(19) Deutsches Patent- und Markenamt

(10) **DE 10 2013 018 380 A1** 2014.05.08

(12) # Offenlegungsschrift

(21) Aktenzeichen: **10 2013 018 380.4**
(22) Anmeldetag: **04.11.2013**
(43) Offenlegungstag: **08.05.2014**

(51) Int Cl.: ***G06F 9/46*** (2006.01)

(30) Unionspriorität:

61/722,661	**05.11.2012**	**US**
13/724,359	**21.12.2012**	**US**

(71) Anmelder:
NVIDIA Corporation, Santa Clara, Calif., US

(74) Vertreter:
Verscht, Thomas K., Dipl.-Phys.(Univ.), 81673, München, DE

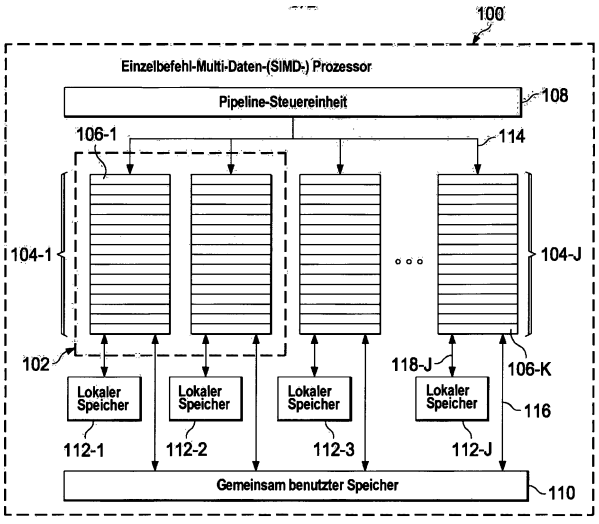
(72) Erfinder:
Lin, Yuan, Santa Clara, Calif., US; Chakrabarti, Gautam, Santa Clara, Calif., US; Marathe, Jaydeep, Santa Clara, Calif., US; Kwon, Okwan, West Lafayette, Ind., US; Sabne, Amit, West Lafayette, Ind., US

Prüfungsantrag gemäß § 44 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Bezeichnung: **System und Verfahren zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms mit Aufteilung-Vereinigung mit Funktionsaufrufen in einem Einzelbefehl-Multi-Strang-Prozessor**

(57) Zusammenfassung: Ein System und ein Verfahren zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms mit Aufteilung-Vereinigung mit Funktionsaufrufen. In einer Ausführungsform umfasst das System: (1) eine Partitionseinheit, die ausgebildet ist, Gruppen in eine übergeordnete Gruppe und mindestens eine Arbeiter-Gruppe zu unterteilen, und (2) eine Strang-Zuordnungseinheit, die mit der Partitionseinheit verbunden und ausgebildet ist, nur einen einzigen Strang aus der übergeordneten Gruppe zur Ausführung und alle Stränge in der mindestens einen Arbeiter-Gruppe für die Ausführung auszuweisen.



Beschreibung

QUERVERWEIS AUF VERWANDTE ANMELDUNG

[0001] Diese Anmeldung beansprucht die Priorität der vorläufigen US-Anmeldung mit der Seriennummer 61/722661, die von Lin et al. am 5. November 2012 eingereicht wurde mit dem Titel „AUSFÜHRUNG EINES SEQUENZIELLEN CODES UNTER ANWENDUNG EINER GRUPPE AUS STRÄNGEN" und der US-Anmeldung mit der Seriennummer 13/724359, die von Lin, et al. am 21. Dezember 2012 eingereicht wurde mit dem Titel „SYSTEM UND VERFAHREN ZUR KOMPILIERUNG ODER LAUFZEITAUSFÜHRUNG EINES DATENPARALLELEN PROGRAMMS MIT AUFTEILUNG-VEREINIGUNG MIT FUNKTIONSAUFRUFEN IN EINEM EINZELBEFEHL-MULTI-STRANG-PROZESSOR", die beide die gleiche Anmelderin wie diese Anmeldung haben und hierin durch Bezugnahme mit eingeschlossen sind.

TECHNISCHES GEBIET

[0002] Diese Anmeldung betrifft generell parallele Prozessoren und insbesondere ein System und ein Verfahren zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms mit Aufteilung-Vereinigung mit Funktionsaufrufen in einem Einzelbefehl-Multi-Strang-(SIMT-)Prozessor.

HINTERGRUND

[0003] Wie der Fachmann auf diesem Gebiet weiß, können Anwendungen oder Programme parallel ausgeführt werden, um ihr Leistungsverhalten zu verbessern. Datenparallele Programme führen den gleichen Prozess gleichzeitig an unterschiedlichen Daten aus. Aufgabenparallele Programme führen unterschiedliche Prozesse gleichzeitig an den gleichen Daten aus. Statische parallele Programme sind Programme mit einem Grad an Parallelität, der vor der Ausführung bestimmt werden kann. Im Gegensatz dazu kann die Parallelität, die von dynamischen parallelen Programmen erreichbar ist, nur ermittelt werden, wenn sie ausgeführt werden. Unabhängig davon, ob das Programm datenparallel oder aufgabenparallel oder statisch oder dynamisch parallel ist, kann es in einer Pipeline bzw. Parallelverarbeitungslinie ausgeführt werden, was häufig der Fall ist für graphische Programme.

[0004] Ein SIMT-Prozessor ist besonders geschickt bei der Ausführung datenparalleler Programme. Eine Steuereinheit in dem SIMT-Prozessor erzeugt Gruppen aus Strängen zur Ausführung und disponiert diese für die Ausführung, während welcher alle Stränge in der Gruppe den gleichen Befehl gleichzeitig ausführen. In einem speziellen Prozessor hat jede Gruppe oder „Wölbung bzw. Kette" 32 Stränge, die 32 Ausführungs-Pipelines oder Bahnen in dem SIMT-Prozessor entsprechen.

[0005] Ein datenparalleles Programm mit Aufteilung-Vereinigung beginnt mit einem Hauptprogramm, das nur einen Strang aufweist. Das Programm ist in dieser Phase in einer sequenziellen Phase oder einem sequenziellen Bereich. Bei einem gewissen Punkt während der Ausführung des Hauptprogramms trifft der Haupt- oder „Master-"Strang auf eine Sequenz aus parallelen Phasen oder Bereichen. Jeder parallele Bereich hat einen unabhängigen Datensatz und kann von mehreren Strängen gleichzeitig ausgeführt werden. Die Anzahl an gleichzeitigen Aufgaben in jedem parallelen Bereich wird bestimmt, wenn der parallele Bereich beginnt, und sich während des parallelen Bereichs nicht ändert. Wenn ein paralleler Bereich angetroffen wird, führt der Haupt-Strang eine Aufteilung in eine Gruppe aus Strängen (die als Arbeiter-Stränge bezeichnet werden) durch, um die parallelen Bereiche parallel abzuarbeiten. Das Programm tritt dann in den parallelen Bereich ein. Wenn ein Arbeiter-Strang einen neuen parallelen Bereich antrifft, wird der neue parallele Bereich serialisiert, d. h. der parallele Bereich wird von dem eintreffenden Arbeiter-Strang selbst ausgeführt. Der Haupt-Strang wartet, bis der parallele Bereich beendet ist. Beim Austritt aus dem parallelen Bereich vereinigen sich die Arbeiter-Stränge mit dem Haupt-Strang, der dann die Ausführung des Hauptprogramms wieder fortgesetzt, wobei dann das Programm einen sequenziellen Bereich betritt.

