(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 698 22 534 T2** 2005.01.27

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 856 796 B1**
(21) Deutsches Aktenzeichen: **698 22 534.1**
(96) Europäisches Aktenzeichen: **98 101 562.1**
(96) Europäischer Anmeldetag: **29.01.1998**
(97) Erstveröffentlichung durch das EPA: **05.08.1998**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **24.03.2004**
(47) Veröffentlichungstag im Patentblatt: **27.01.2005**

(51) Int Cl.[7]: **G06F 12/08**
G06F 12/10, G06F 12/02

(30) Unionspriorität:
794172 **03.02.1997** **US**

(84) Benannte Vertragsstaaten:
DE, FR, GB

(73) Patentinhaber:
Compaq Computer Corp., Houston, Tex., US

(74) Vertreter:
Andrae Flach Haug, 83022 Rosenheim

(72) Erfinder:
Scales, Daniel J., Palo Alto, California 94306, US; Gharachorloo, Kourosh, Menlo Park, California 94025, US; Aggarwal, Anshu, Mountain View, US

(54) Bezeichnung: **Gemeinsame Speicherbenutzung mit variablen Blockgrössen für symmetrische Multiporzessor-Gruppen**

Beschreibung

Gebiet der Erfindung

[0001] Die vorliegende Erfindung bezieht sich allgemein auf symmetrische Multiprozessoren und insbesondere auf die gemeinsame Nutzung von Daten zwischen symmetrischen Multiprozessoren.

Hintergrund der Erfindung

[0002] Verteilte Computersysteme weisen typischerweise mehrere Computer auf, die über ein Kommunikationsnetzwerk miteinander verbunden sind. Bei manchen verteilten Computersystemen können die vernetzten Computer auf gemeinsame Daten zugreifen. Solche Systeme werden manchmal auch als parallele Computer bezeichnet. Wenn eine große Anzahl von Computern vernetzt ist, wird das verteilte System als "massiv" parallel bezeichnet. Ein Vorteil von massiv parallelen Computern ist, dass sie komplexe Rechenaufgaben in vertretbarer Zeit lösen können.

[0003] Bei solchen Systemen werden die Speicher der Computer kollektiv als verteilter gemeinsamer Speicher (Distributed Shared Memory/DSM) bezeichnet. Dabei besteht insofern ein Problem, weil nämlich garantiert werden muss, dass die im verteilten gemeinsamen Speicher gespeicherten Daten in kohärenter Weise abgerufen werden. Kohärenz bedeutet teilweise, dass nur ein Prozessor einen bestimmten Teil der Daten zu einer bestimmten Zeit modifizieren kann, weil sonst der Zustand des Systems nicht deterministisch wäre.

[0004] In Yeung D., "MGS: A Multigrain Shared Memory System" ("Gemeinsames Speichersystem mit mehreren Granularitäten"), Association for Computing Machinery, Bd. 24, Nr. 2, 1996 ist ein verteiltes skalierbares gemeinsames Speicher-Multiprozessorsystem beschrieben. Die Konstruktion dieses gemeinsamen Speichersystems verwendet multiple Granularitäten einer gemeinsamen Nutzung.

[0005] **Fig.** 1 zeigt ein typisches verteiltes gemeinsames Speichersystem **100**, das mehrere Computer **110** aufweist. Jeder Computer **110** enthält einen Einzelprozessor **101**, einen Speicher **102** und Eingabe/Ausgabe(Input/Output/I/O)-Schnittstellen **103**, die über einen Bus **104** miteinander verbunden sind. Die Computer sind über ein Netzwerk **120** miteinander verbunden. Hierbei bilden die Speicher **102** der Computer **110** den gemeinsamen Speicher.

[0006] In letzter Zeit wurden verteilte gemeinsame Speichersysteme als ein Cluster symmetrischer Multiprozessoren (SMP) aufgebaut. Bei SMP-Systemen kann der gemeinsam genutzte Speicher in wirkungsvoller Weise in Hardware implementiert werden, da die Prozessoren symmetrisch sind, z. B. in ihrem Aufbau und Betrieb identisch, und auf einem einzigen gemeinsamen Prozessorbus betrieben werden. SMP-Systeme haben ein gutes Preis/Leistungs-Verhältnis mit vier oder acht Prozessoren. Aufgrund des speziell gefertigten Busses ist es jedoch schwierig, die Größe eines SMP-Systems über zwölf oder sechszehn Prozessoren hinaus zu skalieren.

[0007] In Scales, D. J. et al., "Shasta: a low overhead, software-only approach for supporting fine-grain shared memory" ("Shasta: ein kostengünstiges, nur in Software implementiertes Verfahren zur Unterstützung eines gemeinsam genutzten Speichers feiner Granularität"), Sigplan Notices, Bd. 31, Nr. 9, 1996, ist ein verteiltes, gemeinsam genutztes Speichersystem beschrieben, das einen gemeinsamen Adressraum in Software auf Clustern von Computern mit physisch verteiltem Speicher unterstützt. Ein besonderes Merkmal des Shasta-Systems ist, dass die gemeinsam genutzten Daten in einer feinen Granularität kohärent gehalten werden können und die Kohärenz-Granularität sich in einer einzigen Anwendung über verschiedene gemeinsam genutzte Datenstrukturen verändern kann.

[0008] Es ist wünschenswert, große verteilte gemeinsam genutzte Speichersysteme unter Verwendung symmetrischer Multiprozessoren aufzubauen, die über ein Netzwerk miteinander verbunden sind. Das Ziel ist dabei, dass die Prozesse die Speicher in einer wirkungsvollen Weise nutzen, so dass von einem auf einem ersten SMP ausgeführten Prozess von einem mit einem zweiten SMP verbundenen Speicher abgerufene Daten sofort für alle auf dem ersten SMP ausgeführten Prozesse zur Verfügung stehen.

[0009] Bei den meisten bestehenden verteilten gemeinsamen Speichersystemen signalisiert eine Logik der virtuellen Speicherhardware (Paging) typischerweise, wenn ein Prozess versucht, auf gemeinsame Daten zuzugreifen, die nicht im Speicher des lokalen SMP gespeichert sind, auf dem der Prozess ausgeführt wird. In dem Fall, wo die Daten nicht lokal verfügbar sind, werden die Funktionen der Paging-Fehlerhandhabungsroutinen durch Softwareroutinen ersetzt, die Nachrichten mit auf entfernten Prozessoren ausgeführten Prozessen austauschen.

[0010] Bei dieser Vorgehensweise besteht das Hauptproblem darin, dass die Datenkohärenz nur mit großen (groben) Quantitäten vorgesehen werden kann, da typische virtuelle Speicherseiteneinheiten 4 KB oder 8 KB betragen. Diese Größe kann mit viel kleineren Dateneinheiten, zum Beispiel 32 oder 64 Bytes, auf die von vielen Prozessen zugegriffen werden, inkonsistent sein. Eine Granularität mit einer großen Seitengröße erhöht den Netzverkehr und kann zu einer Verringerung der Systemleistung füh-

ren.

[0011] Außerdem nutzen auf dem gleichen SMP ausgeführte multiple Prozesse typischerweise zusammen Zustandsinformation über gemeinsam genutzte Daten. Es können daher potentiell Wettlaufsituationen (Racing) entstehen. Eine Wettlaufsituation entsteht dann, wenn der Zustand des Systems davon abhängt, welcher Prozess zuerst abgeschlossen wird. Wenn zum Beispiel viele Prozesse Daten an eine identische Adresse schreiben können, werden von der Adresse gelesene Daten von der Ausführungsreihenfolge der Prozesse abhängen. Die Reihenfolge kann jedoch mit Laufzeitbedingungen variieren. Wettlaufsituationen können dadurch vermieden werden, dass dem Prozess vorgeschaltete Synchronisationsüberprüfungen, wie zum Beispiel Sperren oder Flags, hinzugefügt werden. Eine explizite Synchronisation erhöht jedoch die Kosten und kann dazu führen, dass das System nicht mehr praktisch zu implementieren ist.

[0012] Es ist wünschenswert, der Datentransfereinheit zwischen den symmetrischen Multiprozessoren zu erlauben, je nach der Größe der zugegriffenen Datenstrukturen zu variieren. Eine Kohärenzsteuerung für große Datenstrukturen sollte die Übertragung großer Dateneinheiten erlauben, so dass die zum Transfer der Daten benötigte Zeit amortisiert werden kann. Eine Kohärenz für kleinere Datenstrukturen sollte den Transfer kleiner Dateneinheiten erlauben. Außerdem sollte möglich sein, kleine Kohärenzeinheiten für große Datenstrukturen zu verwenden, bei denen eine falsche gemeinsame Nutzung auftreten kann. Eine falsche gemeinsame Nutzung (False Sharing) ist eine Situation, die auftritt, wenn unabhängige Datenelemente, auf die von unterschiedlichen Prozessen zugegriffen wird, in einer kohärenten Dateneinheit gespeichert sind.

Zusammenfassung der Erfindung

[0013] Ein in Software implementiertes Verfahren ermöglicht eine gemeinsame Nutzung von Daten zwischen symmetrischen Multiprozessoren, die ein verteiltes gemeinsames Speichersystem verwenden, indem Datenquantitäten mit variabler Größe verwendet werden. Beim verteilten gemeinsamen Speichersystem sind die symmetrischen Multiprozessoren über ein Netzwerk miteinander verbunden. Jeder symmetrische Multiprozessor enthält mehrere identische Prozessoren, einen Speicher mit Adressen und eine E/A-Schnittstelle zur Verbindung der symmetrischen Multiprozessoren über das Netzwerk.

[0014] Die Erfindung besteht in ihrer weitest gefassten Form aus einem Verfahren zum gemeinsamen Zugreifen auf in den Speichern symmetrischer Multiprozessoren in einem Computersystem gespeicherte Daten, nach Anspruch 1.

[0015] Wie hiernach beschrieben wird, wird ein Satz der Adressen der Speicher kollektiv als virtuelle gemeinsame Adressen zum Speichern gemeinsamer Daten bezeichnet. Die gemeinsamen Daten können durch die Befehle von Programmen abgerufen werden, die auf einem beliebigen der Prozessoren der symmetrischen Multiprozessoren als Prozesse ablaufen. Ein Teil der virtuellen gemeinsamen Adressen wird zum Speichern einer gemeinsamen Datenstruktur zugewiesen, die von den Prozessen als einer oder mehrere Blöcke verwendet wird. Die Daten werden auf der Ebene einzelner Blöcke abgerufen und kohärent gehalten.

[0016] In einer bevorzugten Ausführungsform der Erfindung kann die Größe eines bestimmten zugewiesenen Blocks für eine bestimmte gemeinsame Datenstruktur variieren. Jeder Block enthält eine ganzzahlige Anzahl von Zeilen, und jede Zeile enthält eine vorbestimmte Anzahl von Bytes gemeinsamer Daten.

[0017] Verzeichnisinformation (Directory-Information) eines bestimmten Blocks kann in einem Verzeichnis (Directory) im Speicher eines Prozessors gespeichert werden, der als der "Heim"-Prozessor (Home Processor) bezeichnet wird. Zugewiesene Blöcke werden den verschiedenen Prozessoren in zyklischer Weise zugeteilt. Die Verzeichnisinformation enthält die Größe des bestimmten Blocks, die Identität des Prozessors, der den Block zuletzt modifiziert hat, und die Identitäten aller Prozessoren, die eine Kopie des Blocks haben.

[0018] Vor der Ausführung werden die Programme vorzugsweise statisch analysiert, um Speicherzugriffsbefehle, wie zum Beispiel Lade- und Speicherbefehle, zu lokalisieren. Die Programme werden durch Einfügen zusätzlicher Befehle in die Programme instrumentiert. Die zusätzlichen Befehle können dynamisch überprüfen, ob die Zieladresse von Lade- und Speicherbefehlen auf eine bestimmte Zeile der gemeinsam genutzten Datenstruktur zugreift, und ob die Daten an der Zieladresse einen gültigen Zustand haben.

[0019] Wenn die Daten ungültig sind, wird eine Zugriffsanforderung erzeugt. In Reaktion auf den Empfang der Zugriffsanforderung von einem Anfordernden der Prozessoren wird ein bestimmter die bestimmte Zeile enthaltender Block und die Größe des bestimmten Blocks an den anfordernden Prozessor gesendet. Der Block wird über das Netzwerk gesendet. Hierdurch wird es den symmetrischen Multiprozessoren ermöglicht, gemeinsame Datenstrukturen, die in Blocks verschiedener Größe gespeichert sind, über das Netzwerk auszutauschen.

Kurzbeschreibung der Zeichnungen

[0020] Ein detaillierteres Verständnis der Erfindung geht aus der folgenden Beschreibung einer bevorzugten Ausführungsform hervor, die als Beispiel angeführt ist und anhand der beiliegenden Zeichnungen zu verstehen ist. Es zeigt:

[0021] **Fig.** 1 ein auf einem Uni-Prozessor basierendes verteiltes gemeinsames Speichersystem des Standes der Technik;

[0022] **Fig.** 2 ein Blockdiagramm eines auf symmetrischen Multiprozessoren basierenden verteilten gemeinsamen Speichersystems gemäß einer bevorzugten Ausführungsform der vorliegenden Erfindung;

[0023] **Fig.** 3 ein Fließdiagramm eines Prozesses zum Instrumentieren von Programmen;

[0024] **Fig.** 4 ein Blockdiagramm von Optimierungsschritten;

[0025] **Fig.** 5 ein Blockdiagramm einer Speicherpartitionierung;

[0026] **Fig.** 6 ein Diagramm eines optimierten Speicher-Fehlzugriffs-Prüfcodes;

[0027] **Fig.** 7 ein Diagramm eines Fehlzugriffs-Prüfcodes, der zu einer optimalen Planung (Scheduling) angeordnet ist;

[0028] **Fig.** 8 ein Fließdiagramm eines Verfahrens zum Überprüfen ungültiger Daten auf einem Ladezugriff;

[0029] **Fig.** 9 ein Diagramm von Befehlen, die nach einem ungültigen Flag suchen;

[0030] **Fig.** 10 ein Blockdiagramm einer Ausschlusstabelle;

[0031] **Fig.** 11 ein Blockdiagramm eines Prozesses zur Überprüfung von Stapeln von Zugriffsbefehlen;

[0032] **Fig.** 12 ein Diagramm von Befehlen, die den Prozess von **Fig.** 11 implementieren und für ein optimales Scheduling angeordnet sind;

[0033] **Fig.** 13 ein Blockdiagramm eines Blockverzeichnisses; und

[0034] **Fig.** 14 ein Blockdiagramm von Datenstrukturen, die variable Granularitäten aufweisen.

Detaillierte Beschreibung der bevorzugten Ausführungsform

Systemüberblick

[0035] **Fig.** 2 zeigt ein zur Verwendung der Erfindung geeignetes Computersystem **200** mit einem auf symmetrische Multiprozessoren (SMP) verteiltem gemeinsamen Speicher (DSM). Das DSM-SMP-System **200** weist mehrere SMP-Systeme **210** auf, die über ein Netzwerk **220** miteinander verbunden sind. Jedes SMP-System **210** weist 2, 4, 8 oder mehr symmetrische Prozessoren **211** auf, die über einen Prozessorbus **209** miteinander verbunden sind. Außerdem kann jeder SMP **210** Speicher (M) **212** und Eingabe/Ausgabe-Schnittstellen (E/A) **214** aufweisen, die über einen Systembus **213** mit den symmetrischen Prozessoren **211** verbunden sind.

[0036] Die Speicher **212** können dynamische Speicher mit wahlfreiem Zugriff (Dynamic Random Access Memories/DRAM) sein. Die Speicher **212** können Hochgeschwindigkeits-Hardware-Caches haben, um die räumlichen und zeitlichen Orte von Daten ausnutzen zu können. Häufig benutzte Daten werden mit größerer Wahrscheinlichkeit im Cache gespeichert.

[0037] Die Speicher **212** speichern Programme **215** und Datenstrukturen **216**. Manche Adressen der Speicher **212** können kollektiv als ein einziger Satz gemeinsamer virtueller Adressen bezeichnet werden. Einige der Datenstrukturen können gemeinsame Daten einschließen. Gemeinsame Daten können von jedem beliebigen Prozess, der auf einen beliebigen der Prozessoren **211** eines beliebigen der SMPs **210** ausgeführt wird, unter der Verwendung der virtuellen Adressen abgerufen werden.

[0038] Die Busse **209** und **213** verbinden die Komponenten der SMPs **210** unter der Verwendung von Daten-, Adress- und Steuerleitungen. Das Netzwerk **220** verwendet Netzwerkprotokolle, zum Beispiel einen asynchronen Transfermodus (ATM) oder FDDI-Protokolle, zur Kommunikation von Nachrichten zwischen symmetrischen Multiprozessoren **210**. Alternativ dazu kann das Netzwerk **220** die Form eines Hochleistungs-Cluster-Netzwerks haben, wie zum Beispiel eines Memory Channel, der von Digital Equipment Corporation hergestellt wird.

