(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 809 230**

(51) Int. CI.:

G06F 8/41 (2008.01)
G06F 9/50 (2006.01)
G06F 9/455 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **30.03.2015 PCT/EP2015/056933**

(87) Fecha y número de publicación internacional: **08.10.2015 WO15150342**

(96) Fecha de presentación y número de la solicitud europea: **30.03.2015 E 15713472 (7)**

(97) Fecha y número de publicación de la concesión europea: **29.04.2020 EP 3126971**

(54) Título: **Ejecución del programa sobre plataforma heterogénea**

(30) Prioridad:

30.03.2014 EP 14162520

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
03.03.2021

(73) Titular/es:

UNIVERSITEIT GENT (50.0%)
Sint-Pietersnieuwstraat 25
9000 Gent, BE y
IMEC VZW (50.0%)

(72) Inventor/es:

GOOSSENS, BART

(74) Agente/Representante:

LEHMANN NOVO, María Isabel

DESCRIPCIÓN

Ejecución del programa sobre plataforma heterogénea

CAMPO DE LA INVENCIÓN

La invención se refiere al campo de la ejecución de programas informáticos en plataformas informáticas con al menos dos unidades de ejecución diferentes que tienen una memoria con una ubicación de memoria diferente, tal como, por ejemplo, una CPU y una GPU. Más concretamente, se refiere a un método puesto en práctica por ordenador para ejecutar una secuencia de instrucciones, p. ej., un programa informático, en dicha plataforma informática y productos de programas informáticos relacionados con dicho método puesto en práctica por ordenador.

ANTECEDENTES DE LA INVENCIÓN

Las plataformas informáticas heterogéneas actualmente comprenden una pluralidad de unidades de ejecución que tienen propiedades de procesamiento mutuamente distintas, tales tal como diferentes conjuntos de instrucciones de bajo nivel, p. ej., diferentes codificaciones de instrucciones de lenguaje máquina, diferentes sistemas de organización de memoria, diferentes capacidades de procesamiento y/o diferentes características de rendimiento dependiendo de las instrucciones de bajo nivel y/o tareas ejecutada de alto nivel. Por ejemplo, una plataforma informática heterogénea puede comprender, al menos, un núcleo de unidad central de procesamiento (CPU) que puede ser particularmente adecuado para realizar una amplia variedad de tareas informáticas complejas y para la ejecución en serie de instrucciones a alta velocidad, p. ej., instrucciones formateadas en un lenguaje máquina familiar del conjunto de instrucciones x86. La pluralidad de unidades de ejecución también puede comprender, al menos, un núcleo de unidad de procesamiento de gráficos (GPU), que puede ser particularmente adecuado para realizar un gran número de operaciones relativamente simples en paralelo. La pluralidad de unidades de ejecución también puede comprender un coprocesador para complementar las funciones de una CPU primaria, p. ej., un coprocesador específicamente adaptado para el procesamiento rápido de la señal, el cifrado o la interfaz de entrada/salida (E/S). La pluralidad de unidades de ejecución también puede comprender una matriz de compuertas lógicas programables en campo (FPGA) adaptadas para la reconfiguración de hardware en tiempo de ejecución, p. ej., utilizando un lenguaje de descripción de hardware (HDL). Aunque la plataforma informática heterogénea puede integrarse en un único sistema informático, p. ej., un ordenador personal que comprende una CPU, p. ej., una CPU multinúcleo y al menos una GPU, la plataforma informática heterogénea también puede comprender un sistema informático distribuido, p. ej., que comprende una pluralidad de ordenadores que tienen configuraciones idénticas o distintas, por ejemplo, cada una de las cuales comprende una CPU y una GPU, siendo las CPUs y las GPUs no necesariamente idénticas o compatibles sobre la pluralidad de ordenadores.

Se han descrito varios métodos para gestionar la programación y la ejecución de programas en una plataforma informática heterogénea.

La patente de los Estados Unidos US 8.225.300 da a conocer un método que comprende recibir un programa que incluye uno de entre una construcción paralela o una construcción distribuida, crear un componente objetivo del programa e integrar el componente objetivo en un entorno objetivo para obtener un programa cliente que sea ejecutable en múltiples plataformas de servidor heterogéneas, incluyendo un conjunto no homogéneo de recursos informáticos completos de Turing capaces de comunicarse entre sí. Una o más tareas se distribuyen de manera automática a través de las plataformas de servidor heterogéneas en función de una demanda de procesamiento de tareas. Un entorno de cliente puede proporcionar demandas de procesamiento de recursos y/o tareas a un planificador/administrador de trabajos. El planificador/administrador de trabajos puede determinar una estrategia de asignación para las demandas de procesamiento de recursos/tareas en función de los recursos de hardware disponibles y las demandas de procesamiento de recursos/tareas. Por ejemplo, el planificador/administrador de trabajos puede determinar un subconjunto de unidades de ejecución disponibles que son capaces de ejecutar una demanda de procesamiento de recursos/tareas, y puede asignar, de manera arbitraria, una unidad de este conjunto a la demanda. Sin embargo, es una desventaja de este método que, aunque una unidad de ejecución disponible se asigna de manera automática para gestionar una demanda que tiene las capacidades requeridas, el método dado a conocer puede seleccionar, p. ej., de forma aleatoria, una unidad de ejecución que tiene un bajo rendimiento para ejecutar la tarea en cuestión.

La patente de los Estados Unidos US 8.527.973 también da a conocer un método que comprende recibir un programa creado por un entorno informático técnico, analizar el programa, generar múltiples partes de programa basadas en el análisis del programa, asignar de manera dinámica las partes de múltiples programas a múltiples unidades de software de ejecución para programación paralela, recibir múltiples resultados asociados con las múltiples partes de programa desde las múltiples unidades de ejecución de software, y proporcionar los múltiples resultados o un resultado único al programa.

El documento US2013/0050229 da a conocer un terminal, provisto de una unidad central de procesamiento y de una unidad de procesamiento de gráficos, que realiza un método o ejecuta aplicaciones en adaptación a la carga de la CPU y de la GPU. El documento US2008/0276262 da a conocer un método y un aparato que planifican una pluralidad

de ejecutables en una cola de espera de planificación para su ejecución en uno o más dispositivos informáticos físicos tales tal como CPUs o GPUs de manera simultánea. El documento US2012/0317556 da a conocer métodos, sistemas y productos de programas informáticos para optimizar la ejecución de núcleos, incluyendo marcos de optimización para optimizar el tiempo de ejecución de los núcleos. El documento US2013/0160016 da a conocer un sistema y un método para asignar, de manera óptima, los núcleos informáticos a diferentes tipos de procesadores, tales como CPUs y GPUs en un sistema informático heterogéneo.

SUMARIO DE LA INVENCIÓN

Un objeto de las formas de realización de la presente invención es proporcionar medios y métodos adecuados para ejecutar un programa informático en una plataforma informática heterogénea.

El objetivo anteriormente mencionado se logra mediante un método puesto en práctica por ordenador y un primer, un segundo producto de programa informático y un soporte de datos de conformidad con las reivindicaciones independientes.

La presente invención se refiere a un método puesto en práctica por ordenador que comprende: obtener un objeto de código informático intermedio que comprende, al menos, un conjunto de instrucciones correspondientes a una tarea a realizar, comprendiendo el objeto de código informático intermedio, además, para cada uno de dicho al menos un conjunto de instrucciones, uno o más descriptores de metadatos representativos de al menos una medida de complejidad de dicha tarea a realizar, siendo el objeto de código informático intermedio independiente de la máquina y estando dicha medida de complejidad basada en al menos un análisis del código informático intermedio, y

- ejecutar, en tiempo de ejecución, dicho objeto de código informático intermedio en una plataforma informática que comprende, al menos, dos unidades de ejecución diferentes que tienen una memoria con una ubicación de memoria diferente, comprendiendo dicha ejecución en tiempo de ejecución seleccionar, para cada uno de dichos al menos un conjunto de instrucciones, un unidad de ejecución objetivo de entre dicha pluralidad de unidades de ejecución, teniendo en cuenta dicha selección los uno o más descriptores de metadatos y una regla de decisión que relaciona dicha pluralidad de medidas de complejidad con una característica de rendimiento de la pluralidad de unidades de ejecución, comprendiendo dicha ejecución del objeto de código informático intermedio traducir cada uno de los al menos un conjunto de instrucciones a un formato de nivel máquina ejecutable por la unidad de ejecución objetivo correspondiente.

Una ventaja de las formas de realización de la presente invención es que los descriptores de metadatos se generan a nivel de compilador y están incluidos inherentemente en el objeto de código informático intermedio y que estos descriptores de metadatos se utilizan para influir en las decisiones de tiempo de ejecución. Otra ventaja de las formas de realización de la presente invención es que los parámetros de tiempo de compilación (descriptores de metadatos) extraídos del lenguaje de programación de alto nivel utilizado y los parámetros de tiempo de ejecución disponibles en tiempo de ejecución se combinan para seleccionar el dispositivo más adecuado para ejecutar el objeto de código informático intermedio. Una ventaja de las formas de realización de la presente invención es que se obtiene un método y un sistema eficiente ya que la información de descriptores de metadatos requerida para la ejecución está inherentemente presente en el objeto de código informático intermedio y se suministra de manera automática para su ejecución.

Las al menos dos unidades de ejecución diferentes pueden ser una unidad central de procesamiento CPU y una unidad de procesamiento gráfico GPU.

La correspondiente pluralidad de medidas de complejidad se puede obtener al menos analizando el código intermedio.

Los uno o más descriptores de metadatos pueden ser representativos de una pluralidad correspondiente de medidas de complejidad de dicha tarea a realizar.

Las al menos dos unidades de ejecución diferentes pueden ser unidades de procesamiento gráfico GPU que tienen una memoria con una ubicación de memoria diferente.

El objeto de código informático intermedio puede obtenerse en un formato intermedio independiente de la unidad de ejecución. Una ventaja de las formas de realización de la presente invención es que el método permite una programación tal que la ejecución eficiente en una plataforma informática puede realizarse de manera automática.

La ejecución del objeto de código informático intermedio puede comprender determinar si un primer conjunto de dicho al menos un conjunto de instrucciones y un segundo conjunto de dicho al menos un conjunto de instrucciones puede ejecutarse de manera simultánea.

La ejecución del objeto de código informático intermedio puede comprender proporcionar una asignación de memoria automatizada para proporcionar datos para ser procesados por la ejecución de cada uno de dichos al menos un conjunto de instrucciones a la unidad de ejecución objetivo correspondiente. La asignación automática de memoria

por lo tanto, puede referirse al hecho de que no se requiere intervención del usuario para la asignación de memoria, es decir, que la asignación de memoria se produce de manera automática por el sistema.

La obtención del objeto de código informático intermedio puede comprender compilar el objeto de código informático intermedio a partir de un código de programa informático especificado de conformidad con una especificación de lenguaje de programación de alto nivel.

Conviene señalar que la compilación del código informático intermedio en el código dependiente de la máquina puede realizarse en tiempo de ejecución, pero que, sin embargo, esto no es necesario. Dicho de otro modo, la compilación del código informático intermedio en código dependiente de la máquina puede ocurrir no solamente en tiempo de ejecución.

La obtención del objeto de código informático intermedio puede comprender, además, para cada uno de dichos al menos un conjunto de instrucciones, determinar los uno o más descriptores de metadatos representativos de la pluralidad correspondiente de medidas de complejidad.

El uno o más descriptores de metadatos pueden ser uno o más parámetros determinables a nivel de compilación y que expresen una complejidad de una función núcleo del objeto de código informático intermedio.

La selección también puede tener en cuenta uno o más de entre una longitud de código, un producto de dimensiones de datos, un producto de dimensiones de bloque de GPU, un número total de bloques de GPU, un número de hilos de ejecución de CPU asignados, un tiempo de transferencia de memoria, una ocupación de GPU, un tamaño o carga de colas de espera de comandos de CPU y de GPU o una carga global de colas de espera de CPU y de GPU.

La presente invención también se refiere a un primer producto de programa informático para ejecutar un objeto de código informático intermedio, comprendiendo el primer producto de programa informático

un componente de entrada para obtener un objeto de código informático intermedio que comprende, al menos, un conjunto de instrucciones correspondientes a una tarea a realizar, comprendiendo, además, el objeto de código informático intermedio, para cada uno de dichos al menos un conjunto de instrucciones, uno o más descriptores de metadatos representativos de una pluralidad correspondiente de medidas de complejidad de dicha tarea a realizar, y un componente de tiempo de ejecución para ejecutar dicho objeto de código informático intermedio en una plataforma informática que comprende, al menos, dos unidades de ejecución diferentes que tienen una memoria con una ubicación de memoria diferente, en donde el componente de tiempo de ejecución comprende una unidad de selección para seleccionar, para cada uno de dichos al menos un conjunto de instrucciones, una unidad de ejecución objetivo desde dichas al menos dos unidades de ejecución, teniendo en cuenta dicha selección los uno o más descriptores de metadatos y una regla de decisión que relaciona dicha pluralidad de medida de complejidad con una característica de rendimiento de al menos dos unidades de ejecución diferentes, y dicho componente de tiempo de ejecución está programado para traducir cada una de los al menos un conjunto de instrucciones a un formato de nivel máquina ejecutable por la unidad de ejecución objetivo correspondiente.