[0006] Die nachfolgende Tabelle 1 gibt ein Beispiel für ein datenparalleles Programm mit Aufteilung-Vereinigung an.

```
extern ext();

thread_main()
{
   foo();
   ext();
   #pragma parallel loop
   for (…) {
      foo();
      bar();
   }
   #pragma parallel loop
   for (…) {
      ext();
}
   bar();
}

bar()
{
   #pragma parallel loop
   for (…) {
      …
   }
}

foo()
{
   …
}
```

Tabelle 1 – ein Beispiel eines datenparallelen Programms mit Aufteilung-Vereinigung

[0007] Zum Zwecke des Verständnisses der Tabelle 1 und des Restes dieser Offenbarung sind die Begriffe „foo" und „bar" willkürliche Namen von Funktionen. Es kann daher eine beliebige Funktion anstelle von „foo" oder „bar" verwendet werden.

[0008] Das Datenparallele Modell mit Aufteilung-Verzweigung wird häufig in der parallelen Programmierung eingesetzt. Beispielsweise verwendet der OpenMP-Standard dieses Modell als ein grundlegendes Strang-Ausführungsmodell. Der OpenACC-Standard verwendet dieses Modell für die Arbeiter-Stränge in einer Gruppe, die als eine „Arbeitsgruppe" bezeichnet wird.

ÜBERBLICK

[0009] Ein Aspekt stellt ein System zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms mit Aufteilung-Vereinigung mit Funktionsaufrufen bereit. In einer Ausführungsform umfasst das System: (1) eine Partitionseinheit, die ausgebildet ist, Ketten in eine übergeordnete Kette und mindestens eine Arbeiter-Kette zu unterteilen, und (2) eine Strang-Zuweisungseinheit, die mit der Partitionseinheit verbunden und ausgebildet ist, nur einen einzelnen Strang aus der übergeordneten Kette zur Ausführung und alle Stränge in der mindestens einen Arbeiter-Kette zur Ausführung anzuweisen bzw. zu bestimmen.

[0010] Ein weiterer Aspekt stellt ein Verfahren zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms mit Aufteilung-Vereinigung mit Funktionsaufrufen bereit. In einer Ausführungsform umfasst das Verfahren: (1) Unterteilen von Ketten in eine übergeordnete Kette und mindestens eine Arbeiter-Kette und (2) Anweisen bzw. Bestimmen nur eines einzelnen Strangs aus der übergeordneten Kette zur Ausführung und aller Stränge in der mindestens einen Arbeiter-Kette für die Ausführung.

KURZE BESCHREIBUNG

[0011] Es wird nun auf die folgenden Beschreibungen in Verbindung mit den begleitenden Zeichnungen verwiesen, in denen:

[0012] **Fig.** 1 eine Blockansicht eines SIMT-Prozessors ist, der ausgebildet ist, ein System oder ein Verfahren zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms mit Aufteilung-Verzweigung und mit Funktionsaufrufen zu enthalten oder auszuführen;

[0013] **Fig.** 2 eine Blockansicht einer Ausführungsform eines Systems zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms mit Aufteilung-Verzweigung und mit Funktionsaufrufen ist; und

[0014] **Fig.** 3 ein Flussdiagramm einer Ausführungsform eines Verfahrens zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms mit Aufteilung-Verzweigung und mit Funktionsaufrufen ist.

DETAILLIERTE BESCHREIBUNG

[0015] In einem SIMT-Prozessor werden mehrere Ausführungs-Stränge in Gruppen unterteilt. Alle Stränge in der Gruppe führen den gleichen Befehl zur gleichen Zeit aus. In grafischen Verarbeitungseinheiten (GPUs), die kommerziell von der Firma Nvidia, Santa Clara, Kalifornien erhältlich sind, die eine Art von SIMT-Prozessoren sind, werden Gruppen als „Wölbungen bzw. Ketten" bezeichnet, und sie werden in Blöcken ausgeführt.

[0016] Eine Pipeline-Steuereinheit des SIMT-Prozessors erzeugt, verwaltet, disponiert, führt aus und stellt bereit einen Mechanismus, um Gruppen zu synchronisieren. Nvidia-GPUs stellen einen bar.sync-Befehl zum Synchronisieren von Gruppen bereit. Nvidia-GPUs unterstützen ferner die Ausführung einer „divergenten" bedingten Verzweigung durch eine Gruppe; einige Stränge der Gruppe müssen die Verzweigung nehmen (da die Verzweigungsbedingungs-Vorbestimmung ein „wahr" ermittelt), und die anderen Stränge müssen auf den nächsten Befehl springen (da die Verzweigungsbedingungs-Vorbestimmung ein „falsch" ermittelt). Die Pipeline-Steuereinheit überwacht aktive Stränge in der Gruppe. Sie führt zuerst einen der Pfade (Verzweigung genommen oder Verzweigung nicht genommen) aus und führt dann den anderen Pfad aus; die entsprechenden Stränge werden in jedem Pfad aktiviert.

[0017] Es wird hierin erkannt, dass, während alle Stränge innerhalb eines GPU-Strangblocks an der gleichen Programmadresse starten, die Pipeline-Steuereinheit von einem Software-Mechanismus profitieren würde, der die Stränge in einen Haupt-Strang und Arbeiter-Stränge unterteilt und disponiert, so dass diese in dem Aufteilung-Vereinigungs-Modell ausführbar sind.

[0018] Es wird ferner hierin erkannt, dass gewisse Ausführungsformen des Software-Mechanismus die Stränge als Gruppen verwalten und synchronisieren sollten, da die Pipeline-Steuereinheit die Stränge als Gruppen verwaltet.

[0019] Es wird ferner hierin erkannt, dass, da das Hauptprogramm ein Einzel-Strang-Programm in dem Aufteilung-Vereinigungs-Modell ist, gewisse Ausführungsformen des Software-Mechanismus eine Semantik eines sequenziellen Bereichs erreichen sollten, ohne dass Nebenwirkungen eingeführt werden. Beispiele von Nebenwirkungen hervorrufenden Befehlen sind jene, die gemeinsam benutzte Ressourcen verwenden, etwa einen Lesebefehl oder einen Schreibbefehl für einen gemeinsam benutzten Speicher oder eine beliebige Code-Operation, die eine gemeinsam genutzte Ausnahmebehandlung (beispielsweise Division) aufrufen kann.

[0020] Es wird ferner hierin erkannt, dass gewisse Ausführungsformen des Software-Mechanismus Funktionen unterstützen sollten, die innerhalb des sequenziellen Bereichs und innerhalb des parallelen Bereichs aufgerufen werden können. Und derartige Funktionen können selbst parallele Konstrukte enthalten. Es wird ferner hierin erkannt, dass gewisse Ausführungsformen des Software-Mechanismus Funktionsaufrufe unterstützen sollten, die parallele Bereiche aufteilen-vereinigen können.

