(19) Deutsches Patent- und Markenamt

(10) **DE 10 2018 005 216 A1** 2019.02.21

(12) # Offenlegungsschrift

(21) Aktenzeichen: **10 2018 005 216.9**
(22) Anmeldetag: **29.06.2018**
(43) Offenlegungstag: **21.02.2019**

(51) Int Cl.: ***G06F 9/30*** (2018.01)

(30) Unionspriorität:
15/640,533 **01.07.2017** **US**

(71) Anmelder:
INTEL CORPORATION, Santa Clara, Calif., US

(74) Vertreter:
Samson & Partner Patentanwälte mbB, 80538 München, DE

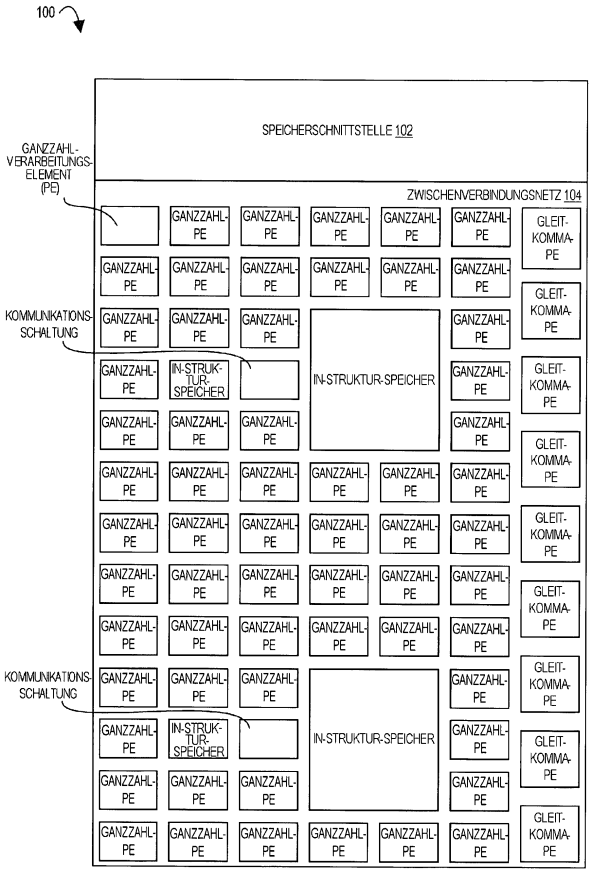
(72) Erfinder:
Fleming, Kermin, Hudson, Mass., US; Glossop, Kent, Nashua, N.H., US; Steely, Simon C. Jr., Hudson, N.H., US; Sury, Samantika S., Westford, Mass., US

Prüfungsantrag gemäß § 44 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen.

(54) Bezeichnung: **Prozessoren, Verfahren und Systeme für einen konfigurierbaren, räumlichen Beschleuniger mit Transaktions- und Wiederholungsmerkmalen**

(57) Zusammenfassung: Es werden Systeme, Verfahren und Vorrichtungen bezüglich eines konfigurierbaren räumlichen Beschleunigers beschrieben. In einer Ausführungsform weist ein Prozessor mehrere Verarbeitungselemente auf; und ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen zum Empfangen einer Eingabe eines Datenflussgraphen, der mehrere Knoten umfasst, wobei der Datenflussgraph in das Zwischenverbindungsnetz und die mehreren Verarbeitungselemente zu überlagern, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist, und die mehreren Verarbeitungselemente eine atomare Operation durchzuführen haben, wenn ein eingehender Operand bei den mehreren Verarbeitungselementen eingeht.



Beschreibung

AUSSAGE BEZÜGLICH DER STAATLICH GEFÖRDERTEN FORSCHUNG UND ENTWICKLUNG

[0001] Diese Erfindung wurde mit Unterstützung der Regierung unter Vertragsnummer H98230A-13-D-0124, verliehen vom Ministerium für Verteidigung, erstellt. Die Regierung besitzt gewisse Rechte auf diese Erfindung.

TECHNISCHES GEBIET

[0002] Die Offenbarung betrifft allgemein Elektronik, und spezifischer betrifft eine Ausführungsform der Offenbarung einen konfigurierbaren räumlichen Beschleuniger.

Hintergrund

[0003] Ein Prozessor oder Satz von Prozessoren führt Befehle aus einem Befehlssatz aus, z. B. der Befehlssatzarchitektur (ISA). Der Befehlssatz ist Teil der Rechnerarchitektur bezüglich der Programmierung und beinhaltet im Allgemeinen die nativen Datentypen, Befehle, Registerarchitektur, Adressiermodi, Speicherarchitektur, Interrupt- und Ausnahmehandhabung und externe Eingabe und Ausgabe (I/O) auf. Es sei angemerkt, dass sich der Begriff Befehl hierin auf einen Makrobefehl, z. B. einen Befehl, der dem Prozessor zur Ausführung bereitgestellt wird, oder auf einen Mikrobefehl, z. B. einen Befehl, der aus einem Prozessor-Decodierer resultiert, der Makrobefehle decodiert, beziehen kann.

Figurenliste

[0004] Die vorliegende Offenbarung ist beispielhaft und nicht einschränkend in den Figuren der beigefügten Zeichnungen veranschaulicht, in denen ähnliche Bezugszeichen ähnliche Elemente angeben und in denen zeigen:

Fig. 1 veranschaulicht eine Beschleuniger-Kachel gemäß Ausführungsformen der Offenbarung;

Fig. 2 veranschaulicht einen Hardware-Prozessor, der mit einem Speicher gekoppelt ist, gemäß Ausführungsformen der Offenbarung;

Fig. 3A veranschaulicht eine Programmquelle gemäß Ausführungsformen der Offenbarung;

Fig. 3B veranschaulicht einen Datenflussgraphen für die Programmquelle aus **Fig. 3A** gemäß Ausführungsformen der Offenbarung;

Fig. 3C veranschaulicht einen Beschleuniger mit mehreren Verarbeitungselementen, die zum Ausführen des Datenflussgraphen aus **Fig. 3B** gemäß Ausführungsformen der Offenbarung konfiguriert ist;

Fig. 4 veranschaulicht eine beispielhafte Ausführung des Datenflussgraphen gemäß Ausführungsformen der Offenbarung;

Fig. 5 veranschaulicht eine Programmquelle gemäß Ausführungsformen der Offenbarung;

Fig. 6 veranschaulicht eine Beschleuniger-Kachel, umfassend ein Array von Verarbeitungselementen gemäß Ausführungsformen der Offenbarung;

Fig. 7A veranschaulicht ein konfigurierbares Datenpfad-Netzwerk gemäß Ausführungsformen der Offenbarung;

Fig. 7B veranschaulicht ein konfigurierbares Flusssteuerpfad-Netzwerk gemäß Ausführungsformen der Offenbarung;

Fig. 8 veranschaulicht eine Hardware-Prozessor-Kachel, umfassend einen Beschleuniger gemäß Ausführungsformen der Offenbarung;

Fig. 9 veranschaulicht ein Verarbeitungselement gemäß Ausführungsformen der Offenbarung;

Fig. 10 veranschaulicht eine Abfrage-Adressdatei (RAF)-Schaltung gemäß Ausführungsformen der Offenbarung;

Fig. 11A veranschaulicht mehrere Abfrage-Adressdatei (RAF)-Schaltungen, die zwischen mehreren Beschleuniger-Kacheln und mehreren Cache-Bänken gemäß Ausführungsformen der Offenbarung gekoppelt sind;

Fig. 11B veranschaulicht einen Transaktionsmechanismus, in dem Cache-Zeilen mit Informationen über die Quelle eines Lese- oder Schreibzugriffs markiert sind, gemäß Ausführungsformen der Offenbarung.

Fig. 11C bis **Fig. 11J** veranschaulicht eine Unterstützung für Backup und Wiederholung unter Verwendung von Epochen im Cache-/Speicheruntersystem gemäß Ausführungsformen der Offenbarung.

Fig. 12 veranschaulicht einen Gleitkomma-Multiplizierer, der in drei Gebiete (Ergebnisgebiet, drei potentielle Übertraggebiete und Gebiet mit Gate) gemäß Ausführungsformen der Offenbarung unterteilt ist;

Fig. 13 veranschaulicht eine In-Flight-Beschleunigerkonfiguration mit mehreren Verarbeitungselementen gemäß Ausführungsformen der Offenbarung;

Fig. 14 veranschaulicht einen Speicherauszug einer zeitverschachtelten In-Flight-Extraktion gemäß Ausführungsformen der Offenbarung;

Fig. 15 veranschaulicht eine Kompilationstoolkette für einen Beschleuniger gemäß Ausführungsformen der Offenbarung;

Fig. 16 veranschaulicht einen Kompilierer für einen Beschleuniger gemäß Ausführungsformen der Offenbarung;

Fig. 17A veranschaulicht einen sequentiellen Assembliercode gemäß Ausführungsformen der Offenbarung;

Fig. 17B veranschaulicht einen Datenfluss-Assembliercode für den sequentiellen Assembliercode aus **Fig. 17A** gemäß Ausführungsformen der Offenbarung;

Fig. 17C veranschaulicht einen Datenflussgraph für den Datenfluss-Assembliercode aus **Fig. 17B** für einen Beschleuniger gemäß Ausführungsformen der Offenbarung;

Fig. 18A veranschaulicht einen C-Quellcode gemäß Ausführungsformen der Offenbarung;

Fig. 18B veranschaulicht einen Datenfluss-Assembliercode für den C-Quellcode aus **Fig. 18A** gemäß Ausführungsformen der Offenbarung;

Fig. 18C veranschaulicht einen Datenflussgraphen für den Datenfluss-Assembliercode aus **Fig. 18B** für einen Beschleuniger gemäß Ausführungsformen der Offenbarung;

Fig. 19A veranschaulicht einen C-Quellcode gemäß Ausführungsformen der Offenbarung;

Fig. 19B veranschaulicht einen Datenfluss-Assembliercode für den C-Quellcode aus **Fig. 19A** gemäß Ausführungsformen der Offenbarung;

Fig. 19C veranschaulicht einen Datenflussgraphen für den Datenfluss-Assembliercode aus **Fig. 19B** für einen Beschleuniger gemäß Ausführungsformen der Offenbarung;

Fig. 20A veranschaulicht ein Flussdiagramm gemäß Ausführungsformen der Offenbarung;

Fig. 20B veranschaulicht ein Flussdiagramm gemäß Ausführungsformen der Offenbarung;

Fig. 21 veranschaulicht einen Durchlaufsgraphen gegenüber der Energie pro Operation gemäß Ausführungsformen der Offenbarung;

Fig. 22 veranschaulicht eine Beschleuniger-Kachel, umfassend ein Array von Verarbeitungselementen und eine lokale Konfigurationssteuerung gemäß Ausführungsformen der Offenbarung;

Fig. 23A-23C veranschaulicht eine lokale Konfigurationssteuerung, die ein Datenpfad-Netzwerk konfiguriert, gemäß Ausführungsformen der Offenbarung;

Fig. 24 veranschaulicht eine Konfigurationssteuerung gemäß Ausführungsformen der Offenbarung;

Fig. 25 veranschaulicht eine Beschleuniger-Kachel, umfassend ein Array von Verarbeitungselementen, ein Konfigurations-Cache und eine lokale Konfigurationssteuerung gemäß Ausführungsformen der Offenbarung;

Fig. 26 veranschaulicht eine Beschleuniger-Kachel, umfassend ein Array von Verarbeitungselementen und eine Konfigurations- und Ausnahmehandhabungssteuerung mit einer Rekonfigurationsschaltung gemäß Ausführungsformen der Offenbarung;

Fig. 27 veranschaulicht eine Rekonfigurationsschaltung gemäß Ausführungsformen der Offenbarung;

Fig. 28 veranschaulicht eine Beschleuniger-Kachel, umfassend ein Array von Verarbeitungselementen und eine Konfigurations- und Ausnahmehandhabungssteuerung mit einer Rekonfigurationsschaltung gemäß Ausführungsformen der Offenbarung;

Fig. 29 veranschaulicht eine Beschleuniger-Kachel, umfassend ein Array von Verarbeitungselementen und einen Mezzanine-Ausnahmeaggregator, der mit einem Kachel-Level-Ausnahmeaggregator gemäß Ausführungsformen der Offenbarung gekoppelt ist;

Fig. 30 veranschaulicht ein Verarbeitungselement mit einem Ausnahmegenerator gemäß Ausführungsformen der Offenbarung;

Fig. 31 veranschaulicht eine Beschleuniger-Kachel, umfassend ein Array von Verarbeitungselementen und eine lokale Extraktionssteuerung gemäß Ausführungsformen der Offenbarung;

Fig. 32A-32C veranschaulicht eine lokale Extraktionssteuerung, die ein Datenpfad-Netzwerk konfiguriert, gemäß Ausführungsformen der Offenbarung;

Fig. 33 veranschaulicht eine Extraktionssteuerung gemäß Ausführungsformen der Offenbarung;

Fig. 34 veranschaulicht ein Flussdiagramm gemäß Ausführungsformen der Offenbarung;

Fig. 35 veranschaulicht ein Flussdiagramm gemäß Ausführungsformen der Offenbarung;

Fig. 36A ist ein Blockdiagramm, das ein allgemeines vektorfreundliches Befehlsformat und Klasse-A-Befehlstemplates gemäß Ausführungsformen der Offenbarung veranschaulicht;

Fig. 36B ist ein Blockdiagramm, welches das allgemeine vektorfreundliche Befehlsformat und Klasse-B-Befehlstemplates davon gemäß Ausführungsformen der Offenbarung veranschaulicht;

Fig. 37A ist ein Blockdiagramm, das Felder für die allgemeinen vektorfreundlichen Befehlsformate in **Fig. 36A** und **Fig. 36B** gemäß Ausführungsformen der Offenbarung veranschaulicht;

Fig. 37B ist ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Befehlsformats aus **Fig. 37A** veranschaulicht, das ein Full-Opcode-Feld gemäß einer Ausführungsform der Offenbarung bildet;

Fig. 37C ist ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Befehlsformats aus **Fig. 37A** veranschaulicht, das ein Registerindex-Feld gemäß einer Ausführungsform der Offenbarung bildet;

Fig. 37D ist ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Befehlsformats aus **Fig. 37A** veranschaulicht, das ein Augmentationsoperationsfeld 3650 gemäß einer Ausführungsform der Offenbarung bildet;

Fig. 38 ist ein Blockdiagramm einer Registerarchitektur gemäß einer Ausführungsform der Offenbarung;

Fig. 39A ist ein Blockdiagramm, das sowohl eine beispielhafte In-Reihenfolge-Pipeline als auch eine beispielhafte Außer-Reihenfolge-Ausgabe/Ausführungspipeline mit Registerumbenennung gemäß Ausführungsformen der Offenbarung veranschaulicht;

Fig. 39B ist ein Blockdiagramm, das sowohl ein Ausführungsbeispiel eines In-Reihenfolge-Architekturkerns als auch einen beispielhaften Außer-Reihenfolge-Ausgabe/Ausführungsarchitekturkem, der in einen Prozessor aufzunehmen ist, gemäß Ausführungsformen der Offenbarung veranschaulicht;

Fig. 40A ist ein Blockdiagramm eines einzelnen Prozessorkerns zusammen mit seiner Verbindung mit On-Die-Zwischenverbindungsnetzen und mit seinem lokalen Level 2 (L2) -Cacheuntersatz gemäß Ausführungsformen der Offenbarung;

Fig. 40B ist eine auseinander gezogene Ansicht des Teils des Prozessorkerns aus **Fig. 40A** gemäß Ausführungsformen der Offenbarung;

Fig. 41 ist ein Blockdiagramm eines Prozessors, der mehr als einen Kern aufweisen kann, eine integrierte Speichersteuerung aufweisen kann und der eine integrierte Grafik gemäß Ausführungsformen der Offenbarung aufweisen kann;

Fig. 42 ist ein Blockdiagramm eines Systems gemäß einer Ausführungsform der vorliegenden Offenbarung;

Fig. 43 ist ein Blockdiagramm eines spezifischeren beispielhaften Systems gemäß einer Ausführungsform der vorliegenden Offenbarung;

Fig. 44 zeigt ein Blockdiagramm eines zweiten spezifischeren beispielhaften Systems gemäß einer Ausführungsform der vorliegenden Offenbarung;

Fig. 45 zeigt ein Blockdiagramm eines SoC (System-on-Chip) gemäß einer Ausführungsform der vorliegenden Offenbarung;

Fig. 46 ist ein Blockdiagramm, das die Verwendung eines Softwarebefehlumwandlers zum Umwandeln von binären Befehlen in einem Quellbefehlssatz zu binären Befehlen in einem Zielbefehlssatz gemäß Ausführungsformen der Offenbarung kontrastiert.

AUSFÜHRLICHE BESCHREIBUNG

[0005] In der folgenden Beschreibung sind zahlreiche spezielle Details dargelegt. Es versteht sich jedoch, dass Ausführungsformen der Offenbarung ohne diese spezifischen Details in die Praxis umgesetzt werden können. In anderen Fällen wurden hinlänglich bekannte Schaltungen, Strukturen und Techniken nicht im Detail gezeigt, um das Verständnis dieser Beschreibung nicht zu verschleiern.

[0006] Bezugnahmen in der Beschreibung auf „eine Ausführungsform“, „Ausführungsform“, „ein Ausführungsbeispiel“ usw. geben an, dass die beschriebene Ausführungsform ein bestimmtes Merkmal, eine bestimmte Struktur oder ein bestimmtes Charakteristikum aufweisen kann, wobei allerdings nicht jede Ausführungsform dieses bestimmte Merkmal, diese bestimmte Struktur oder dieses bestimmte Charakteristikum notwendigerweise aufweist. Darüber hinaus beziehen sich derartige Formulierungen nicht notwendigerweise auf dieselbe Ausführungsform. Wenn ein bestimmtes Merkmal, eine bestimmte Struktur oder ein bestimmtes Charakteristikum in Verbindung mit einer Ausführungsform beschrieben wird, wird außerdem angenommen, dass es im Kenntnisbereich eines Fachmannes liegt, auf ein derartiges Merkmal, eine derartige Struktur oder ein derartiges Charakteristikum in Verbindung mit anderen Ausführungsformen einzuwirken, ob es nun explizit beschrieben wurde oder nicht.

[0007] Ein Prozessor (z. B. mit einem oder mehreren Kernen) kann Befehle ausführen (z. B. einen Thread von Befehlen), um Daten zu bearbeiten, um beispielsweise arithmetische, logische oder andere Funktionen durchzuführen. Zum Beispiel kann Software eine Operation anfordern, und ein Hardware-Prozessor (z. B. ein Kern oder Kerne davon) kann die Operation als Reaktion auf die Anfrage durchführen. Ein nicht einschränkendes Beispiel für eine Operation ist eine Mischoperation, um mehrere Vektorelemente einzugeben und einen Vektor mit mehreren Mischelementen auszugeben. In bestimmten Ausführungsformen wird eine Vielzahl von Operationen mit der Ausführung eines einzelnen Befehls erreicht.

[0008] Exascale-Leistung, z. B. wie vom Ministerium für Energie definiert, kann erfordern, dass die Gleitkomma-Punktleistung auf Systemebene 10^18 Gleitkomma-Operationen pro Sekunde (exaFLOPs) oder mehr innerhalb eines vorgegebenen (z. B. 20 MW) Leistungsbudgets übersteigt. Bestimmte Ausführungsformen hierin sind auf einen konfigurierbaren räumlichen Beschleuniger (CSA - Configurable Spatial Accelerator) gerichtet, der auf Hochleistungsrechnen (HPC - High Performance Computing) abzielt. Bestimmte Ausführungsformen eines CSA zielen auf die direkte Ausführung eines Datenflussgraphen ab, um eine rechenintensive, jedoch energieeffiziente räumliche Mikroarchitektur zu erhalten, die herkömmliche Roadmap-Architekturen weit übersteigt. Im Folgenden wird eine Beschreibung der Architekturphilosophie von Ausführungsformen eines CSA und bestimmter Merkmale davon gegeben. Wie bei jeder revolutionären Architektur kann die Programmierbarkeit ein Risiko darstellen. Zur Abschwächung dieses Problems wurden Ausführungsformen der CSA-Architektur zusammen mit einer Kompilier-Toolkette ausgestaltet, die ebenfalls unten erläutert wird.

EINLEITUNG

[0009] Exascale-Rechenziele können eine enorme Gleitkommaleistung auf Systemebene (z. B. 1 ExaFLOPs) innerhalb eines aggressiven Leistungsbudgets (z. B. 20 MW) erfordern. Die gleichzeitige Verbesserung der Leistung und Energieeffizienz der Programmausführung mit klassischen von Neumann-Architekturen ist jedoch schwierig geworden: Außer-Reihenfolge-Planung, simultanes Multithreading, komplexe Registerdateien und andere Strukturen stellen Leistung bereit, jedoch mit hohen Energiekosten. Bestimmte Ausführungsformen hierin erreichen gleichzeitig die Leistungs- und Energieanforderungen. Exascale-Rechenenergie-Leistungsziele können sowohl einen hohen Durchlauf als auch einen geringen Energieverbrauch pro Operation erfordern. Bestimmte Ausführungsformen hierin stellen dies bereit, indem sie eine große Anzahl von niederkomplexen, energieeffizienten Verarbeitungs- (z. B. Rechen-) Elementen bereitstellen, die den Steuerungsaufwand bisheriger Prozessorausgestaltungen weitgehend beseitigt. Geleitet von dieser Beobachtung weisen bestimmte Ausführungsformen hierin einen konfigurierbaren räumlichen Beschleuniger (CSA) auf, der z. B. ein Array aus

Verarbeitungselementen (PEs) umfasst, die durch einen Satz von leichtgewichtigen Gegendrucknetzwerken verbunden werden. Ein Beispiel für eine CSA-Kachel ist in **Fig. 1** dargestellt. Bestimmte Ausführungsformen von Verarbeitungselementen (z. B. Rechenelementen) sind Datenflussoperatoren, z. B. eine Vielzahl von Datenflussoperatoren, die nur Eingabedaten verarbeiten, wenn sowohl (i) die Eingabedaten beim Datenflussoperator eingegangen sind und (ii) Speicherplatz zum Speichern der Ausgabedaten verfügbar ist, weil z. B. anderenfalls keine Verarbeitung erfolgt. Bestimmte Ausführungsformen (z. B. eines Beschleunigers oder CSA) benutzen keinen getriggerten Befehl.

[0010] Fig. 1 veranschaulicht eine Beschleuniger-Kachel **100** gemäß Ausführungsformen der Offenbarung. Die Beschleuniger-Kachel **100** kann ein Abschnitt einer größeren Kachel sein. Die Beschleuniger-Kachel **100** führt einen oder mehrere Datenflussgraphen aus. Ein Datenflussgraph kann sich im Allgemeinen auf eine explizit parallele Programmbeschreibung beziehen, die bei der Kompilierung von sequentiellen Codes entsteht. Bestimmte Ausführungsformen hierin (z. B. CSAs) ermöglichen, dass Datenflussgraphen direkt auf dem CSA-Array konfiguriert werden, z. B. anstatt in sequentielle Befehlsströme umgewandelt zu werden. Die Ableitung eines Datenflussgraphen aus einem sequentiellen Kompilierungsfluss ermöglicht es Ausführungsformen eines CSA, bekannte Programmiermodelle zu unterstützen und (z. B. ohne Verwendung einer Arbeitstabelle) einen existierenden Hochleistungsrechencode (HPC) direkt auszuführen. Die CSA-Verarbeitungselemente (PE) können energieeffizient sein. In **Fig. 1** kann die Speicherschnittstelle **102** mit einem Speicher (z. B. Speicher **202** in **Fig. 2**) gekoppelt sein, um es der Beschleuniger-Kachel **100** zu ermöglichen, auf Daten des Speichers (z. B. Off-Die) zuzugreifen (z. B. diese zu laden und/oder zu speichern). Die dargestellte Beschleuniger-Kachel **100** ist ein heterogenes Array, das aus verschiedenen Arten von PEs besteht, die über ein Zwischenverbindungsnetz **104** miteinander gekoppelt sind. Die Beschleuniger-Kachel **100** kann eine oder mehr ganzzahlige arithmetische PEs, arithmetische Gleitkomma-PEs, Kommunikationsschaltkreise und in-Struktur-Speicher aufweisen. Die Datenflussgraphen (z. B. kompilierte Datenflussgraphen) können die Beschleuniger-Kachel **100** für die Ausführung überlagern. In einer Ausführungsform handhabt für einen bestimmten Datenflussgraphen jede PE nur eine oder zwei Operationen des Graphen. Das PE-Array kann heterogen sein, z. B. derart, dass kein PE die volle CSA-Datenflussarchitektur unterstützt und/oder eine oder mehrere PEs programmiert sind (z. B. benutzerdefiniert), um nur einige wenige, aber hocheffiziente Operationen durchzuführen. Bestimmte Ausführungsformen hierin ergeben somit einen Beschleuniger mit einem Array aus Verarbeitungselementen, das im Vergleich zu Roadmap-Architekturen rechenintensiv ist und dennoch eine Erhöhung der Energieeffizienz und Leistung in Bezug auf bestehende HPC-Angebote um etwa eine Größenordnung erreicht.

[0011] Die Leistungsanstiege können aus der parallelen Ausführung innerhalb des (z. B. dichten) CSA, wobei jede PE gleichzeitig ausführen kann, z. B. wenn Eingabedaten verfügbar sind. Die Effizienzanstiege können aus der Effizienz jedes PE resultieren, z. B. wenn jede PE-Operation (z. B. Verhalten) einmal pro Konfigurationsschritt (z. B. Mapping) fixiert wird und die Ausführung beim lokalen Eingehen von Daten am PE erfolgt, z. B. ohne Berücksichtigen einer anderen Strukturaktivität. In bestimmten Ausführungsformen ist ein PE ein Datenflussoperator (z. B. jeweils ein einzelner), z. B. ein Datenflussoperator, der nur Eingabedaten verarbeitet, wenn sowohl (i) die Eingabedaten beim Datenflussoperator eingegangen sind als auch (ii) Speicherplatz zum Speichern der Ausgabedaten verfügbar ist, weil z. B. anderenfalls keine Verarbeitung erfolgt. Diese Eigenschaften ermöglichen Ausführungsformen des CSA, paradigmaverschobene Leistungsniveaus und enorme Verbesserungen in der Energieeffizienz über eine breite Klasse bestehender Einzelstrom- und Parallelprogramme bereitzustellen, z. B. aller bei gleichzeitiger Beibehaltung vertrauter HPC-Programmiermodelle. Bestimmte Ausführungsformen eines CSA können auf HPC abzielen, sodass die Gleitkomma-Energieeffizienz extrem wichtig wird. Bestimmte Ausführungsformen des CSA liefern nicht nur überzeugende Leistungsverbesserungen und Energieeinsparungen, sie liefern diese Vorteile auch für bestehende HPC-Programme, die in Mainstream-HPC-Sprachen und für Mainstream-HPC-Frameworks geschrieben sind. Bestimmte Ausführungsformen der CSA-Architektur (z. B. unter Berücksichtigung der Kompilierung) stellen mehrere Erweiterungen bei der direkten Unterstützung der von modernen Kompilierern erzeugten internen Steuerdatenflussrepräsentationen bereit. Bestimmte Ausführungsformen hierin sind direkt an einen CSA-Datenfluss-Kompilierer gerichtet, der z. B. C-, C++- und Fortran-Programmiersprachen akzeptieren kann, um auf eine CSA-Architektur abzuzielen.

[0012] Abschnitt 2 unten offenbart Ausführungsformen der CSA-Architektur. Insbesondere sind neuartige Ausführungsformen der Integration von Speicher innerhalb des Datenfluss-Ausführungsmodells offenbart. Abschnitt 3 taucht in die mikroarchitektonischen Details der Ausführungsformen eines CSA ein. In einer Ausführungsform ist das Hauptziel eines CSA die Unterstützung von vom Kompilierer erzeugten Programmen. Abschnitt 4 unten untersucht die Ausführungsformen einer CSA-Kompilierungstoolkette. Die Vorteile der Ausführungsformen eines CSA werden mit anderen Architekturen bei der Ausführung von kompilierten Codes in Abschnitt 5 verglichen. Schließlich wird die Leistung der Ausführungsformen einer CSA-Mikroarchitektur in

Abschnitt 6 erläutert, weitere CSA-Details werden in Abschnitt 7 erläutert und eine Zusammenfassung in Abschnitt 8 bereitgestellt.

ARCHITEKTUR

[0013] Das Ziel bestimmter Ausführungsformen eines CSA ist das schnelle und effiziente Ausführen von Programmen, z. B. Programmen, die von Kompilierern erzeugt werden. Bestimmte Ausführungsformen der CSA-Architektur stellen Programmierabstraktionen bereit, die den Bedarf an Kompiliertechnologien und Programmierparadigmen unterstützen. Ausführungsformen des CSA führen Datenflussgraphen aus, z. B. eine Programmmanifestation, welche die kompiliereigene interne Repräsentation (IR) von kompilierten Programmen eng imitiert. In diesem Modell wird ein Programm als ein Datenflussgraph dargestellt, der aus Knoten (z. B. Scheitelpunkten), die aus einem Satz von architektonisch definierten Datenflussoperatoren (die z. B. sowohl Rechen- als auch Steueroperationen umschließen) und Rändern besteht, welche die Übertragung von Daten zwischen den Datenflussoperatoren repräsentieren. Die Ausführung kann durch Einfügen von Datenfluss-Token (die z. B. Datenwerte sind oder repräsentieren) in den Datenflussgraph fortschreiten. Die Token können zwischen jedem Knoten (z. B. Scheitelpunkt) fließen und umgewandelt werden und z. B. eine vollständige Berechnung bilden. Ein Probendatenflussgraph und seine Ableitung aus einem High-Level-Quellcode ist in **Fig. 3A-3C** gezeigt, und **Fig. 5** zeigt ein Beispiel der Ausführung eines Datenflussgraphen.

[0014] Ausführungsformen der CSA sind für die Datenflussgrahpausführung durch Bereitstellen exakt solcher Datenfluss-Graph-Ausführungsunterstützungen konfiguriert, die durch die Kompilierer erfordert werden. In einer Ausführungsform ist der CSA ein Beschleuniger (z. B. ein Beschleuniger aus **Fig. 2**), der nicht versucht, einige der notwendigen aber selten verwendeten Mechanismen zu suchen, die auf Allzweck-Verarbeitungskernen verfügbar sind (z. B. einem Kern aus **Fig. 2**), wie z. B. Systemanrufe. Daher kann der CSA in dieser Ausführungsform viele Codes ausführen, aber nicht alle Codes. Im Gegenzug erzielt der CSA signifikante Leistungs- und Energievorteile. Zum Aktivieren der Beschleunigung von Code, der in herkömmlich verwendeten sequentiellen Sprachen geschrieben ist, führen die Ausführungsformen hierin auch verschiedene neuartige Architekturmerkmale zum Unterstützen des Kompilierers ein. Eine besondere Neuheit ist die CSA-Speicherbehandlung, ein Gegenstand, der zuvor ignoriert oder nur dürftig angegangen wurde. Ausführungsformen des CSA sind auch eindeutig bei der Verwendung von Datenflussoperatoren, z. B. im Gegensatz zu Nachschlagetabellen (LUT), als ihre fundamentale Architekturschnittstelle.

[0015] **Fig. 2** veranschaulicht einen Hardware-Prozessor **200**, der mit einem Speicher **202** gemäß Ausführungsformen der Offenbarung gekoppelt (z. B. damit verbunden) ist. In einer Ausführungsform sind der Hardware-Prozessor **200** und der Speicher **202** ein Rechnersystem **201**. In bestimmen Ausführungsformen sind einer oder mehrere der Beschleuniger ein CSA gemäß dieser Offenbarung. In bestimmten Ausführungsformen sind einer oder mehrere der Kerne in einem Prozessor die hierin offenbarten Kerne. Der Hardware-Prozessor **200** (z. B. jeder Kern davon) kann einen Hardware-Decodierer (z. B. Decodiereinheit) und eine Hardware-Ausführungseinheit aufweisen. Der Hardware-Prozessor **200** kann Register aufweisen. Es sei angemerkt, dass die Figuren hierin ggf. nicht alle Datenkommunikationskopplungen (z. B. Verbindungen) darstellen. Ein Durchschnittsfachmann wird zu schätzen wissen, dass dies gewisse Details in den Figuren nicht verschleiert. Es sei darauf hingewiesen, dass ein Doppelpfeil in den Figuren keine Zweiwegekommunikation erfordert, z. B. kann er eine Einwegekommunikation (z. B. zu oder von dieser Komponente oder Vorrichtung) angeben. Jede oder alle Kombinationen von Kommunikationspfaden können in bestimmten Ausführungsformen hierin verwendet werden. Der dargestellte Hardware-Prozessor **200** weist mehrere Kerne (O bis N, wobei N 1 oder mehr sein kann) und Hardware-Beschleuniger (O bis M, wobei M 1 oder mehr sein kann) gemäß Ausführungsformen der Offenbarung auf. Der Hardware-Prozessor **200** (z. B. der/die Beschleuniger und/oder Kern(e) davon) können mit dem Speicher **202** (z. B. Datenspeichervorrichtung) gekoppelt sein. Der Hardware-Decodierer (z. B. des Kerns) kann einen (z. B. einzelnen) Befehl (z. B. Makrobefehl) empfangen und den Befehl decodieren, z. B. in Mikrobefehle und/oder Mikrooperationen. Die Hardware-Ausführungseinheit (z. B. des Kerns) kann den decodierten Befehl (z. B. Makrobefehl) zum Durchführen einer Operation oder Operationen ausführen. Mit erneuter Bezugnahme auf die Ausführungsformen des CSA werden als nächstes die Datenflussoperatoren erläutert.

Datenflussoperatoren

[0016] Die wichtigste Architekturschnittstelle der Ausführungsformen des Beschleunigers (z. B. CSA) ist der Datenflussoperator, z. B. als eine direkte Repräsentation eines Knotens in einem Datenflussgraphen. Aus einer betrieblichen Perspektive verhalten sich die Datenflussoperatoren in einer Streaming- oder datenangesteuerten Weise. Die Datenflussoperatoren können ausführen, sobald ihre eingehenden Operanden verfügbar werden. Die CSA-Datenflussausführung kann (z. B. nur) von einem stark lokalisierten Status abhängig

sein, der z. B. in einer hoch skalierbaren Architektur mit einem verteilten, asynchronen Ausführungsmodell resultiert. Die Datenflussoperatoren können arithmetische Datenflussoperatoren aufweisen, z. B. eine oder mehrere von Gleitkomma-Addition und - Multiplikation, Integer-Addition, Subtraktion und Multiplikation, verschiedene Vergleichsformen, logische Operatoren und Verschiebung. Die Ausführungsformen des CSA können auch einen reichen Satz an Steueroperatoren aufweisen, welche die Verwaltung der Datenfluss-Token in dem Programmgraphen stützen. Beispiele davon weisen einen „Pick-“ Operator auf, der z. B. zwei oder mehr logische Eingabekanäle zu einem einzelnen Ausgabekanal multiplext, sowie einen „Schalt-“ Operator, der z. B. als ein Kanal-Demultiplexer arbeitet (der z. B. einen einzelnen Kanal aus zwei oder mehreren logischen Eingabekanälen ausgibt). Diese Operatoren können einem Kompilierer ermöglichen, Steuerparadigmen wie bedingte Ausdrücke zu implementieren. Bestimmte Ausführungsformen eines CSA können einen begrenzten Datenflussoperatorsatz (z. B. für eine relativ kleine Anzahl an Operationen) aufweisen, um dichte und energieeffiziente PE-Mikroarchitekturen zu ergeben. Bestimmte Ausführungsformen können Datenflussoperatoren für komplexe Operationen aufweisen, die in HPC-Code gewöhnlich sind. Die CSA-Datenflussoperator-Architektur ist für einsatzspezifische Erweiterungen stark anpassungsfähig. Zum Beispiel können komplexere mathematische Datenflussoperatoren, z. B. trigonometrische Funktionen, in bestimmten Ausführungsformen zum Beschleunigen bestimmter mathematikintensiver HPC-Arbeitslasten einschließen. Auf ähnliche Weise kann eine neuralnetzwerkabgestimmte Erweiterung Datenflussoperatoren für eine vektorisierte niederpräzise Arithmetik einschließen.

[0017] **Fig. 3A** veranschaulicht eine Programmquelle gemäß Ausführungsformen der Offenbarung. Der Programmquellcode weist eine Multiplikationsfunktion (func) auf. **Fig. 3B** veranschaulicht ein Datenflussschaubild **300** für die Programmquelle aus **Fig. 3A** gemäß Ausführungsformen der Offenbarung. Der Datenflussgraph **300** weist einen Pick-Knoten **304**, Switch-Knoten **306** und Multiplikationsknoten **308** auf. Ein Puffer kann wahlweise entlang eines oder mehrerer Kommunikationspfade aufgenommen sein. Der dargestellte Datenflussgraph **300** kann eine Operation der Auswahl der Eingabe X mit Pick-Knoten **304** durchführen, X mit Y multiplizieren (z. B. Multiplikationsknoten **308**) und dann das Ergebnis von der übrigen Ausgabe des Switch-Knotens **306** ausgeben. **Fig. 3C** veranschaulicht einen Beschleuniger (z. B. CSA) mit mehreren Verarbeitungselementen **301**, der zum Ausführen des Datenflussgraphen aus **Fig. 3B** gemäß Ausführungsformen der Offenbarung **301** konfiguriert ist; Insbesondere ist der Datenflussgraph **300** in das Array der Verarbeitungselemente **301** überlagert (z. B. und die (z. B. Zwischenverbindungs-)Netzwerk(e) dazwischen), z. B. sodass jeder Knoten des Datenflussgraphen **300** als ein Datenflussoperator in dem Array aus Verarbeitungselementen **301** repräsentiert ist. In einer Ausführungsform dienen eines oder mehrere der Verarbeitungselemente in dem Array von Verarbeitungselementen **301** zum Zugriff auf den Speicher durch die Speicherschnittstelle **302**). In einer Ausführungsform entspricht der Pick-Knoten **304** des Datenflussgraphen **300** somit dem Pick-Operator **304A** (wird z. B. davon repräsentiert), der Switch-Knoten **306** des Datenflussgraphen **300** entspricht also dem Schalt-Operator **306A** (wird z. B. davon repräsentiert) und der Multiplizierer-Knoten **308** des Datenflussgraphen **300** entspricht also dem Multiplizierer-Operator **308A** (wird z. B. dadurch repräsentiert). Ein weiteres Verarbeitungselement und/oder ein Flusssteuerpfad-Netzwerk können die Steuersignale (z. B. Steuer-Token) an den Pick-Operator **304A** und den Schalt-Operator **306A** bereitstellen, um die Operation in **Fig. 3A** durchzuführen. In einer Ausführungsform ist das Array von Verarbeitungselementen **301** zum Ausführen des Datenflussgraphen **300** aus **Fig. 3B** vor Start der Ausführung konfiguriert. In einer Ausführungsform führt der Kompilierer die Umwandlung von **Fig. 3A** bis **Fig. 3B** durch. In einer Ausführungsform bettet die Eingabe der Datenflussgraph-Knoten in das Array aus Verarbeitungselementen den Datenflussgraph logisch in das Array aus Verarbeitungselementen ein, z. B. wie weiter unten besprochen, sodass der Eingabe-/Ausgabepfad zum Erzeugen des gewünschten Ergebnisses konfiguriert ist.

Latenzinsensitive Kanäle

[0018] Kommunikationsbögen sind die zweite Hauptkomponente des Datenflussgraphen. Bestimmte Ausführungsformen eines CSA beschreiben diese Bögen als latenzinsensitive Kanäle, z. B. in-Reihenfolge, Gegendruck- (die z. B. keine Ausgabe erzeugen oder senden, bis ein Platz zum Speichern der Ausgabe vorhanden ist), Point-to-Point-Kommunikationskanäle. Wie bei den Datenflussoperatoren sind die latenzinsensitiven Kanäle fundamental asynchron und geben die Freiheit, viele Typen von Netzwerken zum Implementieren der Kanäle eines bestimmten Graphen zusammenzustellen. Latenzinsensitive Kanäle können willkürlich lange Latenzen aufweisen und die CSA-Architektur weiterhin gewissenhaft implementieren. In bestimmten Ausführungsformen ist es jedoch ein großer Anreiz bezüglich der Leistung und Energie, die Latenzen so klein wie möglich zu machen. Abschnitt 3.2 hierin offenbart eine Netzwerk-Mikroarchitektur, in der die Datenflussgraph-Kanäle zeitverschachtelt mit nicht mehr als einem Latenzzyklus implementiert sind. Ausführungsformen von latenzinsensitiven Kanälen stellen eine kritische Abstraktionsschicht bereit, die mit der CSA-Architektur zum Bereitstellen einer Anzahl von Laufzeitdiensten an den Anwendungsprogrammierer genutzt werden können.

Ein CSA kann beispielsweise die latenzinsensitiven Kanäle bei der Implementierung der CSA-Konfiguration (dem Laden eines Programms auf ein CSA-Array) nutzen.

[0019] **Fig. 4** veranschaulicht eine beispielhafte Ausführung des Datenflussgraphen **400** gemäß Ausführungsformen der Offenbarung. Bei Schritt 1 können Eingabewerte (z. B. 1 für X in **Fig. 3B** und **Fig. 2** für Y in **Fig. 3B)** in den Datenflussgraph **400** geladen werden, um eine 1 x 2-Muliplikationsoperation durchzuführen. Einer oder mehrere Dateneingabewerte können in der Operation (z. B. 1 für X und 2 für Y in Bezug auf **Fig. 3B)** statisch (z. B. konstant) sein oder während der Operation aktualisiert werden. Bei Schritt 2 geben ein Verarbeitungselement (z. B. auf einem Flusssteuerpfad-Netzwerk) oder andere Schaltungsausgaben eine Null in die Steuereingabe (z. B. Mux-Steuersignal) von Pick-Knoten **404** (z. B. um eine eins von Port „0“ zu seiner Ausgabe zu beschaffen) und gibt eine Null zur Steuereingabe (z. B. Mux-Steuersignal) von Switch-Knoten **406** aus (z. B. zum Bereitstellen seiner Eingabe aus Port „0“ zu einem Ziel (z. B. einem nachgeschalteten Verarbeitungselement). Bei Schritt 3 wird der Datenwert 1 vom Pick-Knoten **404** (und z. B. sein Steuersignal „0“ am Pick-Knoten **404** verbraucht) an den Multiplizierer-Knoten **408** ausgegeben, um mit dem Datenwert von 2 bei Schritt 4 multipliziert zu werden. Bei Schritt 4 erreicht die Ausgabe des Multiplizierer-Knotens **408** den Switch-Knoten **406**, was z. B. den Switch-Knoten **406** veranlasst, ein Steuersignal „0“ zu verbrauchen, um den Wert von 2 von Port „2“ von Switch-Knoten **406** bei Schritt 5 auszugeben. Die Operation ist dann abgeschlossen. Ein CSA kann daher dementsprechend programmiert werden, damit ein entsprechender Datenflussoperator für jeden Knoten die Operation in **Fig. 4** durchführt. Obwohl die Ausführung in diesem Beispiel serialisiert ist, können im Prinzip alle Datenfluss-Operationen parallel ausgeführt werden. Die Schritte werden in **Fig. 4** verwendet, um die Datenflussausführung von jeder physischen mikroarchitektonischen Manifestation zu differenzieren. In einer Ausführungsform hat ein nachgeschaltetes Verarbeitungselement ein Signal an den Schalter **406** zu senden (oder kein Bereit-Signal zu senden) (z. B. auf einem Flusssteuerpfad-Netzwerk), um die Ausgabe von Schalter **406** aufzuhalten, z. B. bis das nachgeschaltete Verarbeitungselement für die Ausgabe bereit ist (z. B. Speicherplatz aufweist).

Speicher

[0020] Datenfluss-Architekturen konzentrieren sich im Allgemeinen auf die Kommunikation und Datenmanipulation und beachten dem Status weniger Beachtung. Das Aktivieren von echter Software, insbesondere Programmen, die in sequentiellen veralteten Sprachen geschrieben sind, erfordert eine bedeutende Beachtung der Schnittstelle mit dem Speicher. Bestimmte Ausführungsformen eines CSA verwenden Architekturspeicheroperationen als ihre primäre Schnittstelle zur (z. B. großen) statusbehafteten Speicherung. Aus der Perspektive des Datenflussgraphen ähneln die Speicheroperationen anderen Datenfluss-Operationen, mit der Ausnahme, dass sie den Nebeneffekt der Aktualisierung eines gemeinsam genutzten Speichers aufweisen. Insbesondere haben Speicheroperationen von bestimmten Ausführungsformen hierin die gleiche Semantik wie jeder andere Datenflussoperator, z. B. werden „ausgeführt“, wenn ihre Operanden, z. B. eine Adresse, verfügbar ist und nach einiger Latenz eine Antwort erzeugt wird. Bestimmte Ausführungsformen hierin entkoppeln die Operandeneingabe explizit und resultieren in einer Ausgabe, sodass die Speicheroperatoren natürlich zeitverschachtelt sind und das Potenzial aufweisen, viele simultan ausstehende Anforderungen zu erzeugen, welche diese z. B. ausgezeichnet für die Latenz- und Bandbreitencharakteristika eines Speicheruntersystems geeignet machen. Ausführungsformen eines CSA stellen grundlegende Speicheroperationen wie Last bereit, die einen Adressenkanal nimmt und einen Antwortkanal mit den Werten, die den Adressen entsprechen, und einen Speicher füllt. Ausführungsformen eines CSA können auch erweiterte Operationen wie speicherinterne Atomik- und Konsistenz-Operatoren bereitstellen. Diese Operationen können eine ähnliche Semantik wie ihre von-Neumann-Gegenstücke aufweisen. Ausführungsformen eines CSA können vorhandene Programme beschleunigen, die unter Verwendung sequentieller Sprachen wie C- und Fortran beschrieben werden. Eine Folge der Unterstützung dieser Sprachmodelle ist das Adressieren der Programmspeicherreihenfolge, z. B. der seriellen Anordnung von Speicheroperationen, die typischerweise durch diese Sprachen vorgeschrieben sind.

[0021] **Fig. 5** veranschaulicht eine Programmquelle (z. B. C-Code) 500 gemäß Ausführungsformen der Offenbarung. Gemäß der Speichersemantik der C-Programmiersprache ist die Speicherkopie (memcpy) zu serialisieren. Memcpy kann jedoch mit einer Ausführungsform des CSA parallelisiert werden, wenn die Arrays A und B bekanntermaßen unverbunden sind. **Fig. 5** veranschaulicht ferner das Problem der Programmreihenfolge. Allgemein können Kompilierer nicht nachweisen, dass Array A anders als Array B ist, z. B. entweder für den gleichen Wert des Index oder einen anderen Wert des Index über den Schleifenkörpern. Dies ist als Zeiger- oder Speicher-Aliasing bekannt. Da Kompilierer einen statisch korrekten Code erzeugen müssen, werden sie gewöhnlich zur Serialisierung der Speicherzugriffe gezwungen. Typischerweise verwenden Kompilierer, die auf sequentielle von-Neumann-Architekturen abzielen, die Befehlsreihenfolge als natürliches Mittel zum Durchsetzen der Programmreihenfolge. Ausführungsformen des CSA besitzen jedoch keine Vorstellung

von Befehlen oder befehlsbasierter Programmreihenfolge, wie durch einen Programmzähler definiert. In bestimmten Ausführungsformen sind eingehende Abhängigkeits-Token, die z. B. keine architektonisch sichtbare Information enthalten, wie alle anderen Datenfluss-Token und die Speicheroperationen können nicht ausgeführt werden, bis sie einen Abhängigkeits-Token empfangen haben. In bestimmten Ausführungsformen erzeugen die Speicheroperationen einen ausgehenden Abhängigkeits-Token, sobald ihre Operation für alle logisch nachfolgenden abhängigen Speicheroperationen sichtbar ist. In bestimmten Ausführungsformen gleichen die Abhängigkeits-Token anderen Datenfluss-Token in einem Datenflussgraph. Da Speicheroperationen z. B. in bedingten Kontexten auftreten, können die Abhängigkeits-Token auch unter Verwendung von Steueroperatoren manipuliert werden, wie in Abschnitt 2.1 beschrieben, z. B. wie alle anderen Token. Abhängigkeits-Token können den Effekt der Serialisierung von Speicherzugriffen haben, z. B. dem Kompilierer eine Einrichtung zum architektonischen Definieren der Reihenfolge von Speicherzugriffen bereitstellen.

Laufzeitdienste

[0022] Ein Hauptpunkt für die Berücksichtigung der Architektur von Ausführungsformen des CSA beinhaltet die tatsächliche Ausführung von Programmen auf Benutzerebene, es kann aber auch wünschenswert sein, verschiedene Mechanismen zu unterstützen, die diese Ausführung stärken. Zu den wichtigsten zählen Konfiguration (bei der ein Datenflussgraph in den CSA geladen wird), Extraktion (bei welcher der Status eines ausführenden Graphen in den Speicher verschoben wird) und Ausnahmen (in denen mathematische, weiche und andere Arten von Fehlern in der Struktur erkannt und behandelt werden, möglicherweise durch eine externe Entität). Abschnitt 3.6 unten erläutert die Eigenschaften einer latenzinsensitiven Datenflussarchitektur einer Ausführungsform eines CSA, um effiziente, stark zeitverschachtelte Implementierungen dieser Funktionen zu ergeben. Vom Konzept her kann die Konfiguration den Status eines Datenflussgraphen in die Zwischenverbindungs- und Verarbeitungselemente (z. B. Struktur) laden, z. B. im Allgemeinen von dem Speicher. Während dieses Schrittes können alle Strukturen im CSA mit einem neuen Datenflussgraphen geladen werden und alle Datenfluss-Token in diesem Graphen, z. B. als Folge einer Kontextumschaltung, leben. Die latenzinsensitive Semantik eines CSA kann eine verteilte, asynchrone Initialisierung der Struktur ermöglichen, z. B. kann sie die Ausführung unmittelbar beginnen, sobald die PEs konfiguriert sind. Unkonfigurierte PEs können ihre Kanäle gegendrücken, bis sie konfiguriert werden, z. B. Kommunikationen zwischen konfigurierten und unkonfigurierten Elementen verhindern. Die CSA-Konfiguration kann in einen priviligierten Status und Status auf Benutzerebene partitioniert werden. Eine solche Zwei-Level-Partitionierung kann der primären Konfiguration der Struktur ermöglichen, ohne Aufrufen des Betriebssystems zu erfolgen. Während einer Extraktionsausführungsform wird eine logische Sicht des Datenflussgraphen erfasst und in den Speicher festgeschrieben, z. B. durch Aufnehmen aller Live-Steuer- und Datenfluss-Token und Zustand in dem Graphen.

[0023] Die Extraktion kann auch eine Rolle bei der Bereitstellung von Zuverlässigkeitsgarantien durch die Schaffung von Strukturprüfpunkten spielen. Ausnahmen in einem CSA können allgemein durch die gleichen Ereignisse verursacht werden, die Ausnahmen in Prozessoren bewirken, wie z. B. illegale Operatorargumente oder RAS-Ereignisse (RAS - Reliability (Zuverlässigkeit), Availability (Verfügbarkeit) und Serviceability (Betriebsfähigkeit). In bestimmten Ausführungsformen werden Ausnahmen auf dem Level der Datenflussoperatoren erfasst, wie z. B. durch Prüfen von Argumentwerten oder durch modulare arithmetische Schemata. Nach dem Detektieren einer Ausnahme kann ein Datenflussoperator (z. B. eine Schaltung) halten und eine Ausnahmenachricht emittieren, die z. B. sowohl einen Operationsidentifikator als auch einige Details über die Natur des Problems, das aufgetreten ist, enthält. In einer Ausführungsform bleibt der Datenflussoperator angehalten, bis er rekonfiguriert wurde. Die Ausnahmenachricht kann dann einem zugeordneten Prozessor (z. B. Kern) zur Wartung kommuniziert werden, die z. B. das Extrahieren des Graphen zur Softwareanalyse einschließen kann.

Kachel-Level-Architektur

[0024] Ausführungsformen der CSA-Computerarchitekturen (z. B. auf HPC und Datencenter abzielende Verwendungen) sind nebeneinander angeordnet. **Fig. 6** und **Fig. 8** zeigen den Kachel-Level-Einsatz eines CSA. **Fig. 8** zeigt eine Voll-Kachel-Implementierung eines CSA, die z. B. ein Beschleuniger eines Prozessors mit einem Kern sein kann. Ein Hauptvorteil dieser Architektur kann reduziertes Ausgestaltungsrisiko sein, sodass der CSA und der Kern z. B. bei der Herstellung vollständig entkoppelt sind. Zusätzlich zu der Ermöglichung einer besseren Komponentenwiederverwendung kann dies der Ausgestaltung der Komponenten wie dem CSA-Cache ermöglichen, nur den CSA zu berücksichtigen, statt z. B. die strikteren Latenzanforderungen des Kerns zu integrieren. Schließlich können separate Kacheln die Integration des CSA mit kleinen oder großen Kernen ermöglichen. Eine Ausführungsform des CSA erfasst die meisten vektorparellelen Arbeitslasten, sodass die meisten vektorartigen Arbeitslasten direkt auf dem CSA laufen, aber in bestimmten Ausführungsformen können die vektorartigen Befehle in dem Kern eingeschlossen sein, z. B. zum Unterstützen von veralteten Binaritäten.

MIKROARCHITEKTUR

[0025] In einer Ausführungsform ist das Ziel der CSA-Mikroarchitektur das Bereitstellen einer hochqualitativen Implementierung jedes Datenflussoperators, der durch die CSA-Architektur spezifiziert wird. Ausführungsformen der CSA-Mikroarchitektur sehen vor, dass jedes Verarbeitungselement der Mikroarchitektur ungefähr einem Knoten (z. B. einer Entität) in dem Architekturdatenflussgraphen entspricht. In bestimmten Ausführungsformen führt dies zu Mikroarchitekturelementen, die nicht nur kompakt sind, was zu einem dichten Berechnungsarray führt, sondern auch energieeffizient, zum Beispiel, wenn Verarbeitungselemente (PEs) sowohl einfach als auch stark ungemultiplext sind, z. B. Ausführen eines einzelnen Datenflussoperators für eine Konfiguration (z. B. Programmierung) des CSA ausführen. Um den Energie- und Implementierungsbereich weiter zu reduzieren, kann ein CSA einen konfigurierbaren, heterogenen Strukturstil aufweisen, in dem jedes PE davon nur einen Untersatz von Datenflussoperatoren implementiert. Periphere und unterstützende Teilsysteme, wie z. B. der CSA-Cache, können bereitgestellt werden, um die verteilte Parallelität zu unterstützen, die in der Haupt-CSA-Verarbeitungsstruktur selbst vorherrscht. Die Implementierung von CSA-Mikroarchitekturen kann Datenfluss- und latenzinsensitive Kommunikationsabstraktionen verwenden, die in der Architektur vorhanden sind. In bestimmten Ausführungsformen gibt es (z. B. im Wesentlichen) eine Eins-zu-Eins-Entsprechung zwischen Knoten in dem kompiliererzeugten Graphen und den Datenflussoperatoren (z. B. Datenflussoperator-Rechenelementen) in einem CSA.

[0026] Es folgt eine Erläuterung eines beispielhaften CSA, gefolgt von einer detaillierteren Erläuterung der Mikroarchitektur. Bestimmte Ausführungsformen hierin stellen einen CSA bereit, der eine einfache Kompilierung ermöglicht, z. B. im Gegensatz zu bestehenden FPGA-Kompilierern, die einen kleinen Untersatz einer Programmiersprache (z. B. C oder C++) handhaben und viele Stunden benötigen, um selbst kleine Programme zu kompilieren.

[0027] Bestimmte Ausführungsformen einer CSA-Architektur erlauben heterogene grobkörnige Operationen, wie Gleitkomma mit doppelter Genauigkeit. Programme können in weniger grobkörnigen Operationen ausgedrückt werden, z. B. so, dass der offenbarte Kompilierer schneller läuft als herkömmliche räumliche Kompilierer. Bestimmte Ausführungsformen beinhalten eine Struktur mit neuen Verarbeitungselementen, um sequentielle Konzepte wie programmgeordnete Speicherzugriffe zu unterstützen. Bestimmte Ausführungsformen implementieren Hardware, um grobkörnige datenflussartige Kommunikationskanäle zu unterstützen. Dieses Kommunikationsmodell ist abstrakt und kommt der vom Kompilierer verwendeten Steuerdatenflussrepräsentation sehr nahe. Bestimmte Ausführungsformen hierin beinhalten eine Netzwerkimplementierung, die Einzelzyklus-Latenzzeitkommunikationen unterstützt, z. B. Benutzen (z. B. kleiner) PEs, die einzelne Steuerdatenflussoperationen unterstützen. In bestimmten Ausführungsformen verbessert dies nicht nur die Energieeffizienz und Leistung, sondern vereinfacht auch die Kompilierung, da der Kompilierer eine Eins-zu-eins-Abbildung zwischen High-Level-Datenflusskonstrukten und der Struktur vornimmt. Bestimmte Ausführungsformen hierin vereinfachen somit die Aufgabe des Kompilierens existierender (z. B. C-, C++- oder Fortran-) Programme zu einem CSA (z. B. Struktur).

[0028] Die Energieeffizienz kann ein Hauptanliegen moderner Rechensysteme sein. Bestimmte Ausführungsformen hierin stellen ein neues Schema von energieeffizienten räumlichen Architekturen bereit. In bestimmten Ausführungsformen bilden diese Architekturen eine Struktur mit einer einzigartigen Zusammensetzung aus einer heterogenen Mischung aus kleinen, energieeffizienten, datenflussorientierten Verarbeitungselementen (PEs) mit einem leichtgewichtigen leitungsvermittelten Kommunikationsnetzwerk (z. B. Interconnect), z. B. mit einer gehärteten Unterstützung der Flussteuerung. Aufgrund der Energievorteile davon kann die Kombination dieser zwei Komponenten einen räumlichen Beschleuniger (z. B. als Teil eines Rechners) bilden, der zum Ausführen von kompiliergenerierten parallelen Programmen in einer extrem energieeffizienten Weise geeignet ist. Da diese Struktur heterogen ist, können bestimmte Ausführungsformen an unterschiedliche Anwendungsdomänen durch Einführen neuer domänenspezifischer PEs angepasst werden. Eine Struktur für die Hochleistungsberechnung könnte z. B. einige Anpassungen für doppelte Genauigkeit, fusioniertes Multiply-Add, enthalten, während eine Struktur, die auf tiefe neuronale Netzwerke abzielt, Gleitkomma-Operationen mit niedriger Genauigkeit beinhalten könnte.

[0029] Eine Ausführungsform eines räumlichen Architekturschemas, wie es z. B. in **Fig. 6** veranschaulicht ist, ist die Zusammensetzung von leichtgewichtigen Verarbeitungselementen (PE), die durch ein Inter-PE-Netzwerk verbunden sind. Im Allgemeinen können PEs Datenflussoperatoren umfassen, wo z. B., sobald alle Eingabeoperanden bei dem Datenflussoperator eingegangen sind, einige Operationen (z. B. Mikrobefehl oder Mikrobefehlssatz) ausgeführt werden, und die Ergebnisse an nachgeschaltete Operatoren weitergeleitet werden.

Steuerung, Planung und Datenspeicherung können daher unter den PEs verteilt sein, z. B. durch Entfernen des Aufwands der zentralisierten Strukturen, die klassische Prozessoren dominieren.

[0030] Programme können in Datenflussgraphen umgewandelt werden, die auf die Architektur abgebildet werden, indem PEs und Netzwerk konfiguriert werden, um den Steuerdatenflussgraphen des Programms auszudrücken. Kommunikationskanäle können flussgesteuert und vollständig gegengedrückt sein, z. B. sodass die PEs anhalten, wenn entweder die Quellkommunikationskanäle keine Daten aufweisen oder die Zielkommunikationskanäle voll sind. In einer Ausführungsform fließen Daten zur Laufzeit durch die PEs und Kanäle, die zum Implementieren der Operation konfiguriert wurden (z. B. ein beschleunigter Algorithmus). Zum Beispiel können Daten aus dem Speicher durch die Struktur eingestreamt werden und dann zurück in den Speicher gehen.

[0031] Ausführungsformen einer solchen Architektur können eine bemerkenswerte Leistungseffizienz gegenüber herkömmlichen Mehrkernprozessoren erreichen: die Rechenleistung (z. B. in Form von PEs) kann einfacher, energieeffizienter und umfangreicher sein als in größeren Kernen, und die Kommunikation kann direkt und meist kurzstreckig sein, z. B. im Gegensatz zum Auftreten über ein breites Vollchip-Netzwerk wie in typischen Mehrkernprozessoren. Da Ausführungsformen der Architektur des Weiteren extrem parallel sind, ist eine Anzahl von leistungsfähigen Optimierungen auf Schaltungs- und Geräteebene möglich, ohne den Durchsatz ernsthaft zu beeinträchtigen, z. B. Geräte mit niedrigem Verlust und niedriger Betriebsspannung. Diese Lower-Level-Optimierungen können gegenüber herkömmlichen Kernen noch größere Leistungsvorteile bringen. Die Kombination aus Effizienz auf Architektur-, Schaltungs- und Geräte-Level dieser Ausführungsformen ist zwingend. Ausführungsformen dieser Architektur können größere aktive Bereiche ermöglichen, während gleichzeitig die Transistordichte weiter steigt.

[0032] Ausführungsformen hierin bieten eine einzigartige Kombination aus Datenflussunterstützung und Leitungsvermittlung, um die Struktur kleiner, energieeffizienter zu machen und eine höhere Aggregatleistung gegenüber vorherigen Architekturen bereitzustellen. FPGAs sind im Allgemeinen auf eine feinkörnige Bitmanipulation abgestimmt, während Ausführungsformen hierin auf Gleitkomma-Operationen mit doppelter Genauigkeit in HPC-Anwendungen abgestimmt sind. Bestimmte Ausführungsformen hierin können einen FPGA zusätzlich zu einem CSA gemäß dieser Offenbarung aufweisen.

[0033] Bestimmte Ausführungsformen hierin kombinieren ein leichtgewichtiges Netzwerk mit energieeffizienten Datenflussverarbeitungselementen zum Bilden einer energieeffizienten HPC-Struktur mit hohem Durchsatz und geringer Latenz. Dieses Netzwerk mit geringer Latenz kann den Bau von Verarbeitungselementen mit weniger Funktionen ermöglichen, zum Beispiel nur einen oder zwei Befehle und ggf. ein sichtbares Architekturregister, weil es effizient ist, mehrere PE gemeinsam zu gruppieren, um ein vollständiges Programm zu bilden.

[0034] Bezüglich eines Prozessorkerns können CSA-Ausführungsformen hierin mehr Rechendichte und Energieeffizienz bereitstellen. Wenn zum Beispiel PEs sehr klein sind (z. B. verglichen mit einem Kern), kann der CSA viel mehr Operationen durchführen und hat viel mehr Rechenparallelität als ein Kern, z. B. womöglich etwa das 16-fache der Anzahl der FMAs einer Vektorverarbeitungseinheit (VPU - Vector Processing Unit). Zum Nutzen aller dieser Rechenelemente ist die Energie pro Operation in bestimmten Ausführungsformen sehr gering.

[0035] Die Energievorteile unserer Ausführungsformen dieser Datenflussarchitektur sind zahlreich. Parallelität ist in Datenflussgraphen explizit und Ausführungsformen der CSA-Architektur verbrauchen keine oder nur minimale Energie zur Extraktion davon, z. B. im Gegensatz zu Außer-Reihenfolge-Prozessoren, die jedes Mal, wenn ein Befehl ausgeführt wird, die Parallelität neu erkennen müssen. Da jedes PE in einer Ausführungsform für eine einzelne Operation verantwortlich ist, können die Registerdateien und Port-Zählungen klein sein, z. B. oft nur eins, und verbrauchen daher weniger Energie als ihre Gegenstücke im Kern. Bestimmte CSAs weisen viele PEs auf, von denen jedes Live-Programmwerte enthält, die den Gesamteffekt einer riesigen Registerdatei in einer traditionellen Architektur ergeben, was die Speicherzugriffe drastisch reduziert. In Ausführungsformen, bei denen der Speicher vom Mehrfachporttyp und verteilt ist, kann ein CSA viel mehr anstehende Speicheranforderungen erfüllen und mehr Bandbreite als ein Kern nutzen. Diese Vorteile können kombiniert werden, um einen Energiepegel pro Watt zu ergeben, der nur einen kleinen Prozentsatz der Kosten der bloßen arithmetischen Schaltung darstellt. Zum Beispiel kann im Falle einer Integer-Multiplikation ein CSA nicht mehr als 25% mehr Energie als die zugrundeliegende Multiplikationsschaltung verbrauchen. In Bezug auf eine Ausführungsform eines Kerns verbraucht eine Integeroperation in dieser CSA-Struktur weniger als 1/30 der Energie pro Integeroperation.

[0036] Aus einer Programmierperspektive ergibt die anwendungsspezifische Formbarkeit von Ausführungsformen der CSA-Architektur wichtige Vorteile gegenüber einer Vektorverarbeitungseinheit (VPU). Bei traditionellen, unflexiblen Architekturen müssen die Anzahl funktionaler Einheiten, wie Gleitdivision, oder die verschiedenen transzendentalen mathematischen Funktionen zum Ausgestaltungszeitpunkt basierend auf einem erwarteten Anwendungsfall gewählt werden. In Ausführungsformen der CSA-Architektur können solche Funktionen basierend auf den Anforderungen jeder Anwendung (z. B. durch einen Benutzer und nicht einen Hersteller) in der Struktur konfiguriert werden. Der Anwendungsdurchsatz kann dadurch weiter gesteigert werden. Gleichzeitig verbessert sich die Rechendichte von Ausführungsformen des CSA, indem die Verhärtung solcher Funktionen vermieden wird und stattdessen mehr Instanzen primitiver Funktionen wie Gleitmultiplikation vorgesehen werden. Diese Vorteile können bei HPC-Arbeitslasten von Bedeutung sein, von denen einige 75% der Gleitkomma-Ausführung in transzendentalen Funktionen verbrauchen.

[0037] Bestimmte Ausführungsformen des CSA stellen einen bedeutenden Fortschritt als eine datenflussorientierte räumliche Architektur dar, weil z. B. die PEs dieser Offenbarung kleiner, aber auch energieeffizienter sein können. Diese Verbesserungen können sich direkt aus der Kombination von datenflussorientierten PEs mit einer leichtgewichtigen, leitungsvermittelten Zwischenverbindung ergeben, die zum Beispiel im Gegensatz zu einem paketvermittelten Netzwerk eine Einzelzykluslatenz aufweist (z. B. mit einer 300% höheren Latenz an einem Minimum). Bestimmte Ausführungsformen der PEs unterstützen die 32-Bit- oder 64-Bit-Operation. Bestimmte Ausführungsformen hierin erlauben die Einführung neuer anwendungsspezifischer PEs, z. B. für Maschinenlernen oder Sicherheit, und sind keine rein homogene Kombination. Bestimmte Ausführungsformen hierin kombinieren leichtgewichtige datenflussorientierte Verarbeitungselemente mit einem leichtgewichtigen Niederlatenznetzwerk zum Bilden einer energieeffizienten Rechenstruktur.

[0038] Damit bestimmte räumliche Architekturen erfolgreich sind, müssen Programmierer sie mit relativ geringem Aufwand konfigurieren, z. B. sie gegenüber sequentiellen Kernen eine signifikante Energie- und Leistungsüberlegenheit erhalten. Bestimmte Ausführungsformen hierin stellen einen CSA (z. B. räumliche Struktur) bereit, der leicht programmiert werden kann (z. B. durch einen Kompilierer), energieeffizient und hochparallel ist. Bestimmte Ausführungsformen hierin stellen ein Netzwerk (z. B. Zwischenverbindungsnetz) bereit, dass diese drei Ziele erreicht. Aus einer Programmierbarkeitsperspektive stellen bestimmte Ausführungsformen des Netzwerks flussgesteuerte Kanäle bereit, die z. B. dem Steuerdatenflussgraphen (CDFG - Control-Dataflow Graph)-Modell der Ausführung entsprechen, das in Kompilierern verwendet wird. Bestimmte Netzwerkausführungsformen benutzen zweckgebundene leitungsvermittelte Verknüpfungen, so dass die Programmleistung sowohl von einem Menschen als auch einem Kompilierer leichter zu verstehen ist, weil die Leistung vorhersagbar ist. Bestimmte Netzwerkausführungsformen bieten sowohl hohe Bandbreite als auch niedrige Latenz. Bestimmte Netzwerkausführungsformen (z. B. statische leitungsvermittelte) stellen eine Latenz von 0 bis 1 Zyklus bereit (z. B. je nach der Übertragungsstrecke). Bestimmte Ausführungsformen stellen eine hohe Bandbreite durch paralleles Verlegen verschiedener Netzwerke bereit, z. B. in Low-Level-Metallen. Bestimmte Netzwerkausführungsformen kommunizieren in Low-Level-Metallen und über kurze Strecken und sind somit sehr energieeffizient.

[0039] Bestimmte Ausführungsformen von Netzwerken weisen eine Architekturunterstützung für die Durchflusssteuerung auf. Zum Beispiel können in räumlichen Beschleunigern, die aus kleinen Verarbeitungselementen (PEs) bestehen, die Kommunikationslatenz und die Bandbreite für die Gesamtprogrammleistung bedeutend sein. Bestimmte Ausführungsformen hierin stellen ein leichtgewichtiges leitungsvermitteltes Netzwerk bereit, das die Kommunikation zwischen PEs in räumlichen Verarbeitungsarrays, wie dem in **Fig. 6** gezeigten räumlichen Array, und den mikroarchitektonischen Steuermerkmalen, die notwendig sind, um dieses Netzwerk zu unterstützen, erleichtert. Bestimmte Ausführungsformen eines Netzwerks ermöglichen die Konstruktion von flussgesteuerten Punkt-zu-Punkt-Kommunikationskanälen, welche die Kommunikation der datenflussorientierten Verarbeitungselemente (PEs) unterstützen. Zusätzlich zu den Punkt-zu-Punkt-Kommunikationen können bestimmte Netzwerke hierin auch Multicast-Kommunikationen unterstützen. Die Kommunikationskanäle können durch statisches Konfigurieren des Netzwerks zum Bilden virtueller Schaltungen zwischen den PEs gebildet werden. Schaltungsumschalttechniken hierin können die Kommunikationslatenz verringern und die Netzwerkpufferung entsprechend minimieren, was zum Beispiel sowohl zu einer hohen Leistungsfähigkeit als auch zu einer hohen Energieeffizienz führt. In bestimmten Ausführungsformen eines Netzwerks kann die Inter-PE-Latenz so niedrig wie ein Null-Zyklus sein, was bedeutet, dass das nachgeschaltete PE mit Daten im Zyklus arbeiten kann, nachdem es erzeugt wurde. Zum Erhalten einer noch höheren Bandbreite und zum Zulassen von mehr Programmen kann eine Vielzahl von Netzwerken parallel angeordnet sein, wie z. B. in **Fig. 6** gezeigt.

[0040] Räumliche Architekturen, wie die in **Fig. 6** gezeigte, können die Zusammensetzung von leichtgewichtigen Verarbeitungselementen sein, die durch ein Inter-PE-Netzwerk verbunden sind. Programme, die als Da-

tenflussgraphen angesehen werden, können auf der Architektur durch Konfigurieren der PEs und des Netzwerks abgebildet werden. Im Allgemeinen können PEs als Datenflussoperatoren konfiguriert sein und, sobald alle Eingabeoperanden am PE eingehen, kann dann eine Operation erfolgen und das Ergebnis an die gewünschten nachgeschalteten PEs weitergeleitet werden. PEs können über zweckgebundene virtuelle Schaltungen kommunizieren, die durch statistisches Konfigurieren eines leitungsvermittelten Kommunikationsnetzwerks gebildet werden. Diese virtuellen Schaltungen können flussgesteuert und vollständig gegengedrückt sein, z. B. so, dass die PEs anhalten, wenn entweder die Quelle keine Daten aufweist oder der Zielspeicherplatz voll ist. Bei Laufzeit können Daten durch die PEs fließen und den abgebildeten Algorithmus implementieren. Zum Beispiel können Daten aus dem Speicher durch die Struktur eingestreamt werden und dann zurück in den Speicher gehen. Ausführungsformen dieser Architektur können eine bemerkenswerte Leistungseffizienz im Vergleich zu herkömmlichen Mehrkernprozessoren erreichen: wenn zum Beispiel eine Berechnung in der Form von PEs einfacher und zahlreicher ist als größere Kerne und die Kommunikation direkt ist, z. B. im Gegensatz zu einer Erweiterung des Speichersystems.

[0041] Fig. 6 veranschaulicht eine Beschleuniger-Kachel **600**, umfassend ein Array von Verarbeitungselementen (PEs) gemäß Ausführungsformen der Offenbarung. Das Zwischenverbindungsnetz ist als leitungsvermittelte, statisch konfigurierte Kommunikationskanäle dargestellt. Zum Beispiel eine Gruppe von Kanälen, die durch einen Schalter miteinander verbunden sind (z. B. Schalter **610** in einem ersten Netzwerk und Schalter **611** in einem zweiten Netzwerk). Das erste Netzwerk und das zweite Netzwerk können getrennt oder zusammengekoppelt sein. Der Schalter **610** kann z. B. einen oder mehrere der vier Datenpfade (**612**, **614**, **616**, **618**) zusammenkoppeln, z. B. wie zum Durchführen einer Operation gemäß einem Datenflussgraphen konfiguriert. In einer Ausführungsform kann die Anzahl von Datenpfaden jede beliebige Vielzahl sein. Das Verarbeitungselement (z. B. Verarbeitungselement **604**) kann wie hierin offenbart sein, zum Beispiel, wie in **Fig. 9A**. Die Beschleuniger-Kachel **600** weist eine Speicher-/Cache-Hierarchieschnittstelle **602** auf, z. B. zum Verbinden der Beschleuniger-Kachel **600** mit einem Speicher und/oder Cache. Ein Datenpfad (z. B. **618**) kann sich zu einer anderen Kachel erstrecken oder enden, z. B. am Rand einer Kachel. Ein Verarbeitungselement kann einen Eingabepuffer (z. B. Puffer **606**) und einen Ausgabepuffer (z. B. Puffer **608**) aufweisen.

[0042] Die Operationen können basierend auf der Verfügbarkeit ihrer Eingaben und dem Status des PE ausgeführt werden. Ein PE kann Operanden aus den Eingabekanälen erhalten und die Ergebnisse in Ausgabekanäle schreiben, auch wenn ein interner Registerstatus ebenfalls verwendet werden kann. Bestimmte Ausführungsformen hierin beinhalten ein konfigurierbares datenflussfreundliches PE. **Fig. 9** zeigt ein detailliertes Blockdiagramm eines solchen PE: dem Integer-PE. Dieses PE besteht aus verschiedenen I/O-Puffern, einer ALU, einem Speicherregister, einigen Befehlsregistern und einem Planer. Jeden Zyklus kann der Planer einen Befehl für die Ausführung basierend auf der Verfügbarkeit der Eingabe- und Ausgabepuffer und dem Status des PE auswählen. Das Ergebnis der Operation kann dann entweder in einen Ausgabepuffer oder ein Register (z. B. lokal oder PE) geschrieben werden. Die Daten, die in einen Ausgabepuffer geschrieben werden, können zu einem nachgeschalteten PE zur weiteren Verarbeitung transportiert werden. Dieser PE-Stil kann extrem energieeffizient sein, z. B. weil statt Daten aus einer komplexen Mehrportregisterdatei zu lesen, ein PE die Daten aus einem Register liest. Auf ähnliche Weise können die Befehle direkt in einem Register gespeichert werden, anstelle in einem virtuellen Befehls-Cache.

[0043] Befehlsregister können während eines speziellen Konfigurationsschrittes eingestellt werden. Während dieses Schritts können Hilfssteuerdrähte und Zustand zusätzlich zum inter-PE-Netzwerk zum Streamen in der Konfiguration über die verschiedenen PEs, welche die Struktur umfassen, verwendet werden. Als Ergebnis der Parallelität können bestimmte Ausführungsformen eines solchen Netzwerks eine schnelle Rekonfiguration bereitstellen, z. B. kann ein kachelgroßes Netzwerk in weniger als etwa 10 Mikrosekunden konfiguriert sein.