El componente de tiempo de ejecución, además, puede comprender una unidad de gestión de memoria para la asignación automática de memoria.

El componente de tiempo de ejecución, además, puede comprender una unidad de planificación de tiempo de ejecución adaptada para determinar si un primer conjunto de dicho al menos un conjunto de instrucciones y un segundo conjunto de dicho al menos un conjunto de instrucciones puede ejecutarse de manera simultánea.

El primer producto de programa informático puede ponerse en práctica mediante un conjunto de instrucciones para ejecutar un objeto de código informático intermedio.

La presente invención también se refiere a un segundo producto de programa informático para generar un objeto de código informático intermedio, comprendiendo el segundo producto de programa informático:

- un componente de entrada para obtener un código de programa informático especificado de conformidad con una especificación de lenguaje de programación de alto nivel,

- un componente de compilación para compilar el código del programa informático en un objeto de código informático intermedio que comprende, al menos, un conjunto de instrucciones correspondientes a una tarea a realizar, y

- un componente de análisis para anotar cada uno de dichos al menos un conjunto de instrucciones con uno o más descriptores de metadatos representativos de una pluralidad correspondiente de medidas de complejidad de dicha tarea a realizar, estando la al menos una medida de complejidad basada en un análisis del código informático intermedio y siendo dicho objeto de código informático intermedio independiente de la máquina,

- en donde cuando el objeto de código informático intermedio generado es ejecutado por un producto de programa informático de conformidad con la reivindicación 10 en una plataforma informática que comprende, al menos, dos unidades de ejecución diferentes, se selecciona una unidad de ejecución objetivo desde dichas al menos dos unidades de ejecución teniendo en cuenta dichos uno o más descriptores de metadatos.

El componente de análisis puede adaptarse para determinar los uno o más descriptores de metadatos representativos de al menos una medida relacionada con bifurcaciones condicionales, saltos hacia atrás, asignación de memoria dinámica, llamadas de función indirectas y/o sincronización de hilos de ejecución.

El segundo producto de programa informático puede ponerse en práctica tal como un compilador de software.

La presente invención también se refiere a un soporte de datos que comprende un conjunto de instrucciones para, cuando se ejecuta en un ordenador, ejecutar un objeto de código informático intermedio en una plataforma informática que comprende, al menos, dos unidades de ejecución diferentes que tienen una memoria con una ubicación de memoria diferente de conformidad con un método tal como se describe con anterioridad.

Los aspectos particulares y preferidos de la invención se establecen en las reivindicaciones independientes y dependientes adjuntas. Las características de las reivindicaciones dependientes pueden combinarse con las características de las reivindicaciones independientes y con las características de otras reivindicaciones dependientes según corresponda y no simplemente tal como se establece explícitamente en las reivindicaciones.

Estos y otros aspectos de la invención serán evidentes a partir de, y se dilucidarán con referencia a, las formas de realización descritas a continuación.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

La Figura 1 ilustra un método, a modo de ejemplo, de conformidad con formas de realización de la presente invención.

La Figura 2 ilustra un primer producto de programa informático de conformidad con formas de realización de la presente invención.

La Figura 3 ilustra una forma de realización, a modo de ejemplo, de un primer producto de programa informático según la presente invención.

La Figura 4 ilustra un primer producto de programa informático según formas de realización de la presente invención.

La Figura 5 y la Figura 6 ilustran capturas de pantalla de puestas en práctica de reglas de decisiones, tal como pueden utilizarse en formas de realización ejemplo de la presente invención.

Los dibujos son solamente esquemáticos y no limitativos. En los dibujos, el tamaño de algunos de los elementos puede ser exagerado y no dibujado a escala con fines ilustrativos.

Cualquier signo de referencia en las reivindicaciones no se interpretará como limitativo del alcance.

En los diferentes dibujos, los mismos signos de referencia se refieren a elementos iguales o análogos.

DESCRIPCIÓN DETALLADA DE LAS FORMAS DE REALIZACIÓN ILUSTRATIVAS

La presente invención se describirá con respecto a formas de realización particulares y con referencia a determinados dibujos, pero la invención no está limitada a los mismos sino solamente por las reivindicaciones. Los dibujos descritos son solamente esquemáticos y no limitativos. En los dibujos, el tamaño de algunos de los elementos puede ser exagerado y no dibujado a escala con fines ilustrativos. Las dimensiones absolutas y las dimensiones relativas no corresponden a reducciones reales a la práctica de la invención.

Además, los términos primero, segundo y similares en la descripción y en las reivindicaciones, se utilizan para distinguir entre elementos similares y no necesariamente para describir una secuencia, ya sea temporal, espacial, de clasificación o de cualquier otra manera. Debe entenderse que los términos así utilizados son intercambiables en circunstancias apropiadas y que las formas de realización de la invención descritas en el presente documento pueden funcionar en otras secuencias distintas a las descritas o ilustradas en el mismo.

Además, los términos superior, inferior y similares en la descripción y en las reivindicaciones se utilizan con fines descriptivos y no necesariamente para describir posiciones relativas. Debe entenderse que los términos así utilizados son intercambiables en circunstancias apropiadas y que las formas de realización de la invención descritas en el presente documento pueden funcionar en otras orientaciones que las descritas o ilustradas en el presente documento.

Conviene señalar que el término "que comprende", utilizado en las reivindicaciones, no debe interpretarse tal como restringido a los medios enumerados a continuación; no excluye otros elementos o etapas. Por lo tanto, debe interpretarse como especificando la presencia de las características, números enteros, etapas o componentes indicados, pero no excluye la presencia o adición de una o más características, números enteros, etapas o componentes, o sus grupos. Por lo tanto, el alcance de la expresión "un dispositivo que comprende los medios A y B" no debe limitarse a los dispositivos que consisten únicamente en los componentes A y B. Significa que, con respecto a la presente invención, los únicos componentes pertinentes del dispositivo son A y B.

La referencia a lo largo de esta especificación a "una sola forma de realización" o "una forma de realización" significa que una función, estructura o característica particular descrita en relación con la forma de realización está incluida en al menos una forma de realización de la presente invención. Por lo tanto, la presencia de las expresiones "en una sola forma de realización" o "en una forma de realización" en varios lugares a lo largo de esta especificación no se refieren necesariamente a la misma forma de realización, sino que pueden no serlo. Además, las funciones, estructuras o características particulares pueden combinarse de cualquier manera adecuada, tal como sería evidente para un experto en esta técnica a partir de esta descripción, en una o más formas de realización.

De manera similar, debe apreciarse que, en la descripción de formas de realización a modo de ejemplo de la invención, algunas características de la invención a veces se agrupan en una sola forma de realización, figura o descripción de la misma con el fin de simplificar la divulgación y ayudar a comprender uno o más de los diversos aspectos inventivos. Sin embargo, este método de divulgación no debe interpretarse tal como un reflejo de la intención de que la invención reivindicada requiera más características de las que se mencionan expresamente en cada reivindicación. Por el contrario, tal como reflejan las siguientes reivindicaciones, los aspectos inventivos se encuentran en menos de todas las características de una sola forma de realización descrita con anterioridad. Por lo tanto, las reivindicaciones que siguen a la descripción detallada se incorporan expresamente en esta descripción detallada, y cada una de las reivindicaciones se presenta tal como una forma de realización separada de esta invención.

Además, aunque algunas formas de realización descritas en el presente documento incluyen algunas, pero no otras características incluidas en otras formas de realización, las combinaciones de características de diferentes formas de realización están destinadas a estar dentro del alcance de la invención y forman diferentes formas de realización, tal como lo entenderían los expertos en esta técnica. Por ejemplo, en las siguientes reivindicaciones, cualquiera de las formas de realización reivindicadas puede utilizarse en cualquier combinación.

En la descripción proporcionada en este documento, se exponen numerosos detalles específicos. Sin embargo, se entiende que las formas de realización de la invención se pueden practicar sin estos detalles específicos. En otros casos, los métodos, estructuras y técnicas bien conocidas no se han mostrado en detalle para no perjudicar la comprensión de esta descripción.

Cuando en las formas de realización de la presente invención se hace referencia al "tiempo de ejecución", se hace referencia al momento en que el programa se ejecuta en el ordenador, a diferencia del tiempo de compilación. Cuando en formas de realización de la presente invención se hace referencia a la ejecución, se hace referencia a la ejecución en tiempo de ejecución, a menos que se indique lo contrario.

Cuando en formas de realización de la presente invención se hace referencia a unidades de ejecución que tienen una memoria con una ubicación de memoria diferente, se puede hacer referencia a una CPU y GPU, una GPU y otro tipo de unidad de procesamiento, o dos GPUs, p. ej., parte de una red de GPU que tiene diferentes memorias con una ubicación de memoria diferente.

En un primer aspecto, la presente invención se refiere a un método puesto en práctica por ordenador. Este método comprende obtener un objeto de código informático intermedio que comprende, al menos, un conjunto de instrucciones correspondientes a una tarea a realizar, y para cada uno de dichos al menos un conjunto de instrucciones, una pluralidad de descriptores de metadatos representativos de una pluralidad correspondiente de medidas de complejidad de dicha tarea a realizar. El método comprende, además, ejecutar el objeto de código informático intermedio en la plataforma informática que comprende, al menos, dos unidades de ejecución con una memoria que tiene una ubicación de memoria diferente (es decir, las ubicaciones de memoria para las diferentes unidades de ejecución son diferentes). Dichos sistemas pueden comprender una pluralidad de unidades de ejecución en donde hay al menos una unidad de procesamiento gráfico GPU y al menos una unidad de procesamiento de un tipo diferente, p. ej., una unidad central de procesamiento CPU. Esta última normalmente puede denominarse plataformas informáticas heterogéneas. De manera alternativa, también se incluyen sistemas que comprenden al menos dos GPUs diferentes, por ejemplo, una red de GPU en donde de manera opcional no hay CPU presentes, por lo que las GPUs son unidades de ejecución con diferentes ubicaciones de memoria. Esta etapa de ejecutar el objeto de código informático intermedio comprende seleccionar para cada uno de dichos al menos un conjunto de instrucciones una unidad de ejecución objetivo de la pluralidad de unidades de ejecución. Esta selección, además, tiene en cuenta la pluralidad de descriptores de metadatos y una regla de decisión. La regla de decisión relaciona la pluralidad de medidas de complejidad con una característica de rendimiento de la pluralidad de unidades de ejecución.

Los detalles y ventajas adicionales de las etapas estándar y opcionales de un método puesto en práctica por ordenador de conformidad con al menos algunas formas de realización de la presente invención se describirán ahora de manera adicional con referencia a un método puesto en práctica por ordenador, a modo de ejemplo, y a los dibujos, no estando limitadas las formas de realización de la presente invención a esta circunstancia.

Con referencia a la Figura 1, se muestra un ejemplo de método 1 puesto en práctica en ordenador de conformidad con formas de realización de la presente invención. Este método puede ser un método para ejecutar operaciones de alto nivel, p. ej., operaciones paralelas de alto nivel, en plataformas informáticas que tienen al menos dos unidades de ejecución con una memoria que tiene una ubicación de memoria diferente, p. ej., en dispositivos informáticos heterogéneos. Dicha plataforma informática heterogénea puede comprender una pluralidad de unidades de ejecución, p. ej., una pluralidad de unidades de ejecución en donde al menos algunas son unidades de ejecución funcional y/o estructuralmente distintas, tal como, por ejemplo, un núcleo de unidad central de procesamiento (CPU) y una unidad de procesador de gráficos (GPU). Sin embargo, la plataforma informática también puede comprender una pluralidad de unidades de procesamiento gráfico, p. ej., conectadas a través de una red, en donde al menos dos unidades de procesamiento gráfico tienen una memoria con una ubicación de memoria diferente, p. ej., un sistema informático distribuido. La plataforma informática también puede comprender, por ejemplo, un coprocesador particularmente adecuado para realizar una tarea específica, p. ej., operaciones aritméticas de coma flotante o procesamiento de señales. La plataforma informática heterogénea también puede comprender, por ejemplo, una matriz de compuertas lógicas programables en campo (FPGA).

Según las formas de realización de la presente invención, la plataforma informática heterogénea puede consistir en un único ordenador que comprende, al menos, dos núcleos o puede consistir en un conjunto de ordenadores en donde el conjunto comprende, al menos, dos dispositivos informáticos. El ordenador individual o el conjunto de ordenadores puede comprender una o más unidades CPU y/o GPU que tienen propiedades diferentes, tal como, por ejemplo, GPU1 y GPU2 en donde ambas GPUs tienen propiedades diferentes. Las diferentes CPUs y/o GPUs pueden ser núcleos en diferentes dispositivos informáticos o en el mismo dispositivo informático. El dispositivo informático puede ser una CPU de un único núcleo o de múltiples núcleos.