[0021] Weiterhin wird hierin erkannt, dass gewisse Ausführungsformen des Software-Mechanismus außenstehende bzw. äußere Funktionen unterstützen sollten, das heißt Funktionen, die nicht von dem gleichen Compiler wie das Programm kompiliert werden. Beispielsweise mathematische Funktionen in bestehenden GPU-Mathematik-Bibliotheken, und Systemfunktionen wie malloc, free und print. In gewissen Ausführungsformen sollten sowohl der übergeordnete Strang in den sequenziellen Bereichen als auch die Arbeiter-Stränge in dem parallelen Bereich in der Lage sein, eine äußere Funktion aufzurufen.

[0022] Folglich sind hierin diverse Ausführungsformen eines Systems und eines Verfahrens zur Kompilierung und Ausführung von datenparallelen Programmen mit Aufteilung-Vereinigung und mit Funktionsaufrufen in einem SIMT-Prozessor, etwa einer GPU, beschrieben.

[0023] Vor der Beschreibung gewisser Ausführungsformen des Systems und des Verfahrens wird ein SIMT-Prozessor beschrieben, der ausgebildet ist, ein System oder ein Verfahren zur Kompilierung oder zur Laufzeitausführung von datenparallelen Programmen mit Aufteilung-Vereinigung mit Funktionsaufrufen zu enthalten oder auszuführen.

[0024] **Fig.** 1 ist eine Blockansicht eines SIMT-Prozessors **100**. Der SIMT-Prozessor **100** enthält mehrere Strang-Prozessoren oder Kerne **106**, die in Stranggruppen **104** oder „Wölbungen bzw. Ketten" eingeteilt sind. Der SIMT-Prozessor **100** enthält J Stranggruppen **104-1** bis **104-J**, wovon jede K Kerne **106-1** bis **106-K** aufweist. In gewissen Ausführungsformen können die Stranggruppen **104-1** bis **104-J** weiter in einen oder mehrere Strangblöcke **102** eingeteilt sein. Gewisse Ausführungsformen umfassen zweiunddreißig Kerne **106** pro Stranggruppe **104**. Andere Ausführungsformen können weniger als vier Kerne in einer Stranggruppe und bis zu mehrere zehntausend Kerne enthalten. Gewisse Ausführungsformen teilen die Kerne **106** in eine einzelne Stranggruppe **104** ein, während andere Ausführungsformen Hunderte oder sogar Tausende von Stranggruppen **104** aufweisen. Alternative Ausführungsformen des SIMT-Prozessors **100** können die Kerne **106** ausschließlich in die Stranggruppen **104** einteilen, wobei die Einteilung auf Ebene von Strangblöcken nicht vorhanden ist.

[0025] Der SIMT-Prozessor **100** umfasst ferner eine Pipeline-Steuereinheit **108**, einen von Blöcken gemeinsam benutzten Speicher **110** und ein Array aus lokalen Speichern **112-1** bis **112-J**, die den Stranggruppen **104-1** bis **104-J** zugeordnet sind. Die Pipeline-Steuereinheit **108** verteilt Aufgaben an die diversen Stranggruppen **104-1** bis **104-J** über einen Datenbus **114**. Die Kerne **106** in einer Stranggruppe **106-j** arbeiten parallel zueinander. Die Stranggruppen **104-1** bis **104-J** kommunizieren über einen Speicherbus **116** mit dem von Blöcken gemeinsam benutzten Speicher **110**. Die Stranggruppen **104-1** bis **104-J** kommunizieren entsprechend mit den lokalen Speichern **112-1** bis **112-J** über lokale Busse **118-1** bis **118-J**. Beispielsweise verwendet eine Stranggruppe **104-J** den lokalen Speicher **112-J** mittels Kommunikation über einen lokalen Bus **118-J**. Gewisse Ausführungsformen des SIMT-Prozessors **100** weisen einen gemeinsam benutzten Teilbereich des von Blöcken gemeinsam benutzten Speichers **110** jedem Strangblock **102** zu und ermöglichen Zugriff auf gemeinsam benutzte Teilbereiche des von Blöcken gemeinsam benutzten Speichers **110** für alle Stranggruppen **104** innerhalb eines Strangblocks **102**. Gewisse Ausführungsformen umfassen Stranggruppen **104**, die ausschließlich den lokalen Speicher **112** benutzen. Viele andere Ausführungsformen umfassen Stranggruppen **104**, die eine ausgewogene Nutzung des lokalen Speichers **112** und des von Blöcken gemeinsam benutzten Speichers **110** bewerkstelligen.

[0026] Die Ausführungsform aus **Fig.** 1 enthält eine übergeordnete Stranggruppe **104-1**. Jede der verbleibenden Stranggruppen **104-2** bis **104-J** wird als „Arbeiter-"Stranggruppe betrachtet. Die übergeordnete Stranggruppe **104-1** enthält zahlreiche Kerne, wovon einer ein übergeordneter Kern **106-1**, der schließlich einen übergeordneten Strang ausführt. Programme, die in dem SIMT-Prozessor **100** ausgeführt werden, sind als eine Sequenz aus Kernels aufgebaut. Typischerweise beendet jeder Kernel seine Ausführung, bevor der nächste Kernel beginnt. In gewissen Ausführungsformen kann der SIMT-Prozessor **100** mehrere Kernels parallel ausführen, wobei dies von der Größe der Kernels abhängt. Jeder Kernel ist als eine Hierarchie aus Strängen aufgebaut, die in den Kernen **106** auszuführen sind.

[0027] Nach der Beschreibung eines SIMT-Prozessors, in welchem das System oder das Verfahren, wie sie hierin eingeführt sind, enthalten oder ausgeführt werden kann, werden nunmehr diverse Ausführungsformen des Systems und des Verfahrens beschrieben.

[0028] Eine Ausführungsform des hierin eingeführten Systems umfasst einen Compiler und eine Geräte-Laufzeitbibliothek. Die Geräte-Laufzeitbibliothek realisiert die Funktion der Strang- und Gruppenverwaltung. Der Compiler übersetzt ein datenparalleles Programm mit Aufteilung-Vereinigung in ein Haupt-Strangprogramm und eine Gruppe aus äußeren Funktionen, wovon jede einem parallelen Konstrukt entspricht. Der übersetzte Code nimmt Aufrufe von Funktionen der Geräte-Laufzeitbibliothek vor, um die Strang- und Gruppenverwaltung auszuführen.

