(19) Deutsches Patent- und Markenamt

(10) **DE 10 2021 201 597 A1** 2022.08.25

(12) # Offenlegungsschrift

(21) Aktenzeichen: **10 2021 201 597.2**
(22) Anmeldetag: **19.02.2021**
(43) Offenlegungstag: **25.08.2022**

(51) Int Cl.: ***G06F 8/40*** (2018.01)
G06F 9/44 (2018.01)

(71) Anmelder:
Robert Bosch Gesellschaft mit beschränkter Haftung, 70469 Stuttgart, DE

(72) Erfinder:
Rieber, Dennis Sebastian, 71272 Renningen, DE

(56) Ermittelter Stand der Technik:

SOTOUDEH, Matthew [u.a.]: ISA mapper: A compute and hardware agnostic deep learning compiler. Version 1. 12-10-2018. S. 1-11. URL: https://arxiv.org/pdf/1810.09958v1 [abgerufen am 2021-03-16]. - arXiv:1810.09958v1

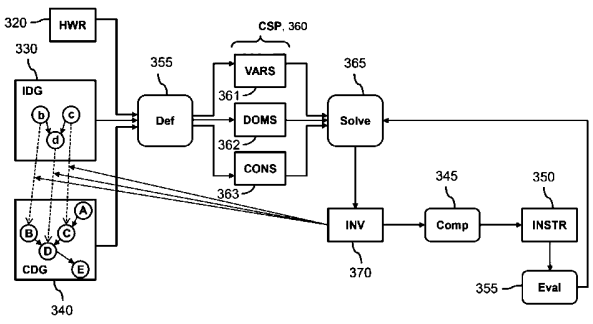
Wikipedia: Constraint-Satisfaction-Problem. 14.2.2020.<https://de.wikipedia.org/w/index.php?title=Constraint-Satisfaction-Problem&oldid=196795360>(recherchiert am 25.10.2021)

Rechercheantrag gemäß § 43 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen.

(54) Bezeichnung: **VERWENDEN VON HARDWARE-BESCHLEUNIGTEN BEFEHLEN**

(57) Zusammenfassung: Die Erfindung bezieht sich auf ein computerimplementiertes Verfahren (600) zum Implementieren einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls eines Prozessorsystems durch Lösen eines Bedingungserfüllungsproblems. Eine Lösung des Bedingungserfüllungsproblems repräsentiert einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung. Das Bedingungserfüllungsproblem weist Knoten eines Datenflussgraphen der Berechnung Knoten eines Datenflussgraphen des Befehls zu. Das Bedingungserfüllungsproblem umfasst Bedingungen, die erzwingen, dass die zugewiesenen Knoten des Berechnungsdatenflussgraphen einen dem Befehlsdatenflussgraphen äquivalenten Datenfluss aufweisen, und Bedingungen, die einschränken, welche Knoten des Berechnungsdatenflussgraphen den Eingaben des Hardware-beschleunigten Befehls zugewiesen werden können, wobei die Einschränkungen durch den Hardware-beschleunigten Befehl und/oder seine Programmierschnittstelle auferlegt sind.



Beschreibung

GEBIET DER ERFINDUNG

[0001] Die Erfindung bezieht sich auf ein computerimplementiertes Verfahren zum Implementieren einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls eines Prozessorsystems, und auf ein entsprechendes System. Die Erfindung bezieht sich ferner auf ein computerlesbares Medium, das Befehle zum Ausführen des vorstehenden Verfahrens und/oder Befehle, die die Berechnung unter Verwendung des Hardware-beschleunigten Befehls implementieren, umfasst.

HINTERGRUND DER ERFINDUNG

[0002] Mit ständig ansteigenden Mengen von Daten, die gesammelt und verarbeitet werden, wird es immer wichtiger, große Berechnungen effizient zu implementieren. Ein Gebiet, auf dem die Notwendigkeit, große Datenmengen zu verarbeiten, besonders verbreitet ist, ist maschinelles Lernen. Beispielsweise bezieht die Bewertung, besonders jedoch auch das Training neuronaler Netze, wie z. B. tiefer neuronaler Netze (DNN), die Bewertung einer großen Anzahl von Matrixmultiplikationen, Faltungen, Skalarprodukten und ähnlicher Operationen ein. Es gibt eine große und sich fortlaufend weiterentwickelnde Vielzahl von DNN-Architekturen, die für spezielle Anwendungen zugeschnitten sind.

[0003] Um solche Berechnungen zu beschleunigen, wird immer mehr spezialisierte Hardware verfügbar. Solche Hardware bezieht Beschleuniger ein, die optimiert sind, um spezielle Befehle wie z. B. Matrixmultiplikationen oder Faltungen effizient auszuführen. Die Verwendung solcher Hardware-beschleunigter Befehle erlaubt es, dass Berechnungen effizienter implementiert werden, insbesondere in Konfigurationen mit beschränkten Betriebsmitteln.

[0004] Um Hardware-beschleunigte Befehle zu verwenden, ist es wichtig, Compiler-Techniken zum Übersetzen von Code höherer Programmiersprachen in maschinenausführbare Befehle, die den Hardware-beschleunigten Befehl aufrufen, bereitzustellen. Das ist insbesondere wichtig, da sich sowohl die auszuführenden Berechnungen als auch die Hardware-Architekturen zu ihrer Ausführung kontinuierlich weiterentwickeln. Es ist jedoch eine komplexe Aufgabe, automatisch zu bestimmen, wie Berechnungen zu implementieren sind, wie z. B. diejenigen, die sich auf Bewertungen von Schichten neuronaler Netze beziehen, so dass sie die Hardware-beschleunigten Befehle am effektivsten verwenden. Einer der Gründe, warum diese Aufgabe so komplex ist, ist, dass Hardware-beschleunigte Befehle komplexe Datenströme aufweisen können, die Hunderte oder Tausende paralleler und sequenzieller Operationen über mehrdimensionale Eingabe- und Ausgabe-Arrays einbeziehen können.

[0005] In dem Dokument „ISA Mapper: A Compute and Hardware Agnostic Deep Learning Compiler“ von M. Sotoudeh u. a. (verfügbar bei http://arxiv.org/abs/1810.09958 und hier durch Bezugnahme mit aufgenommen) sind Techniken zum Ausführen von Codeerzeugung auf einen Befehlssatz, der einen Matrixmultiplikations-Befehl enthält, präsentiert. Ein Berechnungskernel für tiefes Lernen und Hardware-Fähigkeiten sind durch eine Zwischenrepräsentation beschrieben, die auf einer Repräsentation von Operatoren auf Schleifenebene arbeitet. Befehlsabbilden wird angewandt, um die möglichen Arten, auf die eine Berechnung auf einem Hardware-System ausgeführt werden kann, zu bestimmen. In diesem Befehlsabbilden werden eine Anzahl von „IR-Transformationen“ auf die Berechnung angewandt, wonach ein „deterministischer Mapper“ versucht, die transformierten Berechnungen auf die Hardware-Befehle auf einer Schleifenebene abzubilden. Ein Scheduler wählt eine spezifische Abbildung. Um den großen Suchraum von Abbildungen und Planungen zu managen, werden Heuristiken, Kostenmodelle und potentiell maschinelles Lernen verwendet.

KURZFASSUNG DER ERFINDUNG

[0006] Ein Nachteil existierender Techniken zum Implementieren von Berechnungen unter Verwendung Hardware-beschleunigter Befehle ist, dass die Verwendung einer Repräsentation auf Schleifenebene von Operationen implizite Implementierungsentscheidungen wie Schleifen- und Tensor-Reihenfolge oder Zugangsfunktionsnotation einführt. Beispielsweise kann eine Tensor-Zugangsfunktion für eine Schablone in einem Operator x * stride + w sein, während ein Beschleuniger einen Befehl mit der Zugangsfunktion i + k anbieten kann. Für stride = 1 ist eine Übereinstimmung möglich, es ist jedoch schwierig, solche Übereinstimmungen im Allgemeinen automatisch herzustellen, z. B. durch Musterabgleich. Somit können mögliche Implementierungen übersehen werden.

[0007] Außerdem ist die Abhängigkeit existierender Techniken von einer explizit spezifizierten Menge von IR-Transformation nachteilig. Es ist schwierig, eine spezifische Sequenz von Transformationen, die zu einer Einbettung führt, direkt zu detektieren. Schließlich kann das zu einem nichtdeterministischen Suchprozess führen, in dem viele unterschiedliche Implementierungskandidaten erzeugt werden müssen. Außerdem wird der Suchraum umso mehr vergrößert, je mehr Transformation hinzugefügt werden, mit abnehmendem Ertrag für die Anzahl von Lösungen. Somit gibt es eine Spannung zwischen der Erkundung, die benötigt wird, um die optimale Implementierung zu finden, und Suchraumeinschränkungen, die benötigt werden, um eine Lösung innerhalb einer angemessenen Zeitdauer zu finden. Selbst wenn eine große Menge von Transformationen verwendet wird, bleibt es jedoch möglich, dass eine Unterklasse von Implementierungen hinter einer nicht verfügbaren Transformation verborgen ist.

[0008] Aufgrund dieser Nachteile könnten existierende Techniken nicht in der Lage sein, eine Implementierung einer Berechnung, die einen Hardware-beschleunigten Befehl am effektivsten verwendet, zu finden. In einigen Fällen könnten existierende Techniken nicht einmal in der Lage sein herauszufinden, wie der Hardware-beschleunigte Befehl überhaupt in der Berechnung verwendet werden kann. Es ist somit wünschenswert, Techniken zu besitzen, die mehr mögliche Implementierungen einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls finden können. Es ist auch wünschenswert, Techniken zu besitzen, die solche Implementierungen effizienter finden können.

[0009] In Übereinstimmung mit einem ersten Aspekt der Erfindung werden ein computerimplementiertes Verfahren und ein entsprechendes System zum Implementieren einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls eines Prozessorsystems bereitgestellt, wie durch die Ansprüche 1 bzw. 14 definiert ist. In Übereinstimmung mit einem weiteren Aspekt der Erfindung ist ein computerlesbares Medium bereitgestellt, wie durch Anspruch 15 definiert.

[0010] Verschiedene Maßnahmen richten sich auf das Implementieren einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls eines Prozessorsystems. Mit Implementieren der Berechnung unter Verwendung des Hardware-beschleunigten Befehls ist gemeint, dass ein möglicher Aufruf des Befehls in der Berechnung gefunden wird, mit anderen Worten, dass ein Teil der Berechnung gefunden wird, der durch Aufrufen des Befehls ausgeführt werden kann. Der bestimmte Aufruf kann verwendet werden, um automatisch maschinenausführbare Befehle, die die Berechnung implementieren, zu erzeugen und den Hardware-beschleunigten Befehl gemäß dem bestimmten möglichen Aufruf aufzurufen, z. B. die Berechnung in ausführbaren Code zu kompilieren, was jedoch nicht notwendig ist. Das Finden des möglichen Aufrufs an sich ist bereits nützlich, z. B. zum Gebrauch beim manuellen Implementieren der Berechnung; zum Bestimmen, ob eine solche Implementierung überhaupt möglich ist; oder zum Gebrauch durch ein weiteres System, das die eigentliche Kompilierung ausführt.

[0011] Die Berechnung und der Hardware-beschleunigte Befehl können durch jeweilige Datenflussgraphen repräsentiert sein. Knoten der Datenflussgraphen können Eingaben und Operationen der Berechnung oder des Befehls repräsentieren, wobei Kanten Datenflussrelationen zwischen den Knoten repräsentieren, z. B. können eingehende Kanten einer Operation verwendet werden, um Werte, auf denen die Operation arbeitet, zu repräsentieren. Der Hardware-beschleunigte Befehl führt typischerweise eine vordefinierte Operation mit einem festen Datenfluss aus, z. B. kann der Befehlsdatenflussgraph fest sein. Die Operationen sind typischerweise Skalaroperationen, z. B. Operationen, die auf individuelle numerische oder nicht numerische Elemente angewandt werden.

[0012] Interessanterweise haben die Erfinder erkannt, dass das Problem zum Bestimmen eines möglichen Aufrufs des Hardware-beschleunigten Befehls in der Berechnung als ein Bedingungserfüllungsproblem ausgedrückt werden kann, das Knoten des Berechnungsdatenflussgraphen Knoten des Befehlsdatenflussgraphen zuweist. Somit kann eine Lösung des Bedingungserfüllungsproblems Knoten des Berechnungsdatenflussgraphen, z. B. Operationen oder Eingaben der Berechnung, Knoten des Befehlsdatenflussgraphen, z. B. Berechnungen oder Eingaben des Hardware-beschleunigten Befehls, zuweisen. Eine solche Zuweisung ist hier auch als eine „Abbildung“ des Befehlsdatenflussgraphen auf den Berechnungsdatenflussgraphen bezeichnet. Effektiv ist, anstelle des Findens eines möglichen Aufrufs in einer Herangehensweise von oben nach unten durch Transformieren der Berechnung und dann Abbilden der Berechnung auf den Hardware-Befehl, eine Herangehensweise von unten nach oben vorgeschlagen, in der durch Finden einer Lösungen für das Bedingungserfüllungsproblem eine Abbildung von dem Befehl auf die Berechnung schrittweise konstruiert werden kann.

[0013] Das Bedingungserfüllungsproblem kann so definiert sein, dass eine Lösung für das Bedingungserfüllungsproblem einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung repräsentiert. Zu diesem Zweck kann das Bedingungserfüllungsproblem wenigstens zwei Typen von Bedingungen, die als Datenflussbedingungen und Eingabebedingungen bezeichnet sind, umfassen.

[0014] Die Datenflussbedingungen können erzwingen, dass die Knoten des Berechnungsdatenflussgraphen, die den Knoten des Befehlsdatenflussgraphen zugewiesen sind, einen zu dem Befehlsdatenflussgraphen äquivalenten Datenfluss aufweisen. Beispielsweise wird der Hardware-beschleunigte Befehl auf einen Teil der Berechnung, der der Funktionalität des Hardware-beschleunigten Befehls entspricht, abgebildet. Insbesondere kann das Ausführen der Operationen des Befehlsdatenflussgraphen auf den Werten der Berechnung, auf die die Eingaben des Befehlsdatenflussgraphen abgebildet sind, gemäß den Datenflussrelationen, die durch den Befehlsdatenflussgraphen angegeben sind, zu dem gleichen Berechnungsergebnis führen wie das Ausführen der Operationen des Berechnungsdatenflusses, auf den diese Operationen abgebildet sind. Eine Art dafür, das zu erzwingen, ist durch Erzwingen, dass der Teilgraph des Berechnungsdatenflussgraphen, auf den der Befehlsdatenflussgraph abgebildet ist, mit dem Befehlsdatenflussgraphen selbst isomorph ist, obwohl Lockerungen dieser Anforderung möglich sind.

[0015] Wie die Erfinder erkannt haben, ist es jedoch für eine Abbildung von dem Befehlsdatenflussgraphen auf den Berechnungsdatenflussgraphen, um einen möglichen Aufruf des Hardware-beschleunigten Befehls zu repräsentieren, in vielen Fällen nicht ausreichend, nur Datenflussbedingungen zu verwenden. Das liegt daran, dass der Hardware-beschleunigte Befehl und/oder seine Programmierschnittstelle normalerweise zusätzliche Einschränkungen bezüglich der Werte der Berechnung, auf die der Befehl angewandt werden kann, auferlegen. Beispielsweise kann der Hardware-beschleunigte Befehl erfordern, dass seine Eingaben gemäß einem speziellen Speicherlayout organisiert sind, und/oder dass gemäß einem speziellen Zugriffsmuster auf sie zugegriffen wird. Um solche Anforderungen zu repräsentieren, kann das Bedingungserfüllungsproblem außerdem eine oder mehrere Eingabebedingungen enthalten, die einschränken, welche Knoten des Berechnungsdatenflussgraphen den Eingaben des Hardware-beschleunigten Befehls zugewiesen werden können. Somit kann ein möglicher Aufruf ein Aufruf sein, so dass er mit den durch die Einschränkungen, die durch den Hardware-beschleunigten Befehl und/oder seine Programmierschnittstelle auferlegt sind, kompatibel ist.

[0016] Interessanterweise ist es durch Definieren möglicher Aufrufe hinsichtlich eines Bedingungserfüllungsproblems möglich, einen möglichen Aufruf durch Lösen des Bedingungserfüllungsproblems zu bestimmen. Techniken dafür sind an sich bekannt. Der Datenfluss und die Eingabebedingungen können garantieren, dass der Aufruf möglich ist. Es ist nicht notwendig, zuerst eine Transformation auf die vollständige Berechnung anzuwenden und dann auszuprobieren, ob die transformierte Berechnung unter Verwendung des Hardware-beschleunigten Befehls implementiert werden kann, was ineffizient sein kann. Stattdessen kann das Abbilden, das den Aufruf des Hardware-beschleunigten Befehls repräsentiert, durch den Bedingungsauflöser schrittweise konstruiert werden, wobei die Bedingungen garantieren, dass die Lösung ein möglicher Aufruf ist. Es ist außerdem nicht notwendig, eine explizite Menge von Code-Transformationen zu definieren, was sehr schwierig oder unmöglich vollständig zu erhalten ist.