[0044] Fig. 9 repräsentiert eine Beispielkonfiguration eines Verarbeitungselements, in dem z. B. alle Architekturelemente minimal bemessen sind. In anderen Ausführungsformen ist jede der Komponenten eines Verarbeitungselements unabhängig bemessen, um neue PEs zu erzeugen. Zum Handhaben komplizierterer Programme kann z. B. eine größere Anzahl von Befehlen eingefügt werden, die durch ein PE ausführbar sind. Eine zweite Dimension der Konfigurierbarkeit ist abhängig von der PE-Arithmetik-Logik-Einheit (ALU). In **Fig. 9** ist ein Integer-PE dargestellt, das Addition, Subtraktion und verschiedene logische Operationen unterstützen kann. Andere Arten von PEs können durch Ersetzen unterschiedlicher Arten von Funktionseinheiten in dem PE geschaffen werden. Ein Integer-Multiplikations-PE kann beispielsweise keine Register, einen Einzelbefehl und einen einzigen Ausgabepuffer aufweisen. Bestimmte Ausführungsformen eines PE zerlegen eine fusionierte Multiplikationsaddition (FMA) in separate, aber eng gekoppelte, Gleitkomma-Multiplikations- und Gleitkomma-Additionseinheiten zum Verbessern der Unterstützung von Multiply-Add-Schwerarbeitslasten. PEs werden nachstehend erläutert.

[0045] **Fig. 7A** veranschaulicht ein konfigurierbares Datenpfad-Netzwerk **700** (z. B. von Netzwerk eins oder Netzwerk zwei, die in Bezug auf **Fig. 6** erläutert wurden) gemäß Ausführungsformen der Offenbarung. Das Netzwerk **700** weist mehrere Multiplexer (z. B. Multiplexer **702**, **704**, **706**) auf, die konfiguriert werden können (z. B. über zugehörige Steuersignale), um einen oder mehrere Datenpfade (z. B. von PEs) miteinander zu verbinden. **Fig. 7B** veranschaulicht ein konfigurierbares Flusssteuerpfad-Netzwerk **701** (z. B. Netzwerk eins oder Netzwerk zwei, die in Bezug auf **Fig. 6** erläutert wurden) gemäß Ausführungsformen der Offenbarung. Ein Netzwerk kann ein leichtgewichtiges PE-PE-Netzwerk sein. Bestimmte Ausführungsformen eines Netzwerks können als ein Satz von zusammenfügbaren Grundelementen zum Bau von verteilten, Punkt-zu-Punkt-Datenkanälen gedacht sein. **Fig. 7A** zeigt ein Netzwerk, das zwei aktivierte Kanäle aufweist, die fette schwarze Linie und die gepunktete schwarze Linie. Der Kanal der fetten schwarzen Linie ist Multicast, z. B. wird eine einzelne Eingabe an zwei Ausgaben gesendet. Es sei angemerkt, dass sich die Kanäle an einigen Punkten innerhalb eines einzelnen Netzwerks schneiden können, selbst wenn zweckgebundene leitungsvermittelte Pfade zwischen den Kanalendpunkten gebildet werden. Des Weiteren stellt diese Kreuzung keine strukturelle Gefahr zwischen den zwei Kanälen dar, sodass jeder unabhängig und bei voller Bandbreite arbeitet.

[0046] Das Implementieren von verteilten Datenkanälen kann zwei Pfade aufweisen, wie in **Fig. 7A** bis **Fig. 7B** veranschaulicht. Die Vorwärtsverbindung, oder Datenpfad, trägt Daten von einem Erzeuger zu einem Verbraucher. Multiplexer können zum Lenken von Daten und Validieren von Bits von dem Erzeuger zu dem Verbraucher konfiguriert sein, z. B. wie in **Fig. 7A**. Im Fall von Multicast werden die Daten zu einer Vielzahl von Verbraucherendpunkten gelenkt. Der zweite Teil dieser Ausführungsform eines Netzwerks ist die Flusssteuerung oder der Gegendruckpfad, der in Gegenrichtung des Vorwärtsdatenpfads fließt, z. B. wie in **Fig. 7B**. Die Verbraucherendpunkte können geltend gemacht werden, wenn sie zum Annehmen neuer Daten bereit sind. Diese Signale können dann zurück zum Erzeuger unter Verwendung der konfigurierbaren logischen Konjunktionen gelenkt werden, die als Flusssteuerungsfunktion in **Fig. 7B** gekennzeichnet sind (z. B. Rückfluss). In einer Ausführungsform kann jede Flusssteuerungsfunktionsschaltung mehrere Schalter (z. B. Muxes) aufweisen, wie z. B. ähnlich denen aus **Fig. 7A**. Der Flusssteuerpfad kann zurückkehrende Steuerdaten von dem Verbraucher an den Erzeuger handhaben. Konjunktionen können Multicast ermöglichen, wobei z. B. jeder Verbraucher Daten empfangen kann, bevor der Erzeuger voraussetzt, dass diese empfangen wurden. In einer Ausführungsform ist ein PE ein PE, das einen Datenflussoperator wie seine Architekturschnittstelle aufweist. Zusätzlich oder alternativ kann in einer Ausführungsform ein PE eine Art von PE (z. B. in der Struktur) sein, z. B. ein PE, das eine Befehlszweiger-, Triggerbefehl- oder Zustandsmaschinen-Architekturschnittstelle aufweist, aber nicht darauf beschränkt ist.

[0047] Das Netzwerk kann statisch konfiguriert sein, z. B. zusätzlich zu PEs, die statisch konfiguriert sind. Während des Konfigurationsschrittes können Konfigurationsbits an jeder Netzwerkkomponente eingestellt werden. Diese Bits steuern z. B. die Muxauswahl und die Flusssteuerfunktionen. Ein Netzwerk kann mehrere Netzwerke umfassen, z. B. ein Datenpfad-Netzwerk und ein Flusssteuerungsnetzwerk. Eine Netzwerk oder mehrere Netzwerke können Pfade unterschiedlicher Breiten benutzen (z. B. einer ersten Breite und einer engeren oder breiteren Breite). In einer Ausführungsform weist ein Datenpfad-Netzwerk eine breitere (z. B. Bittransport) Breite auf als die Breite eines Flusssteuerpfad-Netzwerks. In einer Ausführungsform weist jedes von einem ersten Netzwerk und einem zweiten Netzwerk sein eigenes Datenpfad-Netzwerk und Flusssteuerpfad-Netzwerk auf, z. B. Datenpfad-Netzwerk A und Flusssteuerpfad-Netzwerk A und ein breiteres Datenpfad-Netzwerk B und ein Flusssteuerpfad-Netzwerk B.

[0048] Bestimmte Ausführungsformen eines Netzwerks sind pufferlos und die Daten müssen sich zwischen dem Erzeuger und Verbraucher in einem Einzelzyklus bewegen. Bestimmte Ausführungsformen eines Netzwerks sind ungebunden, das heißt, das Netzwerk überspannt die gesamte Struktur. In einer Ausführungsform kommuniziert ein PE mit einem anderen PE in einem Einzelzyklus. In einer Ausführungsform können zum Verbessern der Routingbandbreite mehrere Netzwerke parallel zwischen die Reihen der PE gelegt werden.

[0049] Bezüglich der FPGAs haben bestimmte Ausführungsformen von Netzwerken hierin drei Vorteile: Bereich, Frequenz und Programmausdruck. Bestimmte Ausführungsformen von Netzwerken arbeiten grobkörnig, was z. B. die Anzahl an Konfigurationsbits reduziert und dadurch den Netzwerkbereich. Bestimmte Ausführungsformen von Netzwerken erhalten die Bereichsreduktion durch Implementieren der Steuerlogik direkt im Schaltkreis (z. B. Silicium). Bestimmte Ausführungsformen gehärteter Netzwerkimplementierungen genießen gegenüber FPGA auch einen Frequenzvorteil. Aufgrund eines Bereichs- und Frequenzvorteils kann ein Leistungsvorteil vorhanden sein, wenn eine geringere Spannung als Durchsatzparität verwendet wird. Schließlich stellen bestimmte Ausführungsformen von Netzwerken bessere High-Level-Semantiken als FPGA-Drähte bereit, insbesondere in Bezug auf die variable Zeitsteuerung, weshalb solche bestimmten Ausführungsformen leichter durch die Kompilierer anzuzielen sind. Bestimmte Ausführungsformen von Netzwerken hierin können

als ein Satz von zusammensetzbaren Grundelementen zum Bau von verteilten, Punkt-zu-Punkt-Datenkanälen gedacht sein.

[0050] In bestimmten Ausführungsformen kann eine Multicast-Quelle ihre Daten erst dann geltend machen, wenn ein Bereit-Signal von jeder Senke empfangen wurde. Daher können ein zusätzliches Konjunktions- und Steuerbit in dem Multicastfall benutzt werden.

[0051] Wie bestimmte PEs kann das Netzwerk statistisch konfiguriert sein. Während dieses Schrittes werden Konfigurationsbits an jeder Netzwerkkomponente eingestellt. Diese Bits steuern z. B. die Muxauswahl und die Flusssteuerungsfunktion. Der Vorwärtspfad unseres Netzwerks erfordert, dass einige Bits ihre Muxe schwingen. In dem in **Fig. 7A** gezeigten Beispiel werden vier Bits pro Hop erfordert: die Ost- und West-Muxe benutzen jeweils ein Bit, während die südlich gebundenen Muxe zwei Bits benutzen. In dieser Ausführungsform können vier Bits für den Datenpfad benutzt werden, es können aber 7 Bits für die Flussteuerungsfunktion benutzt werden (z. B. in dem Flussteuerpfad-Netzwerk). Andere Ausführungsformen können mehr Bits benutzen, z. B., wenn ein CSA ferner eine Nord-Süd-Richtung benutzt. Die Flusssteuerungsfunktion kann ein Steuerbit für jede Richtung benutzen, aus welcher die Flusssteuerung kommen kann. Dies kann das Einstellen der Sensitivität der Flusssteuerungsfunktion statisch ermöglichen. Die Tabelle 1 unten fasst die Boolsche Algebra-Implementierung der Flussteuerungsfunktion für das Netzwerk in **Fig. 7B** zusammen, wobei die Konfigurationsbits in Großbuchstaben aufgeführt sind. In diesem Beispiel werden sieben Bits benutzt.

Tabelle 1 Flussimplementierung

readyToEast	(EAST_WEST_SENSITIVE+readyFromWest) * (EAST_SOUTH_SENSITIVE+readyFromSouth)
readyToWest	(WEST_EAST_SENSITIVE+readyFromEast) * (WEST_SOUTH_SENSITIVE+readyFromSouth)
readyToNorth	(NORTH_WEST_SENSITIVE+readyFromWest) * (NORTH_EAST_SENSITIVE+readyFromEast) * (NORTH_SOUTH_SENSITIVE+readyFromSouth)

[0052] Für die dritte Flusssteuerungsbox von links in **Fig. 7B** sind EAST_WEST_SENSITIVE (ost-west-sensitiv) und NORTH_SOUTH_SENSITIVE (nord-süd-sensitiv) als Satz dargestellt, um die Flussteuerung für Kanäle in fetter Linie bzw. gestrichelter Linie zu implementieren.

[0053] **Fig. 8** veranschaulicht eine Hardware-Prozessor-Kachel, 800 umfassend einen Beschleuniger **802** gemäß Ausführungsformen der Offenbarung. Der Beschleuniger **802** kann ein CSA gemäß dieser Offenbarung sein. Die Kachel **800** weist mehrere Cache-Bänke (z. B. Cache-Bank **808**) auf. Abfrageadressdatei (RAF)-Schaltungen **810** können aufgenommen sein, z. B. wie unten in Abschnitt 3.2 erläutert. ODI kann sich auf eine On-Die-Zwischenverbindung beziehen, z. B. eine Zwischenverbindung, die sich über den gesamten Chip erstreckt, der alle Kacheln miteinander verbindet. OTI kann sich auf eine On-Tile-Zwischenverbindung beziehen, die sich z. B. über eine Kachel erstreckt, die z. B. die Cache-Bänke auf der Kachel miteinander verbindet.

Verarbeitungselemente

[0054] In bestimmten Ausführungsformen weist ein CSA ein Array aus heterogenen PEs auf, in denen die Struktur aus verschiedenen Typen von PEs zusammengesetzt ist, von denen jedes nur einen Untersatz von Datenflussoperatoren implementiert. Rein beispielhaft zeigt **Fig. 9** eine provisionale Implementierung eines PE, das einen breiten Satz von Integer- und Steueroperationen implementieren kann. Andere PEs, einschließlich solcher, die eine Gleitkomma-Addition, Gleitkomma-Multiplikation, Pufferung und bestimmte Steueroperationen aufweisen, können einen ähnlichen Implementierungsstil aufweisen, z. B. mit der angemessenen (Datenflussoperator)-Schaltung, welche die ALU ersetzt. PEs (z. B. Datenflussoperatoren) eines CSA können vor dem Beginn der Ausführung zum Implementieren einer bestimmten Datenfluss-Operation von einem der Sätze, die das PE unterstützt, konfiguriert (z. B. programmiert) werden. Eine Konfiguration kann eines oder zwei Steuerwörter umfassen, die einen Opcode spezifizieren, der die ALU steuert, die verschiedenen Multiplexer innerhalb des PE lenkt und den Datenfluss in die PE-Kanäle hinein und aus diesen heraus betätigt. Die Datenflussoperatoren können durch Mikrocodieren dieser Konfigurationsbits implementiert werden. Das dargestellte Integer-PE **900** in **Fig. 9** ist als Einzelstufen-Logik-Pipeline, die von oben nach unten fließt, organisiert. Daten treten von einem eines Satzes lokaler Netzwerke ein, in denen sie in einem Eingabepuffer zur nachfolgenden Operation registriert werden. Jedes PE kann eine Anzahl von breiten, datenausgerichteten und schmalen, steuerungsausgerichteten Kanälen unterstützen. Die Anzahl der vorgesehenen Kanäle kann basierend auf der

PE-Funktionalität variieren, jedoch weist eine Ausführungsform eines ganzzahlorientierten PE 2 breite und 1-2 schmale Eingabe- und Ausgabekanäle auf. Obwohl das Integer-PE als eine Einzelzyklus-Pipeline implementiert ist, können andere Wahlen für das Pipelinenetz benutzt werden. Multiplikations-PEs können z. B. eine Mehrzahl von Pipelinestufen aufweisen.

[0055] Die PE-Ausführung kann im Datenflussstil voranschreiten. Basierend auf dem Konfigurationsmikrocode kann der Planer den Status der Eintritts- und Austrittspuffer des PE untersuchen und, wenn alle Eingaben für die konfigurierte Operation eingegangen sind und der Austrittspuffer der Operation verfügbar ist, die tatsächliche Ausführung der Operation durch einen Datenflussoperator (z. B. auf der ALU) inszenieren. Der resultierende Wert kann in dem konfigurierten Austrittspuffer platziert werden. Übertragungen zwischen dem Austrittspuffer eines PE und dem Eintrittspuffer eines anderen PE können asynchron auftreten, wenn eine Pufferung verfügbar wird. In bestimmten Ausführungsformen sind die PEs derart ausgestattet, dass mindestens eine Datenfluss-Operation pro Zyklus abgeschlossen wird. In Abschnitt 2 ist der Datenflussoperator als primitive Operationen umschließend erläutert, wie z. B. Add, Xor oder Pick. Bestimmte Ausführungsformen können Vorteile bei Energie, Bereich, Leistung und Latenz bereitstellen. In einer Ausführungsform können mit einer Erweiterung zu einem PE-Steuerpfad, mehr fusionierte Kombinationen ermöglicht werden. In einer Ausführungsform beträgt die Breite der Verarbeitungselemente 64 Bits, z. B. für die starke Nutzung der Doppelpräzision-Gleitkomma-Berechnung in HPC und zum Unterstützen der 64-Bit-Speicher-Adressierung.

Kommunikationsnetzwerke

[0056] Ausführungsformen der CSA-Mikroarchitektur stellen eine Hierarchie von Netzwerken bereit, die zusammen eine Implementierung der architektonischen Abstraktion latenzinsensitiver Kanäle über mehrere Kommunikationsmaßstäbe bereitstellen. Das niedrigste Level der CSA-Kommunikationshierarchie kann das lokale Netzwerk sein. Das lokale Netzwerk kann ein statisch leitungsvermitteltes sein, das z. B. die Konfigurationsregister zum Schwingen eines oder mehrerer Multiplexer in dem lokalen Netzwerkdatenpfad zum Bilden fester elektrischer Pfade zwischen kommunizierenden PEs verwendet. In einer Ausführungsform wird die Konfiguration des lokalen Netzwerkes einmal pro Datenflussgraph eingestellt, z. B. gleichzeitig mit der PE-Konfiguration. In einer Ausführungsform optimiert die statische Leitungsvermittlung die Energie, z. B. wenn eine große Mehrheit (evtl. größer als 95%) des CSA-Kommunikationsverkehrs das lokale Netzwerk kreuzt. Ein Programm kann Begriffe aufweisen, die in einer Mehrzahl von Ausdrücken verwendet werden. Zum Optimieren in diesem Fall stellen Ausführungsformen hierin eine Hardwareunterstützung für Multicast innerhalb des lokalen Netzwerks bereit. Mehrere verschiedene lokale Netzwerke können zusammengefasst werden, um Routingkanäle zu bilden, die z. B. verschachtelt (als ein Gitter) zwischen Reihen und Spalten der PEs sind. Als eine Optimierung können mehrere verschiedene Netzwerke zum Tragen von Steuerungs-Token aufgenommen sein. Im Vergleich mit einer FPGA-Zwischenverbindung kann ein lokales CSA-Netzwerk an der Granularität des Datenpfads geleitet werden und ein weiterer Unterschied kann die CSA-Behandlung der Steuerung sein. Eine Ausführungsform eines lokalen CSA-Netzwerks ist explizit flussgesteuert (z. B. gegengedrückt). Zum Beispiel muss ein CSA für jeden Vorwärts-Datenpfad und Multiplexer-Satz einen Rückwärtsfluss-Flusssteuerpfad bereitstellen, der physisch mit dem Vorwärts-Datenpfad gepaart ist. Die Kombination der zwei mikroarchitektonischen Pfade kann eine Punkt-zu-Punkt-Implementierung der latenzinsensitiven Kanalabstraktion mit niedriger Latenz, niedriger Energie und niedrigem Bereich bereitstellen. In einer Ausführungsform sind die Flusssteuerungsleitungen eines CSA für das Benutzerprogramm nicht sichtbar, können jedoch durch die Architektur in Betrieb des Benutzerprogramms manipuliert werden. Zum Beispiel können die in Abschnitt 2.2 beschriebenen Ausnahmehandhabungsmechanismen erreicht werden, indem Flusssteuerungsleitungen bei der Erkennung eines Ausnahmezustands in einen Zustand „nicht vorhanden“ gezogen werden. Diese Aktion kann nicht nur die Teile der Pipeline, die an der fehlerhaften Berechnung beteiligt sind, elegant anhalten, sondern kann auch den Maschinenzustand, der zu der Ausnahme geführt hat, bewahren, z. B. für eine Diagnoseanalyse. Die zweite Netzwerkschicht, z. B. das Mezzanine-Netzwerk, kann ein gemeinsam genutztes paketvermitteltes Netzwerk sein. Das Mezzanine-Netzwerk (z. B. das Netzwerk, das schematisch durch die gestrichelte Box in **Fig. 22** dargestellt ist) kann allgemeinere Kommunikationen mit größerer Reichweite auf Kosten von Latenz, Bandbreite und Energie bereitstellen. In gut gerouteten Programmen kann die meiste Kommunikation in dem lokalen Netzwerk stattfinden, weshalb die Mezzanine-Netzwerkbereitstellung im Vergleich beträchtlich reduziert wird, zum Beispiel kann sich jedes PE mit mehreren lokalen Netzwerken verbinden, aber der CSA nur einen Mezzanine-Endpunkt pro logischer Nachbarschaft der PEs bereitstellen. Da das Mezzanine tatsächlich ein gemeinsam genutztes Netzwerk ist, kann jedes Mezzanine-Netzwerk mehrere logisch unabhängige Kanäle tragen und z. B. mit einer Vielzahl von virtuellen Kanälen versehen sein. In einer Ausführungsform ist die Hauptfunktion des Mezzanine-Netzwerks die Bereitstellung von Weitbereichskommunikationen zwischen PEs und zwischen PEs und Speicher. Zusätzlich zu dieser Kapazität kann das Mezzanine-Netzwerk auch ein Laufzeitunterstützungsnetzwerk betreiben, durch das z. B. verschiedene Dienste auf die komplette Struktur auf benutzerpro-

grammtransparente Weise zugreifen. In dieser Eigenschaft kann der Mezzanine-Endpunkt als eine Steuerung für seine lokale Nachbarschaft dienen, z. B. während der CSA-Konfiguration. Zum Bilden der Kanäle, die eine CSA-Kachel überbrücken, können drei Teilkanäle und zwei lokale Netzwerkkanäle (die Verkehr zu und von einem einzelnen Kanal in dem Mezzanine-Netzwerk tragen) benutzt werden. In einer Ausführungsform wird ein Mezzanine-Kanal benutzt, z. B. ein Mezzanine- und zwei lokale = 3 Netzwerk-Hops insgesamt.

[0057] Die Zusammensetzbarkeit von Kanälen über Netzwerkschichten hinweg kann auf Higher-Level-Netzwerkebenen an den Inter-Tile-, Inter-Die- und Fabric-Granularitäten erweitert werden.

[0058] **Fig. 9** veranschaulicht Verarbeitungselement **900** gemäß Ausführungsformen der Offenbarung. In einer Ausführungsform wird das Betriebskonfigurationsregister **919** während der Konfiguration (z. B. Abbildung) geladen und spezifiziert die bestimmte Operation (oder Operationen), die dieses Verarbeitungs- (z. B. Rechen-) Element durchführen soll. Die Aktivität von Register **920** kann durch diese Operation (eine Ausgabe von Mux **916**, die z. B. durch den Planer **914** gesteuert wird) gesteuert werden. Der Planer **914** kann eine Operation oder Operationen des Verarbeitungselements **900** planen, zum Beispiel, wenn Eingabedaten und die Steuereingabe eintreffen. Der Steuereingabepuffer **922** ist mit dem lokalen Netzwerk **902** verbunden (z. B. kann das lokale Netzwerk **902** ein Datenpfad-Netzwerk wie in **Fig. 7A** und ein Flusssteuerpfad-Netzwerk wie in **Fig. 7B** aufweisen) und wird mit einem Wert geladen, wenn er eintrifft (wenn z. B. das Netzwerk ein/mehrere Datenbit/s und eines/mehrere gültige Bit/s aufweist). Der Steuerausgabepuffer **932**, der Datenausgabepuffer **934** und/oder der Datenausgabepuffer **936** können eine Ausgabe des Verarbeitungselements **900** empfangen, z. B. wie durch die Operation (eine Ausgabe des Mux **916**) gesteuert. Das Zustandsregister **938** kann immer dann geladen werden, wenn die ALU **918** ausgeführt wird (wird auch durch die Ausgabe von Mux **916** gesteuert). Die Daten in dem Steuereingabepuffer **922** und Steuerausgabepuffer **932** können ein Einzelbit sein. Der Mux **921** (z. B. Operand A) und Mux **923** (z. B. Operand B) können Quelleingaben sein.

[0059] Angenommen, die Operation dieses Verarbeitungs- (z. B. Rechen-) Elements ist (oder beinhaltet) zum Beispiel, was als ein Pick in **Fig. 3B** bezeichnet wird. Das Verarbeitungselement **900** muss dann die Daten entweder von dem Dateneingabepuffer **924** oder dem Dateneingabepuffer **926** auswählen, z. B. um zu Datenausgabepuffer **934** (z. B. Standard) oder Datenausgabepuffer **936** zu gehen. Das Steuerbit in **922** kann daher eine 0 angeben, wenn es von dem Dateneingabepuffer **924** ausgewählt wird oder 1, wenn es von dem Dateneingabepuffer **926** ausgewählt wird.

[0060] Angenommen, die Operation dieses Verarbeitungs- (z. B. Rechen-) Elements ist (oder beinhaltet) zum Beispiel, was als ein Switch (Schalter) in **Fig. 3B** bezeichnet wird. Das Verarbeitungselement **900** muss dann die Daten zum Datenausgabepuffer **934** oder dem Datenausgabepuffer **936** ausgeben, z. B. vom Dateneingabepuffer **924** (z. B. Standard) oder Dateneingabepuffer **926**. Das Steuerbit in **922** kann daher eine 0 angeben, wenn es zum Datenausgabepuffer **934** ausgegeben wird oder 1, wenn es zum Datenausgabepuffer **936** ausgegeben wird.

[0061] Eine Vielzahl von Netzwerken (z. B. Zwischenverbindungen) kann mit einem Verarbeitungselement verbunden sein, z. B. die (Eingabe-) Netzwerke **902**, **904**, **906** und (Ausgabe-) Netzwerke **908**, **910**, **912**. Die Verbindungen können Schalter sein, wie z. B. in Bezug auf **Fig. 7A** und **Fig. 7B** erläutert. In einer Ausführungsform weist jedes Netzwerk zwei Unternetzwerke (oder zwei Kanäle auf dem Netzwerk) auf, z. B. eines für das Datenpfad-Netzwerk aus **Fig. 7A** und eines für das Flusssteuer- (z. B. Gegendruck-) Pfadnetzwerk aus **Fig. 7B**. Als ein Beispiel ist das lokale Netzwerk **902** (z. B. als Steuerungszwischenverbindung eingerichtet) als mit dem Steuereingabepuffer **922** geschaltet (z. B. verbunden) dargestellt. In dieser Ausführungsform kann ein Datenpfad (z. B. ein Netzwerk wie in **Fig. 7A)** den Steuereingabewert (z. B. Bit oder Bits) (z. B. ein Steuer-Token) tragen, und der Flusssteuerpfad (z. B. das Netzwerk) kann das Gegendrucksignal (z. B. Gegendruck- oder Nicht-Gegendruck-Token) von dem Steuereingabepuffer **922** tragen, um z. B. dem vorgeschalteten Erzeuger (z. B. PE) anzuzeigen, dass ein neuer Steuereingabewert nicht in den Steuereingabepuffer **922** zu laden (z. B. senden) ist, bis das Gegendrucksignal angibt, dass Platz in dem Steuereingabepuffer **922** für den neuen Steuereingabewert (z. B. von einer Steuerausgabepuffer des vorgeschalteten Erzeugers) vorhanden ist. In einer Ausführungsform kann der neue Steuereingabewert nicht in den Steuereingabepuffer **922** eintreten, bis sowohl (i) der vorgeschaltete Erzeuger das Gegendrucksignal „Platz verfügbar“ von dem „Steuereingabe“-Puffer **922** empfängt, als auch (ii) der neue Steuereingabewert von dem vorgeschalteten Erzeuger gesendet wird, und dies z. B. das Verarbeitungselement **900** anhält, bis dies geschieht (und Platz in dem bzw. den Zielausgabepuffer(n) verfügbar ist).

[0062] Der Dateneingabepuffer **924** und der Dateneingabepuffer **926** können in ähnlicher Weise arbeiten, z. B. ist das lokale Netzwerk **904** (z. B. als eine Datenzwischenverbindung (im Gegensatz zur Steuerung) einge-

richtet) als mit dem Dateneingabepuffer **924** geschaltet (z. B. verbunden) dargestellt. In dieser Ausführungsform kann ein Datenpfad (z. B. ein Netzwerk wie in **Fig. 7A)** den Dateneingabewert (z. B. Bit oder Bits) (z. B. ein Datenfluss-Token) tragen, und der Flusssteuerpfad (z. B. das Netzwerk) kann das Gegendrucksignal (z. B. Gegendruck- oder Nicht-Gegendruck-Token) von dem Dateneingabepuffer **924** tragen, um z. B. dem vorgeschalteten Erzeuger (z. B. PE) anzuzeigen, dass ein neuer Dateneingabewert nicht in den Dateneingabepuffer **924** zu laden (z. B. senden) ist, bis das Gegendrucksignal angibt, dass Platz in dem Steuereingabepuffer **924** für den neuen Dateneingabewert (z. B. von einer Datenausgabepuffer des vorgeschalteten Erzeugers) vorhanden ist. In einer Ausführungsform kann der neue Dateneingabewert nicht in den Dateneingabepuffer **924** eintreten, bis sowohl (i) der vorgeschaltete Erzeuger das Gegendrucksignal „Platz verfügbar“ von dem „Dateneingabe“-Puffer **924** empfängt, als auch (ii) der neue Dateneingabewert von dem vorgeschalteten Erzeuger gesendet wird, und dies z. B. das Verarbeitungselement **900** anhält, bis dies geschieht (und Platz in dem bzw. den Zielausgabepuffer(n) verfügbar ist). Ein Steuerausgabewert und/oder Datenausgabewert können in ihren jeweiligen Ausgabepuffern (z. B. **932**, **934**, **936**) angehalten werden, bis ein Gegendrucksignal anzeigt, dass Platz in dem Eingabepuffer für das bzw. die nachgeschalteten Verarbeitungselemente verfügbar ist.

[0063] Ein Verarbeitungselement **900** kann von der Ausführung abgehalten werden, bis seine Operanden (z. B. ein Steuereingabewert und sein bzw. seine entsprechender/n Dateneingabewert oder -werte) empfangen werden und/oder bis Platz in dem/den Ausgabepuffer(n) des Verarbeitungselements **900** für Daten vorhanden ist, die durch die Ausführung der Operation für diese Operanden zu erzeugen sind.

Speicherschnittstelle

[0064] Die Abfrageadressdatei- (RAF) Schaltung, von der eine vereinfachte Version in **Fig. 10** gezeigt ist, kann für die Ausführung von Speicheroperationen verantwortlich sein und dient als Vermittler zwischen der CSA-Struktur und der Speicherhierarchie. Als solche kann die Hauptaufgabe der Mikroarchitektur der RAF darin bestehen, das Außer-Reihenfolge-Speichersubsystem mit der In-Reihenfolge-Semantik der CSA-Struktur zu rationalisieren. In dieser Eigenschaft kann die RAF-Schaltung mit Abschlusspuffern ausgestattet sein, z. B. warteschlangenähnlichen Strukturen, welche die Speicherantworten neu ordnen und diese zur Struktur in der Anforderungsreihenfolge zurückführen. Die zweite Hauptfunktionalität der RAF-Schaltung kann darin bestehen, Unterstützung in Form einer Adressumsetzung und eines Seitenwanderers bereitzustellen. Eingehende virtuelle Adressen können unter Verwendung eines kanalassoziativen Adressenübersetzungspuffers (Translation Lookaside Puffer - TLB) in physische Adressen umgesetzt werden. Zum Bereitstellen einer breiten Speicherbandbreite kann jede CSA-Kachel eine Vielzahl von RAF-Schaltungen aufweisen. Wie bei den verschiedenen PEs der Struktur können die RAF-Schaltungen in einem Datenflussstil durch Prüfen auf Verfügbarkeit der Eingabeargumente und Ausgabepufferung, wenn notwendig, vor dem Auswählen eines Speicherbetriebs zur Ausführung betrieben werden. Im Gegensatz zu einigen PEs wird die RAF-Schaltung jedoch zwischen mehreren verschiedenen gemeinsam angeordneten Speicheroperationen gemultiplext. Eine gemultiplexte RAF-Schaltung kann zum Minimieren des Bereichs oberhalb ihrer verschiedenen Subkomponenten verwendet werden, z. B. zum gemeinsamen Nutzen des Accelerator Cache Interface (ACI) -Ports (ausführlicher in Abschnitt 3.4 beschrieben), der gemeinsam genutzten Virtualspeicher (SVM)-Support-Hardware, Mezzanine-Netzwerkschnittstelle und anderen Hardware-Verwaltungseinrichtungen. Es kann jedoch einige Programmeigenschaften geben, die diese Wahl motivieren. In einer Ausführungsform muss ein (z. B. gültiger) Datenflussgraph den Speicher in einem gemeinsam genutzten virtuellen Speichersystem abfragen. Speicherlatenzgebundene Programme, wie z. B. Graph-Traversierungen, können viele getrennte Speicheroperationen benutzen, um die Speicherbandbreite aufgrund des speicherabhängigen Steuerflusses zu sättigen. Obwohl jede RAF gemultiplext sein kann, kann ein CSA eine Vielzahl (z. B. zwischen 8 und 32) RAFs bei einer Kachel-Granularität aufweisen, um eine adäquate Bandbreite bereitzustellen. RAFs können mit dem Rest der Struktur über sowohl das lokale Netzwerk als auch Mezzanine-Netzwerk kommunizieren. Wenn die RAFs gemultiplext sind, kann jede RAF mit mehreren verschiedenen Ports in dem lokalen Netzwerk vorgesehen sein. Diese Ports können als minimallatenter, hochdeterministischer Pfad zum Speicher zur Verwendung durch latenzsensitive oder Hochbandbreiten-Speicheroperationen dienen. Zusätzlich kann eine RAF mit einem Mezzanine-Endpunkt vorgesehen sein, der z. B. einen Speicherzugriff auf Laufzeitdienste und entfernte Benutzerebenen-Speicherzugangseinrichtungen bereitstellt.

[0065] Fig. 10 veranschaulicht eine Abfrage-Adressdatei (RAF)-Schaltung **1000** gemäß Ausführungsformen der Offenbarung. In einer Ausführungsform kann zur Zeit der Konfiguration die Speicherlast- und -speicheroperationen, die in einem Datenflussgraphen waren, in Registern **1010** spezifiziert werden. Die Bögen zu diesen Speicheroperationen in den Datenflussgraphen können dann mit den Eingabewarteschlangen **1022**, **1024** und **1026** verbunden werden. Die Bögen aus diesen Speicheroperationen müssen somit die Abschlusspuffer **1028**, **1030** oder **1032** verlassen. Abhängigkeits-Token (die einzelne Bits sein können) kommen in die

Warteschlangen **1018** und **1020**. Abhängigkeits-Token müssen die Warteschlange **1016** verlassen. Der Abhängigkeits-Tokenzähler **1014** kann eine kompakte Repräsentation einer Warteschlange sein und eine Anzahl von Abhängigkeits-Token, die für eine vorgegebene Eingabewarteschlange verwendet werden, nachverfolgen. Wenn die Abhängigkeits-Tokenzähler **1014** gesättigt sind, können keine zusätzlichen Abhängigkeits-Token für neue Speicheroperationen generiert werden. Entsprechend kann eine Speicherordnungsschaltung (z. B. eine RAF in **Fig. 11A)** das Planen neuer Speicheroperationen anhalten, bis die Abhängigkeits-Tokenzähler **1014** ungesättigt werden.

[0066] Als ein Beispiel für eine Last geht eine Adresse in der Warteschlange **1022** ein, die der Planer **1012** mit einer Last in **1010** in Übereinstimmung bringt. Ein Abschlusspufferslot für diese Last wird in der Reihenfolge zugeordnet, in der die Adresse eingegangen ist. Unter der Voraussetzung, dass diese bestimmte Last in dem Graphen keine spezifizierten Abhängigkeiten aufweist, werden die Adresse und der Abschlusspufferslot durch den Planer (z. B. durch die Anweisung durch den Speicher **1042**) aus dem Speichersystem gesendet. Wenn das Ergebnis zum Mux **1040** zurückkehrt (schematisch dargestellt), wird dieses in dem Abschlusspufferslot, das dieser spezifiziert, gespeichert (z. B. wenn es den Zielslot vollständig entlang durch das Speichersystem getragen hat). Der Abschlusspuffer sendet die Ergebnisse zurück in das lokale Netzwerk (z. B. lokales Netzwerk **1002**, **1004**, **1006** oder **1008**), in der Reihenfolge, in der die Adressen eingegangen sind.

[0067] Die Speicher können ähnlich sein, außer dass sowohl die Adresse als auch die Daten eingehen müssen, bevor irgendeine Operation an das Speichersystem ausgesendet wird.

Cache

[0068] Datenflussgraphen können eine Profusion von Anfragen (z. B. Wortgranularität) parallel erzeugen. Daher stellen bestimmte Ausführungsformen des CSA in Cache-Untersystem mit ausreichender Bandbreite bereit, um den CSA zu bedienen. Eine stark gestapelte Cache-Mikroarchitektur, wie sie beispielsweise in **Fig. 11A** gezeigt ist, kann benutzt werden. **Fig. 11A** veranschaulicht eine Schaltung **1100** mit mehreren Abfrage-Adressdatei (RAF)-Schaltungen (z. B. RAF-Schaltung (**1**)), die zwischen mehreren Beschleuniger-Kacheln (**1108**, **1110**, **1112**, **1114**) und mehreren Cache-Bänken (z. B. Cache-Bank **1102**) gemäß Ausführungsformen der Offenbarung gekoppelt sind. In einer Ausführungsform kann die Anzahl der RAFs und Cache-Bänke in einem Verhältnis von entweder 1:1 oder 1:2 sein. Cache-Bänke können volle Cache-Zeilen enthalten (z. B. im Gegensatz zur gemeinsamen Nutzung durch Wort), wobei jede Zeile genau eine Heimat im Cache hat. Cache-Zeilen können über eine Pseudozufallsfunktion auf Cache-Bänke abgebildet werden. Der CSA kann das SVM-Modell zur Integration mit anderen gekachelten Architekturen übernehmen. Bestimmte Ausführungsformen weisen ein ACI-Netzwerk (ACI - Accelerator Cache Interconnect, Beschleuniger-Cache-Zwischenverbindung), das die RAFs mit den Cache-Bänken verbindet. Dieses Netzwerk kann die Adresse und Daten zwischen den RAFs und dem Cache übertragen. Die ACI-Topologiekann eine kaskadierte Crossbar sein, z. B. als ein Kompromiss zwischen Latenz- und Implementierungskomplexität.

[0069] Fig. 11B veranschaulicht einen Transaktionsmechanismus, in dem Cache-Zeilen mit Informationen über die Quelle eines Lese- oder Schreibzugriffs markiert sind, gemäß Ausführungsformen der Erfindung. Da Lese- und Schreibgeräte verteilt sind, werden Sätze von Lese- und Schreibgeräten in Transaktionsklassen gruppiert, die keinen Transaktionsfehlschlag induzieren. Die Struktur initiiert und beendet Transaktionen durch eine Sondernachricht.

[0070] Fig. 11B zeigt die Architektur auf Systemebene der Transaktionsschnittstelle. Zu der Kompilationszeit werden Speicherzugriffsstreams mit einem bestimmten Transaktionsidentifizierer in Zusammenhang gebracht. Dies ermöglicht, dass größere Teile der Struktur an derselben Transaktion arbeiten. Zur Laufzeit beginnen Transaktionen durch Senden einer Nachricht zu der Transaktionssteuerung, die die Transaktion als aktiv markiert. Nachfolgende Abfragen von den Transaktionszugangseinrichtungen werden als mit der aktiven Transaktion im Cache in Zusammenhang gebracht markiert.

[0071] Die Transaktion wird durch Senden einer anderen Nachricht zu der Steuerung abgeschlossen, die Markierungsbits im Cache löscht. Wenn keine Konflikte aufgetreten sind, wird die Transaktion erfolgreich abgeschlossen und die Struktur wird informiert. Wenn jedoch ein Konflikt aufgetreten ist, wird die Struktur über einen Fehlschlag informiert. Jegliche Cache-Aktualisierungen werden zurückgesetzt, um den Zustand vor der Transaktion wiederherzustellen.

[0072] Optional wird bei einem Fehlschlag eine Software aufgerufen, um den potentiellen Konflikt zu reparieren. Die Software kann reagieren, indem eine weniger parallele Version des Programms, zum Beispiel eine sequenzielle Version auf einem Kern zu dem Punkt eines vorherigen sicheren Prüfpunkts, ausgeführt wird.

[0073] In Ausführungsformen schließt dieser Mechanismus einen Begriff von Checkpointing und eine Benachrichtigung über den Teil der Struktursoftware ein. Die Struktur beginnt einen Prüfpunkt und fährt mit der Ausführung fort. Zum Ende der Transaktion muss die Struktur möglicherweise den Speicher synchronisieren, zum Beispiel durch das Ausgeben von Speicher-Fences. Eine Erweiterung des Basismechanismus schließt ein Überwachen der Festschreibung von transaktionsbezogenen Aktivitäten in der Hardware ein.

[0074] Von einem Transaktionsmechanismus gemäß Ausführungsformen der Erfindung kann gewünscht sein, dass er atomare Operationen unterstützt, die Operationen sind, in denen ein Speicherort gelesen wird, der Wert modifiziert wird und dann der neue Wert zurück in denselben Speicherort gespeichert wird. Dies wird „atomar“ durchgeführt, was bedeuten soll, dass kein anderer Agent, der am Speicher agiert, auch denselben Lesewert zum Bearbeiten verwenden kann. Sie müssen entweder einen vorherigen Wert oder den resultierenden Wert dieser atomaren Aktion verwenden. Mit anderen Worten, wenn mehrere Agenten jeweils versuchen, einen Speicherort zu inkrementieren, dann wird jeder der individuellen Werte nur durch einen Agenten, der am Speicher agiert, erzeugt.

[0075] Da Arithmetik an Datenwerten in der CSA-Struktur durchgeführt wird, ermöglicht diese Lösung, dass die Modifikation an dem Datenwertteil der atomaren Operation in der räumlichen Struktur stattfindet. Ausführungsformen können einschließen:

- Ausgeben einer Ladeoperation zu dem Cache, die anzeigt, dass eine atomare Operation an den Daten an dem spezifizierten Ort initiiert wird. Der Wert der Daten wird zusammen mit einem Signal, das anzeigt, ob eine atomare Operation erfolgreich initiiert wurde oder nicht, zurückgeleitet. Eine atomare Operation könnte nicht erfolgreich initiiert werden, wenn eine andere atomare Operation durch einen anderen Agenten, der am Speicher agiert, schon im Gange ist.
- Annehmen, dass die atomare Operation initiiert wurde, dann werden die Daten durch die geeigneten Rechenelemente im Graphen modifiziert.
- Ausgeben einer Speicheroperation zu dem Cache mit den modifizierten Daten. Diese Speicheroperation ist eine bedingte Speicherung, die nur erfolgreich sein wird, wenn der Speicherort für die gesamte Zeitspanne von der Ladung zur Speicherung unter der Cache-Kohärenzsteuerung des CSA-Cache gestanden hat. Wenn er unter der Steuerung steht, dann wird die Speicherung durchgeführt und eine Erfolgsangabe wird zurückgeliefert, andernfalls wird eine Speicherungsfehlschlagangabe zurückgeliefert.

[0076] Gemäß diesem Ansatz wird eine Sperre-im-Gange-Adresse an jeder Cache-Bank im Cache gehalten. Sie wird als frei analysiert, wenn die anfängliche atomare Ladeoperation auftritt und dann geschrieben wird, sodass sie durch die atomare Ladung gehalten wird. Sie wird durch eine beliebige Cache-Operation, die in den Cache schreibt, oder eine beliebige Operation, die die Cache-Zeile aus dem Cache entfernt, sodass sie durch einen Agenten geschrieben wird, untersucht. Wenn eines dieser beiden Ereignisse stattfindet, dann wird der Sperre-im-Gange-Adressen-Latch ungültig gemacht. Die resultierende atomare Speicherung wird fehlschlagen, wenn ein anderes Schreibgerät für den Ort in der Zwischenzeit zwischen dem Eintreffen der atomaren Ladung und dem Eintreffen der atomaren Speicherung erschienen ist. Es sei zu beachten, dass das Sperre-im-Gange-Adressregister auch ein Timeout aufweisen wird.

[0077] **Fig. 11C** bis **Fig. 11J** veranschaulicht eine Unterstützung für Backup und Wiederholung unter Verwendung von Epochen im Cache-/Speicheruntersystem gemäß Ausführungsformen der Erfindung. In konventionellen Computer-Pipelines, die einen Befehlszeiger zum Anzeigen eines Strings von Befehlen verwenden, gibt es typischerweise ein Zeitfenster, das von der Initiierung der Arbeit an einem Befehl bis zur Zurückziehung eines Befehls live gehalten werden. In dem Fenster befinden sich viele Befehle in-flight. Während dieses Fensters kann die Pipeline immer zu einem beliebigen Live-Befehl im Fenster zurückgesetzt werden, und dies wird für Ereignisse wie Zweigprädiktionsfehler und andere spekulative Aktionen, die korrekt erneut ausgeführt werden müssen, durchgeführt. Es gibt jedoch weder einen zentralen Steuermechanismus mit Graphausführung noch eine Befehlszeigerangabe, damit eine Erzeugung für ein Arbeitsfenster ermöglicht wird. Ein Backup-Ansatz, der zweckmäßig ist, periodisch Speicherauszüge zu erstellen, die einen Zeitpunkt bei der Ausführung des Graphen repräsentieren und zu denen zurückgesetzt werden kann. Die Zeit von einem solchen Speicherauszug zum nächsten wird in dieser Offenbarung eine „Epoche“ genannt. Ausführungsformen der Erfindung schließen Mechanismen ein, die in der Cache-Hierarchie durchgeführt werden, um die Ausführung mit Speicherauszügen und die Fähigkeit zum Unterstützen eines Backups zu dem jüngsten Speicherauszug zu unter-

stützen. Ein Ansatz mit gestaffelter Extraktion kann in Verbindung mit dem Erstellen von Speicherauszügen in der räumlichen Struktur verwendet werden.

[0078] Die Unterstützung für die Ausführung von Epochen im Cache-/Speichersystem kann einschließen:

- Akkumulieren von Speicherschreibzugriffen, die seit der Initiierung der Epoche aufgetreten sind.
- Bewahren der alten Werte (Datenwert, der zum Start der Epoche vorhanden war), sodass in dem Fall, dass das System zum Start der Epoche zurückgesetzt werden muss, alle ursprünglichen Datenwerte verfügbar sind.
- Steuern der Sichtbarkeit von Änderungen an Speicherdatenwerten in der Mitte einer Epoche von sichtbar für alle anderen Agenten im Cache-Kohärenzsystem.
- Erhalten der Erlaubnis, die eindeutige Kopie einer Zeile zu besitzen, vom Kohärenzprotokoll während der Ausführung der Epoche.
- Am Ende einer Epoche, Sichtbarmachen aller Änderungen, die in der Epoche auftraten, für alle Agenten in einer atomaren Aktion.

[0079] **Fig. 11C** zeigt den Plan, wie eine Epochenunterstützung im Cache-/Speichersystem erreicht werden kann. Neue Werte werden in Cache-Einträgen akkumuliert, während alte Werte, die vor dem Start der Epoche vorhanden waren, als sich im Speicher befindlich garantiert werden. Falls eine Rücksetzung zum Start der Epoche erforderlich ist, werden alle neuen Werte ungültig gemacht. Wenn eine erfolgreiche Bewegung zur nächsten Epoche stattfindet, werden dem Cache-Kohärenzprotokoll alle neuen Werte als sichtbar deklariert.
Bei **Fig. 11C** werden beispielsweise in Schritt **C1** alle neu erstellten Werte **W1** im Cache **CA** gehalten, während Epoche n+1 ausgeführt wird. Falls eine Rücksetzung benötigt wird, werden beispielsweise alle Werte der Epoche n+1 ausgelöscht, um die Werte der Epoche n verfügbar zu machen. Anschließend folgt das Übergehen zu Epoche n+2, umfassend Sichtbarmachen aller erzeugten Werte der Epoche n+1 für das Cache Kohärenzprotokoll, Starten des Akkumulierens von erzeugten Werten der Epoche n+2 in den Cache **CA** und, wie benötigt, Verschieben der Werte der Epoche n+1 in den Speicher **SP**. Für jene geschriebenen Werte **W2** die zum Start der Epoche n+1 live waren und nicht im Cache **CA** verbleiben können, folgt gemäß Schritt **C2**, Sicherstellen, dass eine Kopie im Speicher **SP** ist.

[0080] Das Cache-Kohärenzprotokoll kann drei Operationen zum Unterstützen der Epochenausführung einschließen. Die erste derartige Operation (Speicherabgleich-Schreibzugriff) ist ein Verfahren des Sicherstellens, dass der Speicherwert mit einem im Cache gehaltenen Wert übereinstimmt, wie in **Fig. 11D** gezeigt. Diese Operation ändert einen Cache-Kohärenzzustand nicht; sie stellt lediglich sicher, dass die Daten aus einem Cache-Eintrag in den Speicher geschrieben werden.
Bei **Fig. 11D** wird beispielsweise der Speicher **SP** an den Cache-Wert „Speicherableich-Schreibzugriff“ angepasst und der Cache-Zustand unverändert belassen und die Tag Verzeichnisse **T** unverändert belassen. Dies stellt im Wesentlichen nur eine Buchführungsoperation dar, wobei das Bezugszeichen **D1** eine Zeile in Besitz im Cache veranschaulicht. In unseren gegenwärtigen Protokollen würde diese neue Option unter Verwendung eines Zugmodells implementiert werden (Anfragen zum Verschieben, Warten auf eine Zuganfrage).

[0081] Die zweite Protokolloperation (Besitzfreigabe-Keine-Daten) ist ein Verfahren zum Freigeben des Besitzes einer Zeile und Spezifizieren, dass die Speicherkopie der gegenwärtige zu verwendende Wert ist, wie in **Fig. 11E** gezeigt. Diese Operation ändert den Cache-Kohärenzsteuerzustand im Tag-Verzeichnis, sodass er nicht zu dem Cache als den Ort zum Erhalten von Daten zeigt, sondern sie stattdessen aus dem Speicher zu bekommen.
Bei **Fig. 11E** wird beispielsweise der Besitz der Zeile abgegeben, ohne Daten bereitzustellen (nachfolgende Anforderer würden eine Speicherkopie verwenden; „Besitzfreigabe-Keine-Daten“). Das Tag-Verzeichnis **T** entfernt nur den Datensatz der gecachten Zeile in Besitz, daher ist der Speicher jetzt der Bereitsteller von Daten, wobei das Bezugszeichen **E1** eine Zeile in Besitz im Cache veranschaulicht. Dies liegt sehr nahe an einer Operation einer „sauberen Räumung“ in unseren gegenwärtigen Protokollen.

[0082] Die dritte Operation, in **Fig. 1F** veranschaulicht, ist die Fähigkeit, auf eine Untersuchung, die in diesem Cache nach Daten sucht, zu reagieren und anzuzeigen, dass es keine Daten gibt, und stattdessen die Speicherkopie zu erhalten. Da es gewünscht ist, die Epochen auf eine atomare Art und Weise auszuführen, sodass für jeden Beobachter entweder die gesamten Epochenänderungen stattgefunden haben oder keine Änderungen stattgefunden haben, wird durch diese Operation eine Weise des Handhabens von Untersuchungen von anderen Beobachtern bereitgestellt.

Fig. 11F veranschaulicht die Fähigkeit, einer Untersuchung **U** zu sagen, dass der Cache **CA** tatsächlich nicht die Daten, die man haben will, aufweist, und stattdessen die Kopie im Speicher **SP** verwendet, wobei das Bezugszeichen **F1** eine Zeile in Besitz im Cache veranschaulicht. In gegenwärtigen Protokollen würde dies eine RSPI-Antwort sein. In Laufbedingung mit einem „Speicherabgleich-Schreibzugriff“ würde die Untersuchung entweder den noch nicht aus dem Cache **CA** gezogenen Wert verwenden oder den RSPI empfangen und den Wert aus dem Speicher **SP** holen.

[0083] Ein zusätzlicher Cache-Zustand, in **Fig. 11G** veranschaulicht, kann zum Unterscheiden zwischen kohärenten Zeilen in Besitz gegenüber spekulativen Zeilen in Besitz bereitgestellt werden. Diese Metadaten in Cache-Kohärenzprotokoll-Unterstützungsstrukturen unterscheiden nicht zwischen den beiden unterschiedlichen Besitz-Zuständen und zeichnen nur auf, dass dieser Cache der Besitzer der Zeile ist.
Fig. 11G veranschaulicht daher zwei Zustände **Z1** und **Z2** im Cache **CA**, aber beide sind der in-Besitz-Zustand, soweit es das Tag-Verzeichnis betrifft, wobei Zustand **Z1** der „Spekulativ in Besitz“ Zustand und Zustand **Z2** der „in Besitz“ Zustand ist.

[0084] Als Nächstes, um die Epochenunterstützung zu veranschaulichen, zeigt **Fig. 11H** einen Schreibzugriff von der räumlichen Struktur in den Cache, wie in **Fig. 11 H** gezeigt. Wenn der Schreibzugriff auf eine spekulative Zeile in Besitz trifft, dann wird die Zeile aktualisiert. Wenn der Schreibzugriff eine reguläre Zeile in Besitz trifft, dann wird zuerst mit einer „Speicherabgleich-Schreibzugriff“-Operation veranlasst, dass die Zeile mit dem Speicher übereinstimmt. Dann wird die Zeile zu spekulativ in Besitz und aktualisiert abgeändert. Wenn der Schreibzugriff keine In-Besitz-Version der Zeile findet, dann wird dem Cache-Kohärenzprotokoll eine „Anfrage für den Besitz“(RFO)-Anweisung gesendet. Beim Empfang der Füllung und somit des Besitzes der Zeile wird die Quelle der Füllung untersucht. Wenn sie vom Speicher kam, dann ist der Speicher schon auf dem neuesten Stand und es wird keine „Speicherabgleich-Schreibzugriff“-Anweisung ausgegeben. Wenn sie nicht vom Speicher kam, dann wird eine „Speicherabgleich-Schreibzugriff“-Anweisung ausgegeben. Danach wird die Zeile zu spekulativ in Besitz und aktualisiert abgeändert.

[0085] Es sei zu beachten, dass, wenn die Schreibanfrage den Cache verfehlt und eine Zeile im Cache zuweisen muss, es Platz für die neue Zeile geben muss, ohne eine spekulative Zeile zu räumen. Ein Ansatz zum Handhaben von diesem besteht darin, dass, wenn die Zuweisung für die letzte Zeile im Satz ist, der nicht spekulativ in Besitz ist, dann ein Epochenende deklariert wird und der Transfer von sowohl der räumlichen Struktur als auch des Cache zur nächsten Epoche initiiert wird. Eine beliebige Schwelle würde zum Initiieren des nächsten Speicherauszugs funktionieren.

[0086] **Fig. 11H** veranschaulicht mehrere Schritte, zuerst, Finden einer Zeile in Besitz oder Erhalten von einer, falls notwendig (RFO) **H1**, zweitens, Durchführen eines „Speicherabgleich-Schreibzugriffs“ von ursprünglichen Daten in dem Speicher und drittens, Abändern des Cache-Zustands zu „spekulativ in Besitz“ und Durchführen des Schreibzugriffs SZ. Es sei zu beachten, dass, wenn erforderlich, RFO und Füllung aus Speicher kommen, „Speicherabgleich-Schreibzugriff“ nicht durchgeführt werden muss. Wenn die Zeile schon „spekulativ in Besitz“ ist, dann folgt nur Durchführen **H2** des Schreibzugriffs SZ. Die nächste Aktion ist der Übergang von einer Epoche zur nächsten. Die Cache-Kohärenzprotokoll-Metadaten in den Tag-Verzeichnissen zeichnen auf, dass dieser Cache alle Zeilen besitzt, die in dieser Epoche geschrieben werden. Daher wird eine Flash-Änderungsoperation durchgeführt, um den Kohärenzzustand zu ändern, der im Cache für alle Zeilen, die spekulativ in Besitz oder nur in Besitz sind, aufgezeichnet ist, wie in **Fig. 11I** gezeigt. Keine Aktion wird durch die Cache-Kohärenz-Metadaten benötigt, die sich in den Tag-Verzeichnissen befinden.
Fig. 11I veranschaulicht im Endeffekt eine Flash-Löschoperation, die alle „spekulativ in Besitz“-Zustände zu „in Besitz“ ändert, d.h. alle Schreibzugriffe **SZ** sind jetzt für das Cache-Kohärenzprotokoll sichtbar 11.

[0087] Die letzte Aktion bei der Unterstützung der Epochenausführung ist ein Mechanismus zum Zurücksetzen der Cache-/Speicherhierarchie zu dem Punkt des jüngsten Epochenstarts (dem Speicherauszug). Dies ist in **Fig. 11J** veranschaulicht. Der Ansatz besteht darin, alle spekulativen Zeilen in Besitz ungültig zu machen. Dies kann mit einer Flash-Löschoperation durchgeführt werden. Nun, obwohl die Cache-Kohärenz-Metadaten diese Zeilen als durch diesen Cache zu besitzend aufzeichnen, wenn eine Untersuchung stattfindet, besteht die Antwort darin, die Daten im Speicher zu finden.

[0088] **Fig. 11J** zeigt eine zusätzliche Leistungsverbesserung, die zwei ungültige Zeilentypen benötigen würde, eine reguläre ungültige Zeile und eine Flash-ungültig gemachte Zeile. Beide würden als ungültige Zeilen behandelt werden, mit der Ausnahme eines Demon, der läuft und den Cache nach Flash-ungültig gemachten Zeilen durchkämmt. Wenn diese gefunden werden, wird er „Besitzfreigabe-Keine-Daten“-Anweisungen ausgeben, um sicherzustellen, dass die Cache-Kohärenz-Metadaten aufzeichnen, dass der Speicher der Ort ist,

um die Daten zu erhalten, und nicht dieser Cache. Diese Zeilen würden dann zu dem regulären ungültigen Zustand geändert werden.
Wir können alle „spekulativ in Besitz“-Zeilen Flash-ungültig machen und das RSPI-Merkmal zum Säubern jeglicher Kommunikation von anderen Beobachtern verwenden, was sie zwingen wird, die Speicherkopie zu bekommen. Wenn die Struktur eine Zeile erneut berühren will, muss sie eine RFO zu dem Kohärenzsystem ausgeben, das die Daten aus dem Speicher holen wird. Eine zusätzliche Leistungsverbesserung könnte sein, zwei ungültige Zustände aufzuweisen und einen Demon herumlaufen zu lassen und jegliche Flash-ungültig gemachte Zeilen in wahrlich ungültige Zeilen sauber zu räumen.

Gleitkomma-Unterstützung

[0089] Bestimmte HPC-Anwendungen sind durch ihre Bedarf an einer signifikanten Gleitkomma-Bandbreite gekennzeichnet. Um diesen Bedarf zu erfüllen, können Ausführungsformen eines CSA mit einer Vielzahl (z. B. jeweils zwischen 128 und 256) von Gleitkomma-Additions- und Multiplikations-PEs bereitgestellt werden, z. B. in Abhängigkeit von der Kachelkonfiguration. Ein CSA kann einige andere erweiterte Genauigkeitsmodi bereitstellen, z. B. zum Vereinfachen der Mathematikbibliothek-Implementierung. CSA-Gleitkomma-PEs können sowohl einzelne als auch doppelte Genauigkeit unterstützen, aber PEs mit geringerer Genauigkeit können die Maschinenlern-Arbeitslasten unterstützen. Ein CSA kann eine um eine Größenordnung höhere Gleitkomma-Leistung als ein Prozessorkern bereitstellen. In einer Ausführungsform wird zusätzlich zur Erhöhung der Gleitkomma-Bandbreite die Energie, die in Gleitkomma-Operationen verbraucht wird, reduziert, um alle Gleitkomma-Einheiten zu versorgen. Zum Reduzieren von Energie kann ein CSA selektiv die Bits niedriger Ordnung der Gleitkomma-Multiplizierer-Anordnung durchschalten. Bei der Untersuchung des Verhaltens der Gleitkomma-Arithmetik beeinflussen die Bits niedriger Ordnung des Multiplikationsarrays oft nicht das endgültige, gerundete Produkt. **Fig. 12** veranschaulicht einen Gleitkomma-Multiplizierer **1200**, der in drei Gebiete (Ergebnisgebiet, drei potentielle Übertraggebiete (**1202**, **1204**, **1206**) und Gebiet mit Gate) gemäß Ausführungsformen der Offenbarung unterteilt ist. In bestimmten Ausführungsformen beeinflusst das Übertraggebiet wahrscheinlich das Ergebnisgebiet, und es ist unwahrscheinlich, dass das Gebiet mit Gate das Ergebnisgebiet beeinflusst. Bei der Betrachtung eines Gebiets mit Gate aus g Bits kann der maximale Übertrag wie folgt sein:

$$\begin{aligned} carry_g &\leq \frac{1}{2^g}\sum_1^g i2^{i-1} \\ &\leq \sum_1^g \frac{i}{2^g} - \sum_1^g \frac{i}{2^g} + 1 \\ &\leq g - 1 \end{aligned}$$

Wenn bei diesem maximalen Übertrag das Ergebnis des Übertragsgebiets kleiner als 2c - g ist, wobei das Übertraggebiet c Bits breit ist, kann das Gebiet mit Gate ignoriert werden, da es das Ergebnisgebiet nicht beeinflusst. Das Erhöhen von g bedeutet, dass es wahrscheinlicher ist, dass das Gebiet mit Gate benötigt wird, während das Erhöhen von c bedeutet, dass das Gebiet mit Gate bei zufälliger Annahme unbenutzt bleibt und deaktiviert werden kann, um einen Energieverbrauch zu vermeiden. In Ausführungsformen eines CSA-Gleitkomma-Multiplikations-PE wird ein zweistufiger Pipeline-Ansatz benutzt, bei dem zuerst das Übertraggebiet bestimmt wird und dann das Gebiet mit Gate bestimmt wird, wenn festgestellt wird, dass dies das Ergebnis beeinflusst. Wenn mehr Information über den Kontext der Multiplikation bekannt ist, stimmt ein CSA die Größe des Gebiets mit Gate aggressiver ab. Bei einer FMA kann das Multiplikationsergebnis zu einem Akkumulator addiert werden, der oft viel größer als jeder der Multiplikanden ist. In diesem Fall kann der Addend-Exponent vor der Multiplikation beobachtet werden, und die CSDA kann das Gebiet mit Gate entsprechend einstellen. Eine Ausführungsform des CSA weist ein Schema auf, in dem ein Kontextwert, der das Mindestergebnis einer Berechnung bindet, zugehörigen Multiplizierern bereitgestellt wird, um Minimum-Energie-Gating-Konfigurationen auszuwählen.

Laufzeitdienste

[0090] In einer bestimmten Ausführungsform weist ein CSA eine heterogene und verteilte Struktur auf, und folglich müssen Laufzeitdienst-Implementierungen mehrere Arten von PEs in einer parallelen und verteilten Weise aufnehmen. Obwohl Laufzeitdienste in einem CSA kritisch sein können, können sie im Vergleich zur Berechnung auf Benutzerebene infrequent sein. Bestimmte Implementierungen konzentrieren sich daher auf das Überlagern von Diensten über Hardwareressourcen. Zur Erreichung dieser Ziele können CSA-Laufzeitdienste als eine Hierarchie angegeben werden, wobei z. B. jede Schicht einem CSA-Netzwerk entspricht. Auf Kachel-Level kann eine einzelne nach außen weisende Steuerung Dienstbefehle an einen zugeordneten

Kern mit der CSA-Kachel akzeptieren oder senden. Eine Kachel-Level-Steuerung kann dazu dienen, regionale Steuerungen an den RAFs zu koordinieren, z. B. unter Verwendung des ACI-Netzwerks. Im Gegenzug können regionale Steuerungen lokale Steuerungen an bestimmten Mezzanine-Stopps im Netzwerk koordinieren. Auf dem untersten Level können dienstspezifische Mikroprotokolle über das lokale Netzwerk ausgeführt werden, z. B. während eines speziellen Modus, der durch die Mezzanine-Steuerung gesteuert wird. Die Mikroprotokolle können zulassen, dass jedes PE (z. B. PE-Klasse nach Typ) mit dem Laufzeitdienst gemäß eigenem Bedarf interagiert. Die Parallelität ist somit in dieser hierarchischen Organisation implizit und Operationen auf geringstem Level können gleichzeitig stattfinden. Diese Parallelität kann die Konfiguration einer CSA-Kachel zwischen Hunderten von Nanosekunden bis einigen Mikrosekunden ermöglichen, z. B. abhängig von der Konfigurationsgröße und ihrem Speicherplatz in der Speicherhierarchie. Ausführungsformen des CSA nutzen somit die Eigenschaften von Datenflussgraphen, um die Implementierung jedes Laufzeitdienstes zu verbessern. Eine Hauptbeobachtung ist, dass Laufzeitdienste möglicherweise nur eine legale logische Ansicht des Datenflussgraphen bewahren müssen, z. B. einen Zustand, der durch eine gewisse Reihenfolge von Ausführungen des Datenflussoperators erzeugt werden kann. Dienste brauchen im Allgemeinen keine temporäre Ansicht des Datenflussgraphen garantieren, z. B. den Zustand eines Datenflussgraphen in einem CSA zu einem spezifischen Zeitpunkt. Dies kann es dem CSA ermöglichen, die meisten Laufzeitdienste auf verteilte, zeitverschachtelte und parallele Weise durchzuführen, z. B. unter der Voraussetzung, dass der Dienst abgestimmt ist, um die logische Ansicht des Datenflussgraphen zu bewahren. Das lokale Konfigurationsmikroprotokoll kann ein paketbasiertes Protokoll sein, das dem lokalen Netzwerk überlagert ist. Konfigurationsziele können in einer Konfigurationskette organisiert sein, die z. B. in der Mikroarchitektur festgelegt ist. Strukturziele (z. B. PE-Ziele) können einzeln konfiguriert werden, z. B. unter Verwendung eines einzigen zusätzlichen Registers pro Ziel, um eine verteilte Koordination zu erreichen. Zum Starten der Konfiguration kann eine Steuerung ein Außer-Band-Signal ansteuern, das alle Strukturziele in seiner Nachbarschaft in einen nicht konfigurierten, pausierten Zustand versetzt und Multiplexer in dem lokalen Netzwerk zu einer vordefinierten Konformation schwingt. Wenn die Strukturziele (z. B. PE-Ziele) konfiguriert sind, d. h. sie vollständig ihr Konfigurationspaket empfangen, können sie ihre Konfigurations-Mikroprotokoll-Register setzen und das unmittelbar nachfolgende Ziel (z. B. PE), das als nächstes konfiguriert wird, unter Verwendung des nachfolgenden Pakets mitteilen. Es gibt keine Beschränkung für die Größe eines Konfigurationspakets, und Pakete können eine dynamisch variable Länge aufweisen. Zum Beispiel können PEs, die konstante Operanden konfigurieren, ein Konfigurationspaket aufweisen, das verlängert wird, um das konstante Feld (z. B. X und Y in **Fig. 3B** - **Fig. 3C)** aufzuweisen. **Fig. 13** veranschaulicht eine in-Flight-Konfiguration eines Beschleunigers **1300** mit mehreren Verarbeitungselementen (z. B. PEs **1302**, **1304**, **1306**, **1308**) gemäß Ausführungsformen der Offenbarung. Nach der Konfiguration können die PEs den Gegenstand der Datenflusseinschränkungen ausführen. Kanäle, die unkonfigurierte PEs beinhalten, können jedoch durch die Mikroarchitektur deaktiviert werden, z. B. verhindern, dass undefinierte Operationen stattfinden. Diese Eigenschaften ermöglichen Ausführungsformen eines CSA das verteilte Initialisieren und Ausführen ohne jegliche zentralisierte Steuerung. Aus einem unkonfigurierten Zustand kann die Konfiguration vollständig parallel stattfinden, z. B. in vielleicht nur 200 Nanosekunden. Aufgrund der verteilten Initialisierung von Ausführungsformen eines CSA können PEs jedoch aktiv werden, zum Beispiel Anforderungen an den Speicher senden, lange bevor die gesamte Struktur konfiguriert ist. Die Extraktion kann ähnlich wie die Konfiguration ablaufen. Das lokale Netzwerk kann angepasst werden, um Daten von jeweils einem Ziel gleichzeitig zu extrahieren, und es werden Zustandsbits verwendet, um eine verteilte Koordination zu erreichen. Ein CSA kann die Extraktion so steuern, dass sie nicht-destruktiv ist, d. h. nach Abschluss der Extraktion ist jedes extrahierbare Ziel in seinen Ausgangszustand zurückgekehrt. In dieser Implementierung kann der gesamte Zustand in dem Ziel in einem abtastungsähnlichen Ausgangsregister, das an das lokale Netzwerk gebunden ist, zirkuliert werden. Es kann jedoch eine In-Place-Extraktion erreicht werden, indem neue Pfade auf der Registerübertragungsebene (RTL) eingefügt werden oder bestehende Leitungen verwendet werden, um die gleichen Funktionen mit geringerem Aufwand bereitzustellen. Wie bei der Konfiguration wird die hierarchische Extraktion parallel dazu erreicht.

[0091] Fig. 14 veranschaulicht einen Speicherauszug **1400** einer zeitverschachtelten in-Flight-Extraktion gemäß Ausführungsformen der Offenbarung. In einigen Verwendungsfällen der Extraktion, wie Checkpointing, ist die Latenz möglicherweise kein Problem, solange der Strukturdurchsatz beibehalten wird. In diesen Fällen kann die Extraktion auf zeitverschachtelt erfolgen. Diese Anordnung, die in **Fig. 14** gezeigt ist, erlaubt es dem größten Teil der Struktur, weiter auszuführen, während ein schmales Gebiet für die Extraktion deaktiviert ist. Konfiguration und Extraktion können koordiniert und zusammengesetzt werden, um eine Umschaltung im zeitverschachtelten Kontext zu erreichen. Ausnahmen können sich qualitativ von der Konfiguration und Extraktion darin unterscheiden, indem sie nicht zu einem spezifischen Zeitpunkt auftreten, sondern irgendwo in der Struktur zu jedem Zeitpunkt während der Laufzeit. Dementsprechend kann das Ausnahme-Mikroprotokoll in einer Ausführungsform nicht dem lokalen Netzwerk überlagert werden, das zur Laufzeit durch das Benutzerprogramm belegt ist, und benutzt sein eigenes Netzwerk. Ausnahmen sind jedoch von Natur aus selten und

insensitiv gegenüber Latenz und Bandbreite. Daher verwenden bestimmte Ausführungsformen des CSA ein paketvermitteltes Netzwerk, um Ausnahmen zu dem lokalen Mezzanine-Stopp zu übertragen, wo sie z. B. in der Diensthierarchie nach oben weitergeleitet werden (wie z. B. in **Fig. 29**). Pakete in dem lokalen Ausnahme-Netzwerk können extrem klein sein. In vielen Fällen reicht eine PE-Identifikation (ID) von nur zwei bis acht Bits als ein vollständiges Paket aus, da z. B. der CSA eine eindeutige Ausnahmekennung erzeugen kann, wenn das Paket die Ausnahmediensthierarchie durchläuft. Ein solches Schema kann wünschenswert sein, weil es auch den Bereichsaufwand zum Erzeugen von Ausnahmen an jedem PE reduziert.

KOMPILIERUNG

[0092] Die Fähigkeit, in Hochsprachen geschriebene Programme in einem CSA zu kompilieren, kann für die Übernahme in der Branche von entscheidender Bedeutung sein. Dieser Abschnitt gibt einen allgemeinen Überblick über Kompilierungsstrategien für Ausführungsformen eines CSA. Zunächst wird ein CSA-Softwareframework vorgeschlagen, das die gewünschten Eigenschaften einer idealen Toolkette für die Produktionsqualität veranschaulicht. Danach wird ein Prototyp-Kompilierer-Framework erläutert. Dann wird eine „Control-to-Data-flow-Conversion (Steuerung-zu-Datenfluss-Umwandlung)“ besprochen, z. B. um gewöhnlichen sequentiellen Steuerflusscode in CSA-Datenfluss-Assembliercode umzuwandeln.

Beispiel Produktionsframework

[0093] Fig. 15 veranschaulicht eine Kompilierungs-Toolkette **1500** für einen Beschleuniger gemäß Ausführungsformen der Offenbarung. Diese Toolkette kompiliert Hochsprachen (wie C, C ++ und Fortran) in eine Kombination aus Host-Code- (LLVM) - Zwischenrepräsentation (IR) für die spezifischen Gebiete, die beschleunigt werden sollen. Der CSA-spezifische Teil dieser Kompilierungs-Toolkette verwendet LLVM IR als Eingabe, optimiert und kompiliert diese IR in eine CSA-Assembly, z. B. durch Hinzufügen einer geeigneten Pufferung auf latenzinsensitiven Kanälen für die Leistung. Anschließend platziert und routet er die CSA-Assembly auf der Hardware-Struktur und konfiguriert die PEs und das Netzwerk für die Ausführung. In einer Ausführungsform unterstützt die Toolkette die CSA-spezifische Kompilierung als Just-in-Time (JIT), wobei potentielle Laufzeitrückmeldungen von tatsächlichen Ausführungen einbezogen werden. Eine der wichtigsten Ausgestaltungseigenschaften des Frameworks ist die Kompilierung von (LLVM) IR für den CSA anstelle der Verwendung einer höheren Sprache als Eingabe. Während ein Programm, das in einer höheren Programmiersprache geschrieben wurde, die speziell für den CSA ausgestaltet wurde, maximale Leistung und/oder Energieeffizienz erreichen kann, kann die Übernahme neuer Hochsprachen oder Programmier-Frameworks in der Praxis aufgrund der Schwierigkeit der Umwandlung existierender Codegrundlagen langsam sein. Die Verwendung von (LLVM) IR als Eingabe ermöglicht vielen existierenden Programmen, möglicherweise auf einem CSA ausgeführt zu werden, ohne dass z. B. eine neue Sprache geschaffen werden muss oder das Frontend neuer Sprachen, die auf dem CSA ausgeführt werden sollen, signifikant zu modifizieren.

Prototyp-Kompilierer

[0094] Fig. 16 veranschaulicht einen Kompilierer **1600** für einen Beschleuniger gemäß Ausführungsformen der Offenbarung. Der Kompilierer **1600** konzentriert sich anfänglich auf die vorzeitige Kompilierung von C und C++ durch das Frontend (z. B. Clang). Zum Kompilieren von (LLVM) IR implementiert der Kompilierer ein CSA-Backend-Ziel in LLVM mit drei Hauptstufen. Zuerst verringert das CSA-Back-End LLVM IR zu einem zielspezifischen Maschinenbefehl für die sequentielle Einheit, welche die meisten CSA-Operationen kombiniert mit einer herkömmlichen RISC-artigen Steuerungsflussarchitektur (z. B. mit Verzweigungen und einem Programmzähler) implementiert. Die sequentielle Einheit in der Toolkette kann sowohl für Kompilierer als auch für Anwendungsentwickler eine nützliche Hilfe sein, da sie eine stufenweise Transformation eines Programms von einem Steuerungsfluss (CF) zu einem Datenfluss (DF) ermöglicht, z. B. durch gleichzeitiges Umwandeln eines Codeabschnitts von Steuerfluss zu Datenfluss und zur Validierung der Programmkorrektheit. Die sequentielle Einheit kann auch ein Modell zur Handhabung von Code bereitstellen, der nicht in das räumliche Array passt. Danach wandelt der Kompilierer diesen Steuerfluss in Datenflussoperatoren (z. B. Code) für den CSA um. Diese Phase wird nachstehend in Abschnitt 4.3 beschrieben. Dann kann das CSA-Back-End seine eigenen Optimierungsschritte auf den Datenfluss-Operationen ausführen. Schließlich kann der Kompilierer die Befehle in einem CSA-Assemblierformat ausgeben. Dieses Assemblierformat wird als Eingabe für Tools der späten Stufe verwendet, welche die Datenfluss-Operationen auf der tatsächlichen CSA-Hardware platzieren und routen.

Control-to-Dataflow-Umwandlung

[0095] Ein wichtiger Teil des Kompilierers kann in dem Control-to-Dataflow-Umwandlungs-Durchlauf oder kurz Dataflow-Umwandlungs-Durchlauf implementiert sein. Dieser Durchlauf nimmt eine Funktion ein, die in der Steuerflussform dargestellt ist, z. B. ein Steuerflussgraph (CFG) mit sequentiellen Maschinenbefehlen, die auf virtuellen Registern arbeiten, und wandelt sie in eine Datenflussfunktion um, die konzeptionell ein Graph von Datenfluss-Operationen (Befehlen) ist, der durch latenzinsensitive Kanäle (LICs) verbunden wird. Dieser Abschnitt gibt eine High-Level-Beschreibung dieses Durchlaufs und beschreibt, wie er in bestimmten Ausführungsformen Speicheroperationen, Verzweigungen und Schleifen konzeptionell behandelt.