El método 1 comprende una etapa de obtener 2 un objeto de código informático intermedio. El objeto de código informático intermedio puede obtenerse en un formato intermedio independiente de unidad de ejecución. El objeto de código informático intermedio puede codificar un programa informático, p. ej., un algoritmo para realizar una tarea automatizada en una plataforma informática. El método 1 puede ser particularmente adecuado para ejecutar un algoritmo iterativo que implica operaciones complejas y paralelizables, p. ej., el objeto de código informático intermedio puede codificar dicho algoritmo iterativo paralelo, por ejemplo, un algoritmo de procesamiento de señal, un algoritmo de procesamiento de imagen 2D o un algoritmo de procesamiento de imagen 3D. El objeto de código informático intermedio puede comprender una representación intermedia, por ejemplo, un lenguaje de transferencia de registro, una forma de asignación única estática, una representación de notación polaca inversa (RPN) de códigos de operación y referencias independientes de la plataforma, p. ej., referencias a estructuras de datos, funciones y procedimientos.

Este objeto de código informático intermedio, también denominado representación intermedia, comprende, al menos, un conjunto de instrucciones correspondientes a una tarea a realizar. El al menos un conjunto de instrucciones, p. ej., al menos una secuencia de instrucciones, puede comprender operaciones de alto nivel, p. ej., puede comprender una instrucción para realizar una operación de alto nivel tal como una operación de matriz, una operación de manipulación de imagen o una transformación de Fourier. Por ejemplo, una instrucción en el objeto de código informático intermedio puede indicar la ejecución de una multiplicación de matrices de valor real o complejo o aplicar una transformación goniométrica a cada elemento de un vector.

Para cada uno de los al menos un conjunto de instrucciones, p. ej., instrucciones que definen una función núcleo correspondiente a una tarea a realizar, p. ej., un algoritmo para una operación de matriz por elementos, el objeto de código informático intermedio también comprende una pluralidad de descriptores de metadatos representativos de una pluralidad correspondiente de medidas de complejidad de dicha tarea a realizar.

Las medidas de complejidad se pueden obtener en base al código informático intermedio.

El código intermedio es tal que el código es independiente de la máquina o dispositivo.

El objeto de código informático intermedio puede ser un formato intermedio independiente de unidad de ejecución, p. ej., puede ejecutarse en una pluralidad de dispositivos informáticos estructurales y funcionalmente diferentes. Por ejemplo, el al menos un conjunto de instrucciones correspondientes a una tarea a realizar puede comprender instrucciones que se especifican de manera agnóstica a nivel máquina. Dichas instrucciones pueden ser ejecutadas por un intérprete puesto en práctica por ordenador que traduce cada instrucción en un código informático adecuado para la ejecución en una unidad de ejecución específica. La decisión sobre qué dispositivo informático se ejecuta en el objeto de código de programa intermedio, se suele realizar en el tiempo de ejecución. Además, la pluralidad de descriptores de metadatos también se puede especificar de manera agnóstica a nivel máquina, p. ej., los descriptores pueden ser representativos de una pluralidad correspondiente de medidas de complejidad de la tarea a realizar

independientemente de las características de rendimiento y capacidades de una unidad de ejecución en donde se ejecuta la tarea correspondiente.

Por ejemplo, la pluralidad de descriptores de metadatos puede comprender un conjunto de valores de medidas de complejidad correspondientes determinadas para el conjunto de instrucciones a las que se atribuye. Dichos descriptores de metadatos pueden comprender variables indicadoras, variables enteras, variables de valor real o incluso variables estructuradas, p. ej., un puntero a un nodo en un árbol de clasificación jerárquica. Las medidas de complejidad pueden comprender medidas que son mutuamente independientes o proporcionar información al menos parcialmente complementaria, relacionada con diferentes aspectos de la complejidad informática del conjunto de instrucciones. La complejidad de una función núcleo se define en función de una serie de parámetros. La idea es que una función núcleo que utiliza, por ejemplo, bucles o sincronización de hilos de ejecución, generalmente necesita un tiempo de finalización más largo que las funciones núcleo que solamente consisten en un pequeño número de cálculos sin bucle. Por ejemplo, las medidas de complejidad pueden comprender información sobre estructuras de bucle, profundidad de bucle anidado, ejecución de código condicional, bifurcación, profundidad de bifurcación, asignación de memoria dinámica, saltos hacia atrás en el código durante la ejecución, llamadas a funciones indirectas o requisitos de sincronización tal como la sincronización entre hilos de ejecución. En formas de realización particulares, a nivel del compilador, se puede determinar la complejidad y se le puede asignar una puntuación, p. ej., entre 0 y 10. En un ejemplo, esto puede basarse, por ejemplo, en los siguientes parámetros:

(a) COMPLEXITY_BRANCHES (1): la función núcleo contiene sentencias if

(b) COMPLEXITY_TEN_STATEMENTS (2): la función núcleo tiene al menos 10 declaraciones

(c) COMPLEXITY_DYNAMIC_MEMORY (3): la función núcleo requiere asignación de memoria dinámica

(d) COMPLEXITY_JUMP_BACK (8): la función núcleo contiene un salto hacia atrás (generalmente un bucle)

(e) COMPLEXITY_INDIRECT_CALLS (9): la función núcleo realiza llamadas indirectas (por ejemplo, a través de punteros de función)

(f) COMPLEXITY_SYNCHRONIZATION (10): la función núcleo requiere sincronización de los hilos de ejecución

Como será evidente, lo que antecede es solamente un ejemplo y se pueden hacer diferentes selecciones.

Una ventaja de las formas de realización de la presente invención es que el objeto de código informático intermedio puede ejecutarse en una plataforma que comprende tipos de unidad de ejecución no considerados al compilar el objeto de código informático intermedio, p. ej., una nueva arquitectura de CPU o GPU. La decisión se toma así en tiempo de ejecución. Además, esta ejecución se puede realizar de manera eficiente simplemente adaptando el intérprete de tiempo de ejecución a este tipo de unidad de ejecución y proporcionando una regla de decisión adecuada. La regla o reglas de decisión permitirán o ayudarán a seleccionar qué unidad de ejecución se utilizará. Además, esta adaptación puede realizarse de manera ventajosa con independencia del programa informático a ejecutar.

El método comprende, además, ejecutar 4 el objeto de código informático intermedio en una plataforma informática que comprende, al menos, dos unidades de ejecución con una memoria que tiene una ubicación de memoria diferente. Esta etapa de ejecutar el objeto de código informático intermedio comprende seleccionar 6, para cada una de dichas al menos un conjunto de instrucciones, una unidad de ejecución objetivo de entre la pluralidad de unidades de ejecución. Esta selección, además, tiene en cuenta la pluralidad de descriptores de metadatos y una regla de decisión. La regla de decisión relaciona la pluralidad de medidas de complejidad con una característica de rendimiento de la pluralidad de unidades de ejecución. Por ejemplo, la regla de decisión puede adaptarse para predecir cuál de entre la pluralidad de unidades de ejecución funcionará mejor para realizar la tarea codificada por el conjunto de instrucciones, p. ej., qué unidad de ejecución tiene la mayor probabilidad de proporcionar el mejor rendimiento para la tarea en cuestión. La regla de decisión puede ser, por ejemplo, una expresión clasificadora, p. ej., proporcionar una división del espacio abarcado por la pluralidad de medidas de complejidad en zonas de rendimiento dominante para subconjuntos, p. ej., elementos, de la pluralidad de unidades de ejecución. Dicha regla de decisión puede obtenerse, por ejemplo, perfilando un conjunto de tareas de referencia, p. ej., para lo cual los puntos en el espacio definidos por las medidas de complejidad proporcionan un buen muestreo de este espacio y determinan para cada tarea de referencia la unidad de ejecución con mejor rendimiento. Una ventaja de las formas de realización de la presente invención es que se puede obtener un buen rendimiento en la ejecución de un programa sin requerir un perfil detallado de este programa, p. ej., ejecución manual o automática de componentes del programa, p. ej., diferentes conjuntos de instrucciones que codifican tareas constitutivas, en una pluralidad de unidades de ejecución con el fin de determinar la mejor solución.

Sin embargo, la selección también puede tener en cuenta un parámetro de tiempo de ejecución, por ejemplo, la dimensionalidad o el número de elementos de una estructura de datos en donde al menos un conjunto de instrucciones, p. ej., una función núcleo, opera. Por lo tanto, la regla de decisión puede ser, por ejemplo, una expresión clasificadora, p. ej., proporcionar una división del espacio conjunto abarcado por la pluralidad de medidas de complejidad y el al menos un parámetro de tiempo de ejecución en zonas de rendimiento dominante para

subconjuntos, p. ej., elementos, de la pluralidad de unidades de ejecución. El al menos un parámetro de tiempo de ejecución puede comprender, por ejemplo, una dimensionalidad, número de elementos y/o tipo de datos de una estructura de datos proporcionada tal como parámetro de entrada a al menos un conjunto de instrucciones, p. ej., a una función núcleo. El al menos un parámetro de tiempo de ejecución puede comprender una serie de hilos de ejecución o unidades paralelas asignadas a una solicitud de al menos un conjunto de instrucciones, p. ej., teniendo en cuenta los requisitos de memoria. En algunas formas de realización de la presente invención, la regla de decisión puede tener en cuenta, por ejemplo, además de una complejidad, uno o más o todos los siguientes parámetros:

- Longitud de código: describe el número de "instrucciones" (de nivel medio) que contiene la función núcleo. La longitud de código se define de manera independiente de la arquitectura. Conviene señalar que la "longitud de código" se suele utilizar a menudo tal como una heurística para la función en línea en compiladores modernos.

- Producto de las dimensiones de datos: se aplica una función núcleo a cada elemento de un conjunto de datos (por ejemplo, cada píxel de una imagen). El número total de elementos de datos es, por lo tanto, un indicador importante para el tiempo de cálculo de la función núcleo.

- Producto de las dimensiones del bloque GPU: cuando una GPU ejecuta una función núcleo, los datos se dividen en bloques y cada bloque se procesa de manera secuencial (o a menudo se mezcla secuencialmente/en paralelo por diferentes multiprocesadores). El tamaño del bloque es el resultado de un procedimiento de optimización diferente. Este puede ser un procedimiento numérico, un procedimiento analítico, puede basarse en la elaboración de perfiles, etc. La presente invención, por lo tanto, no está limitada por los procedimientos de optimización específicos utilizados.

- El número total de bloques de GPU: este número se obtiene por las dimensiones de datos dim, que corresponden a la dimensión de datos, divididas por las dimensiones de bloques blk dim, que corresponden a la dimensión de los bloques, es decir, al tamaño del bloque, y calculando el producto del resultado:

$$n = \left\lceil \frac{\dim_1}{blk\,\dim_1} \right\rceil .... \left\lceil \frac{\dim_D}{blk\,\dim_D} \right\rceil$$

donde D es la dimensionalidad de los datos.

- Número de hilos de ejecución de CPU asignados: debido a la granularidad de hilos de ejecución del sistema operativo, para tareas de peso liviano a menudo es más eficiente ejecutar la función núcleo en un núcleo de CPU en lugar de en todos los núcleos disponibles. La selección del número de hilos de ejecución de la CPU suele ser binaria (1 núcleo o todos los núcleos disponibles) y se realiza mediante una regla de decisión separada. El número de hilos de ejecución de la CPU también se puede seleccionar de manera dinámica (por ejemplo, en función de la carga actual de la CPU). Por lo tanto, el número de hilos de ejecución de CPU asignados también puede ser un indicador de si la CPU es una buena opción para la ejecución.

- Tiempo de transferencia de memoria: debido al sistema de memoria distribuida, puede ser necesario realizar transferencias de datos de la CPU a la GPU para ejecutar la función núcleo especificada en la GPU, o de GPU a CPU para ejecutar en la CPU. Debido a que el sistema de tiempo de ejecución sabe exactamente cuántos bytes de memoria deben transferirse, se puede estimar el tiempo de transferencia de memoria multiplicando el número de bytes por una tasa de transferencia media dependiente del dispositivo (agregando una pequeña constante que contiene la sobrecarga del nivel del controlador).

- Ocupación de GPU: definida tal como el número máximo de urdimbres activas en una GPU dividida por el número máximo de urdimbres soportadas, es un indicador de la utilización de la GPU. Por ejemplo, cuando se sabe de antemano que la ocupación de la GPU es bastante baja, puede ser más eficiente ejecutar la función núcleo en la CPU. Conviene señalar que la definición de ocupación de GPU aún ignora dos aspectos:

(a) La ejecución de diferentes funciones núcleo asíncronas en la GPU (solamente considera una función núcleo).

(b) Paralelismo de nivel de instrucción: en algunos casos (raros) la ocupación puede ser baja, pero el rendimiento puede ser alto. Sin embargo, debido a que la utilización de la GPU es difícil de predecir directamente, la ocupación de la GPU sigue siendo un parámetro útil.

