



(10) **DE 10 2020 101 940 A1** 2021.03.11

(12) **Offenlegungsschrift**

(21) Aktenzeichen: **10 2020 101 940.8**

(22) Anmeldetag: **28.01.2020**

(43) Offenlegungstag: **11.03.2021**

(51) Int Cl.: **G06F 12/02 (2006.01)**

(30) Unionspriorität:

16/562,361 05.09.2019 US

(71) Anmelder:

NVIDIA Corporation, Santa Clara, Calif., US

(74) Vertreter:

Kraus & Weisert Patentanwälte PartGmbB, 80539 München, DE

(72) Erfinder:

**Duluk jun., Jerome F., Santa Clara, CA, US;
Palmer, Gregory Scott, Santa Clara, CA, US;
Evans, Jonathon Stuart Ramsay, Santa Clara,**

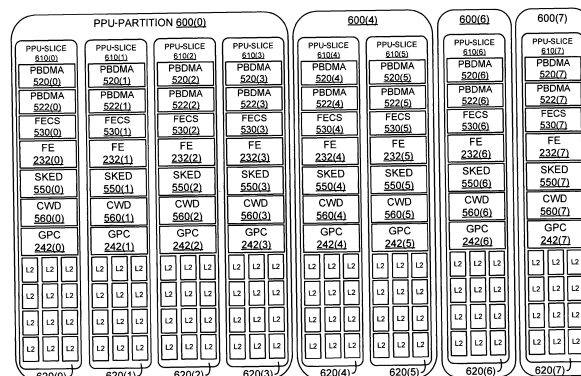
**Calif., US; Singh, Shailendra, Santa Clara, CA, US;
Duncan, Samuel H., Santa Clara, CA, US; Gandhi,
Wishwesh Anil, Santa Clara, CA, US; Shah, Lacky
V., Santa Clara, CA, US; Rock, Eric, Santa Clara,
CA, US; Su, Feiqi, Santa Clara, CA, US; Deming,
James Leroy, Santa Clara, CA, US; Menezes,
Alan, Santa Clara, CA, US; Vaidya, Pranav, Santa
Clara, CA, US; Joginipally, Praveen, Santa Clara,
CA, US; Purcell, Timothy John, Santa Clara, CA,
US; Mandal, Manas, Santa Clara, CA, US**

Prüfungsantrag gemäß § 44 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen.

(54) Bezeichnung: **TECHNIKEN ZUR KONFIGURATION EINES PROZESSORS ZUR FUNKTION ALS MEHRERE SEPARATE PROZESSOREN**

(57) Zusammenfassung: Eine Parallelverarbeitungseinheit (PPU) kann in Partitionen unterteilt sein. Jede Partition ist dazu konfiguriert, ähnlich wie die gesamte PPU zu arbeiten. Eine bestimmte Partition beinhaltet eine Teilmenge der der gesamten PPU zugeordneten Rechen- und Speicherressourcen. Software, die auf einer CPU ausgeführt wird, partitioniert die PPU für einen Verwaltungsbenutzer. Ein Gastbenutzer wird einer Partition zugewiesen und kann innerhalb dieser Partition isoliert von allen anderen Gastbenutzern, die anderen Partitionen zugewiesen sind, Verarbeitungsaufgaben durchführen. Weil die PPU in isolierte Partitionen unterteilt werden kann, können mehrere CPU-Prozesse PPU-Ressourcen effizient nutzen.



Beschreibung**HINTERGRUND**

Gebiet der verschiedenen Ausführungsformen

[0001] Verschiedene Ausführungsformen beziehen sich allgemein auf parallele Verarbeitungsarchitekturen, genauer gesagt auf Techniken zur Konfiguration eines Prozessors zur Funktion desselben als mehrere separate Prozessoren.

Beschreibung des verwandten Standes der Technik

[0002] Eine herkömmliche Zentralverarbeitungseinheit (CPU) beinhaltet in der Regel eine relativ kleine Anzahl von Verarbeitungskernen, die eine relativ kleine Anzahl von CPU-Prozessen ausführen können. Im Gegensatz dazu umfasst eine herkömmliche Grafikverarbeitungseinheit (GPU) in der Regel Hunderte von Verarbeitungskernen, die Hunderte von Threads parallel zueinander ausführen können. Dementsprechend können herkömmliche GPUs in der Regel bestimmte Verarbeitungsaufgaben schneller und effektiver ausführen als herkömmliche CPUs, da bei der Verwendung herkömmlicher GPUs größere Mengen an Verarbeitungsressourcen bereitgestellt werden können.

[0003] In einigen Implementierungen kann ein CPU-Prozess, der auf einer CPU ausgeführt wird, eine bestimmte Verarbeitungsaufgabe an einen Grafikprozessor bzw. eine GPU abgeben bzw. auslagern, damit diese Verarbeitungsaufgabe schneller ausgeführt werden kann. Dabei erzeugt der CPU-Prozess einen Verarbeitungskontext auf dem Grafikprozessor, der einen Soll- bzw. Zielzustand für die verschiedenen GPU-Ressourcen angibt, die zur Durchführung der Verarbeitungsaufgabe implementiert werden sollen. Diese GPU-Ressourcen können u. a. Verarbeitungs-, Grafik- und Speicherressourcen umfassen. Der CPU-Prozess startet dann eine Reihe von Threads auf der GPU in Übereinstimmung mit dem Verarbeitungskontext, und die Reihe von Threads nutzt die verschiedenen GPU-Ressourcen, um die Verarbeitungsaufgabe durchzuführen. Bei vielen dieser Arten von Implementierungen ist die GPU jeweils in Übereinstimmung mit nur einem Verarbeitungskontext konfiguriert. In manchen Situationen muss die CPU jedoch mehr als einen CPU-Prozess während desselben Zeitintervalls an die GPU abgeben. In solchen Situationen kann die CPU den auf der GPU implementierten Verarbeitungskontext zu verschiedenen Zeitpunkten dynamisch ändern, um diese CPU-Prozesse seriell über das Zeitintervall hinweg zu bedienen. Ein Nachteil dieses Ansatzes ist jedoch, dass die von bestimmten CPU-Prozessen ausgelagerten Verarbeitungsaufgaben die Ressourcen der GPU nicht vollständig ausnutzen. Demzufolge können dann, wenn eine oder mehrere Verarbeitungsaufgaben, die die-

sen CPU-Prozessen zugeordnet sind, seriell auf der GPU ausgeführt werden, manche GPU-Ressourcen ungenutzt bleiben, welches die Gesamtleistung und -auslastung der GPU verringert.

[0004] Ein Ansatz zur gleichzeitigen Ausführung mehrerer CPU-Prozesse auf einer GPU besteht darin, mehrere verschiedene Verarbeitungsunterkontexte innerhalb eines gegebenen „übergeordneten“ Verarbeitungskontexts zu erzeugen und jeden verschiedenen Verarbeitungsunterkontext einem anderen CPU-Prozess zuzuweisen. Mehrere CPU-Prozesse können dann gleichzeitig verschiedene Sätze von Threads auf der GPU starten, wobei jeder Thread-Satz spezifische GPU-Ressourcen nutzt, die in Übereinstimmung mit einem bestimmten Verarbeitungsunterkontext konfiguriert sind. Mit diesem Ansatz kann die GPU effizienter genutzt werden, weil mehr als ein CPU-Prozess gleichzeitig Verarbeitungsaufgaben an die GPU abgeben bzw. auslagern kann, wodurch potenziell Situationen vermieden werden können, in denen einige GPU-Ressourcen ungenutzt bleiben.

[0005] Ein Problem bei dem vorstehenden Ansatz besteht darin, dass CPU-Prozesse, die verschiedenen Verarbeitungsunterkontexten zugeordnet sind, GPU-Ressourcen, die über die verschiedenen Verarbeitungsunterkontexte hinweg gleichmäßiger zuzuweisen oder auf diese zu verteilen sind, ungleichmäßig in Anspruch nehmen können. Beispielsweise könnte ein erster CPU-Prozess einen ersten Satz von Threads innerhalb eines ersten Verarbeitungsunterkontexts starten, der eine große Menge von Leseanforderungen ausführt und eine große Menge an verfügbarer GPU-Speicherbandbreite verbraucht. Ein zweiter CPU-Prozess könnte anschließend einen zweiten Satz von Threads innerhalb eines zweiten Verarbeitungsunterkontexts starten, der ebenfalls eine große Menge von Leseanforderungen durchführt. Weil jedoch ein viel der verfügbaren GPU-Speicherbandbreite bereits von dem ersten Satz von Threads verbraucht wird, könnte es bei dem zweiten Satz von Threads zu hohen Latenzen kommen, welches zu einem Stillstand des zweiten CPU-Prozesses führen könnte.

[0006] Ein weiteres Problem bei dem vorstehenden Ansatz besteht darin, dass aufgrund der Tatsache, dass sich Verarbeitungsunterkontexte einen übergeordneten Kontext teilen, alle Fehler, die auftreten, wenn die Threads, die einem Verarbeitungsunterkontext zugeordnet sind, zur Ausführung kommen, die Ausführung anderer Threads stören können, die einem anderen Verarbeitungsunterkontext zugeordnet sind, der denselben übergeordneten Kontext gemeinsam hat. Beispielsweise könnte ein erster CPU-Prozess einen ersten Satz von Threads starten, die einem ersten Verarbeitungsunterkontext zugeordnet sind, um eine erste Verarbeitungsaufgabe auszufüh-

ren. Ein zweiter CPU-Prozess könnte einen zweiten Satz von Threads starten, die einem zweiten Verarbeitungsunterkontext zugeordnet sind, und der zweite Satz von Threads könnte anschließend einem Fehler unterliegen und fehlschlagen. Um sich von dem Fehler zu erholen, müsste die GPU den übergeordneten Kontext zurücksetzen, wodurch sowohl der erste Verarbeitungsunterkontext als auch der zweite Verarbeitungsunterkontext automatisch zurückgesetzt würden. In einem solchen Szenario würde die Ausführung des ersten Satzes von Threads unterbrochen, obwohl der Fehler dem zweiten Satz von Threads und nicht dem ersten Satz von Threads entstammte.

[0007] Wie das Vorstehende veranschaulicht, sind im Stand der Technik effektivere Techniken notwendig zum Konfigurieren einer GPU dazu, Verarbeitungsaufgaben auszuführen, die mehreren Kontexten zugeordnet sind.

KURZBESCHREIBUNG

[0008] Verschiedene Ausführungsformen beinhalten ein computer-implementiertes Verfahren, einschließlich einer Partitionierung eines Satzes von Hardwareressourcen, die in einem Prozessor enthalten sind, um eine erste logische Partition zu erzeugen, die eine erste Teilmenge von Hardwareressourcen beinhaltet, und einer Erzeugung einer Vielzahl von Maschinen innerhalb der ersten logischen Partition, wobei jeder in der Vielzahl von Maschinen enthaltenen Maschine ein anderer Teil der ersten Teilmenge von Hardwareressourcen zugewiesen bzw. allokiert wird und diese in funktionaler Isolation von allen anderen Maschinen, die in der Vielzahl von Maschinen enthalten sind, ausgeführt wird.

[0009] Ein technologischer Vorteil der offenbarten Techniken gegenüber dem Stand der Technik besteht darin, dass mit den offenbarten Techniken eine Parallelverarbeitungseinheit (PPU) (z.B. eine GPU) mehrere Kontexte gleichzeitig und funktional voneinander isoliert unterstützen kann. Dementsprechend können mehrere CPU-Prozesse PPU-Ressourcen effizient nutzen, indem sie gleichzeitig mehrere verschiedene Kontexte ausführen, ohne dass sich die Kontexte gegenseitig stören.

Figurenliste

[0010] Zum Verständnis der Art und Weise der vorstehend wiedergegebenen Merkmale der verschiedenen Ausführungsformen im Einzelnen werden nachstehend die vorstehend kurz zusammengefassten erfinderischen Konzepte unter Bezugnahme auf verschiedene Ausführungsformen, von denen einige in den beigefügten Zeichnungen dargestellt sind, beschrieben. Es wird jedoch darauf hinzuweisen, dass die beigefügten Zeichnungen nur typische

Ausführungsformen der erfinderischen Konzepte darstellen und daher nicht als in irgendeiner Weise beschränkend zu betrachten sind, und dass es andere ebenso effektive Ausführungsformen gibt.

Fig. 1 ist ein Blockdiagramm eines Computersystems, das dazu konfiguriert ist, einen oder mehrere Aspekte der verschiedenen Ausführungsformen zu implementieren;

Fig. 2 ist ein Blockdiagramm einer Parallelverarbeitungseinheit (PPU), die in dem Parallelverarbeitungsuntersystem von **Fig. 1** enthalten ist, gemäß verschiedenen Ausführungsformen;

Fig. 3 ist ein Blockdiagramm eines Universalverarbeitungsclusters, der in der Parallelverarbeitungseinheit von **Fig. 2** enthalten ist, gemäß verschiedenen Ausführungsformen;

Fig. 4 ist ein Blockdiagramm einer in der PPU von **Fig. 2** enthaltenen Partitionseinheit, gemäß verschiedenen Ausführungsformen;

Fig. 5 ist ein Blockdiagramm verschiedener PPU-Ressourcen, die in der PPU von **Fig. 2** enthalten sind, gemäß verschiedenen Ausführungsformen;

Fig. 6 ist ein Beispiel dafür, wie der Hypervisor von **Fig. 1** PPU-Ressourcen logisch in einen Satz von PPU-Partitionen gruppiert, gemäß verschiedenen Ausführungsformen;

Fig. 7 veranschaulicht, wie der Hypervisor von **Fig. 1** einen Satz von PPU-Partitionen dazu konfiguriert, eine oder mehrere SMC (Simultaneous Multiple Context)-Engines zu implementieren, gemäß verschiedenen Ausführungsformen;

Fig. 8A ist eine detailliertere Darstellung des DRAMs von **Fig. 7**, gemäß verschiedenen Ausführungsformen;

Fig. 8B veranschaulicht, wie die verschiedenen DRAM-Abschnitte von **Fig. 8B** adressiert werden, gemäß verschiedenen Ausführungsformen;

Fig. 9 ist ein Datenflussdiagramm, das veranschaulicht, wie der Hypervisor von **Fig. 1** eine PPU partitioniert und konfiguriert, gemäß verschiedenen Ausführungsformen;

Fig. 10 ist ein Ablaufdiagramm von Verfahrensschritten zur Partitionierung und Konfiguration einer PPU im Auftrag eines oder mehrerer Benutzer, gemäß verschiedenen Ausführungsformen;

Fig. 11 veranschaulicht eine Partitionskonfigurationstabelle, gemäß der der Hypervisor von **Fig. 1** eine oder mehrere PPU-Partitionen konfigurieren kann, gemäß verschiedenen Ausführungsformen;

Fig. 12 veranschaulicht, wie der Hypervisor von **Fig. 1** eine PPU partitioniert, um eine oder mehrere PPU-Partitionen zu erzeugen, gemäß verschiedenen Ausführungsformen;

Fig. 13 veranschaulicht, wie der Hypervisor von **Fig. 1** verschiedene PPU-Ressourcen während der Partitionierung zuweist, gemäß verschiedenen Ausführungsformen;

Fig. 14A veranschaulicht, wie mehrere Gastbetriebssysteme mit mehreren laufenden VMs gleichzeitig mehrere Verarbeitungskontexte innerhalb einer oder mehrerer PPU-Partitionen starten, gemäß verschiedenen Ausführungsformen;

Fig. 14B veranschaulicht, wie ein Hostbetriebssystem mehrere Verarbeitungskontexte gleichzeitig innerhalb einer oder mehrerer PPU-Partitionen startet, gemäß verschiedenen Ausführungsformen;

Fig. 15 veranschaulicht, wie der Hypervisor von **Fig. 1** verschiedenen SMC-Engines Virtualladressraumidentifizierer zuweist, gemäß verschiedenen Ausführungsformen;

Fig. 16 veranschaulicht, wie eine Speicherverwaltungseinheit lokale Virtualladressraumidentifizierer bei einer Fehlerabschwächung übersetzt, gemäß verschiedenen Ausführungsformen;

Fig. 17 veranschaulicht, wie der Hypervisor von **Fig. 1** ein sanftes Floorsweeping bei der Migration eines Verarbeitungskontexts zwischen SMC-Engines auf verschiedenen PPUs implementiert, gemäß verschiedenen Ausführungsformen;

Fig. 18 ist ein Ablaufdiagramm von Verfahrensschritten zur Konfiguration von Rechenressourcen innerhalb einer PPU zur Unterstützung von Operationen, die mehreren Verarbeitungskontexten gleichzeitig zugeordnet sind, gemäß verschiedenen Ausführungsformen;

Fig. 19 veranschaulicht einen Satz von Grenzoptionen, in Übereinstimmung mit welchen der Hypervisor von **Fig. 1** eine oder mehrere PPU-Speicherpartitionen erzeugen kann, gemäß verschiedenen Ausführungsformen;

Fig. 20 zeigt ein Beispiel dafür, wie der Hypervisor von **Fig. 1** PPU-Speicher partitioniert, um eine oder mehrere PPU-Speicherpartitionen zu erzeugen, gemäß verschiedenen Ausführungsformen;

Fig. 21 veranschaulicht, wie die Speicherverwaltungseinheit von **Fig. 16** Zugriff auf verschiedene PPU-Speicherpartitionen bereitstellt, entsprechend den verschiedenen Ausführungsformen;

Fig. 22 veranschaulicht, wie die Speicherverwaltungseinheit von **Fig. 16** verschiedene Adressübersetzungen durchführt, gemäß verschiedenen Ausführungsformen;

Fig. 23 veranschaulicht, wie die Speicherverwaltungseinheit von **Fig. 16** Unterstützungsoperationen, die mehreren Verarbeitungskontexten zugeordnet sind, gleichzeitig bereitstellt, gemäß verschiedenen Ausführungsformen;

Fig. 24 ist ein Ablaufdiagramm von Verfahrensschritten zur Konfiguration von Speicherressourcen innerhalb einer PPU zur Unterstützung von Operationen, die mehreren Verarbeitungskontexten gleichzeitig zugeordnet sind, gemäß verschiedenen Ausführungsformen;

Fig. 25 ist ein Satz von Zeitleisten, die ein Zuordnen von Zeitscheiben auf VM-Ebene in Verbindung mit der PPU von **Fig. 2** veranschaulichen, gemäß verschiedenen Ausführungsformen;

Fig. 26 ist ein weiterer Satz von Zeitleisten, der das Zuordnen von Zeitscheiben auf VM-Ebene in Verbindung mit der PPU von **Fig. 2** veranschaulicht, gemäß verschiedenen anderen Ausführungsformen;

Fig. 27 ist eine Zeitleiste, die eine Zuordnung von Zeitscheiben auf SMC-Ebene in Verbindung mit der PPU von **Fig. 2** veranschaulicht, gemäß verschiedenen Ausführungsformen;

Fig. 28 veranschaulicht, wie VMs von einer PPU zu einer anderen PPU migrieren können, gemäß verschiedenen Ausführungsformen;

Fig. 29 ist ein Satz von Zeitleisten, die eine feine VM-Migration in Verbindung mit der PPU von **Fig. 2** veranschaulichen, gemäß verschiedenen Ausführungsformen;

Fig. 30A-30B stellen ein Ablaufdiagramm von Verfahrensschritten zur Zuordnung von Zeitscheiben für VMs in der PPU von **Fig. 2** dar, gemäß verschiedenen Ausführungsformen;

Fig. 31 ist eine Speicherabbildung, die darstellt, wie der BARO-Adressraum auf den privilegierten Registerraum innerhalb der PPU von **Fig. 2** abgebildet wird, gemäß verschiedenen Ausführungsformen;

Fig. 32 ist ein Ablaufdiagramm von Verfahrensschritten zur Adressierung privilegierten Registeradressraums in der PPU von **Fig. 2**, gemäß verschiedenen Ausführungsformen;

Fig. 33 ist ein Blockdiagramm eines Leistungsüberwachungssystems für die PPU von **Fig. 2**, gemäß verschiedenen Ausführungsformen;

Fig. 34A-34B veranschaulichen verschiedene Konfigurationen der Leistungs-Multiplexer-Ein-

heiten von **Fig. 33**, gemäß verschiedenen Ausführungsformen;

Fig. 35 ist ein Blockdiagramm eines Leistungsüberwachungsaggregationssystems für die PPU von **Fig. 2**, gemäß verschiedenen Ausführungsformen;

Fig. 36 veranschaulicht das Format von Triggerpaketen, die dem Leistungsüberwachungsaggregationssystem von **Fig. 35** zugeordnet sind, gemäß verschiedenen Ausführungsformen;

Fig. 37 ist ein Ablaufdiagramm von Verfahrensschritten zur Überwachung der Leistung der PPU von **Fig. 2**, gemäß verschiedenen Ausführungsformen;

Fig. 38 ist ein Blockdiagramm eines Leistungs- und Taktfrequenz-Verwaltungssystems für die PPU von **Fig. 2**, gemäß verschiedenen Ausführungsformen; und

Fig. 39 ist ein Ablaufdiagramm von Verfahrensschritten zur Verwaltung des Stromverbrauchs der PPU **200** von **Fig. 2**, gemäß verschiedenen Ausführungsformen.

DETAILLIERTE BESCHREIBUNG

[0011] In der folgenden Beschreibung werden zahlreiche spezifische Einzelheiten dargelegt, um ein tieferes Verständnis der verschiedenen Ausführungsformen zu ermöglichen. Für den Fachmann versteht sich jedoch, dass die erfinderischen Konzepte auch ohne eines oder mehrerer dieser spezifischen Einzelheiten in die Praxis umgesetzt werden können.

[0012] Wie vorstehend erwähnt wurde, können konventionelle GPUs bestimmte Verarbeitungsaufgaben in der Regel schneller ausführen als konventionelle CPUs. In manchen Konfigurationen kann ein auf einer CPU ausgeführter CPU-Prozess eine bestimmte Verarbeitungsaufgabe an eine GPU abgeben bzw. auslagern, um diese Verarbeitungsaufgabe schneller auszuführen. Dabei erzeugt der CPU-Prozess einen Verarbeitungskontext auf der GPU, der einen Soll- bzw. Zielzustand für verschiedene GPU-Ressourcen spezifiziert, und startet dann eine Reihe von Threads auf der GPU, um die Verarbeitungsaufgabe durchzuführen.

[0013] In einigen Situationen kann es erforderlich sein, dass mehr als ein CPU-Prozess während desselben Zeitintervalls Verarbeitungsaufgaben an die GPU abgeben muss. Die GPU kann jedoch nur für jeweils einen Verarbeitungskontext zur gleichen Zeit konfiguriert sein. In solchen Situationen kann die CPU den Verarbeitungskontext der GPU zu verschiedenen Zeitpunkten dynamisch ändern, um die mehreren CPU-Prozesse über das Zeitintervall hinweg seriell zu bedienen. Es kann jedoch vorkommen, dass bestimmte CPU-Prozesse die GPU-Ressour-

cen bei der Durchführung von Verarbeitungsaufgaben nicht voll austasten, so dass verschiedene GPU-Ressourcen zeitweise ungenutzt bleiben. Um dieses Problem zu beheben, kann die CPU mehrere Verarbeitungsunterkontexte innerhalb eines „übergeordneten“ Verarbeitungskontexts erzeugen und diese Verarbeitungsunterkontexte verschiedenen CPU-Prozessen zuweisen. Diese CPU-Prozesse können dann verschiedene Thread-Datensätze auf der GPU gleichzeitig starten, und jeder Thread-Satz kann bestimmte GPU-Ressourcen nutzen, die entsprechend einem bestimmten Verarbeitungsunterkontext konfiguriert sind. Dieser Ansatz kann zur effizienteren Nutzung von GPU-Ressourcen implementiert werden. Dieser Ansatz leidet jedoch an einigen Nachteilen.

[0014] Erstens können CPU-Prozesse, die verschiedenen Verarbeitungsunterkontexten zugeordnet sind, GPU-Ressourcen in unfairer Weise verbrauchen, die gerecht auf die verschiedenen Verarbeitungsunterkontexte verteilt werden sollten, was zu Situationen führen kann, in denen ein CPU-Prozess den Fortschritt eines anderen CPU-Prozesses blockieren kann. Zweitens können, weil sich die Verarbeitungsunterkontexte einen übergeordneten Verarbeitungskontext teilen, Fehler, die während der Ausführung von Threads auftreten, die einem Verarbeitungsunterkontext zugeordnet sind, die Ausführung von Threads stören, die anderen Verarbeitungsunterkontexten zugeordnet sind, die in demselben übergeordneten Verarbeitungskontext enthalten sind. In einigen Fällen kann ein Fehler, der innerhalb eines Verarbeitungsunterkontexts auftritt, dazu führen, dass alle anderen Verarbeitungsunterkontexte innerhalb desselben übergeordneten Verarbeitungskontexts zurückgesetzt und neu gestartet werden.

[0015] Allgemein schränken die vorstehenden Nachteile, die mit der Verarbeitung von Unterkontexten zugeordnet sind, das Ausmaß ein, in dem konventionelle GPUs Multitenancy bzw. Mandantenfähigkeit unterstützen können. Wie hierin in Bezug genommen, bezieht sich „Multitenancy“ auf GPU-Konfigurationen, bei denen mehrere Benutzer oder „Inhaber“ gleichzeitig oder während sich überlappende Zeitintervalle Verarbeitungsoperationen unter Verwendung von GPU-Ressourcen durchführen. In der Regel stellen konventionelle GPUs Unterstützung für Mandantenfähigkeit bereit, indem sie es verschiedenen Inhabern ermöglichen, verschiedene Verarbeitungsaufgaben unter Verwendung verschiedener Verarbeitungsunterkontexte innerhalb eines gegebenen übergeordneten Verarbeitungskontexts auszuführen. Verarbeitungsunterkontexte sind jedoch keine isolierten Rechenumgebungen, weil sich Verarbeitungsaufgaben, die innerhalb verschiedener Verarbeitungsunterkontexte ausgeführt werden, aus den verschiedenen, vorstehend diskutierten Gründen gegenseitig stören können. Folglich kann sich jeder Inhaber, der eine bestimmte GPU belegt, negativ auf

die Servicequalität auswirken, die die GPU anderen Inhabern bietet. Diese Faktoren können die Attraktivität von Cloud-basierten GPU-Entwicklungen verringern, bei denen mehrere Benutzer gleichzeitig Zugriff auf dieselbe GPU haben können.

[0016] Zur Lösung dieser Probleme beinhalten verschiedene Ausführungsformen eine Parallelverarbeitungseinheit (PPU), die in Partitionen unterteilt sein kann. Jede Partition ist dazu konfiguriert, Verarbeitungsaufgaben, die mehreren Verarbeitungskontexten zugeordnet sind, gleichzeitig auszuführen. Eine gegebene Partition beinhaltet eine oder mehrere logische Gruppierungen oder „Slices“ bzw. „Scheiben“ von GPU-Ressourcen. Jeder Slice stellt ausreichend Rechen-, Grafik- und Speicherressourcen bereit, um den Betriebsablauf der PPU als Ganzes nachzuahmen. Ein Hypervisor, der auf einer CPU ausgeführt wird, führt im Auftrag eines Admin- bzw. Verwaltungsbenutzers verschiedene Verfahren zur Partitionierung der PPU aus. Ein Gastbenutzer wird einer Partition zugewiesen und kann dann innerhalb dieser Partition isoliert von allen anderen Gastbenutzern, die anderen Partitionen zugewiesen sind, Verarbeitungsaufgaben durchführen.

[0017] Ein technologischer Vorteil der offenbarten Techniken gegenüber dem Stand der Technik besteht darin, dass eine PPU mit den offenbarten Techniken mehrere Verarbeitungskontexte gleichzeitig und in funktioneller Isolation voneinander unterstützen kann. Dementsprechend können mehrere CPU-Prozesse PPU-Ressourcen effizient über mehrere verschiedene Verarbeitungskontexte und ohne gegenseitige Beeinträchtigung nutzen. Ein weiterer technologischer Vorteil der offenbarten Techniken besteht darin, dass die PPU, weil sie unter Verwendung der offenbarten Techniken in isolierte Rechenumgebungen partitioniert werden kann, eine robustere Form der Multitenancy unterstützen kann, als dies bei Ansätzen nach dem Stand der Technik der Fall ist, die sich zur Bereitstellung von Multitenancy-Funktionalität auf Verarbeitungsunterkontexte stützen. Dementsprechend eignet sich eine PPU bei Implementierung der offenbarten Techniken besser für Cloud-basierte Entwicklungen, bei denen verschiedenen und potenziell konkurrierenden Entitäten Zugriff auf verschiedene Partitionen innerhalb derselben PPU gewährt werden kann. Diese technologischen Vorteile repräsentieren einen oder mehrere technologische Fortschritte gegenüber bekannten Ansätzen.

Systemübersicht

[0018] Fig. 1 ist ein Blockdiagramm eines Computersystems, das dazu konfiguriert ist, einen oder mehrere Aspekte der Erfindung zu implementieren. Wie gezeigt ist, beinhaltet das Computersystem **100** eine Zentralverarbeitungseinheit (CPU) **110**, einen Systemspeicher **120** und ein Parallelverarbeitungsunter-

system **130**, die über eine Speicherbrücke **132** miteinander gekoppelt sind. Das Parallelverarbeitungs-untersystem **130** ist über einen Kommunikationspfad **134** mit der Speicherbrücke **132** gekoppelt. Eine oder mehrere Anzeigevorrichtungen **136** können mit dem Parallelverarbeitungsuntersystem **130** gekoppelt sein. Das Computersystem **100** beinhaltet ferner eine Systemplatte **140**, eine oder mehrere Add-In- bzw. Zusatzkarten **150** und einen Netzwerkadapter **160**. Die Systemplatte **140** ist mit einer E/A-Brücke **142** gekoppelt. Die E/A-Brücke **142** ist über einen Kommunikationspfad **144** mit der Speicherbrücke **132** gekoppelt und ist auch mit Eingabevorrichtungen **146** gekoppelt. Die Zusatzkarte(n) **150** und der Netzwerkadapter **160** sind über einen Switch **148** miteinander gekoppelt, der wiederum mit der E/A-Brücke **142** gekoppelt ist.

[0019] Die Speicherbrücke **132** ist eine Hardware-Einheit, die Kommunikationen zwischen der CPU **110**, dem Systemspeicher **120** und dem Parallelverarbeitungsuntersystem **130** unter anderen Komponenten des Computersystems **100** ermöglicht. Zum Beispiel könnte Speicherbrücke **132** ein Northbridge-Chip sein. Der Kommunikationspfad **134** ist eine Datenverbindung mit hoher Geschwindigkeit und/oder hoher Bandbreite, die Kommunikationen mit geringer Latenz zwischen dem Parallelverarbeitungsuntersystem **130** und der Speicherbrücke **132** über eine oder mehrere separate Spuren hinweg ermöglicht. Der Kommunikationspfad **134** könnte z.B. eine PCIe-Verbindung (PCIe = Peripheral Component Interconnect Express), ein AGP (= Accelerated Graphics Port), ein HyperTransport oder jede andere technisch realisierbare Art von Kommunikationsbus sein.

[0020] Die E/A-Brücke **142** ist eine Hardware-Einheit, die Eingabe- und/oder Ausgabeoperationen erleichtert, die mit der Systemplatte **140**, den Eingabevorrichtungen **146**, der/den Zusatzkarte(n) **150**, dem Netzwerkadapter **160** und verschiedenen anderen Komponenten des Computersystems **100** durchgeführt werden. Zum Beispiel könnte die E/A-Brücke **142** ein Southbridge-Chip sein. Der Kommunikationspfad **144** ist eine Datenverbindung mit hoher Geschwindigkeit und/oder hoher Bandbreite, die Kommunikationen mit geringer Latenz zwischen der Speicherbrücke **132** und der E/A-Brücke **142** erleichtert. Der Kommunikationspfad **142** könnte zum Beispiel eine PCIe-Verbindung, ein AGP, ein HyperTransport oder jede andere technisch machbare Art von Kommunikationsbus sein. Mit der gezeigten Konfiguration kann jede Komponente, die entweder mit der Speicherbrücke **132** oder der E/A-Brücke **142** gekoppelt ist, mit jeder anderen Komponente kommunizieren, die entweder mit der Speicherbrücke **132** oder der E/A-Brücke **142** gekoppelt ist.

[0021] Die CPU **110** ist ein Prozessor, der dazu konfiguriert ist, den Gesamtbetrieb des Computersys-

tems **100** zu koordinieren. Dabei führt die CPU **110** Anweisungen aus, um Befehle an die verschiedenen anderen Komponenten auszugeben, die in dem Computersystem **100** enthalten sind. Die CPU **110** ist darüber hinaus dazu konfiguriert, Befehle auszuführen, um Daten zu verarbeiten, die von einer der anderen in dem Computersystem **100** enthaltenen Komponenten, einschließlich des Systemspeichers **120** und der Systemplatte **140**, erzeugt und/oder gespeichert werden. Der Systemspeicher **120** und die Systemplatte **140** sind Speichervorrichtungen, die computerlesbare Medien beinhalten, die dazu konfiguriert sind, Daten und Softwareanwendungen zu speichern. Der Systemspeicher **120** beinhaltet einen Gerätetreiber **122** und einen Hypervisor **124**, deren Funktionsweise nachstehend ausführlicher beschrieben wird. Das Parallelverarbeitungsuntersystem **130** beinhaltet eine oder mehrere Parallelverarbeitungseinheiten (PPUs), die dazu konfiguriert sind, über eine hochparallele Verarbeitungsarchitektur mehrere Operationen gleichzeitig auszuführen. Jede PPU beinhaltet eine oder mehrere Rechen-Engines, die allgemeine Rechenoperationen parallel ausführen, und/oder eine oder mehrere Grafik-Engines, die grafikorientierte Operationen parallel ausführen. Eine gegebene PPU kann dazu konfiguriert sein, Pixel für die Anzeige über die Anzeigevorrichtung **136** zu erzeugen. Eine beispielhafte PPU wird nachstehend in Verbindung mit den **Fig. 2-4** ausführlicher beschrieben.

[0022] Der Gerätetreiber **122** ist eine Software-Anwendung, die dann, wenn sie von der CPU **110** ausgeführt wird, als eine Schnittstelle zwischen der CPU **110** und dem Parallelverarbeitungsuntersystem **130** fungiert. Insbesondere ermöglicht es der Gerätetreiber **122** der CPU **110**, verschiedene Verarbeitungsoperationen auf das Parallelverarbeitungsuntersystem **130** zur hochparallelen Ausführung auszulagern, einschließlich allgemeiner Rechenoperationen sowie Grafikverarbeitungsoperationen. Der Hypervisor **124** ist eine Software-Anwendung, die dann, wenn sie von der CPU **110** ausgeführt wird, verschiedene Rechen-, Grafik- und Speicherressourcen, die im dem Parallelverarbeitungsuntersystem **130** enthalten sind, partitioniert, um getrennten Benutzern eine unabhängige Nutzung dieser Ressourcen bereitzustellen, wie nachstehend in Verbindung mit den **Fig. 5-10** näher beschrieben wird.

[0023] In verschiedenen Ausführungsformen können einige oder alle Komponenten des Computersystems **100** in einer Cloud-basierten Umgebung implementiert sein, die potenziell über ein großes geografisches Gebiet verteilt ist. Beispielsweise könnten verschiedene Komponenten des Computersystems **100** in geografisch weit auseinander liegenden Datenzentren eingesetzt sein. In solchen Ausführungsformen können die verschiedenen Komponenten des Computersystems **100** über ein oder mehrere Netzwer-

ke, einschließlich einer beliebigen Anzahl lokaler Intranets und/oder des Internets, miteinander kommunizieren. In verschiedenen anderen Ausführungsformen können bestimmte Komponenten des Computersystems **100** über ein oder mehrere virtualisierte Geräte implementiert sein. Zum Beispiel könnte die CPU **110** als eine virtualisierte Instanz einer Hardware-CPU implementiert sein. In einigen Ausführungsformen kann ein Teil oder die Gesamtheit des Parallelverarbeitungsuntersystems **130** mit einer oder mehreren anderen Komponenten des Computersystems **100** integriert sein, um einen einzigen Chip zu bilden, wie beispielsweise ein System-on-Chip (SoC).

[0024] Für den Fachmann versteht sich, dass die Architektur des Computersystems **100** flexibel genug ist, um in einem breiten Spektrum möglicher Szenarien und Anwendungsfälle implementiert werden zu können. Beispielsweise könnte das Computersystem **100** in einem Cloud-Computing-Center implementiert werden, um einem oder mehreren Benutzern allgemeine Rechenfähigkeiten und/oder allgemeine Grafikverarbeitungsfähigkeiten zur Verfügung zu stellen. Alternativ könnte das Computersystem **100** in einer Automobil-Implementierung eingesetzt werden, um Datenverarbeitungsoperationen in Zusammenhang mit Fahrzeugnavigation durchzuführen. Für den Fachmann versteht sich weiter, dass die verschiedenen Komponenten des Computersystems **100** und die Verbindungstopologie zwischen diesen Komponenten auf jede technisch machbare Weise modifiziert werden können, ohne vom Gesamtumfang und der Idee der vorliegenden Ausführungsformen abzuweichen.

[0025] **Fig. 2** ist ein Blockdiagramm einer PPU, die in dem Parallelverarbeitungsuntersystem von **Fig. 1** enthalten ist, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet eine PPU **200** eine E/A-Einheit **210**, eine Hostschnittstelle **220**, Sys-Pipes **230**, ein Verarbeitungsclusterarray **240**, eine Crossbar-Einheit bzw. Kreuzschieneneinheit **250** und eine Speicherschnittstelle **260**. Die PPU **200** ist mit einem PPU-Speicher **270** gekoppelt. Jede der gezeigten Komponenten kann über jede technisch machbare Art von Hardware und/oder jede technisch machbare Kombination von Hardware und Software implementiert sein.

[0026] Die E/A-Einheit **210** ist über den Kommunikationspfad **134** und die Speicherbrücke **132** mit der CPU **110** von **Fig. 1** gekoppelt. Die E/A-Einheit **210** ist darüber hinaus an die Hostschnittstelle **220** und an die Kreuzschieneneinheit **250** gekoppelt. Die Hostschnittstelle **220** ist mit einer oder mehreren physischen Kopier-Engines (PCEs) **222** gekoppelt, die ihrerseits mit einem oder mehreren PCE-Zählern **224** gekoppelt sind. Die Hostschnittstelle **220** ist darüber hinaus mit den Sys-Pipes **230** gekoppelt. Eine gege-

bene Sys-Pipe **230** beinhaltet ein Frontend **232**, eine Aufgaben-/Arbeits-Einheit **234** und einen Leistungsmonitor (PM) **236** und ist mit dem Verarbeitungsclusterarray **240** gekoppelt. Das Verarbeitungsclusterarray **240** beinhaltet Universalverarbeitungscluster (GPCs) **242(0)** bis **242(A)**, wobei A eine positive ganze Zahl ist. Das Verarbeitungsclusterarray **240** ist mit der Kreuzschieneneinheit **250** gekoppelt. Die Kreuzschieneneinheit **250** ist mit der Speicherschnittstelle **260** gekoppelt. Die Speicherschnittstelle **260** beinhaltet Partitionseinheiten **262(0)** bis **262(B)**, wobei B ein positiver ganzzahliger Wert ist. Jede Partitionseinheit **262** kann separat mit der Kreuzschieneneinheit **250** zugeordnet sein. Der PPU-Speicher **270** beinhaltet dynamische Direktzugriffsspeicher (DRAMs) **272(0)** bis **272(C)**, wobei C ein positiver ganzzahliger Wert ist. Um den gleichzeitigen Betrieb in mehreren Verarbeitungskontexten zu erleichtern, sind verschiedene Einheiten innerhalb der PPU **200** wie folgt repliziert: (a) die Hostschnittstelle **220** beinhaltet PB-DMAAs **520(0)** bis **520(7)**; (b) die Sys-Pipe **230** einschließlich der Sys-Pipe **230(0)** bis **230(7)**, so dass die Aufgaben-/Arbeits-Einheit **234** SKED **500(0)** bis **500(7)** entspricht und die Aufgaben-/Arbeits-Einheit **234** CWD **560(0)** bis **560(7)** entspricht.

[0027] Im Betrieb erhält die E/A-Einheit **210** verschiedene Arten von Befehlsdaten von der CPU **110** und verteilt diese Befehlsdaten zur Ausführung an relevante Komponenten der PPU **200**. Insbesondere erhält die E/A-Einheit **210** Befehlsdaten, die Verarbeitungsaufgaben zugeordnet sind, von der CPU **110** und leitet diese Befehlsdaten an die Hostschnittstelle **220** weiter. Die E/A-Einheit **210** erhält darüber hinaus Befehlsdaten, die Speicherzugriffsoperationen zugeordnet sind, von der CPU **110** und leitet diese Befehlsdaten an die Kreuzschieneneinheit **250** weiter. Befehlsdaten im Zusammenhang mit Verarbeitungsaufgaben beinhalten allgemein einen oder mehrere Zeiger auf Aufgabenmetadaten (TMD), die in einer Befehlswarteschlange innerhalb des PPU-Speichers **270** oder an anderer Stelle innerhalb des Computersystems **100** gespeichert sind. Ein bestimmtes TMD ist eine kodierte Verarbeitungsaufgabe, die Indizes der zu verarbeitenden Daten, die mit diesen Daten auszuführenden Operationen, die mit diesen Operationen zugeordneten Zustandsparameter, eine Ausführungspriorität und andere aufgabenorientierte Verarbeitungsinformationen beschreibt.

[0028] Die Hostschnittstelle **220** empfängt Befehlsdaten im Zusammenhang mit Verarbeitungsaufgaben von der E/A-Einheit **210** und verteilt diese Befehlsdaten dann über einen oder mehrere Befehlsströme an die Sys-Pipes **230**. In einigen Konfigurationen erzeugt die Hostschnittstelle **210** für jede unterschiedliche Sys-Pipe **230** einen anderen Befehlsstrom, wobei ein gegebener Befehlsstrom Zeiger auf TMDs enthält, die für eine entsprechende Sys-Pipe **230** relevant sind.

[0029] Eine gegebene Sys-Pipe **230** führt verschiedene Vorverarbeitungsoperationen mit empfangenen Befehlsdaten durch, um die Ausführung entsprechender Verarbeitungsaufgaben auf den GPCs **242** innerhalb des Verarbeitungsclusterarrays **240** zu erleichtern. Nach dem Empfang von Befehlsdaten, die einer oder mehreren Verarbeitungsaufgaben zugeordnet sind, erhält das Frontend **232** innerhalb der gegebenen Sys-Pipe **230** die zugehörigen Verarbeitungsaufgaben und leitet diese Verarbeitungsaufgaben an die Aufgaben-/Arbeits-Einheit **234** weiter. Die Aufgaben-/Arbeits-Einheit **234** konfiguriert eine oder mehrere GPCs **242** in einen Betriebszustand, der für die Ausführung der Verarbeitungsaufgaben geeignet ist, und überträgt dann die Verarbeitungsaufgaben zur Ausführung an diese GPCs **242**. Jede Sys-Pipe **230** kann Kopieraufgaben an eine oder mehrere PCEs **222** auslagern, die dedizierte Kopieroperationen durchführen. Die PCE-Zähler **224** verfolgen die Verwendung der PCEs **222**, um Arbeitslasten von Kopieroperationen zwischen verschiedenen Sys-Pipes **230** auszugleichen. Der PM **236** überwacht die Gesamtleistung und/oder den Ressourcenverbrauch der entsprechenden Sys-Pipe **230** und kann verschiedene von dieser Sys-Pipe **230** ausgeführte Operationen drosseln, um einen ausgeglichenen Ressourcenverbrauch über alle Sys-Pipes **230** hinweg zu halten.

[0030] Jeder GPC **242** beinhaltet mehrere parallele Verarbeitungskerne, die in der Lage sind, eine große Anzahl von Threads gleichzeitig und mit einem beliebigen Grad an Unabhängigkeit und/oder Isolation von anderen GPCs **242** auszuführen. Beispielsweise könnte ein gegebener GPC **242** Hunderte oder Tausende von Threads gleichzeitig in Verbindung mit oder isoliert von jedem anderen GPC **242** ausführen. Ein Satz gleichzeitiger Threads, die auf einem GPC **242** ausgeführt werden, kann separate Instanzen desselben Programms oder separate Instanzen verschiedener Programme ausführen. In einigen Konfigurationen werden die GPCs **242** von allen Sys-Pipes **230** gemeinsam genutzt, während in anderen Konfigurationen verschiedene Sätze von GPCs **242** für den Betrieb in Verbindung mit bestimmten Sys-Pipes **230** zugewiesen sind. Jeder GPC **242** empfängt Verarbeitungsaufgaben von einer oder mehreren Sys-Pipes **230** und startet im Ansprechen darauf einen oder mehrere Sätze von Threads, um diese Verarbeitungsaufgaben auszuführen und Ausgabedaten zu erzeugen. Nach Abschluss einer bestimmten Verarbeitungsaufgabe überträgt ein bestimmter GPC **242** die Ausgabedaten an einen anderen GPC **242** zur weiteren Verarbeitung oder an die Kreuzschieneneinheit **250** zur geeigneten Weiterleitung. Ein beispielhafter GPC wird nachstehend in Verbindung mit **Fig. 3** ausführlicher beschrieben.

[0031] Die Kreuzschieneneinheit **250** ist ein Schaltmechanismus, der verschiedene Arten von Daten zwischen der E/A-Einheit **210**, dem Verarbeitungs-

clusterarray **240** und der Speicherschnittstelle **260** routet. Wie vorstehend erwähnt wurde, überträgt die E/A-Einheit **210** Befehlsdaten im Zusammenhang mit Speicherzugriffsoperationen an die Kreuzschieneneinheit **250**. Als Antwort sendet die Kreuzschieneneinheit **250** die zugehörigen Speicherzugriffsoperationen zur Verarbeitung an die Speicherschnittstelle **260**. In einigen Fällen leitet die Kreuzschieneneinheit **250** auch Lesedaten, die von der Speicherschnittstelle **260** zurückgegeben werden, zurück an die Komponente, die die Lesedaten anfordert. Die Kreuzschieneneinheit **250** empfängt darüber hinaus Ausgabedaten von den GPCs **242**, wie vorstehend erwähnt wurde, und kann diese Ausgabedaten dann an die E/A-Einheit **210** zur Übertragung an die CPU **110** routen, oder kann diese Daten zur Speicherung und/oder Verarbeitung an die Speicherschnittstelle **260** routen. Die Kreuzschieneneinheit **250** ist allgemein dazu konfiguriert, Daten zwischen den GPCs **242** und von jedem GPC **242** an jede Partitionseinheit **262** zu routen. In verschiedenen Ausführungsformen kann die Kreuzschieneneinheit **250** virtuelle Kanäle implementieren, um Verkehrsströme zwischen den GPCs **242** und den Partitionseinheiten **262** zu trennen. In verschiedenen Ausführungsformen kann die Kreuzschieneneinheit **250** nicht gemeinsam genutzte Pfade zwischen einem Satz von GPCs **242** und einem Satz von Partitionseinheiten **262** ermöglichen.

[0032] Die Speicherschnittstelle **260** implementiert die Partitionseinheiten **262**, um einen Speicherzugriff mit hoher Bandbreite auf DRAMs **272** innerhalb des PPU-Speichers **270** bereitzustellen. Jede Partitionseinheit **262** kann Speicherzugriffsoperationen mit einem anderen DRAM **272** parallel zueinander durchführen und dadurch die verfügbare Speicherbandbreite des PPU-Speichers **270** effizient nutzen. Eine gegebene Partitionseinheit **262** stellt auch Caching-Unterstützung über einen oder mehrere interne Caches bereit. Eine beispielhafte Partitionseinheit **262** wird nachstehend in Verbindung mit **Fig. 4** ausführlicher beschrieben.

[0033] Der PPU-Speicher **270** allgemein und die DRAMs **272** im Besonderen können dazu konfiguriert sein, jeden technisch machbaren Datentyp zu speichern, der universellen Rechenanwendungen und/oder Grafikverarbeitungsanwendungen zugeordnet ist. Beispielsweise könnten DRAMs **272** große Matrizen von Datenwerten speichern, die neuronalen Netzwerken in Universalrechenanwendungen zugeordnet sind, oder alternativ einen oder mehrere Frame-Puffer speichern, die verschiedene Renderingziele in Grafikverarbeitungsanwendungen enthalten. In verschiedenen Ausführungsformen können die DRAMs **272** über jede technisch machbare Speichervorrichtung implementiert sein.

[0034] Die vorstehend dargelegte Architektur ermöglicht es der PPU **200**, eine breite Vielzahl

von Verarbeitungsvorgängen beschleunigt und asynchron zum Betrieb der CPU **110** durchzuführen. Insbesondere ermöglicht die parallele Architektur der PPU **200**, eine große Anzahl von Operationen parallel und mit einem beliebigen Grad an Unabhängigkeit voneinander und von auf der CPU **110** durchgeführten Operationen durchzuführen, wodurch die Gesamtleistung dieser Operationen beschleunigt wird.

[0035] In einer Ausführungsform kann die PPU **200** dazu konfiguriert sein, Universalrechenoperationen durchzuführen, um Berechnungen, die großen Datensätze involvieren, zu beschleunigen. Solche Datensätze können sich unter anderem auf finanzielle Zeitreihen, dynamische Simulationsdaten, Echtzeit-Sensorauslesungen, Gewichtsmatrizen und/oder Tensoren neuronaler Netzwerke und Parameter maschinellen Lernens beziehen. In einer anderen Ausführungsform kann die PPU **200** dazu konfiguriert sein, als Grafikverarbeitungseinheit (GPU) zu arbeiten, die eine oder mehrere Grafik-Rendering-Pipelines implementiert, um Pixeldaten auf der Grundlage von Grafikbefehlen zu erzeugen, die von der CPU **110** generiert werden. Die PPU **200** kann dann die Pixeldaten über die Anzeigevorrichtung **136** als einen oder mehrere Frames ausgeben. Der PPU-Speicher **170** kann dazu konfiguriert sein, als Grafikspeicher zu arbeiten, der einen oder mehrere Frame-Puffer und/oder ein oder mehrere Rendering-Ziele in ähnlicher Weise wie vorstehend erwähnt zu speichern. In einer weiteren Ausführungsform kann die PPU **200** dazu konfiguriert sein, sowohl Universalrechenoperationen als auch Grafikverarbeitungsoperationen gleichzeitig auszuführen. In solchen Konfigurationen können eine oder mehrere Sys-Pipes **230** dazu konfiguriert sein, über eine oder mehrere GPCs **242** allgemeine Rechenoperationen zu implementieren, und können eine oder mehrere andere Sys-Pipes **230** dazu konfiguriert sein, über eine oder mehrere GPCs **242** eine oder mehrere Grafikverarbeitungs-Pipelines zu implementieren.

[0036] Bei jeder der vorstehend genannten Konfigurationen arbeiten der Gerätetreiber **122** und der Hypervisor **124** zusammen, um verschiedene in der PPU **200** enthaltene Rechen-, Grafik- und Speicherressourcen in separate „PPU-Partitionen“ zu unterteilen. Alternativ kann es eine Vielzahl von Gerätetreibern **122** geben, die jeweils einer „PPU-Partition“ zugeordnet sind. Vorzugsweise werden Gerätetreiber auf einem Satz von Kernen in der CPU **110** ausgeführt. Eine gegebene PPU-Partition arbeitet im Wesentlichen ähnlich wie PPU **200** als Ganzes. Insbesondere kann jede PPU-Partition dazu konfiguriert sein, Universalrechenoperationen, Grafikverarbeitungsoperationen oder beide Arten von Operationen relativ isoliert von anderen PPU-Partitionen auszuführen. Darüber hinaus kann eine gegebene PPU-Partition dazu konfiguriert sein, mehrere Verarbeitungskontexte gleichzeitig zu implementieren,

wenn eine oder mehrere virtuelle Maschinen (VMs) gleichzeitig auf den der gegebenen PPU-Partition zugewiesenen Rechen-, Grafik- und Speicherressourcen ausgeführt werden. Logische Gruppierungen von PPU-Ressourcen in PPU-Partitionen werden nachstehend in Verbindung mit den **Fig. 5-8** ausführlicher beschrieben. Techniken zur Partitionierung und Konfiguration von PPU-Ressourcen werden nachstehend in Verbindung mit den **Fig. 9-10** ausführlicher beschrieben.

[0037] **Fig. 3** ist ein Blockdiagramm eines GPC, der in der PPU von **Fig. 2** enthalten ist, gemäß verschiedenen Ausführungsformen der Erfindung. Wie gezeigt ist, ist der GPC **242** mit einer Speicherverwaltungseinheit (MMU) **300** gekoppelt und beinhaltet einen Pipeline-Manager **310**, eine Arbeitsverteilungs-Kreuzschiene **320**, einen oder mehrere Texturverarbeitungs-Cluster (TPCs) **330**, eine oder mehrere Textureinheiten **340**, einen Level 1.5 (L1.5) Cache **350**, einen PM **360** und einen Pre-Raster-Operations-Prozessor (preROP) **370**. Der Pipeline-Manager **310** ist mit der Arbeitsverteilungs-Kreuzschiene **320** und den TPCs **330** gekoppelt. Jeder TPC **330** beinhaltet einen oder mehrere Streaming-Multiprozessoren (SMs) **332** und ist mit einer Textureinheit **340**, der MMU **300**, dem L1.5-Cache **350**, dem PM **360** und dem preROP **370** gekoppelt. Die Textureinheit **340** und der L1.5-Cache **350** sind darüber hinaus mit der MMU **300** und untereinander gekoppelt. Der PreROP **370** ist mit der Arbeitsverteilungs-Kreuzschiene **320** gekoppelt. Jede der gezeigten Komponenten kann über jede technisch machbare Art von Hardware und/oder jede technisch machbare Kombination von Hardware und Software implementiert sein.

[0038] Der GPC **242** ist mit einer hochparallelen Architektur konfiguriert, die die parallele Ausführung einer großen Anzahl von Threads unterstützt. Wie hierin in Bezug genommen, ist ein „Thread“ eine Instanz eines bestimmten Programms, das auf einem bestimmten Satz von Eingabedaten ausgeführt wird, um verschiedene Arten von Operationen durchzuführen, einschließlich allgemeiner Rechenoperationen und Grafikverarbeitungsoperationen. In einer Ausführungsform kann der GPC **242** SIMD (Single-Instruction Multiple-Data)-Techniken implementieren, um die parallele Ausführung einer großen Anzahl von Threads zu unterstützen, ohne notwendigerweise auf mehrere unabhängige Befehlseinheiten angewiesen zu sein.

[0039] In einer anderen Ausführungsform kann der GPC **242** SIMT (Single-Instruction Multiple-Thread)-Techniken implementieren, um die parallele Ausführung einer großen Anzahl von allgemein synchronisierten Threads über eine gemeinsame Befehlseinheit zu unterstützen, die Befehle an eine oder mehrere Verarbeitungs-Engines ausgibt. Für den Fach-

mann versteht sich, dass die SIMT-Ausführung es verschiedenen Threads ermöglicht, leichter divergierenden Ausführungspfaden durch ein gegebenes Programm zu folgen, im Gegensatz zur SIMD-Ausführung, bei der alle Threads allgemein nicht divergierenden Ausführungspfaden durch ein gegebenes Programm zu folgen. Der Fachmann wird erkennen, dass SIMD-Techniken eine funktionelle Teilmenge der SIMT-Techniken repräsentieren.

[0040] Der GPC **242** kann eine große Anzahl paralleler Threads über SMs **332** ausführen, die in TPCs **330** enthalten sind. Jeder SM **332** beinhaltet einen Satz von funktionellen Einheiten bzw. Funktionseinheiten (nicht gezeigt), einschließlich einer oder mehrerer Ausführungseinheiten und/oder einer oder mehrerer Lade-Speicher-Einheiten, die dazu konfiguriert sind, Anweisungen auszuführen, die empfangenen Verarbeitungsaufgaben zugeordnet sind. Eine gegebene Funktionseinheit kann Anweisungen in einer Pipeline-Weise ausführen, was bedeutet, dass eine Anweisung an die Funktionseinheit ausgegeben werden kann, bevor die Ausführung einer vorangehenden Anweisung abgeschlossen ist. In verschiedenen Ausführungsformen können die Funktionseinheiten innerhalb der SMs **332** dazu konfiguriert sein, eine Vielzahl verschiedener Operationen auszuführen, einschließlich von Ganzzahl- und Gleitkommaarithmetik (z.B. Addition und Multiplikation u.a.), Vergleichsoperationen, Boolesche Operationen (z.B. UND, ODER und XOR u.a.), Bitverschiebung und Berechnung verschiedener algebraischer Funktionen (z.B. planare Interpolation und trigonometrische, exponentielle und logarithmische Funktionen u.a.). Jede Funktionseinheit kann Zwischendaten innerhalb eines Level-1 (L1)-Caches speichern, der sich in dem SM **332** befindet.

[0041] Über die vorstehend beschriebenen Funktionseinheiten ist der SM **332** dazu konfiguriert, eine oder mehrere „Thread-Gruppen“ (auch „Warps“ genannt) zu verarbeiten, die gleichzeitig dasselbe Programm mit unterschiedlichen Eingangsdaten ausführen. Jeder Thread innerhalb einer Thread-Gruppe wird allgemein über eine andere Funktionseinheit ausgeführt, obwohl in manchen Situationen nicht alle Funktionseinheiten Threads ausführen. Falls z.B. die Anzahl von in der Thread-Gruppe enthaltenen Threads geringer ist als die Anzahl von Funktionseinheiten, könnten die nicht verwendeten Funktionseinheiten während der Verarbeitung der Thread-Gruppe untätig bleiben. In anderen Situationen werden mehrere Threads innerhalb einer Thread-Gruppe über dieselbe Funktionseinheit zu unterschiedlichen Zeiten ausgeführt. Falls z.B. die Anzahl von in der Thread-Gruppe enthaltenen Threads größer ist als die Anzahl der Funktionseinheiten, dann könnten eine oder mehrere Funktionseinheiten verschiedene Threads über aufeinander folgende Taktzyklen ausführen.

[0042] In einer Ausführungsform kann ein Satz verwandter Thread-Gruppen gleichzeitig in verschiedenen Phasen der Ausführung innerhalb des SMs **332** aktiv sein. Ein Satz verwandter Thread-Gruppen wird hierin als ein „kooperatives Threadarray“ (CTA) oder ein „Threadarray“ bezeichnet. Threads innerhalb desselben CTA oder Threads innerhalb verschiedener CTAs können allgemein Zwischendaten und/oder Ausgabedaten über einen oder mehrere L1-Caches, einschließlich der SMs **332**, des L1.5-Cache **350**, eines oder mehrerer L2-Caches, die von SMs **332** gemeinsam genutzt werden, oder über einen gemeinsam genutzten Speicher, einen globalen Speicher oder einen anderen Speichertyp, der sich auf einer beliebigen Speichervorrichtung in dem Computersystem **100** befindet, miteinander teilen. In einer Ausführungsform kann der L1.5-Cache **350** dazu konfiguriert sein, Anweisungen zwischenzuspeichern, die von Threads auszuführen sind, die auf SMs **332** ausgeführt werden.

[0043] Jedem Thread in einer gegebenen Thread-Gruppe oder CTA ist allgemein eine eindeutige Thread-Kennung (Thread-ID) zugewiesen, auf die der Thread während der Ausführung zugreifen kann. Die einem gegebenen Thread zugewiesene Thread-ID kann als ein eindimensionaler oder mehrdimensionaler numerischer Wert definiert sein. Das Ausführungs- und Verarbeitungsverhalten des gegebenen Threads kann abhängig von der Thread-ID variieren. Beispielsweise könnte der Thread auf der Grundlage der Thread-ID bestimmen, welcher Teil eines Eingangs- bzw. Eingabedatensatzes zu verarbeiten ist und/oder welcher Teil eines Ausgangs- bzw. Ausgabedatensatzes zu schreiben ist.

[0044] In einer Ausführungsform kann eine Sequenz von Pro-Thread-Anweisungen zumindest eine Anweisung enthalten, die ein kooperatives Verhalten zwischen einem gegebenen Thread und einem oder mehreren anderen Threads definiert. Beispielsweise könnte die Sequenz von Pro-Thread-Anweisungen eine Anweisung enthalten, die bei ihrer Ausführung den gegebenen Thread in einem bestimmten Ausführungszustand aussetzt, bis einige oder alle anderen Threads einen entsprechenden Ausführungszustand erreichen. In einem anderen Beispiel könnte die Sequenz von Pro-Thread-Anweisungen eine Anweisung enthalten, die bei ihrer Ausführung bewirkt, dass der angegebene Thread Daten in einem gemeinsamen Speicher speichert, auf den einige oder alle anderen Threads Zugriff haben. In einem weiteren Beispiel könnte die Sequenz von Pro-Thread-Anweisungen eine Anweisung enthalten, die bei ihrer Ausführung bewirkt, dass der betreffende Thread atomar Daten liest und aktualisiert, die in einem gemeinsamen Speicher gespeichert sind, auf den einige oder alle anderen Threads abhängig von den Thread-IDs dieser Threads Zugriff haben. In einem weiteren Beispiel könnte die Sequenz von Pro-

Thread-Anweisungen eine Anweisung enthalten, die bei ihrer Ausführung den gegebenen Thread veranlasst, eine Adresse in einem gemeinsamen Speicher auf der Grundlage einer entsprechenden Thread-ID zu berechnen, um Daten aus diesem gemeinsamen Speicher zu lesen. Mit den vorstehenden Synchronisierungstechniken kann ein erster Thread Daten an eine bestimmte Stelle in einem gemeinsamen Speicher schreiben und kann ein zweiter Thread diese Daten auf vorhersehbare Weise aus dem gemeinsamen Speicher lesen. Demgemäß können Threads dazu konfiguriert sein, eine Vielzahl von Datenfreigabemustern innerhalb einer bestimmten Thread-Gruppe oder eines bestimmten CTA oder über Threads in verschiedenen Thread-Gruppen oder verschiedenen CTAs hinweg implementieren. In verschiedenen Ausführungsformen beschreibt eine in der Programmiersprache CUDA (Compute Unified Device Architecture) geschriebene Software-anwendung das Verhalten und den Betriebsablauf von Threads, die auf dem GPC **242** ausgeführt werden, einschließlich aller vorstehend beschriebenen Verhaltensweisen und Operationen.

[0045] Im Betrieb koordiniert der Pipeline-Manager **310** allgemein die parallele Ausführung von Verarbeitungsaufgaben innerhalb des GPC **242**. Der Pipeline-Manager **310** empfängt Verarbeitungsaufgaben von der Aufgaben-/Arbeits-Einheit **234** und verteilt diese Verarbeitungsaufgaben an die TPCs **330** zur Ausführung über die SMs **332**. Eine bestimmte Verarbeitungsaufgabe ist in der Regel einem oder mehreren CTAs zugeordnet, die auf einem oder mehreren SMs **332** innerhalb eines oder mehrerer TPCs **330** ausgeführt werden können. In einer Ausführungsform kann eine bestimmte Aufgaben-/Arbeits-Einheit **234** eine oder mehrere Verarbeitungsaufgaben an GPC **242** verteilen, indem ein oder mehrere CTAs gestartet werden, die an einen oder mehrere bestimmte TPCs **330** gerichtet sind. Der Pipeline-Manager **310** kann das gestartete CTA von der Aufgaben-/Arbeits-Einheit **234** empfangen und das CTA an den entsprechenden TPC **330** zur Ausführung über eine oder mehrere in dem TPC **330** enthaltene SMs **332** übertragen. Während oder nach der Ausführung einer bestimmten Verarbeitungsaufgabe erzeugt jeder SM **332** Ausgabedaten und überträgt die Ausgabedaten abhängig von einer aktuellen Konfiguration und/oder Art der aktuellen Verarbeitungsaufgabe an verschiedene Stellen.

