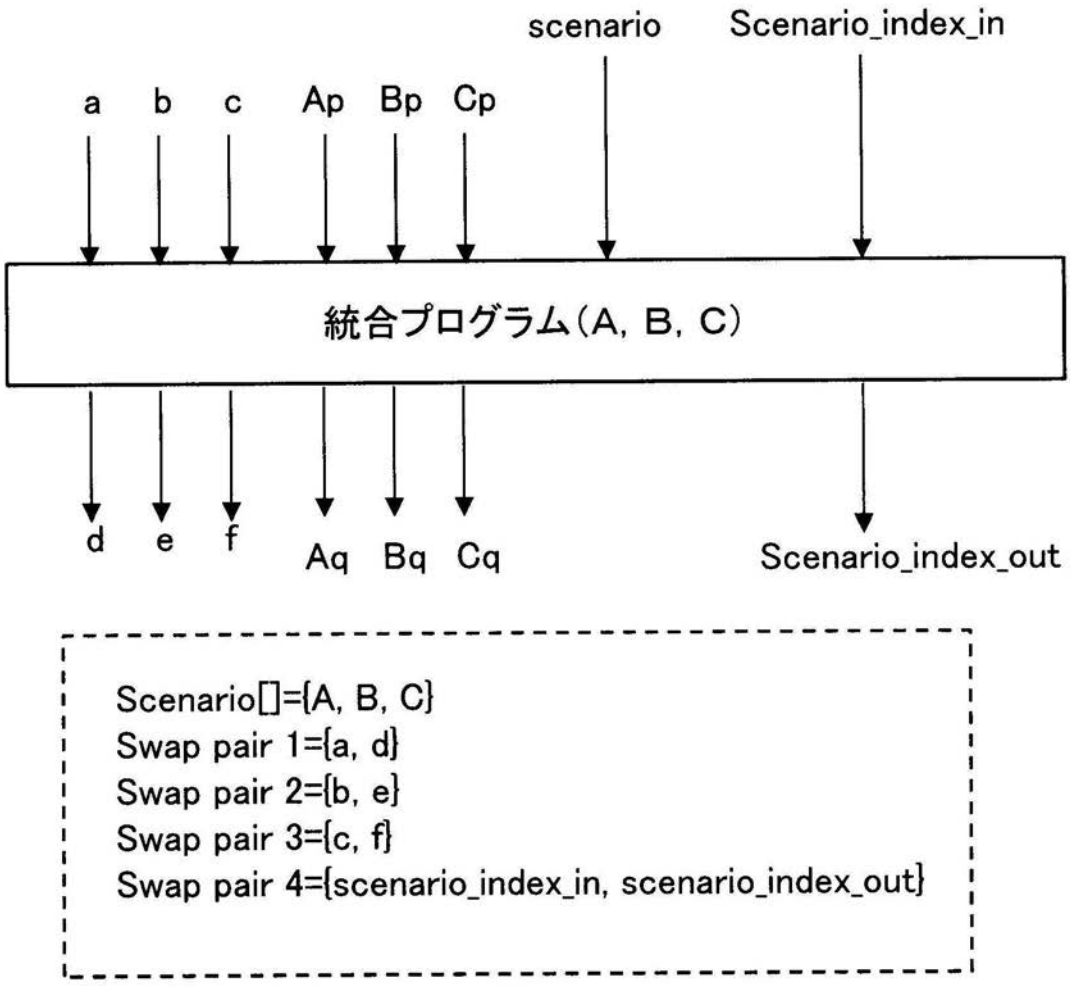
【図１】


10
11
B
13
CPU
補助記憶装置
12
16
主記憶装置
通信I／F回路
15
17
出力装置
VRAM
18
入力装置
GPU
14
情報処理装置

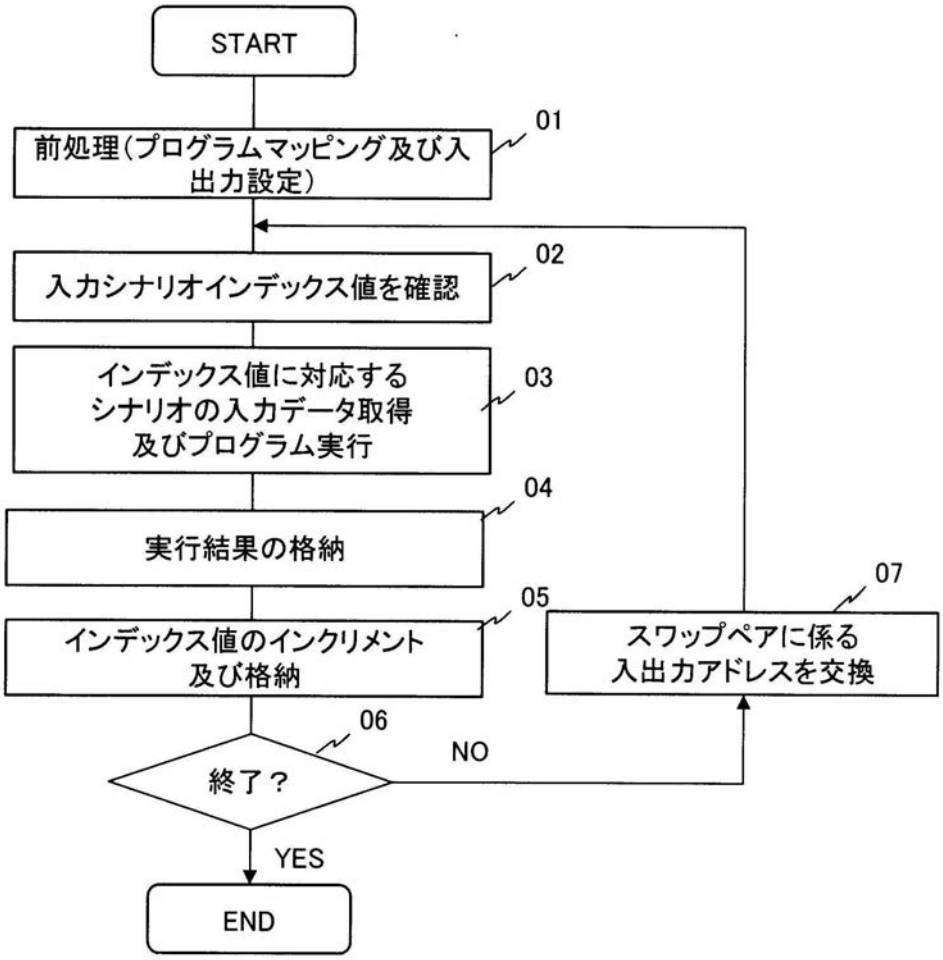
【図２】



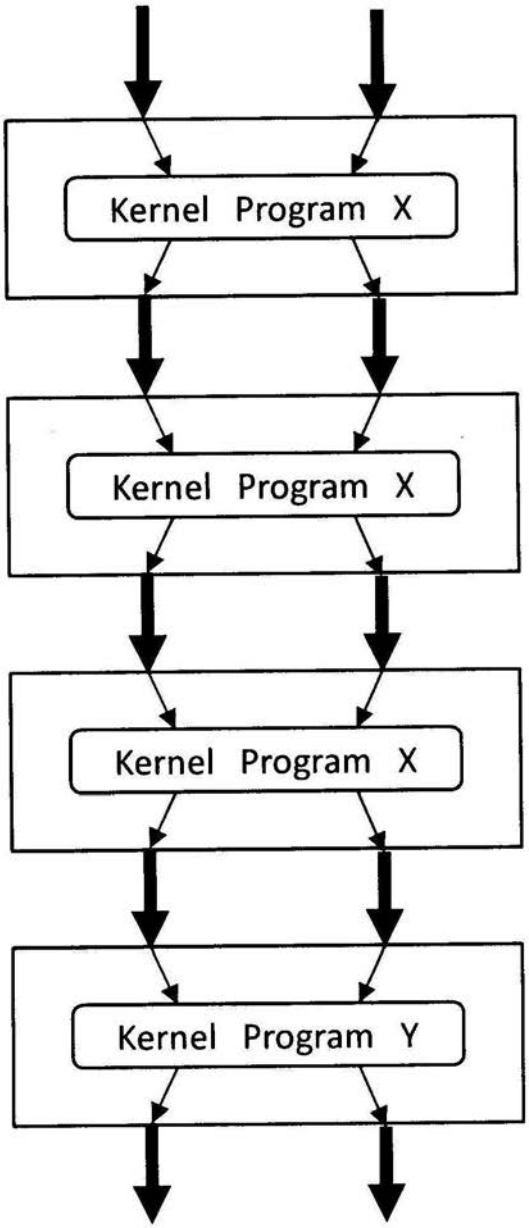
【図３】


scenario
Scenario_index_in
a
b
c
Ap
Bp
Cp
統合プログラム(A, B, C)
d
e
f
Aq
Bq
Cq
Scenario_index_out
Scenario[]={A, B, C}