Allgemeiner Systembetrieb

[0039] Während des Betriebs des SMP-DSM-Systems **200** werden Befehle der Programme **215** durch die Prozessoren **211** als Ausführungsstränge oder -prozesse ausgeführt. Die Befehle können unter der Verwendung von Lade- und Speicherbefehlen auf die Datenstrukturen **216** zugreifen. Es ist wünschenswert, dass ein Beliebiges der Programme **215**, das

auf einem Beliebigen der Prozessoren **211** ausgeführt wird, auf eine Beliebige der gemeinsamen Datenstrukturen **216** zugreifen kann, die in einem Beliebigen der Speicher **212** abgelegt ist.

Instrumentierung

[0040] Vorzugsweise werden die Programme **215**, wie hier beschrieben, vor der Ausführung instrumentiert. Unter Instrumentierung versteht man einen Vorgang, der Zugriffsbefehle (Lade- und Speichervorgänge) in den Programmen **215** statisch lokalisiert. Die Instrumentierung lokalisiert auch Befehle, die Teile der Speicher **211** zuweisen und aberkennen.

[0041] Nachdem die Befehle lokalisiert wurden, können zusätzliche Befehle, z. B. Fehlzugriffs-Prüfcode, vor den Zugriffsbefehlen in die Programme eingefügt werden, um sicherzustellen, dass Speicherzugriffe korrekt ausgeführt werden. Der Fehlzugriffs-Prüfcode wird optimiert, um den zusätzlichen Aufwand (Overhead) zu minimieren, der zur Durchführung der zusätzlichen Befehle benötigt wird. Die zusätzlichen Befehle, die für Zuweisungs- und Aberkennungsbefehle eingefügt werden, unterhalten Kohärenzsteuerinformation, wie zum Beispiel die Größe der zugewiesenen Blöcke.

[0042] Wie oben erwähnt, können die Programme **215** einen Teil der Adressen der verteilten Speicher **212** als gemeinsamen Speicher betrachten. Für bestimmte Zieladressen des gemeinsamen Speichers kann ein Befehl auf eine lokale Kopie der Daten zugreifen, oder es muss eine Nachricht an einen anderen Prozessor gesendet werden, um eine Kopie der Daten anzufordern.

Zugriffszustände

[0043] Bei einem beliebigen SMP können die im gemeinsamen Speicher abgelegten Daten zwei mögliche Zustände einnehmen: ungültig (Invalid) oder gültig (Valid). Der gültige Zustand kann die Unterzustände "gemeinsam" (Shared) oder "exklusiv" (Exclusive) haben. Wenn der Zustand der Daten ungültig ist, ist ein Zugriff auf die Daten nicht erlaubt. Wenn der Zustand gemeinsam ist, existiert eine lokale Kopie und andere SMPs haben ebenfalls eine Kopie. Wenn der Zustand exklusiv ist, hat nur ein SMP eine gültige Kopie der Daten, und keine anderen SMPs können auf die Daten zugreifen. Außerdem können die Daten, wie unten beschrieben, auch in einem Übergangszustand bzw. "schwebend" (Pending) sein.

[0044] Die Zustände der Daten werden durch über das Netzwerk **220** mitgeteilte Kohärenzsteuernachrichten gepflegt. Die Nachrichten werden durch Prozeduren erzeugt, die vom Fehlzugriffs-Prüfcode der instrumentierten Programme aufgerufen werden.

[0045] Daten können nur dann direkt vorn Speicher eines lokalen SMP aufgerufen werden, wenn die Daten einen gemeinsamen oder exklusiven Zustand haben. Daten können nur dann im lokalen Speicher gespeichert werden, wenn der Zustand exklusiv ist. Eine Kommunikation wird benötigt, wenn ein Prozessor versucht, Daten zu laden, die in einem ungültigen Zustand sind oder wenn ein Prozessor versucht, Daten zu speichern, die in einem ungültigen oder gemeinsamen Zustand sind. Diese Zugriffe, die Kommunikationen erfordern, werden als Fehlzugriffe (Misses) bezeichnet.

[0046] Die Adressen der Speicher **212** können dynamisch zum Speichern gemeinsamer Daten zugewiesen werden. Einige der Adressen können statisch zugewiesen werden, um private Daten zu speichern, auf die nur von Prozessen zugegriffen werden kann, die auf einem lokalen Prozessor ausgeführt werden. Zusätzlicher Aufwand kann dadurch verringert werden, dass einige der Adressen für private Daten reserviert werden, da Zugriffe auf die privaten Daten durch den lokalen Prozessor nicht auf Fehlzugriffe überprüft werden müssen.

[0047] Wie bei einem durch Hardware gesteuerten gemeinsamen Speichersystem sind die Adressen der Speicher **212** in zuweisbare Blöcke partitioniert. Alle Daten innerhalb eines Blocks werden als eine kohärente Einheit abgerufen. Als ein Merkmal des Systems **200** können Blöcke für unterschiedliche Bereiche von Adressen variable Größen haben. Zum Vereinfachen des unten beschriebenen Fehlzugriffs-Prüfcodes sind die variabel abgemessene Blöcke weiter in Adressbereiche fester Größe, die als "Zeilen" (lines) bezeichnet werden, partitioniert.

[0048] Zustandsinformation wird in Zustandstabellen zeilenweise unterhalten. Die Größe der Zeile, typischerweise 32, 64 oder 128 Bytes, wird zu der Zeit bestimmt, zu der ein bestimmtes Programm **215** instrumentiert wird. Ein Block kann eine ganzzahlige Anzahl von Zeilen enthalten.

[0049] Während des Betriebs des Systems **200** stellt vor der Ausführung eines Speicherzugriffbefehls der Fehlzugriffs-Prüfcode fest, ob die Zieladresse in einem privaten Speicher ist. Wenn die Zieladresse in einem privaten Speicher ist, dann kann der Fehlzugriffs-Prüfcode sofort abgeschlossen werden, da private Daten immer von einem lokalen Prozessor abgerufen werden können. Ansonsten berechnet der Fehlzugriffs-Prüfcode, welche Zeile eines bestimmten Blocks die Zieladresse des Befehls enthält und stellt fest, ob die Zeile für den Zugriff im korrekten Zustand ist. Wenn der Zustand nicht korrekt ist, dann wird eine Fehlzugriffs-Behandlungsroutine aufgerufen, um die Daten aus dem Speicher eines entfernten SMP abzurufen.

Instrumentierungsvorgang

[0050] **Fig.** 3 zeigt ein Fließdiagramm eines Vorgangs **300**, der zum Instrumentieren von Programmen so verwendet werden kann, dass der für die zusätzlichen Befehle benötigte zusätzliche Aufwand verringert wird. Außerdem erlaubt der Vorgang **300** eine Kohärenzsteuerung für Datenquantitäten variabler Größe, auf die von symmetrischen Multiprozessoren zugegriffen wird. Der Vorgang **300** enthält ein Analysier-Modul **320**, ein Optimier-Modul **330** und einen Generator **340** für ein ausführbares Image.

[0051] Maschinenausführbare Programme **310** werden einem Analysier-Modul **320** vorgelegt. Das Analysier-Modul **320** unterteilt die Programme **310** in Prozeduren **301** und die Prozeduren **301** in grundlegende Ausführungsblöcke **302**. Ein grundlegender Block **302** ist als ein Satz von Befehlen definiert, die alle ausgeführt werden, wenn der erste Befehlt des Satzes ausgeführt wird. Die Befehle von Prozeduren und den grundlegenden Blöcken werden analysiert, um die Programm-Aufruf- und -Fließ-Graphen **303** zu bilden.

[0052] Die Graphen **303** können verwendet werden, um einen Daten- und Ausführungsfluss der Programme **310** zu erstellen. Die grundlegenden Blöcke und Graphen **303** werden analysiert, um Befehle zu lokalisieren, die Speicheradressen zuteilen und Zugriffe auf zugewiesene Adressen durchführen. Wenn ein Befehl auf gemeinsame Teile der Speicher **212** zugreift, wird ein Fehlzugriffs-Prüfcode eingefügt, um sicherzustellen, dass der Zugriff in einer kohärenten Weise erfolgt.

[0053] Der Fehlzugriffs-Prüfcode wird durch das Optimier-Modul **330** eingefügt, wie unten eingehend beschrieben ist. Nachdem die Programme **310** instrumentiert wurden, erzeugt der Image-Generator **340** ein modifiziertes maschinenausführbares Image **350**. Das modifizierte Image **350** enthält instrumentierte Programme **351** mit Fehlzugriffs-Prüfcode, Fehlzugriffs-Behandlungs-Protokollprozeduren **352** und einer Nachrichten-Weiterleitungsbibliothek **353**. Das Image **350** kann die Programme **310** ersetzen.

[0054] **Fig.** 4 zeigt die Schritte, die vom Optimier-Modul **330** von **Fig.** 3 ausgeführt werden. Diese Schritte enthalten Speicherpartitionierungs- **410**, Registeranalyse-**420**, Codescheduling- **430**, Ladeprüfanalyse- **440** und Stapelbildungs(Batching)-Schritte **450**.

Speicheraufbau

[0055] **Fig.** 5 zeigt eine Zuweisung von Adressen zu den Speichern **212** von **Fig.** 2. Die Adressen werden in **Fig.** 5 von unten nach oben größer. Die Adressen werden für Stapel **510**, Programmtext **520**, statisch zugewiesene private Daten **530**, Zustandstabellen **540** und dynamisch zugewiesene gemeinsame Daten **550** reserviert.

[0056] Während des Betriebs verringern sich die von den Stapeln **510** verwendeten Adressen zum Stapelüberlaufbereich **505** hin. Der Textraum **520** wird zum Speichern der ausführbaren Befehle, z. B. des Images **350** von **Fig.** 3, verwendet. Die für Text zugewiesenen Adressen steigen zum Textüberlaufbereich **525** hin an. Die Adressen des privaten Datenabschnitts **530** werden zum Speichern von Datenstrukturen verwendet, die exklusiv von einem einzigen lokalen Prozessor verwendet werden, z. B. werden die Daten nicht gemeinsam genutzt. Die Adressen in diesem Teil des Speichers werden statisch zugewiesen, wenn ein bestimmtes Programm zur Ausführung geladen wird.

Zustandstabellen (State Tables)

[0057] Die Zustandstabellen **540** enthalten eine gemeinsame Zustandstabelle **541**, private Zustandstabellen **542** und Ausschlusstabellen **1000**. Die Ausschlusstabellen **1000** können auch einen gemeinsam **1001** und einen privaten **1002** Teil enthalten.

[0058] Die gemeinsamen und privaten Zustandstabellen **541** enthalten für jede Zeile zugewiesene Adressen ein Byte gemeinsamer bzw. privater Zustandseinträge **545**. Die Bits der Zustandseinträge **545** können dazu verwendet werden, die verschiedenen Zustände der entsprechenden Datenzeile anzuzeigen. Eine oder mehrere Datenzeilen bilden einen Block.

[0059] Gemäß der bevorzugten Umsetzung können alle Prozessoren **211** eines bestimmten SMP **210** die gleichen Daten gemeinsam nutzen. Daher werden die Zustandstabelleneinträge **545** für alle Prozessoren des SMP **210** gemeinsam genutzt. Dies bedeutet, dass beim Abrufen eines Blocks, z. B. eine oder mehrere Datenzeilen, von einem entfernten SMP und beim Ändern des Zustands des Blocks von ungültig auf gemeinsam oder exklusiv die gemeinsam genutzte Speicherhardware des SMP den Zustand der Daten erkennt und ein beliebiger Prozessor **211** des SMP auf die neuen Daten zugreifen kann.

[0060] Da es sein kann, dass mehr als ein Prozessor des bestimmten SMP gleichzeitig auf einen Zustandstabelleneintrag zuzugreifen versucht, wird der Eintrag gesperrt, bevor ein Zugriff auf den Eintrag erfolgt. Die Fehlzugriffsprüfungen, die im Code eingefügt wurden, können auch einen Zugriff auf den Zustandstabelleneintrag erfordern. In diesem Fall wird jedoch der Eintrag nicht gesperrt, um den zusätzlichen Aufwand gering zu halten. Stattdessen unterhält jeder Prozessor eine entsprechende private Zustandstabelle **542**, auf die durch einen vorgeschalte-

ten Code ohne zusätzlichen Aufwand zugegriffen werden kann.

[0061] Die Einträge der privaten Zustandstabellen **542** der Prozessoren werden durch zwei unterschiedliche Mechanismen aktualisiert.

[0062] In dem Fall, in dem ein Prozessor versucht, auf ungültige Daten zuzugreifen, wird ein Fehlzugriffszustand eintreten, und die Daten werden von einem entfernten SMP abgerufen. Nach Empfang wird der Zustand der Daten gültig. Dies wird als "Höherstufen" (Upgrade) des Zustands bezeichnet, da die Daten nun verfügbar sind, während sie vorher nicht verfügbar waren. Die Daten sind jedoch in den privaten Zustandstabellen der anderen Prozessoren auf dem gleichen SMP **210** immer noch als ungültig markiert.

[0063] Wenn einer dieser anderen Prozessoren nun versucht, auf die Daten zuzugreifen, werden die anderen Prozessoren in ihren privaten Zustandstabellen **542** immer noch einen ungültigen Zustand vorfinden. Der andere Prozessor kann eine Sperrung der gemeinsamen Zustandstabelle **541** erlangen und feststellen, dass die Daten für den lokalen SMP gültig sind und seine private Zustandstabelle **542** entsprechend aktualisieren. Nachfolgende Zugriffe auf Daten können dann durchgeführt werden, ohne dass auf die gemeinsame Zustandstabelle **541** zugegriffen werden muss.

[0064] In dem Fall, wo der Zustand der Daten auf ungültig zurückgesetzt werden muss, z. B. braucht ein Prozessor eines anderen SMP die Daten, wird der Zustand der Daten "herabgestuft" (Downgrade). In diesem Fall sendet der die Anforderung empfangende Prozessor eine interne Nachricht an andere auf dem lokalen SMP tätige Prozessoren, so dass der in ihren privaten Zustandstabellen **542** unterhaltene Zustand herabgestuft werden kann. Dieses "Herabstufen" einer Zeile ist erst dann abgeschlossen, bis alle Prozessoren ihre privaten Zustandstabellen geändert haben.

[0065] Es wird darauf hingewiesen, dass eine Wettlaufsituation entstehen kann, wenn der die Invalidierungsanfrage empfangende Prozessor alle privaten Zustandstabellen aller Prozessoren des lokalen SMPs direkt ändern würde. Zum Beispiel würde eine Wettlaufsituation dadurch entstehen, dass ein erster Prozessor einen gültigen Zustand sieht, während er die vorgeschaltete Überprüfung für einen Speichervorgang durchführt, ein zweiter Prozessor jedoch den Zustand der Daten auf ungültig herabstufen würde, bevor der erste Prozessor die Möglichkeit hat, die modifizierten Daten zu speichern.

[0066] Eine Möglichkeit zur Vermeidung von Wettlaufsituationen besteht darin, Zustandstabellensperren mit dem vorgeschalteten Fehlzugriffs-Prüfcode zu erlangen. Dies lässt jedoch den zusätzlichen Aufwand aufgrund der Sperrung anwachsen. Dies trifft insbesondere auf Prozessoren mit einem entspannten Speichermodell zu, wie zum Beispiel einen Alpha-Prozessor, der von Digital Equipment Corporation hergestellt wird. Daher ist die Verwendung privater Zustandstabellen zum wirkungsvollen Vermeiden von Wettlaufsituationen wichtig.

[0067] Die Verwendung privater Zustandstabellen **542** vermeidet nicht nur Wettlaufsituationen im Fehlzugriffs-Prüfcode, sondern verringert auch die Anzahl von Nachrichten, die weitergeleitet werden müssen, während der Zustand von Daten innerhalb eines SMP **210** herabgestuft wird. Wenn zum Beispiel ein lokaler Prozessor niemals auf Daten zugreift, die innerhalb eines lokalen SMPs gültig sind, dann braucht seine private Zustandstabelle nicht aktualisiert zu werden.

Gemeinsame Daten

[0068] Die Adressen der gemeinsamen Daten **550** werden durch die Programme während der Ausführung dynamisch zugewiesen. Ein Vorteil hierbei ist, dass die Adressen der gemeinsamen Daten **550** in Blöcken variabler Größe **551** zugewiesen werden können. Die Blöcke werden weiter in Zeilen **552** unterteilt.

[0069] Bei dem in **Fig.** 5 gezeigten Aufbau brauchen nicht alle Zugriffsbefehle instrumentiert zu werden. Zum Beispiel sind in den Programmstapeln **510** gespeicherte Daten nicht gemeinsam genutzt. Daher brauchen nicht alle Befehle, die das Stapelzeigerregister (SP) als Basis verwenden, mit einem Fehlzugriffs-Prüfcode versehen zu werden. Außerdem brauchen nicht alle Befehle instrumentiert zu werden, die unter der Verwendung eines Privatdaten-Zeigerregisters (PR) auf private Daten **530** zugreifen.