[0017] Da die Abbildung auf einer Operationsebene ausgeführt wird, im Gegensatz z. B. zu einer Schleifenebene, und da irgendeine Abbildung, die die Bedingungen erfüllt, durch den Bedingungsauflöser gefunden werden kann, können implizite Implementierungsentscheidungen wie z. B. Schleifenreihenfolge, Speicherlayout und Speicherzugriffsfunktionen vermieden werden. Insbesondere können Aufrufe mit einem Speicherlayout, das flexibler ist als dasjenige, das durch bekannte Herangehensweisen basierend auf einer vordefinierten Menge von Transformation bereitgestellt ist, bestimmt werden. Experimente mit Benchmarks zeigen, dass es diese gesteigerte Flexibilität in der Praxis erlaubt, Aufrufe mit besserer Leistung als bekannte Techniken zu finden. Somit kann ein großer Bereich möglicher Aufrufe des Hardware-beschleunigten Befehls gefunden werden, und das kann auf eine effizientere, z. B. rechnerisch weniger aufwändige, Weise stattfinden. Diese effektiveren Aufrufe des Hardware-beschleunigten Befehls können gefunden werden, oder Aufrufe können selbst für Berechnungen gefunden werden, für die das vorher nicht machbar war.

[0018] Ein weiterer Vorteil der Verwendung eines Bedingungsauflösers für das Finden möglicher Aufrufe ist, dass es möglich ist, Anforderungen, die ein möglicher Aufruf erfüllen muss, von Präferenzen, die als vorteilhaft betrachtet werden, zu trennen. Insbesondere durch Anpassen der Variablenauswahlstrategie und/oder Werteauswahlstrategie, die durch den Bedingungsauflöser angewandt wird, ist es möglich, Kontrolle über den Lösungsraum während der Bedingungsauflösung zu besitzen, jedoch ohne zu beeinflussen, was es bedeutet, dass eine Lösung gültig ist. Durch Steuern des Bedingungsauflösers ist es möglich, den Bedin-

gungsauflöser zu lenken, so dass er eine spezielle Lösung zuerst findet. Somit kann die Qualität der bestimmten möglichen Aufrufe verbessert sein, und es ist möglich, die Effizienz des Bedingungsauflösers durch effektives Vorschlagen von Teilen potentieller Lösungen zum Ausprobieren weiter zu verbessern, ohne jedoch Teile des Lösungsraums zu verwerfen: Falls der Vorschlag nicht zu einer Lösung führt, kann der Bedingungsauflöser fortfahren, andere mögliche Lösungen zu betrachten.

[0019] Optional können maschinenausführbare Befehle, die die Berechnung implementieren, erzeugt werden. Diese Befehle können den Hardware-beschleunigten Befehl gemäß dem bestimmten möglichen Aufruf aufrufen. Auf diese Weise kann eine effiziente Implementierung der Berechnung, die den Hardware-beschleunigten Befehl verwendet, erhalten werden. Somit kann ein Verfahren zum Kompilieren einer Berechnung in maschinenausführbare Befehle bereitgestellt sein.

[0020] Optional können die Berechnung und der Befehlsdatenflussgraph Knoten umfassen, die Skalaroperationen repräsentieren. Z. B. kann eine Skalaroperation auf einer festen Anzahl von einem oder mehreren separaten Werten (üblicherweise einem oder zwei) arbeiten und gibt eine feste Anzahl von einem oder mehreren separaten Werten (üblicherweise einen) aus. Somit kann das Bedingungserfüllungsproblem Skalaroperationen auf Skalaroperationen abbilden, im Gegensatz zum Arbeiten auf einer Schleifenebene. In einigen Ausführungsformen kann jeder Knoten der Graphen eine Skalaroperation repräsentieren, mit der möglichen Ausnahme kommutativer Reduktionsoperationen, wie an anderer Stelle beschrieben. Wie ebenfalls an anderer Stelle erläutert ist, sind durch Ausführen der Abbildung auf einer Skalarebene implizite Modellierungsannahmen, die andernfalls einschränken können, welche Aufrufe gefunden werden können, minimiert, was ermöglicht, dass Aufrufe mit besserer Leistung identifiziert werden.

[0021] Optional kann der Hardware-beschleunigte Befehl eines oder mehrere aus einer Matrixmultiplikation, einer Faltung, einem Skalarprodukt, einem Matrix-Vektor-Produkt, einer kumulativen Summe, einem Pooling und einem Hadamard-Produkt implementieren. Das sind übliche Operationen, die in großen Berechnungen wiederholt ausgeführt werden und für die Hardware-Beschleunigung verfügbar ist, und somit ist es besonders relevant, sie zu optimieren.

[0022] Allgemein ist die Berechnung eine Berechnung, die Operationen umfasst, von denen erwartet wird, dass sie durch den Hardware-beschleunigten Befehl implementierbar sind. Beispielsweise kann eine Berechnung eine Teilberechnung einer größeren Berechnung sein, wie z. B. die Anwendung eines Faltungsoperators in einer Bewertung eines neuronalen Netzes, der manuell dafür ausgewählt ist, dass er potentiell für die Anwendung des Hardware-beschleunigten Befehls zugänglich ist. Hinsichtlich des Datenflusses können viele verschiedene Arten vorhanden sein, auf die der Befehl in der Berechnung aufgerufen werden kann, jedoch nur einige dieser Arten können die Einschränkungen, die durch den Hardware-beschleunigten Befehl und/oder seine Programmierschnittstelle auferlegt sind, befolgen, und diese möglichen Aufrufe können sich hinsichtlich der Leistung in hohem Maße unterscheiden. Somit ist es wichtig, automatisch fähig zu sein, die möglichen Aufrufe zu finden und/oder aufzulisten.

[0023] Optional kann der Befehlsdatenflussgraph Knoten für nur eine Teilmenge von Ausgaben des Hardware-beschleunigten Befehls enthalten. Sowohl andere Ausgaben des Hardware-beschleunigten Befehls als auch Operationen, die verwendet werden, um diese anderen Ausgaben zu berechnen, können weggelassen sein, was das Bedingungserfüllungsproblem effizienter lösbar macht. Ein bestimmter möglicher Aufruf kann dann auf die vollständige Berechnung, die durch den Befehl basierend auf dem bestimmten möglichen Aufruf ausgeführt wird, verallgemeinert werden. Beispielsweise kann für einen Matrixmultiplikationsbefehl ein Datenflussgraph einer Berechnung eines einzigen Ausgabeelements ausreichend sein, um auf Eingaben und Ausgaben der vollständigen Matrixmultiplikation zu extrapolieren. Eine Teilmenge der Ausgaben, die alle Eingaben des Hardware-beschleunigten Befehls verwendet, kann ausgewählt werden, so dass der mögliche Aufruf alle Eingaben abbildet, was die Extrapolation einfacher macht; oder das Abbilden der Befehlseingaben in die Berechnung kann ebenfalls extrapoliert werden.

[0024] Optional kann die Berechnung eine Berechnung sein, die in einer Bewertung eines neuronalen Netzes angewandt wird. Beispielsweise kann die Berechnung eine Faltung, eine Matrixmultiplikation und/oder eine Pooling-Operation, die z. B. auf die Eingabe eines Schicht eines neuronalen Netzes angewandt wird, umfassen. Die Berechnung kann jeden Knoten der Eingabe der Schicht des neuronalen Netzes berühren. Bewertung eines neuronalen Netzes ist eine Operation, die sowohl während des Trainings als auch während der Verwendung eines neuronalen Netzes stattfinden, und beinhaltet das wiederholte Anwenden von Operationen, die für Hardware-Beschleunigung zugänglich sind, wie z. B. Matrixmultiplikation, und ist somit ein besonders relevantes Optimierungsziel. Außerdem sind hinsichtlich des Datenflusses viele Arten vorhanden,

wie diese Operationen angewandt werden können. Insbesondere ist es, damit das Training eines neuronalen Netzes effektiv ist, typischerweise notwendig, das neuronale Netz mit großen Datenmengen zu trainieren; während die Bewertung neuronaler Netze häufig in Umgebungen mit eingeschränkten Betriebsmitteln stattfindet. In beiden Fällen ist Optimierung von besonderer Wichtigkeit. Außerdem ist es angesichts der großen Vielzahl und konstanten Entwicklung von Schichten neuronaler Netze und Beschleunigungs-Hardware besonders wichtig fähig zu sein, Implementierungen neuer Typen von Schichten neuronaler Netze und/oder für neue Hardware-Ziele zu erhalten.

[0025] Optional kann wenigstens ein Teil der Berechnung durch eine polyedrische Repräsentation spezifiziert sein. Eine polyedrische Repräsentation kann eine Menge von Knoten des Berechnungsdatenflussgraphen symbolisch definieren. Wenn das Bedingungserfüllungsproblem gelöst wird, kann die polyedrische Repräsentation instanziiert werden, um eine konkrete Kennzeichnung dieser Menge von Knoten zu erhalten, und ein Knoten des Befehlsdatenflussgraphen kann dann auf diesen erhaltenen Knoten abgebildet werden.

[0026] Polyedrische Repräsentationen sind an sich als eine Art zum kompakten Repräsentieren schleifenbasierter Programme bekannt. Interessanterweise haben die Erfinder erkannt, dass zusätzlich zum Modellieren von Schleifen eines schleifenbasierten Programms die polyedrische Repräsentation verwendet werden kann, um auch den Datenfluss zu modellieren. Das Bestimmen eines Aufrufs durch direktes Übersetzen polyedrischer Repräsentationen ineinander ist unerwünscht, beispielsweise weil die Repräsentation Modellierungsannahmen wie Schleifen- oder Tensordimensionsreihenfolge repräsentiert und weil das das Definieren expliziter Transformationen der Repräsentationen erfordern kann, mit den zugeordneten Nachteilen, die an anderer Stelle beschrieben sind. Interessanterweise werden durch Verwenden der polyedrischen Repräsentation eines Berechnungsdatenflussgraphen, auf den ein Befehlsdatenflussgraph abgebildet werden soll, diese Nachteile vermieden.

[0027] In Kombination mit den vorliegenden Techniken kann die polyedrische Repräsentation verwendet werden, um Mengen von Polyedern des Berechnungsdatenflussgraphen symbolisch zu definieren. Interessanterweise kann jedoch die Abbildung zwischen dem Befehlsdatenflussgraphen und dem Berechnungsdatenflussgraphen immer noch nicht hinsichtlich Mengen von Knoten, z. B. Mengen von Operationen, ausgeführt werden, jedoch hinsichtlich individueller Knoten, z. B. individueller Operationen. Die polyedrische Notation kann vermeiden, dass der vollständige Berechnungsdatenflussgraph gespeichert werden muss; nur diese individuellen Elemente, auf die von dem Befehlsdatenflussgraph abgebildet wird, müssen instanziiert werden. In vielen Fällen reduziert das die rechnerische und Speicher-Komplexität in hohem Maße.

[0028] Die polyedrische Repräsentation wird typischerweise nicht verwendet, um den Berechnungsdatenflussgraph zu repräsentieren, oder eine polyedrische Repräsentation wird wenigstens nicht während der Bedingungsauflösung verwendet. Die polyedrischen Knoten des Berechnungsdatenflussgraphen werden typischerweise individuell als Variablen des Bedingungserfüllungsproblems gespeichert.

[0029] Optional können die eine oder die mehreren Eingabebedingungen definiert sein, um zu erzwingen, dass Eingaben des Hardware-beschleunigten Befehls ein zulässiges Speicherlayout und/oder Speicherzugriffsmuster aufweisen. Allgemein können der Hardware-beschleunigte Befehl und/oder seine Programmierschnittstelle erzwingen, dass die Eingaben, auf die der Befehl angewandt wird, im Speicher gemäß einer speziellen Anordnung gespeichert sind, z. B. dass die Eingaben einen fortlaufenden Speicherblock bilden, die Speicheradressen auf eine spezielle Weise ausgerichtet sind, usw. Dieses sind Beispiele für Speicherlayout-Anforderungen, die einschränken, auf welche Eingaben der Hardware-beschleunigte Befehl angewandt werden kann. Diese Anforderungen können als Eingabebedingungen repräsentiert sein, um sicherzustellen, dass ihnen gefolgt wird. Ähnlich kann der Hardware-beschleunigte Befehl eine vorgegebene Menge von Zugriffsmustern unterstützen, gemäß denen der Befehl angewandt werden kann, z. B. ob auf die Eingaben gemäß einer gegebenen Schrittweite, einem gegebenen Schablonenmuster usw. zugegriffen wird. Auf diese Weise kann weiter begrenzt werden, auf welche Eingaben der Hardware-beschleunigte Befehl angewandt werden kann.

[0030] Es ist auch möglich, Speicherlayout- und/oder Speicherzugriffsmuster-Einschränkungen, die sich auf Zwischenergebnisse und Ausgaben des Hardware-beschleunigten Befehls in die Menge von Bedingungen beziehen, aufzunehmen, z. B. um Befehle zu repräsentieren, die durch den Befehl und/oder seine Programmierschnittstelle auferlegt sind.

[0031] Optional können eine oder mehrere Bedingungen definiert sein, die erzwingen, dass Paare aus wechselseitig parallelisierbaren Operationen des Hardware-beschleunigten Befehls auf Paare von wechselseitig

parallelisierbaren Operationen der Berechnung abgebildet werden. Paare können als wechselseitig parallelisierbar markiert werden, z. B. durch Aufnehmen eines speziellen Typs von Kante, der die Parallelisierbarkeit angibt, in die Datenflussgraphen. Obwohl es nicht notwendig ist, Paare von Operationen als wechselseitig parallelisierbar zu markieren und das in der Abbildung zu erzwingen, ist es vorteilhaft für die Leistung der Bedingungsauflösung, weil es ermöglicht, dass der Lösungsprozess potentielle Zuweisungen, in denen wechselseitig parallelisierbare Operationen nicht auf wechselseitig parallelisierbare Operationen abgebildet sind, direkt zu eliminieren.

[0032] Interessanterweise haben die Erfinder erkannt, dass es für eine Menge von Operationen des Befehlsdatenflussgraphen, die alle wechselseitig parallelisierbar sind, nicht notwendig ist, Parallelisierbarkeitsbedingungen für jedes separate Paar aufzunehmen. Das Aufnehmen der Bedingung für nur eine Teilmenge von Paaren verbessert die Leistung des Lösers, da die Effizient des Lösers allgemein mit der Anzahl von Bedingungen skaliert; und auch eine Teilmenge dem Löser hilft, die Zuweisungen zu eliminieren. Außerdem kann durch Transitivität das Erzwingen von Parallelisierbarkeitsbedingungen für eine Menge von Paaren des Befehlsdatenflussgraphen implizieren, dass Parallelisierbarkeitsbedingungen für jedes Paar in der transitiven Hülle ebenfalls erfüllt sind. In dem Berechnungsdatenflussgraphen kann Parallelisierbarkeit für jedes Paar von Operationen in einer Menge wechselseitig parallelisierbarer Operationen angegeben sein, um nicht unnötigerweise zu beschränken, welche Zuweisungen vorgenommen werden können. Für den Berechnungsdatenflussgraphen kann das weniger Leistungseinbußen auferlegen, insbesondere wenn dieser Graph während der Lösung nicht vollständig instanziiert wird, z. B. aufgrund der Verwendung einer polyedrischen Repräsentation.

[0033] Optional kann ein Knoten des Befehlsdatenflussgraphen eine kommutative Reduktionsoperation repräsentieren. Beispielsweise kann ein Knoten für eine kommutative Reduktionsoperation, z. B. Summe oder Produkt, repräsentieren, dass diese Operation auf alle eingehenden Kanten des Knotens angewandt wird. Die Verwendung solcher Knoten kann die Anzahl von Knoten und Kanten, die benötigt werden, um übliche Hardware-beschleunigte Operationen, die z. B. Matrixmultiplikation oder Faltung, zu spezifizieren, in hohem Maße reduzieren, ohne die Korrektheit der Datenflussgraphen als Repräsentationen der Berechnung oder des Befehls zu beeinflussen. Sie ermöglicht außerdem flexiblere Abbildung, da der Graph keine spezielle Reihenfolge, in der die Elemente reduziert werden müssen, spezifiziert. Obwohl kommutative Reduktionsoperation implizit definiert sein können, z. B. dadurch, dass der Knoten mehr als zwei eingehende Kanten besitzt, ist es vorzuziehen, Knoten explizit als eine kommutative Reduktionsoperation repräsentierend zu markieren. Beispielsweise kann das durch Aufnehmen einer Eigenkante in den Graphen stattfinden. Eine solche explizite Markierung kann es für den Bedingungsauflöser effizienter machen, Abbildungen für kommutative Reduktionsoperationen zu bestimmen.