Swap pair 1={a, d}
Swap pair 2={b, e}
Swap pair 3={c, f}
Swap pair 4={scenario_index_in, scenario_index_out}

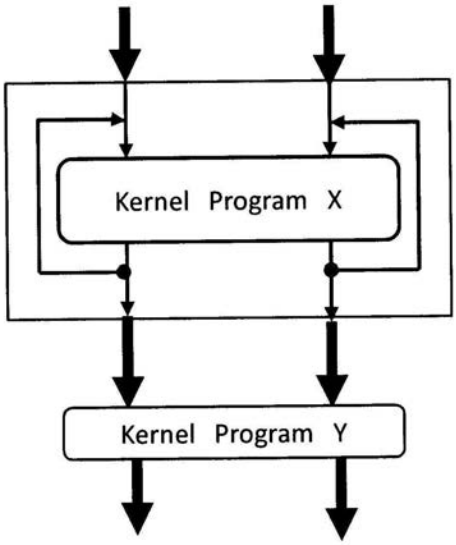
【図４】


START
前処理(プログラムマッピング及び入出力設定)
01
入力シナリオインデックス値を確認
02
インデックス値に対応するシナリオの入力データ取得及びプログラム実行
03
実行結果の格納
04
インデックス値のインクリメント及び格納
05
スワップペアに係る入出力アドレスを交換
07
06
終了？
NO
YES
END