Registergebrauch

[0070] Das Analysier-Modul **320** von **Fig.** 3 verwendet die Graphen **303** und Datenflussanalyse zum Nachverfolgen des Inhalts von Allzweckregistern zum Feststellen, ob in den Registern gespeicherte Werte aus Adressen stammen, die auf dem SP oder PR-Register basieren. Dann braucht nämlich ein auf den Stapel oder die privaten Daten über eine abgeleitete Adresse zugreifender Befehl nicht instrumentiert zu werden. Das Analysier-Modul **320** kann auch alle Register lokalisieren, die zu der Zeit frei sind, zu der der Fehlzugriffs-Prüfcode anzuwenden ist, wodurch sich die Notwendigkeit des Speicherns und erneuten Speicherns der vom Fehlzugriffs-Prüfcode verwendeten Register erübrigt.

[0071] Durch Starten der privaten Zustandstabelle

540 bei der Adresse 0x2000000000 im privaten Adressraum eines jeden Prozessors kann eine Verschiebung der Zielzugriffsadresse direkt die Adresse des entsprechenden Eintrags **545** in der privaten Zustandstabelle **540** erzeugen. Auch wenn der in **Fig.** 5 gezeigte Aufbau der Adressen für einen Prozessor mit 64-Bit-Adressierung gedacht ist, versteht es sich, dass der Aufbau **500** für Prozessoren mit 32-Bit- und anderen Adressierungen modifiziert werden kann.

Optimierter Fehlzugriffs-Prüfcode

[0072] **Fig.** 6 zeigt den Fehlzugriffs-Prüfcode **600**, der für den Speicheraufbau von **Fig.** 5 optimiert ist. Die Zieladresse für einen Zugriff kann durch den Befehl **601** festgestellt werden. Wenn die Ziel-Basisadresse jedoch schon in einem Register zum Beispiel durch einen zuvor ausgeführten Lade- oder Speicherbefehl eingerichtet wurde, ist der Befehl **601**, der die Ziel-Basisadresse lädt, nicht erforderlich.

[0073] Der Verschiebungsbefehl **602** stellt fest, ob die Zieladresse innerhalb des gemeinsamen Datenbereichs **550** ist. Der Verzweigungsbefehl **603** geht direkt zum Ausführen des ursprünglichen Speicherbefehls weiter, wenn dies nicht der Fall ist. Der Verschiebungsbefehl **604** erzeugt die Adresse des Eintrags in der Zustandstabelle, die der die Zieladresse enthaltenden Zeile entspricht. Dadurch, dass der Wert des Zustands durch eine Null auf "exklusiv" gesetzt wird, erübrigt sich die Notwendigkeit eines Vergleichs mit einem konstanten Wert. Stattdessen kann ein einfacher Sprungbefehl **607** zur Überprüfung eines Fehlzugriffs durchgeführt werden. Die Befehle **605** bis **606** rufen den Zustandstabelleneintrag ab. Der Fehlzugriffs-Behandlungscode **608** wird in dem Fall eines Fehlzugriffs ausgeführt, und der ursprüngliche Speicherbefehl wird bei **609** ausgeführt.

[0074] Der Fehlzugriffs-Prüfcode **600** erfordert nur die Ausführung von drei Befehlen, wenn die Zieladresse nicht im gemeinsamen Datenbereich ist. In dem Fall eines Zugriffs auf gemeinsame Daten müssen sieben Befehle ausgeführt werden.

Code-Scheduling

[0075] Im Schritt **430** von **Fig.** 4 können Befehls-Scheduling-Verfahren zur weiteren Verringerung des zusätzlichen Aufwands für den Fehlzugriffs-Prüfcode **600** verwendet werden. In modernen Prozessoren, die im Pipelining-Verfahren arbeiten und superskalar sind, kann der hinzugefügte Fehlzugriffs-Prüfcode in vielen Fällen so ausgelegt sein, dass er minimale Pipelineverzögerungen einführt und das Potential für eine Vielzahl von Befehlen erhöht, die während eines einzigen Prozessorzyklus ausgegeben werden.

[0076] Zum Beispiel ergibt sich in manchen Prozessoren nur eine Verzögerung von einem einzigen Zyklus, bevor das Ergebnis eines Verschiebungsvorgangs verwendet werden kann. Wenn daher der zweite Verschiebungsbefehl **604** von **Fig.** 6 vorgeschoben wird, um den Verzögerungsschlitz zu besetzen, der aus dem ersten Verschiebungsbefehl **702** resultiert, wird die Verzögerung zwischen dem verlegten zweiten Verschiebungsbefehl **703** und dem ldq_u-Befehl **705** ausgeschlossen. Dies bedeutet, dass der Code **700** in weniger Maschinenzyklen als der Code **600** abgeschlossen werden kann. Es wird darauf hingewiesen, dass sich wie beim Code **600** die Notwendigkeit für den Befehl **701** in vielen Fällen erübrigt. Die Befehle **705**– **707** laden und prüfen den Datenzustand.

[0077] Zur weiteren Verringerung zusätzlicher Kosten bei Mehrfachausgabeprozessoren können die Befehle des Fehlzugriffs-Prüfcodes **700** so platziert werden, dass sie während Pipelineverzögerungen im ursprünglichen ausführbaren Code oder gleichzeitig mit den Befehlen des ausführbaren Images ausgegeben werden. Es wird darauf hingewiesen, dass die Ausführung der ersten drei Befehle **701**–**703** in einem grundlegenden Befehlsblock vorgeschoben werden kann, solange die Register (r1 und r2) frei bleiben. In vielen Fällen können nämlich alle drei Befehle so weit vorgeschoben werden, dass sie den zusätzlichen Aufwand der Ausführung der Befehle vollständig verbergen. Daher ist es ganz klar von Vorteil, den Code wie in **Fig.** 7 gezeigt anzuordnen.

Speicherprüfung

[0078] Der Fehlzugriffs-Prüfcode kann weiter optimiert werden, wenn der Zugriffsbefehl ein Speicherbefehl **710** ist. In diesem Fall werden die ersten drei Befehle **701**–**703** vor dem Speicherbefehl **710** angeordnet. Die verbleibenden Befehle **704**–**707** werden nach dem Speicherbefehl **710** angeordnet. Diese Anordnung ist in Fällen vorteilhaft, wo Befehle mit langen Latenzen dem Speicherbefehl **710** unmittelbar vorausgehen, während das Programm den zu speichernden Wert berechnet. In diesem Fall muss der Speicherbefehl **710** warten, bis der Wert verfügbar wird. Daher kann der zusätzliche Aufwand beim Ausführen der vorgeschobenen Befehle vollständig verborgen werden.

Ladeprüfung

[0079] Wie in den **Fig.** 8 und 9 gezeigt, können die durch einen Ladebefehl geladenen Daten analysiert werden, um den zusätzlichen Aufwand für den Fehlzugriffs-Prüfcode weiter zu verringern. Immer wenn Daten einer Zeile ungültig werden, wird ein "Flag" **801** an allen der Zeile zugeordneten Adressen **810**–**811** gespeichert. Der Flag **801** ist zum Beispiel 0xFFFFFF03 (–253). Dann kann anstelle der Feststellung des Zustands einer Zeile über Zustandsta-

belleneinträge in fast allen Fällen der Zustand aus den geladenen Daten bestimmt werden.

[0080] Zum Beispiel werden die Daten an Zieladressen mit einem Ladebefehl **901** abgerufen, Schritt **820**. Bei Schritt **830** wird das Komplement **840** des Flags hinzugefügt, z. B. **253**. Bei Schritt **850** wird überprüft, ob die vom Speicher geladenen Daten aller Wahrscheinlichkeit nach einen ungültigen Zustand anzeigen. Wenn dies zutrifft, wird mit dem Fehlzugriffs-Prüfcode **870** fortgefahren, ansonsten wird mit Schritt **860** fortgefahren, der keinen Fehlzugriff darstellt. In dem Fall, wo ein vermuteter Fehlzugriff auftritt, kann der Fehlzugriffscode **870** dies durch Überprüfen des Eintrags für die Zeile in der Zustandstabelle **540** prüfen. Hierbei werden die seltenen Fälle behandelt, wo das Programm tatsächlich die dem Flag entsprechenden Daten verwendet.

[0081] Der Flag wird so gewählt, dass ein einzelner Befehl **902** zum Überprüfen nach ungültigen Daten verwendet werden kann. Hierbei ist es möglich, dass so gut wie jede Konstante verwendet werden kann. Es wird darauf hingewiesen, dass bei der Verwendung eines Werts Null zum Anzeigen eines ungültigen Zustands ein einfacher Sprungbefehl genügen würde. In Fällen, wo jedoch eine Null oder eine andere kleine ganze Zahl, z. B. –1, 0, +1, verwendet wird, wird der gemessene zusätzliche Aufwand des Fehlzugriffs-Prüfcodes anscheinend größer, weil die größere Anzahl falscher Fehlzugriffe bewältigt werden muss. In der Praxis treten bei der Verwendung des Flags 0xFFFFFF03 falsche Fehlzugriffe nur selten auf, daher verringert der optimierte Fehlzugriffs-Prüfcode **900**, wie er in **Fig.** 9 gezeigt ist, den Fehlzugriffs-Prüfcode für Ladebefehle beträchtlich, z. B. auf zwei Befehle.

[0082] Neben einer Verringerung des zusätzlichen Aufwands hat das Flagverfahren auch weitere Vorteile. Der Hauptvorteil besteht darin, dass die Notwendigkeit einer Überprüfung der Zustandstabelle in Fällen wegfällt, wo der Ladezugriff gültig ist. Außerdem geschieht das Laden der "Flag"-Daten von der Zieladresse und die Zustandsüberprüfung atomisch. Durch diese Atomizität werden mögliche Wettlaufsituationen zwischen dem Ladebefehl und Protokolloperationen für die gleiche Adresse ausgeschlossen, die auf einem anderen Prozessor des gleichen SMP auftreten können.

[0083] Das Flag-Prüfverfahren kann auch für Gleitkomma-Lade-Zugriffsbefehle verwendet werden. In diesem Fall lädt der Fehlzugriffs-Prüfcode die Daten der Zieladresse in ein Gleitkommaregister, gefolgt von einem Gleitkomma-Additions- und -Vergleichsvorgang. Auf manchen Prozessoren bringen Gleitkommabefehle jedoch lange Verzögerungen mit sich. Deswegen kann der Gleitkomma-Fehlzugriffscode dadurch optimiert werden, dass ein ganzzahliges Laden für die gleiche Zieladresse eingefügt wird und die Flagüberprüfung, wie oben für die **Fig.** 8 und 9 beschrieben, implementiert wird. Auch mit dem zusätzlichen Ladebefehl hat dieses Verfahren immer noch einen größeren Wirkungsgrad als die Überprüfung eines Eintrags der Zustandstabelle.

[0084] Alternativ dazu können die Gleitkommadaten direkt vom Gleitkommaregister an das ganzzahlige Register übertragen werden, wenn ein solcher Vorgang auf dem zugrundeliegenden Prozessor verfügbar ist.

[0085] Es versteht sich, dass das Befehlsscheduling auf die Befehle von **Fig.** 9 für Lade-Fehlzugriffs-Codeprüfungen angewendet werden kann. Bei einer bevorzugten Umsetzung versucht der Schedulingschritt **430** von **Fig.** 4, die Ausführung der Befehle **902** und **903** zu verzögern, um einen Pipelinestau (Stalling) zu vermeiden, wenn der Wert des Ladevorgangs zu verwenden ist.

Cache-Fehlzugriffe

[0086] Beim Laden von Einträgen aus der Zustandstabelle **540** können Fehlzugriffe in einem Cache eine potentielle Quelle von zusätzlichem Aufwand für den Fehlzugriffs-Prüfcode sein. Wenn das Programm eine gute räumliche Lokalität hat, dann werden beim Programm nicht viele Hardware-Cache-Fehlzugriffe auftreten. Wenn 64-Byte-Zeilen verwendet werden, dann ist der für die Zustandstabelle benötigte Speicher nur 1/64 des Speichers der entsprechenden Zeilen. Wenn das Programm jedoch keine gute räumliche Lokalität hat, dann sind Cache-Fehlzugriffe auf die Daten sowie Fehlzugriffe auf die Zustandstabelle wahrscheinlicher.

Ausschlusstabelle (Exclusion Table)

[0087] **Fig.** 10 zeigt die gemeinsam genutzte Ausschlusstabelle **1001**. Die privaten Ausschlusstabellen **1002** von **Fig.** 5, eine für jeden Prozessor, können in ihrem Aufbau ähnlich sein. Der Zweck der Ausschlusstabellen **1000** ist es, Hardware-Cache-Fehlzugriffe zu verringern, die dadurch verursacht werden, dass der Fehlzugriffs-Prüfcode Zustandstabelleneinträge für Speicherbefehle lädt. Die Ausschlusstabelle **1001** hat Biteinträge **1010**, jeweils ein Bit für jede entsprechende Zeile. Ein Bit wird auf eine logische Eins gesetzt, wenn die entsprechende Zeile den exklusiven Zustand hat, sonst wird das Bit auf eine logische Null gesetzt.

[0088] Anstelle des Prüfens der Einträge **545** der Zustandstabelle **540** kann der Speicher-Fehlzugriffs-Prüfcode die Bits **1010** der Ausschlusstabelle **1000** prüfen, um festzustellen, ob eine entsprechende Zeile den exklusiven Zustand hat. Wenn die Zeile den exklusiven Zustand hat, dann kann die Speiche-

rung sofort durchgeführt werden.

[0089] Bei 64-Byte-Zeilen ist der von der Ausschlusstabelle **1000** verwendete Speicher 1/512 des von den Zeilen verwendeten Speichers. Daher kann die Anzahl von Hardware-Cache-Fehlzugriffen, die vom Speicher-Fehlzugriffs-Prüfcode unter der Verwendung der Ausschlusstabelle **1001** verursacht werden, ein Achtel der Hardware-Cache-Fehlzugriffe sein, die lediglich unter der Verwendung der Zustandstabellen auftreten würden. Es wird darauf hingewiesen, dass die Verwendung der Ausschlusstabellen **1000** für Speicher-Fehlzugriffs-Codeprüfungen teilweise durch den Ungültig-Flag **801** von **Fig.** 8 ermöglicht wird. Der Lade-Fehlzugriffs-Prüfcode für Ladevorgänge muss in dem Fall nicht auf die Zustandstabelle **540** zugreifen, wo die Daten gültig sind. Daher wird nur durch den Fehlzugriffs-Prüfcode für Speicherbefehle auf die Ausschlusstabellen **1000** zugegriffen.

Stapelbildung (Batching)

[0090] Der Batch-Optimierungsschritt **450** von **Fig.** 4 erkennt, dass Lade- und Speichervorgänge von Daten häufig in Stapeln (Batches) im Verhältnis zu einem gemeinsamen Basisregister durchgeführt werden. Zum Beispiel ist es in Programmen häufig der Fall, dass Daten in einer Reihenfolge gemäß ihrer Adressen abgerufen und manipuliert werden. Der Batch-Optimierschritt **450** erfasst einen Satz von Befehlen, die auf einen Bereich von Zieladressen zugreifen, der nicht größer als die Größe einer Zeile ist, z. B. ist der Bereich höchstens 64 Bytes groß. Ein solcher Satz von Lade- und Speicherbefehlen kann höchstens auf Daten in zwei unmittelbar nebeneinanderliegenden Zeilen und in manchen Fällen nur auf eine einzige Zeile zugreifen.

[0091] In diesem Fall stellt der Fehlzugriffs-Prüfcode fest, ob die beiden Zeilen in einem korrekten Zustand sind. Wenn dies der Fall ist, können alle Lade- und/oder Speicherbefehle im Satz ohne irgendwelche zusätzlichen Überprüfungen durchgeführt werden. Es versteht sich, dass eine Stapelüberprüfung auch für einen Bereich von Zieladressen durchgeführt werden kann, die sich über eine einzige Zeile erstrecken. Jedoch kann der Code, der zwei nebeneinander liegende Zeilen überprüft, ohne viel zusätzlichen Aufwand auch eine einzige Zeile überprüfen.

[0092] Eine Einschränkung besteht darin, dass die gestapelten Lade- und Speicherbefehle nicht mit anderen Lade- und Speichervorgängen gemischt werden können, die ihren eigenen Fehlzugriffs-Prüfcode haben. Fehlzugriffe, die durch andere Lade- und Speichervorgänge hervorgerufen werden, können den Zustand einer Zeile verändern, wodurch sich für die gestapelten Lade- und Speicherbefehle ein fehlerhaftes Ergebnis ergibt. Jedoch können Lade- und Speichervorgänge über mehrere Basisregister gestapelt werden, solange richtige Fehlzugriffsüberprüfungen für entsprechende Leitungen durchgeführt werden, die über die entsprechenden Basisregister referenziert sind.

[0093] Eine weitere Einschränkung besteht darin, dass das vom Stapel von Befehlen verwendete Basisregister nicht durch eine Variable modifiziert werden kann, während der Stapel auf Zieladressen im überprüften Bereich zugreift. Dies würde die anfängliche Prüfung für den Stapel ungültig machen. Es ist möglich, das Basisregister durch eine Konstante zu modifizieren, da in diesem Fall die Bereichsprüfung statisch für der Ausführung der gestapelten Zugriffsbefehle durchgeführt werden kann.