[0046] In Konfigurationen, die sich auf allgemeine Computer- oder Grafikverarbeitung beziehen, kann der SM **332** Ausgabedaten an die Arbeitsverteilungs-Kreuzschiene **320** übertragen, und die Arbeitsverteilungs-Kreuzschiene **320** leitet die Ausgabedaten dann zur weiteren Verarbeitung an einen oder mehrere GPCs **242** oder zur weiteren Weiterleitung an die Kreuzschieneneinheit **250** weiter. Die Kreuzschieneneinheit **250** kann die Ausgabedaten u.a. an ei-

nen L2-Cache, der in einer bestimmten Partitionseinheit **262** enthalten ist, an den PPU-Speicher **270** oder an den Systemspeicher **120** weiterleiten. Der Pipeline-Manager **310** koordiniert allgemein das Routing der Ausgabedaten, das von der Arbeitsverteilungskreuzschiene **320** durchgeführt wird, auf der Grundlage der mit diesen Ausgabedaten zugeordneten Verarbeitungsaufgaben.

[0047] In Konfigurationen, die für Grafikverarbeitung spezifisch sind, kann der SM **332** Ausgabedaten an die Textureinheit **340** und/oder den preROP **370** übertragen. In einigen Ausführungsformen kann der preROP **370** einige oder alle der Rasteroperationen implementieren, die in einer 3D-Grafik-API spezifiziert sind, in welchem Fall der preROP **370** einige oder alle der Operationen implementiert, die ansonsten über einen ROP **410** ausgeführt werden. Die Textureinheit **340** führt allgemein Texturabbildungsoperationen aus, darunter beispielsweise eine Bestimmung von Texturprobenpositionen, ein Lesen von Texturdaten und ein Filtern von Texturdaten. Der PreROP **370** führt allgemein rasterorientierte Operationen durch, darunter z. B. die Organisation von Pixelfarbdaten und die Durchführung von Optimierungen zur Farbmischung. Der PreROP **370** kann auch Adressübersetzungen durchführen und Ausgabedaten, die von SMs **332** empfangen wurden, an eine oder mehrere ROP (Raster Operation Processor)-Einheiten innerhalb Partitionseinheiten **262** senden.

[0048] In jeder der vorstehend genannten Konfigurationen überwachen ein oder mehrere PMs **360** die Leistung der verschiedenen Komponenten des GPC **242**, um Benutzern Leistungsdaten zur Verfügung zu stellen und/oder die Auslastung von Rechen-, Grafik- und/oder Speicherressourcen über Gruppen von Threads hinweg auszugleichen und/oder die Auslastung dieser Ressourcen mit der anderer GPC **242** auszugleichen. Ferner können in jeder der vorstehend genannten Konfigurationen der SM **332** und andere Komponenten innerhalb des GPC **242** Speicherzugriffsoperationen mit der Speicherschnittstelle **260** über die MMU **300** durchführen. Die MMU **300** schreibt allgemein Ausgabedaten in verschiedene Speicherbereiche und/oder liest Eingabedaten aus verschiedenen Speicherbereichen im Auftrag des GPC **242** und den darin enthaltenen Komponenten. Die MMU **300** ist dazu konfiguriert, virtuelle Adressen über einen Satz von Seitentabelleneinträgen (PTEs) und einen oder mehrere optionale Adressübersetzungs-Lookaside-Puffer (TLBs) in physische Adressen abgebildet werden. Die MMU **300** kann verschiedene Daten in dem L1.5-Cache **350** zwischenspeichern, einschließlich von gelesenen Daten, die von der Speicherschnittstelle **260** zurückgegeben wurden. In der gezeigten Ausführungsform ist die MMU **300** extern an den GPC **242** gekoppelt und kann potenziell mit anderen GPCs **242** gemeinsam genutzt werden. In anderen Ausführungs-

formen kann der GPC **242** eine dedizierte Instanz von MMU **300** beinhalten, die Zugriff auf eine oder mehrere Partitionseinheiten **262** bereitstellt, die in der Speicherschnittstelle **260** enthalten sind.

[0049] Fig. 4 ist ein Blockdiagramm einer Partitionseinheit **262**, die in der PPU **200** von Fig. 2 enthalten ist, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet die Partitionseinheit **262** einen L2-Cache **400**, eine Frame Buffer (FB)-DRAM-Schnittstelle **410**, einen Raster Operations Processor (ROP) **420** und einen oder mehrere PMs **430**. Der L2-Cache **400** ist zwischen der FB-DRAM-Schnittstelle **410**, dem ROP **420** und dem PM **430** gekoppelt.

[0050] Der L2-Cache **400** ist ein Lese-/Schreib-Cache, der Lade- und Speicheroperationen durchführt, die von der Kreuzschieneneinheit **250** und dem ROP **420** empfangen wurden. Der L2-Cache **400** gibt Lesefehler und dringende Rückschreibanforderungen zur Verarbeitung an die FB-DRAM-Schnittstelle **410** aus. Der L2-Cache **400** überträgt auch schmutzige Aktualisierungen zur opportunistischen Verarbeitung an die FB-DRAM-Schnittstelle **410**. In einigen Ausführungsformen überwachen die PMs **430** während des Betriebs die Auslastung des L2-Cache **400**, um die Speicherzugriffsbandbreite gerecht auf verschiedene GPCs **242** und andere Komponenten der PPU **200** aufzuteilen. Die FB-DRAM-Schnittstelle **410** hat eine direkte Schnittstelle zu einem spezifischen DRAM **272**, um Speicherzugriffsoperationen durchzuführen, einschließlich des Schreibens von Daten in den DRAM **272** und des Lesens von Daten aus dem DRAM **272**. In einigen Ausführungsformen ist der Satz von DRAMs **272** auf mehrere DRAM-Chips aufgeteilt, wobei Teile mehrerer DRAM-Chips jedem DRAM **272** entsprechen.

[0051] In Konfigurationen mit Bezug zu Grafikverarbeitung führt der ROP **420** Rasteroperationen zur Erzeugung von Grafikdaten durch. Der ROP **420** könnte beispielsweise Schablonenoperationen, z-Test-Operationen, Mischoperationen und Komprimierungs- und/oder Dekomprimierungsoperationen u.a. an z- oder Farbdaten durchführen. Der ROP **420** kann dazu konfiguriert sein, verschiedene Arten von Grafikdaten, einschließlich Pixeldaten, Grafikobjekten, Fragmentdaten usw. zu erzeugen. Der ROP **420** kann auch Grafikverarbeitungsaufgaben auf andere Recheneinheiten verteilen. In einer Ausführungsform beinhaltet jeder GPC **242** einen dedizierten ROP **420**, der Rasteroperationen im Namen des entsprechenden GPC **242** durchführt.

[0052] Für den Fachmann versteht sich, dass die in den Fig. 1-4 beschriebene Architektur den Schutzbereich der vorliegenden Ausführungsformen in keiner Weise beschränkt und dass die hierin offenbarten Techniken auf jeder ordnungsgemäß konfigurierten Verarbeitungseinheit, einschließlich, aber nicht

beschränkt auf, einer oder mehrerer CPUs, einer oder mehrerer Mehrkern-CPU's, einer oder mehrerer PPU's **200**, eines oder mehrerer GPC's **242**, einer oder mehrerer GPU's oder anderer Sonderverarbeitungseinheiten usw., ohne vom Schutzbereich und der Idee der vorliegenden Ausführungsformen abzuweichen.

Logische Gruppierungen von Hardwareressourcen

[0053] Fig. 5 ist ein Blockdiagramm verschiedener PPU-Ressourcen, die in der PPU von Fig. 2 enthalten sind, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhalten PPU-Ressourcen **500** die Sys-Pipes **230(0)** bis **230(7)**, eine Steuerungskreuzschiene und SMC-Arbiter **510**, einen PRI (Privileged Register Interface)-Hub **512**, die GPC's **242**, die Kreuzschieneneinheit **250** und den L2-Cache **400**. Der L2-Cache **400** ist hier als eine Sammlung von „L2-Cache-Scheiben“ bzw. „L2-Cache-Slices“ gezeigt ist, die jeweils einem anderen Bereich des DRAM **262** entsprechen. Die Sys-Pipes **230**, die GPC's **242** und der PRI-Hub **512** sind über eine Steuerungskreuzschiene und SMC-Arbiter **510** miteinander gekoppelt. Die GPC's **242** und einzelne Slices des L2-Cache **400** sind über die Kreuzschieneneinheit **250** miteinander gekoppelt. In dem hierin diskutierten Beispiel beinhaltet die PPU-Ressource **500** acht Sys-Pipes **230**, acht GPC's **242** und eine bestimmte Anzahl anderer Komponenten. Für den Fachmann versteht sich jedoch, dass die PPU-Ressource **500** jede beliebige technisch machbare Anzahl dieser Komponenten beinhalten kann.

[0054] Jede Sys-Pipe **230** beinhaltet allgemein PBDMAs **520** und **522**, einen Frontend-Kontext-Switch (FECS) **530**, ein Rechen (COMP)-Frontend (FE) **540**, einen Scheduler (SKED) **550** und einen CUDA-Arbeitsverteiler (CWD) **560**. Die PBDMAs **520** und **522** sind Hardware-Speichersteuereinrichtungen bzw. Hardware-Speicher-Controller, die Kommunikationen zwischen dem Gerätetreiber **122** und der PPU **200** verwalten. Der FECS **530** ist eine Hardware-Einheit, die Kontextwechsel verwaltet. Das Rechen-FE **540** ist eine Hardware-Einheit, die die Verarbeitung von Rechenaufgaben zur Ausführung vorbereitet. Der SKED **550** ist eine Hardware-Einheit, die Verarbeitungsaufgaben zur Ausführung plant. Der CWD **560** ist eine Hardware-Einheit, die dazu konfiguriert ist, ein oder mehrere Thread-Grids in eine Warteschlange zu stellen und an einen oder mehrere GPC's **242** zu senden, um eine oder mehrere Verarbeitungsaufgaben auszuführen. In einer Ausführungsform kann eine bestimmte Verarbeitungsaufgabe in einem CUDA-Programm spezifiziert werden. Über die vorstehend genannten Komponenten können die Sys-Pipes **230** dazu konfiguriert werden, allgemeine Rechenoperationen auszuführen und/oder zu verwalten.

[0055] Die Sys-Pipe **230(0)** beinhaltet ferner eine Grafik-Frontend (FE)-Einheit **542** (gezeigt als GFX FE **542**), einen Zustandsänderungs-Controller SCC **552** und Primitive- bzw. Stammfunktions-Verteiler Phase A / Phase B-Einheiten (PDA/PDB) **562**. Das Graphik FE **542** ist eine Hardware-Einheit, die Graphikverarbeitungsaufgaben zur Ausführung vorbereitet. Der SCC **552** ist eine Hardware-Einheit, die die Parallelisierung der Arbeit mit verschiedenen API-Zuständen (z.B. ein Shader-Programm, von einem Shader verwendete Konstanten und die Art und Weise, wie eine Textur abgetastet wird) verwaltet, um die geordnete Anwendung des API-Zustands aufrechtzuerhalten, auch wenn Grafik-Primitive bzw. Grafik-Stammfunktionen nicht der Reihe nach verarbeitet werden. Die PDA/PDB **562** ist eine Hardware-Einheit, die Grafik-Primitive (z.B. Dreiecke, Linien, Punkte, Vierecke, Netze usw.) an die GPC's **242** verteilt. Über diese zusätzlichen Komponenten kann die Sys-Pipe **230(0)** weiter dazu konfiguriert sein, Grafikverarbeitungsoperationen durchzuführen. In verschiedenen Ausführungsformen können einige oder alle Sys-Pipes **230** dazu konfiguriert sein, ähnliche Komponenten wie die Sys-Pipe **230(0)** zu beinhalten und daher in der Lage sein, entweder universelle Rechenoperationen oder Grafikverarbeitungsoperationen durchzuführen. Alternativ dazu können in verschiedenen anderen Ausführungsformen einige oder alle Sys-Pipes **230** dazu konfiguriert sein, ähnliche Komponenten wie die Sys-Pipes **230(1)** bis **230(7)** zu beinhalten und daher in der Lage sein, nur universelle Berechnungsoperationen auszuführen. Generell kann das Frontend **232** von Fig. 2 dazu konfiguriert sein, das Rechen-FE **540**, das Grafik-FE **542** oder sowohl das Rechen-FE **540** als auch das Grafik-FE **542** zu beinhalten. Demgemäß wird nachstehend in Bezug auf das Rechen-FE **540** und das Graphik-FE **542** oder beide allgemein auf das Frontend **232** verwiesen.

[0056] Die Steuerungskreuzschiene und SMC-Arbiter **510** erleichtern Kommunikationen zwischen den Sys-Pipes **230** und den GPC's **242**. In manchen Konfigurationen sind ein oder mehrere spezifische GPC's **242** programmierbar zugewiesen, um Verarbeitungsaufgaben im Namen einer spezifischen Sys-Pipe **230** durchzuführen. In solchen Konfigurationen sind die Steuerungskreuzschiene und SMC-Arbiter **510** dazu konfiguriert, Daten zwischen einem oder mehreren beliebigen GPC(s) **242** und der/den entsprechenden Sys-Pipe(s) **230** zu routen. Der PRI-Hub **512** stellt durch die CPU **110** und/oder die PPU **200**-Einheiten Zugriff auf einen Satz privilegierter Register bereit, um die Konfiguration der PPU **200** zu steuern. Der Registeradressraum mit der PPU **200** kann durch ein PRI-Register konfiguriert werden, und dabei wird der PRI-Hub **512** dazu verwendet, die Zuordnung bzw. Abbildung von PRI-Registeradressen zwischen einem generischen PRI-Adressraum und einem für jede Sys-Pipe **230** separat definierten PRI-Adress-

raum zu konfigurieren. Diese PRI-Adressraum-Konfiguration ermöglicht eine Übertragung an mehrere PRI-Register von SMC-Engines, wie unten in Verbindung mit **Fig. 7** beschrieben wird. Die GPCs **242** schreiben Daten in den L2-Cache **400** und Lesen Daten aus dem L2-Cache über die Kreuzschieneneinheit **250** in der zuvor beschriebenen Weise. In manchen Konfigurationen ist jedem GPC **242** ein separater Satz von L2-Slices zugewiesen, die von dem L2-Cache **400** abgeleitet sind, und kann jeder beliebige GPC **242** Schreib-/Leseoperationen mit dem entsprechenden Satz von L2-Slices durchführen.

[0057] Jede der vorstehend diskutierten PPU-Ressourcen **500** kann logisch gruppiert oder in eine oder mehrere PPU-Partitionen unterteilt sein, die jeweils in gleicher Weise wie die PPU **200** als Ganzes funktionieren. Insbesondere kann eine bestimmte PPU-Partition mit ausreichenden Rechen-, Grafik- und Speicherressourcen konfiguriert sein, um jede technisch machbare Operation durchzuführen, die von der PPU **200** durchgeführt werden kann. Ein Beispiel dafür, wie PPU-Ressourcen **500** logisch in Partitionen gruppiert werden können, wird nachstehend in Verbindung mit **Fig. 6** ausführlicher beschrieben.

[0058] **Fig. 6** ist ein Beispiel dafür, wie der Hypervisor von **Fig. 1** PPU-Ressourcen logisch in einen Satz von PPU-Partitionen gruppiert, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhalten PPU-Partitionen **600** eine oder mehrere PPU-Slices **610**. Insbesondere beinhaltet eine PPU-Partition **600(0)** PPU-Slices **610(0)** bis **610(3)**, beinhaltet eine PPU-Partition **600(4)** PPU-Slices **610(4)** und **610(5)**, beinhaltet eine PPU-Partition **600(6)** einen PPU-Slice **610(6)** und beinhaltet eine PPU-Partition **600(7)** einen PPU-Slice **610(7)**. In dem hier diskutierten Beispiel beinhalten die PPU-Partitionen **600** die spezifische Anzahl der PPU-Slices **610** wie gezeigt. In anderen Konfigurationen können die PPU-Partitionen **600** jedoch eine andere Anzahl von PPU-Slices **610** beinhalten.

[0059] Jeder PPU-Slice **610** beinhaltet verschiedene Ressourcen, die von einer Sys-Pipe **230** abgeleitet sind, darunter die PBDMAAs **520** und **522**, der FECS **530**, das Frontend **232**, der SKED **550** und der CWD **560**. Jeder PPU-Slice **610** beinhaltet ferner außerdem einen GPC **242**, einen Satz L2-Slices **620** und entsprechende Teile von DRAM **272** (hier nicht gezeigt). Die verschiedenen Ressourcen, die in einem bestimmten PPU-Slice **610** enthalten sind, verleihen ihr eine ausreichende Funktionalität dahingehend, dass ein bestimmter PPU-Slice **610** zumindest einige der universellen Rechen- und/oder Grafikverarbeitungsvorgänge ausführen kann, die die PPU **200** ausführen kann.

[0060] Beispielsweise könnte ein PPU-Slice **610** Verarbeitungsaufgaben über das Frontend **232** emp-

fangen und dann diese Verarbeitungsaufgaben zur Ausführung über den SKED **550** einplanen. Der CWD **560** könnte dann Gitter von Threads ausgeben, um diese Verarbeitungsaufgaben auf dem GPC **242** auszuführen. Der GPC **242** könnte zahlreiche Thread-Gruppen parallel in der vorstehend in Verbindung mit **Fig. 3** beschriebenen Weise ausführen. Die PBDMAAs **520** und **522** könnten Speicherzugriffsoperationen für die verschiedenen in dem PPU-Slice **610** enthaltenen Komponenten ausführen. In einigen Ausführungsformen holen die PBDMAAs **520** und **522** Befehle aus dem Speicher und senden die Befehle zur Verarbeitung an das FE **232**. Bedarfsabhängig könnten verschiedene Komponenten des PPU-Slices **610** Daten in den entsprechenden Satz von L2-Cache-Slices **620** schreiben und daraus lesen. Die Komponenten des PPU-Slices **610** könnten bei Bedarf auch mit externen, in der PPU **200** enthaltenen Komponenten zugeordnet werden, u.a. mit der E/A-Einheit **210** und/oder den PCEs **222**. Der FECS **530** könnte Kontextwechseloperationen durchführen, wenn eine oder mehrere VMs auf den verschiedenen in dem PPU-Slice **610** enthaltenen Ressourcen zeitgeschnitten werden.

[0061] In der gezeigten Ausführungsform beinhaltet jeder PPU-Slice **610** Ressourcen, die von einer Sys-Pipe **230** abgeleitet sind, die dazu konfiguriert ist, allgemeine Rechenoperationen zu koordinieren. Demgemäß sind die PPU-Slices **610** dazu konfiguriert, nur Allzweck-Verarbeitungsaufgaben auszuführen. In anderen Ausführungsformen kann jeder PPU-Slice **610** jedoch auch Ressourcen enthalten, die von einer Sys-Pipe **230** abgeleitet sind, die dazu konfiguriert ist, Grafikverarbeitungsvorgänge zu koordinieren, wie z.B. die Sys-Pipe **230(0)**. In diesen Ausführungsformen können die PPU-Slices **610** dazu konfiguriert sein, zusätzlich Grafikverarbeitungsaufgaben auszuführen.

[0062] Allgemein ist jede PPU-Partition **600** eine harte Partitionierung von Ressourcen, die einem oder mehreren Benutzern eine dedizierte parallele Computerumgebung bereitstellt, die von anderen PPU-Partitionen **600** isoliert ist. Eine bestimmte PPU-Partition **600** beinhaltet, wie gezeigt ist, eine oder mehrere dedizierte PPU-Slices **610**, die gemeinsam die verschiedenen universellen bzw. allgemeinen Rechen-, Grafikverarbeitungs- und Speicherressourcen bereitstellen, die erforderlich sind, um zumindest bis zu einem gewissen Grad die übergreifende Funktionalität der PPU **200** als Ganzes nachzuahmen. Demgemäß kann ein bestimmter Benutzer parallele Verarbeitungsvorgänge innerhalb einer bestimmten PPU-Partition **600** in der gleichen Weise ausführen wie ein Benutzer, der dieselben parallelen Verarbeitungsvorgänge auf der PPU **200** ausführt, wenn die PPU **200** nicht partitioniert ist. Jede PPU-Partition **600** ist gegenüber anderen PPUs **600** fehlerunempfindlich, und jede PPU-Partition kann unabhängig von und oh-

ne Unterbrechung des Betriebsablaufs anderer PPU-Partitionen **600** zurückgesetzt werden. Verschiedene Ressourcen, die hier nicht spezifisch gezeigt werden, werden im Verhältnis zur Größe dieser verschiedenen PPU-Partitionen **600** gerecht auf die verschiedenen PPU-Partitionen **600** verteilt, wie nachstehend ausführlicher beschrieben wird.

[0063] In der hierin diskutierten Beispielkonfiguration von PPU-Partitionen **600** werden der PPU-Partition **600(0)** vier von acht PPU-Slices **610** zugewiesen, so dass diese mit der Hälfte der PPU-Ressourcen **500**, einschließlich verschiedener Arten von Bandbreite, wie z.B. Speicherbandbreite, ausgestattet wird. Dementsprechend wäre die PPU-Partition **610(0)** darauf beschränkt, die Hälfte der verfügbaren Systemspeicherbandbreite, die Hälfte der verfügbaren PPU-Speicherbandbreite, die Hälfte der verfügbaren PCE 212-Bandbreite usw. zu verbrauchen. In ähnlicher Weise werden der PPU-Partition **600(4)** zwei von acht PPU-Slices **610** zugewiesen, so dass sie mit einem Viertel der PPU-Ressourcen **500** ausgestattet ist. Dementsprechend wäre die PPU-Partition **610(4)** darauf beschränkt, ein Viertel der verfügbaren Systemspeicherbandbreite, ein Viertel der verfügbaren PPU-Speicherbandbreite, ein Viertel der verfügbaren PCE 212-Bandbreite usw. zu verbrauchen. Die anderen PPU-Partitionen **600(6)** und **600(7)** würden in analoger Weise eingeschränkt werden. Der Fachmann wird verstehen, wie die vorstehend diskutierte beispielhafte Partitionierung und die damit zugeordnete Ressourcenbereitstellung mit jeder anderen technisch machbaren Konfiguration von PPU-Partitionen **600** umgesetzt werden kann.

[0064] In einigen Ausführungsformen führt jede PPU-Partition **600** Kontexte für eine virtuelle Maschine (VM) aus. In einer Ausführungsform kann die PPU **200** verschiedene Leistungsmonitore und Drosselungszähler implementieren, die die Menge der lokalen und/oder systemweiten Ressourcen aufzeichnen, die von jeder PPU-Partition **600** verbraucht werden, um einen proportionalen Ressourcenverbrauch über alle PPU-Partitionen **600** hinweg aufrechtzuerhalten. Die Zuweisung des entsprechenden Anteils der PPU-Speicherbandbreite an eine PPU-Partition **600** kann durch die Zuweisen desselben Anteils von L2-Slices **400** an die PPU-Partition **600** erreicht werden.

[0065] Allgemein können PPU-Partitionen **600** dazu konfiguriert sein, funktionell voneinander isoliert zu arbeiten. Wie hierin erwähnt wurde, bedeutet der Begriff „funktionelle Isolierung“, wie er auf einen Satz von PPU-Partitionen **600** angewendet wird, allgemein, dass jede PPU-Partition **600** eine oder mehrere Operationen unabhängig von den Operationen durchführen kann, die von einer anderen PPU-Partition **600** im Satz von PPU-Partitionen **600** durchge-

führt werden, ohne diese zu stören und ohne von ihr gestört zu werden.

[0066] Eine bestimmte PPU-Partition **600** kann dazu konfiguriert sein, Verarbeitungsaufgaben, die mehreren Verarbeitungskontexten zugeordnet sind, gleichzeitig auszuführen. Der Begriff „Verarbeitungskontext“ oder „Kontext“ bezieht sich allgemein auf den Zustand von Hardware-, Software- und/oder Speicherressourcen während der Ausführung eines oder mehrerer Threads und entspricht allgemein einem Prozess auf der CPU **110**. Die einer bestimmten PPU-Partition **600** zugeordneten mehreren Verarbeitungskontexte können verschiedene Verarbeitungskontexte oder verschiedene Instanzen desselben Verarbeitungskontexts sein. Bei einer solchen Konfiguration werden spezifische PPU-Ressourcen, die der bestimmten PPU-Partition **600** zugewiesen sind, logisch in separate „SMC-Engines“ bzw. „SMC-Maschinen“ gruppiert, die separate Verarbeitungsaufgaben ausführen, die separaten Verarbeitungskontexten zugeordnet sind, wie nachstehend in Verbindung mit **Fig. 7** näher beschrieben wird. Ein gegebener Verarbeitungskontext könnte beispielsweise Hardware-Einstellungen, Pro-Thread-Anweisungen und/oder mit Threads zugeordnete Registerinhalte umfassen, die innerhalb einer SMC-Engine **700** ausgeführt werden.

[0067] **Fig. 7** veranschaulicht ein Beispiel dafür, wie der Hypervisor von **Fig. 1** einen Satz von PPU-Partitionen konfiguriert, um eine oder mehrere SMC (Simultaneous Multiple Context)-Engines zu implementieren, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhalten die PPU-Partitionen **600** eine oder mehrere SMC-Engines **700**. Insbesondere beinhaltet die PPU-Partition **600(0)** die SMC-Engines **700(0)** und **700(2)**, die PPU-Partition **600(4)** die SMC-Engine **700(4)**, die PPU-Partition **600(6)** die SMC-Engine **700(6)** und die PPU-Partition **600(7)** die SMC-Engine **700(7)**. Jede SMC-Engine **700** kann dazu konfiguriert sein, einen oder mehrere Verarbeitungskontexte auszuführen und/oder dazu konfiguriert sein, eine oder mehrere mit einem bestimmten Verarbeitungskontext zugeordnete Verarbeitungsaufgaben auszuführen, in gleichartiger Weise wie die PPU **200** als Ganzes.

[0068] Eine bestimmte SMC-Engine **700** beinhaltet allgemein Rechen- und Speicherressourcen, die zumindest einem PPU-Slice **610** zugeordnet sind. Beispielsweise beinhalten die SMC-Engines **700(6)** und **700(7)** die Rechen- und Speicherressourcen, die den PPU-Slices **610(6)** bzw. **610(7)** zugeordnet sind. Jede SMC-Engine **700** beinhaltet auch einen Satz von Virtuellengine-Identifikatoren (VEIDs) **702**, die lokal auf einen oder mehrere Unterkontexte verweisen, wobei ein VEID mit einem Virtualladressraum-Identifikator assoziiert ist und mit diesem identisch sein kann, der zur Auswahl eines virtuellen Adress-

raums verwendet wird, wobei die Seiten der virtuellen Adressräume durch Seitentabellen beschrieben werden, die von der MMU1600 verwaltet werden. Eine bestimmte SMC-Engine **700** kann darüber hinaus Rechen- und Speicherressourcen beinhalten, die mehreren PPU-Slices **610** zugeordnet sind. Zum Beispiel beinhaltet die SMC-Engine **700(0)** die Rechenressourcen, die den PPU-Slices **610(0)** und **610(1)** zugeordnet sind, verwendet aber nicht die Sys-Pipe **230(1)**. Die SMC-Engine **700()** beinhaltet und verwendet die L2-Slices in vier PPU-Slices **610(0)**, **610(1)**, **610(2)** und **610(3)**. In einigen Ausführungsformen teilen sich die SMC-Engines **700** innerhalb derselben PPU-Partition **600** die L2-Slices innerhalb der PPU-Partition **600**. In dieser Konfiguration ist die Sys-Pipe **230(1)** der PPU-Partition **600(1)**, wie gezeigt ist, unbenutzt, da eine SMC-Engine in der Regel jeweils nur einen Verarbeitungskontext ausführt und nur eine Sys-Pipe **230** für einen Verarbeitungskontext benötigt wird. Die SMC-Engine **700(2)** ist in ähnlicher Weise konfiguriert wie die SMC-Engine **700(0)**. Die in einer bestimmten PPU-Partition **600** enthaltenen Speicherressourcen, die einer oder mehreren SMC-Engines **700** innerhalb dieser bestimmten PPU-Partition **700** zugewiesen und/oder auf diese verteilt werden können, sind als PPU-Speicherpartitionen **710** gezeigt.

[0069] Eine bestimmte PPU-Speicherpartition **710** beinhaltet den Satz von L2-Slices, der in der PPU-Partition **600** enthalten ist, und entsprechende Teile von DRAM **272**. Allgemein teilen sich mehrere SMC-Engines **700** eine PPU-Speicherpartition **710**, falls diese SMC-Engines **700** in der gleichen PPU-Partition **600** enthalten sind. Die Zuweisungen an jede SMC-Engine **700** werden den auf diesen SMC-Engines **700** laufenden Kontexten zur Verfügung gestellt, und die Zuweisungen innerhalb der PPU-Speicherpartition **710** werden basierend auf Seiten implementiert.

[0070] Jede SMC-Engine **700** kann dazu konfiguriert sein, Verarbeitungsaufgaben, die einem bestimmten Verarbeitungskontext zugeordnet sind, unabhängig voneinander auszuführen. Demgemäß kann die PPU-Partition **600(0)**, die zwei SMC-Engines **700(0)** und **700(2)** aufweist, dazu konfiguriert sein, Verarbeitungsaufgaben, die zwei getrennten Verarbeitungskontexten zugeordnet sind, zu jedem beliebigen Zeitpunkt gleichzeitig auszuführen. Die PPU-Partitionen **600(4)**, **600(6)** und **600(7)** hingegen, die jeweils eine SMC-Engine **700(4)**, **700(6)** bzw. **700(7)** beinhalten, können dazu konfiguriert sein, Verarbeitungsaufgaben, die jeweils einem Verarbeitungskontext zugeordnet sind, gleichzeitig auszuführen. In einigen Ausführungsformen können Kontexte, die auf SMC-Engines **700** in verschiedenen PPU-Partitionen **600** laufen, Daten gemeinsam nutzen, indem sie eine oder mehrere Seiten in einer oder beiden PPU-Partitionen **600** gemeinsam nutzen.

[0071] Jede beliebige SMC-Engine **700** kann ferner dazu konfiguriert sein, verschiedene Verarbeitungskontexte über verschiedene Zeitintervalle in Zeitschnitten zu unterteilen. Demgemäß kann jede SMC-Engine **700** unabhängig die Ausführung von Verarbeitungsaufgaben unterstützen, die mehreren Verarbeitungskontexten zugeordnet sind, wenn auch nicht unbedingt gleichzeitig. Beispielsweise könnte die SMC-Engine **700(6)** vier verschiedene Verarbeitungskontexte über vier verschiedene Zeitintervalle zeitlich aufteilen und es dadurch diesen vier Verarbeitungskontexten zugeordneten Verarbeitungsaufgaben erlauben, innerhalb der PPU-Partition **600(6)** ausgeführt zu werden. In einigen Ausführungsformen werden VMs auf einer oder mehreren der PPU-Partitionen **600** in Zeitintervalle unterteilt. Beispielsweise kann die PPU-Partition **600(0)** zwischen zwei VMs in Zeitintervalle unterteilen, wobei jede VM gleichzeitig zwei Verarbeitungskontexte ausführt, einen auf jeder SMC-Engine **700(0)** und **700(1)**. In diesen Ausführungsformen wird es bevorzugt, alle Verarbeitungskontexte aus einer ersten VM aus dem Kontext zu schalten, bevor in den Verarbeitungskontexten von einer zweiten VM kontextumgeschaltet wird, welches von Vorteil ist, wenn sich die auf der PPU-Partition **600(0)** laufenden Verarbeitungskontexte die L2-Slices **400** teilen, die sich innerhalb der PPU-Partition **600(0)** befinden.

[0072] In einer Ausführungsform kann eine bestimmte VM mit einer GPU-Funktions-ID (GFID) assoziiert sein. Eine bestimmte GFID kann ein oder mehrere Bits beinhalten, die einer Physischen Funktion (PF) entsprechen, die Hardware zugeordnet ist, auf der die VM ausgeführt wird. Die bestimmte GFID kann auch einen Satz von Bits beinhalten, der einer Virtuellen Funktion (VF) entspricht, die der VM eindeutig zugeordnet ist. Eine bestimmte GFID kann u. a. dazu verwendet werden, Fehler an ein entsprechendes Gastbetriebssystem einer VM zu routen.

[0073] SMC-Engines **700** innerhalb verschiedener PPU-Partitionen **600** arbeiten in der Regel isoliert voneinander, weil, wie bereits diskutiert wurde, jede PPU-Partition **600** eine harte Partitionierung von PPU-Ressourcen **500** ist. Mehrere SMC-Engines **700** innerhalb derselben PPU-Partition **600** können allgemein unabhängig voneinander arbeiten und insbesondere unabhängig voneinander den Kontext wechseln. Beispielsweise könnte die SMC-Engine **700(0)** innerhalb der PPU-Partition **600(0)** unabhängig und asynchron relativ zu der SMC-Engine **700(2)** kontextwechselnd arbeiten. In einigen Ausführungsformen können mehrere SMC-Engines **700** innerhalb derselben PPU-Partition **600** den Kontextwechsel synchronisieren, um bestimmte Betriebsarten, wie beispielsweise eine Unterteilung in Zeitscheiben zwischen zwei VMs, zu unterstützen.

[0074] Allgemein arbeiten der Gerätetreiber **122** und der Hypervisor **124** von **Fig. 1** zusammen, um die PPU **200** in der bisher beschriebenen Weise zu partitionieren. Ferner arbeiten der Gerätetreiber **122** und der Hypervisor **124** zusammen, um jede PPU-Partition **600** in eine oder mehrere SMC-Engines **700** zu konfigurieren. Dabei konfigurieren der Gerätetreiber **122** und der Hypervisor **124** die DRAMs **272** und/oder den L2-Cache **400**, um den Satz von L2-Slices in Gruppen zu unterteilen, die jeweils eine SMC-Speicherpartition **710** sind, wie nachstehend in Verbindung mit **Fig. 8** näher beschrieben wird. In einigen Ausführungsformen reagiert der Hypervisor **124** auf Steuerungen durch einen Systemadministrator, so dass der Systemadministrator Konfigurationen von PPU-Partitionen erstellen kann. Diese PPU-Partitionen **600** werden an das Gastbetriebssystem **916** einer VM übergeben, und das Gastbetriebssystem **916** sendet anschließend Anfragen an den Hypervisor **124**, um eine zugehörige PPU-Partition **600** in eine oder mehrere SMC-Engines **700** zu konfigurieren. In einigen Ausführungsformen kann das Gastbetriebssystem die SMC-Engines **700** direkt innerhalb einer PPU-Partition **700** konfigurieren, weil eine ausreichende Isolierung hinzugefügt ist, um zu verhindern, dass ein Gastbetriebssystem die PPU-Partition **700** eines anderen Gastbetriebssystems beeinträchtigt.

[0075] **Fig. 8A** ist eine detailliertere Darstellung des DRAMs von **Fig. 7**, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, ist der DRAM **272**, der jeden der DRAMs **272(0)** bis **272(7)** von **Fig. 7** beinhaltet, über L2-Slices **800** zugänglich. Jeder L2-Slice **800** entspricht einem anderen Teil des L2-Cache **400** und ist für den Zugriff auf eine entsprechende Untergruppe von Speicherorten innerhalb des DRAM **272** konfiguriert. Allgemein entspricht die Partitionierung des DRAM **272** einem 2D-Rohadressraum, der in ähnlicher Weise organisiert ist wie der hier gezeigte DRAM **272**.

[0076] Wie ebenfalls gezeigt ist, ist der DRAM **272** in einen oberen Abschnitt **810**, einen partitionierbaren Abschnitt **820** und einen unteren Abschnitt **830** unterteilt. Der obere Abschnitt **810** und der untere Abschnitt **830** sind Speicherausschnitte bzw. Speicher-Carve-Outs, die von den oberen bzw. unteren Abschnitten aller DRAMs **272(0)** bis **272(7)** abgeleitet sind. Der Gerätetreiber **122**, der Hypervisor **124** und andere Einheiten bzw. Entitäten auf Systemebene haben Zugriff auf den oberen Abschnitt **810** und/oder den unteren Abschnitt **830**, und in einigen Ausführungsformen sind diese Abschnitte für PPU-Partitionen **600** nicht zugänglich. Der partitionierbare Abschnitt **820** hingegen ist für die Verwendung durch PPU-Partitionen **600** allgemein und SMC-Engines **700** im Besonderen vorgesehen. In einigen Ausführungsformen befinden sich sichere Daten in dem oberen Abschnitt **810** oder in dem unteren Abschnitt

830 und sind für alle PPU-Partitionen **600** zugänglich. In einigen Ausführungsformen werden der obere Abschnitt **810** oder der untere Abschnitt **830** für Hypervisor-Daten verwendet, auf die die VMs keinen Zugriff haben.

[0077] In der gezeigten beispielhaften Speicherpartitionierung beinhaltet der partitionierbare Abschnitt **820** den DRAM-Abschnitt **822(0)**, der der PPU-Speicherpartition **710(0)** innerhalb der PPU-Partition **600** (0) entspricht, den DRAM-Abschnitt **822(4)**, der der PPU-Speicherpartition **710(4)** innerhalb der PPU-Partition **600(2)** entspricht, den DRAM-Abschnitt **822(6)**, der der PPU-Speicherpartition **710(6)** innerhalb der PPU-Partition **600(6)** entspricht, und den DRAM-Abschnitt **822(7)**, der der PPU-Speicherpartition **710(7)** innerhalb der PPU-Partition **600(7)** entspricht. Jeder DRAM-Abschnitt **822** entspricht dem mittleren Teil der Adressen, die einem Satz von L2-Cache-Slices **800** entsprechen. Ein bestimmter DRAM-Abschnitt **822** kann weiter unterteilt werden, um separate Sätze von L2-Cache-Slices für verschiedene VMs bereitzustellen, die unterschiedlichen Verarbeitungskontexten zugeordnete Verarbeitungsaufgaben ausführen. Beispielsweise könnte der DRAM-Abschnitt **822(4)** in zwei oder mehr Bereiche unterteilt werden, um zwei oder mehr VMs zu unterstützen, die Verarbeitungsaufgaben ausführen, die zwei oder mehr Verarbeitungskontexten zugeordnet sind. Sobald ein DRAM-Abschnitt konfiguriert und in Gebrauch ist, wird er allgemein von jeweils einer VM verwendet, die auf einer PPU-Partition **600** läuft.

[0078] Im Betrieb führen der Gerätetreiber **122** und der Hypervisor **124** Speicherzugriffsoperationen innerhalb des oberen Abschnitts **810** und/oder des unteren Abschnitts **830** über obere Teile und untere Teile von Adressbereichen entsprechend allen L2-Cache-Slices **800** auf relativ ausgewogene Weise durch, wodurch Speicherbandbreite über jeden L2-Slice **800** proportional bestraft wird. In einigen Ausführungsformen führen die SMC-Engines **700** Speicherzugriffsoperationen auf den Systemspeicher **120** über die L2-Cache-Slices **800** mit einem Durchsatz durch, der von den Drosselzählern **840** gesteuert wird. Jeder Drosselzähler **840** überwacht die Speicherbandbreite, die verbraucht wird, wenn die SMC-Engines **700** über die L2-Cache-Slices **800**, die den entsprechenden PPU-Speicherpartitionen **710** zugeordnet sind, auf den Systemspeicher **120** zugreifen, um für jede PPU-Partition **600** eine proportionale Speicherbandbreite bereitzustellen. Wie vorstehend diskutiert wurde, erhalten die PPU-Partitionen **600** Zugriff auf verschiedene systemweite Ressourcen im Verhältnis zu der Konfiguration dieser PPU-Partitionen **600**. In dem gezeigten Beispiel wird der PPU-Partition **600(0)** eine Hälfte der PPU-Ressourcen **500** zugewiesen, und damit eine Hälfte des partitionierbaren Abschnitts **820** (gezeigt als DRAM-Abschnitt **822(0)**) und dementsprechend eine Hälfte der verfüg-

baren Speicherbandbreite dem Systemspeicher **120** zugewiesen. Die Partitionierung des DRAM **272** wird nachstehend in Verbindung mit den **Fig. 19-24** ausführlicher beschrieben.

[0079] **Fig. 8B** veranschaulicht, wie die verschiedenen DRAM-Abschnitte von **Fig. 8B** adressiert werden, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet ein eindimensionaler (1D) SPA (System Physical Address)-Raum **850** die obere Adressen **852**, die dem oberen Abschnitt **810** entsprechen, partitionierbare Adressen **854**, die in Adressbereiche **856** unterteilt sind und den DRAM-Abschnitten **822** entsprechen, und untere Adressen **858**, die dem unteren Abschnitt **830** entsprechen. Die oberen Adressen **852** sind über alle L2-Slices **800** vertauscht (d. h. pseudozufällig basierend auf der SPA-Adresse verschachtelt) und entsprechen oberen Abschnitten dieser L2-Slices. Die unteren Adressen **858** sind über alle L2-Scheiben **800** vertauscht und entsprechen unteren Abschnitten dieser L2-Scheiben. Die oberen Adressen **852** und die unteren Adressen **858** sind allgemein nur für Entitäten auf Systemebene, wie beispielsweise den Hypervisor **124**, und/oder alle Entitäten, die über eine physische Funktion (PF) arbeiten, zugänglich. Partitionierbare Adressen **854** sind den PPU-Partitionen **600** zugewiesen. Insbesondere ist eine Adressbereich **856(0)** der PPU-Partition **600(0)**, eine Adressbereich **856(4)** der PPU-Partition **600(4)**, eine Adressbereich **856(6)** der PPU-Partition **600(6)** und eine Adressbereich **856(7)** der PPU-Partition **600(7)** zugewiesen. Die Adressbereiche **856** dürfen nur für die SMC-Engine(s) **700** zugänglich sein, die innerhalb entsprechender PPU-Partitionen **600** ausgeführt werden.

[0080] Allgemein auf die **Fig. 5-8B** Bezug nehmen, unterstützt der vorstehende Ansatz zur Partitionierung der PPU-Ressourcen **500** eine Reihe von Nutzungsszenarien, einschließlich Einzelbenutzer- und Multi-Benutzer-Nutzungsszenarien. In einem Einzelbenutzer-Nutzungsszenario kann die PPU **200** partitioniert werden, um verschiedenen Benutzern, die einem einzigen Benutzer bzw. Inhaber zugeordnet sind, unabhängigen Zugriff auf PPU-Ressourcen zu ermöglichen. Beispielsweise könnten verschiedene Benutzer, die einem bestimmten Inhaber zugeordnet sind, unterschiedliche kuratierte Arbeitslasten über verschiedene PPU-Partitionen **600** hinweg ausführen. In Einzelbenutzer-Nutzungsszenarien kann einer einzigen Entität Zugriff auf die Gesamtheit der PPU-Ressourcen **500** gewährt werden. In einem Multi-Inhaber-Nutzungsszenario kann die PPU **200** partitioniert werden, um einem oder mehreren Benutzern, die einem oder mehreren verschiedenen Inhabern zugeordnet sind, unabhängigen Zugriff auf PPU-Ressourcen zu ermöglichen. In einem Multi-Inhaber-Nutzungsszenario können mehrere Entitäten Zugriff auf verschiedene PPU-Partitionen **600** erhalten, die ver-

schiedene Teile der PPU-Ressourcen **500** beinhalten.

[0081] In jedem beliebigen Anwendungsszenario arbeiten der Gerätetreiber **122** und der Hypervisor **124** zusammen, um einen zweistufigen Prozess durchzuführen, der erstens die Partitionierung von PPU **200** in PPU-Partitionen **600** und zweitens die Konfiguration dieser PPU-Partitionen **600** in SMC-Engines **700** umfasst. Dieser Prozess wird nachstehend in Verbindung mit den **Fig. 9-10** ausführlicher beschrieben.

Techniken zur Konfiguration
logischer Gruppierungen

von Hardwareressourcen

[0082] **Fig. 9** ist ein Ablaufdiagramm, das veranschaulicht, wie der Hypervisor von **Fig. 1** eine PPU partitioniert und konfiguriert, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet die Hypervisor-Umgebung **900** eine Gastumgebung **910** und eine Hostumgebung **920**, die durch eine Hypervisor-Vertrauensgrenze **930** voneinander getrennt sind. Die Gastumgebung **910** beinhaltet eine Systemverwaltungsschnittstelle (SMI) **912**, einen Kernel-Treiber **914** und ein Gastbetriebssystem (OS) **916**. Die Hostumgebung **920** beinhaltet eine SMI **922**, ein virtuelles GPU (vGPU)-Plugin **924**, ein Hostbetriebssystem **926** und einen Kernel-Treiber **928**. In der Gastumgebung **910** enthaltene Module, die oberhalb der Hypervisor-Vertrauensgrenze **930** liegen, werden in der Regel mit einer niedrigeren Berechtigungsstufe ausgeführt als Module, die in der Hostumgebung **920** enthalten sind und unterhalb der Hypervisor-Vertrauensgrenze **930** liegen, einschließlich wiederholter Instanzen desselben Moduls, wie dem SMI **912** und dem SMI **922**. Der Hypervisor **124** wird mit einem Satz von Berechtigungen auf Kernel-Ebene ausgeführt und kann jedem der gezeigten Module entsprechende Berechtigungen erteilen. In einigen Ausführungsformen besteht eine Eins-zu-Eins-Entsprechung zwischen einer VM und einer Gastumgebung **910**; und wenn mehrere VMs nicht kontextabhängig auf eine PPU-Partition **600** umgeschaltet werden, besteht allgemein eine Eins-zu-Entsprechung zwischen Gastumgebungen und PPU-Partitionen.

[0083] Im Betrieb interagiert ein Verwaltungsbenutzer der PPU **200** mit der PPU **200** über die Hostumgebung **920** und das Host-OS **926**, um die PPU-Partitionen **600** zu konfigurieren. Insbesondere stellt der Verwaltungsbenutzer die Partitionierungseingabe **904** der SMI **922** zur Verfügung. Als Antwort gibt die SMI **922** einen Befehl „Partition erstellen“ an den Kernel-Treiber **928** aus, der die Zielkonfiguration der PPU-Partitionen **600** angibt. Der Kernel-Treiber **928** überträgt den Befehl „Partition erstellen“ an die Hostschnittstelle **220** innerhalb der PPU **200**, um die verschiedenen PPU-Ressourcen **500** zu partitionieren.

Auf diese Weise kann der Verwaltungsbenutzer die PPU **200** initialisieren, um eine spezifische Konfiguration der PPU-Partitionen **600** zu erhalten. Allgemein hat der Verwaltungsbenutzer uneingeschränkten Zugriff auf die PPU **200**. Der Verwaltungsbenutzer könnte z. B. der Systemadministrator in einem Rechenzentrum sein, in dem sich die PPU **200** befindet. Der Verwaltungsbenutzer könnte der Systemadministrator in einem Rechenzentrum sein, in dem sich mehrere PPUs **200** befinden. Der Verwaltungsbenutzer partitioniert die PPU **200** in der beschriebenen Weise, um die verschiedenen PPU-Partitionen **600** so vorzubereiten, dass sie unabhängig voneinander konfiguriert und von verschiedenen Gastbenutzern verwendet werden können, wie nachstehend ausführlicher beschrieben wird.

[0084] Ein Gastbenutzer der PPU **200** interagiert mit einer bestimmten „Gast“-PPU-Partition **600** über eine VM, die innerhalb der Gastumgebung **910** ausgeführt wird, um die SMC-Engines **700** innerhalb dieser Gast-PPU-Partition **600** zu konfigurieren. Konkret stellt der Gastbenutzer die Konfigurationseingaben **902** bis SMI **912** zur Verfügung. SMI **912** gibt dann einen Befehl „Partition konfigurieren“ an den Kernel-Treiber **914** aus, der die Zielkonfiguration der SMC-Engines **700** angibt. Der Kernel-Treiber **914** überträgt den Befehl „Partition konfigurieren“ über das Gastbetriebssystem **916** über die Hypervisor-Vertrauensgrenze **930** an das vGPU-Plugin **924**. Das vGPU-Plugin **924** gibt verschiedene VM-Aufrufe an den Kernel-Treiber **928** aus. Der Kernel-Treiber **928** überträgt den Befehl „Partition konfigurieren“ an die Hostschnittstelle **220** innerhalb der PPU **200**, um verschiedene Ressourcen der Gast-PPU-Partition **600** zu konfigurieren. Auf diese Weise kann der Gastbenutzer eine bestimmte PPU-Partition **600** so konfigurieren, dass eine spezifische Konfiguration der SMC-Engines **700** vorliegt. Allgemein hat der Gastbenutzer nur eingeschränkten Zugriff auf einen Teil der PPU-Ressourcen **500**, die der Gast-PPU-Partition **600** zugeordnet sind. Der Gastbenutzer könnte z.B. ein Kunde des Rechenzentrums sein, in dem sich die PPU **200** befindet, der Zugang zu einem Bruchteil der PPU **200** erwirbt. In einer Ausführungsform können Gastbetriebssysteme **916** mit ausreichenden Sicherheitsmaßnahmen konfiguriert sein, so dass jedes Gastbetriebssystem **916** eine entsprechende PPU-Partition **600** ohne Beteiligung der Hostumgebung **920** und/oder des Hypervisors **124** konfigurieren kann.

[0085] Fig. **10** ist ein Ablaufdiagramm von Verfahrensschritten zur Partitionierung und Konfiguration einer PPU im Auftrag eines oder mehrerer Benutzer, gemäß verschiedenen Ausführungsformen. Obwohl die Verfahrensschritte in Verbindung mit den Systemen der Fig. **1-9** beschrieben werden, versteht sich für den Fachmann, dass jedes beliebige System, das dazu konfiguriert ist, die Verfahrensschritte in belie-

biger Reihenfolge auszuführen, in den Anwendungsbereich der vorliegenden Ausführungsformen fällt.

[0086] Wie gezeigt ist, beginnt ein Verfahren bei Schritt **1000**, wobei der Hypervisor **124** die Partitionierungseingabe **904** über die Hostumgebung **920** empfängt. Die Hostumgebung **920** wird mit einer erhöhten Berechtigungsstufe ausgeführt, wodurch ein Verwaltungsbenutzer direkt mit PPU **200** interagieren kann. Die Partitionierungseingabe **904** spezifiziert eine Sol- bzw. Zielkonfiguration von PPU-Partitionen **600**, einschließlich einer gewünschten Größe und Anordnung der PPU-Partitionen **600**. In einer Ausführungsform kann die Partitionierungseingabe von dem Verwaltungsbenutzer über die Hostumgebung empfangen werden.

[0087] In Schritt **1004** erzeugt der Hypervisor **124** eine oder mehrere PPU-Partitionen **600** innerhalb der PPU **200** auf der Grundlage der in Schritt **1002** empfangenen Partitionierungseingabe **904**. Insbesondere implementiert der Hypervisor **124** die SMI **922**, um einen „Partition erstellen“-Befehl an den Kernel-Treiber **928** auszugeben. Im Ansprechen darauf interagiert der Kernel-Treiber **928** mit der Hostschnittstelle **220** der PPU **200**, um eine oder mehrere PPU-Partitionen **600** mit der gewünschten Konfiguration zu erstellen.

[0088] In Schritt **1006** verteilt der Hypervisor **124** die Speicherressourcen über die PPU-Partition(en) **600**, die in Schritt **1004** generiert wurden. Insbesondere unterteilt der Hypervisor **124** über den vorstehend besprochenen Befehl „Partition erstellen“ DRAM **272** in der vorstehend in Verbindung mit Fig. **8A** beschriebenen Weise, um verschiedene Bereiche von DRAM **272** und entsprechende L2-Cache-Slices **800** den PPU-Partition(en) **600** zuzuweisen. In einer Ausführungsform kann der Hypervisor **124** auch eine Adresszuordnungseinheit konfigurieren, um partitionsspezifische Vertauschungs- bzw. Swizzle-Operationen durchzuführen, um den Zugriff auf diese verschiedenen Bereiche von DRAM **272** über L2-Cache-Slices **800** zu ermöglichen. Diese spezielle Ausführungsform wird nachstehend in Verbindung mit Fig. **22** ausführlicher beschrieben.

[0089] In Schritt **1008** verteilt der Hypervisor **124** die PPU-Berechnungs- und/oder Grafikressourcen über die PPU-Partition(en) **600**. Dabei weist der Hypervisor **124** über den Befehl „Partition erstellen“ eine oder mehrere Sys-Pipes **230** und eine oder mehrere GPCs **242** diesen PPU-Partition(en) **600** zu. In einer Ausführungsform kann der Hypervisor **124** die Schritte **1006** und **1008** implementieren, indem er einen oder mehrere PPU-Slices **610** logisch den PPU-Partition(en) **600** zuweist, wodurch Speicherressourcen und Rechen-/Grafikressourcen gemeinsam zugewiesen werden. Wenn die Schritte **1002**, **1004**, **1006** und **1008** des Verfahrens **1000** abgeschlossen sind, wird

die PPU **200** partitioniert, und kann ein Gastbenutzer dann eine oder mehrere dieser PPU-Partitionen **600** konfigurieren, wie nachstehend beschrieben wird.

[0090] In Schritt **1010** empfängt der Hypervisor **124** über die Gastumgebung **910** eine Konfigurationseingabe **902**, die einer ersten PPU-Partition zugeordnet ist. Die Gastumgebung **910** wird mit einer reduzierten Berechtigungsstufe ausgeführt, wodurch ein Gastbenutzer nur mit der ersten PPU-Partition **600** interagieren kann. Die Konfigurationseingabe **902** spezifiziert eine Soll- bzw. Zielkonfiguration der SMC-Engines **700** innerhalb der ersten PPU-Partition **600**, einschließlich einer gewünschten Größe und Anordnung der SMC-Engines **700**. In einer Ausführungsform kann die Konfigurationseingabe von dem Gastbenutzer über die Gastumgebung empfangen werden.

[0091] In Schritt **1012** generiert der Hypervisor **124** über den Befehl „Partition konfigurieren“ eine oder mehrere SMC-Engines **700** innerhalb der ersten PPU-Partition **600** auf der Grundlage der in Schritt **1010** empfangenen Konfigurationseingabe **902**. Eine bestimmte SMC-Engine **700** kann Rechen- und/oder Grafikressourcen beinhalten, die von einer Sys-Pipe **230** und einem oder mehreren GPCs **242** abgeleitet sind, die von einem oder mehreren PPU-Slices **610** abgeleitet sind. Eine bestimmte SMC-Engine **700** hat Zugriff auf zumindest einen Teil einer PPU-Speicherpartition **710**, die in der PPU-Partition **600** enthalten ist, in der sich die SMC-Engine **700** befindet, wobei diese PPU-Speicherpartition **710** einen oder mehrere Sätze von L2-Cache-Slices und entsprechende Teile von DRAM **272** beinhaltet.

[0092] In Schritt **1014** verteilt der Hypervisor **124** die der ersten PPU-Partition **600** zugewiesenen Speicherressourcen über die in Schritt **1012** erzeugte(n) SMC-Engine(s) **700**. Allgemein teilen sich die SMC-Engine(s) **700** die erste PPU-Speicherpartition **710**, wenn diese SMC-Engine(s) **700** in derselben PPU-Partition **600** enthalten sind. Zuweisungen an jede SMC-Engine **700** werden für die Kontexte bereitgestellt, die auf diesen SMC-Engines **700** laufen, und die Zuweisungen innerhalb der PPU-Speicherpartition **710** werden auf Seitenbasis implementiert. In einigen Ausführungsformen führt das Gastbetriebssystem **916** einer VM Schritt **1014** aus, indem es Speicherressourcen auf Kontexte verteilt, die auf SMC-Engines **700** laufen, die Teil der PPU-Partition **600** sind, die zu der Gastumgebung **910** gehört. In anderen Ausführungsformen führt der Hypervisor **124** eine Speicherressourcenzuweisung **1014** für mehrere VMs unter Verwendung einer PPU-Partition **600** durch, und verteilt jede VM darüber hinaus Speicherressourcen **1024** an Kontexte, die auf SMC-Engines **700** laufen.

[0093] In Schritt **1016** verteilt der Hypervisor **124** die der ersten PPU-Partition **600** zugewiesenen Rechen- und/oder Grafikressourcen über die in Schritt **1012** generierte(n) SMC-Engine(s) **700**. Über den Befehl „Partition konfigurieren“ weist der Hypervisor **124** jeder SMC-Engine **700** eine bestimmte Sys-Pipe **230** zu, die in der Gast-PPU-Partition **600** enthalten ist. Der Hypervisor **124** ordnet darüber hinaus einen oder mehrere GPCs **242** jeder SMC-Engine **700** zu. Wenn die Schritte **1010**, **1012**, **1014** und **1016** des Verfahrens **1000** abgeschlossen sind, ist die erste PPU-Partition **600** konfiguriert, und kann der Gastbenutzer dann Verarbeitungsvorgänge auf der/den SMC-Engine(s) **700** innerhalb dieser PPU-Partition **600** einleiten.

[0094] In Schritt **1018** veranlasst der Hypervisor **124** die erste PPU-Partition **600**, eine oder mehrere VMs über die SMC-Engine(s) **700**, die innerhalb der ersten PPU-Partition **600** konfiguriert ist/sind, hinweg in Zeitscheiben zu unterteilen. Die in Zeitscheiben unterteilten VMs können unabhängig von anderen VMs arbeiten, die innerhalb der ersten PPU-Partition **600** ausgeführt werden, und isoliert von anderen VMs arbeiten, die innerhalb anderer PPU-Partitionen **600** ausgeführt werden. In einer Ausführungsform können eine oder mehrere VMs gleichzeitig über die SMC-Engine(n) **700** in Zeitscheiben unterteilt werden. Auf diese Weise ermöglichen die offenbarten Techniken, dass eine fraktionierte PPU die parallele Ausführung von Verarbeitungsaufgaben unterstützt, die mehreren unterschiedlichen Verarbeitungskontexten zugeordnet sind.

[0095] In einigen Ausführungsformen arbeiten die hierin offenbarten Techniken in einem nicht virtualisierten System. Der Fachmann erkennt, dass ein einzelnes Betriebssystem-Nutzungsmodell auf einer PPU **200** oder einem Satz von PPUs **200** alle in Verbindung mit VMs beschriebenen Mechanismen verwenden kann. In einigen Ausführungsformen entsprechen Container der Beschreibung von VMs, was bedeutet, dass Container auf einem einzelnen Betriebssystem die Verarbeitungs isolation erreichen können, die den hierin beschriebenen VMs gewährt wird.

Aufteilen von Rechenressourcen zur Unterstützung
gleichzeitiger Mehrfachkontexte

[0096] In verschiedenen Ausführungsformen empfängt dann, wenn der Hypervisor **124** die PPU **200** im Auftrag eines Verwaltungsbenutzers in der vorstehend beschriebenen Weise partitioniert, der Hypervisor **124** eine Eingabe von dem Verwaltungsbenutzer, die verschiedene Grenzen zwischen den PPU-Partitionen **600** angeben. Auf der Grundlage dieser Eingabe gruppiert der Hypervisor **124** logisch PPU-Slices **610** in PPU-Partitionen **600**, weist jeder

PPU-Partition **600** verschiedene Hardwareressourcen zu und koordiniert verschiedene andere Operationen, um die gleichzeitige Implementierung mehrerer Verarbeitungskontexte innerhalb einer bestimmten PPU-Partition **600** zu unterstützen. Der Hypervisor **124** führt darüber hinaus zusätzliche Techniken aus, um die Migration von Verarbeitungskontexten zwischen PPU-Partitionen **600**, die auf verschiedenen PPU-Slices **200** konfiguriert sind, zu unterstützen. Diese verschiedenen Techniken werden nachstehend in Verbindung mit den **Fig. 11-18** ausführlicher beschrieben.

[0097] **Fig. 11** veranschaulicht eine Ausführungsform einer Partitionskonfigurationstabelle, gemäß der der Hypervisor **124** von **Fig. 1** eine oder mehrere PPU-Partitionen konfigurieren kann, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet die Partitionskonfigurationstabelle **1100** Partitionierungsoptionen **0** bis **14**. Die Partitionierungsoptionen **0-14** sind über den PPU-Slices **610** dargestellt. Jede der Partitionierungsoptionen **0-14** erstreckt sich über eine andere Gruppierung von PPU-Slices **610** und repräsentiert auf diese Weise eine andere mögliche PPU-Partition **600**. Insbesondere überspannt die Partitionierungsoption **0** die PPU-Slices **610(0)** bis **610(7)** und repräsentiert daher eine PPU-Partition **600**, die alle acht PPU-Slices **610** beinhaltet. In ähnlicher Weise überspannt die Partitionierungsoption **1** die PPU-Slices **610(0)** bis **610(3)** und repräsentiert daher eine PPU-Partition **600**, die nur die ersten vier PPU-Slices **610** beinhaltet, ähnlich zu der in den **Fig. 6-7** dargestellten PPU-Partition **600(0)**. Die Partitionierungsoption **2** erstreckt sich über die PPU-Slices **610(4)** bis **610(7)** und repräsentiert daher eine PPU-Partition **600**, die nur die letzten vier PPU-Slices **610** beinhaltet. Die Partitionierungsoptionen **3, 4, 5** und **6** überspannen verschiedene Gruppierungen von zwei benachbarten PPU-Slices **610**, während die Partitionierungsoptionen **7, 8, 9, 10, 11, 12, 13** und **14** nur einen entsprechenden PPU-Slice **610** überspannt.

[0098] Die Partitionskonfigurationstabelle **1100** beinhaltet darüber hinaus Grenzoptionen **1110**, die verschiedene mögliche Positionen für Partitionsgrenzen repräsentieren. Insbesondere repräsentieren Grenzoptionen **1110(1)** und **1110(9)** Grenzen der Partitionierungsoption **0**. Grenzoptionen **1110(1)** und **1110(5)** repräsentieren Grenzen der Partitionierungsoption **1**, während Grenzoptionen **1110(5)** und **1110(9)** Grenzen der Partitionierungsoption **2** repräsentieren. Grenzoptionen **1110(1), 1110(3), 1110(5), 1110(7)** und **1110(9)** repräsentieren Grenzen der Partitionierungsoptionen **3, 4, 5** und **6**. Die Grenzoptionen **1110(1)** bis **1110(9)** repräsentieren Grenzen, die mit den Partitionierungsoptionen **7** bis **14** assoziiert sind. Es wird begrüßt, dass der Fachmann viele verschiedene Schemata erstellen können, um die gleiche Funktion wie die Konfigurationstabelle **1100** zu erreichen,

d.h. einen Satz von Freigabebits, eine Liste vordefinierter Auswahlmöglichkeiten oder jede beliebige andere Form, die eine Kontrolle darüber ermöglicht, wie PPU-Slices **610** in PPU-Partitionen **600** partitioniert werden können. Darüber hinaus versteht sich für den Fachmann, dass die Partitionskonfigurationstabelle **1100** außer denjenigen, die in **Fig. 11** gezeigt sind, jede technisch machbare Anzahl von Einträgen beinhalten kann.

[0099] Während der Partitionierung empfängt der Hypervisor **124** oder der Gerätetreiber **122**, der auf der Hypervisor-Ebene läuft, eine Partitionierungseingabe von dem Verwaltungsbenutzer, die spezifische Partitionierungsoptionen angibt, gemäß welchen die PPU **200** zu partitionieren ist. Der Hypervisor **124** oder der Gerätetreiber **122** aktiviert dann einen spezifischen Satz von Grenzoptionen **1110**, die eine oder mehrere Gruppen von PPU-Slices **610** logisch voneinander isolieren, um die gewünschte Partitionierung zu implementieren, wie nachstehend beispielhaft in Verbindung mit **Fig. 12** näher beschrieben wird.

[0100] **Fig. 12** veranschaulicht, wie der Hypervisor **124** oder der Gerätetreiber **122** von **Fig. 1** eine PPU partitioniert, um eine oder mehrere PPU-Partitionen zu erzeugen, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, empfängt der Hypervisor **124** oder der Gerätetreiber **122**, der auf der Hypervisor-Ebene läuft, während der Partitionierung eine Eingabe von dem Verwaltungsbenutzer, die angibt, dass die PPU **200** gemäß den Partitionierungsoptionen **1, 5, 13** und **14** zu partitionieren ist (zur Verdeutlichung hervorgehoben). Im Ansprechen darauf aktiviert der Hypervisor **124** oder der Gerätetreiber **122** die Grenzoptionen **1110(1), 1110(5), 1110(7), 1110(8)** und **1110(9)** und deaktiviert die anderen Grenzoptionen, um die PPU-Partitionen **600(0), 600(4), 600(6)** und **600(7)** zu erzeugen. Diese beispielhafte Konfiguration von PPU-Partitionen **600** ist auch in den **Fig. 6-7** gezeigt.