[0034] Die Datenflussbedingungen können erzwingen, dass Knoten, die kommutative Reduktionsoperationen des Befehlsdatenflussgraphen repräsentieren, auf entsprechende Knoten des Berechnungsdatenflussgraphen abgebildet werden. Ein entsprechender Knoten kann ein Knoten sein, der eine kommutative Reduktionsoperation der Berechnung repräsentiert, z. B. ein Knoten, der eingehende Kanten für jedes Element, auf dem die kommutative Reduktionsoperation arbeitet, aufweist. Das ist jedoch nicht notwendig. Es ist auch möglich, individuelle Reduktionen der gesamten Reduktionsoperation separat in dem Befehlsdatenflussgraphen zu repräsentieren. In solchen Fällen kann der Knoten des Befehlsdatenflussgraphen auf einen Knoten, der eine individuelle Reduktion repräsentiert, abgebildet werden. Das kann es vereinfachen, Berechnungen zu modellieren, da kein separater Modellierungsaufwand für kommutative Reduktionsoperationen benötigt wird.

[0035] Optional kann beim Lösen eines Bedingungserfüllungsproblems eine Variablenauswahlstrategie verwendet werden, die rückwärts durch den Befehlsdatenflussgraphen propagiert. Es wurde in der Praxis beobachtet, dass für den Typ eines Bedingungserfüllungsproblems, der durch Datenfluss und Eingabebedingungen definiert ist, das Verwenden dieser Variablenauswahlstrategie das größte Propagierungspotential aufweist und somit beim Finden von Lösungen allgemein am effizientesten ist. Es wurde gefunden, dass das insbesondere für Berechnungen von Bewertungen von Schichten neuronaler Netze der Fall ist.

[0036] Optional kann das Bedingungserfüllungsproblem so definiert sein, dass es das Abbilden von Operationen des Hardware-beschleunigten Befehls auf Dummy-Operationen, die die Berechnung nicht beeinflussen, die z. B. zu dem gleichen Berechnungsergebnis führen, erlaubt. Das Lösen des Bedingungserfüllungsproblems kann das Abbilden eines Knotens auf eine Dummy-Operation umfassen, z. B. kann ein Knoten auf eine Operation abgebildet werden, die nicht in dem Berechnungsdatenflussgraph enthalten ist, oder mehrere Knoten können auf denselben Knoten in dem Berechnungsdatenflussgraphen abgebildet werden. Durch

Erlauben von Dummy-Operationen wird die Flexibilität beim Finden möglicher Aufrufe gesteigert. Insbesondere kann es erlauben, den Hardware-beschleunigten Befehl auf Arbeitslasten anzuwenden, die anderweitig durch den Befehl nicht ausführbar sind. Beispielsweise kann ein relevanter Teil der Berechnung kleiner sein als der Hardware-beschleunigte Befehl, z. B. nur kann eine Teilmenge der Knoten in dem Befehlsdatenflussgraphen auf Knoten des Berechnungsdatenflussgraphen abgebildet sein. Beispielsweise kann der Befehl eine Summe von vier Elementen berechnen, während die Berechnung eine Summe von nur drei Elementen erfordert. Durch Aufnehmen von Dummy-Operationen, die die Berechnung nicht beeinflussen, z. B. Hinzufügen neutraler Elemente für Operationen, die durch den Befehl ausgeführt werden, wie z. B. Null für Addition oder Eins für Multiplikation, kann der relevante Teil der Berechnung immer noch durch den Hardware-beschleunigten Befehl ausgeführt werden.

[0037] Optional können eine oder mehrere Bedingungen in dem Bedingungserfüllungsproblem enthalten sein, die erzwingen, dass Eingaben des Hardware-beschleunigten Befehls ein Hyper-Rechteck bilden. Diese Bedingung erlaubt, dass der Bedingungsauflöser einen großen Teil des Suchraums schnell ausschließt, z. B. bereits nachdem nur einige wenige Zuweisungen zu Befehlsdatenflussknoten vorgenommen worden sind. Das verbessert die Effizienz der Suche in hohem Maße und garantiert außerdem, dass reguläre Lösungen bereitgestellt werden, die leichter verallgemeinert werden können, um Aufrufe für den Rest der Berechnung zu erhalten. Da in vielen praktischen Anwendungen typische Aufrufe des Hardware-beschleunigten Befehls Hyper-Rechteck-förmige Eingaben aufweisen, führen diese Bedingungen in solchen Fällen nicht dazu, dass mögliche Aufrufe mit guter Leistung übersehen werden.

[0038] Optional können mehrere Lösungen für das Bedingungserfüllungsproblem bestimmt werden. Ihre Leistung kann dann bewertet werden, und basierend auf dieser Leistungsbewertung kann eine Lösung aus den mehreren Lösungen ausgewählt werden. Der Bedingungsauflöser kann mögliche Aufrufe finden, die die Bedingung erfüllen, optimiert jedoch typischerweise die gefundenen Lösungen nicht hinsichtlich der Leistung (obwohl der Bedingungsauflöser konfiguriert sein kann, spezielle Lösungen zu priorisieren, wie hier ebenfalls beschrieben ist). Der Bedingungsauflöser kann jedoch konfiguriert sein, mehrere Lösungen für das Bedingungserfüllungsproblem auszugeben oder sogar alle möglichen Lösungen aufzulisten. Durch Bewerten der Leistung dieser mehreren Lösungen ist es möglich, eine Lösung auszuwählen, die die gewünschte Leistung aufweist. Verschiedene Strategien können eingesetzt werden, z. B. kann eine feste Anzahl von Lösungen bestimmt werden, oder eine feste Zeitdauer kann verwendet werden, um Lösungen zu finden, und die gefundenen Lösungen können dann bewertet werden; oder Lösungen können iterativ bestimmt und bewertet werden, bis eine Lösung mit einer wünschenswerten Leistung gefunden ist oder bis keine weitere Verbesserung in einer speziellen Zeitspanne gefunden wird; usw.

[0039] Es ist jedoch durch Fachleute zu verstehen, dass zwei oder mehr der vorstehend genannten Ausführungsformen, Implementierungen und/oder optionalen Aspekte der Erfindung auf jede als nützlich erachtete Weise kombiniert sein können.

[0040] Modifikationen und Variationen des Systems und/oder irgendeines computerlesbaren Medium, die den beschriebenen Modifikationen und Variationen eines entsprechenden computerimplementierten Verfahrens entsprechen, können durch einen Fachmann auf der Basis der vorliegenden Beschreibung ausgeführt werden.

Figurenliste

[0041] Diese und andere Aspekte der Erfindung werden aus den Ausführungsformen, die als Beispiel in der folgenden Beschreibung und mit Bezug auf die begleitenden Zeichnungen beschrieben sind, offensichtlich und mit Bezug darauf weiter erläutert; es zeigen:

Fig. 1 ein System zum Implementieren einer Berechnung;

Fig. 2 ein System zum Ausführen einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls;

Fig. 3 ein ausführliches Beispiel dafür, wie eine Berechnung zu implementieren ist;

Fig. 4a ein ausführliches Beispiel eines Datenflussgraphen;

Fig. 4b ein ausführliches Beispiel einer polyedrischen Repräsentation, die einen Datenflussgraphen definiert;

Fig. 5a-5d ein ausführliches Beispiel für das Anwenden einer Hyper-Rechteck-Bedingung;

Fig. 6 ein computerimplementiertes Verfahren zum Implementieren einer Berechnung;

Fig. 7 ein computerlesbares Medium, das Daten umfasst.

[0042] Es wird darauf hingewiesen, dass die Figuren rein schematisch sind und nicht maßstabsgerecht gezeichnet sind. In den Figuren können Elemente, die bereits beschriebenen Elementen entsprechen, die gleichen Bezugszeichen aufweisen.

AUSFÜHRLICHE BESCHREIBUNG VON AUSFÜHRUNGSFORMEN

[0043] **Fig. 1** zeigt ein System 100 zum Implementieren einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls eines Prozessorsystems. Beispielsweise kann die Berechnung unter Verwendung des Prozessorsystems 240 von System 200 von **Fig. 2** implementiert sein, z. B. kann das System 100 Befehle produzieren, die durch das Prozessorsystem 240 ausführbar sind, oder wenigstens einen Aufruf eines Hardware-beschleunigten Befehls, der durch das Prozessorsystem 240 unterstützt wird, der verwendet werden kann, um die Berechnung auf dem Prozessorsystem 240 auszuführen, bestimmen. Die Systeme 100 und 200 können kombiniert sein, z. B. kann das Prozessorsystem 140 von System 100 (weiter unten diskutiert) den Hardware-beschleunigten Befehl unterstützen.

[0044] Das System 100 kann eine Datenschnittstelle zum Zugreifen auf Berechnungsdaten 030 und Befehlsdaten 040 umfassen. Die Berechnungsdaten 030 können einen Berechnungsdatenflussgraphen, der die Berechnung repräsentiert, definieren. Ein Knoten des Berechnungsdatenflussgraphen kann eine Eingabe oder eine Operation der Berechnung repräsentieren. Die Befehlsdaten 040 können einen Befehlsdatenflussgraphen, der den Hardware-beschleunigten Befehl repräsentiert, definieren. Ein Knoten des Befehlsdatenflussgraphen kann eine Eingabe oder eine Operation des Hardware-beschleunigten Befehls repräsentieren.

[0045] Der Befehlsdatenflussgraph kann wenigstens 16, wenigstens 32 oder wenigstens 128 Knoten aufweisen. Die Anzahl von Knoten kann höchstens 2048 oder höchstens 4096 sein. Der Berechnungsdatenflussgraph weist typischerweise eine größere Anzahl von Knoten auf als der Befehlsdatenflussgraph, z. B. um einen Faktor 2^6=64, 2^10=1024 oder 2^12=4096. Die Anzahl von Knoten des Befehlsdatenflussgraphen kann beispielsweise wenigstens 2^16=65536, 2^32=4294967296 oder 2^48 sein.

[0046] Beispielsweise, wie auch in **Fig. 1** dargestellt ist, kann die Datenschnittstelle auch durch eine Daten-Speicherschnittstelle 120 gebildet sein, die auf die Daten 030, 040 aus einem Datenspeicher 021 zugreift. Beispielsweise kann die Daten-Speicherschnittstelle 120 eine Speicherschnittstelle oder eine persistente Speicherschnittstelle sein, z. B. eine Festplatte oder eine SSD-Schnittstelle, jedoch auch eine Schnittstelle eines persönlichen, lokalen oder Weitbereichs-Netzes wie z. B. eine Bluetooth-, Zigbee- oder Wi-Fi-Schnittstelle oder eine Ethernet- oder Lichtleiter-Schnittstelle. Der Datenspeicher 021 kann ein interner Datenspeicher des Systems 100 sein, wie z. B. ein Festplattenlaufwerk oder eine SSD, kann jedoch auch ein externer Datenspeicher sein, z. B. ein über ein Netz zugänglicher Datenspeicher. In einigen Ausführungsformen kann auf die Daten 030, 040 jeweils von einem unterschiedlichen Datenspeicher zugegriffen werden, z. B. über ein anderes Teilsystem der Daten-Speicherschnittstelle 120. Jedes Teilsystem kann von einem Typ sein, wie er vorstehend für die Daten-Speicherschnittstelle 120 beschrieben ist.

[0047] Die Datenschnittstelle, z. B. die Datenspeicherschnittstelle 120, kann auch zum Zugreifen auf einen bestimmten möglichen Aufruf des Hardware-beschleunigten Befehls oder, wie in der Figur gezeigt, maschinenausführbare Befehle 050, die die Berechnung durch Aufrufen des Hardware-beschleunigten Befehls gemäß einem bestimmten möglichen Aufruf implementieren, dienen. Diese Befehle 050 können zum Gebrauch durch ein Prozessorsystem, das die Berechnung ausführt, z. B. das Prozessorsystem 240 von **Fig. 2**, dienen.

[0048] Das System 100 kann ferner ein Prozessorteilsystem 140 umfassen, das konfiguriert sein kann, während des Betriebs des Systems 100 die Berechnungsdaten 030 und die Befehlsdaten 040 zu erhalten.

[0049] Das Prozessorteilsystem 140 kann ferner konfiguriert sein, basierend auf den Berechnungsdaten und den Befehlsdaten ein Bedingungserfüllungsproblem zu definieren. Ein Bedingungserfüllungsproblem kann durch eine Menge von Variablen, eine Menge von Domänen und eine Menge von Bedingungen definiert sein. Mit „Definieren“ des Bedingungserfüllungsproblems ist gemeint, dass Informationen über das Bedingungserfüllungsproblem in einer Form verfügbar gemacht werden, die es ermöglicht, dass das Bedingungserfüllungsproblem durch einen Bedingungsauflöser gelöst wird. Es ist beispielsweise nicht notwendig, dass Mengen von Variablen, Domänen und/oder Bedingungen explizit und individuell gespeichert sind, sie können

z. B. implizit definiert und/oder bequem bewertet werden, wie an sich bekannt ist. Außerdem kann ein Teil des oder das gesamte Bedingungserfüllungsproblem in der Form der Berechnungsdaten 030 und/oder der Befehlsdaten 040 verfügbar gemacht werden, z. B. können die Befehlsdaten 040 die Menge von Knoten aus Knoten des Befehlsdatenflussgraphen und dadurch die Menge von Variablen des Bedingungserfüllungsproblems umfassen oder sie auf andere Weise definieren.

[0050] Eine Lösung für das Bedingungserfüllungsproblem kann einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung repräsentieren. Das Bedingungserfüllungsproblem kann Knoten des Berechnungsdatenflussgraphen Knoten des Befehlsdatenflussgraphen zuweisen. Das Bedingungserfüllungsproblem kann eine oder mehrere Datenflussbedingungen umfassen, die erzwingen, dass die zugewiesenen Knoten des Berechnungsdatenflussgraphen einen dem Befehlsdatenflussgraphen äquivalenten Datenfluss aufweisen. Das Bedingungserfüllungsproblem kann ferner eine oder mehrere Eingabebedingungen umfassen, die einschränken, welche Knoten des Berechnungsdatenflussgraphen den Eingaben des Hardware-beschleunigten Befehls zugewiesen werden können. Diese letzteren Einschränkungen können durch den Hardware-beschleunigten Befehl und/oder seine Programmierschnittstelle auferlegt werden. Die Bedingungen können z. B. dadurch definiert werden, dass sie automatisch basierend auf den Berechnungsdaten 030 und/oder den Befehlsdaten 040 definiert werden, durch Empfangen manueller Definitionen der Bedingungen oder eine Kombination aus beiden.

[0051] Das Prozessorteilsystem 140 kann ferner konfiguriert sein, das Bedingungserfüllungsproblem zu lösen, um einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung zu bestimmen. Das Prozessorteilsystem kann ferner konfiguriert sein, Daten, die den möglichen Aufruf definieren, auszugeben.

[0052] Das System 100 kann ferner eine Ausgabeschnittstelle zum Ausgeben von Daten, die den möglichen Aufruf definieren, beispielsweise die Lösung für das Bedingungserfüllungsproblem oder maschinenausführbare Befehle 050, die von der Lösung abgeleitet sind, umfassen. Beispielsweise kann, wie auch in **Fig. 1** dargestellt ist, die Ausgabeschnittstelle durch die Datenschnittstelle 120 gebildet sein, wobei diese Schnittstelle in diesen Ausführungsformen eine Eingabe/Ausgabe- (‚I/O'-) Schnittstelle ist, über die die Daten in dem Datenspeicher 021 gespeichert werden können. In einigen Ausführungsformen kann die Ausgabeschnittstelle von der Daten-Speicherschnittstelle 120 separat sein, kann jedoch allgemein von einem Typ sein, wie er vorstehend für die Datenspeicherschnittstelle 120 beschrieben ist. Die Ausgabeschnittstelle kann außerdem irgendein anderer Typ einer Ausgabeschnittstelle sein, z. B. eine Netzschnittstelle.

[0053] **Fig. 2** zeigt ein System 200 zum Ausführen einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls.

[0054] Das System 200 kann eine Datenschnittstelle 220 zum Zugreifen auf Befehle 050 zum Ausführen der Berechnung umfassen. Beispielsweise können die Befehle 050 durch das System 100 von **Fig. 1** oder basierend auf einem möglichen Aufruf, der durch das System 100 bestimmt ist, erzeugt werden. Beispielsweise, wie auch in **Fig. 2** dargestellt ist, kann die Datenschnittstelle auch durch eine Datenspeicherschnittstelle 220 gebildet sein, die auf die Befehle 050 aus einem Datenspeicher 022 zugreift. Im Allgemeinen können die Datenschnittstelle 220 und der Datenspeicher 022 vom gleichen Typ sein, wie er mit Bezug auf **Fig. 1** für die Datenschnittstelle 120 und den Datenspeicher 021 beschrieben ist.