[0094] Das Stapelbildungsverfahren ist beim Verringern des zusätzlichen Aufwands für den Fehlzugriffs-Prüfcode immer erfolgreich. Das Verfahren ist jedoch insbesondere für Befehle einer Schleife nützlich, die "entrollt" wurde. Eine entrollte Schleife enthält Befehle, die linear und nicht in einer iterativen zirkulären Weise durchgeführt werden. Hier funktionieren Zugriffsbefehle typischerweise innerhalb eines kleinen Bereichs eines Basisregisters, das während der Iterationen nicht modifiziert wird. In diesem Fall kann das Batching-Verfahren fast immer und auf sehr wirkungsvolle Weise angewendet werden.

[0095] Auch wenn das Batching immer für Befehle eines einzelnen Basisblocks versucht wird, kann es auch möglich sein, das Batching für Lade- und Speicherbefehle durchzuführen, die sich über mehrere Basisblocks hinweg erstrecken. Wenn Lade- und Speichervorgänge über mehrere Basisblocks hinweg gestapelt werden, ergeben sich zusätzliche Einschränkungen. Der gestapelte Satz von Befehlen kann keine Sub-Routinen-Aufrufe enthalten, da diese Aufrufe die Ausführungen von Lade- und Speichervorgängen mit unbekannten Zieladressen in den aufgerufenen Subroutinen auslösen könnten. Außerdem können die gestapelten Befehle keine Schleife enthalten, da erst bei Ausführung der Befehle des Stapels festgestellt werden kann, wie oft die Schleife wiederholt wird. Außerdem muss in einem Stapel, der bedingte Sprünge enthält, ein Speichervorgang, der in einem der abgezweigten Ausführungspfade auftritt, in allen Pfaden auftreten. Nur dann kann festgestellt werden, welche Speicherzugriffe durchgeführt werden, wenn die gestapelten Befehle ausgeführt werden.

[0096] Der Batching-Vorgang kann willkürlich viele Lade- und Speichervorgänge im Verhältnis zu einer beliebigen Anzahl von Basisregistern und über einen oder mehrere Basisblocks hinweg stapeln.

[0097] Es kann auch ein "gieriger" Batching-Algorithmus verwendet werden. Der gierige Algorithmus

lokalisiert so viele Lade- und Speicherbefehle wie möglich, um sie in einen Batch einzubinden. Der Algorithmus wird abgeschlossen, wenn ein Beendigungszustand, wie unten beschrieben, erreicht ist. Wenn sich in einem Batch nur ein einziger Lade- oder Speicherbefehl befindet, wird der gestapelte Fehlzugriffs-Prüfcode nicht verwendet.

[0098] Wenn ein bedingter Abzweigbefehl angetroffen wird, der zwei mögliche Ausführungspfade zum Ergebnis hat, dann werden beide Pfade nach Befehlen untersucht, die in einen Batch aufgenommen werden könnten. Die Abtastung der zwei getrennten Ausführungspfade wird zusammengeführt, wenn die Ausführung der beiden Pfade zusammengeführt wird.

[0099] Beendigungszustände können die Folgenden sein: ein Lade- oder Speicherbefehl, der ein Basisregister verwendet, das durch eine Variable modifiziert ist; ein Lade- oder Speicherbefehl, der eine Zieladresse außerhalb der überprüften Zeilen hat; das Aufrufen einer Subroutine; ein bedingter Abzweigbefehl, der eine Schleife verursacht, z. B. das erneute Ausführen eines oder mehrerer Befehle; das Ende einer Subroutine wird erreicht; ein Speicherbefehl in einem von mehreren Zweigen; und das Abtasten eines Zweigs, der mit einem parallelen Zweig zusammengeführt wird, wobei jedoch die Abtastung des parallelen Zweigs schon abgeschlossen ist.

Fehlzugriffs-Prüfcode für Stapel von Befehlen

[0100] Die **Fig.** 11 und 12 zeigen den Fluss **1100** bzw. den Fehlzugriffs-Prüfcode **1200** für eine Gruppe gestapelter Ladebefehle, die auf einen Bereich von Zieladressen **1130** zugreifen. Der Bereich **1130** kann ganz leicht dadurch überprüft werden, dass eine Fehlzugriffs-Codeüberprüfung **1140**–**1141** an der ersten Adresse **1111** und der letzten Adresse **1121** des Bereichs **1130** von Adressen durchgeführt wird, auf die vom Satz von Zugriffsbefehlen zugegriffen wird. Die ersten und letzten Adressen müssen in der ersten bzw. letzten Zeile **1110** bzw. **1120** sein, siehe Befehle **1201**–**1204**. Die Befehle **1205** und **1206** überprüfen den Ungültig-Flag.

[0101] Wenn entweder die Adresse **1111** oder die Adresse **1121** ungültig ist (**1150**), dann wird der Fehlzugriffs-Behandlungscode **1160** aufgerufen. Wenn sowohl die erste als auch die letzte Adresse gültige Daten speichern, können alle Befehle des Satzes ohne weitere Überprüfung ausgeführt werden. Ein Vorteil des Fehlzugriffs-Prüfcodes **1200** besteht darin, dass die Endpunktadressen miteinander verschachtelt sein können, um Pipeline-Staus (Stalls) wirkungsvoll auszuschließen.

Nachrichten-Weiterreichungs-Bibliothek

[0102] Die Nachrichten-Weiterreichungs-Bibliothek **353** von **Fig.** 3 liefert die notwendigen Prozeduren, um es symmetrischen Multiprozessoren **210** zu erlauben, über das Netzwerk **220** zu kommunizieren. Wenn zum Beispiel das Netzwerk **220** ATM-Protokolle verwendet, teilen die Routinen der Bibliothek **353** Nachrichten des ATM-Typs mit. Die Routinen der Bibliothek **353** können Nachrichten einer beliebigen Größe senden und empfangen. Außerdem können die Routinen periodisch nach eintreffenden Nachrichten sehen.

Fehlzugriffs-Behandlungsprotokoll

[0103] Der andere Code, der mit dem instrumentierten Programm **351** von **Fig.** 3 in Verbindung steht, ist der Fehlzugriffs-Behandlungsprotokollcode **352**. Dieser Code kann Daten aus dem Speicher eines anderen symmetrischen Multiprozessors abrufen, Kohärenz zwischen gemeinsam genutzten Kopien von Daten aufrechterhalten und garantieren, dass ein Prozessor, der das Speichern von Daten versucht, die alleinige Verfügung (Ownership) über die Daten hat.

[0104] Der Protokollcode **352** implementiert auch Synchronisierungsoperationen, wie zum Beispiel "Sperren" und "Barrieren". Der Code **352** wird aufgerufen, wenn der Fehlzugriffs-Prüfcode einen Lade- oder Speicherfehlzugriff erfasst oder wenn ein Synchronisationsvorgang erforderlich ist.

[0105] Der Protokollcode **352** ist ein verzeichnisbasiertes Invalidierungsprotokoll. Für jeden Block **551** gemeinsam genutzter Daten **550** von **Fig.** 5 wird einer der Prozessoren als der "Heim"-Prozessor zugeteilt. Blöcke können unterschiedlichen Heimprozessoren in einer zyklischen Weise zugewiesen werden, z. B. nach einer Zuweisungsreihenfolge. Blöcke können ausdrücklich einem bestimmten Prozessor zugeteilt werden, wenn von einem der Programme **310** von **Fig.** 3 Platzierungshinweise abgegeben werden.

[0106] Ein Heimprozessor ist für das Initialisieren der an den Adressen des Blocks gespeicherten Daten verantwortlich. Der Heimprozessor stellt auch die anfänglichen Zustände der Zeilen des zugewiesenen Blocks her, wobei der Zustand zum Beispiel eine exklusive Verfügung (Ownership) über die Daten reflektieren kann. Der Heimprozessor erzeugt auch die anfängliche Verzeichnisinformation über den Block.

[0107] Das Verzeichnis zeigt auch, wie unten beschrieben, an, welche Prozessoren eine Kopie des dem Heimprozessor zugewiesenen Blocks haben. Wenn ein Prozessor, der nicht der Heimprozessor ist, auf die Daten des Blocks zugreifen möchte, sendet er eine Nachricht an den Heimprozessor, die anzeigt, dass er Daten des Blocks entweder laden oder spei-

chern möchte. Wenn er sie speichern möchte, wird auch eine Ownership-Anforderung gesendet.

Heimprozessorverzeichnis

[0108] Wie in **Fig.** 13 gezeigt, unterhält jeder Prozessor **210** ein Verzeichnis **1300**, das Information über in Blöcken enthaltene Zeilen speichern kann, für die der Prozessor der Heimprozessor ist. Außerdem hat zu jeder Zeit jede Zeile eines bestimmten Blocks einen "steuernden" Prozessor. Der Prozessor, der eine Zeile steuert, kann der Prozessor sein, der die Zeile zum letzten Mal exklusiv besessen hat.

[0109] Für jeden von einem Heimprozessor besessenen Block hat das Verzeichnis **1300** einen Eintrag **1301** für jede Zeile im Block. Jeder Eintrag **301** enthält eine Identifizierung (ID) **1310**, eine Blockgröße **1315** und einen Bitvektor **1320**. Die ID **1310** zeigt an, welcher Prozessor derzeit den Block steuert, und der Vektor **1320** hat ein Bit **1321** für jeden Prozessor, der eine Kopie des Blocks hat. Die Größe des Blocks **1315** kann, wie unten im Einzelnen beschrieben, variiert werden.

Protokollnachrichten

[0110] Die Prozessoren **211** tauschen über das Netzwerk **220** von **Fig.** 2 Nachrichten aus. Bei den Nachrichten gibt es die folgenden allgemeinen Typen. Anforderungsnachrichten können Kopien von Daten zum Zwecke des Ladens und Speicherns anfordern, und Antwortnachrichten können die angeforderten Daten enthalten. Anforderungen nach Daten werden typischerweise an den Heimprozessor gesendet. Wenn der Heimprozessor keine Kopie der Daten hat, dann wird die Anforderung an den steuernden Prozessor weitergeleitet. Der steuernde Prozessor kann dem Prozessor direkt antworten, von dem die Anforderung ausging.

[0111] Manche Nachrichten werden auch zur Prozesssynchronisation verwendet. Zwei Typen von Synchronisationsmechanismen können eingesetzt werden. Zuerst können Prozessoren mit einer vorgegebenen "Barrieren"-Adresse synchronisiert werden. Wenn sie mit einer Barrierenadresse synchronisiert sind, warten Prozessoren, die die Barrierenadresse erreicht haben, bis alle anderen Prozessoren ebenfalls die Barrierenadresse erreicht haben.

[0112] Ein weiterer Typ einer Synchronisation geschieht über eine Sperre. Eine "Sperre" kann von einem beliebigen Prozessor an einer bestimmten Adresse des gemeinsamen Speichers ausgeübt werden. Ein anderer Prozessor kann erst dann an der gleichen Adresse eine Sperre ausüben, wenn die Sperre gelöst wurde.

[0113] Die Einzelheiten der vom Fehlzugriffs-Behandlungscode **352** unterstützten Nachrichten sind in den folgenden Abschnitten beschrieben.

Lesenachricht

[0114] Eine Lesenachricht fordert Daten von einem bestimmten Prozessor zum Lesen an. Diese Nachricht enthält die Adresse des Blocks, der die angeforderten Daten speichert, sowie eine Identität des anfordernden Prozessors. Auf die Nachricht wird ein gesamter Block, der die angeforderten Daten enthält, abgerufen.

Schreibnachricht

[0115] Die Schreibnachricht enthält die Adresse der angeforderten Daten und eine Identität des anfordernden Prozessors. Diese Nachricht fordert einen Block von Daten zum Zweck des Speicherns neuer Daten im Block an, wenn der anfordernde Prozessor nicht selbst eine Kopie der Daten hat. Daher fordert die Nachricht auch die Verfügung (Ownership) über den Datenblock an.

Ownership-Nachricht

[0116] Diese Nachricht fordert die Verfügung (Ownership) über die Daten in dem Fall an, wo der anfordernde Prozessor keine Kopie der Daten hat. Diese Nachricht wird verwendet, wenn der anfordernde Prozessor seine Kopie der Daten modifizieren möchte. Die Ownership-Nachricht enthält die Adresse der Daten und eine Identität des anfordernden Prozessors.

Sauber-Nachricht

[0117] Diese Nachricht wird zum Mitteilen einer Anforderung nach einer (sauberen) Nur-Lese-Kopie der Daten verwendet. Die Sauber-Nachricht enthält die Adresse der angeforderten Daten, die Anzahl von Bytes und eine Identität des anfordernden Prozessors. Eine Optimierung besteht darin, dass die Anforderung an keinen weiteren Prozessor weitergeleitet zu werden braucht, wenn der Heimprozessor eine Kopie der angeforderten Daten hat.

Weiterleitungsnachricht

[0118] Diese Nachricht fordert an, dass eine schreibbare Kopie der Daten vom Prozessor, der derzeit die Daten steuert, an den Prozessor geschickt wird, der eine Anforderung der Daten gesendet hat. Die Weiterleitungsnachricht enthält die Adresse der angeforderten Daten, die Anzahl von Bytes und eine Identität des anfordernden Prozessors.

Invalidierungsnachricht

[0119] Diese Nachricht fordert, dass eine Kopie der

Daten invalidiert wird. Wenn die Invalidierung abgeschlossen ist, wird an den anfordernden Prozessor eine Bestätigung geschickt. Die Invalidierungsnachricht enthält die Adresse der angeforderten Daten, die Anzahl von zu invalidierenden Bytes und eine Identität des anfordernden Prozessors.

Herabstufungsnachricht

[0120] Diese Nachricht wird, wenn der Zustand eines Blocks herabgestuft wird, lokal innerhalb eines SMP an Prozessoren gesendet, deren private Zustandstabellen ebenfalls herabgestuft werden müssen. Die Herabstufungsnachricht enthält den Typ der Herabstufung, die Adresse der angeforderten Daten, die Anzahl von Bytes und die Identität des anfordernden Prozessors. Der letzte Prozessor, der die Herabstufungsnachricht erhält, schließt den mit der Anforderung zusammenhängenden Vorgang ab, welcher die Herabstufung anstieß.

Sauber-Antwort-Nachricht

[0121] Diese Nachricht enthält eine Kopie der tatsächlich in der Sauber-Nachricht angeforderten Daten. Die Sauber-Antwort-Nachricht enthält die Adresse der angeforderten Daten, die Anzahl der Bytes und die Daten.

Weiterleitungs-Antwort-Nachricht

[0122] Diese Nachricht enthält eine schreibbare Kopie der angeforderten Daten. Die Weiterleitungs-Antwort-Nachricht enthält die Adresse der angeforderten Daten, die Anzahl von Bytes und die Daten.

Invalidierungs-Antwort-Nachricht

[0123] Diese Nachricht ist eine Bestätigung, dass Daten invalidiert wurden. Die Invalidierungs-Antwort-Nachricht enthält die Adresse der angeforderten Daten und die Anzahl invalidierter Bytes.

Barrieren-Warte-Nachricht

[0124] Diese Nachricht fordert eine an den anfordernden Prozessor gerichtete Mitteilung an, wenn alle Prozessoren eine bestimmte Barrierenadresse erreicht haben. Die Barrieren-Warte-Nachricht enthält die Barrierenadresse und die Identität des anfordernden Prozessors.

Barriere-Fertig-Nachricht

[0125] Diese Nachricht zeigt an, dass die Bedingungen der Barrieren-Warte-Nachricht erfüllt sind. Die Barriere-Fertig-Nachricht enthält die Barrierenadresse.

Sperrnachricht

[0126] Diese Nachricht fordert die Verfügung (Ownership) über eine Sperre an. Bei der vorliegenden Umsetzung wird die Sperre auf einer spezifischen Adresse des gemeinsamen Speichers ausgeführt. Die an der Adresse gespeicherten Daten spielen hinsichtlich der Sperrnachricht keine Rolle. Die Sperrnachricht enthält die der Sperre zugeordnete Adresse.

Sperre-Weiterleitungsnachricht

[0127] Diese Nachricht leitet eine Sperranforderung an einen Prozessor weiter, der derzeit die Verfügung (Ownership) über die gesperrte Adresse hat. Die Sperr-Weiterleitungsnachricht enthält die Sperradresse.

Sperr-Antwortnachricht

[0128] Diese Nachricht überträgt die Verfügung (Ownership) über die gesperrte Adresse an den anfordernden Prozessor. Die Sperr-Antwortnachricht enthält die gesperrte Adresse.

Schmutzige Daten

[0129] Die oben beschriebenen Protokollnachrichten erlauben die gemeinsame Nutzung "schmutziger" Daten. Dies bedeutet, dass der Heimprozessor eines Blocks nicht eine saubere, aktuelle Kopie der Daten haben muss. Zum Beispiel könnte ein anderer Prozessor seine Kopie der Daten geändert haben und die modifizierte Kopie der Daten mit anderen Prozessoren außer dem Heimprozessor geteilt haben. Dieses Merkmal bewirkt, dass ein Rückschreiben an den Heimprozessor optional wird. Ansonsten ist ein Rückschreiben an den Heimprozessor immer dann erforderlich, wenn ein Prozessor eine Kopie schmutziger Daten von einem anderen Prozessor liest.