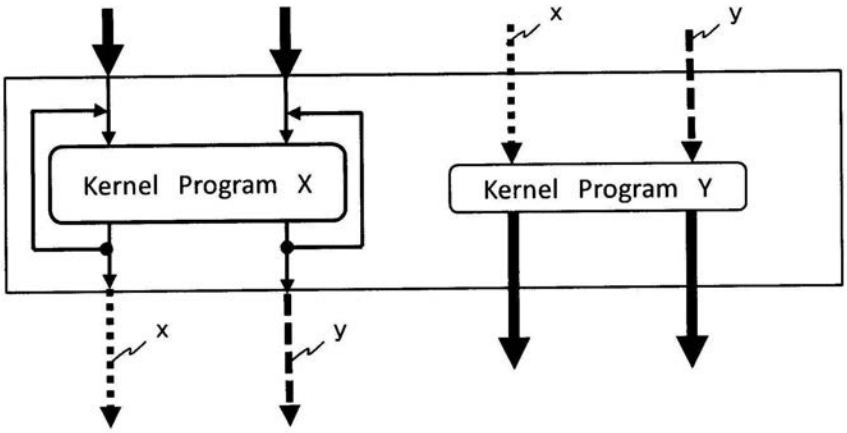
【図５】


Kernel Program X
Kernel Program X
Kernel Program X
Kernel Program Y

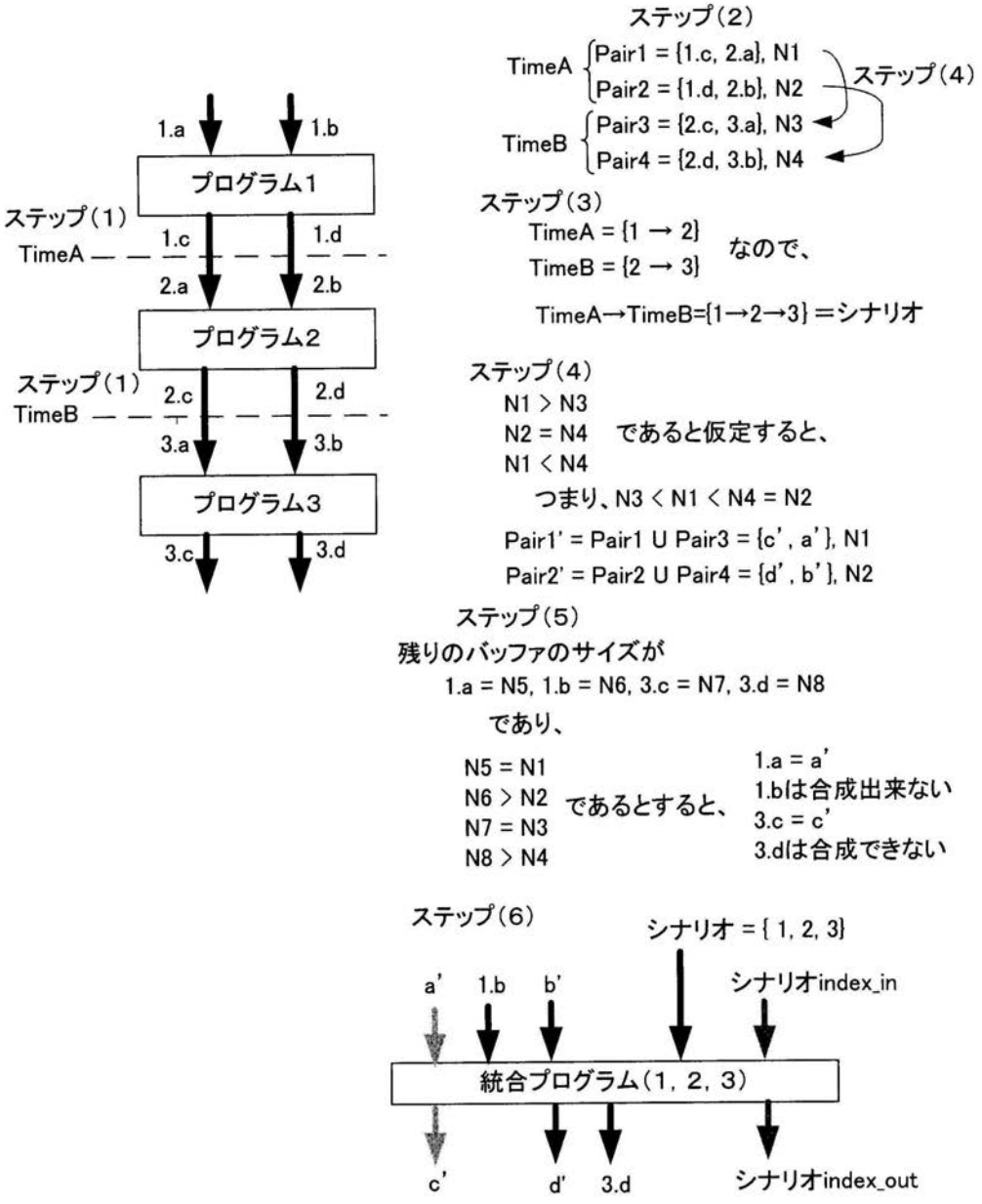
【図６】


Kernel Program X
Kernel Program Y

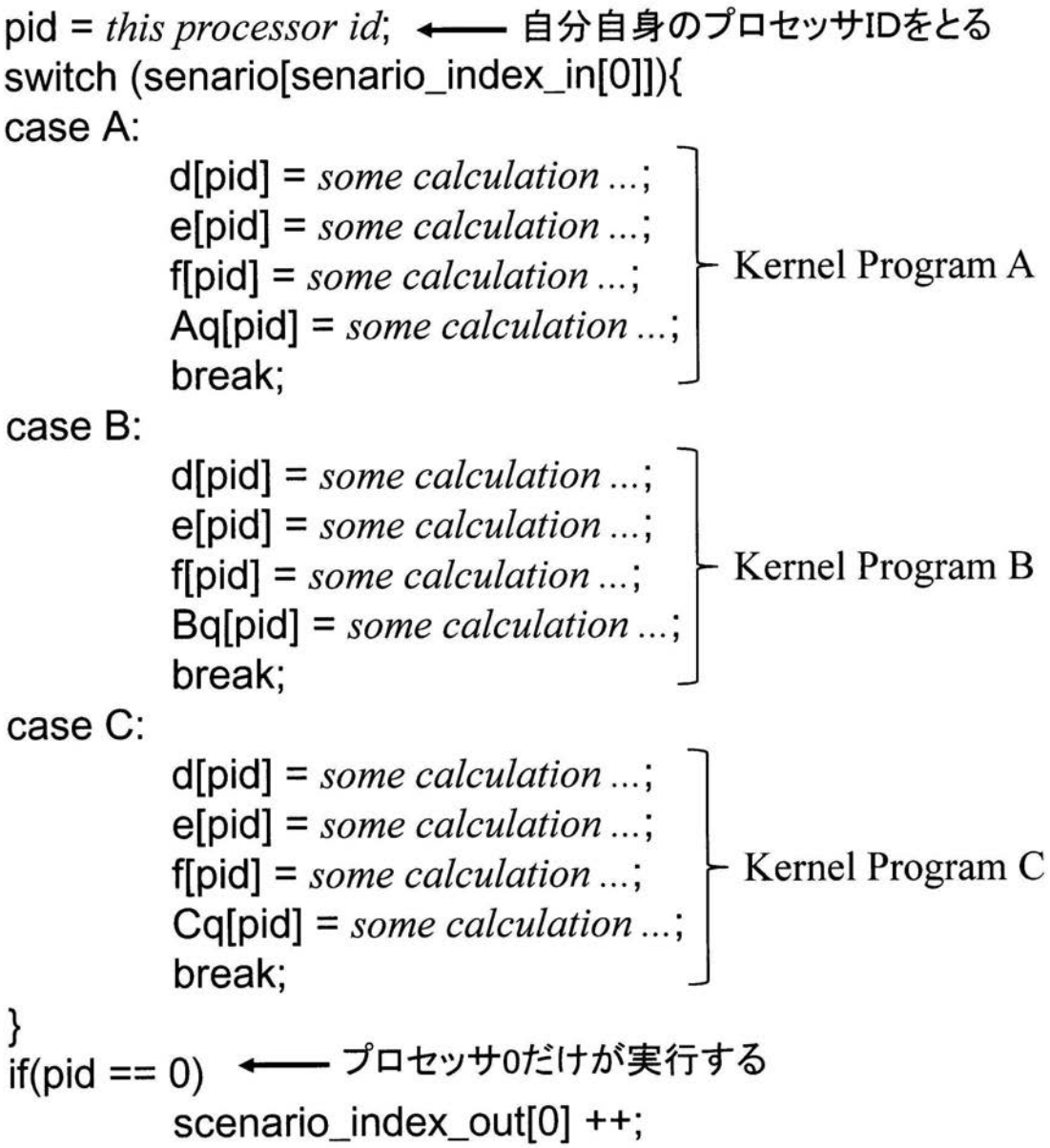
【図７】


Kernel Program X
Kernel Program Y
x
y
x
y

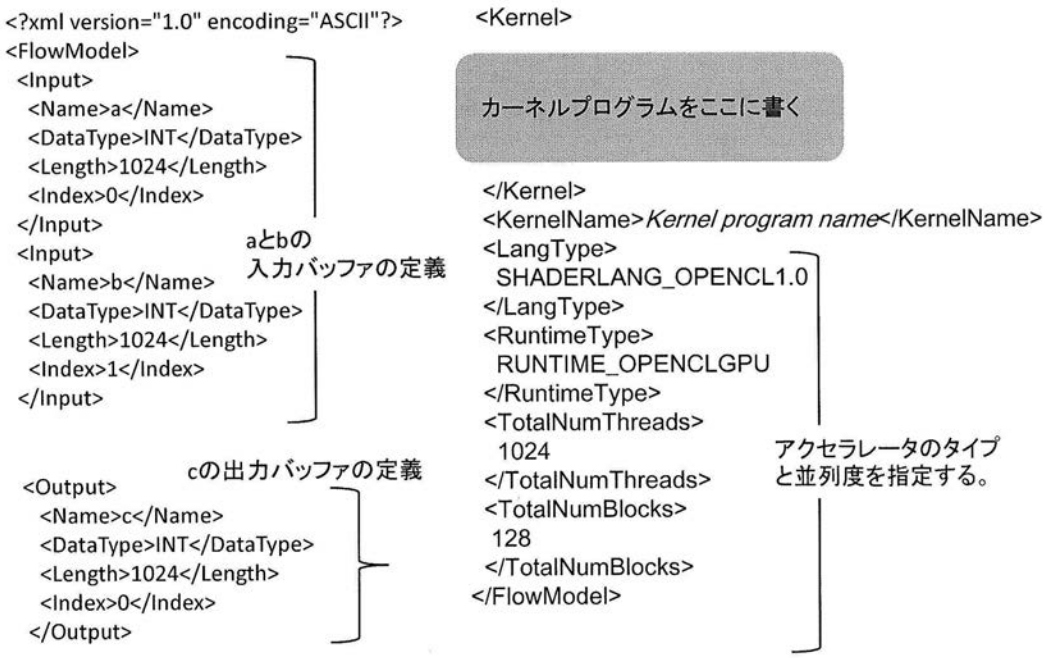