[0055] Das System 200 kann ferner ein Prozessorteilsystem 240 umfassen. Das Prozessorteilsystem 240 kann einen Hardware-beschleunigten Befehl unterstützen. Allgemein ist ein Hardware-beschleunigter Befehl ein Befehl, der durch das Prozessorteilsystem 240 unterstützt wird, der eine dedizierte Aufgabe ausführt. Die dedizierte Aufgabe ist kein elementarer CPU-Befehl wie z. B. eine Skalaraddition, Skalarmultiplikation, bitweise und so weiter. Stattdessen kann ein einziger Aufruf für den Hardware-beschleunigten Befehl dem Ausführen mehrerer dieser elementaren Befehle entsprechen; das Ausführen des Hardware-beschleunigten Befehls ist jedoch typischerweise schneller. Das Ausführen des Hardware-beschleunigten Befehls kann die Leistung der Berechnung verbessern, führt z. B. zu einer verringerten Latenz und/oder erhöhtem Durchsatz.

[0056] Der Hardware-beschleunigte Befehl weist typischerweise mehrere skalare Eingaben auf (z. B. numerische Werte wie z. B. Ganzzahl oder Gleitkomma), z. B. wenigstens 4, wenigstens 16 oder wenigstens 64. Ein Hardware-beschleunigter Befehl kann eine einzige Ausgabe aufweisen (z. B. einen numerischen Wert wie z. B. Ganzzahl oder Gleitkomma), kann jedoch auch mehrere solche Ausgaben aufweisen, z. B. wenigstens 4, wenigstens 16 oder wenigstens 64. Der Hardware-beschleunigte Befehl kann einer speziellen Anzahl N elementarer CPU-Befehle entsprechen. Beispielsweise kann ein elementarer CPU-Befehl einem Knoten in

dem Befehlsdatenflussgraphen entsprechen. Somit kann N gleich der Anzahl von Knoten des Befehlsdatenflussgraphen sein, oder N kann größer sein, falls der Befehlsdatenflussgraph nur einen Teil des Befehls modelliert. Beispielsweise kann N wenigstens 2^12=4096, wenigstens 2^16=65536 oder wenigstens 2^24=16777216 sein.

[0057] Allgemein kann das Prozessorteilsystem 240 durch eine einzige zentrale Verarbeitungseinheit (CPU), jedoch auch durch eine Kombination oder ein System aus solchen CPUs und/oder anderen Typen von Verarbeitungseinheiten, verwirklicht sein. Der Hardware-beschleunigte Befehl kann durch das Prozessorsystem 240 auf verschiedene Arten unterstützt werden. In einigen Ausführungsformen ist der Hardware-beschleunigte Befehl ein Befehl der CPU des Prozessorteilsystems 240, z. B. kann der Hardware-beschleunigte Befehl on-Chip implementiert sein. In einigen Ausführungsformen kann, wie in der Figur dargestellt, der Hardware-beschleunigte Befehl durch einen Hardware-Beschleuniger 241, der von der CPU separat ist, z. B. eine spezialisierte Verarbeitungseinheit, ausgeführt werden. Der Hardware-Beschleuniger 241 kann beispielsweise ein Coprozessor; ein Prozessor mit anwendungsspezifischem Befehlssatz (ASIP); ein im Feld programmierbares Gatter-Array (FPGA); eine anwendungsspezifische integrierte Schaltung (ASIC); eine komplexe programmierbare Logikvorrichtung (CPLD); oder ein anderer Typ von programmierbarer Logik sein. Beispielsweise kann der Hardware-Beschleuniger 241 ein VTA-Beschleuniger sein, wie er in T. Moreau u. a., „A Hardware-Software Blueprint for Flexible Deep Learning Specialization“ (verfügbar bei https://arxiv.org/abs/1807.04188 und hier durch Bezugnahme mit aufgenommen) diskutiert ist.

[0058] Die Befehle 050 können den Hardware-beschleunigten Befehl aufrufen. Das Prozessorteilsystem 240 kann konfiguriert sein, die Berechnung, die das Aufrufen des Hardware-beschleunigten Befehls gemäß den Befehlen 050 umfasst, z. B. durch Aufrufen des Hardware-Beschleunigers 241, auszuführen. Weil ein bestimmter möglicher Aufruf verwendet wird, kann die Effizienz der Berechnung verbessert sein.

[0059] Optional kann die Berechnung Teil der Anwendung eines trainierten Maschinenlernmodells sein, z. B. eines tiefen neuronalen Netzes. Das Prozessorteilsystem 240 kann konfiguriert sein, eine Eingabe für das Maschinenlernmodell von einem Sensor über eine Sensorschnittstelle, z. B. einem Bildsensor, einem Lidarsensor, einem Radarsensor, einem Drucksensor, einem Inhaltstemperatursensor usw. zu erhalten. Das Prozessorteilsystem 240 kann konfiguriert sein, das Maschinenlernmodell auf die Eingabe anzuwenden, um eine Ausgabe zu erhalten. Basierend auf der Ausgabe können Steuerdaten abgeleitet und für einen Aktor, z. B. einen elektrischen, hydraulischen, pneumatischen, thermischen, magnetischen und/oder mechanischen Aktor, über eine Aktorschnittstelle bereitgestellt werden. Beispielsweise kann das System 200 verwendet werden, um einen Roboter, ein autonomes Fahrzeug und dergleichen zu steuern.

[0060] Im Allgemeinen kann jedes System, das in dieser Spezifikation beschrieben ist, das, ohne jedoch darauf beschränkt zu sein, das System 100 von **Fig. 1** und das System 200 von **Fig. 2** enthält, als eine oder in einer einzelnen Vorrichtung oder Einrichtung wie z. B. einem Arbeitsplatzrechner oder einem Server verwirklicht sein. Die Vorrichtung kann eine eingebettete Vorrichtung sein. Die Vorrichtung oder Einrichtung kann einen oder mehrere Mikroprozessoren umfassen, die geeignete Software ausführen. Die Beispiele, in dieser Hinsicht für das Prozessorsystem 240 von **Fig. 2** gegeben sind, treffen auch für das Prozessorsystem 140 von **Fig. 1** zu. Die Software, die auf den Prozessorsystemen abläuft, kann heruntergeladen und/oder in einem entsprechenden Speicher, z. B. einem flüchtigen Speicher wie z. B. RAM oder einem nichtflüchtigen Speicher wie z. B. Flash, gespeichert sein. Allgemein kann jede Funktionseinheit der jeweiligen Systeme 100, 200 in der Form einer Schaltung implementiert sein. Es wird darauf hingewiesen, dass die gleichen Überlegungen und Implementierungsoptionen im Allgemeinen für das System 200 wie für das System 100 von **Fig. 1** gelten können, sofern nicht anders angemerkt.

[0061] **Fig. 4a** zeigt ein ausführliches und dennoch nicht einschränkendes Beispiel eines Datenflussgraphen. Ein Datenfluss ist eine Repräsentation einer Berechnung hinsichtlich Operationen, in die die Berechnung unterteilt werden kann, und Datenflussrelationen zwischen diesen Operationen, Insbesondere kann der Graph Knoten und Kanten umfassen. Knoten können Eingaben und Operationen der Berechnung repräsentieren. Kanten können Datenflussrelationen zwischen den Knoten repräsentieren. Das enthält Eingabe/Ausgabe-Relationen (hier als Kanten vom sequenziellen Typ bezeichnet), kann jedoch optional auch andere Typen von Relationen enthalten. Neben diesen Knoten und Kanten können auch andere Knoten und Kanten in dem Graphen vorhanden sein. Datenflussgraphen sind typischerweise gerichtet, beinhalten z. B. wenigstens eine gerichtete Kante. Ein Datenflussgraph kann ein verbundener Graph sein, was jedoch streng genommen nicht notwendig ist.

[0062] Für anschauliche Zwecke zeigt die Figur einen Datenflussgraphen der Berechnung $\sum_{i=1}^{4} x_i \cdot y_i$ Beispielsweise findet diese Berechnung als die Berechnung eines Ausgabeelements einer 4x4-Matrixmultiplikation statt. Dieses Beispiel kann beispielsweise als ein Berechnungsdatenflussgraph oder ein Befehlsdatenflussgraph oder als ein Teil davon verwendet werden. Kreise repräsentieren Knoten des Graphen. Pfeile bezeichnen Kanten des Graphen.

[0063] Ein oder mehrere Knoten des Datenflussgraphen können Eingaben der Berechnung repräsentieren. Diese Knoten sind hier auch als Datenknoten bezeichnet. Typischerweise weisen Datenknoten nur ausgehende Kanten vom sequenziellen Typ auf, z. B. keine eingehenden Kanten irgendeines Typs oder ausgehende Kanten anderer Typen. Die Figur zeigt die Datenknoten 411, 412, 413 und 414, die den jeweiligen Eingaben x1, x2, x3 und x4 der durch den Graphen repräsentierten Berechnung entsprechen; und die Datenknoten 421, 422, 423 und 424, die den jeweiligen Eingaben y1, y2, y3, y4 entsprechen. Die Figur zeigt einen weiteren Eingabeknoten 461, der dem konstanten Wert 0 entspricht.

[0064] Ein oder mehrere Knoten des Datenflussgraphen können Operationen der Berechnung repräsentieren. Diese Knoten sind hier auch als Operationsknoten bezeichnet. Ein Operationsknoten kann dementsprechend gekennzeichnet sein, welche Operation er ausführt. Obwohl Datenflussgraphen an sich typischerweise verwendet werden können, um irgendeinen Typ einer Operation zu repräsentieren, sind Operationen hier typischerweise Skalaroperationen, z. B. eine Operation, die auf einer festen Anzahl numerischer Werte (z. B. eine Skalaraddition, eine Skalarmultiplikation, eine Skalardivision, eine Negation) oder nicht numerischer Werte (z. B. ein bitweises AND, ein bitweises XOR, ein bitweises NOT usw.) ausgeführt werden. Beispielsweise kann ein Skalarwert ein Wert mit einer Repräsentation fester Größe im Speicher sein, z. B. höchstens 256, höchstens 128, höchstens 64 oder höchstens 32 Bits.

[0065] Typischerweise weist ein Operationsknoten eine oder mehrere eingehende sequenzielle Kanten auf, die den Daten entsprechen, die durch die Operation verbraucht werden. Beispielsweise zeigt die Figur einen Operationsknoten 431, der eine Skalarmultiplikation der Eingaben X1, 411 und y1, 421 repräsentiert; einen Operationsknoten 432, der eine Skalarmultiplikation der Eingaben x2, 412 und y2, 422 repräsentiert; einen Operationsknoten 433, der eine Skalarmultiplikation der Eingaben x3, 413 und y3, 423 repräsentiert; und einen Operationsknoten 434, der eine Skalarmultiplikation der Eingaben x4, 414 und y4, 424 repräsentiert. Wie in diesem Beispiel kann die Operation eine kommutative Operation sein, wobei in diesem Fall die Reihenfolge der Eingaben der Operation typischerweise nicht explizit repräsentiert ist; es ist jedoch auch möglich, die Reihenfolge der Eingaben zu repräsentieren, z. B. für nicht kommutative Operationen, z. B. durch Kennzeichnen der eingehenden sequenziellen Kanten.

[0066] Interessanterweise kann ein Operationsknoten verwendet werden, um eine kommutative Reduktionsoperation wie z. B. Additionen oder Multiplikation zu repräsentieren. Beispielsweise repräsentiert der Knoten 441 eine Addition seiner eingehenden Knoten, der Operationsknoten 431-434, und der Knoten 461 repräsentiert den Initialisierungswert der Operation. Hinzufügen eines Knotens für den Initialisierungswert ist optional, z. B. kann angenommen werden, dass der Initialisierungswert, sofern er nicht explizit gegeben ist, gleich dem neutralen Element der Operation ist, z. B. Null oder Eins. Ein Initialisierungswert der Operation kann auf einen gleichen Initialisierungswert der Berechnung abgebildet werden. Das Verwenden eines einzigen Knotens 441, um diese Operation zu repräsentieren, weist mehrere Vorteile auf. Es vermindert die Notwendigkeit, separate Knoten für separate paarweise Anwendungen der Operationen, z. B. Addieren der Werte 431, 432; Addieren des Wertes 433 zu dem Ergebnis und Addieren des Werts 434 zu diesem Ergebnis, aufzunehmen. Das steigert die Effizienz und verhindert außerdem, dass der Datenflussgraph erzwingt, dass die Addition in einer speziellen Reihenfolge ausgeführt wird, z. B. durch Verbieten, die Summe durch zuerst Addieren der Werte 433 und 434; dann Addieren 432; und dann Addieren des Werts 431 zu bestimmen. Knoten, die kommutative Reduktionsoperationen repräsentieren, können in einem Befehlsdatenflussgraphen, in einem Berechnungsdatenflussgraph oder in beiden verwendet werden.

[0067] Wie in diesem Beispiel gezeigt ist, kann ein Knoten in dem Graphen markiert sein, dass er eine kommutative Reduktionsoperation repräsentiert. In diesem Beispiel ist das durch Hinzufügen einer Eigenkante 451 zu dem Knoten 441 erfolgt. Andere Arten zum Markieren sind ebenfalls möglich, z. B. durch Kennzeichnen des Knotens, dass er eine kommutative Reduktionsoperation repräsentiert. Dieses Markieren besitzt den Vorteil, dass es für den Bedingungsauflöser effizienter wird, Abbildungen für kommutative Reduktionsoperationen zu bestimmen, z. B. wenn er wählt, auf welchen Knoten eine kommutative Reduktionsoperation abzubilden ist, kann der Bedingungsauflöser fähig sein, Knoten, die keine Eigenkante aufweisen, einfacher zu verwerfen als z. B. durch Versuchen, Eingaben der Operationen abzubilden.

[0068] Die Verwendung von Eigenkanten zum Markieren von kommutativen Reduktionsoperationen ist besonders günstig in dem Fall, in dem kommutative Reduktionsknoten in dem Befehlsdatenflussgraphen, jedoch nicht in dem Berechnungsdatenflussgraph verwendet werden. Beispielsweise kann, wenn ein kommutative Reduktionsknoten A des Befehlsdatenflussgraphen auf einen entsprechenden Knoten B des Berechnungsdatenflussgraphen abgebildet wird, die Eigenkante auf eine entsprechende Kante des Berechnungsdatenflussgraphen zwischen nachfolgenden Anwendungen der Reduktionsoperation abgebildet werden. Das Erzwingen der Existenz einer solchen entsprechenden Kante kann es einfacher machen, die Abbildung zwischen den beiden Graphen zu detektieren.

[0069] Ein Operationsknoten kann eine oder mehrere ausgehende sequenzielle Kanten zu anderen Operationen, die das Ergebnis der Operation verwenden, produzieren. Knoten, die keine ausgehenden sequenziellen Kanten zu anderen Operationen aufweisen (jedoch mögliche ausgehende Eigenkanten zu sich selbst), können Ausgaben der Berechnung, die durch den Graphen repräsentiert ist, repräsentieren. Beispielsweise kann der Knoten 441 eine Ausgabe repräsentieren. Somit kann in einem Datenflussgraphen eine Ausgabe mit der Operation, die zu ihr führt, gleichgesetzt werden. Es ist außerdem möglich, Ausgaben explizit zu markieren, z. B. durch Kennzeichnen. Ausgabeknoten können mit zusätzlichen Informationen gekennzeichnet sein, die z. B. eine Form und/oder einen Datentyp der Ausgabe repräsentieren, die verwendet werden können, um zwischen Berechnungen und Befehlen abzugleichen.

[0070] Optional können Paare von Operationen des Datenflussgraphen als wechselseitig parallelisierbar markiert sein. Das kann z. B. durch Hinzufügen einer Kante zwischen dem Paar von Knoten vorgenommen werden (in dieser Figur nicht gezeigt). Solche Kanten sind hier als räumliche Kanten bezeichnet. Insbesondere können räumliche Kanten zwischen Knoten, die die gleiche Operation, jedoch einen anderen Teil der Berechnung, ausführen, z. B. für ein anderes Ausgabeelement, hinzugefügt werden. Räumliche Kanten können wieder verwendet werden, um den Bedingungsauflöser dabei zu unterstützen, wechselseitig parallelisierbare Operationen auf wechselseitig parallelisierbare Operationen abzubilden, wie hier beschrieben ist.

[0071] Eine Reduktion der Anzahl von räumlichen Knoten und somit eine Verbesserung der Effizienz kann für eine Menge von wechselseitig parallelisierbaren Operationen durch Aufnehmen nur einer Teilmenge der Menge von Kanten erreicht werden. Beispielsweise kann, anstatt dass eine Menge von k Knoten vollständig durch räumliche Kanten verbunden sind, die Anzahl der Kanten durch Ausdünnen der Verbindungen, während dieselbe verbundene Komponente beibehalten wird, reduziert werden. Beispielsweise kann eine Sternkonfiguration gewählt werden, wo ein innerer Knoten durch räumliche Kanten mit k-1 Blättern verbunden ist. Durch Transitivität können Parallelisierbarkeitsinformationen beibehalten werden.