Polling (Abfragen)

[0130] Ein Polling-Mechanismus wird zum Verarbeiten der Nachrichten verwendet, die von den Prozessoren **211** erzeugt wurden. Zum Beispiel wird das Netzwerk **220** nach einer eintreffenden Nachricht abgefragt, wenn ein Prozessor auf eine Antwort zu einer Anforderungsnachricht wartet. Hierdurch wird eine gegenseitige Sperrsituation vermieden.

[0131] Außerdem werden, um angemessene Antwortzeiten für Anforderungen zu garantieren, die Programme so instrumentiert, dass sie immer dann eintreffende Nachrichten abfragen, wenn die Programme einen Funktionsaufruf durchführen. Wenn das Netzwerk **220** ein Netzwerk ist, das kurze Latenzen hat, kann ein Polling (Abfragen) auch häufiger durchgeführt werden, wie zum Beispiel an jeder Pro-

grammsteuerungs-Hinterflanke. Eine Programmsteuerungshinterflanke kann ein Sprungbefehl sein, der verursacht, dass eine Schleife iterativ erneut durchgeführt wird. Daher wird für jede Iteration einer Schleife eine Hinterflankenabfrage durchgeführt.

[0132] Nachrichten könnten unter der Verwendung eines Unterbrechungsmechanismus behandelt werden. Jedoch dauert das Behandeln einer Unterbrechung üblicherweise länger in der Verarbeitung, da der Zustand, der zur Zeit der Unterbrechung besteht, zunächst gespeichert und in der Folge wieder hergestellt werden muss. Außerdem ist beim Abfragen (Polling) die Aufgabe der Umsetzung unteilbarer Protokollvorgänge vereinfacht.

[0133] Aufgrund des relativ hohen Aufwands des Sendens von Nachrichten zwischen Prozessoren werden äußere Protokoll-Kohärenznachrichten minimiert. Da ein Heimprozessor eines Blocks die Behandlung der Anforderung durch eine Weiterleitung der Anforderung an den derzeit steuernden Prozessor garantiert, können alle Nachrichten, die Information im Verzeichnis **1300** ändern, abgeschlossen werden, wenn die Nachrichten den Heimprozessor erreicht haben. Es besteht daher keine Notwendigkeit, eine zusätzliche Nachricht zu schicken, um zu bestätigen, dass eine Weiterleitungsanforderung erfüllt wurde. Außerdem werden alle Invalidierungsbestätigungen, die auf Exklusiv-Anforderungen erzeugt wurden, direkt an den anfordernden Prozessor und nicht über den Heimprozessor geleitet.

Sperr-freier Cache (Lock-up Free Cache)

[0134] Das Protokoll **352** sieht auch ein Lösungs-Konsistenzmodell vor, das im Wesentlichen äquivalent mit einem in Hardware implementierten sperr-freien Cache ist, der nicht blockierende Lade- und Speichervorgänge erlaubt. Daten, die in den verteilten gemeinsam genutzten Speichern zwischengespeichert werden, können einen der folgenden Zustände haben: ungültig (Invalid), gemeinsam (Shared), exklusiv (Exclusive), schwebend-ungültig (Pending-Invalid) oder schwebend-gemeinsam (Pending-Shared). Die schwebenden Zustände sind vorübergehende Zustände einer Zeile, wenn eine Anforderung nach dem die Zeile enthaltenden Block ansteht. Der Schwebend-Ungültig-Zustand besteht für Daten, bei denen eine Lese- oder Schreibanforderung ansteht. Der Schwebend-Gemeinsam-Zustand besteht für Daten, bei denen eine Ownership-Anforderung ansteht.

[0135] Nicht blockierende Speichervorgänge werden dadurch unterstützt, dass ein Prozessor mit der Verarbeitung von Befehlen fortfährt, nachdem eine Anforderung nach Daten gemacht wurde. Während die Anforderung ansteht, erkennt das Protokoll die Adressen aller Daten, die in der lokalen Kopie des Blocks modifiziert sind. Dann können, wenn der angeforderte Block von Daten verfügbar wird, die modifizierten Daten mit den angeforderten Daten zusammengeführt werden. Es wird darauf hingewiesen, dass das oben beschriebene Stapeln von Lade- und Speichervorgängen nicht blockierende Ladevorgänge erlaubt, da das Stapeln von Ladevorgängen dazu führen kann, dass für eine einzige Überprüfung mehrere anstehende Ladevorgänge vorliegen.

[0136] Ein sperr-freies Verhalten kann auch für Daten unterstützt werden, die einen schwebenden Zustand haben. Das Speichern von Daten an Adressen schwebender Daten kann zur Durchführung zugelassen werden, indem die Adressen aufgezeichnet werden, wo die Daten gespeichert sind, und die Adressen an den Fehlzugriffs-Behandlungscode **352** von **Fig.** 3 übergeben werden.

[0137] Alle Speichervorgänge in einem Block in einem schwebenden Zustand werden innerhalb der Protokollroutine abgeschlossen, während auf dem entsprechenden Zustandstabelleneintrag eine Sperre liegt. Dieses Verfahren der Durchführung schwebender Speichervorgänge ist wichtig, um sicherzustellen, dass die Speichervorgänge für alle Prozessoren sichtbar sind, die später am selben Block einen Protokollvorgang durchführen.

[0138] Ladungsvorgänge von Adressen von Daten, die einen Schwebend-Gemeinsam-Zustand haben, werden sofort zugelassen, da der Prozessor schon eine Kopie der Daten hat. Ladungsvorgänge von Adressen von Daten eines Blocks, der den Schwebend-Ungültigzustand hat, können ebenfalls durchgeführt werden, solange die Ladevorgänge von Adressen einer Zeile des Blocks sind, der gültige Daten speichert. Gültige Ladevorgänge an schwebende Zeilen werden aufgrund der Verwendung des Ungültig-Flags **801** von **Fig.** 8 schneller durchgeführt. Ein gültiger Ladevorgang einer schwebenden Zeile kann sofort durchgeführt werden, weil der geladene Wert nicht gleich dem Ungültig-Flag ist.

Variable Granularitäten

[0139] Ein Merkmal der hier beschriebenen Protokolle ist die Möglichkeit variabler Granularitäten für die Kohärenz, auch innerhalb eines einzigen Programms oder einer einzigen Datenstruktur. Variable Granularitäten sind möglich, weil alle Überprüfungen nach Fehlzugriffen von Softwarebefehlen durchgeführt werden, die auf Daten sehr kleiner Granularitäten zugreifen, z. B. Bytes, lange Wörter (long) und Quad-Wörter. Im Gegensatz dazu verwenden andere verteilte Speichersysteme virtuelle Speicherhardware zur Durchführung von Fehlzugriffsüberprüfungen an festen und groben granularen Adressen, die durch eine virtuelle Speicherseitengröße, typischerweise 4096 oder 8192 Bytes, bestimmt werden.

[0140] Unterschiedliche Typen von Daten, die von einem Programm verwendet werden, werden am natürlichsten und wirkungsvollsten mit variablen Granularitäten abgerufen. Zum Beispiel werden Datenblöcke, die von aufeinanderfolgenden Massenadressen von Eingabe/Ausgabe-Geräten gelesen und an diese geschrieben werden, am besten mit groben Granularitäten, z. B. 2 K, 4 K usw., behandelt. Jedoch erfordern viele Programme auch einen wahlfreien Zugriff auf Bereiche von Adressen, die beträchtlich kleiner, z. B. 32, 256, 1024 Bytes, sind.

[0141] Dass Anwendungsprogramme und Datenstrukturen variable Zugriffsgranularitäten haben dürfen, kann die Leistung steigern, weil Daten in der wirkungsvollsten Übertragungseinheit kommuniziert werden können. Daten mit guter räumlicher Lokalität, z. B. in Blöcke "geklumpte" Daten, können mit groben Granularitäten transportiert werden, um die Zeit langer Kommunikationslatenzen zu amortisieren. Im Gegensatz dazu können Daten, die einer "falschen Gemeinsamkeit" unterliegen, mit feineren Granularitäten kommuniziert werden.

[0142] Eine falsche Gemeinsamkeit (False Sharing) ist ein Zustand, bei dem unabhängige Teile von Daten, z. B. Feldelemente, in der Datenstruktur, z. B. in einem oder mehreren Blöcken, gespeichert und von vielen symmetrischen Multiprozessoren abgerufen werden. Bei Blöcken variabler Größe erübrigt sich die Notwendigkeit des wiederholten Transfers großer Datenquantitäten fester Größe, die kleinere unabhängige Teile falscher gemeinsamer Daten enthalten, zwischen den symmetrischen Multiprozessoren.

[0143] Demnach ist der Vorgang **300** von **Fig.** 3 dahingehend optimiert, Datentransfereinheiten zu verarbeiten, die variable Granularitäten besitzen. Eine Datentransfereinheit, z. B. ein Block, kann eine beliebige geradzahlige Vielfache von Zeilen sein, je nach der für das Programm gewählten festen Zeilengröße, z. B. können unterschiedliche Programme auf Daten mit unterschiedlichen Zeilengrößen (32, 64, 128 Byte-Zeilen) zugreifen.

[0144] Um eine entsprechende Blockgröße für eine bestimmte Datenstruktur zu wählen, kann eine auf der zugewiesenen Größe basierende Heuristik verwendet werden. Die zu Grunde liegende Heuristik wählt eine Blockgröße, die gleich der Größe der zugewiesenen Datenstruktur ist, bis zu einer vorbestimmten Schwellengröße der Datenstruktur, z. B. 1 K oder 2 K Bytes. Für zugewiesene Datenstrukturen, die größer als die vorbestimmte Schwellengröße sind, kann die Granularität einfach die Größe einer Zeile sein. Die der Heuristik zu Grunde liegende Argumentation besteht darin, dass kleine Datenstrukturen als eine Einheit übertragen werden sollten, wenn auf sie zugegriffen wird, während große Datenstrukturen, wie zum Beispiel Felder, in feinen Granularitäten übertragen werden sollten, um eine falsche Gemeinsamkeit zu vermeiden.

[0145] Die Heuristik kann dadurch modifiziert werden, dass spezielle Zuweisungsbefehle in die Programme eingefügt werden, die explizit die Blockgröße definieren. Da die Größe zugewiesener Blöcke die Korrektheit des Programms nicht beeinträchtigt, kann die Blockgröße für die maximale Leistung empirisch festgestellt werden.

[0146] Wie in **Fig.** 13 gezeigt, wird die Blockgröße **1315** eines zuweisbaren Datenstücks vom Heimprozessor in einem Verzeichnis **1300** unterhalten. Jeder Zeileneintrag enthält die Größe **1315** des entsprechenden Blocks. Prozessoren erhalten Kenntnis über die Größe eines Blocks, wenn Daten des Blocks an einen anfordernden Prozessor transportiert werden.

[0147] Da Prozessoren die Größe von Blöcken nicht zu wissen brauchen, können die Größen dynamisch festgelegt werden. Zum Beispiel kann ein Heimprozessor die Granularität einer gesamten Datenstruktur dadurch ändern, dass zuerst alle Zeilen invalidiert werden, welche die Datenstruktur enthalten, und dann die Blockgrößen in den Verzeichniseinträgen **1301** geändert werden.

[0148] Der Heimprozessor kann die Größe eines Blocks nachschlagen, wenn eine Zugriffsanforderung, z. B. Lesen, Schreiben, Ownership, an einer Zieladresse einer bestimmten Zeile empfangen wird. Dann kann der Heimprozessor die korrekte Anzahl von Zeilen, die den gesamten Block enthalten, an den anfordernden Prozessor senden. Alle anderen Kopien der Zeilen können unter der Verwendung des Vektors **1320** entsprechend vom Prozessor behandelt werden. In Reaktion auf eine Zugriffsanforderung, die nicht die anfängliche Anforderung ist, werden alle Protokolloperationen an allen Zeilen des Blocks durchgeführt.

[0149] Um den Fehlzugriffs-Prüfcode zu vereinfachen, werden die Zustände der Datenstücke Zeile für Zeile überprüft und gepflegt. Das Protokoll **352** stellt jedoch sicher, dass alle Zeilen eines Blocks immer im selben Zustand sind. Daher kann der vorgeschaltete Fehlzugriffs-Prüfcode Zustände für Blöcke variabler Größe wirkungsvoll pflegen.

[0150] Im Fall Granularitäten variabler Größe kann es sein, dass ein Prozessor die Größe eines Blocks nicht weiß, der eine angeforderte Zeile enthält. Zum Beispiel fordert ein Prozessor an, auf Daten an einer Adresse A und an einer Adresse A + 64 zuzugreifen. In dem Fall, wo der Prozessor die Größe des Blocks nicht weiß, kann es sein, dass er zwei Anforderungen unter der Annahme einer Zeilengröße von 64 Bytes macht, jeweils eine für jede Zieladresse, auch wenn die Adressen im selben Block sind.

[0151] Ein Vorteil hierbei ist jedoch, dass das Protokoll, wie hier beschrieben, in einer einzigen Nachricht den gesamten die Zeilen enthaltenden Block überträgt. Danach kann der die anfängliche Anforderung verarbeitende Heimprozessor auch erkennen, dass die zweite Anforderung nicht benötigt wird. Dies trifft in allen Fällen zu, außer wenn ein anderer Prozessor eine Anforderung zum Zugreifen auf die erste Zeile macht, bevor die Anforderung für die zweite Zeile voll verarbeitet ist. In diesem Fall muss die zweite Anforderung als eine anfängliche Anforderung behandelt werden, da die aktuellen Zustände der Daten nicht immer feststellbar sind.

[0152] **Fig.** 14 zeigt Datenstrukturen, die variable Granularitäten besitzen. Die Speicher **1401** sind einem ersten Prozessor (PROC1) zugeordnet, und die Speicher **1402** sind einem zweiten Prozessor (PROC2) zugeordnet.

[0153] Innerhalb der Speicher **1401** des ersten Prozessors wurden einem ersten Programm (P1) **1411** Datenstrukturen mit Zeilen von 64 Bytes zugeordnet, und einem zweiten Programm (P2) **1441** sind Datenstrukturen mit Zeilen von 32 Bytes zugeordnet.

[0154] Das erste Programm **1411** enthält Datenstrukturen **1421** und **1431**. Die Datenstrukturen **1421** enthalten einen Block von 128 Bytes, z. B. zwei Zeilen pro Block. Die Datenstrukturen **1431** haben acht Blöcke von 64 Bytes, z. B. eine Zeile pro Block.

[0155] Das zweite Programm enthält Datenstrukturen **1451**, **1461** und **1471**. Die Datenstrukturen **1451** enthalten acht Blöcke von jeweils 32 Bytes (einer Zeile). Die Datenstrukturen **1461** enthalten drei Blöcke von jeweils 128 Bytes (vier Zeilen). Die Datenstrukturen **1471** enthalten einen Block von 256 Bytes, z. B. acht Zeilen.

[0156] Die Speicher **1402** des zweiten Prozessors enthalten vergleichbare Programme **1412** und **1442** und ihre Datenstrukturen. Wie oben beschrieben, kommunizieren die Prozessoren Daten in Datenübertragungseinheiten von einem Block. Zum Beispiel übertragen die ersten Programme **1411** und **1412** Daten unter der Verwendung der Blöcke **1403** und die zweiten Programme **1441** und **1442** übertragen Blöcke **1404**. Ein Vorteil hierbei ist, dass die Blöcke **1403** und **1404** unterschiedliche Größen, z. B. variable Granularitäten, und unterschiedliche Zeilengrößen, z. B. 32 und 64 Bytes, haben können.

[0157] Die vorliegende Erfindung wurde unter der Verwendung spezifischer Begriffe und Beispiele beschrieben. Es versteht sich, dass verschiedene andere Anpassungen und Modifikationen innerhalb des Umfangs der Erfindung vorgenommen werden können. Die nachfolgenden Ansprüche decken alle solche Variationen und Modifikationen ab, die im Umfang der Erfindung enthalten sind.