[0101] Allgemein auf die **Fig. 11-12** Bezug nehmend, implementiert der Hypervisor **124** oder der Gerätetreiber **122** die vorstehenden Techniken, indem er jede Auswahl einer Partitionierungsoption auf einen bestimmten Binärwert abbildet, der anschließend zum Aktivieren und Deaktivieren der Grenzoptionen **1110** verwendet wird. Der einer bestimmten Partitionierungsoption zugeordnete Binärwert wird hierin als „Vertauschungsidentifikator“ bzw. „Swizzle Identifier“ (swizID) bezeichnet. Die verschiedenen von dem Hypervisor **124** implementierten swizIDs sind nachstehend in Tabelle 1 tabellarisch dargestellt:

Tabelle 1

Partitionierungsoption	SwizID
0	11000000011
1	10000100011
2	11000100001
3	10000001011
4	10000101001
5	10010100001
6	11010000001
7	10000000111
8	10000001101
9	10000011001
10	10000110001
11	10001100001
12	10011000001
13	10110000001
14	11100000001

[0102] Der Hypervisor **124** aktiviert oder deaktiviert die Grenzooptionen **1110** für eine gegebene Partitionierungsoption basierend auf der swizID, die der gegebenen Partitionierungsoption zugeordnet ist. Beispielsweise könnte der Hypervisor **124** die Grenzooptionen **1110(1)** und **1110(3)** aktivieren, um die PPU **200** gemäß der Partitionierungsoption **3** auf der Grundlage der entsprechenden swizID, 10000001011, zu konfigurieren. Bits **1** und **3** dieser swizID aktivieren die Grenzooptionen **1110(1)** bzw. **1110(3)**, und Bits **2** und **4-9** deaktivieren die übrigen Grenzooptionen. Bits **0** und **10** aller swizIDs werden auf eins (1) gesetzt, um die Grenzen innerhalb des L2-Cache **400** zu aktivieren, wie nachstehend in Verbindung mit den **Fig. 19-20** näher beschrieben wird. Der Hypervisor **124** sammelt die verschiedenen swizIDs für die verschiedenen ausgewählten Konfigurationsoptionen und berechnet eine ODER-Operation über alle gesammelten swizIDs hinweg, um eine Konfigurations-swizID zu erzeugen, die die Konfiguration der PPU-Partitionen **600** definiert. Die Konfigurations-SwizID gibt alle Grenzooptionen **1110** an, die zu aktivieren und zu deaktivieren sind, um die gewünschte Konfiguration der PPU-Partitionen **600** zu erreichen.

[0103] Der Fachmann wird erkennen, dass bestimmte Kombinationen von Partitionierungsoptionen nicht durchführbar sind. Zum Beispiel können die Partitionierungsoptionen **0** und **1** nicht in Verbindung miteinander implementiert werden, weil die Partitionierungsoptionen **0** und **1** einander überlappen. Während der Partitionierung erkennt der Hypervisor **124** automatisch undurchführbare Kombina-

tionen von Partitionierungsoptionen und korrigiert diese Kombinationen, indem er eine oder mehrere Partitionierungsoptionen und/oder entsprechende swizIDs modifiziert oder eine oder mehrere Partitionierungsoptionen und/oder entsprechende swizIDs auslässt.

[0104] Darüber hinaus kann der Hypervisor **124** dynamisch Hardwarefehler erfassen, die dazu führen, dass bestimmte Partitionierungsoptionen nicht durchführbar sind. Beispielsweise sei angenommen dass der PPU-Slice **610(0)** einen nicht funktionsfähigen GPC **242** beinhaltet, der während der Herstellung ausgeschlossen und abgeschmolzen wird. In dieser Situation würde eine PPU-Partition **600**, die nur den PPU-Slice **610(0)** beinhaltet, keine ausreichenden Rechenressourcen für den Betrieb haben und wäre daher nicht implementierbar. In dieser Situation würde der Hypervisor **124** die Auswahl der Partitionierungsoption **7** und/oder die Verwendung der entsprechenden swizID verbieten, weil jede PPU-Partition **600**, die in Übereinstimmung mit dieser Partitionierungsoption konfiguriert wurde, keine Rechenoperationen durchführen könnte und daher nicht korrekt funktionieren würde.

[0105] In manchen Situationen kann der Hypervisor **124** bestimmte Konfigurationsoptionen zulassen, die eine gewisse Menge an nicht funktionsfähiger Hardware umfassen, solange eine PPU-Partition **600**, die in Übereinstimmung mit solchen Konfigurationsoptionen konfiguriert wurde, noch bis zu einem gewissen Grad funktionieren kann. Im vorstehenden Beispiel könnte der Hypervisor **124** die Auswahl der Konfigurationsoption **3** erlauben, solange der PPU-Slice **610(1)** einen funktionsfähigen GPC **242** enthält. Jede gemäß der Konfigurationsoption **3** konfigurierte PPU-Partition **600** würde weiterhin funktionieren, würde aber nur die Hälfte der Rechenressourcen im Vergleich zu einer ähnlichen PPU-Partition **600** ohne nicht funktionsfähige Hardware beinhalten.

[0106] Im Anschluss an die Partitionierung der PPU **200** in der vorstehend beschriebenen Weise weist der Hypervisor **124** den resultierenden PPU-Partitionen **600** verschiedene Hardwareressourcen zu. Einige dieser Ressourcen werden statisch verschiedenen PPU-Slices **610** zugewiesen und stellen dedizierte Unterstützung für bestimmte Operationen bereit, während andere Ressourcen über verschiedene PPU-Slices **610** innerhalb derselben PPU-Partition **600** oder innerhalb verschiedener PPU-Partitionen **600** gemeinsam genutzt werden, wie nachstehend in Verbindung mit **Fig. 13** näher beschrieben wird.

[0107] **Fig. 13** veranschaulicht, wie der Hypervisor von **Fig. 1** verschiedene PPU-Ressourcen während der Partitionierung zuweist, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, sind die PCEs **222(0)** bis **222(7)** an die PPU-Slices **610(0)** bis **610(7)**

gekoppelt. In diesem Beispiel beinhaltet die PPU **200** eine Anzahl von PCEs **222**, die gleich der Anzahl der PPU-Slices **610** ist. Demgemäß kann der Hypervisor **124** jede PCE **222** statisch einem anderen PPU-Slice **610** zuweisen und diese PCE **222** dazu konfigurieren, Kopieroperationen im Auftrag des entsprechenden PPU-Slices **610** auf eine dedizierte Weise durchführen.

[0108] Andere Hardwareressourcen, die in der PPU **200** enthalten sind, können nicht auf die vorstehend beschriebene Weise statisch zugewiesen werden, weil diese Ressourcen vergleichsweise knapp sein können. In dem gezeigten Beispiel beinhaltet die PPU **200** nur zwei Decoder **1300**, die über acht PPU-Slices **610** hinweg allokiert werden müssen. Demgemäß weist der Hypervisor **124** den in der PPU-Partition **600(0)** enthaltenen PPU-Slices **610(0)** bis **610(3)** den Decoder **1300(0)** dynamisch zu. Darüber hinaus weist der Hypervisor **124** den in der PPU-Partition **600(4)** enthaltenen PPU-Slices **610(4)** und **610(5)**, dem in der PPU-Partition **600(6)** enthaltenen PPU-Slice **610(6)** und dem in der PPU-Partition **600(7)** enthaltenen PPU-Slice **600(7)** den Decoder **1300(1)** dynamisch zu.

[0109] In der gezeigten Konfiguration wird der Decoder **1300(0)** dynamisch zugewiesen, um Decodieroperationen für die PPU-Partition **600(0)** in einer dedizierten Weise durchzuführen, aber der Decoder **1300(1)** wird über die PPU-Partitionen **600(4)**, **600(6)** und **600(7)** hinweg gemeinsam genutzt. In verschiedenen Ausführungsformen können ein oder mehrere Leistungsmonitore die Nutzung von Hardwareressourcen verwalten, die auf die beschriebene Weise gemeinsam genutzt werden, um die Ressourcennutzung über verschiedene PPU-Slices **610** hinweg auszugleichen. Der Hypervisor **124** führt die vorstehend beschriebenen Techniken aus, um beliebige technisch machbare Ressourcen der PPU **200** den PPU-Partitionen **600** zuzuweisen.

[0110] Wenn die Partitionierung durchgeführt wurde und die verschiedenen Ressourcen der PPU **200** statisch oder dynamisch den jeweiligen PPU-Slices **610** zugewiesen sind, ist der Hypervisor **124** bereit, es VMs zu erlauben, die Ausführung von Verarbeitungsaufgaben innerhalb dieser PPU-Partitionen **600** zu beginnen. Dabei können VMs innerhalb einer bestimmten PPU-Partition **600** gleichzeitig mehrere Verarbeitungskontexte starten, und zwar isoliert von anderen Verarbeitungskontexten, die anderen PPU-Partitionen **600** zugeordnet sind, wie vorstehend erwähnt wurde und wie nachstehend in Verbindung mit den **Fig. 14A-14B** näher beschrieben wird. PPU-Slices **610**, die nicht in Verwendung sind, können in andere PPU-Partitionen **600** umpartitioniert werden, während andere PPU-Slices in aktiven PPU-Partitionen **600** in Verwendung sind.

[0111] **Fig. 14A** veranschaulicht, wie ein Mehrfachgast-OS **916**, auf dem mehrere VMs ausgeführt werden, gleichzeitig mehrere Verarbeitungskontexte innerhalb einer oder mehrerer PPU-Partitionen starten, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet das Gastbetriebssystem **916** verschiedene Verarbeitungskontexte **1400**, die verschiedenen PPU-Partitionen **600** zugeordnet sind. Die Verarbeitungskontexte **1400(0)** und **1400(1)** sind der PPU-Partition **600(0)** zugeordnet und können entweder auf der SMC-Engine **700(0)** oder der SMC-Engine **700(1)** gestartet werden. In einigen Ausführungsformen verbleibt ein Verarbeitungskontext, nachdem er einer SMC-Engine **700** zugeordnet wurde, bis zum Abschluss auf dieser SMC-Engine **700**. Der Verarbeitungskontext **1400(4)** ist der PPU-Partition **600(4)** zugeordnet und kann auf der SMC-Engine **700(4)** gestartet werden. Die Verarbeitungskontexte **1400(6)** und **1400(7)** sind den PPU-Partitionen **600(6)** bzw. **600(7)** zugeordnet und können auf den SMC-Engines **700(6)** bzw. **700(7)** gestartet werden.

[0112] Wie bereits in Verbindung mit **Fig. 7** erwähnt wurde, kann jede SMC-Engine **700** Verarbeitungsaufgaben in Verbindung mit einem bestimmten Verarbeitungskontext **1400** isoliert von anderen SMC-Engines **700** ausführen, die Verarbeitungsaufgaben in Verbindung mit einem bestimmten Verarbeitungskontext **1400** ausführen. Die Verarbeitungsaufgaben, die von einer bestimmten SMC-Engine **700** in Verbindung mit einem bestimmten Verarbeitungskontext **1400** ausgeführt werden, werden unabhängig von anderen Verarbeitungsaufgaben geplant, die von anderen SMC-Engines **700** in Verbindung mit einem beliebigen anderen Verarbeitungskontext **1400** ausgeführt werden. Darüber hinaus können, wie nachstehend in Verbindung mit den **Fig. 15-16** näher beschrieben wird, SMC-Engines **700** unabhängig voneinander Fehler und/oder Störungen erfahren und ohne Unterbrechung bzw. Störung des Betriebs anderer SMC-Engines **700** zurückgesetzt werden.

[0113] Ferner kann jede SMC-Engine **700** dazu konfiguriert sein, Verarbeitungsaufgaben auszuführen, die einem oder mehreren Verarbeitungsunterkontexten **1410** zugeordnet sind, die in einem einzigen übergeordneten Verarbeitungskontext **1400** enthalten sind und/oder von diesem abgeleitet wurden. Wie gezeigt ist, beinhaltet ein gegebener Verarbeitungskontext **1400(0)** einen oder mehrere Verarbeitungsunterkontexte **1410(0)** und beinhaltet ein gegebener Verarbeitungskontext **1400(1)** einen oder mehrere Verarbeitungsunterkontexte **1410(1)**. Der Hypervisor **124** konfiguriert den Verarbeitungsunterkontext **1410** und jeweilige Gerätetreiber. Verarbeitungsunterkontexte **1410**, die einem gegebenen übergeordneten Verarbeitungskontext **1400** zugeordnet sind, werden auf derselben SMC-Engine **700** gestartet, auf der auch der übergeordnete Verarbeitungskontext **1400** gestartet wird. In dem gezeigten Beispiel werden al-

so die Verarbeitungsunterkontexte **1410(0)** auf der SMC-Engine **700(0)** und die Verarbeitungsunterkontexte **1410(1)** auf der SMC-Engine **700(1)** gestartet. In einer Ausführungsform kann jedes Gastbetriebssystem **916** in der Lage sein, eine jeweilige PPU-Partition **600** unabhängig von dem Hypervisor **124** zu konfigurieren, ohne die Konfiguration anderer PPU-Partitionen **600** stören zu können.

[0114] In bestimmten Ausführungsformen, in denen keine Virtualisierung verwendet wird, können der Hypervisor **124** und Gastbetriebssysteme **916** fehlen und kann das Hostbetriebssystem **926** Verarbeitungs-kontexte **1400** und Verarbeitungsunterkontexte **1410** konfigurieren und starten, wie nachstehend in Verbindung mit **Fig. 14B** ausführlicher beschrieben wird.

[0115] **Fig. 14B** veranschaulicht, wie ein Hostbetriebssystem mehrere Verarbeitungs-kontexte gleichzeitig innerhalb einer oder mehrerer PPU-Partitionen startet, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet das Hostbetriebssystem **926** die Verarbeitungs-kontexte **1400** und die Verarbeitungsunterkontexte **1410**. In der gezeigten Ausführungsform ist das Hostbetriebssystem **926** dazu konfiguriert, Verarbeitungs-kontexte **1400** und Verarbeitungsunterkontexte **1410** auf SMC-Engines **700** ohne Beteiligung eines Hypervisors oder anderer Virtualisierungssoftware zu starten. Die gezeigte Ausführungsform kann in einem „Bare-Metal“-Szenario implementiert sein.

[0116] Allgemein auf die **Fig. 14A-14B** Bezug nehmend, werden Verarbeitungsaufgaben, die Verarbeitungsunterkontexten **1410** innerhalb desselben übergeordneten Verarbeitungs-kontexts **1400** zugeordnet sind, allgemein nicht unabhängig voneinander geplant und teilen sich in der Regel die Ressourcen einer entsprechenden SMC-Engine **700**. Ferner können die Verarbeitungsunterkontexte **1410**, die innerhalb einer bestimmten SMC-Engine **700** gestartet werden, in einigen Situationen Fehler und/oder Irrtümer verursachen, die dazu führen, dass die SMC-Engine **700** zurückgesetzt wird und alle relevanten Verarbeitungs-kontexte **1400** und/oder Verarbeitungsunterkontexte **1410** neu gestartet werden. Den Verarbeitungs-kontexten **1400** und/oder Verarbeitungsunterkontexten **1410** wird ein lokaler Virtualladressraumidentifizierer zugewiesen, der von einem globalen Virtualladressraumidentifizierer **1510** abgeleitet ist, der der PPU **200** als Ganzes zugeordnet ist, wie nachstehend in Verbindung mit **Fig. 15** näher beschrieben wird.

[0117] In einigen Ausführungsformen gibt es keine Virtualisierung und daher auch keinen Hypervisor, aber es ist für den Fachmann klar, dass ein Einzelbetriebssystem-Nutzungsmodell auf einer PPU **200** oder einem Satz von PPUs **200** alle die Mechanis-

men nutzen kann, die als VMs zugehörig beschrieben werden. In einigen Ausführungsformen entsprechen Container der Beschreibung von VMs, was bedeutet, dass Container auf einem Einzelbetriebssystem die Verarbeitungs-isolation erreichen können, die den VMs in den Beschreibungen hierin gewährt wird. Beispiele für Container sind LXC (LinuX-Container) und Docker-Container, wie sie in der Computerindustrie bekannt sind. Beispielsweise kann jeder Docker-Container einer PPU Partition **600** entsprechen, so dass die Erfindung eine Isolierung zwischen mehreren, unter einem Betriebssystem laufenden Docker-Containern bereitstellt.

[0118] **Fig. 15** veranschaulicht, wie der Hypervisor von **Fig. 1** den verschiedenen SMC-Engines Virtualladressraumidentifizierer zuweist, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhalten die Virtualladressraumidentifizierer **1500** einen separaten Bereich von virtuellen Adressen für jede SMC-Engine **700**. Jeder Bereich von virtuellen Adressen beginnt bei null (0), um die Konsistenz über die SMC-Engines **700** hinweg zu gewährleisten, aber jeder Bereich von virtuellen Adressen entspricht einem anderen Teil von globalen Virtualladressraumidentifizierern **1510**. Zum Beispiel entsprechen der SMC-Engine **700(0)** zugewiesene Virtualladressraumidentifizierer **0-15** globalen Virtualladressraumidentifizierern **0-15**, aber entsprechen der SMC-Engine **700(1)** zugewiesene Virtualladressraumidentifizierer **0-15** globalen Virtualladressraumidentifizierern **16-31**. In einer Ausführungsform kann ein globaler Satz von virtuellem Adressraum **1510** ein virtueller Adressraum oder ein physischer Adressraum sein. In einigen Ausführungsformen gibt es auch einen Virtualladressraumidentifizierer pro PPU-Partition, damit das Gastbetriebssystem einer VM einen nullbasierten Satz von Virtualladressraumidentifizierern für alle SMC-Engines **700** hat, die es besitzt.

[0119] Der Hypervisor **124** weist einer bestimmten SMC-Engine **700** einen bestimmten Bereich von Virtualladressraumidentifizierern zu, abhängig von der Anzahl von PPU-Slices **610**, aus welchen der SMC-Engine **700** Ressourcen zugewiesen werden. In dem gezeigten Beispiel weist der Hypervisor **124** die Virtualladressraumidentifizierer **0-15** der SMC-Engine **700(0)**, die Virtualladressraumidentifizierer **0-15** der SMC-Engine **700(1)** und die Virtualladressraumidentifizierer **0-15** der SMC-Engine **700(4)** zu. Der Hypervisor **124** weist den SMC-Engines **700(0)**, **700(1)** und **700(4)** 16 Virtualladressraumidentifizierer zu, weil diese SMC-Engines Ressourcen von zwei PPU-Slices **610** beziehen, wie in **Fig. 7** gezeigt ist. Im Gegensatz dazu weist der Hypervisor **124** die Virtualladressraumidentifizierer **0-7** den SMC-Engines **700(6)** und **700(7)** zu, weil diese SMC-Engines **700** Ressourcen von nur je einem PPU-Slice **610** beziehen. Der Hypervisor **124** kann ferner die einer bestimmten SMC-Engine **700** zugewiesenen Virtualladressraum-

identifizierer **0-7** unterteilen, um mehrere Verarbeitungskontexte **1400** zu unterstützen. Beispielsweise könnte der Hypervisor **124** die Virtuelladressraumidentifizierer **0-15**, die der SMC-Engine **700(0)** zugewiesen sind, in zwei Bereiche, **0-7** und **0-7**, unterteilen, von welchen jeder einem anderen Verarbeitungskontext **1400** zugewiesen werden könnte. Dieses Beispiel zeigt, wie die globalen Virtuelladressraumidentifizierer **1510** proportional in Gruppen wie **0-15**, **16-31**, **32-47**, **48-55** und **55-63** verteilt sind. In einigen Ausführungsformen sind die Virtuelladressraumidentifizierer eindeutig, so dass das vorstehende Beispiel die Virtuelladressraumidentifizierer **0-15**, **16-31**, **32-47**, **48-55** und **55-63** hätte und nicht **0-15**, **0-15**, **0-15**, **0-7** und **0-7** wie in Fig. 15 gezeigt. In einigen Ausführungsformen werden die globalen Virtuelladressraumidentifizierer **1510** nicht proportional zur Anzahl der PPU-Slices **610** zugewiesen, und steht es dem Hypervisor frei, jede Teilmenge der globalen Virtuelladressraumidentifizierer **1510** den PPU-Partitionen **600** oder SMC-Engines **700** zuzuweisen.

[0120] Der Hypervisor **124** allokiert Virtuelladressraumidentifizierer in der beschriebenen Weise, um es verschiedenen SMC-Engines **700** zu ermöglichen, Verarbeitungsaufgaben auszuführen, die einem beliebigen bestimmten Verarbeitungskontext **1400** zugeordnet sind, ohne dass virtuelle Adressen, die durch diese Verarbeitungsaufgaben spezifiziert wurden, neu zugeordnet werden müssen. Demgemäß kann der Hypervisor **124** Verarbeitungskontexte **1400** dynamisch zwischen SMC-Engines **700** migrieren, ohne dass diese Verarbeitungskontexte wesentlich verändert werden müssen. Während der Ausführung verschiedener Verarbeitungsaufgaben, die einem bestimmten Verarbeitungskontext **1400** zugeordnet sind, kann es bei einer beliebigen bestimmten SMC-Engine **700** gelegentlich zu Fehlern kommen, und ist diese dazu konfiguriert, diese Fehler unter Verwendung der lokal zugewiesenen virtuellen Adressen zu melden, wie nachstehend in Verbindung mit Fig. 16 näher beschrieben wird. Nach erfolgreicher Migration verwendet der migrierte Verarbeitungskontext immer noch die gleichen Virtuelladressraumidentifizierer **1500**, aber diese könnten verschiedenen globalen Virtuelladressraumidentifizierern **1510** entsprechen.

[0121] Fig. 16 veranschaulicht, wie eine Speicherverwaltungseinheit lokale Virtuelladressraumidentifizierer **1500** in globale Virtuelladressraumidentifizierer **1510** übersetzt, wenn sie Fehler abschwächt, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, können SMC-Engines **700** während der Ausführung Fehler und/oder Irrtümer erfahren und unabhängig voneinander abstürzen, wie vorangehend diskutiert wurde. In dem gezeigten Beispiel tritt bei der SMC-Engine **700(1)** ein Fehler auf, der die Ausgabe eines lokalen Fehleridentifizierers **1610** an eine Speicherverwaltungseinheit (MMU) **1600** verursacht. Ein Zu-

griff durch eine SMC-Engine **700** auf eine nicht gemappte Seite kann die MMU veranlassen, einen Fehler zu erzeugen, der ebenfalls einen lokalen Fehleridentifizierer verursacht.

[0122] Die MMU **1600** unterhält ein Mapping zwischen lokalen Virtuelladressraumidentifizierern **1500** und den globalen Virtuelladressraumidentifizierern **1510**. Auf der Grundlage dieser Zuordnung generiert die MMU **1600** einen globalen Fehleridentifizierer **1620** und überträgt den globalen Fehleridentifizierer **1620** an das Gastbetriebssystem OS **916(0)**. Im Ansprechen auf den Empfang des globalen Fehleridentifizierers **1620** kann das Gastbetriebssystem **916(0)** die SMC-Engine **700(1)** zurücksetzen, ohne den Betrieb der anderen SMC-Engines **700** zu unterbrechen, und dann den Verarbeitungskontext **1400(1)** neu starten. Bei diesem Ansatz arbeitet jede SMC-Engine **700** mit verschiedenen Sätzen von Virtuelladressraumidentifizierern, die bei null beginnen und sich über potenziell ähnliche Bereiche hinweg erstrecken, aber unterschiedlichen Teilen des globalen Speichers entsprechen. Demgemäß kann der globale Virtuelladressraumidentifizierer **1510** auf die SMC-Engines **700** aufgeteilt werden, aber das Erscheinungsbild eines dedizierten Adressraums bewahren. In einigen Ausführungsformen können die Fehleridentifizierer **1620** für die gesamte PPU-Partition **600** Null sein. In anderen Ausführungsformen kann der Fehleridentifizierer **1620** ein Identifizierer für die SMC-Engine **700** und den Virtuelladressraumidentifizierer **1500** sein.

[0123] In einer Ausführungsform können die globalen Fehleridentifizierer **1620** an den Hypervisor **124** gemeldet werden, und kann der Hypervisor **124** verschiedene Operationen durchführen, um die zugeordneten Fehler zu beheben. In einer anderen Ausführungsform können einige Arten von Fehlern an das zugehörige Gast-OS **916** gemeldet werden, und können andere Arten von Fehlern, wie z. B. harte Fehler, die in dem oberen Abschnitt **810** oder in dem unteren Abschnitt **830** des DRAM **272** auftreten, an der Hypervisor **124** gemeldet werden. Im Ansprechen auf solche Fehler kann der Hypervisor **124** einige oder alle SMC-Engines **700** zurücksetzen. In verschiedenen anderen Ausführungsformen kann ein bestimmter globaler Fehleridentifizierer **1620** virtualisiert sein und daher nicht direkt einem echten globalen Identifizierer entsprechen. Im Betrieb kann die MMU **1600** Fehler an geeignete VMs weiterleiten, auf der Grundlage der diesen VMs zugeordneten GFIDs. GFIDs sind vorstehend in Verbindung mit Fig. 7 diskutiert.

[0124] Allgemein auf die Fig. 15-16 Bezug nehmend, kann der Hypervisor **124** zu den vorstehend beschriebenen analogen Techniken implementieren, um verschiedenen Hardwareressourcen, die jeder PPU-Partition **600** und/oder jeder SMC-Engine **700**

zugeordnet sind, Identifizierer zuzuweisen. Beispielsweise könnte der Hypervisor **124** jedem GPC **242**, der in einer bestimmten PPU-Partition **600** enthalten ist, einen lokalen GPC-Identifizierer (GPC-ID) aus einem Bereich lokaler GPC-IDs zuweisen, der mit Null (**0**) beginnt. Jeder lokale GPC-ID würde einem anderen globalen GPC-ID entsprechen. Dieser Ansatz kann mit jeder PPU-Ressource implementiert werden, um einen Satz von Identifizierern zu unterhalten, der innerhalb einer gegebenen PPU-Partition **600** und/oder SMC-Engine **700** intern konsistent ist. Wie erwähnt wurde, erleichtert dieser Ansatz die Migration von Verarbeitungskontexten **1400** zwischen SMC-Engines **700** und ermöglicht ferner die Migration von Verarbeitungskontexten **1400** zwischen verschiedenen PPUs **200**.

[0125] Wenn der Hypervisor **124** einen Verarbeitungskontext **1400** zwischen verschiedenen SMC-Engines **700** migriert, die sich auf verschiedenen PPUs **200** befinden, führt der Hypervisor **124** eine hier als „soft floorsweeping“ bezeichnete Technik aus, um eine Ziel-PPU **200** mit ähnlichen Hardwareressourcen wie die Quell-PPU **200** zu konfigurieren, wie nachstehend in Verbindung mit **Fig. 17** näher beschrieben wird.

[0126] **Fig. 17** veranschaulicht, wie der Hypervisor von **Fig. 1** bei der Migration eines Verarbeitungskontexts zwischen SMC-Engines auf verschiedenen PPUs soft floorsweeping implementiert, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet eine Rechenumgebung **1700(0)** eine Instanz des Hypervisors **124(0)** und eine PPU-Partition **600(0)**. Die PPU-Partition **600(0)** ist mit einer SMC-Engine **700(0)** konfiguriert. Die SMC-Engine **700(0)** führt Verarbeitungsaufgaben aus, die einem Verarbeitungskontext **1710** zugeordnet sind. Der SMC-Engine **700(0)** sind die Ressourcen **1720(0)** und **1720(1)** zugewiesen, aber die Ressource **1720(1)** ist nicht funktionsfähig. Daher wird die Ressource **1720(1)** während der Herstellung abgeschmolzen („floorswept“). Die Ressource **1720** kann eine beliebige der bisher beschriebenen Rechen-, Grafik- oder Speicherressourcen sein. Beispielsweise könnte eine bestimmte Ressource **1720** unter anderem ein GPC **242**, ein TPC **330** innerhalb eines GPC **242**, ein SM **332** innerhalb eines TPC **330**, ein GFX FE **542** oder ein L2-Cache-Slice **800** sein.

[0127] Unter verschiedenen Umständen kann der Hypervisor **124(0)** bestimmen, dass der Verarbeitungskontext **1710** aus der Rechenumgebung **1700(0)** in die Rechenumgebung **1700(1)** zu migrieren ist. Beispielsweise könnte die Rechenumgebung **1700(0)** für eine geplante Ausfallzeit bzw. Downtime eingeplant sein, und um einen kontinuierlichen Dienst aufrechtzuerhalten, bestimmt der Hypervisor **124(0)**, dass der Verarbeitungskontext **1710** zumindest vorübergehend in eine andere Rechenumgebung zu mi-

grieren ist, während die Rechenumgebung **1700(0)** nicht verfügbar ist.

[0128] In solchen Situationen interagiert der Hypervisor **124(0)** mit einem entsprechenden Hypervisor **124(1)**, der in der Rechenumgebung **1700(1)** ausgeführt wird, um eine PPU-Partition **600(1)** dazu zu konfigurieren, dieselben oder ähnliche Ressourcen wie die PPU-Partition **600(0)** bereitzustellen. Wie gezeigt ist, beinhaltet die PPU-Partition **600(1)** Ressourcen **1720(2)** und **1720(3)**, aber **1720(3)** wird nicht verfügbar gemacht, um die Menge an Ressourcen zu imitieren, die von der PPU-Partition **600(0)** zur Verfügung gestellt wird. Daher kann der Verarbeitungskontext **1710** von der SMC-Engine **700(0)** innerhalb der PPU-Partition **600(0)** zu der SMC-Engine **700(1)** innerhalb der PPU-Partition **600(1)** migriert werden, ohne dass sich die Dienstqualität merklich ändert. Dieser Ansatz trägt dazu bei, den Anschein zu wahren, dass jede beliebige gegebene PPU-Partition **600** auf die gleiche Weise wie die PPU **200** funktioniert, indem der Zugriff auf einen konsistenten Satz von Ressourcen ermöglicht und gleichzeitig die Migration von Verarbeitungskontexten über verschiedene Hardware hinweg ermöglicht wird. Der Hypervisor **124** kann den vorstehenden Ansatz auch für die Migration von SMC-Engines **700** zwischen Partitionen **600** innerhalb derselben PPU **200** implementieren. In einer Ausführungsform können die Hypervisoren **124(0)** und **124(1)** als eine einheitliche Software-Entität ausgeführt werden, die den Betrieb mehrerer PPUs **200** in verschiedenen Rechenumgebungen **1700** verwaltet.

[0129] Allgemein auf die **Fig. 11-17** Bezug nehmend, implementiert der Hypervisor **124** die vorstehend genannten Techniken zur Aufteilung der PPU-Ressourcen in einer Weise, die die Ausführung von Verarbeitungsaufgaben in Verbindung mit mehreren Verarbeitungskontexten gleichzeitig unterstützt. Diese Techniken werden nachstehend in Verbindung mit **Fig. 18** ausführlicher beschrieben.

[0130] **Fig. 18** ist ein Ablaufdiagramm von Verfahrensschritten zur Konfiguration von Rechenressourcen innerhalb einer PPU zur Unterstützung von Operationen, die mehreren Verarbeitungskontexten gleichzeitig zugeordnet sind, gemäß verschiedenen Ausführungsformen. Obwohl die Verfahrensschritte in Verbindung mit den Systemen der **Fig. 1-17** beschrieben werden, versteht es sich für den Fachmann, dass jedes System, das dazu konfiguriert ist, die Verfahrensschritte in beliebiger Reihenfolge auszuführen, in den Anwendungsbereich der vorliegenden Ausführungsformen fällt.

[0131] Wie gezeigt ist, beginnt ein Verfahren **1800** mit Schritt **1802**, bei dem der Hypervisor **124** von **Fig. 1** die PPU **200** auswertet, um einen Satz verfügbarer Hardwareressourcen zu bestimmen. Bestimmte Hardwareressourcen werden bei der Herstellung

einer bestimmten PPU **200** manchmal nicht korrekt hergestellt und können nicht funktionsfähig sein. In der Praxis werden diese nicht funktionsfähigen Hardwareressourcen abgeschmolzen und nicht verwendet. Andere Hardwareressourcen innerhalb der gegebenen PPU **200** sind jedoch funktionsfähig, so dass die PPU **200** als Ganzes weiterhin funktionieren kann, wenn auch mit geringerer Leistung. Die Rettung teilweise funktionsfähiger PPUs und anderer Arten von Einheiten auf die beschriebene Weise ist als „floorsweeping“ bekannt.

[0132] In Schritt **1804** bestimmt der Hypervisor **124** einen Satz verfügbarer swizIDs auf der Grundlage der in Schritt **1802** ermittelten verfügbaren Hardwareressourcen. Wie vorstehend in Verbindung mit **Fig. 11** beschrieben wurde, definiert eine bestimmte swizID einen Satz von Hardwaregrenzen, die aktiviert und deaktiviert werden können, um verschiedene Gruppen von PPU-Slices **610** innerhalb der PPU **200** zu isolieren und PPU-Partitionen **600** zu bilden. In Situationen, in denen bestimmte Hardwareressourcen nicht verfügbar sind, bestimmt der Hypervisor **124**, dass einige swizIDs nicht realisierbaren Partitionskonfigurationen entsprechen und nicht verfügbar zu machen sind.

[0133] In Schritt **1806** generiert der Hypervisor **124** einen Satz von swizIDs auf der Grundlage einer Partitionierungseingabe. Beispielsweise könnte der Hypervisor **124** eine Eingabe von dem Verwaltungsbenuer empfangen, die einen Satz von Partitionierungsoptionen angeben, und diese Partitionierungsoptionen dann einem entsprechenden Satz von swizIDs zuordnen, der aus dem in Schritt **1804** ermittelten Satz verfügbarer swizIDs abgeleitet ist. Alternativ könnte der Hypervisor **124** den Satz von swizIDs direkt von dem Verwaltungsbenuer empfangen und dann beliebige dieser swizIDs, die nicht im Satz verfügbarer swizIDs enthalten sind, modifizieren.

[0134] In Schritt **1808** konfiguriert der Hypervisor **124** eine Reihe von Grenzen zwischen Hardwareressourcen auf der Grundlage des in Schritt **1806** generierten Satzes von swizIDs. Dabei berechnet der Hypervisor **124** eine logische ODER-Verknüpfung über den Satz von swizIDs, um eine Konfigurations-SwizID (oder „lokale“ swizID) zu erzeugen, die angibt, welche Grenzooptionen als Grenzen zu aktivieren sind und welche Grenzooptionen zu deaktivieren sind. Ein beispielhafter Satz von Partitionierungsoptionen und entsprechenden Grenzooptionen ist vorstehend in Verbindung mit **Fig. 12** beschrieben.

[0135] In Schritt **1810** startet ein Gastbetriebssystem **916** eine Reihe von Verarbeitungskontexten innerhalb einer PPU-Partition **600**, die einem Gastbenuer auf der Grundlage zumindest teilweise der einen oder der mehreren swizIDs zugewiesen ist. Der Hypervisor **124** weist der PPU-Partition **600** einen Satz

von Virtualladressraumidentifizierern **1500** zu, der einem Teil der globalen Virtualladressraumidentifizierer **1510** von **Fig. 15** entspricht. Der Hypervisor **124** oder eine SMC-Engine **700** innerhalb dieser PPU-Partition **600** kann den Satz von Virtualladressraumidentifizierern **1500** in verschiedene Bereiche unterteilen, die wiederum verschiedenen Verarbeitungskontexten zugeordnet sind. Dieser Ansatz ermöglicht es jedem Verarbeitungskontext, mit einem konsistenten Satz virtueller Adressräume über alle SMC-Engines **700** hinweg zu arbeiten, wodurch Verarbeitungskontexte leichter migriert werden können.

[0136] Bei Schritt **1812** setzt der Hypervisor **124** oder das entsprechende Gastbetriebssystem **916** im Ansprechen auf einen oder mehrere Fehler eine Teilmenge der bei Schritt **1810** gestarteten Verarbeitungskontexte zurück. Die Fehler könnten u.a. auf der Ebene der Ausführungseinheit, auf der Ebene der SMC-Engine oder auf VM-Ebene auftreten. Wichtig ist, dass Fehler, die während der Ausführung von Verarbeitungsaufgaben, die einem Verarbeitungskontext zugeordnet sind, erzeugt werden, allgemein die Ausführung von Verarbeitungsaufgaben, die anderen Verarbeitungskontexten zugeordnet sind, nicht beeinflussen. Diese Fehlerisolierung zwischen Verarbeitungskontexten befasst sich speziell mit Problemen, die in Ansätzen des Standes der Technik zu finden sind, die sich auf Verarbeitungskontexte stützen. Optional kann zwischen den Schritten **1810** und **1812** ein Debugger aufgerufen werden, um die SMC-Engine **700** zu steuern, bei der ein Fehler aufgetreten ist.

[0137] In Schritt **1814** konfiguriert der Hypervisor **124** ein Migrationsziel auf der Grundlage der verfügbaren Hardwareressourcen, die der PPU **200** zugeordnet sind. Das Migrationsziel kann eine andere PPU **200** sein, aber in einigen Situationen ist das Migrationsziel **200** eine andere SMC **700** innerhalb einer bestimmten PPU-Partition **600** oder eine andere PPU-Partition **600** innerhalb der PPU **200**. Bei der Konfiguration des Migrationsziels kann der Hypervisor **124** eine hierin als „soft floorsweeping“ bezeichnete Technik ausführen, um zu bewirken, dass das Migrationsziel ähnliche Hardwareressourcen bereitstellt wie diejenigen, die von dem Satz von Verarbeitungskontexten verwendet werden.

[0138] In Schritt **1816** migriert der Hypervisor **124** den Satz von Verarbeitungskontexten zu dem Migrationsziel. Die diesen Verarbeitungskontexten zugeordneten Verarbeitungsaufgaben können mit wenig oder keiner Unterbrechung und mit ähnlich verfügbaren Hardwareressourcen fortgesetzt werden. Demgemäß ermöglichen diese Techniken die Bereitstellung einer ausgewogenen Dienstqualität unter Umständen, in denen Verarbeitungskontexte über verschiedene PPU-Partitionen **600** oder verschiedene PPUs **200** hinweg verschoben werden müssen.

[0139] Allgemein auf die **Fig. 11-18** Bezug nehmend, führen der Hypervisor **124**, Gastbetriebssysteme **916** und/oder das Hostbetriebssystem **926** die offenbarten Techniken durch, um die verschiedenen Rechenressourcen, die der PPU **200** zugeordnet sind, in isolierte und unabhängige PPU-Partitionen **600** zu unterteilen, innerhalb derer verschiedene Verarbeitungskontexte gleichzeitig aktiv sein können. Demgemäß können die Ressourcen der PPU **200** im Vergleich zu herkömmlichen Ansätzen, die jeweils nur einen Verarbeitungskontext unterstützen, der die PPU möglicherweise nicht vollständig ausnutzt, effizienter genutzt werden. Die verschiedenen PPU-Partitionen **600** können darüber hinaus von mehreren verschiedenen Inhabern unabhängig voneinander aufgerufen und konfiguriert werden. Auf diese Weise stellen die offenbarten Techniken eine robuste Unterstützung für Multitenancy bzw. Mandantenfähigkeit bereit und können daher die Nachfrage der Verbraucher nach einer effizienten Cloud-basierten Parallelverarbeitungsplattform befriedigen.

Aufteilen von Speicherressourcen zur Unterstützung gleichzeitiger Mehrfachkontexte

[0140] Zusätzlich zur Partitionierung der der PPU **200** zugeordneten Rechenressourcen zur gleichzeitigen Unterstützung mehrerer Verarbeitungskontexte partitioniert der Hypervisor **124** auch die der PPU **200** zugeordneten Speicherressourcen, um mehrere Kontexte gleichzeitig zu unterstützen und somit eine robuste Unterstützung für Mandantenfähigkeit bereitzustellen. Der Hypervisor **124** implementiert verschiedene Techniken bei der Partitionierung von mit der PPU **200** assoziierten Speicherressourcen, die nachstehend in Verbindung mit den **Fig. 19-24** ausführlicher beschrieben werden.

[0141] **Fig. 19** veranschaulicht einen Satz von Grenzoptionen, in Übereinstimmung mit welchen der Hypervisor von **Fig. 1** eine oder mehrere PPU-Speicherpartitionen erzeugen kann, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet der DRAM **272** einen Satz von Grenzoptionen **1900**, die während der Partitionierung aktiviert werden können, um den L2-Cache in verschiedene Abschnitte und Partitionen zu unterteilen.

[0142] Insbesondere Grenzoptionen **1900(0)**, **1900(1)**, **1900(9)** und **1900(10)** unterteilen den DRAM **272** in den oberen Abschnitt **810**, den partitionierbaren Abschnitt **820** und den unteren Abschnitt **830** von **Fig. 8A**. Die Grenzoption **1900(0)** bildet die untere Grenze des unteren Abschnitts **830**, und die Grenzoption **1900(1)** bildet die obere Grenze des unteren Abschnitts **830**. Die Grenzoption **1900(1)** bildet auch die untere Grenze des partitionierbaren Abschnitts **820** sowie die linke Grenze des partitionierbaren Abschnitts **820**. Die Grenzoption **1900(9)** bildet die rechte Grenze des partitionierbaren Ab-

schnitts **820** sowie die obere Grenze des partitionierbaren Abschnitts **820**. Die Grenzoption **1900(9)** bildet auch die untere Grenze des oberen Abschnitts **810**, und die Grenzoption **1900(10)** bildet die obere Grenze des oberen Abschnitts **810**. Die Grenzoptionen **1900(1)** bis **1900(8)** unterteilen den partitionierbaren Abschnitt **820** weiter in verschiedene Speicherpartitionen, die nachstehend in Verbindung mit **Fig. 20** näher beschrieben werden.

[0143] Wie ebenfalls gezeigt ist, hat der DRAM **272** eine Gesamtgröße von M, hat der obere Abschnitt **810** eine Gesamtgröße von T, hat der partitionierbare Abschnitt **820** eine Gesamtgröße von P und hat der untere Abschnitt **830** eine Gesamtgröße von B. Darüber hinaus ist der Anteil eines gegebenen Cache-Slices, der dem partitionierbaren Abschnitt **820** entspricht, durch F gegeben und ist der Anteil des gegebenen Cache-Slices, der dem unteren Abschnitt entspricht, durch W gegeben. F und W sind konfigurierbare Parameter, die über der Hypervisor **124** eingestellt werden können und die in einigen Ausführungsformen die Werte von T, P und B relativ zu M vollständig beschränken können.

[0144] Während der Konfiguration konfiguriert der Hypervisor **124** den DRAM **272** auf der Grundlage von M, F und W in den oberen Abschnitt **810**, den partitionierbaren Abschnitt **820** und den unteren Abschnitt **830**. Dabei ermittelt der Hypervisor **124** Werte für T, P und B auf der Grundlage der Werte von M, F und W. Der Hypervisor **124** aktiviert außerdem bestimmte Grenzoptionen **1900** auf der Grundlage der Konfigurations-SwizID, die über Interaktionen mit dem Verwaltungsbenutzer generiert wird, wie vorstehend in Verbindung mit den **Fig. 11-12** beschrieben wurde. Eine beispielhafte Aktivierung der Grenzoptionen **1900** wird nachstehend in Verbindung mit **Fig. 20** beschrieben.

[0145] **Fig. 20** veranschaulicht ein Beispiel dafür, wie der Hypervisor von **Fig. 1** PPU-Speicher partitioniert, um eine oder mehrere PPU-Speicherpartitionen zu erzeugen, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, werden die Grenzoptionen **1900(0)**, **1900(1)**, **1900(9)** und **1900(10)** aktiviert, wodurch der obere Abschnitt **810**, der partitionierbare Abschnitt **820** und der untere Abschnitt **830** des DRAM **272** ausgebildet werden. Die Grenzoptionen **1900(1)**, **1900(5)**, **1900(7)** und **1900(8)** werden ebenfalls aktiviert, wodurch die PPU-Speicherpartitionen **710(0)**, **710(4)**, **710(6)** und **710(7)** ausgebildet werden, die den DRAM-Abschnitten **822(0)**, **822(4)**, **822(6)** bzw. **822(7)** innerhalb des partitionierbaren Abschnitts **820** entsprechen. Die Grenzoptionen **1900(2)**, **1900(3)**, **1900(4)** und **1900(6)** sind nicht aktiviert und wurden daher weggelassen. Der Hypervisor **124** konfiguriert den DRAM **272** in der gezeigten Weise auf der Grundlage einer Konfigurations-swizID, die gleich „11110100011“ ist.

[0146] Die dem DRAM 272 zugeordneten Grenzooptionen 1900 entsprechen logisch den in den Fig. 11-12 gezeigten Grenzooptionen 1110. Wie vorstehend in Verbindung mit den Fig. 11-12 diskutiert wurde, zeigt jedes Bit einer gegebenen Konfigurations-swizID an, ob eine entsprechende Grenzooption 1110 zu aktivieren oder zu deaktivieren ist, um PPU-Slices 610 zusammenzugruppieren. In gleicher Weise zeigt, wie hier in Fig. 20 gezeigt ist, jedes Bit der beispielhaften Konfigurations-SwizID „11110100011“ an, ob eine entsprechende Grenzooption 1900, die mit dem DRAM 272 assoziiert ist, zu aktivieren oder zu deaktivieren ist.

[0147] Die Bits 0 und 10 der beispielhaften swizID sind standardmäßig auf eins gesetzt, um die Grenzooptionen 1900(0) und 1900(10) zu aktivieren. Die Bits 1 und 9 der beispielhaften swizID sind auf eins gesetzt, um die Grenzooptionen 1900(1) und 1900(9) zu aktivieren und den partitionierbaren Abschnitt 820 einzurichten. Die Bits 5, 7 und 8 der beispielhaften swizID sind auf eins gesetzt, um die Grenzooptionen 1900(5), 1900(7) und 1900(8) zu aktivieren und den partitionierbaren Abschnitt 820 in DRAM-Abschnitte 822 zu unterteilen, die den PPU-Speicherpartitionen 710 zugeordnet sind. Die anderen Bits der swizID sind auf null gesetzt, um die entsprechenden Grenzooptionen zu deaktivieren. Die hier gezeigte Partitionierung des DRAM 272 entspricht der in Fig. 12 gezeigten beispielhaften Konfiguration der PPU-Slices 610. Nach dieser Partitionierung über den Hypervisor 124 können die SMC-Engines 700, die innerhalb der PPU-Partitionen 600 ausgeführt werden, Speicherzugriffsoperationen über L2-Cache-Slices 800 in der nachstehend in Verbindung mit den Fig. 21-23 beschriebenen Weise durchführen.

[0148] Fig. 21 veranschaulicht, wie die Speicherverwaltungseinheit von Fig. 16 Zugriff auf verschiedene PPU-Speicherpartitionen bereitstellt, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, ist die MMU 1600 von Fig. 16 zwischen dem DRAM 272 und dem 1 D-SPA-Raum 850 gekoppelt. Der 1D-SPA-Raum 850 ist in obere Adressen 852, die dem oberen Abschnitt 810 entsprechen, partitionierbare Adressen 854, die dem partitionierbaren Abschnitt 820 entsprechen, und untere Adressen 856, die dem unteren Abschnitt 830 entsprechen, unterteilt, wie ebenfalls in Fig. 8B gezeigt ist. Während der Partitionierung erzeugt der Hypervisor 124 auf der Grundlage der Konfiguration des DRAM 272 den 1D-SPA-Raum 850.

[0149] Die MMU 1600 beinhaltet eine Adresszuordnungseinheit bzw. Adressabbildungseinheit (AMAP) 2110, die dazu konfiguriert ist, die oberen Adressen 852, die partitionierbaren Adressen 854 und die unteren Adressen 858 in Rohadressen umzuwandeln, die dem oberen Abschnitt 810, dem partitionierbaren Abschnitt 820 bzw. dem unteren Abschnitt 830 zuge-

ordnet sind. Auf diese Weise bedient die MMU 1600 Speicherzugriffsanforderungen, die von dem Hypervisor 124 empfangen werden und auf den oberen Abschnitt 810 und/oder den unteren Abschnitt 830 abzielen, sowie Speicherzugriffsanforderungen, die von den SMC-Engines 700 empfangen werden und auf den partitionierbaren Abschnitt 820 abzielen, wie nachstehend in Verbindung mit Fig. 22 näher beschrieben wird.

[0150] Fig. 22 veranschaulicht, wie die Speicherverwaltungseinheit von Fig. 16 verschiedene Adressübersetzungen durchführt, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhalten die partitionierbaren Adressen 854 die Adressbereich 856(0), die Adressen enthält, die der PPU-Speicherpartition 710(0) entsprechen, wie vorstehend in Verbindung mit Fig. 8B diskutiert wurde. Die MMU 1600 übersetzt physische Adressen, die in der Adressbereich 856(0) enthalten sind, über die AMAP 2110 in Rohadressen, die dem DRAM-Abschnitt 822(0) zugeordnet sind. Die AMAP 2110 ist dazu konfiguriert, Adressen aus dem Adressbereich 856(0) über die L2-Cache-Slices 800(0), die in der PPU-Speicherpartition 710(0) enthalten sind, zu verschachteln, um Situationen zu vermeiden, in denen durch Schrittfolgen wiederholt auf denselben L2-Cache-Slice 800(0) zugegriffen wird (auch als „caching“ bekannt).

[0151] In einer Ausführungsform kann die AMAP 2110 eine „Speicherzugriffs“-swizID implementieren, die einen Speicher-Interleave-Faktor bzw. Speicher-verschachtelungsfaktor für einen bestimmten Speicherbereich identifiziert. Eine bestimmte Speicherzugriffs-SwizID bestimmt einen Satz von L2-Cache-Slices, die für verschiedene Arten von Speicherzugriffen, einschließlich Videospeicher-, Systempeicher- und Peerspeicherzugriff, verschachtelt sind. Verschiedene PPU-Partitionen 600 implementieren allgemein unterschiedliche und nicht überlappende Speicherbereiche 822 innerhalb des partitionierbaren Abschnitts 829, um Interferenzen zwischen gleichzeitig ausgeführten Aufträgen zu minimieren. Der Hypervisor 124 kann eine Speicherzugriffs-swizID von Null verwenden, um Speicherzugriffsoperationen über L2-Cache-Slices auszugleichen, die allgemein entweder auf den oberen Abschnitt 810 oder den unteren Abschnitt 830 zugreifen würden.

[0152] Eine gegebene Speicherzugriffs-swizID kann eine „lokale“ swizID sein, die auf der Grundlage einer physischen Systemadresse berechnet wird und dazu dient, Speicherzugriffsanforderungen über relevante L2-Slices und entsprechende Abschnitte des DRAMs zu verschachteln oder zu vertauschen. Eine bestimmte lokale swizID, die einer bestimmten PPU-Partition 600 zugeordnet ist, kann einer swizID entsprechen, die zur Konfiguration dieser PPU-Partition verwendet wird. Mit diesem Ansatz kann die AMAP 2110 die Adressen innerhalb der Grenzen einer be-

stimmten PPU-Speicherpartition auf der Grundlage der SwizID, die zur Aktivierung dieser Grenzen verwendet wurde, vertauschen. Das Vertauschen von Adressen auf der Grundlage von Speicherzugriffs-SwizIDs erlaubt der MMU **1600**, den DRAM **272** so zu verschachteln, dass jede PPU-Partition **600** ihren Teil des partitionierbaren Abschnitts **820** im linearen physischen Systemadressraum **850** als zusammenhängend betrachtet. Dieser Ansatz kann die Isolation zwischen PPU-Partitionen **600** und die Datenintegrität in Verbindung mit diesen PPU-Partitionen **600** aufrechterhalten.

[0153] Eine gegebene Speicherzugriffs-swizID kann alternativ eine „entfernte“ swizID sein, die von dem Gerätetreiber **122** oder dem Hypervisor **124** bereitgestellt wird und dazu dient, Speicherzugriffsanforderungen über L2-Slices für Systemspeicherzugriffsoperationen zu verschachteln. Die lokale swizID und die „entfernte“ swizID können für Verarbeitungsvorgänge, die innerhalb einer bestimmten PPU-Partition **600** stattfinden, identisch sein. Unterschiedliche PPU-Partitionen **600** haben allgemein unterschiedliche Remote-SwizIDs, um zu ermöglichen, dass Systemspeicherzugriffsoperationen nur über L2 Slices **800** laufen, die zu der PPU-Partition **600** gehören.

[0154] Die MMU **1600** stellt auch Unterstützung für die Übersetzung virtueller Adressen, die eine Virtuelladressraumidentifizierer **1500** zugeordnet sind, in eine physische Systemadresse in dem physischen 1 D-Systemadressraum **850** bereit. Beispielsweise sei angenommen, dass die SMC-Engine **700(0)** von **Fig. 7** unter Verwendung der PPU-Speicherpartition **710(0)** und des entsprechenden DRAM-Abschnitts **822(0)** ausgeführt wird und dabei einen Speicherfehler verursacht. Die MMU **1600** würde einen Fehler mit einem lokalen Fehleridentifizierer bzw. einer lokalen Fehlerkennung **1610** ausgeben. Die MMU **1600** würde wiederum den lokalen Fehleridentifizierer **1610** in einen globalen Fehleridentifizierer **1620** übersetzen. Fehler und Störungen können in Übereinstimmung mit der öffentlichen SR-IOV-Spezifikation an virtuelle Funktionen gemeldet werden.

[0155] Die MMU **1600** erleichtert darüber hinaus die Unterteilung der Adressbereiche **856** und der PPU-Speicherpartitionen **710**, um Unterstützung für mehrere SMC-Engines **700**, mehrere VMs und/oder mehrere Verarbeitungskontexte **1400** bereitzustellen, die innerhalb einer bestimmten PPU-Partition **600** ausgeführt werden, wie nachstehend in Verbindung mit **Fig. 23** näher beschrieben wird.

[0156] **Fig. 23** veranschaulicht, wie die Speicherverwaltungseinheit von **Fig. 16** Unterstützung für Operationen bereitstellt, die gleichzeitig mehreren Verarbeitungskontexten zugeordnet sind. Wie gezeigt ist, umfasst der Adressbereich **856(0)** mehrere virtuelle Speicherseiten **2310** unterschiedlicher Größe.

Für die SMC-Engine **700(0)** wird ein Virtuellspeicher-raumidentifizierer **1500** auf einen globalen Virtuellspeicherraumidentifizierer **1510** abgebildet, der die Seitentabelle für einen bestimmten virtuellen Adressraum auswählt, der von einem Verarbeitungskontext auf der SMC-Engine **700(0)** verwendet wird. Durch eine Seitentabelle A spezifizierte Seiten wählen Seiten **2310(A)** innerhalb des DRAM-Abschnitts **822(0)** aus. Gleichzeitig kann die SMC-Engine **700(2)** Seiten verwenden, die durch eine Seitentabelle B spezifiziert sind, die die Seiten **2310(B)** auch innerhalb des DRAM-Abschnitts **822(0)** auswählt. Durch ein seitenbasiertes virtuelles Speicherverwaltungsschema können Seiten innerhalb des DRAM-Abschnitts **822(0)** verschiedenen Unterkontexten oder verschiedenen Verarbeitungskontexten zugewiesen werden. Es wird angemerkt, dass ein Verarbeitungskontext mehrere Virtuelladressraumidentifizierer **1500** verwenden kann, weil er viele Unterkontexte ausführen kann.

[0157] Ein Unterteilen des Adressbereichs **856(0)** und des DRAM-Abschnitts **822(0)** entsprechend PPU-Speicherpartitionen **710(0)** in der gezeigten Weise versorgt unterschiedliche SMC-Engines **700**, die innerhalb einer entsprechenden PPU-Partition **600** ausgeführt werden, mit dedizierten Speicherressourcen innerhalb der PPU-Speicherpartition **822(0)**. Demgemäß können mehrere SMC-Engines **700** in verschiedenen PPU 600-Partitionen Verarbeitungsaufgaben innerhalb verschiedener Verarbeitungskontexte gleichzeitig ausführen, ohne sich gegenseitig in Bezug auf Bandbreite zu stören.

[0158] Der vorstehend beschriebene seitenbasierte Ansatz kann auch auf eine einzelne SMC-Engine **700** angewendet werden, die mehrere Verarbeitungsunterkontexte ausführt, wobei jeder Verarbeitungsunterkontext einen dedizierten Teil der PPU-Speicherpartition **710(0)** benötigt. Ebenso kann der vorstehend beschriebene Ansatz auf verschiedene VMs angewendet werden, die auf einer oder mehreren SMC-Engines **700** ausgeführt werden und dedizierte Teile der PPU-Speicherpartition **710(0)** benötigen.

[0159] Allgemein auf die **Fig. 19-23** Bezug nehmend, ermöglichen die offenbarten Techniken, dass eine bestimmte PPU-Partition **600**, die in der vorstehend in Verbindung mit den **Fig. 11-12** beschriebenen Weise konfiguriert ist, mehrere Verarbeitungskontexte gleichzeitig sicher starten kann. Insbesondere weist die beschriebene Partitionierung des L2-Caches DRAM-Abschnitte **822** gerecht den verschiedenen PPU-Partitionen **600** zu. Ferner nutzen die verschiedenen Adressübersetzungen, die über die MMU **1600** und die AMAP **2110** implementiert werden, die Speicherbandbreite effizient und gerecht, wodurch allen Inhabern der PPU **200** eine konsistente Dienstqualität bereitgestellt wird. Die in Verbindung mit den **Fig. 19-23** beschriebenen Techniken werden

nachstehend in Verbindung mit **Fig. 24** ebenfalls ausführlicher beschrieben.

[0160] **Fig. 24** ist ein Ablaufdiagramm von Verfahrensschritten zur Konfiguration von Speicherressourcen innerhalb einer PPU zur Unterstützung von Operationen, die mehreren Verarbeitungskontexten gleichzeitig zugeordnet sind, gemäß verschiedenen Ausführungsformen. Obwohl die Verfahrensschritte in Verbindung mit den Systemen der **Fig. 1-23** beschrieben werden, versteht es sich für den Fachmann, dass jedes beliebige System, das dazu konfiguriert ist, die Verfahrensschritte in beliebiger Reihenfolge auszuführen, in den Anwendungsbereich der vorliegenden Ausführungsformen fällt.

[0161] Wie gezeigt ist, beginnt eine Methode **2400** mit Schritt **2402**, in dem der Hypervisor **124** von **Fig. 1** einen Satz von Speicherkonfigurationsparametern bestimmt, mit welchen der DRAM **272** zu partitionieren ist. Der Satz von Speicherkonfigurationsparametern kann jeden beliebigen technisch machbaren Satz von Parametern angeben, die ein beliebiges Attribut des DRAM **272** beschreibt, einschließlich der Gesamtgröße des DRAM **272** (M), der Größe T des oberen Abschnitts **810**, der Größe P des partitionierbaren Abschnitts **820**, der Größe B des unteren Abschnitts **830**, der Anzahl von L2-Cache-Slices 800 (z. B. 96), der Größe jedes Cache-Slice-Abschnitts F, der dem partitionierbaren Abschnitt **820** entspricht, und/oder der Größe jedes Cache-Slice-Abschnitts W, der dem unteren Abschnitt **830** entspricht. In einer Ausführungsform braucht der Satz von Konfigurationsparametern nur die Parameter M, F und W zu enthalten.

[0162] In Schritt **2404** aktiviert der Hypervisor **124** einen ersten Satz von Grenzoptionen auf der Grundlage der in Schritt **2402** bestimmten Speicherkonfigurationsparameter, um den DRAM **272** in Abschnitte zu unterteilen. Insbesondere aktiviert der Hypervisor **124** die in **Fig. 19** dargestellten Grenzoptionen **1900(0)**, **1900(1)**, **1900(9)** und **1900(10)**, um den DRAM **272** in den oberen Abschnitt **810**, den partitionierbaren Abschnitt **820** und den unteren Abschnitt **820** zu unterteilen. In einer Ausführungsform kann der Hypervisor **124** die Platzierung einer gegebenen Grenzoption ändern, um die Größe eines entsprechenden Abschnitts des DRAM **272** anzupassen.

[0163] In Schritt **2406** bestimmt der Hypervisor **124** eine Konfigurations-swizID basierend auf der Partitionierungseingabe. Die Partitionierungseingabe kann über Schritt **1002** des vorstehend in Verbindung mit **Fig. 10** beschriebenen Verfahrens **1000** erhalten werden. Die Partitionierungseingabe gibt einen Satz von Ziel-PPU-Partitionen für die PPU **200** an. Der Hypervisor **124** kann eine Konfigurations-swizID für den Zielsatz von Partitionen über die vorstehend in Verbindung mit **Fig. 11-12** beschriebenen Techniken be-

stimmen. In einer Ausführungsform kann die Konfigurations-SwizID vor Schritt **2404** ermittelt werden, und kann dann der erste Satz von Grenzoptionen auf der Grundlage dieser Konfigurations-SwizID aktiviert werden.

[0164] In Schritt **2408** aktiviert der Hypervisor einen zweiten Satz von Grenzoptionen auf der Grundlage der Konfigurations-swizID, um eine oder mehrere PPU-Speicherpartitionen **710** innerhalb des partitionierbaren Abschnitts **820** des DRAM **272** zu erzeugen. Der zweite Satz von Grenzoptionen kann jede der in **Fig. 19** gezeigten Grenzoptionen **1900(2)** bis **1900(8)** beinhalten. Diese verschiedenen Grenzoptionen können den partitionierbaren Abschnitt **820** in eine den PPU-Speicherpartitionen **710** entsprechende Anzahl von DRAM-Abschnitten **822** unterteilen, die gleich oder kleiner als die Anzahl von PPU-Slices **610** ist. In verschiedenen Ausführungsformen können die Schritte **2404** und **2408** des Verfahrens **2400** in Verbindung miteinander auf der Grundlage einer Konfigurations-swizID durchgeführt werden, die über eine Verwaltungsbenutzereingabe erhalten oder auf der Grundlage dieser Eingabe generiert wurde.

[0165] In Schritt **2410** bestimmt der Hypervisor **124** einen Satz von partitionierbaren Adressen **854** auf der Grundlage des Satzes von Speicherkonfigurationsparametern und/oder der Konfigurations-SwizID. Dabei unterteilt der Hypervisor **124** einen 1D-SPA-Raum **850** in die oberen Adressen **852**, die partitionierbaren Adressen **854** und die untere Adresse **858**, wie in **Fig. 21** gezeigt ist. Die oberen Adressen **852** können in Rohadressen übersetzt werden, die dem oberen Abschnitt **810** zugeordnet sind, die partitionierbaren Adressen **854** können in Rohadressen übersetzt werden, die dem partitionierbaren Abschnitt **820** zugeordnet sind, und die unteren Adressen **856** können in Rohadressen übersetzt werden, die dem unteren Abschnitt **830** zugeordnet sind.

[0166] In Schritt **2412** bedient die MMU **1600** Speicherzugriffsanforderungen durch Vertauschen partitionierbarer Adressen **854** über die L2-Cache-Slices 800 innerhalb einer PPU-Speicherpartition **710** entsprechend einem DRAM-Abschnitt **822**. Die MMU **1600** vertauscht partitionierbare Adressen über die AMAP **2110** auf der Grundlage einer Speicherzugriffs-SwizID (oder „entfernten“ SwizID), die der PPU-Speicherpartition **710** zugeordnet ist. In einer Ausführungsform wird die Speicherzugriffs-SwizID von einer swizID abgeleitet, in Übereinstimmung mit der die PPU-Speicherpartition **710** konfiguriert ist. Ein Vertauschen partitionierbarer Adressen **854** auf diese Weise kann einen wiederholten Zugriff auf einzelne L2-Cache-Slices um **800** (auch als „camping“ bezeichnet) verringern.

[0167] In Schritt **2414** empfängt die MMU **1600** einen lokalen Fehleridentifizierer, der einem Speicher-

fehler zugeordnet ist, und übersetzt den lokalen Fehleridentifizierer in einen globalen Fehleridentifizierer. Dabei kann die MMU **1600** eine virtuelle Adresse, die dem lokalen Fehleridentifizierer zugeordnet ist, in eine globale Adresse übersetzen, die dem globalen Fehleridentifizierer zugeordnet ist. Der Speicherfehler könnte z.B. verursacht werden, wenn eine bestimmte SMC-Engine **700** während eines Speicherlesevorgangs oder eines Speicherschreibvorgangs, der mit einer PPU-Speicherpartition **710** ausgeführt wird, auf einen Fehler stößt. Die Implementierung von Fehler-IDs in einem lokalen virtuellen Adressraum ermöglicht es den SMC-Engines **700**, mit ähnlichen Adressräumen zu arbeiten, wodurch eine einfachere Migration von Verarbeitungskontexten zwischen PPU-Partitionen **600** ermöglicht wird, wie vorstehend in Verbindung mit **Fig. 17** beschrieben wurde. Die Übersetzung dieser Fehler-IDs in einen globalen Fehleridentifizierer **1620** erlaubt es dem Hypervisor **124**, Fehler, die diesen Fehler-IDs entsprechen, aus der Perspektive eines globalen Virtualladressraumidentifizierers **1510** zu adressieren.