La detección del paralelismo en el nivel de instrucción es mucho más difícil porque requiere herramientas de modelado analítico específicas del dispositivo (que no consideraremos en este documento). La ocupación de la GPU se puede calcular de la misma manera que en la hoja de cálculo Excel de NVidia Occupancy Calculator (que luego se ha integrado en los programas NVidia nSight Profiler y NVidia Visual Profiler).

- Tamaño o carga de las colas de espera de comandos de CPU y GPU. Debido al hecho de que la relación de rendimiento es suficientemente alta, por lo general suele ser más eficiente planificar una función núcleo en la cola

de espera de comandos de la GPU, incluso cuando esta cola está llena. Además, cuando la decisión sería ejecutar la función núcleo en la CPU, aún se necesitarían algunas transferencias de memoria. Estas transferencias de memoria tendrían que planificarse en la cola de espera de comandos de la GPU de todos modos. Para los dispositivos Fermi, la gestión en paralelo de las transferencias de memoria y de las funciones núcleo es bastante limitada (es decir, generalmente se realizan de forma secuencial o con solapamiento parcial). En consecuencia, suponiendo que al menos sea necesaria una transferencia de memoria, no tiene mucho sentido ejecutar una función núcleo en la CPU incluso cuando la GPU está ocupada. Sin embargo, tal como las GPUs avanzarán en el futuro, el efecto del parámetro puede volverse más pertinente.

La carga global de las colas de espera de comandos de la CPU y de la GPU: en lugar de contar la cantidad de funciones núcleo que esperan ser procesadas por la CPU o la GPU, también se puede tener en cuenta su complejidad de núcleos en esta métrica. Esto proporciona una predicción más precisa cuando se utilizan funciones núcleo heterogéneas.

Por lo tanto, en las formas de realización de conformidad con la presente invención, se puede lograr un buen rendimiento de ejecución de un bloque de código sin requerir una operación extensiva de creación de perfiles del bloque de código en una pluralidad de posibles unidades de ejecución, p. ej., una CPU y una GPU, pero también pueden proporcionar un buen rendimiento ajustado al tamaño, tipo y/o dimensionalidad de los datos de entrada determinados durante la ejecución. Por ejemplo, un fragmento de código, p. ej., una función núcleo, se puede ejecutar preferiblemente en la CPU de un ordenador cuando los datos de entrada son relativamente pequeños, p. ej., debido a una sobrecarga de programar la ejecución en una GPU, traduciendo las instrucciones a un formato compatible con GPU y/o transferencia de memoria a una memoria gráfica dedicada. Cuando se solicite el mismo fragmento de código con una estructura de datos de entrada más grande, se podría preferir la ejecución en la GPU, ya que las capacidades de procesamiento paralelo en la ejecución de la tarea superarían el coste general. Sin embargo, una tarea diferente podría ser más compleja y se beneficiaría de la selección de un dispositivo GPU en lugar de un dispositivo CPU en un umbral de tamaño de estructura de datos de entrada más pequeño. Además, debe tenerse en cuenta que la complejidad de dos tareas puede ser sustancialmente diferente en más de una manera, p. ej., una primera tarea podría implicar muchas condiciones de bifurcación, mientras que una segunda tarea podría implicar muchos bucles anidados, de modo que una representación multidimensional de los parámetros que influyen en la complejidad inherente de una tarea puede proporcionar buenos medios para determinar una unidad de ejecución adecuada sin requerir un conocimiento a priori, durante la preparación del objeto de código informático intermedio, con respecto a la combinación específica de unidades de ejecución en una plataforma informática en donde se va a ejecutar el código y las soluciones de compromiso de rendimiento asociadas con esta combinación.

La ejecución 4 del objeto de código informático intermedio también puede comprender determinar 5 si un primer conjunto de dicho al menos un conjunto de instrucciones y un segundo conjunto de dicho al menos un conjunto de instrucciones puede ejecutarse de manera simultánea, p. ej., teniendo en cuenta las dependencias de datos y las unidades de ejecución objetivo correspondientes. Por lo tanto, ejecutar 4 el objeto de código informático intermedio puede comprender una planificación de tiempo de ejecución con el fin de definir el orden en donde los conjuntos de al menos un conjunto de instrucciones se ejecutarán en cada una de las unidades de ejecución objetivo correspondientes.

La ejecución 4 del objeto de código informático intermedio puede comprender, además, proporcionar 7 una asignación de memoria automatizada, p. ej., uso compartido de memoria, duplicación de memoria y/o etiquetado no claro de copias de memoria, para proporcionar datos procesados por la ejecución de cada uno de dichos al menos un conjunto de instrucciones a la unidad de ejecución objetivo correspondiente. Por ejemplo, un primer conjunto de instrucciones del al menos un conjunto de instrucciones puede tener una primera unidad de ejecución seleccionada como unidad de ejecución objetivo, mientras que un segundo conjunto de instrucciones del al menos un conjunto de instrucciones puede tener una segunda unidad de ejecución seleccionada que difiere de la primera unidad de ejecución. La ejecución del objeto de código informático intermedio puede tener así en cuenta las dependencias de datos entre el primer conjunto de instrucciones y el segundo conjunto de instrucciones con el fin de copiar de manera automática los datos entre la memoria disponible para la primera unidad de ejecución y la segunda unidad de ejecución. Proporcionar 7 una asignación de memoria automatizada puede comprender el seguimiento de la copia de datos modificada más recientemente en las memorias disponibles para diferentes unidades de ejecución, con el fin de sincronizar dichas copias cuando sea necesario durante la ejecución. Esta asignación de memoria automatizada puede comprender, además, la conversión de datos automatizada entre formatos de datos específicos a nivel máquina, p. ej., para dar cuenta de las diferencias en la arquitectura de las unidades de ejecución. Por ejemplo, dicha conversión de datos automatizada puede comprender operaciones simples tales como cambiar una representación de primer bit más significativo (MSB) a una representación de primer bit menos significativo (LSB) u operaciones más complejas, tal como cambiar el orden en donde las dimensiones de la matriz son almacenadas internamente con el fin de hacer que las características de procesamiento ventajosas de una unidad de ejecución específica estén disponibles, o cambiar un formato de coma flotante que no sea compatible con una unidad de ejecución específica a un formato de menor precisión que sea compatible, o en un agregado de dichos formatos de menor precisión para evitar pérdida de información debido a errores de redondeo.

La ejecución 4 del objeto de código informático intermedio puede comprender, además, traducir 8 cada uno de al menos un conjunto de instrucciones a un formato de nivel máquina ejecutable por la unidad de ejecución objetivo correspondiente, p. ej., la unidad de ejecución objetivo seleccionada para el conjunto de instrucciones. Por ejemplo, esta traducción puede comprender una compilación justo a tiempo del conjunto de instrucciones en un formato de nivel máquina adecuado, ejecutando un programa de intérprete en el formato de nivel máquina adecuado para interpretar el conjunto de instrucciones, una combinación de las mismas o una alternativa de traducir un conjunto de instrucciones no nativas a un formato de nivel máquina nativo tal como se conoce en la técnica.

En un método de conformidad con las formas de realización de la presente invención, obtener 2 el objeto de código informático intermedio puede comprender compilar 11 el objeto de código informático intermedio a partir de un código de programa informático especificado de conformidad con una especificación de lenguaje de programación de alto nivel, p. ej., un lenguaje de programación que proporciona una fuerte abstracción de los detalles de la ejecución del programa por parte de una unidad de ejecución, p. ej., un procesador. Dicho lenguaje de programación de alto nivel puede, por ejemplo, utilizar elementos de lenguaje natural, puede adaptarse para facilitar su uso, puede automatizar y, preferiblemente, ocultar por completo, zonas significativas de programación de sistemas informáticos de bajo nivel, tales como la gestión de memoria y las operaciones de registro del procesador, y puede proporcionar un código legible y fácilmente comprensible.

La obtención 2 del objeto de código informático intermedio puede comprender, además, para cada uno de los al menos un conjunto de instrucciones, determinar 12 la pluralidad de descriptores de metadatos representativos de la correspondiente pluralidad de medidas de complejidad. Por ejemplo, un programa compilador o un programa de soporte para un compilador puede analizar el código del programa informático de alto nivel, determinar conjuntos de instrucciones correspondientes a la ejecución de tareas específicas, p. ej., tareas altamente paralelizables, y derivar una pluralidad de medidas de complejidad relacionadas con cada conjunto de instrucciones.

En un segundo aspecto, la presente invención también se refiere a un primer producto de programa informático. La Figura 2 ilustra de manera esquemática un primer producto 21 de programa informático, a modo de ejemplo, de conformidad con formas de realización de la presente invención.

El primer producto de programa informático 21 está adaptado para ejecutar un objeto de código informático intermedio, p. ej., puede ser un intérprete o código auxiliar en tiempo de ejecución para ejecutar el objeto de código informático intermedio.

El primer producto de programa informático comprende un componente de entrada 22 para obtener un objeto de código informático intermedio que comprende, al menos, un conjunto de instrucciones, p. ej., al menos una secuencia de instrucciones o al menos una construcción de fragmento de código que comprende instrucciones, correspondientes a una tarea a realizar. Este componente de entrada 22 puede, por ejemplo, recuperar el objeto de código informático intermedio de un archivo en el disco o de una zona de memoria precargada. El componente de entrada 22 puede estar adaptado para proporcionar una cola de espera de comandos de instrucciones codificadas en el objeto de código informático intermedio. En algunas formas de realización, el objeto de código informático intermedio puede incorporarse en el primer producto de programa informático, p. ej., se puede empaquetar en un único archivo ejecutable. En otras formas de realización, el objeto de código informático intermedio puede proporcionarse, por ejemplo, en un archivo separado que está siendo cargado por el primer producto de programa informático. El objeto de código informático intermedio comprende, además, para cada uno de los al menos un conjunto de instrucciones una pluralidad de descriptores de metadatos representativos de una pluralidad correspondiente de medidas de complejidad de la tarea a realizar. Por lo tanto, el objeto de código informático intermedio puede comprender un código de programa legible por ordenador de alto nivel que tiene bloques definidos en el mismo que están anotados por los correspondientes metadatos de complejidad. Conviene señalar que estos metadatos de complejidad pueden ser altamente independientes de la máquina, p. ej., puede caracterizar aspectos relacionados con la complejidad de una tarea a ejecutar sin tener en cuenta las características de rendimiento específicas de la máquina. Los metadatos de complejidad pueden ser similares a los descritos con anterioridad. En un ejemplo particular, los metadatos de complejidad pueden relacionarse, por ejemplo, con:

- el número de instrucciones requeridas para realizar la tarea,
- la presencia, el número y/o la profundidad de anidamiento de los bucles de ejecución requeridos,
- la presencia, el número y/o la profundidad de anidamiento de las bifurcaciones condicionales requeridas,
- la presencia, el número o la profundidad de las declaraciones recursivas,
- el uso de asignación de memoria dinámica,
- los requerimientos de espacio de almacenamiento de memoria estática y/o dinámica, y/o

- el uso de referencias de funciones indirectas, herencia operativa de objetos, envío dinámico de objetos, tipos de datos abstractos, encapsulado de objetos y/o recursión abierta.

El primer producto de programa informático también comprende un componente de tiempo de ejecución 23 para ejecutar el objeto de código informático intermedio en una plataforma informática que comprende, al menos, dos unidades de ejecución que tienen una memoria con una ubicación de memoria diferente. Este componente de tiempo de ejecución 23 comprende una unidad de selección 24 para seleccionar, para cada uno de los al menos un conjunto de instrucciones, una unidad de ejecución objetivo de entre la pluralidad de unidades de ejecución. Además, esta selección tiene en cuenta la pluralidad de descriptores de metadatos y una regla de decisión, en donde la regla de decisión relaciona la pluralidad de medidas de complejidad con una característica de rendimiento de la pluralidad de unidades de ejecución. Por lo tanto, el primer producto de programa informático puede adaptarse para una plataforma informática heterogénea específica o tipo de plataforma, p. ej., al proporcionar capacidades de código nativo de esta plataforma y al comprender una regla de decisión adaptada para esta plataforma o tipo de plataforma, al tiempo que puede ejecutar un objeto de código informático intermedio independiente de la plataforma de manera altamente optimizada.

El componente de tiempo de ejecución 23 también puede comprender una unidad de gestor de memoria 25. El objeto de código informático intermedio puede estructurarse de manera que permita una portabilidad adecuada de las estructuras de datos entre diferentes unidades de ejecución, p. ej., entre diferentes arquitecturas de CPU y/o GPU. En particular, el formato de objeto de código informático intermedio puede definir tipos de datos que pueden convertirse con facilidad a formatos nativos adecuados para una amplia gama de unidades de ejecución. Por ejemplo, la unidad de gestión de memoria puede proporcionar operaciones de contabilidad de memoria, tal como el seguimiento de un puntero de CPU y de un puntero de GPU a una estructura de datos y el seguimiento de bits no claros para indicar la versión modificada más recientemente de una estructura de datos. La unidad de gestión de memoria también puede proporcionar, de forma automatizada, funciones de copia, reproducción especular, desplazamiento y/o traducción de datos entre unidades de ejecución, p. ej., teniendo en cuenta las diferencias de bajo nivel en la arquitectura y puesta en práctica de dichas unidades de ejecución.