Patentansprüche

1. In Software implementiertes Verfahren zum gemeinsamen Zugriff auf Daten, die in Speichern (**212**) symmetrischer Multi-Prozessoren (**210**) in einem Computersystem (**200**) gespeichert sind, das mehrere symmetrische Multi-Prozessoren aufweist, wobei jeder symmetrische Multi-Prozessor mehrere Prozessoren (**211**), einen Speicher mit Adressen und eine Eingabe/Ausgabe-Schnittstelle (**214**), die über einen Bus (**213**) miteinander verbunden sind, aufweist, wobei die Eingabe/Ausgabe-Schnittstellen die symmetrischen Multi-Prozessoren durch ein Netzwerk (**220**) miteinander verbinden, mit den folgenden Schritten:
– Bezeichnen eines Satzes der Adressen der Speicher als virtuelle gemeinsam genutzte Adressen zum Speichern gemeinsam genutzter Daten (**550**),
– Zuweisen eines Teils der virtuellen gemeinsam genutzten Adressen zum Speichern einer gemeinsam genutzten Datenstruktur (**551**) als einen oder mehrere Blöcke, auf die durch Programme (**310**) zugegriffen werden kann, die in einem der Prozessoren ausgeführt werden, wobei die Größe eines bestimmten zugewiesenen Blocks mit der Größe der gemeinsam genutzten Datenstruktur variiert, wobei jeder Block eine ganzzahlige Anzahl von Zeilen (**552**) enthält, wobei jede Zeile eine vorbestimmte Anzahl von Bytes gemeinsam genutzter Daten enthält;
– Unterhalten einer gemeinsam genutzten Zustandstabelle (**541**), die mehrere gemeinsam genutzte Zustandseinträge (**545**) enthält, wobei es für jede Zeile des einen oder mehr Blocks einen gemeinsam genutzten Tabelleneintrag gibt, wobei jeder gemeinsam genutzte Zustandseintrag einen möglichen Zustand der Zeile anzeigt, wobei die möglichen Zustände ungültig, gemeinsam, exklusiv und schwebend sind;
– Unterhalten einer privaten Zustandstabelle (**542**) für jeden Prozessor der mehreren symmetrischen Multi-Prozessoren, wobei jede private Zustandstabelle mehrere private Zustandseinträge (**545**) aufweist, wobei die privaten Zustandstabelleneinträge einer bestimmten privaten Zustandstabelle einen möglichen Zustand einer bestimmten Zeile anzeigen, auf die vom zugeordneten bestimmten Prozessor zugegriffen wird;
– Speichern von Verzeichnisinformation eines bestimmten Blocks der gemeinsam genutzten Datenstruktur im Speicher eines Heimprozessors, wobei die Verzeichnisinformation die Größe (**1315**) des bestimmten Blocks enthält;
– Instrumentieren der Programme (**310**) bei Befehlen, die auf die gemeinsamen Daten zugreifen, um zu überprüfen, ob die Daten verfügbar sind; und
– in Reaktion auf den Empfang einer Zugriffsanforderung von einem Anfordernden der Prozessoren zum Zugreifen auf die gemeinsam genutzten Daten, Senden eines bestimmten Blocks, der die bestimmte Zei-

le und die Größe des bestimmten Blocks enthält, an den anfordernden Prozessor über das Netzwerk, um es den Prozessoren zu ermöglichen, in Blöcken mit variabler Größe gespeicherte gemeinsam genutzte Datenstrukturen über das Netzwerk auszutauschen;
– wobei das Instrumentieren **dadurch gekennzeichnet** ist, dass es die folgenden Schritte aufweist:
– Unterteilen von Programmen (**310**) mit einer Analysiereinrichtung (**320**) in Prozeduren (**301**) und die Prozeduren (**301**) in Basis-Ausführungsblöcke (**302**), wobei ein Basis-Ausführungsblock aus einem Satz von Befehlen besteht, die ausgeführt werden, wenn der erste Befehl des Satzes ausgeführt wird;
– Analysieren der Basis-Ausführungsblöcke und der Daten und eines Ausführungsflusses (**303**) zum Lokalisieren von Befehlen, die Speicheradressen zuteilen und Zugriffe auf die zugewiesenen Adressen in den gemeinsam genutzten Teilen der Speicher (**212**) durchführen;
– Einfügen mit einem Optimier-Modul (**330**) von Befehlen in die Programme (**310**) zum Überprüfen, ob Daten verfügbar sind, um sicherzustellen, dass der Zugriff in einer kohärenten Weise erfolgt;
– Erzeugen mit einem Image-Generator (**340**) eines modifizierten maschinenausführbaren Images (**350**), das instrumentierte Programme (**351**) mit den Befehlen zum Überprüfen und einer Prozedur für Fehlzugriffs-Handhabungs-Protokollprozeduren (**352**) und eine Nachrichten-Weiterleitungs-Bibliothek (**353**) enthält.

2. Verfahren nach Anspruch 1, weiter mit dem folgenden Schritt:
– Ablegen der Verzeichnisinformation in einem Verzeichnis (**1300**), das vom Heimprozessor unterhalten wird, wobei das Verzeichnis für jede Zeile (**552**) des einen oder der mehreren Blöcke der gemeinsam genutzten Datenstruktur (**551**) einen Eintrag (**1301**) enthält, wobei jeder Eintrag die Größe (**1315**) des bestimmten die Zeile enthaltenden Blocks enthält.

3. Verfahren nach Anspruch 2, weiter mit dem folgenden Schritt:
– Unterhalten im Eintrag (**1301**) für jede Zeile (**552**) des bestimmten Blocks einer Identität (**1310**) eines Steuernden der Prozessoren (**211**), wobei der steuernde Prozessor als Letztes eine exklusive Kopie des bestimmten die bestimmte Zeile enthaltenden Blocks aufweist.

4. Verfahren nach Anspruch 3, weiter mit dem folgenden Schritt:
– Unterhalten eines Bitvektors (**1320**) im Eintrag (**1301**), wobei der Bitvektor ein Bit (**1321**) für jeden Prozessor (**211**) enthält, wobei das Bit jeweils anzeigt, ob ein entsprechender Prozessor eine gemeinsam genutzte Kopie des bestimmten Blocks hat.

5. Verfahren nach Anspruch 1, weiter mit dem folgenden Schritt:
– dynamisches Ändern der Größe des einen oder der mehreren Blöcke, die für die gemeinsam genutzte Datenstruktur (**551**) zugewiesen wurden, während die Programme (**310**) ausgeführt werden.

6. Verfahren nach Anspruch 1, weiter mit den folgenden Schritten:
– Sperren der gemeinsam genutzten Zustandstabelle (**541**) vor dem Modifizieren eines der gemeinsam genutzten Tabelleneinträge (**545**), weiter mit dem folgenden Schritt:
– Setzen des Zustands jeder Zeile (**552**) des einen oder der mehreren Blöcke auf ungültig, bevor die Größe des einen oder der mehreren Blöcke dynamisch geändert wird.

7. Verfahren nach Anspruch 6, weiter mit dem folgenden Schritt:
– Modifizieren einer der privaten Zustandstabellen (**512**) ausschließlich durch den Prozessor (**211**), der der privaten Zustandstabelle zugeordnet ist.

8. Verfahren nach Anspruch 7, weiter mit dem folgenden Schritt:
– selektives Senden einer Nachricht von einem Bestimmten der Prozessoren (**211**) eines bestimmten symmetrischen Multi-Prozessors (**210**) an andere Prozessoren der bestimmten symmetrischen Multi-Prozessoren, wenn Zustände in der privaten Zustandstabelle (**542**), die dem bestimmten Prozessor zugeordnet ist, herabgestuft werden.

9. Verfahren nach Anspruch 1, bei dem die Anzahl von Zeilen des einen oder der mehreren Blöcke einer ersten gemeinsam genutzten Datenstruktur (**1421**) sich von der Anzahl von Zeilen des einen oder der mehreren Blöcke einer zweiten Datenstruktur (**1431**) unterscheidet.

10. Verfahren nach Anspruch 1, bei dem die Anzahl von Bytes in einer der Zeilen der ersten Datenstruktur (**1421**) in einem Programm (**1411**) sich von der Anzahl von Bytes in einer der Zeilen einer zweiten Datenstruktur (**1451**) in einem anderen Programm (**1441**) unterscheidet.

11. System, mit:
– einem Netzwerk (**220**);
– mehreren symmetrischen Multi-Prozessoren (**210**), die durch das Netzwerk miteinander verbunden sind, wobei jeder symmetrische Multi-Prozessor mehrere Prozessoren (**211**) enthält;
– einem Speicher (**212**), der eine Anordnung von Adressen für jeden symmetrischen Multi-Prozessor hat, wobei jede Speicheradresse einen zugewiesenen Satz virtueller gemeinsam genutzter Adressen zum Speichern gemeinsam genutzter Daten (**550**) hat, wobei in einem Teil der virtuellen gemeinsam genutzten Adressen eine gemeinsam genutzte Datenstruktur (**551**) als einen oder mehrere Blöcke spei-

chert, auf die durch Programme (**310**) zugegriffen werden kann, die in einem beliebigen der Prozessoren ausgeführt werden, wobei die Größe eines bestimmten zugewiesenen Blocks mit einer Größe der gemeinsam genutzten Datenstruktur variiert, wobei jeder Block eine ganzzahlige Anzahl von Zeilen (**552**) enthält, wobei jede Zeile eine vorbestimmte Anzahl von Bytes gemeinsam genutzter Daten enthält; und
– eine Einrichtung (**320**) zum Instrumentieren der Programme (**351**) bei Befehlen, die auf die gemeinsam genutzten Daten zugreifen, um zu überprüfen, ob die Daten verfügbar sind; und
– wobei die Anordnung aufweist:
i) eine gemeinsame Zustandstabelle (**541**), die mehrere gemeinsame Zustandseinträge (**545**) enthält, wobei jeweils ein gemeinsamer Eintrag für jede Zeile des einen oder der mehreren Blöcke ist, wobei jeder gemeinsame Eintrag einen möglichen Zustand der Zeile anzeigt, wobei die möglichen Zustände ungültig, gemeinsam, exklusiv und schwebend sind;
ii) eine private Zustandstabelle (**542**) für jeden Prozessor der mehreren symmetrischen Multi-Prozessoren, wobei jede private Zustandstabelle mehrere private Zustandseinträge (**545**) hat, wobei die privaten Zustandseinträge einer bestimmten privaten Zustandstabelle einen möglichen Zustand einer bestimmten Zeile anzeigen, auf die vom zugeordneten bestimmten Prozessor zugegriffen wird;
– wobei auf die gemeinsam genutzten Daten zugegriffen wird, um zu überprüfen, ob die Daten verfügbar sind, und ein bestimmter Block von einem anderen Prozessor über das Netzwerk an einen anfordernden Prozessor gesendet wird, um in Blöcken variabler Größe gespeicherte gemeinsam genutzte Datenstrukturen auszutauschen;
– wobei die Einrichtung zum Instrumentieren dadurch gekennzeichnet ist, dass sie aufweist:
– ein Analysier-Modul (**320**) zum Unterteilen von Programmen (**310**) in Prozeduren (**301**) und die Prozeduren (**301**) in Basis-Ausführungsblöcke (**302**), wobei ein Basis-Ausführungsblock aus einem Satz von Befehlen besteht, die ausgeführt werden, wenn der erste Befehl des Satzes ausgeführt wird;
– wobei das Analysier-Modul auch zum Analysieren der Basis-Ausführungsblöcke und Daten und eines Ausführungsflusses (**303**) ist, um Befehle zu lokalisieren, die Speicheradressen zuweisen und Zugriffe auf die zugewiesenen Adressen in den gemeinsam genutzten Teilen der Speicher (**212**) ausführen;
– ein Optimier-Modul (**330**) zum Einfügen von Befehlen in die Programme (**310**), um zu überprüfen, ob Daten verfügbar sind, um sicherzustellen, dass der Zugriff in einer kohärenten Weise erfolgt;
– einen Image-Generator (**340**) zum Erzeugen eines modifizierten maschinenausführbaren Images (**350**), das instrumentierte Programme (**351**) mit den Befehlen zum Überprüfen und eine Prozedur für Fehlzugriffs-Handhabungs-Protokollprozeduren (**352**) and eine Nachrichten-Weiterleitungs-Bibliothek (**353**) enthält.

12. System nach Anspruch 11, bei dem die Zustandstabellen (**540**) eine Ausschlusstabelle (**1000**) enthalten.

13. System nach Anspruch 12, bei dem die Ausschlusstabelle (**1000**) einen gemeinsam genutzten Teil (**1001**) und einen privaten Teil (**1002**) enthält.