[0168] Allgemein auf die **Fig. 19-24** Bezug nehmend, ergänzen die offenbarten Techniken zur Partitionierung von PPU-Speicherressourcen die vorstehend in Verbindung mit den **Fig. 11-18** beschriebenen Techniken zur Partitionierung von PPU-Rechenressourcen. Über diese Techniken kann der Hypervisor **124** eine PPU **200** so konfigurieren und partitionieren, dass mehrere Verarbeitungskontexte gleichzeitig unterstützt werden. Mit dieser Funktionalität kann die PPU **200** eine Vielzahl unterschiedlicher Verarbeitungsaufgaben im Namen mehrerer CPU-Prozesse sicher ausführen, ohne dass sich diese CPU-Prozesse gegenseitig stören können, wodurch PPU-Ressourcen effizienter genutzt werden können. Darüber hinaus können die offenbarten Techniken angewandt werden, um robuste Unterstützung für Mandantenfähigkeit in Cloud-basierten PPU-Bereitstellungen bereitzustellen und dadurch eine Produktnachfrage zu erfüllen, die in der Vergangenheit von früheren Ansätzen nicht erfüllt wurde.

Unterteilung in Zeitscheiben mehrerer VMs und Verarbeitungskontexte

[0169] Wie hierin weiter diskutiert wurde, unterstützt die PPU **200** von **Fig. 2** zwei Partitionierungsebenen. In einer ersten Partitionierungsebene, die hierin als „PPU-Partitionierung“ bezeichnet wird, werden die PPU-Ressourcen **500** der PPU **200** in PPU-Partitionen **600** partitioniert, die hier auch als „fraktionierte PPU“ bezeichnet werden. In einigen Ausführungsformen können der PP-Speicher **270**, der DRAM **272** oder beide in SMC-Speicherpartitionen **710** partitioniert sein. Auf dieser Partitionierungsebene führt jede PPU-Partition **600** jeweils eine VM zu einem bestimmten Zeitpunkt aus. In einer zweiten Partitionierungsebene, die hier als „SMC-Partitionierung“ be-

zeichnet wird, wird jede PPU-Partition **600** weiter in SMC-Engines **700** unterteilt. Jede PPU-Partition **600** beinhaltet eine oder mehrere SMC-Engines **700**. Auf dieser Partitionierungsebene führt jede SMC-Engine **700** jeweils einen Verarbeitungskontext für eine VM zu einem bestimmten Zeitpunkt aus.

[0170] Mit der Zeit wechseln SMC-Engines **700** von der Ausführung eines bestimmten Verarbeitungskontexts für eine VM zur Ausführung eines anderen Verarbeitungskontexts für dieselbe VM oder zur Ausführung eines anderen Verarbeitungskontexts für eine andere VM. Dieser Prozess wird hierin als „Unterteilung in Zeitscheiben“ bezeichnet, weil die Ausführungszeit für die SMC-Engines **700** auf mehrere Verarbeitungskontexte „unterteilt“ bzw. „aufgeteilt“ wird, die einer oder mehreren VMs entsprechen.

[0171] Jede SMC-Engine **700** unterteilt in Zeitscheiben zwischen Verarbeitungskontexten, die auf einer Ablaufliste aufgeführt sind, wie sie von den PBDMAAs **520** und **522** der SMC-Engine **700** verwaltet wird. Allgemein werden dann, wenn zwischen VMs umgeschaltet wird, die Ablauflisten auf allen betroffenen SMC-Engines **700** ersetzt, so dass ein anderer Satz von Verarbeitungskontexten in Zeitscheiben unterteilt wird. Falls mehrere SMC-Engines **700** aktiv sind, werden die Ablauflisten gleichzeitig ersetzt. Diese Art der Einplanung durch Ersetzen von Ablauflisten wird hierin als „Software-Scheduling“ bzw. „Software-Einplanung“ bezeichnet. Das Umschalten zwischen Verarbeitungskontexten innerhalb derselben VM kann in ähnlicher Weise ebenfalls das Ersetzen von Ablauflisten beinhalten und ähnelt sehr dem Umschalten von VMs, außer dass sich die VM durch den Kontextwechsel nicht ändert. Infolgedessen ist für diese Art des Kontextwechsels keine zusätzliche Hardware-Unterstützung erforderlich. In einigen Ausführungsformen können VMs zahlreiche unterschiedliche Verarbeitungskontexte unterschiedlicher Größe haben. In solchen Ausführungsformen kann die Software-Einplanung überlegen, wie diese Verarbeitungskontexte zur korrekten und effizienten Ausführung in SMC-Engines **700** gepackt werden können. Außerdem kann die Software-Einplanung die Anzahl der PPU-Partitionen **600** und die Anzahl der SMC-Engines innerhalb jeder PPU-Partition **600** rekonfigurieren, um die Verarbeitungskontexte für die VM korrekt und effizient auszuführen.

[0172] Damit die PPU-Ressourcen **500** die beiden vorstehend beschriebenen Partitionierungsebenen unterstützen können, unterstützt die PPU **200** entsprechend zwei Ebenen der Unterteilung in Zeitscheiben. Entsprechend der PPU-Partitionierung führt die PPU **200** eine Unterteilung in Zeitscheiben auf VM-Ebene durch, wobei jede PPU-Partition **600** zwischen mehreren virtuellen Maschinen in Zeitscheiben unterteilt. Entsprechend der SMC-Partitionierung führt die PPU **200** eine Unterteilung in Zeitscheiben auf SMC-

Ebene durch, wobei jede VM **600** zwischen mehreren Verarbeitungskontexten in Zeitscheiben unterteilt. In verschiedenen Ausführungsformen behalten beide Ebenen der Unterteilung in Zeitscheiben die gleiche Anzahl von TPCs in jedem GPC **242** über die Zeit bei. In verschiedenen Ausführungsformen kann die Unterteilung in Zeitscheiben die Änderung der Anzahl von TPCs in einem oder mehreren GPCs **242** über die Zeit involvieren. Die beiden Ebenen der Unterteilung in Zeitscheiben werden nun beschrieben.

[0173] Fig. 25 ist ein Satz von eine Unterteilung in Zeitscheiben auf VM-Ebene veranschaulichenden Zeitleisten **2500**, die PPU-Partitionen **600** der PPU **200** von Fig. 2 zugeordnet sind, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet der Satz von Zeitleisten **2500**, ohne darauf beschränkt zu sein, vier PPU-Partitions-Zeitleisten **2502(0)**, **2502(4)**, **2502(6)** und **2502(7)**. In einigen Ausführungsformen können die PPU-Partitions-Zeitleisten **2502(0)**, **2502(4)**, **2502(6)** und **2502(7)** den PPU-Partitionen **600(0)**, **600(4)**, **600(6)** bzw. **600(7)** von Fig. 6 entsprechen. In solchen Ausführungsformen kann die PPU-Partitions-Zeitleiste **2502(0)** den PPU-Slices **610(0)-610(3)** entsprechen, und kann die PPU-Partitions-Zeitleiste **2502(4)** den PPU-Slices **610(4)-610(5)** entsprechen. In ähnlicher Weise kann die PPU-Partitions-Zeitleiste **2502(6)** dem PPU-Slice **610(6)** entsprechen, und kann die PPU-Partitions-Zeitleiste **2502(7)** dem PPU-Slice **610(7)** entsprechen. Infolgedessen kann die PPU-Partition **600(0)** bis zu vier Verarbeitungskontexte gleichzeitig ausführen, kann die PPU-Partition **600(4)** bis zu zwei Verarbeitungskontexte gleichzeitig ausführen, und kann jede der PPU-Partitionen **600(6)** und **600(7)** jeweils einen Verarbeitungskontext gleichzeitig ausführen.

[0174] Wie in der PPU-Partitions-Zeitleiste **2502(0)** gezeigt ist, unterteilt die PPU-Partition **600(0)** zwischen zwei VMs, die als VM A und VM B bezeichnet werden, in Zeitscheiben. Die Zeitleiste **2502(0)** veranschaulicht die Unterteilung in Zeitscheiben zwischen der VM A, bei der der VM A zugeordnete Verarbeitungskontexte in Form eines KONTEXT 2510 (Ax-y) gezeigt sind, und der VM B, bei der der mit VM B zugeordnete Verarbeitungskontexte in Form eines KONTEXT 2510 (Bx-y) gezeigt sind. Von einem Zeitpunkt t0 bis zu einem Zeitpunkt t1 führt die PPU-Partition **600(0)** die Verarbeitungskontexte der VM A aus. Eine erste SMC-Engine **700(0)** der PPU-Partition **600(0)** führt sequentiell den Verarbeitungskontext 2510(A0-1), den Verarbeitungskontext 2510(A1-1) und den Verarbeitungskontext 2510(A2-1) aus. Gleichzeitig führt eine zweite SMC-Engine **700(2)** der PPU-Partition **600(0)** sequentiell den Verarbeitungskontext 2510(A3-1), den Verarbeitungskontext 2510(A4-1) und den Verarbeitungskontext 2510(A5-1) aus. Zum Zeitpunkt t1 stoppt die PPU-Partition **600(0)** die Ausführung der Verarbeitungskontexte, die der VM A zugeordnet sind, und wechselt zu Ver-

arbeitungskontexten, die der VM B zugeordnet sind. Der Verarbeitungskontext 2510(A2-1) und der Verarbeitungskontext 2510(A5-1) werden aus den entsprechenden SMC-Engines **700(0)** und **700(2)** heraus kontextgeschaltet. Die PPU-Partition **600(0)** wird von zwei SMC-Engines **700(0)**, einschließlich zweier GPCs **230(0)** und **230(1)**, und **700(2)**, einschließlich zweier GPCs **230(2)** und **230(3)**, aufweisend zu einer SMC-Engine **700(0)**, einschließlich vier GPCs **230(0)**, **230(1)**, **230(2)** und **230(3)**, aufweisend umkonfiguriert. Sobald die Umkonfiguration abgeschlossen ist, beginnt die PPU-Partition **600(0)** mit der Ausführung des Verarbeitungskontexts 2510(B0-1) der VM B.

[0175] Vom Zeitpunkt t1 bis zu einem Zeitpunkt t4 führt die PPU-Partition **600(0)** die Verarbeitungskontexte der VM B aus. Eine erste SMC-Engine **700(0)** der PPU-Partition **600(0)** führt sequentiell den Verarbeitungskontext 2510(B0-1), den Verarbeitungskontext 2510(B1-1) und den Verarbeitungskontext 2510(B2-1) aus. Zum Zeitpunkt t4 stoppt die PPU-Partition **600(0)** die Ausführung der der VM B zugeordneten Verarbeitungskontexte und wechselt zu den der VM A zugeordneten Verarbeitungskontexten. Der Verarbeitungskontext 2510(B2-1) wird aus der entsprechenden SMC-Engine **700(0)** heraus kontextgeschaltet. Die PPU-Partition **600(0)** wird von einer SMC-Engine **700(0)**, einschließlich vier GPCs **230(0)**, **230(1)**, **230(2)** und **230(3)**, aufweisend auf zwei SMC-Engines **700(0)**, einschließlich zwei GPCs **230(0)** und **230(1)**, und **700(2)**, einschließlich zwei GPCs **230(2)** und **230(3)**, aufweisend umkonfiguriert. Sobald die Umkonfiguration abgeschlossen ist, beginnt die PPU-Partition **600(0)** mit der Ausführung der Verarbeitungskontexte 2510(A2-2) und 2510(A5-2) der VM A. Ab dem Zeitpunkt t4 führt die PPU-Partition **600(0)** erneut die Verarbeitungskontexte der VM A aus. Eine erste SMC-Engine **700(0)** der PPU-Partition **600(0)** führt sequentiell den Verarbeitungskontext 2510(A2-2) und den Verarbeitungskontext 2510(A0-2) aus. Gleichzeitig führt eine zweite SMC-Engine **700(2)** der PPU-Partition **600(0)** sequentiell den Verarbeitungskontext 2510(A5-2) und den Verarbeitungskontext 2510(A3-2) aus.

[0176] Wie in der PPU-Partitions-Zeitleiste **2502(4)** gezeigt ist, unterteilt die PPU-Partition **600(4)** zwischen zwei VMs, die als VM C und VM D bezeichnet werden, in Zeitscheiben. Die Zeitleiste **2502(4)** veranschaulicht die Unterteilung in Zeitscheiben zwischen der VM C, wobei die der VM C zugeordneten Verarbeitungskontexte in Form von KONTEXT 2510 (Cx-y) dargestellt sind, und der VM D, wobei die der VM D zugeordneten Verarbeitungskontexte in Form von KONTEXT 2510 (Dx-y) dargestellt sind. Vom Zeitpunkt t0 bis zu einem Zeitpunkt t2 ist die PPU-Partition **600(4)** inaktiv und führt keine Verarbeitungskontexte aus. Vom Zeitpunkt t2 bis zu einem Zeitpunkt t5 führt die PPU-Partition **600(4)** Verarbei-

tungskontexte der VM C aus. Eine erste SMC-Engine **700(4)** der PPU-Partition **600(4)** führt sequentiell den Verarbeitungskontext 2510(C0-1) und den Verarbeitungskontext 2510(C1-1) aus. Zum Zeitpunkt t5 beendet die PPU-Partition **600(4)** die Ausführung des Verarbeitungskontexts 2510(C1-1) und konfiguriert sich neu, um mit der Ausführung von Verarbeitungskontexten der VM D zu beginnen. Ab dem Zeitpunkt t5 führt die PPU-Partition **600(4)** Verarbeitungskontexte der VM D aus. Eine erste SMC-Engine **700(4)** der PPU-Partition **600(4)** führt den Verarbeitungskontext 2510(D1-1) aus.

[0177] Wie in der PPU-Partitions-Zeitleiste **2502(6)** gezeigt ist, unterteilt die PPU-Partition **600(6)** zwischen zwei VMs, die als VM E und VM F bezeichnet werden, in Zeitscheiben. Die Zeitleiste **2502(6)** veranschaulicht die Unterteilung in Zeitscheiben zwischen der VM E, wobei die der VM E zugeordneten Verarbeitungskontexte in Form von KONTEXT 2510 (Ex-y) gezeigt sind, und der VM F, wobei die der VM F zugeordneten Verarbeitungskontexte in Form von KONTEXT 2510(Fx-y) gezeigt sind. Vom Zeitpunkt t0 bis zu einem Zeitpunkt t3 ist die PPU-Partition **600(6)** inaktiv und führt keine Verarbeitungskontexte aus. Vom Zeitpunkt t3 bis zu einem Zeitpunkt t6 führt die PPU-Partition **600(6)** Verarbeitungskontexte der VM E aus. Eine erste SMC-Engine **700(6)** der PPU-Partition **600(6)** führt sequentiell den Verarbeitungskontext 2510(E0-1) und den Verarbeitungskontext 2510 (E1-1) aus. Zum Zeitpunkt t6 beendet die PPU-Partition **600(6)** die Ausführung des Verarbeitungskontexts 2510(E1-1) und konfiguriert sich neu, um mit der Ausführung von Verarbeitungskontexten der VM F zu beginnen. Ab dem Zeitpunkt t6 führt die PPU-Partition **600(4)** Verarbeitungskontexte der VM F aus. Eine erste SMC-Engine **700(6)** der PPU-Partition **600(6)** führt den Verarbeitungskontext 2510(F0-1) aus.

[0178] Wie in der PPU-Partitions-Zeitleiste **2502(7)** gezeigt ist, unterteilt die PPU-Partition **600(7)** innerhalb einer einzelnen VM, die als VM G bezeichnet wird, in Zeitscheiben. Die Zeitleiste **2502(6)** veranschaulicht die Unterteilung in Zeitscheiben für die VM G, wobei die der VM G zugeordneten Verarbeitungskontexte in Form von KONTEXT 2510(Gx-y) gezeigt sind. Vom Zeitpunkt t0 bis zum Zeitpunkt t5 ist die PPU-Partition **600(7)** inaktiv und führt keine Verarbeitungskontexte aus. Ab dem Zeitpunkt t5 führt die PPU-Partition **600(7)** den Verarbeitungskontext G aus. Eine erste SMC-Engine **700(7)** der PPU-Partition **600(7)** führt den Verarbeitungskontext 2510(G0-1) aus.

[0179] Auf diese Weise unterteilt jede der PPU-Partitionen **600(0)**, **600(4)**, **600(6)** und **600(7)** in Zeitscheiben zwischen Verarbeitungskontexten entsprechend einer oder mehreren VMs. Jede der PPU-Partitionen **600(0)**, **600(4)**, **600(6)** und **600(7)** geht unabhängig voneinander von einem Verarbeitungskontext zu einem anderen Verarbeitungskontext über.

Beispielsweise könnte die PPU-Partition **600(0)** von der Ausführung eines Verarbeitungskontexts für eine bestimmte VM zu einem anderen Verarbeitungskontext für dieselbe oder eine andere VM wechseln, ohne Rücksicht darauf, ob eine oder mehrere der PPU-Partitionen **600(4)**, **600(6)** und **600(7)** den Verarbeitungskontext wechseln oder nicht. Während des dargestellten Zeitraums behält jede der PPU-Partitionen **600(0)**, **600(4)**, **600(6)** und **600(7)** eine konstante Anzahl von PPU-Slices **610** bei. In einigen Ausführungsformen kann sich die Anzahl der PPU-Slices **610** in jeder PPU-Partition **600** ändern, wie nun beschrieben wird.

[0180] Fig. 26 ist eine weiterer Satz von Zeitleisten **2600**, die das Unterteilen in Zeitintervalle auf VM-Ebene in Zuordnung mit der PPU **200** von Fig. 2 veranschaulicht, gemäß verschiedenen anderen Ausführungsformen. Die in dem Satz von Zeitleisten **2600** veranschaulichten Verarbeitungskontexte funktionieren im Wesentlichen genauso wie der Satz von Zeitleisten **2500** von Fig. 25, ausgenommen wie nachstehend beschrieben. Wie gezeigt ist, beinhaltet der Satz von Zeitleisten **2600**, ohne darauf beschränkt zu sein, vier PPU-Partitions-Zeitleisten **2602(0)**, **2602(4)**, **2602(6)** und **2602(7)**. In einigen Ausführungsformen können die PPU-Partitions-Zeitleisten **2602(0)**, **2602(4)**, **2602(6)** und **2602(7)** den PPU-Partitionen **600(0)**, **600(4)**, **600(6)** bzw. **600(7)** von Fig. 6 entsprechen.

[0181] Wie in der PPU-Partitions-Zeitleiste **2602(0)** gezeigt ist, unterteilt die PPU-Partition **600(0)** zwischen zwei VMs, die als VM A und VM B bezeichnet werden, in Zeitscheiben. Die Zeitleiste **2602(0)** veranschaulicht die Unterteilung in Zeitscheiben zwischen der VM A, wobei der VM A zugeordnete Verarbeitungskontexte in Form von KONTEXT 2610(Ax-y) gezeigt sind, und der VM B, wobei der VM B zugeordnete Verarbeitungskontexte in Form von KONTEXT 2610(Bx-y) gezeigt sind. Vom Zeitpunkt t0 bis zum Zeitpunkt t1 führt die PPU-Partition **600(0)** Verarbeitungskontexte der VM B aus. Eine erste SMC-Engine **700(0)** der PPU-Partition **600(0)** führt sequentiell den Verarbeitungskontext 2610(B1-1) und den Verarbeitungskontext 2610(B2-1) aus. Zum Zeitpunkt t1 stoppt die PPU-Partition **600(0)** die Ausführung der mit VM B zugeordneten Verarbeitungskontexte und wechselt zu den mit VM A zugeordneten Verarbeitungskontexten. Der Verarbeitungskontext 2610(B2-1) wird aus der entsprechenden SMC-Engine **700(0)** heraus kontextgeschaltet. Die PPU-Partition **600(0)** wird von einer SMC-Engine **700(0)**, einschließlich vier GPCs **230(0)**, **230(1)**, **230(2)** und **230(3)**, aufweisend auf zwei SMC-Engines **700(0)**, einschließlich zwei GPCs **230(0)** und **230(1)**, und **700(2)**, einschließlich zwei GPCs **230(2)** und **230(3)**, aufweisend umkonfiguriert. Sobald die Umkonfiguration abgeschlossen ist, beginnt die PPU-Partition **600(0)** mit

der Ausführung von Verarbeitungskontexten der VM A. Vom Zeitpunkt t1 bis zum Zeitpunkt t3 führt die PPU-Partition **600(0)** Verarbeitungskontexte von A aus. Eine erste SMC-Engine **700(0)** der PPU-Partition **600(0)** führt sequentiell den Verarbeitungskontext 2610(A2-1), den Verarbeitungskontext 2610(A0-1) und den Verarbeitungskontext 2610(A1-1) aus. Gleichzeitig führt eine zweite SMC-Engine **700(2)** der PPU-Partition **600(0)** sequentiell den Verarbeitungskontext 2610(A5-1), den Verarbeitungskontext 2610(A3-1) und den Verarbeitungskontext 2610(A4-1) aus. Zum Zeitpunkt t3 stoppt die PPU-Partition **600(0)** die Ausführung der Verarbeitungskontexte, die der VM A zugeordnet sind, und wechselt zu Verarbeitungskontexten, die der VM B zugeordnet sind. Der Verarbeitungskontext 2610(A1-1) und der Verarbeitungskontext 2610(A4-1) werden aus den entsprechenden SMC-Engines **700(0)** und **700(2)** heraus kontextgeschaltet. Die PPU-Partition **600(0)** wird von zwei SMC-Engines **700(0)**, einschließlich zweier GPCs **230(0)** und **230(1)**, und **700(2)**, einschließlich zweier GPCs **230(2)** und **230(3)**, aufweisend auf eine SMC-Engine **700(0)**, einschließlich vier GPCs **230(0)**, **230(1)**, **230(2)** und **230(3)**, aufweisend umkonfiguriert. Sobald die Rekonfiguration abgeschlossen ist, beginnt die PPU-Partition **600(0)** mit der Ausführung von Verarbeitungskontexten der VM B. Ab dem Zeitpunkt t3 führt die PPU-Partition **600(0)** erneut Verarbeitungskontexte der VM B aus. Eine erste SMC-Engine **700(0)** der PPU-Partition **600(0)** führt sequentiell den Verarbeitungskontext 2610(B2-2) und den Verarbeitungskontext 2610(B0-1) aus.

[0182] Wie in der PPU-Partitions-Zeitleiste **2602(4)** gezeigt ist, führt von dem Zeitpunkt t0 bis zum Zeitpunkt T2 eine erste SMC-Engine **700(4)** der PPU-Partition **600(4)** sequentiell den Verarbeitungskontext 2610(D0-1) und den Verarbeitungskontext 2610(D1-1) aus. Die erste SMC-Engine **700(4)** der PPU-Partition **600(4)** befindet sich dann im Leerlauf. Wie in der PPU-Partitions-Zeitleiste **2602(6)** gezeigt ist, führt eine erste SMC-Engine **700(6)** der PPU-Partition **600(6)** vom Zeitpunkt t0 bis zum Zeitpunkt t2 sequentiell den Verarbeitungskontext 2610(F0-1) und den Verarbeitungskontext 2610(F1-1) aus. Die erste SMC-Engine **700(6)** der PPU-Partition **600(6)** befindet sich dann im Leerlauf. Wie in der Zeitleiste **2602(7)** der PPU-Partition gezeigt ist, befindet sich die PPU-Partition **600(7)** vom Zeitpunkt t0 bis zum Zeitpunkt t2 im Leerlauf.

[0183] Zum Zeitpunkt t2 verschmelzen die PPU-Partitionen **600(4)**, **600(6)** und **600(7)** zu einer einzigen PPU-Partition **600(4)** mit vier SMC-Engines **700(4)**-**700(7)**. Wie in der PPU-Partitions-Zeitleiste **2604(4)** gezeigt ist, führt die zusammengeführte PPU-Partition **600(4)** Verarbeitungskontexte von einer VM H aus. Beginnend zum Zeitpunkt t2 führt eine erste SMC-Engine **700(4)** der PPU-Partition **600(4)** sequentiell den Verarbeitungskontext 2610

(H0-1), den Verarbeitungskontext 2610(H1-1), den Verarbeitungskontext 2610(H2-1) und den Verarbeitungskontext 2610(H0-2) aus.

[0184] Auf diese Weise können die PPU-Partitionen **600** für Partitionen unterschiedlicher Größe während der Unterteilung in Zeitscheiben zusammengeführt und/oder aufgeteilt werden. Die PPU-Partitionen **600** können unabhängig voneinander zusammengeführt und/oder aufgeteilt werden. In den **Fig. 25** und **Fig. 26** wird jede VM auf einer festen Anzahl von SMC-Engines **700** ausgeführt, was zu einer konstanten Anzahl von gleichzeitig ausgeführten Verarbeitungskontexten für eine bestimmte VM führt. Wie gezeigt ist, wird die VM A gleichzeitig auf zwei SMC-Engines **700** ausgeführt, während die übrigen VMs auf jeweils einer SMC-Engine **700** ausgeführt werden. In einigen Ausführungsformen kann eine bestimmte VM die Anzahl der SMC-Engines **700** ändern, auf denen die VM ausgeführt wird, wie nun beschrieben wird.

[0185] **Fig. 27** ist eine Zeitleiste **2700**, die die Unterteilung in Zeitscheiben auf SMC-Ebene in Verbindung mit der PPU **200** von **Fig. 2** veranschaulicht, gemäß verschiedenen Ausführungsformen. Die in der Zeitleiste **2700** dargestellten Verarbeitungskontexte funktionieren im Wesentlichen genauso wie jeweils die Sätze von Zeitleisten **2500** und **2600** der **Fig. 25** und **Fig. 26**, ausgenommen wie nachstehend weiter beschrieben. Wie gezeigt ist, repräsentiert die Zeitleiste **2700** eine einzelne PPU-Partitions-Zeitleiste. In einigen Ausführungsformen kann die PPU-Partitions-Zeitleiste, die durch die Zeitleiste **2700** repräsentiert wird, jeder beliebigen PPU-Partition **600** von **Fig. 6** entsprechen, die zumindest zwei SMC-Engines **700** beinhaltet.

[0186] Wie in der Zeitleiste **2700** gezeigt ist, unterteilt PPU-Partition **600** innerhalb einer einzelnen VM, die als VM A bezeichnet wird, in Zeitscheiben. Die Zeitleiste **2700** veranschaulicht die Unterteilung in Zeitscheiben für die VM A, wobei der VM A zugeordnete Verarbeitungskontexte in der Form von KONTEXT 2710(Ax-y) gezeigt sind. Beginnend zum Zeitpunkt t0 wird die VM A auf zwei SMC-Engines **700** ausgeführt. Eine erste SMC-Engine **700**, die in der PPU-Partition **600** enthalten ist, führt den Verarbeitungskontext 2710(A0-1) aus und bleibt dann bis zum Zeitpunkt t1 im Leerlauf. Gleichzeitig führt eine zweite SMC-Engine **700**, die in der PPU-Partition **600** enthalten ist, den Verarbeitungskontext 2710(A1-1) aus und bleibt dann bis zum Zeitpunkt t1 im Leerlauf. Die Dauer zwischen dem Zeitpunkt t0 und dem Zeitpunkt t1 ist ausreichend lang, um genügend Zeit für die Verarbeitung der Kontexte 2710(A0-1) und 2710(A1-1) bis zum Abschluss der Ausführung und für den Eintritt der SMC-Engines in den Leerlaufzustand zu gewährleisten. In einigen Ausführungsformen können der Verarbeitungskontext 2710(A0-1) und der Verarbeitungskontext 2710(A1-1) die gleichen Aufgaben

auf zwei getrennten SMC-Engines **700** gleichzeitig ausführen, wodurch räumliche Redundanz bereitgestellt wird. In solchen Ausführungsformen können der Verarbeitungskontext 2710(A0-1) und der Verarbeitungskontext 2710(A1-1) Aufgaben auf redundanten SMC-Engines **700** ausführen, die die gleiche Konfiguration haben, und dann die Ergebnisse auf Genauigkeit und Determinismus vergleichen.

[0187] Zwischen dem Zeitpunkt t1 und dem Zeitpunkt t2 werden die Ablauflisten für die Verarbeitungskontexte 2710(A0-1) und 2710(A1-1) aus der PPU-Partition **600** entfernt. Die PPU-Partition **600** wird von zwei SMC-Engines **700** auf eine SMC-Engine **700**, die alle Ressourcen der beiden SMC-Engines **700** beinhaltet, umkonfiguriert. Die PPU-Partition **600** verwendet dann neue Ablauflisten für die Ausführung der Verarbeitungskontexte 2710(B2-1), 2710(B3-1) und 2710(B4-1).

[0188] Zwischen dem Zeitpunkt t2 und dem Zeitpunkt t3 wird die VM B auf einer SMC-Engine **700** ausgeführt. Die in der PPU-Partition **600** enthaltene SMC-Engine **700** führt sequentiell den Verarbeitungskontext 2710(B2-1), den Verarbeitungskontext 2710(B3-1) und den Verarbeitungskontext 2710(B4-1) aus. In einigen Ausführungsformen können die Verarbeitungskontexte 2710(B2-1), 2710(B3-1) und 2710(B4-1) leistungsintensive Aufgaben ausführen, die von der Ausführung auf einer einzigen SMC-Engine **700**, die über mehr Rechenressourcen verfügt als die SMC-Engines, die die Verarbeitungskontexte 2710(A0-1) und 2710(A1-1) ausführen, profitieren können. Die SMC-Engine **700** läuft dann bis zum Zeitpunkt t3 im Leerlauf. In einigen Ausführungsformen führt die SMC-Engine **700** während dieser Leerlaufphase Offline-Planungsaufgaben durch.

[0189] Zwischen dem Zeitpunkt t3 und dem Zeitpunkt t4 werden die Ablauflisten für die Verarbeitungskontexte 2710(B2-1), 2710(B3-1) und 2710(B4-1) aus der PPU-Partition **600** entfernt. Die PPU-Partition **600** wird von einer SMC-Engine **700** auf zwei SMC-Engines **700** umkonfiguriert. Die beiden SMC-Engines **700** beinhalten jeweils einen Teil der Ressourcen, die in der einen SMC-Engine **700** enthalten sind. Die PPU-Partition **600** verwendet dann neue Ablauflisten für die Ausführung der Verarbeitungskontexte 2710(A0-2) und 2710(A1-2).

[0190] Ab dem Zeitpunkt t4 wird erneut die VM A auf zwei SMC-Engines **700** ausgeführt. Eine erste SMC-Engine **700** führt den Verarbeitungskontext 2710(A0-2) aus und bleibt dann bis zum Zeitpunkt t5 im Leerlauf. Gleichzeitig führt eine zweite SMC-Engine **700** den Verarbeitungskontext 2710(A1-2) aus und bleibt dann bis zum Zeitpunkt t5 im Leerlauf. Die Dauer zwischen dem Zeitpunkt t4 und dem Zeitpunkt t5 ist ausreichend lang, um genügend Zeit für die Verarbeitung der Kontexte 2710(A0-2) und 2710(A1-2)

bis zum Abschluss der Ausführung und für Eintritt der SMC-Engines in einen Leerlaufzustand zu gewährleisten. In einigen Ausführungsformen können der Verarbeitungskontext 2710(A0-2) und der Verarbeitungskontext 2710(A1-2) die gleichen Aufgaben auf zwei getrennten SMC-Engines **700** gleichzeitig ausführen, wodurch räumliche Redundanz bereitgestellt wird. In solchen Ausführungsformen können der Verarbeitungskontext 2710(A0-2) und der Verarbeitungskontext 2710(A1-2) Aufgaben auf redundanten SMC-Engines **700** ausführen, die die gleiche Konfiguration haben, und dann die Ergebnisse auf Genauigkeit und Determinismus vergleichen.

[0191] Zwischen dem Zeitpunkt t5 und dem Zeitpunkt t6 werden die Ablauflisten für die Verarbeitungskontexte 2710(A0-2) und 2710(A1-2) aus der PPU-Partition **600** entfernt. Die PPU-Partition **600** wird von zwei SMC-Engines **700** auf eine SMC-Engine **700** umkonfiguriert, die alle Ressourcen der beiden SMC-Engines **700** beinhaltet. Die PPU-Partition **600** verwendet dann neue Ablauflisten für die Ausführung des Verarbeitungskontexts 2710(B3-2). Beginnend zum Zeitpunkt t6 wird erneut die VM B auf einer SMC-Engine **700** ausgeführt. Die SMC-Engine **700** führt den Verarbeitungskontext 2710(B3-2) aus.

[0192] In einigen Ausführungsformen kann die PPU-Partition **600** schnell zwischen der Ausführung auf einer SMC-Engine und der Ausführung auf zwei SMC-Engines rekonfigurieren, ein Vorgang, der hierin als „schnelle Rekonfiguration“ bezeichnet wird. Die schnelle Rekonfiguration erhöht die Auslastung der Ressourcen der PPU **200** und stellt gleichzeitig einen Mechanismus für mehrere Verarbeitungskontexte bereit, die auf einer einzigen PPU-Partition **600** in verschiedenen Modi ausgeführt werden können. Der Kernaltreiber **914** und/oder der Hardware-Mikrocode innerhalb der PPU **200** beinhalten verschiedene Optimierungen, die eine schnelle Rekonfiguration ermöglichen. Diese Optimierungen werden nun beschrieben.

[0193] Während der Rekonfiguration werden bestimmte Ressourcen in der PPU **200**, wie beispielsweise FECS **530** und GPC **242** Kontextwechsel, nicht zurückgesetzt, solange die Ressource keinen Fehler erzeugt. Infolgedessen kann das Laden von Mikrocode in diese Ressourcen während der Rekonfiguration in mehrere Phasen unterteilt werden. Insbesondere kann die Mikrocode-Ladesequenz für den FECS **530** und den GPC **242** in eine Lade- bzw. LOAD-Phase und eine Initialisierungs- bzw. INIT-Phase unterteilt werden. Die LOAD-Phase wird für alle verfügbaren FECS **530** und GPC **242** Kontextwechsel innerhalb der PPU-Partition **600** parallel ausgeführt, wodurch die zum Laden von Mikrocode in diese Ressourcen benötigte Zeit reduziert wird. Die INIT-Phase wird während der Rekonfiguration durchgeführt, wodurch die Initialisierung der FECS **530** und GPC

242 Kontextschalter parallel zur Rekonfiguration der PPU-Partition **600** durchgeführt wird. Während der Initialisierungsphase stellt die PPU **200** sicher, dass die LOAD-Phase für alle FECS **530** und GPC **242** Kontextwechsel abgeschlossen ist. Dadurch wird die Zeit zum Laden und Initialisieren der Ressourcen von PPU Partition **600** reduziert. Darüber hinaus speichert die PPU **200** für jede mögliche Konfiguration der PPU-Partitionen **600** einen Cache mit standardisierten Verarbeitungskontextbildern, die hierin als „goldene Verarbeitungskontextbilder“ bezeichnet werden. Die geeigneten goldenen Verarbeitungskontextbilder werden während der LOAD- und INIT-Phase abgerufen und geladen, wodurch die Zeit zum Laden und Initialisieren der Ressourcen der PPU-Partition **600** weiter reduziert wird.

[0194] Wie hierin beschrieben wurde, kann eine bestimmte VM die Anzahl von SMC-Engines **700** ändern, auf denen die VM im Laufe der Zeit ausgeführt wird. In einem bestimmten Beispiel beinhaltet die VM A verschiedene Aufgaben in Verbindung mit einem selbstfahrenden Fahrzeug. Bestimmte Aufgaben des selbstfahrenden Fahrzeugs sind kritischer als andere Aufgaben. Beispielsweise würden Aufgaben im Zusammenhang mit dem Selbstfahren, wie das Erkennen von Ampeln und die Vermeidung von Unfällen, als kritischer angesehen als Aufgaben im Zusammenhang mit dem Unterhaltungssystem des Fahrzeugs. Diese kritischeren Aufgaben können bestimmten Vorschriften oder Industrienormen unterliegen. Eine dieser Normen weist eine Klassifizierungsstufe zu, die als „Automotive Safety Integrity Level“ (ASIL) bekannt ist. ASIL umfasst vier Stufen, die in der Reihenfolge steigender Integritätsstufen als ASIL-A, ASIL-B, ASIL-C und ASIL-D bezeichnet werden. Aufgaben wie beispielsweise das Erkennen von Ampeln und die Vermeidung von Unfällen würden als ASIL-D klassifiziert werden. Weniger kritische Aufgaben können auf niedrigeren ASIL-Stufen klassifiziert werden. Aufgaben, die keine Sicherheitsrelevanz haben, wie z. B. Aufgaben im Zusammenhang mit dem Unterhaltungssystem des Fahrzeugs, können als QM klassifiziert werden, was bedeutet, dass nur Standardqualitätsmanagementpraktiken anwendbar sind.

[0195] In dieser Hinsicht können die Verarbeitungskontexte 2710(A0-1) und 2710(A1-1) zwei Instanzen derselben Aufgabe auf ASIL-D-Ebene beinhalten, die gleichzeitig auf zwei verschiedenen SMC-Engines **700** einer PPU-Partition **600** ausgeführt werden. Nachdem die Verarbeitungskontexte 2710(A0-1) und 2710(A1-1) vollständig ausgeführt wurden, werden die Ergebnisse der Verarbeitungskontexte 2710(A0-1) und 2710(A1-1) miteinander verglichen. Falls der Verarbeitungskontext 2710(A0-1) und der Verarbeitungskontext 2710(A1-1) die gleichen Ergebnisse erzeugen, werden die Ergebnisse validiert, und das Fahrzeug fährt entsprechend den Ergebnissen fort. Andererseits kann ein Fehler in einer oder mehreren

Komponenten, die entweder dem Verarbeitungskontext 2710(A0-1) oder dem Verarbeitungskontext 2710(A1-1) zugeordnet sind, dazu führen, dass der betroffene Verarbeitungskontext inkorrekte Ergebnisse generiert. Falls also der Verarbeitungskontext 2710(A0-1) und der Verarbeitungskontext 2710(A1-1) unterschiedliche Ergebnisse erzeugen, werden die Ergebnisse für ungültig erklärt, und das Fahrzeug führt eine geeignete Ausweichaktion aus, wie beispielsweise langsames Fahren in Richtung der nächstgelegenen Stelle außerhalb des Verkehrsflusses.

[0196] Nach der vollständigen Ausführung der Verarbeitungskontexte 2710(A0-1) und 2710(A1-1) wird die PPU-Partition **600** so umkonfiguriert, dass sie nur noch eine SMC-Engine **700** enthält. Die SMC-Engine **700** führt die Verarbeitungskontexte 2710(B2-1), 2710(B3-1) und 2710(B4-1) auf QM-Ebene sequentiell aus. Diese Verarbeitungskontexte umfassen weniger kritische Aufgaben, wie beispielsweise Aufgaben im Zusammenhang mit dem Unterhaltungssystem des Fahrzeugs. Nach der vollständigen Ausführung der Verarbeitungskontexte 2710(B2-1), 2710(B3-1) und 2710(B4-1) wird die PPU-Partition **600** so umkonfiguriert, dass sie zwei SMC-Engines **700** enthält. Die SMC-Engines **700** führen gleichzeitig die Verarbeitungskontexte 2710(A0-2) und 2710(A1-2) aus, welche zwei Instanzen derselben Aufgabe auf ASIL-D-Ebene sind. Nachdem die Verarbeitungskontexte 2710(A0-2) und 2710(A1-2) vollständig ausgeführt wurden, rekonfiguriert sich die PPU-Partition **600** erneut so, dass sie nur eine SMC-Engine **700** enthält, und führt den Verarbeitungskontext 2710(B3-2) auf QM-Ebene aus.

[0197] Auf diese Weise rekonfiguriert sich die PPU-Partition **600** dynamisch zwischen mehreren SMC-Engines **700**, die Aufgaben auf ASIL-D-Ebene ausführen, und einer einzigen SMC-Engine **700**, die Aufgaben auf QM-Ebene ausführt. Die Dauer zwischen aufeinanderfolgenden ASIL-D-Verarbeitungskontexten, wie beispielsweise die Dauer zwischen dem Zeitpunkt t0 und dem Zeitpunkt t4, wird als ein „Rahmen“ bzw. „Frame“ bezeichnet, wobei der Teil des Rahmens zwischen dem Zeitpunkt t0 und dem Zeitpunkt t1 für die Ausführung von ASIL-D-Aufgaben allokiert ist.

[0198] Fig. 28 veranschaulicht, wie VMs von einer PPU **200(1)** zu einer anderen PPU **200(2)** migrieren können, gemäß verschiedenen Ausführungsformen. Wie in einem Diagramm **2800** gezeigt ist, führt PPU **200(1)** vier VMs **2810A**, **2810B**, **2810C** und **2810D** aus. Jede dieser VMs **2810A**, **2810B**, **2810C** und **2810D** wird auf einer anderen SMC-Engine **700** ausgeführt, die in der PPU **200(1)** enthalten ist. In ähnlicher Weise führt die PPU **200(2)** vier VMs **2810E**, **2810F**, **2810G** und **2810H** aus. Jede dieser VMs **2810E**, **2810F**, **2810G** und **2810H** wird auf einer an-

deren SMC-Engine **700** ausgeführt, die in der PPU **200(1)** bzw. PPU **200(2)** enthalten ist.

[0199] Im Laufe der Zeit können VMs aus verschiedenen Gründen von einer PPU **200** auf eine andere PPU **200** migriert werden, einschließlich, und ohne darauf beschränkt zu sein, zur Vorbereitung einer Systemwartung, zur Konsolidierung von VMs auf weniger PPUs **200** für bessere Auslastung oder Energieeinsparungen und zur Effizienzsteigerung durch Migrieren auf verschiedene Rechenzentren. In einem ersten Beispiel könnten VMs gezwungen sein, auf andere PPUs **200** zu migrieren, wenn das System, auf dem die VMs aktuell ausgeführt werden, kurz vor dem Herunterfahren für die Systemwartung steht. In einem zweiten Beispiel könnten die Verarbeitungskontexte einer oder mehrerer VMs für eine unbestimmte Zeit inaktiv sein. Falls alle Verarbeitungskontexte in einer oder mehreren VMs im Leerlauf sind, könnten die VMs von einer PPU **200** zu einer anderen PPU **200** migrieren, um die Auslastung der PPU **200** zu verbessern oder den Stromverbrauch zu senken. In einem dritten Beispiel könnten VMs, die einem bestimmten Benutzer oder einer bestimmten Gruppe von Benutzern zugeordnet sind, von einem geografisch entfernten Rechenzentrum in ein näher gelegenes Rechenzentrum migriert werden, um die Kommunikationslatenz zu verbessern. Ganz allgemein können VMs auf verschiedene PPUs **200** migriert werden.

[0200] Allgemeiner gesagt können VMs jederzeit von einer PPU **200** auf eine andere PPU **200** migrieren, wenn ein Kontext über eine Kontextspeicherung von der Ausführung auf Hardware entfernt wurde. Das relevante Betriebssystem, wie beispielsweise das Gastbetriebssystem **916** von **Fig. 9**, kann jederzeit einen Kontext vorwegnehmen und eine Kontextspeicherung erzwingen, nicht nur dann, wenn sich die entsprechende VM im Leerlauf befindet. Ein Kontext kann vorweggenommen werden, wenn alle Arbeiten für die entsprechende VM abgeschlossen sind. Darüber hinaus kann ein Kontext vorweggenommen werden, indem der Kontext gezwungen wird, keine weitere Arbeit mehr abzuliefern und die laufende Arbeit zu erledigen, selbst wenn die VM zusätzliche Arbeit zu leisten hat. In jedem Fall kann die VM von einer PPU **200** zu einer anderen PPU **200** migriert werden, sobald die laufende Arbeit für den Kontext erledigt und der Kontext gespeichert wurde. Während der VM-Migration kann es bei der VM zu einer Aussetzung der Ausführung in der Größenordnung von einigen Millisekunden kommen.

[0201] In einigen Ausführungsformen kann eine VM nur zu einer PPU-Partition **600** in einer anderen PPU **200**, die dieselbe Konfiguration wie die PPU-Partition **600** hat, die die VM gerade ausführt, migrieren. Beispielsweise kann die VM darauf beschränkt sein, nur zu einer PPU-Partition **600** in einer ande-

ren PPU **200** zu migrieren, die dieselbe Anzahl von GPCs **242** hat wie die PPU-Partition **600**, die derzeit die VM ausführt. Wie in einem Diagramm **2802** gezeigt ist, befinden sich vier der VMs in einem Leerlaufzustand. Die PPU **200(1)** führt zwei VMs **2810A** und **2810C** aus. Die beiden anderen VMs **2810B** und **2810D**, die zuvor auf der PPU **200(1)** ausgeführt wurden, befinden sich im Leerlauf. In ähnlicher Weise führt die PPU **200(2)** zwei VMs **2810E** und **2810H** aus. Die beiden anderen VMs **2810F** und **2810G**, die zuvor auf der PPU **200(1)** ausgeführt wurden, befinden sich im Leerlauf. Infolgedessen sind PPU **200(1)** und die PPU **200(2)** nicht voll ausgelastet. In solchen Fällen können die aktuell ausgeführten VMs migrieren, um die verfügbaren PPU-Ressourcen besser auszunutzen. In einem Beispiel könnten die auf der PPU **200(1)** ausgeführten VMs die Hälfte der auf der PPU **200(1)** verfügbaren Hardwareressourcen verbrauchen. Ebenso könnten die auf der PPU **200(2)** ausgeführten VMs die Hälfte der auf der PPU **200(2)** verfügbaren Hardwareressourcen verbrauchen. Infolgedessen würden die PPU **200(1)** und die PPU **200(2)** jeweils mit etwa 50 % Kapazität arbeiten. Falls alle VMs, die auf die PPU **200(2)** ausgeführt werden, auf die PPU **200(1)** migriert werden, dann würde die PPU **200(1)** mit etwa 100 % Kapazität arbeiten. Die PPU **200(2)** würde mit 0 % Kapazität arbeiten. Infolgedessen könnte die Versorgungsspannung für die PPU **200(2)** reduziert werden, um den Stromverbrauch zu senken. Wie in einem Diagramm **2804** gezeigt ist, haben die VMs **2810E** und **2810H** von der PPU **200(2)** auf die PPU **200(1)** migriert. Infolgedessen führt die PPU **200(1)** die vier VMs **2810A**, **2810E**, **2810C** und **2810H** aus. Daher wird die PPU **200(1)** besser ausgelastet. Nach der VM-Migration führt die PPU **200(2)** keine VMs mehr aus. Infolgedessen kann die PPU **200(2)** heruntergefahren werden, um den Stromverbrauch zu reduzieren. Falls anschließend weitere VMs mit der Ausführung beginnen, kann die PPU **200(2)** hochgefahren werden, um die zusätzlichen VMs auszuführen.

[0202] **Fig. 29** ist ein Satz von Zeitleisten **2900**, die eine feine VM-Migration in Verbindung mit der PPU **200** aus **Fig. 2** veranschaulichen, gemäß verschiedenen Ausführungsformen. Der Satz von Zeitleisten **2900** funktioniert im Wesentlichen genauso wie die Sätze von Zeitleisten **2500** und **2600** von **Fig. 25** bzw. **Fig. 26** und die Zeitleiste **2700** von **Fig. 27**, ausgenommen wie nachstehend näher beschrieben. Wie gezeigt ist, beinhaltet der Satz der Zeitleisten **2900**, ohne darauf beschränkt zu sein, vier PPU-Partitions-Zeitleisten **2902(0)**, **2902(1)**, **2902(2)** und **2902(3)**. In einigen Ausführungsformen können die PPU-Partitions-Zeitleisten **2902(0)**, **2902(1)**, **2902(2)** und **2902(3)** den vier PPU-Partitionen **600** von **Fig. 6** entsprechen. Jede PPU-Partition **600** beinhaltet eine SMC-Engine **700**. Daher kann jede PPU-Partition **600** jeweils eine VM auf einmal ausführen. Wie in den PPU-Partitions-Zeitleisten **2902(0)**, **2902(1)**, **2902(2)**

und **2902(3)** gezeigt ist, führt jede PPU-Partition **600** Zeitscheiben zwischen fünf VMs aus, die als VM A bis VM F bezeichnet werden. Infolge dessen migriert jede der fünf VMs zwischen den vier PPU-Partitionen **600**.

[0203] Während der in **Fig. 29** gezeigten Zeitspanne führt die VM A den Verarbeitungskontext 2910 (A0) auf einer ersten PPU-Partition aus, wie auf der PPU-Partitions-Zeitleiste **2902(3)** gezeigt ist. Anschließend migriert die VM A zu einer zweiten PPU-Partition und führt den Verarbeitungskontext 2910 (A1) auf der PPU-Partitions-Zeitleiste **2902(2)** aus. Anschließend migriert die VM A wiederum zu einer dritten PPU-Partition und einer vierten PPU-Partition und führt Verarbeitungskontexte 2910(A2) und 2910(A3) auf den PPU-Partitions-Zeitleisten **2902(1)** bzw. **2902(0)** aus. Die VM A migriert dann zurück zu der ersten PPU-Partition und führt den Verarbeitungskontext 2910(A4) auf der PPU-Partitions-Zeitleiste **2902(3)** aus. Schließlich migriert die VM A erneut zu der zweiten PPU-Partition und führt den Verarbeitungskontext 2910(A5) auf der PPU-Partitions-Zeitleiste **2902(2)** aus.

[0204] In ähnlicher Weise wird die VM B, die die Verarbeitungskontexte 2910(B0) bis 2910(B4) ausführt, auf der ersten PPU-Partition **600** ausgeführt und migriert dann zwischen den anderen drei PPU-Partitionen, wie in den PPU-Partitions-Zeitleisten **2902(0)**, **2902(1)**, **2902(2)** und **2902(3)** gezeigt ist. Die übrigen drei VMs migrieren gleichermaßen zwischen den vier PPU-Partitionen, wobei die VM C die Verarbeitungskontexte 2910(C0) bis 2910(C5), die VM D die Verarbeitungskontexte 2910(D0) bis 2910(D5) und die VM E die Verarbeitungskontexte 2910(E0) bis 2910(E5) ausführt.

[0205] Auf diese Weise migrieren fünf VMs zwischen vier PPU-Partitionen **600**, wobei jede VM im Wesentlichen auf die gleiche Menge an PPU-Ressourcen zugreift. Insgesamt sind die fünf VMs jeweils in der Lage, etwa 80% der Zeit auszuführen, wobei vier PPU-Partitionen **600** geteilt durch fünf VMs **4/5** oder 80% entsprechen. Infolgedessen führt die feine VM-Migration einen Lastausgleich zwischen einem Satz von VMs durch, unabhängig von der Anzahl von VMs relativ zu der Anzahl von PPU-Partitionen **600**.

[0206] Die **Fig. 30A-30B** stellen ein Ablaufdiagramm von Verfahrensschritten für die Unterteilung von VMs in der PPU **200** von **Fig. 2** in Zeitscheiben dar, gemäß verschiedenen Ausführungsformen. Obwohl die Verfahrensschritte in Verbindung mit den Systemen der **Fig. 1-15** beschrieben werden, versteht sich für den Fachmann, dass jedes System, das dazu konfiguriert ist, die Verfahrensschritte in beliebiger Reihenfolge auszuführen, in den Anwendungsbereich der Erfindung fällt.

[0207] Wie gezeigt ist, beginnt ein Verfahren **3000** bei Schritt **3002**, wobei die PPU **200** ermittelt, dass zumindest eine VM dabei ist, von einem ersten Satz eines oder mehrerer Verarbeitungskontexte zu einem zweiten Satz eines oder mehrerer Verarbeitungskontexte zu wechseln. Eine VM kann aus jedem technisch machbaren Grund im Begriff sein, den/die Verarbeitungskontext(e) zu wechseln, einschließlich und ohne Beschränkung darauf, wenn die VM die Ausführung aller Aufgaben abschließt, die VM in einen Ruhezustand übergeht, die VM für eine maximal zugewiesene Zeitspanne ausgeführt wurde oder die VM einen Fehler erzeugt hat.

[0208] In Schritt **3004** bestimmt die PPU **200**, ob die PPU **200** eine Intra-PPU-Partitionsänderung durchführen muss, um den/die neuen Verarbeitungskontext(e) zu berücksichtigen. Eine Intra-PPU-Partitionsänderung erfolgt, wenn eine PPU-Partition **600** während des Kontextwechsels die gleiche Anzahl von PPU-Slices **610** beibehält, aber die Anzahl aktiver SMC-Engines **700** innerhalb der PPU-Partition **600** ändert. Falls die PPU **200** keine Intra-PPU-Partitionsänderung durchzuführen braucht, um den neuen Verarbeitungskontext/die neuen Verarbeitungskontexte zu berücksichtigen, schreitet das Verfahren **3000** zu Schritt **3008** fort. Falls jedoch die PPU **200** eine Intra-PPU-Partitionsänderung durchführen muss, um den neuen Verarbeitungskontext/die neuen Verarbeitungskontexte zu berücksichtigen, schreitet das Verfahren **3000** zu Schritt **3006** fort, in dem die PPU **200** die PPU-Partition **600** so rekonfiguriert, dass die gleiche Anzahl von PPU-Slices **610** beibehalten wird, während die Anzahl aktiver SMC-Engines **700** geändert wird.

[0209] In Schritt **3008** ermittelt die PPU **200**, ob die PPU **200** eine Inter-PPU-Partitionsänderung durchführen muss, um den/die neuen Verarbeitungskontext(e) zu berücksichtigen. Eine Inter-PPU-Partitionsänderung erfolgt, wenn eine PPU-Partition **600** die Anzahl der PPU-Slices **610** während des Kontextwechsels durch Zusammenführen oder Aufteilen einer oder mehrerer PPU-Partitionen **600** ändert. Infolgedessen ändert auch die PPU **200** die Anzahl der PPU-Partitionen **600** innerhalb von 700. Abhängig von den neuen Verarbeitungskontexten kann die PPU **200** die Anzahl aktiver SMC-Engines **700** innerhalb jeder PPU-Partition **600** ändern oder nicht. Falls die PPU **200** keine Inter-PPU-Partitionsänderung durchzuführen braucht, um den/die neuen Verarbeitungskontext(e) zu berücksichtigen, dann schreitet das Verfahren **3000** zu Schritt **3012** fort. Falls jedoch die PPU **200** eine Inter-PPU-Partitionsänderung durchführen muss, um den/die neuen Verarbeitungskontext(e) zu berücksichtigen, schreitet das Verfahren **3000** zu Schritt **3010** fort, in dem die PPU **200** die PPU-Partition **600** rekonfiguriert, um die Anzahl von in der PPU-Partition **600** enthaltenen PPU-Slices **610** zu ändern. Um die Anzahl von PPU-Slices **610** in ei-

ner PPU-Partition **600** zu ändern, führt die PPU **200** zwei oder mehr PPU-Partitionen **600** zu einer einzigen PPU-Partition **600** zusammen. Zusätzlich oder alternativ teilt die PPU **200** eine PPU-Partition **600** in zwei oder mehr PPU-Partitionen **600** auf.

[0210] In Schritt **3012** ermittelt die PPU **200**, ob alle VMs, die auf einer bestimmten PPU **200** ausgeführt werden, im Leerlauf sind. Falls eine oder mehrere VMs auf der gegebenen PPU **200** nicht in Leerlauf (aktiv) sind, schreitet das Verfahren zu Schritt **3018** fort. Falls dagegen alle VMs, die auf einer bestimmten PPU **200** ausgeführt werden, im Leerlauf sind, schreitet das Verfahren **3000** zu Schritt **3014** fort, in dem die PPU **200** bestimmt, ob eine oder mehrere andere PPUs **200** die Ressourcen zur Ausführung der sich im Leerlauf befindenden VMs zur Verfügung haben. Falls die Ressourcen auf einer oder mehreren anderen PPUs **200** nicht verfügbar sind, schreitet das Verfahren zu Schritt **3018** fort. Falls dagegen die Ressourcen auf einer oder mehreren anderen PPUs **200** nicht verfügbar sind, schreitet das Verfahren zu Schritt **3016** fort, in dem die PPU **200** die sich im Leerlauf befindenden VMs auf eine oder mehrere andere PPUs **200** migriert.

[0211] In Schritt **3018** beginnt nach der Durchführung von Inter-PPU-Partitionsänderungen, Inter-PPU-Partitionsänderungen und/oder VM-Migrationen die PPU **200** mit der Ausführung der neuen Verarbeitungskontexte. Dann wird das Verfahren **3000** beendet. In verschiedenen Ausführungsformen ermittelt PPU **200** die Notwendigkeit der Durchführung von Inter-PPU-Änderungen unabhängig von der Notwendigkeit der Durchführung von Inter-PPU-Änderungen. In ähnlicher Weise ermittelt in verschiedenen Ausführungsformen die PPU **200** die Notwendigkeit, Inter-PPU-Änderungen durchzuführen, unabhängig von der Ermittlung der Notwendigkeit, Inter-PPU-Änderungen durchzuführen.

Adresszuordnung für privilegierte Register

[0212] Wie hierin weiter beschrieben wird, ermöglicht der PRI-Hub **512** von **Fig. 5** und ein interner PRI-Bus (nicht gezeigt) es einer CPU **110** und/oder einer beliebigen Einheit in der PPU **200**, privilegierte Register, auch „PRI-Register“ genannt, die über die gesamte PPU **200** verteilt sind, zu lesen und zu schreiben. Dabei ist der PRI-Hub **212** dazu konfiguriert, PRI-Bus-Adressen zwischen einem generischen Adressraum, der alle PRI-Bus-Register abdeckt, und einem für jede Sys-Pipe **230** separat definierten Adressraum abzubilden. Bei der Kommunikation über eine PCIe-Verbindung, allgemein von der CPU aus, erfolgt der Zugriff auf die PRI-Register über einen PCIe-Adressraum, der hierin als der „Basisadressregister 0“-Raum oder einfacher als der „BAR0“-Adressraum bezeichnet wird, wie er für an PCIe-Busse angebundene Geräte typisch ist. Vorwiegend ist der adressier-

bare Speicherbereich des BARO-Adressraums für die PPU **200** auf 16 Megabyte (MB) begrenzt, da eine große Anzahl von Geräten alle in den BARO-Adressraum passen müssen. Der Adressbereich von 16 MB ist für den Zugriff auf die privilegierten Register für eine einzelne SMC-Engine **700** angemessen. Um jedoch mehrere SMC-Engines **700** zu unterstützen, kann der Adressbereich **16** MB überschreiten. Daher stellt der PRI-Hub **512** zwei Adressierungsmodi bereit, um die Ausführung mit mehreren SMC-Engines **300** zu unterstützen. Der erste Adressierungsmodus, hierin als „Legacy-Modus“ bezeichnet, gilt für Operationen, die eine einzelne SMC-Engine **700** involvieren. Der zweite Adressierungsmodus, hierin als „SMC-Engine-Adressierungsmodus“ bezeichnet, gilt für Operationen, die mehrere SMC-Engines **700** involvieren. Die Adressierungsmodi werden nun beschrieben.

[0213] **Fig. 31** ist eine Speicherabbildung, die veranschaulicht, wie der BARO-Adressraum **3110** auf den privilegierten Registeradressraum **3120** innerhalb der PPU **200** von **Fig. 2** abgebildet wird, gemäß verschiedenen Ausführungsformen. Der BARO-Adressraum **3110** beinhaltet, ohne darauf beschränkt zu sein, einen ersten Adressraum **3112**, einen Grafikregister (GFX REG)-Adressraum **3114** und einen zweiten Adressraum **3116**. Ein privilegierte Register-Adressraum **3120** beinhaltet, ohne darauf beschränkt zu sein, einen ersten Adressraum **3122**, einen Legacy-Grafikregister-Adressraum **3124**, einen zweiten Adressraum **3126** und SMC-Grafikregister-Adressräume **3128(0) - 3128(7)**. Der BARO-Adressraum **3110** unterstützt zwei Adressierungsmodi, den Legacy-Adressierungsmodus und den SMC-Adressierungsmodus. Allgemein sind die Ziele der beiden Modi **(1)** ein Legacy-Modus, bei dem die gesamte PPU **200** als eine einzige Maschine mit einem Satz von PRI-Registern behandelt wird; und **(2)** ein SMC-Modus, bei dem jede PPU-Partition **600** so adressiert wird, als ob jede PPU-Partition **600** eine separate Maschine wäre und als ob jede PPU-Partition **600** eine eigenständige gesamte PPU **200** wäre. Der SMC-Modus ermöglicht es Treibersoftware **122**, identisch zu sein, wenn sie mit der gesamten PPU im Legacy-Modus und mit nur einer PPU-Partition **600** arbeitet. Das heißt, der Treiber kann einmal geschrieben und in beiden Szenarien, im Legacy-Modus und im SMC-Modus, verwendet werden.

[0214] Im Legacy-Adressierungsmodus führt die PPU **200** Aufgaben als einen einzigen Cluster von Hardwareressourcen aus, und nicht als separate PPU-Partitionen **600** mit separaten SMC-Engines **700**. Im Legacy-Modus wird beim Lesen oder Schreiben eines Speichers auf eine Speicheradresse in Richtung des ersten Adressraums **3112** oder des zweiten Adressraums **3116** des BARO-Adressraums **3110** auf eine entsprechende Speicheradresse innerhalb des ersten Adressraums **3122** bzw. des zweiten

Adressraums **3126** des privilegierte Register-Adressraums **3120** zugegriffen. In ähnlicher Weise wird beim Lesen oder Schreiben eines Speichers auf eine Speicheradresse in Richtung des Graphikregister-Adressraums **3114** des BARO-Adressraums **3110** auf eine entsprechende Speicheradresse innerhalb des Legacy-Graphikregister-Adressraums **3124** des privilegierte Register-Adressraums **3120** zugegriffen. Der Legacy-Graphikregister-Adressraum **3124** beinhaltet einen Adressbereich für verschiedene Komponenten innerhalb der PPU **200**, einschließlich, aber nicht beschränkt auf, das Rechen-FE **540**, das Grafik-FE **542**, den SKED **550**, die CWD **560** und die PDA/PDB **562**. Darüber hinaus beinhaltet der Legacy-Graphikregister-Adressraum **3124** einen Adressbereich für jeden von GPCs **242**. Die GPCs **242** sind abzüglich der GPCs **242**, die aufgrund von Floorsweeping entfernt wurden, über dedizierte Adressbereiche innerhalb des Legacy-Graphikregister-Adressraums **3124** individuell adressierbar. Zusätzlich oder alternativ dazu beinhaltet der Legacy-Graphikregister-Adressraum **3124** Adressbereiche für die gleichzeitige Übertragung von Daten an alle GPCs **242**. Diese GPC-Broadcast-Adressräume können nützlich sein, wenn alle GPCs **242** identisch konfiguriert sind.

[0215] Im SMC-Adressierungsmodus führt die PPU **200** Aufgaben als separate PPU-Partitionen **600** mit separaten SMC-Engines **700** aus. Wie im Legacy-Modus wird beim Lesen oder Schreiben eines Speichers auf eine Speicheradresse in Richtung des ersten Adressraums **3112** oder des zweiten Adressraums **3116** des BARO-Adressraums **3110** auf eine entsprechende Speicheradresse innerhalb des ersten Adressraums **3122** bzw. des zweiten Adressraums **3126** des privilegierte Register-Adressraums **3120** zugegriffen. Im SMC-Modus sind die SMC-Graphikregister-Adressräume **3128(0) - 3128(7)** für den Zugriff auf die verschiedenen Komponenten innerhalb jeder der SMC-Engines **700(0) - 700(7)** bereitgestellt. Diese Komponenten beinhalten, ohne darauf beschränkt zu sein, die Rechen-FE **540(0) - 540(7)**, die Grafik-FE **542(0) - 542(7)**, die SKED **550(0) - 550(7)**, die CWD **560(0) - 560(7)** und die PDA/PDB **562(0) - 562(7)**. Die GPCs **242** für eine bestimmte entsprechende SMC-Engine **700** sind abzüglich der GPCs **242**, die aufgrund von Floorsweeping entfernt wurden, über dedizierte Adressbereiche innerhalb des Legacy-Graphikregister-Adressraums **3124** individuell adressierbar. Zusätzlich oder alternativ beinhaltet der Legacy-Graphikregister-Adressraum **3124** einen Adressbereich für die gleichzeitige Übertragung von Daten an alle GPCs **242** für eine bestimmte entsprechende SMC-Engine **700**. Der BARO-Adressraum **3110** stellt zwei Mechanismen für den Zugriff auf die SMC-Graphikregister-Adressräume **3128(0) - 3128(7)** bereit.