Geradliniger Code

[0096] **Fig. 17A** veranschaulicht einen sequentiellen Assembliercode **1702** gemäß Ausführungsformen der Offenbarung. **Fig. 17B** veranschaulicht einen Datenfluss-Assembliercode **1704** für den sequentiellen Assembliercode **1702** aus **Fig. 17A** gemäß Ausführungsformen der Offenbarung. **Fig. 17C** veranschaulicht einen Datenflussgraphen **1706** für den Datenfluss-Assembliercode **1704** aus **Fig. 17B** für einen Beschleuniger gemäß Ausführungsformen der Offenbarung.

[0097] Zunächst ist der einfache Fall der Umwandlung eines geradlinigen Codes zu Datenfluss in Betracht zu ziehen. Der Datenfluss-Umwandlungsdurchlauf kann einen Basisblock aus sequentiellem Code umwandeln, wie z. B. den Code aus **Fig. 17A** in CSA-Assembliercode, wie in **Fig. 17B** gezeigt. Vom Konzept her repräsentiert die CSA-Anordnung in **Fig. 17B** den Datenflussgraphen aus **Fig. 17C**. In diesem Beispiel wird jeder sequentielle Befehl in eine übereinstimmende CSA-Assembly übersetzt. Die .lic-Angaben (z. B. für Daten) deklarieren latenzinsensitive Kanäle, die den virtuellen Registern im sequentiellen Code (z. B. Rdata) entsprechen. In der Praxis kann die Eingabe in den Datenfluss-Umwandlungsdurchlauf in nummerierten virtuellen Registern erfolgen. Der Klarheit halber verwendet dieser Abschnitt beschreibende Registernamen. Es sei zu beachten, dass Lade- und Speicheroperationen in dieser Ausführungsform in der CSA-Architektur unterstützt werden, so dass viel mehr Programme ausgeführt werden können als in einer Architektur, die nur reinen Datenfluss unterstützt. Da der sequentielle Code, der in den Kompilierer eingegeben wird, in SSA-Form (Single Static Assignment - Einzelstatikzuweisung) vorliegt, kann der Control-to-Dataflow-Durchlauf für einen einfachen Basisblock jede virtuelle Registerdefinition in die Erzeugung eines Einzelwertes auf einem latenzinsensitiven Kanal umwandeln. Die SSA-Form ermöglicht eine Vielzahl von Verwendungen einer einzelnen Definition eines virtuellen Registers, wie z. B. in Rdata2. Zur Unterstützung dieses Modells, unterstützt der CSA-Assembliercode eine Vielzahl von Verwendungen derselben LIC (z. B. data2), wobei der Simulator implizit die erforderlichen Kopien der LICs erzeugt. Ein Hauptunterschied zwischen sequentiellem Code und Datenflusscode liegt in der Behandlung von Speicheroperationen. Der Code in **Fig. 17A** ist vom Konzept her seriell, was bedeutet, dass die load32 (ld32) von addr3 nach dem st32 von addr erfolgen sollte, falls sich die addr- und addr3-Adressen überschneiden.

Verzweigungen

[0098] Zum Konvertieren von Programmen mit einer Vielzahl von Basisblöcken und Bedingtheiten in dem Datenfluss generiert der Kompilierer spezielle Datenflussoperatoren, um die Verzweigungen zu ersetzen. Genauer verwendet der Kompilierer Schaltoperatoren, um ausgehende Daten am Ende eines Basisblocks in dem ursprünglichen CFG zu steuern, und Operatoren zum Auswählen von Werten aus dem angemessenen eingehenden Kanal am Anfang eines Basisblocks auszusuchen. Als ein konkretes Beispiel betrachte man den Code und den entsprechenden Datenflussgraphen aus **Fig. 18A** - **Fig. 18C**, die bedingt einen Wert von y basierend auf verschiedenen Eingaben berechnen: a, i, x und n. Nach dem Berechnen des Verzweigungsbedingungstests verwendet der Datenflusscode einen Switch-Operator (siehe z. B. **Fig. 3B** - **Fig. 3C)**, der den Wert in Kanal x zu Kanal xF steuert, wenn der Test 0 ist, oder Kanal xT, wenn der Test 1 ist. Ähnlich wird ein Pick-Operator (siehe z. B. **Fig. 3B** - **Fig. 3C)** verwendet, um den Kanal yF an y zu senden, wenn der Test 0 ist, oder den Kanal yT an y zu senden, wenn der Test 1 ist. In diesem Beispiel stellt sich heraus, dass, obwohl der Wert von a nur in der wahren Verzweigung der Bedingung verwendet wird, der CSA einen Switch-Operator aufweisen muss, der diese lenkt, um aT zu kanalisieren, wenn der Test 1 ist, und den Wert verbraucht (frisst), wenn der Test 0 ist. Letzterer Fall wird durch das Einstellen der falschen Eingabe des Schalters als %ign ausgedrückt. Es ist möglicherweise nicht richtig, Kanal a direkt mit dem wahren Pfad zu verbinden, da in den Fällen, in denen die Ausführung tatsächlich den falschen Pfad nimmt, dieser Wert von „a“ im Graphen übrig bleibt, was zu einem inkorrekten Wert von a für die nächste Ausführung der Funktion führt. Dieses Beispiel hebt die Eigenschaft der Steueräquivalenz hervor, einer Haupteigenschaft in Ausführungsformen der korrekten Datenflussumwandlung.

[0099] Steueräquivalenz: Man betrachte einen Single-Entry-Single-Exit-Steuerflussgraphen G mit zwei Basisblöcken A und B. A und B sind steueräquivalent, wenn alle vollständigen Steuerflusspfade durch G A und B die gleiche Anzahl von Malen besuchen.

[0100] LIC-Ersetzung: In einem Steuerflussgraphen G wird angenommen, dass eine Operation im Basisblock A ein virtuelles Register x definiert, und eine Operation im Basisblock B, der x verwendet. Dann kann eine korrekte Control-to-Dataflow-Transformation x durch einen latenzinsensitiven Kanal nur dann ersetzen, wenn A und B steueräquivalent sind. Die Steueräquivalenz-Beziehung trennt die Basisblöcke eines CFG in starke steuerungsabhängige Gebiete. **Fig. 18A** veranschaulicht einen C-Quellcode **1802** gemäß Ausführungsformen der Offenbarung. **Fig. 18B** veranschaulicht einen Datenfluss-Assembliercode **1804** für den C-Quellcode **1802** aus **Fig. 18A** gemäß Ausführungsformen der Offenbarung. **Fig. 18C** veranschaulicht einen Datenflussgraphen **1806** für den Datenfluss-Assembliercode **1804** aus **Fig. 18B** für einen Beschleuniger gemäß Ausführungsformen der Offenbarung. In dem Beispiel aus **Fig. 18A-18C** sind der Basisblock vor und nach den Bedingungen steueräquivalent miteinander, aber die Basisblöcke in dem wahren und falschen Pfad befinden sich jeweils in ihrem eigenen Steuerabhängigkeitsgebiet. Ein korrekter Algorithmus zum Umwandeln eines CFG in einen Datenfluss besteht darin, dass der Kompiliereinsatz (1) schaltet, um die Nichtübereinstimmung in der Ausführungsfrequenz für jeden Wert zu kompensieren, der zwischen Basisblöcken fließt, die nicht steueräquivalent sind, und (2) zu Beginn der Basisblöcke aussucht, um aus beliebigen eingehenden Werten in einem Basisblock zu wählen. Das Erzeugen der geeigneten Steuersignale für diese Picks und Switches kann der Schlüsselteil der Datenflussumwandlung sein.

Schleifen (Loops)

[0101] Eine weitere wichtige Klasse der CFGs bei der Datenflussumwandlung sind CFGs für Single-Entry-Single-Exit-Schleifen, eine herkömmliche Form von Schleife, die in (LLVM) IR generiert wird. Diese Schleifen können nahezu azyklisch sein, mit Ausnahme eines einzelnen Randes vom Ende der Schleife zurück zu einem Schleifenkopfblock. Der Datenfluss-Umwandlungsdurchlauf kann dieselbe High-Level-Strategie zum Umwandeln von Schleifen wie Verzweigungen verwenden, z. B. fügt er Switches am Ende der Schleife ein, um Werte aus der Schleife zu leiten (entweder außerhalb des Schleifenaustritts oder um den hinteren Rand herum zum Beginn der Schleife) und fügt Picks zu Beginn der Schleife ein, um zwischen anfänglichen Werten, die in die Schleife eintreten, und Werten, die durch den hinteren Rand kommen, zu wählen. **Fig. 19A** veranschaulicht einen C-Quellcode **1902** gemäß Ausführungsformen der Offenbarung. **Fig. 19B** veranschaulicht einen Datenfluss-Assembliercode **1904** für den C-Quellcode **1902** aus **Fig. 19A** gemäß Ausführungsformen der Offenbarung. **Fig. 19C** veranschaulicht einen Datenflussgraphen **1906** für den Datenfluss-Assembliercode **1904** aus **Fig. 19B** für einen Beschleuniger gemäß Ausführungsformen der Offenbarung. **Fig. 19A-19C** zeigen C- und CSA-Assembliercode für eine beispielhafte Do-While-Schleife, die Werte zu einer Schleifeninduktionsvariable i zuaddiert, sowie den entsprechenden Datenflussgraphen. Für jede Variable, welche die Schleife (i und Summe) vom Konzept her umkreist, hat dieser Graph ein entsprechendes Pick-/Switch-Paar, das den Fluss dieser Werte steuert. Es sei zu beachten, dass dieses Beispiel auch ein Pick-/Switch-Paar zum Schalten des Wertes n um die Schleife verwendet, obgleich n schleifeninvariant ist. Diese Wiederholung von n ermöglicht die Umwandlung des virtuellen Registers von n in eine LIC, da sie die Ausführungsfrequenzen zwischen einer begrifflichen Definition von n außerhalb der Schleife und der einen oder mehreren Verwendungen von n innerhalb der Schleife in Einklang bringt. Im Allgemeinen werden für eine korrekte Datenflussumwandlung Register, die live-in in einer Schleife sind, für jede Iteration innerhalb des Schleifenkörpers einmal wiederholt, wenn das Register in eine LIC umgewandelt wird. In ähnlicher Weise müssen Register, die innerhalb einer Schleife aktualisiert werden, live-out der Schleife verbraucht werden, z. B. mit einem eindeutigen Abschlusswert, der aus der Schleife gesendet wird. Schleifen führen eine Falte in den Datenfluss-Umwandlungsprozess ein, nämlich indem die Steuerung für eine Wahl an der Oberseite der Schleife und der Schalter für die Unterseite der Schleife versetzt sind. Wenn z. B. die Schleife in **Fig. 18A** drei Iterationen ausführt und beendet, sollte die Steuerung für die Wahl 0, 1, 1 sein, während die Steuerung für den Schalter 1, 1, 0 sein sollte. Diese Steuerung wird durch Starten des Auswahlkanals mit einer anfänglichen zusätzlichen 0 implementiert, wenn die Funktion in Zyklus 0 beginnt (der in der Assembly durch die Direktiven .value 0 und .avail 0 spezifiziert wird) und dann der Ausgabeschalter in den Picker kopiert. Es sei zu beachten, dass die letzte 0 in dem Umschalter eine abschließende 0 in dem Picker wiederherstellt, sodass der Abschlusszustand des Datenflussgraphen garantiert mit dem anfänglichen Zustand übereinstimmt.

[0102] **Fig. 20A** veranschaulicht ein Flussdiagramm **2000** gemäß Ausführungsformen der Offenbarung. Der dargestellte Fluss **2000** beinhaltet Decodieren eines Befehls mit einem Decodierer eines Kerns eines Prozessors in einen decodierten Befehl **2002**; Ausführen des decodierten Befehls mit einer Ausführungseinheit des Kerns des Prozessors zum Durchführen einer ersten Operation **2004**; Empfangen einer Eingabe eines Da-

tenflussgraphen, der mehrere Knoten **2006** umfasst; Überlagern des Datenflussgraphen über mehrere Verarbeitungselemente des Prozessors und ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen des Prozessors, wobei jeder Knoten einen Datenflussoperator in den mehreren Verarbeitungselementen **2008** repräsentiert; und Durchführen einer zweiten Operation des Datenflussgraphen mit dem Zwischenverbindungsnetz und den mehreren Verarbeitungselementen durch einen jeweiligen eingehenden Operandensatz, der an jedem der Datenflussoperatoren der mehreren Verarbeitungselemente **2010** eingeht.

[0103] Fig. 20B veranschaulicht ein Flussdiagramm **2001** gemäß Ausführungsformen der Offenbarung. Der dargestellte Fluss **2001** beinhaltet das Empfangen einer Eingabe eines Datenflussgraphen, umfassend mehrere Knoten **2003**; und Überlagern des Datenflussgraphen über mehrere Verarbeitungselemente eines Prozessors, eines Datenpfad-Netzwerks zwischen die mehreren Verarbeitungselemente und eines Flusssteuerpfad-Netzwerks zwischen die mehreren Verarbeitungselemente, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen **2005** repräsentiert ist.

[0104] In einer Ausführungsform schreibt der Kern einen Befehl in eine Speicherwarteschlange und ein CSA (z. B. die mehreren Verarbeitungselemente) überwacht die Speicherwarteschlange und beginnt mit der Ausführung, wenn der Befehl gelesen wird. In einer Ausführungsform führt der Kern einen ersten Teil eines Programms aus, und ein CSA (z. B. die mehreren Verarbeitungselemente) führt einen zweiten Teil des Programms aus. In einer Ausführungsform vollzieht der Kern andere Arbeit, während der CSA seine Operationen ausführt.

CSA-Vorteile

[0105] In bestimmten Ausführungsformen stellt die CSA-Architektur und - Mikroarchitektur tiefgreifende Vorteile in Bezug auf Energie, Leistung und Benutzerfreundlichkeit gegenüber Roadmap-Prozessorarchitekturen und FPGAs bereit. In diesem Abschnitt werden diese Architekturen mit Ausführungsformen des CSA verglichen und die Überlegenheit des CSA beim Beschleunigen von parallelen Datenflussgraphen in Bezug auf jeden davon hervorgehoben.

Prozessoren

[0106] Fig. 21 veranschaulicht einen Graphen von Durchsatz vs. Energie pro Operation **2100** gemäß Ausführungsformen der Offenbarung. Wie in **Fig. 21** gezeigt, sind kleine Kerne im Allgemeinen energieeffizienter als große Kerne, und bei einigen Arbeitslasten kann dieser Vorteil durch höhere Kernzahlen in absolute Leistung umgesetzt werden. Die CSA-Mikroarchitektur befolgt diese Beobachtungen bis ihrem Abschluss und entfernt (z. B. die meisten) energieraubenden Steuerstrukturen, die mit von-Neumann-Architekturen in Zusammenhang gebracht werden, einschließlich des größten Teils der befehlsseitigen Mikroarchitektur. Durch Entfernen dieser allgemeinen Aufwendungen und Implementieren einfacher Einzeloperations-PEs erhalten Ausführungsformen eines CSA ein dichtes, effizientes räumliches Array. Im Gegensatz zu kleinen Kernen, die normalerweise ziemlich seriell sind, kann ein CSA seine PEs gruppieren, z. B. über das leitungsvermittelte lokale Netzwerk, um explizit parallele aggregierte Datenflussgraphen zu bilden. Das Ergebnis ist Leistung in nicht nur parallelen Anwendungen, sondern auch in seriellen Anwendungen. Im Gegensatz zu Kernen, die für Leistung in Bezug auf Fläche und Energie teuer bezahlen, ist ein CSA in seinem nativen Ausführungsmodell bereits parallel. In bestimmten Ausführungsformen erfordert ein CSA weder Spekulation, um die Leistung zu erhöhen, noch muss er wiederholt Parallelität aus einer sequentiellen Programmdarstellung extrahieren, wodurch zwei der Hauptenergieverbraucher in von Neumann-Architekturen vermieden werden. Die meisten Strukturen in Ausführungsformen eines CSA sind verteilt, klein und energieeffizient, im Gegensatz zu den zentralisierten, voluminösen, energiehungrigen Strukturen, die in Kernen zu finden sind. Zu berücksichtigen sei der Fall von Registern im CSA: jedes PE kann einige wenige (z. B. 10 oder weniger) Speicherregister aufweisen. Allein genommen, können diese Register effizienter als herkömmliche Registerdateien sein. Zusammengefasst können diese Register die Wirkung einer großen, strukturinternen Registerdatei bereitstellen. Als ein Ergebnis vermeiden Ausführungsformen eines CSA die meisten Stapelverschüttungen und -füllungen, die bei klassischen Architekturen auftreten, und brauchen gleichzeitig weniger Energie pro Statuszugriff. Selbstverständlich können die Anwendungen weiter auf den Speicher zugreifen. In Ausführungsformen eines CSA sind Speicherzugriffsanfrage und -antwort architektonisch entkoppelt, wodurch Arbeitslasten ermöglicht werden, die viele weitere ausstehende Speicherzugriffe pro Einheit Bereich und Energie beibehalten. Diese Eigenschaft bietet eine wesentlich höhere Leistung für cachegebundene Arbeitslasten und reduziert den Bereich und die Energie, die zum Sättigen des Hauptspeichers in speichergebundenen Arbeitslasten benötigt werden. Ausführungsformen eines CSA legen neue Formen der Energieeffizienz frei, die in nichtvon-Neumann-Architekturen einzigartig sind. Eine Folge des Ausführens einer einzelnen Operation (z. B. eines Befehls) an einem (z. B. den meisten) PEs ist eine reduzierte Operandenentropie. Im Fall einer Inkrementierungsoperation kann

jede Ausführung zu einer Handvoll Umschaltmöglichkeiten auf der Schaltungsebene und einem geringen Energieverbrauch führen, ein Fall, der in Abschnitt 6.2 im Detail untersucht wird. Demgegenüber sind von Neumann-Architekturen gemultiplext, was zu einer höheren Anzahl von Bitübergängen führt. Der asynchrone Stil von Ausführungsformen eines CSA ermöglicht auch mikroarchitektonische Optimierungen, wie die in Abschnitt 3.5 beschriebenen Gleitkomma-Optimierungen, die in eng geplanten Kern-Pipelines schwierig zu realisieren sind. Da PEs relativ einfach sein können und ihr Verhalten in einem bestimmten Datenflussgraphen statisch bekannt ist, können Takt-Gating- und Power-Gating-Techniken effektiver angewendet werden als in gröberen Architekturen. Der Graphenausführungsstil, die kleine Größe und die Formbarkeit von Ausführungsformen der CSA-PEs und dem Netzwerk ermöglichen zusammen den Ausdruck vieler Arten von Parallelität: Befehls-, Daten-, Pipeline-, Vektor-, Speicher-, Thread- und Aufgabenparallelität können alle implementiert werden. Zum Beispiel kann in Ausführungsformen eines CSA eine Anwendung arithmetische Einheiten verwenden, um einen hohen Grad an Adressbandbreite bereitzustellen, während eine andere Anwendung dieselben Einheiten zur Berechnung verwenden kann. In vielen Fällen können mehrere Arten von Parallelität kombiniert werden, um noch mehr Leistung zu erzielen. Viele wichtige HPC-Operationen können sowohl repliziert als auch in Pipelines ausgeführt werden, was zu Leistungszugewinne von Größenordnung führt. Im Gegensatz dazu optimieren von-Neumann-Kerne typischerweise einen von den Architekten sorgfältig gewählten Parallelitätsstil, was zu einem Versagen der Erfassung aller wichtigen Anwendungskerne führt. So wie Ausführungsformen eines CSA viele Formen von Parallelität zeigen und erleichtern, wird keine bestimmte Form von Parallelität vorgeschrieben oder, schlimmer noch, ist keine bestimmte Subroutine in einer Anwendung vorhanden, um von dem CSA zu profitieren. Viele Anwendungen, einschließlich Einzel-Stream-Anwendungen, können sowohl Leistungs- als auch Energievorteile von Ausführungsformen eines CSA erhalten, z. B. sogar dann, wenn sie ohne Modifikation kompiliert werden. Dies kehrt den langen Trend um, dass ein signifikanter Programmieraufwand erforderlich ist, um einen wesentlichen Leistungsgewinn in Einzel-Stream-Anwendungen zu erzielen. Tatsächlich erhalten Ausführungsformen eines CSA in einigen Anwendungen mehr Leistung von funktionell äquivalenten, aber weniger „modernen“ Codes als von ihren verschachtelten zeitgenössischen Cousins, die gefoltert wurden, um Vektoranweisungen zu erzielen.

Vergleich zwischen CSA-Ausführungsformen und FGPAs

[0107] Die Wahl der Datenflussoperatoren als grundlegende Architektur von Ausführungsformen eines CSA unterscheidet diese CSAs von einem FGPA, und insbesondere ist der CSA als überlegener Beschleuniger für HPC-Datenflussgraphen, die aus traditionellen Programmiersprachen stammen. Die Datenflussoperatoren sind grundlegend asynchron. Dies ermöglicht Ausführungsformen eines CSA nicht nur eine große Freiheit bei der Implementierung in der Mikroarchitektur, sondern ermöglicht auch die einfache und prägnante Aufnahme abstrakter Architekturkonzepte. Zum Beispiel nehmen Ausführungsformen eines CSA natürlich viele Speicher-Mikroarchitekturen auf, die im Wesentlichen asynchron sind, mit einer einfachen Lade-Speicher-Schnittstelle. Man braucht nur einen FPGA DRAM-Controller zum Schätzen des Unterschieds in der Komplexität zu untersuchen. Ausführungsformen eines CSA nutzen Asynchronität auch, um schnellere und vollständiger ausgestattete Laufzeitdienste wie Konfiguration und Extraktion bereitzustellen, von denen angenommen wird, dass sie vier bis sechs Größenordnungen schneller sind als ein FPGA. Durch Verengen der Architekturschnittstelle ermöglichen Ausführungsformen eines CSA die Steuerung der meisten Zeitsteuerpfade auf Mikroarchitekturebene. Dies ermöglicht Ausführungsformen eines CSA bei einer viel höheren Frequenz als der allgemeinere Steuerungsmechanismus zu arbeiten, der in einem FPGA geboten wird. In ähnlicher Weise sind Takt und Reset, die für FPGAs architektonisch grundlegend sein können, im CSA mikroarchitektonisch, wodurch zum Beispiel ihre Unterstützung als programmierbare Einheiten überflüssig wird. Die Datenflussoperatoren können größtenteils grobkörnig sein. Indem nur grobe Operatoren behandelt werden, verbessern Ausführungsformen eines CSA sowohl die Dichte der Struktur als auch ihren Energieverbrauch: Der CSA führt Operationen direkt aus, anstelle sie mit Nachschlagetabellen zu emulieren. Eine zweite Konsequenz der Grobkörnigkeit ist eine Vereinfachung des Platzier- und Routingproblems. CSA-Datenflussgraphen sind viele Größenordnungen kleiner als FPGA-Netzlisten, und Platzier- und Routingzeit sind in Ausführungsformen eines CSA gleichfalls reduziert. Die signifikanten Unterschiede zwischen den Ausführungsformen eines CSA und eines FPGA machen den CSA als Beschleuniger überlegen, z. B. für Datenflussgraphen, die aus traditionellen Programmiersprachen stammen.

AUSWERTUNG

[0108] Beim CSA handelt es sich um eine neuartige Computerarchitektur, die im Vergleich zu Roadmap-Prozessoren ein enormes Potential zur Bereitstellung von Leistungs- und Energievorteilen innehat. Man betrachte den Fall des Berechnens einer einzelnen ausgreifenden Adresse zum Durchlaufen eines Arrays. Dieser Fall kann in HPC-Anwendungen wichtig sein, die z. B. einen bedeutenden Integer-Aufwand bei der Berechnung von Adress-Offsets verbringen. Bei der Adressberechnung und insbesondere bei der ausgreifenden Adress-

berechnung ist ein Argument konstant und das andere variiert nur geringfügig pro Berechnung. Daher werden in den meisten Fällen nur eine Handvoll Bits pro Zyklus umgeschaltet. Tatsächlich kann gezeigt werden, dass unter Verwendung einer Ableitung, die den in Abschnitt 3.5 beschriebenen gebundenen Gleitkomma-Übertragsbits gleicht, durchschnittlich weniger als zwei Bits Eingabe pro Berechnung für eine Schrittberechnung umschalten, wodurch die Energie um 50% gegenüber einer zufälligen Umschaltverteilung reduziert wird. Würde ein zeitgemultiplexter Ansatz verwendet, gingen viele dieser Energieeinsparungen verloren. In einer Ausführungsform erzielt der CSA eine ungefähr 3-fache Energieeffizienz gegenüber eines Kerns und erzielt einen 8-fachen Leistungszugewinn. Die Parallelitätszugewinne, die durch Ausführungsformen eines CSA erreicht werden, können zu reduzierten Programmlaufzeiten führen, was zu einer verhältnismäßigen wesentlichen Reduktion der Verlustenergie führt. Auf PE-Ebene sind die Ausführungsformen eines CSA extrem energieeffizient. Eine zweite wichtige Frage für den CSA ist, ob der CSA eine angemessene Menge an Energie auf Kachel-Level verbraucht. Da Ausführungsformen eines CSA jedes Gleitkomma-PE in der Struktur bei jedem Zyklus ausüben können, dient er als eine vernünftige Obergrenze für den Energie- und Leistungsverbrauch, sodass z . B. der größte Teil der Energie in Gleitkomma-Multiplikation und -Addition geht.

WEITERE CSA-DETAILS

[0109] Dieser Abschnitt bespricht weitere Details zur Konfiguration und Ausnahmehandhabung.

Mikroarchitektur zum Konfigurieren eines CSA

[0110] Dieser Abschnitt offenbart Beispiele zum Konfigurieren eines CSA (z. B. Struktur), wie diese Konfiguration schnell erreicht werden kann und wie der Ressourcenaufwand der Konfiguration minimiert werden kann. Die schnelle Konfiguration der Struktur kann bei der Beschleunigung kleiner Teile eines größeren Algorithmus und folglich bei der Erweiterung der Anwendbarkeit eines CSA von herausragender Bedeutung sein. Der Abschnitt offenbart ferner Merkmale, die Ausführungsformen eines CSA die Programmierung mit Konfigurationen unterschiedlicher Länge ermöglichen.

[0111] Ausführungsformen eines CSA (z. B. Struktur) können sich von herkömmlichen Kernen dadurch unterscheiden, dass sie einen Konfigurationsschritt verwenden, in dem (z. B. große) Teile der Struktur vor der Programmausführung mit einer Programmkonfiguration geladen werden. Ein Vorteil der statischen Konfiguration kann sein, dass sehr wenig Energie zur Laufzeit der Konfiguration verbraucht wird, z. B. im Gegensatz zu sequentiellen Kernen, die Energie für fast jeden Zyklus zum Abrufen von Konfigurationsinformationen (eines Befehls) verbrauchen. Der bisherige Nachteil der Konfiguration besteht darin, dass es sich um einen grobkörnigen Schritt mit einer möglicherweise großen Latenz handelt, was eine Untergrenze für die Größe des Programms darstellt, die aufgrund der Kosten der Kontextumschaltung in der Struktur beschleunigt werden kann. Diese Offenbarung beschreibt eine skalierbare Mikroarchitektur zum schnellen Konfigurieren eines räumlichen Arrays in einer verteilten Art und Weise, die z. B. die bisherigen Nachteile vermeidet.

[0112] Wie oben erläutert, kann ein CSA leichtgewichtige Verarbeitungselemente aufweisen, die durch ein Inter-PE-Netzwerk verbunden sind. Programme, die als Steuerdatenflussgraphen angesehen werden, werden dann auf die Architektur abgebildet, indem die konfigurierbaren Strukturelemente (CFEs) konfiguriert werden, z. B. PEs und die Zwischenverbindungs (Struktur) -Netze. Im Allgemeinen können PEs als Datenflussoperatoren konfiguriert sein und, sobald alle Eingabeoperanden am PE eingetroffen sind, kann eine Operation stattfinden und die Ergebnisse an ein weiteres PE oder PEs zum Verbrauch oder Ausgabe weitergeleitet werden. PEs können über zweckgebundene virtuelle Schaltungen kommunizieren, die durch statistisches Konfigurieren eines leitungsvermittelten Kommunikationsnetzwerks gebildet werden. Diese virtuellen Schaltungen können flussgesteuert und vollständig gegengedrückt sein, z. B. so, dass die PEs anhalten, wenn entweder die Quelle keine Daten aufweist oder der Zielspeicherplatz voll ist. Bei Laufzeit können Daten durch die PEs fließen und den abgebildeten Algorithmus implementieren. Zum Beispiel können Daten aus dem Speicher durch die Struktur eingestreamt werden und dann zurück in den Speicher gehen. Eine solche räumliche Architektur kann eine bemerkenswerte Leistungseffizienz im Vergleich zu herkömmlichen Mehrkernprozessoren erreichen: die Berechnung, in der Form von PEs kann einfacher und zahlreicher als größere Kerne sein und die Kommunikation direkt sein, z. B. im Gegensatz zu einer Erweiterung des Speichersystems.

[0113] Ausführungsformen eines CSA brauchen keine (z. B. softwaregesteuerte) Paketvermittlung verwenden, z. B. Paketvermittlung, die eine erhebliche Softwareunterstützung zur Ausführung erfordert, welche die Konfiguration verlangsamt. Ausführungsformen eines CSA weisen eine Außer-Band-Signalisierung in dem Netzwerk (z. B. von nur zwei bis drei Bits, abhängig von dem unterstützten Merkmalssatz) und eine Topologie mit fester Konfiguration auf, um den Bedarf an einer signifikanten Softwareunterstützung zu vermeiden.

[0114] Ein Hauptunterschied zwischen Ausführungsformen eines CSA und dem Ansatz, der in FPGAs verwendet wird, besteht darin, dass ein CSA-Ansatz ein breites Datenwort verwenden kann, verteilt ist und Mechanismen aufweist, um Programmdaten direkt aus dem Speicher abzurufen. Ausführungsformen eines CSA können im Interesse der Bereichseffizienz keine JTAG-artigen Einzelbitkommunikationen verwenden, weil dies z. B. Millisekunden benötigt, um eine große FPGA-Struktur vollständig zu konfigurieren.

[0115] Ausführungsformen eines CSA beinhalten ein verteiltes Konfigurationsprotokoll und eine Mikroarchitektur, um dieses Protokoll zu unterstützen. Anfangs kann sich der Konfigurationsstatus im Speicher befinden. Mehrere (z. B. verteilte) lokale Konfigurationssteuerungen (Boxes) (LCCs) können Teile des Gesamtprogramms in ihr lokales Gebiet der räumlichen Struktur streamen, z. B. unter Verwendung einer Kombination aus einem kleinen Satz von Steuersignalen und dem Netzwerk mit Struktur. Statuselemente können an jedem CFE verwendet werden, um Konfigurationsketten zu bilden, z. B. um einzelnen CFEs zu ermöglichen, sich ohne globale Adressierung selbst zu programmieren.

[0116] Ausführungsformen eines CSA weisen eine spezifische Hardwareunterstützung für die Bildung von Konfigurationsketten auf, z. B. keine Software, die diese Ketten dynamisch auf Kosten einer zunehmenden Konfigurationszeit erstellt. Ausführungsformen eines CSA sind nicht rein paketvermittelt und weisen zusätzliche Außer-Band-Steuerdrähte (z. B. wird die Steuerung nicht durch den Datenpfad gesendet, was zusätzliche Zyklen erfordert, um diese Information abzutasten und diese Information zu reserialisieren) auf. Ausführungsformen eines CSA verringern die Konfigurationslatenz durch Festlegen der Konfigurationsreihenfolge und durch Bereitstellen einer expliziten Außer-Band-Steuerung (z. B. um mindestens einen Faktor von zwei), während die Netzwerkkomplexität nicht wesentlich zunimmt.

[0117] Ausführungsformen eines CSA verwenden keinen seriellen Mechanismus zur Konfiguration, bei dem Daten Bit für Bit unter Verwendung eines JTAG-artigen Protokolls in die Struktur gestreamt werden. Ausführungsformen eines CSA benutzen einen grobkörnigen Strukturansatz. In bestimmten Ausführungsformen ist das Hinzufügen einiger Steuerdrähte oder Zustandselemente zu einer 64- oder 32-Bit-orientierten CSA-Struktur kostengünstiger als das Hinzufügen derselben Steuermechanismen zu einer 4- oder 6-Bit-Struktur.

[0118] **Fig. 22** veranschaulicht eine Beschleuniger-Kachel **2200**, umfassend ein Array von Verarbeitungselementen (PE) und eine lokale Konfigurationssteuerung (**2202**, **2206**) gemäß Ausführungsformen der Offenbarung. Jedes PE, jede Netzwerksteuerung und jeder Schalter können konfigurierbare Strukturelemente (CFEs) sein, die z. B. durch Ausführungsformen der CSA-Architektur konfiguriert (z. B. programmiert) werden.

[0119] Ausführungsformen eines CSA weisen Hardware auf, die eine effiziente, verteilte Konfiguration mit niedriger Latenzzeit für eine heterogene räumliche Struktur bereitstellt. Dies kann gemäß vier Techniken erreicht werden. Zuerst wird eine Hardwareentität, die lokale Konfigurationssteuerung (LCC - Local Configuration Controller) benutzt, wie in **Fig. 22** - **Fig. 24**. Eine LCC kann einen Stream aus Konfigurationsinformation aus einem (z. B. virtuellen) Speicher abrufen. Zweitens kann ein Konfigurationsdatenpfad enthalten sein, der z. B. so breit wie die ursprüngliche Breite der PE-Struktur ist und der die PE-Struktur überlagern kann. Drittens können neue Steuersignale in der PE-Struktur empfangen werden, die den Konfigurationsprozess durchführen. Viertens können Zustandselemente an jedem konfigurierbaren Endpunkt angeordnet sein (z. B. in einem Register), die den Status benachbarter CFEs verfolgen, so dass sich jedes CFE ohne zusätzliche Steuersignale eindeutig selbst konfigurieren kann. Diese vier mikroarchitektonischen Merkmale können einem CSA das Konfigurieren von Ketten seiner CFEs ermöglichen. Zum Erhalten einer geringen Konfigurationslatenz kann die Konfiguration durch Bilden vieler LCCs und CFE-Ketten partitioniert werden. Zur Konfigurationszeit können diese unabhängig voneinander arbeiten, um die Struktur parallel zu laden, z. B. um die Latenz drastisch zu reduzieren. Als ein Ergebnis dieser Kombinationen können Strukturen, die unter Verwendung von Ausführungsformen eines CSA-Architektur konfiguriert sind, vollständig konfiguriert sein (z. B. in Hunderten von Nanosekunden). Im Folgenden wird der Betrieb der verschiedenen Komponenten von Ausführungsformen eines CSA-Konfigurationsnetzwerks detailliert beschrieben.

[0120] **Fig. 23A-23C** veranschaulicht eine lokale Konfigurationssteuerung **2302**, die ein Datenpfad-Netzwerk gemäß Ausführungsformen der Offenbarung konfiguriert. Das dargestellte Netzwerk weist mehrere Multiplexer (z. B. Multiplexer **2306**, **2308**, **2310**) auf, die konfiguriert werden können (z. B. über zugehörige Steuersignale), um einen oder mehrere Datenpfade (z. B. von PEs) miteinander zu verbinden. **Fig. 23A** veranschaulicht das Netzwerk **2300** (z. B. Struktur), das für eine bisherige Operation oder Programm konfiguriert (z. B. eingestellt) wurde. **Fig. 23B** zeigt die lokale Konfigurationssteuerung **2302** (die z. B. eine Netzwerkschnittstellenschaltung **2304** zum Senden und/oder Empfangen von Signalen aufweist), die ein Konfigurationssignal abtastet und das lokale Netzwerk auf eine Standardkonfiguration (z. B. wie dargestellt) setzt, die es der LCC erlaubt, Konfigu-

rationsdaten an alle konfigurierbaren Strukturelemente (CFEs), z. B. Muxe, zu senden. **Fig. 23C** veranschaulicht die LCC-Abtast-Konfigurationsinformation über das Netzwerk, wobei CFEs in einer vorbestimmten (z. B. siliciumdefinierten) Sequenz konfiguriert werden. In einer Ausführungsform kann die Operation unverzüglich beginnen, wenn die CFEs konfiguriert werden. In einer anderen Ausführungsform warten die CFEs mit dem Beginn der Operation, bis die Struktur vollständig konfiguriert ist (z. B. wie durch den Konfigurationsterminator (z. B. Konfigurationsterminator **2504** und Konfigurationsterminator **2508** in **Fig. 25**) für jede lokale Konfigurationssteuerung signalisiert). In einer Ausführungsform erhält die LCC die Steuerung über die Netzwerkstruktur durch Senden einer Sondernachricht oder Ansteuern eines Signals. Er tastet dann Konfigurationsdaten (z. B. über eine Periode mehrerer Zyklen) zu den CFEs in der Struktur ab. In diesen Figuren sind die Multiplexernetzwerke Analoga des „Schalters (oder Switches)“, der in bestimmten Figuren gezeigt ist (z. B. **Fig. 6**).

Lokale Konfigurationssteuerung

[0121] **Fig. 24** veranschaulicht eine (z. B. lokale) Konfigurationssteuerung **2402** gemäß Ausführungsformen der Offenbarung. Eine lokale Konfigurationssteuerung (LCC - Local Configuration Controller) kann die Hardwareentität sein, die für das Laden der lokalen Teile (z. B. in einem Untersatz einer Kachel oder auf andere Weise) des Strukturprogramms, das Interpretieren dieser Programmteile und dann das Laden dieser Programmteile in die Struktur verantwortlich ist, indem sie das angemessene Protokoll auf den verschiedenen Konfigurationsdrähten ansteuert. In dieser Eigenschaft kann die LCC ein spezieller sequentieller Mikrocontroller sein.

[0122] Die LCC-Operation kann beginnen, wenn ein Zeiger zu einem Codesegment empfangen wird. Je nach der LCC-Mikroarchitektur kann dieser Zeiger (z. B. im Zeigerregister **2406** gespeichert) entweder über ein Netzwerk (z. B. von innerhalb des CSA (Struktur) selbst) oder über einen Speichersystemzugriff auf die LCC kommen. Bei Empfangen eines solchen Zeigers entzieht die LCC optional den relevanten Status von ihrem Teil der Struktur für den Kontextspeicher und fährt dann fort, den Teil der Struktur, für den sie verantwortlich ist, sofort zu rekonfigurieren. Das von der LCC geladene Programm kann eine Kombination von Konfigurationsdaten für die Struktur- und Steuerbefehle für die LCC sein, die z. B. leicht codiert sind. Wenn die LCC in den Programmabschnitt streamt, kann sie das Programm als einen Befehlsstream interpretieren und die geeignete codierte Aktion ausführen, um die Struktur zu konfigurieren (z. B. zu laden).

[0123] Zwei unterschiedliche Mikroarchitekturen für die LCC sind in **Fig. 22** gezeigt, wobei z. B. eine oder beide in einem CSA benutzt werden. Die erste setzt die LCC **2202** als Speicherschnittstelle. In diesem Fall kann die LCC direkte Anforderungen an das Speichersystem zum Laden von Daten stellen. Im zweiten Fall ist die LCC **2206** auf einem Speichernetzwerk angeordnet, in dem sie Anforderungen an den Speicher nur indirekt stellen kann. In beiden Fällen bleibt die logische Operation der LCC unverändert. In einer Ausführungsform wird eine LCC über das zu ladende Programm informiert, beispielsweise durch einen Satz von (z. B. OS-sichtbaren) Steuerstatusregistern, die verwendet werden, um einzelne LCCs über neue Programmzeiger usw. zu informieren.

Zusätzliche Außer-Band-Steuerkanäle (z. B. Drähte)

[0124] In bestimmten Ausführungsformen verlässt sich die Konfiguration auf 2 bis 8 zusätzliche Außer-Band-Steuerkanäle, um die Konfigurationsgeschwindigkeit wie unten definiert zu verbessern. Zum Beispiel kann die Konfigurationssteuerung **2402** die folgenden Steuerkanäle aufweisen, z. B. den CFG_START Steuerkanal **2408**, den CFG_VALID Steuerkanal **2410** und den CFG_DONE Steuerkanal **2412**, wobei Beispiele von jedem einzelnen in der nachstehenden Tabelle 2 besprochen sind.

Tabelle 2 Steuerkanäle

CFG_START	Asserted at beginning of configuration. Sets configuration state at each CFE and sets the configuration bus.
CFG_VALID	Denotes validity of values on configuration bus.
CFG_DONE	Optional. Denotes completion of the configuration of a particular CFE. This allows configuration to be short circuited in case a CFE does not require additional configuration

[0125] Im Allgemeinen kann die Handhabung von Konfigurationsinformation dem Implementierer eines bestimmten CFE überlassen werden. Zum Beispiel kann ein auswählbares Funktions-CFE eine Vorkehrung zum

Setzen von Registern unter Verwendung eines existierenden Datenpfads aufweisen, während eine festes Funktions-CFE einfach ein Konfigurationsregister einstellen kann.

[0126] Aufgrund der langen Drahtverzögerungen beim Programmieren langer CFE-Sätze kann das CFG_ VALID Signal als eine Takt-/Latch-Aktivierung für die CFE-Komponenten behandelt werden. Da dieses Signal als ein Takter verwendet wird, beträgt in einer Ausführungsform der Arbeitszyklus der Leitung höchstens 50%. Als Ergebnis wird der Konfigurationsdurchsatz in etwa halbiert. Optional kann ein zweites CFG_VALID Signal hinzugefügt werden, um eine kontinuierliche Programmierung zu ermöglichen.

[0127] In einer Ausführungsform wird nur CFG_START strikt an eine unabhängige Kopplung (z. B. Draht) kommuniziert, zum Beispiel können CFG_VALID und CFG_DONE über andere Netzwerkkopplungen überlagert werden.

Wiederverwendung von Netzwerkressourcen

[0128] Zum Reduzieren des Konfigurationsaufwands nutzen bestimmte Ausführungsformen eines CSA die vorhandene Netzwerkinfrastruktur zur Kommunikation von Konfigurationsdaten. Eine LCC kann sowohl eine Chipebenen-Speicherhierarchie als auch Strukturebenen-Kommunikationsnetzwerke zum Bewegen von Daten vom Speicher in die Struktur verwenden. Als Ergebnis trägt die Konfigurationsinfrastruktur in bestimmten Ausführungsformen eines CSA nicht mehr als 2% zum gesamten Strukturbereich und zur Gesamtleistung bei.

[0129] Die Wiederverwendung von Netzwerkressourcen in bestimmten Ausführungsformen eines CSA kann ein Netzwerk dazu veranlassen, einige Hardwareunterstützung für einen Konfigurationsmechanismus zu erlangen. Leitungsvermittelte Netzwerke von Ausführungsformen eines CSA bewirken, dass eine LCC ihre Multiplexer auf eine spezifische Weise für die Konfiguration setzt, wenn das Signal ‚CFG_START' angegeben wird. Paketvermittelte Netzwerke erfordern keine Erweiterung, obwohl LCC-Endpunkte (z. B. Konfigurationsterminatoren) eine spezifische Adresse in dem paketvermittelten Netzwerk verwenden. Die Netzwerkwiederverwendung ist optional und einige Ausführungsformen finden ggf. eigens vorgesehene Konfigurationsbusse angemessener.

Per-CFE-Status

[0130] Jedes CFE kann ein Bit halten, das angibt, ob es konfiguriert wurde oder nicht (siehe z. B. **Fig. 13**). Dieses Bit kann deaktiviert werden, wenn das Konfigurationsstartsignal angesteuert wird, und dann aktiviert werden, sobald das bestimmte CFE konfiguriert wurde. In einem Konfigurationsprotokoll sind die CFEs angeordnet, um Ketten mit dem CFE-Konfigurationsstatussbit zu bilden, das die Topologie der Kette bestimmt. Ein CFE kann das Konfigurationsstatussbit des unmittelbar angrenzenden CFE lesen. Wenn dieses benachbarte CFE konfiguriert ist und das derzeitige CFE nicht konfiguriert ist, kann das CFE bestimmen, dass sämtliche derzeitigen Konfigurationsdaten auf das derzeitige CFE abzielen. Wenn das „CFG_DONE"-Signal aktiviert wird, kann das CFE sein Konfigurationsbit setzen, z. B. vorgeschaltete CFEs zum Konfigurieren aktivieren. Als ein Basisfall für den Konfigurationsprozess kann ein Konfigurations-Terminator (z. B. Konfigurations-Terminator **2204** für LCC **2202** oder Konfigurations-Terminator **2208** für LCC **2206** in **Fig. 22**), der bestätigt, dass er konfiguriert ist, am Ende einer Kette aufgenommen werden.

[0131] CFE-intern kann dieses Bit zum Ansteuern der flusssteuerbereiten Signale verwendet werden. Wenn zum Beispiel das Konfigurationsbit deaktiviert wird, können Netzwerksteuersignale automatisch auf einen Wert geklemmt werden, der verhindert, dass Daten fließen, während innerhalb der PEs keine Operationen oder andere Aktionen geplant werden.

Behandeln von Konfigurationspfaden mit hoher Verzögerung

[0132] Eine Ausführungsform einer LCC kann ein Signal über eine lange Distanz, z. B. durch viele Multiplexer und mit vielen Lasten, treiben. Daher kann es für ein Signal schwierig sein, an einem entfernten CFE innerhalb eines kurzen Taktzyklus einzugehen. In bestimmten Ausführungsformen sind die Konfigurationssignale in einer bestimmten Division (z. B. einem Bruchteil von) der Haupttaktfrequenz (z. B. CSA), um eine digitale Zeitdisziplin bei der Konfiguration sicherzustellen. Die Taktteilung kann in einem Außer-Band-Signalisierungsprotokoll benutzt werden und erfordert keine Modifikation des Haupttaktbaums.

Sicherstellen des konsistenten Strukturverhaltens während der Konfiguration

[0133] Da bestimmte Konfigurationsschemata verteilt sind und aufgrund von Programm- und Speichereffekten eine nicht-deterministische Zeitsteuerung aufweisen, können verschiedene Teile der Struktur zu unterschiedlichen Zeiten konfiguriert werden. Als ein Ergebnis stellen bestimmte Ausführungsformen eines CSA Mechanismen bereit, um eine inkonsistente Operation zwischen konfigurierten und unkonfigurierten CFEs zu verhindern. Allgemein wird Konsistenz als eine Eigenschaft angesehen, die von den CFEs selbst gefordert und beibehalten wird, z. B. unter Verwendung des internen CFE-Zustands. Wenn ein CFE z. B. in einem unkonfigurierten Status ist, kann es beanspruchen, dass seine Eingabepuffer voll sind und dass seine Ausgabe ungültig ist. Beim Konfigurieren werden diese Werte auf den wahren Status der Puffer eingestellt. Da genügend von der Struktur aus der Konfiguration kommt, können diese Techniken den Beginn der Operation zulassen. Dies hat den Effekt der weiteren Reduktion der Kontext-Switching-Latenz, z. B. wenn Speicheranfragen mit langer Latenz früh ausgegeben werden.

Konfiguration mit variabler Breite

[0134] Unterschiedliche CFEs können unterschiedliche Konfigurationswortbreiten aufweisen. Bei kleineren CFE-Konfigurationswörtern können die Implementierer die Verzögerung kompensieren, indem sie CFE-Konfigurationslasten über die Netzwerkleitungen gleichmäßig zuweisen. Zum Ausgleichen der Ladung von Netzwerkleitungen ist eine Möglichkeit, Konfigurationsbits verschiedenen Abschnitten von Netzwerkleitungen zuzuweisen, um die Nettoverzögerung auf einen einzigen Draht zu begrenzen. Breite Datenwörter können durch Serialisierungs-/Deserealisierungstechniken gehandhabt werden. Diese Entscheidungen können auf einer Per-Fabric-Basis getroffen werden, um das Verhalten eines spezifischen CSA (z. B. Struktur) zu optimieren. Die Netzwerksteuerung (z. B. eine oder mehrere von Netzwerksteuerung **2210** und Netzwerksteuerung **2212** können mit jeder Domäne (z. B. Untersatz) des CSA (z. B. Struktur) kommunizieren, um z. B. Konfigurationsinformationen an eine oder mehrere LCCs zu senden.

Mikroarchitektur für die Niederlatenz-Konfiguration eines CSA und zum rechtzeitigen Abrufen von Konfigurationsdaten für einen CSA

[0135] Ausführungsformen eines CSA können ein energieeffizientes und leistungsstarkes Mittel sein, um Benutzeranwendungen zu beschleunigen. Bei der Berücksichtigung, ob ein Programm (z. B. ein Datenflussgraph davon) erfolgreich durch einen Beschleuniger beschleunigt werden kann, können sowohl die Zeit zum Konfigurieren des Beschleunigers als auch die Zeit zum Ausführen des Programms in Betracht gezogen werden. Wenn die Laufzeit kurz ist, kann die Konfigurationszeit eine große Rolle bei der Bestimmung der erfolgreichen Beschleunigung spielen. Um die Domäne von beschleunigbaren Programmen zu maximieren, wird daher in einigen Ausführungsformen die Konfigurationszeit so kurz wie möglich gemacht. Einer oder mehrere Konfigurations-Caches können in einem CSA enthalten sein, z. B. derart, dass der Speicher mit hoher Bandbreite und niedriger Latenz eine schnelle Rekonfiguration ermöglicht. Im Folgenden wird eine Beschreibung verschiedener Ausführungsformen eines Konfigurations-Caches gegeben.

[0136] In einer Ausführungsform greift während der Konfiguration die Konfigurationshardware (z. B. LCC) optional auf den Konfigurations-Cache zu, um neue Konfigurationsinformationen zu erhalten. Der Konfigurations-Cache kann entweder als ein traditioneller adressbasierter Cache oder in einem OS-verwalteten Modus arbeiten, in dem Konfigurationen in dem lokalen Adressraum gespeichert sind und durch Bezugnahme auf diesen Adressraum adressiert werden. Wenn sich der Konfigurationsstatus in dem Cache befindet, dann müssen in bestimmten Ausführungsformen keine Anforderungen an den Sicherungsspeicher gestellt werden. In bestimmten Ausführungsformen ist dieser Konfigurations-Cache von sämtlichen (z. B. Lower-Level) gemeinsam genutzten Caches in der Speicherhierarchie getrennt.

[0137] **Fig. 25** veranschaulicht eine Beschleuniger-Kachel **2500**, umfassend ein Array von Verarbeitungselementen, ein Konfigurations-Cache (z. B. **2518** oder **2520**) und eine lokale Konfigurationssteuerung (z. B. **2502** oder **2506**) gemäß Ausführungsformen der Offenbarung. In einer Ausführungsform ist der Konfigurations-Cache **2514** mit der lokalen Konfigurationssteuerung **2502** gemeinsam angeordnet. In einer Ausführungsform befindet sich der Konfigurations-Cache **2518** in der Konfigurationsdomäne der lokalen Konfigurationssteuerung **2506**, z. B. mit einer ersten Domäne, die am Konfigurations-Terminator **2504** endet, und einer zweiten Domäne, die am Konfigurations-Terminator **2508** endet. Ein Konfigurations-Cache kann es einer lokalen Konfigurationssteuerung ermöglichen, während der Konfiguration auf den Konfigurations-Cache Bezug zu nehmen, z. B. in der Hoffnung, einen Konfigurationsstatus mit einer niedrigeren Latenz als einen Bezug auf den Speicher

zu erhalten. Ein Konfigurations-Cache (Speicher) kann entweder dediziert sein oder kann als ein Konfigurationsmodus eines strukturinternen Speicherelements, z. B. des lokalen Caches **2516**, zugänglich sein.
Cachemodi

1. Demand Caching - In diesem Modus arbeitet das Konfigurations-Cache als true Cache. Die Konfigurationssteuerung gibt adressbasierte Anfragen aus, die auf Tags im Cache überprüft werden. Fehler werden in den Cache geladen und können dann während einer zukünftigen Neuprogrammierung erneut referenziert werden.

2. In-Fabric Storage (Scratchpad) Caching - In diesem Modus empfängt der Konfigurations-Cache einen Verweis auf eine Konfigurationssequenz in seinem eigenen kleinen Adressraum und nicht im größeren Adressraum des Hosts. Dies kann die Speicherdichte verbessern, da der Teil des Caches, der zum Speichern von Tags verwendet wird, stattdessen zum Speichern der Konfiguration verwendet werden kann.

[0138] In bestimmten Ausführungsformen kann ein Konfigurations-Cache die Konfigurationsdaten darin vorgeladen aufweisen, z. B. entweder durch externe Richtung oder interne Richtung. Dies kann die Reduktion der Latenz zum Laden von Programmen ermöglichen. Bestimmte Ausführungsformen hierin stellen eine Schnittstelle zu einem Konfigurations-Cache bereit, die das Laden eines neuen Konfigurationsstatus in den Cache erlaubt, z. B. selbst wenn eine Konfiguration bereits in der Struktur läuft. Die Initiierung dieser Ladung kann entweder von einer internen oder externen Quelle erfolgen. Ausführungsformen des Vorlademechanismus reduzieren die Latenz weiter, indem sie die Latenz der Cacheladung von dem Konfigurationspfad entfernen.
Pre-Fetching-Modi

1. Explicit Prefetching - Ein Konfigurationspfad wird mit einem neuen Befehl augmentiert, ConfigurationCachePrefetch. Statt der Programmierung der Struktur bewirkt dieser Befehl lediglich das Laden der relevanten Programmkonfiguration in einen Konfigurations-Cache, ohne die Struktur zu programmieren. Da dieser Mechanismus auf der vorhandenen Konfigurationsinfrastruktur pendelt, wird er sowohl innerhalb der Struktur als auch extern offengelegt, z. B. für Kerne und andere Entitäten, die auf den Speicherraum zugreifen.

2. Implicit prefetching -Eine globale Konfigurationssteuerung kann einen Prefetch-Prädiktor beibehalten und diesen verwenden, um das explizite Vorladen zu einem Konfigurations-Cache, z. B. automatisch, zu initiieren.

Hardware zur schnellen Rekonfiguration eines CSA als Reaktion auf eine Ausnahme

[0139] Bestimmte Ausführungsformen eines CSA (z. B. eine räumliche Struktur) weisen große Mengen eines Befehls- und Konfigurationsstatus auf, der z. B. während des Betriebs des CSA weitgehend statisch ist. Daher kann der Konfigurationsstatus anfällig für weiche Fehler sein. Die schnelle und fehlerfreie Wiederherstellung dieser weichen Fehler kann für die langfristige Zuverlässigkeit und Leistung von räumlichen Systemen entscheidend sein.

[0140] Bestimmte Ausführungsformen hierin stellen eine schnelle Konfigurationswiederherstellungsschleife bereit, bei der z. B. Konfigurationsfehler erkannt und Teile der Struktur sofort rekonfiguriert werden. Bestimmte Ausführungsformen hierin weisen eine Konfigurationssteuerung auf, z. B. mit Merkmalen zur Umprogrammierung der Zuverlässigkeit, Verfügbarkeit und Wartungsfreundlichkeit (RAS). Bestimmte Ausführungsformen des CSA weisen einen Schaltkreis für eine Hochgeschwindigkeitskonfiguration, eine Fehlerberichterstattung und eine Paritätsprüfung innerhalb der räumlichen Struktur auf. Unter Verwendung einer Kombination dieser drei Merkmale und optional eines Konfigurations-Cache kann sich eine Konfigurations-/Ausnahmehandhabungsschaltung von weichen Fehlern in der Konfiguration erholen. Nach der Erkennung können weiche Fehler zu einem Konfigurations-Cache übertragen werden, der eine sofortige Rekonfiguration der Struktur initiiert (z. B. dieses Teils). Bestimmte Ausführungsformen stellen eine dedizierte Rekonfigurationsschaltung bereit, die z. B. schneller ist als jede Lösung, die indirekt in die Struktur implementiert würde. In bestimmten Ausführungsformen kooperieren die co-lokalisierte Ausnahme- und Konfigurationsschaltung, um die Struktur bei der Konfigurationsfehlererkennung neu zu laden.

[0141] **Fig. 26** veranschaulicht eine Beschleuniger-Kachel **2600**, umfassend ein Array von Verarbeitungselementen und eine Konfigurations- und Ausnahmehandhabungssteuerung (**2602**, **2606**) mit einer Rekonfigurationsschaltung (**2618**, **2622**) gemäß Ausführungsformen der Offenbarung. In einer Ausführungsform sendet, wenn ein PE einen Konfigurationsfehler durch seine lokalen RAS-Merkmale erkennt, es eine Nachricht (z. B. einen Konfigurationsfehler oder einen Rekonfigurationsfehler) durch seinen Ausnahmegenerator an die Konfigurations- und Ausnahmehandhabungssteuerung (z. B. **2602** oder **2606**). Bei Empfang dieser Nachricht in-

itiiert die Konfigurations- und Ausnahmehandhabungssteuerung (z. B. **2602** oder **2606**) die co-lokalisierte Rekonfigurationsschaltung (z. B. **2618** oder **2622**), um den Konfigurationsstatus neu zu laden. Die Konfigurations-Mikroarchitektur geht weiter und lädt (z. B. nur) den Konfigurationsstatus neu, und in bestimmten Ausführungsformen nur den Konfigurationsstatus für das PE, das den RAS-Fehler meldet. Nach Abschluss der Rekonfiguration kann die Struktur die normale Operation wieder aufnehmen. Zum Verringern der Latenz kann der Konfigurationsstatus, der von der Konfigurations- und Ausnahmehandhabungssteuerung (z. B. **2602** oder **2606**) verwendet wird, aus einem Konfigurations-Cache bezogen werden. Als ein Basisfall für den Konfigurations- und Rekonfigurationsprozess kann ein Konfigurations-Terminator (z. B. Konfigurations-Terminator **2604** für die Konfigurations- und Ausnahmehandhabungssteuerung **2602** oder Konfigurations-Terminator **2608** für die Konfigurations- und Ausnahmehandhabungssteuerung **2606** in **Fig. 26**), der bestätigt, dass er konfiguriert (oder rekonfiguriert) ist, am Ende einer Kette aufgenommen werden.

[0142] Fig. 27 veranschaulicht eine Rekonfigurationsschaltung **2718** gemäß Ausführungsformen der Offenbarung. Die Rekonfigurationsschaltung **2718** weist ein Konfigurationsstatusregister **2720** auf, um den Konfigurationsstatus (oder einen Zeiger darauf) zu speichern.

Hardware für eine strukturinitiierte Rekonfiguration eines CSA

[0143] Einige Teile einer Anwendung, die auf ein CSA (z. B. räumliches Array) abzielen, können infrequent laufen oder können sich gegenseitig mit anderen Teilen des Programms ausschließen. Zum Sparen von Fläche und zum Verbessern der Leistung und/oder Reduzieren von Energie, kann es nützlich sein, Teile der räumlichen Struktur zwischen mehreren verschiedenen Teilen des Programm-Datenflussgraphen zeitzumultiplexen. Bestimmte Ausführungsformen hierin weisen eine Schnittstelle auf, durch die ein CSA (z. B. über das räumliche Programm) anfordern kann, dass ein Teil der Struktur umprogrammiert wird. Die kann dem CSA ermöglichen, sich gemäß dem dynamischen Steuerfluss selbst zu verändern. Bestimmte Ausführungsformen hierin erlauben eine strukturinitiierte Rekonfiguration (z. B. Umprogrammierung). Bestimmte Ausführungsformen hierin stellen einen Satz für Schnittstellen zum Triggern der Konfiguration von innerhalb der Struktur bereit. In einigen Ausführungsformen gibt ein PE eine Rekonfigurationsanfrage basierend auf einer Entscheidung in dem Programmdatenflussgraphen aus. Diese Anfrage kann durch ein Netzwerk zu unserer neuen Konfigurationsschnittstelle gehen, wo sie die Rekonfiguration triggert. Sobald die Rekonfiguration abgeschlossen ist, kann eine Nachricht optional zurückgeleitet werden, die den Abschluss mitteilt. Bestimmte Ausführungsformen eines CSA stellen somit ein Programm (z. B. Datenflussgraph) bereit, das sich an die Rekonfigurationskapazität richtet.

[0144] Fig. 28 veranschaulicht eine Beschleuniger-Kachel **2800**, umfassend ein Array von Verarbeitungselementen und eine Konfigurations- und Ausnahmehandhabungssteuerung **2806** mit einer Rekonfigurationsschaltung **2818**, gemäß Ausführungsformen der Offenbarung. Hier gibt ein Teil der Struktur eine Anforderung zur (Re-) Konfiguration an eine Konfigurationsdomäne aus, z. B. der Konfigurations- und Ausnahmehandhabungssteuerung **2806** und/oder Rekonfigurationsschaltung **2818**. Die Domäne (re)konfiguriert sich selbst, und wenn die Anforderung erfüllt ist, gibt die Konfigurations- und Ausnahmehandhabungssteuerung **2806** und/oder die Rekonfigurationsschaltung **2818** eine Antwort an die Struktur aus, um die Struktur darüber zu informieren, dass die (Re-) Konfiguration abgeschlossen ist. In einer Ausführungsform deaktivieren die Konfigurations- und Ausnahmehandhabungssteuerung **2806** und/oder die Rekonfigurationsschaltung **2818** die Kommunikation während der Zeit, in der die (Re-) Konfiguration läuft, sodass das Programm während der Operation keine Konsistenzprobleme aufweist.

Konfigurationsmodi

[0145] Configure-by-address - In diesem Modus fordert die Struktur das Laden von Konfigurationsdaten von einer bestimmten Adresse direkt an.

[0146] Configure-by-reference - In diesem Modus fordert die Struktur das Laden einer neuen Konfiguration an, z. B. durch eine vorbestimmte Referenz-ID. Dies kann die Bestimmung des zu ladenden Codes vereinfachen, da der Speicherort des Codes abstrahiert wurde.

Konfigurieren einer Vielzahl von Domänen

[0147] Ein CSA kann eine Higher-Level-Konfigurationssteuerung aufweisen, um einen Multicast-Mechanismus zu unterstützen, um Konfigurationsanforderungen (z. B. über ein Netzwerk, das durch die gestrichelte Box angezeigt wird) an mehrere (z. B. verteilte oder lokale) Konfigurationssteuerungen zu übertragen. Dies kann

ermöglichen, dass eine einzelne Konfigurationsanforderung über größere Teile der Struktur repliziert werden kann, z. B. durch Triggern einer breiteren Rekonfiguration.

Ausnahmeaggregatoren

[0148] Bestimmte Ausführungsformen eines CSA können auch eine Ausnahme erfahren (z. B. eine Ausnahmebedingung), z. B. einen Gleitkomma-Underflow. Bei Auftreten dieser Bedingungen kann ein spezieller Handler aufgerufen werden, um das Programm entweder zu korrigieren oder es zu beenden. Bestimmte Ausführungsformen hierin stellen eine Architektur auf Systemebene zum Handhaben von Ausnahmen in räumlichen Strukturen bereit. Da bestimmte räumliche Strukturen die Bereichseffizienz hervorheben, minimieren die Ausführungsformen hierin die Gesamtfläche und stellen gleichzeitig einen allgemeinen Ausnahmemechanismus bereit. Bestimmte Ausführungsformen hierin stellen eine Niederbereichseinrichtung zum Signalisieren von außergewöhnlichen Zuständen bereit, die innerhalb eines CSA (z. B. einem räumlichen Array) auftreten. Bestimmte Ausführungsformen hierin stellen ein Schnittstellen- und Signalisierungsprotokoll zum Übermitteln solcher Ausnahmen sowie eine PE-Ebenen-Ausnahmesemantik bereit. Bestimmte Ausführungsformen hierin sind dedizierte Ausnahmehandhabungskapazitäten, und erfordern z. B. keine explizite Handhabung durch den Programmierer.

[0149] Eine Ausführungsform eines CSA-Ausnahmearchitektur besteht aus vier Teilen, die z. B. in den **Fig. 29** bis **Fig. 30** gezeigt sind. Diese Teile können in einer Hierarchie angeordnet sein, in der Ausnahmen von dem Erzeuger und schließlich bis zu dem Kachel-Level-Ausnahmeaggregator (z. B. Handler) fließen, der sich mit einem Ausnahmebediner, z. B. einem Kern, treffen kann. Die vier Teile können Folgende sein:

1. PE-Ausnahmegenerator
2. Lokales Ausnahme-Netzwerk
3. Mezzanine-Ausnahmeaggregator
4. Kachel-Level-Ausnahmeaggregator

[0150] **Fig. 29** veranschaulicht eine Beschleuniger-Kachel **2900**, umfassend ein Array von Verarbeitungselementen und einen Mezzanine-Ausnahmeaggregator **2902**, der mit einem Kachel-Level-Ausnahmeaggregator **2904** gemäß Ausführungsformen der Offenbarung gekoppelt ist. **Fig. 30** veranschaulicht ein Verarbeitungselement **3000** mit einem Ausnahmegenerator **3044** gemäß Ausführungsformen der Offenbarung.

PE-Ausnahmegenerator

[0151] Das Verarbeitungselement **3000** kann beispielsweise das Verarbeitungselement **900** aus **Fig. 9** aufweisen, wobei ähnliche Nummern ähnliche Komponenten sind, z. B. das lokale Netzwerk **902** und das lokale Netzwerk **3002**. Das zusätzliche Netzwerk **3013** (z. B. Kanal) kann ein Ausnahme-Netzwerk sein. Ein PE kann eine Schnittstelle zu einem Ausnahme-Netzwerk implementieren (z. B. Ausnahme-Netzwerk **3013** (z. B. Kanal) in **Fig. 30**). Zum Beispiel zeigt **Fig. 30** die Mikroarchitektur einer solchen Schnittstelle, wobei das PE einen Ausnahme-Erzeuger **3044** aufweist (z. B. zum Initiieren einer endlichen Ausnahmestatusmaschine (FSM) **3040**, um ein Ausnahmepaket (z. B. BOXID **3042**) an das Ausnahme-Netzwerk auszugeben. BOXID **3042** kann ein eindeutiger Identifizierer für eine Ausnahmeerzeugungsentität (z. B. ein PE oder eine Box) in einem lokalen Ausnahme-Netzwerk sein. Wenn eine Ausnahme erkannt wird, misst der Ausnahmegenerator **3044** das Ausnahme-Netzwerk und tastet BOXID ab, wenn das Netzwerk als frei befunden wird. Ausnahmen können durch viele Bedingungen verursacht werden, z. B. arithmetische Fehler, fehlgeschlagener ECC-Prüfeinschaltzustand usw., aber nicht darauf beschränkt. Es kann jedoch auch sein, dass eine Ausnahmedatenflussoperation mit der Idee von Unterstützungskonstrukten wie Unterbrechungspunkten eingeführt wird.

[0152] Die Initiierung der Ausnahme kann entweder explizit, durch die Ausführung eines vom Programmierer eingegebenen Befehls, oder implizit, wenn eine gehärtete Fehlerbedingung (z. B. ein Gleitkomma-Underflow) erkannt wird, stattfinden. Bei einer Ausnahme kann das PE **3000** in einen Wartezustand eintreten, in dem es darauf wartet, von dem eventuellen Ausnahme-Handler bedient zu werden, z. B. außerhalb des PE **3000**. Die Inhalte des Ausnahmepakets hängen von der Implementierung des bestimmten PE ab, wie unten beschrieben.

Lokales Ausnahme-Netzwerk

[0153] Ein (z. B. lokales) Ausnahme-Netzwerk lenkt Ausnahmepakete vom PE **3000** zum Mezzanine-Ausnahme-Netzwerk. Das Ausnahme-Netzwerk (z. B. 3013) kann ein serielles paketvermitteltes Netzwerk sein, das

aus einem (z. B. einzelnen) Steuerdraht und einem oder mehreren Datendrähten besteht, z. B. in einer Ring- oder Baumtopologie organisiert ist, z. B. für einen PE-Untersatz. Jedes PE kann einen (z. B. Ring-)Stopp im (z. B. lokalen) Ausnahme-Netzwerk aufweisen, in dem es z. B. entscheiden kann, Nachrichten in das Ausnahme-Netzwerk einzuspeisen.

[0154] PE-Endpunkte, die ein Ausnahmepaket einspeisen müssen, können ihren lokalen Ausnahme-Netzwerkaustrittspunkt beobachten. Wenn das Steuersignal „besetzt“ anzeigt, muss das PE warten, um mit dem Einspeisen seines Pakets zu beginnen. Wenn das Netzwerk nicht besetzt ist, d. h. der nachgeschaltete Stopp kein Paket zum Weiterleiten aufweist, geht das PE zum Beginn der Einspeisung über.

[0155] Netzwerkpakete können von variabler oder fester Länge sein. Jedes Paket kann mit einem Feld einer festen Längenkopfzeile beginnen, die das Quell-PE des Pakets identifiziert. Danach kann eine variable Anzahl von PE-spezifischen Feldern folgen, die Informationen enthalten, einschließlich Fehlercodes, Datenwerte oder andere nützliche Statusinformationen.

Mezzanine-A usnahmeaggregator

[0156] Der Mezzanine-Ausnahmeaggregator **2904** ist dafür verantwortlich, ein lokales Ausnahme-Netzwerk zu größeren Paketen zu assemblieren und sie an den Kachel-Level-Ausnahmeaggregator **2902** zu senden. Der Mezzanine-Ausnahmeaggregator **2904** kann das lokale Ausnahmepaket mit seiner eigenen eindeutigen ID voranstellen, z. B. um sicherzustellen, dass Ausnahmenachrichten unzweideutig sind. Der Mezzanine-Ausnahmeaggregator **2904** kann eine Schnittstelle mit einem speziellen virtuellen Nur-Ausnahme-Kanal im Mezzanine-Netzwerk aufweisen, die z. B. die Stillstandfreiheit der Ausnahmen sicherstellen.

[0157] Der Mezzanine-Ausnahmeaggregator **2904** kann auch direkt bestimmte Ausnahmeklassen bedienen. Eine Konfigurationsanforderung von der Struktur kann z. B. aus dem Mezzanine-Netzwerk unter Verwendung von Caches bedient werden, die für den Mezzanine-Netzwerkstopp lokal sind.

Kachel-Level-Ausnahmeaggregator

[0158] Die letzte Stufe des Ausnahmesystems ist der Kachel-Level-Ausnahmeaggregator **2902**. Der Kachel-Level-Ausnahmeaggregator **2902** ist verantwortlich für das Sammeln von Ausnahmen aus den verschiedenen Mezzanine-Level-Ausnahmeaggregatoren (z. B. **2904**) und für das Weiterleiten davon an die geeignete Service-Hardware (z. B. Kern). Daher kann der Kachel-Level-Ausnahmeaggregator **2902** einige interne Tabellen und Steuerungen aufweisen, um bestimmte Nachrichten Handler-Routinen zuzuordnen. Diese Tabellen können entweder direkt oder mit einer kleinen Statusmaschine indiziert werden, um bestimmte Ausnahmen zu lenken.

[0159] Wie der Mezzanine-Ausnahmeaggregator kann der Kachel-Level-Ausnahmeaggregator einige Ausnahmeanforderungen bedienen. Beispielsweise kann er die Umprogrammierung eines Großteils der PE-Struktur als Reaktion auf eine spezifische Ausnahme initiieren.

Extraktionssteuerungen

[0160] Bestimmte Ausführungsformen eines CSA weisen eine oder mehrere Extraktionssteuerungen zum Extrahieren von Daten aus der Struktur auf. Nachstehend werden Ausführungsformen erläutert, wie diese Extraktion schnell vonstatten gehen kann und wie der Ressourcenaufwand der Konfiguration minimiert werden kann. Die Datenextraktion kann für solche wichtigen Aufgaben wie Ausnahmehandhabung und Kontextumschaltung benutzt werden. Bestimmte Ausführungsformen hierin extrahieren Daten aus einer heterogenen räumlichen Struktur durch Einführen von Merkmalen, die extrahierbare Strukturelemente (EFEs) (z. B. PEs, Netzwerksteuerungen und/oder Schalter) mit variablen und dynamisch variablen Mengen des zu extrahierenden Status zulassen.

[0161] Ausführungsformen eines CSA beinhalten ein verteiltes Datenextraktionsprotokoll und eine Mikroarchitektur, um dieses Protokoll zu unterstützen. Bestimmte Ausführungsformen eines CSA beinhalten eine Vielzahl von lokalen Extraktionssteuerungen (LECs), die Programmdaten aus ihrem lokalen Gebiet der räumlichen Struktur unter Verwendung einer Kombination aus einem (z. B. kleinen) Satz von Steuersignalen und dem von der Struktur bereitgestellten Netzwerk streamen. Zustandselemente können an jedem extrahierbaren Strukturelement (EFE) verwendet werden, um Extraktionsketten zu bilden, z. B. um einzelnen EFEs zu erlauben, sich ohne globale Adressierung selbst zu extrahieren.

[0162] Ausführungsformen eines CSA verwenden kein lokales Netzwerk, um Programmdaten zu extrahieren. Ausführungsformen eines CSA weisen eine spezifische Hardware-Unterstützung (z. B. eine Extraktionssteuerung) für die Bildung von z. B. Extraktionsketten auf, und verlassen sich nicht auf Software, um diese Ketten dynamisch zu erstellen, z. B. auf Kosten der Extraktionszeit. Ausführungsformen eines CSA sind nicht rein paketvermittelt und weisen zusätzliche Außer-Band-Steuerdrähte (z. B. wird die Steuerung nicht durch den Datenpfad gesendet, was zusätzliche Zyklen zum Abtasten und Reserialisieren dieser Information erfordert) auf. Ausführungsformen eines CSA verringern die Extraktionslatenz durch Festlegen der Extraktionsreihenfolge und durch Bereitstellen einer expliziten Außer-Band-Steuerung (z. B. um mindestens einen Faktor von zwei), während die Netzwerkkomplexität nicht wesentlich zunimmt.

[0163] Ausführungsformen eines CSA verwenden keinen seriellen Mechanismus zur Datenextraktion, bei der Daten Bit für Bit unter Verwendung eines JTAG-artigen Protokolls in die Struktur gestreamt werden. Ausführungsformen eines CSA benutzen einen grobkörnigen Strukturansatz. In bestimmten Ausführungsformen ist das Hinzufügen einiger Steuerdrähte oder Zustandselemente zu einer 64- oder 32-Bit-orientierten CSA-Struktur kostengünstiger als das Hinzufügen derselben Steuermechanismen zu einer 4- oder 6-Bit-Struktur.

[0164] **Fig. 31** veranschaulicht eine Beschleuniger-Kachel **3100**, umfassend ein Array von Verarbeitungselementen und eine lokale Extraktionssteuerung (**3102**, **3106**) gemäß Ausführungsformen der Offenbarung. Jedes PE, jeder Netzwerkcontroller und jeder Schalter können extrahierbare Strukturelemente (EFEs) sein, die z. B. durch Ausführungsformen der CSA-Architektur konfiguriert (z. B. programmiert) werden.

[0165] Ausführungsformen eines CSA weisen Hardware auf, die eine effiziente, verteilte Extraktion mit niedriger Latenzzeit aus einer heterogenen räumlichen Struktur bereitstellt. Dies kann gemäß vier Techniken erreicht werden. Zuerst wird eine Hardwareentität, die lokale Extraktionssteuerung (LEC - Local Extraction Controller) benutzt, wie in **Fig. 31** - **Fig. 33**. Eine LEC kann Befehle von einem Host (z. B. einem Prozessorkern) annehmen, z. B. einen Datenstream aus dem räumlichen Array extrahieren, und diese Daten zurück in den virtuellen Speicher zur Überprüfung durch den Host schreiben. Zweitens kann ein Extraktionsdatenpfad enthalten sein, der z. B. so breit wie die ursprüngliche Breite der PE-Struktur ist und der über die PE-Struktur überlagert sein kann. Drittens können neue Steuersignale in der PE-Struktur empfangen werden, die den Extraktionsprozess anleiten. Viertens können Zustandselemente an jedem konfigurierbaren Endpunkt angeordnet sein (z. B. in einem Register), die den Status benachbarter EFEs verfolgen, so dass jedes EFE ohne zusätzliche Steuersignale seinen Status eindeutig exportiert. Diese vier mikroarchitektionischen Merkmale können einem CSA das Extrahieren von Daten aus Ketten der EFEs ermöglichen. Um eine geringe Datenextraktionslatenz zu erhalten, können bestimmte Ausführungsformen das Extraktionsproblem durch Einschließen mehrerer (z. B. vieler) LECs und EFE-Ketten in die Struktur partitionieren. Zur Extraktion können diese unabhängig voneinander arbeiten, um die Struktur unabhängig zu extrahieren, z. B. um die Latenz drastisch zu reduzieren. Als ein Ergebnis dieser Kombinationen kann ein CSA ein vollständige Statusabbild (z. B. in Hunderten von Nanosekunden) durchführen.