[0029] Die nachfolgende Tabelle 2 zeigt ein Beispielprogramm, um die Compiler-Übersetzung und die Realisierung der Geräte-Laufzeit darzustellen.

```
extern ext();

main()
{
   foo();
   ext();
   #pragma parallel loop
   for (…) {
      foo();
      bar();
   }
   #pragma parallel loop
   for (…)    {
      ext();
}
   bar();
}
```

```
bar()
{
   Statements 1;
   #pragma parallel loop
   for (…) {
      bar_par_body;
   }
   Statement 2;
}

foo()
{
   foo_body;
}
```

Tabelle 2 – Beispielprogramm für die Compiler-Übersetzung und die Realisierung der Geräte-Laufzeit

[0030] Der Ablauf des main()-Programms aus Tabelle 2 beginnt mit dem einzelnen übergeordneten Strang. Der übergeordnete Strang ruft die Funktion foo() auf, die einen Körper aufweist, der für diesen Compiler transparent ist und von diesem übersetzt wird. Der übergeordnete Strang ruft dann eine Funktion ext() auf, d. h. eine externe oder außen liegende Funktion mit einem Körper, der für diesen Compiler unsichtbar ist. Aufrufe von außen liegenden Funktionen werden übersetzt, wie sie sind, ohne dass eine spezielle Handhabung durch den Compiler erfolgt. Der übergeordnete Strang trifft dann auf den ersten parallelen Bereich. Arbeiter-Stränge führen den parallelen Bereich aus, während der übergeordnete Strang auf deren Beendigung wartet. Innerhalb des parallelen Bereichs ruft jeder Arbeiter-Strang die Funktion foo() und bar() auf. Die Funktion bar() enthält einen weiteren parallelen Bereich; jedoch liegt bar() bereits innerhalb eines parallelen Bereichs. Da bar() bereits innerhalb eines Parallelenbereichs liegt, wird der parallele Bereich innerhalb von bar() sequenziell von jedem Arbeiter-Strang ausgeführt.

[0031] Nach dem ersten parallelen Bereich trifft der übergeordnete Strang auf einen zweiten parallelen Bereich. Innerhalb des zweiten parallelen Bereichs ruft jeder Arbeiter-Strang die externe außen liegende Funktion ext() auf. Nach dem zweiten parallelen Bereich ruft der übergeordnete Strang die Funktion bar() auf. Innerhalb von bar() trifft der übergeordnete Strang auf einen dritten parallelen Bereich, der wiederum von den Arbeiter-Strängen abgearbeitet wird.

[0032] Die Funktion main() ist als eine Eintrittsfunktion bekannt, da dies der Punkt ist, an welchem das Programm startet. Funktionen, etwa foo() und bar() sind Nicht-Eintrittsfunktionen.

[0033] Für eine Eintrittsfunktion stellt der Compiler zuerst eine geklonte Kopie, die als main_core() bezeichnet wird. Die geklonte Kopie wird dann als eine Nicht-Eintrittsfunktion verarbeitet, wie nachfolgend beschrieben ist. Für die main()-Funktion erzeugt der Compiler einen Code, der in Tabelle 3 nachfolgend gezeigt ist, wobei groupID() die ID der Stranggruppe zurückgibt, die einen Befehl ausführenden Strang enthält. threadID() gibt die ID des Strangs zurück. init(), signal_done() und scheduler() sind Funktionen in der Geräte-Laufzeitbibliothek.

```
main()
{
   if (groupID() == 0) {
      if (threadID() == 0) {
         init();
         main_core();
         signal_done();
      }
  }
  else
      scheduler();
}
```

Tabelle 3 – Compiler-erzeugter Beispiels-Code

[0034] Wenn ein GPU-Strangblock startet, führen alle Stränge innerhalb des Blocks main() aus; jedoch nehmen sie unterschiedliche Pfade. Der Strang 0 ist der übergeordnete Strang und führt init(), main_core() und signal_done() aus. Andere Stränge in der Gruppe 0 gehen geradewegs zum Ende der main()-Funktion und warten dort. Die Stränge in den verbleibenden Gruppen führen scheduler() aus.

[0035] Für eine Nicht-Eintrittsfunktion wie foo(), bar() und main core() übersetzt der Compiler den Code so, als ob kein paralleles Konstrukt existieren würde. Wenn eine Nicht-Eintrittsfunktion ein paralleles Konstrukt enthält, dann erzeugt der Compiler für jedes parallele Konstrukt eine Funktion, die den Körper des parallelen Konstrukts enthält (eine ausgelagerte Funktion), und erzeugt dann eine bedingte Verzweigung, die prüft, ob der gerade ausgeführte Strang der übergeordnete Strang ist. In der falschen Verzweigung fügt der Compiler Code ein, der die Schleife ausführt. In der wahren Verzweigung fügt der Compiler Aufrufe in die Geräte-Laufzeitbibliothek ein, um Aufgaben zuzuordnen, Arbeiter-Stränge aufzuwecken und eine Barriere auszuführen. Die Bedingung ist wahr, wenn die Nicht-Eintrittsfunktion außerhalb der parallelen Bereiche aufgerufen wird. Die Bedienung ist falsch, wenn die Nicht-Eintrittsfunktion innerhalb eines parallelen Bereichs aufgerufen wird, in welchem Falle die parallele Schleife von dem ausführenden Strang sequenziell ausgeführt wird.

[0036] Zum Beispiel ist der übersetzte Code für die Funktion bar() nachfolgend in Tabelle 4 gezeigt.

```
bar()
{
        Anweisungen 1;

   if (groupID() == 0 && threadID() == 0)  {
      signal_task(bar_par_func);
      barrier();
   }
   else {
      for (…) {
         bar_par_body();
      }
   }
        Anweisungen 2;

}
```

```
bar_par_func()
{
        for (…) { // Schleifen Iteration ist auf die

                        Arbeiter-Stränge aufgeteilt


      bar_par_body;
   }
}
```

Tabelle 4 – übersetzter Code für die Funktion bar ()

[0037] signal task() und barrier() sind Funktionen in der Geräte-Laufzeitbibliothek. bar_par_frunc() ist die ausgelagerte Funktion, die dem parallelen Konstrukt in der ursprünglichen Funktion bar() entspricht.

[0038] In dieser Ausführungsform enthält die Geräte-Laufzeitbibliothek unter anderem die folgenden Funktionen: init(), scheduler(), signal task(), signal done(), und barrier(). Die Bibliothek realisiert auch die folgenden Funktionen für die interne Verwendung: signal(), wait() und fetch_task().