Es folgen 14 Blatt Zeichnungen

Anhängende Zeichnungen

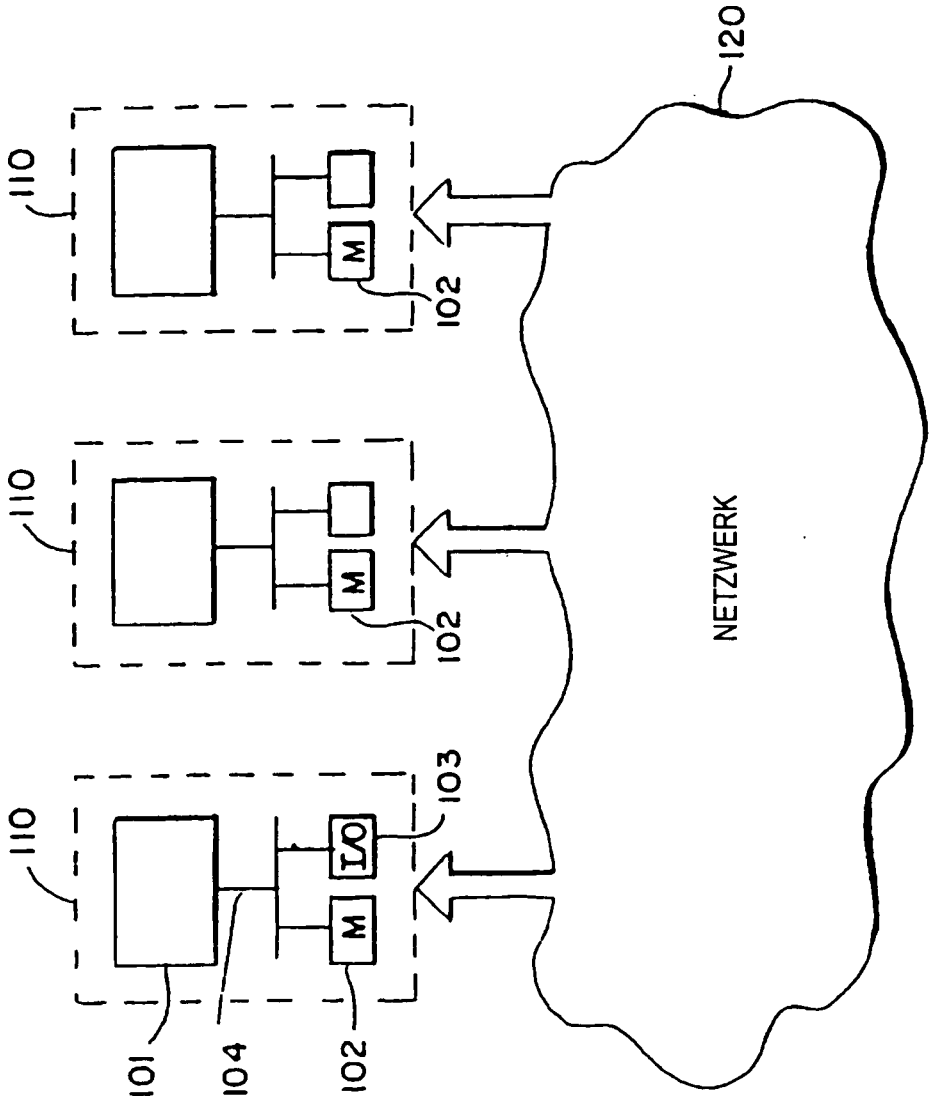

FIG. 1

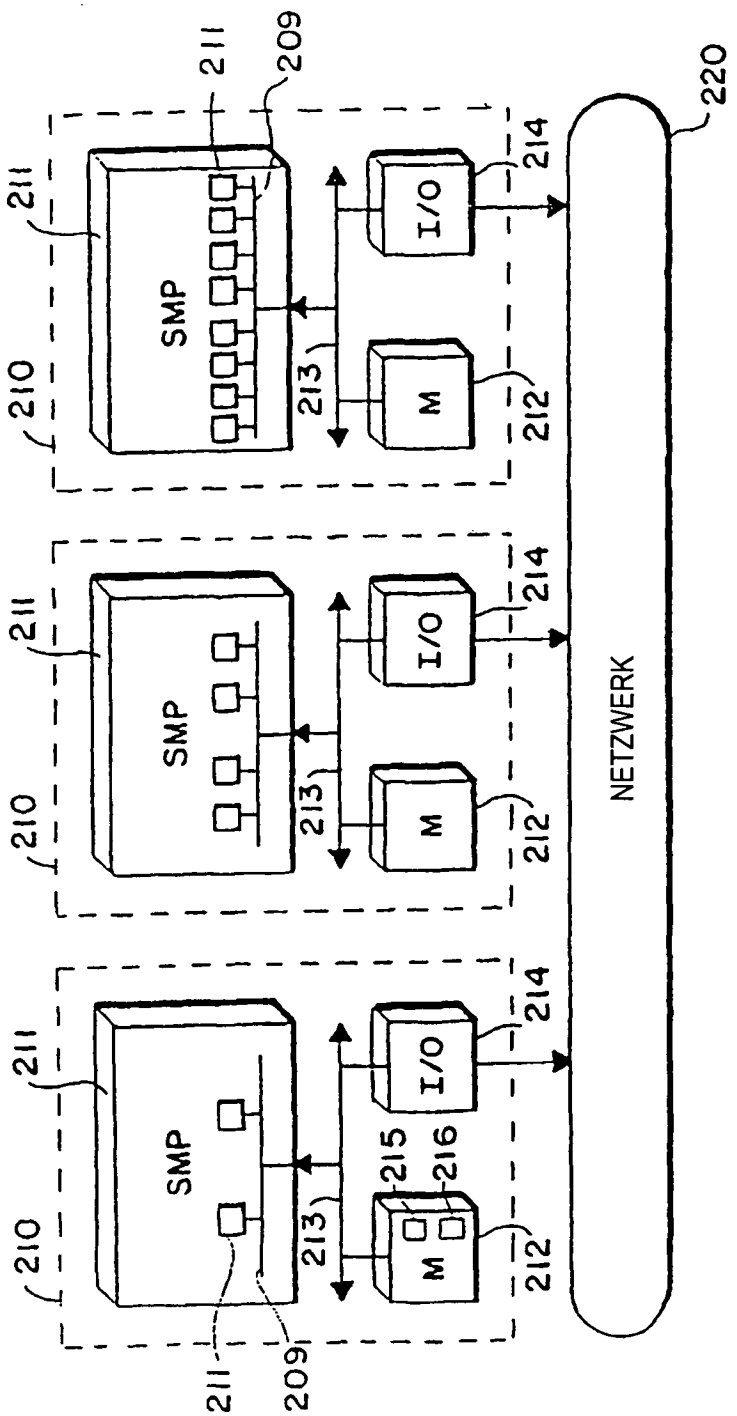

FIG. 2

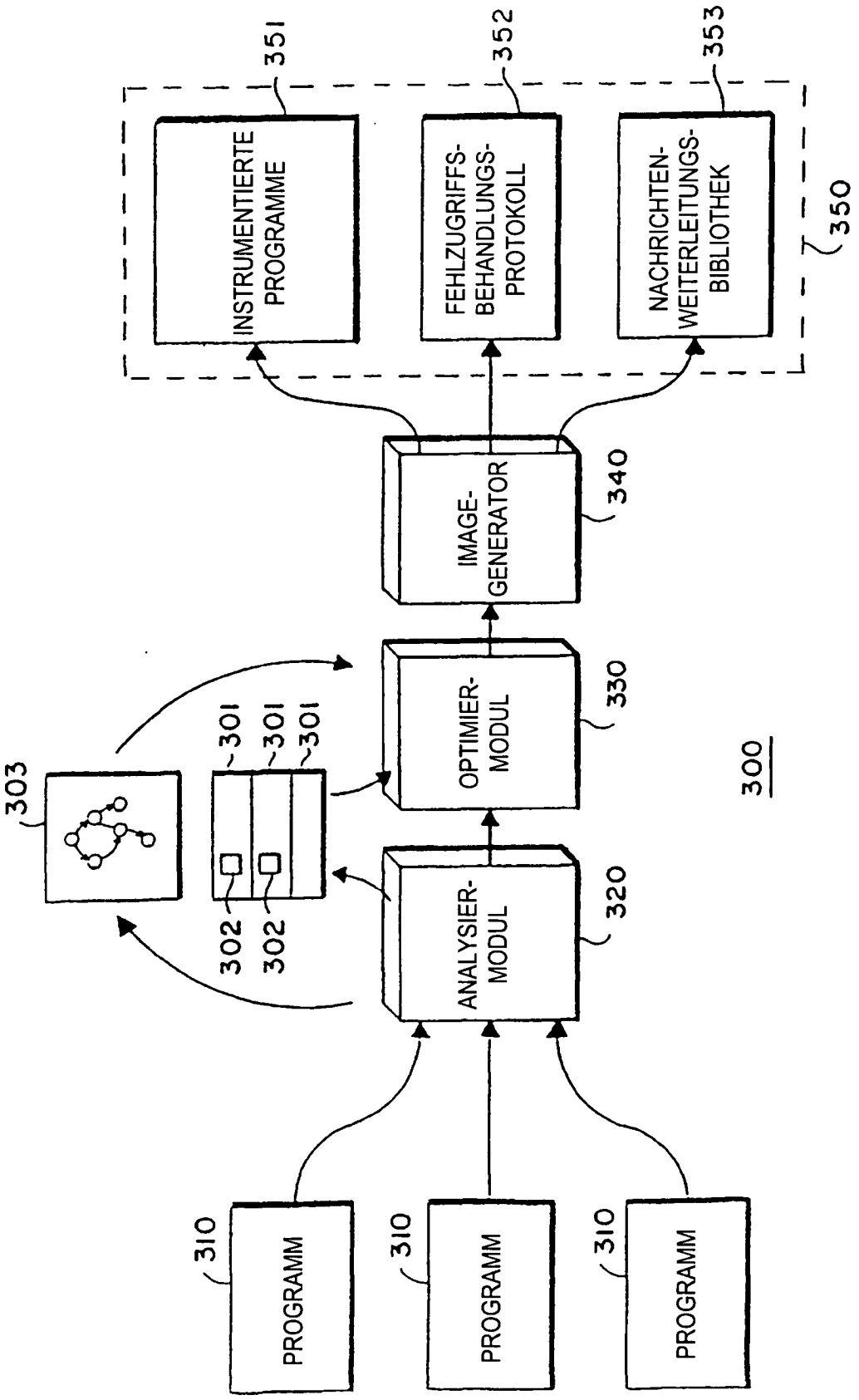
351
352
353
INSTRUMENTIERTE PROGRAMME
FEHLZUGRIFFS-BEHANDLUNGS-PROTOKOLL
NACHRICHTEN-WEITERLEITUNGS-BIBLIOTHEK
350
IMAGE-GENERATOR
340
OPTIMIER-MODUL
330
ANALYSIER-MODUL
320
303
301
301
301
302
302
300
310
PROGRAMM
310
PROGRAMM
310
PROGRAMM

FIG.3

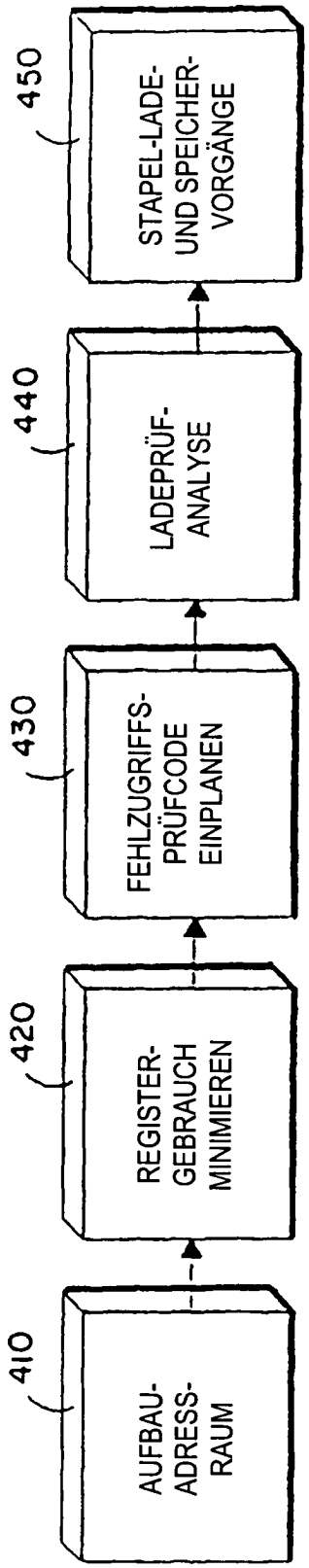
410
AUFBAU-ADRESS-RAUM
420
REGISTER-GEBRAUCH MINIMIEREN
430
FEHLZUGRIFFS-PRÜFCODE EINPLANEN
440
LADEPRÜF-ANALYSE
450
STAPEL-LADE- UND SPEICHER-VORGÄNGE
330

FIG. 4

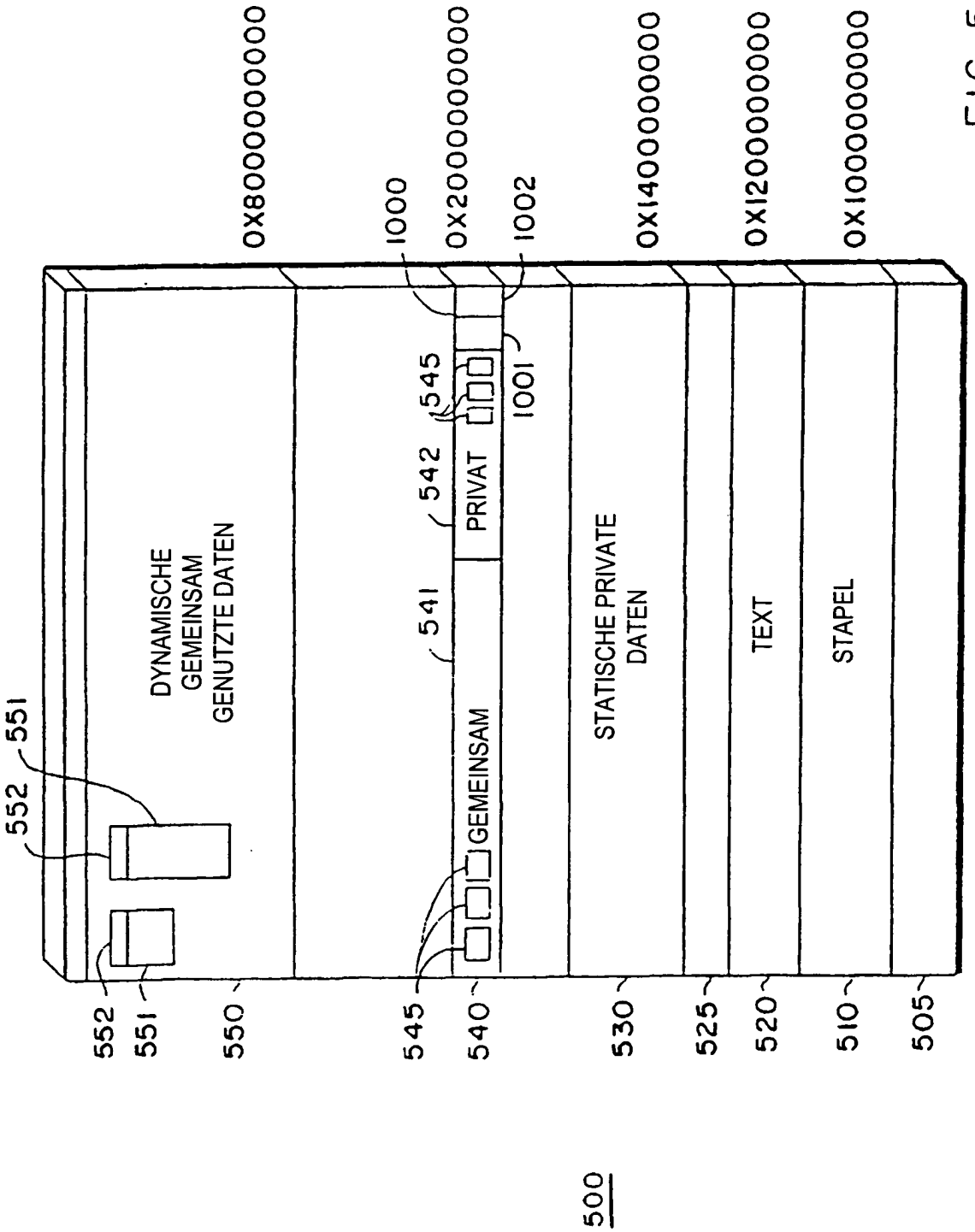
0X8000000000
0X2000000000
0X1400000000
0X1200000000
0X1000000000
1000
1002
1001
545
542
541
552
551
550
540
530
525
520
510
505
DYNAMISCHE GEMEINSAM GENUTZTE DATEN
PRIVAT
GEMEINSAM
STATISCHE PRIVATE DATEN
TEXT
STAPEL
500

FIG. 5

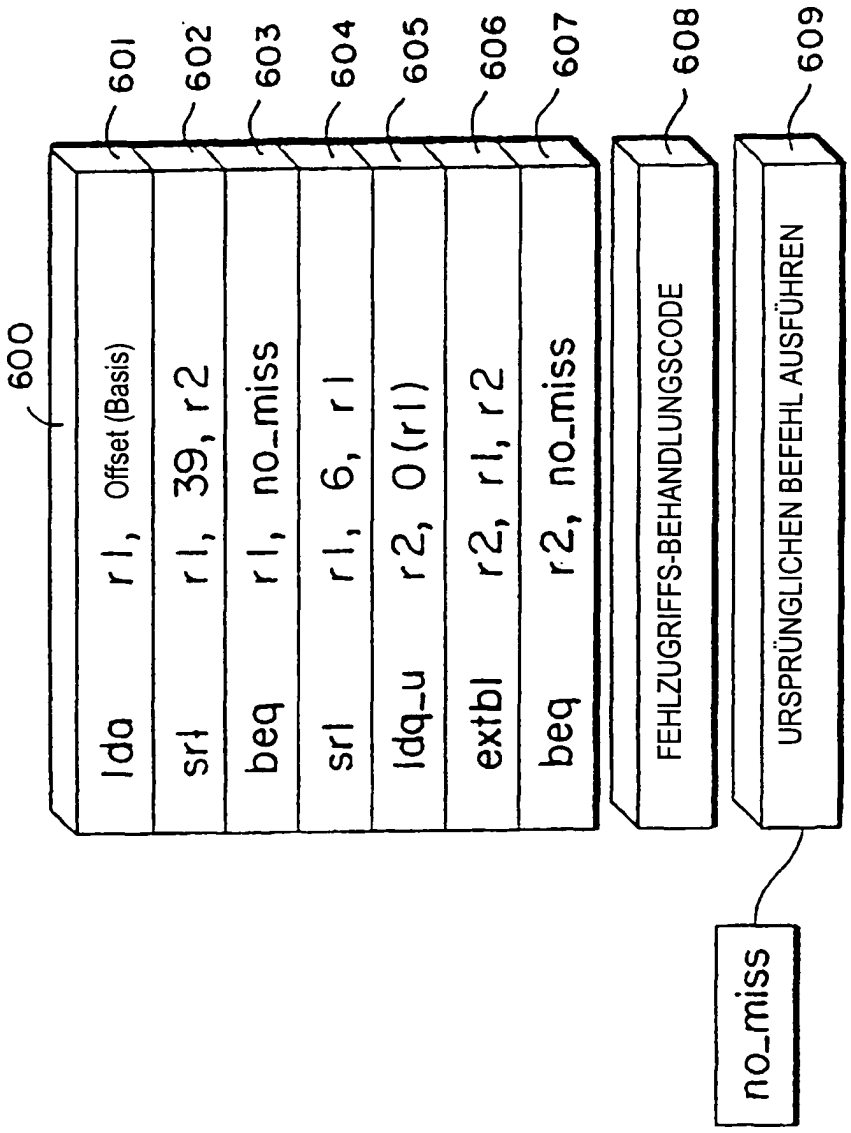
600
601
602
603
604
605
606
607
608
609
lda r1, Offset (Basis)
srl r1, 39, r2
beq r1, no_miss
srl r1, 6, r1
ldq_u r2, 0(r1)
extbl r2, r1, r2
beq r2, no_miss
FEHLZUGRIFFS-BEHANDLUNGSCODE
URSPRÜNGLICHEN BEFEHL AUSFÜHREN
no_miss

FIG.6

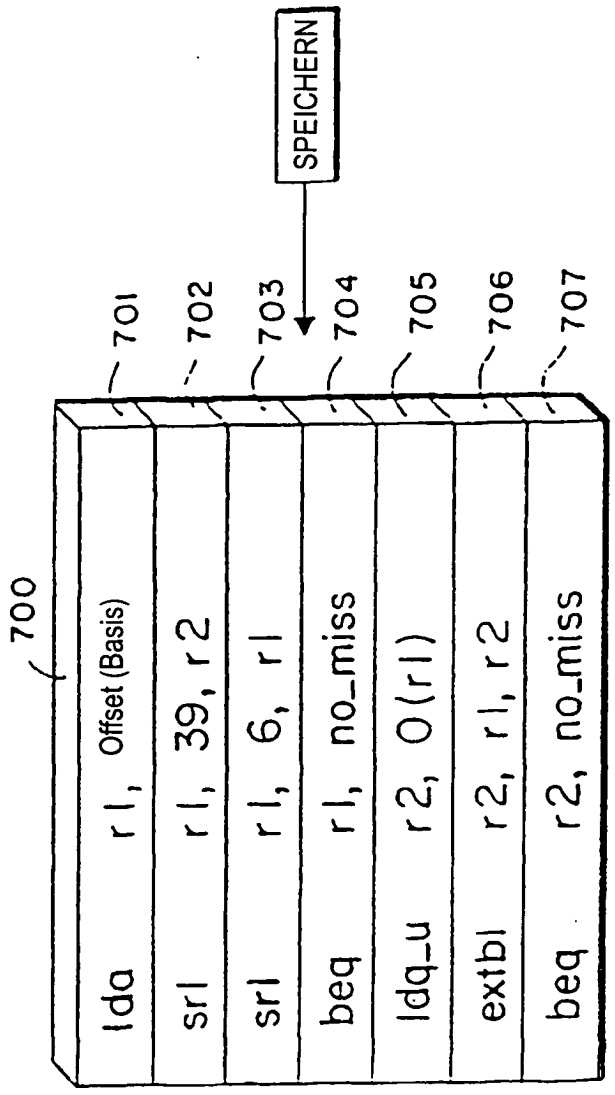
SPEICHERN
700
701
702
703
704
705
706
707
lda r1, Offset (Basis)
srl r1, 39, r2
srl r1, 6, r1
beq r1, no_miss
ldq_u r2, 0(r1)
extbl r2, r1, r2
beq r2, no_miss