【図８】


ステップ(2)
TimeA
Pair1 = {1.c, 2.a}, N1
Pair2 = {1.d, 2.b}, N2
ステップ(4)
TimeB
Pair3 = {2.c, 3.a}, N3
Pair4 = {2.d, 3.b}, N4
1.a
1.b
プログラム1
ステップ(1)
TimeA
1.c
1.d
2.a
2.b
プログラム2
ステップ(1)
TimeB
2.c
2.d
3.a
3.b
プログラム3
3.c
3.d
ステップ(3)
TimeA = {1 → 2}
TimeB = {2 → 3}
なので、
TimeA→TimeB={1→2→3}＝シナリオ
ステップ(4)
N1 > N3
N2 = N4 であると仮定すると、
N1 < N4
つまり、N3 < N1 < N4 = N2
Pair1' = Pair1 U Pair3 = {c', a'}, N1
Pair2' = Pair2 U Pair4 = {d', b'}, N2
ステップ(5)
残りのバッファのサイズが
1.a = N5, 1.b = N6, 3.c = N7, 3.d = N8
であり、
N5 = N1
N6 > N2
N7 = N3
N8 > N4
であるとすると、
1.a = a'
1.bは合成出来ない
3.c = c'
3.dは合成できない
ステップ(6)
シナリオ = { 1, 2, 3}
a'
1.b
b'
シナリオindex_in
統合プログラム(1, 2, 3)
c'
d'
3.d
シナリオindex_out