[0166] **Fig. 32A-32C** veranschaulichen eine lokale Extraktionssteuerung **3202**, die ein Datenpfad-Netzwerk gemäß Ausführungsformen der Offenbarung konfigurieren. Das dargestellte Netzwerk weist mehrere Multiplexer (z. B. Multiplexer **3206**, **3208**, **3210**) auf, die konfiguriert werden können (z. B. über zugehörige Steuersignale), um einen oder mehrere Datenpfade (z. B. von PEs) miteinander zu verbinden. **Fig. 32A** veranschaulicht das Netzwerk **3200** (z. B. Struktur), das für eine bisherige Operation oder Programm konfiguriert (z. B. eingestellt) wurde. **Fig. 32B** zeigt die lokale Extraktionssteuerung **3202** (die beispielsweise eine Netzwerkschnittstellenschaltung **3204** zum Senden und/oder Empfangen von Signalen aufweist), die ein Extraktionssignal abtastet und alle von der LEC gesteuerten PEs in den Extraktionsmodus eingibt. Das letzte PE in der Extraktionskette (oder ein Extraktions-Terminator) kann die Extraktionskanäle (z. B. Bus) mastern und Daten entweder gemäß (1) Signalen von der LEC oder (2) intern erzeugten Signalen (z. B. von einem PE) senden. Nach Abschluss kann ein PE sein Abschlussflag setzen und z. B. dem nächsten PE das Extrahieren seiner Daten ermöglichen. **Fig. 32C** zeigt, dass das am weitesten entfernte PE den Extraktionsprozess abgeschlossen hat und als Ergebnis sein Extraktionszustandsbit oder -bits gesetzt hat, die z. B. die Muxe in das benachbarte Netzwerk schwingen, um es dem nächsten PE zu ermöglichen, mit dem Extraktionsprozess zu beginnen. Das extrahierte PE kann seine normale Operation wieder aufnehmen. In einigen Ausführungsformen kann das PE deaktiviert bleiben, bis eine andere Aktion unternommen wird. In diesen Figuren sind die Multiplexernetzwerke Analoga des „Schalters (oder Switches)“, der in bestimmten Figuren gezeigt ist (z. B. **Fig. 6**).

[0167] Die Folgenden Abschnitte beschreiben die Operation der verschiedenen Komponenten von Ausführungsformen eines Extraktionsnetzwerks.

Lokale Extraktionssteuerung

[0168] **Fig. 33** veranschaulicht eine Extraktionssteuerung **3302** gemäß Ausführungsformen der Offenbarung. Eine lokale Extraktionssteuerung (Local Extraction Controller, LEC) kann die Hardwareentität sein, die dafür verantwortlich ist, Extraktionsbefehle zu akzeptieren, den Extraktionsprozess mit den EFEs zu koordinieren und/oder extrahierte Daten z. B. in einem virtuellen Speicher zu speichern. In dieser Eigenschaft kann die LEC ein sequentieller Spezialzweck-Mikrocontroller sein.

[0169] Die LEC-Operation kann beginnen, wenn sie einen Zeiger auf einen Puffer (z. B. im virtuellen Speicher) empfängt, in den der Strukturstatus und optional ein Befehl geschrieben wird, wie stark die Struktur extrahiert werden wird. Je nach der LEC-Mikroarchitektur kann dieser Zeiger (z. B. im Zeigerregister **3304** gespeichert) entweder über ein Netzwerk oder über einen Speichersystemzugriff auf die LEC gelangen. Wenn sie einen solchen Zeiger (z. B. Befehl) empfängt, geht die LEC zum Extraktionsstatus von dem Teil der Struktur, für den sie verantwortlich ist. Die LEC kann diese extrahierten Daten aus der Struktur in den Puffer streamen, der durch den externen Anrufer bereitgestellt wird.

[0170] Zwei unterschiedliche Mikroarchitekturen für die LEC sind in **Fig. 31** gezeigt. Die erste platziert die LEC **3102** an der Speicherschnittstelle. In diesem Fall kann die LEC direkte Anfragen an das Speichersystem zum Schreiben von extrahierten Daten stellen. Im zweiten Fall ist die LEC **3106** auf einem Speichernetzwerk angeordnet, in dem sie Anforderungen an den Speicher nur indirekt stellen kann. In beiden Fällen kann die logische Operation der LEC unverändert bleiben. In einer Ausführungsform werden die LECs über den Wunsch informiert, Daten aus der Struktur zu extrahieren, beispielsweise durch einen Satz von (z. B. OS-sichtbaren) Steuerstatusregistern, die verwendet werden, um einzelne LECs über neue Befehle zu informieren.

Zusätzliche Außer-Band-Steuerkanäle (z. B. Drähte)

[0171] In bestimmten Ausführungsformen verlässt sich die Extraktion auf 2 bis 8 zusätzliche Außer-Bandsignale, um die Konfigurationsgeschwindigkeit wie unten definiert zu verbessern. Durch die LEC angesteuerte Signale können als LEC markiert sein. Durch EFE (z. B. PE) angesteuerte Signale können als EFE bezeichnet werden. Die Konfigurationssteuerung **3302** kann die folgenden Steuerkanäle aufweisen, z. B. den LEC_EXTRACT Steuerkanal **3406**, LEC_START Steuerkanal **3308**, LEC_STROBE Steuerkanal **3310** und den EFE_COMPLETE Steuerkanal **3312**, wobei Beispiele von jedem einzelnen in Tabelle 3 unten besprochen sind.

TABELLE 3: Extraktionskanäle

LEC_EXTRACT	Optional signal asserted by the LEC during extraction process. Lowering this signal causes normal operation to resume.
LEC_START	Signal denoting start of extraction, allowing setup of local EFE state
LEC_STROBE	Optional strobe signal for controlling extraction related state machines at EFEs. EFEs may generate this signal internally in some implementations.
EFE_COMPLETE	Optional signal strobed when EFE has completed dumping state. This helps LEC identify the completion of individual EFE dumps.

[0172] Im Allgemeinen kann die Handhabung der Extraktion dem Implementierer eines bestimmten EFE überlassen werden. Zum Beispiel kann ein auswählbares Funktions-EFE eine Vorkehrung zum Abbilden von Registern unter Verwendung eines existierenden Datenpfads aufweisen, während eine festes Funktions-EFE einfach einen Multiplexer aufweisen kann.

[0173] Aufgrund der langen Drahtverzögerungen beim Programmieren langer EFE-Sätze kann das LEC_STROBE Signal als eine Takt-/Latch-Aktivierung für die EFE-Komponenten behandelt werden. Da dieses Signal als ein Takter verwendet wird, beträgt in einer Ausführungsform der Arbeitszyklus der Leitung höchstens 50%. Als Ergebnis wird der Extraktionsdurchsatz in etwa halbiert. Optional kann ein zweites LEC_STROBE Signal hinzugefügt werden, um eine kontinuierliche Extraktion zu ermöglichen.

[0174] In einer Ausführungsform wird nur LEC_START strikt an eine unabhängige Kopplung (z. B. Draht) kommuniziert, zum Beispiel können Steuerkanäle über ein existierendes Netzwerk überlagert werden.

Wiederverwendung von Netzwerkressourcen

[0175] Zum Reduzieren des Datenextraktionsaufwands nutzen bestimmte Ausführungsformen eines CSA die vorhandene Netzwerkinfrastruktur zur Kommunikation von Extraktionsdaten. Eine LEC kann sowohl eine Chipebenen-Speicherhierarchie als auch Strukturebenen-Kommunikationsnetzwerke zum Bewegen von Daten von der Struktur in den Speicher nutzen. Als Ergebnis trägt die Extraktionsinfrastruktur in bestimmten Ausführungsformen eines CSA nicht mehr als 2% zur gesamten Strukturfläche und zur Gesamtleistung bei.

[0176] Die Wiederverwendung von Netzwerkressourcen in bestimmten Ausführungsformen eines CSA kann ein Netzwerk dazu veranlassen, einige Hardwareunterstützung für ein Extraktionsprotokoll aufzuweisen. Leitungsvermittelte Netzwerke erfordern von bestimmten Ausführungsformen eines CSA, dass eine LEC ihre Multiplexer auf eine spezifische Weise für die Konfiguration setzt, wenn das Signal ‚LEC_START' bestätigt wird. Paketvermittelte Netzwerke erfordern keine Erweiterung, obwohl LEC-Endpunkte (z. B. Extraktions-Terminatoren) eine spezifische Adresse in dem paketvermittelten Netzwerk verwenden. Die Netzwerkwiederverwendung ist optional und einige Ausführungsformen finden ggf. eigens vorgesehene Konfigurationsbusse angemessener.

Per EFE-Status

[0177] Jedes EFE kann ein Bit halten, das angibt, ob es seinen Status exportiert hat oder nicht. Dieses Bit kann deaktiviert werden, wenn das Konfigurationsstartsignal angesteuert wird, und dann aktiviert werden, sobald das bestimmte EFE die Extraktion beendet hat. In einem Extraktionsprotokoll sind die EFEs angeordnet, um Ketten mit dem EFE-Extraktionszustandsbit zu bilden, das die Topologie der Kette bestimmt. Ein EFE kann das Extraktionszustandsbit des unmittelbar angrenzenden EFE lesen. Wenn dieses benachbarte EFE sein Extraktionsbit gesetzt hat und das aktuelle EFE dies nicht tut, kann das EFE bestimmen, dass es den Extraktionsbus besitzt. Wenn ein EFE seinen letzten Datenwert abbildet, kann es das Signal ‚EFE_DONE' ansteuern und sein Extraktionsbit setzen, z. B. indem es vorgeschaltete EFEs die Konfiguration für die Extraktion ermöglicht. Das Netzwerk benachbart des EFE kann dieses Signal beobachten und auch seinen Status einstellen, um den Übergang zu bewältigen. Als ein Basisfall für den Extraktionsprozess kann ein Extraktions-Terminator (z. B. Extraktions-Terminator **3104** für LEC **3102** oder Extraktions-Terminator **3108** für LEC **3106** in **Fig. 22**), der bestätigt, dass die Extraktion abgeschlossen ist, am Ende einer Kette enthalten sein.

[0178] EFE-intem kann dieses Bit zum Ansteuern der flusssteuerungsbereiten Signale verwendet werden. Wenn zum Beispiel das Extraktionsbit deaktiviert wird, können Netzwerksteuersignale automatisch auf einen Wert geklemmt werden, der verhindert, dass Daten fließen, während innerhalb der PEs keine Operationen oder andere Aktionen geplant werden.

Behandeln von Pfaden mit hoher Verzögerung

[0179] Eine Ausführungsform einer LEC kann ein Signal über eine lange Distanz, z. B. durch viele Multiplexer und mit vielen Lasten, ansteuern. Daher kann es für ein Signal schwierig sein, an einem entfernten EFE innerhalb eines kurzen Taktzyklus einzugehen. In bestimmten Ausführungsformen sind die Extraktionssignale in einer bestimmten Division (z. B. einem Bruchteil von) der Haupttaktfrequenz (z. B. CSA), um eine digitale Zeitdisziplin bei der Extraktion sicherzustellen. Die Taktteilung kann in einem Außer-Band-Signalisierungsprotokoll benutzt werden und erfordert keine Modifikation des Haupttaktbaums.

Sicherstellen des konsistenten Strukturverhaltens während der Extraktion

[0180] Da bestimmte Extraktionsschemata verteilt sind und aufgrund von Programm- und Speichereffekten eine nicht-deterministische Zeitsteuerung aufweisen, können verschiedene Elemente der Struktur zu unterschiedlichen Zeiten extrahiert werden. Während des Ansteuerns von LEC_EXTRACT können alle Netzwerkflusssignale logisch niedrig angesteuert werden, z. B. durch Einfrieren der Operation eines bestimmten Segments der Struktur.

[0181] Ein Extraktionsprozess kann nicht destruktiv sein. Daher kann ein Satz von PEs als operational betrachtet werden, sobald die Extraktion abgeschlossen wurde. Eine Erweiterung eines Extraktionsprotokolls kann den PEs ermöglichen, nach der Extraktion deaktiviert zu werden. Alternativ wird die beginnende Konfiguration während des Extraktionsprozesses in Ausführungsformen einen ähnlichen Effekt haben.

Einzel-PE-Extraktion

[0182] In einigen Fällen kann es zweckmäßig sein, ein einzelnes PE zu extrahieren. In diesem Fall kann ein optionales Adresssignal als Teil des Beginns des Extraktionsprozesses angesteuert werden. Dies kann dem für die Extraktion angezielten PE ermöglichen, direkt aktiviert zu werden. Sobald dieses PE extrahiert wurde, kann der Extraktionsprozess durch das Absenken des LEC_EXTRACT Signals enden. Auf diese Weise kann ein einzelnes PE selektiv extrahiert werden, z. B. durch die lokale Extraktionssteuerung.

Handhaben des Extraktionsgegendrucks

[0183] In einer Ausführungsform, in der die LEC extrahierte Daten in den Speicher schreibt (z. B. zum Nachverarbeiten z. B. in Software), kann sie Gegenstand einer begrenzten Speicherbandbreite sein. In dem Fall, dass die LEC ihre Pufferkapazität erschöpft oder erwartet, ihre Pufferkapazität zu erschöpfen, kann sie das Abtasten des LEC_STROBE Signals anhalten, bis das Pufferproblem gelöst ist.

[0184] Es sei zu beachten, dass in bestimmten Figuren (z. B. **Fig. 22**, **Fig. 25**, **Fig. 26**, **Fig. 28**, **Fig. 29** und **Fig. 31**) Kommunikationen schematisch dargestellt sind. In bestimmten Ausführungsformen entstehen diese Kommunikationen über das (z. B. Interconnect-) Netzwerk.

Flussdiagramme

[0185] **Fig. 34** veranschaulicht ein Flussdiagramm **3400** gemäß Ausführungsformen der Offenbarung. Der dargestellte Fluss **3400** beinhaltet das Decodieren eines Befehls mit einem Decodierer eines Kerns eines Prozessors in einen decodierten Befehl **3402**; Ausführen des decodierten Befehls mit einer Ausführungseinheit des Kerns des Prozessors zum Durchführen einer ersten Operation **3404**; Empfangen einer Eingabe eines Datenflussgraphen, der mehrere Knoten **3406** umfasst; Überlagern des Datenflussgraphen über ein Array aus Verarbeitungselementen des Prozessors, wobei jeder Knoten als Datenflussoperator in dem Array von Verarbeitungselementen **3408** repräsentiert ist; und Durchführen einer zweiten Operation des Datenflussgraphen mit dem Array aus Verarbeitungselementen, wenn ein eingehender Operandensatz an dem Array aus Verarbeitungselementen **3410** eingeht.

[0186] **Fig. 35** veranschaulicht ein Flussdiagramm **3500** gemäß Ausführungsformen der Offenbarung. Der dargestellte Fluss **3500** beinhaltet das Decodieren eines Befehls mit einem Decodierer eines Kerns eines Prozessors in einen decodierten Befehl **3502**; Ausführen des decodierten Befehls mit einer Ausführungseinheit des Kerns des Prozessors zum Durchführen einer ersten Operation **3504**; Empfangen einer Eingabe eines Datenflussgraphen, der mehrere Knoten **3506** umfasst; Überlagern des Datenflussgraphen über mehrere Verarbeitungselemente des Prozessors und ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen des Prozessors, wobei jeder Knoten einen Datenflussoperator in den mehreren Verarbeitungselementen **3508** repräsentiert; und Durchführen einer zweiten Operation des Datenflussgraphen mit dem Zwischenverbindungsnetz und den mehreren Verarbeitungselementen, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen **3510** eingeht.

KURZDARSTELLUNG

[0187] Supercomputing auf der ExaFLOP-Skala kann eine Herausforderung im Hochleistungsrechnen sein, eine Herausforderung, die von konventionellen von Neumann-Architekturen wahrscheinlich nicht erfüllt wird. Zum Erreichen von ExaFLOPs stellen Ausführungsformen eines CSA ein heterogenes räumliches Array bereit, das auf die direkte Ausführung von (z. B. vom Kompilierer erzeugten) Datenflussgraphen abzielt. Zusätzlich zu dem Auslegen der Architekturprinzipien von Ausführungsformen eines CSA beschreibt und bewertet das Vorstehende auch Ausführungsformen eines CSA, der eine Leistung und Energie von mehr als dem 10-fachen gegenüber existierenden Produkten zeigte. Kompiliererzeugter Code kann bedeutende Leistungs- und Energiezugewinne gegenüber Roadmap-Architekturen haben. Als eine heterogene, parametrische Architektur können Ausführungsformen eines CSA leicht an alle Computeranwendungen angepasst werden. Zum Beispiel könnte eine mobile Version von CSA auf 32 Bits abgestimmt sein, während ein auf maschinelles Lernen fokussiertes Array eine signifikante Anzahl von vektorisierten 8-Bit-Multiplikationseinheiten aufweisen könnte. Die Hauptvorteile von Ausführungsformen eines CSA sind eine hohe Leistung und eine extreme Energieeffizienz, Eigenschaften, die für alle Formen des Rechnens von Supercomputing und Rechenzentrum bis zum Internet der Dinge relevant sind.

[0188] In einer Ausführungsform weist ein Prozessor mehrere Verarbeitungselemente auf; und ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen zum Empfangen einer Eingabe eines Datenflussgraphen, umfassend mehrere Knoten, wobei der Datenflussgraph in das Zwischenverbindungsnetz und die mehreren Verarbeitungselemente zu überlagern ist, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist, und die mehreren Verarbeitungselemente eine atomare Operation durchzuführen haben, wenn ein eingehender Operand bei den mehreren Verarbeitungselementen eingeht.

[0189] Ein Verarbeitungselement der Vielzahl von Verarbeitungselementen kann die Ausführung anhalten, wenn ein Gegendrucksignal von einem nachgeschalteten Verarbeitungselement anzeigt, dass kein Speicher in dem nachgeschalteten Verarbeitungselement für eine Ausgabe des Verarbeitungselements verfügbar ist. Der Prozessor kann ein Flusssteuerpfad-Netzwerk zum Übertragen des Gegendrucksignals gemäß dem Datenflussgraphen aufweisen. Ein Datenfluss-Token kann bewirken, dass eine Ausgabe von einem Datenflussoperator, der das Datenfluss-Token empfängt, zu einem Eingabepuffer eines bestimmten Verarbeitungselements der mehreren Verarbeitungselemente gesendet wird. Die atomare Operation kann einen Speicherzugriff beinhalten, und die mehreren Verarbeitungselemente umfassen einen Speicherzugriffs-Datenflussoperator, der den Speicherzugriff nicht durchzuführen hat, bis er ein Speicherabhängigkeits-Token von einem logisch vorherigen Datenflussoperator empfängt. Die mehreren Verarbeitungselemente können einen ersten Typ von Verarbeitungselement und einen zweiten, unterschiedlichen Typ von Verarbeitungselement aufweisen.

[0190] In einer Ausführungsform beinhaltet ein Prozessor mehrere Verarbeitungselemente; ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen, um einen Eingang eines Datenflussgraphen zu empfangen, der mehrere Knoten umfasst, wobei der Datenflussgraph in das Zwischenverbindungsnetz und die mehreren Verarbeitungselemente zu überlagern ist, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist und die mehreren Verarbeitungselemente eine Operation durchzuführen haben, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht; und eine Transaktionssteuerung, um mehrere Speicherzugriffe zu gruppieren, die mit der Operation in Zusammenhang stehen.

[0191] Die Transaktionssteuerung kann die mehreren Speicherzugriffe in eine Transaktion durch Markieren, mit einem Transaktionsidentifizierer, einer Cache-Zeile, die durch die Operation zu modifizieren ist, gruppieren. Eine erste Nachricht kann zu der Transaktionssteuerung in Verbindung mit einem Start der Transaktion gesendet werden. Eine zweite Nachricht kann zu der Transaktionssteuerung in Verbindung mit einem Ende der Transaktion gesendet werden. Die Transaktionssteuerung kann den Transaktionsidentifizierer als Reaktion auf die zweite Nachricht aus der Cache-Zeile löschen. Die mehreren Speicherzugriffe können einen Lesezugriff durch ein erstes der mehreren Verarbeitungselemente einschließen. Die mehreren Speicherzugriffe können einen Schreibzugriff durch ein zweites der mehreren Verarbeitungselemente einschließen. Das erste und das zweite der mehreren Verarbeitungselemente sind unterschiedliche Verarbeitungselemente. Das erste und das zweite der mehreren Verarbeitungselemente sind das gleiche Verarbeitungselement.

[0192] In einer Ausführungsform beinhaltet ein Prozessor mehrere Verarbeitungselemente; ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen, um einen Eingang eines Datenflussgraphen zu empfangen, der mehrere Knoten umfasst, wobei der Datenflussgraph in das Zwischenverbindungsnetz und die mehreren Verarbeitungselemente zu überlagern ist, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist und die mehreren Verarbeitungselemente eine Operation durchzuführen haben, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht; und einen Cache, wobei der Cache in einem Speicheruntersystem einzuschließen ist, das Speicheruntersystem auch einen Speicher einzuschließen hat, in dem mehrere alte Datenwerte zu speichern sind, um eine Ausführung vom Start einer Epoche zu wiederholen, wobei die Epoche die Operation einzuschließen hat.

[0193] Der erste der mehreren alten Datenwerte kann bis zum Ende der Epoche im Speicher bewahrt werden, als Reaktion darauf, dass ein entsprechender neuer Datenwert in einer Zeile des Cache durch eines der mehreren Verarbeitungselemente gespeichert wird. Der neue Datenwert kann von einem Schreibzugriff von einem der mehreren Verarbeitungselemente sein. Der erste der mehreren alten Datenwerte kann gemäß einem Cache-Kohärenzprotokoll bewahrt werden.

[0194] In einer Ausführungsform kann ein Verfahren beinhalten Empfangen eines Eingangs eines Datenflussgraphen, der mehrere Knoten umfasst; Überlagern des Datenflussgraphen in mehrere Verarbeitungselemente des Prozessors und ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen des Prozessors, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert

ist; Durchführen einer Operation des Datenflussgraphen mit dem Zwischenverbindungsnetz und den mehreren Verarbeitungselementen, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht; und Bewahren mehrerer alter Datenwerte in einem Speicher während einer Epoche, wobei die Epoche ein Schreiben eines neuen Datenwerts von einem der mehreren Verarbeitungselemente einschließt, wobei der neue Wert einem der mehreren alten Datenwerte entspricht.

[0195] Das Verfahren kann auch beinhalten Erhalten, durch einen Cache gemäß einem Cache-Kohärenzprotokoll, des Besitzes einer Cache-Zeile, in die der neue Datenwert zu speichern ist. Das Verfahren kann auch beinhalten, als Reaktion auf das Bestimmen, dass die Cache-Zeile kohärent in Besitz des Cache ist, Schreiben der Cache-Zeile in den Speicher. Das Verfahren kann auch beinhalten Aktualisieren der Cache-Zeile zu dem neuen Wert nach dem Schreiben der Cache-Zeile in den Speicher. Das Verfahren kann auch beinhalten Ändern der Cache-Zeile von kohärent in Besitz zu spekulativ in Besitz nach dem Schreiben der Cache-Zeile in den Speicher. Das Verfahren kann auch beinhalten, als Reaktion auf das Bestimmen, dass die Cache-Zeile spekulativ in Besitz des Cache ist, Aktualisieren der Cache-Zeile zu dem neuen Wert, ohne ein Schreiben der Zeile in den Speicher.

[0196] In einer anderen Ausführungsform umfasst ein Verfahren Empfangen einer Eingabe eines Datenflussgraphen umfassend mehrere Knoten; Überlagern des Datenflussgraphen über mehrere Verarbeitungselemente des Prozessors und eines Zwischenverbindungsnetzes zwischen den mehreren Verarbeitungselementen des Prozessors, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist; und Durchführen einer atomaren Operation des Datenflussgraphen mit dem Zwischenverbindungsnetz und den mehreren Verarbeitungselementen durch einen jeweiligen eingehenden Operandensatz, der an jedem der Datenflussoperatoren der mehreren Verarbeitungselemente eingeht. Das Verfahren kann das Anhalten der Ausführung durch ein Verarbeitungselement der mehreren Verarbeitungselemente beinhalten, wenn ein Gegendrucksignal von einem nachgeschalteten Verarbeitungselement anzeigt, dass kein Speicher in dem nachgeschalteten Verarbeitungselement für eine Ausgabe des Verarbeitungselements verfügbar ist. Das Verfahren kann das Senden des Gegendrucksignals auf einem Flusssteuerpfad-Netzwerk gemäß dem Datenflussgraphen beinhalten. Ein Datenfluss-Token kann bewirken, dass eine Ausgabe von einem Datenflussoperator, der das Datenfluss-Token empfängt, zu einem Eingabepuffer eines bestimmten Verarbeitungselements der mehreren Verarbeitungselemente gesendet wird. Das Verfahren kann umfassen, dass kein Speicherzugriff ausgeführt wird, bis ein Speicherabhängigkeits-Token von einem logisch vorherigen Datenflussoperator empfangen wird, wobei die atomare Operation den Speicherzugriff umfasst und die mehreren Verarbeitungselemente einen Speicherzugriffs-Datenflussoperator umfassen. Das Verfahren kann das Bereitstellen eines ersten Typs von Verarbeitungselement und eines zweiten, unterschiedlichen Typs von Verarbeitungselement beinhalten.

[0197] In noch einer anderen Ausführungsform weist eine Vorrichtung ein Datenpfad-Netzwerk zwischen mehreren Verarbeitungselementen auf; und ein Flusssteuerpfad-Netzwerk zwischen den mehreren Verarbeitungselementen, wobei das Datenpfad-Netzwerk und das Flusssteuerpfad-Netzwerk eine Eingabe eines Datenflussgraphen empfangen müssen, der mehrere Knoten umfasst, wobei der Datenflussgraph in das Datenpfad-Netzwerk und das Flussteuerpfad-Netzwerk und die mehreren Verarbeitungseinheiten zu überlagern ist, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist, und die mehreren Verarbeitungselemente eine atomare Operation durch einen jeweiligen eingehenden Operandensatz durchzuführen haben, der an jedem der Datenflussoperatoren der mehreren Verarbeitungselemente eingeht. Das Flusssteuerpfad-Netzwerk kann Gegendrucksignale zu mehreren Datenflussoperatoren gemäß dem Datenflussgraphen tragen. Ein Datenfluss-Token, das auf dem Datenpfad-Netzwerk zu einem Datenflussoperator gesendet wurde, kann bewirken, dass eine Ausgabe von dem Datenflussoperator, der an einen Eingabepuffer eines bestimmten Verarbeitungselements der mehreren Verarbeitungselemente auf dem Datenpfad-Netzwerk gesendet wird. Das Datenpfad-Netzwerk kann ein statisches leitungsvermitteltes Netzwerk sein, um den jeweiligen Eingabeoperandensatz gemäß dem Datenflussgraphen zu jedem der Datenflussoperatoren zu tragen. Das Flussteuerpfad-Netzwerk kann ein Gegendrucksignal gemäß dem Datenflussgraphen von einem nachgeschalteten Verarbeitungselement übertragen, um anzuzeigen, dass kein Speicher in dem nachgeschalteten Verarbeitungselement für eine Ausgabe des Verarbeitungselements verfügbar ist. Mindestens ein Datenpfad des Datenpfad-Netzwerks und mindestens ein Flussteuerpfad des Flussteuerpfad-Netzwerks können eine kanalisierte Schaltung mit Gegendrucksteuerung bilden. Das Flusssteuerpfad-Netzwerk kann mindestens zwei der mehreren Verarbeitungselemente zeitverschachteln.

[0198] In einer weiteren Ausführungsform beinhaltet ein Verfahren das Empfangen einer Eingabe eines Datenflussgraphen, umfassend mehrere Knoten; und Überlagern des Datenflussgraphen über mehrere Verarbeitungselemente eines Prozessors, eines Datenpfad-Netzwerks zwischen die mehreren Verarbeitungselemente

und eines Flusssteuerpfad-Netzwerk zwischen die mehreren Verarbeitungselemente, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist. Das Verfahren kann das Tragen des Gegendrucksignals mit dem Flusssteuerpfad-Netzwerk zu mehreren Datenflussoperatoren gemäß dem Datenflussgraphen beinhalten. Das Verfahren kann das Senden eines Datenfluss-Token auf dem Datenpfad-Netzwerk beinhalten, um zu bewirken, dass eine Ausgabe von dem Datenflussoperator an einen Eingabepuffer eines bestimmten Verarbeitungselements der mehreren Verarbeitungselemente auf dem Datenpfad-Netzwerk gesendet wird. Das Verfahren kann das Einstellen mehrerer Schalter des Datenpfad-Netzwerks und/oder mehrerer Schalter des Flusssteuerpfad-Netzwerks zum Tragen des jeweiligen Eingabeoperandensatzes zu jedem der Datenflussoperatoren gemäß dem Datenflussgraphen beinhalten, wobei das Datenpfad-Netzwerk ein statisches leitungsvermitteltes Netzwerk ist. Das Verfahren kann das Übertragen eines Gegendrucksignals mit dem Flusssteuerpfad-Netzwerk gemäß dem Datenflussgraphen von einem nachgeschalteten Verarbeitungselement beinhalten, um anzuzeigen, dass kein Speicher in dem nachgeschalteten Verarbeitungselement für eine Ausgabe des Verarbeitungselements verfügbar ist. Das Verfahren kann das Bilden einer kanalisierten Schaltung mit Gegendrucksteuerung mit mindestens einem Datenpfad des Datenpfad-Netzwerks und mindestens einem Flusssteuerpfad des Flussteuerpfad-Netzwerks beinhalten.

[0199] In noch einer anderen Ausführungsform weist ein Prozessor mehrere Verarbeitungselemente auf; und eine Netzwerkeinrichtung zwischen den mehreren Verarbeitungselementen zum Empfangen einer Eingabe eines Datenflussgraphen, umfassend mehrere Knoten, wobei der Datenflussgraph in die Netzwerkeinrichtung und die mehreren Verarbeitungseinheiten überlagert wird, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist, und die mehreren Verarbeitungselemente eine atomare Operation durch einen jeweiligen eingehenden Operandensatz durchzuführen haben, der an jedem der Datenflussoperatoren der mehreren Verarbeitungselemente eingeht.

[0200] In einer weiteren Ausführungsform weist eine Vorrichtung eine Datenpfad-Einrichtung zwischen mehreren Verarbeitungselementen auf; und eine Flusssteuerpfad-Einrichtung zwischen den mehreren Verarbeitungselementen, wobei die Datenpfad-Einrichtung und die Flusssteuerpfad-Einrichtung eine Eingabe eines Datenflussgraphen zu empfangen haben, der mehrere Knoten umfasst, wobei der Datenflussgraph in die Datenpfad-Einrichtung und die Flusssteuerpfad-Einrichtung und die mehreren Verarbeitungseinheiten zu überlagern ist, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist, und die mehreren Verarbeitungselemente eine atomare Operation durch einen jeweiligen eingehenden Operandensatz durchzuführen haben, der an jedem der Datenflussoperatoren der mehreren Verarbeitungselemente eingeht.

[0201] In einer Ausführungsform weist ein Prozessor ein Array aus Verarbeitungselementen zum Empfangen einer Eingabe eines Datenflussgraphen, der mehrere Knoten umfasst, auf, wobei der Datenflussgraph in das Array aus Verarbeitungselementen zu überlagern ist, wobei jeder Knoten für einen Datenflussoperator in dem Array aus Verarbeitungselementen steht, und das Array aus Verarbeitungselementen eine atomare Operation durchzuführen hat, wenn ein eingehender Operandensatz am Array aus Verarbeitungselementen eingeht. Das Array aus Verarbeitungselementen kann die zweite Operation nicht durchführen, bis der eingehende Operandensatz an dem Array aus Verarbeitungselementen eingeht und der Speicher in dem Array aus Verarbeitungselementen zur Ausgabe der atomaren Operation verfügbar ist. Das Array aus Verarbeitungselementen kann ein Netzwerk (oder einen oder mehrere Kanäle) aufweisen, um die Datenfluss-Tokens und Steuertokens zu mehreren Datenflussoperatoren zu tragen. Die atomare Operation kann einen Speicherzugriff aufweisen, und das Array aus Verarbeitungselementen kann einen Speicherzugriff-Datenflussoperator aufweisen, der den Speicherzugriff nicht durchzuführen hat, bis er ein Speicherabhängigkeits-Token von einem logisch vorherigen Datenflussoperator empfängt. Jedes Verarbeitungselement kann nur eine oder zwei Operationen des Datenflussgraphen durchführen.

[0202] In einer weiteren Ausführungsform beinhaltet ein Verfahren das Empfangen einer Eingabe eines Datenflussgraphen, der mehrere Knoten umfasst; Überlagern des Datenflussgraphen über ein Array aus Verarbeitungselementen des Prozessors, wobei jeder Knoten als ein Datenflussoperator in dem Array von Verarbeitungselementen repräsentiert ist; und Durchführen einer atomaren Operation des Datenflussgraphen mit dem Array aus Verarbeitungselementen, wenn ein eingehender Operandensatz an dem Array aus Verarbeitungselementen eingeht. Das Array aus Verarbeitungselementen kann die atomare Operation nicht durchführen, bis der eingehende Operandensatz an dem Array aus Verarbeitungselementen eingeht und der Speicher in dem Array aus Verarbeitungselementen zur Ausgabe der atomaren Operation verfügbar ist. Das Array aus Verarbeitungselementen kann ein Netzwerk aufweisen, das die Datenfluss-Tokens und Steuertokens zu mehreren Datenflussoperatoren trägt. Die atomare Operation kann einen Speicherzugriff aufweisen, und das Array aus Verarbeitungselementen kann einen Speicherzugriff-Datenflussoperator, der den Speicherzugriff nicht

durchführen darf, bis er ein Speicherabhängigkeits-Token von einem logisch vorherigen Datenflussoperator empfängt, aufweisen. Jedes Verarbeitungselement kann nur eine oder zwei Operationen des Datenflussgraphen durchführen.

[0203] In noch einer anderen Ausführungsform veranlasst ein nichtflüchtiges maschinenlesbares Medium, das Code speichert, der, wenn er von einer Maschine ausgeführt wird, die Maschine veranlasst, ein Verfahren durchzuführen, aufweisend das Empfangen einer Eingabe eines Datenflussgraphen, der mehrere Knoten umfasst; Überlagern des Datenflussgraphen über ein Array aus Verarbeitungselementen des Prozessors, wobei jeder Knoten als ein Datenflussoperator in dem Array von Verarbeitungselementen repräsentiert ist; und Durchführen einer atomaren Operation des Datenflussgraphen mit dem Array aus Verarbeitungselementen, wenn ein eingehender Operandensatz an dem Array aus Verarbeitungselementen eingeht. Das Array des Verarbeitungselements darf die atomare Operation nicht durchführen, bis der eingehende Operandensatz an dem Array aus Verarbeitungselementen eingeht und der Speicher in dem Array aus Verarbeitungselementen zur Ausgabe der zweiten Operation verfügbar ist. Das Array aus Verarbeitungselementen kann ein Netzwerk aufweisen, das die Datenfluss-Tokens und Steuertokens zu mehreren Datenflussoperatoren trägt. Die atomare Operation kann einen Speicherzugriff aufweisen, und das Array aus Verarbeitungselementen kann einen Speicherzugriff-Datenflussoperator, der den Speicherzugriff nicht durchführen darf, bis er ein Speicherabhängigkeits-Token von einem logisch vorherigen Datenflussoperator empfängt, aufweisen. Jedes Verarbeitungselement kann nur eine oder zwei Operationen des Datenflussgraphen durchführen.

[0204] In einer weiteren Ausführungsform weist ein Prozessor Einrichtungen zum Empfangen einer Eingabe eines Datenflussgraphen, der mehrere Knoten umfasst, auf, wobei der Datenflussgraph in die Einrichtung zu überlagern ist, wobei jeder Knoten für einen Datenflussoperator in der Einrichtung steht, und die Einrichtung eine atomare Operation durchzuführen hat, wenn ein eingehender Operandensatz an der Einrichtung eingeht.

[0205] In einer Ausführungsform weist ein Prozessor einen Kern mit einem Decodierer zum Decodieren eines Befehls in einem decodierten Befehl und eine Ausführungseinheit zum Ausführen des decodierten Befehls zum Durchführen einer ersten Operation; mehrere Verarbeitungselemente; und ein Zwischenverbindungsnetzwerk zwischen den mehreren Verarbeitungselementen zum Empfangen einer Eingabe eines Datenflussgraphen, umfassend mehrere Knoten, auf, wobei der Datenflussgraph in das Zwischenverbindungsnetzwerk und die mehreren Verarbeitungselemente zu überlagern ist, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist, und die mehreren Verarbeitungselemente eine zweite Operation durchzuführen haben, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht, wobei die zweite Operation eine atomare Operation ist. Der Prozessor kann ferner mehrere Konfigurationssteuerungen umfassen, wobei jede Konfigurationssteuerung mit einem jeweiligen Untersatz der mehreren Verarbeitungselemente gekoppelt ist und jede Konfigurationssteuerung Konfigurationsinformation aus dem Speicher zu laden und eine Kopplung des jeweiligen Untersatzes der mehreren Verarbeitungselemente gemäß der Konfigurationsinformation zu veranlassen hat. Der Prozessor kann mehrere Konfigurations-Caches enthalten, und jede Konfigurationssteuerung ist mit einem jeweiligen Konfigurations-Cache gekoppelt, um die Konfigurationsinformation für den jeweiligen Untersatz der mehreren Verarbeitungselemente abzurufen. Die erste Operation, die durch die Ausführungseinheit durchgeführt wird, kann Konfigurationsinformation in jeden der mehreren Konfigurations-Caches abrufen. Jede der mehreren Konfigurationssteuerungen kann eine Rekonfigurationsschaltung aufweisen, um bei Empfang einer Konfigurationsfehlermeldung von dem mindestens einen Verarbeitungselement eine Rekonfiguration für mindestens ein Verarbeitungselement des jeweiligen Untersatzes von Verarbeitungselementen zu veranlassen. Jede der mehreren Konfigurationssteuerungen kann eine Rekonfigurationsschaltung aufweisen, um bei Empfang einer Rekonfigurationsanforderungsnachricht eine Rekonfiguration des jeweiligen Untersatzes der mehreren Verarbeitungselemente zu veranlassen und die Kommunikation mit dem jeweiligen Untersatz der mehreren Verarbeitungselemente zu deaktivieren, bis die Rekonfiguration abgeschlossen ist. Der Prozessor kann mehrere Ausnahmeaggregatoren enthalten, und jeder Ausnahmeaggregator ist mit einem jeweiligen Untersatz der mehreren Verarbeitungselemente gekoppelt, um Ausnahmen von dem jeweiligen Untersatz der mehreren Verarbeitungselemente zu sammeln und die Ausnahmen an den Kern zum Bedienen weiterzuleiten. Der Prozessor kann mehrere Extraktionssteuerungen aufweisen, wobei jede Extraktionssteuerung mit einem jeweiligen Untersatz der mehreren Verarbeitungselemente gekoppelt ist und jede Extraktionssteuerung veranlassen muss, dass Statusdaten aus dem jeweiligen Untersatz der mehreren Verarbeitungselemente in dem Speicher gespeichert werden.

[0206] In einer weiteren Ausführungsform beinhaltet ein Verfahren das Decodieren eines Befehls mit einem Decodierer eines Kerns eines Prozessors in einen decodierten Befehl; Ausführen des decodierten Befehls mit einer Ausführungseinheit des Kerns des Prozessors zum Durchführen einer ersten Operation; Empfangen einer Eingabe eines Datenflussgraphen, der mehrere Knoten umfasst; Überlagern des Datenflussgraphen über

mehrere Verarbeitungselemente des Prozessors und eines Zwischenverbindungsnetzes zwischen den mehreren Verarbeitungselementen des Prozessors, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist; und Durchführen einer zweiten Operation des Datenflussgraphen mit dem Zwischenverbindungsnetz und den mehreren Verarbeitungselementen, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht, wobei die zweite Operation eine atomare Operation ist.

[0207] Das Verfahren kann das Laden von Konfigurationsinformationen aus dem Speicher für jeweilige Untersätze der mehreren Verarbeitungselemente und das Veranlassen der Kopplung für jeden jeweiligen Untersatz der mehreren Verarbeitungselemente gemäß der Konfigurationsinformation beinhalten. Das Verfahren kann das Abrufen der Konfigurationsinformation für den jeweiligen Untersatz der mehreren Verarbeitungselemente von einem jeweiligen Konfigurations-Cache mehrerer Konfigurations-Caches beinhalten. Die erste Operation, die durch die Ausführungseinheit durchgeführt wird, kann das Abrufen von Konfigurationsinformation in jedem der mehreren Konfigurations-Caches sein. Das Verfahren kann bei Empfang einer Konfigurationsfehlermeldung von dem mindestens einen Verarbeitungselement das Veranlassen einer Rekonfiguration für mindestens ein Verarbeitungselement des jeweiligen Untersatzes von Verarbeitungselementen beinhalten. Das Verfahren kann bei Empfang einer Rekonfigurationsanforderungsnachricht das Veranlassen einer Rekonfiguration des jeweiligen Untersatzes der mehreren Verarbeitungselemente und das Deaktivieren der Kommunikation mit dem jeweiligen Untersatz der mehreren Verarbeitungselemente bis Abschluss der Rekonfiguration beinhalten. Das Verfahren kann das Sammeln von Ausnahmen aus einem jeweiligen Untersatz der mehreren Verarbeitungselemente; und das Weiterleiten der Ausnahmen zum Kern für die Bedienung beinhalten. Das Verfahren kann das Veranlassen, dass Statusdaten von einem jeweiligen Untersatz der mehreren Verarbeitungselemente in dem Speicher gespeichert werden, beinhalten.

[0208] In noch einer anderen Ausführungsform veranlasst ein nichtflüchtiges maschinenlesbares Medium, das Code speichert, der, wenn er von einer Maschine ausgeführt wird, die Maschine veranlasst, ein Verfahren durchzuführen, aufweisend das Decodieren eines Befehls mit einem Decodierer eines Kerns eines Prozessors in einen decodierten Befehl; Ausführen des decodierten Befehls mit einer Ausführungseinheit des Kerns des Prozessors zum Durchführen einer ersten Operation; Empfangen einer Eingabe eines Datenflussgraphen, der mehrere Knoten umfasst; Überlagern des Datenflussgraphen über mehrere Verarbeitungselemente des Prozessors und eines Zwischenverbindungsnetzes zwischen den mehreren Verarbeitungselementen des Prozessors, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist; und Durchführen einer zweiten Operation des Datenflussgraphen mit dem Zwischenverbindungsnetz und den mehreren Verarbeitungselementen, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht, wobei die zweite Operation eine atomare Operation ist. Das Verfahren kann das Laden von Konfigurationsinformationen aus dem Speicher für jeweilige Untersätze der mehreren Verarbeitungselemente und das Veranlassen der Kopplung für jeden jeweiligen Untersatz der mehreren Verarbeitungselemente gemäß der Konfigurationsinformation beinhalten. Das Verfahren kann das Abrufen der Konfigurationsinformation für den jeweiligen Untersatz der mehreren Verarbeitungselemente von einem jeweiligen Konfigurations-Cache mehrerer Konfigurations-Caches beinhalten. Die erste Operation, die durch die Ausführungseinheit durchgeführt wird, kann das Abrufen von Konfigurationsinformation in jedem der mehreren Konfigurations-Caches sein. Das Verfahren kann bei Empfang einer Konfigurationsfehlermeldung von dem mindestens einen Verarbeitungselement das Veranlassen einer Rekonfiguration für mindestens ein Verarbeitungselement des jeweiligen Untersatzes von Verarbeitungselementen beinhalten. Das Verfahren kann bei Empfang einer Rekonfigurationsanforderungsnachricht das Veranlassen einer Rekonfiguration des jeweiligen Untersatzes der mehreren Verarbeitungselemente und das Deaktivieren der Kommunikation mit dem jeweiligen Untersatz der mehreren Verarbeitungselemente bis Abschluss der Rekonfiguration beinhalten. Das Verfahren kann das Sammeln von Ausnahmen aus einem jeweiligen Untersatz der mehreren Verarbeitungselemente; und das Weiterleiten der Ausnahmen zum Kern zur Bedienung beinhalten. Das Verfahren kann das Veranlassen, dass Statusdaten von einem jeweiligen Untersatz der mehreren Verarbeitungselemente in dem Speicher gespeichert werden, beinhalten.

[0209] In einer weiteren Ausführungsform weist ein Prozessor mehrere Verarbeitungselemente auf; und eine Einrichtung zwischen den mehreren Verarbeitungselementen zum Empfangen einer Eingabe eines Datenflussgraphen, umfassend mehrere Knoten, wobei der Datenflussgraph in die Einrichtung und die mehreren Verarbeitungseinheiten zu überlagern ist, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist, und die mehreren Verarbeitungselemente eine atomare Operation durchzuführen, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht.

[0210] In noch einer weiteren Ausführungsform umfasst eine Vorrichtung ein Datenspeichergerät, das Code speichert, der, wenn er durch einen Hardware-Prozessor ausgeführt wird, veranlasst, dass der Hardware-Prozessor jedes hierin offenbarte Verfahren durchführt. Eine Vorrichtung kann wie eine in der detaillierten Beschreibung beschriebene sein. Ein Verfahren kann wie ein in der ausführlichen Beschreibung beschriebenes sein.

[0211] In einer weiteren Ausführungsform veranlasst ein nichtflüchtiges maschinenlesbares Medium, das Code speichert, der, wenn er von einer Maschine ausgeführt wird, die Maschine veranlasst, ein Verfahren durchzuführen, das jedes beliebige hierin offenbarte Verfahren umfasst.

[0212] Ein Befehlssatz (z. B. zur Ausführung durch den Kern) kann eines oder mehrere Befehlsformate aufweisen. Ein gegebenes Befehlsformat kann verschiedene Felder (z. B. Anzahl an Bits, Ort von Bits) definieren, um unter anderem die auszuführende Operation (z. B. Opcode) und den/die Operand(en), an dem/denen die Operation durchzuführen ist, und/oder (ein) andere(s) Datenfeld(er) (z. B. Maske) zu spezifizieren. Manche Befehlsformate sind durch die Definition von Befehlstemplates (oder Subformaten) weiter aufgeschlüsselt. Zum Beispiel können die Befehlstemplates eines gegebenen Befehlsformats so definiert sein, dass sie unterschiedliche Untersätze der Felder des Befehlsformats aufweisen (die enthaltenen Felder sind typischerweise in der gleichen Reihenfolge, aber wenigstens manche weisen unterschiedliche Bitpositionen auf, weil weniger Felder enthalten sind), und/oder so definiert sein, dass sie ein gegebenes Feld aufweisen, das unterschiedlich interpretiert wird. Dementsprechend wird jeder Befehl einer ISA unter Verwendung eines gegebenen Befehlsformats (und, falls definiert, in einem gegebenen der Befehlstemplates jenes Befehlsformats) ausgedrückt und beinhaltet jeder Felder zum Spezifizieren der Operation und der Operanden. Zum Beispiel weist ein beispielhafter ADD-Befehl einen speziellen Opcode und ein Befehlsformat auf, das ein Opcode-Feld zum Spezifizieren dieses Opcodes und ein Operandenfeld zum Auswählen von Operanden (Quelle 1/Ziel und Quelle 2) beinhaltet; und ein Auftreten dieses ADD-Befehls in einem Befehlsstrom wird spezielle Inhalte in den Operandenfeldern aufweisen, die spezielle Operanden auswählen. Ein Satz von SIMD-Erweiterungen, die als Advanced Vector Extensions (AVX) (AVX1 und AVX2) bezeichnet werden und das Vektorerweiterungs (VEX) -Codierschema verwenden, wurde herausgegeben und/oder veröffentlicht (siehe z. B. Intel® 64 und IA-32 Architectures Software Developer's Manual, Juni 2016; und siehe Intel® Architecture Instruction Set Extensions Programming Reference, Februar 2016).

Beispielhafte Befehlsformate

[0213] Ausführungsformen des/der hier beschrieben Befehls/Befehle können in verschiedenen Formaten umgesetzt werden. Außerdem sind beispielhafte Systeme, Architekturen und Pipelines unten ausführlich beschrieben. Ausführungsformen des/der Befehls/Befehle können auf solchen Systemen, Architekturen und Pipelines ausgeführt werden, sind aber nicht auf jene ausführlich beschriebenen beschränkt.

Allgemeines vektorfreundliches Befehlsformat

[0214] Ein Vektorfreundliches Befehlsformat ist ein Befehlsformat, das für Vektorbefehle geeignet ist (z. B. gibt es gewisse Felder, die für Vektoroperationen spezifisch sind). Obgleich Ausführungsformen beschrieben sind, bei denen sowohl Vektor- als auch Skalaroperationen durch das vektorfreundliche Befehlsformat unterstützt werden, verwenden alternative Ausführungsformen nur Vektoroperationen durch das vektorfreundliche Befehlsformat.

[0215] **Fig. 36A** -**Fig. 36B** sind Blockdiagramme, die ein allgemeines vektorfreundliches Befehlsformat und Befehlstemplates davon gemäß Ausführungsformen der Offenbarung veranschaulichen. **Fig. 36A** ist ein Blockdiagramm, das ein allgemeines vektorfreundliches Befehlsformat und Klasse-A-Befehlstemplates gemäß Ausführungsformen der Offenbarung veranschaulicht; während **Fig. 36B** ein Blockdiagramm ist, welches das allgemeine vektorfreundliche Befehlsformat und Klasse-B-Befehlstemplates gemäß Ausführungsformen der Offenbarung veranschaulicht. Speziell ein allgemeines vektorfreundliches Befehlsformat **3600**, für das Klasse-A- und Klasse-B-Befehlstemplates definiert sind, die beide Befehlstemplates ohne Speicherzugriff **3605** und Befehlstemplates mit Speicherzugriff **3620** beinhalten. Der Ausdruck allgemein in dem Zusammenhang des vektorfreundlichen Befehlsformats verweist darauf, dass das Befehlsformat nicht an irgendeinen speziellen Befehlssatz gebunden ist.

[0216] Wenngleich Ausführungsformen der Offenbarung beschrieben werden, in denen das vektorfreundliche Befehlsformat Folgendes unterstützt: eine 64-Byte-Vektoroperandenlänge (oder -größe) mit 32-Bit- (4-Byte) oder 64-Bit- (8-Byte-) Datenelementbreiten (oder -größen) (weshalb ein 64-Byte-Vektor aus entweder 16 dop-

pelwortgroßen Elementen oder alternativ 8 vierwortgroßen Elementen besteht); eine 64-Byte-Vektoroperandenlänge (oder -größe) mit 16 Bit (2 Byte) oder 8 Bit (1 Byte) Datenelementbreiten (oder -größen); eine 32-Byte-Vektoroperandenlänge (oder -größe) mit 32 Bit (4 Byte), 64 Bit (8 Byte), 16 Bit (2 Byte) oder 8 Bit (1 Byte) Datenelementbreiten (oder -größen); und eine 16-Byte-Vektoroperandenlänge (oder -größe) mit 32 Bit (4 Byte), 64 Bit (8 Byte), 16 Bit (2 Byte) oder 8 Bit (1 Byte) Datenelementbreiten (oder -größen); können alternative Ausführungsformen mehr, weniger und/oder unterschiedliche Vektoroperandengrößen (z. B. 256 Byte-Vektoroperanden) mit mehr, weniger oder unterschiedlichen Datenelementbreiten (z. B. 128 Bit (16 Byte) Datenelementbreiten) unterstützen.

[0217] Die Klasse-A-Befehlstemplates aus **Fig. 36A** weisen auf: 1) innerhalb der Befehlstemplates ohne Speicherzugriff **3605** ist ein Befehlstemplate für eine Vollrundungssteuerungsoperation ohne Speicherzugriff **3610** und ein Befehlstemplate für eine Datentransformationsoperation ohne Speicherzugriff **3610** gezeigt; und 2) innerhalb der Befehlstemplates mit Speicherzugriff **3620** ist ein temporäres Speicherzugriffsbefehlstemplate **3625** und ein nicht temporäres Speicherzugriffsbefehlstemplate **3630** gezeigt. Die Klasse-B-Befehlstemplates aus **Fig. 36B** weisen auf: 1) innerhalb der Befehlstemplates ohne Speicherzugriff **3605** ist ein Befehlstemplate für eine Schreibmaskensteuerung-Teilrundungssteuerungsoperation ohne Speicherzugriff **3612** und ein Befehlstemplate für eine Schreibmaskensteuerung-VSIZE-Operation mit Speicherzugriff **3610** gezeigt; und 2) innerhalb der Befehlstemplates mit Speicherzugriff **3620** ist ein Speicherzugriffsschreibmaskensteuerungsbefehlstemplate **3627** gezeigt.

[0218] Das allgemeine vektorfreundliche Befehlsformat **3600** weist die folgenden Felder auf, die unten in der in **Fig. 36A** bis **Fig. 36B** aufgelisteten Reihenfolge veranschaulicht sind.

[0219] Formatfeld **3640** - ein spezieller Wert (ein Befehlsformatkennungswert) in diesem Feld identifiziert das vektorfreundliche Befehlsformat und dementsprechend Vorkommnisse von Befehlen in dem vektorfreundlichen Befehlsformat in Befehlsströmen eindeutig. Von daher ist dieses Feld in dem Sinn optional, dass es nicht für einen Befehlssatz benötigt wird, der nur das allgemeine vektorfreundliche Befehlsformat aufweist.

[0220] Basisoperationsfeld **3642** - sein Inhalt unterscheidet verschiedene Basisoperationen.

[0221] Registerindexfeld **3644** - sein Inhalt spezifiziert die Orte der Quellen- und Zieloperanden, seien sie in Registern oder in einem Speicher, direkt oder durch Adressenerzeugung. Diese beinhalten eine ausreichende Anzahl an Bits, um N Register aus einer P×Q(z. B. 32×512, 16×128, 32×1024, 64×1024)-Registerbank auszuwählen. Während bei einer Ausführungsform N bis zu drei Quellen- und ein Zielregister sein kann, können alternative Ausführungsformen mehr oder weniger Quellen- und Zielregister unterstützen (können z. B. bis zu zwei Quellen unterstützen, wobei eine dieser Quellen auch als das Ziel wirkt, können bis zu drei Quellen unterstützen, wobei eine dieser Quellen auch als das Ziel wirkt, können bis zu zwei Quellen und ein Ziel unterstützen).

[0222] Modifiziererfeld **3646** - sein Inhalt unterscheidet das Auftreten von Befehlen im allgemeinen Vektorbefehlsformat, die den Speicherzugriff spezifizieren, von denen, die dies nicht tun; das heißt, zwischen Befehltemplates ohne Speicherzugriff **3605** und Befehlstemplates mit Speicherzugriff **3620**. Speicherzugriffsoperationen lesen und/oder schreiben in die Speicherhierarchie (in manchen Fällen Schreiben der Quellen- und/oder Zieladressen unter Verwendung von Werten in Registern), während Operationen ohne Speicherzugriff dies nicht tun (z. B. sind die Quelle und die Ziele Register). Während bei einer Ausführungsform dieses Feld auch zwischen unterschiedlichen Arten des Durchführens von Speicheradressenberechnungen wählt, können alternative Ausführungsformen mehr, weniger oder unterschiedliche Arten zum Durchführen von Speicheradressenberechnungen unterstützen.

[0223] Ergänzungsoperationsfeld **3650** - sein Inhalt unterscheidet zwischen einer Vielzahl verschiedener Operationen, die zusätzlich zu der Basisoperation durchzuführen sind. Dieses Feld ist kontextspezifisch. In einer Ausführungsform der Offenbarung ist dieses Feld in ein Klassenfeld **3668**, ein Alphafeld **3652** und ein Betafeld **3654**. Das Ergänzungsoperationsfeld **3650** ermöglicht, dass gemeinsame Gruppen von Operationen in einem einzigen Befehl statt in 2, 3 oder 4 Befehlen durchgeführt werden.

[0224] Skalierungsfeld **3660** - sein Inhalt ermöglicht die Skalierung des Inhalts des Indexfelds für eine Speicheradressenerzeugung (z. B. für eine Adressenerzeugung, die $2^{\text{Skalierung}}$ * Index + Basis verwendet).

[0225] Verschiebungsfeld **3662A**- sein Inhalt wird als Teil der Speicheradressenerzeugung verwendet (z. B. für eine Adressenerzeugung, die $2^{\text{Skalierung}}$ * Index + Basis + Verschiebung).

[0226] Verschiebungsfaktorfeld **3662B** (man beachte, dass die Nebeneinanderstellung des Verschiebungsfelds **3662A** direkt über dem Verschiebungsfaktorfeld **3662B** anzeigt, dass das eine oder das andere verwendet wird) - sein Inhalt wird als Teil der Adressenerzeugung verwendet; er spezifiziert einen Verschiebungsfaktor, der durch die Größe eines Speicherzugriffs (N) zu skalieren ist - wobei N die Anzahl der Bytes im Speicherzugriff ist (z. B. für eine Adressenerzeugung, die $2^{\text{Skalierung}}$ * Index + Basis + skalierte Verschiebung). Redundante Bits niedriger Ordnung werden ignoriert und daher wird der Inhalt des Verschiebungsfaktorfelds mit der Speicheroperandengesamtgröße (N) multipliziert, um die abschließende Verschiebung zu erzeugen, die beim Berechnen einer effektiven Adresse zu verwenden ist. Der Wert von N wird durch die Prozessorhardware zur Laufzeit basierend auf dem vollständigen Opcode-Feld **3674** (das hierin später beschrieben wird) und dem Datenmanipulationsfeld **3654C** bestimmt. Das Verschiebungsfeld **3662A** und das Verschiebungsfaktorfeld **3662B** sind optional in dem Sinne, dass sie nicht für Befehltemplates ohne Speicherzugriff **3605** verwendet werden und/oder unterschiedliche Ausführungsformen können nur eines oder keines der beiden implementieren.

[0227] Datenelementbreitenfeld **3664** - sein Inhalt unterscheidet, welche von einer Anzahl an Datenelementbreiten zu verwenden ist (bei manchen Ausführungsformen für alle Befehle; bei anderen Ausführungsformen für nur manche der Befehle). Dieses Feld ist in dem Sinne optional, dass es nicht benötigt wird, falls nur eine Datenelementbreite unterstützt wird und/oder Datenelementbreiten unterstützt werden, die einen gewissen Aspekt der Opcodes verwenden.

[0228] Schreibmaskenfeld **3670** - sein Inhalt steuert auf einer Basis je Datenelementposition, ob die Datenelementposition in dem Zielvektoroperanden das Ergebnis der Basisoperation und der Ergänzungsoperation reflektiert. Klasse-A-Befehlstemplates unterstützen Zusammenlegungsschreibmaskierung, während Klasse-B-Befehlstemplates sowohl Zusammenlegungs- als auch Nullungsschreibmaskierung unterstützen. Beim Zusammenlegen ermöglichen Vektormasken, dass jeder Satz von Elementen im Ziel während der Ausführung einer Operation (spezifiziert durch die Basisoperation und die Augmentationsoperation) vor Aktualisierungen geschützt wird; in einer anderen Ausführungsform, dass der alte Wert jedes Elements des Ziels bewahrt wird, wenn das entsprechende Maskenbit eine 0 aufweist. Demgegenüber ermöglichen Nullungsvektormasken, dass jeder Satz von Elementen im Ziel während der Ausführung einer Operation (spezifiziert durch die Basisoperation und die Augmentationsoperation) genullt wird; in einer anderen Ausführungsform, dass ein Element des Ziels auf 0 gesetzt wird, wenn das entsprechende Maskenbit einen 0-Wert aufweist. Ein Untersatz dieser Funktion ist die Fähigkeit, die Vektorlänge der ausgeführten Operation zu steuern (d. h. die Spanne der Elemente, die modifiziert werden, von der ersten bis zur letzten); es ist jedoch nicht notwendig, dass die Elemente, die modifiziert werden, konsekutiv sind. Somit erlaubt das Schreibmaskenfeld **3670** Teilvektoroperationen, einschließlich Lade-, Speicher-, Arithmetik-, Logikoperationen usw. Während Ausführungsformen der Offenbarung beschrieben werden, in denen der Inhalt des Schreibmaskenfelds **3670** eines aus einer Anzahl von Schreibmaskenregistern auswählt, welche die zu verwendende Schreibmaske (und somit der Inhalt des Schreibmaskenfelds **3670** indirekt die auszuführende Maskierung identifiziert) enthält, ermöglichen alternative Ausführungsformen stattdessen oder zusätzlich, dass der Inhalt des Maskenschreibfelds **3670** direkt die auszuführende Maskierung spezifiziert.

[0229] Unmittelbarfeld **3672** - sein Inhalt ermöglicht die Spezifikation eines Unmittelbaren. Dieses Feld ist in dem Sinne optional, dass es bei einer Implementierung des allgemeinen vektorfreundlichen Formats, das einen Unmittelbaren nicht unterstützt, nicht vorhanden ist und bei Befehlen, die keinen Unmittelbaren verwenden, nicht vorhanden ist.

[0230] Klassenfeld **3668** - sein Inhalt unterscheidet zwischen unterschiedlichen Klassen von Befehlen. Mit Bezug auf **Fig. 36A-B** wählen die Inhalte dieses Feldes zwischen Klasse A- und Klasse B-Befehlen. In **Fig. 36A-B** werden gerundete Eckenquadrate zum Anzeigen verwendet, dass ein spezifischer Wert in einem Feld (z. B. Klasse A 3668A bzw. Klasse B 3668B für das Klassenfeld **3668** in **Fig. 36A-B**) vorhanden ist.

Befehlstemplates der Klasse A

[0231] Im Fall der Klasse-A-Befehlstemplates ohne Speicherzugriff 3605wird das Alphafeld **3652** als ein RS-Feld **3652A** interpretiert, dessen Inhalt unterscheidet, welche der unterschiedlichen Augmentationsoperationstypen auszuführen sind (z. B. Rundung **3652A**.1 und Datentransformation **3652A**.2 sind jeweils für die Rundungstypoperation ohne Speicherzugruff **3610** und den Transformationstypoperations-Befehlstemplates ohne Speicherzugriff **3615** spezifiziert), während das Betafeld **3654** unterscheidet, welche der Operationen des spezifizierten Typs durchzuführen sind. In den Befehlstemplates ohne Speicherzugriff **3605** sind das Skalierungsfeld **3660**, das Verschiebungsfeld **3662A** und das Verschiebungsskalierungsfeld **3662B** nicht vorhanden.

Befehlstemplates ohne Speicherzugriff - Vollrundungssteuertypoperation

[0232] In dem Befehlstemplate der Vollrundungssteuerungstypoperation ohne Speicherzugriff **3610** wird das Betafeld **3654** als ein Rundungssteuerfeld **3654A** interpretiert, dessen Inhalt(e) eine statische Rundung bereitstellt/en. Während in den beschriebenen Ausführungsformen der Offenbarung das Rundungssteuerungsfeld **3654A** ein Feld **3656** zur Unterdrückung aller Gleitkomma-Ausnahmen (SAE) und ein Rundungsoperationssteuerfeld **3658** aufweist, können alternative Ausführungsformen beide Konzepte in das gleiche Feld codieren/unterstützen oder nur eines oder das andere dieser Konzepte/Felder aufweisen (z. B. nur das Rundungsoperationssteuerfeld **3658** aufweisen).

[0233] SAE-Feld **3656** - sein Inhalt unterscheidet, ob die Ausnahmeereignisberichte deaktiviert werden oder nicht; wenn der SAE-Feld 3656-Inhalt anzeigt, dass die Unterdrückung aktiviert ist, gibt eine gegebener Befehl keine Art von Gleitkomma-Ausnahmeflag an und ruft keinen Gleitkomma-Ausnahmehandler auf.

[0234] Rundungsoperationssteuerfeld **3658** - sein Inhalt unterscheidet, welche einer Gruppe von Rundungsoperationen durchzuführen ist (z. B. Aufrunden, Abrunden, Runden zu Null und Runden zum Nächsten). Dementsprechend ermöglicht das Rundungssteuerfeld **3658** das Ändern des Rundungsmodus auf einer Basis je Befehl. In einer Ausführungsform der Offenbarung, in der ein Prozessor ein Steuerregister zum Spezifizieren von Rundungsmodi aufweist, übergeht der Rundungsoperationssteuerfeld 3650-Inhalt diesen Registerwert.

Befehlstemplates ohne Speicherzugriff - Datentransformationstypoperation

[0235] In der Befehlstemplate-Datentransformationstypoperation ohne Speicherzugriff **3615** wird das Betafeld **3654** als ein Datentransformationsfeld **3654B** interpretiert, dessen Inhalt unterscheidet, welche einer Anzahl von Datentransformationen auszuführen ist (z. B. keine Datentransformation, Swizzle, Broadcast).

[0236] Im Fall eines A-Klasse-Befehltemplates mit Speicherzugriff **3620** wird das Alphafeld **3652** als Räumungshinweisfeld **3652B** interpretiert, dessen Inhalt unterscheidet, welcher der Räumungshinweise zu verwenden ist (in **Fig. 36A**, werden Temporär **3652B**.1 und Nicht-Temporär **3652B**.2 jeweils für das temporäre Befehlstemplate mit Speicherzugriff **3625** und das nicht nicht temporäre Befehlstemplate mit Speicherzugriff **3630** spezifiziert, während das Betafeld **3654** als Datenmanipulationsfeld **3654C** interpretiert wird, dessen Inhalt unterscheidet, welche einer Anzahl von Datenmanipulationsoperationen (auch als Primitive bekannt) auszuführen ist (z. B. keine Manipulation; Broadcast; Aufwärtsumwandlung einer Quelle; und Abwärtsumwandlung eines Ziels). Die Befehlstemplates mit Speicherzugriff **3620** weisen das Skalierungsfeld **3660**, das Verschiebungsfeld **3662A** und das Verschiebungsskalierungsfeld **3662B** auf.

[0237] Vektorspeicherbefehle führen Vektorladen aus dem und Vektorspeichern in den Speicher mit Umwandlungsunterstützung durch. Wie bei regulären Vektorbefehlen übertragen Vektorspeicherbefehle Daten von dem/in den Speicher auf eine datenelementweise Art, wobei die Elemente, die tatsächlich übertragen werden, durch die Inhalte der Vektormaske diktiert werden, die als die Schreibmaske ausgewählt wird.

Speicherzugriffsbefehlstemplates - Temporal

[0238] Temporale Daten werden wahrscheinlich bald genug wiederverwendet, um von Caching zu profitieren. Dies ist jedoch ein Hinweis und verschiedene Prozessoren können ihn auf verschiedene Weisen, einschließlich vollständigen Ignorierens des Hinweises, implementieren.

Speicherzugriffsbefehlstemplates - Nichttemporal

[0239] Nichttemporale Daten werden wahrscheinlich nicht bald genug wiederverwendet, um von einem Caching in dem 1.-Level-Cache zu profitieren und sollten Priorität zum Ausräumen erhalten. Dies ist jedoch ein Hinweis und verschiedene Prozessoren können ihn auf verschiedene Weisen, einschließlich vollständigen Ignorierens des Hinweises, implementieren.

Befehlstemplates der Klasse B

[0240] Im Fall der Klasse-B-Befehlstemplates wird das Alphafeld **3652** als ein Schreibmaskensteuerfeld (Z) **3652C** interpretiert, dessen Inhalt unterscheidet, ob die durch das Schreibmaskenfeld **3670** gesteuerte Schreibmaskierung eine Zusammenlegung oder eine Nullung sein sollte.

[0241] Im Fall der Klasse-B-Befehlstemplates ohne Speicherzugriff **3605** wird ein Teil des Betafelds **3654** als ein RL-Feld **3657A** interpretiert, dessen Inhalt unterscheidet, welche der unterschiedlichen Augmentationsoperationstypen auszuführen sind (z. B. Rundung **3657A**.1 und Vektorlänge (VSIZE) **3657A**.2 sind jeweils für das Befehlstemplate der Schreibmaskensteuerungs-Teilrundungstypoperation ohne Speicherzugriff **3612** und das Befehlstemplate für die Schreibmaskensteuerungs-VSIZE-Typoperation ohne Speicherzugriff **3617** spezifiziert), während der Rest des Betafelds **3654** unterscheidet, welche der Operationen des spezifizierten Typs durchzuführen sind. In den Befehlstemplates ohne Speicherzugriff **3605** sind das Skalierungsfeld **3660**, das Verschiebungsfeld **3662A** und das Verschiebungsskalierungsfeld **3662B** nicht vorhanden.

[0242] In dem Befehlstemplate der Schreibmaskensteuerungs-Teilrundungssteuertypoperation ohne Speicherzugriff **3610** wird der Rest des Beta-Feldes **3654** als ein Rundungsoperationsfeld **3659A** interpretiert, und der Ausnahmeereignisbericht deaktiviert (ein gegebener Befehl meldet keine Art von Gleitkomma-Ausnahme-Flag und löst keinen Gleitkomma-Ausnahme-Handler aus).

[0243] Rundoperationssteuerfeld **3659A** - ebenso wie das Rundungsoperationssteuerfeld **3658** unterscheidet sein Inhalt, welche von einer Gruppe von Rundungsoperationen auszuführen ist (z. B. Aufrunden, Abrunden, Runden zu Null und Runden zum Nächsten). Daher erlaubt das Rundungsoperationssteuerfeld **3659A** das Verändern des Rundungsmodus auf einer Basis je Befehl. In einer Ausführungsform der Offenbarung, in der ein Prozessor ein Steuerregister zum Spezifizieren von Rundungsmodi aufweist, übergeht der Inhalt des Rundungsoperationssteuerfelds **3650** diesen Registerwert.

[0244] In dem Befehlstemplate Schreibmaskensteuerungs-VSIZE-Typoperation **3617** wird der Rest des Betafelds **3654** als ein Vektorlängenfeld **3659B** interpretiert, dessen Inhalt unterscheidet, welche einer Anzahl von Datenvektorlängen auszuführen ist (z. B. 128, 256 oder 512 Byte).

[0245] Im Fall eines Klasse-B-Befehlstemplates mit Speicherzugriff **3620** wird ein Teil des Betafeldes **3654** als ein Broadcast-Feld **3657B** interpretiert, dessen Inhalt unterscheidet, ob die Datenmanipulationsoperation des Broadcast-Typs auszuführen ist oder nicht, während der Rest des Betafeldes **3654** als das Vektorlängenfeld **3659B** interpretiert wird. Die Befehlstemplates mit Speicherzugriff **3620** weisen das Skalierungsfeld **3660**, das Verschiebungsfeld **3662A** und das Verschiebungsskalierungsfeld **3662B** auf.

[0246] Mit Bezug auf das allgemeine vektorfreundliche Befehlsformat **3600** ist ein Voll-Opcode-Feld **3674** einschließlich des Formatfelds **3640**, des Basisoperationsfelds **3642** und des Datenelementbreitenfelds **3664** gezeigt. Während eine Ausführungsform gezeigt ist, bei der das Voll-Opcode-Feld **3674** alle dieser Felder beinhaltet, beinhaltet das Voll-Opcode-Feld **3674** bei Ausführungsformen, die nicht alle von ihnen unterstützen, weniger als alle dieser Felder. Das Voll-Opcode-Feld **3674** stellt den Operationscode (Opcode) bereit.

[0247] Das Ergänzungsoperationsfeld **3650**, das Datenelementbreitenfeld **3664** und das Schreibmaskenfeld **3670** ermöglichen, dass diese Merkmale auf Basis je Befehl in dem allgemeinen vektorfreundlichen Befehlsformat spezifiziert werden.

[0248] Die Kombination aus Schreibmaskenfeld und Datenelementbreitenfeld erschafft insofern typisierte Befehle, als dass sie ermöglicht, dass die Maske basierend auf unterschiedlichen Datenelementbreiten angewandt wird.

[0249] Die verschiedenen Befehlstemplates, die innerhalb von Klasse A und Klasse B gefunden werden, sind in verschiedenen Situationen vorteilhaft. In einigen Ausführungsformen der Offenbarung können unterschiedliche Prozessoren oder unterschiedliche Kerne innerhalb eines Prozessors nur Klasse A, nur Klasse B oder beide Klassen unterstützen. Zum Beispiel kann ein Hochleistungs-Allzweck-Außer-Reihenfolge-Kern, der für allgemeine Rechenzwecke gedacht ist, nur Klasse B unterstützen, ein Kern, der primär für Grafik- und/oder wissenschaftliches (Durchsatz-) Berechnung gedacht ist, nur Klasse A unterstützen, und ein Kern, der für beide gedacht ist, beide unterstützen (natürlich ist ein Kern, der eine Mischung aus Templates und Befehlen von beiden Klassen, aber nicht allen Templates und Befehlen von beiden Klassen aufweist, innerhalb des Geltungsbereichs der Offenbarung). Außerdem kann ein einziger Prozessor mehrere Kerne beinhalten, von denen alle die gleiche Klasse unterstützen oder bei denen unterschiedliche Kerne eine unterschiedliche Klasse unterstützen. Zum Beispiel kann in einem Prozessor mit separaten Grafik- und Mehrzweckkernen einer der Grafikkerne, der primär für Grafik- und/oder wissenschaftliches Rechnen gedacht ist, nur Klasse A unterstützen, während einer oder mehrere der Mehrzweckkerne Hochleistungs-Allzweckkerne mit Außer-Reihenfolge-Ausführung und Registerumbenennung sein können, die für Allzweck-Berechnung gedacht sind, die nur Klasse B unterstützen. Ein anderer Prozessor, der keinen separaten Grafikkern aufweist, kann einen oder mehrere

allgemeine In-Reihenfolge- oder Außer-Reihenfolge-Kerne aufweisen, die sowohl Klasse A als auch Klasse B unterstützen. Natürlich können Merkmale aus einer Klasse auch in der anderen Klasse in unterschiedlichen Ausführungsformen der Offenbarung implementiert sein. Programme, die in einer höheren Sprache geschrieben sind, würden in eine Vielzahl unterschiedlicher ausführbarer Formen gelegt werden (z. B. just-in-time-kompiliert oder statisch kompiliert), die Folgendes beinhalten: 1) eine Form, die nur Befehle der Klasse(n) enthält, die durch den Zielprozessor zur Ausführung unterstützt wird/werden; oder 2) eine Form mit alternativen Routinen, die unter Verwendung unterschiedlicher Kombinationen der Befehle aller Klassen geschrieben sind und einen Steuerflusscode aufweisen, der die auszuführenden Routinen basierend auf den Befehlen auswählt, die von dem Prozessor unterstützt werden, der gerade den Code ausführt.

Beispielhaftes spezielles vektorfreundliches Befehlsformat

[0250] **Fig. 37** ist ein Blockdiagramm, das ein beispielhaftes spezifisches vektorfreundliches Befehlsformat gemäß Ausführungsformen der Offenbarung veranschaulicht. **Fig. 37** zeigt ein spezifisches vektorfreundliches Befehlsformat **3700**, das in dem Sinne spezifisch ist, dass es den Ort, die Größe, die Interpretation und Reihenfolge von Feldern sowie Werten von manchen dieser Felder spezifiziert. Das spezielle vektorfreundliche Befehlsformat **3700** kann verwendet werden, um den x86-Befehlssatz zu erweitern, und dementsprechend sind manche der Felder jenen, die in dem existierenden x86-Befehlssatz und einer Erweiterung davon (z. B. AVX) verwendet werden, ähnlich oder die gleichen wie diese. Das Format bleibt konsistent mit dem Präfixcodierungsfeld, Real-Opcode-Byte-Feld, MOD-R/M-Feld, SIB-Feld, Verschiebungsfeld und den Unmittelbarfeldern des existierenden x86-Befehlssatzes mit Erweiterungen. Die Felder aus **Fig. 36**, in welche die Felder aus **Fig. 37** abgebildet werden, sind veranschaulicht.