El componente de tiempo de ejecución 23 puede comprender, además, una unidad de planificación de tiempo de ejecución 26. Por lo tanto, el componente de tiempo de ejecución puede adaptarse para planificar la ejecución simultánea de tareas en la pluralidad de unidades de ejecución, teniendo en cuenta las dependencias de datos entre las tareas.

La Figura 3 ilustra, además, una forma de realización, a modo de ejemplo, de un primer producto de programa informático de conformidad con formas de realización de la presente invención, p. ej., un sistema de tiempo de ejecución. El producto de programa informático puede, cuando se ejecuta en un ordenador, ejecutarse en un hilo de ejecución de proceso host 31, p. ej., en una CPU de la plataforma informática. Por lo tanto, el primer producto de programa informático puede adaptarse para ejecutarse en un procesador host, p. ej., una CPU, de la plataforma informática que comprende una pluralidad de unidades de ejecución.

Un intérprete/motor de ejecución 32 puede proporcionar funciones para recuperar el objeto de código informático intermedio, p. ej., puede integrarse u operar conjuntamente con el componente de entrada 22. Una capa de abstracción de dispositivo de cálculo informático 33 forma una interfaz entre el código de hardware-agnóstico, p. ej., independiente de la plataforma, del objeto de código informático intermedio y de la plataforma informática. Un planificador de tiempo de ejecución 26 que se ejecuta en los hilos de ejecución de proceso host 31 puede recuperar fragmentos de código, p. ej., un conjunto de instrucciones que definen una tarea específica a realizar, desde el intérprete/motor de ejecución 32 a través de la capa de abstracción del dispositivo de cálculo 33.

Por ejemplo, el intérprete/motor de ejecución 32 puede procesar el objeto de código informático intermedio mediante la evaluación de expresiones almacenadas en el mismo, p. ej., codificadas en una notación polaca inversa. Por ejemplo, cuando se encuentra un código de operando, una referencia a un objeto correspondiente se puede insertar en una pila, p. ej., una pila gestionada por un gestor de objetos 27. Cuando se encuentra un operador al evaluar la expresión, esto se puede pasar a la cola de espera de comandos 35 de una unidad de ejecución a través del planificador de tiempo de ejecución 26.

El producto de programa informático, a modo de ejemplo, puede adaptarse, además, para interactuar con una pluralidad de dispositivos de cálculo informático, p. ej., una pluralidad de unidades de ejecución de la plataforma informática. Por ejemplo, el producto del programa informático puede adaptarse para ejecutarse en un dispositivo host, p. ej., una CPU de la plataforma informática y para interactuar con al menos un motor de cálculo específico del dispositivo, p. ej., una biblioteca vinculada dinámicamente adaptada para controlar al menos un dispositivo de cálculo. Por ejemplo, el producto de programa informático 21 puede estar vinculado de manera dinámica a un motor de cálculo para ejecutar un código en un dispositivo de CPU de uso general, p. ej., la CPU host y a un motor de cálculo para ejecutar un código en un dispositivo GPU. Evidentemente, un solo motor de cálculo también puede controlar una pluralidad de unidades de ejecución diferentes, o puede estar vinculado, de manera estática, al producto del programa, p. ej., puede integrarse en el producto del programa informático. Sin embargo, cuando el producto del programa informático está adaptado para conectarse dinámicamente a una pluralidad de motores de cálculo a través de una

interfaz estandarizada, el producto del programa informático puede adaptarse fácilmente para operar en dispositivos informáticos configurados de manera diferente.

El producto de programa informático puede configurarse para interactuar con, por ejemplo, un motor de cálculo de CPU, un motor de cálculo CUDA y un motor de cálculo openCL.

El planificador de tiempo de ejecución 26, además, puede seleccionar, para cada conjunto de instrucciones, p. ej., cada bloque de código forma una unidad coherente o fragmento de código, definiendo una tarea para realizar la unidad de ejecución objetivo más prometedora mediante la aplicación de una regla de decisión a los metadatos asociados con el conjunto de instrucciones. Por lo tanto, una cola de espera de comando 35 para cada una de entre la pluralidad de unidades de ejecución puede llenarse con tareas a ejecutar en cada unidad de ejecución objetivo, teniendo en cuenta las dependencias de datos entre los conjuntos de instrucciones del código intermedio. El planificador de tiempo de ejecución 26 puede, por ejemplo, poner en práctica un método similar a la ejecución fuera de orden en microprocesadores para el código de alto nivel correspondiente al formato de código intermedio.

El planificador de tiempo de ejecución 26 puede determinar la dimensionalidad de la operación en paralelo de un conjunto de instrucciones antes de la ejecución de este conjunto de instrucciones, p. ej., la función núcleo codificada en este conjunto de instrucciones. Por ejemplo, dependiendo de la dimensionalidad de los datos en donde operará el conjunto de instrucciones, puede ser necesario un número diferente de ejecuciones paralelas y/o un número diferente de hilos de ejecución. Esta determinación del tiempo de ejecución de los parámetros que influyen en la complejidad de la tarea complementa los metadatos con respecto a la complejidad inherente del bloque de código, que puede determinarse durante la compilación del objeto de código intermedio. Basado en el descriptor de metadatos multidimensional, p. ej., complementado por la dimensionalidad de la operación en paralelo determinada durante el tiempo de ejecución, el planificador de tiempo de ejecución determina la mejor unidad de ejecución para asignar para ejecutar el conjunto de instrucciones, p. ej., la mejor unidad teniendo en cuenta un modelo de pronóstico probabilístico representado por la regla de decisión. Los parámetros de tiempo de ejecución también pueden comprender información sobre, por ejemplo, la disponibilidad actual de una estructura de datos de entrada en una memoria accesible para una unidad de ejecución específica, p. ej., para tener en cuenta el coste de las operaciones de memoria al seleccionar esta unidad de ejecución.

La regla de decisión puede estar predeterminada para una configuración específica de la plataforma informática, y puede tener en cuenta, por ejemplo, el número máximo de hilos de ejecución paralelos que pueden generarse para cada unidad de ejecución. La regla de decisión también puede excluir o incluir una unidad de ejecución específica teniendo en cuenta la complejidad de la tarea a realizar, p. ej., un conjunto de instrucciones puede comprender demasiadas bifurcaciones o bucles anidados para ser ejecutables en una unidad de ejecución específica o el conjunto de instrucciones necesita funcionar en un bloque de datos que es demasiado grande para integrarse en una memoria accesible a una unidad de ejecución específica. La regla de decisión también puede tener en cuenta los parámetros globales de tiempo de ejecución, por ejemplo, una unidad de ejecución específica puede tener una cola de espera de comandos completa.

La cola de espera de comandos 35 para una CPU puede ponerse en práctica, por ejemplo, utilizando manipuladores de espera de eventos y sincronización, tal como se conoce en esta técnica. Para otras unidades de ejecución, p. ej., una GPU, la cola de espera de comandos 35 puede requerir una puesta en práctica más compleja. Por ejemplo, OpenCL puede utilizarse para interactuar con la GPU, puesto que ya admite la cola de espera de comandos, p. ej., una vez que se resuelven las dependencias de datos para un bloque de código, es suficiente pasar las dependencias al tiempo de ejecución de OpenCL y la solicitud del bloque de código, p. ej., se puede agregar a la cola de espera de comandos una función núcleo correspondiente a un conjunto de instrucciones en el objeto de código informático intermedio. Sin embargo, la cola de espera de comandos 35 para una GPU también se puede poner en práctica en la plataforma CUDA, p. ej., mediante la puesta en práctica de un mapeado de correspondencia adicional a los flujos de CUDA.

En las formas de realización de conformidad con la presente invención, las reglas de decisión pueden basarse en la evaluación de parámetros según se describió con anterioridad. A continuación, se describen dos ejemplos de reglas de decisión, que pueden utilizarse en formas de realización de la presente invención, no estando limitada la invención a las mismas.

En el primer ejemplo, en primer lugar, se verifica si el gestor de memoria de la GPU tiene suficiente espacio para transferir los argumentos de las funciones núcleo a la memoria de la GPU. Es posible que la función núcleo se refiera a varios bloques de memoria muy amplios que no pueden integrarse en la memoria de la GPU. En el ejemplo, para evitar una memoria de GPU insuficiente, la opción es ejecutar la función núcleo en la CPU. Sin embargo, otras opciones también son posibles. Una alternativa es realizar una compactación de la memoria de la GPU (debido a la compactación, podría liberarse algo de memoria adicional, lo que podría permitir que la función núcleo se ejecute en la GPU). Otra alternativa es utilizar una técnica de desalojo de memoria (con una política de desalojo, por ejemplo, la que se utilizó menos recientemente). Con esta técnica, los bloques de memoria que residen en la memoria de la GPU se vuelven a copiar en la memoria de la CPU, de modo que queda memoria adicional disponible para esta función núcleo. Otra alternativa es realizar un mapeado de correspondencia de memoria host. Es posible mapear la memoria

host de la CPU al espacio de direcciones de la GPU. Los tiempos de acceso a la memoria son bastante altos, sin embargo, esta técnica puede ser útil para las funciones núcleo que utilizan matrices de gran magnitud (por ejemplo, 1024 × 1024 × 1024 en formato de doble precisión).

Conviene señalar que todas estas alternativas tienen su propio coste asociado. Durante la evaluación de la regla de decisión, se pueden incluir estimaciones de costes de estas técnicas para guiar la decisión. Por simplicidad, en el ejemplo, simplemente ejecuta la función núcleo en la CPU en caso de que no haya suficiente memoria de GPU. A continuación, se compara el nivel de complejidad (parámetro 1) de una función núcleo con un primer umbral y también el producto de las dimensiones de la función núcleo (parámetro 4) con un segundo umbral. Cuando ambos parámetros son más pequeños que los umbrales correspondientes, se tendrá la seguridad de que se está gestionando con una función núcleo liviana con paralelismo limitado. En este caso, la opción preferida sería ejecutar la función núcleo en la CPU. Sin embargo, hay un coste de transferencia de memoria asociado a esta elección. Conviene señalar que algunos argumentos de entrada de la función núcleo pueden almacenarse en la memoria de la CPU, otros pueden almacenarse en la memoria de la GPU (o en ambas). Para calcular el coste de transferencia de memoria, se puede inspeccionar cada variable de manera individual (lo que requiere información tanto en tiempo de ejecución como en tiempo de compilación):

- información de tiempo de ejecución: por ejemplo, las dimensiones de la matriz, si está almacenada actualmente en la memoria de la CPU o en la memoria de la GPU (indicadores no claros, etc.).
- información en tiempo de compilación: el tipo de datos de la variable, pero también el modo de lectura/escritura de las zonas variables y/o las zonas actualizadas. Por ejemplo, en caso de que una variable que representa una matriz se utilice en modo de escritura y es seguro que todos los elementos de la matriz se sobrescriben (esto puede verificarse en tiempo de compilación), los datos de la matriz original pueden descartarse, eliminando la necesidad para transferencias de datos.

Algunas variables necesitan transferirse desde la GPU a la CPU, mientras que otras variables necesitan transferirse desde la CPU a la GPU. La función Mem_transfer_bytes calcula el número total de bytes que deben copiarse en una dirección, teniendo en cuenta los indicadores "no claros" de la variable (por ejemplo, cuando la variable se almacena en ambas memorias de la CPU y de la GPU y el indicador indica que la variable no tiene una etiqueta clara, por lo que no hay necesidad de transferir este bloque). Con esta información, se calcula la diferencia entre los tiempos de transferencia de memoria (parámetro 8) para copias de GPU a CPU y de CPU a GPU. Lo que antecede se realiza utilizando algunas constantes Avg_transfer_rate (GPU a CPU) y Avg_transfer_rate (CPU a GPU) que contienen (estimaciones para) los tiempos de transferencia de memoria (es decir, tiempos de lectura y escritura, respectivamente en la GPU). Estos parámetros se pueden obtener de antemano (por ejemplo, midiendo el tiempo que lleva copiar N bytes a/desde la GPU). En caso de que la diferencia Delta sea menor que un umbral dado T_Delta1, la decisión es ejecutar la función núcleo en la CPU. Conviene señalar que el umbral T_Delta1 puede ser mayor que 0 para admitir el hecho de que se sepa que se está gestionando con una función núcleo liviana, es decir, se sabe que la GPU no traerá muchos beneficios de rendimiento de todos modos. En el otro caso (las dimensiones de los datos o la complejidad del núcleo son suficientemente altas), la función núcleo es candidata para ejecutarse en la GPU. A continuación, se calcula la ocupación de la función núcleo. La ocupación indica cuántas deformaciones estarán activas en comparación con el número total de deformaciones admitidas en la GPU. La ocupación se calcula utilizando una serie de parámetros de GPU (tales como el número de registros utilizados por la función, la cantidad de memoria compartida, pero también las dimensiones de datos y las dimensiones de bloque). Todos estos parámetros están disponibles en tiempo de ejecución. En caso de que este número sea demasiado bajo (por ejemplo, debido a la gran cantidad de memoria compartida que se está utilizando), puede ser de utilidad ejecutar la función núcleo en la CPU de todos modos. En este caso, se verifica de nuevo la transferencia de memoria multiplicada por un umbral, T_Delta2. A modo de ilustración, el ejemplo de cómo se pone en práctica la regla de decisión se muestra en la Figura 5.