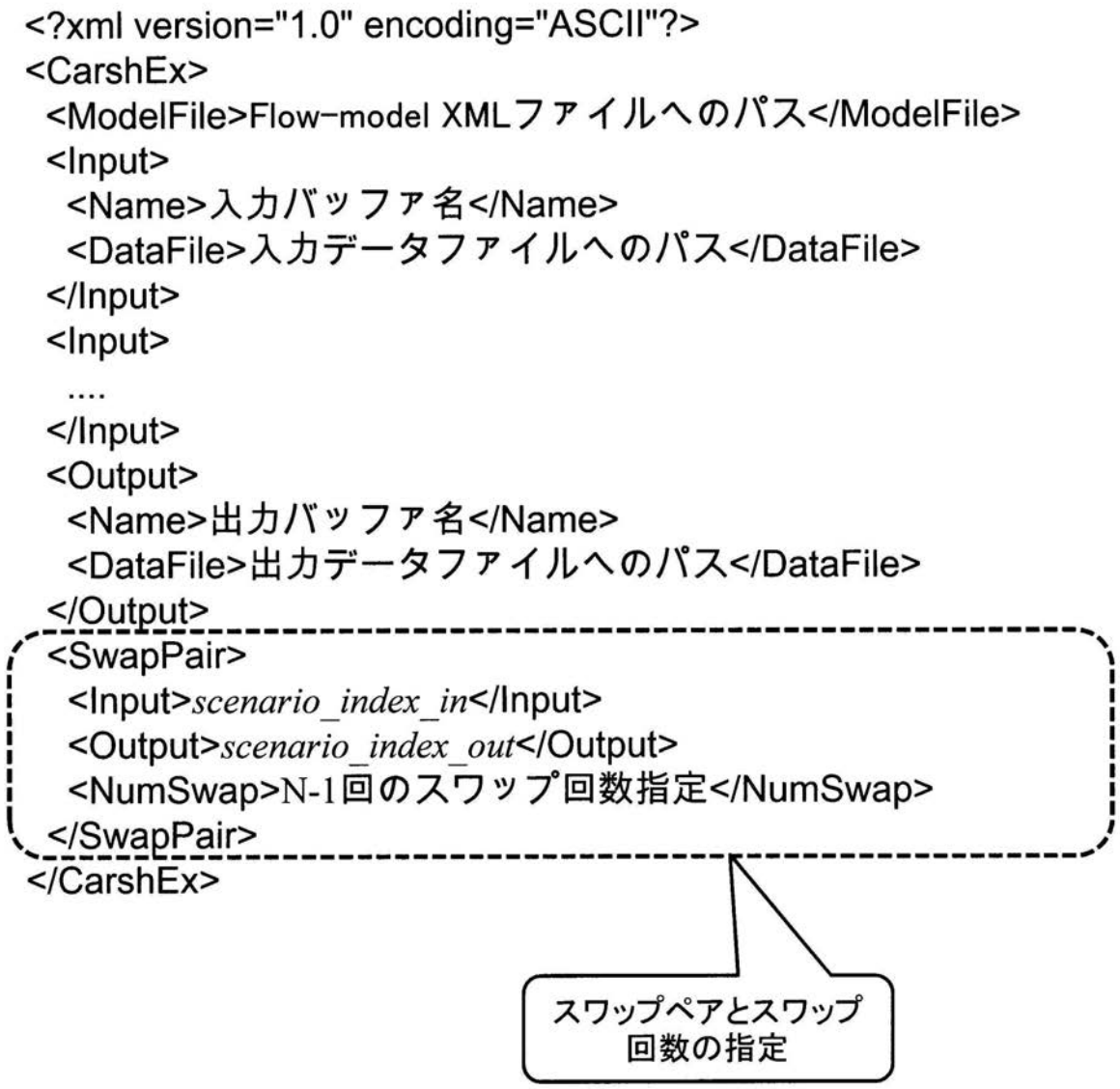
【図９】


pid = this processor id; ← 自分自身のプロセッサIDをとる
switch (senario[senario_index_in[0]]){
case A:
d[pid] = some calculation ...;
e[pid] = some calculation ...;
f[pid] = some calculation ...;
Aq[pid] = some calculation ...;
break;
Kernel Program A
case B:
d[pid] = some calculation ...;
e[pid] = some calculation ...;
f[pid] = some calculation ...;
Bq[pid] = some calculation ...;
break;
Kernel Program B
case C:
d[pid] = some calculation ...;
e[pid] = some calculation ...;
f[pid] = some calculation ...;
Cq[pid] = some calculation ...;
break;
Kernel Program C
}
if(pid == 0) ← プロセッサ0だけが実行する
scenario_index_out[0] ++;

【図１０】


<?xml version="1.0" encoding="ASCII"?>
<FlowModel>
<Input>
<Name>a</Name>
<DataType>INT</DataType>
<Length>1024</Length>
<Index>0</Index>
</Input>
<Input>
<Name>b</Name>
<DataType>INT</DataType>
<Length>1024</Length>