[0251] Es versteht sich, dass, obwohl Ausführungsformen der Offenbarung unter Bezugnahme auf das spezifische vektorfreundliche Befehlsformat **3700** in dem Kontext des allgemeinen vektorfreundlichen Befehlsformats **3600** zu Veranschaulichungszwecken beschrieben sind, die Offenbarung nicht auf das spezifische vektorfreundliche Befehlsformat **3700** beschränkt ist, sofern nicht anderweitig beansprucht. Zum Beispiel beabsichtigt das allgemeine vektorfreundliche Befehlsformat **3600** eine Vielzahl möglicher Größen für die verschiedenen Felder, während das spezielle vektorfreundliche Befehlsformat **3700** als Felder mit speziellen Größen aufweisend gezeigt ist. Wenngleich ein spezifisches Beispiel das Datenelementbreitenfeld **3664** als ein Ein-Bit-Feld in dem spezifischen vektorfreundlichen Befehlsformat **3700** veranschaulicht, ist die Offenbarung nicht darauf beschränkt (das heißt, das allgemeine vektorfreundliche Befehlsformat **3600** berücksichtigt andere Größen des Datenelementbreitenfelds **3664**).

[0252] Das allgemeine vektorfreundliche Befehlsformat **3600** weist die folgenden Felder auf, die unten in der in **Fig. 37A** aufgelisteten Reihenfolge veranschaulicht sind.

[0253] EVEX-Präfix (Bytes 0-3) **3702** - ist in einer Vier-Bit-Form codiert.

[0254] Formatfeld **3640** (EVEX Byte 0, Bits [7: 0]) - das erste Byte (EVEX Byte 0) ist das Formatfeld **3640** und enthält 0x62 (der eindeutige Wert, der zum Unterscheiden des vektorfreundlichen Befehlsformats in einer Ausführungsform der Offenbarung verwendet wird).

[0255] Die zweiten bis vierten Bytes (EVEX-Bytes 1-3) beinhalten eine Anzahl an Bitfeldern, die eine spezielle Fähigkeit bereitstellen.

[0256] REX-Feld **3705** (EVEX Byte 1, Bits [7-5]) - besteht aus einem EVEX.R-Bitfeld (EVEX Byte 1, Bit [7] -R), EVEX.X-Bitfeld (EVEX Byte 1, Bit [6]-X), und 3657BEX Byte 1, Bit [5]-B). Die EVEX.R-, EVEX.X- und EVEX.B-Bitfelder stellen die gleiche Funktionalität wie die entsprechenden VEX-Bitfelder bereit und sind unter Verwendung einer Einerkomplementform codiert, d. h. ZMM0 ist als 1111B codiert, ZMM15 ist als 0000B codiert. Andere Felder der Befehle codieren die unteren drei Bits der Registerindices wie in der Technik bekannt (rrr, xxx und bbb), so dass Rrrr, Xxxx und Bbbb durch Hinzufügen von EVEX.R, EVEX.X und EVEX.B gebildet werden können.

[0257] REX‘-Feld 3610 - dies ist der erste Teil des REX‘-Felds 3610 und ist das EVEX.R‘-Bitfeld (EVEX-Byte 1, Bit [4] - R'), das verwendet wird, um entweder die oberen 16 oder die unteren 16 des erweiterten 32-Register-Satzes zu codieren. In einer Ausführungsform der Offenbarung wird dieses Bit zusammen mit anderen, wie unten angegeben, in einem bitinvertierten Format gespeichert, um sich (in dem bekannten x86 32-Bit-Modus) von dem BOUND-Befehl zu unterscheiden, dessen reales Opcode-Byte **62** ist, aber akzeptiert im MOD R/M-Feld (unten beschrieben) nicht den Wert **11** im MOD-Feld; alternative Ausführungsformen der

Offenbarung speichern dies und die anderen angezeigten Bits in dem invertierten Format nicht. Ein Wert von 1 wird verwendet, um die unteren 16 Register zu codieren. Mit anderen Worten wird R'Rrrr durch Kombinieren von EVEX.R', EVEX.R und des anderen RRR von anderen Feldern gebildet.

[0258] Opcode-Map-Feld **3715** (EVEX-Byte 1, Bits [3:0] - mmmm) - sein Inhalt codiert ein impliziertes führendes Opcode-Byte (0F, 0F 38 oder 0F 3).

[0259] Datenelementbreitenfeld **3664** (EVEX-Byte 2, Bit [7] - W) - wird durch die Notation EVEX.W repräsentiert. EVEX.W wird verwendet, um die Granularität (Größe) des Datentyps (entweder 32-Bit-Datenelemente oder 64-Bit-Datenelemente) zu definieren.

[0260] EVEX.vvv 3720 (EVEX-Byte 2, Bits [6:3]-vvvv) - die Rolle von EVEX.vvvv kann das Folgende beinhalten: 1) EVEX.vvvv codiert den ersten Quellregisteroperanden, der in invertierter (ls-Komplement-) Form spezifiziert ist und für Befehle mit 2 oder mehr Quelloperanden gültig ist; 2) EVEX.vvvv codiert den Zielregisteroperanden, der für bestimmte Vektorverschiebungen in ls-Komplementform spezifiziert ist; oder 3) EVEX.vvvv codiert keinen Operanden, das Feld ist reserviert und sollte **1111b** enthalten. Dementsprechend codiert das EVEX.vvvv-Feld 3720 die 4 Bits niedriger Ordnung des ersten Quellenregisterspezifikationssymbols, die in invertierter (Einerkomplement) Form gespeichert werden. In Abhängigkeit von dem Befehl wird ein zusätzliches verschiedenes EVEX-Bitfeld verwendet, um die Spezifikationssymbolgröße auf 32 Register zu erweitern.

[0261] EVEX.U 3668 Klassenfeld (EVEX-Byte 2, Bit [2]-U) - falls EVEX.U = 0, gibt es Klasse A oder EVEX.U0 an; falls EVEX.U = 1, gibt es Klasse B oder EVEX.U1 an.

[0262] Präfixcodierungsfeld **3725** (EVEX-Byte 2, Bits [1:0]-pp) - stellt zusätzliche Bits für das Basisoperationsfeld bereit. Zusätzlich zu dem Bereitstellen einer Unterstützung für die veralteten SSE-Befehle in dem EVEX-Präfix-Format weist dies auch den Vorteil des Kompaktierens des SIMD-Präfixes auf (statt ein Byte zum Ausdrücken des SIMD-Präfixes zu benötigen, benötigt das EVEX-Präfix nur 2 Bit). In einer Ausführungsform werden zum Unterstützen der veralteten SSE-Befehle, die ein SIMD-Präfix (66H, F2H, F3H) sowohl im veralteten Format als auch im EVEX-Präfixformat verwenden, diese veralteten SIMD-Präfixe in das SIMD-Präfix-Codierfeld codiert; und zur Laufzeit werden sie in das veraltete SIMD-Präfix expandiert, bevor sie der PLA des Decodierers bereitgestellt werden (so kann die PLA sowohl das veraltete als auch das EVEX-Format dieser veralteten Befehle ohne Modifikation ausführen). Obwohl neuere Befehle den Inhalt des EVEX-Präfixcodierungsfeld direkt als eine Opcode-Erweiterung verwenden könnten, erweitern bestimmte Ausführungsformen zur Konsistenz auf eine ähnliche Weise, erlauben aber, dass unterschiedliche Bedeutungen durch diese veralteten SIMD-Präfixe spezifiziert werden. Eine alternative Ausführungsform kann den PLA umgestalten, um die 2-Bit-SIMD-Präfixcodierungen zu unterstützen, und benötigt dementsprechend die Erweiterung nicht.

[0263] Alphafeld **3652** (EVEX-Byte 3, Bit [7]-EH; auch bekannt als EVEX.EH, EVEX.rs, EVEX.RL, EVEX.-Schreibmaskensteuerung und EVEX.N; ebenfalls mit α veranschaulicht) - wie zuvor beschrieben ist dieses Feld kontextspezifisch.

[0264] Betafeld **3654** (EVEX-Byte 3, Bits [6: 4]-SSS, auch bekannt als EVEX.$s_{2-0,}$ EVEX.$r_{2-0,}$ EVEX.rr1, EVEX.LL0, EVEX.LLB; auch veranschaulicht mit βββ) - wie zuvor beschrieben, ist dieses Feld kontextspezifisch.

[0265] REX'-Feld 3610 - dies ist der Rest des REX'-Felds und ist das EVEX.V'-Bitfeld (EVEX-Byte 3, Bit [3] - V'), das verwendet werden kann, um entweder die oberen 16 oder die unteren 16 des erweiterten 32-Register-Satzes zu codieren. Dieses Bit wird in bitinvertiertem Format gespeichert. Ein Wert von 1 wird verwendet, um die unteren 16 Register zu codieren. Mit anderen Worten wird V'VVVV durch Kombinieren von EVEX.V', EVEX.vvvv gebildet.

[0266] Schreibmaskenfeld **3670** (EVEX-Byte 3, Bits [2:0]-kkk) - sein Inhalt spezifiziert den Index eines Registers in den Schreibmaskenregistern, wie zuvor beschrieben wurde. In einer Ausführungsform der Offenbarung hat der spezifische Wert EVEX.kkk = 000 ein spezielles Verhalten, das impliziert, dass keine Schreibmaske für den bestimmten Befehl verwendet wird (dies kann auf eine Vielzahl von Arten einschließlich der Verwendung einer für alle Einsen fest verdrahteten Schreibmaske oder Hardware, welche die Maskierungshardware umgeht, implementiert werden).

[0267] Real-Opcode-Feld **3730** (Byte 4) ist auch als das Opcode-Byte bekannt. Ein Teil des Opcodes ist in diesem Feld spezifiziert.

[0268] MOD-R/M-Feld 3740 (Byte 5) beinhaltet MOD-Feld **3742**, Reg-Feld **3744** und R/M-Feld 3746. Wie zuvor beschrieben, unterscheidet der Inhalt des MOD-Felds **3742** zwischen Operationen mit Speicherzugriff und Operationen ohne Speicherzugriff. Die Rolle des Reg-Felds **3744** kann in zwei Situationen zusammengefasst werden: Codieren von entweder dem Zielregisteroperanden oder einem Quellenregisteroperanden oder als eine Opcode-Erweiterung behandelt werden und nicht zum Codieren irgendeines Befehlsoperanden verwendet werden. Die Rolle des R/M-Felds 3746 kann das Folgende beinhalten: Codieren des Befehlsoperanden, der eine Speicheradresse referenziert, oder Codieren von entweder dem Zielregisteroperanden oder einem Quellenregisteroperanden.

[0269] Byte für Skalierung, Index, Basis (SIB) (Byte **6**) - Wie zuvor beschrieben, wird der Inhalt des Skalierungsfelds **3650** für eine Speicheradressenerzeugung verwendet. SIB.xxx **3754** und SIB.bbb **3756** - auf die Inhalte dieser Felder wurde zuvor mit Bezug auf die Registerindices Xxxx und Bbbb Bezug genommen.

[0270] Verschiebungsfeld **3662A** (Bytes 7-10) - wenn das MOD-Feld **3742** 10 enthält, sind die Bytes 7-10 das Verschiebungsfeld **3662A**, und es funktioniert genauso wie die veraltete 32-Bit-Verschiebung (disp32) und funktioniert auf Bytegranularität.

[0271] Verschiebungsfaktorfeld **3662B** (Byte 7) - wenn das MOD-Feld **3742** 01 enthält, ist Byte 7 das Verschiebungsfaktorfeld **3662B**. Der Ort dieses Felds ist der gleiche wie jener der veralteten 8-Bit-Verschiebung (disp8) des X86-Befehlssatzes, die auf Bytegranularität funktioniert. Da disp8 vorzeichenerweitert ist, kann es nur zwischen -128 und 127 Bytes Offsets adressieren; in Bezug auf 64-Byte-Cache-Zeilen verwendet disp8 8 Bits, die auf nur vier wirklich nützliche Werte eingestellt werden können -128, -64, 0 und 64; da oft ein größerer Bereich benötigt wird, wird disp32 verwendet; disp32 benötigt jedoch 4 Bytes. Im Gegensatz zu disp8 und disp 32 ist das Verschiebungsfaktorfeld **3662B** eine Neuinterpretation von disp8; bei Verwendung des Verschiebungsfaktorfelds **3662B** wird die tatsächliche Verschiebung durch den Inhalt des Verschiebungsfaktorfeldes mit der Größe des Speicheroperandenzugriffs (N) multipliziert. Die Verschiebung wird als disp8*N bezeichnet. Dies reduziert die durchschnittliche Befehlslänge (es wird ein einzelnes Byte für die Verschiebung verwendet, aber mit einem viel größeren Bereich). Eine solche komprimierte Verschiebung basiert auf der Annahme, dass die effektive Verschiebung ein Vielfaches der Granularität des Speicherzugriffs ist, und daher die redundanten Bits niedriger Ordnung des Adressenoffsets nicht codiert zu werden brauchen. Mit anderen Worten ersetzt das Verschiebungsfaktorfeld **3662B** die veraltete x86-Befehlssatz-8-Bit-Verschiebung. Somit ist das Verschiebungsfaktorfeld **3662B** auf die gleiche Weise wie eine x86-Befehlssatz-8-Bit-Verschiebung (also keine Änderungen in den ModRM/SIB-Codierregeln) mit der einzigen Ausnahme codiert, dass disp8 zu disp8*N überladen ist. Mit anderen Worten gibt es keine Änderungen in den Codierungsregeln oder Codierungslängen, sondern nur in der Interpretation des Verschiebungswertes durch Hardware (die die Verschiebung mit der Größe des Speicheroperanden skalieren muss, um einen byteweisen Adressenversatz zu erhalten). Unmittelbarfeld **3672** arbeitet wie zuvor beschrieben.

Voll-Opcode-Feld

[0272] **Fig. 37B** ist ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Befehlsformats **3700** veranschaulicht, das ein Voll-Opcode-Feld **3674** gemäß einer Ausführungsform der Offenbarung ausmacht. Speziell beinhaltet das Voll-Opcode-Feld **3674** das Formatfeld **3640**, das Basisoperationsfeld **3642** und das Datenelementbreiten(W)-Feld 3664. Das Basisoperationsfeld **3642** beinhaltet das Präfixcodierungsfeld **3725**, das Opcode-Map-Feld **3715** und das Real-Opcode-Feld **3730**.

Registerindexfeld

[0273] **Fig. 37C** ist ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Befehlsformats **3700** veranschaulicht, das ein Registerindexfeld **3644** gemäß einer Ausführungsform der Offenbarung ausmacht. Speziell beinhaltet das Registerindexfeld **3644** das REX-Feld **3705**. das REX'-Feld **3710**, das M.DR/M.reg-Feld **3744**, das MODR/M.r/m-Feld 3746, das VVVV-Feld **3720**, das xxx-Feld **3754** und das bbb-Feld **3756**.

Erweiterungsoperationsfeld

[0274] **Fig. 37D** ist ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Befehlsformats **3700** veranschaulicht, das ein Augmentationsoperationsfeld **3650** gemäß einer Ausführungsform der Offenbarung ausmacht. Wenn das Feld **3668** der Klasse (U) 0 enthält, bedeutet es EVEX.U0 (Klasse A 3668A); wenn es 1 enthält, bedeutet dies EVEX.U1 (Klasse B 3668B). Wenn U=0 ist und das MOD-Feld **3742** 11 enthält

(was eine Nicht-Speicherzugriffsoperation bedeutet), wird das Alphafeld **3652** (EVEX-Byte 3, Bit [7]-EH) als das rs-Feld **3652A** interpretiert. Wenn das rs-Feld 3652A eine 1 enthält (gerundet **3652A.1**), wird das Betafeld **3654** (EVEX-Byte 3, Bits [6:4]-SSS) als das Rundungssteuerfeld **3654A** interpretiert. Das Rundungssteuerfeld **3654A** enthält ein Ein-Bit-SAE-Feld **3656** und ein Zwei-Bit-Rundungsoperationsfeld **3658**. Wenn das rs-Feld **3652A** eine 0 enthält (Datentransformation **3652A.2**), wird das Betafeld **3654** (EVEX-Byte 3, Bits [6:4]-SSS) als ein Drei-Bit-Datentransformationsfeld **3654B** interpretiert. Wenn U=0 ist und das MOD-Feld **3742** 00, 01 oder 10 enthält (was eine Speicherzugriffsoperation bedeutet), wird das Alphafeld **3652** (EVEX-Byte 3, Bit [7] - EH) als das Räumungshinweis (EH) -Feld **3652B** interpretiert und das Betafeld **3654** (EVEX-Byte **3**, Bits [6: 4]- SSS) wird als ein 3-Bit-Datenmanipulationsfeld 3654C interpretiert.

[0275] Wenn U=1 ist, wird das Alphafeld **3652** (EVEX-Byte **3**, Bit [7]-EH) als das Schreibmaskensteuerfeld (Z) **3652C** interpretiert. Wenn U=1 ist und das MOD-Feld **3742 11** (was eine Nicht-Speicherzugriffsoperation bedeutet) enthält, wird ein Teil des Betafelds **3654** (EVEX-Byte **3**, bit [4]- S_0) als das RL-Feld **3657A** interpretiert; wenn es eine 1 (gerundet 3657A.1) enthält, wird der Rest des Betafelds **3654** (EVEX-Byte **3**, Bit [6-5]- S_{2-1}) als das Rundungsoperationsfeld **3659A** interpretiert, während, wenn das RL-Feld **3657A** eine 0 (VSIZE 3657.A2) enthält, der Rest des Betafelds **3654** (EVEX-Byte **3**, bit [6-5]- S_{2-1}) als das Vektorlängenfeld **3659B** (EVEX-Byte **3**, bit [6-5]- L_{1-0}) interpretiert wird. Wenn U=1 ist und das MOD-Feld **3742 00**, **01** oder **10** enthält (was eine Speicherzugriffsoperation bedeutet), wird das Beta-Feld **3654** (EVEX-Byte **3**, Bits [6: 4] - SSS) als Vektorlängenfeld **3659B** (EVEX-Byte **3**, Bit [6-5] - L_{1-0}) und das Broadcastfeld **3657B** (EVEX-Byte **3**, Bit [4] - B) interpretiert.

Beispielhafte Registerarchitektur

[0276] Fig. 38 ist ein Blockdiagramm einer Registerarchitektur **3800** gemäß einer Ausführungsform der Offenbarung. Bei der veranschaulichten Ausführungsform gibt es 32 Vektorregister **3810**, die 512 Bit breit sind; diese Register werden als zmm0 bis zmm31 bezeichnet. Die 256 Bit niedriger Ordnung der unteren 16 zmm-Register werden auf Register ymm0-16 überlagert. Die 128 Bit niedriger Ordnung der unteren 16 zmm-Register (die 128 Bit niedriger Ordnung der ymm-Register) werden auf Register xmm0-15 überlagert. Das spezielle vektorfreundliche Befehlsformat **3700** arbeitet auf diesen überlagerten Registerbänken, wie in den Tabellen unten veranschaulicht ist.

Anpassbare Vektorlänge	Klasse	Operationen	Register
Befehlstemplates, die kein Vektorlängenfeld 3659B aufweisen	A (**Fig. 36A**; U=0)	3610, 3615, 3625, 3630	zmm-Register (die Vektorlänge beträgt 64 Byte)
	B (**Fig. 36B**; U=1)	3612	zmm-Register (die Vektorlänge beträgt 64 Byte)
Befehlstemplates, die ein Vektorlängenfeld 3659B aufweisen	B (**Fig. 36B**; U=1)	3617,3627	zmm-, ymm- oder xmm-Register (die Vektorlänge beträgt 64 Byte, 32 Byte oder 16 Byte), abhängig von dem Vektorlängenfeld 3659B

[0277] Mit anderen Worten wählt das Vektorlängenfeld **3659B** zwischen einer maximalen Länge und einer oder mehreren anderen kürzeren Längen aus, wobei jede solche kürzere Länge die Hälfte der Länge der vorhergehenden Länge ist; und Befehlstemplates ohne das Vektorlängenfeld **3659B** arbeiten bei maximaler Vektorlänge. Ferner arbeiten die Klasse-B-Befehlstemplates des speziellen vektorfreundlichen Befehlsformats **3700** bei einer Ausführungsform an gepackten oder skalaren Single/Double-Precision-Gleitkommadaten und gepackten oder skalaren Integerdaten. Skalare Operationen sind Operationen, die an der Datenelementposition niedrigster Ordnung in einem zmm-/ymm-/xmm-Register ausgeführt werden; die Datenelementpositionen höherer Ordnung bleiben entweder gleich wie vor dem Befehl oder werden in Abhängigkeit von der Ausführungsform auf Null gesetzt.

[0278] Schreibmaskenregister **3815** - bei der veranschaulichten Ausführungsform gibt es 8 Schreibmaskenregister (k0 bis k7), jeweils mit einer Größe von 64 Bit. Bei einer alternativen Ausführungsform weisen die Schreibmaskenregister **3815** eine Größe von 16 Bit auf. Wie zuvor beschrieben, kann in einer Ausführungsform der Offenbarung das Vektormaskenregister k0 nicht als eine Schreibmaske verwendet werden; wenn die Codierung, die normalerweise k0 angeben würde, für eine Schreibmaske verwendet wird, wählt sie eine

festverdrahtete Schreibmaske von OxFFFF aus, wodurch die Schreibmaskierung für diesen Befehl wirksam deaktiviert wird.

[0279] Mehrzweckregister **3825** - bei der veranschaulichten Ausführungsform gibt es sechzehn 64-Bit-Mehrzweckregister, die zusammen mit den existierenden x86-Adressierungsmodi zum Adressieren von Speicheroperanden verwendet werden. Diese Register werden mit den Namen RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP und R8 bis R15 bezeichnet.

[0280] Skalare Gleitkommastapel-Registerdatei (x87-Stapel) 3845, auf der die MMXgepackte ganzzahlige flache Registerdatei **3850** aliasiert ist - in der dargestellten Ausführungsform ist der x87-Stapel ein Stapel mit acht Elementen, der zur Durchführung von skalaren Gleitkomma-Operationen auf 32/64/80-Bit-Gleitkommadaten unter Verwendung der x87-Befehlssatzerweiterung verwendet wird; während die MMX-Register verwendet werden, um Operationen an gepackten 64-Bit-Integerdaten durchzuführen, sowie um Operanden für manche Operationen zu halten, die zwischen den MMX- und XMM-Registern durchgeführt werden.

[0281] Alternative Ausführungsformen der Offenbarung verwenden breitere oder engere Register. Außerdem können alternative Ausführungsformen mehr, weniger oder andere Registerdateien oder Register verwenden.

Beispielhafte Kernarchitekturen, Prozessoren und Computerarchitekturen

[0282] Prozessorkerne können auf verschiedene Arten, für verschiedene Zwecke und in unterschiedlichen Prozessoren implementiert werden. Beispielsweise können Implementierungen solcher Kerne Folgendes beinhalten: 1) einen Allzweck-In-Reihenfolge-Kern, der zur Allzweckberechnung gedacht ist; 2) ein Hochleistungs-Allzweck-Außer-Reihenfolge-Kern, der zur Allzweckberechnung gedacht ist; 3) ein Spezialzweck-Kern, der primär zur Grafik- und/oder wissenschaftlichen (Durchsatz-) Berechnung gedacht ist. Implementierungen unterschiedlicher Prozessoren können Folgendes beinhalten: 1) eine CPU, die einen oder mehrere Allzweck-In-Reihenfolge-Kerne, die zur Allzweckberechnung gedacht sind, und/oder einen oder mehrere Allzweck-Außer-Reihenfolge-Kerne, die zur Allzweckberechnung gedacht sind; und 2) einen Koprozessor, der einen oder mehrere Spezialzweck-Kerne aufweist, die primär für Grafik und/oder Wissenschaft (Durchsatz) gedacht sind. Solche unterschiedlichen Prozessoren führen zu unterschiedlichen Computersystemarchitekturen, die Folgendes beinhalten können: 1) den Koprozessor auf einem separaten Chip von der CPU; 2) den Koprozessor auf einem separaten Die im gleichen Gehäuse wie die CPU; 3) den Koprozessor auf dem gleichen Die wie die CPU (in welchem Fall solch ein Koprozessor manchmal als eine Spezialzwecklogik bezeichnet wird, wie eine integrierte Grafik- und/oder wissenschaftliche (Durchsatz-) Logik oder als Spezialzweckkerne); und 4) ein System-On-Chip, das auf dem gleichen Die die beschriebene CPU aufweist (die manchmal als der bzw. die Anwendungskerne oder Anwendungsprozessoren, der oben beschriebene Koprozessor und zusätzliche Funktionen bezeichnet wird). Beispielhafte Kernarchitekturen sind als Nächstes beschrieben, gefolgt von Beschreibungen beispielhafter Prozessoren und Computerarchitekturen.

Beispielhafte Kernarchitekturen

In-Reihenfolge- und Außer-Reihenfolge-Kern-Blockdiagramm

[0283] **Fig. 39A** ist ein Blockdiagramm, das sowohl eine beispielhafte In-Reihenfolge-Pipeline als auch eine beispielhafte Außer-Reihenfolge-Ausgabe/Ausführungspipeline mit Registerumbenennung gemäß Ausführungsformen der Offenbarung veranschaulicht; **Fig. 39B** ist ein Blockdiagramm, das sowohl ein Ausführungsbeispiel eines In-Reihenfolge-Architekturkerns als auch einen beispielhafte Außer-Reihenfolge-Ausgabe/Ausführungsarchitekturkern, der in einem Prozessor enthalten sein soll, gemäß Ausführungsformen der Offenbarung veranschaulicht. Die Boxen mit durchgezogener Linie aus **Fig. 39A-B** veranschaulichen eine In-Reihenfolge-Pipeline und einen In-Reihenfolge-Kern, während die optionale Addition der gestrichelten Boxen die Registerumbenennung, Außer-Reihenfolge-Ausgabe/Ausführungspipeline und -kern veranschaulicht. Unter der Annahme, dass der In-Reihenfolge-Aspekt eine Untersatz des Außer-Reihenfolge-Aspekts ist, wird der Außer-Reihenfolge-Aspekt beschrieben.

[0284] In **Fig. 39A** weist eine Prozessorpipeline **3900** eine Abrufstufe **3902**, eine Längendecodierstufe **3904**, eine Decodierstufe **3906**, eine Zuweisungsstufe **3908**, eine Umbenennungsstufe **3910**, eine Planungsstufe (auch bekannt als Versende- oder Ausgabestufe) **3912**, eine Register-Lese-/Speicher-Lese-Stufe **3914**, eine Ausführungsstufe **3916**, eine Rückschreib-/Speicher-Schreib-Stufe **3918**, eine Ausnahmehandhabungsstufe **3922** und eine Festschreibungsstufe **3924** auf.

[0285] Fig. 39B zeigt den Prozessorkern **3990**, der eine Frontend-Einheit **3930** aufweist, die mit einer Ausführungs-Engine-Einheit **3950** gekoppelt ist, und beide mit einer Speichereinheit **3970** gekoppelt sind. Der Kern **3990** kann ein RISC-Kern (RISC: Reduced Instruction Set Computing -Berechnung mit reduziertem Befehlssatz), ein CISC-Kern (CISC: Complex Instruction Set Computing - Berechnung mit komplexem Befehlssatz), ein VLIW-Kern (VLIW: Very Long Instruction Word - sehr langes Befehlswort) oder ein hybrider oder alternativer Kerntyp sein. Als noch eine andere Option kann der Kern **3990** ein Spezialkern, wie etwa zum Beispiel ein Netz- oder Kommunikationskern, eine Kompression-Engine, ein Koprozessorkern, ein GPGPU-Kern (GPGPU: General Purpose Computing Graphics Processing Unit - Vielzweck-Berechnung-Grafikverarbeitung-Einheit) oder dergleichen sein.

[0286] Die Frontend-Einheit **3930** beinhaltet eine Zweigprädiktionseinheit **3932**, die mit einer Befehls-Cacheeinheit **3934** gekoppelt ist, die mit einem Übersetzungspuffer (TLB: Translation Lookaside Buffer) **3936** gekoppelt ist, der mit einer Befehlsabrufeinheit **3938** gekoppelt ist, die mit einer Decodierungseinheit **3940** gekoppelt ist. Die Decodierungseinheit **3940** (oder der Decodierer oder Decodiereinheit) kann Befehle decodieren (z. B. Makrobefehle) und als eine Ausgabe eine oder mehrere Mikrooperationen, Mikrocode-Eintrittspunkte, Mikrobefehle, andere Befehle oder andere Steuersignale erzeugen, die von den ursprünglichen Befehlen decodiert werden oder die in anderer Weise davon reflektiert werden oder von diesen abgeleitet sind. Die Decodierungseinheit **3940** kann unter Verwendung zahlreicher verschiedener Mechanismen implementiert werden. Beispiele für geeignete Mechanismen schließen ein, sind aber nicht beschränkt auf, Nachschlagetabellen, Hardware-Implementierungen, programmierbare Logik-Arrays (PLAs), Mikrocode-Nur-Lese-Speicher (ROMs) usw. In einer Ausführungsform enthält der Kern **3990** einen Mikrocode-ROM oder ein anderes Medium, das Mikrocode für bestimmte Makrobefehle speichert (z. B. in der Decodierungseinheit **3940** oder anderweitig in der Front-End-Einheit **3930**). Die Decodierungseinheit **3940** ist mit einer Umbenennung/Zuordnung-Einheit **3952** in der Ausführung-Engine-Einheit **3950** gekoppelt.

[0287] Die Ausführung-Engine-Einheit **3950** beinhaltet die Umbenennung/Zuordnung-Einheit **3952**, die mit einer Zurückzieheinheit **3954** und einem Satz aus einer oder mehreren Planereinheit(en) **3956** gekoppelt ist. Die Planereinheit(en) **3956** repräsentiert/en eine beliebige Anzahl unterschiedlicher Planer, einschließlich Reservierungsstationen, zentraler Instruktionsfenster usw. Die Planereinheit(en) **3956** ist (sind) mit der/den physikalischen Registerdatei(en) -Einheit(en) 3958 gekoppelt. Jede der physikalischen Registerdatei(en)-Einheiten **3958** repräsentiert eine oder mehrere physische Registerdateien, von denen verschiedene einen oder mehrere unterschiedliche Datentypen speichern, wie zum Beispiel skalare Integer, skalare Gleitkommazahl, gepackte Integer, gepackte Gleitkommazahl, Vektorzahl, Vektor-Gleitkomma, Status (z. B. ein Befehlszeiger, der die Adresse des nächsten auszuführenden Befehls ist) usw. In einer Ausführungsform umfasst die physikalische Registerdatei(en)-Einheit **3958** eine Vektorregistereinheit, eine Schreibmaskenregistereinheit und eine Skalarregistereinheit. Diese Registereinheiten können Architekturvektorregister, Vektormaskenregister und Mehrzweckregister bereitstellen. Die physikalische(n) Registereinheit(en) **3958** wird (werden) von der Rückhalteeinheit **3954** überlappt, um verschiedene Wege zu veranschaulichen, wie Registerumbenennung und Außer-Reihenfolge-Ausführung implementiert werden können (z. B. unter Verwendung eines oder mehrerer Neuordnungspuffer und Auslagerungsregisterdatei(en); Verwenden einer oder mehrerer zukünftiger Dateien, eines oder mehrerer Verlaufspuffer und einer oder mehrerer Auslagerungsregisterdateien; Verwenden einer Registerkarte und eines Registerpools; usw.). Die Zurückzieheinheit **3954** und die physische(n) Registerbank (en)einheit(en) 3958 sind mit dem (den) Ausführungscluster(n) 3960 gekoppelt. Das/die Ausführungscluster **3960** beinhaltet/beinhalten einen Satz aus einer oder mehreren Ausführungseinheiten **3962** und einen Satz aus einer oder mehreren Speicherzugriffseinheiten **3964**. Die Ausführungseinheiten **3962** können verschiedene Operationen (z. B. Verschiebungen, Addition, Subtraktion, Multiplikation) und an verschiedenen Typen von Daten (z. B. Skalargleitkomma, gepackter Integer, gepacktes Gleitkomma, Vektorinteger, Vektorgleitkomma) durchführen. Während manche Ausführungsformen eine Anzahl an Ausführungseinheiten beinhalten können, die für spezielle Funktionen oder Sätze von Funktionen dediziert sind, können andere Ausführungsformen nur eine Ausführungseinheit oder mehrere Ausführungseinheiten, die alle Funktionen durchführen, beinhalten. Die Planereinheit(en) **3956**, die physische(n) Registerbank(en)einheit(en) 3958 und das (die) Ausführungscluster **3960** sind als möglicherweise mehrere gezeigt, weil gewisse Ausführungsformen getrennte Pipelines für gewisse Typen von Daten/Operationen erschaffen (z. B. eine Skalarinteger-Pipeline, eine Skalargleitkomma-/Gepackter-Integer-/Gepacktes-Gleitkomma-/Vektorinteger-/Vektorgleitkomma-Pipeline und/oder eine Speicherzugriff-Pipeline, die jeweils ihre/n eigene/n Planereinheit, physische Registerbank(en)einheit und/oder Ausführungscluster aufweisen - und im Fall einer getrennten Speicherzugriff-Pipeline sind gewisse Ausführungsformen implementiert, bei denen nur der Ausführungscluster dieser Pipeline die Speicherzugriffseinheit(en) **3964** aufweist). Es versteht sich auch, dass, wenn getrennte Pipelines verwendet werden, eine oder mehrere dieser Pipelines eine Außer-Reihenfolge-Ausgabe/Ausführung und der Rest In-Reihenfolge sein können.

[0288] Der Satz von Speicherzugriffseinheiten **3964** ist mit der Speichereinheit **3970** gekoppelt, die eine Daten-TLB-Einheit **3972** beinhaltet, die mit einer Datencacheeinheit **3974** gekoppelt ist, die mit einer Level-2(L2) -Cache-Einheit 3976 gekoppelt ist. Bei einem Ausführungsbeispiel können die Speicherzugriffseinheiten **3964** eine Ladeeinheit, eine Adressenspeichereinheit und eine Datenspeichereinheit beinhalten, von denen jede mit der Daten-TLB-Einheit **3972** in der Speichereinheit **3970** gekoppelt ist. Die Befehlscacheeinheit **3934** ist ferner mit einer Level-2(L2)-Cache-Einheit **3976** in der Speichereinheit **3970** gekoppelt. Die **L2**-Cache-Einheit **3976** ist mit einem oder mehreren anderen Leveln eines Caches und schlussendlich mit einem Hauptspeicher verbunden.

[0289] Beispielsweise kann die beispielhafte Registerumbenennung-Außer-Reihenfolge-Ausgabe/Ausführung-Kern-Architektur die Pipeline **3900** wie folgt implementierten: 1) der Befehlsabruf **3938** führt den Abruf und die Längendecodierungsstufen **3902** und **3904** durch; 2) die Decodierungseinheit **3940** führt die Decodierungsstufe **3906** durch; 3) die Umbenennung/Zuweisung-Einheit **3952** führt die Zuweisungsstufe **3908** und Umbenennungsstufe **3910** durch; 4) die Planereinheit(en) **3956** führt (führen) die Planungsstufe **3912** durch; 5) die physische Registerdateieinheit(en) **3958** und die Speichereinheit **3970** führen die Register-Lese-/Speicher-Lese-Stufe **3914** durch; die Ausführungscluster **3960** führen die Ausführungsstufe **3916** durch; 6) die Speichereinheit **3970** und die eine oder mehreren physischen Registerdateieinheiten **3958** führen die Rückschreibe-/Speicher-Schreib-Stufe **3918** durch; 7) verschiedene Stufen können an der Ausnahmehandhabungsstufe **3922** beteiligt sein; und 8) die Auslagerungseinheit **3954** und die eine oder mehreren physischen Registerdateieinheiten **3958** führen die Festschreibungsstufe **3924** durch.

[0290] Der Kern **3990** kann einen oder mehrere Befehlssätze unterstützen (z. B. den x86-Befehlssatz (mit einigen Erweiterungen, die mit neueren Versionen hinzugefügt wurden), den MIPS-Befehlssatz von MIPS Technologies aus Sunnyvale, CA, den ARM-Befehlssatz (mit optionalen zusätzlichen Erweiterungen wie NEON) von ARM Holdings aus Sunnyvale, CA), einschließlich der hierin beschriebenen Befehle. Bei einer Ausführungsform beinhaltet der Kern **3990** eine Logik zum Unterstützen einer Gepackte-DatenBefehlssatzerweiterung (z. B. AVX1, AVX2), wodurch ermöglicht wird, dass die Operationen, die durch viele Multimediaanwendungen verwendet werden, unter Verwendung gepackter Daten durchgeführt werden.

[0291] Es versteht sich, dass der Kern Multithreading (Ausführen von zwei oder mehr parallelen Sätzen von Operationen oder Threads) unterstützen kann und dies auf vielfältige Weisen vornehmen kann, einschließlich Zeitscheiben-Multithreading, Simultan-Multithreading (wobei ein einziger physischer Kern einen logischen Kern für jeden der Threads bereitstellt, die der physische Kern simultan im Multithreading behandelt), oder eine Kombination davon (z. B. Zeitscheiben-Abruf und -Decodierung und simultanes Multithreading danach, wie etwa bei der Hyperthreading-Technologie von Intel®).

[0292] Während eine Registerumbenennung in dem Zusammenhang einer Außer-Reihenfolge-Ausführung beschrieben ist, versteht es sich, dass eine Registerumbenennung in einer In-Reihenfolge-Architektur verwendet werden kann. Während die veranschaulichte Ausführungsform des Prozessors auch getrennte Befehls- und Datencacheeinheiten **3934**/**3974** und eine geteilte L2-Cache-Einheit **3976** beinhaltet, können alternative Ausführungsformen einen einzigen internen Cache für sowohl Befehle als auch Daten aufweisen, wie etwa zum Beispiel einen internen Level-1(L1)-Cache oder mehrere Level eines internen Caches. Bei manchen Ausführungsformen kann das System eine Kombination eines internen Caches und eines externen Caches, der extern zu dem Kern und/oder dem Prozessor ist, beinhalten. Alternativ dazu kann der gesamte Cache extern zu dem Kern und/oder dem Prozessor sein.

Spezielle beispielhafte In-Reihenfolge-Kernarchitektur

[0293] **Fig. 40A-B** zeigen ein Blockdiagramm einer spezifischeren beispielhaften In-Reihenfolge-Kernarchitektur, wobei der Kern einer von mehreren Logikblöcken (einschließlich anderer Kerne desselben Typs und/oder unterschiedlichen Typs) in einem Chip sein würde. Die Logikblöcke kommunizieren durch ein Zwischenverbindungsnetz mit hoher Bandbreite (z. B. ein Ringnetz) mit, in Abhängigkeit von der Anwendung, einer festen Funktionslogik, Speicher-E/A-Schnittstellen und anderer notwendiger E/A-Logik.

[0294] **Fig. 40A** ist ein Blockdiagramm eines einzelnen Prozessorkerns zusammen mit seiner Verbindung mit On-Die-Zwischenverbindungsnetzen **4002** und mit seinem lokalen Untersatz des Level 2 (L2) -Caches 4004 gemäß Ausführungsformen der Offenbarung. In einer Ausführungsform unterstützt eine Befehlsdecodierungseinheit **4000** den x86-Befehlssatz mit einer gepackten Datenbefehlssatzerweiterung. Ein L1-Cache 4006 ermöglicht Zugriffe mit geringer Latenz auf einen Cachespeicher in die Skalar- und Vektoreinheiten. Während in einer Ausführungsform (zur Vereinfachung der Ausgestaltung) eine Skalareinheit **4008** und eine Vektoreinheit

4010 separate Registersätze (jeweils Skalarregister **4012** und Vektorregister **4014**) verwenden und Daten, die dazwischen übertragen werden, in den Speicher geschrieben und dann aus einem Level 1 (L1)-Cache 4006 zurückgelesen werden, können alternative Ausführungsformen der Offenbarung einen anderen Ansatz verwenden (z. B. einen Einzelregistersatz verwenden oder einen Kommunikationspfad aufweisen, der die Übertragung von Daten zwischen den zwei Registerdateien ermöglicht, ohne dass diese geschrieben und zurückgelesen werden).

[0295] Die lokale Untersatz des L2-Caches **4004** ist Teil eines globalen L2-Caches, der in getrennte lokale Teilsätze, einen pro Prozessorkern, unterteilt ist. Jeder Prozessorkern weist einen direkten Zugriffspfad auf seine eigene lokale Untersatz des L2-Caches **4004** auf. Daten, die durch einen Prozessorkern gelesen werden, werden in seiner L2-Cache-Untersatz **4004** gespeichert und auf sie kann schnell parallel zu anderen Prozessorkernen, die auf ihre eigenen lokalen L2-Cache-Teilsätze zugreifen, zugegriffen werden. Daten, die durch einen Prozessorkern geschrieben werden, werden in seiner eigenen L2-Cache-Untersatz **4004** gespeichert und werden bei Bedarf aus anderen Teilsätzen ausgeräumt. Das Ringnetz stellt eine Kohärenz für geteilte Daten sicher. Das Ringnetz ist bidirektional, um zu ermöglichen, dass Agenten, wie etwa Prozessorkerne, L2-Caches und andere Logikblöcke, miteinander innerhalb des Chips kommunizieren. Jeder Ringdatenpfad ist pro Richtung 1012 Bit breit.

[0296] **Fig. 40B** ist eine auseinander gezogene Ansicht des Teils des Prozessorkerns in **Fig. 40A** gemäß Ausführungsformen der Offenbarung; **Fig. 40B** weist einen Ll-Datencache **4006A** auf, der Teil des L1-Cache **4004** ist, sowie mehr Details im Hinblick auf die Vektoreinheit **4010** und die Vektorregister **4014**. Speziell ist die Vektoreinheit **4010** eine 16-breite Vektorverarbeitungseinheit (VPU: Vector Processing Unit) (siehe die 16-breite ALU **4028**), die Integer- und/oder Single-Precision-Gleit- und/oder Double-Precision-Gleitbefehle ausführt. Die VPU unterstützt das Swizzling von Registereingaben mit der Swizzle-Einheit **4020**, die numerische Umwandlung mit den numerischen Umwandlungseinheiten **4022A-B** und die Replikation mit der Replikationseinheit **4024** auf der Speichereingabe. Schreibmaskenregister **4026** ermöglichen eine Vorhersage resultierender Vektorschreibvorgänge.

[0297] **Fig. 41** ist ein Blockdiagramm eines Prozessors **4100**, der mehr als einen Kern aufweisen kann, eine integrierte Speichersteuerung aufweisen kann und der eine integrierte Grafik gemäß Ausführungsformen der Offenbarung aufweisen kann. Die durchgezogenen Boxen in **Fig. 41** veranschaulichen einen Prozessor **4100** mit einem einzelnen Kern **4102A**, einem Systemagenten **4110**, einem Satz von einer oder mehreren Busssteuerungseinheiten **4116**, während die optionale Hinzunahme der gestrichelten Boxen einen alternativen Prozessor **4100** mit mehreren Kernen **4102A-N**, einen Satz aus einer oder mehreren integrierten Speichersteuereinheit **4114** in der Systemagenteneinheit **4110** und eine Spezialzwecklogik **4108** veranschaulicht.

[0298] Dementsprechend können unterschiedliche Implementierungen des Prozessors **4100** Folgendes beinhalten: 1) eine CPU mit der Spezialzwecklogik **410**, die eine integrierte Grafik- und/oder wissenschaftliche (Durchsatz-) Logik ist (die einen oder mehrere Kerne aufweisen kann), und die Kerne **4102A-N**, die einer oder mehrere Mehrzweckkerne sind (z. B. Allzweck-In-Reihenfolge-Kerne, Allzweck-Außer-Reihenfolge-Kerne, eine Kombination aus den zwei); 2) einen Koprozessor mit den Kernen **4102A-N**, die eine große Anzahl von Spezialzweckkernen sind, die primär für Grafik und/oder Wissenschaft (Durchsatz) gedacht sind; und 3) einen Koprozessor mit den Kernen **4102A-N**, der eine große Anzahl aus Allzweck-In-Reihenfolge-Kernen ist. Dementsprechend kann der Prozessor **4100** ein Mehrzweckprozessor, ein Koprozessor oder Spezialprozessor, wie etwa ein Netz- oder Kommunikationsprozessor, eine Kompression-Engine, ein Grafikprozessor, GPGPU (Mehrzweckgrafikverarbeitungseinheit), ein Hochdurchsatz-MIC-Koprozessor (MIC: Many Integrated Core - viele integrierte Kerne) (der 30 oder mehr Kerne beinhaltet), ein eingebetteter Prozessor oder dergleichen sein. Der Prozessor kann auf einem oder mehreren Chips implementiert sein. Der Prozessor **4100** kann Teil eines oder mehrerer Substrate, die eine beliebige einer Anzahl an Prozesstechnologien verwenden, wie etwa zum Beispiel BiCMOS, CMOS oder NMOS, sein und/oder auf solchen implementiert sein.

[0299] Die Speicherhierarchie beinhaltet ein oder mehrere Levels eines Caches innerhalb der Kerne, eine Menge aus einem oder mehreren geteilten Cacheeinheiten **4106** und einen (nicht gezeigten) externen Speicher, der mit dem Satz aus integrierten Speichersteuereinheiten **4114** gekoppelt ist. Die Menge geteilter Cache-Einheiten **4106** kann einen oder mehrere Mid-Level-Caches, wie etwa Level 2 (L2), Level **3** (L3), Level 4 (L4) oder andere Level eines Caches, einen Last-Level-Cache (LLC) und/oder Kombinationen davon beinhalten. Während bei einer Ausführungsform eine ringbasierte Zwischenverbindungseinheit **4112** die integrierte Grafiklogik **4108**, die Menge geteilter Cacheeinheiten **4106** und die Systemagentenheinheit 4110/integrierte Speichersteuereinheit(en) **4114** miteinander verbindet, können alternative Ausführungsformen eine beliebige Anzahl wohlbekannter Techniken zum Zwischenverbinden solcher Einheiten verwenden. Bei einer Ausfüh-

rungsform wird eine Kohärenz zwischen einer oder mehreren Cacheeinheiten **4106** und Kernen **4102-A-N** beibehalten.

[0300] In einigen Ausführungsformen können einer oder mehrere der Kerne **4102A-N** multithreadingfähig sein. Der Systemagent **4110** enthält die Komponenten, welche die Kerne **4102A-N** koordinieren und betreiben. Die Systemagenteinheit **4110** kann beispielsweise eine Leistungssteuereinheit (PCU) und eine Anzeigeeinheit aufweisen. Die PCU kann eine Logik und Komponenten sein oder aufweisen, die zum Regeln des Leistungszustands der Kerne **4102A-N** und der integrierten Grafiklogik **4108** benötigt werden. Die Anzeigeeinheit dient dem Ansteuern einer oder mehrerer extern verbundener Anzeigen.

[0301] Die Kerne **4102A-N** können hinsichtlich des Architekturbefehlssatzes homogen oder heterogen sein; das heißt, zwei oder mehr der Kerne **4102A-N** können den gleichen Befehlssatz ausführen, während andere nur einen Untersatz dieses Befehlssatzes oder einen anderen Befehlssatz ausführen können.

Beispielhafte Computerarchitekturen

[0302] **Fig. 42-45** sind Blockdiagramme beispielhafter Computerarchitekturen. Andere Systemgestaltungen und Konfigurationen, die in der Technik für Laptops, Desktops, Handheld PCs, persönliche digitale Assistenten, technische Workstations, Server, Netzwerkvorrichtungen, Netzwerk-Hubs, Switches, eingebettete Prozessoren, digitale Signalprozessoren (DSPs), Grafikvorrichtungen, Videospielvorrichtungen, Set-Top-Boxes, Mikrocontroller, Mobiltelefone, portable Medienabspieler, tragbare Vorrichtungen und verschiedene andere elektronische Vorrichtungen bekannt sind, sind ebenfalls geeignet. Allgemein ist eine große Vielzahl an Systemen oder elektronischen Vorrichtungen, die zum Einbinden eines Prozessors und/oder einer anderen Ausführungslogik, wie hier offenbart, fähig sind, allgemein geeignet.

[0303] Mit Bezug auf **Fig. 42** ist ein Blockdiagramm eines Systems **4200** gemäß einer Ausführungsform der vorliegenden Offenbarung gezeigt. Das System **4200** kann einen oder mehrere Prozessoren **4210**, **4215** beinhalten, die mit einem Steuer-Hub **4220** gekoppelt sind. In einer Ausführungsform weist der Steuerungs-Hub **4220** einen Grafikspeicher-Steuerungs-Hub (GMCH) **4290** und einen Eingabe/Ausgabe-Hub (IOH) **4250** auf (die sich auf separaten Chips befinden können); der GMCH **4290** weist Speicher- und Grafiksteuerungen auf, mit denen ein Speicher **4240** und ein Koprozessor **4245** verbunden sind; der IOH **4250** koppelt Eingabe/Ausgabe (I/O) -Geräte 4260 mit dem GMCH **4290**. Alternativ dazu sind eine oder beide der Speicher- und Grafiksteuerungen in dem Prozessor integriert (wie hierin beschrieben), der Speicher **4240** und der Koprozessor **4245** sind direkt mit dem Prozessor **4210** gekoppelt und der Steuerungs-Hub **4220** ist in einem einzigen Chip mit dem IOH **4250** integriert. Der Speicher **4240** kann ein Kompilierermodul **4240A** aufweisen, z. B. zum Speichern von Code, der beim Ausführen davon einen Prozessor veranlasst, jedes Verfahren dieser Offenbarung durchzuführen.

[0304] Die optionale Natur zusätzlicher Prozessoren **4215** ist in **Fig. 42** mit gestrichelten Linien gekennzeichnet. Jeder Prozessor **4210**, **4215** kann einen oder mehrere hier beschriebene Verarbeitungskerne beinhalten und kann irgendeine Version des Prozessors **4100** sein.

[0305] Der Speicher **4240** kann zum Beispiel dynamischer Direktzugriffsspeicher (DRAM: Direct Random Access Memory), Phasenwechselspeicher (PCM: Phase Change Memory) oder eine Kombination von den beiden sein. Für wenigstens eine Ausführungsform kommuniziert der Steuer-Hub **4220** mit dem(den) Prozessor(en) **4210**, **4215** über einen Multi-Drop-Bus, wie etwa einen Front-Side-Bus (FSB), eine Punkt-zu-Punkt-Schnittstelle, wie etwa QuickPath-Interconnect (QPI) oder eine ähnliche Verbindung **4295**.

[0306] Bei einer Ausführungsform ist der Koprozessor **4245** ein Spezialprozessor, wie etwa zum Beispiel ein Hochdurchsatz-MIC-Prozessor, ein Netz- oder Kommunikationsprozessor, eine Kompression-Engine, ein Grafikprozessor, eine GPGPU, ein eingebetteter Prozessor oder dergleichen. Bei einer Ausführungsform kann der Steuer-Hub **4220** einen integrierten Grafikbeschleuniger beinhalten.

[0307] Die physikalischen Ressourcen **4210**, **4215** können sehr unterschiedlich im Hinblick auf das Spektrum der Leistungsmetrik, einschließlich Architektur-, Mikroarchitektur-, Wärme-, Stromverbrauchsmerkmale und dergleichen sein.

[0308] Bei einer Ausführungsform führt der Prozessor **4210** Befehle aus, die Datenverarbeitungsoperationen eines allgemeinen Typs steuern. Eingebettet in die Befehle können Koprozessorbefehle sein. Der Prozessor **4210** erkennt diese Koprozessorbefehle als von einem Typ, der durch den angehängten Koprozessor **4245**

ausgeführt werden sollte. Entsprechend gibt der Prozessor **4210** diese Koprozessorbefehle (oder Steuersignale, die Koprozessorbefehle repräsentieren) auf einem Koprozessorbus oder einer anderen Zwischenverbindung an den Koprozessor **4245** aus. Der/die Koprozessor(en) **4245** nehmen die empfangen Koprozessorbefehle an und führen diese aus.

[0309] Mit Bezug auf **Fig. 43** ist ein Blockdiagramm eines ersten spezifischeren beispielhaften Systems **4300** gemäß einer Ausführungsform der vorliegenden Offenbarung gezeigt. Wie in **Fig. 43** gezeigt, ist das Mehrfachprozessorsystem **4300** ein Punkt-zu-Punkt-Zwischenverbindungssystem und beinhaltet einen ersten Prozessor **4370** und einen zweiten Prozessor **4380**, der über eine Punkt-zu-Punkt-Zwischenverbindung **4350** gekoppelt ist. Jeder der Prozessoren **4370** und **4380** kann irgendeine Version des Prozessors **4100** sein. In einer Ausführungsform der Offenbarung, sind die Prozessoren **4370** und **4380** jeweils Prozessoren **4210** und **4215**, während der Prozessor **4338** der Koprozessor **4245** ist. Bei einer anderen Ausführungsform sind die Prozessoren **4370** und **4380** der Prozessor **4210** bzw. der Koprozessor **4245**.

[0310] Die Prozessoren **4370** und **4380** sind einschließlich IMC-Einheiten (IMC: Integrated Memory Controller - Integrierter-Speicher-Steuerung) **4372** bzw. **4382** gezeigt. Der Prozessor **4370** weist auch, als Teil seiner Bussteuereinheiten, die Punkt-zu-Punkt- (P-P)-Schnittstellen **4376** und **4378** auf; in ähnlicher Weise weist der zweite Prozessor **4380** die P-P-Schnittstellen **4386** und **4388** auf. Die Prozessoren **4370** und **4380** können Daten über eine Punkt-zu-Punkt(P-P)-Schnittstelle **4350** unter Verwendung von P-P-Schnittstellenschaltungen **4378** bzw. **4388** austauschen. Wie in **Fig. 43** gezeigt, koppeln IMCs **4372** und **4382** die Prozessoren mit jeweiligen Speichern, nämlich einem Speicher **4332** und einem Speicher **4334**, die Teile eines Hauptspeichers sein können, die lokal an die jeweiligen Prozessoren angehängt sind.

[0311] Die Prozessoren **4370**, **4380** können jeweils Informationen über individuelle P-P-Schnittstellen **4352**, **4354** mit einem Chipsatz **4390** unter Verwendung von Punkt-zu-Punkt-Schnittstelle-Schaltkreisen **4376**, **4394**, **4386**, **4398** austauschen. Der Chipsatz **4390** kann optional Informationen mit dem Koprozessor **4338** über eine Hochleistungsschnittstelle **4339** austauschen. Bei einer Ausführungsform ist der Koprozessor **4338** ein Spezialprozessor, wie etwa zum Beispiel ein Hochdurchsatz-MIC-Prozessor, ein Netz- oder Kommunikationsprozessor, eine Kompression-Engine, ein Grafikprozessor, eine GPGPU, ein eingebetteter Prozessor oder dergleichen.

[0312] Ein (nichtgezeigter) geteilter Cache kann in beiden Prozessoren oder außerhalb beider Prozessoren enthalten sein, jedoch über eine P-P-Zwischenverbindung mit den Prozessoren verbunden sein, so dass lokale Cacheinformationen von einem oder beiden der Prozessoren in dem geteilten Cache gespeichert werden können, falls ein Prozessor in einen Niederleistungsmodus gesetzt wird.

[0313] Der Chipsatz **4390** kann über eine Schnittstelle **4396** mit einem ersten Bus **4316** gekoppelt werden. In einer Ausführungsform kann der erste Bus **4316** ein PCI-(Peripheral Component Interconnect)-Bus oder ein Bus wie ein PCI Express-Bus oder ein anderer I/O-Interconnect-Bus der dritten Generation sein, wenngleich der Schutzumfang der vorliegenden Offenbarung nicht darauf eingeschränkt ist.

[0314] Wie in **Fig. 43** gezeigt, können verschiedene E/A-Vorrichtungen **4314** zusammen mit einer Busbrücke **4318**, die den ersten Bus **4316** mit einem zweiten Bus **4320** koppelt, mit dem ersten Bus **4316** gekoppelt sein. Bei einer Ausführungsform sind ein oder mehrere zusätzliche Prozessoren **4315**, wie etwa Koprozessoren, Hochdurchsatz-MIC-Prozessoren, GPGPUs, Beschleuniger (wie etwa z. B. Grafikbeschleuniger oder DSP-Einheiten (DSP: Digital Signal Processing - digitale Signalverarbeitung)), vor Ort programmierbare Gate-Arrays oder ein beliebiger anderer Prozessor, mit dem ersten Bus **4316** gekoppelt. In einer Ausführungsform kann der zweite Bus **4320** ein LPC-Bus (Low Pin Count) sein. Verschiedene Vorrichtungen können mit einem zweiten Bus **4320** gekoppelt sein, der zum Beispiel eine Tastatur und/oder Maus **4322**, Kommunikationsvorrichtungen **4327** und eine Speichereinheit **4328** wie ein Plattenlaufwerk oder eine andere Massenspeichervorrichtung, die Befehle/Code und Daten **4330** aufweisen kann, in einer Ausführungsform umfassen. Ferner kann ein Audio-E/A **4324** mit dem zweiten Bus **4320** gekoppelt sein. Es wird angemerkt, dass andere Architekturen möglich sind. Zum Beispiel kann ein System statt der Punkt-zu-Punkt-Architektur aus **Fig. 43** einen Multi-Drop-Bus oder eine andere solche Architektur implementieren.

[0315] Mit Bezug auf **Fig. 44** ist ein Blockdiagramm eines zweiten spezifischeren beispielhaften Systems **4400** gemäß einer Ausführungsform der vorliegenden Offenbarung gezeigt. Gleiche Elemente in **Fig. 43** und **Fig. 44** tragen gleiche Bezugsziffern und gewisse Aspekte aus **Fig. 43** wurden in **Fig. 44** weggelassen, um eine Verschleierung anderer Aspekte von **Fig. 44** zu vermeiden.

[0316] **Fig. 44** veranschaulicht, dass die Prozessoren **4370**, **4380** eine Integrierter-Speicher-und-E/A-Steuerlogik („CL“: Control Logic) **4372** bzw. **4382** beinhalten können. Dementsprechend beinhaltet die CL **4372**, **4382** integrierte Speichersteuereinheiten und beinhaltet eine E/A-Steuerlogik. **Fig. 44** veranschaulicht ferner, dass nicht nur die Speicher **4332**, **4334** mit der CL **4372**, **4382** gekoppelt sind, sondern auch, dass die E/A-Vorrichtungen **4414** ebenfalls mit der Steuerlogik **4372**, **4382** gekoppelt sind. Veraltete E/A-Vorrichtungen 4415 sind mit dem Chipsatz **4390** gekoppelt.

[0317] Mit Bezug auf **Fig. 45** ist ein Blockdiagramm eines SoC 4500 gemäß einer Ausführungsform der vorliegenden Offenbarung gezeigt. Gleiche Elemente in **Fig. 41** tragen gleiche Bezugsziffern. Außerdem Kästen sind mit Kästen gestrichelten Linien optionale Merkmale auf fortschrittlicheren SoCs. In **Fig. 45** ist eine Verbindungseinheit **4502** gekoppelt mit: einem Anwendungsprozessor **4510**, der einen Satz von einem oder mehreren Kernen **202A-N** und gemeinsame Cache-Einheit(en) **4106** aufweist; eine Systemagenteneinheit **4110**; eine Bussteuereinheit **4116**; eine integrierte Speichersteuereinheit **4114**; einen Satz von oder einen oder mehrere Koprozessoren **4520**, die integrierte Grafiklogik, einen Bildprozessor, einen Audioprozessor und einen Videoprozessor aufweisen kann/können; eine SRAM-Einheit (SRAM - statischer Zufallszugriffspeicher) 4530; eine DMA-Einheit (DMA - Direktzugriffsspeicher) **4532**; und eine Anzeigeeinheit **4540** zum Koppeln mit einer oder mehreren externen Anzeigen aufweist. In einer Ausführungsform weist/weisen der/die Koprozessor(en) **4520** einen Spezialprozessor auf, wie zum Beispiel einen Netzwerk- oder Kommunikationsprozessor, eine Kompressions-Engine, GPGPU, einen Hochdurchsatz-MIC-Prozessor, einen eingebetteten Prozessor oder dergleichen.

[0318] Hierin offenbarte Ausführungsformen (z. B. der Mechanismen) können in Hardware, Software, Firmware oder einer Kombination solcher Implementierungsansätze implementiert werden. Ausführungsformen der Offenbarung können als Computerprogramme oder Programmcode implementiert werden, die auf programmierbaren Systemen ausgeführt werden, die mindestens einen Prozessor, ein Speichersystem (einschließlich flüchtiger und nichtflüchtiger Speicher und/oder Speicherelemente), mindestens eine Eingabevorrichtung und mindestens eine Ausgabevorrichtung umfassen.

[0319] Ein Programmcode, wie etwa der in **Fig. 43** veranschaulichte Code **4330**, kann auf Eingabebefehle angewandt werden, um die hier beschriebenen Funktionen durchzuführen und Ausgabeinformationen zu erzeugen. Die Ausgabeinformationen können auf eine oder mehrere Ausgabevorrichtungen auf bekannte Weise angewandt werden. Zum Zweck dieser Anmeldung beinhaltet ein Verarbeitungssystem ein beliebiges System, das einen Prozessor aufweist, wie etwa zum Beispiel einen digitalen Signalprozessor (DSP), einen Mikrocontroller, einen anwendungsspezifischen integrierten Schaltkreis (ASIC: Application Specific Integrated Circuit) oder einen Mikroprozessor.

[0320] Der Programmcode kann in einer höheren prozeduralen oder objektorientierten Programmiersprache implementiert werden, um mit einem Verarbeitungssystem zu kommunizieren. Der Programmcode kann, falls gewünscht, auch in einer Assembler- oder Maschinensprache implementiert werden. Tatsächlich sind die hier beschriebenen Mechanismen in dem Schutzumfang nicht auf irgendeine bestimmte Programmiersprache beschränkt. In jedem Fall kann die Sprache eine kompilierte oder interpretierte Sprache sein.

[0321] Ein oder mehrere Aspekte von mindestens einer Ausführungsform können durch repräsentative Befehle, die auf einem maschinenlesbaren Medium gespeichert sind, das verschiedene Logik innerhalb des Prozessors repräsentiert, implementiert sein, welche, wenn sie durch eine Maschine gelesen werden, die Maschine veranlassen, Logik zu fabrizieren zum Durchführen der hier beschriebenen Techniken. Derartige Repräsentationen, als „IP-Kerne“ bekannt, können auf einem greifbaren maschinenlesbaren Medium gespeichert sein und an verschiedene Kunden oder Herstellungseinrichtungen geliefert werden, um in die Fabrikationsmaschinen geladen zu werden, die tatsächlich die Logik oder den Prozessor herstellen.

[0322] Solche maschinenlesbaren Speichermedien können ohne Einschränkung nichtflüchtige, greifbare Anordnungen von Artikeln einschließen, die durch eine Maschine oder ein Gerät hergestellt oder gebildet werden, einschließlich Speichermedien wie Festplatten, jede andere Art von Laufwerken, einschließlich Disketten, optische Disketten, CD-ROMs, CD-RWs und magnetooptische Platten, Halbleitervorrichtungen wie Nur-Lese-Speicher (ROMs), Direktzugriffsspeicher (RAMs) wie dynamische Direktzugriffsspeicher (DRAMs), statische Direktzugriffsspeicher (SRAMs), löschbare programmierbare Nur-Lese-Speicher (EPROMs), Flash-Speicher, elektrisch löschbare programmierbare Nur-Lese-Speicher (EEPROMs), Phasenwechselspeicher (PCM), magnetische oder optische Karten oder dergleichen oder jede andere Art von Medien, die zum Speichern elektronischer Anweisungen geeignet sind.

[0323] Dementsprechend schließen Ausführungsformen der Offenbarung auch nicht transitorische, greifbare maschinenlesbare Medien ein, die Befehle enthalten oder Ausgestaltungsdaten enthalten, wie zum Beispiel Hardware Description Language (HDL), die hierin beschriebene Strukturen, Schaltungen, Vorrichtungen, Prozessoren und/oder Systemmerkmale definiert. Solche Ausführungsformen können als Programmprodukte bezeichnet werden.

Emulation (einschließlich Binärübersetzung, Codeumformung usw.)

[0324] In manchen Fällen kann ein Befehlsumwandler verwendet werden, um einen Befehl von einem Quellenbefehlssatz zu einem Zielbefehlssatz umzuwandeln. Zum Beispiel kann der Befehlsumwandler einen Befehl in einen oder mehrere andere Befehle, die durch den Kern zu verarbeiten sind, übersetzen (z. B. unter Verwendung statischer Binärübersetzung, dynamischer Binärübersetzung einschließlich dynamischer Kompilation), umformen, emulieren oder anderweitig umwandeln. Der Befehlsumwandler kann in Software, Hardware, Firmware oder einer Kombination davon implementiert sein. Der Befehlsumwandler kann auf dem Prozessor, außerhalb des Prozessors oder teilweise auf und teilweise außerhalb des Prozessors sein.

[0325] **Fig. 46** ist ein Blockdiagramm, das die Verwendung eines Softwarebefehlumwandlers zum Umwandeln von binären Befehlen einem Quellbefehlssatz zu binären Befehlen in einem Zielbefehlssatz gemäß Ausführungsformen der Offenbarung kontrastiert. Bei der veranschaulichten Ausführungsform ist der Befehlsumwandler ein Softwarebefehlsumwandler, obwohl alternativ dazu der Befehlsumwandler in Software, Firmware, Hardware oder verschiedenen Kombinationen davon implementiert werden kann. **Fig. 46** zeigt ein Programm in einer höheren Sprache **4602**, das unter Verwendung eines x86-Kompilierers **4604** kompiliert werden kann, um einen x86-Binärcode **4606** zu erzeugen, der durch einen Prozessor mit wenigstens einem x86-Befehlsatz-Kern **4616** nativ ausgeführt werden kann. Der Prozessor mit wenigstens einem x86-Befehlssatz-Kern **4616** repräsentiert einen beliebigen Prozessor, der im Wesentlichen die gleichen Funktionen wie ein Intel-Prozessor mit wenigstens einem x86-Befehlssatz-Kern durch kompatibles Ausführen oder anderweitiges Verarbeiten (**1**) eines wesentlichen Teils des Befehlssatzes des Intel-x86-Befehlssatz-Kerns oder (2) von Objektcodeversionen von Anwendungen oder anderer Software, die auf einem Intel-Prozessor mit wenigstens einem x86-Befehlssatz-Kern ablaufen soll, durchführen kann, um im Wesentlichen das gleiche Ergebnis wie ein Intel-Prozessor mit einem x86-Befehlssatz-Kern zu erreichen. Der x86-Kompilierer **4604** repräsentiert einen Kompilierer, der dazu funktionsfähig ist, einen x86-Binärcode **4606** (z. B. Objektcode) zu erzeugen, der mit oder ohne zusätzliche Verknüpfungsverarbeitung auf dem Prozessor mit wenigstens einem x86-Befehlssatz-Kern **4616** ausgeführt werden kann. Gleichermaßen zeigt **Fig. 46**, dass das Programm in der höheren Sprache **4602** unter Verwendung eines Alternativer-Befehlssatz-Kompilierers **4608** kompiliert werden kann, um einen Alternativer-Befehlssatz-Binärcode **4610** zu erzeugen, der durch einen Prozessor ohne wenigstens einen x86-Befehlssatz-Kern **4614** (z. B. einen Prozessor mit Kernen, die den MIPS-Befehlssatz von MIPS Technologies of Sunnyvale, **CA**, USA ausführen und/oder den ARM-Befehlssatz von ARM Holdings of Sunnyvale, **CA**, USA ausführen) nativ ausgeführt werden kann. Der Befehlsumwandler **4612** wird verwendet, um den x86-Binärcode **4606** in einen Code umzuwandeln, der durch den Prozessor ohne einen x86 Befehlssatz-Kern **4614** nativ ausgeführt werden kann. Dieser umgewandelte Code ist wahrscheinlich nicht der gleiche wie der alternative Befehlssatz-Binärcode **4610**, da ein Befehlsumwandler, der dies kann, schwierig herzustellen ist; der umgewandelte Code wird jedoch die allgemeine Operation ausführen und aus Befehlen aus dem alternativen Befehlssatz bestehen. Dementsprechend repräsentiert der Befehlsumwandler **4612** Software, Firmware, Hardware oder eine Kombination davon, die durch Emulation, Simulation oder einen beliebigen anderen Prozess ermöglicht, dass ein Prozessor oder eine andere elektronische Vorrichtung, der/die keinen x86-Befehlssatz-Prozessor oder -Kern aufweist, den x86-Binärcode **4606** ausführt.

Patentansprüche

1. Prozessor, umfassend:

einen Kern mit einem Decodierer, um einen Befehl in einen decodierten Befehl zu decodieren, und eine Ausführungseinheit, um den decodierten Befehl auszuführen, damit eine erste Operation durchgeführt wird;
mehrere Verarbeitungselemente;
ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen, um einen Eingang eines Datenflussgraphen zu empfangen, der mehrere Knoten umfasst, wobei der Datenflussgraph in das Zwischenverbindungsnetz und die mehreren Verarbeitungselemente zu überlagern ist, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist, die mehreren Verarbeitungselemente eine zweite Operation durchzuführen haben, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht, wobei die zweite Operation eine atomare Operation ist.

2. Prozessor, umfassend:
einen Kern mit einem Decodierer, um einen Befehl in einen decodierten Befehl zu decodieren, und eine Ausführungseinheit, um den decodierten Befehl auszuführen, damit eine erste Operation durchgeführt wird;
mehrere Verarbeitungselemente;
ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen, um einen Eingang eines Datenflussgraphen zu empfangen, der mehrere Knoten umfasst, wobei der Datenflussgraph in das Zwischenverbindungsnetz und die mehreren Verarbeitungselemente zu überlagern ist, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist, die mehreren Verarbeitungselemente eine zweite Operation durchzuführen haben, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht; und
eine Transaktionssteuerung, um mehrere Speicherzugriffe zu gruppieren, die mit der zweiten Operation in Zusammenhang stehen.

3. Prozessor nach Anspruch 2, wobei die Transaktionssteuerung die mehreren Speicherzugriffe in eine Transaktion durch Markieren, mit einem Transaktionsidentifizierer, einer Cache-Zeile, die durch die zweite Operation zu modifizieren ist, zu gruppieren hat.

4. Prozessor nach Anspruch 3, wobei eine erste Nachricht zu der Transaktionssteuerung in Verbindung mit einem Start der Transaktion zu senden ist.

5. Prozessor nach Anspruch 4, wobei eine zweite Nachricht zu der Transaktionssteuerung in Verbindung mit einem Ende der Transaktion zu senden ist.

6. Prozessor nach Anspruch 5, wobei die Transaktionssteuerung als Reaktion auf die zweite Nachricht den Transaktionsidentifizierer aus der Cache-Zeile zu löschen hat.

7. Prozessor nach Anspruch 2, wobei die mehreren Speicherzugriffe einen Lesezugriff durch ein erstes der mehreren Verarbeitungselemente einschließen.

8. Prozessor nach Anspruch 7, wobei die mehreren Speicherzugriffe einen Schreibzugriff durch ein zweites der mehreren Verarbeitungselemente einschließen.

9. Prozessor nach Anspruch 8, wobei das erste und das zweite der mehreren Verarbeitungselemente unterschiedliche Verarbeitungselemente sind.

10. Prozessor nach Anspruch 8, wobei das erste und das zweite der mehreren Verarbeitungselemente das gleiche Verarbeitungselement sind.

11. Prozessor, umfassend:
einen Kern mit einem Decodierer, um einen Befehl in einen decodierten Befehl zu decodieren, und eine Ausführungseinheit, um den decodierten Befehl auszuführen, damit eine erste Operation durchgeführt wird;
mehrere Verarbeitungselemente;
ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen, um einen Eingang eines Datenflussgraphen zu empfangen, der mehrere Knoten umfasst, wobei der Datenflussgraph in das Zwischenverbindungsnetz und die mehreren Verarbeitungselemente zu überlagern ist, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist, die mehreren Verarbeitungselemente eine zweite Operation durchzuführen haben, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht; und
einen Cache, wobei der Cache in einem Speicheruntersystem einzuschließen ist, das Speicheruntersystem auch einen Speicher einzuschließen hat, in dem mehrere alte Datenwerte zu speichern sind, um eine Ausführung vom Start einer Epoche zu wiederholen, wobei die Epoche die Operation einzuschließen hat.

12. Prozessor nach Anspruch 11, wobei ein erster der mehreren alten Datenwerte bis zum Ende der Epoche im Speicher zu bewahren ist, als Reaktion darauf, dass ein entsprechender neuer Datenwert in einer Zeile des Cache durch eines der mehreren Verarbeitungselemente gespeichert wird.

13. Prozessor nach Anspruch 12, wobei der neue Datenwert von einem Schreibzugriff von einem der mehreren Verarbeitungselemente ist.

14. Prozessor nach Anspruch 13, wobei der erste der mehreren alten Datenwerte gemäß einem Cache-Kohärenzprotokoll zu bewahren ist.

15. Verfahren, umfassend:
Decodieren eines Befehls mit einem Decodierer eines Kerns eines Prozessors in einen decodierten Befehl;
Ausführen des decodierten Befehls mit einer Ausführungseinheit des Kerns des Prozessors, damit eine erste Operation durchgeführt wird;
Empfangen eines Eingangs eines Datenflussgraphen, der mehrere Knoten umfasst;
Überlagern des Datenflussgraphen in mehrere Verarbeitungselemente des Prozessors und ein Zwischenverbindungsnetz zwischen den mehreren Verarbeitungselementen des Prozessors, wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist;
Durchführen einer zweiten Operation des Datenflussgraphen mit dem Zwischenverbindungsnetz und den mehreren Verarbeitungselementen, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht; und
Bewahren mehrerer alter Datenwerte in einem Speicher während einer Epoche, wobei die Epoche ein Schreiben eines neuen Datenwerts von einem der mehreren Verarbeitungselemente einschließt, wobei der neue Wert einem der mehreren alten Datenwerte entspricht.

16. Verfahren nach Anspruch 15, ferner umfassend Erhalten, durch einen Cache gemäß einem Cache-Kohärenzprotokoll, des Besitzes einer Cache-Zeile, in die der neue Datenwert zu speichern ist.

17. Verfahren nach Anspruch 16, ferner umfassend, als Reaktion auf das Bestimmen, dass die Cache-Zeile kohärent in Besitz des Cache ist, Schreiben der Cache-Zeile in den Speicher.

18. Verfahren nach Anspruch 17, ferner umfassend Aktualisieren der Cache-Zeile zu dem neuen Wert nach dem Schreiben der Cache-Zeile in den Speicher.

19. Verfahren nach Anspruch 18, ferner umfassend Ändern der Cache-Zeile von kohärent in Besitz zu spekulativ in Besitz nach dem Schreiben der Cache-Zeile in den Speicher.

20. Verfahren nach Anspruch 16, ferner umfassend, als Reaktion auf das Bestimmen, dass die Cache-Zeile spekulativ in Besitz des Cache ist, Aktualisieren der Cache-Zeile zu dem neuen Wert, ohne ein Schreiben der Zeile in den Speicher.

21. Prozessor, umfassend:
einen Kern mit einem Decodierer, um einen Befehl in einen decodierten Befehl zu decodieren, und eine Ausführungseinheit, um den decodierten Befehl auszuführen, damit eine erste Operation durchgeführt wird;
mehrere Verarbeitungselemente;
Mittel zum Empfangen eines Eingangs eines Datenflussgraphen, der mehrere Knoten umfasst, wobei der Datenflussgraph in die Mittel und die mehreren Verarbeitungselemente zu überlagern ist,
wobei jeder Knoten als ein Datenflussoperator in den mehreren Verarbeitungselementen repräsentiert ist; die mehreren Verarbeitungselemente eine zweite Operation durchzuführen haben, wenn ein eingehender Operandensatz bei den mehreren Verarbeitungselementen eingeht; und
eine Transaktionssteuerung, um mehrere Speicherzugriffe zu gruppieren, die mit der zweiten Operation in Zusammenhang stehen.

22. Prozessor nach Anspruch 21, wobei die Transaktionssteuerung die mehreren Speicherzugriffe in eine Transaktion durch Markieren, mit einem Transaktionsidentifizierer, einer Cache-Zeile, die durch die zweite Operation zu modifizieren ist, zu gruppieren hat.

23. Prozessor nach Anspruch 22, wobei eine erste Nachricht zu der Transaktionssteuerung in Verbindung mit einem Start der Transaktion zu senden ist.

24. Prozessor nach Anspruch 23, wobei eine zweite Nachricht zu der Transaktionssteuerung in Verbindung mit einem Ende der Transaktion zu senden ist.

25. Prozessor nach Anspruch 24, wobei die Transaktionssteuerung als Reaktion auf die zweite Nachricht den Transaktionsidentifizierer aus der Cache-Zeile zu löschen hat.