[0039] Alle Arbeiter-Stränge führen die scheduler()-Funktion aus. Die Arbeiter-Stränge durchlaufen einen Schlaf-Aufwach-Ausführen-Zyklus, bis sie angewiesen werden, zu enden.

```
shared bool exit_flag ;

scheduler()
{
   while (1) {
      wait();
      if (exit_flag)
         return;
      }
```

```
        task = fetch_task();
        task();
        barrier();
    }
}
```

Tabelle 5 – Beispiel-Code unter Verwendung einer Marke zum Verlassen des Programms

[0040] Es wird eine Bool'sche Variable 'exit_flag' in dem von Blöcken gemeinsam benutzten Speicher abgelegt und auf diese kann von allen Strängen innerhalb des Strangblocks zugegriffen werden. Sie wird von dem übergeordneten Strang verwendet, um den Arbeiter-Strängen mitzuteilen, ob sie alle die Ausführung beenden sollen. Die 'exit_flag' wird in der init()-Funktion auf falsch gesetzt, und wird in der signal_done()-Funktion auf wahr gesetzt. Beide Funktionen werden von dem übergeordneten Strang aufgerufen.

```
init()
{
    exit_flag = false;
}

signal_done()
{
    exit_flag = true;
    signal();
}
```

Tabelle 6 – Beispiel-Code für Änderung des Zustands der Marke für das Verlassen des Programms

[0041] Ein weiterer Teil des von Blöcken gemeinsam benutzten Speichers wird verwendet, um die aktuelle Aufgabe mitzuteilen. Die aktuelle Aufgabe wird von dem übergeordneten Strang in der signal_task()-Funktion festgelegt und wird von dem Arbeiter-Strang in der fetch_task()-Funktion abgeholt. Der von Blöcken gemeinsam benutzte Speicher enthält den Zeiger auf die ausgelagerte Funktion, die dem parallelen Konstrukt entspricht.

```
shared function_ptr task;
signal_task(function_ptr a_task)
{
    task = a_task;
    signal();
}
fetch_task()
{
    return task;
}
```

Tabelle 7 – Beispiel-Code zur Verwendung eines Zeigers, um eine aktuelle Aufgabe zu kennzeichnen

[0042] Da die parallelen Bereiche der Reihe nach innerhalb eines Strangblocks ausgeführt werden, ist zu jedem Zeitpunkt nur eine einzelne Aufgabe aktiv. Wenn die parallelen Bereiche asynchron ausgeführt werden können, wird typischerweise eine kompliziertere Datenstruktur, etwa ein Stapel, eine Warteschlange oder ein Datenbaum benötigt, um die aktiven Aufgaben zu speichern.

[0043] barrier(), signal() und wait()-Funktionen werden unter Anwendung einer Hardware-Barriere realisiert.

```
barrier()
{
   bar.sync;
}
```

```
wait()
{
   bar.sync;
}
signal()
{
   bar.sync;
}
```

Tabelle 8 – Beispielfunktionen für barrier(), signal() und wait()

[0044] **Fig.** 2 ist eine Blockansicht einer Ausführungsform eines Systems **200** zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms unter Aufteilung-Vereinigung mit Funktionsaufrufen. Das Ein Programm **210** enthält Eintrittsfunktionen **212**, Nicht-Eintrittsfunktionen **214** und äußere Funktionen **216**. Das System **200** umfasst eine Partitionseinheit **202**, eine Strang-Zuordnungseinheit **204**, eine Strang-Disponiereinheit **206**, einen Funktionsprozessor **208**, eine Geräte-Laufzeitbibliothek **218** und einen SIMT-Prozessor **100** aus **Fig.** 1.

[0045] Der SIMT-Prozessor **100** umfasst die Pipeline-Steuereinheit **108**, den Datenbus **114**, die lokalen Busse **118-1** und **118-2** und den gemeinsam benutzten Speicher **110** aus **Fig.** 1. In der Ausführungsform aus **Fig.** 2 ist der SIMT-Prozessor **100** so dargestellt, dass er einen einzelnen Strangblock hat, der zwei Stranggruppen enthält: die übergeordnete Stranggruppe **104-1** und die Arbeiter-Stranggruppe **104-2**. Die Stranggruppen **104-1** und **104-2** enthalten jeweils Stränge **106**.

[0046] Die Partitionseinheit **202** bestimmt bzw. weist die Stranggruppe **104-1** als die übergeordnete Stranggruppe und die verbleibenden Stranggruppen als Arbeiter-Stranggruppen aus. In der Ausführungsform aus **Fig.** 2 ist eine einzelne Arbeiter-Stranggruppe **104-2** gezeigt. In alternativen Ausführungsformen können viele Arbeiter-Stranggruppen verwendet werden. Die Strang-Zuordnungseinheit **204** bestimmt einen übergeordneten Strang **106-1** der übergeordneten Stranggruppe **104-1**. Alle anderen Stränge in der übergeordneten Stranggruppe **104-1** sind untätig. Die Strang-Zuordnungseinheit **204** bestimmt bzw. weist auch jeden der Stränge **106** in der Arbeiter-Stranggruppe **104-2** als Arbeiter-Stränge aus.

[0047] Die Strang-Disponiereinheit **206** übersetzt das Programm **210** derart, dass die Pipeline-Steuereinheit **108** die Ausführung des übergeordneten Strangs **106-1** und der diversen Arbeiter-Stränge in der Arbeiter-Stranggruppe **104-2** in geeigneter Weise gesteuert. Die Strang-Disponiereinheit **206** übersetzt das Programm **210** derart, dass, wenn die Ausführung des übergeordneten Strangs beginnt, eine Marke für das Verlassen eines Programms zurückgesetzt wird. Die Strang-Disponiereinheit **206** disponiert den übergeordneten Strang **106-1** so, dass der ausgeführt wird, bis ein paralleler Bereich oder das Ende des Programms **210** erreicht sind. Wenn ein paralleler Bereich des Programms **210** erreicht wird, legt die Strang-Disponiereinheit **206** eine parallele Aufgabe fest und die Arbeiter-Stränge in der Arbeiter-Stranggruppe **104-2** beginnen mit der Ausführung. Die Strang-Disponiereinheit **206** legt ferner eine Barriere für jeden der Arbeiter-Stränge so fest, dass, wenn in die Barriere eingetreten wird, der übergeordnete Strang **106-2** seine Ausführung fortsetzt. Wenn das Ende des Programms **210** erreicht wird, wird die Marke für das Verlassen des Programms aktiviert, wodurch bewirkt wird, dass alle Arbeiter-Stränge die Ausführung beenden.

[0048] Der Funktionsprozessor **208** operiert auf den Funktionen des Programms **210**. Die Verarbeitung der Eintrittsfunktionen **212** beinhaltet die Erzeugung einer Klon-Kopie einer Eintrittsfunktion, die dann als eine Nicht-Eintrittsfunktion verarbeitet wird. Die ursprüngliche Eintrittsfunktion wird so verarbeitet, dass der übergeordnete Strang **106-1** die Klon-Kopie zusätzlich zu anderen Aufrufen verarbeiten wird, und die Arbeiter-Stränge führen Zyklen mit Schlafmodus, Aufwachen, Abholung und Ausführung der parallelen Aufgabe aus, die von der Strang-Disponiereinheit **206** festgelegt ist.

[0049] Der Funktionsprozessor **208** übersetzt die Nicht-Eintrittsfunktionen **214** auf zwei Arten. Wenn kein paralleles Konstrukt in einer Nicht-Eintrittsfunktion vorhanden ist, dann wird die Funktion einfach verarbeitet, wie sie ist. Wenn ein paralleles Konstrukt vorhanden ist, wird eine äußere Funktion, die den Körper des parallelen Konstrukts enthält, erzeugt. Der Funktionsprozessor **208** erzeugt dann eine Verzweigungsbedingung, die das parallele Konstrukt entweder sequenziell ausführt oder die Geräte-Laufzeitbibliothek **218** verwendet, um eine Aufgabe zuzuweisen, Arbeiter-Stränge aufzuwecken und eine Barriere auszuführen, wie dies zuvor beschrieben ist. Die Aufwachen- und Schlaffunktion werden unter Verwendung von Hardware-Barrierenfunktionen der Geräte-Laufzeitbibliothek **218** realisiert. Stränge an Barrieren werden nicht für die Ausführung durch die Hardware disponiert, so dass sie keine Arbeitszyklen verschwenden. In der übergeordneten Stranggruppe **104-1** nimmt nur der übergeordnete Strang **106-1** an den Barrieren teil. Dies ist deswegen so, weil die Hardware-Barriere gruppenbasiert ist. Eine Gruppe wird als an einer Barriere liegend erachtet, wenn jeder Strang innerhalb der Gruppe an der Barriere steht.