[0072] Mathematisch kann ein Datenflussgraph, wie er hier verwendet ist, beispielsweise formalisiert sein als ein gekennzeichneter gerichteter Graph, der als $G = (N, E, l)$ definiert ist, wobei N die Menge der Knoten ist, die Menge gerichteter Kanten $E \subseteq NxN$ ist, und l() eine Funktion ist, die Kennzeichen aus der Menge $L_N \cup L_E$ zuweist. L_N = {{Operation}, {Data}} ist die Menge von Knotenkennzeichenklassen, die z. B. Tensorformen, Datentypen und/oder Arithmetikoperationen enthält. L_E = {Spatial, Sequential} ist die Menge von Kantenkennzeichen.

[0073] In verschiedenen Aspekten kann ein Datenflussgraph, der als der Berechnungsdatenflussgraph bezeichnet ist, verwendet werden, um eine Berechnung zu repräsentieren. Ein Datenflussgraph, der als der Befehlsdatenflussgraph bezeichnet ist, kann verwendet werden, um einen Befehl zu repräsentieren. Diese Datenflussgraphen können das vollständige Eingabe/Ausgabe-Verhalten der Berechnung oder des Befehls repräsentieren, das z. B. alle Eingaben und Ausgaben der Berechnung oder des Befehls und Knoten, die modellieren, wie die Ausgaben aus den Eingaben folgen, enthält. In einigen Fällen ist es auch möglich, nur eine Teilmenge der Berechnung und/oder des Befehls zu modellieren. Beispielsweise kann der Befehlsdatenflussgraph nur Operationen und/oder Eingaben repräsentieren, die für die Berechnung einer Teilmenge seiner Ausgaben verwendet werden, wobei die restlichen Eingaben, Operationen und/oder Ausgaben durch Extrapolation bestimmt werden, wie auch an anderer Stelle beschrieben ist.

[0074] Es ist auch nicht notwendig, dass Operationen der Graphen explizit durch die Berechnung oder den Hardware-beschleunigten Befehl ausgeführt werden, z. B. kann eine Additionen oder Multiplikation in einem Befehlsdatenflussgraphen separat modelliert werden, obwohl sie durch die Hardware durch ein kombinierte Multiplizieren-und-Addieren ausgeführt werden. Somit entsprechen Knoten, die Zwischenergebnisse des Befehls oder der Berechnung repräsentieren, nicht notwendigerweise Werten, die an einem Punkt während des Befehls oder der Berechnung explizit berechnet oder gespeichert werden. Die Eingaben und Ausgaben werden jedoch in einer Implementierung typischerweise explizit berechnet.

[0075] In verschiedenen Aspekten werden Daten verwendet, die einen Datenflussgraphen, z. B. einen Berechnungsdatenflussgraphen oder einen Befehlsdatenflussgraph, repräsentieren. Durch „Definieren“ des Datenflussgraphen ist gemeint, dass die Mengen von Knoten und Kanten aus den Daten ableitbar sind, oder zumindest dass für eine/n gegebenen Knoten oder Kante bestimmt werden kann, ob der Knoten oder die Kante zu dem Graphen gehört. Insbesondere ist es nicht notwendig, dass die Daten selbst explizite Datenelemente, die individuellen Knoten und Kanten entsprechen, beinhalten. Beispielsweise ist es auch möglich, Mengen von Knoten und Kanten symbolisch zu definieren, wie auch mit Bezug auf **Fig. 4b** diskutiert ist, oder Kanten können implizit definiert sein, z. B. kann das Vorhandensein einer Kante eines speziellen Typs durch die Daten durch Angeben von Paaren von Knoten, die die Kante nicht aufweisen, definiert sein. usw. Verschiedene Alternativen sind möglich.

[0076] **Fig. 3** zeigt ein ausführliches, dennoch nicht einschränkendes, Beispiel dafür, wie eine Berechnung unter Verwendung eines Hardware-beschleunigten Befehls zu implementieren ist.

[0077] In der Figur gezeigt ist ein Befehlsdatenflussgraph (Befehls-DFG) IDG, 330, der den Hardware-beschleunigten Befehl, mit anderen Worten die Funktionalität oder Berechnung, die der Hardware-beschleunigte Befehl ausführt, repräsentiert. Beispielsweise kann der Befehls-DFG IDG wie mit Bezug auf **Fig. 4a** beschrieben sein. Insbesondere können die Knoten des Graphen IDG Eingaben und Operationen des Hardware-beschleunigten Befehls repräsentieren.

[0078] In der Figur ist auch ein Berechnungs-DFG CDG, 340, gezeigt, der die Berechnung repräsentiert, die unter Verwendung des Hardware-beschleunigten Befehls zu implementieren ist, z. B. wie in **Fig. 4a**. Knoten des Graphen CDG können Eingaben und Operationen der Berechnung repräsentieren. Der Befehlsdatenflussgraph IDG und der Berechnungsdatenfluss CDG sind unterschiedlich, besitzen z. B. keine gemeinsamen Knoten oder Kanten.

[0079] Der Befehls- und der Berechnungs-DFG sind zwei unterscheidbare Graphen, die jeweils durch entsprechende Berechnungs- und Befehlsdaten (nicht gezeigt) repräsentiert sein können. Wie diskutiert können diese Daten implizit oder explizit die Knoten und Kanten des Graphen definieren, z. B. ermöglichen, dass die Knoten und Kanten aufgelistet werden, oder wenigstens, dass eine Mitgliedschaft der Mengen von Knoten und Kanten getestet wird. Die Graphen IDG, CDG sind allgemein über die gleiche Menge von Operationen definiert, z. B. weisen sie die gleiche Granularität auf. Das macht es einfacher und somit effizienter, Operationen der beiden Graphen aufeinander abzubilden.

[0080] In einer definierenden Operation Def, 335, können die Befehls- und Berechnungs-DFGs IDG, CDG verwendet werden, um ein Bedingungserfüllungsproblem CSP, 360, zu definieren. Eine Lösung für dieses Problem CSP kann einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung repräsentieren.

[0081] Wie an sich bekannt ist, ist ein Bedingungserfüllungsproblem CSP typischerweise durch eine Menge von Variablen VAR, 361; eine Menge entsprechender Domänen DOMS, 362, für die jeweiligen Variablen; und eine Menge von Bedingungen CONS, 363, definiert. Mathematisch kann das Bedingungserfüllungsproblem CSP durch ein Tripel (X, D, C) repräsentiert sein, wobei:

- $X = \{x_j | 0 \leq j \leq n\}$ eine Menge von Variablen ist, für das Problem einen Wert finden soll.
- $D = \{d_j | 0 \leq j \leq n\}$ die Menge von Wertdomänen ist, aus denen Werte den jeweiligen Variablen zugewiesen werden können. Eine Zuweisung $Asn(d_j, x_j)$: $x_j = v$ kann einen Wert $v \in d_j$ auswählen, der für ein x_j anzunehmen ist. Die Variable x_j kann somit darauf eingeschränkt sein, Werte nur aus der Domäne d_j zu anzunehmen.
- $C = \{c_i | 0 \leq i \leq m\}$ die Menge von Bedingungen ist. Eine Bedingung c_i kann über eine Teilmenge von Variablen $g_x \subset x$ gebildet sein und bewertet, ob alle Zuweisungen $Asn(g_d, g_x)$ mit $g_d \subset D$ gültig sind.

[0082] Eine Lösung des Bedingungserfüllungsproblems CSP kann eine Zuweisung sein, die Werte allen Variablen zuweist und wo keine Zuweisung die Vereinigung der Bedingungen C verletzt.

[0083] Das Bedingungserfüllungsproblem CSP kann das Problem zum Finden möglicher Aufrufe des Hardware-beschleunigten Befehls in der Berechnung durch Abbilden von Knoten des Berechnungsdatenflussgraphen CDG auf Knoten des Befehlsdatenflussgraphen IDG repräsentieren. Die Menge von Variablen VAR kann eine Variable für jeden Knoten des Befehlsdatenflussgraphen enthalten. Jeder Knoten des Berechnungsdatenflussgraphen kann in einer Domäne DOM für wenigstens eine der Variablen enthalten sein. Die

Menge von Bedingungen CONS kann Datenfluss- und Eingabebedingungen enthalten. Die Anzahl von Bedingungen kann beispielsweise wenigstens 10, wenigstens 100 oder wenigstens einmal, zweimal oder fünfmal die Anzahl von Knoten des Befehlsdatenflussgraphen sein.

[0084] Somit kann das Einbetten des Befehls in die Berechnung effektiv auf einer Skalarebene beschrieben sein. Interessanterweise kann jede potentielle Zuweisung zwischen Knoten in dem Befehl und Knoten in der Berechnung als eine Abbildung zwischen Knoten des Graphen repräsentiert sein, und somit kann die Formulierung verwendet werden, um jede mögliche Lösung für das Einbettungsproblem zu erfassen. Das Abbildungsproblem selbst ist nicht auf implizite Implementierungsentscheidungen wie Schleifenreihenfolge oder Speicherlayout von Tensoren oder Zugriffsfunktionen begrenzt. Das entfernt die Notwendigkeit von Transformationen in der Suche, um diese Entscheidungen zu berücksichtigen. Stattdessen können solche Transformationen aus dem Ergebnis der Einbettung abgeleitet werden.

[0085] Insbesondere kann eine Variable in der Menge von Variablen VARS für jeden Knoten des Befehls-DFG IDG enthalten sein, z. B. $X = \{x|\forall_x \in N_i\}$. Dementsprechend können jede Skalaroperation und jedes Datenelement in dem Hardware-beschleunigten Befehl durch eine Variable repräsentiert sein. Die Menge von Domänen DOMS kann als $D = \{d|d \subseteq S_d\}$ definiert sein. Hier kann S_d alle Knoten des Berechnungs-DFG CDG enthalten oder kann eine Teilmenge der Menge aller Knoten enthalten, z. B. die Teilmenge aller Knoten, die die gleiche Operation wie Knoten d aufweisen, usw. Es ist jedoch auch möglich, diese letztere Einschränkung als eine Bedingung aufzuerlegen. Die Domänen können zusätzliche Werte, die z. B. Dummy-Operationen oder Eingaben entsprechen, enthalten.

[0086] Mit den gegebenen Variablen VARS und Domänen DOMS kann die Menge von Bedingungen CONS verwendet werden, um Voraussetzungen zu repräsentieren, die eine Zuweisung erfüllen muss, um einen möglichen Aufruf des Hardware-beschleunigten Befehls zu repräsentieren.

[0087] Ein wichtiger Aspekt, der durch die Bedingungen CONS erzwungen werden kann, ist, dass der Datenfluss der Knoten des Befehlsdatenflussgraphen IDG dem Datenfluss der Knoten des Berechnungsdatenflussgraphen CDG, denen sie zugewiesen sind, äquivalent ist. Beispielsweise kann das Nehmen des Teils des Berechnungsdatenflussgraphen, der durch die zugewiesenen Knoten gegeben ist, und sein Ersetzen durch die Knoten des Befehlsdatenflussgraphen zu einem Datenflussgraphen einer Berechnung mit denselben Ausgaben führen. Bedingungen, die diese Äquivalenz erzwingen, können als Datenflussbedingungen bezeichnet sein.

[0088] Insbesondere können die Datenflussbedingungen erzwingen, dass der Befehlsdatenflussgraph IDG zu dem Teilgraphen des Berechnungsdatenflussgraphen CDG, der durch die Lösung für das Bedingungserfüllungsproblem CSP induziert ist, isomorph ist. In diesem Fall kann das Ersetzen des abgebildeten Teilgraphen des Berechnungsdatenflussgraphen durch den Befehlsdatenflussgraphen zu einem Graphen führen, der hinsichtlich der Verbindung zwischen den und der Kennzeichnung der Knoten identisch ist. Strikter Isomorphismus kann jedoch auf verschiedene hier beschriebene Weisen gelockert sein.

[0089] Um die Äquivalenz zu erzwingen, können für eine Kante des Befehlsdatenflussgraphen eine oder mehrere Bedingungen, die ihren Datenfluss repräsentieren, in die Datenflussbedingungen aufgenommen werden. In mathematischer Notation können für einen gegebenen Befehls-DFG $G_i = (N_i, E_i, l_i)$ und Berechnungs-DFG $G_o = (N_o, E_o, l_o)$ die Datenflussbedingungen erfordern, dass die Lösung für das Bedingungserfüllungsproblem CSP eine injektive Funktion f: $G_i \rightarrow G_o$ repräsentiert, die eine unterscheidbare Teilmenge von Knoten und Kanten in G_o, die mit G_i übereinstimmen, beschreibt, z. B.: $V(s, t) \in E_i \Rightarrow (f(s), f(t)) \in E_o$, und die die Kennzeichnung beibehält, z. B. $\forall s \in N_i$: $l_i(s) \equiv l_o(f(s))$. Injektivität kann dadurch erzwungen werden, dass erforderlich ist, dass jeder Knoten des Berechnungsdatenflussgraphen nur einmal in der Lösung existiert. Eine globale „AllDiff“-Bedingung, die das erzwingt, kann in den Datenflussbedingungen enthalten sein.

[0090] Beispielsweise kann der folgende Pseudocode verwendet werden, um den Datenfluss zu modellieren:

```
for (s,t) in E:
        if label(s,t) == sequential:
                edge(s,t,type=sequential)
        else:
                edge(s,t,type=spatial)
AllDiff(N)
```

[0091] Die Datenflussbedingungen können erzwingen, dass Paare aus wechselseitig parallelisierbaren Operationen des Hardware-beschleunigten Befehls auf Paare von wechselseitig parallelisierbaren Operationen der Berechnung abgebildet werden. In diesem Beispiel erfolgt das durch Aufnehmen von Kanten vom räumlichen Typ in die jeweiligen Datenflussgraphen IDG, CDG und dann Erzwingen, dass die räumlichen Kanten des Befehlsdatenflussgraphen auf räumliche Kanten des Berechnungsdatenflussgraphen abgebildet werden. Wie diskutiert kann das Aufnehmen einer Bedingung für jedes Paar in einer Menge wechselseitig parallelisierbarer Operationen zu einer großen Anzahl von Bedingungen führen. Die transitive Eigenschaft der paarweisen Bedingungen kann eingesetzt werden, um das zu vermeiden. Anstatt wechselseitige Parallelisierbarkeit für jedes Paar in der Menge zu erzwingen, kann sie nur für eine Teilmenge von Paaren erzwungen werden. Insbesondere kann, falls die Teilmenge die vollständige Menge als transitive Hülle aufweist, immer noch Parallelisierbarkeit für die vollständige Menge impliziert sein. Beispielsweise kann ein beliebiger erster Knoten in dem Befehls-DFG ausgewählt sein, und eine Bedingung kann zu jedem parallelen Knoten, den er aufweist, hinzugefügt werden. Falls irgendeinem Knoten, der parallel zu dem ersten Knoten ist, ein Wert zugewiesen wird, kann die Domäne des ersten Knotens ausgedünnt werden, so dass sie nur Knoten parallel zu dem Zugewiesenen enthält. Das kann sich zu allen anderen Knoten parallel zu dem ersten Knoten ausbreiten. Somit kann die gleiche Anzahl von Werten effizient ausgedünnt werden.

[0092] Beim Lösen des Bedingungserfüllungsproblems CSP können Variablen VARS Werte aus ihren jeweiligen Domänen DOMS zugewiesen werden. Während des Lösens kann ein Knoten des Berechnungsdatenflussgraphen CDG als ein möglicher Wert für einen Knoten des Befehlsdatenflussgraphen IDG ausgewählt werden. Der Bedingungsauflöser kann einen sogenannten Propagierungs-Algorithmus anwenden, der versucht, einen Wert einer Variablen zuzuweisen, und überprüft, ob diese Zuweisung durch die Bedingungen CONS erlaubt ist und/oder ob diese Zuweisung Zuweisungen zu anderen Variablen VARS gemäß den Bedingungen erzwingt. Das Folgende ist ein beispielhafter Propagierungs-Algorithmus, der Datenflussbedingungen erzwingt. Für einen gegebenen Knoten s, dem ein Wert zugewiesen ist und der eine Kante zu einem anderen Knoten t aufweist, überprüft dieser beispielhafte Algorithmus, ob die Zuweisung zu s korrekt ist und welche korrekten Werte dem anderen Knoten t zugewiesen werden können:

```
if label(s) != l(s.val):        # überprüfe die Variable label gegen das label des zugewiesenen
Knotens
        return Failed

# berechne die mögliche Lösung für die Relation in Bezug auf den Knotentypen und
zugewiesenen Wert
solution = evaluate_relation(T, l(s), l(t), s.val)
if not solution:
        return Failed

t.intersect(solution)

if t.size == 1:
        t.val = t[0]            # weise den einzigen verbleibenden Wert in der Domäne zu
        return Finished                                  # Bedingung ist beendet
return                  # Wert aus t.domain muss durch den Löser ausgewählt werden
```

[0093] In diesem Beispiel filtert der Propagator Werte direkt basierend auf den Relationen zwischen den Knoten in T. Er bewertet die Relation („evaluate_relation“) und entfernt Werte aus der Domäne des Partner-Knotens, wo keine Verbindung existiert („t.intersect“). Falls die Relation zwischen dem Paar funktional ist, kann er direkt eine Lösung zuweisen („ift.size==1“). Selbst wenn das nicht der Fall ist, ist die gezeigte Propagierung mächtig genug, die Domäne zusammenzufassen, was bedeutet, dass nur gültige Lösungen für diese Bedingung in den Domänen verbleiben und keine weitere Propagierung notwendig ist. Die verbleibenden Domänenwerte werden in Bezug auf die anderen Bedingungen über ihre Variable bewertet. Falls eine Propagierung zu einer leeren Domäne führt, ist die Zuweisung fehlgeschlagen.