Es folgen 67 Seiten Zeichnungen

Anhängende Zeichnungen

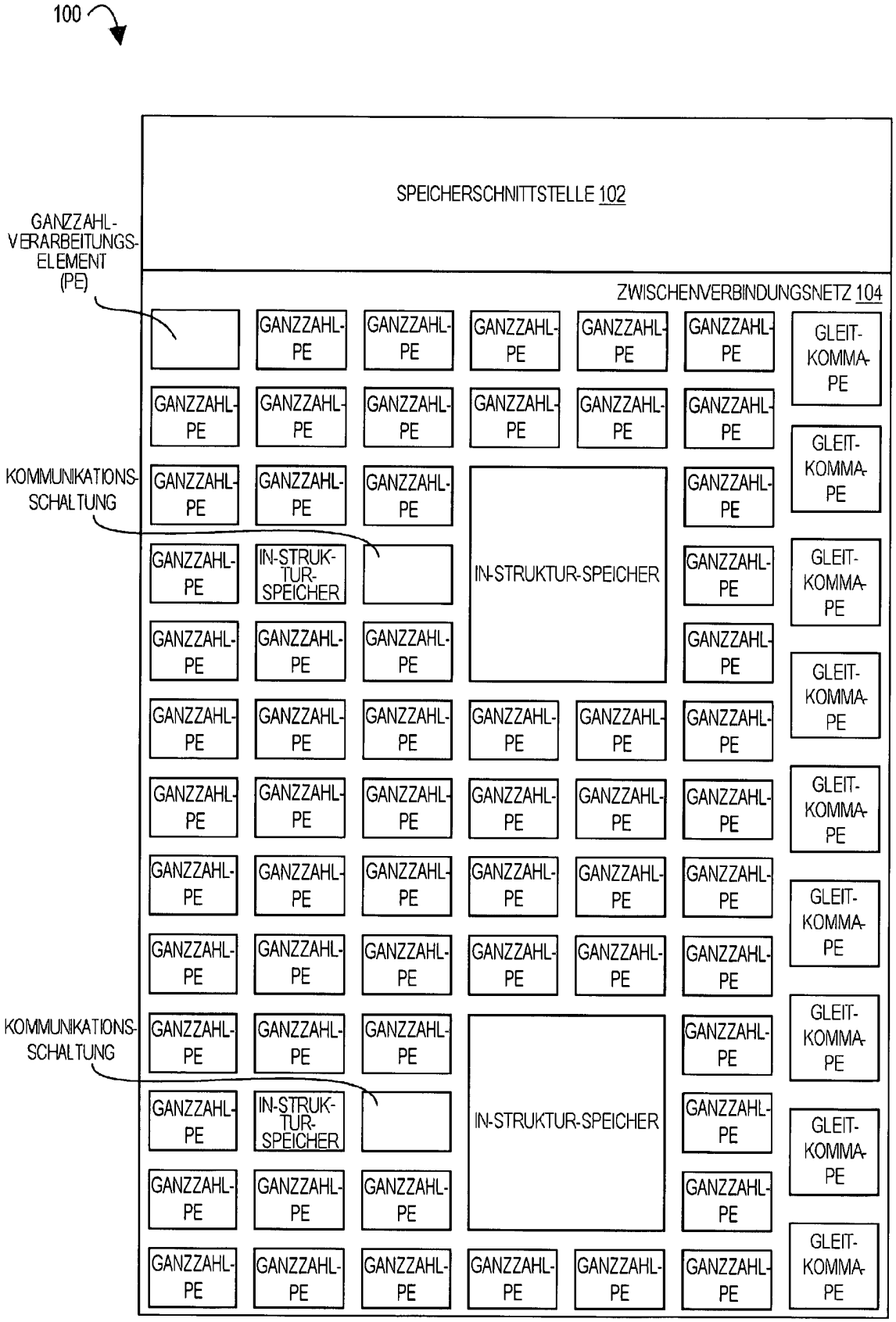


FIG. 1

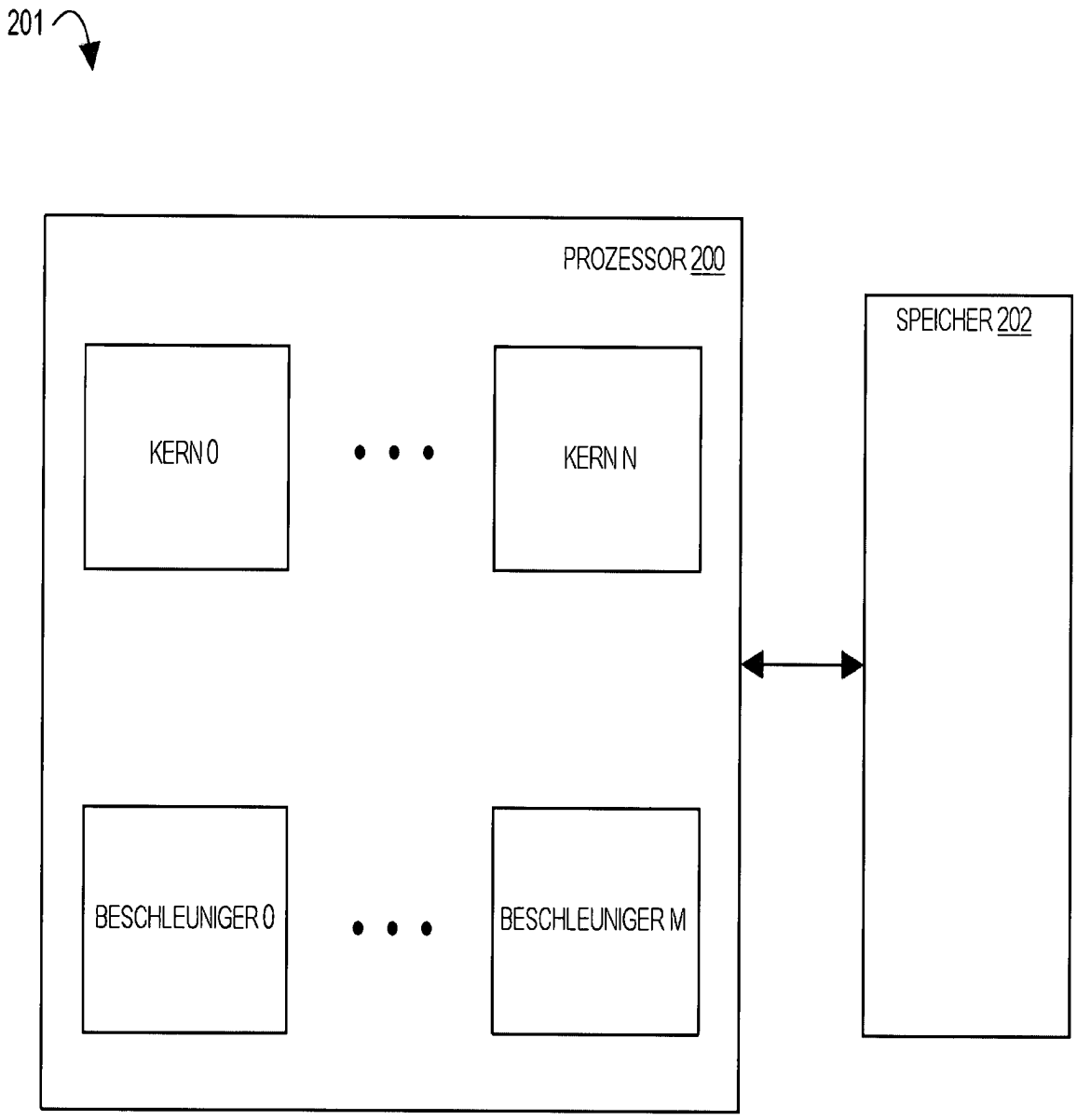
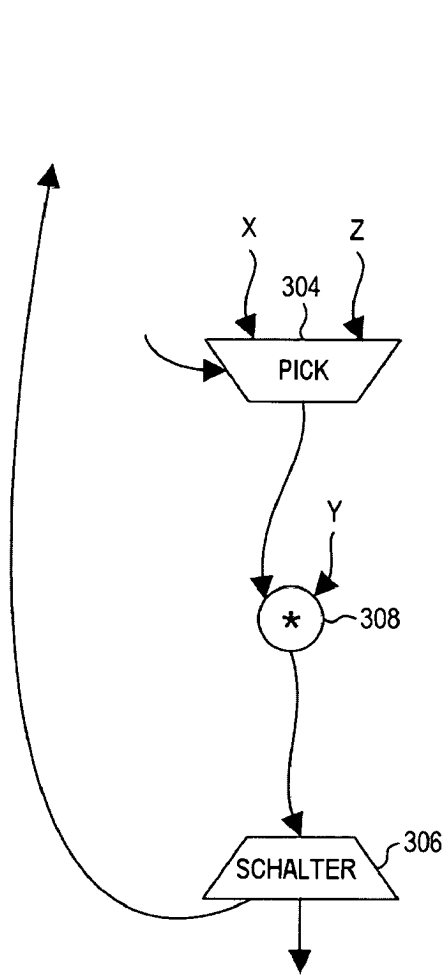

201
PROZESSOR 200
KERN 0
KERN N
SPEICHER 202
BESCHLEUNIGER 0
BESCHLEUNIGER M

FIG. 2

```
LEERFUNK (INT x, y) {
    x = x * y;
    ZURÜCK X
    }
```

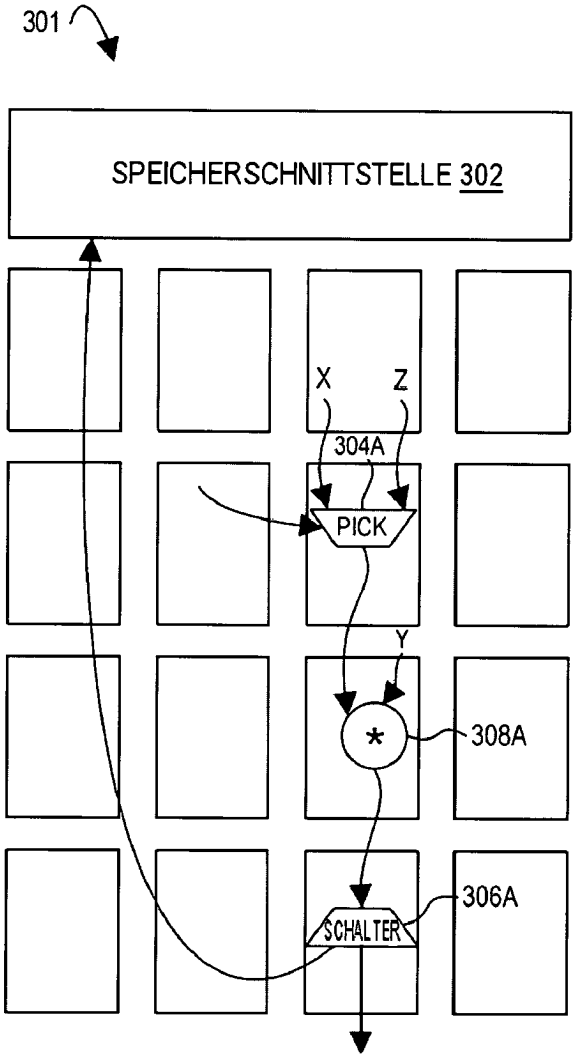
PROGRAMMQUELLE

FIG. 3A



DATENFLUSSGRAPH

FIG. 3B



KONFIGURIERTE CSA

FIG. 3C

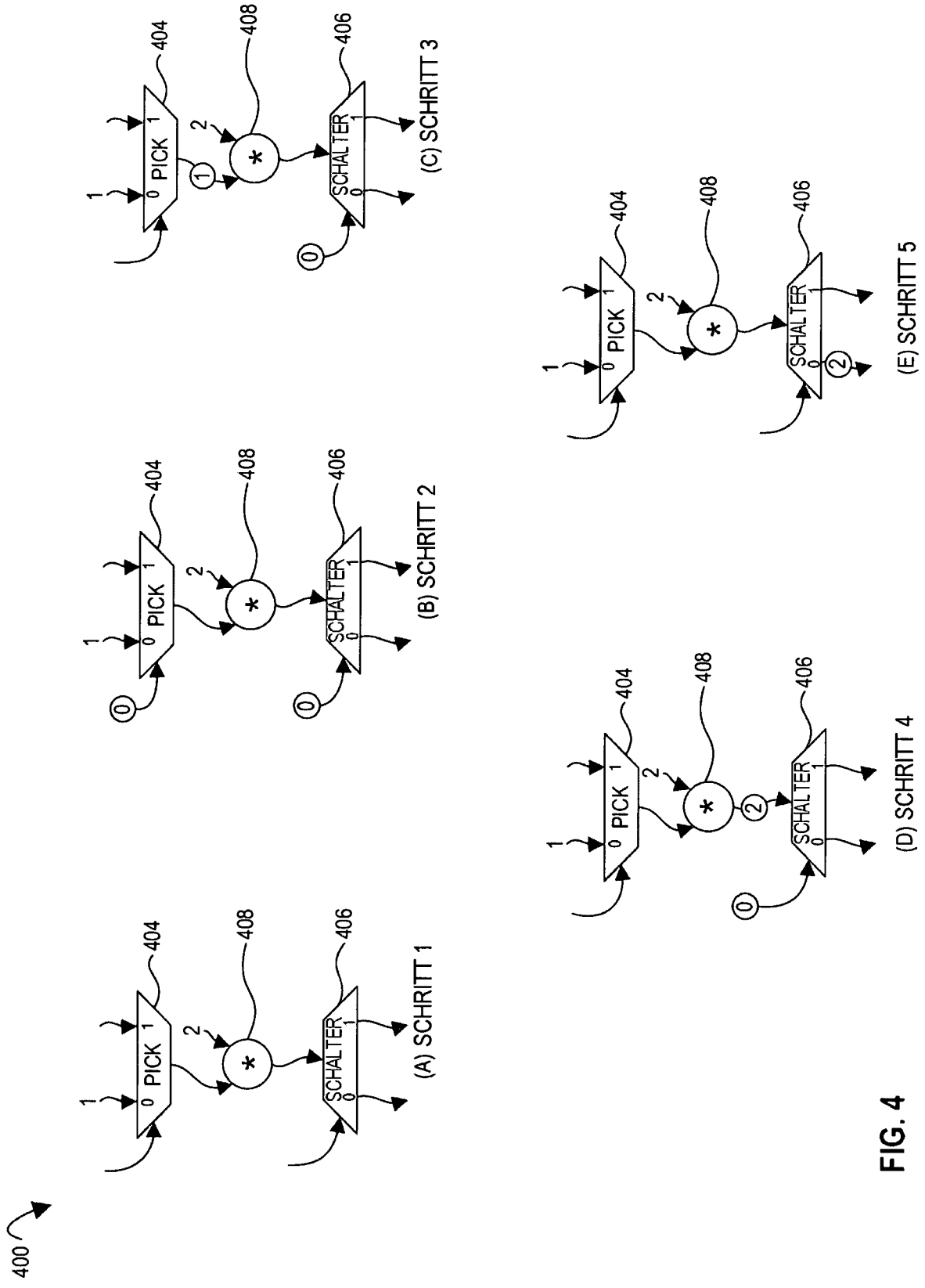

400
PICK
SCHALTER
404
406
408
(A) SCHRITT 1
(B) SCHRITT 2
(C) SCHRITT 3
(D) SCHRITT 4
(E) SCHRITT 5

FIG. 4

500

```
LEERES MEMCPY (LEER *A, LEER *B, INT N) {
FÜR (INT INDEX = 0; INDEX < N; INDEX ++) {
        a [INDEX] = b [INDEX]
    }
}
```

FIG. 5

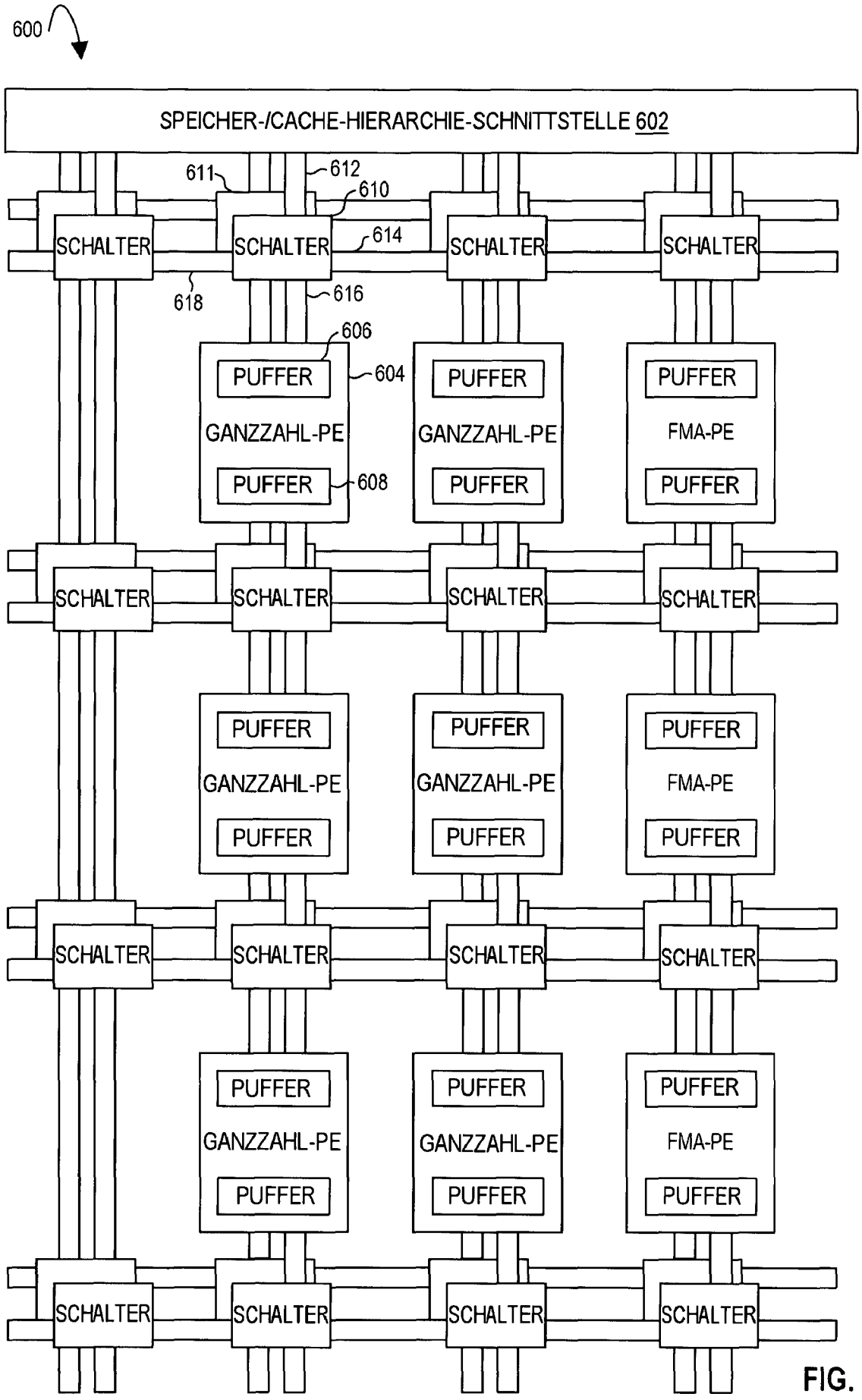

600
SPEICHER-/CACHE-HIERARCHIE-SCHNITTSTELLE 602
611
612
610
614
618
616
606
604
608
SCHALTER
PUFFER
GANZZAHL-PE
FMA-PE

FIG. 6

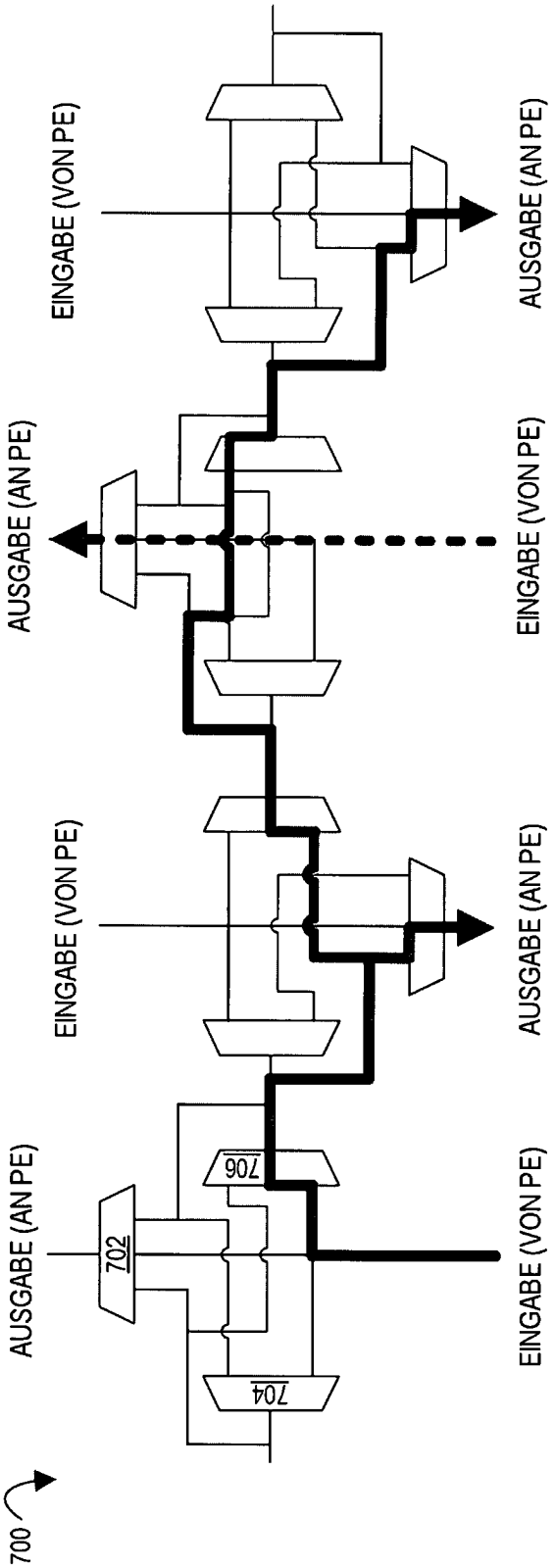

700
AUSGABE (AN PE)
EINGABE (VON PE)
AUSGABE (AN PE)
EINGABE (VON PE)
AUSGABE (AN PE)
EINGABE (VON PE)
AUSGABE (AN PE)
EINGABE (VON PE)
702
704
706

FIG. 7A

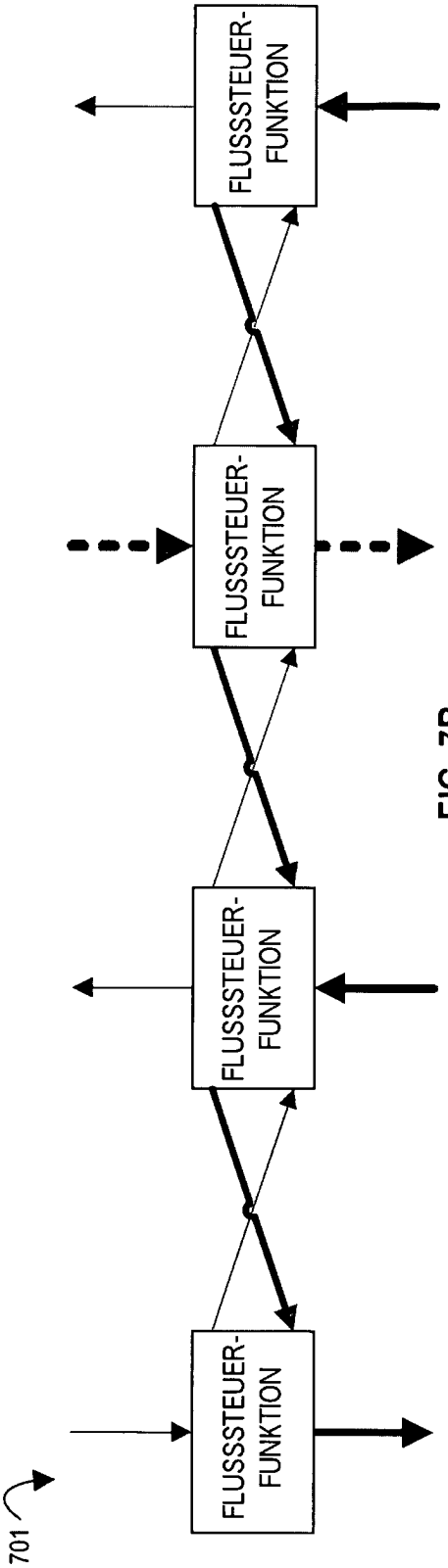

701
FLUSSSTEUER-FUNKTION
FLUSSSTEUER-FUNKTION
FLUSSSTEUER-FUNKTION
FLUSSSTEUER-FUNKTION

FIG. 7B

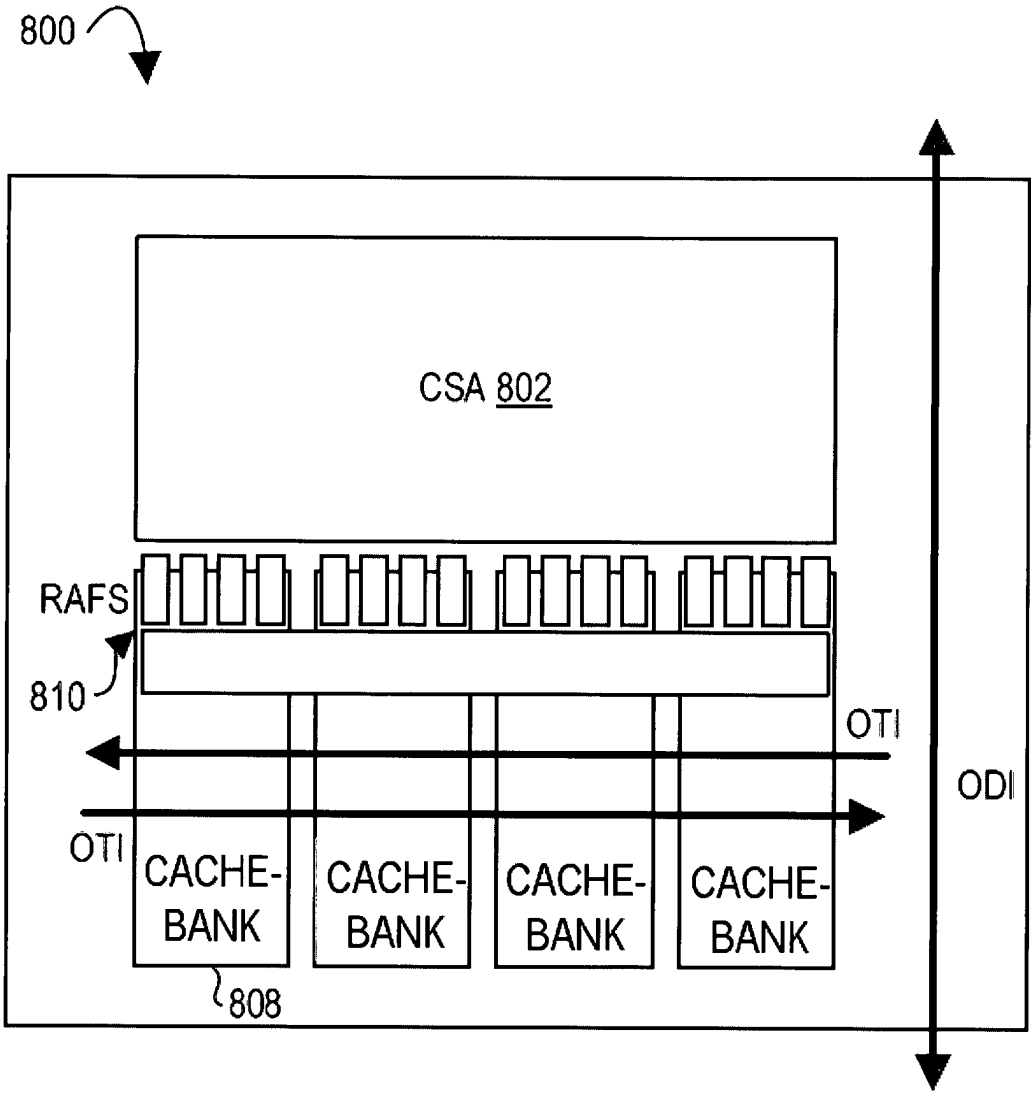
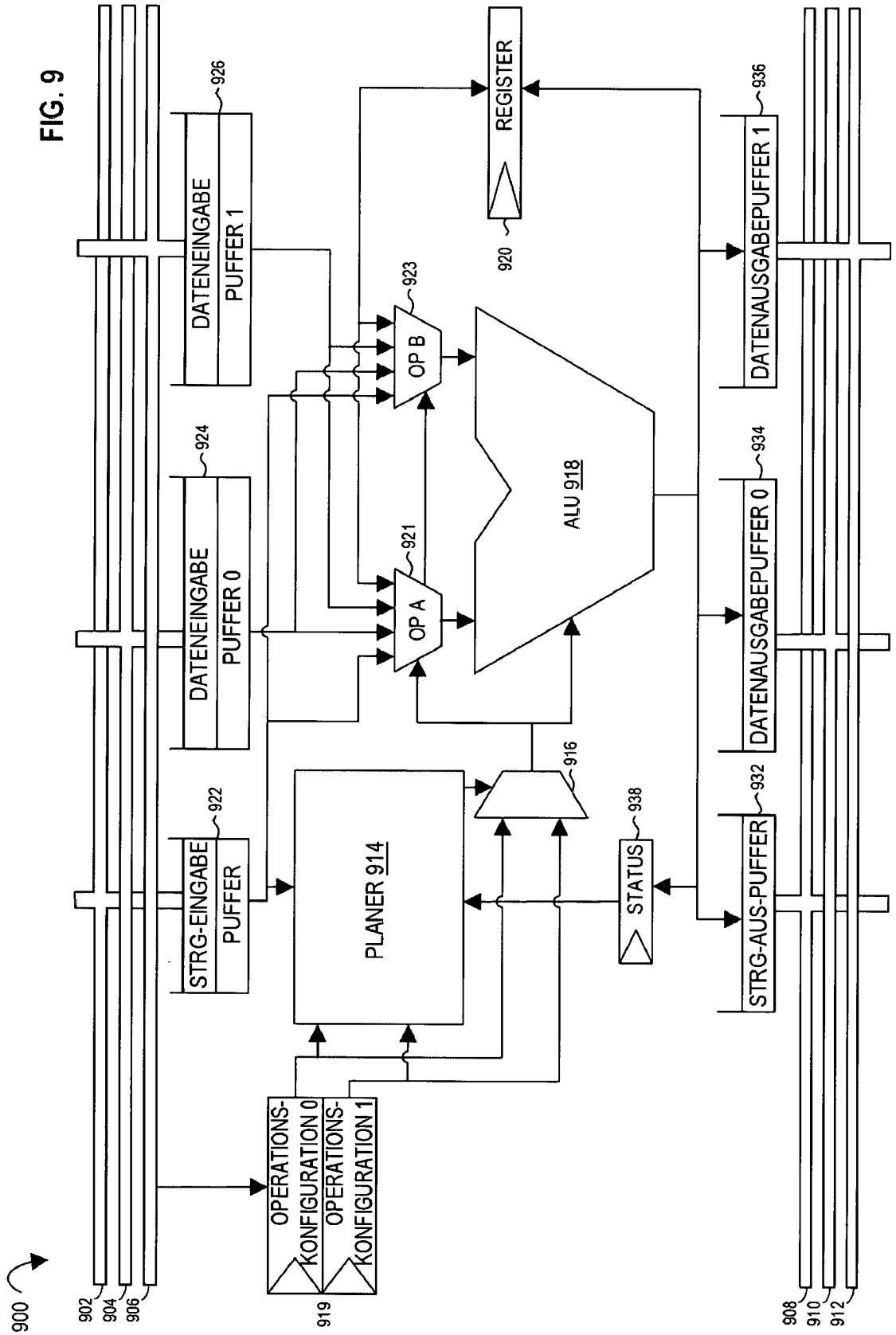

800
CSA 802
RAFS
810
OTI
OTI
ODI
CACHE-BANK
CACHE-BANK
CACHE-BANK
CACHE-BANK
808

FIG. 8

FIG. 9


900
902
904
906
908
910
912
919
920
921
922
923
924
926
932
934
936
938
DATENEINGABE PUFFER 1
DATENEINGABE PUFFER 0
STRG-EINGABE PUFFER
OPERATIONS-KONFIGURATION 0
OPERATIONS-KONFIGURATION 1
REGISTER
OP A
OP B
ALU 918
PLANER 914
STATUS
DATENAUSGABEPUFFER 1
DATENAUSGABEPUFFER 0
STRG-AUS-PUFFER

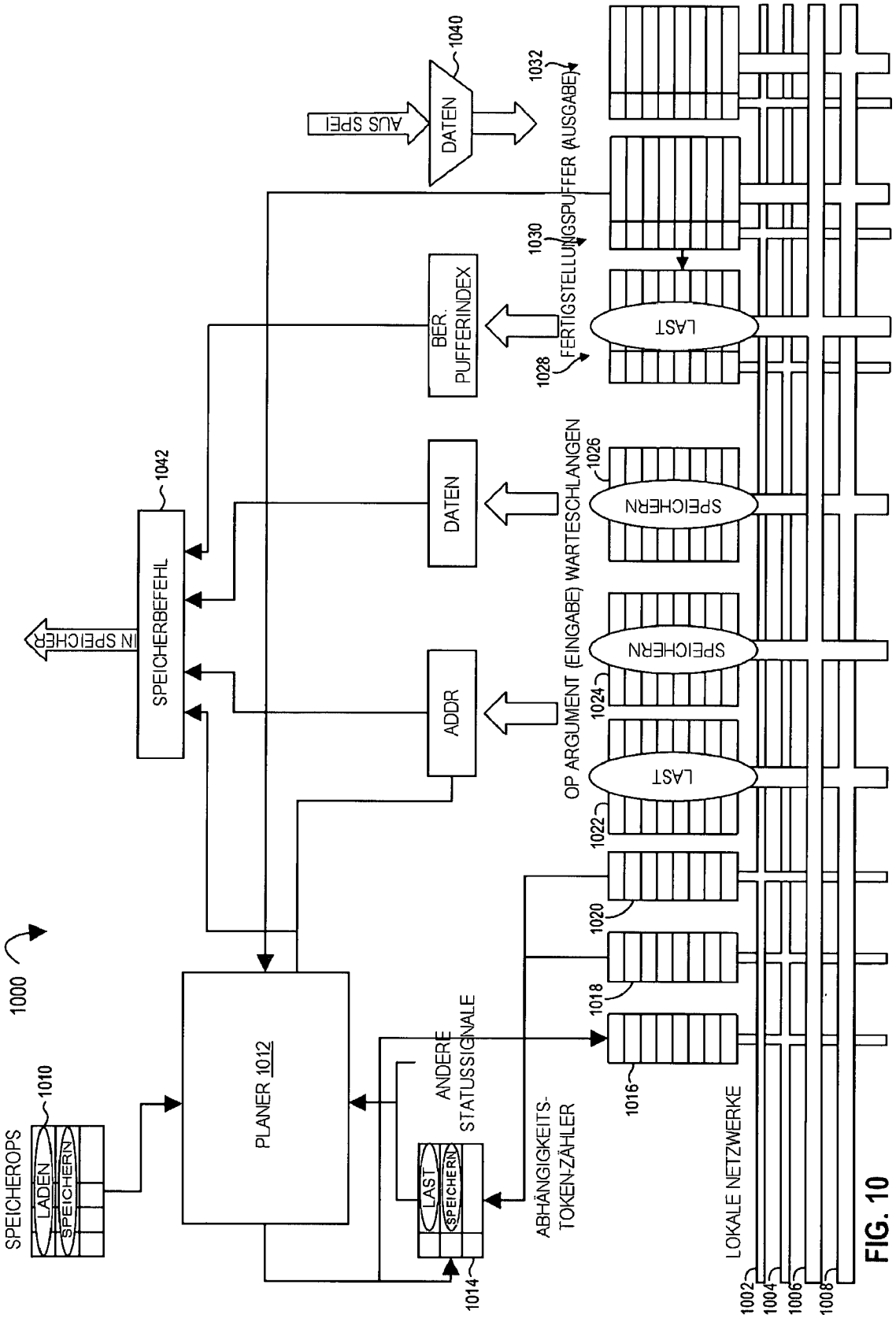

1000
SPEICHEROPS
1010
LADEN
SPEICHERN
PLANER 1012
1014
LAST
SPEICHERN
ANDERE STATUSSIGNALE
ABHÄNGIGKEITS-TOKEN-ZÄHLER
1016
1018
1020
1022
LAST
1024
SPEICHERN
1026
SPEICHERN
OP ARGUMENT (EINGABE) WARTESCHLANGEN
LOKALE NETZWERKE
1002
1004
1006
1008
1028
FERTIGSTELLUNGSPUFFER (AUSGABE)
1030
LAST
1032
DATEN
1040
AUS SPEI
BER. PUFFERINDEX
DATEN
ADDR
1042
SPEICHERBEFEHL
IN SPEICHER

FIG. 10

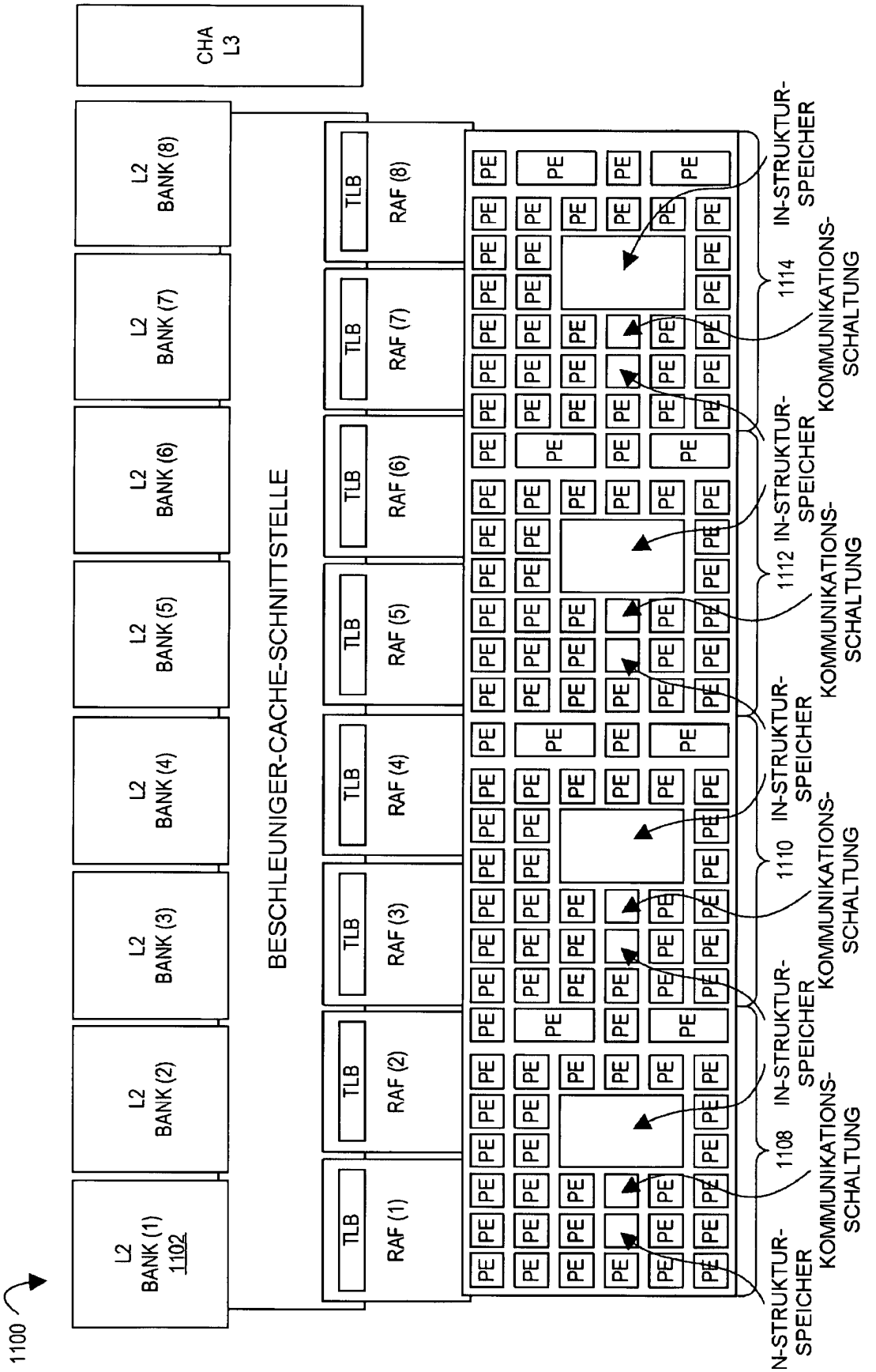

1100
CHA L3
L2 BANK (8)
L2 BANK (7)
L2 BANK (6)
L2 BANK (5)
L2 BANK (4)
L2 BANK (3)
L2 BANK (2)
L2 BANK (1) 1102
BESCHLEUNIGER-CACHE-SCHNITTSTELLE
TLB
RAF (8)
RAF (7)
RAF (6)
RAF (5)
RAF (4)
RAF (3)
RAF (2)
RAF (1)
PE
IN-STRUKTUR-SPEICHER
KOMMUNIKATIONS-SCHALTUNG
1114
1112
1110
1108

FIG. 11A

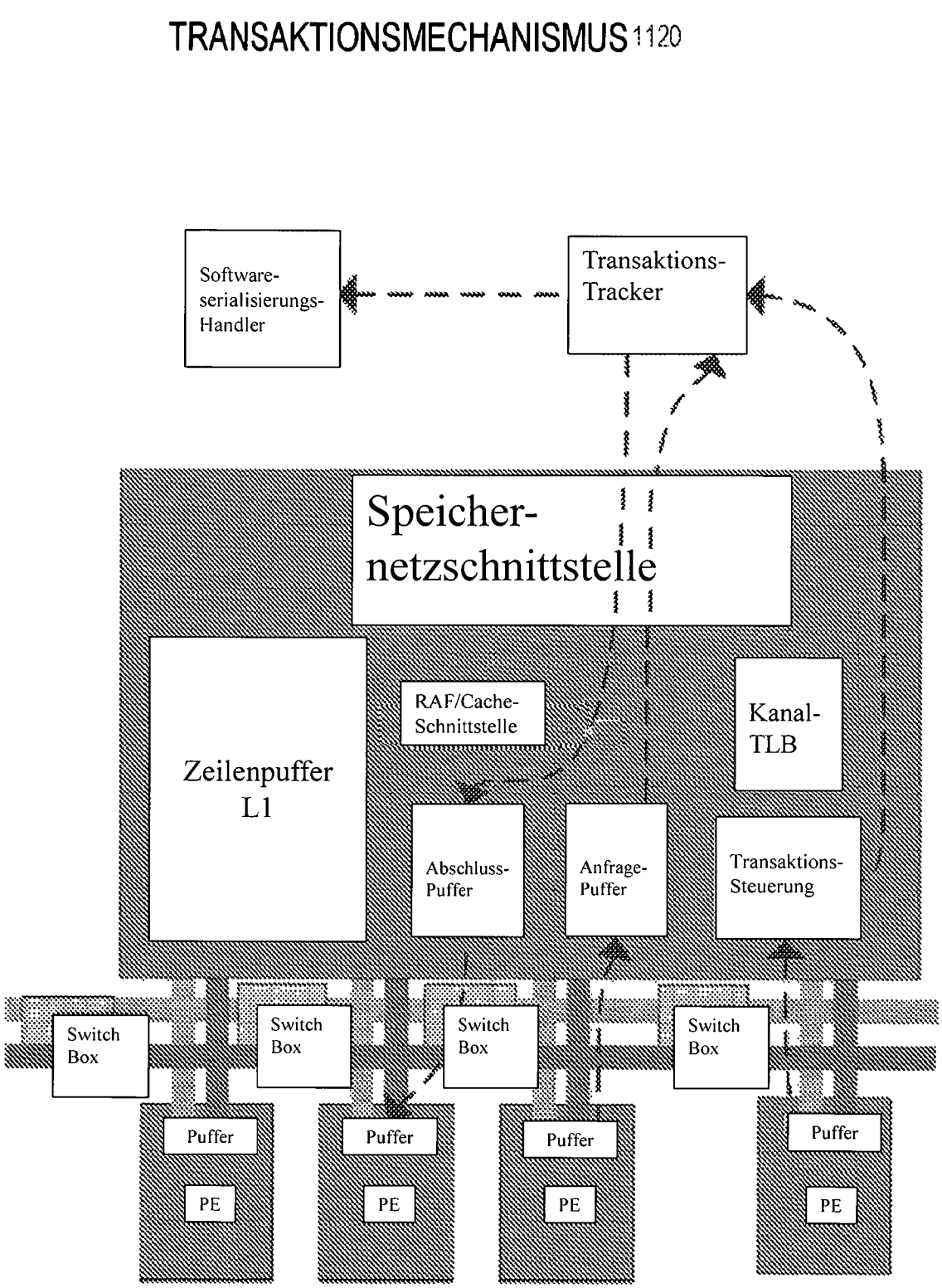

TRANSAKTIONSMECHANISMUS 1120
Software-serialisierungs-Handler
Transaktions-Tracker
Speicher-netzschnittstelle
Zeilenpuffer L1
RAF/Cache-Schnittstelle
Kanal-TLB
Abschluss-Puffer
Anfrage-Puffer
Transaktions-Steuerung
Switch Box
Switch Box
Switch Box
Switch Box
Puffer
Puffer
Puffer
Puffer
PE
PE
PE
PE

FIG. 11B

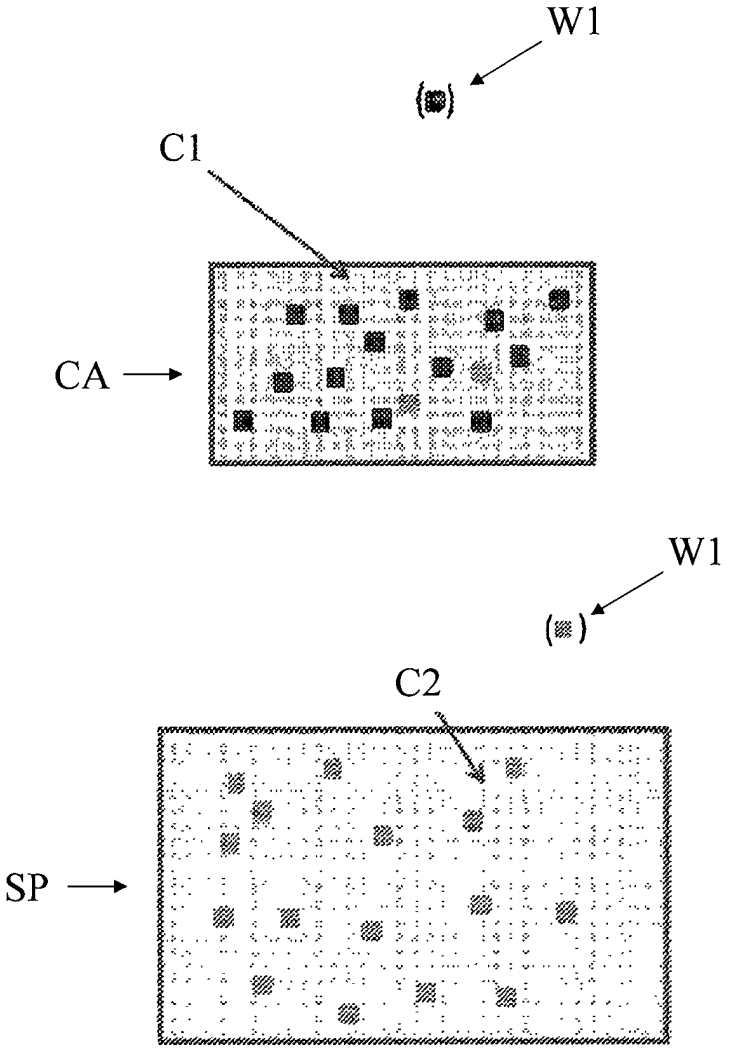


FIG. 11C

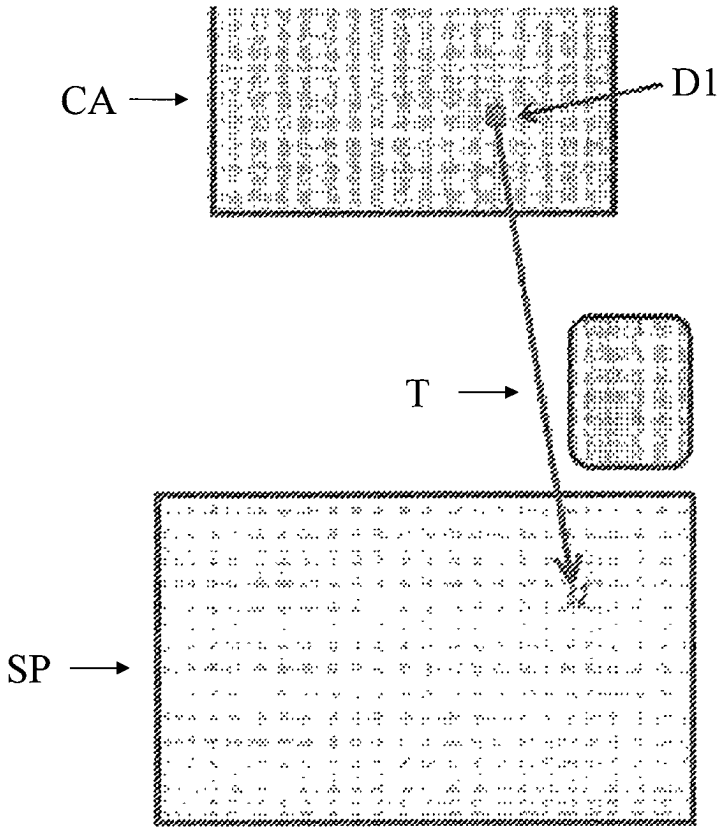


FIG. 11D

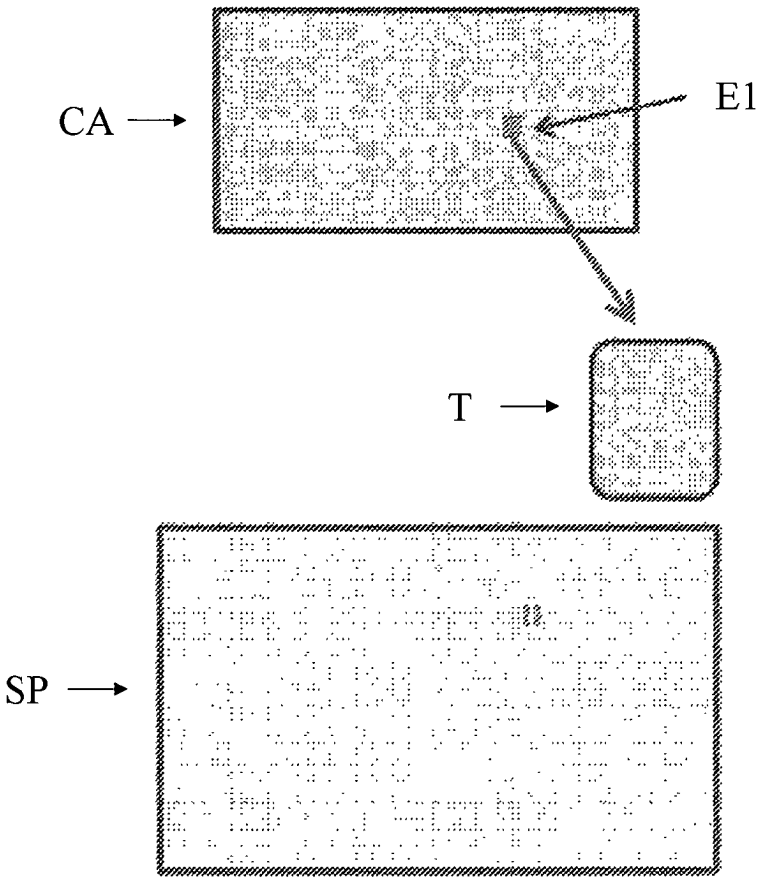


FIG. 11E

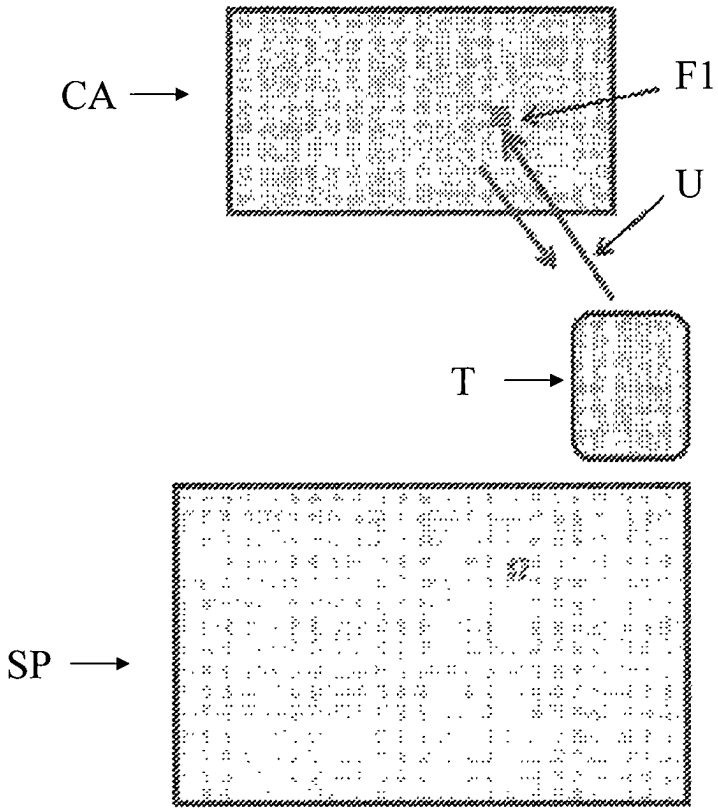


FIG. 11F

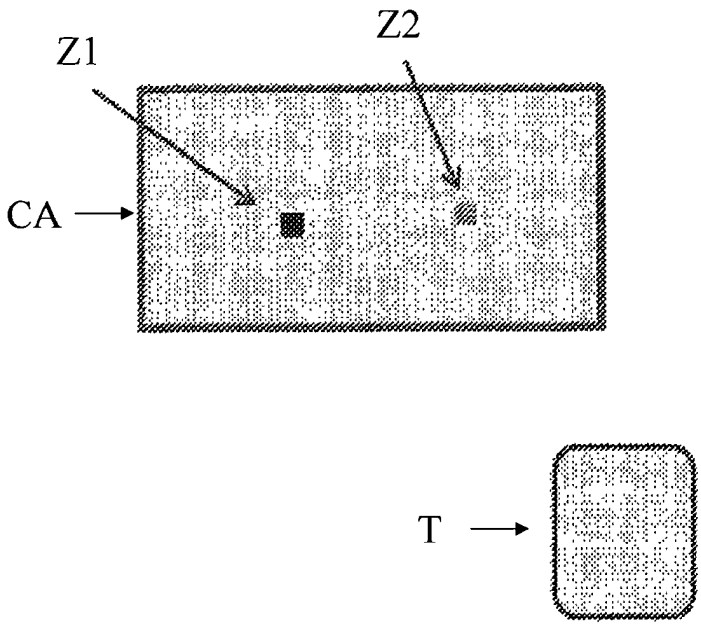


FIG. 11G

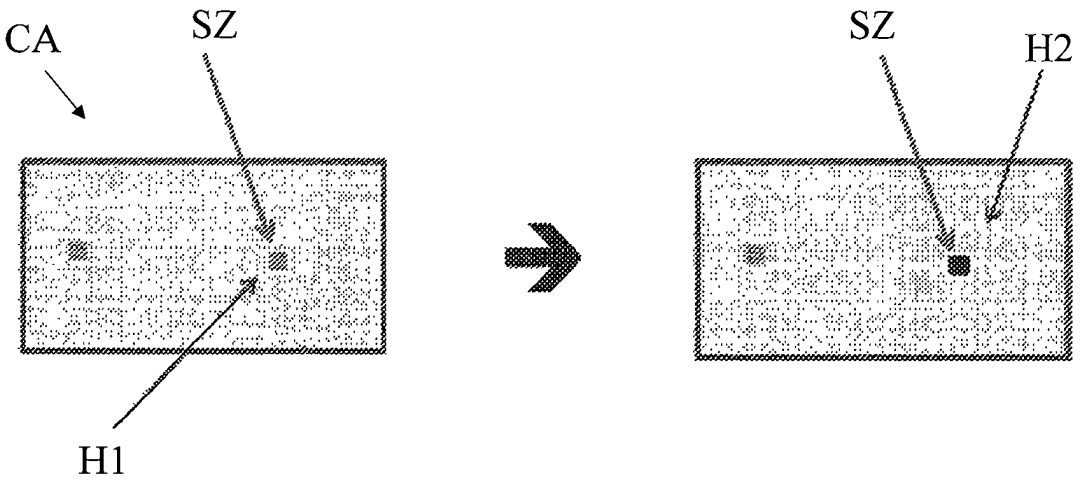


FIG. 11H

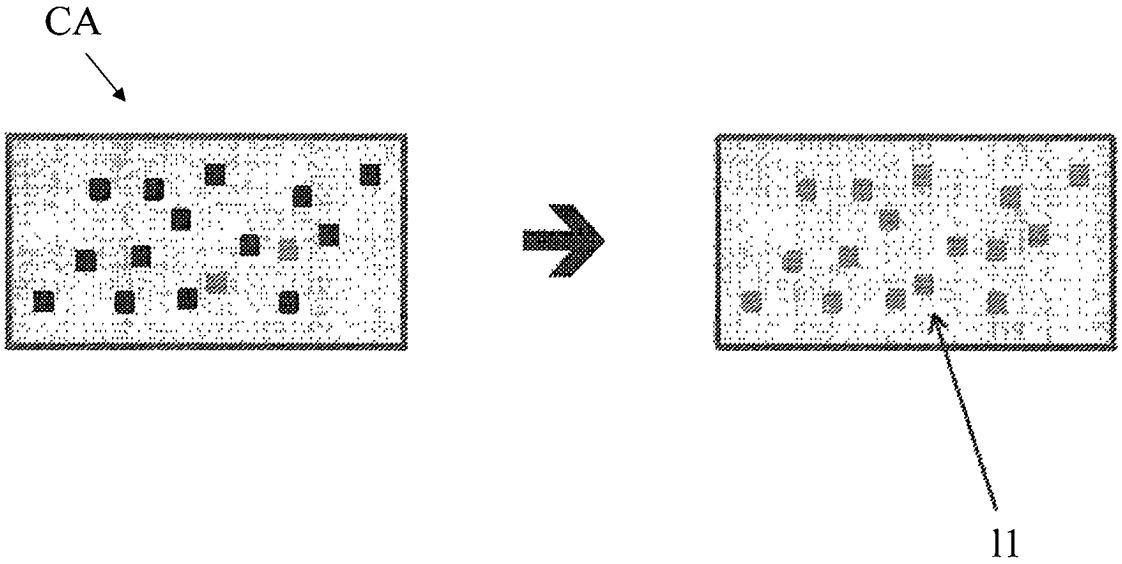


FIG. 11I

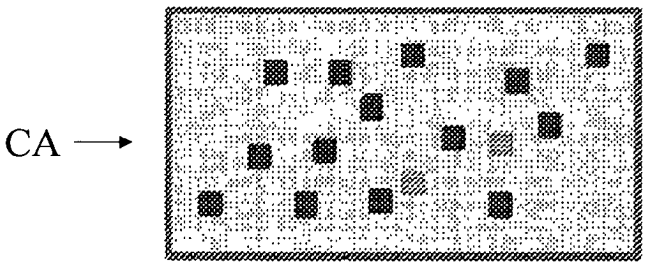


FIG. 11J

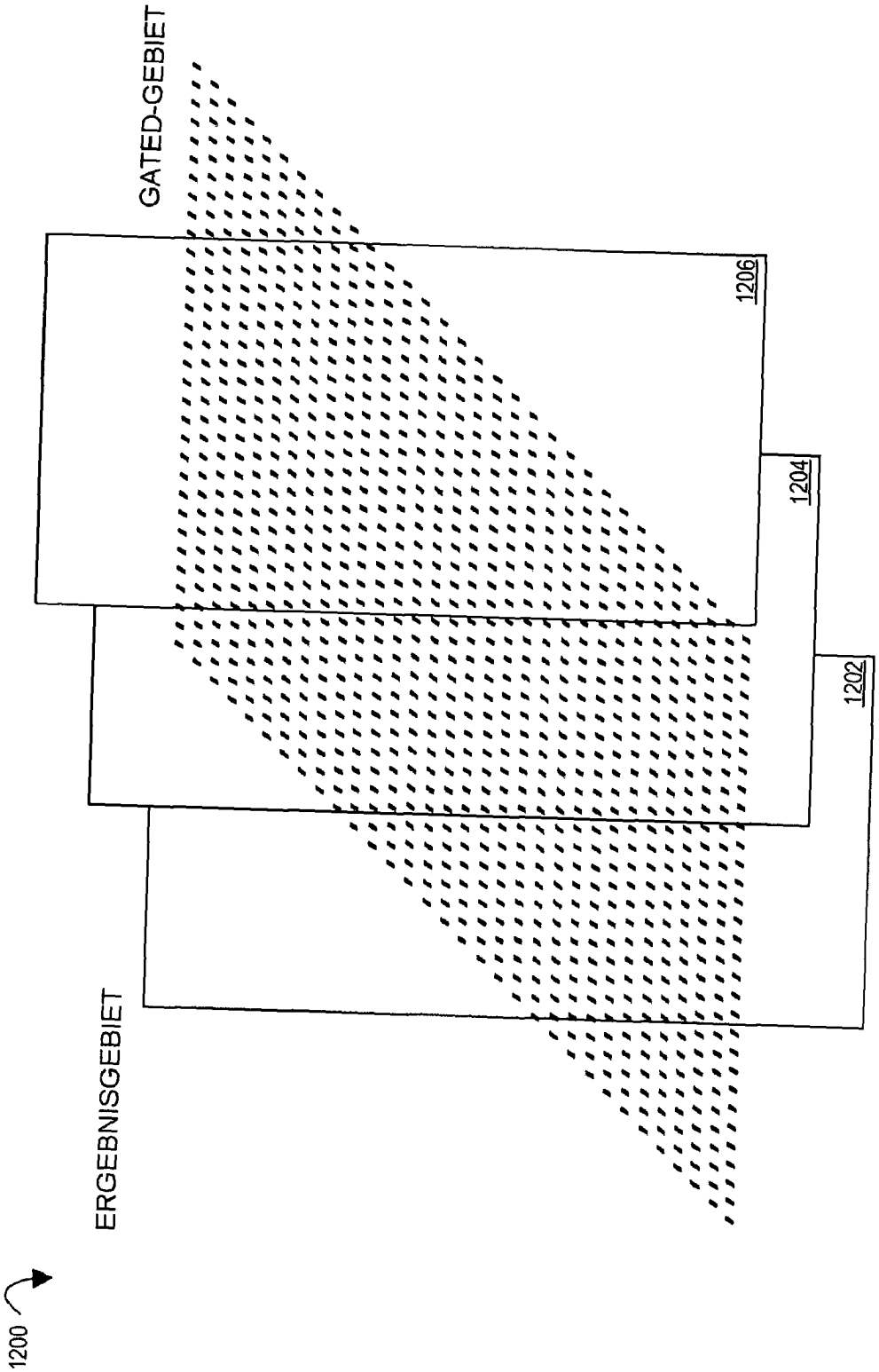


FIG. 12

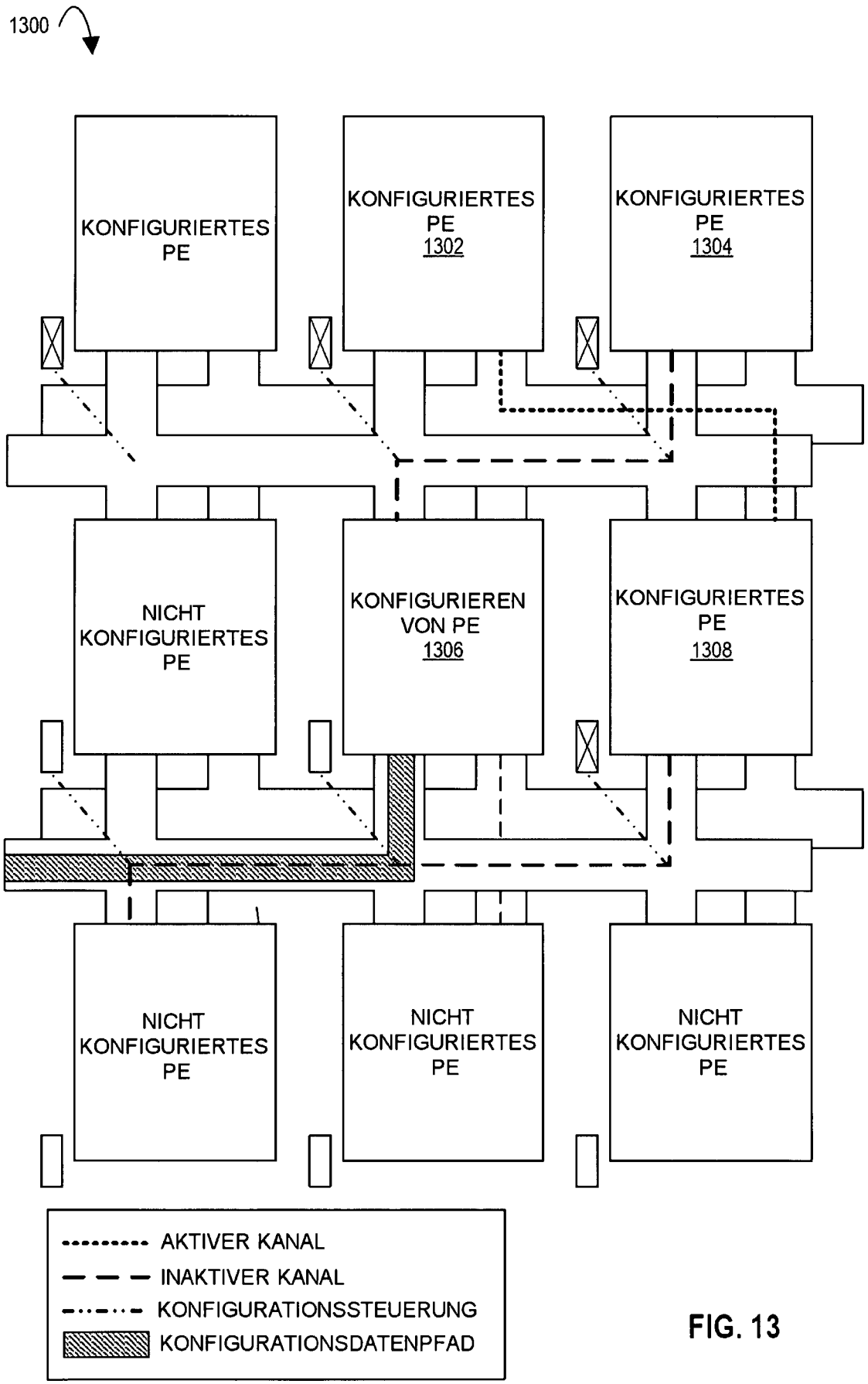

1300
KONFIGURIERTES PE
KONFIGURIERTES PE 1302
KONFIGURIERTES PE 1304
NICHT KONFIGURIERTES PE
KONFIGURIEREN VON PE 1306
KONFIGURIERTES PE 1308
NICHT KONFIGURIERTES PE
NICHT KONFIGURIERTES PE
NICHT KONFIGURIERTES PE
AKTIVER KANAL
INAKTIVER KANAL
KONFIGURATIONSSTEUERUNG
KONFIGURATIONSDATENPFAD

FIG. 13

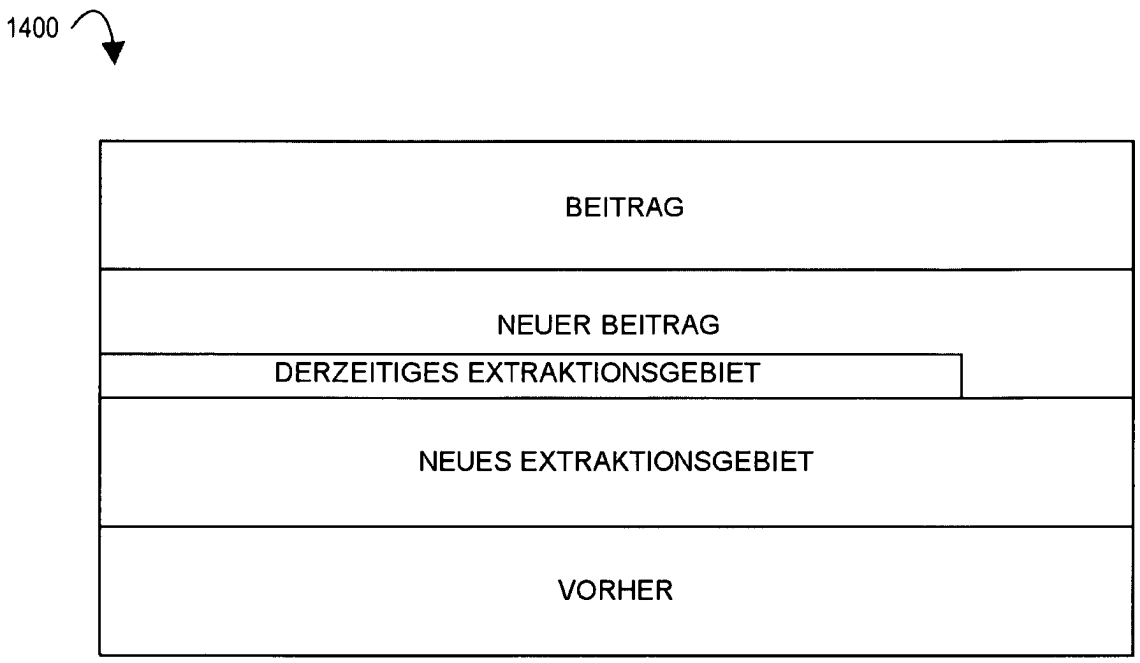

1400
BEITRAG
NEUER BEITRAG
DERZEITIGES EXTRAKTIONSGEBIET
NEUES EXTRAKTIONSGEBIET
VORHER

FIG. 14

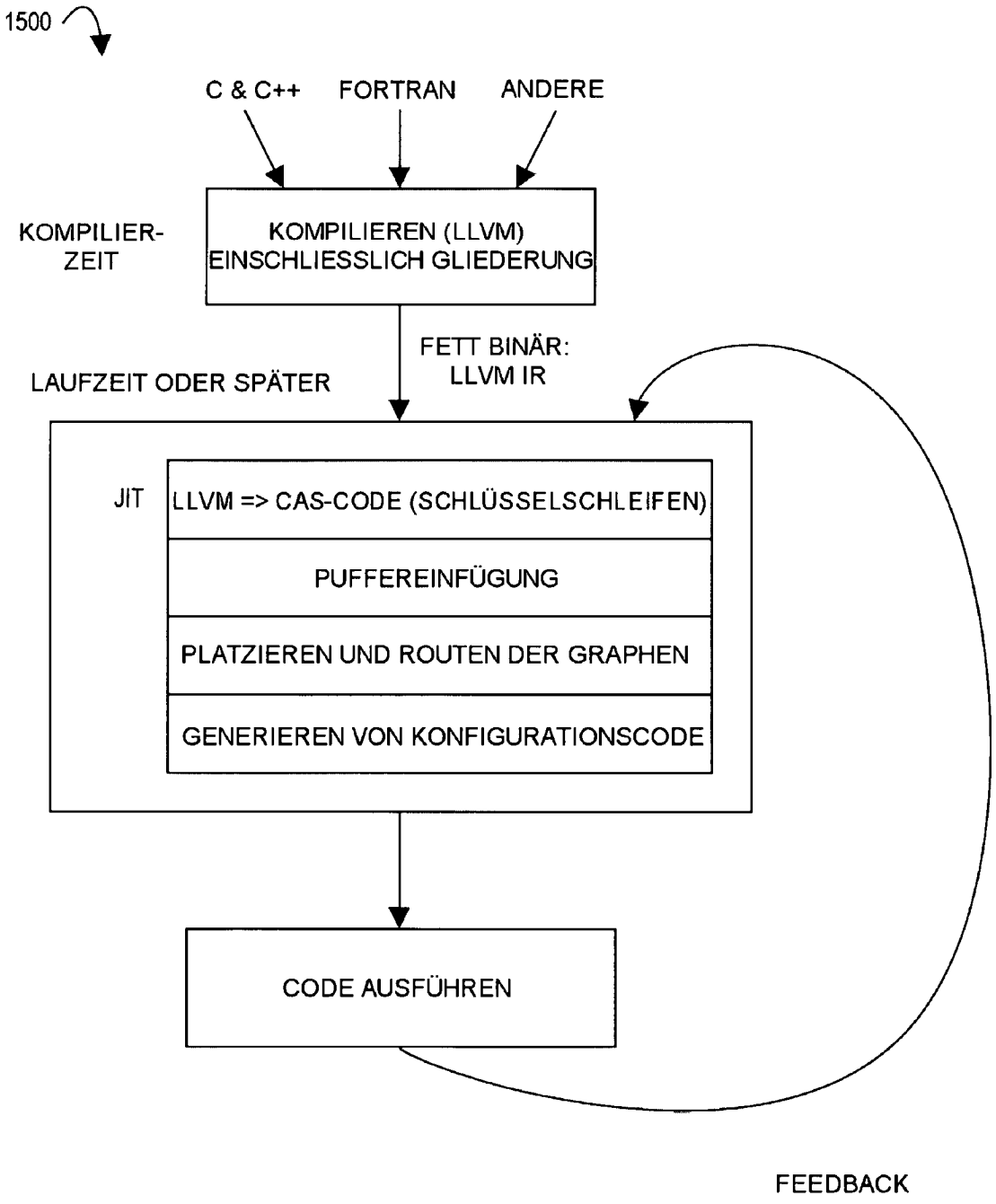

1500
C & C++
FORTRAN
ANDERE
KOMPILIER-ZEIT
KOMPILIEREN (LLVM) EINSCHLIESSLICH GLIEDERUNG
FETT BINÄR: LLVM IR
LAUFZEIT ODER SPÄTER
JIT
LLVM => CAS-CODE (SCHLÜSSELSCHLEIFEN)
PUFFEREINFÜGUNG
PLATZIEREN UND ROUTEN DER GRAPHEN
GENERIEREN VON KONFIGURATIONSCODE
CODE AUSFÜHREN
FEEDBACK