Como segundo ejemplo, se pueden tener en cuenta las magnitudes de las colas de espera de comandos de la CPU y de la GPU. En primer lugar, se compara el nivel de complejidad de nuevo con el umbral, así como el producto de las dimensiones de datos prod(dims) y la longitud del código. La verificación del tiempo de transferencia de memoria del primer ejemplo se omite en el presente ejemplo, pero también se puede incluir. Después del cálculo de la ocupación, y en el caso de que se trate de una función núcleo que tenga capacidad informática suficiente, se verifican las magnitudes de las colas de espera de comandos de la CPU y de la GPU. En este ejemplo, se toma el número total de funciones núcleo que están planificadas para las colas de espera de comandos de la CPU y de la GPU, sin embargo, esto también puede ser una estimación de la carga de las colas de espera de comandos de la CPU y de la GPU (por ejemplo, calculadas a través de las dimensiones de datos y niveles de complejidad de las funciones núcleo que ya están planificadas). Se compara las diferencias de las magnitudes de las colas de espera de comandos con un umbral T_size. Por último, se comprueban los tiempos de transferencia de memoria con el fin de tomar una decisión final. Una puesta en práctica de cómo se puede aplicar dicha regla de decisión se muestra en la Figura 6.

El planificador de tiempo de ejecución 26 puede, además, interactuar con un gestor de memoria 25 para la asignación automática de memoria con el fin de proporcionar datos para ser procesados por la ejecución de cada uno de dichos al menos un conjunto de instrucciones a la unidad de ejecución objetivo correspondiente. Por ejemplo, el producto de programa 31 puede configurarse para operar en una plataforma informática que consiste en un ordenador que

comprende dos unidades GPUs con diferente ubicación de memoria o una pluralidad de núcleos de CPU y una o más unidades GPUs. Esta configuración puede realizarse, por ejemplo, configurando el producto del programa para que se vincule con un motor de cálculo de CPU y un motor de cálculo CUDA. El gestor de memoria 25 puede proporcionar operaciones de asignación de memoria en la memoria RAM compartida accesible para la CPU y la memoria RAM gráfica accesible para la GPU. El gestor de memoria 25 puede transferir, además, estructuras de datos entre ambas memorias cuando el planificador de tiempo de ejecución detecta una dependencia de datos de un primer conjunto de instrucciones destinadas a la ejecución, por ejemplo, en la CPU, y un segundo conjunto de instrucciones destinadas a la ejecución, por ejemplo, la GPU. El gestor de memoria también puede adaptarse para realizar un seguimiento de varias copias, p. ej., en diferentes memorias y sincronizando copias a la versión actualizada más reciente cuando surja la necesidad. El gestor de memoria también puede adaptarse para la recogida de desechos y/o para liberar memoria para una unidad de ejecución específica desplazando los datos que actualmente no están en uso por la unidad de ejecución específica a una memoria asociada con una unidad de ejecución diferente.

El gestor de memoria puede proporcionar una asignación de memoria automatizada, de modo que, por ejemplo, las traducciones entre representaciones de datos específicas de hardware adecuadas para diferentes unidades de ejecución se realicen de manera automática, y sin requerir instrucciones específicas en el objeto de código informático intermedio para realizar dichas operaciones.

Las formas de realización de la presente invención tienen la ventaja de que el usuario, p. ej., el programador no tiene que copiar manualmente datos entre memorias asignadas a una primera unidad de ejecución, p. ej., la CPU y una segunda unidad de ejecución, tal como una GPU. Se conoce en esta técnica que programar explícitamente dichas operaciones puede ser tedioso, p. ej., para gráficos de objetos con punteros. Por ejemplo, sin dicha gestión de memoria automatizada, los objetos de datos, que pueden estar fragmentados en la memoria, podrían requerir varias copias de bloques de memoria que tengan un coste de tiempo asociado y/o las construcciones complejas de punteros de referencia deben atravesarse en la memoria de origen y reflejarse en la memoria de destino. Además, las restricciones de hardware o software de bajo nivel pueden requerir una organización de datos sustancialmente diferente para el dispositivo destino que la que estaba en uso para el dispositivo origen.

Las formas de realización de la presente invención también tienen la ventaja de que el usuario, p. ej., el programador, no necesita especificar qué bloques de código, p. ej., funciones núcleo, pueden ejecutarse en paralelo y cuáles necesitan ejecutarse en serie. Por lo tanto, se puede lograr un uso eficiente del tiempo durante la programación. Además, en el proceso de desarrollo, las dependencias de datos pueden cambiar, por lo que requeriría una reevaluación de las propiedades de concurrencia, y posiblemente podría conducir a errores de dependencia, en un sistema convencional. Por lo tanto, este inconveniente también se supera mediante formas de realización de la presente invención.

Puesto que las formas de realización de la presente invención pueden proporcionar gestión de memoria automática, se puede obtener un buen uso de los recursos de memoria. Por ejemplo, la gestión manual de la memoria puede ser difícil y podría conducir a prácticas de asignación previa también costosas, aunque simples. Además, la gestión manual de la memoria puede conducir fácilmente a pérdidas de memoria que son difíciles de detectar y corregir.

Cuando se inicia un conjunto de instrucciones, p. ej., una función núcleo, el planificador de tiempo de ejecución 26 puede crear un objeto representativo de un comando e inspeccionar las estructuras de datos asociadas con este comando, p. ej., operandos pasados a la función núcleo. Los objetos de datos correspondientes a estas estructuras de datos en una memoria accesible para la unidad de ejecución objetivo seleccionada, para la ejecución de este conjunto de instrucciones, pueden fijarse para evitar que este objeto de datos se desplace o se desasigne durante la ejecución. Esta fijación puede comprender agregar un indicador a un bloque de memoria en una estructura de memoria asignada para uso del gestor de memoria 25, indicando el indicador que el bloque de memoria referenciado no puede desplazarse ni desasignarse. Dichas disposiciones de fijación de memoria son bien conocidas en esta técnica, p. ej., utilizadas en muchos sistemas operativos para la gestión de memoria virtual. Por ejemplo, CUDA también admite una disposición similar en donde la memoria de la CPU está fijada para proporcionar acceso a la memoria host de la CPU a una GPU a través del acceso directo a la memoria (DMA).

Sin embargo, el planificador de tiempo de ejecución puede proporcionar fijación de nivel de objeto, en oposición al nivel de gestión de memoria, porque un único objeto puede tener múltiples copias en espacios de memoria separados, p. ej., correspondiente a diferentes unidades de ejecución. Por lo tanto, un objeto puede estar fijado o no fijado, p. ej., marcado o no marcado, para uno o más dispositivos. Por ejemplo, para una configuración que comprende una sola CPU que tiene una memoria de sistema dedicada y una sola GPU que tiene una memoria de dispositivo dedicada, se puede fijar un objeto: en la memoria del dispositivo cuando está en uso por la GPU, en la memoria del sistema cuando está en uso por la CPU, en ambas memorias cuando se utilizan en paralelo por la CPU y la GPU (por ejemplo, cuando solamente se realizan operaciones de lectura), o se pueden no fijar en ambas memorias cuando no están en uso. En el último caso, el gestor de memoria puede desplazar una copia del objeto dentro de una de las memorias, p. ej., para compactar bloques de memoria, o puede desplazar el objeto entre las memorias, p. ej., para que esté disponible para la GPU cuando la copia en la memoria de la CPU se haya actualizado más recientemente.

Se utiliza un indicador para la fijación que puede comprender también un indicador de lectura/escritura. Por ejemplo, cuando se establece un indicador de escritura para la memoria del dispositivo, se debe realizar una copia de la memoria en la memoria del sistema cuando la CPU necesita operar en el objeto de datos. En consecuencia, cuando se establece un inductor de solamente lectura, esta operación no es necesaria.

Además, el sistema de fijación puede ponerse en práctica de manera jerárquica, de modo que, para un gráfico de objeto, se evita la fijación de todos los nodos individuales del gráfico mediante el uso de una fijación de nodo principal.

Cuando se inicia un conjunto de instrucciones, el planificador de tiempo de ejecución 26 puede crear un objeto representativo de un comando, y las fijaciones de objeto de los objetos de datos asociados al conjunto de instrucciones, p. ej., los operandos pasados a la llamada de función pueden estar vinculados al objeto de comando de modo que el planificador pueda efectuar un seguimiento de las dependencias de datos. Además, cada objeto puede tener una lista asociada de comandos en donde participan los datos. Por lo tanto, cuando se inicia un conjunto de instrucciones, p. ej., una función núcleo, las dependencias entre los comandos en la cola de espera de comandos 35 se pueden determinar con facilidad, inspeccionando la lista de comandos asociados con las fijaciones del objeto. Además, al inspeccionar los accesos de lectura/escritura de los objetos, el planificador puede determinar si una ejecución simultánea o en serie del comando es apropiada. Por ejemplo, los accesos de lectura posteriores a un objeto se pueden agregar, ya que no introducen una nueva dependencia de datos. Además, las operaciones de lectura después de escritura (RaW), escritura después de lectura (WaR) y escritura después de escritura (WaW) en un objeto deben ser serializadas, de modo que se introduzca una nueva dependencia de datos.

En un tercer aspecto, la presente invención también se refiere a un segundo producto de programa informático, p. ej., un producto funcionalmente interrelacionado con el primer programa informático. Este segundo producto de programa informático puede ser, por ejemplo, un compilador de software. El segundo producto de programa informático, p. ej., el compilador y el primer producto de programa informático, p. ej., un intérprete de ejecución en tiempo de ejecución, según las formas de realización, forman un par de productos interrelacionados, p. ej., se complementan y trabajan de manera conjunta. En particular, el segundo producto de programa informático de conformidad con formas de realización puede utilizarse para, p. ej., adaptarse para generar un objeto de código informático intermedio que comprende, al menos, un conjunto de instrucciones, p. ej., al menos una secuencia de instrucciones, cada una de las cuales forma un bloque de código coherente, tal como una función o rutina, y los metadatos que lo acompañan correspondientes a este al menos un conjunto de instrucciones, mientras que el primer producto de programa informático, según las formas de realización, se puede utilizar para, p. ej., adaptarse para ejecutar este objeto de código informático intermedio en una plataforma informática. Por lo tanto, cada producto de programa informático está vinculado a través del objeto de código informático intermedio que permite una compilación eficiente independiente de la plataforma mientras disfruta de una ejecución optimizada dependiente de la plataforma en una plataforma informática que comprende una pluralidad de unidades de ejecución que pueden tener arquitecturas mutuamente casi distintas y, por lo tanto, capacidades mutuamente distintas y los puntos fuertes y débiles del rendimiento relativo entre sí.

La Figura 4 ilustra un segundo producto de programa informático 41, a modo de ejemplo, de conformidad con formas de realización de la presente invención. El segundo producto de programa informático 41 para generar un objeto de código informático intermedio comprende un componente de entrada 42 para obtener un código de programa informático especificado de conformidad con una especificación de lenguaje de programación de alto nivel. Este lenguaje de programación de alto nivel puede incluir, por ejemplo, C++, Pascal, BASIC, Smalltalk o incluso un lenguaje de programación de muy alto nivel (VHLL). Dicho lenguaje de programación de alto nivel puede ampliarse para aprovechar todas las características del segundo producto de programa informático, p. ej., por directivas #pragma o palabras clave adicionales no definidas en la norma de idioma. Sin embargo, el lenguaje de programación de alto nivel también puede diseñarse específicamente para aprovechar al máximo las características de la presente invención, p. ej., para proporcionar clases de variables nativas que proporcionan una alta portabilidad y/o estructuras de control específicas para sistemas paralelos y/o distribuidos, tales como los bucles paralelos.

El segundo producto de programa informático 41 comprende, además, un componente de compilación 43 para compilar el código del programa informático en un objeto de código informático intermedio que comprende, al menos, un conjunto de instrucciones correspondientes a una tarea a realizar. Este objeto de código informático intermedio puede ser, por ejemplo, un código de nivel de byte para una máquina abstracta. Por ejemplo, el código fuente de un programa se traduce a una forma más adecuada para las transformaciones que mejoran el código antes de utilizarse, p. ej., mediante un primer producto de programa informático según las formas de realización de la presente invención, para generar código máquina para un procesador objetivo durante el tiempo de ejecución. El lenguaje intermedio al que confirma el objeto de código informático intermedio puede tener, por ejemplo, una operación fundamental correspondiente a cada instrucción, donde una operación fundamental puede relacionarse con una operación, posiblemente implicada de forma informática, tal como una multiplicación de matriz. Dicho código informático intermedio puede definirse, además, sin especificar directamente las operaciones típicas de bajo nivel tales como una manipulación de registros o una manipulación de punteros de instrucción.