[0094] Wenn Werte beiden Knoten s und t des Befehlsdatenflussgraphen IDG zugewiesen sind, kann die Bedingung auf Richtigkeit überprüfen durch Verifizieren, dass eine Kante in dem Berechnungsdatenflussgraphen CDG vorhanden ist, die das Paar verbindet, z. B. $Asn((s,t), (d_s, d_t)) \in E_o$. Insbesondere kann das das Erzwingen einbeziehen, dass eine Eigenkante des Berechnungsdatenflussgraphen IDG, die eine kommutative Reduktionsoperation repräsentiert, auf eine Eigenkante des Berechnungsdatenflussgraphen CDG abgebildet ist.

[0095] Es ist nicht notwendig, dass die Datenflussbedingungen einen strengen Isomorphismus erzwingen. Verschiedene Lockerungen können eingesetzt werden. Beispielsweise kann es, anstatt zu fordern, dass eine Quelloperation des Befehlsdatenflussgraphen IDG auf eine identische Zieloperation des Berechnungsdatenflussgraphen CDG abgebildet wird, ausreichend sein zu fordern, dass die Quelloperation eine Verallgemeinerung der Zieloperation ist, z. B. eine Operation mit wenigstens einer Genauigkeit, die durch die Zieloperation gefordert ist, oder eine Operation, die durch die Quelloperation instanziiert werden kann. Wie auch an anderer Stelle diskutiert können Lockerungen eingesetzt werden, um das Abbilden einer kommutativen Reduktionsoperation eines Befehlsdatenflussgraphen auf einen Knoten des Berechnungsdatenflussgraphen, der eine einzelne Reduktion der kommutativen Reduktionsoperation repräsentiert, zu ermöglichen. Ein weiteres Beispiel für Nicht-Erzwingen eines strikten Isomorphismus ist die Verwendung von Dummy-Operationen, wie auch an anderer Stelle diskutiert.

[0096] Wie die Erfinder erkannt haben, ist es, um zu definieren, was einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung begründet, in vielen Fällen nicht ausreichend, die Äquivalenz des Datenflusses mit Hilfe von Datenflussbedingungen zu definieren. Und zwar können, wie in der Figur gezeigt ist, der Hardware-beschleunigte Befehl und seine Programmierschnittstelle eine Menge von Hardware-Einschränkungen HWR, 320, für mögliche Aufrufe auferlegen. Das Definieren Def des Bedingungserfüllungs-

problems CSP kann das Definieren einer oder mehrerer Bedingungen CONS, die diese Hardware-Einschränkungen repräsentieren, umfassen. Die Hardware-Einschränkungen HWR können auf verschiedene Arten repräsentiert sein, z. B. als Bedingungen, als Vorlagen, aus denen Bedingungen instanziiert werden, wenn das Bedingungserfüllungsproblem definiert wird, als Code, der implizit Bedingungen basierend auf dem Befehls- und/oder Berechnungsdatenflussgraphen definiert, usw.

[0097] Insbesondere können die Hardware-Einschränkungen eine oder mehrere Eingabebedingungen auferlegen, die einschränken, welche Knoten des Berechnungsdatenflussgraphen CDG den Eingaben der Hardware-beschleunigten Befehle zugewiesen werden können, z. B. den Eingabeknoten des Befehlsdatenflussgraphen IDG zugewiesen werden können. Solche Bedingungen können beispielsweise erzwingen, dass die Eingaben ein zulässiges Speicherlayout und/oder ein zulässiges Zugriffsmuster aufweisen. Ähnliche Bedingungen können auch auf den Ausgaben des Hardware-beschleunigten Befehls platziert werden, z. B. falls die Ausgaben in einer weiteren Berechnung oder einem anderen Teil der aktuellen Berechnung, die ein spezielles Speicherlayout erfordert, verwendet werden sollen. Neben Einschränkungen, die durch den Befehl auferlegt sind, ist es auch möglich, Einschränkungen aufzunehmen, um einen Aufruf mit anderen erwünschten Eigenschaften zu erhalten, z. B. einen Aufruf, der das erste Element des Eingabe-Array verwendet. usw. Das kann es einfacher machen, den gefundenen Aufruf zu verallgemeinern, um Code für die vollständige Berechnung zu erzeugen. Allgemein kann eine geeignete Menge von Bedingungen CONS wenigstens von dem Hardware-Ziel und den verfügbaren Codeerzeugungsverfahren abhängen.

[0098] Beispielsweise können die Bedingungen CONS eines oder mehrere aus dem Folgenden enthalten:

- Einschränken von Eingaben auf zulässige Speicherzugriffsmuster; beispielsweise kann das Eingabezugriffsmuster auf eine Schablonenberechnung oder ein Zugriffsmuster mit regelmäßiger Schrittweite und/oder Versatz eingeschränkt sein; Propagierung kann Werte in solchen Dimensionen auf null begrenzen, was einen großen Teil des Suchraums in Berechnungen, in denen diese Zugriffsmuster existieren, entfernen kann;

- Einschränken des Abbildens von Befehlseingaben auf spezifische Achsen in der Berechnung;

- Einschränken, dass der Befehl auf spezielle Werte in der Berechnung angewandt wird; beispielsweise kann durch Einschränken der ersten Ausgabe in dem Befehl das erste Ausgabeelement in der Berechnung die Suche auf die Suche von Werten um diese Punkte festgelegt sein.

[0099] Interessanterweise kann das Aufnehmen zusätzlicher weiterer Bedingungen den Lösungsraum spezifischer machen, während das Lockern von Bedingungen als ein Werkzeug zum Implementierung von Strategie-Erkundung dienen kann. Bedingungsauflösung ist vorteilhaft, weil es die Aussagekraft der Programmformulierung bereitstellt und anpassbare Propagierungs- und Such-Algorithmen aufweist.

[0100] Das Bedingungserfüllungsproblem CSP kann auf eine solche Weise definiert sein, dass Operationen des Hardware-beschleunigten Befehls auf Dummy-Operationen, die die Berechnung nicht beeinflussen, abgebildet werden können. Effektiv kann eine „nicht perfekte“ (z. B. nicht isomorphe) Einbettung des Befehls in die Berechnung erhalten werden. Das kann ermöglichen, Aufrufe zu finden, wo das andernfalls nicht möglich ist. Beispielsweise kann es nicht möglich sein, eine Berechnung gleichmäßig in kleinere Teilprobleme, die durch den Befehl berechnet werden, zu unterteilen. Das Erlauben von Dummy-Operationen ermöglicht es, den Hardware-beschleunigten Befehl auch in solchen Situationen zu verwenden. Beispielsweise kann eine 8x8-Matrixmultiplikation mit einem Hardware-beschleunigten Befehl für 16×16-Matrixmultiplikation ausgeführt werden. Dieser Befehl kann durch Auffüllen des Eingabe- und Ausgabe-Tensors verwendet werden: die Form kann dann mit dem Befehl übereinstimmen, und der Befehl kann verwendet werden. Dummy-Operationen können in dem Bedingungserfüllungsproblem durch irgendeine Kombination aus Erlauben oder Beschränken von überlappenden/duplizierten Operationen; Erlauben oder Beschränken von überlappenden/duplizierten Speicherzugriffen; und Erlauben oder Beschränken der Einfügung von Dummy-Werten in die Berechnung erlaubt sein. Typischerweise wird als Dummy-Wert ein neutrales Element einer Operation aufgenommen, so dass das Ergebnis der Berechnung nicht beeinflusst wird, z. B. null (z. B. für Addition) oder eins (z. B. für Multiplikation). Diese Werte können zu den Domänen DOM der jeweiligen Variablen hinzugefügt werden.

[0101] Es ist auch möglich, Bedingungen für die Menge von Bedingungen CONS aufzunehmen, um das Bedingungserfüllungsproblem CSP zu vereinfachen und dadurch effizienter lösbar zu machen. Beispielsweise kann die Domänengröße einer Gruppe von Variablen reduziert werden durch Anwenden einer Ausdünnungsbedingung auf eine Gruppe von Variable des Befehlsdatenflussgraphen IDG, die dieselbe Domäne aufweisen, beispielsweise alle Variablen, die die Ausgabeoperation des Befehls beschreiben. Beispielsweise kann die Größe einer Domäne d_m durch einen Vektor (e_o, ..., e_m) beschrieben werden, z. B. kann d_m =

(4,4,4) einen Würfel der Kantenlänge 4 bezeichnen. Eine Koordinate in dem Würfel kann somit als (i,j.k) beschrieben werden, mit $0 \leq i < 4{,}0 \leq j < 4{,}0 \leq k < 4$. Somit kann durch Einstellen der oberen Begrenzungen e., z. B. Einstellen von $e_i = 2$, die Größe einer Domäne gesteuert werden. Beispielsweise kann die folgende unäre Ausdünnungsbedingung verwendet werden für das Festlegen eines Schwellenwerts für die Größe aller Dimensionen in einer m-dimensionalen Domäne $d_m = (e_0, \ldots, e_m) \subset S$ für alle Variablen in g verwendet durch:

$$\forall e_i \in (e_0,\ldots,e_m) | 0 \leq i \leq m : e_i = \begin{cases} e_i & \text{falls } \#g * \text{Schrittweite} \leq e_i \\ \#g * \text{Schrittweite} & \text{sonst} \end{cases},$$

wobei #g die Anzahl von Variablen ist, die dieser Gruppen zugewiesen sind, und e_i die Größe einer Dimension ist. Es ist auch möglich, eine Koordinate einer Domäne für eine Gruppe von Variablen auf irgendeinen anderen Schwellenwert zu begrenzen, z. B. auf die Länge einer Dimension in dem Befehl, usw. Eine Ausdünnungsbedingung ist eine Bedingung mit geringem Overhead, die den Suchraum reduziert.

[0102] Dementsprechend kann das Bedingungserfüllungsproblem CSP Knoten des Berechnungsdatenflussgraphen Knoten des Befehlsdatenflussgraphen zuweisen. In einigen Ausführungsformen wird ein Knoten des Berechnungs-DFG jedem Knoten des Befehls-DFG zugewiesen. In einigen Ausführungsformen wird jedem Knoten des Befehlsdatenflussgraphen entweder ein Knoten des Berechnungsdatenflussgraphen oder ein Knoten, der eine Dummy-Eingabe oder -Operation repräsentiert, zugewiesen. In einigen Ausführungsformen wird jedem Knoten des Befehls-DFG ein unterschiedlicher Wert zugewiesen, z. B. kann jeder Knoten des Berechnungs-DFG höchstens einmal zugewiesen werden.

[0103] In der Figur ist auch eine Operation Solve, 365, gezeigt, die das Bedingungserfüllungsproblem CSP aufnimmt und es löst, um einen möglichen Aufruf INV, 370, des Hardware-beschleunigten Befehls in der Berechnung, falls er existiert, zu bestimmen. Der Aufruf INV kann Knoten des Befehlsdatenflussgraphen IDG auf Knoten des Berechnungsdatenflussgraphen CDG abbilden, wie in der Figur durch gestrichelte Pfeile, die von den Knoten b, c, d des Befehlsdatenflussgraphen IDG zu den Knoten B, C, D des Berechnungsdatenflussgraphen CDG verlaufen, dargestellt ist. Einige wünschenswerte Eigenschaften des Aufrufs INV sind in diesem Beispiel dargestellt. Wie dargestellt kann jeder Knoten des Befehlsdatenflussgraphen IDG auf einen Knoten des Berechnungsdatenflussgraphen CDG abgebildet werden; Der Berechnungsdatenflussgraph weist jedoch typischerweise zusätzliche Knoten auf: die Knoten A und E in diesem Beispiel. Das Vorhandensein einer Kante zwischen jeweiligen Knoten des Befehlsdatenflussgraphen IDG kann das Vorhandensein einer Kante zwischen den entsprechenden Knoten des Berechnungsdatenflussgraphen CDG, auf die sie abgebildet sind, implizieren; es können jedoch zusätzliche Kanten vorhanden sein, z. B. die Kante zwischen den Knoten A und C und die Kante zwischen dem Knoten D und dem Knoten E in diesem Beispiel. Ein Nicht-Eingangsknoten des Befehlsdatenflussgraphen, z. B. des Knoten d, wird jedoch typischerweise auf einen Knoten des Berechnungsdatenflussgraphen, z. B. den Knoten D, mit der/dem gleichen Anzahl und/oder Typ eingehender Kanten abgebildet.

[0104] Um die Lösungsoperation Solve auszuführen, sind verschiedene Bedingungsprogrammiertechniken in der Literatur an sich bekannt und können hier angewandt werden. Interessanterweise kann der Algorithmus, der durch den Bedingungsauflöser verwendet wird, um Lösungen für das Problem zu finden, in vielen Fällen angepasst werden. Das ermöglicht die Kontrolle über den Lösungsraum, z. B. Priorisierung spezieller Lösungen, ohne zu ändern, was es dafür bedeutet, dass eine Lösung gültig ist oder nicht. Insbesondere können viele existierende Bedingungsauflöser Solve die Anpassung eines Propagierungs-Algorithmus, einer Variablenauswahlstrategie und/oder einer Werteauswahlstrategie erlauben.

[0105] Der Propagator kann verwendet werden, um Werte aus den Domänen DOMS, die nicht Teil einer gültigen Lösung sein können, zu entfernen. Der Propagator kann ein monotoner Filteralgorithmus sein in dem Sinn, dass er Werte aus einer Domäne entfernt, jedoch keine hinzufügt. Der Propagator ist typischerweise für jede Bedingung CONS spezifisch. Der Propagator kann ableiten, welche Werte aus einer Domäne zu entfernen sind, basierend auf den Domänen DOMS und Zuweisungen anderer Variablen unter derselben Bedingung. Um eine Lösung zu finden, kann der Löser Solve einen Suchalgorithmus verwenden, um Zuweisungen systematisch auszuführen und die Zuweisungen durch die Domänen DOMS zu propagieren. Ein beispielhafter Propagierungs-Algorithmus zum Erzwingen von Datenflussbedingungen ist an anderer Stelle in dieser Spezifikation gegeben.

[0106] Der Löser Solve verwendet in vielen Fällen einen rückverfolgungsbasierten Suchalgorithmus, um alle möglichen Lösungen in einem gegebenen Problem zu finden. Das kann eine Variablenauswahlstrategie bein-

halten, die bestimmt, welche Variable $x \in X$ aus der Menge von Variablen VARS als Nächstes einem Wert zugewiesen wird. Eine Werteauswahlstrategie kann bestimmen, in welcher Reihenfolge die Domäne einer Variablen zu durchsuchen ist, z. B. kann sie als eine spezifische Implementierung von Asn(d, x) betrachtet werden. Variablen- und Werteauswahl wirken sich auf die Zeit bis zur Lösung aus, und beeinflussen auch die Reihenfolgen, in der Lösungen ausgegeben werden. Somit können durch Optimieren des Werte- und Variablenauswahlstrategien sowohl die Effizienz des Lösens als auch die Qualität der bestimmten Lösungen beeinflusst werden.

[0107] Für die Variablenauswahlstrategie ist es bevorzugt, eine Strategie zu verwenden, die rückwärts durch den Befehlsdatenflussgraphen propagiert. Die Ausgaben können zuerst ausgewählt werden, und dann kann das Lösen mit rückwärts durch den Graphen Gehen fortschreiten. Es wurde gefunden, dass diese Strategie das größte Propagierungspotential für relevante Berechnungen, z. B. für Bewertungen von Schichten neuronaler Netze, aufweist.