FIG. 15

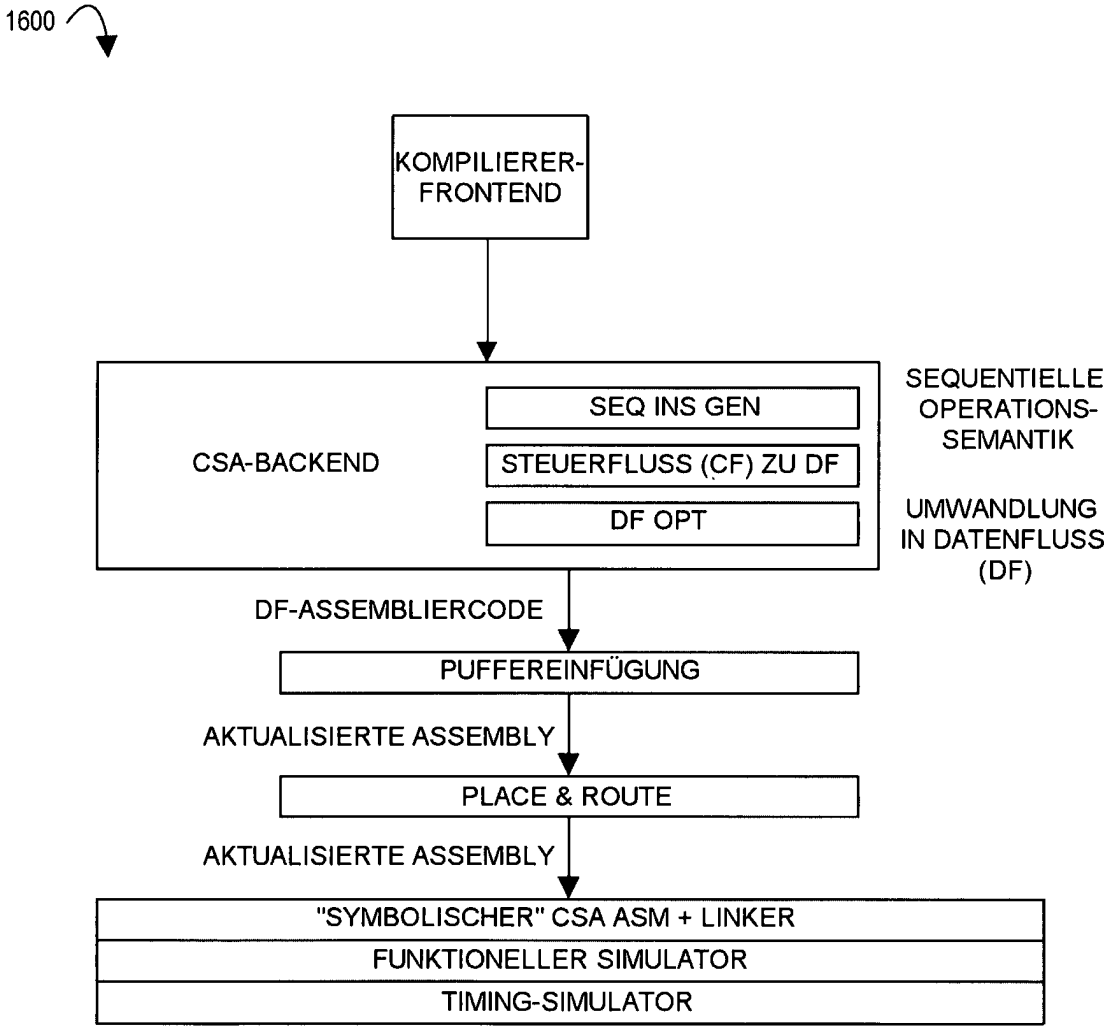

1600
KOMPILIERER-FRONTEND
CSA-BACKEND
SEQ INS GEN
STEUERFLUSS (CF) ZU DF
DF OPT
SEQUENTIELLE OPERATIONS-SEMANTIK
UMWANDLUNG IN DATENFLUSS (DF)
DF-ASSEMBLIERCODE
PUFFEREINFÜGUNG
AKTUALISIERTE ASSEMBLY
PLACE & ROUTE
AKTUALISIERTE ASSEMBLY
"SYMBOLISCHER" CSA ASM + LINKER
FUNKTIONELLER SIMULATOR
TIMING-SIMULATOR

FIG. 16

```
ld32 Rdata, Raddr
ld32 Rdata2, Raddr2
mul32 Rv0, Rdata, 17
mul32 Rv1, Rdata2, Rdata2
add32 Rres, Rv0, Rv1
st32 Raddr, Rres
ld32 Rdata3, Raddr3
```

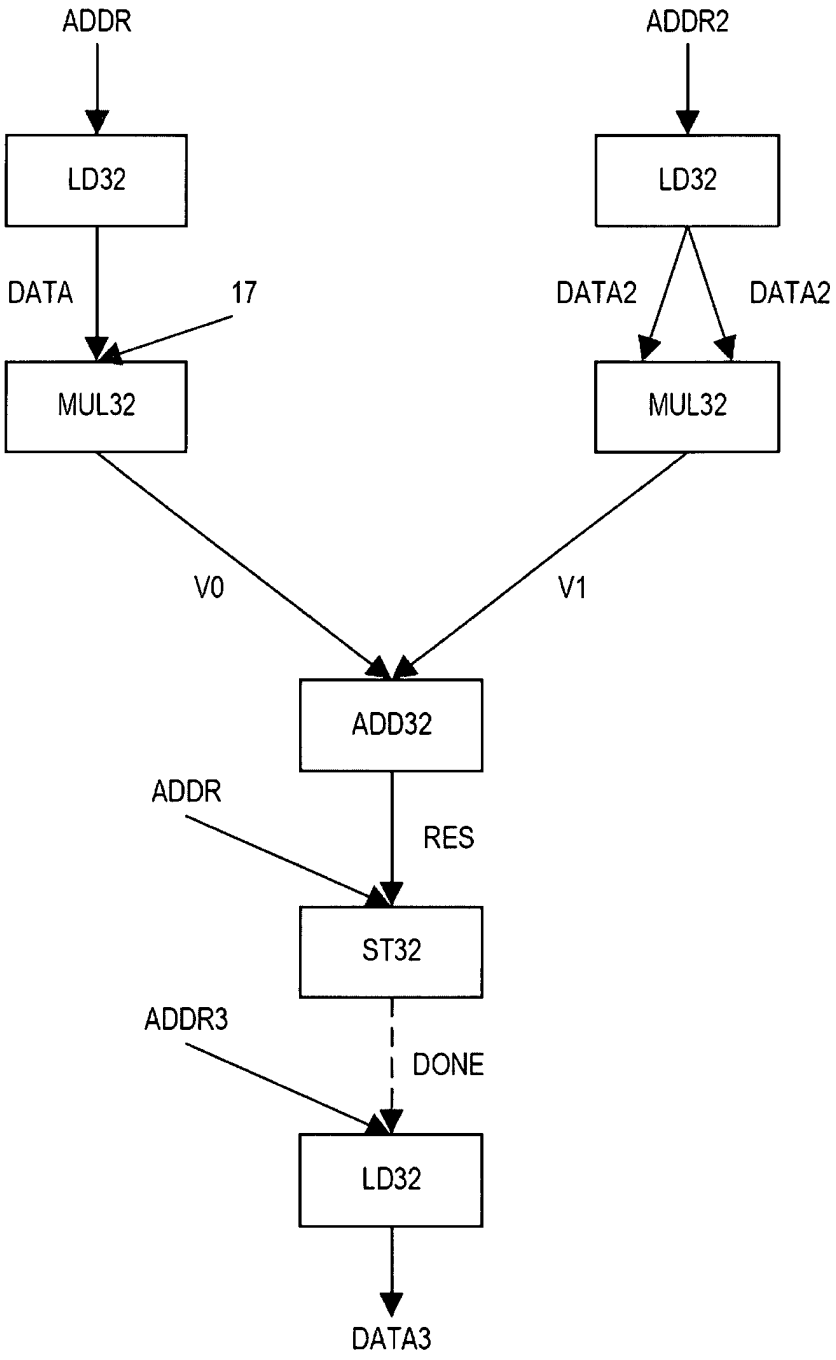
SEQUENTIELLE ASSEMBLY 1702

FIG. 17A

```
.lic .i32 data; .lic .i64 addr;
.lic .i32 data2; .lic .i64 addr2;
.lic .i32 data3; .lic .i64 addr3;
.lic .i32 v0; .lic .i32 v1; .lic .i32 res
ld32 data, addr
ld32 data2, addr2
mul32 v0, data, 17
mul32 v1, data2, data2
add32 res, v0, v1
st32 addr, res, done, %ign
ld32 data3, addr3, %ign, done
```

DATENFLUSS-ASSEMBLY 1704

FIG. 17B



DATENFLUSSGRAPH 1706

FIG. 17C

```
if (i < n)
    y = x + a;
else
    y = i + x;
```

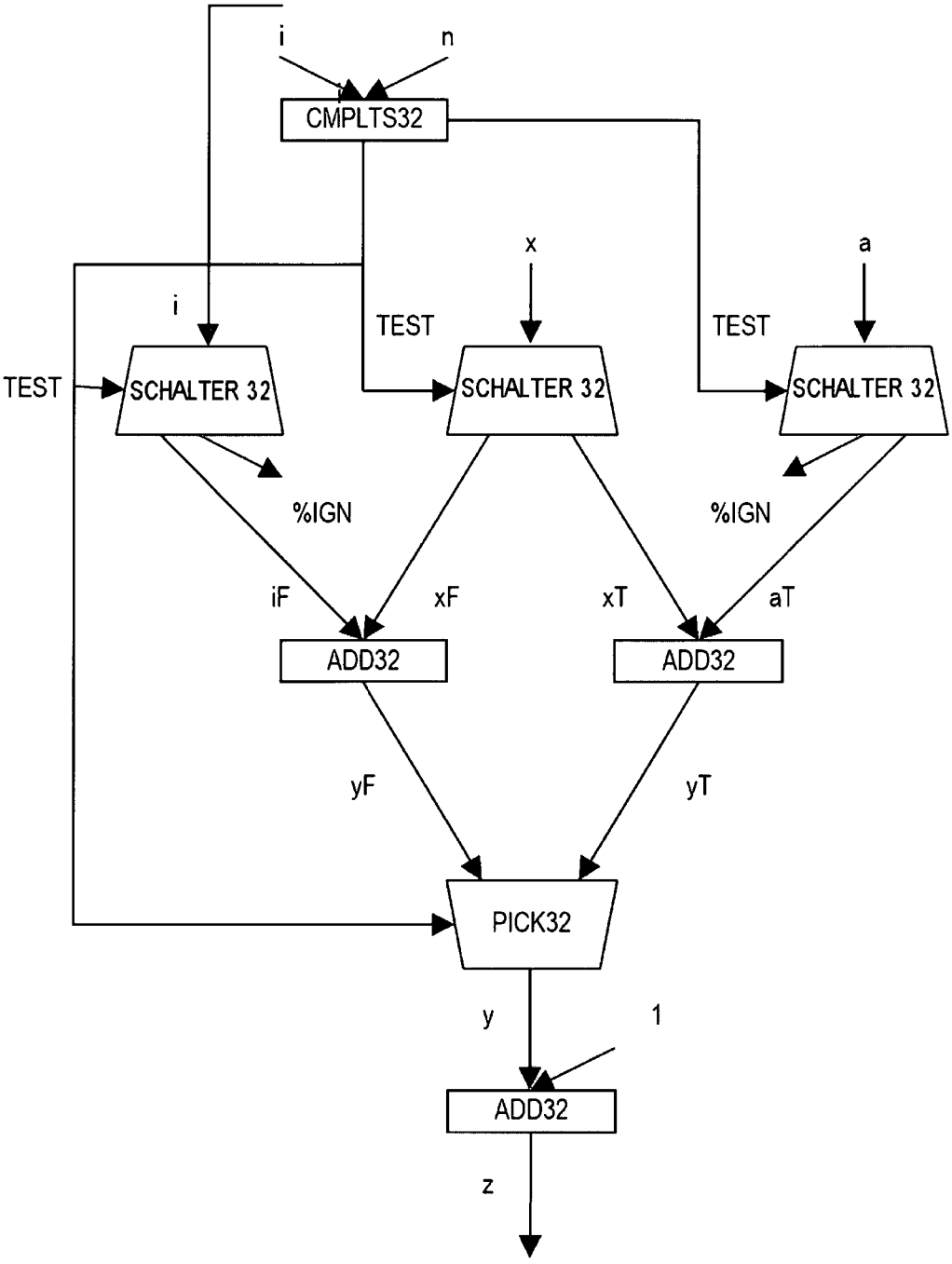
C-QUELLCODE 1802

FIG. 18A

```
.lic .i1 test
cmplts32 test, i, n
switch32 %ign, aT, test, a
switch32 iF, %ign, test, i
switch32 xF, xT, test, x
add32 yT, xT, aT      # True path
add32 yF, iF, xF      # False path
pick32 y, test, yF, yT
add32 z, y, 1
```

DATENFLUSS-ASSEMBLY 1804

FIG. 18B



DATENFLUSSGRAPH 1806

FIG. 18C

```
int i = 0;
int sum = 0;
do {
    sum = sum + i;
    i = i + 1;
} while (i < n);
return sum;
```

C-QUELLCODE 1902

FIG. 19A

```
# Schleifensteuerkanäle.
.lic .i1 picker
.lic .i1 switcher

# Offset-Werte im Picker mit einer Anfänglichen 0.
.curr picker; .value 0; .avail 0

# Erzeuge den Wert von i für jede Schleifeniteration
pick32 top_i, picker, init_i, loopback_i
add32 bottom_i, top_i, 1
switch32 %ign, loopback_i, switcher, bottom_i

# Wiederhole den Wert von n für jede Ausführung der Schleife.
pick32 loop_n, picker, init_n, loopback_n
switch32 %ign, loopback_n, switcher, loop_n

# Vergleich am Ende der Schleife.
cmplts32 switcher, bottom_i, loop_n
mov1 picker, switcher

# Addiere die Summe um die Schleifeniteration.
pick32 top_sum, picker, init_sum, loopback_sum
add32 bottom_sum, top_sum, top_i
switch32 out_sum, loopback_sum, switcher, bottom_sum
```

DATENFLUSS-ASSEMBLY 1904

FIG. 19B

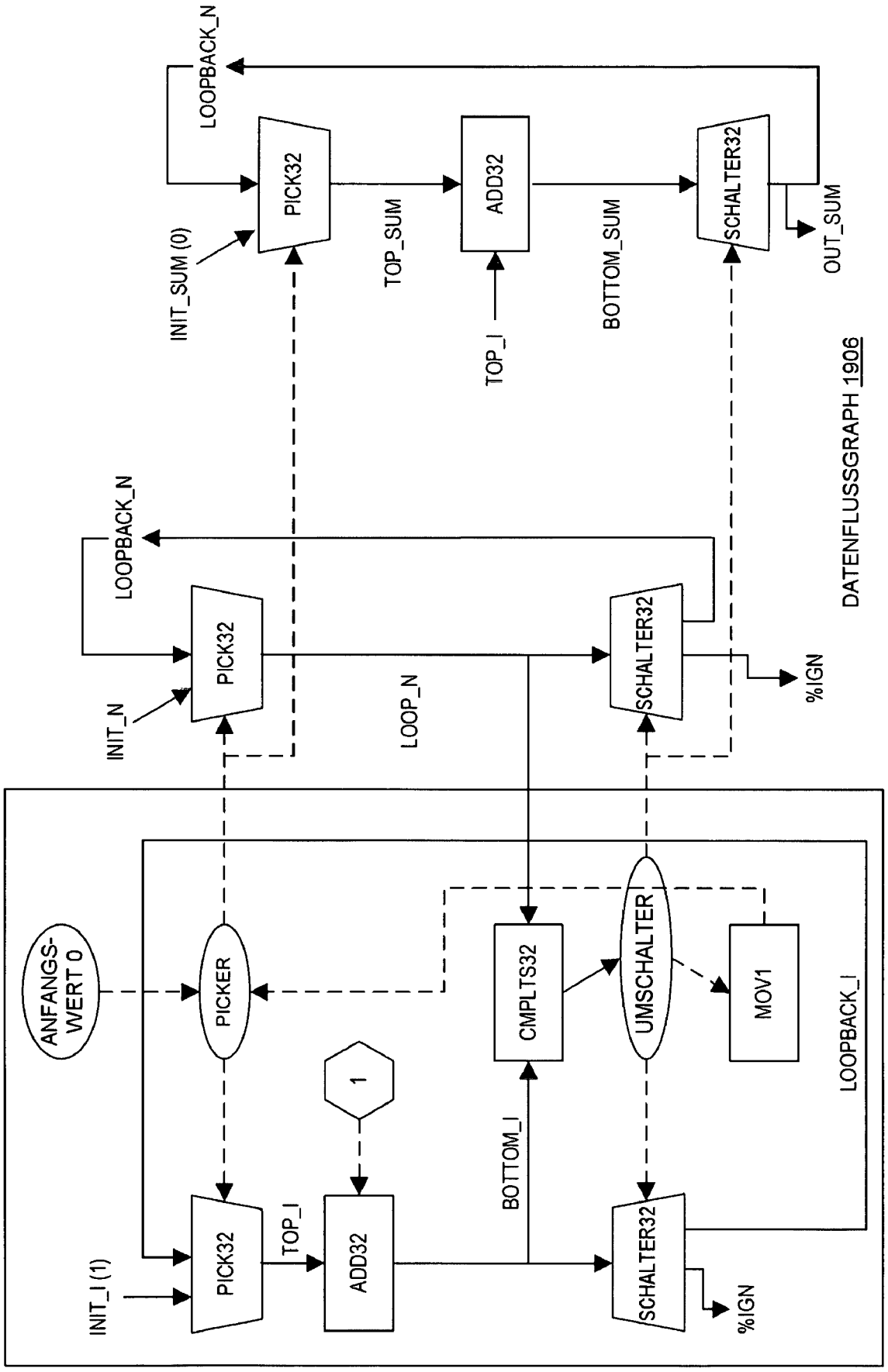

LOOPBACK_N
PICK32
INIT_SUM (0)
TOP_SUM
ADD32
TOP_I
BOTTOM_SUM
SCHALTER32
OUT_SUM
LOOPBACK_N
PICK32
INIT_N
LOOP_N
SCHALTER32
%IGN
DATENFLUSSGRAPH 1906
ANFANGS-WERT 0
PICKER
CMPLTS32
UMSCHALTER
MOV1
LOOPBACK_I
1
BOTTOM_I
PICK32
INIT_I (1)
TOP_I
ADD32
SCHALTER32
%IGN

FIG. 19C

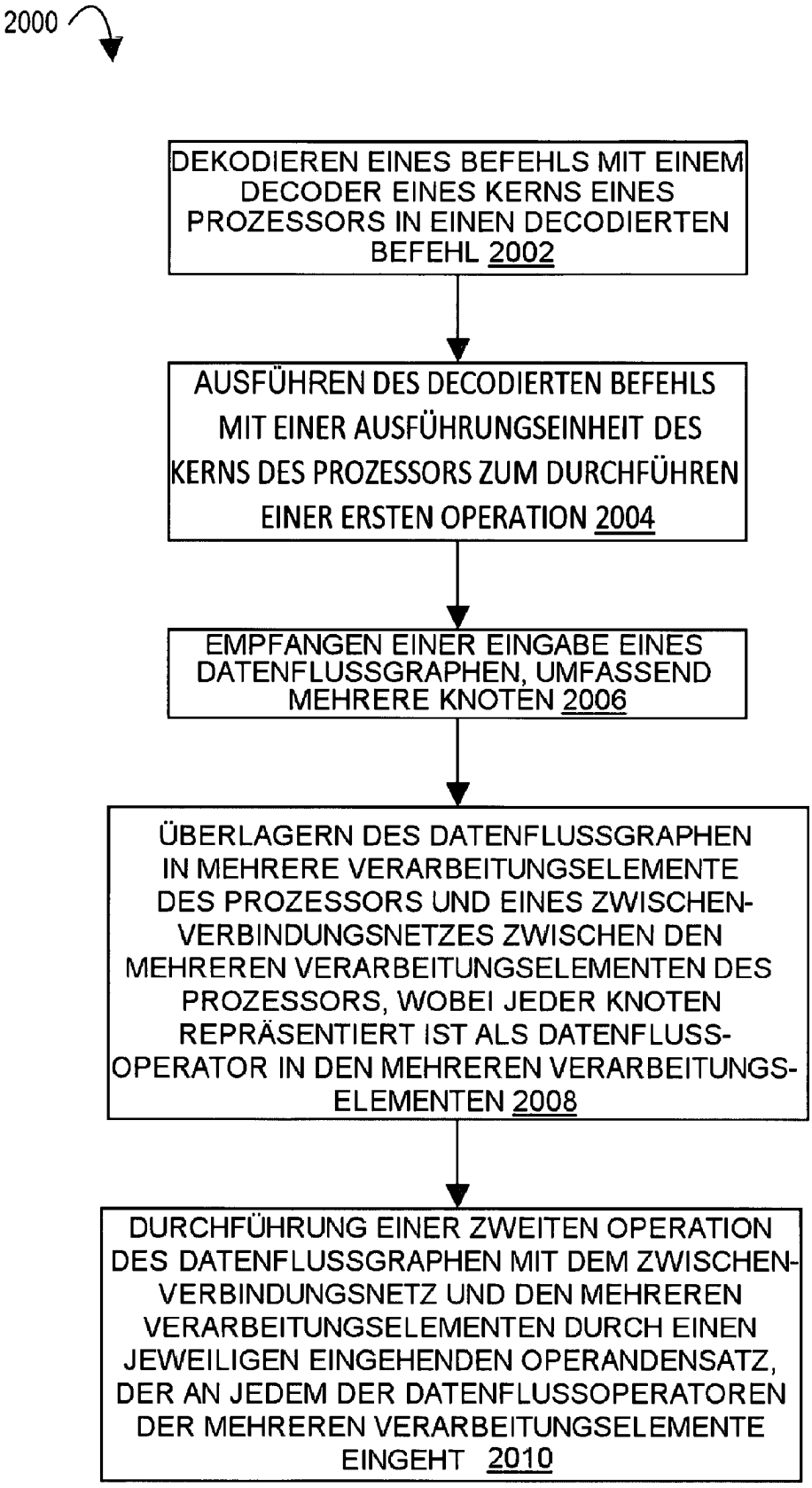

2000
DEKODIEREN EINES BEFEHLS MIT EINEM DECODER EINES KERNS EINES PROZESSORS IN EINEN DECODIERTEN BEFEHL 2002
AUSFÜHREN DES DECODIERTEN BEFEHLS MIT EINER AUSFÜHRUNGSEINHEIT DES KERNS DES PROZESSORS ZUM DURCHFÜHREN EINER ERSTEN OPERATION 2004
EMPFANGEN EINER EINGABE EINES DATENFLUSSGRAPHEN, UMFASSEND MEHRERE KNOTEN 2006
ÜBERLAGERN DES DATENFLUSSGRAPHEN IN MEHRERE VERARBEITUNGSELEMENTE DES PROZESSORS UND EINES ZWISCHEN-VERBINDUNGSNETZES ZWISCHEN DEN MEHREREN VERARBEITUNGSELEMENTEN DES PROZESSORS, WOBEI JEDER KNOTEN REPRÄSENTIERT IST ALS DATENFLUSS-OPERATOR IN DEN MEHREREN VERARBEITUNGS-ELEMENTEN 2008
DURCHFÜHRUNG EINER ZWEITEN OPERATION DES DATENFLUSSGRAPHEN MIT DEM ZWISCHEN-VERBINDUNGSNETZ UND DEN MEHREREN VERARBEITUNGSELEMENTEN DURCH EINEN JEWEILIGEN EINGEHENDEN OPERANDENSATZ, DER AN JEDEM DER DATENFLUSSOPERATOREN DER MEHREREN VERARBEITUNGSELEMENTE EINGEHT 2010

FIG. 20A

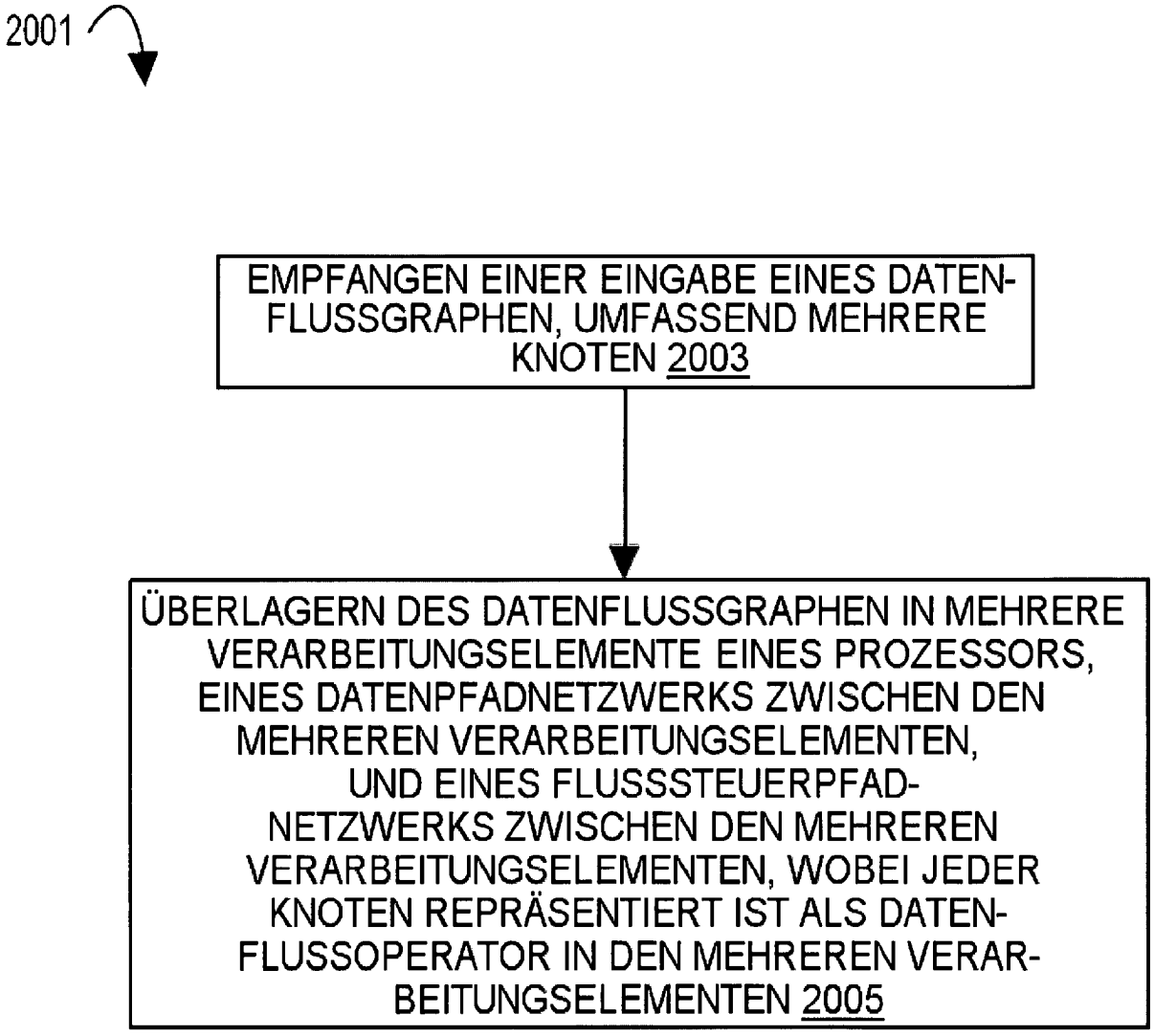

2001
EMPFANGEN EINER EINGABE EINES DATEN-FLUSSGRAPHEN, UMFASSEND MEHRERE KNOTEN 2003
ÜBERLAGERN DES DATENFLUSSGRAPHEN IN MEHRERE VERARBEITUNGSELEMENTE EINES PROZESSORS, EINES DATENPFADNETZWERKS ZWISCHEN DEN MEHREREN VERARBEITUNGSELEMENTEN, UND EINES FLUSSSTEUERPFAD-NETZWERKS ZWISCHEN DEN MEHREREN VERARBEITUNGSELEMENTEN, WOBEI JEDER KNOTEN REPRÄSENTIERT IST ALS DATEN-FLUSSOPERATOR IN DEN MEHREREN VERAR-BEITUNGSELEMENTEN 2005

FIG. 20B

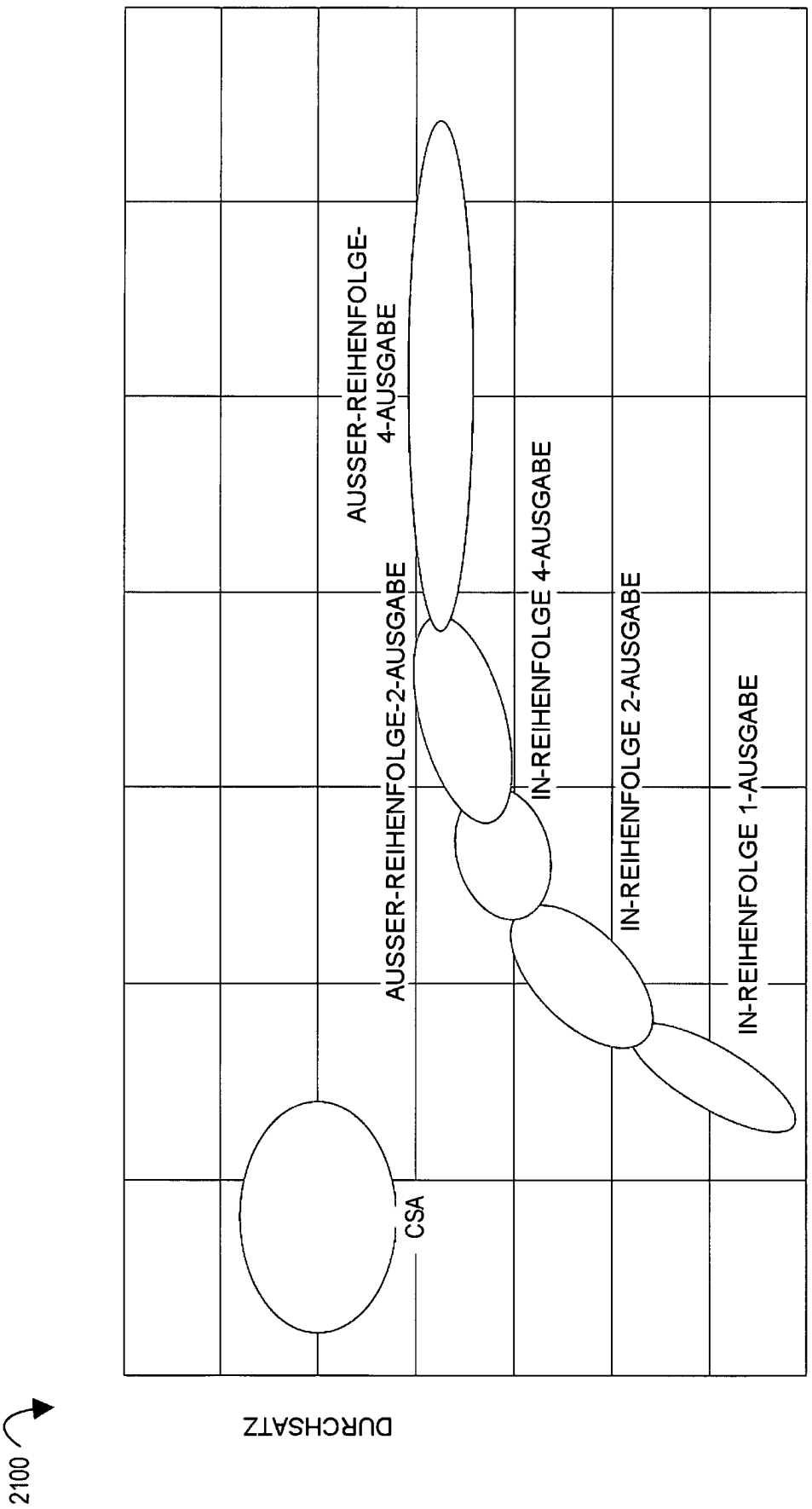

2100
DURCHSATZ
ENERGIE PRO OPERATION
CSA
AUSSER-REIHENFOLGE-2-AUSGABE
AUSSER-REIHENFOLGE-4-AUSGABE
IN-REIHENFOLGE 1-AUSGABE
IN-REIHENFOLGE 2-AUSGABE
IN-REIHENFOLGE 4-AUSGABE

FIG. 21

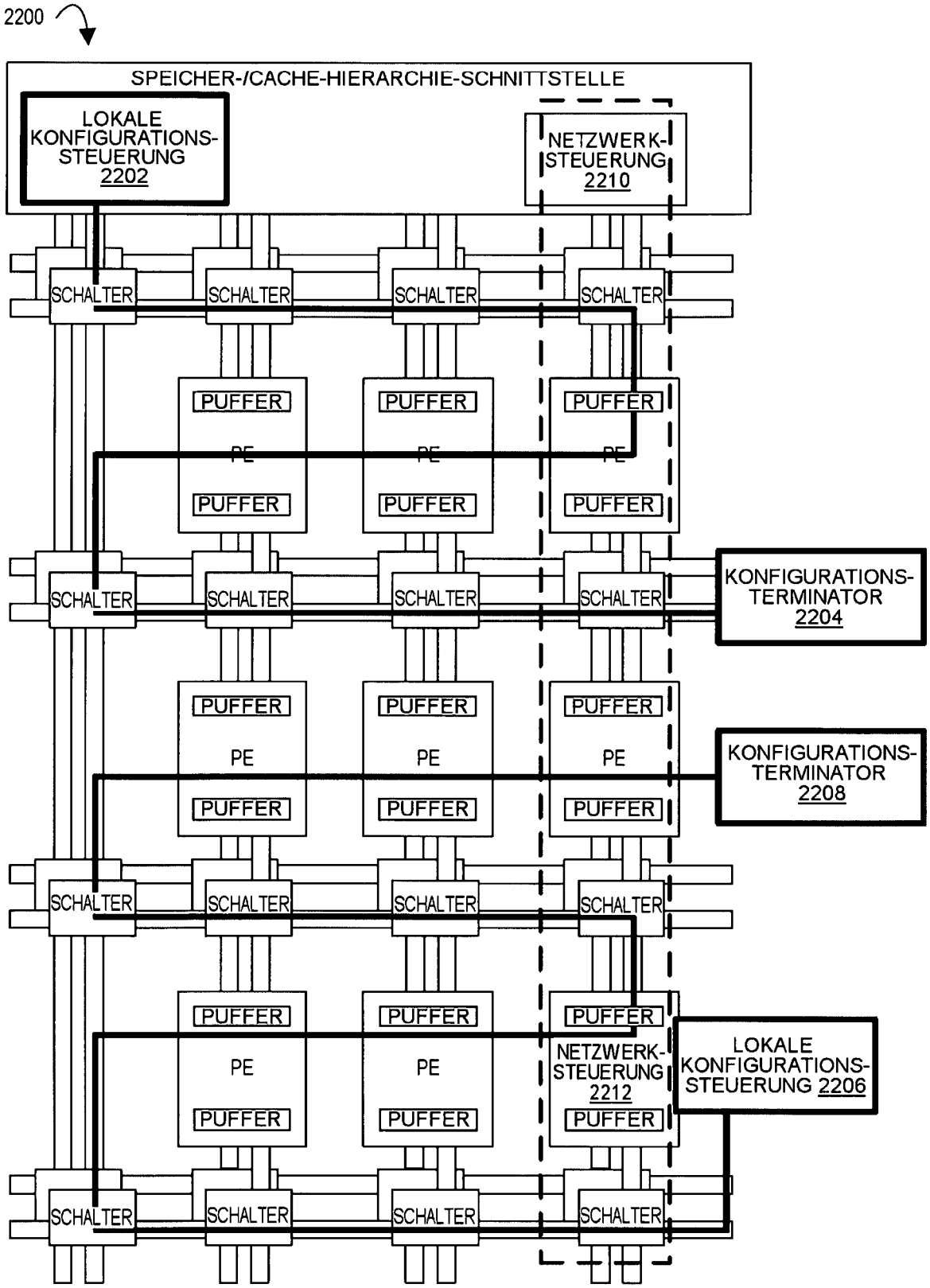

2200
SPEICHER-/CACHE-HIERARCHIE-SCHNITTSTELLE
LOKALE KONFIGURATIONS-STEUERUNG 2202
NETZWERK-STEUERUNG 2210
SCHALTER
PUFFER
PE
KONFIGURATIONS-TERMINATOR 2204
KONFIGURATIONS-TERMINATOR 2208
NETZWERK-STEUERUNG 2212
LOKALE KONFIGURATIONS-STEUERUNG 2206

FIG. 22

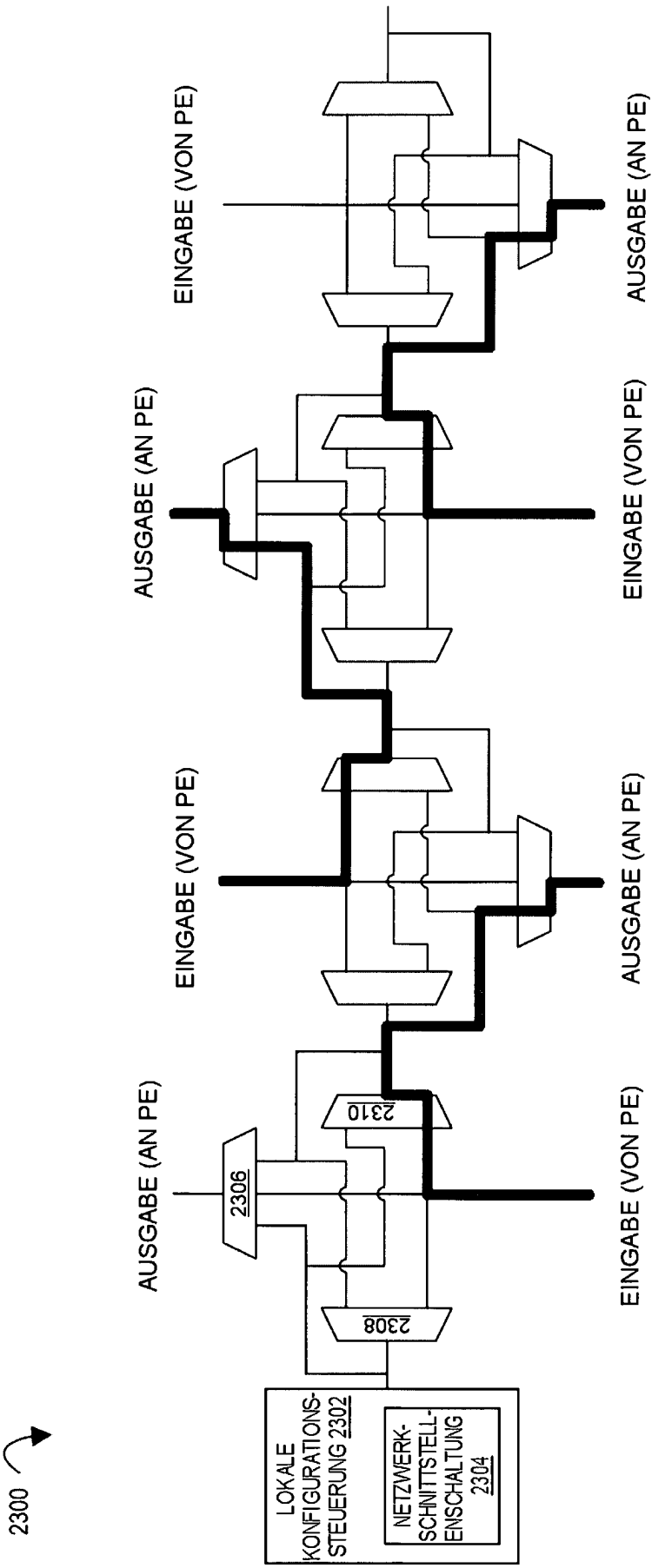

2300
LOKALE KONFIGURATIONS-STEUERUNG 2302
NETZWERK-SCHNITTSTELL-ENSCHALTUNG 2304
2308
2310
2306
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)

FIG. 23A

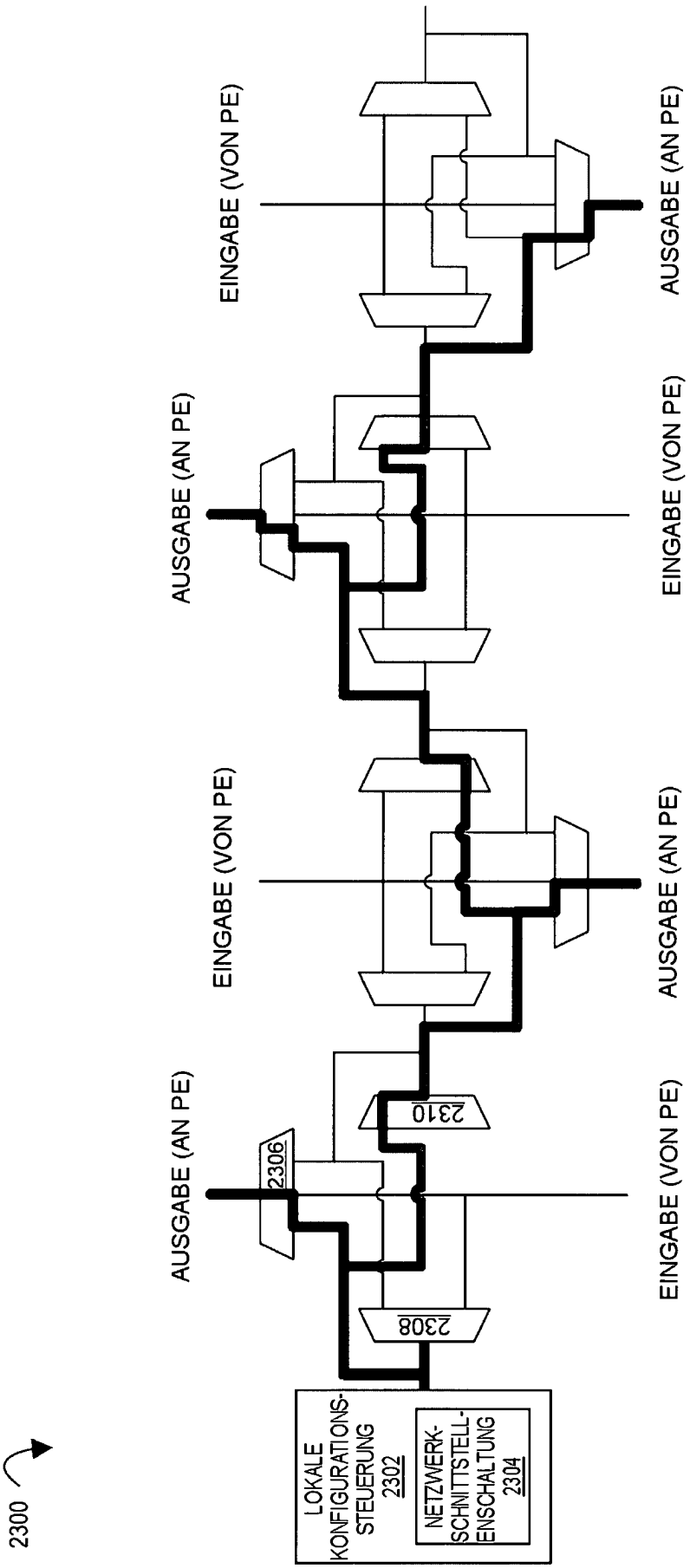

2300
LOKALE KONFIGURATIONS-STEUERUNG 2302
NETZWERK-SCHNITTSTELL-ENSCHALTUNG 2304
2306
2308
2310
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)

FIG. 23B

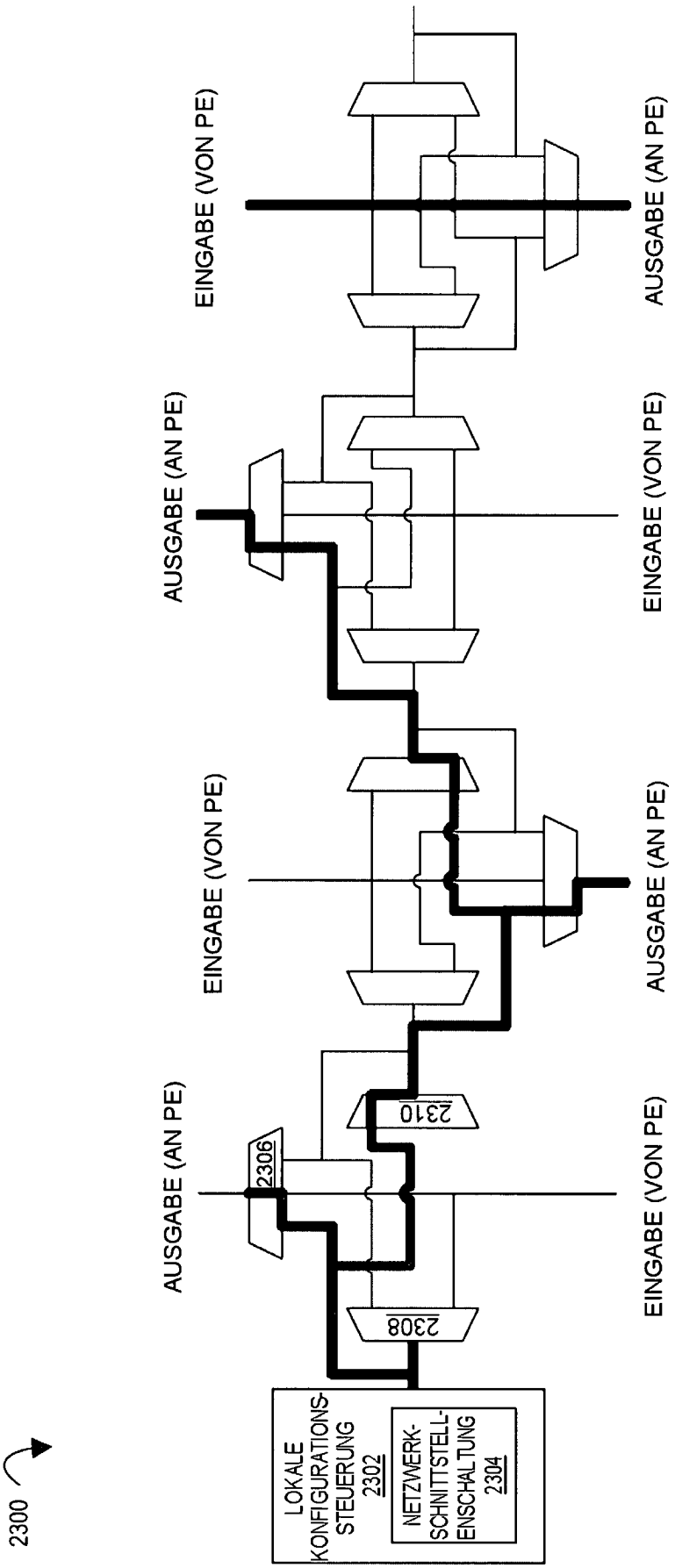

2300
LOKALE KONFIGURATIONS-STEUERUNG 2302
NETZWERK-SCHNITTSTELL-ENSCHALTUNG 2304
2308
2310
2306
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)

FIG. 23C

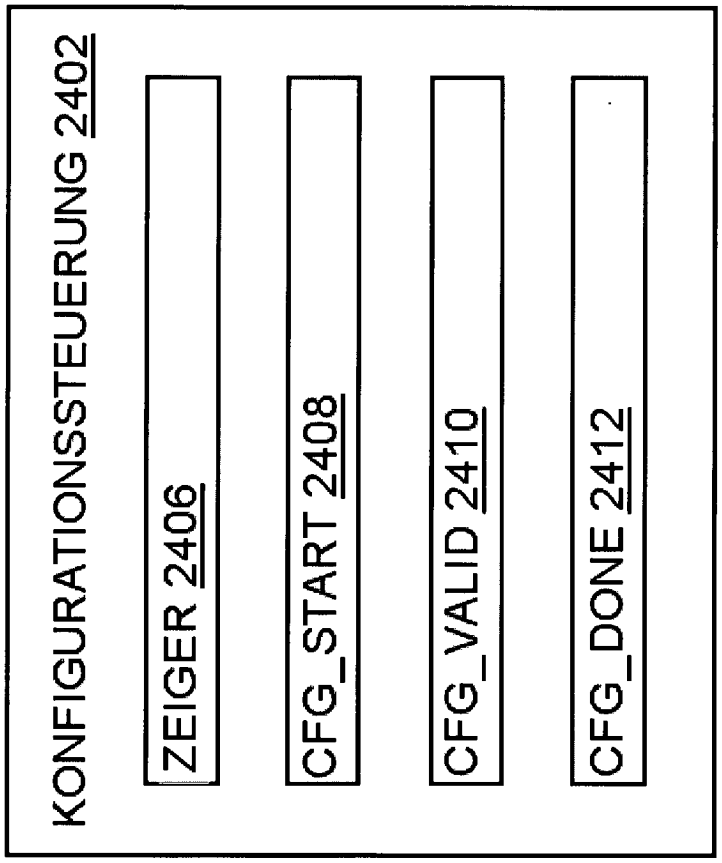

KONFIGURATIONSSTEUERUNG 2402
ZEIGER 2406
CFG_START 2408
CFG_VALID 2410
CFG_DONE 2412

FIG. 24

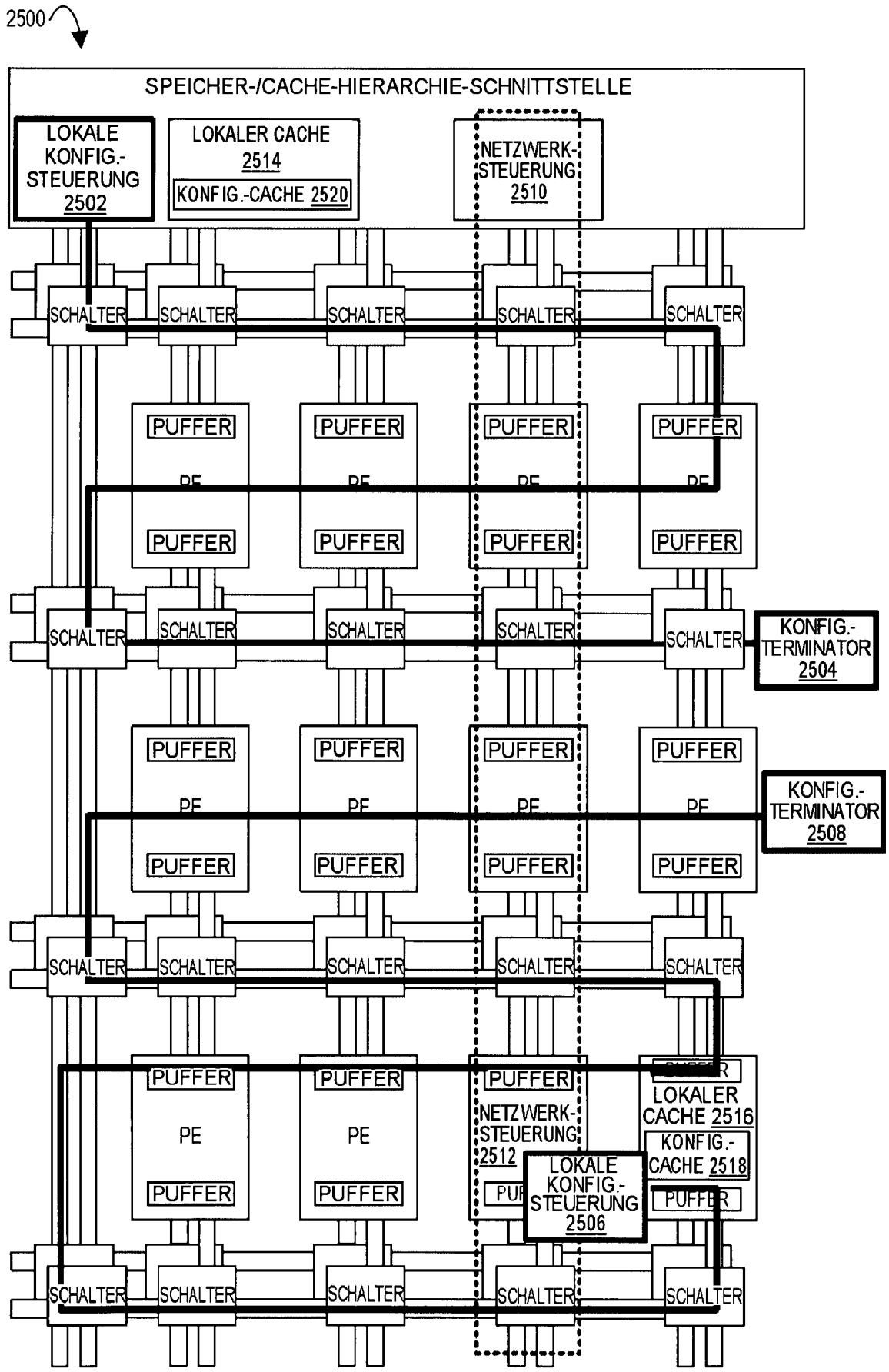

2500
SPEICHER-/CACHE-HIERARCHIE-SCHNITTSTELLE
LOKALE KONFIG.-STEUERUNG 2502
LOKALER CACHE 2514
KONFIG.-CACHE 2520
NETZWERK-STEUERUNG 2510
SCHALTER
PUFFER
PE
KONFIG.-TERMINATOR 2504
KONFIG.-TERMINATOR 2508
NETZWERK-STEUERUNG 2512
LOKALER CACHE 2516
KONFIG.-CACHE 2518
LOKALE KONFIG.-STEUERUNG 2506

FIG. 25

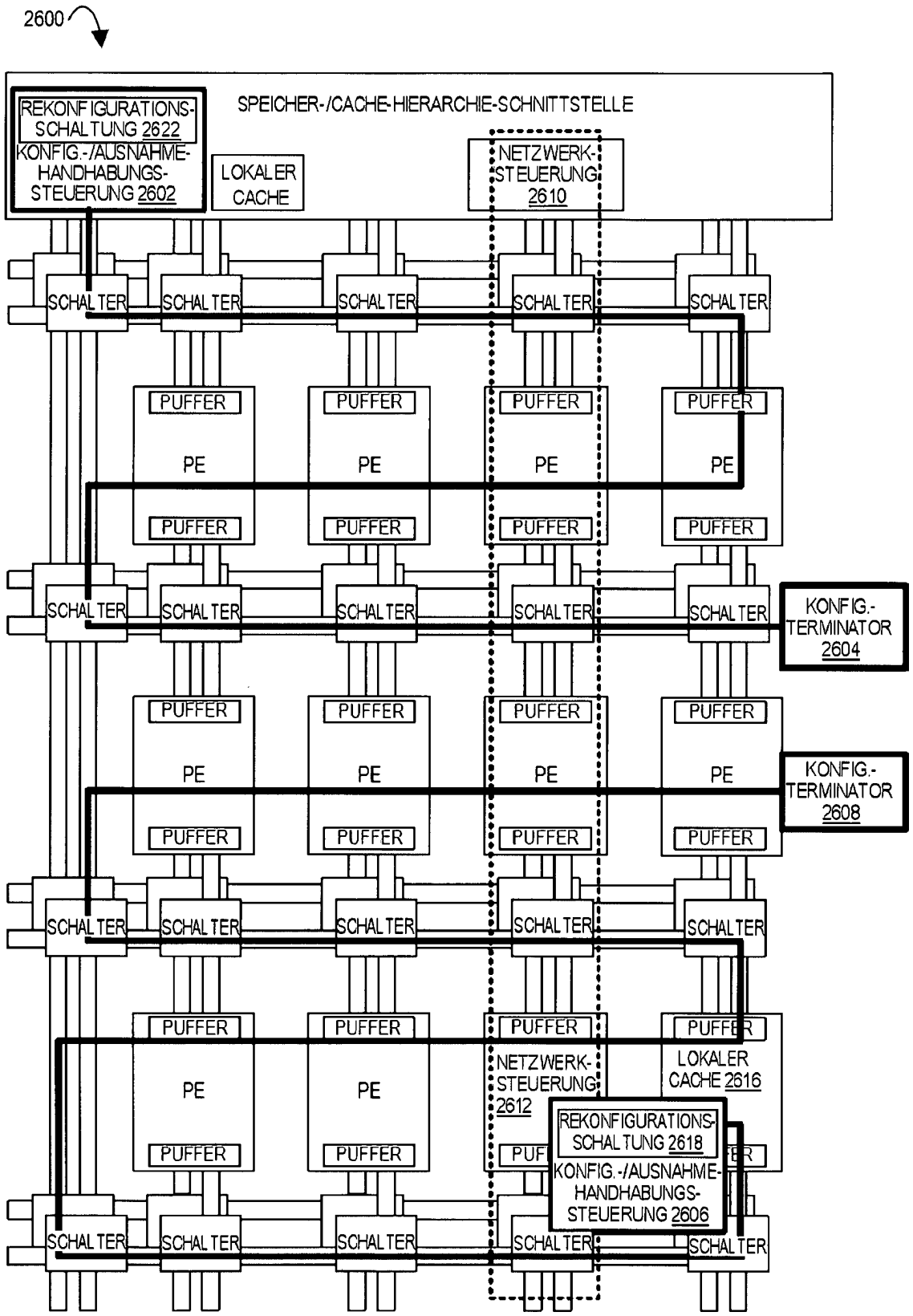

2600
REKONFIGURATIONS-SCHALTUNG 2622
KONFIG.-/AUSNAHME-HANDHABUNGS-STEUERUNG 2602
LOKALER CACHE
SPEICHER-/CACHE-HIERARCHIE-SCHNITTSTELLE
NETZWERK-STEUERUNG 2610
SCHALTER
PUFFER
PE
KONFIG.-TERMINATOR 2604
KONFIG.-TERMINATOR 2608
NETZWERK-STEUERUNG 2612
LOKALER CACHE 2616
REKONFIGURATIONS-SCHALTUNG 2618
KONFIG.-/AUSNAHME-HANDHABUNGS-STEUERUNG 2606

FIG. 26

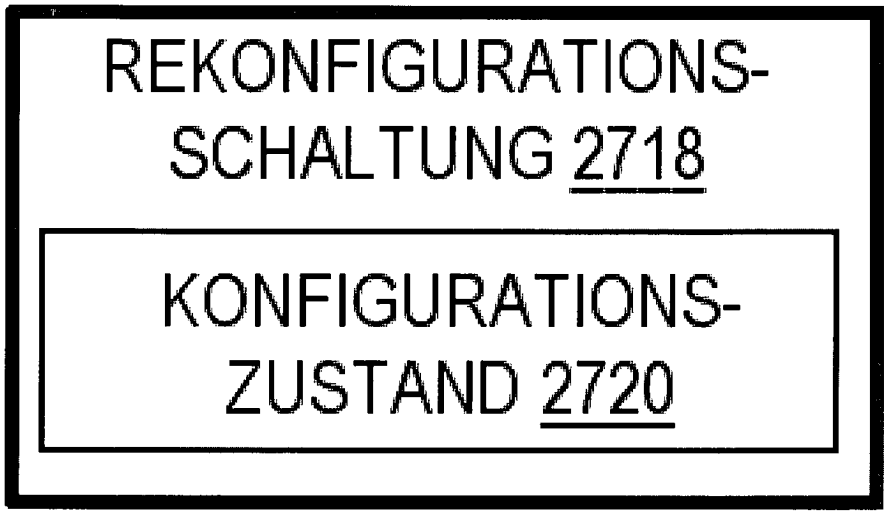

REKONFIGURATIONS-
SCHALTUNG 2718
KONFIGURATIONS-
ZUSTAND 2720

FIG. 27

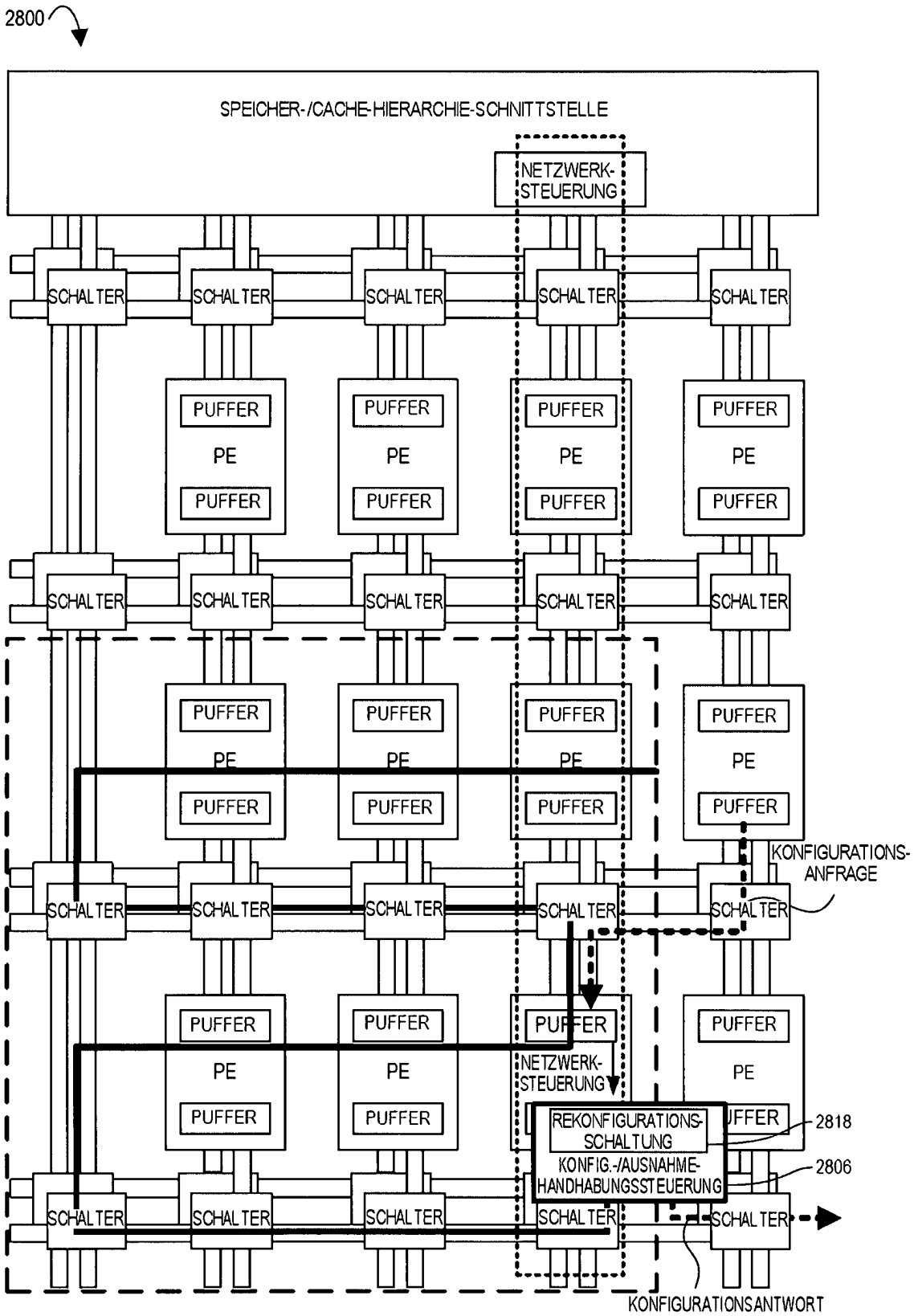

2800
SPEICHER-/CACHE-HIERARCHIE-SCHNITTSTELLE
NETZWERK-STEUERUNG
SCHALTER
PUFFER
PE
KONFIGURATIONS-ANFRAGE
NETZWERK-STEUERUNG
REKONFIGURATIONS-SCHALTUNG
2818
KONFIG.-/AUSNAHME-HANDHABUNGSSTEUERUNG
2806
KONFIGURATIONSANTWORT

FIG. 28

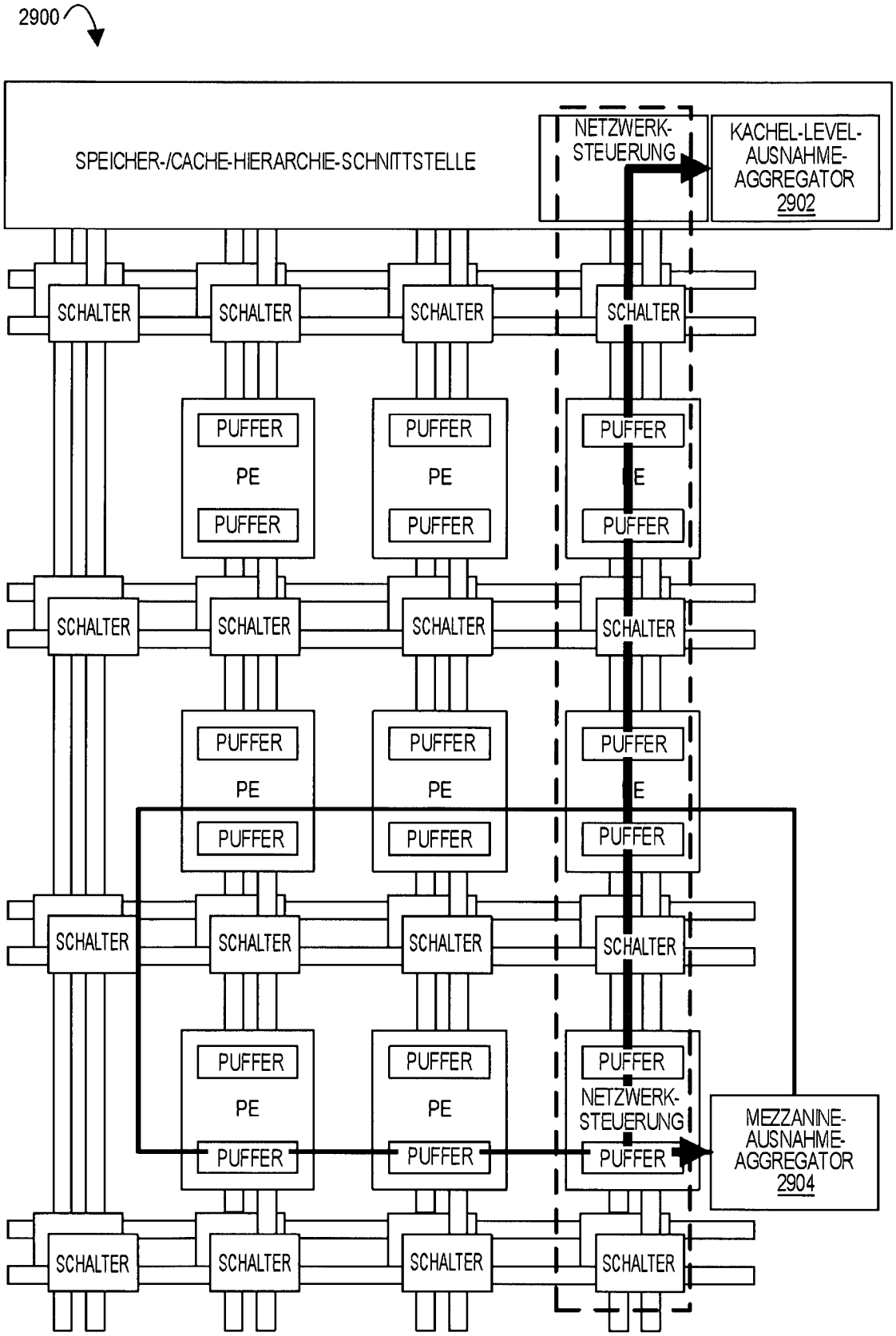
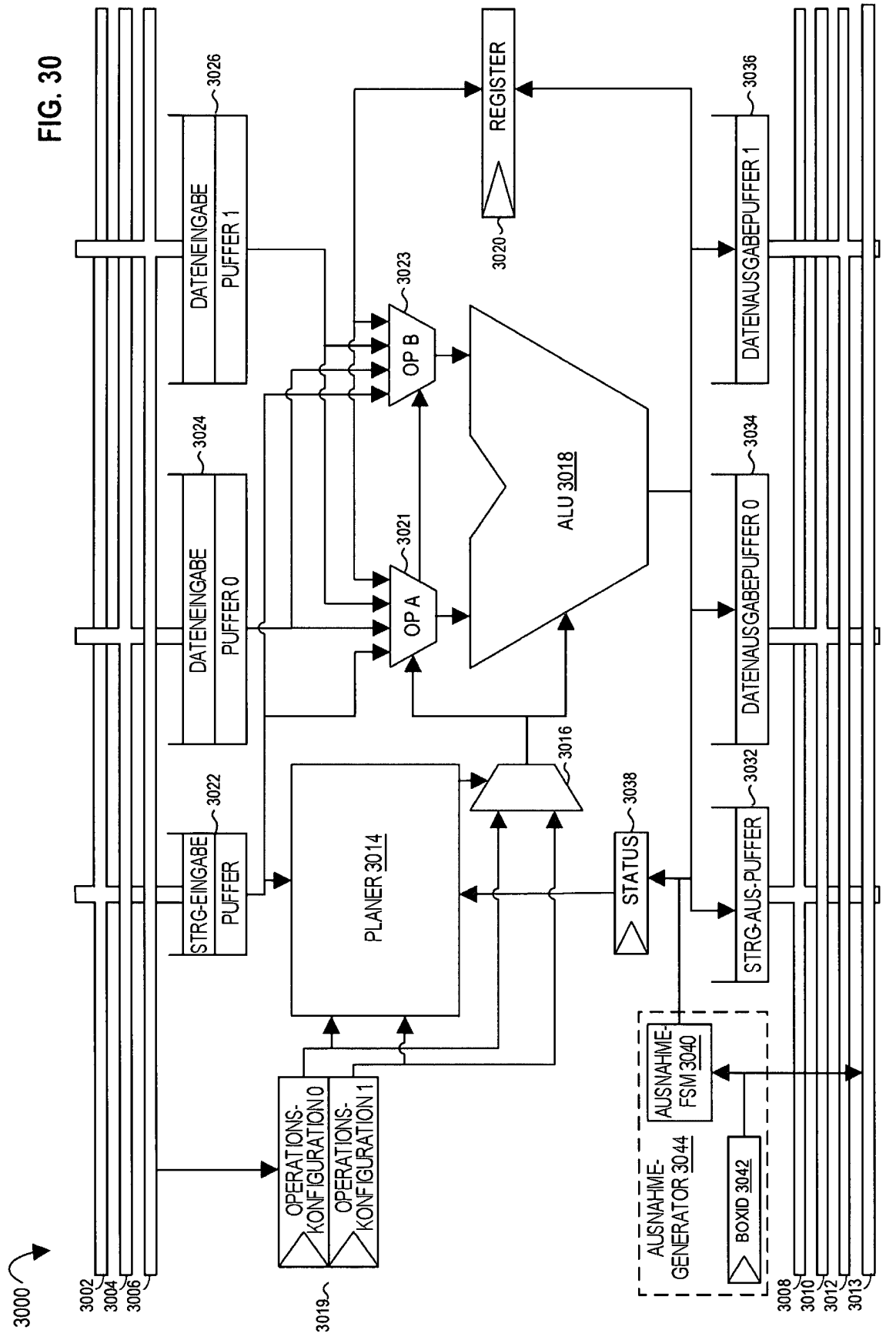

2900
SPEICHER-/CACHE-HIERARCHIE-SCHNITTSTELLE
NETZWERK-STEUERUNG
KACHEL-LEVEL-AUSNAHME-AGGREGATOR 2902
SCHALTER
PUFFER
PE
NETZWERK-STEUERUNG
MEZZANINE-AUSNAHME-AGGREGATOR 2904

FIG. 29

FIG. 30


3000
3002
3004
3006
3008
3010
3012
3013
3019
DATENEINGABE PUFFER 1
3026
DATENEINGABE PUFFER 0
3024
STRG-EINGABE PUFFER
3022
OPERATIONS-KONFIGURATION 0
OPERATIONS-KONFIGURATION 1
PLANER 3014
3016
OP A
3021
OP B
3023
ALU 3018
REGISTER
3020
STATUS
3038
DATENAUSGABEPUFFER 1
3036
DATENAUSGABEPUFFER 0
3034
STRG-AUS-PUFFER
3032
AUSNAHME-GENERATOR 3044
AUSNAHME-FSM 3040
BOXID 3042

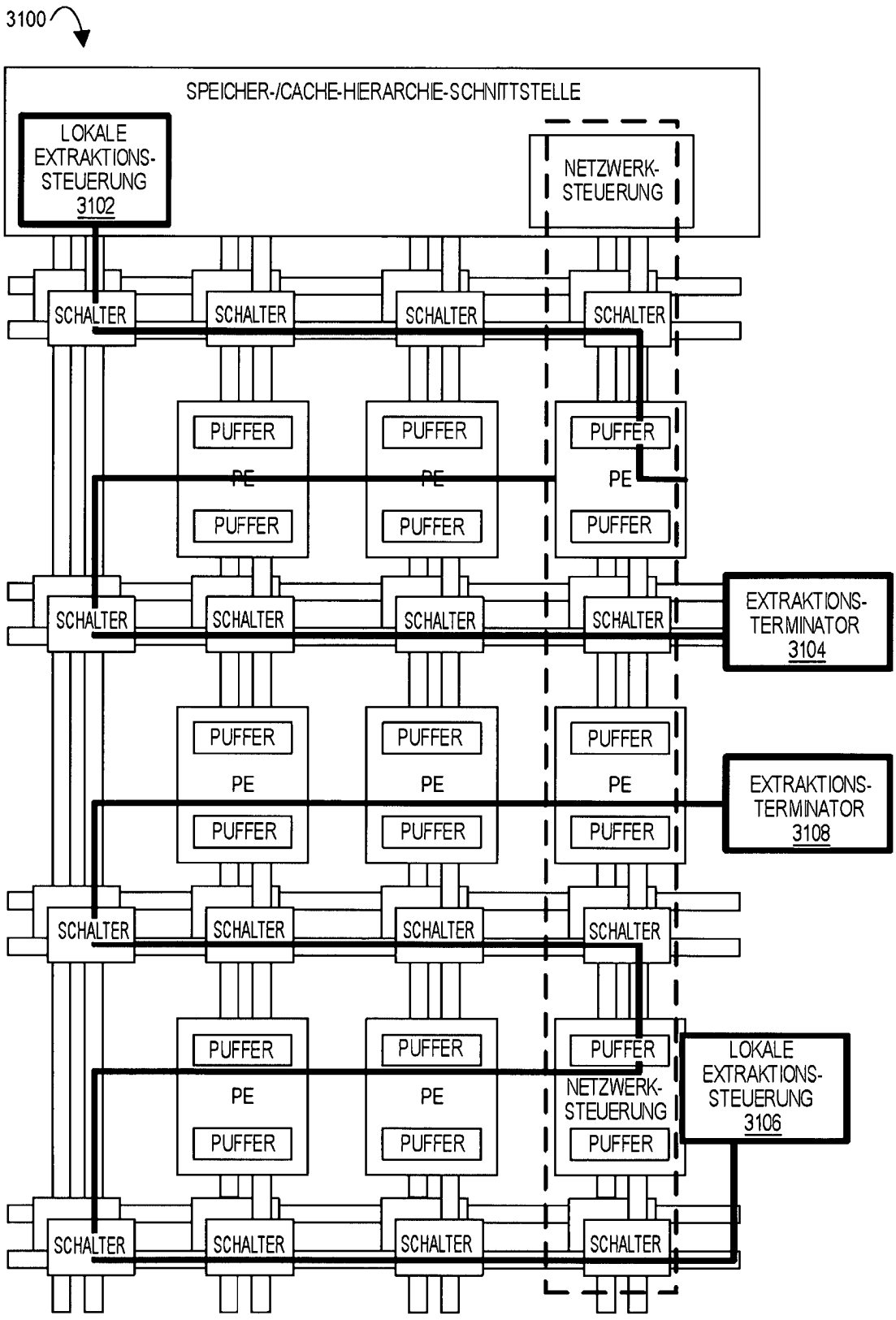

3100
SPEICHER-/CACHE-HIERARCHIE-SCHNITTSTELLE
LOKALE EXTRAKTIONS-STEUERUNG 3102
NETZWERK-STEUERUNG
SCHALTER
PUFFER
PE
EXTRAKTIONS-TERMINATOR 3104
EXTRAKTIONS-TERMINATOR 3108
LOKALE EXTRAKTIONS-STEUERUNG 3106
NETZWERK-STEUERUNG