El segundo producto de programa informático 41 comprende, además, un componente de análisis 44 para anotar cada uno de los al menos un conjunto de instrucciones con una pluralidad de descriptores de metadatos representativos de

una pluralidad correspondiente de medidas de complejidad que caracterizan el menos un conjunto de instrucciones, p. ej., representativo de la complejidad de la tarea a realizar. Por ejemplo, el componente de análisis 44 puede adaptarse para determinar la pluralidad de descriptores de metadatos representativos de al menos una medida relacionada con bifurcaciones condicionales, saltos hacia atrás, asignación de memoria dinámica, llamadas indirectas a funciones y/o sincronización de hilos de ejecución. Conviene señalar que estos metadatos de complejidad pueden ser altamente independientes de la máquina, p. ej., pueden caracterizar aspectos relacionados con la complejidad de una tarea a ejecutar sin tener en cuenta las características de rendimiento específicas de la máquina. Dichos metadatos de complejidad pueden relacionarse, por ejemplo, con:

- el número de instrucciones requeridas para realizar la tarea,
- la presencia, el número y/o la profundidad de anidamiento de los bucles de ejecución requeridos,
- la presencia, el número y/o la profundidad de anidamiento de las bifurcaciones condicionales requeridas,
- la presencia, número o profundidad de las declaraciones recursivas,
- el uso de asignación de memoria dinámica,
- los requerimientos de espacio de almacenamiento de memoria estática y/o dinámica, y/o
- el uso de referencias de funciones indirectas, herencia operativa de objetos, envío dinámico de objetos, tipos de datos abstractos, encapsulado de objetos y/o recursión abierta.

Las formas de realización de métodos descritas con anterioridad para ejecutar un objeto de código informático intermedio pueden ponerse en práctica como software en un procesador. Una configuración de dicho procesador puede incluir, por ejemplo, al menos un componente informático programable acoplado a un subsistema de memoria que incluye al menos una forma de memoria, por ejemplo, memoria RAM, memoria ROM, etc. Conviene señalar que el componente informático o los componentes informáticos pueden ser de uso general, o de uso especial, y pueden incluirse en un dispositivo, por ejemplo, un circuito integrado que tenga otros componentes que realizan otras funciones. Por lo tanto, uno o más aspectos de la presente invención pueden ponerse en práctica en circuitos electrónicos digitales, o en hardware, firmware, software informático, o sus combinaciones. Por ejemplo, cada una de las etapas del método para ejecutar un objeto de código informático intermedio puede ser una etapa puesta en práctica por ordenador tal como una o un conjunto de instrucciones. Por lo tanto, mientras que un procesador tal como es en la técnica anterior, un sistema que incluye las instrucciones para poner en práctica aspectos de los métodos para ejecutar el objeto de código informático intermedio no técnica anterior.

Por lo tanto, la presente invención también incluye un producto de programa informático que proporciona la funcionalidad de cualquiera de los métodos de conformidad con la presente invención cuando se ejecuta en un dispositivo informático.

A la inversa, los productos de programas informáticos descritos con anterioridad pueden ponerse en práctica como hardware en dispositivos informáticos. De manera alternativa, los productos de programas informáticos pueden ponerse en práctica como métodos puestos en práctica por ordenador y, por lo tanto, la presente invención también se refiere a los métodos puestos en práctica por ordenador correspondientes.

En otro aspecto, la presente invención se refiere a un soporte de datos para transportar un producto de programa informático tal como se describió con anterioridad. Dicho soporte de datos puede comprender un producto de programa informático incorporado de manera tangible en el mismo y puede incluir código legible por máquina para su ejecución por un procesador programable. Por lo tanto, la presente invención se refiere a un medio portador que incluye un producto de programa informático que, cuando se ejecuta en medios informáticos, proporciona instrucciones para ejecutar cualquiera de los métodos descritos con anterioridad. El término "medio portador" se refiere a cualquier medio que participa en el suministro de instrucciones a un procesador para su ejecución. Dicho medio puede adoptar muchas formas, incluyendo, sin limitación, medios no volátiles y medios de transmisión. Los medios no volátiles incluyen, por ejemplo, discos ópticos o magnéticos, tal como un dispositivo de almacenamiento que forma parte del almacenamiento masivo. Las formas comunes de medios legibles por ordenador incluyen, un CD-ROM, un DVD, un disco flexible o disquete, una cinta, un circuito integrado de memoria o cartucho de memoria o cualquier otro medio desde el cual un ordenador pueda efectúa su lectura. Diversas formas de medios legibles por ordenador pueden estar implicadas en transmitir una o más secuencias de una o más instrucciones a un procesador para su ejecución. El producto del programa informático también se puede transmitir a través de una onda portadora en una red, tal como una red LAN, una red WAN o Internet. Los medios de transmisión pueden adoptar la forma de ondas acústicas o de luz, tal como las generadas durante las comunicaciones de datos de ondas de radio e infrarrojos. Los medios de transmisión incluyen cables coaxiales, cable de cobre y fibra óptica, incluyendo los cables que comprenden un bus dentro de un ordenador.

REIVINDICACIONES

1. Un método puesto en práctica por ordenador (1) que comprende:

- obtener (2) un objeto de código informático intermedio que comprende, al menos, un conjunto de instrucciones correspondiente a una tarea a realizar, comprendiendo el objeto de código informático intermedio, además, para cada uno de dichos al menos un conjunto de instrucciones, uno o más descriptores de metadatos representativos de al menos una medida de complejidad de dicha tarea a realizar, siendo dicho objeto de código informático intermedio independiente de la máquina y estando dicha medida de complejidad basada al menos en un análisis del código informático intermedio; y

- ejecutar (4) en tiempo de ejecución dicho objeto de código informático intermedio en una plataforma informática que comprende, al menos, dos unidades de ejecución diferentes que tienen una memoria con una ubicación de memoria diferente, comprendiendo dicha ejecución en tiempo de ejecución seleccionar (6), para cada uno de dichos al menos un conjunto de instrucciones, una unidad de ejecución objetivo de entre dicha pluralidad de unidades de ejecución, teniendo en cuenta dicha selección los uno o más de los descriptores de metadatos y una regla de decisión que relacione dicha al menos una medida de complejidad con una característica de rendimiento de la pluralidad de unidades de ejecución, comprendiendo dicha ejecución (4) del objeto de código informático intermedio traducir (8) cada uno de al menos un conjunto de instrucciones a un formato de nivel máquina ejecutable por la unidad de ejecución objetivo correspondiente.

2. El método puesto en práctica por ordenador según la reivindicación 1, en donde las al menos dos unidades de ejecución diferentes son una unidad central de procesamiento CPU y una unidad de procesamiento gráfico GPU o en donde las al menos dos unidades de ejecución diferentes son unidades de procesamiento gráfico GPU que tienen una memoria con una ubicación de memoria diferente.

3. El método puesto en práctica por ordenador según cualquiera de las reivindicaciones anteriores, en donde dichos uno o más descriptores de metadatos son representativos de una medida correspondiente de al menos una complejidad de dicha tarea a realizar.

4. El método puesto en práctica por ordenador según cualquiera de las reivindicaciones anteriores, en donde dicho objeto de código informático intermedio se obtiene (2) en un formato intermedio independiente de unidad de ejecución.

5. El método puesto en práctica por ordenador según cualquiera de las reivindicaciones anteriores, en donde ejecutar (4) el objeto de código informático intermedio comprende determinar (5) si un primer conjunto de dicho al menos un conjunto de instrucciones y un segundo conjunto de dicho en al menos un conjunto de instrucciones puede ejecutarse de manera simultánea.

6. El método puesto en práctica por ordenador según cualquiera de las reivindicaciones anteriores, en donde ejecutar (4) el objeto de código informático intermedio comprende proporcionar (7) asignación de memoria automatizada para proporcionar datos para ser procesados por la ejecución de cada uno de dichos al menos un conjunto de instrucciones para la unidad de ejecución objetivo correspondiente.

7. El método puesto en práctica por ordenador según cualquiera de las reivindicaciones anteriores, en donde obtener (2) el objeto de código informático intermedio comprende compilar (11) el objeto de código informático intermedio a partir de un código de programa informático especificado de conformidad con una especificación de lenguaje de programación de alto nivel y/o en donde obtener (2) el objeto de código informático intermedio comprende, además, para cada uno de dichos al menos un conjunto de instrucciones que determinan (12) los uno o más de los descriptores de metadatos representativos de la medida de complejidad correspondiente.

8. El método puesto en práctica por ordenador según cualquiera de las reivindicaciones anteriores, en donde los uno o más descriptores de metadatos son uno o más parámetros determinables a nivel de compilador y que expresan una complejidad de una función núcleo del objeto de código informático intermedio.

9. El método puesto en práctica por ordenador según cualquiera de las reivindicaciones anteriores, en donde dicha selección también tiene en cuenta una o más de entre una longitud de código, un producto de dimensiones de datos, un producto de dimensiones de bloques de GPU, un número total de bloques de GPU, un número de hilos de ejecución de CPU asignados, un tiempo de transferencia de memoria, una ocupación de GPU, un tamaño o carga de colas de espera de comando de CPU y GPU o una carga global de colas de espera de CPU y GPU.

10. Un primer producto de programa informático (21) para ejecutar un objeto de código informático intermedio, comprendiendo el primer producto de programa informático:

- un componente de entrada (22) para obtener un objeto de código informático intermedio que comprende, al menos, un conjunto de instrucciones correspondientes a una tarea a realizar, comprendiendo, además, el objeto de código informático intermedio, para cada uno de dichos al menos un conjunto de instrucciones, uno o más de

descriptores de metadatos representativos de una medida de complejidad correspondiente de dicha tarea a realizarse, estando basada al menos una medida de complejidad en un análisis del código informático intermedio y siendo dicho objeto de código informático intermedio independiente de la máquina, y

- un componente de tiempo de ejecución (23) para ejecutar dicho objeto de código informático intermedio en una plataforma informática que comprende, al menos, dos unidades de ejecución diferentes que tienen una memoria con una ubicación de memoria diferente,

en donde el componente de tiempo de ejecución (23) comprende una unidad de selección (24) para seleccionar, para cada una de dichos al menos un conjunto de instrucciones, una unidad de ejecución objetivo de dichas al menos dos unidades de ejecución, teniendo en cuenta dicha selección los uno o más descriptores de metadatos y una regla de decisión que relacione dicha al menos una medida de complejidad con una característica de rendimiento de las al menos dos unidades de ejecución diferentes, y estando dicho componente de tiempo de ejecución (23) programado para traducir (8) cada uno de los al menos un conjunto de instrucciones a un formato de nivel máquina ejecutable por la unidad de ejecución objetivo correspondiente.

11. El primer producto de programa informático según la reivindicación 10, en donde el componente de tiempo de ejecución (23) comprende, además, una unidad de gestión de memoria (25) para asignación de memoria automatizada y/o una unidad de planificación de tiempo de ejecución (26) adaptada para determinar (5) si un primer conjunto de dicho al menos un conjunto de instrucciones y un segundo conjunto de dicho al menos un conjunto de instrucciones pueden ejecutarse de manera simultánea.

12. Un segundo producto de programa informático (41) para generar un objeto de código informático intermedio, comprendiendo el segundo producto de programa informático:

- un componente de entrada (42) para obtener un código de programa informático especificado de conformidad con una especificación de lenguaje de programación de alto nivel,

- un componente de compilación (43) para compilar el código del programa informático en un objeto de código informático intermedio que comprende, al menos, un conjunto de instrucciones correspondientes a una tarea a realizar, y

- un componente de análisis (44) para anotar cada uno de dichos al menos un conjunto de instrucciones con uno o más descriptores de metadatos representativos de al menos una medida de complejidad correspondiente de dicha tarea a realizar, estando basada al menos una medida de complejidad en un análisis del código informático intermedio y siendo dicho objeto de código informático intermedio independiente de la máquina,

en donde cuando el objeto de código informático intermedio generado es ejecutado por un producto de programa informático, según la reivindicación 10, en una plataforma informática que comprende, al menos, dos unidades de ejecución diferentes, se selecciona una unidad de ejecución objetivo de dichas al menos dos unidades de ejecución teniendo en cuenta dichos uno o más descriptores de metadatos.

13. El segundo producto de programa informático (41) según la reivindicación 12, en donde el componente de análisis (44) está adaptado para determinar los uno o más descriptores de metadatos representativos de al menos una medida relacionada con bifurcaciones condicionales, saltos hacia atrás, asignación de memoria dinámica, llamadas de función indirectas y/o sincronización de hilos de ejecución.