[0108] Ein Beispiel für die Wertestrategie ist, eine lexikografische Suche zu verwenden. In vielen Fällen kann die Werteauswahlstrategie basierend auf zusätzlichen Informationen über die Struktur der Eingabedaten des Hardware-beschleunigten Befehls optimiert werden. Beispielsweise kann in vielen Fällen der Hardware-beschleunigte Befehl eine mehrdimensionale Eingabe aufweisen, und die Werte, auf die die Befehle angewandt werden, können ebenfalls mehrere Dimensionen aufweisen. In solchen Fällen kann eine Portfoliosuchstrategie als eine Werteauswahlstrategie verwendet werden. In einer solchen Suche werden mehrere sogenannte Assets, die jeweils eine Reihenfolge bereitstellen, in der Werte einer Variablen zugewiesen werden, parallel durchsucht. Die unterschiedlichen Assets können unterschiedlichen Abbildungen zwischen Dimensionen des Hardware-beschleunigten Befehls und Dimensionen möglicher Eingabewerte entsprechen. Ein Asset kann Zuweisungen gemäß seiner jeweiligen Dimensionsabbildung priorisieren. Diese unterschiedlichen Assets können parallel durchsucht werden, was zu einer Stabilisierung der Suchdauer über einen weiten Bereich von Berechnungen führt.

[0109] Interessanterweise kann, wenn eine Portfoliosuchstrategie verwendet wird, die Anzahl möglicher Assets durch Begrenzen, welche Dimensionen des Befehls auf welche Dimensionen der Berechnung abgebildet werden können, begrenzt werden. Das hängt von dem Befehl und der Berechnung, die vorliegen, ab und kann beispielsweise durch manuelle Annotation ausgeführt werden. Beispielsweise können die Dimensionen des Befehls und die Operation in Reduktionsdimensionen nicht Nichtreduktionsdimensionen unterteilt werden, z. B. für Matrixmultiplikationen. In der Portfoliosuchstrategie können die Reduktionsdimensionen der Operation dann nur auf Reduktionsdimensionen der Berechnung abgebildet werden, und ähnlich für Nichtreduktionsdimensionen. Auf diese Weise kann die Anzahl von Assets erheblich begrenzt werden, was die Leistung verbessert. Beispielsweise weist eine Matrixmultiplikation über (I,J,K) zwei Raumdimensionen (I,J) und ein Reduktionsdimension (K) auf. Die Werteauswahlstrategie kann versuchen, zwischen Dimensionen in den Berechnungen und den Dimensionen des Matrixmultiplizierens direkt einzubetten. In einem Asset kann die Suche die drei ausgewählten Dimensionen priorisieren. Falls n_s die Anzahl von Raumdimensionen und n_k die Anzahl von Reduktionen in der Berechnung ist, kann die Anzahl von Assets $\left(\frac{n_s!}{(n_s-2)!} \cdot \frac{n_k!}{(n_k-1)!}\right)$ sein. Das ist eine erhebliche Reduktion im Vergleich zu $(n_s + n_k + 1)!$ möglichen Permutationen der Domäne, die andernfalls möglich sein können. Innerhalb eines Assets kann beispielsweise lexikografische Suche als eine Werteauswahlstrategie angewandt werden.

[0110] Falls der Löser Solve keine Lösung in diesen initialen Portfolios finden kann, können die Bedingungen für den Suchraum gelockert werden, und die Suche kann mit einem größeren Lösungsraum wiederholt werden.

[0111] Allgemein (und anstelle des oder zusätzlich zu dem Wiederholen der Suche, wenn Portfoliosuche wie vorstehend diskutiert verwendet wird) kann, falls der Löser Solve keine Lösung für das Bedingungserfüllungsproblem CSP finden kann, das Bedingungserfüllungsproblem CSP gelockert werden, und das Lösen Solve kann wiederholt werden, z. B. bis eine Lösung gefunden wurde oder keine geeigneten Lockerungen mehr verfügbar sind. Beispielsweise kann das Bedingungserfüllungsproblem gelockert werden, um Dummy-Operationen zu erlauben, Bedingungen, die erzwingen, dass spezielle Eingaben verwendet werden, können gelockert werden; usw. Solche Lockerungen können zu weniger optimalen Lösungen führen, z. B. werden weniger Operationen aus den Operationen optimiert, oder mehr Vorverarbeitung wird benötigt, um die Daten in die richtige Form zu bringen, damit sie die den Hardware-beschleunigten Befehl zu verarbeiten sind, jedoch kann es durch Ausführen dieser Lockerung ermöglicht werden, dass der Hardware-beschleunigte Befehl auch in solchen nicht ganz idealen Situationen angewandt werden kann.

[0112] In der Figur ist auch eine optionale Kompilierungsoperation Comp, 345, gezeigt, die den bestimmten Aufruf INV verwenden kann, um maschinenausführbare Befehle INSTR, 350, die die Berechnung implementieren, zu erzeugen. Die Befehle INSTR können den Hardware-beschleunigten Befehl gemäß dem bestimmten möglichen Aufruf INV aufrufen. Beispielsweise kann der Aufruf INV verwendet werden, um kompilierbaren Code zu erzeugen, der dann kompiliert wird, um die Befehle INSTR zu erhalten. Die Regeln, die verwendet werden, um den Code aus der Aufruf-Kompilierung INV für einen speziellen Hardware-beschleunigten Befehl zu erzeugen, hängen typischerweise von der Codeerzeugungsschnittstelle ab, die angeboten ist. Allgemein kann durch Aufnehmen einer ausreichenden Eingabebedingung für das Bedingungserfüllungsproblem CSP dementsprechend, wie die Programmierschnittstelle das Aufrufen des Hardware-beschleunigten Befehls erlaubt, das Übersetzen des Aufrufs INV in die Befehle INSTR relativ unkompliziert ausgeführt werden.

[0113] In einigen Ausführungsformen kann der Befehlsdatenflussgraph Knoten für nur eine Teilmenge von Ausgaben des Hardware-beschleunigten Befehls enthalten. Um maschinenausführbare Befehle INSTR zu erzeugen, kann der Aufruf INV verwendet werden, um eine Abbildung einiger oder aller verbleibenden Ausgaben des Hardware-beschleunigten Befehls auf Operationen der Berechnung abzuleiten. Dementsprechend kann der Hardware-beschleunigte Befehl auf einen größeren Teil der Berechnung angewandt werden, als durch das Abbilden des möglichen Aufrufs INV angegeben ist. Beispielsweise verwendet in einer Matrixmultiplikation eine Berechnung eines Ausgabeelements eine Menge von Eingabeknoten mit seinen Nachbarn gemeinsam, jedoch keine Zwischenergebnisse. Somit kann der Aufruf INV eine Abbildung für nur ein Ausgabeelement bereitstellen, und diese Abbildung kann dann dadurch verallgemeinert werden, dass sie verwendet wird, um eine vollständige Matrixmultiplikation der Berechnung zu implementieren.

[0114] Interessanterweise kann das Erzeugen von Befehlen INSTR auch das Verallgemeinern des bestimmten Aufrufs INV beinhalten, um weitere Aufrufe des Hardware-beschleunigten Befehls zu erhalten. Für viele praktische Berechnungen, die Arbeitslasten neuronaler Netze wie z. B. Faltungen enthalten, ist es möglich, eine Implementierung der Berechnung mit mehreren Aufrufen aus einem einzigen bestimmten Aufruf INV zu extrapolieren. Zu diesem Zweck können ein oder mehrere Merkmale aus dem bestimmten möglichen Aufruf INV extrahiert werden, auf denen basierend der bestimmte Aufruf INV verallgemeinert werden kann. Beispielsweise kann die Berechnung Comp vollständig durch Aufrufe des Hardware-beschleunigten Befehls auf diese Weise implementiert sein.

[0115] Beispielsweise können für eine Faltung die Variablen des Berechnungsdatenflussgraphen CDG, die den Eingabe- und Ausgabewerten des Befehlsdatenflussgraphen IDG zugeordnet sind, bewertet werden, um zu berechnen, welche Dimension des Befehls mit welcher Dimension der Arbeitslast übereinstimmt und was die Kachelungsfaktoren sind. Daraus können Befehle INSTR, die die vollständige Faltung implementieren, implementiert werden. Beispielsweise können die übereinstimmenden Dimensionen mit den gefundenen Faktoren gekachelt und in die innersten Dimensionen verlagert werden. Diese Kachelung und Umordnungen können für das Einbetten fest sein. Andere Schleifen- und Tensor-Dimensionen können immer noch für weitere Leistungsoptimierungen transformierbar sein, z. B. können Schleifen-Kachelung, Umordnen, Zusammenführen usw. angewandt werden. Solche weiteren Leistungsoptimierungen können unter Verwendung von AutoTVM oder ähnlichen Werkzeugen automatisch bestimmt werden. Die Befehle INSTR mit eingebetteten Aufrufen des Hardware-beschleunigten Befehls können beispielsweise durch den VTA-Programmierwerkzeugablauf von TVM erzeugt werden.

[0116] Es ist auch eine optionale Bewertungsoperation Eval, 355, gezeigt. In vielen Fällen ist der Bedingungsauflöser Solve fähig, mehrere Aufrufe INV, die das Bedingungserfüllungsproblem CSP lösen, zu bestimmen, z. B. alle möglichen Lösungen für das Bedingungserfüllungsproblem CSP aufzulisten. In der Bewertungsoperation Eval kann die Leistung mehrerer solcher Aufrufe bewertet werden, und basierend auf dieser Leistungsbewertung kann eine Lösung aus den mehreren Lösungen ausgewählt werden. Die Bewertung kann basierend auf den bestimmten maschinenausführbaren Befehlen INSTR ausgeführt werden, z. B. durch Ausführen oder Simulieren der Befehle und Messen der Leistung; oder durch Berechnen einer (typischerweise Hardware-spezifischen) Kostenfunktion direkt auf dem Aufruf INV, ohne zuerst die Befehle INSTR zu bestimmen.

[0117] Allgemein können die hier beschriebenen Techniken auf verschiedene Typen von Berechnungen und Hardware-beschleunigten Befehlen angewandt werden. Der Hardware-beschleunigte Befehl kann eines oder mehrere aus einer Matrixmultiplikation, einer Faltung, einem Skalarprodukt, einem Matrix-Vektor-Produkt, einer kumulativen Summe, einem Pooling und einem Hadamard-Produkt implementieren. Die Berechnung kann beispielsweise eine Bewertung eines Faltungsoperators, eine Multiplikation oder eine Pooling-Operation einer Schicht eines neuronalen Netzes umfassen.

[0118] Typischerweise hängen wenigstens die Hardware-Einschränkungen HWR von der Ziel-Hardware ab, während der Berechnungsdatenflussgraph CDG von der auszuführenden Berechnung abhängt. Somit ist es beispielsweise möglich, Implementierungen für mehrere Berechnungen durch Anwenden der präsentierten Herangehensweise auf mehrere unterschiedliche Berechnungsdatenflussgraphen CDG unter Verwendung desselben Befehlsdatenflussgraphen IDG und Hardware-Einschränkungen HWR zu bestimmen; und mehrere Implementierungen derselben Berechnung durch Anwenden des präsentierten Herangehensweise unter Verwendung desselben Berechnungsdatenflussgraphen CDG und mehrerer unterschiedlicher Hardware-Einschränkungen HWR und/oder Befehlsdatenflussgraphen IDG zu bestimmen.

[0119] Fig. 4b zeigt ein ausführliches und dennoch nicht einschränkendes Beispiel einer polyedrischen Repräsentation, die einen Datenflussgraphen definiert. In verschiedenen Ausführungsformen kann ein Berechnungsdatenflussgraph oder ein Teil davon durch eine polyedrische Repräsentation repräsentiert sein.

[0120] Wie mit Bezug auf **Fig. 3** diskutiert kann ein möglicher Aufruf eines Hardware-beschleunigten Befehls in einer Berechnung durch Lösen eines Bedingungserfüllungsproblems, in dem Knoten eines Berechnungsdatenflussgraphen Knoten eines Befehlsdatenflussgraphen zugewiesen werden, bestimmt werden. In vielen praktischen Fällen kann die Anzahl von Knoten des Berechnungsdatenflussgraphen ziemlich groß sein, z. B. wenigstens 1024, wenigstens 4096 oder wenigstens 104876 Knoten. Interessanterweise haben die Erfinder jedoch erkannt, dass durch Verwenden der polyedrischen Repräsentation zum Speichern des Berechnungsdatenflussgraphen oder wenigstens Teilen davon vermieden werden kann, eine Liste aller Knoten und/oder Kanten des Berechnungsdatenflussgraphen explizit zu bestimmen und zu speichern. Das verbessert die Leistung des Bedingungsauflösers.

[0121] Allgemein wird in einer polyedrischen Repräsentation eine Menge von Knoten des Datenflussgraphen symbolisch repräsentiert, z. B. wird eine Repräsentation verwendet, die beschreibt, welche Knoten in der Menge sind, ohne sie explizit aufzulisten. Auch Mengen von Kanten, z. B. eingehende und/oder ausgehende Kanten der Menge von Knoten, können symbolisch repräsentiert sein. Wenn das Bedingungserfüllungsproblem gelöst wird, kann ein spezieller Knoten aus der Menge von Knoten durch Instanziieren der polyedrischen Repräsentation erhalten werden, z. B. durch Auswählen des Knoten gemäß einer Werteauswahlstrategie wie hier beschrieben. Ein Knoten des Befehlsdatenflussgraphen kann dann auf diesen instanziierten Knoten abgebildet werden.

[0122] Als ein Beispiel repräsentiert die Figur grafisch einen Datenflussgraphen unter Verwendung polyedrischer Repräsentationen. Wie in **Fig. 4a** ist eine 4x4-Matrixmultiplikation repräsentiert, z. B. ein Tensorausdruck T: $A_{i,j} = \Sigma_k X_{i,k} \bullet Y_{k,j}$. Diese Berechnung kann als die Schleifenverschachtelung repräsentiert sein: for i in I: for j in J: for k in K: A[i][j] += X[i][k] * Y[k][j].

[0123] Eingegebene Tensoren können durch eine Menge ihrer Form beschrieben werden. Beispielsweise ist in der Figur eine polyedrische Repräsentation x, 410, einer Menge von Eingabeknoten des Datenflussgraphen gezeigt. Diese Menge entspricht beispielsweise einer Menge, die die Knoten 411-414 von **Fig. 4a** enthält und kann beschrieben werden durch die Menge

$$X = \{(i,k) | 0 \le i < 1, 0 \le k < K\}.$$

[0124] Es ist auch die polyedrische Repräsentation y, 420, einer weiteren Menge von Eingabeknoten des Datenflussgraphen gezeigt, die die Knoten 421-424 von **Fig. 4a** enthält. Diese Menge kann beschrieben werden als:

$$Y = \{(k,j) | 0 \le k < K, 0 \le j < J\}$$

[0125] Die Skalarmultiplikationsoperationen der Matrixmultiplikationsberechnung können durch eine polyedrische Repräsentation *, 430, repräsentiert sein, die z. B. die Multiplikationen 431-434 von **Fig. 4a** enthält. Die Skalaradditionsoperationen der Matrixmultiplikationsberechnung können durch eine polyedrische Repräsentation +, 440, repräsentiert sein, die z. B. die Addition 441 von **Fig. 4a** enthält. Die Skalarmultiplikationen und -additionen können gemeinsam repräsentiert werden durch eine Menge:

$$S = \{(i,j,k,n) | 0 \le \mathrm{i} < \mathrm{I}, 0 \le \mathrm{j} < \mathrm{J}, 0 \le \mathrm{k} < \mathrm{K}, 0 \le \mathrm{n} < \#\mathrm{T}\},$$

I,J,K sind die Domänengrenzen. S enthält außerdem eine zusätzliche Dimension, die beschreibt, welches Element des ursprünglichen Tensorausdrucks T ausgewählt ist, z. B. in diesem Fall, ob der Knoten eine Addition oder eine Multiplikation repräsentiert.

[0126] Die Menge von Knoten des Datenflussgraphen kann als eine Vereinigung von Mengen, die durch polyedrische Repräsentationen definiert sind gebildet werden, z. B. $N_o = S \cup X \cup Y$.

[0127] Außerdem können Mengen von Kanten zwischen den Knoten des Datenflussgraphen durch polyedrische Repräsentationen definiert sein. Eine polyedrische Repräsentation kann einer binären Relation zwischen zwei Domänen entsprechen. Zwischen zwei Instanzen $(s1, s2) \in S$ ist eine Kante, z. B. ein Datenfluss, vorhanden, falls eine Relation existiert, für die $s1 \rightarrow s2 \neq \varnothing$ gilt.