[0216] In einem ersten Mechanismus wird der Graphikregister-Adressraum **3114** des BARO-Adress-

raums **3110** auf einen der SMC-Graphikregister-Adressräume **3128(0) - 3128(7)** in dem privilegierte Register-Adressraum **3120** abgebildet. Eine bestimmte Adresse in dem BARO-Adressraum **3110** greift auf ein SMC-Fensterregister zu. Das SMC-Fensterregister enthält zwei Felder. Diese zwei Felder beinhalten ein SMC-Freigabefeld und ein SMC-Indexfeld. Das SMC-Freigabefeld enthält einen binären Logikwert, der entweder FALSE oder TRUE ist. Wenn das SMC-Freigabefeld FALSE ist, dann greift der BARO-Adressraum **3110** auf den privilegierte Register-Adressraum **3120** in dem Legacy-Adressierungsmodus zu, wie hierin beschrieben wurde. Wenn das SMC-Freigabefeld TRUE ist, dann greift der BARO-Adressraum **3110** auf den privilegierte Register-Adressraum **3120** in dem SMC-Adressierungsmodus zu, basierend auf den Werten des SMC-Indexfelds. Der Wert des SMC-Indexfelds gibt an, welche SMC-Engine **700** derzeit auf den BARO-Adressraum **3110** abgebildet wird. Falls beispielsweise der Wert des SMC-Indexfelds **0** ist, dann würde der SMC-Graphikregister-Adressraum **3128(0)** des privilegierte Register-Adressraums **3120** auf den Graphikregister-Adressraum **3114** des BARO-Adressraums **3110** abgebildet werden. Gleichermäßen würde dann, wenn der Wert des SMC-Indexfelds **1** ist, der SMC-Graphikregister-Adressraum **3128(1)** des privilegierte Register-Adressraums **3120** auf den Graphikregister-Adressraum **3114** des BARO-Adressraums **3110** abgebildet, und so weiter. Wenn ein Lese- oder Schreibzugriff auf eine Speicheradresse in Richtung des Graphikregister-Adressraums **3114** des BARO-Adressraums **3110** erfolgt, wird auf eine entsprechende Speicheradresse innerhalb des durch das SMC-Indexfeld spezifizierten SMC-Graphikregister-Adressraums **3128** zugegriffen. Über diesen ersten Mechanismus ist der durch das SMC-Indexfeld spezifizierte SMC-Graphikregister-Adressraum **3128** für die SMC-Engine **700** zugänglich, während der Zugriff auf die übrigen SMC-Graphikregister-Adressräume **3128** verboten ist.

[0217] In einem zweiten Mechanismus sind einzelne Adressen innerhalb eines der SMC-Graphikregister-Adressräume **3128(0) - 3128(7)** durch bestimmte privilegierte Komponenten, wie z.B. den Hypervisor **124**, zugänglich. Dieser zweite Mechanismus greift auf die SMC-Graphikregisteradressräume **3128(0) - 3128(7)** über zwei bestimmte Adressen innerhalb des BARO-Adressraums **3110** zu. Eine der zwei Adressen greift auf ein SMC-Adressregister zu. Die andere der zwei Adressen greift auf ein SMC-Datenregister zu. Auf eine bestimmte Speicheradresse irgendwo innerhalb der SMC-Graphikregister-Adressräume **3128(0) - 3128(7)** wird in zwei Schritten zugegriffen. In einem ersten Schritt wird das SMC-Adressregister mit einer Adresse beschrieben, die einer Adresse in den SMC-Graphikregister-Adressräumen **3128(0) - 3128(7)** entspricht. In einem zweiten Schritt wird das SMC-Datenregister mit einem Datenwert gelesen oder beschrieben. Ein Lesen oder

Schreiben des SMC-Datenregisters in dem BARO-Adressraum **3110** bewirkt ein entsprechendes Lesen oder Schreiben in den privilegierte Register-Adressraum **3120** an der im SMC-Adressregister spezifizierten Speicheradresse. Das SMC-Adressregister wird dann dereferenziert, wodurch das SMC-Adressregister und das SMC-Datenregister für eine nachfolgende Transaktion freigegeben werden. Das Lesen oder Schreiben des SMC-Adressregisters führt nicht zu einem Lesen oder Schreiben in den privilegierte Register-Adressraum **3120**.

[0218] Fig. 32 ist ein Ablaufdiagramm von Verfahrensschritten zur Adressierung des privilegierte Register-Adressraums in der PPU **200** von Fig. 2, gemäß verschiedenen Ausführungsformen. Obwohl die Verfahrensschritte in Verbindung mit den Systemen der Fig. 1-17 beschrieben werden, versteht sich für den Fachmann, dass jedes System, das dazu konfiguriert ist, die Verfahrensschritte in beliebiger Reihenfolge auszuführen, in den Anwendungsbereich der Erfindung fällt.

[0219] Wie gezeigt ist, beginnt ein Verfahren **3200** bei Schritt **3202**, in dem die PPU **200** einen Speicherzugriff erfasst, der auf den privilegierte Register-Adressraum **3120** gerichtet ist. Genauer gesagt erfasst die PPU **200** einen Speicherzugriff, der auf den BARO-Adressraum **3110** gerichtet ist. In Schritt **3204** ermittelt die PPU **200**, ob der Speicherzugriff auf den Grafikregister-Adressraum **3114** gerichtet ist. Falls der Speicherzugriff nicht auf den Graphikregister-Adressraum **3114** gerichtet ist, schreitet das Verfahren **3200** zu Schritt **3214** fort, wobei die PPU **200** eine Speichertransaktion in die durch den Speicherzugriff spezifizierte Adresse erzeugt. Das Verfahren **3200** wird dann beendet.

[0220] Zu Schritt **3204** zurückkehrend, schreitet dann, wenn der Speicherzugriff auf den Grafikregister-Adressraum **3114** gerichtet ist, das Verfahren **3200** zu Schritt **3206** fort, in dem die PPU **200** ermittelt, ob der Speicherzugriff im Legacy-Modus erfolgt. Falls der Speicherzugriff im Legacy-Modus erfolgt, schreitet das Verfahren **3200** zu Schritt **3214** fort, in dem die PPU **200** eine Speichertransaktion in die durch den Speicherzugriff spezifizierte Adresse erzeugt. Das Verfahren **3200** endet dann. Falls demgegenüber der Speicherzugriff nicht im Legacy-Modus erfolgt, schreitet das Verfahren **3200** zu Schritt **3208** fort, in dem die PPU **200** ermittelt, ob der Speicherzugriff im Fenstermodus erfolgt.

[0221] Falls der Speicherzugriff im Fenstermodus erfolgt, schreitet das Verfahren zu Schritt **3212** fort, in dem die PPU **200** eine Speichertransaktion auf der Grundlage des Werts des in dem SMC-Fensterregister spezifizierten SMC-Indexfelds erzeugt. Der Wert des SMC-Indexfelds spezifiziert, welche SMC-Engine **700** aktuell auf den BARO-Adressraum **3110** ab-

gebildet wird. Falls z.B. der Wert des SMC-Indexfelds **0** ist, dann würde der SMC-Grafikregister-Adressraum **3128(0)** des privilegierte Register-Adressraums **3120** auf den Grafikregister-Adressraum **3114** des BARO-Adressraums **3110** abgebildet werden. Gleichmaßen würde dann, wenn der Wert des SMC-Indexfelds **1** ist, der SMC-Grafikregister-Adressraum **3128(1)** des privilegierte Register-Adressraums **3120** auf den Grafikregister-Adressraum **3114** des BARO-Adressraums **3110** abgebildet, und so weiter. Wenn ein Speicher-Lese- oder Schreibzugriff auf eine Speicheradresse in Richtung des Graphikregister-Adressraums **3114** des BARO-Adressraums **3110** geleitet wird, wird auf eine entsprechende Speicheradresse innerhalb des SMC-Graphikregister-Adressraums **3128** zugegriffen, die durch das SMC-Indexfeld spezifiziert wird. Über diesen ersten Mechanismus ist der SMC-Grafikregister-Adressraum **3128** für die SMC-Engine **700** spezifiziert durch das SMC-Indexfeld zugänglich, während der Zugriff auf die übrigen SMC-Grafikregister-Adressräume **3128** verboten ist. Das Verfahren **3200** endet dann.

[0222] Zu Schritt **3208** zurückkehrend schreitet dann, wenn der Speicherzugriff nicht im Fenstermodus erfolgt, das Verfahren zu Schritt **3210** fort, in dem die PPU **200** eine Speichertransaktion auf der Grundlage der Werte des SMC-Adressregisters und des SMC-Datenregisters erzeugt. Genauer gesagt greift die PPU **200** auf die SMC-Grafikregister-Adressräume **3128(0) - 3128(7)** über zwei bestimmte Adressen innerhalb des BARO-Adressraums **3110** zu. Eine der zwei Adressen greift auf ein SMC-Adressregister zu. Die andere der zwei Adressen greift auf ein SMC-Datenregister zu. Auf eine bestimmte Speicheradresse irgendwo innerhalb der SMC-Grafikregister-Adressräume **3128(0) - 3128(7)** wird in zwei Schritten zugegriffen. In einem ersten Schritt wird das SMC-Adressregister mit einer Adresse beschrieben, die einer Adresse in den SMC-Grafikregister-Adressräumen **3128(0) - 3128(7)** entspricht. In einem zweiten Schritt wird das SMC-Datenregister gelesen oder mit einem Datenwert beschrieben. Ein Lesen oder Schreiben des SMC-Datenregisters in dem BARO-Adressraum **3110** bewirkt ein entsprechendes Lesen oder Schreiben in den privilegierte Register-Adressraum **3120** an der in dem SMC-Adressregister angegebenen Speicheradresse. Das SMC-Adressregister wird dann dereferenziert, wodurch das SMC-Adressregister und das SMC-Datenregister für eine nachfolgende Transaktion freigegeben werden. Das Lesen oder Schreiben des SMC-Adressregisters führt nicht zu einem Lesen oder Schreiben in den privilegierte Register-Adressraum **3120**. Das Verfahren **3200** endet dann.

Leistungsüberwachung mit mehreren SMC-Engines

[0223] Wie hierin weiter diskutiert wurde, überwachen Leistungsmonitore (PMs), wie beispielsweise

der PM **236** von **Fig. 2**, der PM **360** von **Fig. 3** und der PM **430** von **Fig. 4**, die Gesamtleistung und/oder den Ressourcenverbrauch der entsprechenden Komponenten, die in der PPU **200** enthalten sind. Die Leistungsmonitore (PMs) sind in einem Leistungsüberwachungssystem enthalten, das eine Leistungsüberwachung und Profilerstellung über mehrere SMC-Engines **700** hinweg bereitstellt. Das Leistungsüberwachungssystem profiliert gleichzeitig oder im Wesentlichen gleichzeitig mehrere VMs und Verarbeitungskontexte, die in den VMs ausgeführt werden. Das Leistungsüberwachungssystem isoliert die mehreren VMs und die mehreren Verarbeitungskontexte, die in den VMs ausgeführt werden, in Bezug darauf voneinander, wie Leistungsdaten erzeugt und erfasst werden, um ein Heraustreten von Leistungsdaten zwischen den VMs zu verhindern. Die PMs **232** und zugehörige Zähler innerhalb des Leistungsdaten-Überwachungssystems nachverfolgen die Zuordnung von Leistungsdaten zu bestimmten SMC-Engines **700**. Im Fall gemeinsam genutzter Ressourcen und Einheiten, bei denen die Zuweisung nicht zu einer bestimmten SMC-Engine **700** zurückverfolgt werden kann, erfasst eine Vorrichtung mit einer höher privilegierten Entität, wie beispielsweise der Hypervisor **124**, Leistungsdaten für die gemeinsam genutzten Ressourcen und Einheiten. Das Leistungsüberwachungssystem profiliert gleichzeitig Rechen-Engines und Grafik-Engines, und profiliert VMs, während die VMs zu anderen PPU-Partitionen **600** und/oder anderen PPUs **200** migrieren. Das Leistungsüberwachungssystem wird nun beschrieben.

[0224] **Fig. 33** ist ein Blockdiagramm eines Leistungsüberwachungssystems **3300** für die PPU **200** von **Fig. 2**, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet das Leistungsüberwachungssystem **3300**, ohne Beschränkung darauf, einen Leistungsmonitor **3310**, Auswahlmultiplexer **3320**, einen Überwachungsbus **3330** und Leistungsmultiplexereinheiten **3340**. Zusammen bilden der Leistungsmonitor **3310** und die Auswahlmultiplexer **3320** ein Leistungsüberwachungsmodul (PMM). Jeder GPC **242**, jede Partitionseinheit **262** und jede Sys-Pipe **230** beinhaltet zumindest ein PMM. Die Leistungsmultiplexereinheiten **3340** sind innerhalb jeder Einheit enthalten, die überwacht wird. Die Logik innerhalb der Leistungsmultiplexereinheiten **3340** ist in einer oder mehreren der FE **540**, SKED **550**, CWD **560** und/oder anderen geeigneten Funktionseinheiten enthalten. Wie hierin weiter beschrieben, kommunizieren alle Komponenten der verschiedenen Leistungsüberwachungssysteme **3300** mit einem Leistungsüberwachungsaggregator (in **Fig. 33** nicht gezeigt). Das Leistungsüberwachungssystem **3300** funktioniert im Wesentlichen genauso wie der PM **236** von **Fig. 2**, der PM **360** von **Fig. 3** und der PM **430**, ausgenommen wie nachstehend weiter beschrieben.

[0225] Im Betrieb ermöglichen die Leistungsmultiplexereinheiten **3340(O) - 3340(P)** eine programmierbare Auswahl von Gruppen von Signalen innerhalb der PPU **200**, die durch einen entsprechenden Leistungsmonitor **3340** überwacht werden können. Jeder Leistungsmultiplexer **3340** kann eine Gruppe von Signalen auswählen, die an den Überwachungsbus **3330** übertragen wird. Eine Teilmenge der Signale von dem Überwachungsbus **3330** wird für die Überwachung durch die Auswahlmultiplexer **3320** ausgewählt. Signale von dem Überwachungsbus **3330**, die nicht über die Auswahlmultiplexer **3320** ausgewählt werden, werden nicht überwacht. Signale von innerhalb der PPU **200** werden gruppenweise mit den Leistungsmultiplexereinheiten **3340(O) - 3340(P)** verbunden, so dass jeweils eine Gruppe zur Überwachung ausgewählt werden kann. Die Leistungsmultiplexereinheit **3340** multiplext die Signale so, dass die Signale in einer bestimmten Signalgruppe als eine Gruppe ausgewählt werden. Die Auswahlwege zu den in der Leistungsmultiplexereinheit **3340** enthaltenen Multiplexern werden über ein oder mehrere Register programmiert, die in dem privilegierten Register-Adressraum **3120** enthalten sind. Infolgedessen sind die bestimmten Signale, die von der Leistungsmultiplexereinheit **3340** an den Überwachungsbus **3330** übertragen werden, programmierbar.

[0226] Der Überwachungsbus **3330** empfängt Signalgruppen von den Leistungsmultiplexereinheiten **3340(O) - 3340(P)**. Jedes an den Überwachungsbus **3330** übertragene Signal wird als Eingangssignal an jeden der ausgewählten Multiplexer **3320** gelegt.

[0227] Ausgewählte Multiplexer **3320** beinhalten einen Satz individueller Multiplexer **3322(O) - 3322(M)** und **3324(O) - 3324(N)**. Die Eingangsseite jedes Multiplexers **3322(O) - 3322(M)** und **3324(O) - 3324(N)** empfängt alle Signale von dem Überwachungsbus **3330** und wählt ein Signal zur Übertragung aus. Die Auswahlwege der Multiplexer **3322(O) - 3322(M)** und **3324(O) - 3324(N)** werden über ein oder mehrere Register programmiert, die in dem privilegierten Register-Adressraum **3120** enthalten sind. Infolgedessen ist der Satz bestimmter Signale, die von den Auswahlmultiplexern **3320** übertragen werden, programmierbar. Die Auswahlmultiplexer **3320** übertragen die ausgewählten Signale an den Leistungsmonitor **3310**.

[0228] Die bestimmten PPU-Signale, die von dem Leistungsmonitor **3310** überwacht werden, sind infolge der Zusammensetzung der programmierbaren Leistungsmultiplexereinheiten **3340** und der programmierbaren Auswahlmultiplexer **3320** programmierbar.

[0229] Der Leistungsmonitor **3310** beinhaltet ein Leistungszählerarray **3312**, ein Schattenzählerarray **3314** und eine Triggerfunktionstabelle **3316**.

Der Leistungsmonitor **3310** empfängt Signale, die von den Auswahlmultiplexern **3320** übertragen werden. Genauer gesagt empfängt das Schattenzählerarray **3314** Signale, die von den Multiplexern **3322(0) - 3322(M)** übertragen werden. In ähnlicher Weise empfängt die Triggerfunktionstabelle **3316** Signale, die von den Multiplexern **3324(0) - 3324(N)** übertragen werden. Wie weiter beschrieben wird, werden Zähler innerhalb des Schattenzählerarrays **3314** auf der Grundlage von von den Multiplexern **3322(0) - 3322(M)** empfangenen Signalen und auf der Grundlage verschiedener Triggerbedingungen aktualisiert. Allgemein beinhaltet das Schattenzählerarray **3314** einen Satz von einem oder mehreren Signalzählern, wobei jeder Zähler immer dann inkrementiert, wenn sich ein von einem entsprechenden Multiplexer **3322** empfangenes Signal in einem bestimmten logischen Zustand befindet. Werte in dem Schattenzählerarray **3314** werden auf der Grundlage bestimmter Signale in Form von Triggerbedingungen an das Leistungszählerarray **3312** übertragen. Das Leistungszählerarray **3312** beinhaltet einen Satz von einem oder mehreren Signalzählern, die den in dem Schattenzählerarray enthaltenen Signalzählern entsprechen.

[0230] In einer Betriebsart werden die Zähler in dem Schattenzählerarray **3314** nach der Übertragung an das Leistungszählerarray **3312** auf null zurückgesetzt, so dass die in dem Schattenzählerarray **3314** gespeicherten Schattenzählerwerte immer der Aktivität seit dem vorherigen Trigger bzw. Auslöser entsprechen.

[0231] Der Leistungsmonitor **3310** ist in Übereinstimmung mit verschiedenen Zählmodi konfigurierbar, die die Anzahl von Zählern definieren, die in dem Leistungszählerarray **3312** und in dem Schattenzählerarray **3314** enthalten sind. Die Zählmodi definieren ferner, wie und wann das Leistungszählerarray **3312** und das Schattenzählerarray **3314** ausgelöst bzw. getriggert werden und wie Daten von dem Leistungszählerarray **3312** an andere Vorrichtungen innerhalb der PPU **200** übertragen werden. Diese verschiedenen Zählmodi können in zwei Hauptleistungsüberwachungsmodi gruppiert werden - Nicht-Streaming-Leistungsüberwachung und Streaming-Leistungsüberwachung.

[0232] In dem Nicht-Streaming-Leistungsüberwachungsmodus ist die Triggerfunktionstabelle **3316** dazu programmiert, von den Multiplexern **3324(0) - 3324(N)** empfangene Signale gemäß bestimmten spezifizierten logischen Signalausdrücken zu kombinieren. Wenn die Bedingungen eines oder mehrerer dieser logischen Signalausdrücke erfüllt sind, sendet die Triggerfunktionstabelle **3316** ein Signal in Form eines Logikauslösers bzw. Logiktriggers **3350** an das Leistungszählerarray **3312**. Im Ansprechen auf den Empfang des Logiktriggers **3350**

tastet das Leistungszählerarray **3312** die aktuellen Werte ab und speichert sie in dem Schattenzählerarray **3314**. Die Werte in dem Leistungszählerarray **3312** sind dann über den privilegierte Register-Adressraum **3120** lesbar.

[0233] In dem Streaming-Leistungsüberwachungsmodus sendet ein Leistungsüberwachungsaggregator (PMA) ein Signal in Form eines PMA-Triggers **3352** an das Leistungszählerarray **3312**. Im Ansprechen auf den Empfang des PMA-Triggers **3352** tastet das Leistungszählerarray **3312** die aktuellen Werte ab und speichert sie in dem Schattenzählerarray **3314**. Der Leistungsmonitor **3310** erzeugt Leistungsmonitor (PMM)-Datensätze, die, ohne darauf beschränkt zu sein, die Werte in dem Leistungszählerarray **3312** zum Zeitpunkt des Empfangs des PMA-Triggers **3352** von dem PMA, eine Zählung der Gesamtzahl von PMA-Triggern, auf die der Leistungsmonitor **3310** angesprochen hat, eine SMC-Engine-ID und eine PMM-ID, die das PMM, das den Datensatz erzeugt hat, in dem System eindeutig identifiziert, beinhalten können. Diese PMM-Aufzeichnungen werden dann an einen PMM-Router übertragen, der einem oder mehreren Leistungsmonitoren **3310** zugeordnet ist. Der PMM-Router wiederum überträgt die PMM-Datensätze an den PMA. In einigen Ausführungsformen kann die PMM-ID für jeden Leistungsmonitor **3310** über ein oder mehrere Register programmiert werden, die in dem privilegierte Register-Adressraum **3120** enthalten sind.

[0234] Allgemein befindet sich ein bestimmter Leistungsmonitor **3310** in einem bestimmten Leistungsüberwachungssystem **3300** in der PPU **200** innerhalb derselben Taktfrequenzdomäne wie die Signale, die von diesem bestimmten Leistungsmonitor **3310** überwacht werden. Ein bestimmter Leistungsmonitor **3310** kann sich jedoch innerhalb derselben Taktfrequenzdomäne oder innerhalb einer anderen Taktfrequenzdomäne relativ zu einem anderen Leistungsmonitor in PPU **200** befinden.

[0235] Nachstehend werden verschiedene Konfigurationen von Leistungsmultiplexereinheiten **3340** beschrieben.

[0236] Die Fig. **34A-34B** veranschaulichen verschiedene Konfigurationen der Leistungsmultiplexereinheiten **3340** vom Fig. **33**, entsprechend verschiedenen Ausführungsformen.

[0237] Wie in Fig. **34A** gezeigt ist, beinhaltet eine erste Konfiguration einer Leistungsmultiplexereinheit **3340(0)**, ohne Beschränkung darauf, Signalgruppen A **3420(0) - 3420(P)**, Signalgruppen B **3430(0) - 3430(Q)** sowie Multiplexer **3412(0)** und **3412(1)**. Im Betrieb wählt der Multiplexer **3412(0)** eine der Signalgruppen A **3420(0) - 3420(P)** aus, wobei jede der Signalgrup-

pen A **3420(0)** - **3420(P)** eine Untergruppe innerhalb einer größeren Signalgruppe C ist. Der Multiplexer **3412(0)** wählt eine der Signalgruppen A **3420(0)** - **3420(P)** aus und überträgt die ausgewählte Signalgruppe an den Überwachtungsbus **3330**. In ähnlicher Weise wählt der Multiplexer **3412(1)** eine der Signalgruppen B **3430(0)** - **3430(Q)** aus, wobei jede der Signalgruppen B **3430(0)** - **3430(Q)** eine Untergruppe innerhalb einer größeren Signalgruppe D ist. Der Multiplexer **3412(1)** wählt eine der Signalgruppen B **3430(0)** - **3430(Q)** aus und überträgt die ausgewählte Untergruppe an den Überwachtungsbus **3330**. Die Auswahlwege der Multiplexer **3412**, die in der Leistungsmultiplexereinheit **3340(0)** enthalten sind, werden über ein oder mehrere Register programmiert, die in dem privilegierte Register-Adressraum **3120** enthalten sind. Infolgedessen ist der von der Leistungsmultiplexereinheit **3340(0)** übertragene Satz von Signalen programmierbar.

[0238] Wie in **Fig. 34B** gezeigt ist, beinhaltet eine zweite Konfiguration einer Leistungsmultiplexereinheit **3340(1)**, ohne Beschränkung darauf, Signalgruppen C **3440(0)** - **3440(R)** und einen Multiplexer **3412(2)**. Im Betrieb wählt der Multiplexer **3412(2)** eine der Signalgruppen C **3440(0)** - **3440(R)** aus, wobei jede der Signalgruppen C **3440(0)** - **3440(R)** eine Untergruppe innerhalb einer größeren Signalgruppe E ist. Der Multiplexer **3412(2)** wählt eine der Signalgruppen C **3440(0)** - **3440(R)** aus und überträgt die ausgewählte Signalgruppe an den Überwachtungsbus **3330**. In der Konfiguration der Leistungsmultiplexereinheiten **3340(1)** werden mehrere Signale an mehrere Signalgruppen übertragen. Insbesondere wird ein Signal **C1 3450** sowohl an die Signalgruppe C **3440(0)** als auch an die Signalgruppe C **3440(1)** übertragen. In ähnlicher Weise wird ein Signal **C2 3452** sowohl an die Signalgruppe C **3440(1)** als auch an die Signalgruppe C **3440(2)** übertragen. Die in der Leistungsmultiplexereinheit **3340(1)** enthaltenen Auswahlwege für den Multiplexer **3412(2)** werden über ein oder mehrere Register programmiert, die im dem privilegierte Register-Adressraum **3120** enthalten sind. Infolgedessen ist der von der Leistungsmultiplexereinheit **3340(1)** übertragene Satz von Signalen programmierbar. Die Konfiguration der Leistungsmultiplexereinheit **3340(1)** kann nützlich sein, wenn die Bereitstellung von Signalen in mehreren Signalgruppen **3440** die Sichtbarkeit bestimmter Signalgruppen in einem einzigen Durchlauf des Leistungsüberwachungssystems **3300** erleichtert.

[0239] **Fig. 35** ist ein Blockdiagramm eines Leistungsmonitor-Aggregationssystems **3500** für die PPU **200** von **Fig. 2**, gemäß verschiedenen Ausführungsformen. Wie gezeigt ist, beinhaltet das Leistungsmonitor-Aggregationssystem **3500**, ohne Beschränkung darauf, die GPCs **242(0)** - **242(M)**, die Partitionseinheiten **262(0)** - **262(N)**, eine Kreuzschieneneinheit

250, eine Steuerungskreuzschiene und SMC-Arbitr **510**, ein PM-Verwaltungssystem **3530** und ein Leistungsanalysesystem **3540**.

[0240] Im Betrieb führen die GPCs **242(0)** - **242(M)** verschiedene Verarbeitungsaufgaben für eine oder mehrere Sys-Pipes **230** aus. Jeder GPC **242** beinhaltet mehrere Parallelverarbeitungskerne, die in der Lage sind, eine große Anzahl von Threads gleichzeitig und mit einem beliebigen Grad an Unabhängigkeit und/oder Isolierung von anderen GPCs **242** auszuführen. Jeder der GPCs **242(0)** - **242(M)** beinhaltet eine oder mehrere PMs **360(0)** - **360(M)** und einen GPC-PPMM-Router **3514(0)** - **3514(M)**. Die PMs **360(0)** - **360(M)** funktionieren im Wesentlichen ähnlich wie der Leistungsmonitor **3310** in **Fig. 33**. Die PMs **360(0)** - **360(M)** erzeugen PMM-Datensätze, die Leistungsdaten für die entsprechenden GPCs **242(0)** - **242(M)** beinhalten. Die PMs **360(0)** - **360(M)** übertragen diese PMM-Datensätze an die entsprechenden GPC-PPMM-Router **3514(0)** - **3514(M)** und empfangen Daten von diesen. Die GPC-PPMM-Router **3514(0)** - **3514(M)** übertragen die PMM-Datensätze über die Kreuzschieneneinheit **250** an das PMM-Verwaltungssystem **3530**.

[0241] Die Partitionseinheiten **262(0)** - **262(N)** stellen einen Speicherzugriff mit hoher Bandbreite für die DRAMS innerhalb des PPU-Speichers bereit (in **Fig. 35** nicht gezeigt). Jede Partitionseinheit **262** führt Speicherzugriffsoperationen mit einem anderen DRAM parallel zueinander aus, wodurch die verfügbare Speicherbandbreite des PPU-Speichers effizient genutzt wird. Jede der Partitionseinheiten **262(0)** - **262(N)** beinhaltet einen oder mehrere PMs **430(0)** - **430(N)** und einen Partitionseinheit (PU)-PMM-Router **3524(0)** - **3524(N)**. Die PMs **430(0)** - **430(N)** funktionieren im Wesentlichen ähnlich wie der Leistungsmonitor **3310** von **Fig. 33**. Die PMs **430(0)** - **430(N)** erzeugen PMM-Datensätze, die Leistungsdaten für die entsprechenden Partitionseinheiten **262(0)** - **262(N)** enthalten. Die PMs **430(0)** - **430(N)** übertragen diese PMM-Datensätze an die entsprechenden PU-PPMM-Router **3524(0)** - **3524(N)** und empfangen Daten von diesen. Die PU-PPMM-Router **3524(0)** - **3524(N)** wiederum übertragen die PMM-Datensätze über die Steuerungskreuzschiene und SMC-Arbitr **510** an das PMM-Verwaltungssystem **3530**.

[0242] Das PM-Managementsystem **3530** steuert die Sammlung von PMM-Datensätzen und speichert die PMM-Datensätze zu Berichtszwecken. Das PM-Verwaltungssystem **3530** beinhaltet, ohne Beschränkung darauf, Systemleistungsmonitore **3532**, einen System (SYS)-PMM-Router **3534**, einen Leistungsmonitoraggregator (PMA) **3536**, einen Hochgeschwindigkeits-Hub (HSHUB) und Transferlogik **3539**.

[0243] Die System-PMs **3532** funktionieren im Wesentlichen ähnlich wie der Leistungsmonitor **3310** von **Fig. 33**. Die System-PMs **3532** erzeugen PMM-Datensätze, die Leistungsdaten für systemweite Komponenten enthalten, die nicht in einem bestimmten GPC **242** oder einer bestimmten Partitionseinheit **262** enthalten sind. Die System-PMs **3532** übertragen diese PMM-Datensätze an den System-PMM-Router **3534** und empfangen Daten von diesem. Der System-PMM-Router **3534** wiederum überträgt die PMM-Datensätze an den PMA **3536**.

[0244] Der PMA **3536** erzeugt Trigger für die verschiedenen Leistungsmonitore, einschließlich der PMs **360(O) - 360(M)**, der PMs **430(O) - 430(N)** und der System PMs **3532**. Der PMA **3536** erzeugt diese Trigger über zwei Techniken. Bei der ersten Technik erzeugt der PMA **3536** Trigger im Ansprechen auf Signale, die von jeder der Sys-Pipes **230** gesendet werden, wenn die Sys-Pipes **230** Befehle von der Hostschnittstelle **220** empfangen. Bei einer zweiten Technik erzeugt der PMA **3536** Trigger durch periodisches Übertragen von programmgesteuerten Auslöser- bzw. Triggerimpulsen an die PMs. Allgemein beinhaltet das Leistungsmonitor-Aggregationssystem **3500** zumindest einen programmierbaren Triggerimpuls-generator, der jeder Sys-Pipe **230** entspricht, zusätzlich zu einem weiteren Triggerimpuls-generator, der von jeder Sys-Pipe **230** unabhängig ist. Der PMA **3536** überträgt die Trigger über die Steuerungskreuzschiene und SMC-Arbitr **510** an die GPC-PMM-Router **3514(O) - 3514(M)** und die PU-PMs **3524(O) - 3524(N)**. Der PMA **3536** überträgt die Trigger über eine in dem PM-Managementsystem **3530** interne Kommunikationsverbindung direkt an den System-PMM-Router **3534**. Die PMM-Router übertragen die PMA-Trigger an die entsprechenden PMs. In einigen Ausführungsformen nehmen diese Trigger die Form von Triggerpaketen an, die nachstehend in Verbindung mit **Fig. 36** beschrieben werden.

[0245] **Fig. 36** veranschaulicht das Format von Triggerpaketen, die dem Leistungsmonitor-Aggregationssystem **3500** von **Fig. 3500** zugeordnet sind, gemäß verschiedenen Ausführungsformen. Der Zweck der Triggerpakete besteht darin, Informationen über die Quelle des Triggers an die Leistungsmonitore **3310** zu übermitteln. Jeder der Leistungsmonitore **3310** nutzt diese Informationen zum Bestimmen, ob er auf einen bestimmten Trigger reagieren soll oder nicht. In dieser Hinsicht enthalten die Triggerpakete Informationen, die von jedem Leistungsmonitor **3310** verwendet werden können, um zu bestimmen, ob auf ein bestimmtes Triggerpaket reagiert werden soll oder nicht. Jeder Leistungsmonitor **3310**, der einem bestimmten SMC-Modul **700** zugeordnet ist, wird mit einer SMC-Engine-ID programmiert, die diesem SMC-Modul **700** entspricht. Ein solcher Leistungsmonitor **3310** reagiert auf einzelne SMC-Triggerpakete, die die gleiche SMC-Engine-ID enthalten. Jeder Leis-

tungsmonitor **3310**, der nicht mit einer bestimmten SMC-Engine **700** assoziiert ist oder mit einer ungültigen SMC-Engine-ID programmiert ist, reagiert nicht auf Pro-SMC-Triggerpakete. Stattdessen reagieren solche Leistungsmonitore **3310** auf gemeinsam genutzte („shared“) Triggerpakete.

[0246] Ein Diagramm **3600** veranschaulicht das allgemeine Format von Triggerpaketen. Wie gezeigt ist, beinhaltet das Diagramm **3600** einen Pakettyp **3602**, der anzeigt, dass das Paket einen PM-Trigger ist, einen Triggertyp **3604** und eine Triggernutzlast **3606**. Der PM-Triggertyp **3604** ist ein Aufzählungswert, der den Typ des Triggerformats identifiziert. Um beispielsweise drei verschiedene Triggerpakettypen zu identifizieren, könnte der PM-Triggertyp **3604** ein 2-Bit-Wert sein. Die Triggernutzlast **3606** beinhaltet Daten, die sich abhängig von dem PM-Triggertyp **3604** unterscheiden. Nachstehend werden drei verschiedene Typen von Triggerpaketen beschrieben, wobei die drei Typen von Triggerpaketen den drei Kategorien von Leistungsüberwachungsdaten (Legacy-Daten, Pro-SMC-Daten und gemeinsam genutzte Daten) entsprechen.

[0247] Ein Diagramm **3610** veranschaulicht das Format von Legacy-Triggerpaketen. Legacy-Triggerpakete beinhalten einen Pakettyp **3602**, der angibt, dass das Paket ein PM-Trigger ist, und einen Triggertyp **3614**, der angibt, dass das Triggerpaket ein Legacy-Triggerpaket ist. Die Triggernutzlast **3606** des Legacy-Triggerpakets beinhaltet ein ungenutztes Feld **3616**.

[0248] Ein Diagramm **3620** veranschaulicht das Format von Pro-SMC-Triggerpaketen. Pro-SMC-Triggerpakete beinhalten einen Pakettyp **3602**, der angibt, dass das Paket ein PM-Trigger ist, und einen Triggertyp **3624**, der angibt, dass das Triggerpaket ein SMC-Triggerpaket ist. Die Triggernutzlast **3606** des Pro-SMC-Triggerpakets beinhaltet ein SMC-Engine-ID-Feld. Das SMC-Engine-ID-Feld **3626** identifiziert die bestimmte SMC-Engine **700**, auf die sich der Trigger bezieht.

[0249] Ein Diagramm **3630** veranschaulicht das Format von gemeinsam genutzten Triggerpaketen. Gemeinsam genutzte Triggerpakete beinhalten einen Pakettyp **3602**, der angibt, dass das Paket ein PM-Trigger ist, und einen Triggertyp **3634**, der angibt, dass das Triggerpaket ein gemeinsam genutztes Triggerpaket ist. Die Triggernutzlast **3606** des gemeinsam genutzten Triggerpakets beinhaltet ein ungenutztes Feld.

[0250] Der Typ des von dem PMA **3536** gesendeten Triggerpakets wird durch die Quelle des Triggers und ein oder mehrere Register bestimmt, die entsprechend jeder Triggerquelle in dem privilegierte Register-Adressraum **3120** enthalten sind und dazu pro-

grammiert sind, den Triggerpakettyp anzugeben, den der PMA für diese Quelle senden soll. In einer Betriebsart ist der PMA so programmiert, dass Triggerpakete, die im Ansprechen auf eine einer SMC-Engine zugeordneten Quelle erzeugt werden, pro SMC-Triggerpakete sind, wobei die SMC-Engine-ID auf die entsprechende SMC-Engine eingestellt ist und Triggerpakete, die im Ansprechen auf Quellen erzeugt werden, die nicht einer SMC-Engine zugeordnet sind, gemeinsam genutzte Triggerpakete sind. Triggerpakete können mit jeder technisch machbaren Rate erzeugt werden, bis hin zu und einschließlich eines Triggerpakets pro Rechenzyklus.

[0251] Im Ansprechen auf den Empfang eines Triggerpakets von dem PMA **3536** prüft jeder PM den Triggertyp und die Triggernutzlast, um zu bestimmen, ob der PM auf den Trigger reagieren soll. Jeder PM antwortet unbedingt auf Legacy-Triggerpakete. Im Fall von Pro-SMC-Triggerpaketen antwortet der PM nur dann, wenn die in der Triggernutzlast enthaltene SMC-Engine-ID mit der SMC-Engine-ID übereinstimmt, die dem PM durch Programmierung eines Registers in dem privilegierte Register-Adressraums **3120** zugewiesen wurde. Die Programmierung einer ungültigen SMC-Engine-ID in dieses Register stellt sicher, dass der PM nicht auf Pro-SMC-Triggerpakete reagiert. Im Fall gemeinsam genutzter Triggerpakete antwortet der PM nur dann, wenn der PM über ein Register in dem privilegierte Register-Adressraum **3120** entsprechend programmiert wurde. In einer Betriebsart sind alle PMs, die Überwachungseinheiten sind, die eindeutig einer SMC-Engine zugeordnet sind, dazu programmiert, auf Pro-SMC-Triggerpakete mit der entsprechenden SMC-Engine-ID-Nutzlast zu reagieren, und sind alle anderen PMs dazu programmiert, auf gemeinsam genutzte Triggerpakete, aber nicht auf Pro-SMC-Triggerpakete zu reagieren.

[0252] In dem Fall, in dem ein PM ermittelt, dass eine Reaktion auf den Trigger gerechtfertigt ist, fragt der PM die in dem jeweiligen PM enthaltenen Zähler ab. Die antwortenden PMs übermitteln dann PMM-Datensätze, die die abgefragten Zählerwerte, die Gesamtzahl von Triggern, auf die reagiert wurde, die dem PM zugewiesene SMC-Engine-ID und die PMM-ID, die den PM innerhalb des Systems eindeutig identifiziert, enthalten. Die PMM-Router, der PMA **3536** und/oder das Leistungsanalysesystem **3540** verwenden die PMM-ID, um zu identifizieren, welcher PM den entsprechenden PMM-Datensatz übertragen hat. Die PMM-Router übertragen diese Datensätze dann an den PMA **3536**. Genauer gesagt übertragen die GPC-PMM-Router **3514(O)** - **3514(M)** die PMM-Datensätze über die Kreuzschieneneinheit **250** und den Hochgeschwindigkeits-Hub **3538** an den PMA **3536**. Die PU-PPMM-Router **3524(O)** - **3524(N)** übertragen die PMM-Datensätze über die Steuerungskreuzschiene und SMC-Arbiter **510** an den PMA

3536. Der System-PPMM-Router **3534** überträgt die PMM-Datensätze über eine interne Kommunikationsverbindung direkt an das PM-Verwaltungssystem **3530**. Auf diese Weise empfängt der PMA **3536** PMM-Datensätze von allen relevanten PMs in der PPU **200**.

[0253] In einigen Ausführungsformen erzeugt der PMA **3536** bei der Übertragung eines Triggers zusätzlich PMA-Datensätze. Allgemein besteht der Zweck eines PMA-Datensatzes darin, zu einem Zeitstempel aufzuzeichnen, bei dem ein bestimmter PMA-Trigger erzeugt wird, und die entsprechenden PMM-Datensätze von den Leistungsmonitoren **3310**, die auf den PMA-Auslöser zu diesem Zeitstempel geantwortet haben, zuzuordnen. Die PMA-Datensätze enthalten, ohne Beschränkung darauf, einen Zeitstempel, eine der Quelle des PMA-Triggers zugeordnete SMC-Engine-ID, eine Gesamtzahl von Triggern, die von Quellen mit derselben SMC-Engine-ID erzeugt wurden, und zugehörige Metadaten. Wenn die Leistungsmonitore **3310** einen Trigger empfangen, erzeugen die Leistungsmonitore **3310** ebenfalls PMM-Datensätze mit einer Triggerzählung. Anschließend können beim Parsen von PMA- und PMM-Datensätzen PMM-Datensätze mit einer bestimmten Triggeranzahl PMA-Einträgen mit derselben Triggeranzahl zugeordnet werden. Auf diese Weise wird der den PMM-Datensätzen entsprechende Zeitstempel auf der Grundlage des Zeitstempels des zugehörigen PMA-Datensatzes erstellt. Infolgedessen wird das Verhalten der PPU **200**, das von den PMM-Datensätzen widerspiegelt wird, genau einem Zeitbereich zugeordnet, der durch zwei benachbarte PMA-Trigger aus derselben Quelle begrenzt wird.

[0254] Beim Empfangen von PMM-Datensätzen und bei der Erzeugung von PMA-Datensätzen speichert der PMA **3536** die PMM-Datensätze und die PMA-Datensätze über den Hochgeschwindigkeits-Hub **3538** in einem Datenspeicher in Form eines Datensatzpuffers in dem PPU-Speicher. Der Hochgeschwindigkeits-Hub **3538** überträgt die PMM-Datensätze und die PMA-Datensätze an die Partitionseinheiten **262(O)** - **262(N)**. Die Partitionseinheiten speichern dann die PMM-Datensätze und die PMA-Datensätze in dem Datensatzpuffer in dem PPU-Speicher. Zusätzlich oder alternativ überträgt der Hochgeschwindigkeits-Hub **3538** die PMM-Datensätze und die PMA-Datensätze über die Transferlogik **3539** an das Leistungsanalysesystem **3540**. In einigen Ausführungsformen können der Hochgeschwindigkeits-Hub **3538**, die Übertragungslogik **3539** und das Leistungsanalysesystem **3540** über eine PCIe-Verbindung miteinander kommunizieren. Ein Benutzer kann die PMM-Datensätze und die PMA-Datensätze auf dem Leistungsanalysesystem **3540** einsehen, um das Verhalten der PPU **200** zu charakterisieren, wie es in den PMM-Datensätzen widerspiegelt wird. Das Leistungsanalysesystem **3540**

sammelt PMM-Datensätze und PMA-Datensätze, die die gleiche Triggeranzahl haben. Dann verwendet das Leistungsanalysesystem **3540** den Zeitstempel aus dem PMA-Datensatz und die Leistungsdaten aus den PMM-Datensätzen, die die gleiche Triggeranzahl haben, um den den Leistungsdaten zugeordneten Zeitstempel zu bestimmen. In einigen Ausführungsformen kann das Leistungsanalysesystem **3540** auf den Leistungsdatensatzpuffer als virtuellen Speicher zugreifen. Infolge der Platzierung von Leistungsdatensatzpuffern in verschiedenen virtuellen Adressräumen können Leistungsüberwachungsdaten für verschiedene SMC-Engines **700** voneinander isoliert sein, wie nachstehend beschrieben wird.

[0255] Der PMA **3536** stellt eine Isolierung der Leistungsüberwachungsdaten zwischen den verschiedenen SMC-Engines **700** bereit. Insbesondere sortiert der PMA **3536** die PMM-Datensätze und die PMA-Datensätze auf der Grundlage der Betriebsart in Kategorien. Wenn die PPU **200** im Legacy-Modus arbeitet, führt die PPU **200** Aufgaben als ein einzelner Cluster von Hardwareressourcen aus, und nicht als separate PPU-Partitionen **600** mit separaten SMC-Engines **700**. Im Legacy-Modus speichert der PMA **3536** die PMM-Datensätze und die PMA-Datensätze in einer einzigen Kategorie als einen einzigen Satz von Leistungsüberwachungsdaten. Wenn die PPU **200** im SMC-Modus arbeitet, führt die PPU **200** Aufgaben als separate PPU-Partitionen **600** mit separaten SMC-Engines **700** aus. Im SMC-Modus sortiert und speichert der PMA **3536** die PMM-Datensätze und die PMA-Datensätze unter Verwendung des SMC-Engine-ID-Felds der PMM-Datensätze und der PMA-Datensätze in verschiedenen Kategorien. Aufzeichnungen bzw. Datensätze mit jeder SMC-Engine-ID werden in einem separaten Datenspeicher in Form eines Aufzeichnungs- bzw. Datensatzpuffers gespeichert, auf den von einem eigenen virtuellen Adressraum aus zugegriffen werden kann, der mit der entsprechenden SMC-Engine **700** übereinstimmt. Wie vorstehend beschrieben wurde, enthalten die PMM-Datensätze und die PMA-Datensätze, die nicht zu einer bestimmten SMC-Engine **700** rückverfolgbar sind, eine ungültige SMC-Engine-ID. Der PMA speichert diese Datensätze in einem separaten Datenspeicher in Form eines Nicht-SMC-Datensatzpuffers in einem virtuellen Adressraum, auf den jede autorisierte Entität zugreifen kann, die über ausreichende Privilegien verfügt, um auf die Daten aller SMC-Engines **700** zuzugreifen. Solche autorisierten Einheiten beinhalten unter anderem einen Hypervisor **124** in einer virtualisierten Umgebung und einen Root-Benutzer oder ein Betriebssystemkernel in einer nicht virtualisierten Umgebung. Jede SMC-Engine **700** kann auf einige oder alle Leistungsüberwachungsdaten in dem Nicht-SMC-PMA-Datensatzpuffer durch Anfordern der Daten von der autorisierten Entität zugreifen.

[0256] In einigen Ausführungsformen ist der PMA **3536** derart konfiguriert, dass Trigger, die jeder SMC-Engine **700** entsprechen, so generiert werden, dass sie mit Kontextwechsel-Ereignissen für dieselbe SMC-Engine **700** zusammenfallen. In solchen Ausführungsformen sind die PMs dazu konfiguriert, die Zähler in dem Schattenzählerarray **3314** nach jedem Trigger auf null zurückzusetzen, so dass die von den PMs an den PMA **3536** übertragenen Daten für jede SMC-Engine-ID einem individuellen Kontext oder einer individuellen VM zugeordnet werden können, während die Unterteilung in Zeitscheiben aktiviert ist.

[0257] Fig. 37 ist ein Ablaufdiagramm von Verfahrensschritten für die Überwachung der Leistung der PPU **200** von Fig. 2, gemäß verschiedenen Ausführungsformen. Obwohl die Verfahrensschritte in Verbindung mit den Systemen der Fig. 1-23 beschrieben werden, versteht sich für den Fachmann, dass jedes System, das dazu konfiguriert ist, die Verfahrensschritte in beliebiger Reihenfolge auszuführen, in den Anwendungsbereich der Erfindung fällt.

[0258] Wie gezeigt ist, beginnt ein Verfahren **3700** bei Schritt **3702**, in dem der PMA **3536** einen Trigger erzeugt und an die PMs **3310** überträgt. Ferner erzeugt der PMA **3536** einen entsprechenden PMA-Datensatz, der einen Zeitstempel und optional eine SMC-Engine-ID entsprechend der Quelle des Triggers enthält. Die PMs **3310** empfangen einen Trigger zur Abtastung von Leistungsdaten.

[0259] In Schritt **3704** bestimmen die PMs **3310** im Ansprechen auf den Empfang des Triggers, ob eine Reaktion auf den Trigger gerechtfertigt ist. Jeder PM **3310** prüft den Triggertyp und die Triggerlast aus dem Triggerpaket, um zu bestimmen, ob der PM **3310** auf den Trigger reagieren soll. Jeder PM **3310** antwortet bedingungslos auf unbedingt auf Legacy-Triggerpakete. Im Fall von Per-SMC-Triggerpaketen reagiert der PM **3310** nur dann, wenn die in der Triggernutzlast enthaltene SMC-Engine-ID mit der SMC-Engine-ID übereinstimmt, die dem PM **3310** durch Programmierung eines Registers in dem privilegierte Register-Adressraum **3120** zugewiesen wurde. Die Programmierung einer ungültigen SMC-Engine-ID in dieses Register stellt sicher, dass der PM **3310** nicht auf die Pro-SMC-Triggerpakete reagiert. Im Fall von gemeinsam genutzten Triggerpaketen antwortet der PM **3310** nur dann, wenn der PM **3310** über ein Register in dem privilegierte Register-Adressraum **3120** dazu programmiert wurde. In einer Betriebsart sind alle PMs **3310**, die Überwachungseinheiten sind, die eindeutig einer SMC-Engine **700** zugeordnet sind, dazu programmiert, mit der entsprechenden SMC-Engine-ID-Nutzlast auf Pro-SMC-Triggerpakete zu reagieren. Alle anderen PMs **3310** sind so programmiert, dass sie auf gemeinsam genutzte Triggerpakete reagieren, nicht aber auf Pro-SMC-Triggerpakete.

[0260] Falls eine Antwort gerechtfertigt ist, fragt das Leistungszählerarray **3312** die aktuellen Werte ab und speichert diese in dem Schattenzählerarray **3314**. Im Nicht-Streaming-Modus können andere Komponenten die Werte in den Leistungszählerarrays über den privilegierte Register-Schnittstellen-Hub **512** lesen.

[0261] Im Streaming-Modus schreitet das Verfahren zu Schritt **3706** fort, in dem die PMs **3310** abgetastete Leistungsdaten übertragen und der PMA **3536** die abgetasteten Leistungsdaten von den PMs **3310** empfängt. Die PMs **3310** erzeugen PMM-Datensätze, die die Werte in dem Leistungszählerarray **3312** zu dem Zeitpunkt, zu dem der PMA-Trigger **3352** von dem PMA **3536** empfangen wurde, enthalten. Diese PMM-Datensätze werden dann an einen PMM-Router übertragen, der dem jeweiligen Leistungsmonitor **3310** zugeordnet ist. Der PMM-Router wiederum überträgt die PMM-Datensätze an den PMA **3536**.

[0262] In Schritt **3708** sortiert der PMA **3536** die PMM-Datensätze und die PMA-Datensätze auf der Grundlage der Betriebsart in Kategorien. Wenn die PPU **200** im Legacy-Modus arbeitet, sortiert der PMA **3536** die PMM-Datensätze und die PMA-Datensätze als einen einzigen Satz von Leistungsüberwachungsdaten in eine einzige Kategorie. Wenn die PPU **200** im SMC-Modus arbeitet, sortiert der PMA **3536** die PMM-Datensätze und die PMA-Datensätze unter Verwendung des SMC-Engine-ID-Felds der PMM-Datensätze und der PMA-Datensätze in verschiedene Kategorien. Wie vorstehend beschrieben wurde, enthalten die PMM-Datensätze und die PMA-Datensätze, die nicht zu einer bestimmten SMC-Engine **700** rückverfolgbar sind, eine ungültige SMC-Engine-ID. Der PMA **3536** sortiert diese PMM-Datensätze und diese PMA-Datensätze in eine separate Kategorie.

[0263] In Schritt **3710** speichert der PMA **3536** die PMM-Datensätze und/oder die PMA-Datensätze in einem PMA-Datensatzpuffer in dem PPU-Speicher. Wenn die PPU **200** im Legacy-Modus arbeitet, speichert der PMA **3536** die PMM-Datensätze und die PMA-Datensätze in einer einzigen Kategorie als einen einzigen Satz von Leistungsüberwachungsdaten. Wenn die PPU **200** im SMC-Modus arbeitet, speichert der PMA **3536** die PMM-Datensätze und die PMA-Datensätze, die jeder SMC-Engine-ID zugeordnet sind, in einem separaten Datenspeicher, auf den von einem eigenen virtuellen Adressraum aus zugegriffen werden kann, der mit der entsprechenden SMC-Engine **700** übereinstimmt. Der PMA **3536** speichert die PMM-Datensätze und die PMA-Datensätze, die nicht zu einer bestimmten SMC-Engine **700** rückverfolgbar sind, in einem separaten Datenspeicher in Form eines Nicht-SMC-PMA-Satzpuffers in einem virtuellen Adressraum, auf den nur eine autorisierte Entität zugreifen kann, die über ausreichenden

de Berechtigungen für den Zugriff auf die Daten aller SMC-Engines **700** verfügt. Solche autorisierte Entitäten beinhalten, ohne Beschränkung darauf, einen Hypervisor **124** in einer virtualisierten Umgebung und einen Root-Benutzer oder einen Betriebssystemkernel in einer nicht virtualisierten Umgebung. Jede SMC-Engine **700** kann auf einige oder alle Leistungsüberwachungsdaten in dem Nicht-SMC-PMA-Datensatzpuffer durch Anfordern der Daten von der autorisierten Einheit zugreifen.

[0264] Genauer gesagt streamt der PMA **3536** die PMM-Datensätze und die PMA-Datensätze zu dem Hochgeschwindigkeits-Hub **3538**. Der Hochgeschwindigkeits-Hub **3538** überträgt die PMM-Datensätze und die PMA-Datensätze an die Partitionseinheiten **262(0)** - **262(N)**. Die Partitionseinheiten speichern dann die PMM-Datensätze und die PMA-Datensätze in dem PMA-Datensatzpuffer in dem PPU-Speicher. Der PMA-Datensatzpuffer für jeden Datensatz wird auf der Grundlage des SMC-Engine-ID-Felds des Datensatzes so gewählt, dass sich jeder Datensatz letztlich in dem PPU-Speicher befindet, auf den in einem virtuellen Adressraum zugegriffen werden kann, der der SMC-Engine entspricht, der der PMA-Datensatzpuffer entspricht. Die PMM-Datensätze und die PMA-Datensätze, die nicht zu einer bestimmten SMC-Engine **700** rückverfolgbar sind, befinden sich in einem separaten Datenspeicher, auf den nur eine autorisierte Entität Zugriff hat.

[0265] In Schritt **3712** überträgt der PMA **3536** die PMA-Datensätze und/oder die PMM-Datensätze über den Hochgeschwindigkeits-Hub **3538** und die Übertragungslogik **3539** an das Leistungsanalyzesystem **3540**. Zusätzlich oder alternativ greift das Leistungsanalyzesystem **3540** über eine oder mehrere virtuelle Adressen in einem virtuellen Adressraum auf die PMA-Datensätze und/oder die PMM-Datensätze zu. Allgemein beinhaltet das Leistungsanalyzesystem **3540** eine Softwareanwendung, die auf der CPU **110** und/oder einem anderen technisch machbaren Prozessor ausgeführt wird. Das Leistungsanalyzesystem **3540** greift direkt auf den virtuellen Speicher zu, um auf die PMA-Datensätze und/oder die PMM-Datensätze zuzugreifen. Der virtuelle Speicher kann der PPU **200** und/oder der CPU **110** zugeordnet sein. Ein Benutzer kann die PMA-Datensätze und/oder die PMM-Datensätze auf dem Leistungsanalyzesystem **3540** einsehen, um das Verhalten der PPU **200** zu charakterisieren, wie es sich in den PMA-Datensätzen und/oder den PMM-Datensätzen widerspiegelt. Das Verfahren **3700** wird dann beendet.

Leistungs- und Taktfrequenzverwaltung bei SMC-Engines

[0266] Komplexe Systeme, wie beispielsweise die PPU **200** in **Fig. 2**, können erhebliche Mengen an Energie verbrauchen. Genauer gesagt können be-

stimmte Komponenten innerhalb der PPU **200** zu verschiedenen Zeitpunkten unterschiedlich viel Strom verbrauchen. In einem Beispiel kann eine Komponente in einer PPU-Partition **600** rechen- und/oder grafikintensive Aufgaben ausführen und dadurch den Stromverbrauch relativ zu anderen PPU-Partitionen **600** erhöhen. In einem anderen Beispiel kann eine PPU-Partition **600** aufgrund von Leckstrom und verwandten Faktoren auch im Leerlauf Strom verbrauchen. Ferner kann ein erhöhter Stromverbrauch zu einer höheren Betriebstemperatur führen, was wiederum zu einer verringerten Leistung führen kann. Infolgedessen beinhaltet die PPU **200** eine Leistungs- und Taktfrequenz-Verwaltung, die berücksichtigt, wie sich der Stromverbrauch innerhalb einer PPU-Partition **600** negativ auf die Leistung anderer PPU-Partitionen **600** auswirken kann.

[0267] Fig. 38 ist ein Blockdiagramm eines Leistungs- und Taktfrequenz-Verwaltungssystems **3800** für die PPU **200** von Fig. 2, gemäß verschiedenen Ausführungsformen. Das Leistungs- und Taktfrequenz-Verwaltungssystem **3800** beinhaltet, ohne Beschränkung darauf, Schaltungsunterabschnitte **3810(0) - 3810(N)**, eine Leistungs-Gate-Steuereinrichtung **3820** und eine Taktfrequenz-Steuereinrichtung **3830**.

[0268] Jeder der Schaltungsunterabschnitte **3810(0) - 3810(N)** beinhaltet einen beliebigen Satz von Komponenten, die in der PPU **200** auf jeder Granularitätsebene enthalten sind. In dieser Hinsicht kann jeder der Schaltungsunterabschnitte **3810(0) - 3810(N)**, ohne Beschränkung darauf, eine Sys-Pipe **230**, eine PPU-Partition **600**, eine PPU-Scheibe bzw. einen PPU-Slice **610**, eine SMC-Engine **700** oder jede technisch machbare Teilmenge davon enthalten.

[0269] Im Betrieb überwacht die Leistungs-Gate-Steuereinrichtung **3820** den Aktivitätsstatus der Schaltungsunterabschnitte **3810(0) - 3810(N)**. Falls die Leistungs-Gate-Steuereinrichtung **3820** ermittelt, dass sich ein bestimmter Schaltungsunterabschnitt, wie beispielsweise der Schaltungsunterabschnitt **3810(2)**, im Leerlaufzustand befindet, dann reduziert die Leistungs-Gate-Steuereinrichtung **3820** die Versorgungsspannung für den Schaltungsunterabschnitt **3810(2)** auf eine Spannung, die kleiner als eine Betriebsspannung ist, behält aber die im Speicher gespeicherten Daten bei. Alternativ kann die Leistungs-Gate-Steuereinrichtung **3820** die Leistung aus dem Schaltungsunterabschnitt **3810(2)** abziehen und dadurch den Schaltungsunterabschnitt **3810(2)** abschalten. Anschließend, falls der Schaltungsunterabschnitt **3810(2)** zur Ausführung bestimmter Aufgaben benötigt wird, erhöht die Leistungs-Gate-Steuereinrichtung **3820** die Versorgungsspannung des Schaltungsunterabschnitts **3810(2)** auf eine für den Betrieb geeignete Spannung.

[0270] Die Taktfrequenz-Steuereinrichtung **3830** überwacht die Leistungsaufnahme der Schaltungsunterabschnitte **3810(0) - 3810(N)**. Falls die Taktfrequenz-Steuereinrichtung **3830** ermittelt, dass ein bestimmter Schaltungsunterabschnitt, wie beispielsweise der Schaltungsunterabschnitt **3810(3)**, relativ zu anderen Schaltungsunterabschnitten **3810** mehr Leistung verbraucht, dann reduziert die Taktfrequenz-Steuereinrichtung **3830** die Frequenz von dem Schaltungsunterabschnitt **3810(3)** zugeordneten Taktsignalen. Infolgedessen wird die von dem Schaltungsunterabschnitt **3810(3)** verbrauchte Leistung reduziert. Darauf folgend erhöht dann, wenn die Taktfrequenz-Steuereinrichtung **3830** anschließend ermittelt, dass der Schaltungsunterabschnitt **3810(3)** relativ zu anderen Schaltungsunterabschnitten **3810** weniger Leistung verbraucht, die Taktfrequenz-Steuereinrichtung **3830** die Frequenz von dem Schaltungsunterabschnitt **3810(3)** zugeordneten Taktsignalen, wodurch die Leistung des Schaltungsunterabschnitts **3810(3)** erhöht wird.

[0271] Auf diese Weise reduzieren die Leistungs-Gate-Steuereinrichtung **3820** und die Taktfrequenz-Steuereinrichtung **3830** den Gesamtstromverbrauch der PPU **200** und verringern negative Auswirkungen einer PPU-Partition **600** auf eine andere PPU-Partition **600** aufgrund von Temperatureffekten.

[0272] Fig. 39 ist ein Ablaufdiagramm von Verfahrensschritten zur Verwaltung des Stromverbrauchs der PPU **200** von Fig. 2, gemäß verschiedenen Ausführungsformen. Obwohl die Verfahrensschritte in Verbindung mit den Systemen der Fig. 1-25 beschrieben werden, versteht sich für den Fachmann, dass jedes System, das dazu konfiguriert ist, die Verfahrensschritte in beliebiger Reihenfolge auszuführen, in den Anwendungsbereich der Erfindung fällt.

[0273] Wie gezeigt ist, beginnt ein Verfahren **3900** bei Schritt **3902**, in dem das Leistungs- und Taktfrequenz-Verwaltungssystem **3800** der PPU **200** den Aktivitätsstatus von VMs überwacht, die auf verschiedenen Schaltungsunterabschnitten **3810** der PPU **200** ausgeführt werden. In Schritt **3904** ermittelt das Leistungs- und Taktfrequenz-Verwaltungssystem **3800**, ob ein Schaltungsunterabschnitt **3810** im Leerlauf ist. Falls keine Schaltungsunterabschnitte **3810** im Leerlauf sind, schreitet das Verfahren zu Schritt **3908** fort. Falls jedoch eine oder mehrere Schaltungsunterabschnitte **3810** im Leerlauf sind, schreitet das Verfahren zu Schritt **3906** fort, in dem das Leistungs- und Taktfrequenz-Verwaltungssystem **3800** die Versorgungsspannung für die sich im Leerlauf befindenden Schaltungsunterabschnitte **3810** reduziert. Insbesondere reduziert die Leistungs-Gate-Steuereinrichtung **3820** innerhalb des Leistungs- und Taktfrequenz-Verwaltungssystems **3800** die Versorgungsspannung für den Schaltungsunterabschnitt **3810(2)** auf eine Spannung, die

kleiner als eine Betriebsspannung ist, behält aber die im Speicher gespeicherten Daten bei. Alternativ kann die Leistungs-Gate-Steuereinrichtung **3820** die Leistung aus dem Schaltungsunterabschnitt **3810(2)** abziehen und dadurch den Schaltungsunterabschnitt **3810(2)** abschalten.

[0274] In Schritt **3908** überwacht das Leistungs- und Taktfrequenz-Verwaltungssystem **3800** den Stromverbrauch für jede SMC-Engine **700** innerhalb der PPU. In Schritt **3910** ermittelt das Leistungs- und Taktfrequenz-Verwaltungssystem **3800**, ob eine oder mehrere SMC-Engines **700** relativ zu anderen SMC-Engines **700** übermäßig Leistung verbrauchen. Wenn keine SMC-Engine **700** übermäßig Leistung verbraucht, schreitet das Verfahren zu Schritt **3902** fort, um die Überwachung fortzusetzen. Falls jedoch eine oder mehrere SMC-Engines **700** übermäßig Leistung verbrauchen, schreitet das Verfahren zu Schritt **3912** fort, in dem die Taktfrequenz-Steuereinrichtung **3830** in dem Leistungs- und Taktfrequenz-Verwaltungssystem **3800** die Taktfrequenz für einen oder mehrere Schaltungsunterabschnitte **3810** reduziert, die SMC-Engines **700** zugeordnet sind, die übermäßig Leistung verbrauchen. Das Verfahren schreitet dann zu Schritt **3902** fort, um die Überwachung fortzusetzen.