[0050] Ähnlich zu der Verarbeitung von Nicht-Eintrittsfunktionen, die kein paralleles Konstrukt aufweisen, werden äußere Funktionen **216** von dem Funktionsprozessor **208** so verarbeitet, wie sie sind.

[0051] Ein datenparalleles Programm mit Aufteilung-Vereinigung wird in ein übergeordnetes Programm und eine Gruppe aus parallelen Aufgaben unterteilt. Das übergeordnete Programm ist das Programm, das von dem übergeordneten Strang ausgeführt wird. Eine parallele Aufgabe entspricht einem parallelen Bereich, der von den Arbeiter-Strängen ausgeführt wird. Das übergeordnete Programm enthält Disponierpunkte, an denen der übergeordnete Strang eine parallele Aufgabe zuweisen bzw. bestimmen wird, die Arbeiter-Stränge aufwecken wird und darauf wartet, dass die Arbeiter-Stränge abgeschlossen sind.

[0052] Der spezielle übergeordnete Strang in der speziellen übergeordneten Gruppe wird den sequenziellen Bereich des Programms ausführen.

[0053] Alternativ kann ein Einzel–Strangverhalten in dem sequenziellen Bereich nachgebildet werden, während alle Stränge in der Gruppe den Code ausführen. Jedoch weisen Nachbildungsschemata Beschränkungen im Hinblick auf die Komplexität hinsichtlich des Leistungsverhaltens und der Erzeugung auf, so dass diese weniger brauchbar sind. Die notwendige Vorbestimmung und Synchronisierung ergibt zusätzlichen Aufwand bei der Ausführung. Ferner müssen alle Funktionen, die von dem sequenziellen Bereich und dem parallelen Bereich aufgerufen werden, unterschiedlich geklont und kontrolliert werden.

[0054] Bei gegebener Unterteilung der Stränge und Gruppen vollführen die Arbeiter-Stränge und der übergeordnete Strang die folgenden Lebenszyklen aus: Eine Ausführungsform eines Arbeiter-Strangs durchläuft die folgenden Phasen in einem Lebenszyklus:

1) der Strangblock startet;
2) im Schlafmodus, bis das Aufwecken durch den übergeordneten Strang erfolgt;
3) Verlassen des Programms, wenn die Marke zum Verlassen des Programms auf wahr gesetzt ist;
4) Abholen und Ausführung der Aufgabe, die von dem übergeordneten Strang zugewiesen wird;
5) Eintreten in eine Barriere und
6) Zurückkehren zur Stufe 2.

[0055] Eine Ausführungsform eines übergeordneten Strangs durchläuft die folgenden Stufen in einem Lebenszyklus:

1) der Strangblock startet;
2) Setzen der Marke für das Verlassen des Programms auf falsch;
3) Ausführen des übergeordneten Programms, bis ein paralleler Bereich oder das Ende des übergeordneten Programms erreicht werden;
4) am Anfang eines parallelen Bereichs:
a. Festlegen einer parallelen Aufgabe,
b. Aufwecken der Arbeiter-Stränge,
c. Eintreten in eine Barriere, und
d. Fortsetzen des übergeordneten Programms (Stufe 3); und
5) am Ende des übergeordneten Programms:
a. Setzen der Marke für das Verlassen eines Programms auf wahr,
b. Aufwecken der Arbeiter-Stränge, und c. Ende.

[0056] Die anderen Stränge in der übergeordneten Gruppe warten im Wesentlichen auf das Ende des Programms in untätiger Weise. Das Programm wird abwechselnd von dem übergeordneten Strang und den Ar-

beiter-Strängen ausgeführt. Dies führt zu einer guten Auslastung des Befehls-Cache-Speichers, die besser ist als die Auslastung, die durch ein Verfahren hervorgerufen wird, in welchem sowohl der übergeordnete Strang als auch die Arbeiter-Stränge aktiv sind und unterschiedliche Code-Pfade ausführen.

[0057] Fig. 3 ist ein Flussdiagramm einer Ausführungsform eines Verfahrens zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms unter Aufteilung-Vereinigung mit Funktionsaufrufen. Das Verfahren beginnt in einem Startschritt **310**. In einem Schritt **320** werden Stranggruppen innerhalb eines Strangblocks in eine übergeordnete Stranggruppe und mindestens eine Arbeiter-Stranggruppe aufgeteilt. In einem Schritt **330** wird ein einzelner Strang aus der übergeordneten Stranggruppe als der übergeordnete Strang ausgewiesen bzw. bestimmt. Die verbleibenden Stränge der übergeordneten Gruppe sind während der Ausführung im Wesentlichen untätig. Ferner werden im Schritt **330** alle Stränge in der mindestens einen Arbeiter-Stranggruppe als Arbeiter-Stränge ausgewiesen bzw. bestimmt. Das Verfahren endet in einem Schlussschritt **340**.

[0058] Der Fachmann auf diesem Gebiet, an den sich diese Anmeldung richtet, erkennt, dass andere und weitere Hinzufügungen, Streichungen, Ersetzungen und Modifizierungen an den beschriebenen Ausführungsformen vorgenommen werden können.

Patentansprüche

1. Ein System zur Kompilierung oder Laufzeitausführung eines datenparallelen Programms mit Aufteilung-Vereinigung mit Funktionsaufrufen, mit:
einer Partitionseinheit, die ausgebildet ist, Gruppen in eine übergeordnete Gruppe und mindestens eine Arbeiter-Gruppe aufzuteilen; und
einer Strang-Zuordnungseinheit, die mit der Partitionseinheit verbunden und ausgebildet ist, nur einen Strang aus der übergeordneten Gruppe zur Ausführung als übergeordneten Strang zu bestimmen und alle Stränge in der mindestens einen Arbeiter-Gruppe zur Ausführung als Arbeiter-Stränge zu bestimmen.

2. Das System nach Anspruch 1, das ferner eine Strang-Disponiereinheit aufweist, die mit der Strang-Zuordnungseinheit verbunden und ausgebildet ist, eine Pipeline-Steuereinheit eines Einzelbefehl-Multi-Strang-Prozessors zu veranlassen, die Ausführung des übergeordneten Strangs wie folgt zu disponieren:
es wird eine Marke zum Verlassen eines Programms auf einen ersten Zustand gesetzt, wenn der übergeordnete Strang mit der Ausführung beginnt;
der übergeordnete Strang wird ausgeführt, bis er einen parallelen Bereich oder ein Ende des Programms erreicht;
beim Erreichen des parallelen Bereichs wird eine parallele Aufgabe festgelegt, in eine erste Barriere eingetreten, in eine zweite Barriere eingetreten und die Ausführung des übergeordneten Strangs fortgesetzt; und
beim Erreichen des Endes wird die Marke zum Verlassen eines Programms auf einen zweiten Zustand gesetzt, in eine Barriere eingetreten und der übergeordnete Strang beendet die Ausführung.