14. Un soporte de datos que comprende un conjunto de instrucciones para, cuando se ejecuta en un ordenador, ejecutar un objeto de código informático intermedio en una plataforma informática que comprende, al menos, dos unidades de ejecución diferentes que tienen una memoria con una ubicación de memoria diferente de conformidad con un método según cualquiera de las reivindicaciones 1 a 9.

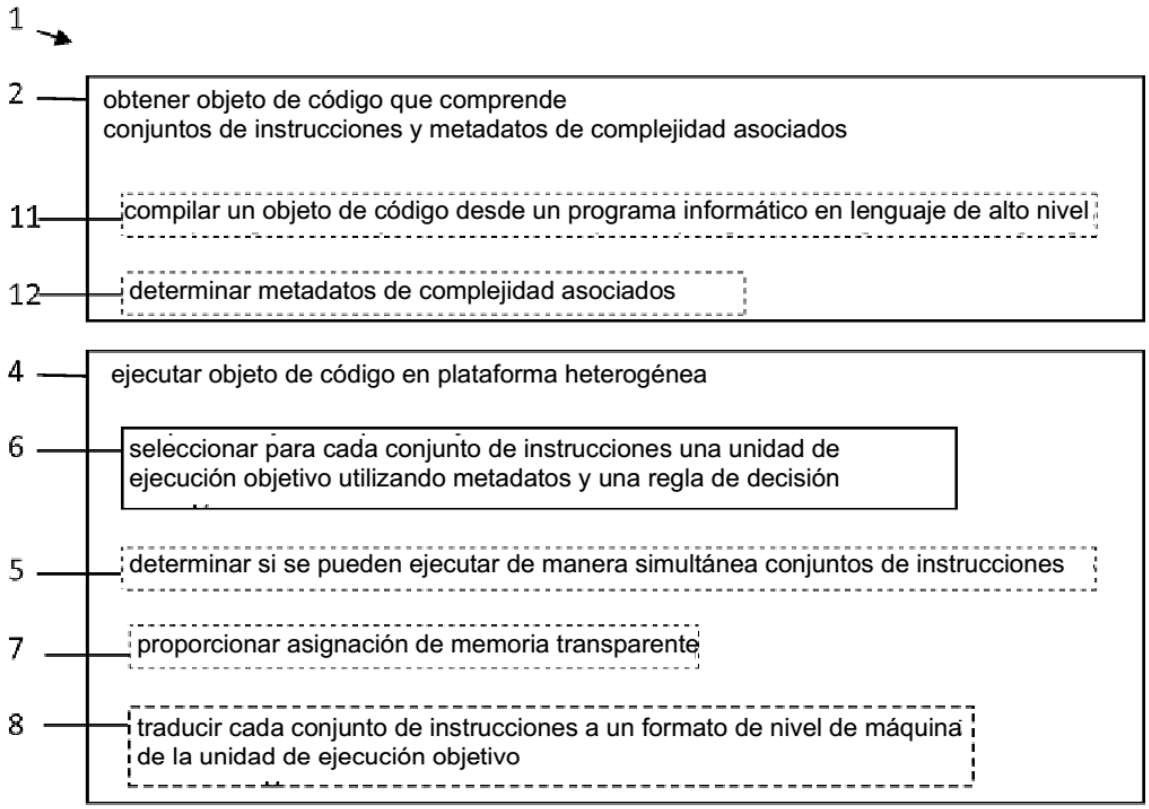
1
2
obtener objeto de código que comprende
conjuntos de instrucciones y metadatos de complejidad asociados
11
compilar un objeto de código desde un programa informático en lenguaje de alto nivel
12
determinar metadatos de complejidad asociados
4
ejecutar objeto de código en plataforma heterogénea
6
seleccionar para cada conjunto de instrucciones una unidad de ejecución objetivo utilizando metadatos y una regla de decisión
5
determinar si se pueden ejecutar de manera simultánea conjuntos de instrucciones
7
proporcionar asignación de memoria transparente
8
traducir cada conjunto de instrucciones a un formato de nivel de máquina de la unidad de ejecución objetivo

FIG. 1

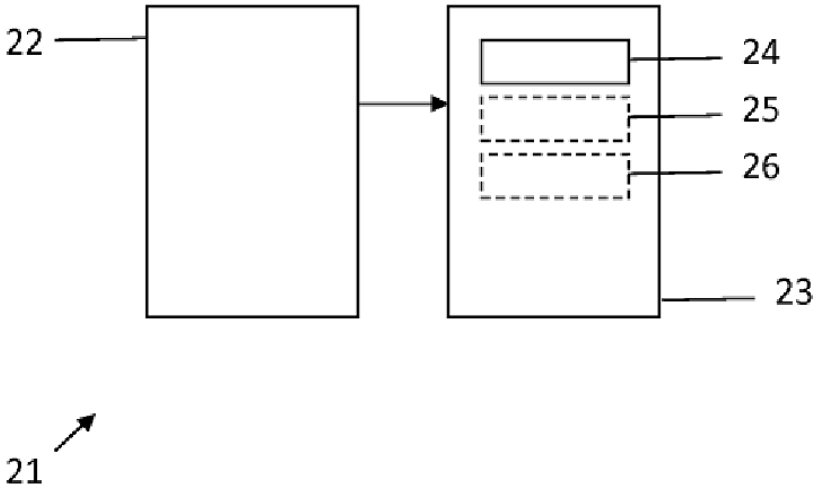
22
24
25
26
23
21

FIG. 2

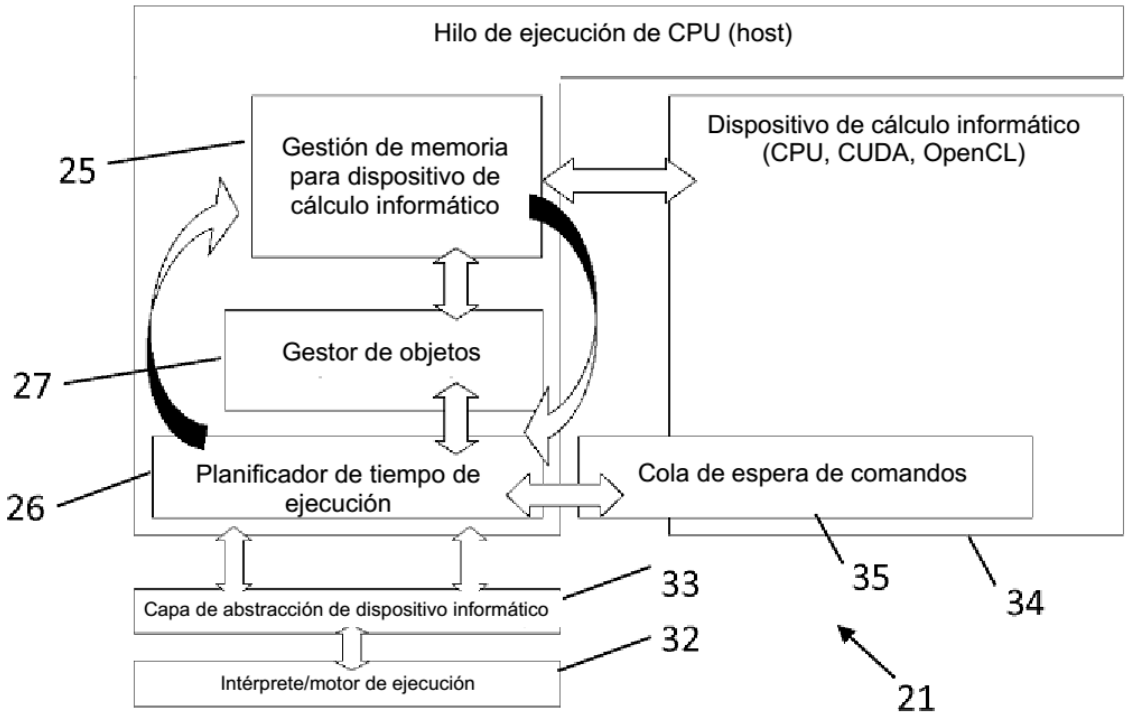
Hilo de ejecución de CPU (host)
Gestión de memoria para dispositivo de cálculo informático
Dispositivo de cálculo informático (CPU, CUDA, OpenCL)
25
Gestor de objetos
27
Planificador de tiempo de ejecución
26
Cola de espera de comandos
35
34
Capa de abstracción de dispositivo informático
33
Intérprete/motor de ejecución
32
21

FIG. 3

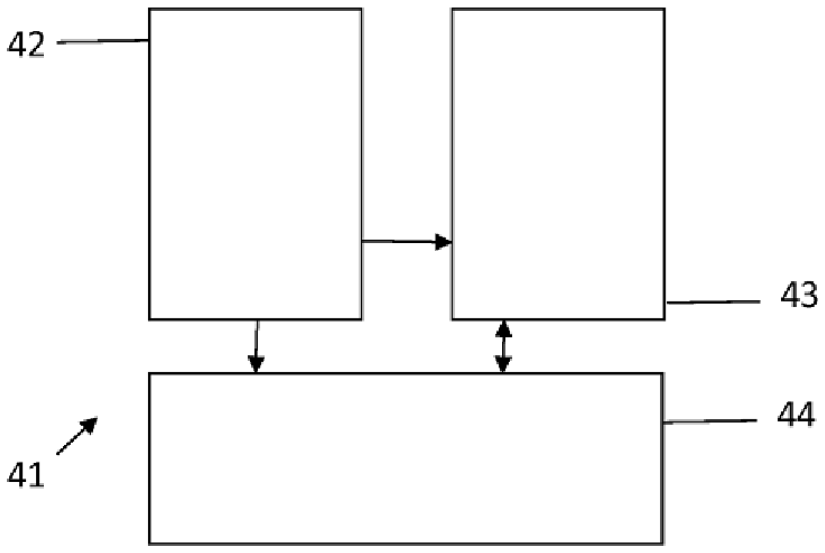
42
43
44
41

FIG. 4

```
% Entradas: núcleo, argumentos, dimensiones, dimensiones bloques
% Salidas: objetivo
% Umbrales: T_C, T_dims, T_Delta1, T_Delta2, T_Occupancy
% Parámetros utilizados: 1, 3, 4, 5, 8, 9

% Hemos de ejecutarlos probablemente en la GPU
% Comprobar si se dispone de memoria suficiente para copiar todos los argumentos
% para la GPU

if !gpu_memory_manager.sufficient_space(args)
    target = TARGET_CPU
elseif Complexity_level < T_C && prod(dims) < T_dims
    % Función núcleo de peso liviano; es probable que sea más eficiente su ejecución en la CPU
    % => Calcular el tiempo de transferencia de memoria desde GPU -> CPU
    Delta = Mem_transfer_bytes(args, GPU->CPU) * Avg_transfer_rate(GPU->CPU) -
            Mem_transfer_bytes(args, CPU->GPU) * Avg_transfer_rate(CPU->GPU)

    if Delta < T_Delta1
        target = TARGET_CPU   % Ejecutar en la CPU
    endif
else
    % Calcular la ocupación; si la ocupación es demasiado baja, podemos considerar
    % la ejecución en la GPU

    if occupancy < T_occupancy
        %-> Calcular el tiempo de transferencia de memoria desde GPU -> CPU
        Delta = Mem_transfer_bytes(args, GPU->CPU) * Avg_transfer_rate(GPU->CPU) -
                Mem_transfer_bytes(args, CPU->GPU) * Avg_transfer_rate(CPU->GPU)

        if Delta < T_Delta2
            target = TARGET_CPU   % Ejecutar en la CPU
        else
            target = TARGET_GPU
        endif
    endif
endif
```

FIG. 5

```
% Entradas: núcleo, argumentos, dimensiones, dimensiones de bloques, cola de espera (CPU), cola de espera (GPU)
% Salidas: objetivo
% Umbrales: T_C, T_dims, T_Occupancy, T_size, T_Delta, T_code
% Parámetros utilizados: 1, 2, 3, 4, 5, 8, 9, 10

if Complexity_level < T_C && code_len < T_code && prod(dims) < T_dims
    % Función núcleo de peso liviano; es probable que sea más eficiente su ejecución en la CPU
    objetivo = TARGET_CPU
else
    % Calcular la ocupación; si la ocupación es demasiado baja, podemos considerar
    % la ejecución en la GPU
    occupancy = calcOccupancy(kernel, dims, blkdims)
    if occupancy < T_occupancy
        target = TARGET_CPU
    endif

    if size(cmdqueue(GPU)) - size(cmdqueue(CPU)) < T_size
        target = TARGET_GPU
    else
        % Calcular el tiempo de transferencia de memoria desde GPU -> CPU
        Delta = Mem_transfer_bytes(args, GPU->CPU) * Avg_transfer_rate(GPU->CPU) -
                Mem_transfer_bytes(args, CPU->GPU) * Avg_transfer_rate(CPU->GPU)
        if Delta > T_Delta
            target = TARGET_GPU
        else
            target = TARGET_CPU
        endif
    endif
endif
```

FIG. 6