[0275] Zusammenfassend beinhalten verschiedene Ausführungsformen eine Parallelverarbeitungseinheit (PPU), die in Partitionen unterteilt werden kann. Jede Partition ist dazu konfiguriert, Verarbeitungsaufgaben, die mehreren Verarbeitungskontexten zugeordnet sind, gleichzeitig auszuführen. Eine bestimmte Partition beinhaltet eine oder mehrere logische Gruppierungen oder „Scheiben“ bzw. „Slices“ von GPU-Ressourcen. Jeder Slice stellt ausreichend Rechen-, Grafik- und Speicherressourcen bereit, um den Betriebsablauf der PPU als Ganzes nachzuahmen. Ein Hypervisor, der auf einer CPU ausgeführt wird, führt im Auftrag eines Verwaltungsbenutzers verschiedene Techniken zur Partitionierung der PPU durch. Ein Gastbenutzer wird einer Partition zugewiesen und kann dann innerhalb dieser Partition isoliert von allen anderen Gastbenutzern, die anderen Partitionen zugewiesen sind, Verarbeitungsaufgaben durchführen.

[0276] Ein technologischer Vorteil der offenbarten Techniken gegenüber dem Stand der Technik besteht darin, dass mit den offenbarten Techniken eine PPU mehrere Verarbeitungskontexte gleichzeitig und funktional voneinander isoliert unterstützen kann. Demgemäß können mehrere CPU-Prozesse PPU-Ressourcen effizient über mehrere verschiedene Verarbeitungskontexte und ohne gegenseitige Beeinträchtigung nutzen. Ein weiterer technologischer Vorteil der offenbarten Techniken besteht darin, dass die PPU, weil die PPU unter Verwendung der offenbarten Techniken in isolierte Rechenumgebungen parti-

tioniert werden kann, eine robustere Form der Multitenancy bzw. Mandantenfähigkeit unterstützen kann, als dies bei bekannten Ansätzen der Fall ist, die sich zur Bereitstellung von Multitenancy-Funktionalität auf Verarbeitungsunterkontexte stützen. Demgemäß eignet sich eine PPU, wenn sie die offenbarten Techniken implementiert, besser für Cloud-basierte Implementierungen geeignet wird, bei denen verschiedene und potenziell konkurrierende Entitäten Zugriff auf verschiedene Partitionen innerhalb derselben PPU gewährt werden kann. Diese technologischen Vorteile repräsentieren einen oder mehrere technologische Fortschritte gegenüber dem Stand der Technik.

[0277] 1. Einige Ausführungsformen umfassen ein computer-implementiertes Verfahren, umfassend ein Partitionieren eines Satzes von Speicherressourcen, die in einem Prozessor enthalten sind, um eine Vielzahl von logischen Speicherpartitionen zu erzeugen, wobei jede logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, zumindest einer Maschine entspricht, die in einer Vielzahl von Maschinen enthalten ist, ein Empfangen einer Vielzahl von Speicherzugriffsanforderungen von der Vielzahl von Maschinen, und ein Übertragen jeder Speicherzugriffsanforderung, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist, an eine jeweils andere logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, wobei jede Speicherzugriffsanforderung von der jeweiligen logischen Speicherpartition unabhängig von allen anderen Speicherzugriffsanforderungen bedient wird, die in der Vielzahl von Speicherzugriffsanforderungen enthalten sind.

[0278] 2. Computer-implementiertes Verfahren nach Punkt 1, wobei das Partitionieren des Satzes von Speicherressourcen ein Aktivieren eines Satzes von logischen Grenzen umfasst, die dem Satz von Speicherressourcen auf der Grundlage eines Satzes von Bits zugeordnet sind, wobei jedes in dem Satz von Bits enthaltene Bit einer anderen logischen Grenze entspricht, die in dem Satz von logischen Grenzen enthalten ist.

[0279] 3. Computer-implementiertes Verfahren nach einem der Punkte 1 bis 2, wobei eine erste Speicherzugriffsanforderung, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist, an eine erste logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, auf der Grundlage eines Satzes von Bits übertragen wird, wobei der Satz von Speicherressourcen zumindest teilweise auf der Grundlage des Satzes von Bits partitioniert ist.

[0280] 4. Computer-implementiertes Verfahren nach einem der Punkte 1 bis 3, ferner umfassend ein Übersetzen einer ersten Adresse, die einer ersten Spei-

cherzugriffsanforderung zugeordnet ist, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist, basierend auf einem Satz von Bits, um eine zweite Adresse zu erzeugen, wobei die zweite Adresse einer Speicherstelle zugeordnet ist, die sich innerhalb einer ersten logischen Speicherpartition befindet, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, und der Satz von Speicherressourcen zumindest teilweise auf der Grundlage des Satzes von Bits partitioniert ist.

[0281] 5. Computer-implementiertes Verfahren nach einem der Punkte 1 bis 4, ferner umfassend ein Bestimmen eines ersten Teils des Satzes von Speicherressourcen für einen Zugriff durch eine autorisierte Entität, wobei der erste Teil des Satzes von Speicherressourcen nicht in der Vielzahl von logischen Speicherpartitionen enthalten ist und die autorisierte Entität den Satz von Speicherressourcen partitioniert, um die Vielzahl von logischen Speicherpartitionen zu erzeugen.

[0282] 6. Computer-implementiertes Verfahren nach einem der Punkte 1 bis 5, ferner umfassend ein Bestimmen eines ersten Fehlers, der einer ersten Speicherzugriffsanforderung zugeordnet ist, die einer Speicherstelle zugeordnet ist, die sich innerhalb einer ersten logischen Speicherpartition befindet, die in der Vielzahl der logischen Speicherpartitionen enthalten ist, ein Übersetzen eines virtuellen Fehleridentifizierers, der dem ersten Fehler entspricht, in einen globalen Fehleridentifizierer, der dem ersten Fehler entspricht; und ein Rücksetzen einer ersten Maschine auf der Grundlage des globalen Fehleridentifizierers, wobei die erste Maschine in der Vielzahl von Maschinen enthalten ist und dazu konfiguriert ist, auf die erste logische Speicherpartition zuzugreifen.

[0283] 7. Computer-implementiertes Verfahren nach einem der Punkte 1 bis 6, , ferner umfassend ein Übersetzen eines virtuellen Fehleridentifizierers, der in einem lokalen Adressraum enthalten ist, der einer ersten logischen Speicherpartition zugeordnet ist, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, um einen globalen Fehleridentifizierer zu erzeugen, der in einem globalen Adressraum enthalten ist, der dem Satz von Speicherressourcen zugeordnet ist, wobei der virtuelle Fehleridentifizierer von der ersten logischen Speicherpartition erzeugt wird, wenn eine erste Speicherzugriffsanforderung bedient wird, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist.

[0284] 8. Computer-implementiertes Verfahren nach einem der Punkte 1 bis 7, ferner umfassend ein Konfigurieren einer ersten logischen Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, dazu, einen ersten Speicherbereich zu enthalten, der Daten speichert, die einer ersten Maschine zugeordnet sind, die in der Vielzahl von Maschi-

nen enthalten ist; und ein Konfigurieren der ersten logischen Speicherpartition dazu, einen zweiten Speicherbereich zu enthalten, der Daten speichert, die einer zweiten Maschine zugeordnet sind, die in der Vielzahl von Maschinen enthalten ist.

[0285] 9. Computer-implementiertes Verfahren nach einem der Punkte 1 bis 8, wobei die erste Maschine einen ersten Satz von Verarbeitungsaufgaben, die einem ersten Verarbeitungskontext zugeordnet sind, unter Verwendung des ersten Speicherbereichs ausführt und die zweite Maschine einen zweiten Satz von Verarbeitungsaufgaben, die einem zweiten Verarbeitungskontext zugeordnet sind, unter Verwendung des zweiten Speicherbereichs ausführt.

[0286] 10. Computer-implementiertes Verfahren nach einem der Punkte 1 bis 10, ferner umfassend ein Konfigurieren einer ersten logischen Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, dazu, einen ersten Speicherbereich zu enthalten, der Daten speichert, die einem ersten Verarbeitungsunterkontext zugeordnet sind, der von einem ersten Verarbeitungskontext abgeleitet ist; und ein Konfigurieren der ersten logischen Speicherpartition dazu, einen zweiten Speicherbereich zu enthalten, der Daten speichert, die einem zweiten Verarbeitungsunterkontext zugeordnet sind, der ebenfalls aus dem ersten Verarbeitungskontext abgeleitet ist.

[0287] 11. Einige Ausführungsformen umfassen ein nichtflüchtiges computerlesbares Medium, das Programmweisungen speichert, die, wenn sie von einem Prozessor ausgeführt werden, den Prozessor veranlassen, die folgenden Schritte auszuführen: Partitionieren eines Satzes von Speicherressourcen, die in einem Prozessor enthalten sind, um eine Vielzahl von logischen Speicherpartitionen zu erzeugen, wobei jede logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, zumindest einer Maschine entspricht, die in einer Vielzahl von Maschinen enthalten ist; und Übertragen jeder Speicherzugriffsanforderung, die in einer Vielzahl von Speicherzugriffsanforderungen enthalten ist, die von der Vielzahl von Maschinen empfangen werden, an eine jeweils andere logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, wobei jede Speicherzugriffsanforderung von der jeweiligen logischen Speicherpartition unabhängig von allen anderen Speicherzugriffsanforderungen bedient wird, die in der Vielzahl von Speicherzugriffsanforderungen enthalten sind.

[0288] 12. Nichtflüchtiges computerlesbares Medium nach Punkt 11, wobei der Schritt des Partitionierens des Satzes von Speicherressourcen ein Aktivieren eines Satzes von logischen Grenzen umfasst, die dem Satz von Speicherressourcen auf der Grundlage eines Satzes von Bits zugeordnet sind, wobei jedes

in dem Satz von Bits enthaltene Bit einer anderen logischen Grenze entspricht, die in dem Satz von logischen Grenzen enthalten ist.

[0289] 13. Nichtflüchtiges computerlesbares Medium nach einem der Punkte 11 bis 12, wobei eine erste Speicherzugriffsanforderung, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist, an eine erste logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, auf der Grundlage eines Satzes von Bits übertragen wird, wobei der Satz von Speicherressourcen zumindest teilweise auf der Grundlage des Satzes von Bits partitioniert ist.

[0290] 14. Nichtflüchtiges computerlesbares Medium nach einem der Punkte 11 bis 13, ferner umfassend den Schritt des Bestimmens eines ersten Teils des Satzes von Speicherressourcen für einen Zugriff durch eine autorisierte Entität, wobei der erste Teil des Satzes von Speicherressourcen nicht in der Vielzahl von logischen Speicherpartitionen enthalten ist und die autorisierte Entität den Satz von Speicherressourcen partitioniert, um die Vielzahl von logischen Speicherpartitionen zu erzeugen.

[0291] 15. Nichtflüchtiges computerlesbares Medium nach einem der Punkte 11 bis 14, ferner umfassend die Schritte: Konfigurieren einer ersten logischen Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, dazu, einen ersten Speicherbereich zu enthalten, der Daten speichert, die einer ersten Maschine zugeordnet sind, die in der Vielzahl von Maschinen enthalten ist, und Konfigurieren der ersten logischen Speicherpartition dazu, einen zweiten Speicherbereich zu enthalten, der Daten speichert, die einer zweiten Maschine zugeordnet sind, die in der Vielzahl von Maschinen enthalten ist, wobei die erste Maschine einen ersten Satz von Verarbeitungsaufgaben, die einem ersten Verarbeitungskontext zugeordnet sind, unter Verwendung des ersten Speicherbereichs ausführt, und die zweite Maschine einen zweiten Satz von Verarbeitungsaufgaben, die einem zweiten Verarbeitungskontext zugeordnet sind, unter Verwendung des zweiten Speicherbereichs ausführt.

[0292] 16. Nichtflüchtiges computerlesbares Medium nach einem der Punkte 11 bis 15, ferner umfassend die Schritte: Konfigurieren einer ersten logischen Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, dazu, einen ersten Speicherbereich zu enthalten, der Daten speichert, die einem ersten Verarbeitungsunterkontext zugeordnet sind, der von einem ersten Verarbeitungskontext abgeleitet ist, und Konfigurieren der ersten logischen Speicherpartition dazu, einen zweiten Speicherbereich zu enthalten, der Daten speichert, die einem zweiten Verarbeitungsunterkontext zuge-

ordnet sind, der ebenfalls aus dem ersten Verarbeitungskontext abgeleitet ist.

[0293] 17. Nichtflüchtiges computerlesbares Medium nach einem der Punkte 11 bis 16, wobei die erste Maschine entsprechend dem ersten Verarbeitungskontext konfiguriert ist, um einen ersten Satz von Verarbeitungsaufgaben, die dem ersten Unterkontext zugeordnet sind, unter Verwendung des ersten Speicherbereichs auszuführen und um einen zweiten Satz von Verarbeitungsaufgaben, die dem zweiten Unterkontext zugeordnet sind, unter Verwendung des zweiten Speicherbereichs auszuführen.

[0294] 18. Nichtflüchtiges computerlesbares Medium nach einem der Punkte 11 bis 17, ferner umfassend die Schritte: Empfangen einer Partitionierungseingabe, die einer Hostcomputerumgebung zugeordnet ist, und Empfangen einer Konfigurationseingabe, die einer Hostcomputerumgebung zugeordnet ist, wobei der Satz von Speicherressourcen auf der Grundlage der Partitionierungseingabe partitioniert ist, und wobei eine erste logische Speicherpartition, die in der Vielzahl von logischen Speichern enthalten ist, auf der Grundlage der Konfigurationseingabe partitioniert ist.

[0295] 19. Nichtflüchtiges computerlesbares Medium nach einem der Punkte 11 bis 18, ferner umfassend die Schritte: Empfangen einer ersten Eingabe, die einer ersten Rechenumgebung zugeordnet ist, wobei ein erster Satz von Operationen auf dem Prozessor innerhalb der ersten Rechenumgebung auf der Grundlage eines ersten Satzes von Berechtigungen ausgeführt wird, und Empfangen einer zweiten Eingabe, die einer zweiten Rechenumgebung zugeordnet ist, wobei ein zweiter Satz von Operationen auf dem Prozessor innerhalb der zweiten Rechenumgebung auf der Grundlage eines zweiten Satzes von Berechtigungen ausgeführt wird, und der erste Satz von Berechtigungen eine größere Anzahl von Berechtigungen als der zweite Satz von Berechtigungen enthält, wobei der Satz von Speicherressourcen auf der Grundlage der Partitionierungseingabe partitioniert ist, und wobei eine erste logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, auf der Grundlage der zweiten Eingabe konfiguriert ist.

[0296] 20. Einige Ausführungsformen beinhalten ein System, umfassend einen Speicher, der eine Software-Anwendung speichert; und einen Prozessor, der bei der Ausführung der Softwareanwendung dazu konfiguriert ist, die Schritte eines Partitionierens eines Satzes von Speicherressourcen, die in einem Prozessor enthalten sind, um eine Vielzahl von logischen Speicherpartitionen zu erzeugen, wobei jede logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, zumindest einer Maschine entspricht, die in einer Vielzahl

von Maschinen enthalten ist, eines Empfangens einer Vielzahl von Speicherzugriffsanforderungen von der Vielzahl von Maschinen und eines Übertragens jeder Speicherzugriffsanforderung, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist, an eine jeweils andere logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, durchzuführen, wobei jede Speicherzugriffsanforderung von der jeweiligen logischen Speicherpartition unabhängig von allen anderen Speicherzugriffsanforderungen bedient wird, die in der Vielzahl von Speicherzugriffsanforderungen enthalten sind.

[0297] Jede und alle Kombinationen der in einem beliebigen der Ansprüche genannten Anspruchselemente und/oder der in dieser Anmeldung beschriebenen Elemente fallen in beliebiger Weise in die angedachte Reichweite der vorliegenden Ausführungsformen und den angedachten Schutz.

[0298] Die Beschreibungen der verschiedenen Ausführungsformen wurden zu Zwecken der Veranschaulichung präsentiert, sollen aber weder erschöpfend noch auf die offenbarten Ausführungsformen beschränkt sein. Viele Modifikationen und Variationen werden sich für den Fachmann ergeben, ohne den Rahmen und die Idee der beschriebenen Ausführungsformen zu verlassen.

[0299] Aspekte der vorliegenden Ausführungsformen können als ein System, ein Verfahren oder ein Computerprogrammprodukt verkörpert sein. Dementsprechend können Aspekte der Erfindung die Form einer Ausführungsform vollständig in Hardware, einer Ausführungsform vollständig in Software (einschließlich Firmware, residenter Software, Mikrocode usw.) oder einer Ausführungsform, die Software- und Hardware-Aspekte kombiniert, annehmen, die hierin alle allgemein als ein „Modul“, „System“ oder „Computer“ bezeichnet sein können. Darüber hinaus kann jede Hardware- und/oder Software-Technik, jeder Prozess, jede Funktion, jede Komponente, jede Maschine bzw. Engine, jedes Modul oder jedes System, das bzw. die in der vorliegenden Offenlegung beschrieben werden, als eine Schaltung oder ein Satz von Schaltungen implementiert sein. Ferner können Aspekte der Erfindung die Form eines Computerprogrammprodukts annehmen, das in einem oder mehreren computerlesbaren Medien mit darauf enthaltenem computerlesbarem Programmcode verkörpert ist.

[0300] Eine beliebige Kombination von einem oder mehreren computerlesbaren Medium bzw. Medien kann genutzt werden. Das computerlesbare Medium kann ein computerlesbares Signalmedium oder ein computerlesbares Speichermedium sein. Ein computerlesbares Speichermedium kann zum Beispiel, ohne darauf beschränkt zu sein, ein elektronisches, magnetisches, optisches oder elektromagnetisches Sys-

tem, ein Infrarotsystem oder ein Halbleitersystem, ein entsprechendes Gerät oder eine entsprechende Vorrichtung oder jede beliebige geeignete Kombination der vorgenannten sein. Spezifischere Beispiele (eine nicht erschöpfende Liste) der computerlesbaren Speichermedien würden beinhalten: eine elektrische Verbindung mit einer oder mehreren Leiterdrähten, eine tragbare Computerdiskette, eine Festplatte, ein Speicher mit wahlfreiem Zugriff (RAM), ein Festwertspeicher (ROM), ein löschbarer programmierbarer Festwertspeicher (EPROM oder Flash-Speicher), eine optische Faser, ein tragbarer Compact-Disc-Festwertspeicher (CD-ROM), eine optische Speichervorrichtung, eine magnetische Speichervorrichtung oder jede beliebige geeignete Kombination der vorgenannten. Im Kontext dieses Dokuments kann ein computerlesbares Speichermedium jedes beliebige dinghafte Medium sein, das ein Programm zur Verwendung durch oder in Verbindung mit einem Anweisungsausführungs-System, einer -Vorrichtung oder einem -Gerät enthalten oder speichern kann.

[0301] Aspekte der Erfindung sind vorstehend unter Bezugnahme auf Ablaufdiagramm-Darstellungen und/oder Blockdiagramme von Verfahren, Vorrichtungen (Systemen) und Computerprogrammprodukten in Übereinstimmung mit Ausführungsformen der Erfindung beschrieben. Es versteht sich, dass jeder Block der Ablaufdiagramm-Darstellungen und/oder Blockdiagramme, und ebenso Kombinationen von Blöcken in den Ablaufdiagramm-Darstellungen und/oder Blockdiagrammen, durch Computerprogramm-anweisungen implementiert sein kann. Diese Computerprogrammanweisungen können einem Prozessor eines Universalrechners, eines Spezialrechners oder einer anderen programmierbaren Datenverarbeitungsvorrichtung zur Herstellung einer Maschine zur Verfügung gestellt werden. Wenn die Anweisungen über den Prozessor des Computers oder einer anderen programmierbaren Datenverarbeitungsvorrichtung ausgeführt werden, ermöglichen sie die Implementierung der in dem Ablaufdiagramm und/oder in dem/den Blockdiagrammblock/Blockdiagrammböcken angegebenen Funktionen/ Aktionen. Solche Prozessoren können, ohne darauf beschränkt zu sein, Universalprozessoren, Spezialprozessoren, anwendungsspezifische Prozessoren oder feldprogrammierbare Gate-Arrays sein.

[0302] Die Ablaufdiagramme und die Blockdiagramme in den Figuren veranschaulichen die Architektur, die Funktionalität und die Funktionsweise möglicher Implementierungen von Systemen, Verfahren und Computerprogramm-produkten gemäß verschiedenen Ausführungsformen der Erfindung. In dieser Hinsicht kann jeder Block in den Ablaufdiagrammen oder den Blockdiagrammen ein Modul, ein Segment oder einen Abschnitt von Code repräsentieren, der eine oder mehrere ausführbare Anweisungen zur Im-

plementierung der spezifizierten logischen Funktion (en) umfasst. Es wird darüber hinaus angemerkt, dass in einigen alternativen Implementierungen die in dem Block angegebenen Funktionen außerhalb der in den Figuren vermerkten Reihenfolge auftreten können. Beispielsweise können zwei nacheinander dargestellte Blöcke tatsächlich im Wesentlichen gleichzeitig ausgeführt werden, oder können die Blöcke manchmal in umgekehrter Reihenfolge ausgeführt werden, abhängig von der involvierten Funktionalität. Es wird darüber hinaus angemerkt, dass jeder Block der Blockdiagramme und/oder der Ablaufdiagrammdarstellung sowie Kombinationen von Blöcken in den Blockdiagrammen und/oder der Ablaufdiagrammdarstellung durch auf Spezialzweck-Hardware basierende Systeme, die die spezifizierten Funktionen oder Aktionen ausführen, oder Kombinationen von Spezialzweck-Hardware und Computerbefehlen implementiert werden kann.

[0303] Während sich das Vorstehende auf Ausführungsformen der Erfindung bezieht, können andere und weitere Ausführungsformen der Erfindung ausgestaltet werden, ohne vom grundsätzlichen Umfang der Erfindung abzuweichen, so dass der Umfang derselben durch die folgenden Ansprüche bestimmt wird.

Patentansprüche

1. Computer-implementiertes Verfahren, umfassend:

Partitionieren eines Satzes von Speicherressourcen, die in einem Prozessor enthalten sind, um eine Vielzahl von logischen Speicherpartitionen zu erzeugen, wobei jede logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, zumindest einer Maschine entspricht, die in einer Vielzahl von Maschinen enthalten ist;

Empfangen einer Vielzahl von Speicherzugriffsanforderungen von der Vielzahl von Maschinen; und Übertragen jeder Speicherzugriffsanforderung, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist, an eine jeweils andere logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, wobei jede Speicherzugriffsanforderung von der jeweiligen logischen Speicherpartition unabhängig von allen anderen Speicherzugriffsanforderungen bedient wird, die in der Vielzahl von Speicherzugriffsanforderungen enthalten sind.

2. Computer-implementiertes Verfahren nach Anspruch 1, wobei das Partitionieren des Satzes von Speicherressourcen ein Aktivieren eines Satzes von logischen Grenzen umfasst, die dem Satz von Speicherressourcen auf der Grundlage eines Satzes von Bits zugeordnet sind, wobei jedes in dem Satz von Bits enthaltene Bit einer anderen logischen Grenze entspricht, die in dem Satz von logischen Grenzen enthalten ist.

3. Computer-implementiertes Verfahren nach Anspruch 1 oder 2, wobei eine erste Speicherzugriffsanforderung, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist, an eine erste logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, auf der Grundlage eines Satzes von Bits übertragen wird, wobei der Satz von Speicherressourcen zumindest teilweise auf der Grundlage des Satzes von Bits partitioniert ist.

4. Computer-implementiertes Verfahren nach einem der vorangehenden Ansprüche, ferner umfassend ein Übersetzen einer ersten Adresse, die einer ersten Speicherzugriffsanforderung zugeordnet ist, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist, basierend auf einem Satz von Bits, um eine zweite Adresse zu erzeugen, wobei die zweite Adresse einer Speicherstelle zugeordnet ist, die sich innerhalb einer ersten logischen Speicherpartition befindet, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, und der Satz von Speicherressourcen zumindest teilweise auf der Grundlage des Satzes von Bits partitioniert ist.

5. Computer-implementiertes Verfahren nach einem der vorangehenden Ansprüche, ferner umfassend ein Bestimmen eines ersten Teils des Satzes von Speicherressourcen für einen Zugriff durch eine autorisierte Entität, wobei der erste Teil des Satzes von Speicherressourcen nicht in der Vielzahl von logischen Speicherpartitionen enthalten ist und die autorisierte Entität den Satz von Speicherressourcen partitioniert, um die Vielzahl von logischen Speicherpartitionen zu erzeugen.

6. Computer-implementiertes Verfahren nach einem der vorangehenden Ansprüche, ferner umfassend:

Bestimmen eines ersten Fehlers, der einer ersten Speicherzugriffsanforderung zugeordnet ist, die einer Speicherstelle zugeordnet ist, die sich innerhalb einer ersten logischen Speicherpartition befindet, die in der Vielzahl der logischen Speicherpartitionen enthalten ist;

Übersetzen eines virtuellen Fehleridentifizierers, der dem ersten Fehler entspricht, in einen globalen Fehleridentifizierer, der dem ersten Fehler entspricht; und Rücksetzen einer ersten Maschine auf der Grundlage des globalen Fehleridentifizierers, wobei die erste Maschine in der Vielzahl von Maschinen enthalten ist und dazu konfiguriert ist, auf die erste logische Speicherpartition zuzugreifen.

7. Computer-implementiertes Verfahren nach einem der vorangehenden Ansprüche, ferner umfassend ein Übersetzen eines virtuellen Fehleridentifizierers, der in einem lokalen Adressraum enthalten ist, der einer ersten logischen Speicherpartition zugeordnet ist, die in der Vielzahl von logischen Spei-

cherpartitionen enthalten ist, um einen globalen Fehleridentifizierer zu erzeugen, der in einem globalen Adressraum enthalten ist, der dem Satz von Speicherressourcen zugeordnet ist, wobei der virtuelle Fehleridentifizierer von der ersten logischen Speicherpartition erzeugt wird, wenn eine erste Speicherzugriffsanforderung bedient wird, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist.

8. Computer-implementiertes Verfahren nach einem der vorangehenden Ansprüche, ferner umfassend:

Konfigurieren einer ersten logischen Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, dazu, einen ersten Speicherbereich zu enthalten, der Daten speichert, die einer ersten Maschine zugeordnet sind, die in der Vielzahl von Maschinen enthalten ist; und

Konfigurieren der ersten logischen Speicherpartition dazu, einen zweiten Speicherbereich zu enthalten, der Daten speichert, die einer zweiten Maschine zugeordnet sind, die in der Vielzahl von Maschinen enthalten ist.

9. Computer-implementiertes Verfahren nach Anspruch 8, wobei die erste Maschine einen ersten Satz von Verarbeitungsaufgaben, die einem ersten Verarbeitungskontext zugeordnet sind, unter Verwendung des ersten Speicherbereichs ausführt und die zweite Maschine einen zweiten Satz von Verarbeitungsaufgaben, die einem zweiten Verarbeitungskontext zugeordnet sind, unter Verwendung des zweiten Speicherbereichs ausführt.

10. Computer-implementiertes Verfahren nach einem der vorangehenden Ansprüche, ferner umfassend:

Konfigurieren einer ersten logischen Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, dazu, einen ersten Speicherbereich zu enthalten, der Daten speichert, die einem ersten Verarbeitungsunterkontext zugeordnet sind, der von einem ersten Verarbeitungskontext abgeleitet ist; und Konfigurieren der ersten logischen Speicherpartition dazu, einen zweiten Speicherbereich zu enthalten, der Daten speichert, die einem zweiten Verarbeitungsunterkontext zugeordnet sind, der ebenfalls aus dem ersten Verarbeitungskontext abgeleitet ist.

11. Nichtflüchtiges computerlesbares Medium, das Programmanweisungen speichert, die, wenn sie von einem Prozessor ausgeführt werden, den Prozessor veranlassen, die folgenden Schritte auszuführen: Partitionieren eines Satzes von Speicherressourcen, die in einem Prozessor enthalten sind, um eine Vielzahl von logischen Speicherpartitionen zu erzeugen, wobei jede logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, zumindest einer Maschine entspricht, die in einer Vielzahl von Maschinen enthalten ist; und

Übertragen jeder Speicherzugriffsanforderung, die in einer Vielzahl von Speicherzugriffsanforderungen enthalten ist, die von der Vielzahl von Maschinen empfangen werden, an eine jeweils andere logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, wobei jede Speicherzugriffsanforderung von der jeweiligen logischen Speicherpartition unabhängig von allen anderen Speicherzugriffsanforderungen bedient wird, die in der Vielzahl von Speicherzugriffsanforderungen enthalten sind.

12. Nichtflüchtiges computerlesbares Medium nach Anspruch 11, wobei der Schritt des Partitionierens des Satzes von Speicherressourcen ein Aktivieren eines Satzes von logischen Grenzen umfasst, die dem Satz von Speicherressourcen auf der Grundlage eines Satzes von Bits zugeordnet sind, wobei jedes in dem Satz von Bits enthaltene Bit einer anderen logischen Grenze entspricht, die in dem Satz von logischen Grenzen enthalten ist.

13. Nichtflüchtiges computerlesbares Medium nach Anspruch 11 oder 12, wobei eine erste Speicherzugriffsanforderung, die in der Vielzahl von Speicherzugriffsanforderungen enthalten ist, an eine erste logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, auf der Grundlage eines Satzes von Bits übertragen wird, wobei der Satz von Speicherressourcen zumindest teilweise auf der Grundlage des Satzes von Bits partitioniert ist.

14. Nichtflüchtiges computerlesbares Medium nach einem der Ansprüche 11 bis 13, ferner umfassend den Schritt des Bestimmens eines ersten Teils des Satzes von Speicherressourcen für einen Zugriff durch eine autorisierte Entität, wobei der erste Teil des Satzes von Speicherressourcen nicht in der Vielzahl von logischen Speicherpartitionen enthalten ist und die autorisierte Entität den Satz von Speicherressourcen partitioniert, um die Vielzahl von logischen Speicherpartitionen zu erzeugen.

15. Nichtflüchtiges computerlesbares Medium nach einem der Ansprüche 11 bis 14, ferner umfassend die Schritte:

Konfigurieren einer ersten logischen Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, dazu, einen ersten Speicherbereich zu enthalten, der Daten speichert, die einer ersten Maschine zugeordnet sind, die in der Vielzahl von Maschinen enthalten ist; und

Konfigurieren der ersten logischen Speicherpartition dazu, einen zweiten Speicherbereich zu enthalten, der Daten speichert, die einer zweiten Maschine zugeordnet sind, die in der Vielzahl von Maschinen enthalten ist,

wobei die erste Maschine einen ersten Satz von Verarbeitungsaufgaben, die einem ersten Verarbei-

tungskontext zugeordnet sind, unter Verwendung des ersten Speicherbereichs ausführt, und die zweite Maschine einen zweiten Satz von Verarbeitungsaufgaben, die einem zweiten Verarbeitungskontext zugeordnet sind, unter Verwendung des zweiten Speicherbereichs ausführt.

16. Nichtflüchtiges computerlesbares Medium nach einem der Ansprüche 11 bis 15, ferner umfassend die Schritte:

Konfigurieren einer ersten logischen Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, dazu, einen ersten Speicherbereich zu enthalten, der Daten speichert, die einem ersten Verarbeitungsunterkontext zugeordnet sind, der von einem ersten Verarbeitungskontext abgeleitet ist; und Konfigurieren der ersten logischen Speicherpartition dazu, einen zweiten Speicherbereich zu enthalten, der Daten speichert, die einem zweiten Verarbeitungsunterkontext zugeordnet sind, der ebenfalls aus dem ersten Verarbeitungskontext abgeleitet ist.

17. Nichtflüchtiges computerlesbares Medium nach Anspruch 16, wobei die erste Maschine entsprechend dem ersten Verarbeitungskontext konfiguriert ist, um einen ersten Satz von Verarbeitungsaufgaben, die dem ersten Unterkontext zugeordnet sind, unter Verwendung des ersten Speicherbereichs auszuführen und um einen zweiten Satz von Verarbeitungsaufgaben, die dem zweiten Unterkontext zugeordnet sind, unter Verwendung des zweiten Speicherbereichs auszuführen.

18. Nichtflüchtiges computerlesbares Medium nach einem der Ansprüche 11 bis 17, ferner umfassend die Schritte:

Empfangen einer Partitionierungseingabe, die einer Hostcomputerumgebung zugeordnet ist; und Empfangen einer Konfigurationseingabe, die einer Hostcomputerumgebung zugeordnet ist, wobei der Satz von Speicherressourcen auf der Grundlage der Partitionierungseingabe partitioniert ist, und wobei eine erste logische Speicherpartition, die in der Vielzahl von logischen Speichern enthalten ist, auf der Grundlage der Konfigurationseingabe partitioniert ist.

19. Nichtflüchtiges computerlesbares Medium nach einem der Ansprüche 11 bis 18, ferner umfassend die Schritte:

Empfangen einer ersten Eingabe, die einer ersten Rechenumgebung zugeordnet ist, wobei ein erster Satz von Operationen auf dem Prozessor innerhalb der ersten Rechenumgebung auf der Grundlage eines ersten Satzes von Berechtigungen ausgeführt wird; und Empfangen einer zweiten Eingabe, die einer zweiten Rechenumgebung zugeordnet ist, wobei ein zweiter Satz von Operationen auf dem Prozessor innerhalb

der zweiten Rechenumgebung auf der Grundlage eines zweiten Satzes von Berechtigungen ausgeführt wird, und der erste Satz von Berechtigungen eine größere Anzahl von Berechtigungen als der zweite Satz von Berechtigungen enthält,

wobei der Satz von Speicherressourcen auf der Grundlage der Partitionierungseingabe partitioniert ist, und

wobei eine erste logische Speicherpartition, die in der Vielzahl von logischen Speicherpartitionen enthalten ist, auf der Grundlage der zweiten Eingabe konfiguriert ist.

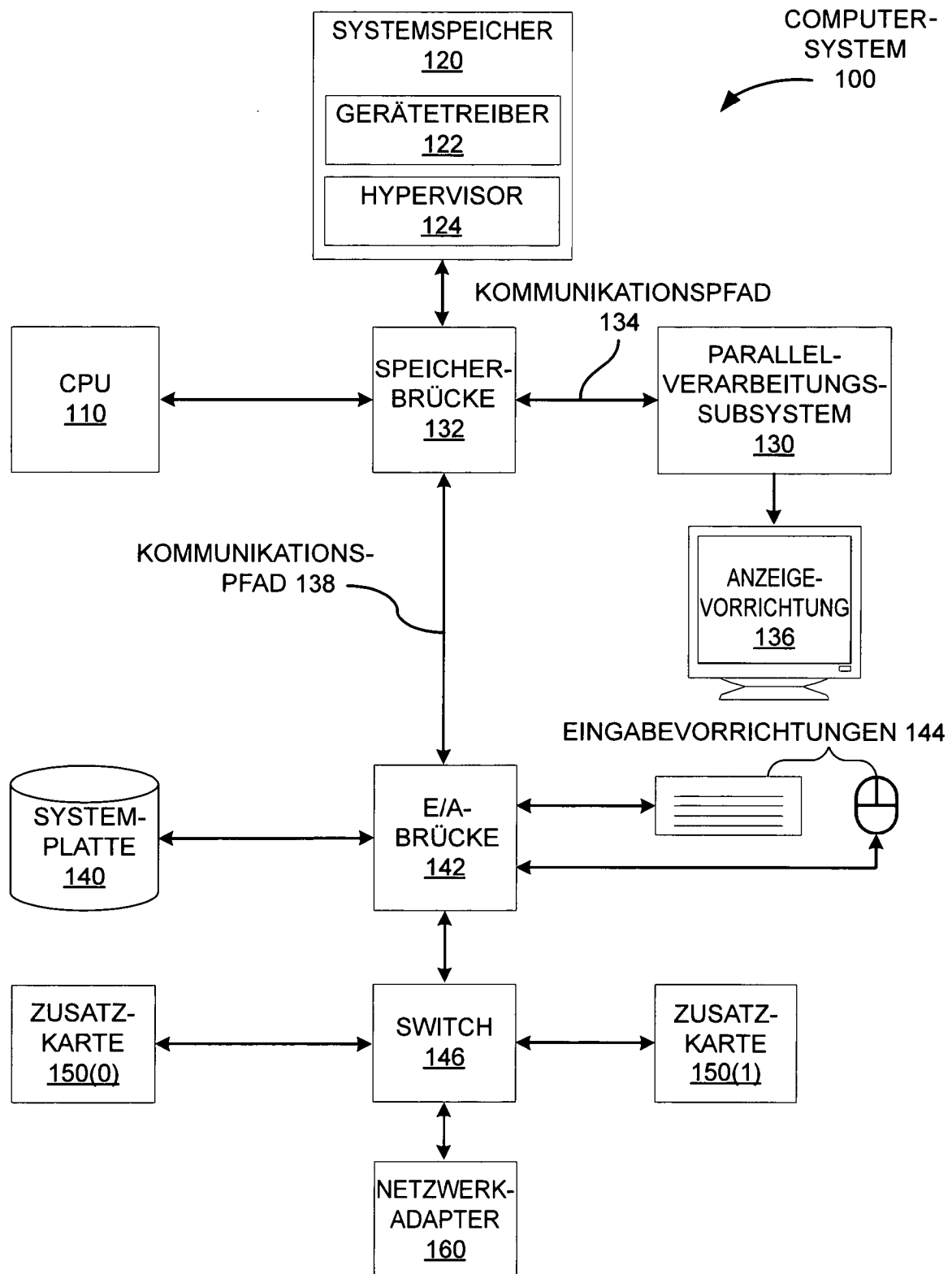
20. System, umfassend:

einen Speicher, der eine Software-Anwendung speichert; und

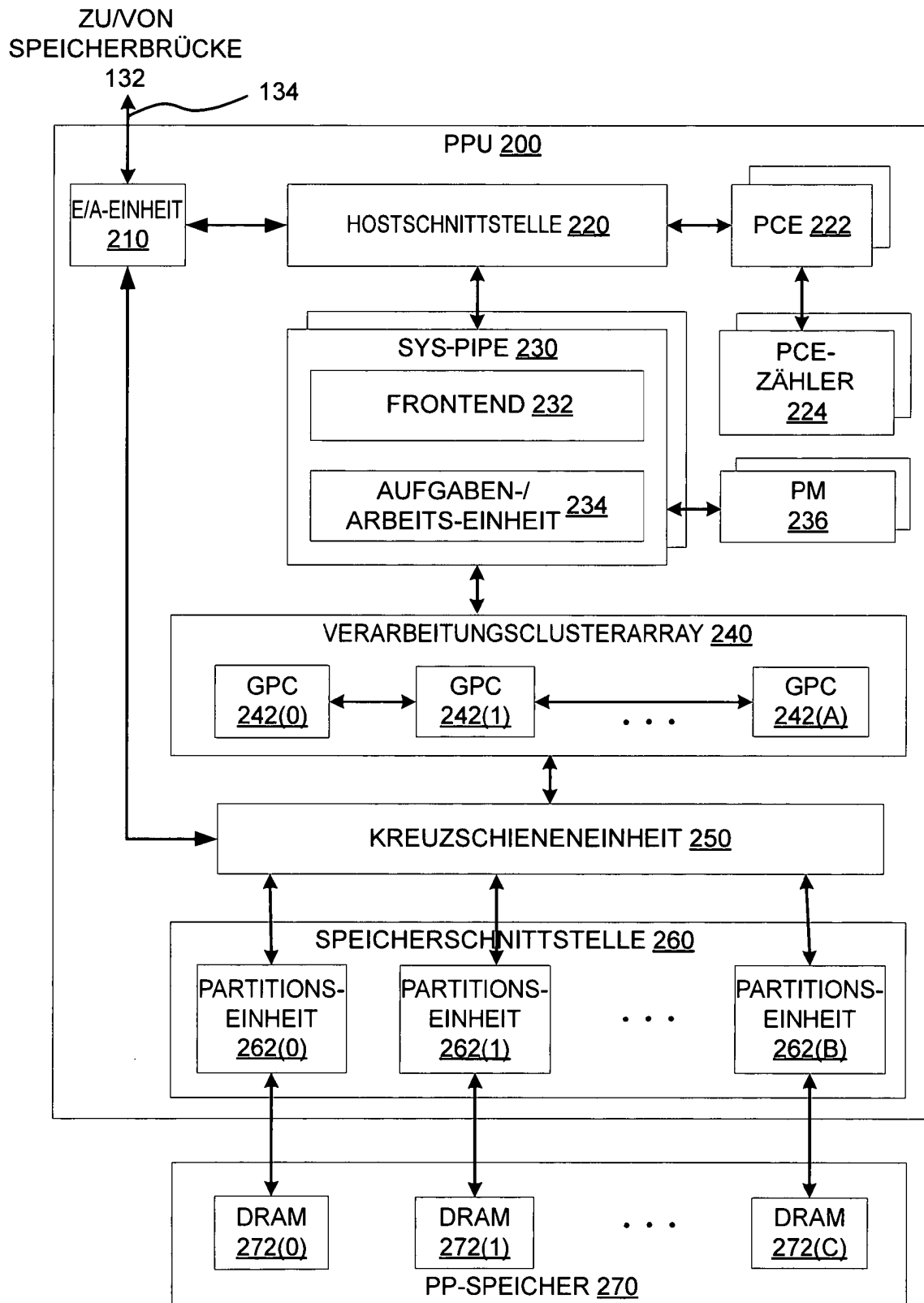
einen Prozessor, der bei der Ausführung der Softwareanwendung dazu konfiguriert ist, die Schritte eines Verfahrens nach einem der Ansprüche 1 bis 10 auszuführen.

Es folgen 43 Seiten Zeichnungen

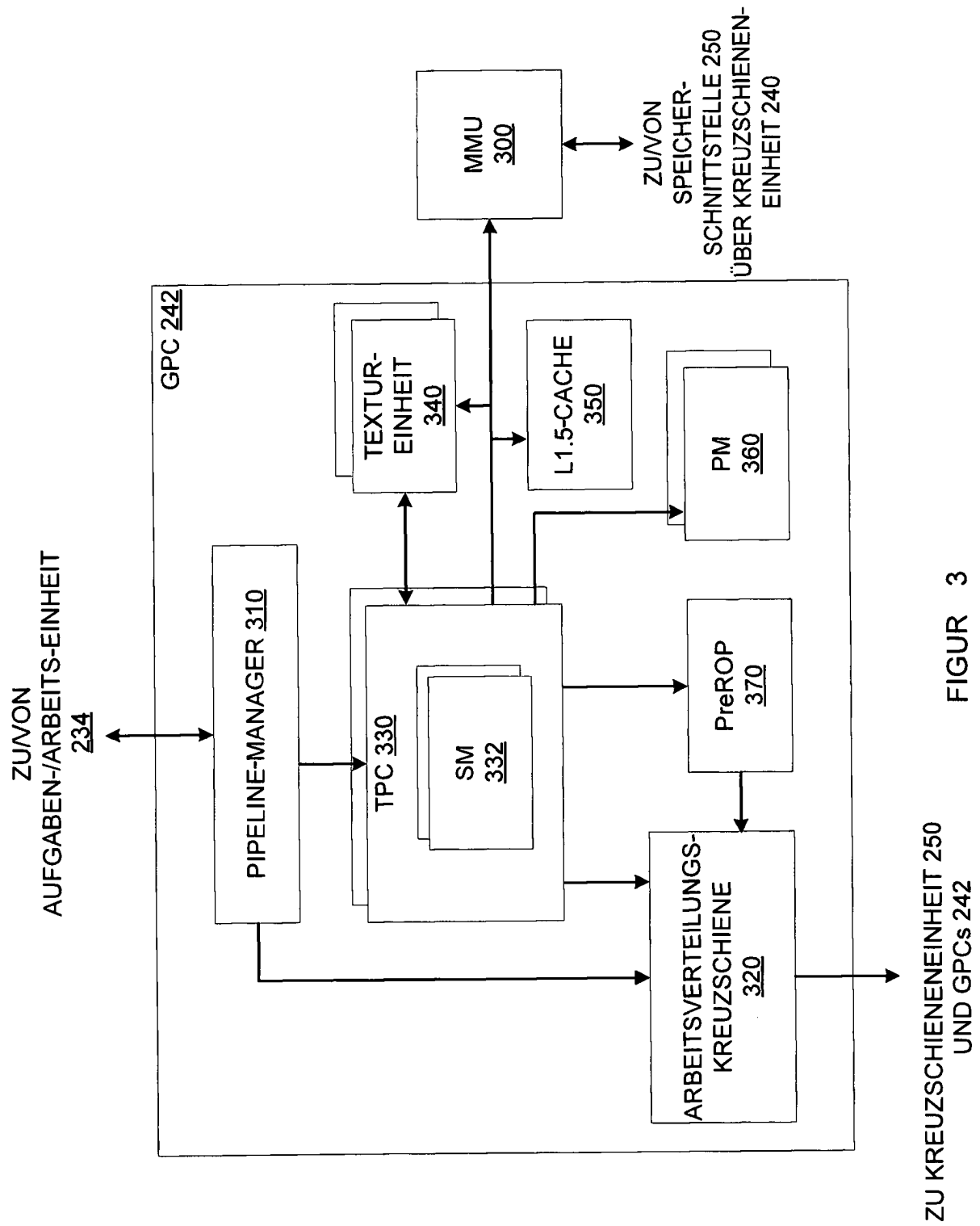
Anhängende Zeichnungen



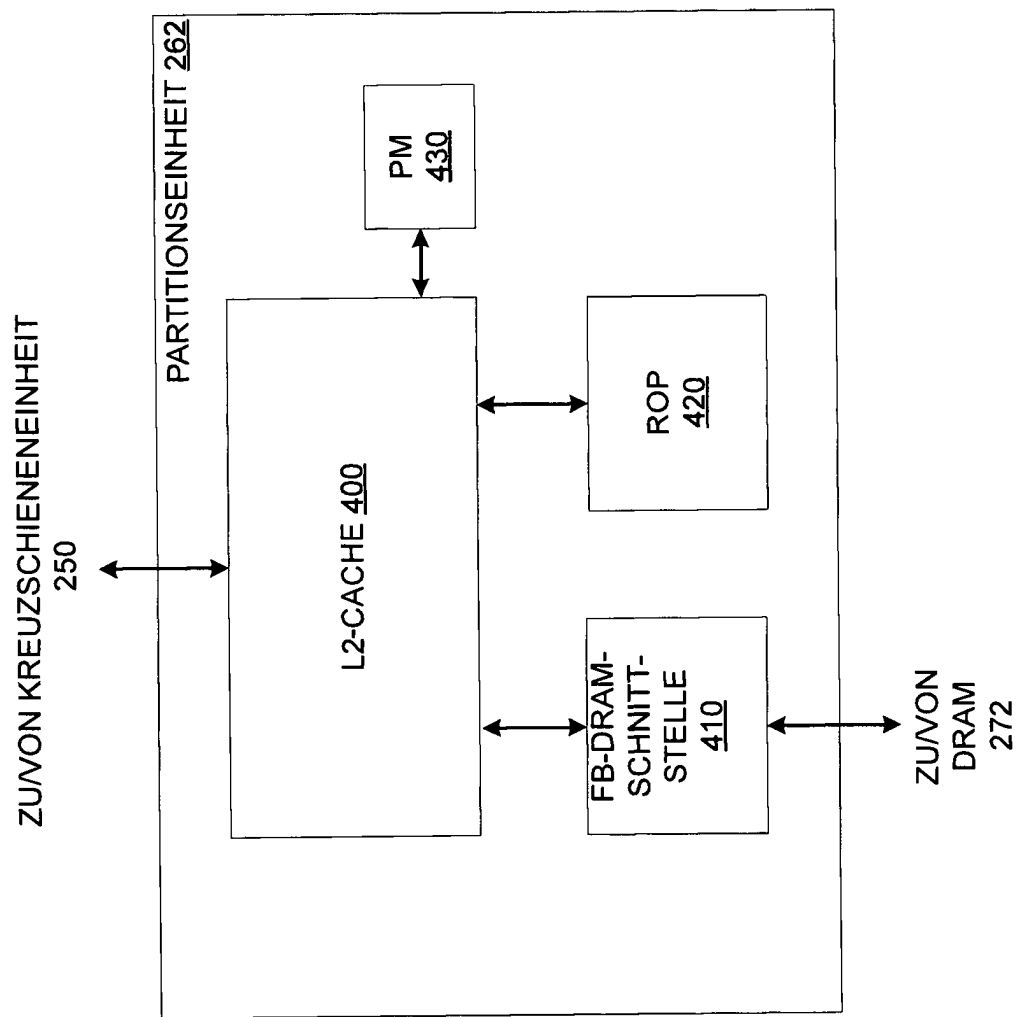
FIGUR 1



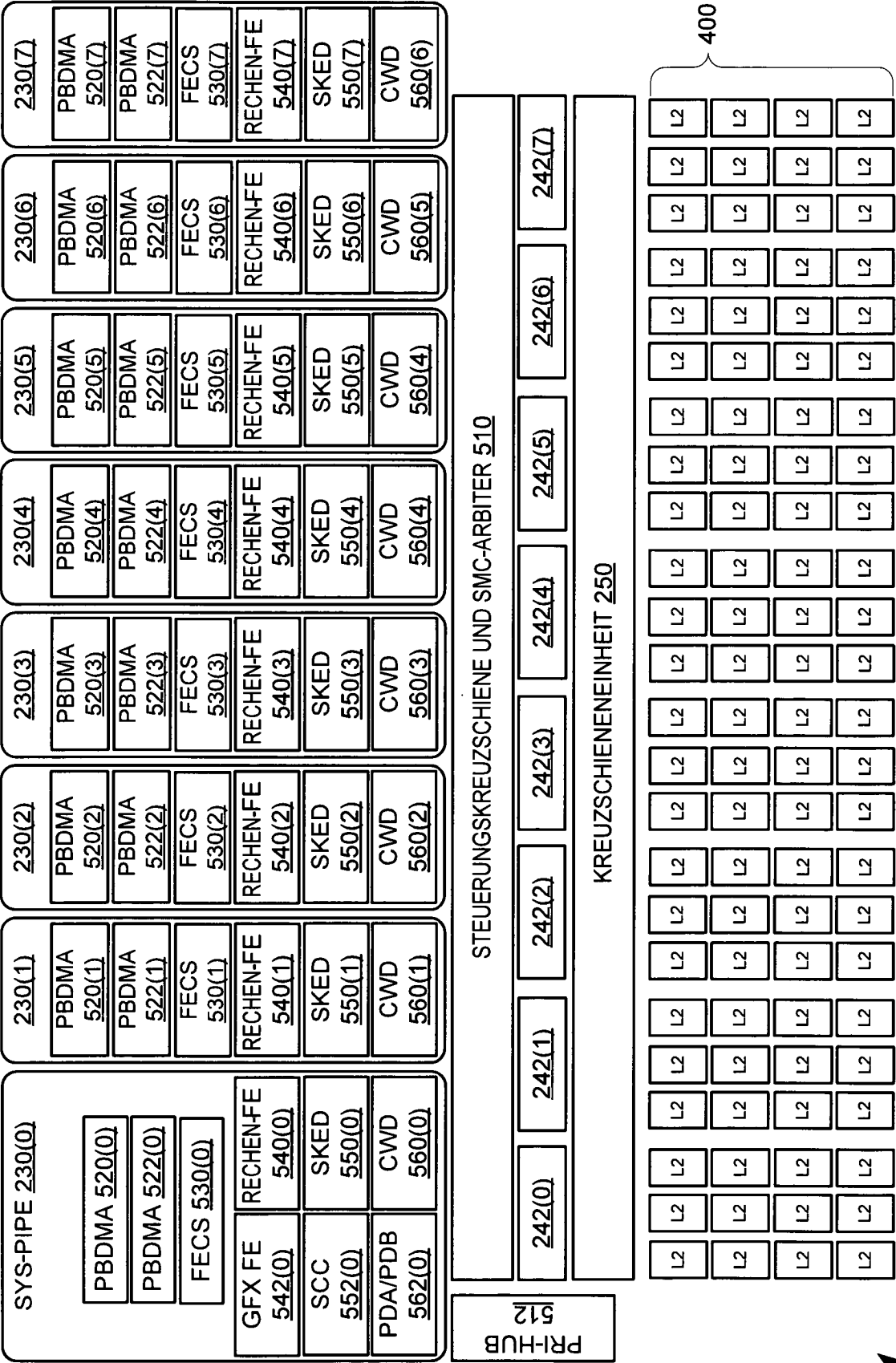
FIGUR 2



FIGUR 3



FIGUR 4



FIGUR 5

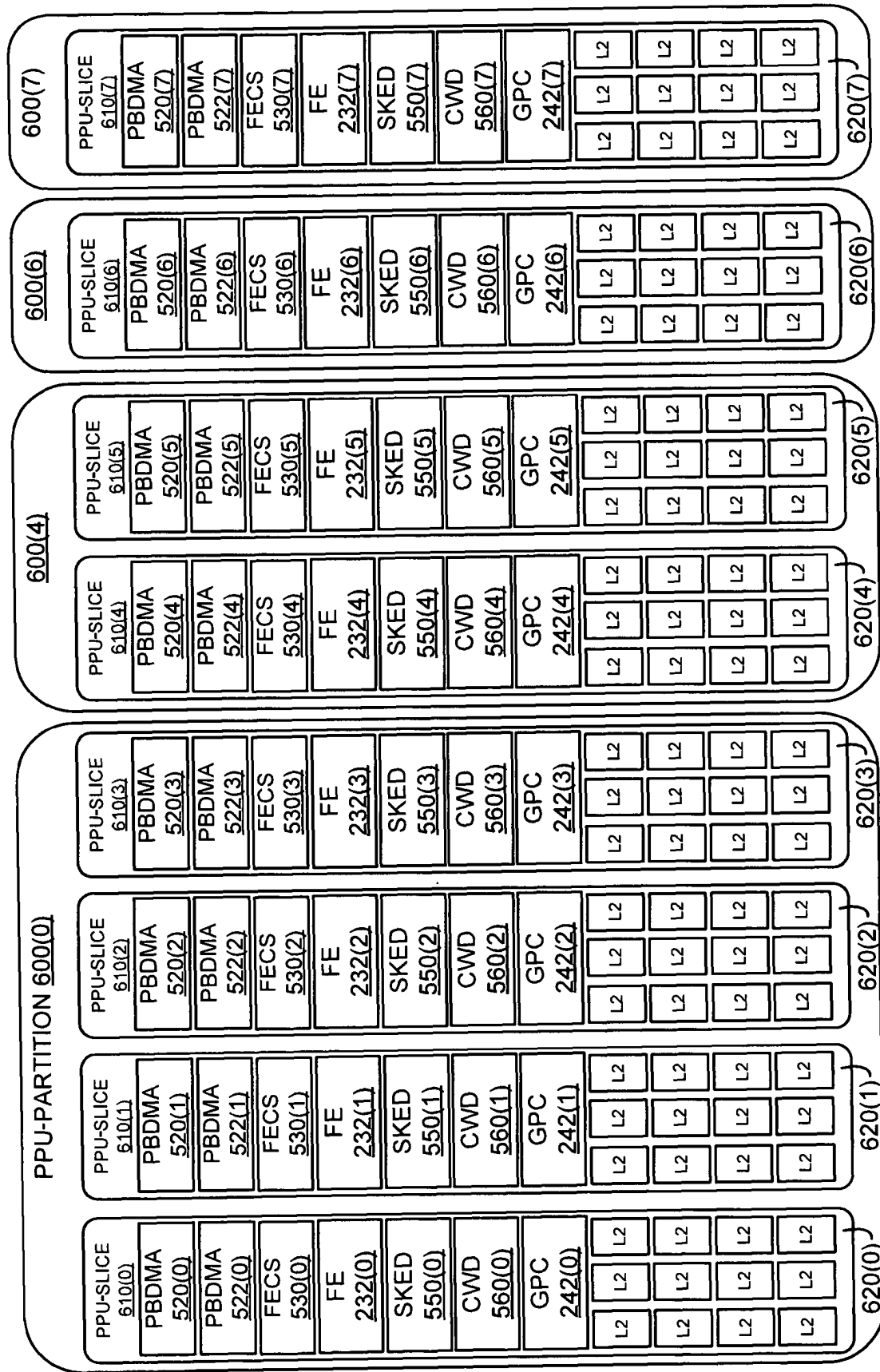
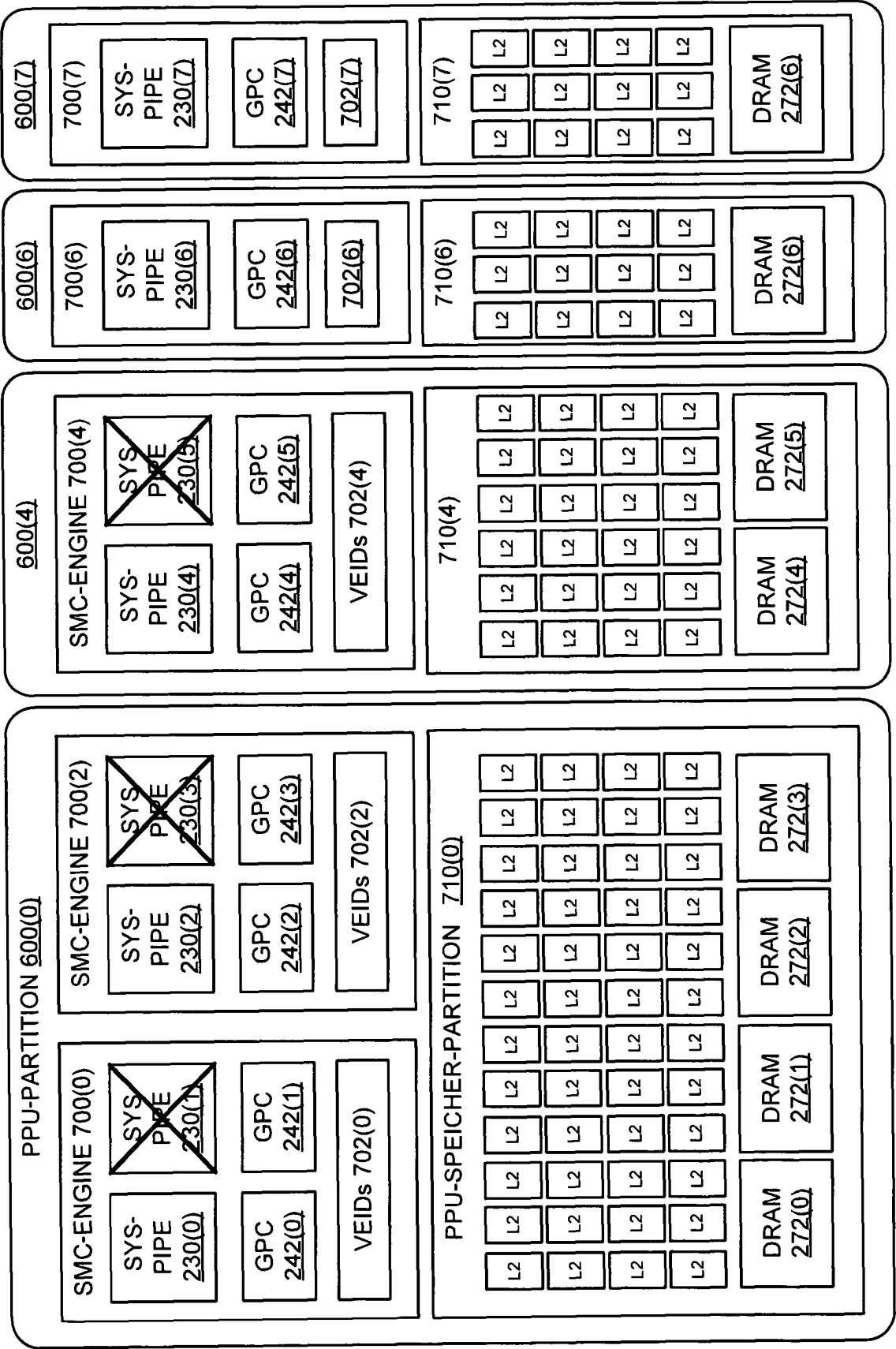
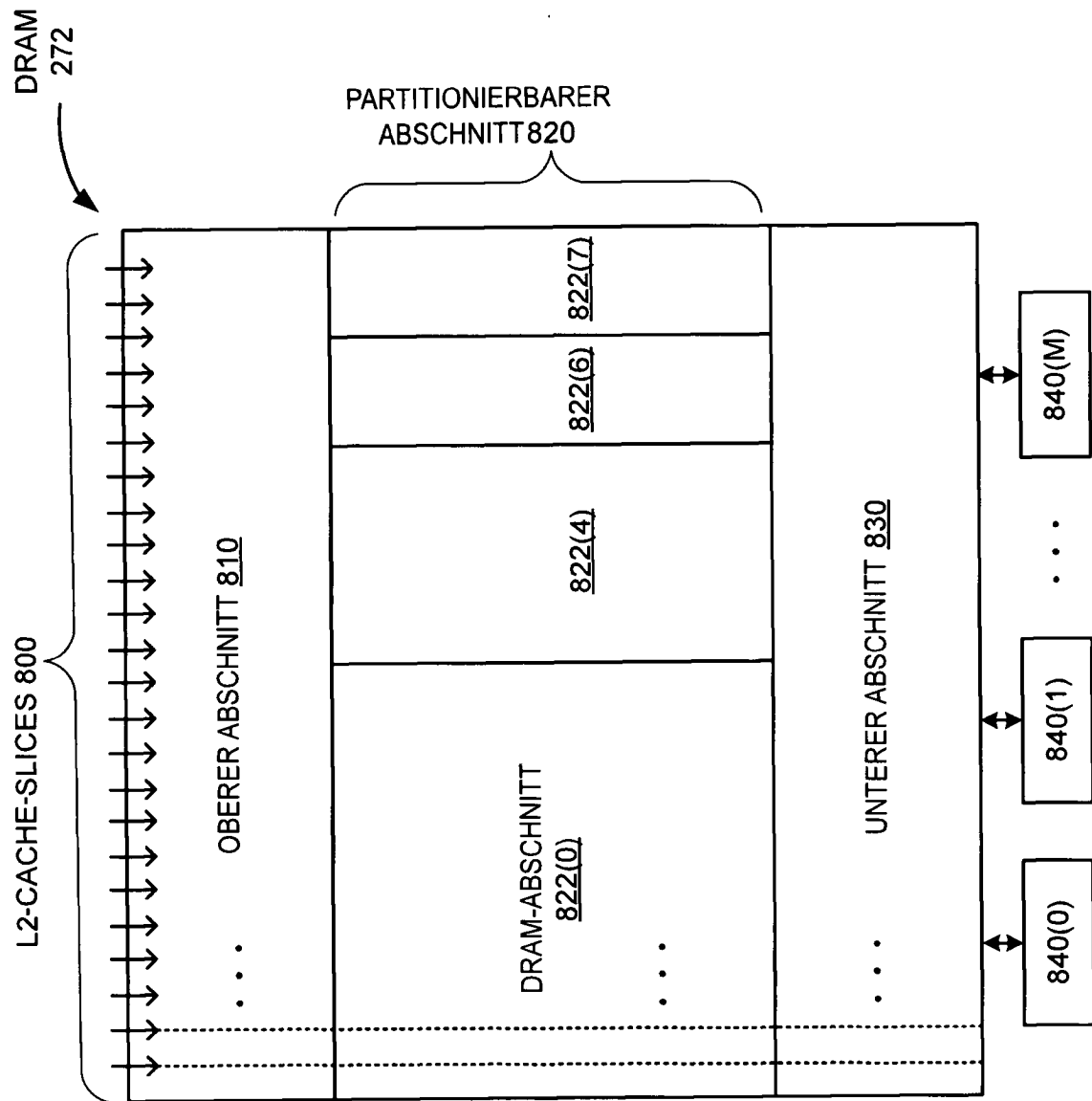