3. Das System nach Anspruch 1 oder 2, das ferner eine Strang-Disponiereinheit aufweist, die mit der Strang-Zuordnungseinheit verbunden und ausgebildet ist, eine Pipeline-Steuereinheit eines Einzelbefehl-Multi-Strang-Prozessors zu veranlassen, die Ausführung der Arbeiter-Stränge wie folgt zu disponieren:
die Arbeiter-Stränge treten in eine erste Barriere ein;
die Arbeiter-Stränge beginnen die Ausführung einer parallelen Aufgabe, die von dem übergeordneten Strang festgelegt ist, wenn der übergeordnete Strang einen parallelen Bereich erreicht und der übergeordnete Strang in eine Barriere eintritt;
die Arbeiter-Stränge treten in eine zweite Barriere bei Abschluss der parallelen Aufgabe ein; und
die Arbeiter-Stränge enden, wenn eine Flagge für das Verlassen eines Programms auf einen zweiten Zustand gesetzt wird und der übergeordnete Strang in eine Barriere eintritt.

4. Das System nach einem der Ansprüche 1–3, das ferner eine Strang-Disponiereinheit umfasst, die mit der Strang-Zuordnungseinheit verbunden und ausgebildet ist, eine Barrierenfunktion einer Pipeline-Steuereinheit eines Einzelbefehl-Multi-Strang-Prozessors zur Steuerung der Ausführung und Beendigung der Arbeiter-Stränge zu verwenden.

5. Das System nach einem der Ansprüche 1–4, das ferner einen Funktionsprozessor aufweist, der mit der Strang-Zuordnungseinheit verbunden und ausgebildet ist, eine geklonte Kopie einer Eintrittsfunktion zu erzeugen und die Eintrittsfunktion als eine Nicht-Eintrittsfunktion zu verarbeiten.

6. Das System nach einem der Ansprüche 1–5, das ferner einen Funktionsprozessor aufweist, der mit der Strang-Zuordnungseinheit verbunden und ausgebildet ist, eine Nicht-Eintrittsfunktion zu übersetzen, die ein paralleles Konstrukt aufweist, durch:
Erzeugung einer Funktion, die einen Körper des parallelen Konstrukts enthält; und
Einfügung von Aufrufen, die in eine Geräte-Laufzeitbibliothek zielen, wenn die Funktion in dem übergeordneten Strang ausgeführt wird.

7. Das System nach einem der Ansprüche 1–6, das ferner einen Funktionsprozessor aufweist, der mit der Strang-Zuordnungseinheit verbunden und ausgebildet ist, Aufrufe von äußeren Funktionen unverändert zu übersetzen.

8. Das System nach einem der Ansprüche 1–7, das ferner einen Funktionsprozessor aufweist, der mit der Strang-Zuordnungseinheit verbunden und ausgebildet ist, eine Geräte-Laufzeitbibliothek zu verwenden, die Funktionen bereitstellt, die von kompiliertem Anwender-Code und internen Funktionen aufgerufen werden können.

9. Das System nach einem der Ansprüche 1–8, das ferner einen Funktionsprozessor aufweist, der mit der Strang-Zuordnungseinheit verbunden und ausgebildet ist, eine Austrittsmarke zu verwenden, die in einem gemeinsam benutzten Speicher gespeichert ist und verwendet wird, um dem übergeordneten Strang zu ermöglichen, um den Arbeiter-Strängen mitzuteilen, wann sie ihre Ausführung beenden sollen.

10. Das System nach einem der Ansprüche 1–9, wobei das System ausgebildet ist, einen gemeinsam benutzten Speicher eines Einzelbefehl-Multi-Prozessors zur Kennzeichnung einer aktuellen Aufgabe zu konfigurieren.