FIG. 7

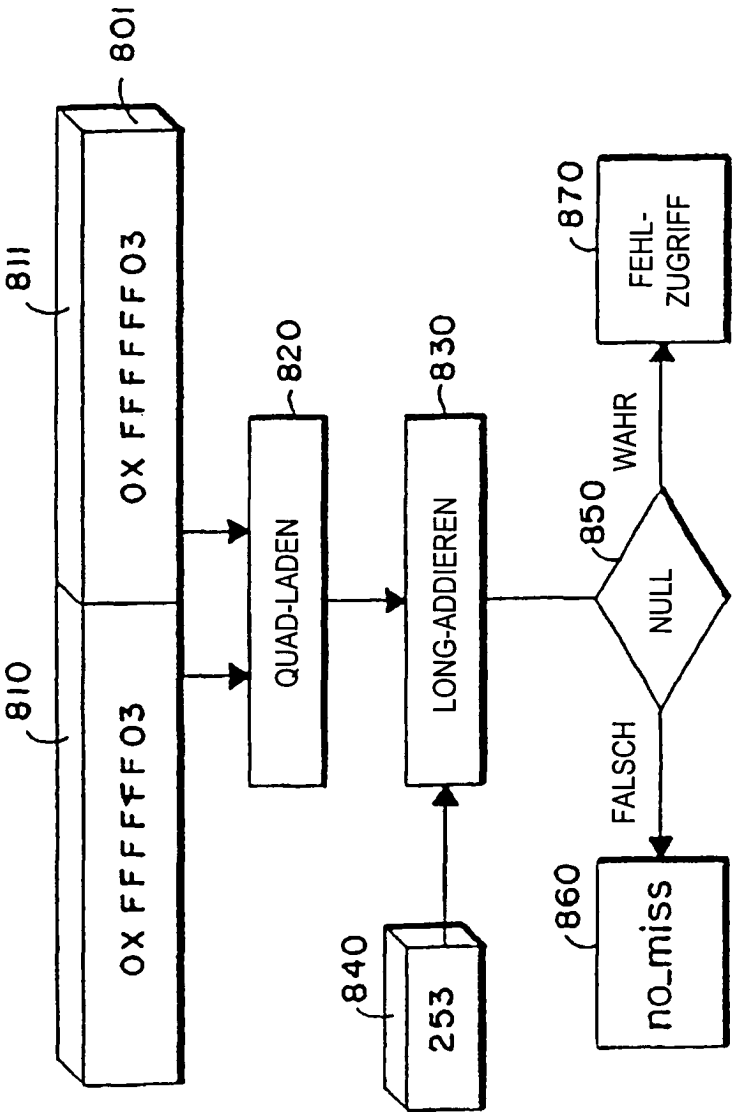
801
811
OX FFFFFF03
810
OX FFFFFF03
820
QUAD-LADEN
830
LONG-ADDIEREN
840
253
850
NULL
WAHR
FALSCH
870
FEHL-ZUGRIFF
860
no_miss

FIG.8

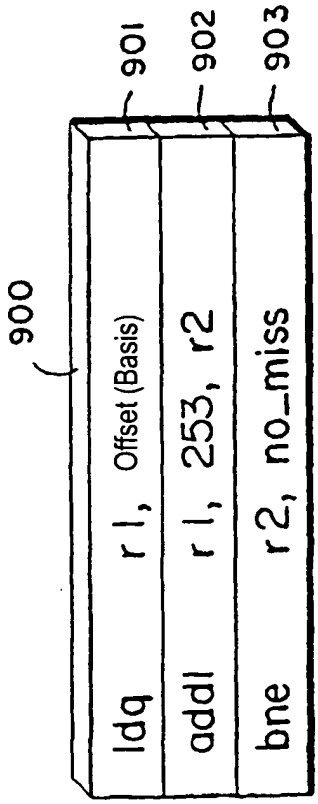
900
901
902
903
ldq r1, Offset (Basis)
addl r1, 253, r2
bne r2, no_miss

FIG. 9

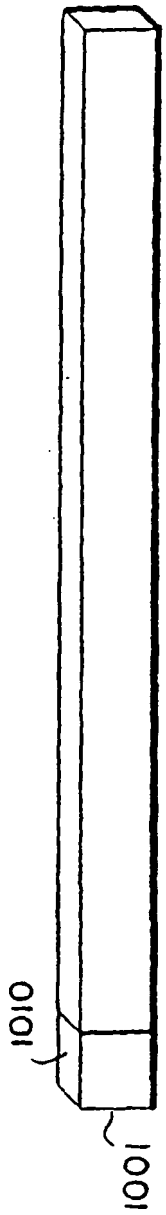

FIG. 10

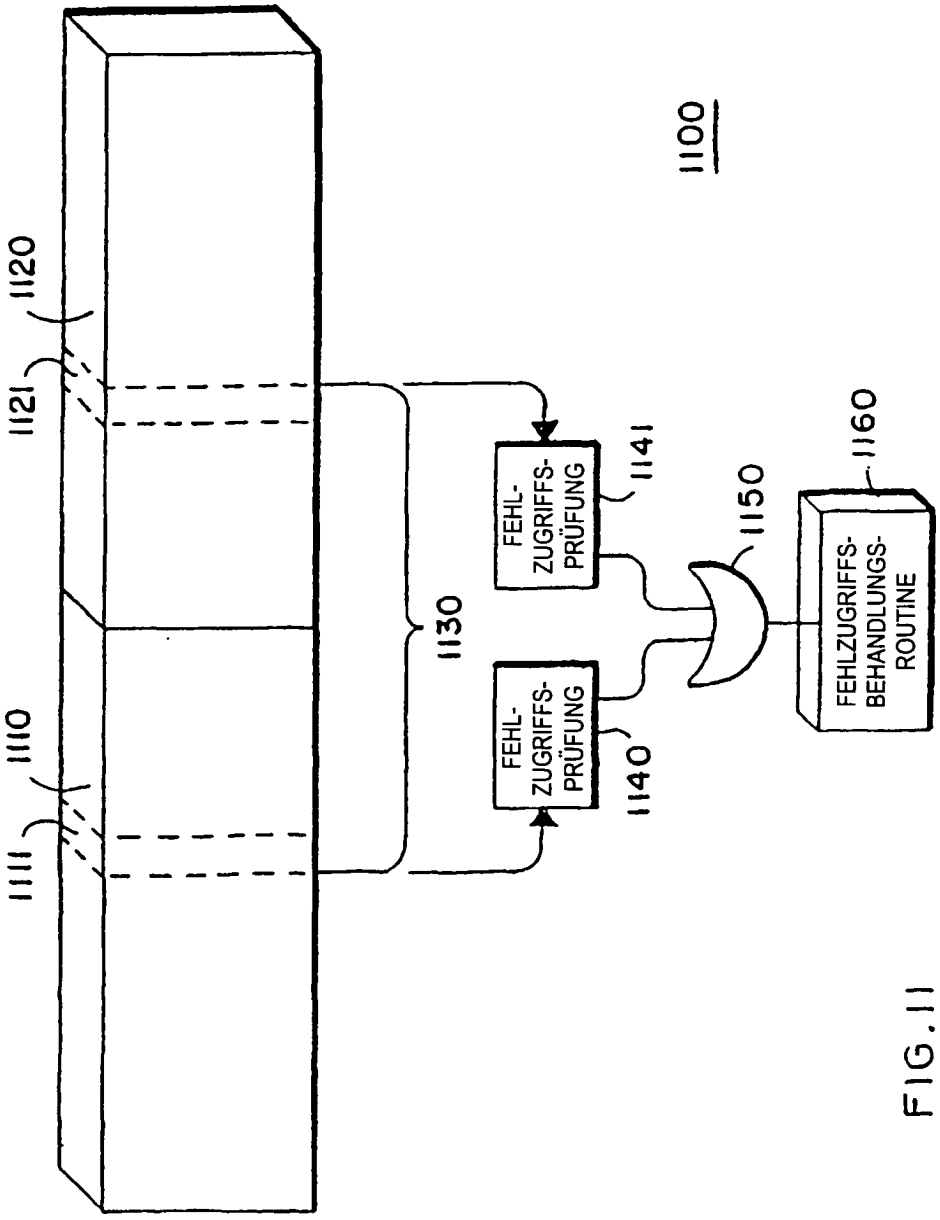
1100
1120
1121
1110
1111
1130
FEHL-ZUGRIFFS-PRÜFUNG
1141
FEHL-ZUGRIFFS-PRÜFUNG
1140
1150
1160
FEHLZUGRIFFS-BEHANDLUNGS-ROUTINE

FIG. 11

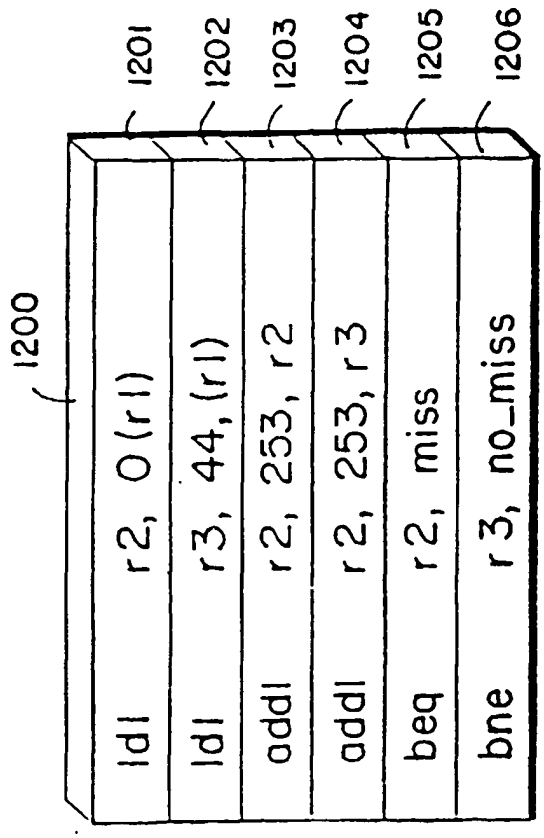
1200
1201
1202
1203
1204
1205
1206
ldl r2, 0(r1)
ldl r3, 44, (r1)
addl r2, 253, r2
addl r2, 253, r3
beq r2, miss
bne r3, no_miss

FIG. 12

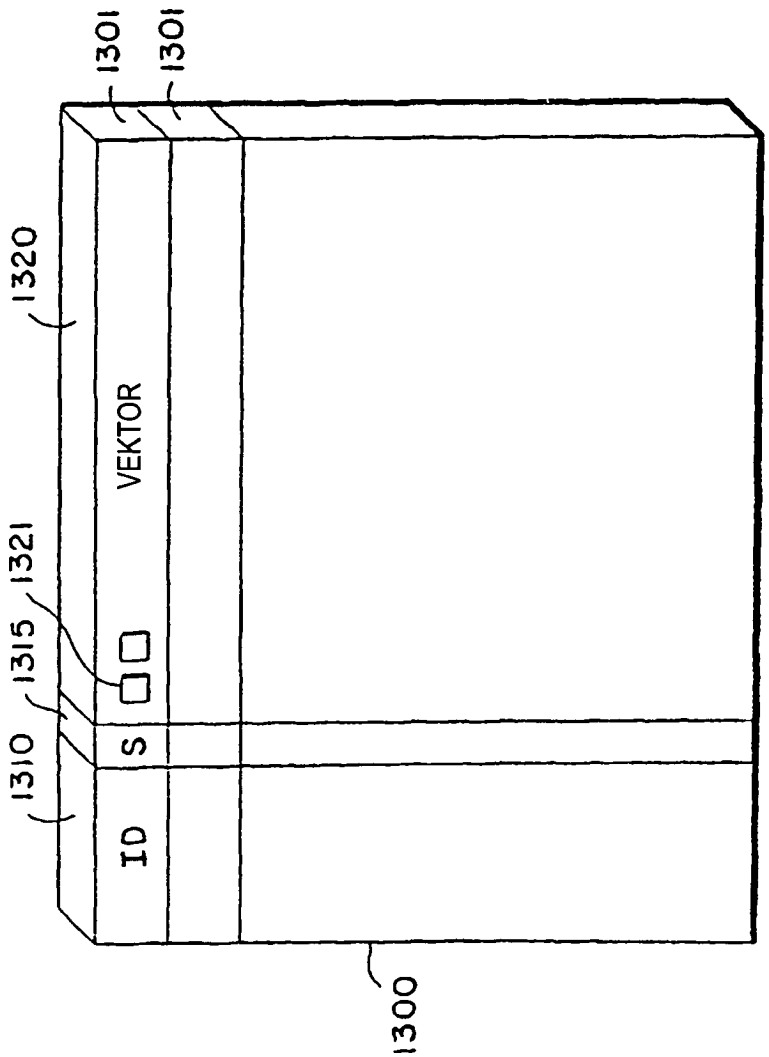

FIG. 13

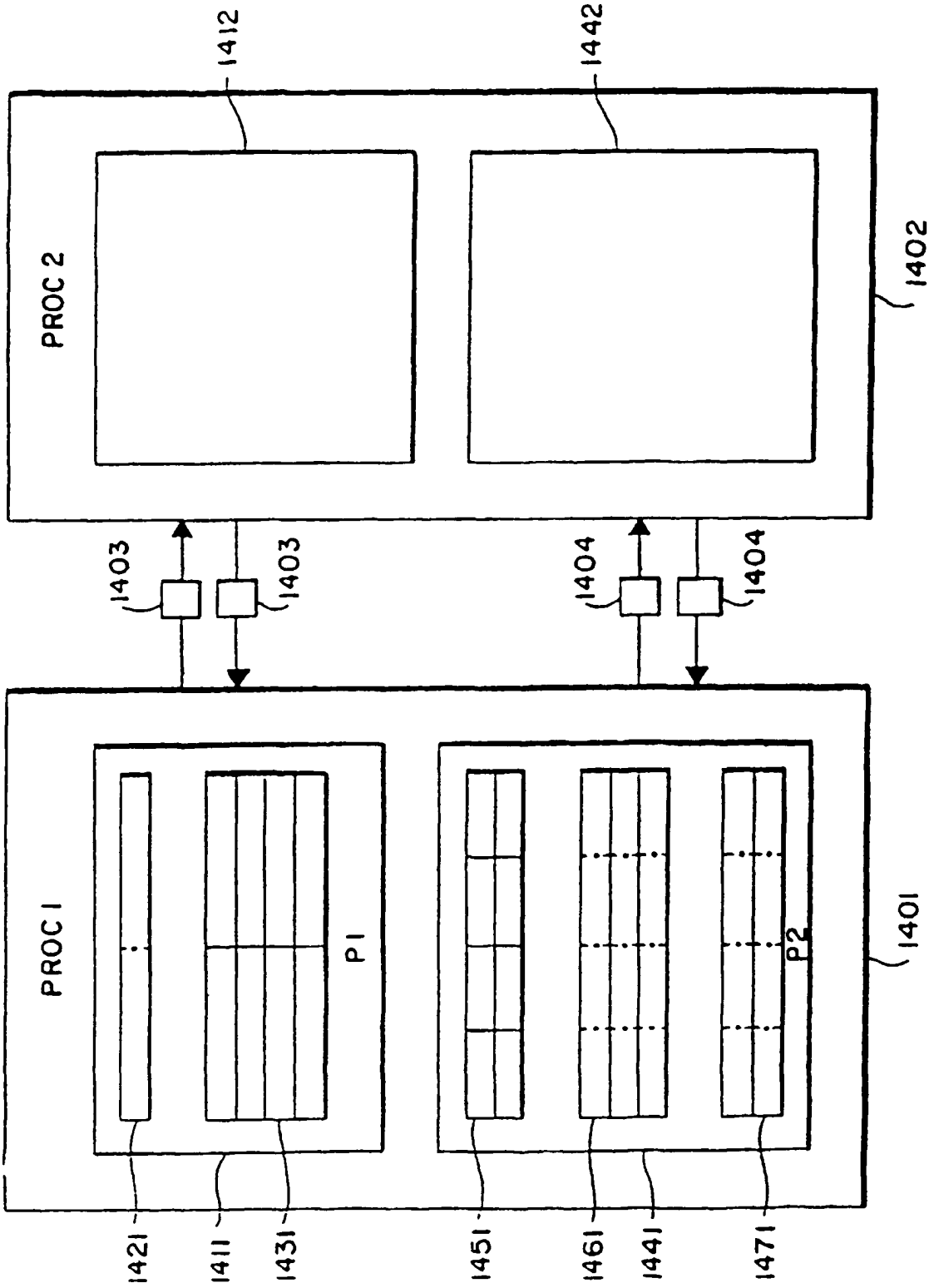
1412
1442
PROC 2
1402
1403
1403
1404
1404
PROC 1
P1
P2
1401
1421
1411
1431
1451
1461
1441
1471

FIG. 14