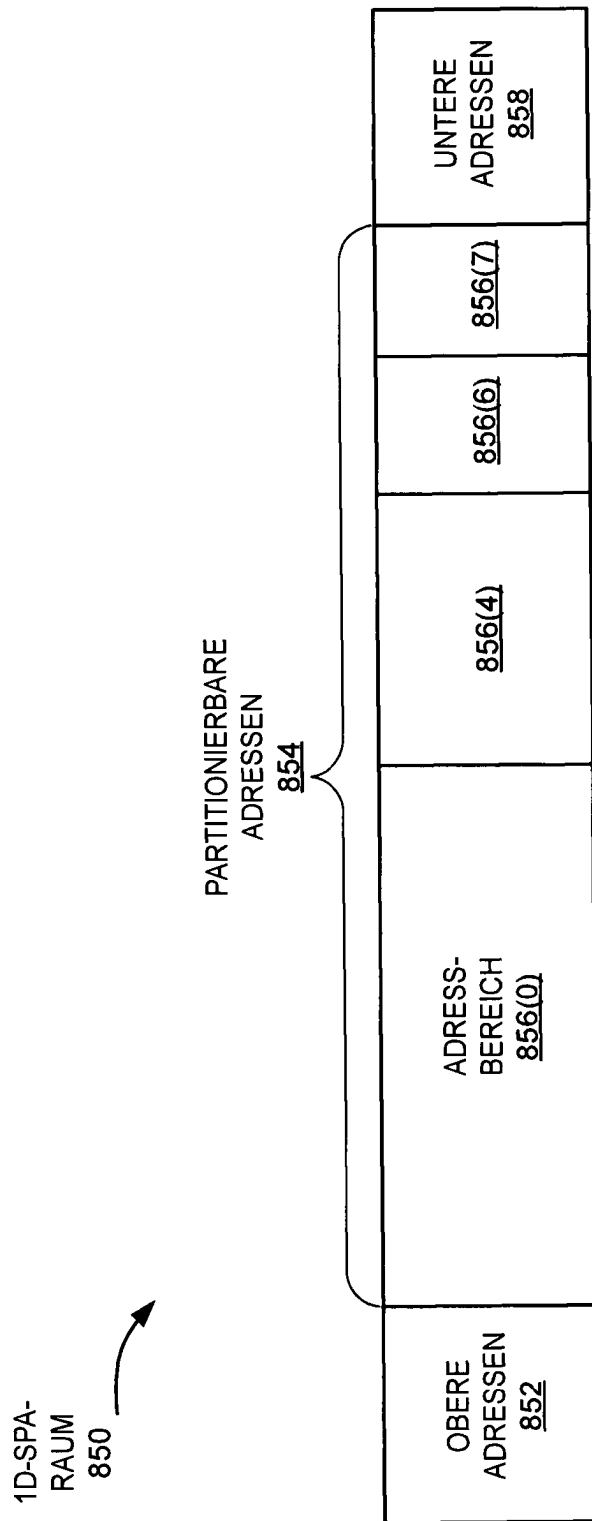
FIGURE 6



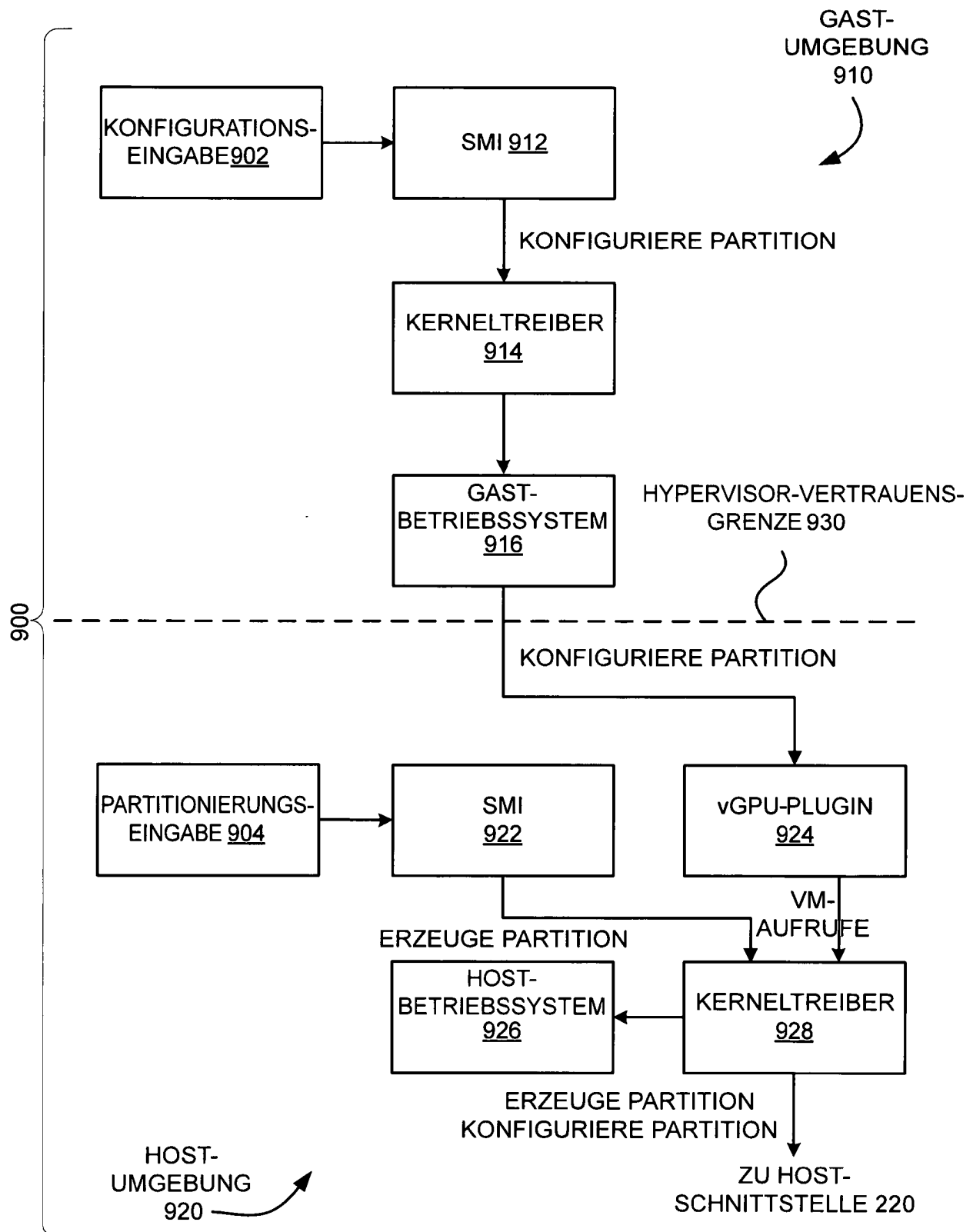
FIGUR 7



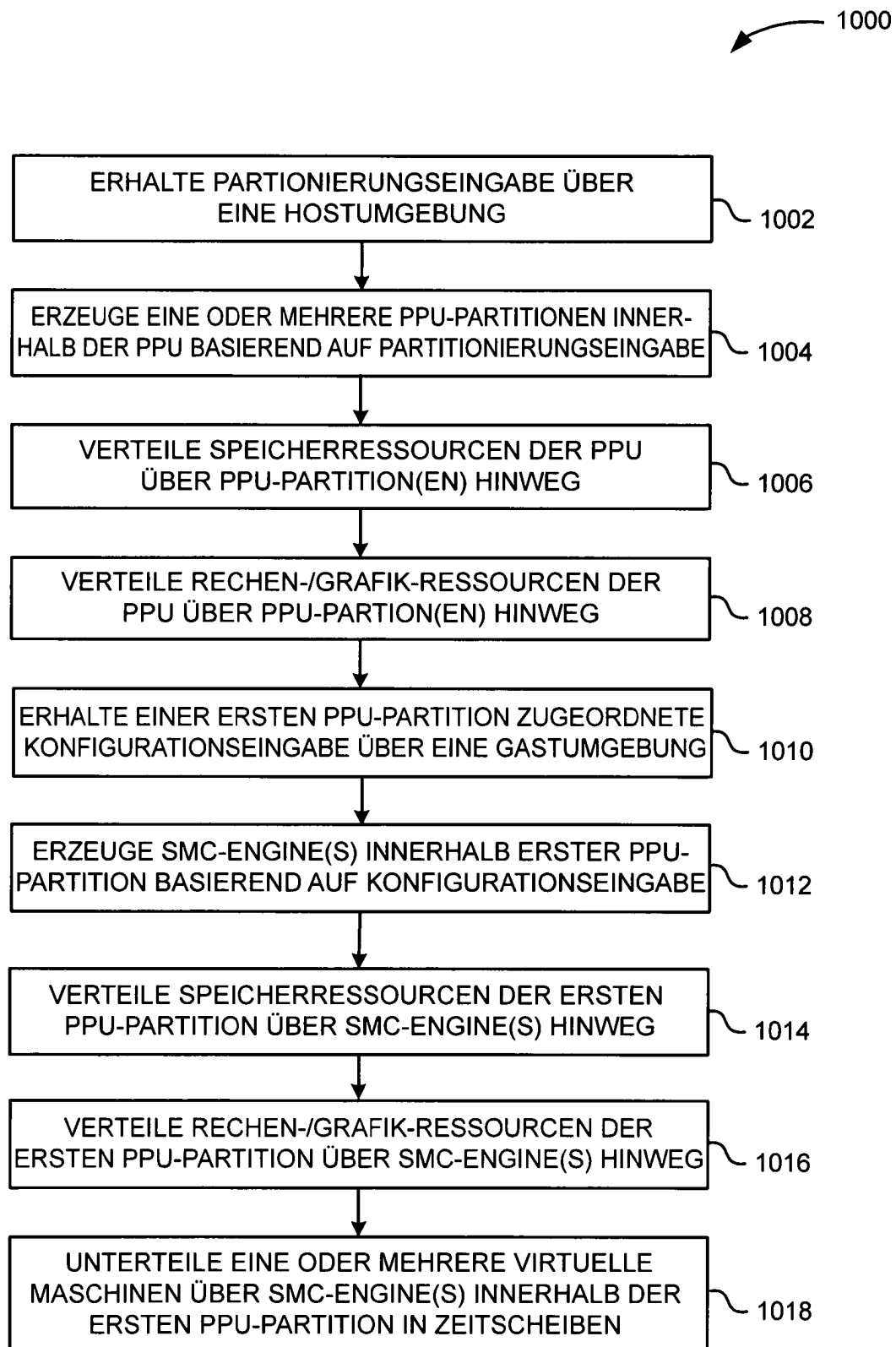
FIGUR 8A



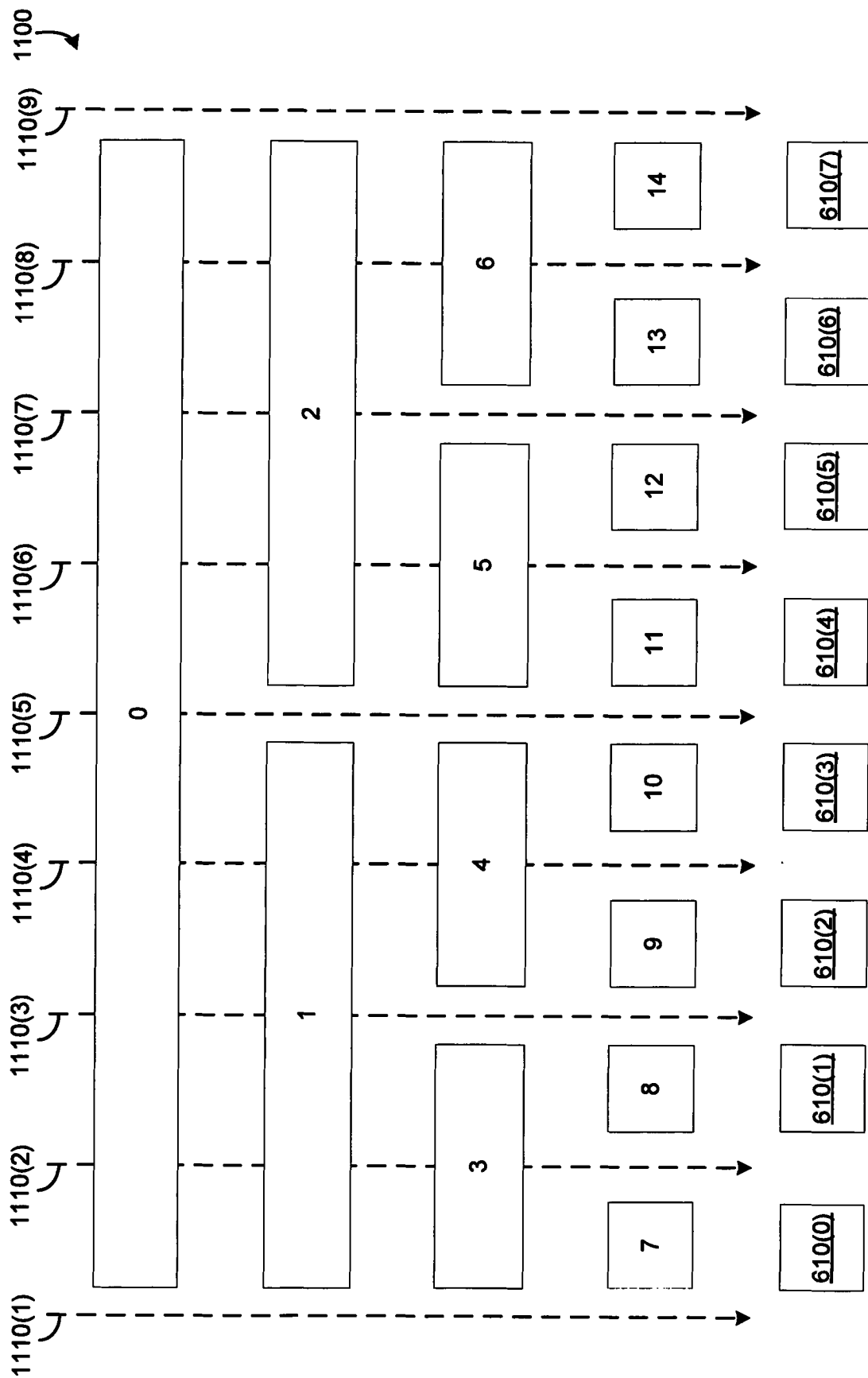
FIGUR 8B



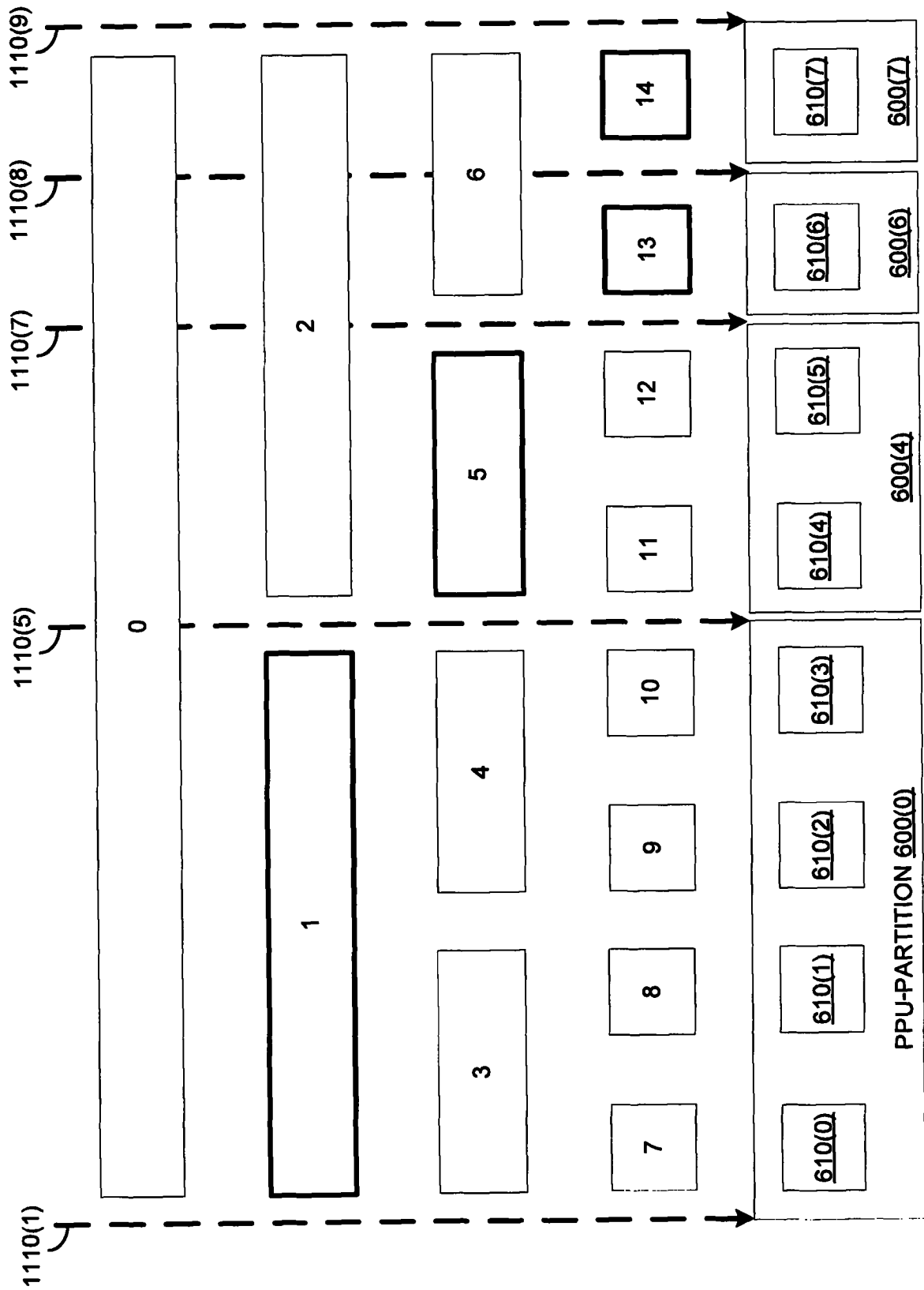
FIGUR 9



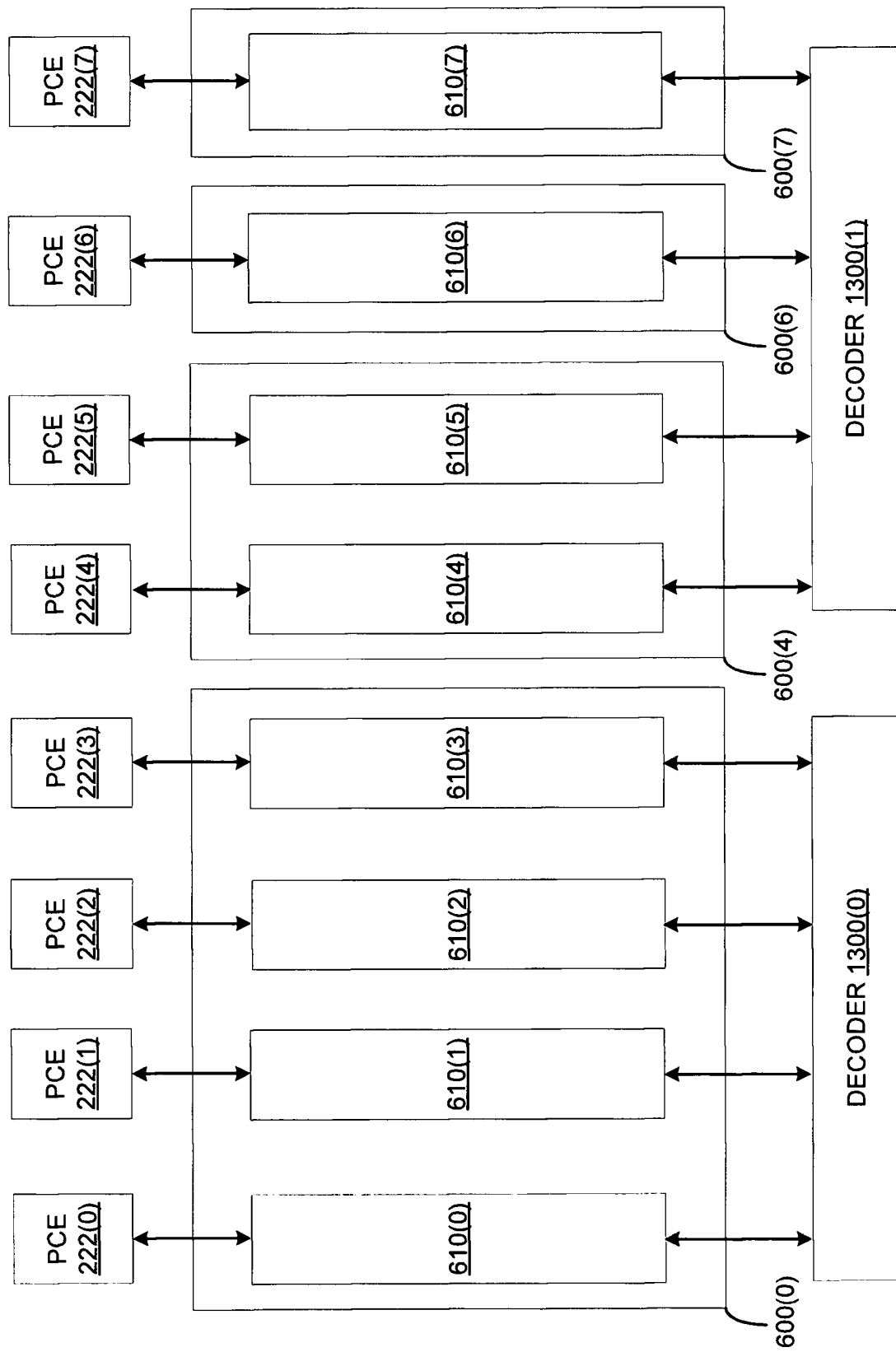
FIGUR 10



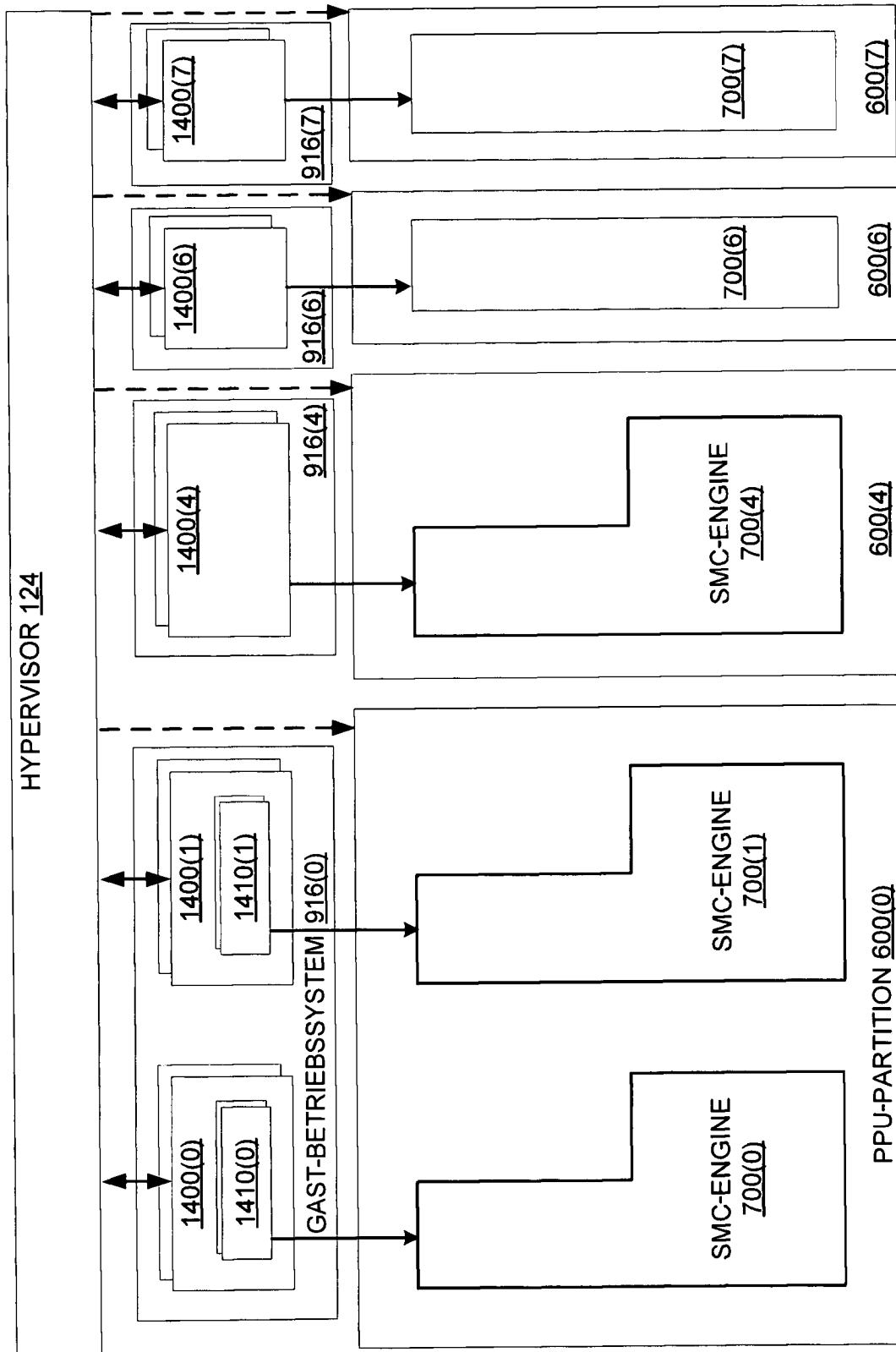
FIGUR 11



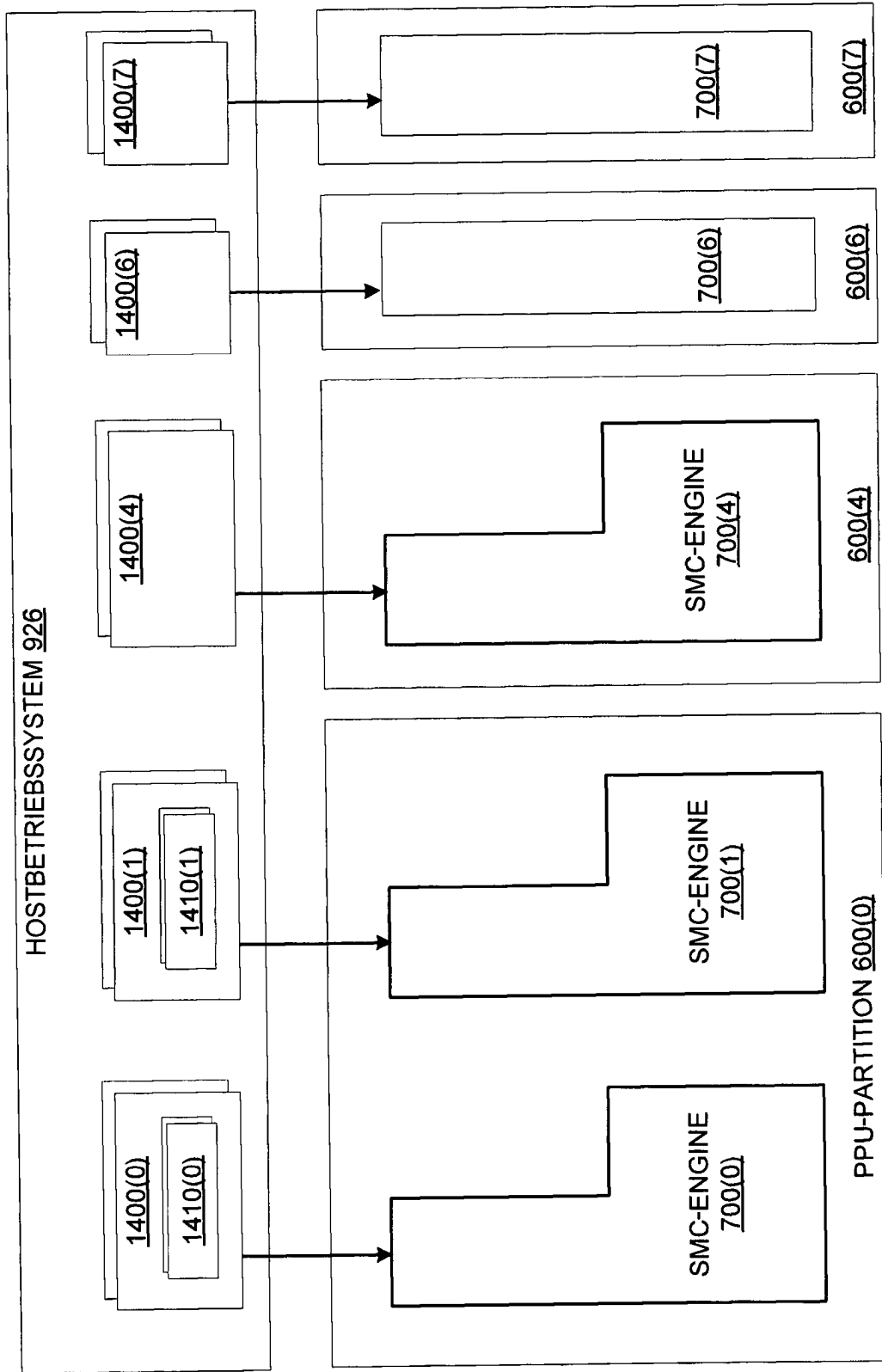
FIGUR 12



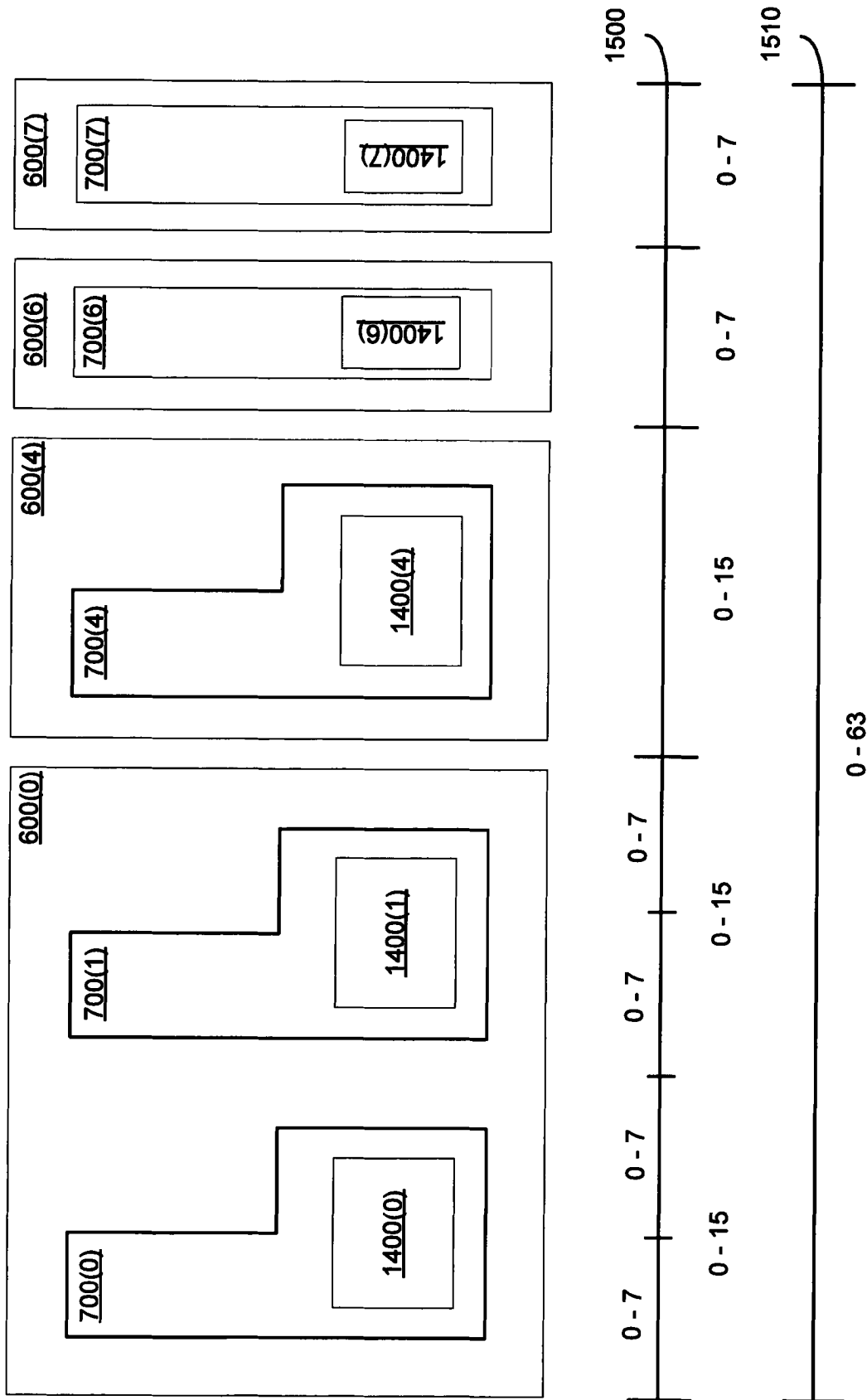
FIGUR 13



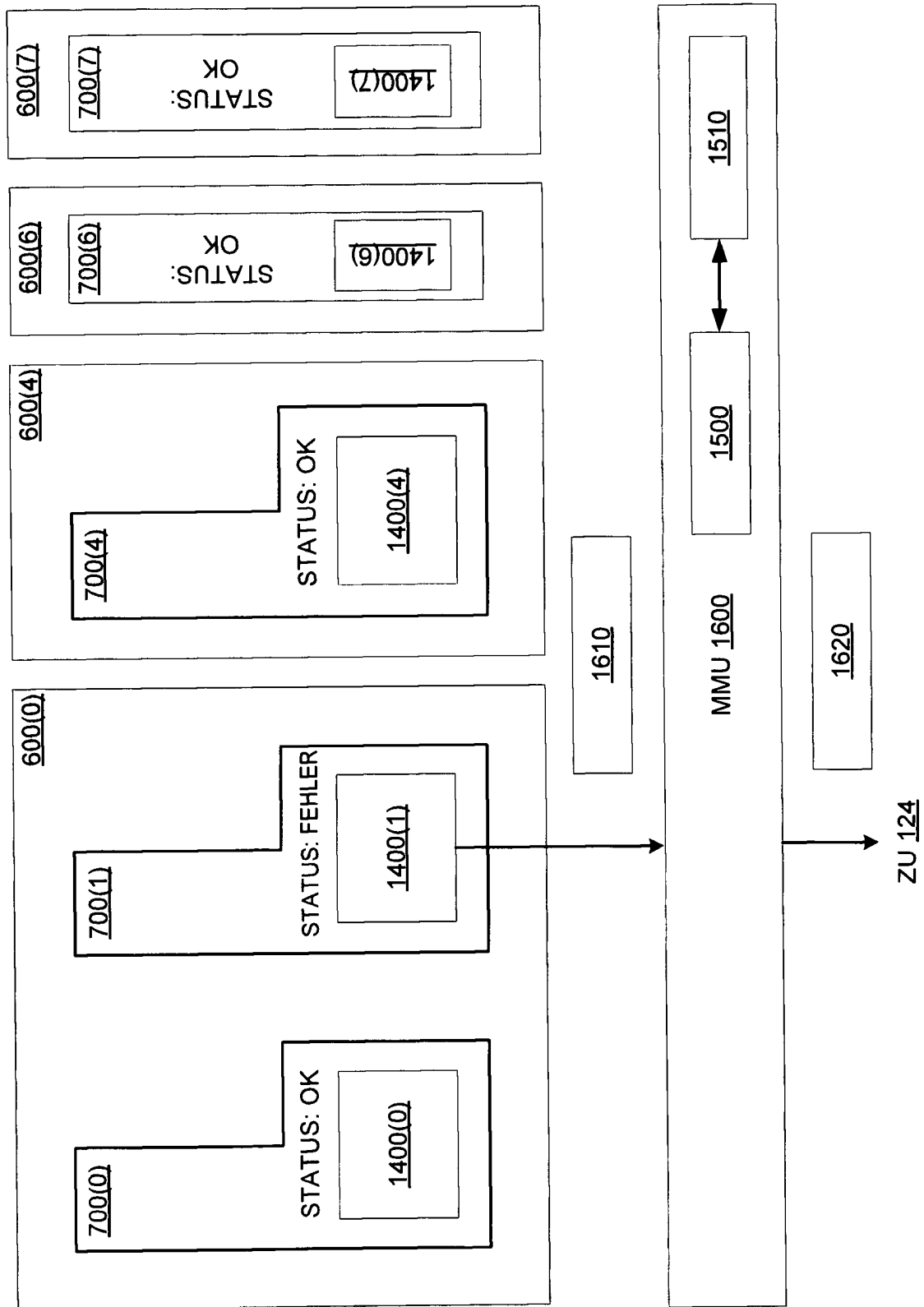
FIGUR 14A



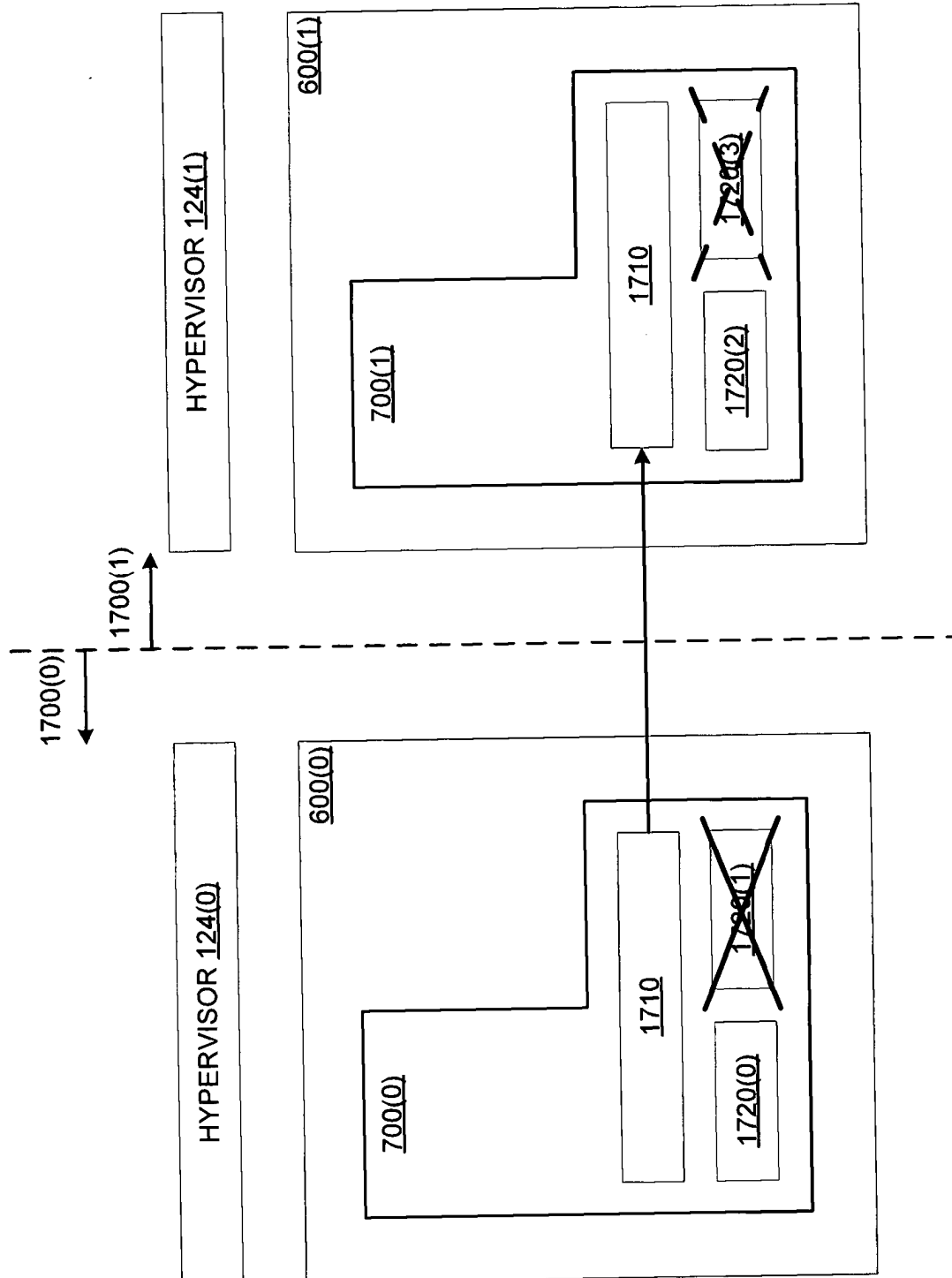
FIGUR 14B



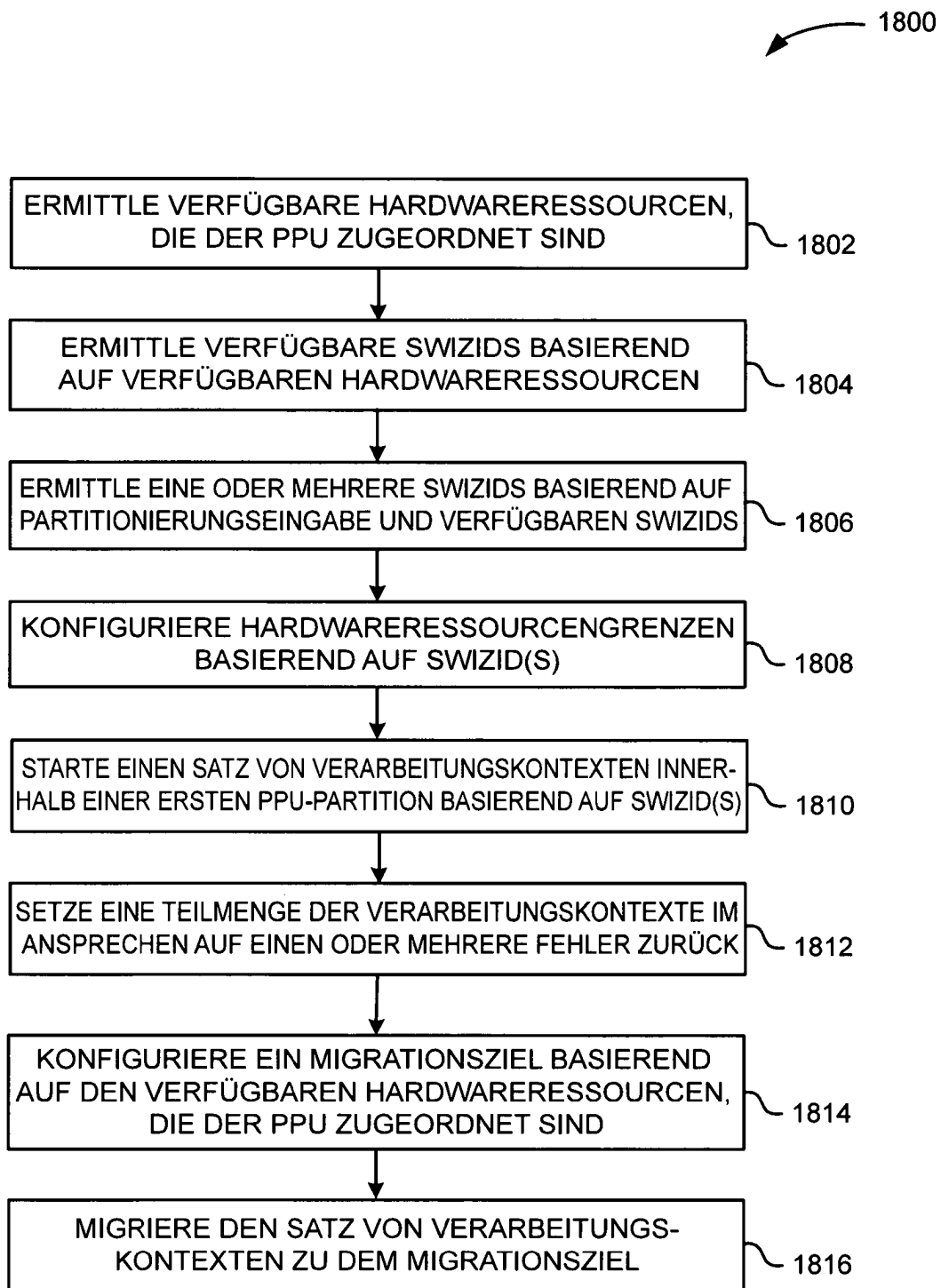
FIGUR 15



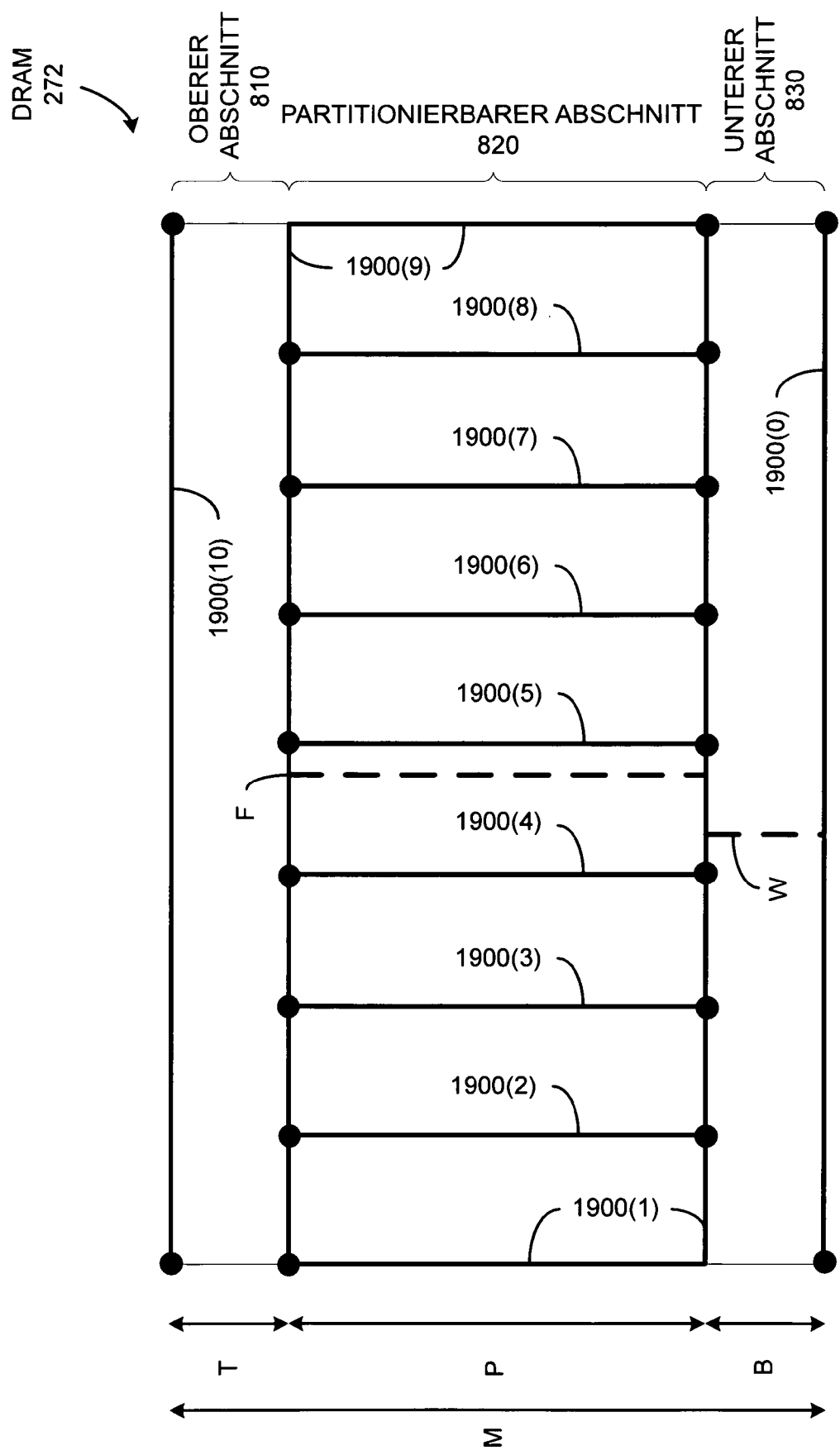
FIGUR 16



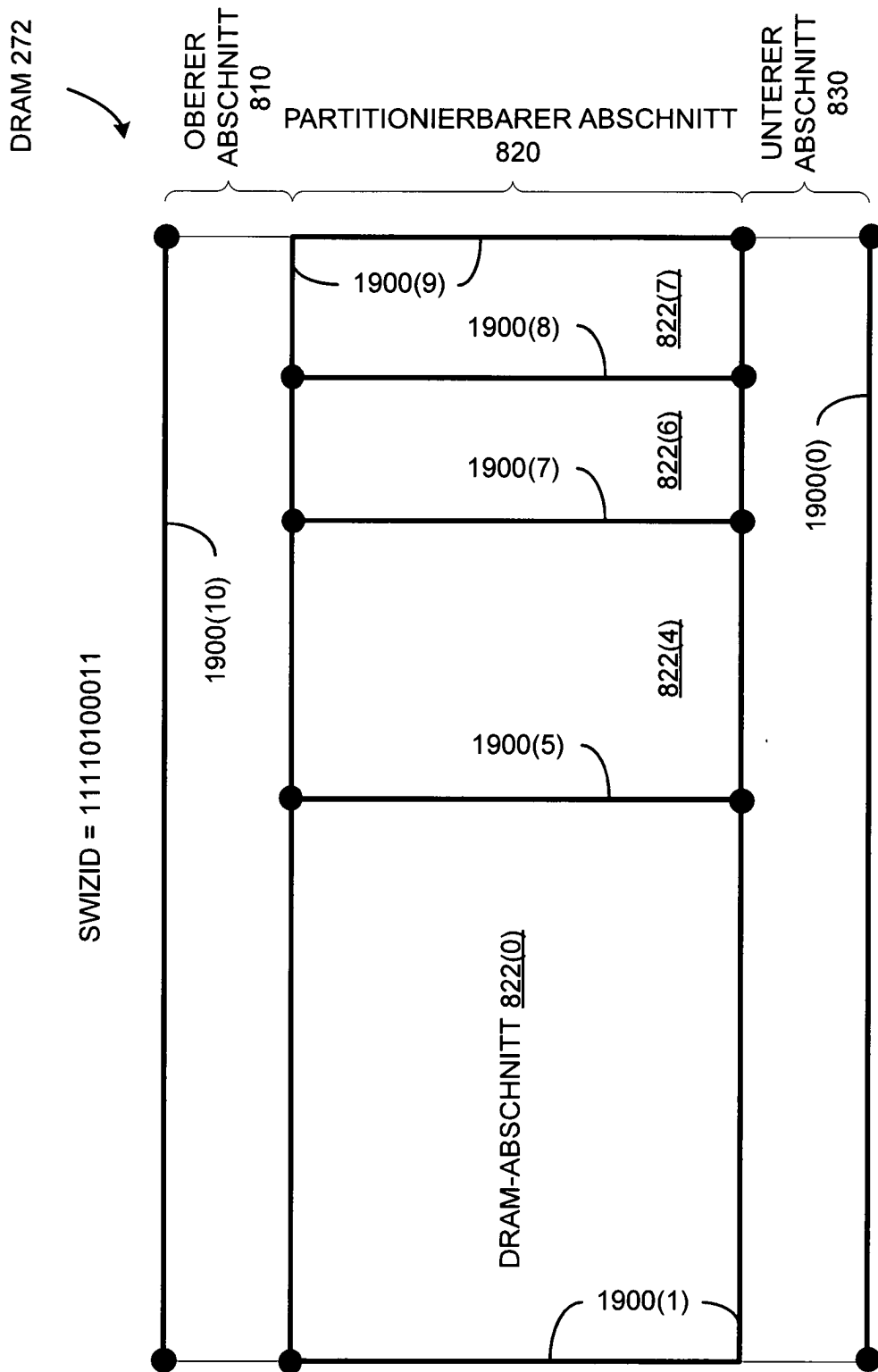
FIGUR 17



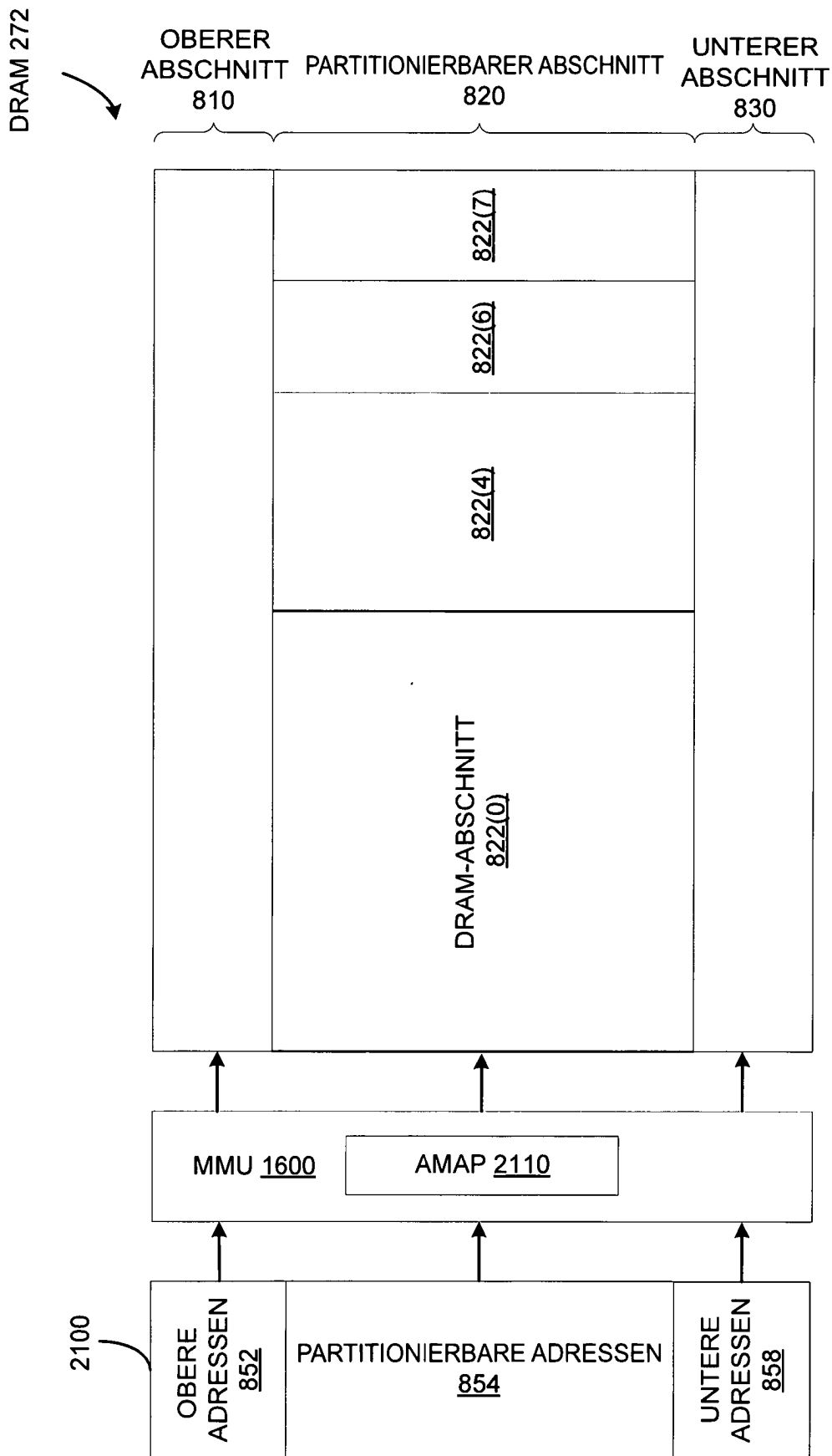
FIGUR 18



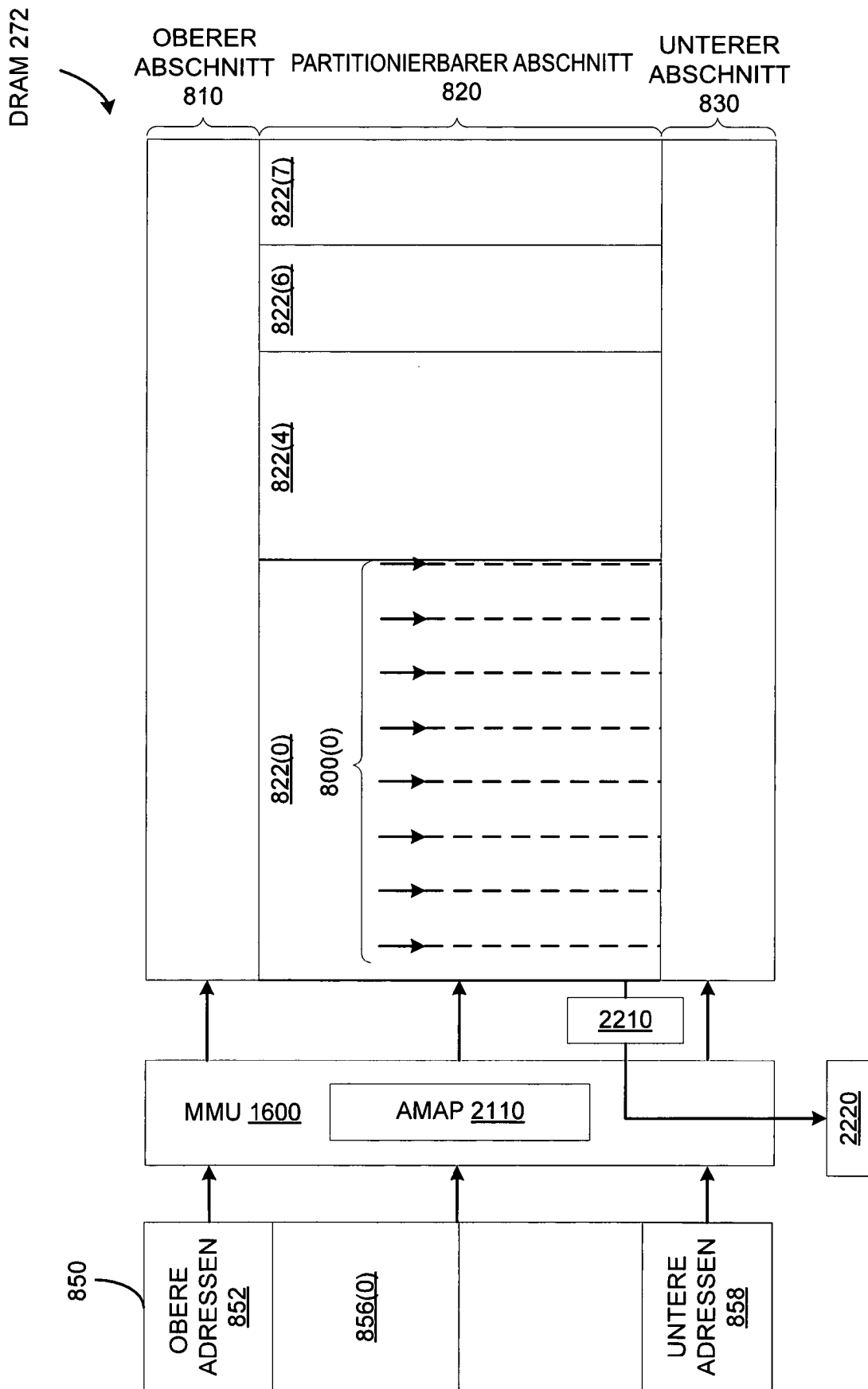
FIGUR 19



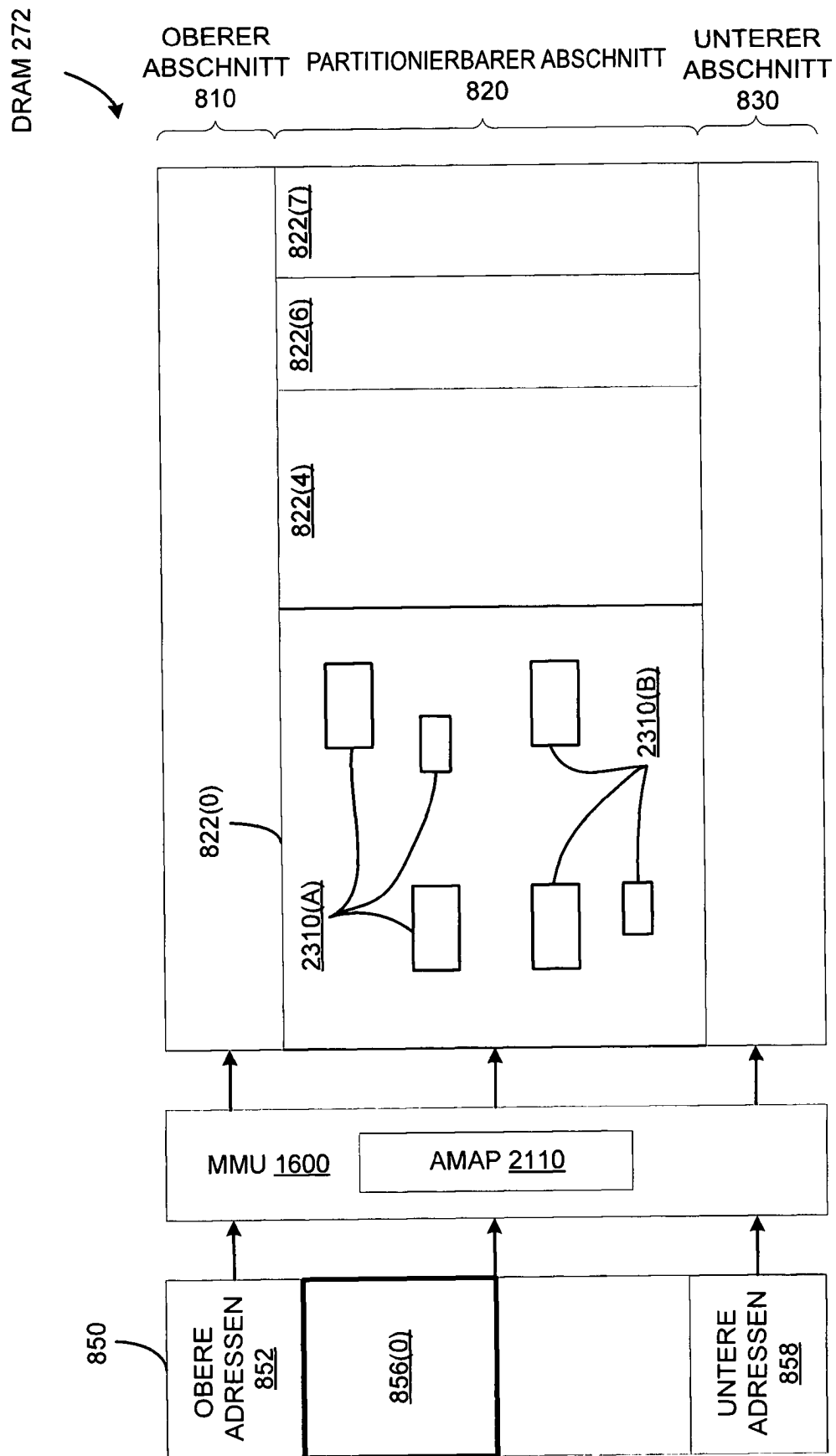
FIGUR 20



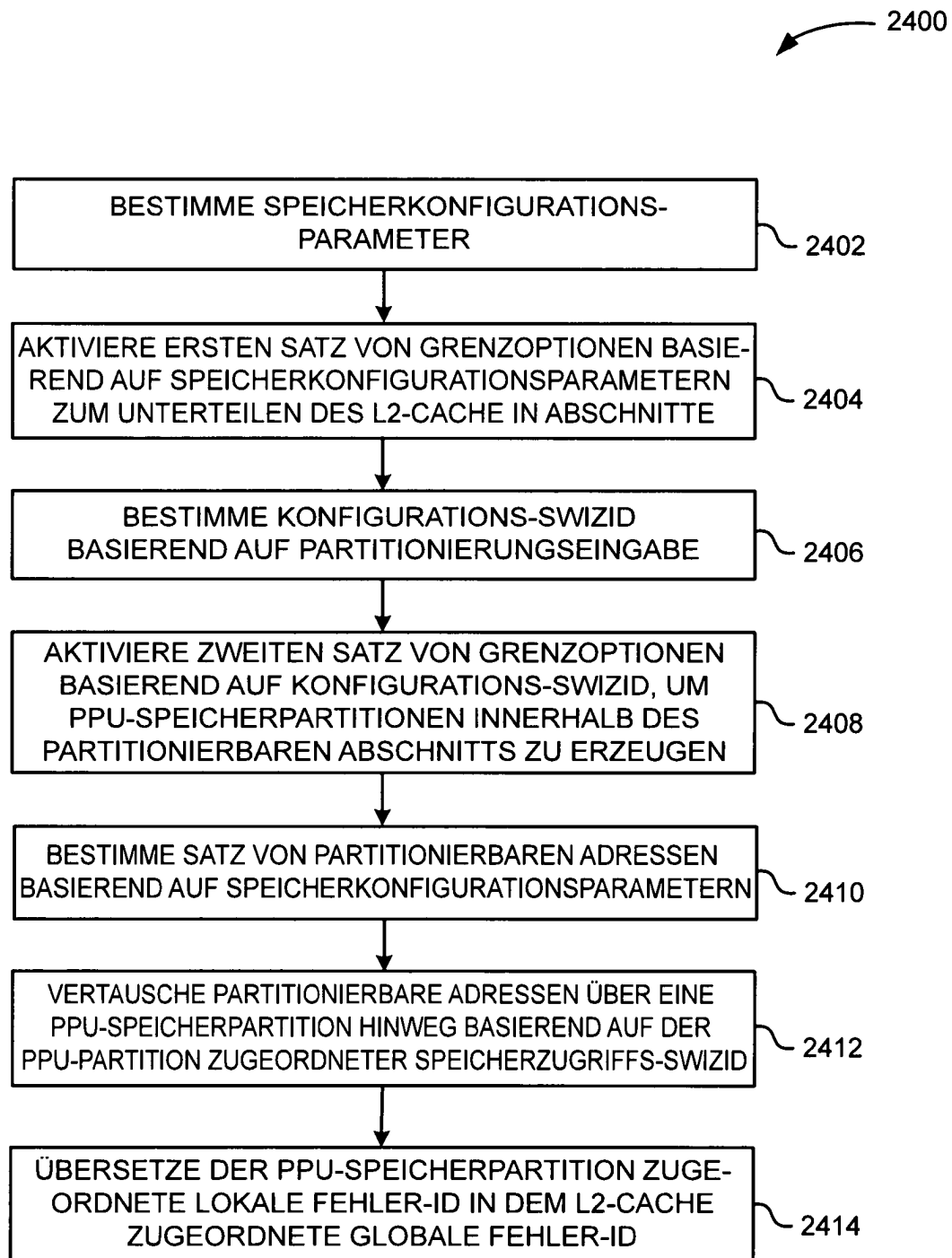
FIGUR 21