Es folgen 2 Seiten Zeichnungen

Anhängende Zeichnungen

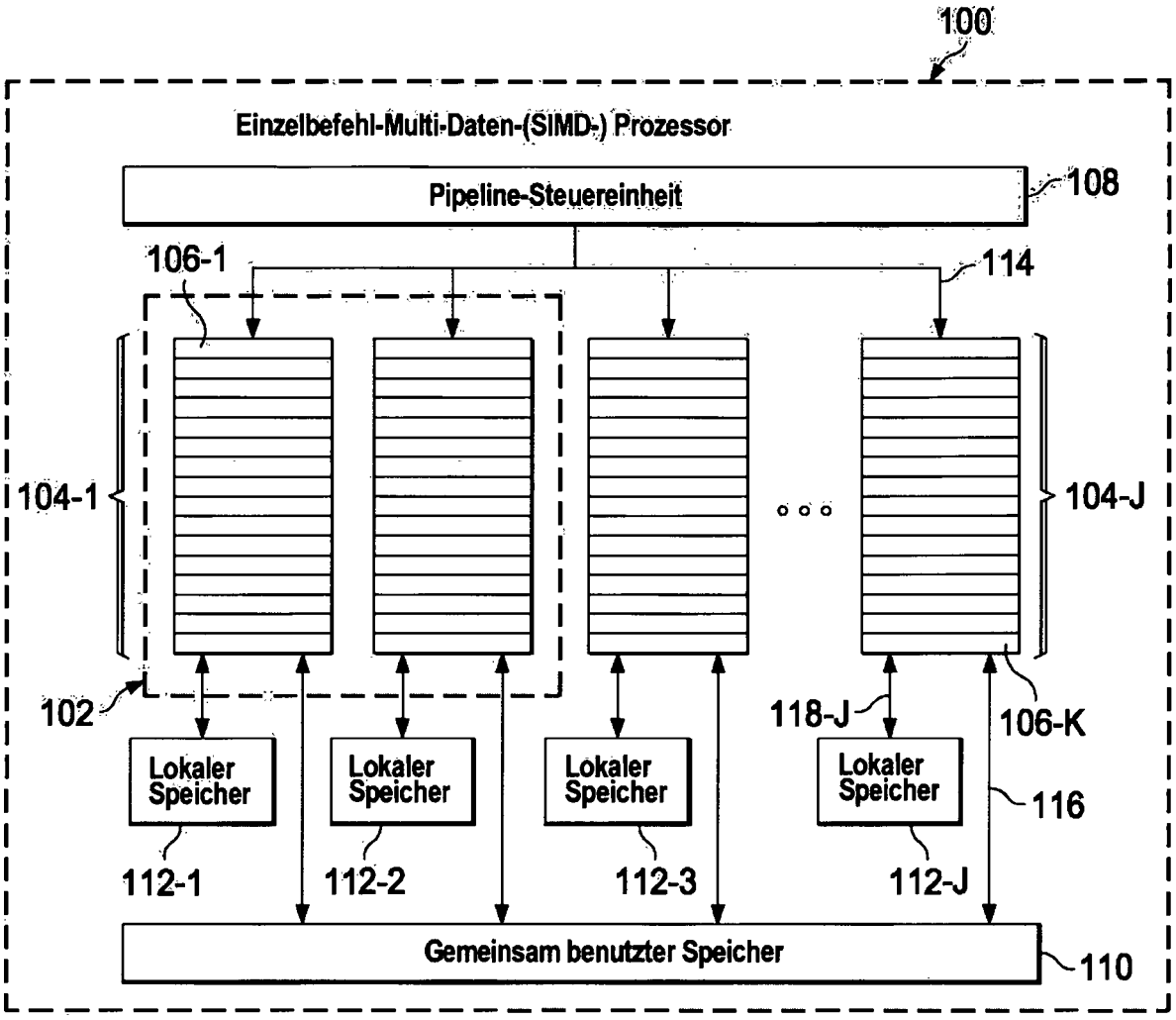


FIG. 1

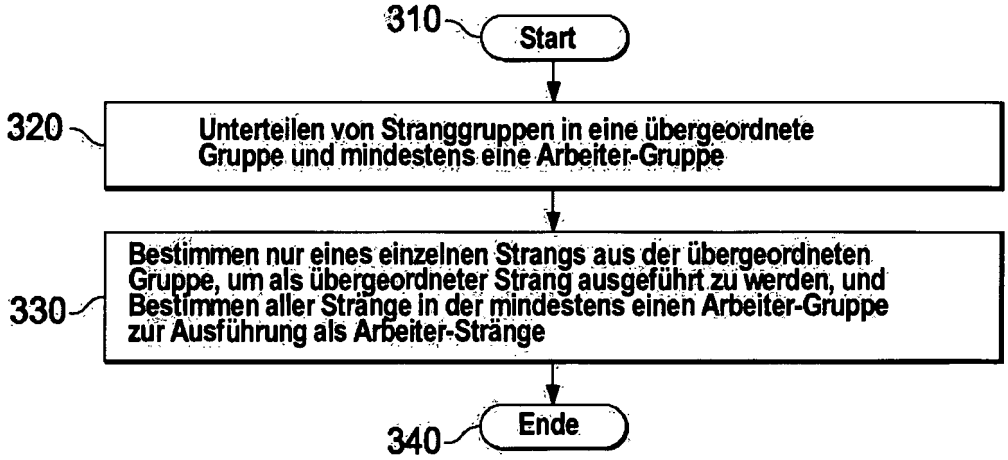


FIG. 3

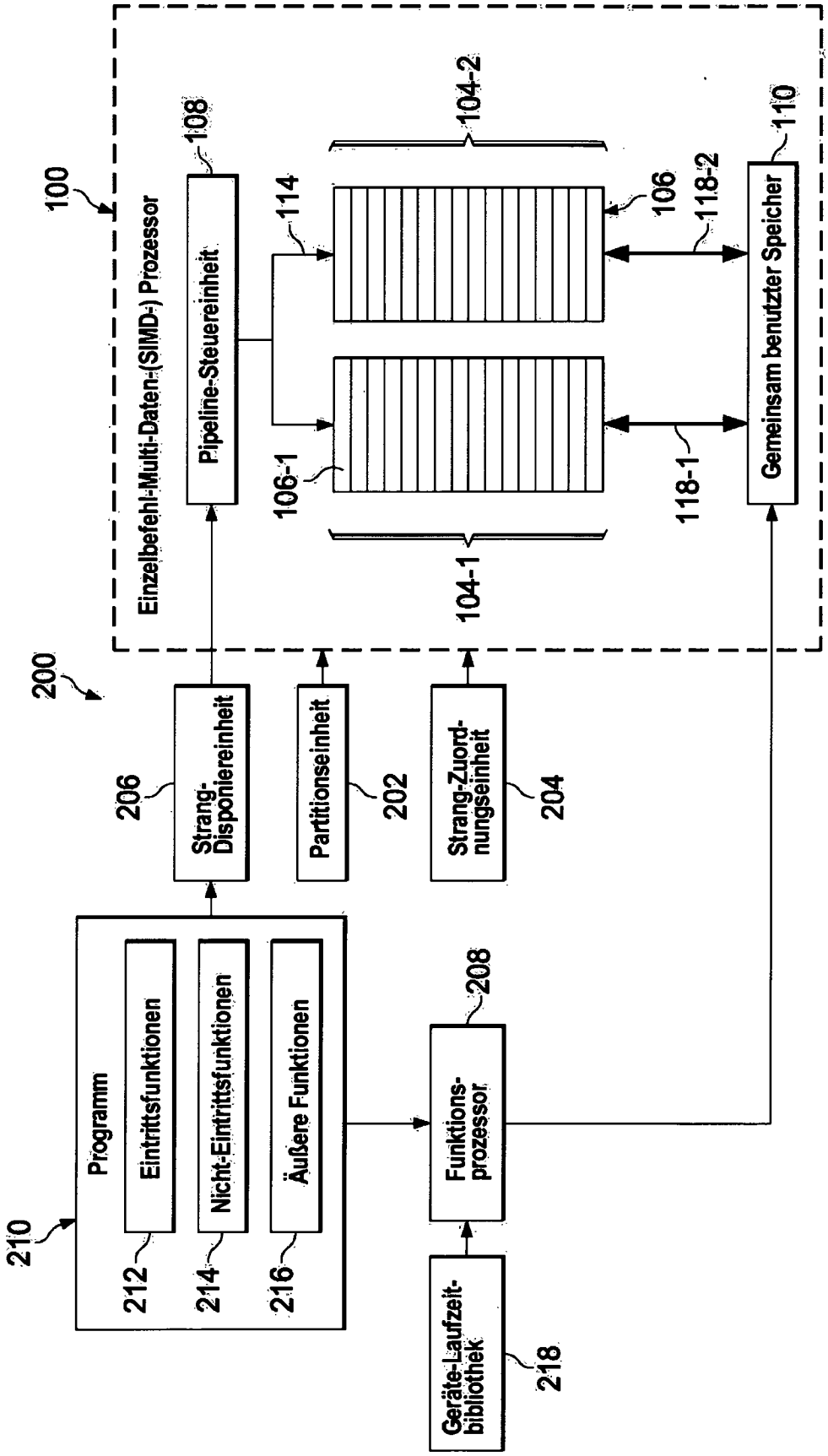

100
Einzelbefehl-Multi-Daten-(SIMD-) Prozessor
108
Pipeline-Steuereinheit
114
104-2
106
118-2
110
Gemeinsam benutzter Speicher
106-1
118-1
104-1
200
206
Strang-Dispo-niereinheit
Partitionseinheit
202
Strang-Zuord-nungseinheit
204
Programm
Eintrittsfunktionen
Nicht-Eintrittsfunktionen
Äußere Funktionen
208
Funktions-prozessor
210
212
214
216
Geräte-Laufzeit-bibliothek
218

FIG. 2