FIG. 31

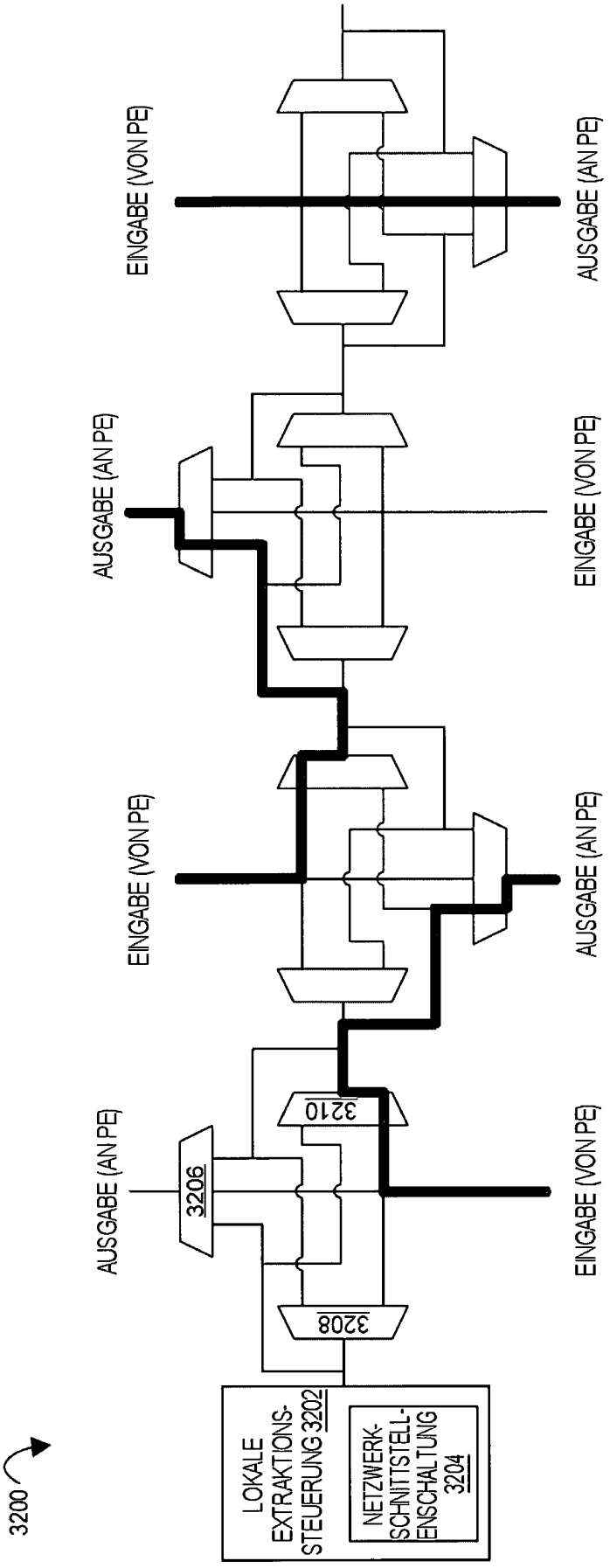

3200
LOKALE EXTRAKTIONS-STEUERUNG 3202
NETZWERK-SCHNITTSTELL-ENSCHALTUNG 3204
3208
3210
3206
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)

FIG. 32A

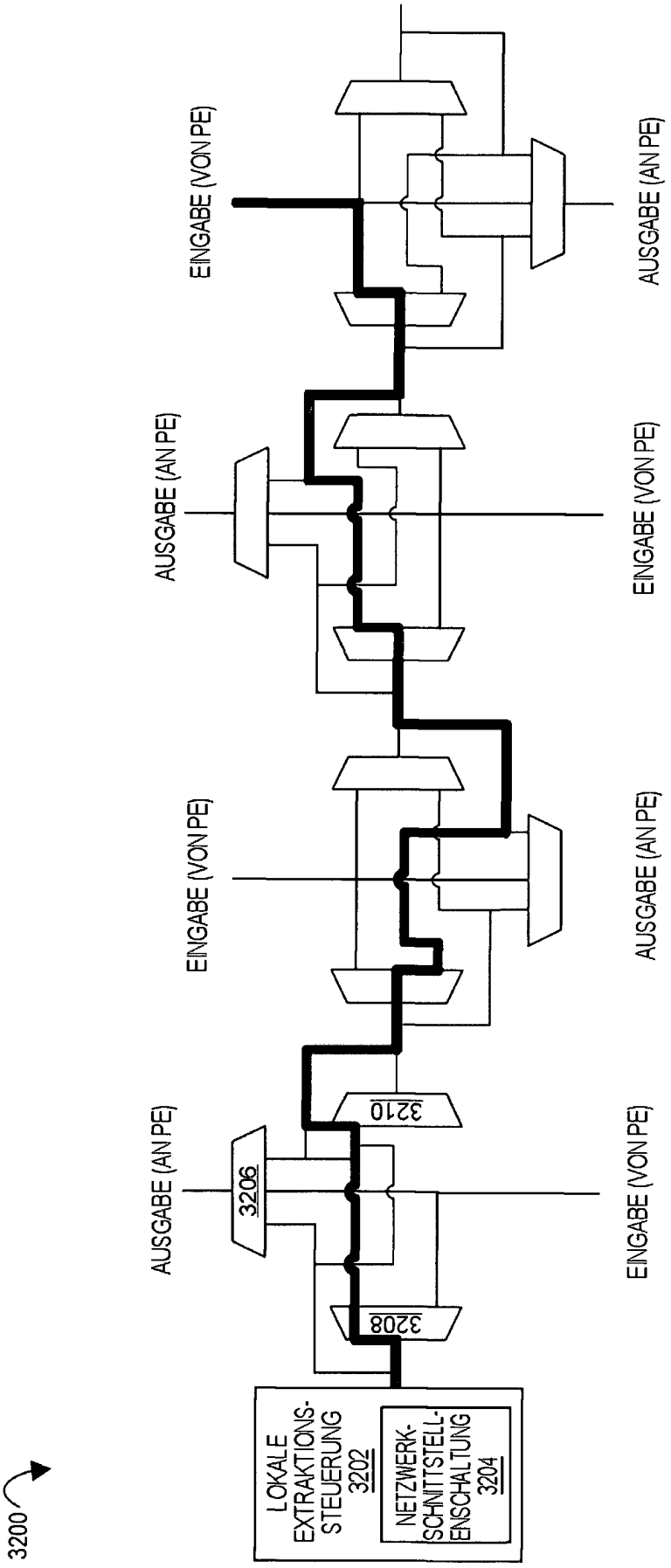

3200
LOKALE EXTRAKTIONS-STEUERUNG 3202
NETZWERK-SCHNITTSTELL-ENSCHALTUNG 3204
3208
3210
3206
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)

FIG. 32B

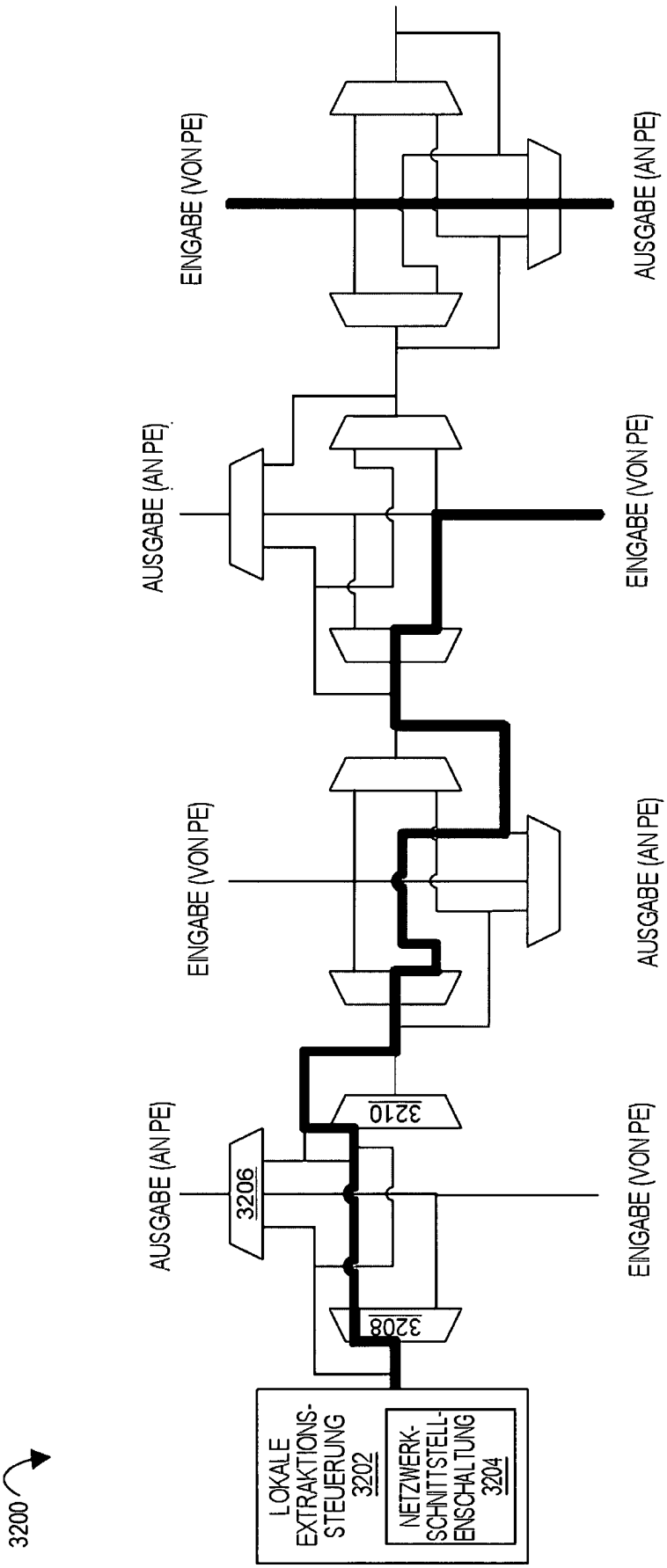

3200
LOKALE EXTRAKTIONS-STEUERUNG 3202
NETZWERK-SCHNITTSTELL-ENSCHALTUNG 3204
3208
3210
3206
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)
AUSGABE (AN PE)
EINGABE (VON PE)
EINGABE (VON PE)
AUSGABE (AN PE)

FIG. 32C


EXTRAKTIONSSTEUERUNG 3302
ZEIGER 3304
LEC_STROBE 3310
EFE_COMPLETE 3312
LEC_EXTRACT 3306
LEC_START 3308

FIG. 33

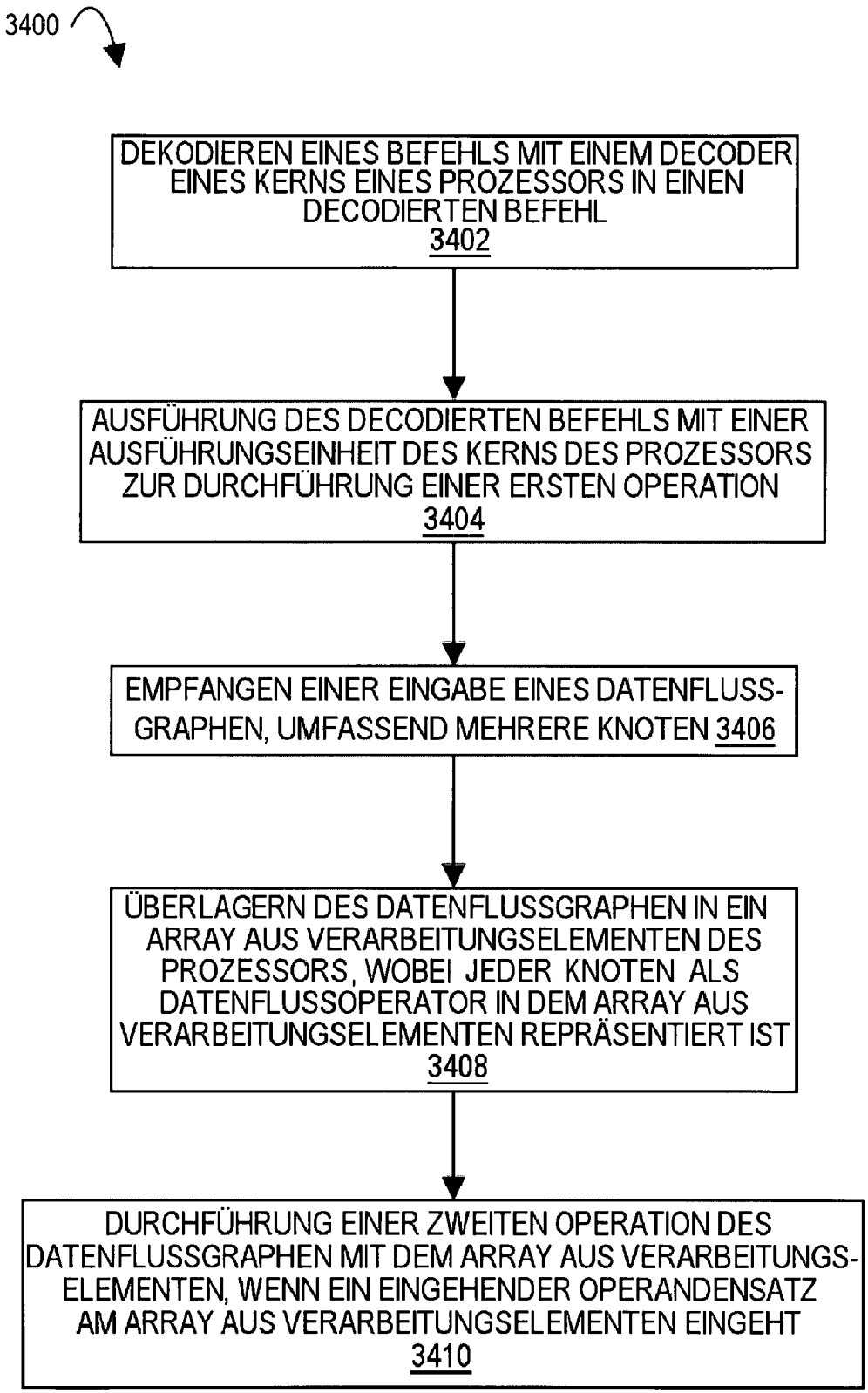

3400
DEKODIEREN EINES BEFEHLS MIT EINEM DECODER EINES KERNS EINES PROZESSORS IN EINEN DECODIERTEN BEFEHL
3402
AUSFÜHRUNG DES DECODIERTEN BEFEHLS MIT EINER AUSFÜHRUNGSEINHEIT DES KERNS DES PROZESSORS ZUR DURCHFÜHRUNG EINER ERSTEN OPERATION
3404
EMPFANGEN EINER EINGABE EINES DATENFLUSS-GRAPHEN, UMFASSEND MEHRERE KNOTEN 3406
ÜBERLAGERN DES DATENFLUSSGRAPHEN IN EIN ARRAY AUS VERARBEITUNGSELEMENTEN DES PROZESSORS, WOBEI JEDER KNOTEN ALS DATENFLUSSOPERATOR IN DEM ARRAY AUS VERARBEITUNGSELEMENTEN REPRÄSENTIERT IST
3408
DURCHFÜHRUNG EINER ZWEITEN OPERATION DES DATENFLUSSGRAPHEN MIT DEM ARRAY AUS VERARBEITUNGS-ELEMENTEN, WENN EIN EINGEHENDER OPERANDENSATZ AM ARRAY AUS VERARBEITUNGSELEMENTEN EINGEHT
3410

FIG. 34

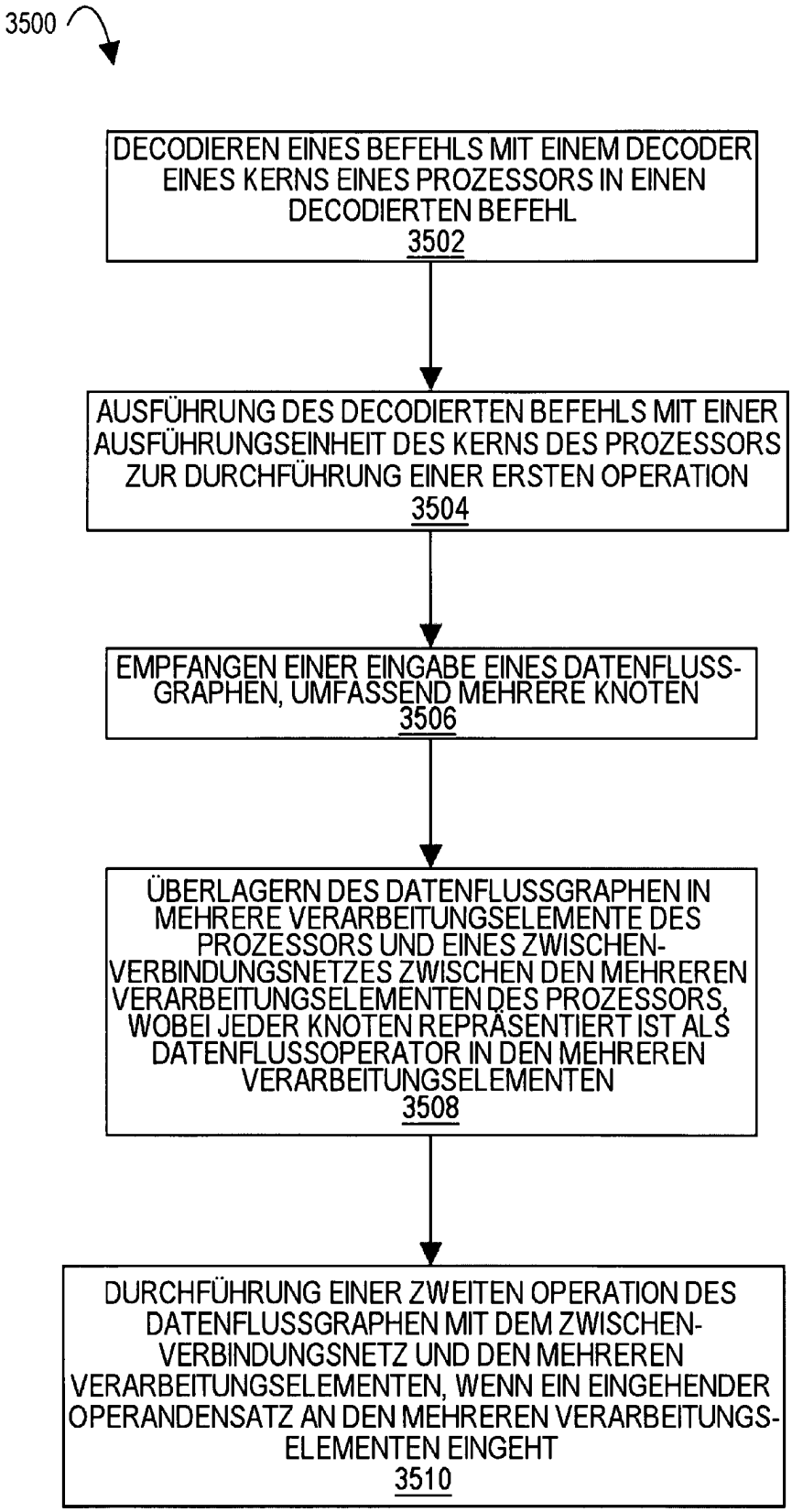

3500
DECODIEREN EINES BEFEHLS MIT EINEM DECODER EINES KERNS EINES PROZESSORS IN EINEN DECODIERTEN BEFEHL
3502
AUSFÜHRUNG DES DECODIERTEN BEFEHLS MIT EINER AUSFÜHRUNGSEINHEIT DES KERNS DES PROZESSORS ZUR DURCHFÜHRUNG EINER ERSTEN OPERATION
3504
EMPFANGEN EINER EINGABE EINES DATENFLUSS-GRAPHEN, UMFASSEND MEHRERE KNOTEN
3506
ÜBERLAGERN DES DATENFLUSSGRAPHEN IN MEHRERE VERARBEITUNGSELEMENTE DES PROZESSORS UND EINES ZWISCHEN-VERBINDUNGSNETZES ZWISCHEN DEN MEHREREN VERARBEITUNGSELEMENTEN DES PROZESSORS, WOBEI JEDER KNOTEN REPRÄSENTIERT IST ALS DATENFLUSSOPERATOR IN DEN MEHREREN VERARBEITUNGSELEMENTEN
3508
DURCHFÜHRUNG EINER ZWEITEN OPERATION DES DATENFLUSSGRAPHEN MIT DEM ZWISCHEN-VERBINDUNGSNETZ UND DEN MEHREREN VERARBEITUNGSELEMENTEN, WENN EIN EINGEHENDER OPERANDENSATZ AN DEN MEHREREN VERARBEITUNGS-ELEMENTEN EINGEHT
3510

FIG. 35

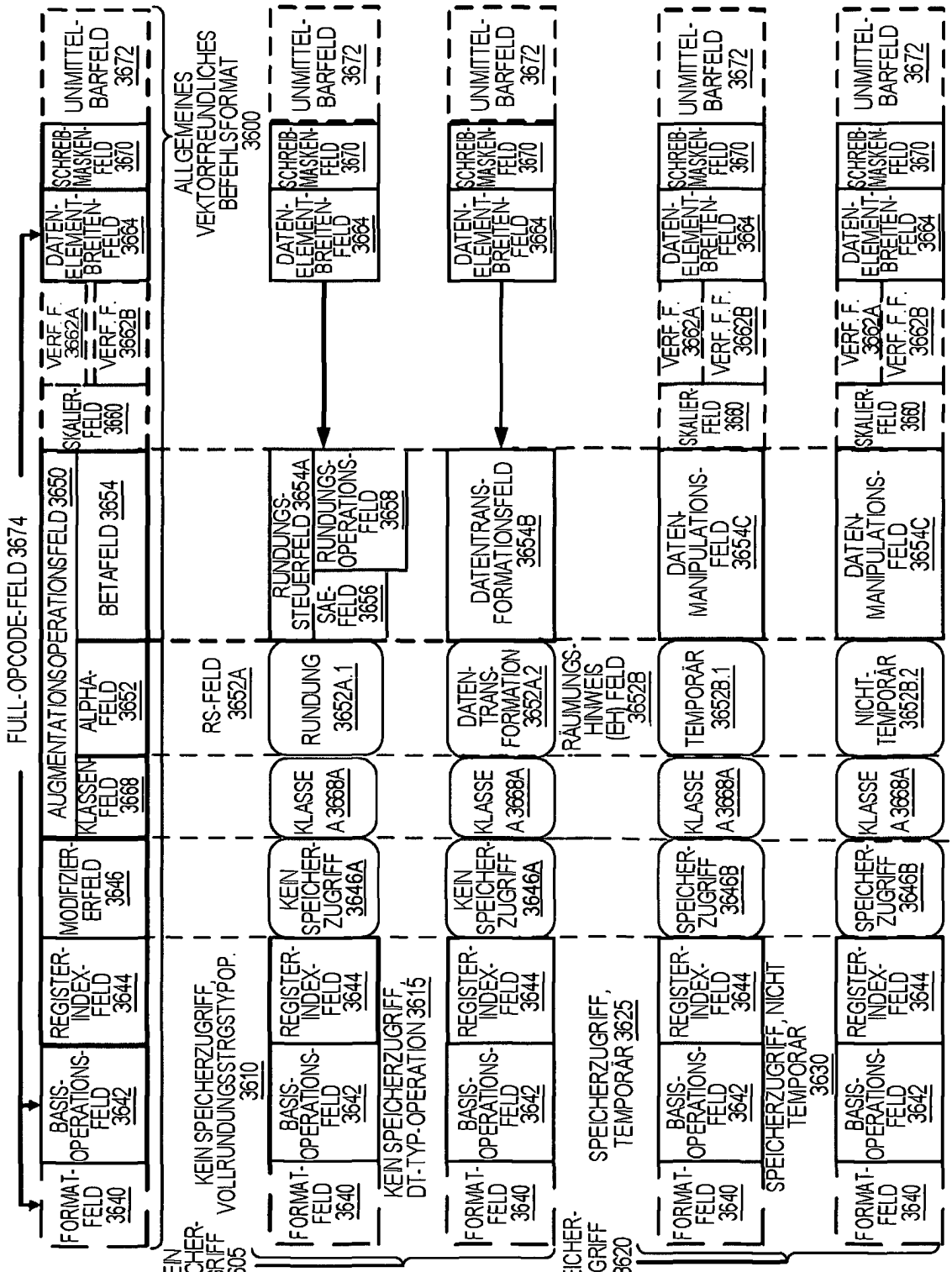

FULL-OPCODE-FELD 3674
ALLGEMEINES VEKTORFREUNDLICHES BEFEHLSFORMAT 3600
FORMAT-FELD 3640
BASIS-OPERATIONS-FELD 3642
REGISTER-INDEX-FELD 3644
MODIFIZIER-ERFELD 3646
AUGMENTATIONSOPERATIONSFELD 3650
KLASSEN-FELD 3668
ALPHA-FELD 3652
BETAFELD 3654
SKALIER-FELD 3660
VERF. F. 3662A
VERF. F. 3662B
DATEN-ELEMENT-BREITEN-FELD 3664
SCHREIB-MASKEN-FELD 3670
UNMITTEL-BARFELD 3672
KEIN SPEICHERZUGRIFF 3605
KEIN SPEICHERZUGRIFF, VOLLRUNDUNGSSTRGSTYP/OP. 3610
KEIN SPEICHERZUGRIFF DT-TYP-OPERATION 3615
KEIN SPEICHER-ZUGRIFF 3646A
KLASSE A 3668A
RS-FELD 3652A
RUNDUNG 3652A.1
RUNDUNGS-STEUERFELD 3654A
SAE-FELD 3656
RUNDUNGS-OPERATIONS-FELD 3658
DATENTRANS-FORMATIONSFELD 3654B
DATEN-TRANS-FORMATION 3652A.2
SPEICHERZUGRIFF 3620
SPEICHERZUGRIFF, TEMPORÄR 3625
SPEICHER-ZUGRIFF 3646B
RÄUMUNGS-HINWEIS (EH) FELD 3652B
TEMPORÄR 3652B.1
DATEN-MANIPULATIONS-FELD 3654C
SPEICHERZUGRIFF, NICHT TEMPORAR 3630
NICHT-TEMPORÄR 3652B.2

FIG.36A

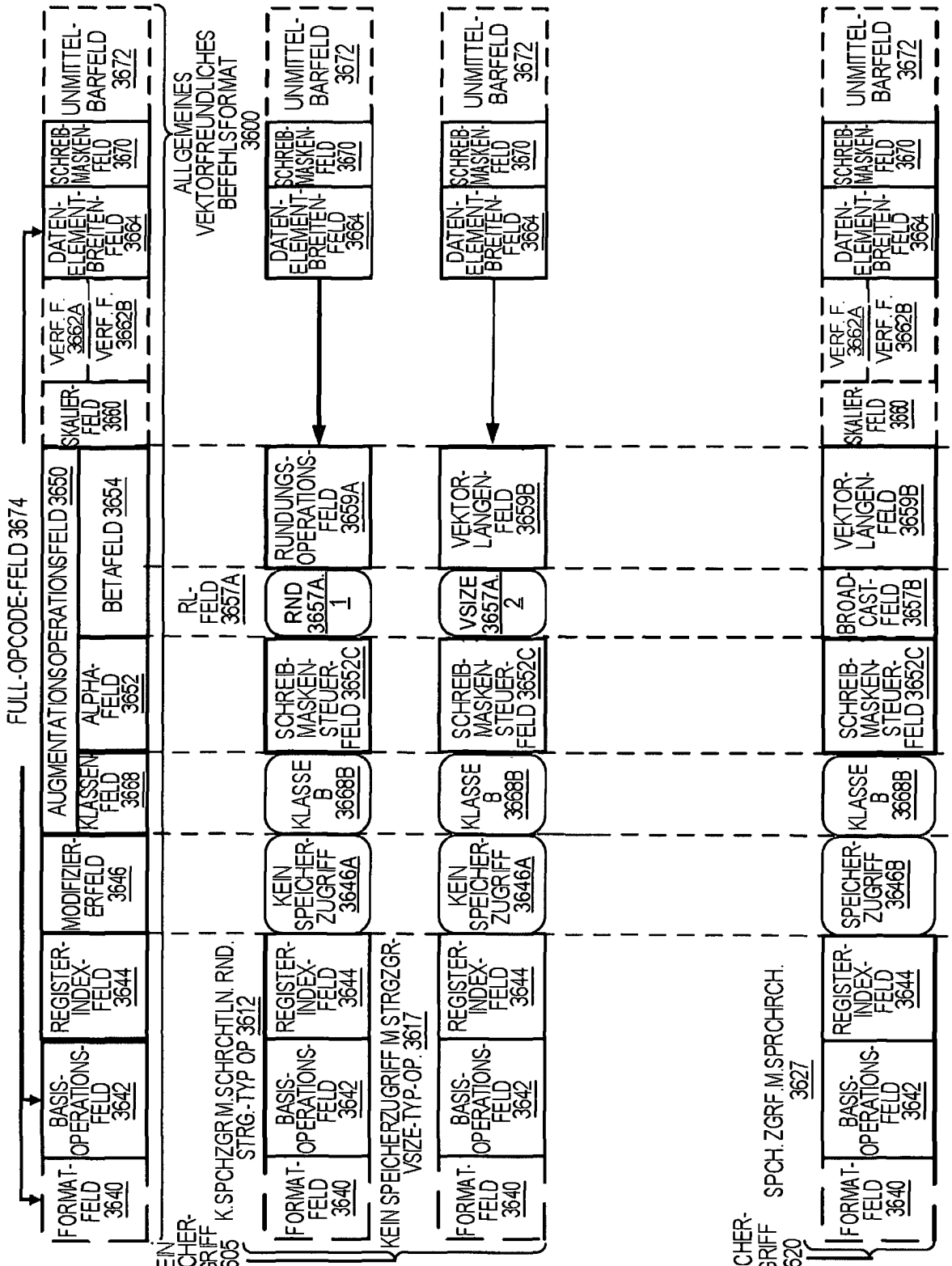
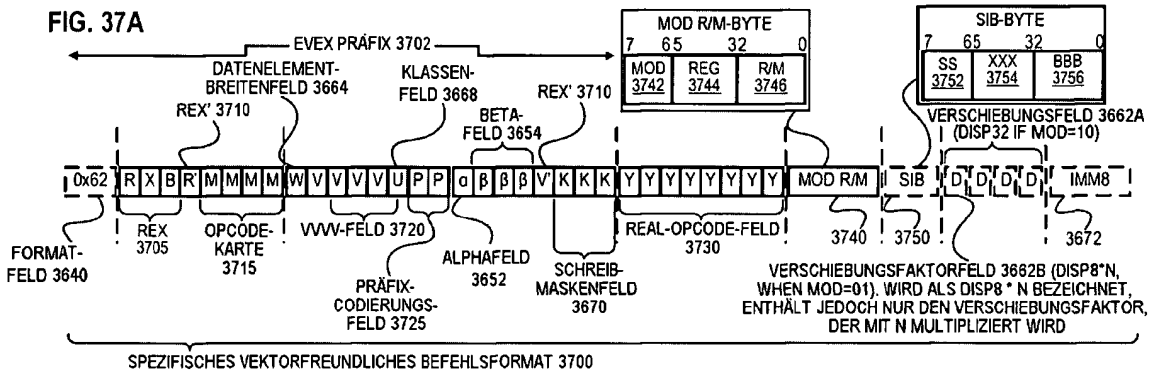

FULL-OPCODE-FELD 3674
FORMAT-FELD 3640
BASIS-OPERATIONS-FELD 3642
REGISTER-INDEX-FELD 3644
MODIFIZIER-ERFELD 3646
AUGMENTATIONSOPERATIONSFELD 3650
KLASSEN-FELD 3668
ALPHA-FELD 3652
BETAFELD 3654
SKALIER-FELD 3660
VERF. F. 3662A
VERF. F. 3662B
DATEN-ELEMENT-BREITEN-FELD 3664
SCHREIB-MASKEN-FELD 3670
UNMITTEL-BARFELD 3672
ALLGEMEINES VEKTORFREUNDLICHES BEFEHLSFORMAT 3600
KEIN SPEICHER-ZUGRIFF 3605
K.SPCHZGR M.SCHRCHTLN. RND. STRG.-TYP OP 3612
KEIN SPEICHERZUGRIFF M STRGZGR-VSIZE-TYP-OP. 3617
SPEICHER-ZUGRIFF 3620
SPCH. ZGRF. M. SPRCHRCH. 3627
KEIN SPEICHER-ZUGRIFF 3646A
SPEICHER-ZUGRIFF 3646B
KLASSE B 3668B
SCHREIB-MASKEN-STEUER-FELD 3652C
RL-FELD 3657A
RND 3657A. 1
VSIZE 3657A. 2
RUNDUNGS-OPERATIONS-FELD 3659A
VEKTOR-LÄNGEN-FELD 3659B
BROAD-CAST-FELD 3657B

FIG. 36B

FIG. 37A


EVEX PRÄFIX 3702
DATENELEMENT-BREITENFELD 3664
KLASSEN-FELD 3668
REX' 3710
BETA-FELD 3654
REX' 3710
MOD R/M-BYTE
7 65 32 0
MOD 3742
REG 3744
R/M 3746
SIB-BYTE
7 65 32 0
SS 3752
XXX 3754
BBB 3756
VERSCHIEBUNGSFELD 3662A (DISP32 IF MOD=10)
0x62
R X B R' M M M M W V V V V U P P α β β β V' K K K Y Y Y Y Y Y Y
MOD R/M
SIB
D D D D
IMM8
FORMAT-FELD 3640
REX 3705
OPCODE-KARTE 3715
VVVV-FELD 3720
PRÄFIX-CODIERUNGS-FELD 3725
ALPHAFELD 3652
SCHREIB-MASKENFELD 3670
REAL-OPCODE-FELD 3730
3740
3750
3672
VERSCHIEBUNGSFAKTORFELD 3662B (DISP8*N, WHEN MOD=01). WIRD ALS DISP8 * N BEZEICHNET, ENTHÄLT JEDOCH NUR DEN VERSCHIEBUNGSFAKTOR, DER MIT N MULTIPLIZIERT WIRD
SPEZIFISCHES VEKTORFREUNDLICHES BEFEHLSFORMAT 3700

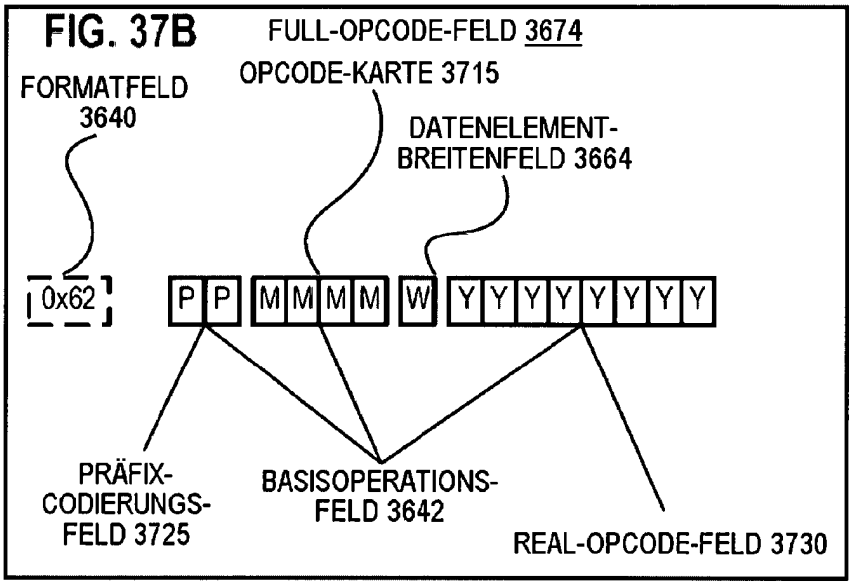


FIG. 37B
FULL-OPCODE-FELD 3674
FORMATFELD 3640
OPCODE-KARTE 3715
DATENELEMENT-BREITENFELD 3664
0x62
P P M M M M W Y Y Y Y Y Y Y Y
PRÄFIX-CODIERUNGS-FELD 3725
BASISOPERATIONS-FELD 3642
REAL-OPCODE-FELD 3730

FIG. 37C

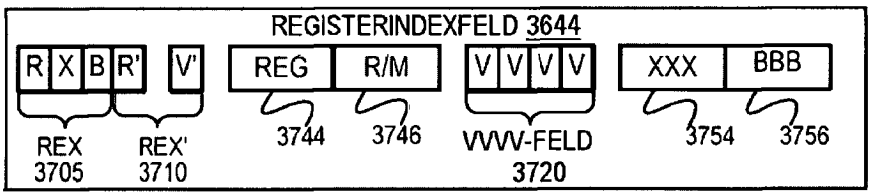
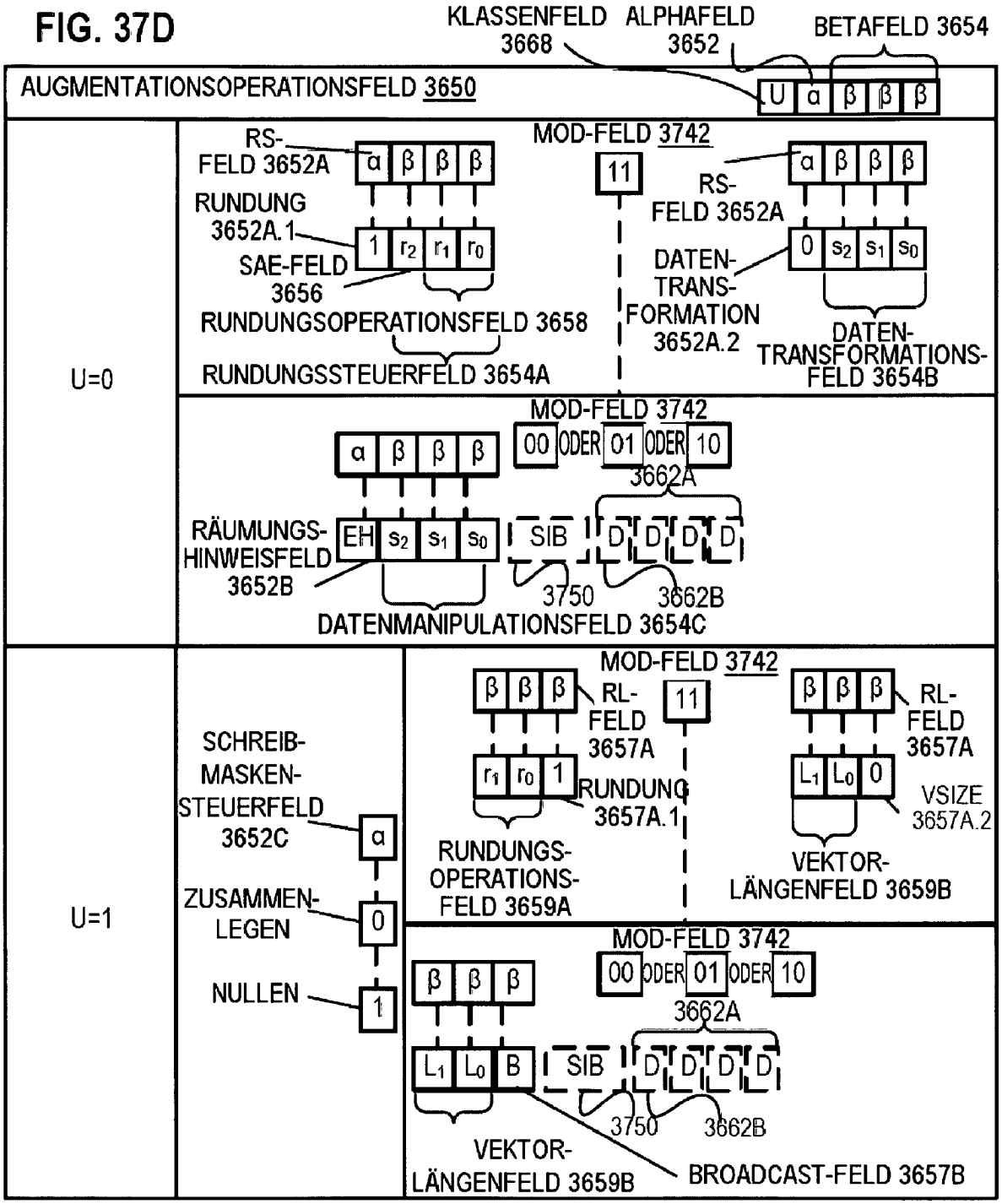

REGISTERINDEXFELD 3644
R
X
B
R'
V'
REG
R/M
V
V
V
V
XXX
BBB
REX 3705
REX' 3710
3744
3746
VVVV-FELD 3720
3754
3756

FIG. 37D


KLASSENFELD 3668
ALPHAFELD 3652
BETAFELD 3654
AUGMENTATIONSOPERATIONSFELD 3650
U α β β β
U=0
RS-FELD 3652A
RUNDUNG 3652A.1
SAE-FELD 3656
RUNDUNGSOPERATIONSFELD 3658
RUNDUNGSSTEUERFELD 3654A
MOD-FELD 3742
11
RS-FELD 3652A
DATEN-TRANS-FORMATION 3652A.2
DATEN-TRANSFORMATIONS-FELD 3654B
MOD-FELD 3742
00 ODER 01 ODER 10
3662A
RÄUMUNGS-HINWEISFELD 3652B
EH
SIB
D
3750
3662B
DATENMANIPULATIONSFELD 3654C
U=1
SCHREIB-MASKEN-STEUERFELD 3652C
ZUSAMMEN-LEGEN
NULLEN
MOD-FELD 3742
RL-FELD 3657A
RUNDUNG 3657A.1
RUNDUNGS-OPERATIONS-FELD 3659A
VSIZE 3657A.2
VEKTOR-LÄNGENFELD 3659B
MOD-FELD 3742
00 ODER 01 ODER 10
3662A
B
SIB
3750
3662B
VEKTOR-LÄNGENFELD 3659B
BROADCAST-FELD 3657B

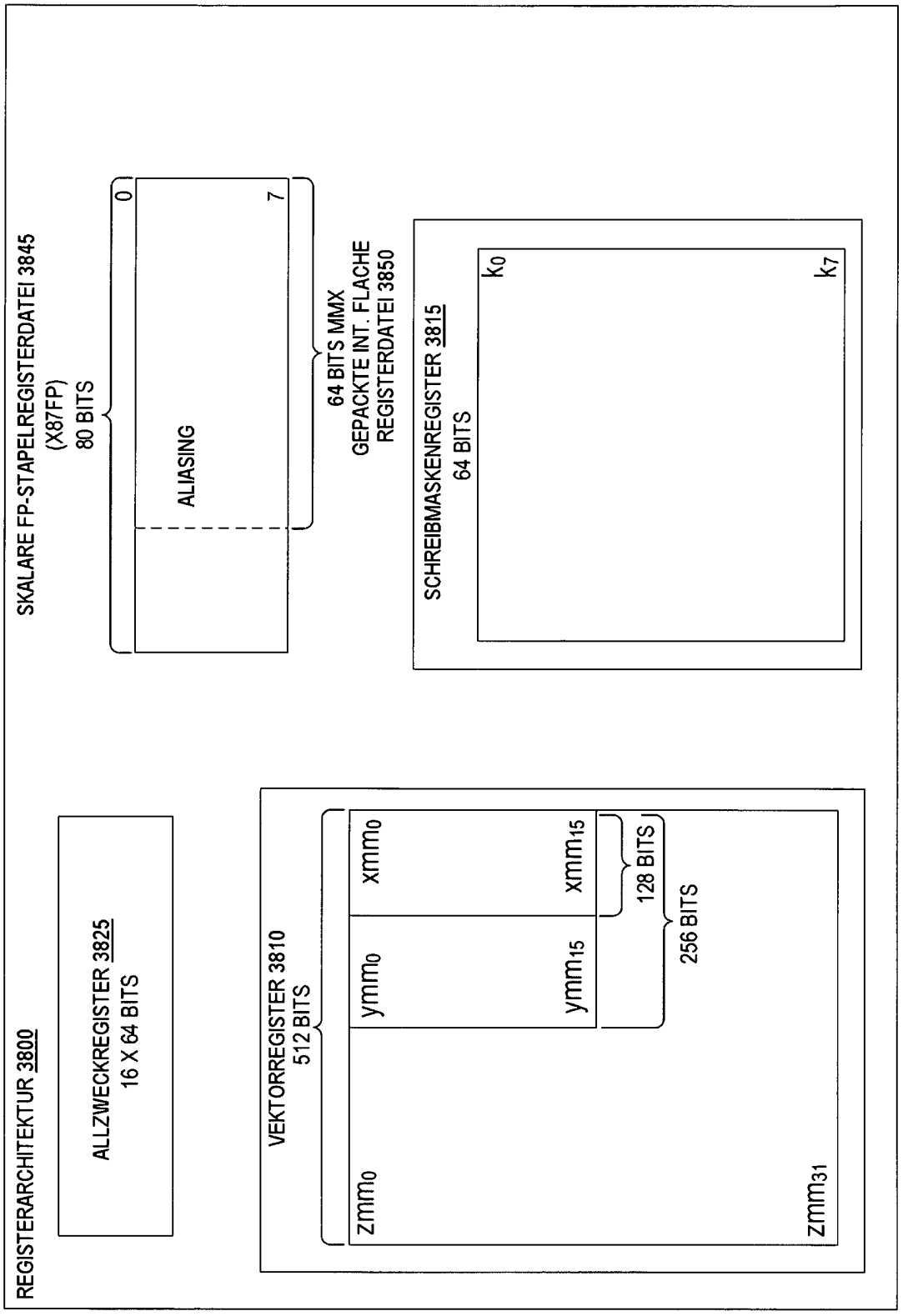

REGISTERARCHITEKTUR 3800
ALLZWECKREGISTER 3825
16 X 64 BITS
VEKTORREGISTER 3810
512 BITS
zmm0
ymm0
xmm0
ymm15
xmm15
128 BITS
256 BITS
zmm31
SKALARE FP-STAPELREGISTERDATEI 3845
(X87FP)
80 BITS
0
7
ALIASING
64 BITS MMX
GEPACKTE INT. FLACHE
REGISTERDATEI 3850
SCHREIBMASKENREGISTER 3815
64 BITS
k0
k7

FIG. 38

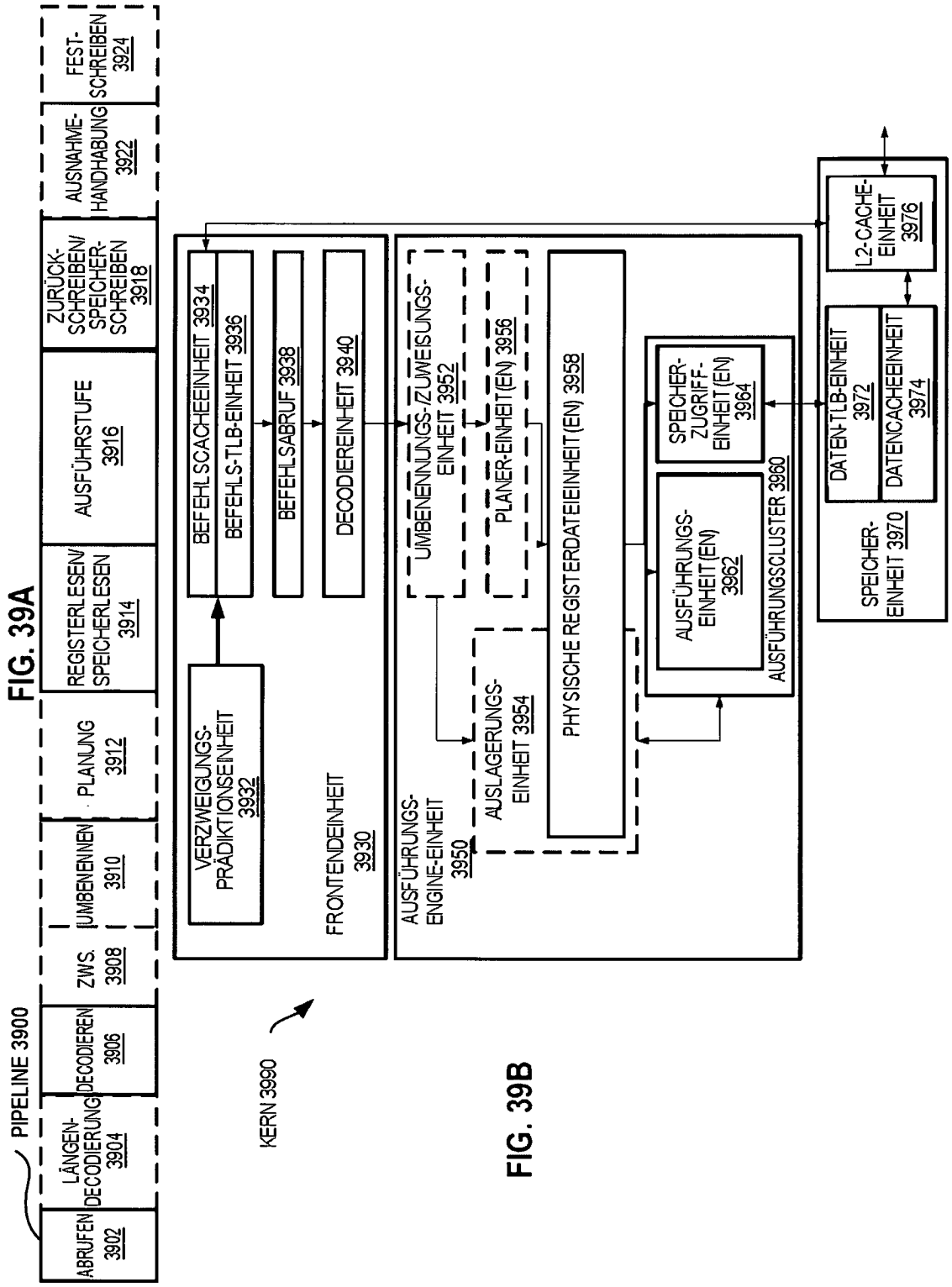


FIG. 39A

FIG. 39B

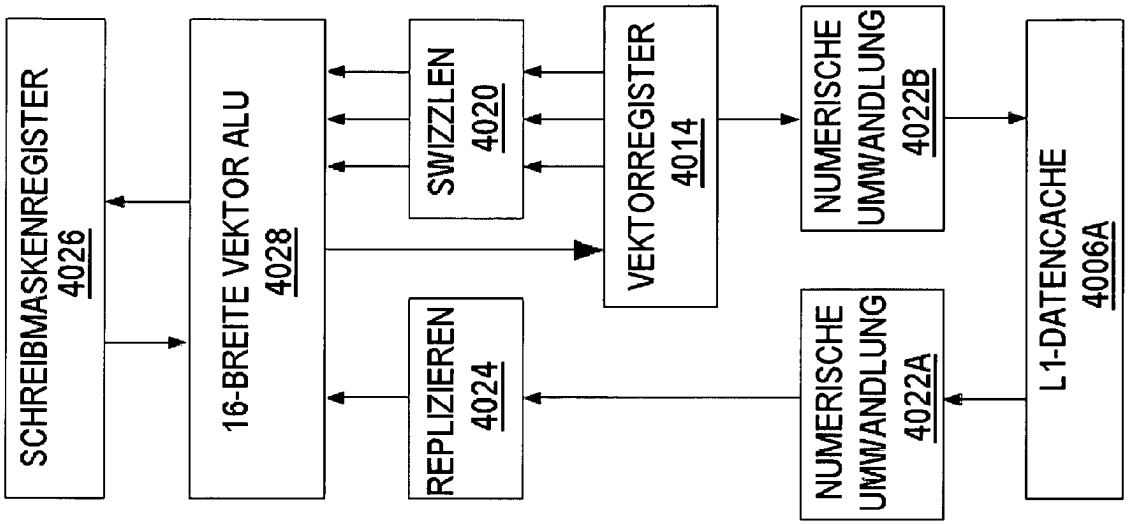


FIG. 40B

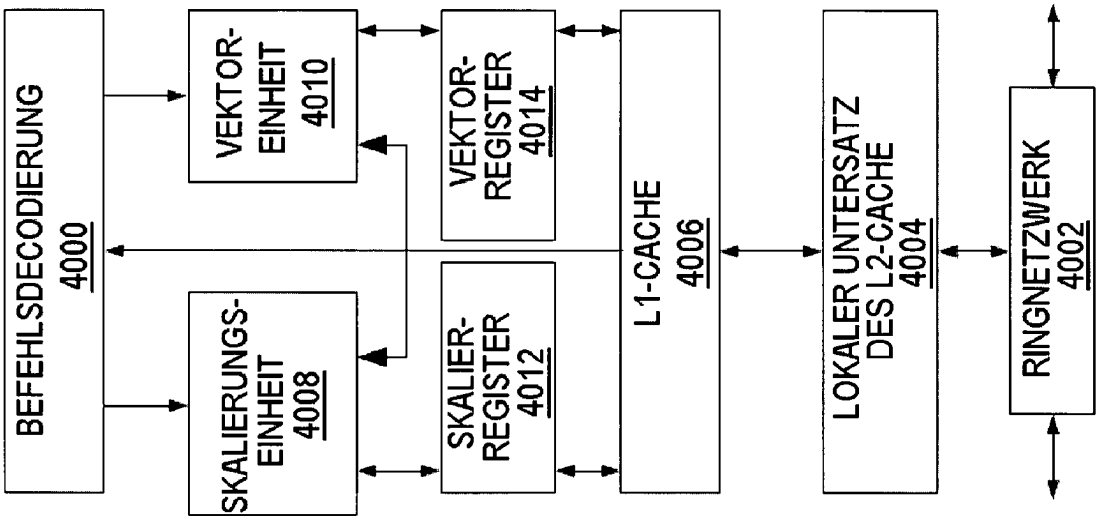


FIG. 40A

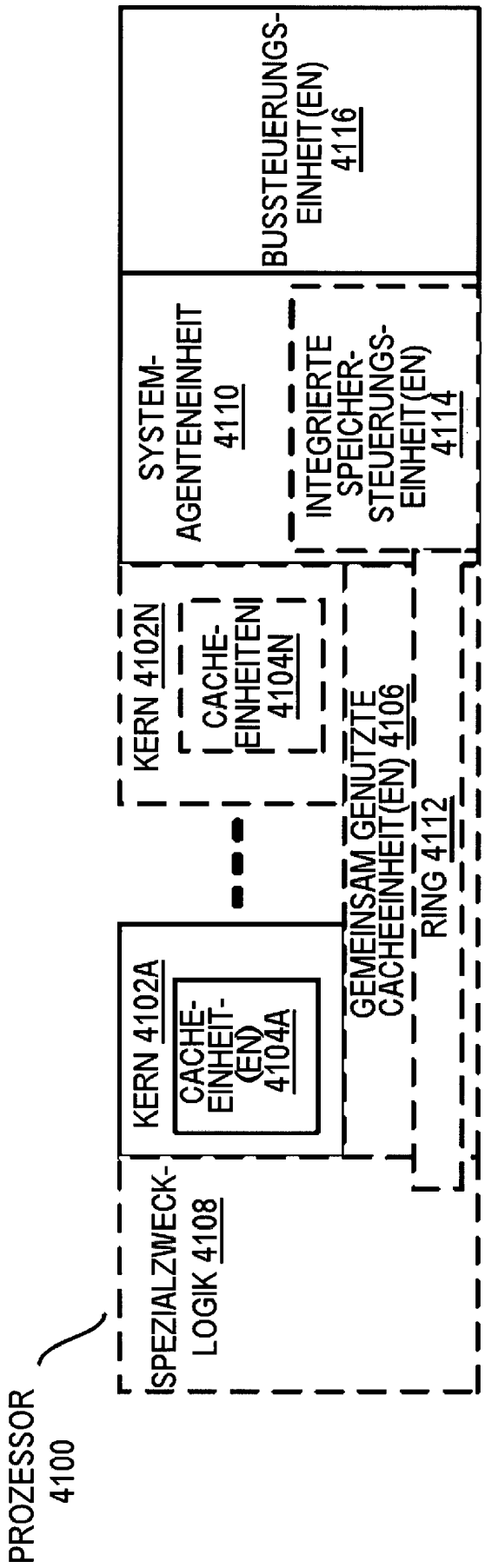

PROZESSOR 4100
SPEZIALZWECK-LOGIK 4108
KERN 4102A
CACHE-EINHEIT-(EN) 4104A
KERN 4102N
CACHE-EINHEITEN 4104N
GEMEINSAM GENUTZTE CACHEEINHEIT(EN) 4106
RING 4112
SYSTEM-AGENTENEINHEIT 4110
INTEGRIERTE SPEICHER-STEUERUNGS-EINHEIT(EN) 4114
BUSSTEUERUNGS-EINHEIT(EN) 4116

FIG. 41

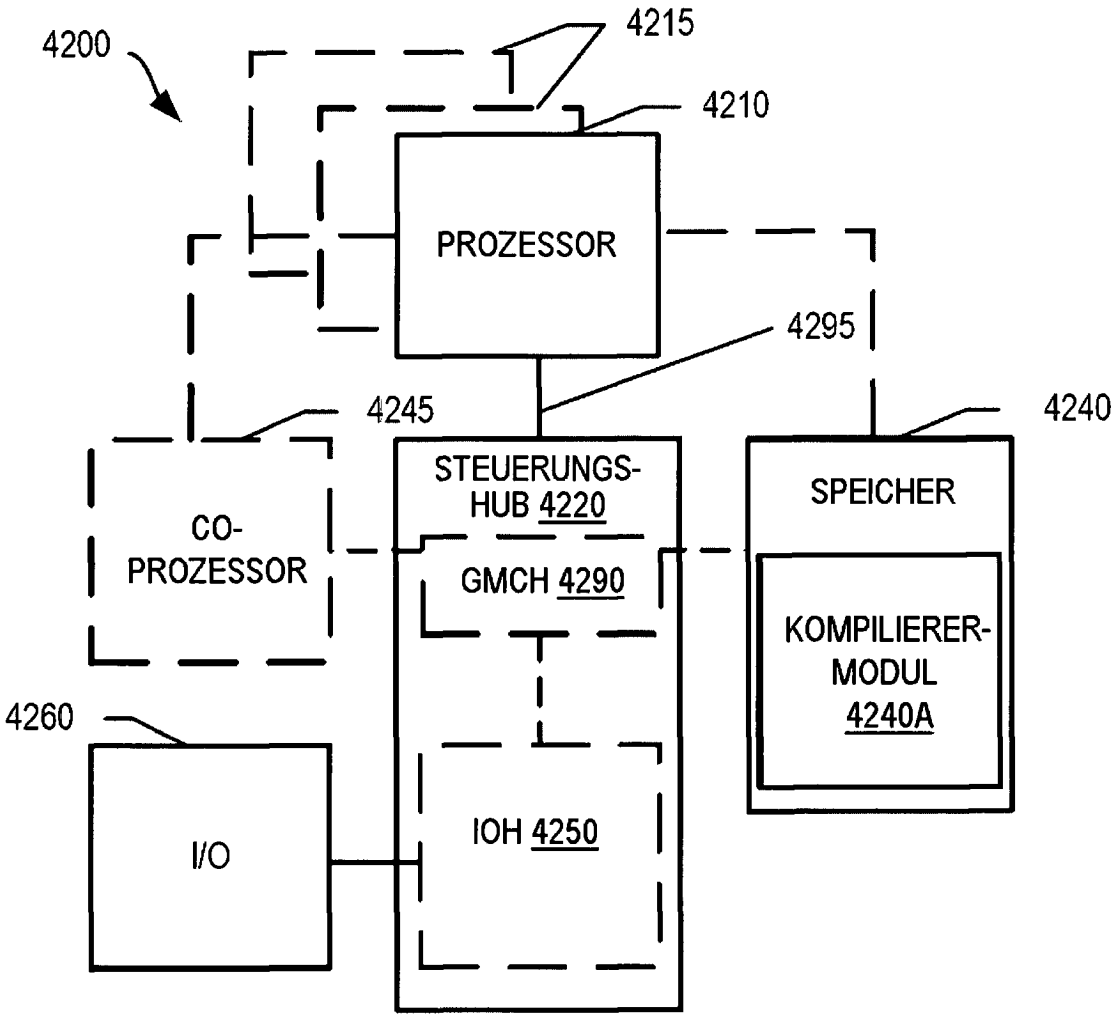

4200
4215
4210
PROZESSOR
4295
4245
4240
STEUERUNGS-
HUB 4220
SPEICHER
CO-
PROZESSOR
GMCH 4290
KOMPILIERER-
MODUL
4240A
4260
I/O
IOH 4250

FIG. 42

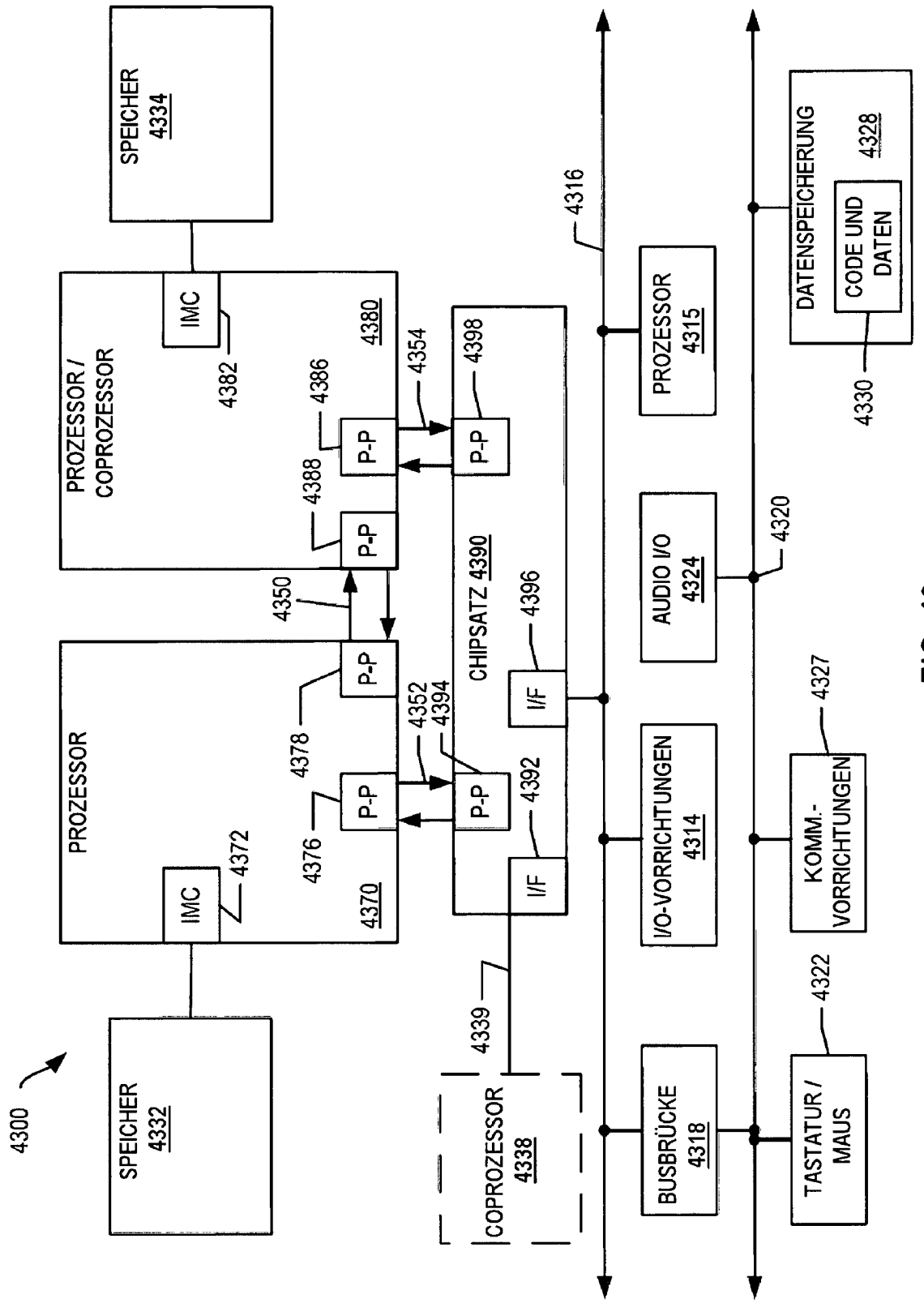

4300
SPEICHER 4332
SPEICHER 4334
PROZESSOR
IMC
4372
4370
PROZESSOR / COPROZESSOR
IMC
4382
4380
P-P
4376
4378
4386
4388
4350
4352
4354
4394
4398
CHIPSATZ 4390
I/F
4396
4392
I/F
4339
COPROZESSOR 4338
4316
PROZESSOR 4315
AUDIO I/O 4324
4320
I/O-VORRICHTUNGEN 4314
KOMM.-VORRICHTUNGEN
4327
BUSBRÜCKE 4318
TASTATUR / MAUS
4322
DATENSPEICHERUNG
CODE UND DATEN
4328
4330

FIG. 43

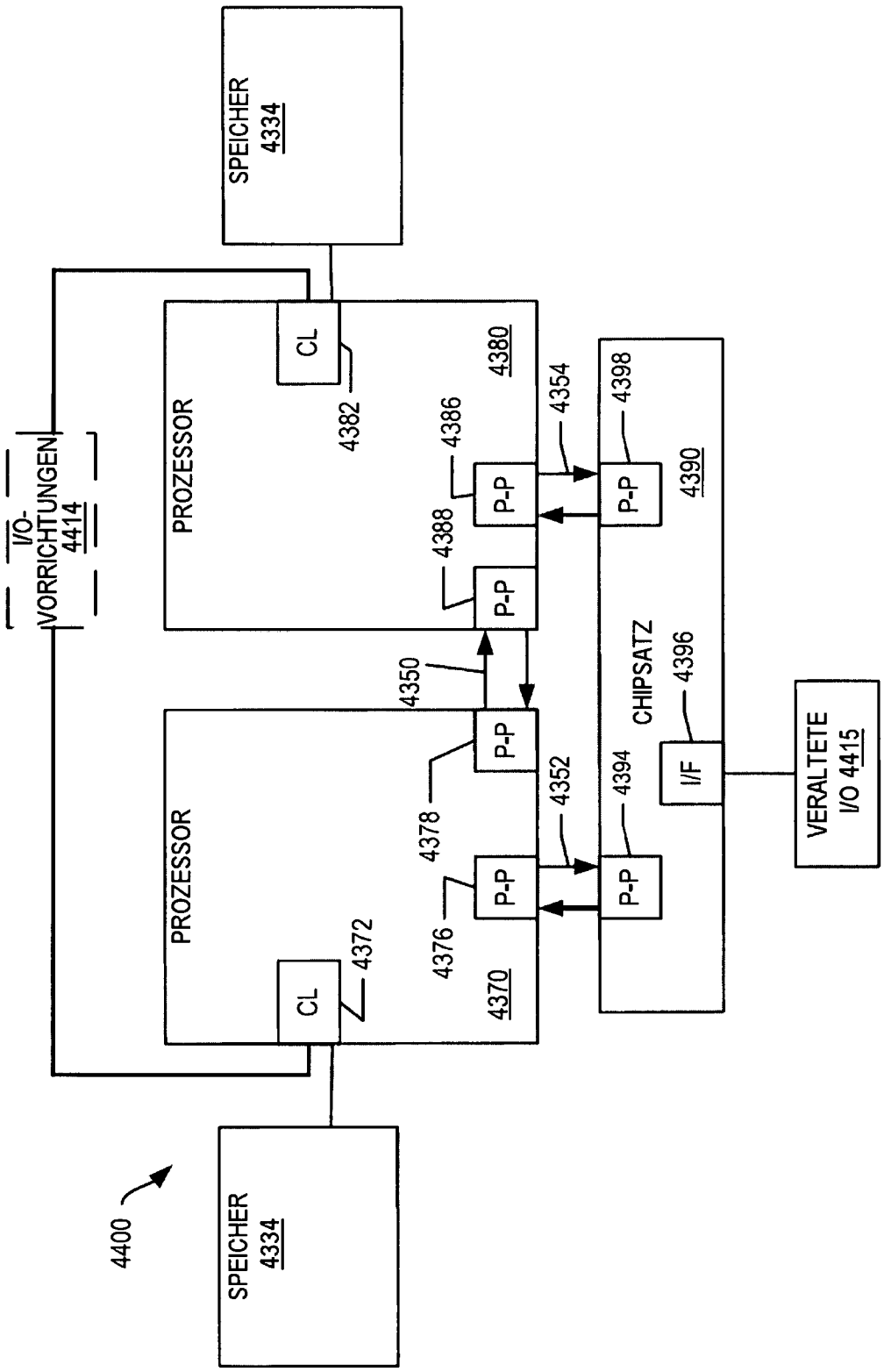

4400
SPEICHER 4334
SPEICHER 4334
I/O-VORRICHTUNGEN 4414
PROZESSOR
CL
4372
4370
PROZESSOR
CL
4382
4380
P-P
4376
P-P
4378
P-P
4386
P-P
4388
4350
4352
4354
CHIPSATZ
4390
P-P
4394
P-P
4398
I/F
4396
VERALTETE I/O 4415

FIG. 44

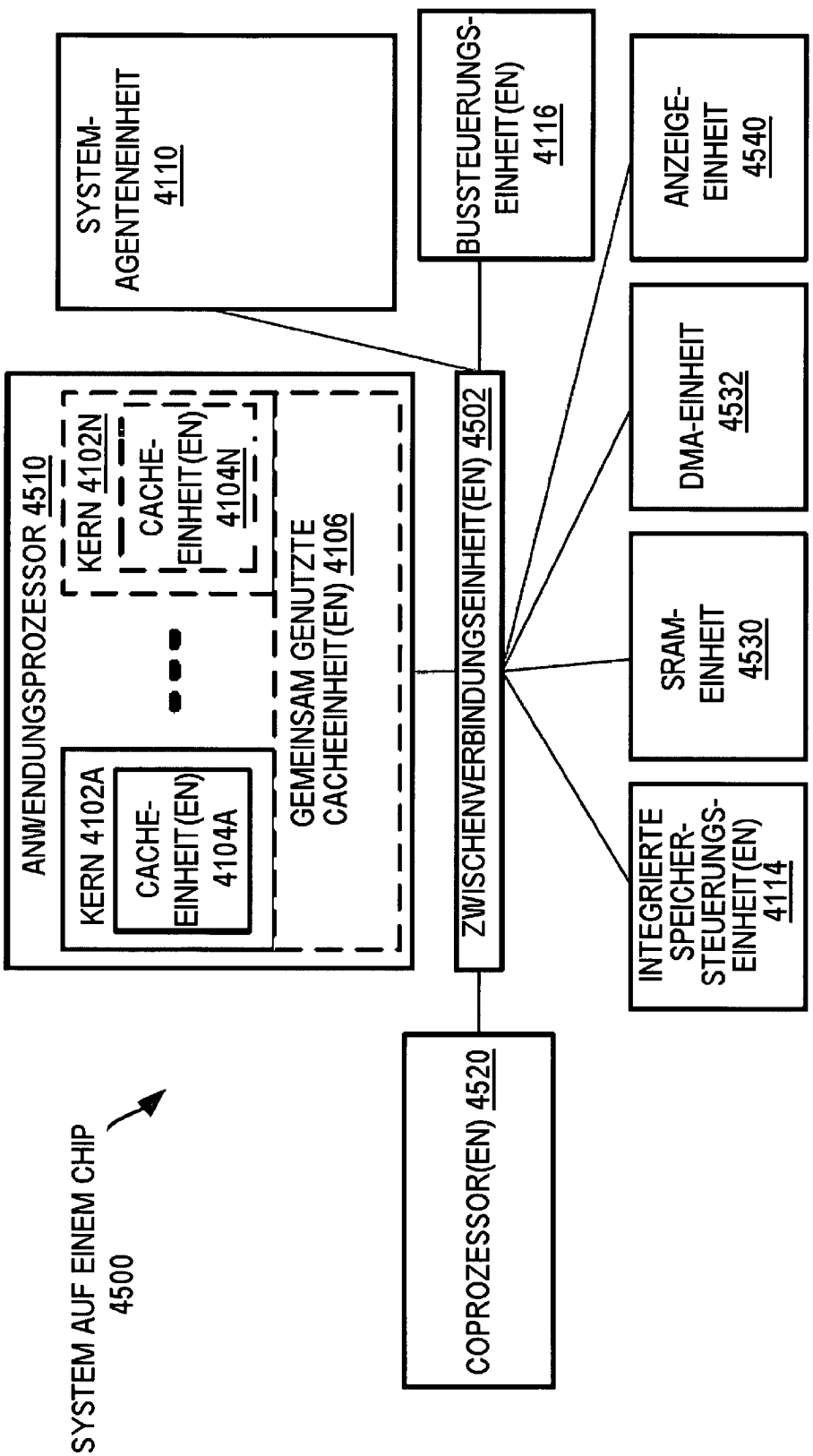

SYSTEM AUF EINEM CHIP
4500
SYSTEM-AGENTENEINHEIT
4110
BUSSTEUERUNGS-EINHEIT(EN)
4116
ANZEIGE-EINHEIT
4540
ANWENDUNGSPROZESSOR 4510
KERN 4102A
CACHE-EINHEIT(EN) 4104A
KERN 4102N
CACHE-EINHEIT(EN) 4104N
GEMEINSAM GENUTZTE CACHEEINHEIT(EN) 4106
ZWISCHENVERBINDUNGSEINHEIT(EN) 4502
DMA-EINHEIT
4532
SRAM-EINHEIT
4530
INTEGRIERTE SPEICHER-STEUERUNGS-EINHEIT(EN)
4114
COPROZESSOR(EN) 4520

FIG. 45

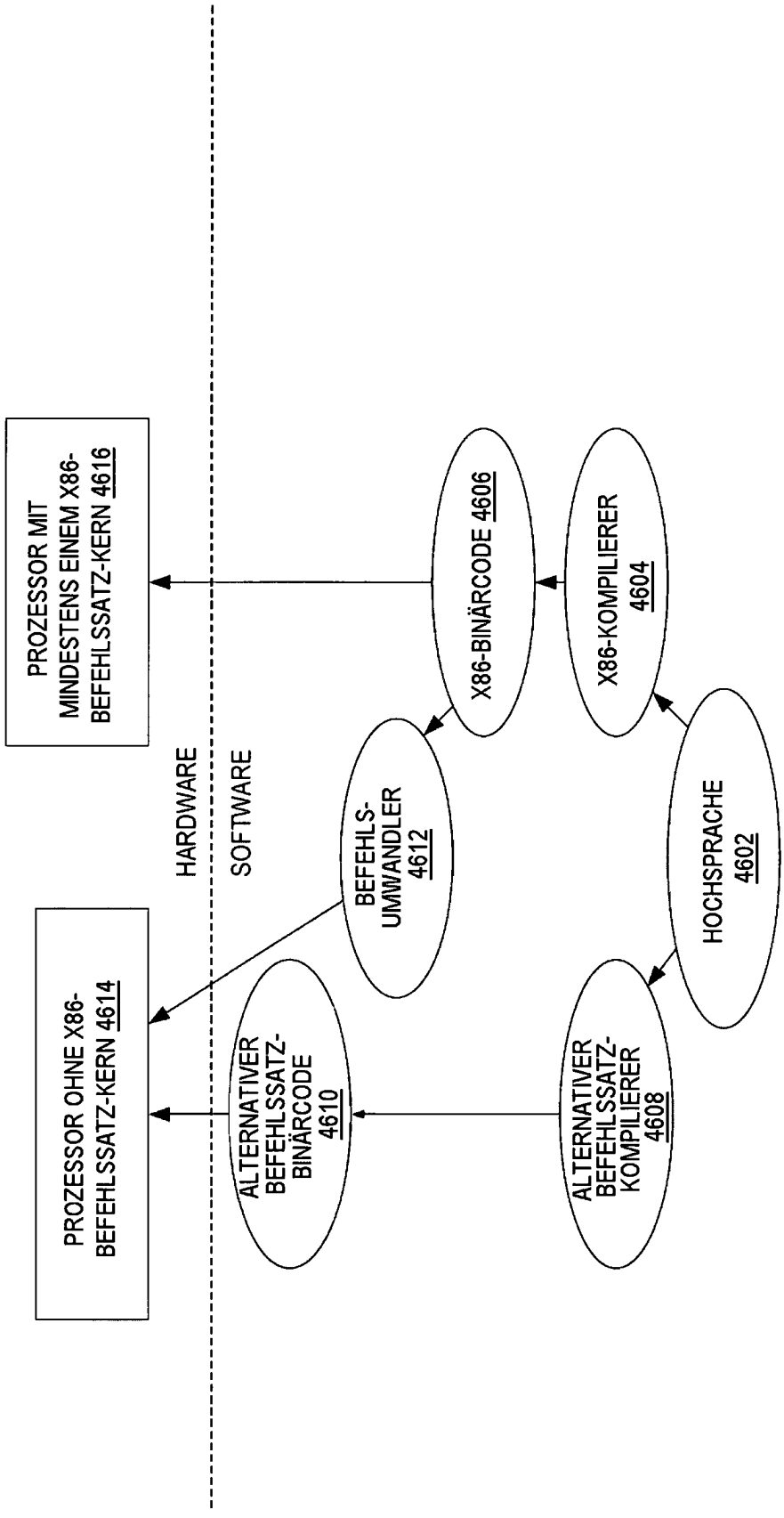

PROZESSOR MIT MINDESTENS EINEM X86-BEFEHLSSATZ-KERN 4616
PROZESSOR OHNE X86-BEFEHLSSATZ-KERN 4614
HARDWARE
SOFTWARE
X86-BINÄRCODE 4606
X86-KOMPILIERER 4604
BEFEHLS-UMWANDLER 4612
HOCHSPRACHE 4602
ALTERNATIVER BEFEHLSSATZ-BINÄRCODE 4610
ALTERNATIVER BEFEHLSSATZ-KOMPILIERER 4608

FIG. 46