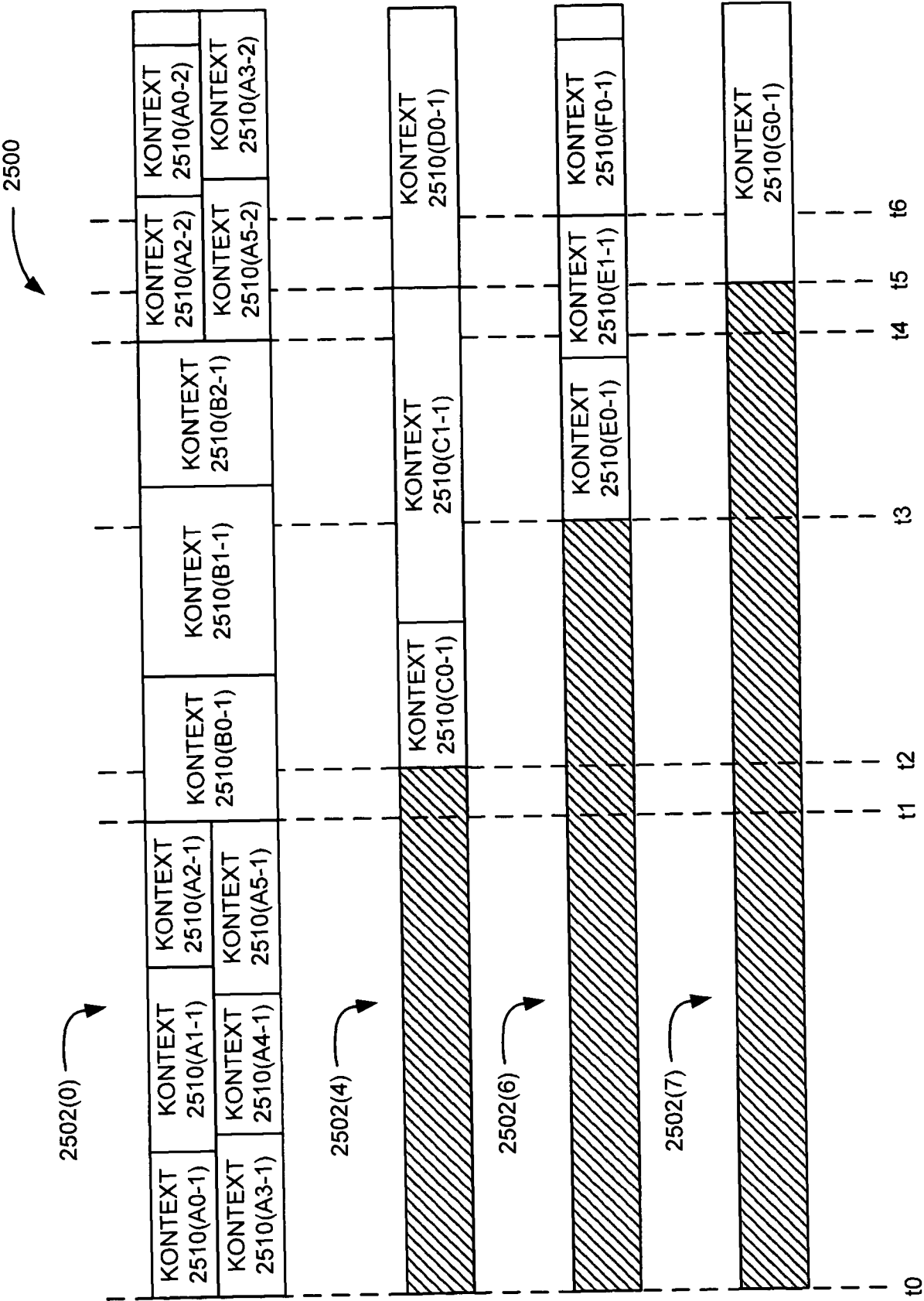
FIGUR 22



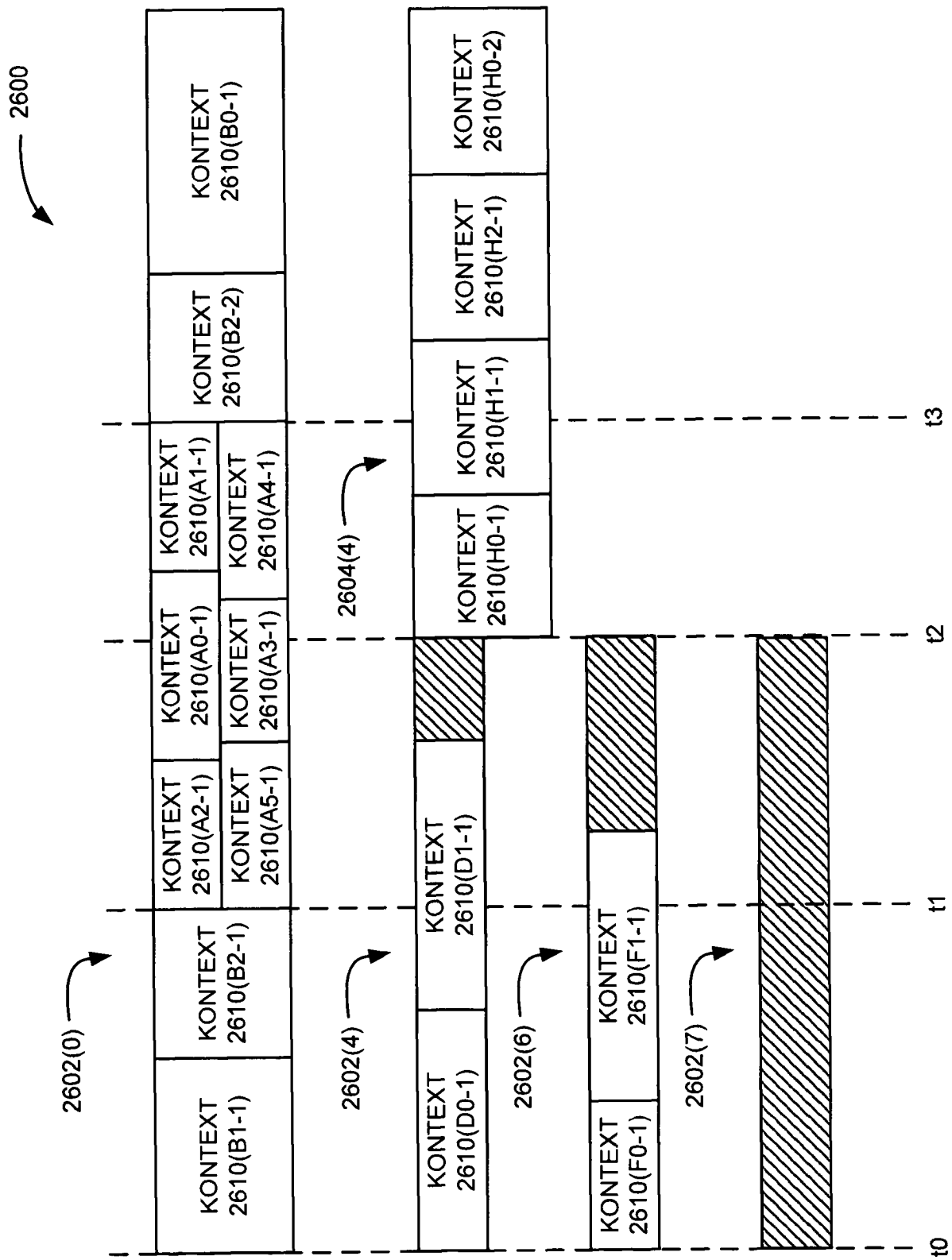
FIGUR 23



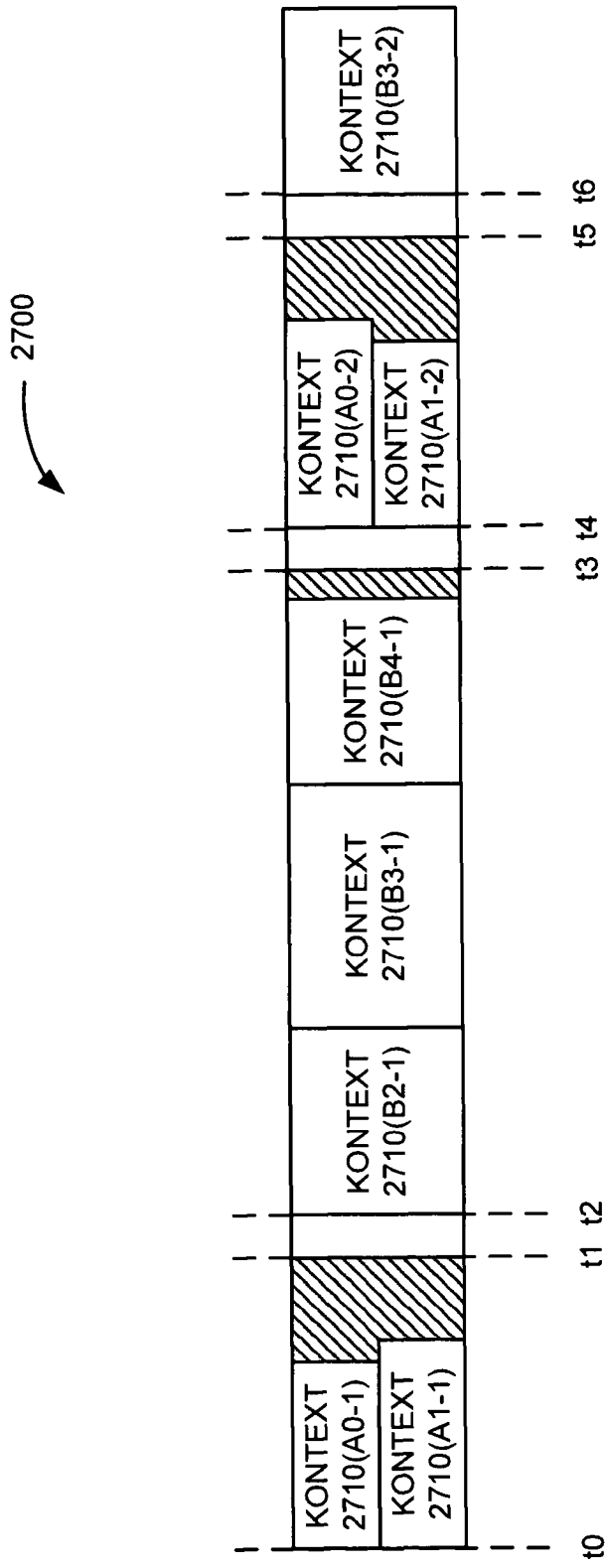
FIGUR 24



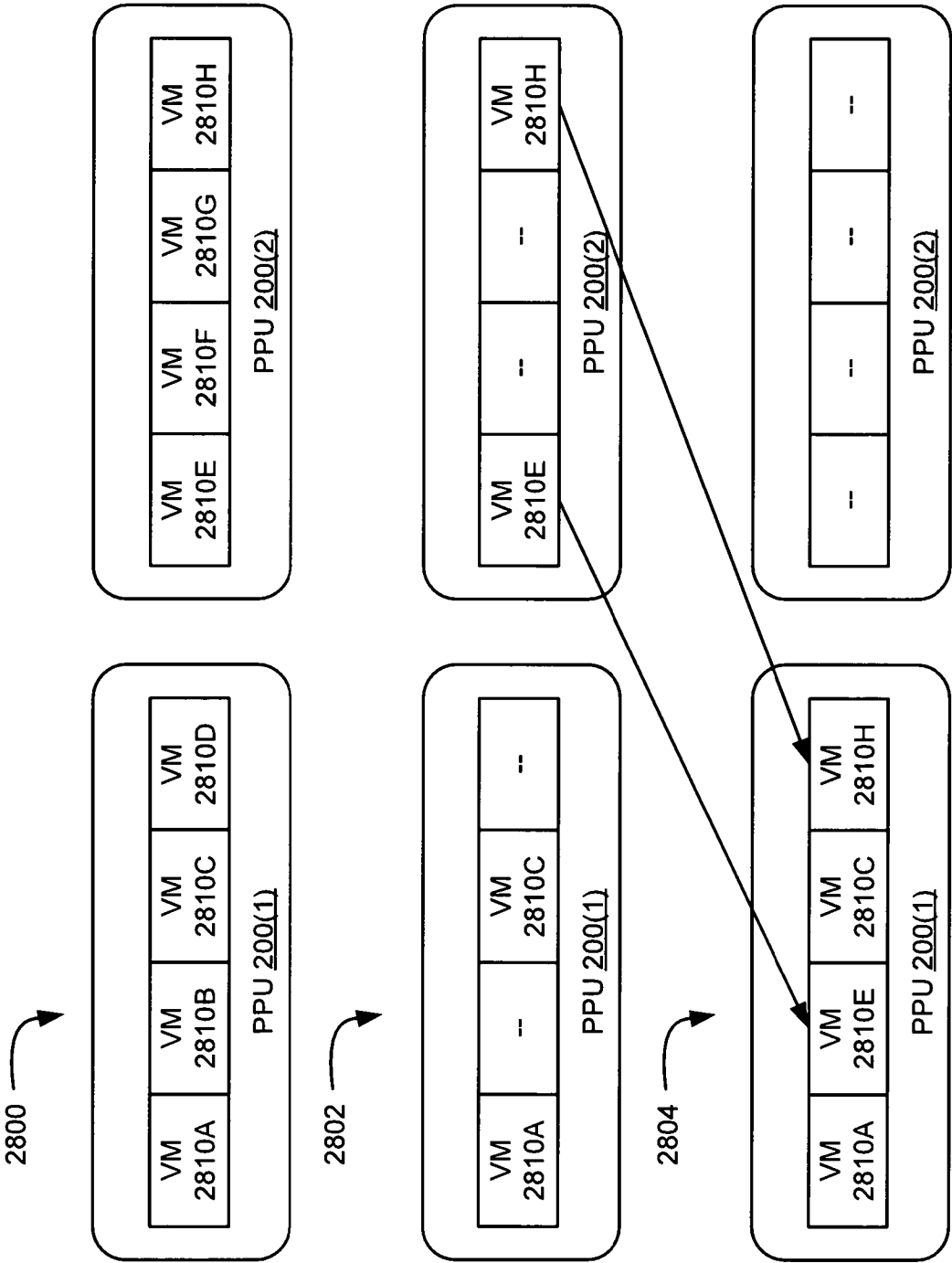
FIGUR 25



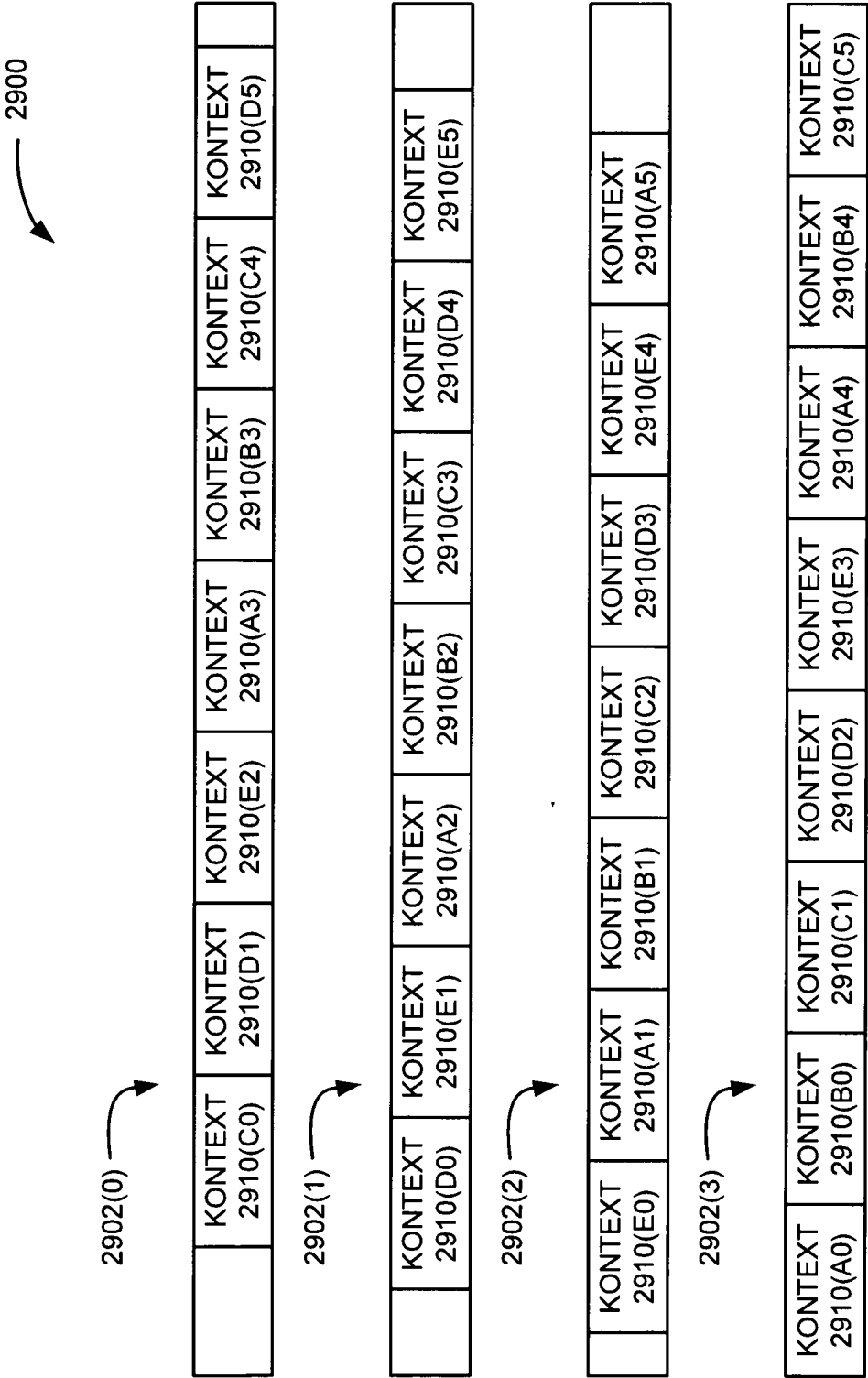
FIGUR 26



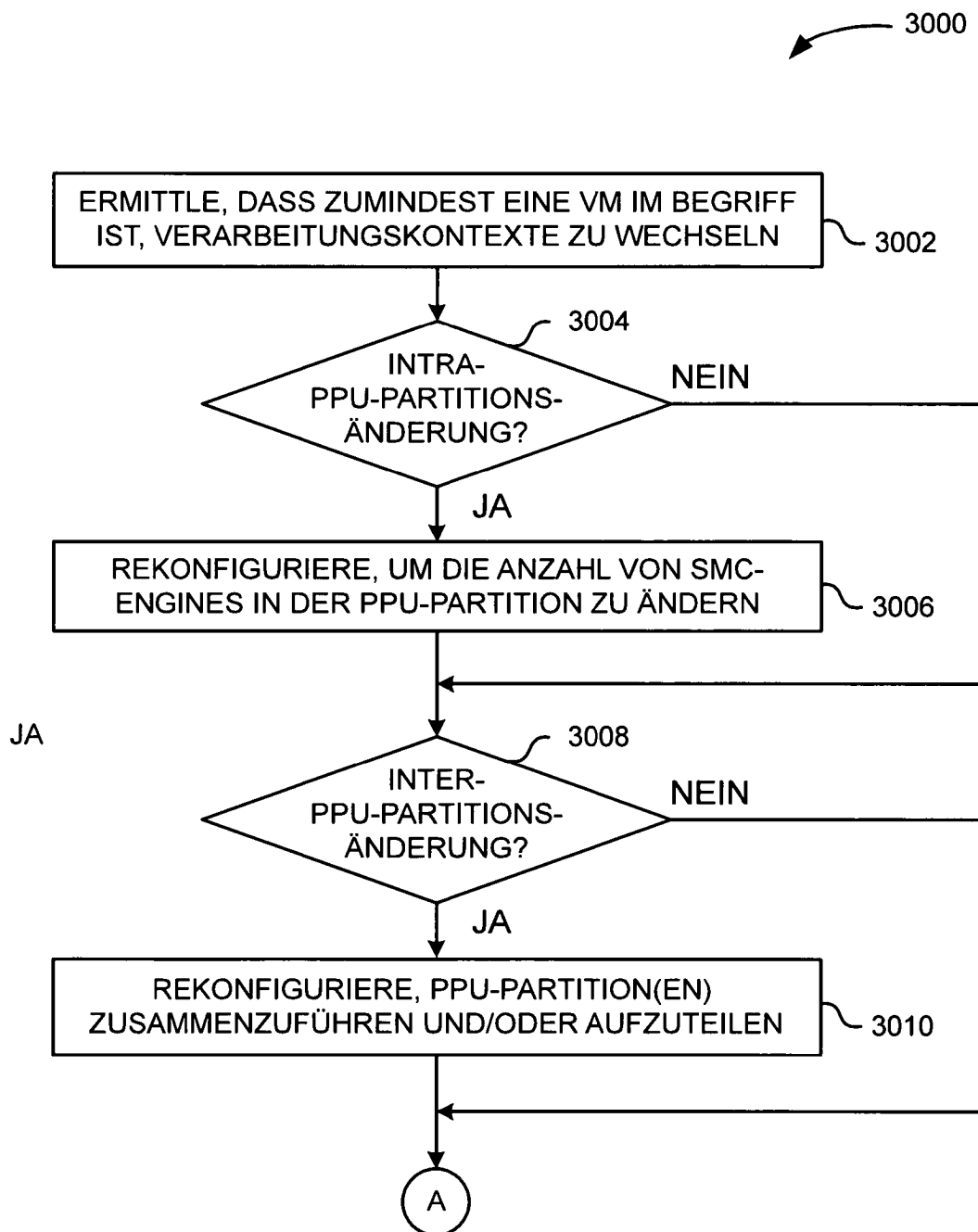
FIGUR 27



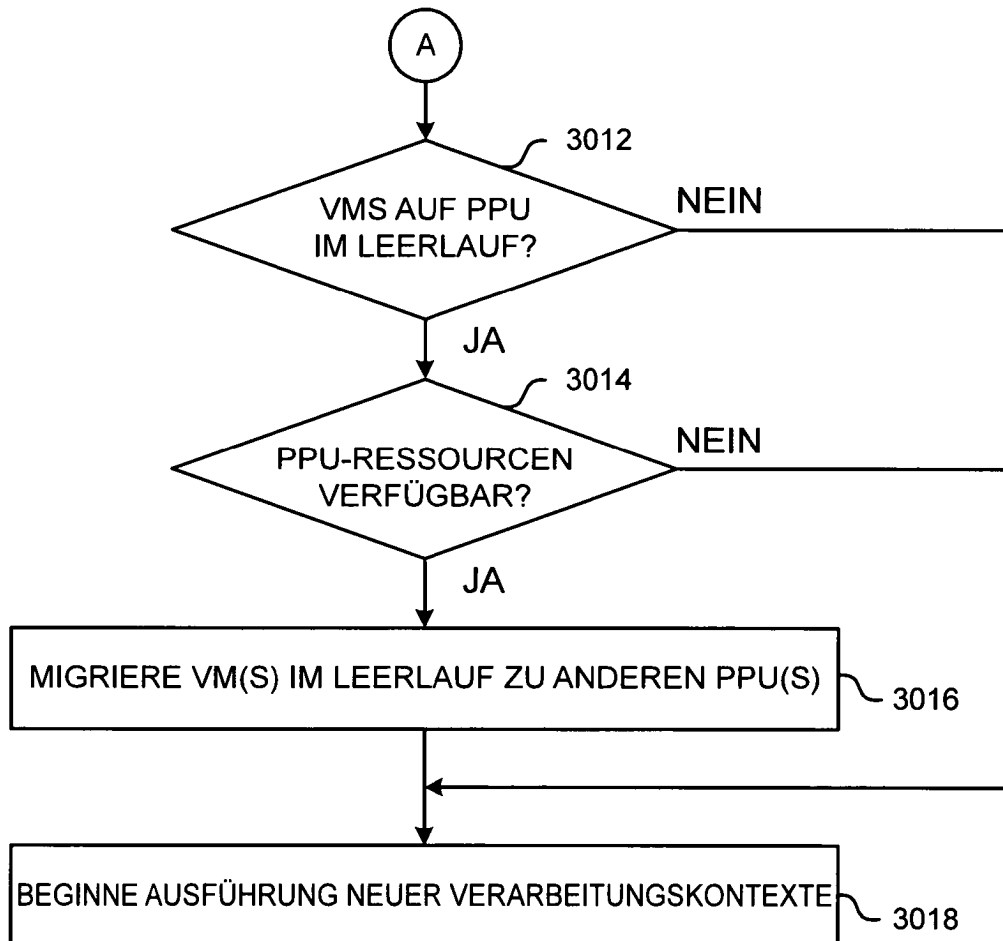
FIGUR 28



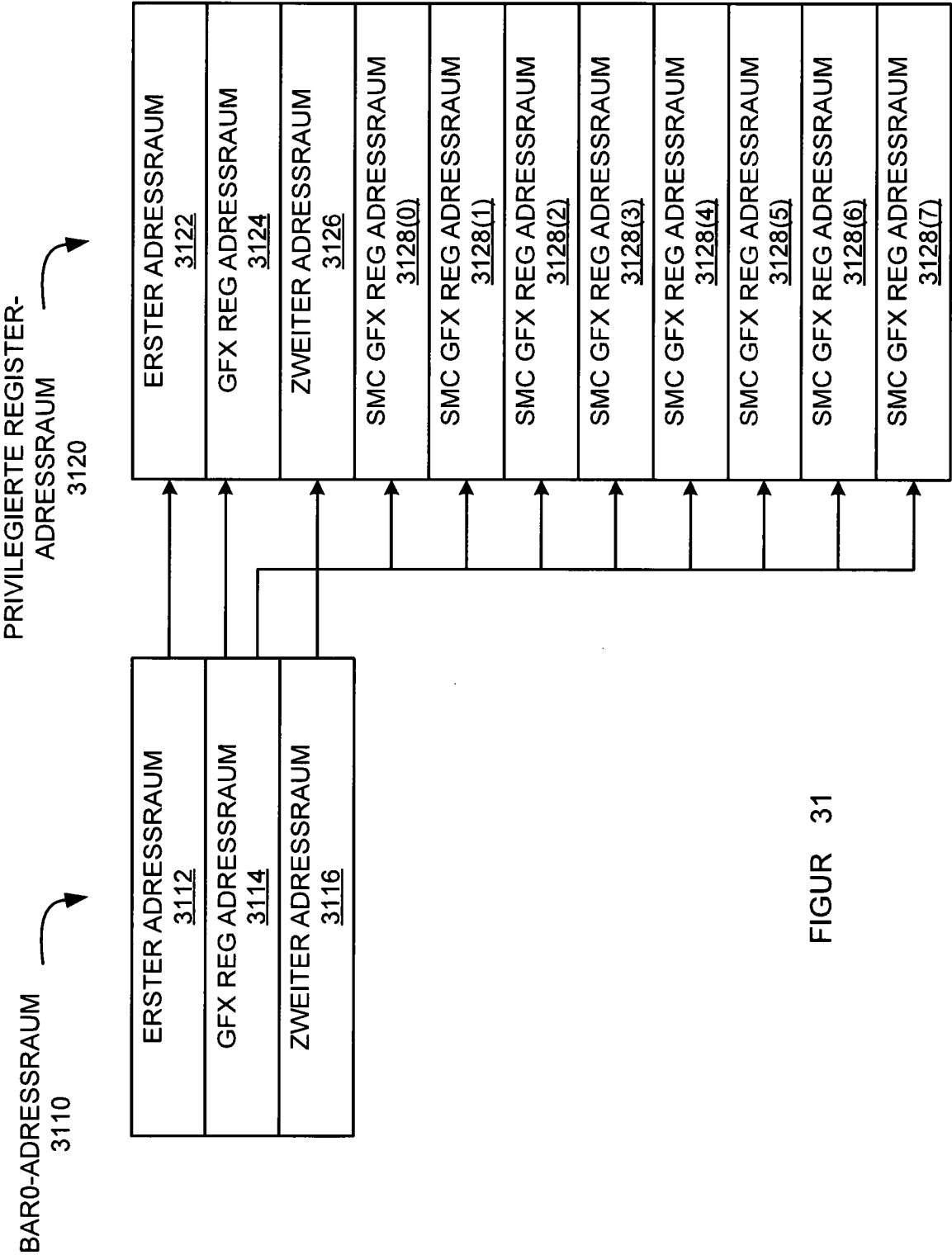
FIGUR 29



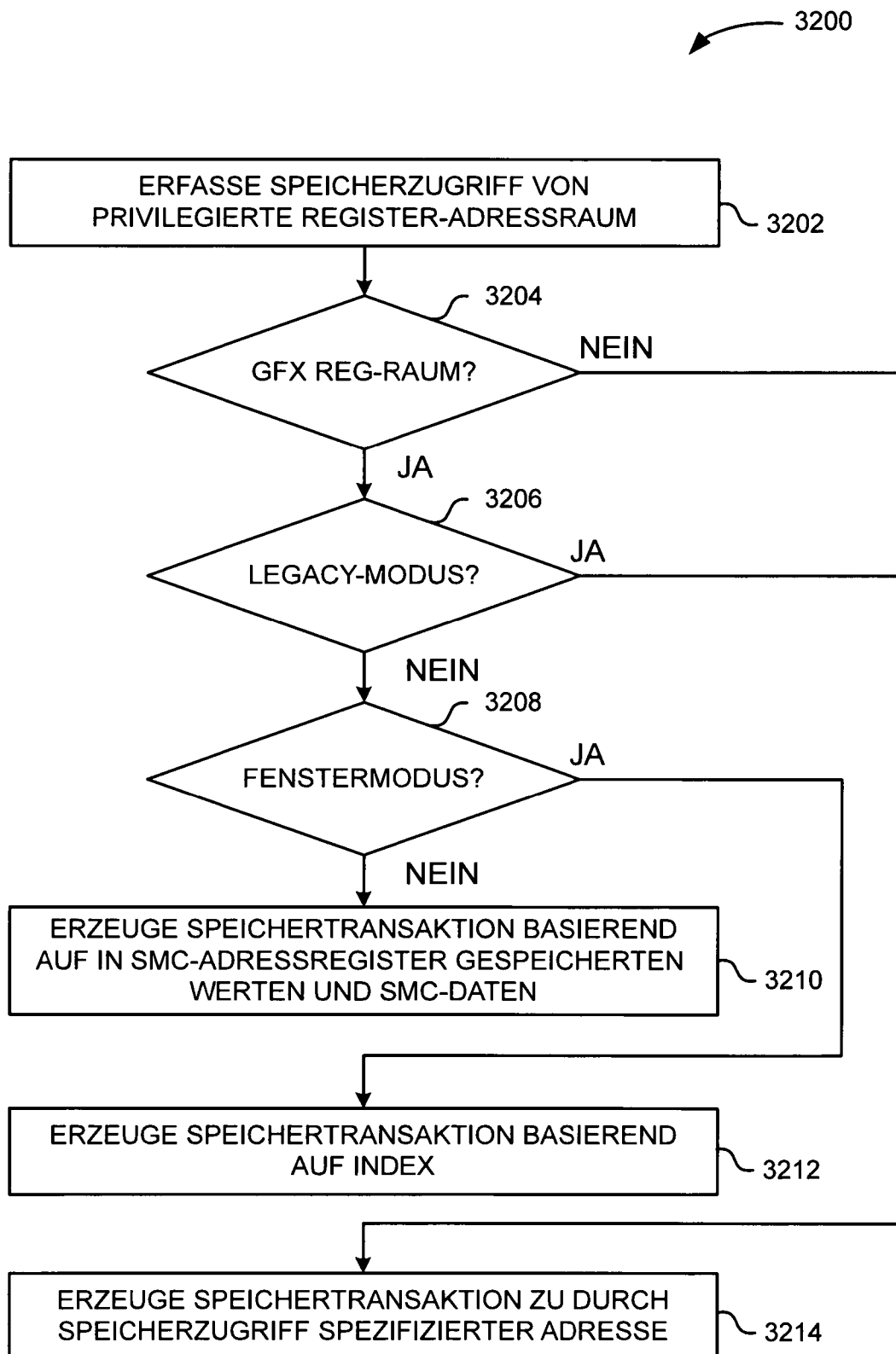
FIGUR 30A



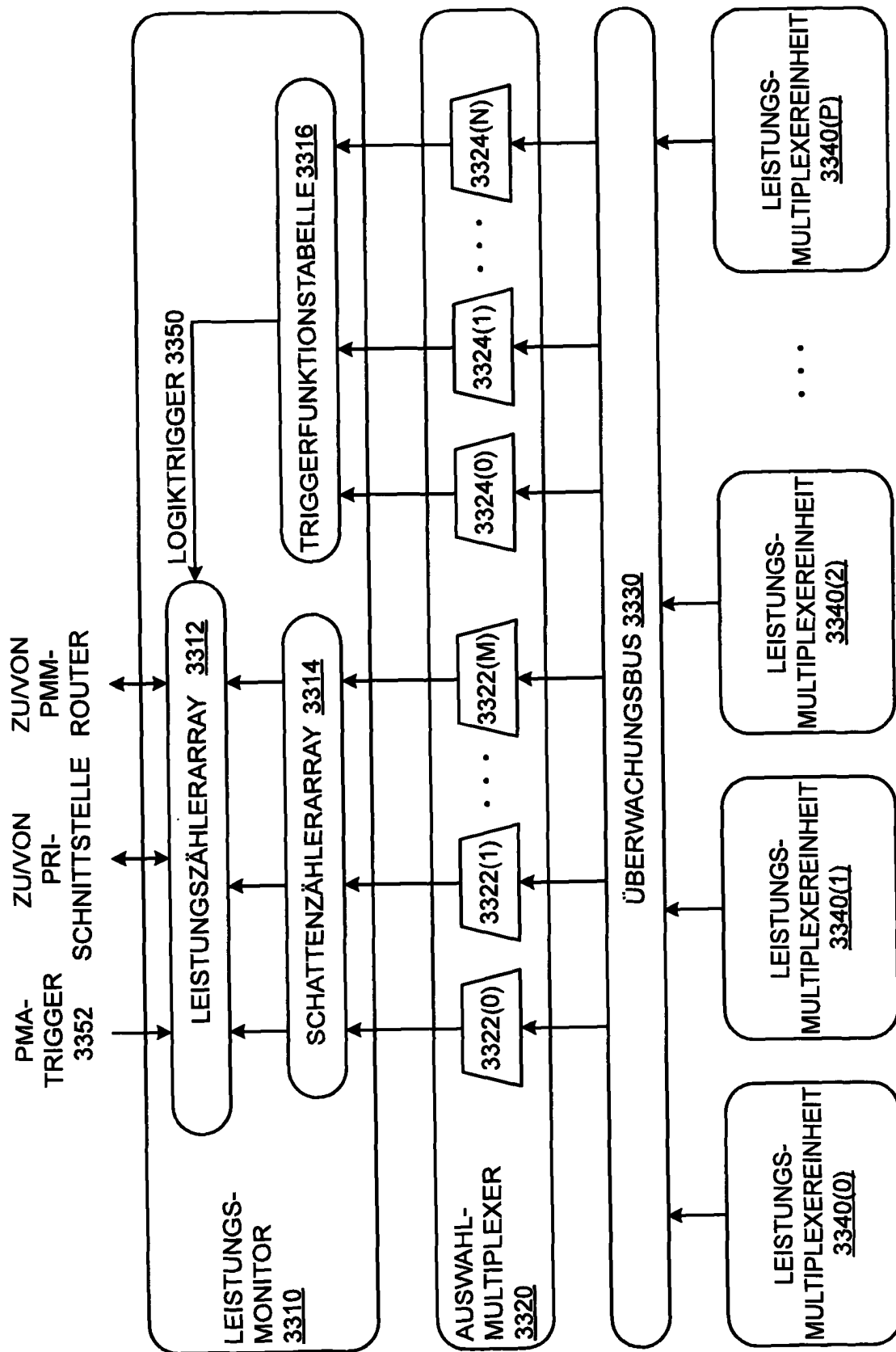
FIGUR 30B



FIGUR 31

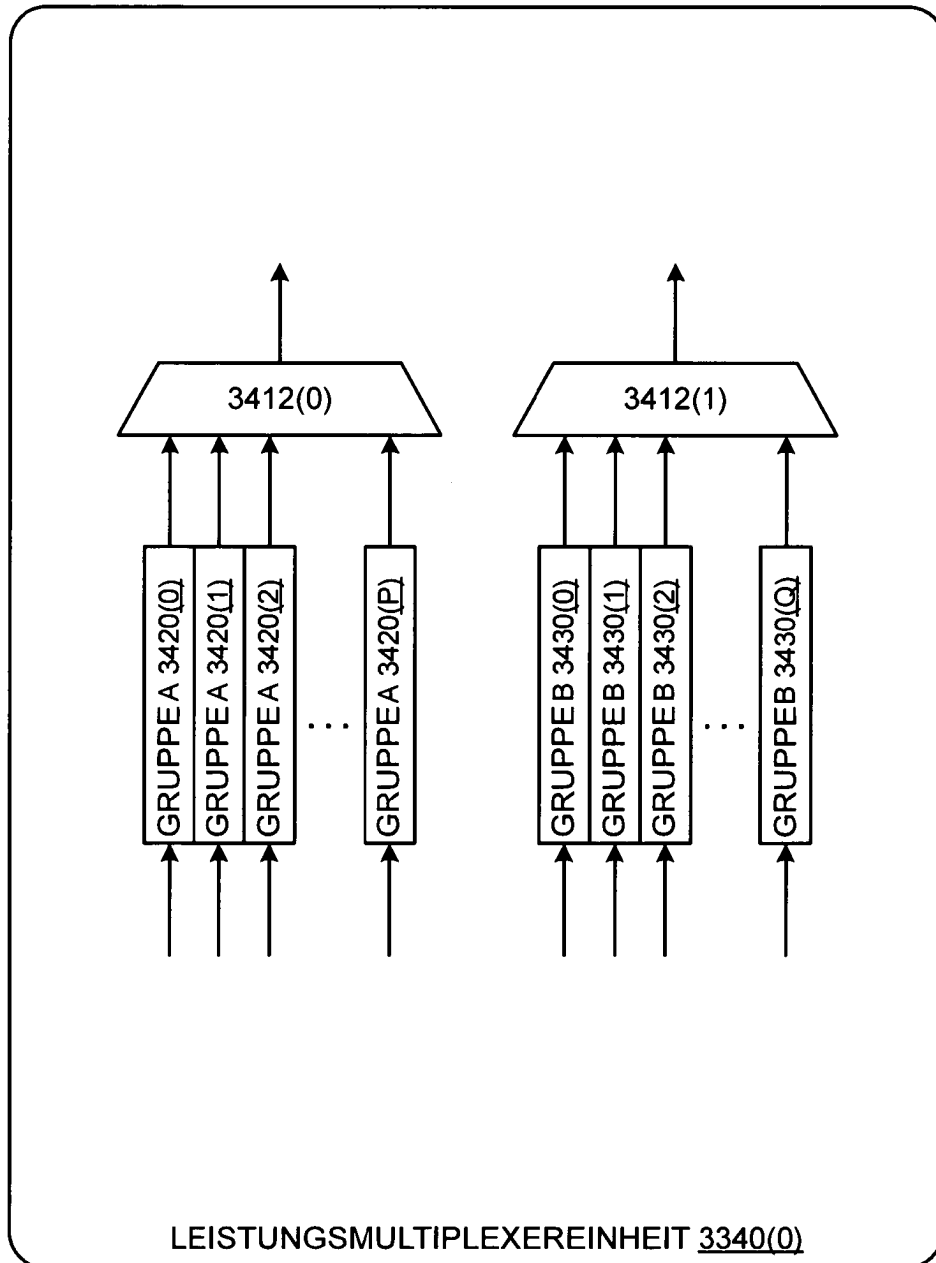


FIGUR 32

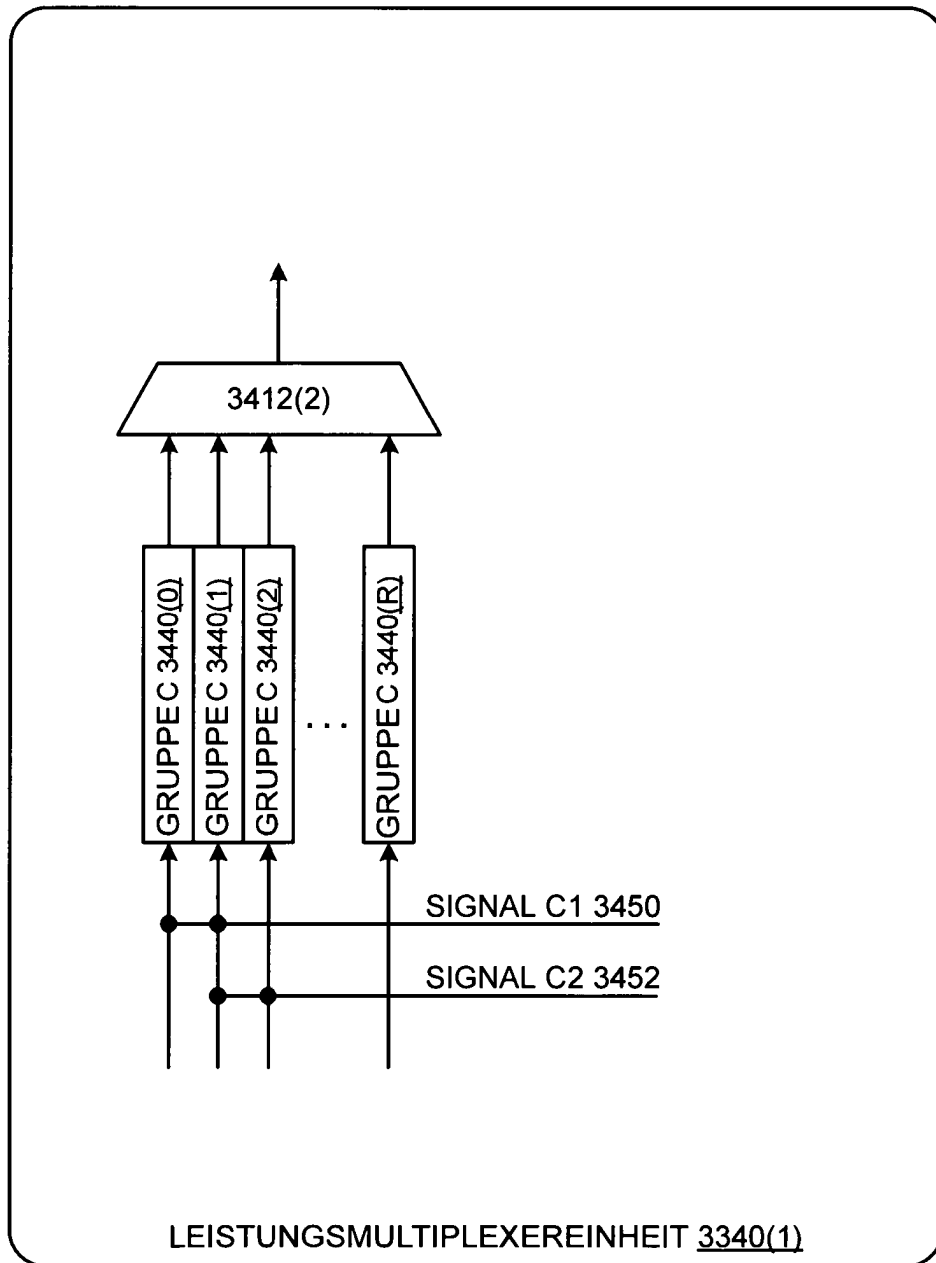


FIGUR 33

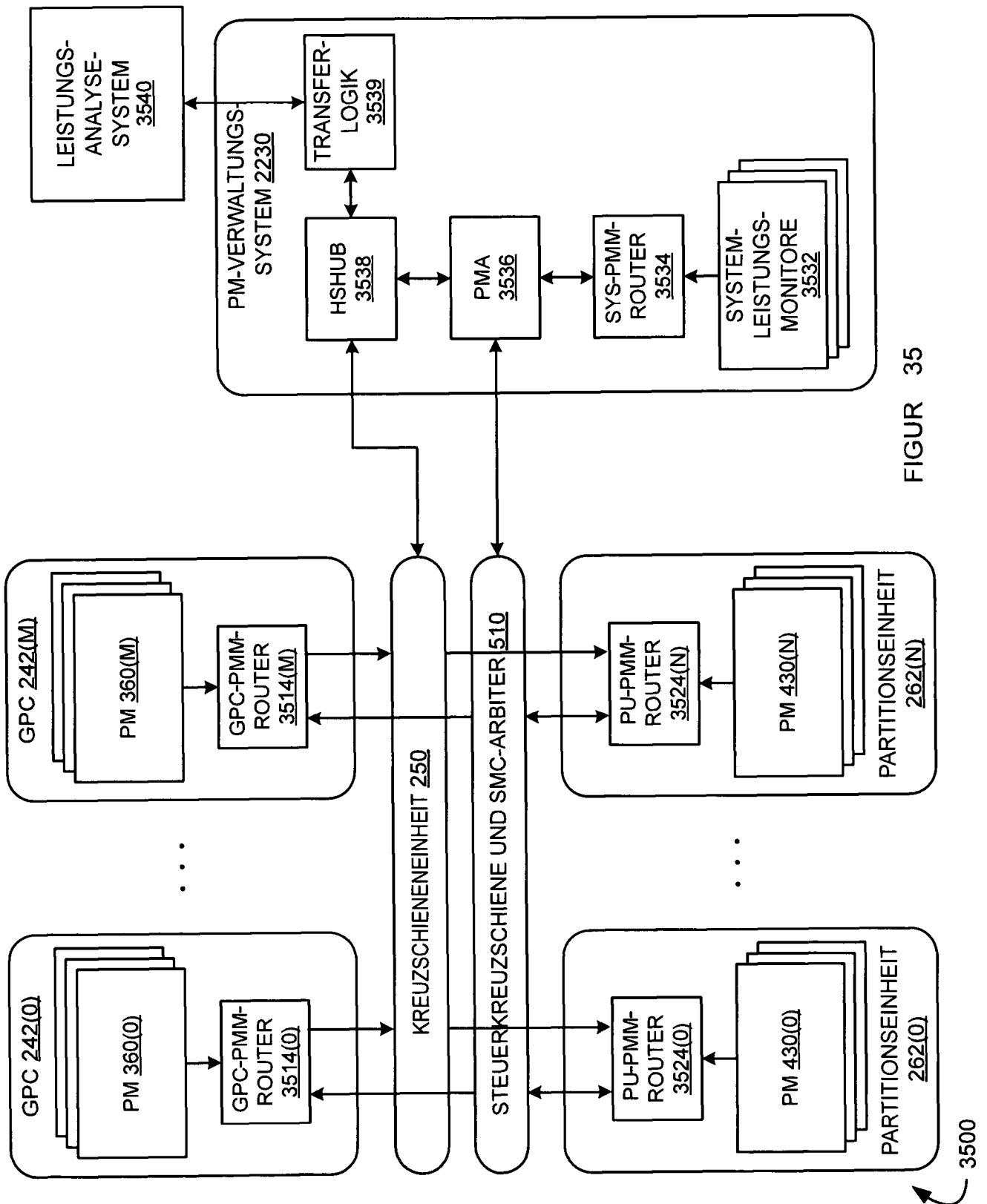
3300



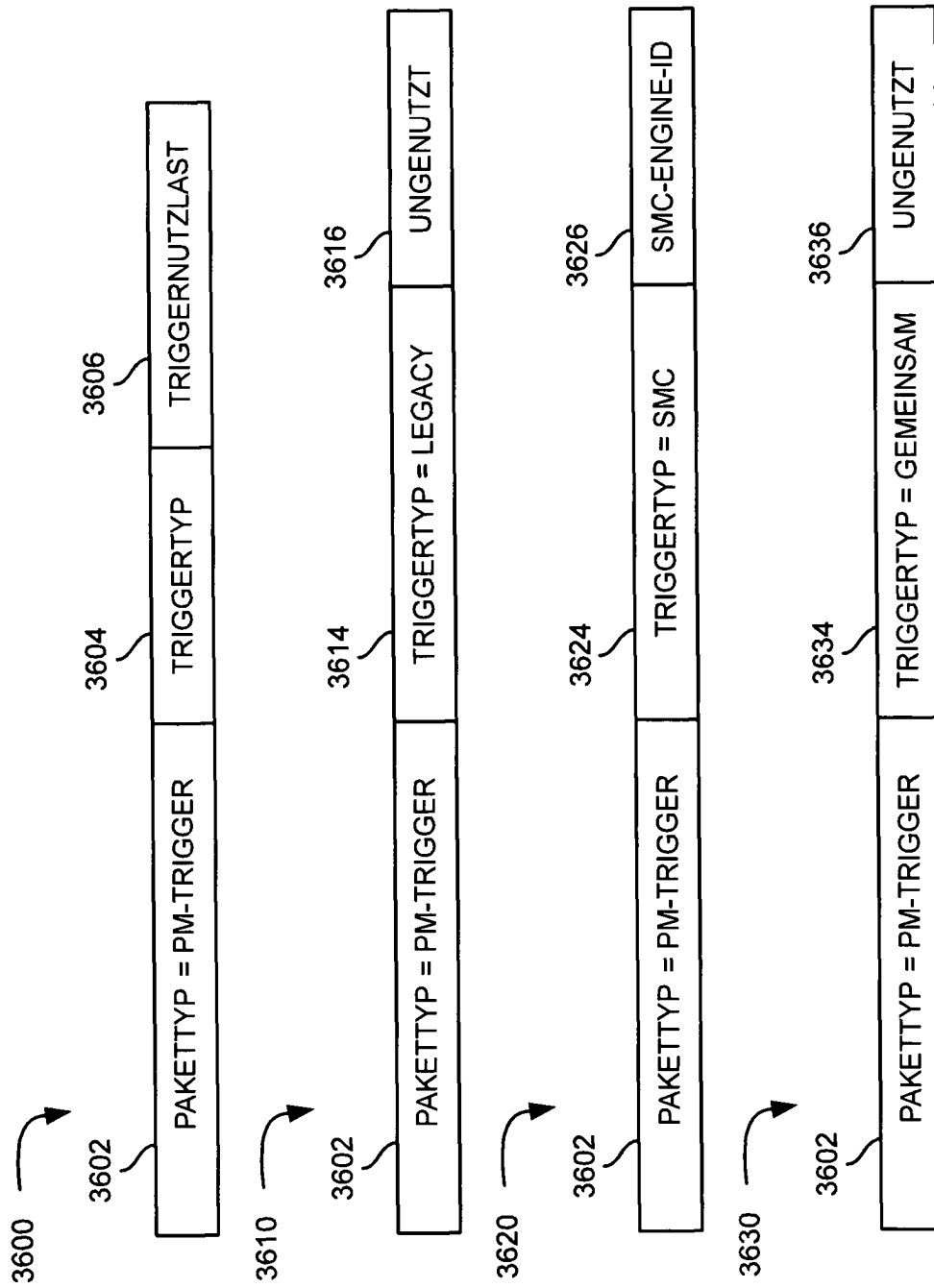
FIGUR 34A



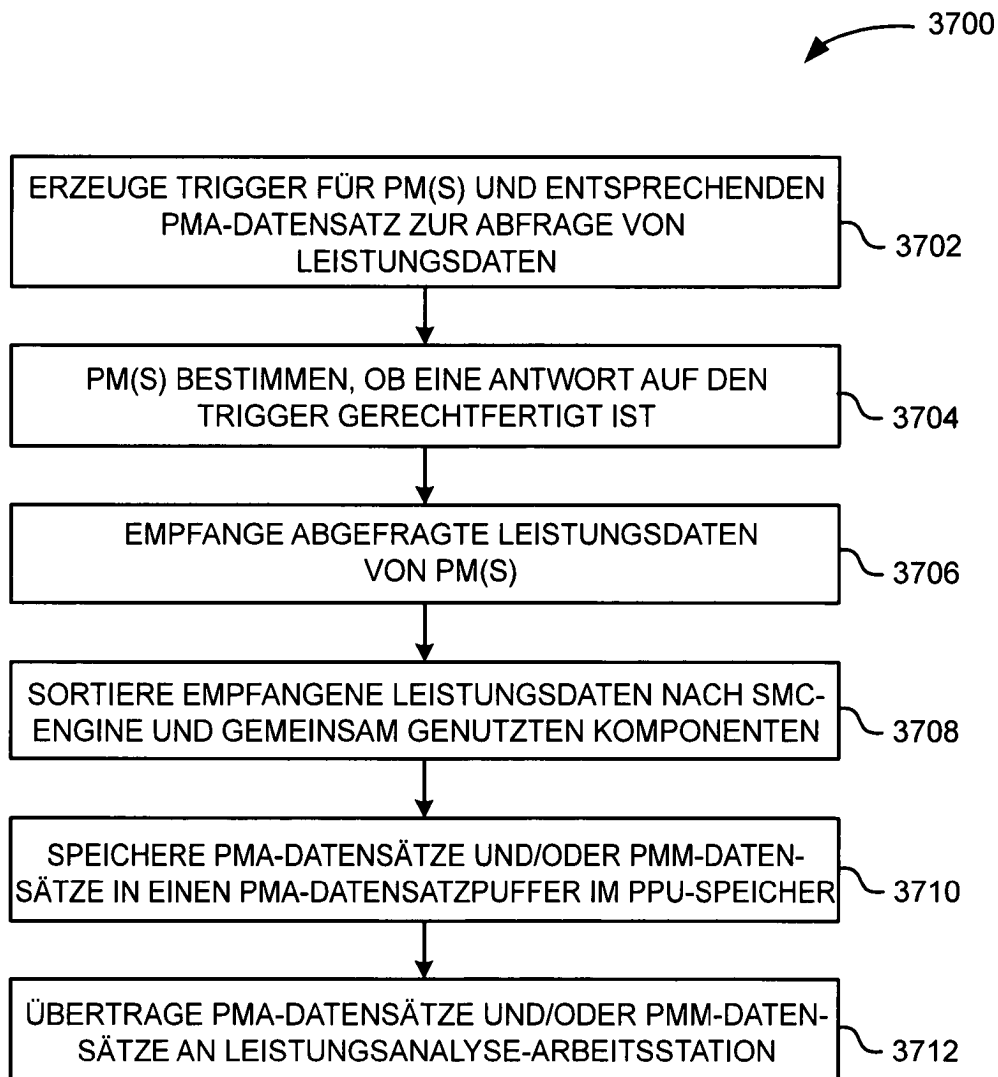
FIGUR 34B



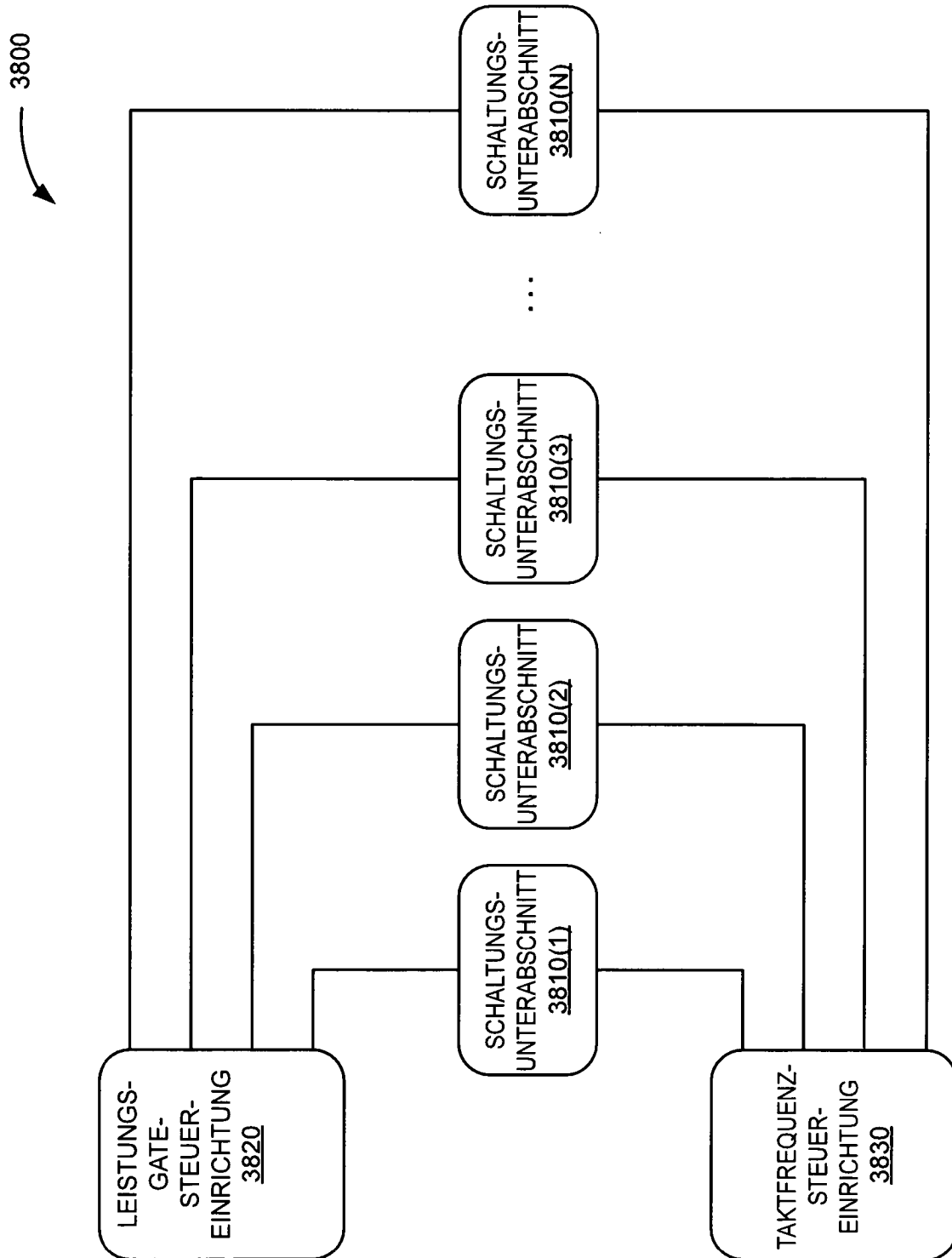
FIGUR 35



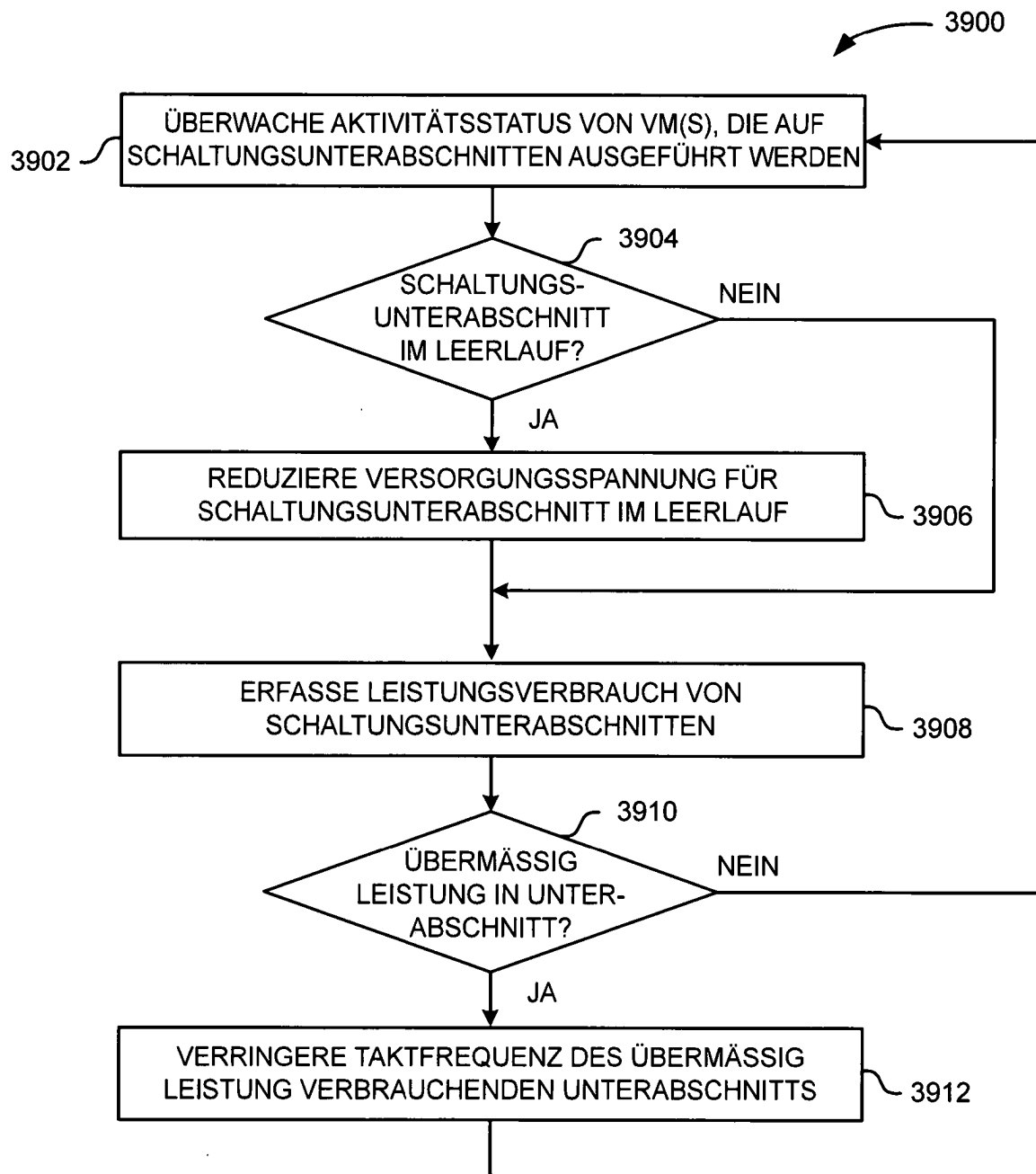
FIGUR 36



FIGUR 37



FIGUR 38



FIGUR 39