<Index>1</Index>
</Input>
aとbの
入力バッファの定義
cの出力バッファの定義
<Output>
<Name>c</Name>
<DataType>INT</DataType>
<Length>1024</Length>
<Index>0</Index>
</Output>
<Kernel>
カーネルプログラムをここに書く
</Kernel>
<KernelName>Kernel program name</KernelName>
<LangType>
SHADERLANG_OPENCL1.0
</LangType>
<RuntimeType>
RUNTIME_OPENCLGPU
</RuntimeType>
<TotalNumThreads>
1024
</TotalNumThreads>
<TotalNumBlocks>
128
</TotalNumBlocks>
</FlowModel>
アクセラレータのタイプ
と並列度を指定する。

【図１１】


<?xml version="1.0" encoding="ASCII"?>
<CarshEx>
<ModelFile>Flow-model XMLファイルへのパス</ModelFile>
<Input>
<Name>入力バッファ名</Name>
<DataFile>入力データファイルへのパス</DataFile>
</Input>
<Input>
....
</Input>
<Output>
<Name>出力バッファ名</Name>
<DataFile>出力データファイルへのパス</DataFile>
</Output>
<SwapPair>
<Input>scenario_index_in</Input>
<Output>scenario_index_out</Output>
<NumSwap>N-1回のスワップ回数指定</NumSwap>
</SwapPair>
</CarshEx>
スワップペアとスワップ
回数の指定