[0128] Beispielsweise können in **Fig. 4b** die Kanten zwischen jeweiligen Mengen von Knoten 410, 420, 430, 440 durch die folgenden Relationen definiert sein:

$$R1* : \rightarrow + = \left\{ i,j,k,n \rightarrow i',j',k',n' \middle| i' = \mathrm{i}, j' = j', k' = \mathrm{k}, \mathrm{n} = n_*, n' = n_+ \right\};$$

$$R2 : + \rightarrow +' = \left\{ i,j,k,n \rightarrow i',j',k',n' \middle| i' = \mathrm{i}', j' = \mathrm{j}, k' = \mathrm{k}+1, n' = n = n_+ \right\};$$

$$A_X : \mathrm{X} \rightarrow * = \left\{ i,k \rightarrow i',j',k',n' \middle| i' = \mathrm{i}, k' = \mathrm{k}, n' = n_* \right\};$$

$$A_Y : \mathrm{Y} \rightarrow * = \left\{ \mathrm{j},\mathrm{k} \rightarrow k',j',i'n' \middle| j' = \mathrm{j}, k' = \mathrm{k}, n' = n_* \right\}.$$

[0129] In dem vorstehenden Beispiel spezifiziert die Relation R1, dass Multiplikation und Addition in derselben Schleifeniteration stattfinden, jedoch durch ihre Position in dem Tensorausdruck T geordnet sind. R2 kann als zwei Additionsoperationen interpretiert werden, die sequenziell in der Iterationsdimension k stattfinden. A_X und A_Y codieren die Zugriffsfunktion selbst und durch welchen Knoten der Zugriff ausgeführt wird, in diesem Fall die Multiplikation in T. Die Vereinigung aller Relationen kann die Kanten des Datenflussgraphen G_o beschreiben, und spezifisch für das Beispiel $E_o = R1 \cup R2 \cup A_X \cup A_Y$.

[0130] In diesem Beispiel sind eine Menge von Knoten 440 und eine entsprechende Relation R2 verwendet, um eine kommutative Reduktionsoperation, in diesem Beispiel Addition, zu repräsentieren. Wie mit Bezug auf **Fig. 4a** diskutiert ist, kann in einem Befehlsdatenflussgraphen eine solche kommutative Reduktionsoperation durch einen einzigen Knoten repräsentiert sein, z. B. den Knoten 441 von **Fig. 4a**. In solchen Fällen kann der Knoten 441 des Befehlsdatenflussgraphen effektiv der Menge von Knoten 440 des Berechnungsdatenflussgraphen entsprechen. Effektiv kann die Relation R1 eingehenden Nicht-Eigenkanten des Knotens 441 entsprechen, während die Relation R2 der Eigenkante entsprechen kann. In diesem Fall kann das Bedingungserfüllungsproblem konfiguriert sein zu erlauben, dass der Knoten 441 auf irgendeinen der Knoten in der Menge von Knoten 440 abgebildet wird. Um das Abbilden von kommutativen Reduktionsoperationen auf einen Knoten aus der Menge von Knoten 440 aufzunehmen, kann der Term für Kontrollieren der Reduktionsreihenfolge gelockert werden, das heißt der Term für k' in R1 und R2 kann gelockert werden, so dass er keine Reihenfolge der Reduktion bestimmt. Durch diese Lockerung kann jeder Knoten der Menge 440 effektiv als ein kommutativer Reduktionsknoten betrachtet werden. Interessanterweise kann, obwohl diese Lockerung die Anzahl von Kanten in dem Graphen in hohem Maße steigert, er effizient unter Verwendung der polyedrischen Repräsentation implementiert werden, weil es nicht notwendig ist, den Berechnungsdatenflussgraphen vollständig zu instanziieren. Alternativ kann eine kommutative Reduktionsoperation auch bei Verwendung der polyedrischen Repräsentation als ein einziger Knoten repräsentiert sein.

[0131] Kennzeichen von Knoten und/oder Kanten des Datenflussgraphen können beispielsweise als gemeinsame Kennzeichen für die Menge oder als symbolisch definierte separate Kennzeichen bereitgestellt werden. Die Kennzeichen können Typen von Eingaben und Ausgaben; einen Typ der Operation usw. angeben. Beispielsweise können die Knoten 410 als „OutType: int8“; Knoten 420 als „OutType: int8“; Knoten 430 als „InType: int8; OutType: f32; Op: mul“; und Knoten 440 als „InType: f32; OutType: f32; Op: add“ gekennzeichnet sein.

[0132] Auch Paare von wechselseitig parallelisierbaren Operationen können in polyedrischer Notation angegeben sein. Beispielsweise können, wie auch mit Bezug auf **Fig. 4a** diskutiert ist, Paare von wechselseitig parallelisierbaren Operationen durch räumliche Kanten angegeben sein. Eine Menge solcher Kanten kann auch in polyedrischer Repräsentation repräsentiert sein.

[0133] Mit Bezug auf die Relationen, die die Menge von Kanten beschreiben, können mehrere Fälle auftreten. Einige der oder alle Relationen können nicht symmetrisch sein. Eine oder mehrere Relationen, wie z. B. die Relation *→ X in diesem Beispiel, können surjektiv und funktional sein, z. B. kann eine Multiplikation auf genau ein Tensorelement in X abbilden, das sie verbraucht. Eine oder inverse Relationen, wie z. B. die Relation X →*, können nicht-funktional sein, z. B. kann ein Eingabelelement in X durch mehrere, jedoch nicht alle, Multiplikationen verwendet werden. Das kann durch die Relation von A_X und A_Y beobachtet werden, da sie keinen Term mit Eigenschaften für i' bzw. $_j$' beinhalten. Für einen spezifischen Eingabewert in X kann die Relation die Teilmenge aller Multiplikationen, die diesen Wert verwenden, beschreiben.

[0134] Allgemein kann die Menge S von Operationen des Datenflussgraphen, die durch polyedrischen Repräsentationen definiert sind, als die Instanzmenge bezeichnet sein, wobei Elemente als dynamische Ausführungsinstanzen bezeichnet sind. Die Menge S kann durch eine Menge von Ganzzahltupeln beschrieben sein, wobei ein Ganzzahltupel eine dynamische Ausführungsinstanz, z. B. einen Knoten, beschreibt. Um die Menge von Ganzzahltupeln zu beschreiben, kann beispielsweise die folgende Notation verwendet werden:

$$\left\{\left(e_0,\ldots,e_n\right)\middle|\tau_0,\ldots,\tau_n\right\}$$

[0135] Beispielsweise kann ein jeweiliger Term τ_j feste untere und/oder obere Begrenzungen eines entsprechenden Tupelelements e_j bereitstellen. Die Vereinigung aller τ Terme kann somit eine vollständige Menge definieren. Eine Domäne des Bedingungserfüllungsproblems kann als eine Teilmenge einer Menge von Ganzzahltupeln, z. B. eine durch den Knotentyp definierte, definiert sein. Für Eingabeknoten kann die Domäne durch eine Menge von Tupeln in der Form der entsprechenden Tensoren beschrieben sein. Für Operationsknoten kann die Domäne die Menge von Tupeln, die Operationsknoten repräsentieren, oder eine Teilmenge davon sein.

[0136] Die Vereinigung binärer Relationen zwischen Paaren von Instanzen, z. B. sequenzielle Kanten des Graphen, ist manchmal als eine Datenabhängigkeitsrelation D bezeichnet. Eine Relation zwischen Elementen aus einer Quellmenge zu einer Zielmenge kann bezeichnet sein als:

$$s_{source} \rightarrow s_{target} = \left\{\left(e_0,\ldots,e_n\right) \rightarrow \left(e'_0,\ldots,e'_n\right)\middle|\Phi_0,\ldots\Phi_m\right\}$$

wobei ein Term Φ_k eine Voraussetzung in der Relation beschreibt. Ein Element in dem Zieltupel kann als e' bezeichnet sein. Die Relationsvoraussetzung Φ_k kann verwendet sein, um ein Quellelement e zu beschreiben, das auf das Element e' in der Zielmenge abbildet. Die Vereinigung aller Φ Terme kann die gesamte Relationsdomäne beschreiben.

[0137] Die Verwendung polyedrischer Repräsentationen ist besonders vorteilhaft für Berechnungen, die Teil einer Bewertung einer Schicht eines neuronalen Netzes ist, wie z. B. eine Faltung, eine Matrixmultiplikation oder eine Pooling-Operation. Solche Berechnungen arbeiten typischerweise über Tensoren, z. B. n-dimensionale Arrays, mit zur Zeit der Kompilierung bekannten Grenzen. Typischerweise werden unterschiedliche Ausgabeelemente auf die gleiche Weise berechnet, jedoch mit einer unterschiedlichen „Slice“ von Eingabewerten. Die Berechnungen beinhalten typischerweise tiefe Schleifenverschachtelungen ohne bedingte Anweisungen und sind in hohem Maße strukturiert. Das macht es besonders effizient, sie unter Verwendung polyedrischer Repräsentationen zu repräsentieren.

[0138] Interessanterweise bringt die Verwendung polyedrischer Repräsentationen in einem Bedingungserfüllungsproblem wie hier beschrieben keinen Verlust der Allgemeinheit mit sich. Das Abbilden des Befehls auf die Berechnung wird immer noch auf der Ebene individueller Knoten vorgenommen, z. B. auf der Skalarebene. Ein Bedingungsauflöser kann irgendeinen Berechnungsdatenflussgraphknoten aus einer Menge beschriebener Knoten einem Befehlsdatenflussgraphknoten zuweisen, unabhängig davon, wie die Menge beschrieben ist, z. B. unabhängig von der Ordnung der Dimensionen. Das ist anders als andere Herangehensweisen, in denen explizite Transformationen auf polyedrische Repräsentationen angewandt werden, wobei in diesem Fall die Art und Weise, in der die Menge beschrieben ist, das Modellieren von Annahmen, die schwer zu bewältigen sind, einschließt.

[0139] Die **Fig. 5a-5d** zeigen ein genaues und dennoch nicht einschränkendes Beispiel für das Anwenden von Hyper-Rechteck-Bedingungen. In verschiedenen Konfigurationen ist es vorteilhaft, die Eingaben des Hardware-beschleunigten Befehls einzuschränken, um eine regelmäßige Form zu bilden. Das kann zu Implementierungen mit effizienterem Speicherzugriff und/oder für die effiziente Speicherlayout-Transformationen

verfügbar sind, um zu erlauben, dass der Hardware-beschleunigte Befehl auf den Eingaben aufgerufen wird, führen. Insbesondere zeigt dieses Beispiel eine Menge von Hyper-Rechteck-Bedingungen, die verwendet werden können, um zu erzwingen, dass die Eingaben ein Hyper-Rechteck bilden.

[0140] Der Begriff Hyper-Rechteck ist hier verwendet, um allgemein ein Rechteck mit irgendeiner Anzahl von Dimensionen n > 0 zu bezeichnen, z. B. n = 1, n = 2, n = 3 oder n ≥ 4. Die Punkte können auf dem Hyper-Rechteck als ein quadratisches Gitter, das z. B. eine Menge von Punkten $[x_1 + ... + x_n) + \delta_1 \cdot (s_1, 0, ...,0) + \delta 2 \cdot (0, s_2, 0, ...,0) + ...$ für $0 \leq \delta_i \leq \Delta_i$ bildet, angeordnet sein, wobei die s_i Schrittweiten sind, die separat festgelegt werden können, z. B. auf eins eingestellt, oder die dynamisch bestimmt werden können.

[0141] Das Verwenden eines Hyper-Rechtecks ist beispielsweise für Berechnungen geeignet, die in Bewertungen neuronaler Netze auftreten, da solche Berechnungen typischerweise (häufig große) Tensoren, z. B. n-dimensionale Arrays, einbeziehen. Eine Hyper-Rechteckform ist besonders vorteilhaft für effizienten Speicherzugriff und um zu ermöglichen, dass der Hardware-beschleunigte Befehl effizient unter Verwendung einer Speicherlayout-Transformation, z. B. einer Transponierten oder einer Kachelung von Tensordimensionen, aufgerufen wird. Bedingungen können verwendet werden, um sicherzustellen, dass die Variablen, die den Eingabewerten der Lösung zugewiesen sind, ein Hyper-Rechteck bilden, z. B. ein Hyper-Rechteck, das an einer Vektorbasis des Tensorraums ausgerichtet ist. Ein weiterer Vorteil ist, dass das Einschränken von Eingaben auf Bilden von Hyper-Rechtecken dem Bedingungsauflöser erlauben kann, potentielle Zuweisungen zu Eingabeknoten des Befehlsdatenflussgraphen relativ einfach zu eliminieren, was die Effizienz des Bedingungsauflösens in hohem Maße verbessern kann.

[0142] Die Hyper-Rechteck-Bedingungen können konfiguriert sein, die Form eines Hyper-Rechtecks mit einer Anzahl von Dimensionen n > 0 aus einem geordneten Tupel von Punkten $V = (v_0,..., v_n)$, das z. B. den Eingaben des Hardware-beschleunigten Befehls entspricht, abzugleichen und zu propagieren. Die Bedingung kann konfiguriert sein, einen Begrenzungsrahmen aus der Menge von Eingabepunkten, denen bisher Werte der Berechnung zugewiesen sind, und aus der Gesamtzahl von Punkten in v abzuleiten. Werte, die außerhalb des Begrenzungsrahmens sind, können als mögliche Werte für die verbleibenden Eingabepunkt des Hardware-beschleunigten Befehls eliminiert werden. Dementsprechend ist es schon nach einigen wenigen Zuweisungen typischerweise möglich, einen großen Teil einer Domäne zu eliminieren. Beispielsweise kann nach dem Auswählen von nur zwei Punkten entlang einer Achse des Tensors diese Dimension auf eine Größe von $bound = \#v \cdot |v_0 - v_1|$ begrenzt werden.

[0143] Das Rechteck kann durch Ausführen einer linearen Iteration der Menge von Punkten v berechnet werden, was den iterativen Versuch abzuleiten, ob die Punkte ein Rechteck mit irgendeiner Anzahl beliebiger Dimensionen erzeugen, umfasst. Insbesondere kann, falls die Punkte ein Gitter in lexikografischer Reihenfolge beschreiben, kann eine Iteration das Ausführen eines der folgenden Schritte enthalten. Ein Basisschritt kann eine Bewegung von einem Element in der innersten Dimension zu dem nächsten Element repräsentieren. Ein Dimensionssprung kann das Bewegen zu einer nächsten Zeile der innersten Dimension, z. B. Bewegen diagonal durch den Raum, repräsentieren. Um eine Gitterstruktur zu erzwingen, kann es erforderlich sein, dass Basisschritte gleich sind und in einer festen Anzahl wiederholt werden, bevor ein Dimensionssprung stattfindet. Ein Dimensionssprung kann dem Rechteck neue Dimensionen hinzufügen. Für eine Dimension $[n_0,...,n_m] \in Rec$ kann es erforderlich sein, dass ein Dimensionssprung, n_k mit $0 \leq k \leq m$, $\prod_{i=0}^{k}$ d_i-mal stattfindet. Ein Dimensionssprung kann eine Form aufweisen, die durch einen Vektor $jmp = v_a - v_{a+1}$ definiert ist, wobei erforderlich sein kann, dass er gleich $v_o = (v_0 - v_a) + v_b$ ist, wobei v_0 der erste Punkt in V ist und v_b einer der Basisvektoren des Tensors skaliert mit einem Ganzzahlfaktor ist. Durch Einschränken von v_b auf ein Element der Vektorbasis kann sichergestellt sein, dass rechte Winkel an den Ecken sind und dass das Rechteck an einer Achse ausgerichtet ist.

[0144] **Fig. 5a-5d** zeigen ein Beispiel für das Erzwingen von Hyper-Rechteck-Bedingungen, die eine Gitterstruktur auf der Menge von Punkten erzwingen. Die Figur zeigt ein Beispiel dafür, wie die Bedingung erzwungen und propagiert werden kann. In den Figuren repräsentieren die gefüllten Punkte 512, 522, 532 Domänenwerte; Kreuze 511, 521, 531, 541 repräsentieren eine Auswahl für eine von 16 Variablen; nicht gefüllte Punkte 533, 543 repräsentieren Werte, die durch Propagierung entfernt werden.

[0145] In diesem Beispiel kann in den ersten 4 Schritten noch keine Propagierung möglich sein (**Fig. 5a-5b)**, da die Größe der Dimension entlang der x-Achse, 8, kleiner ist als die Gesamtzahl von Variablen, 16. Sobald jedoch der fünfte Wert ausgewählt ist, können Werte aus der Domäne entfernt werden. in diesem Beispiel können Werte (y, x) mit x > 3 entfernt werden, da das die Größe der innersten Dimension ist und diese für

jede äußere Dimension gleich sein muss. Aus dem Wert (1,0), der für die fünfte Variable ausgewählt ist, kann der Propagator ableiten, dass die Erweiterung in die y Dimension nicht größer sein kann als:

$$d_y = \frac{\#V}{\prod_{i=y+1}^{n} d_i \cdot stride_i}$$

[0146] In diesem Beispiel ist das $d_y = \frac{16}{4} \cdot 1 = 4$. Somit können die Punkte 533 eliminiert werden (**Fig. 5c**). Die verbleibenden Punkte werden wie in **Fig. 5d** gezeigt abgebildet.

[0147] Interessanterweise können durch Implementieren der Hyper-Rechteck-Bedingung durch Iterieren über die Menge von Punkten v die erhaltenen Bedingungen für die relative Dimensionsreihenfolge zwischen Befehl und Arbeitslast transparent sein. Als ein Ergebnis kann irgendeine Dimension in der Arbeitslast auf irgendeine Dimension in dem Befehl abgebildet werden, ohne dass z. B. eine Transponierungsoperation ausgeführt werden muss. Interessanterweise kann die Abbildung verwendet werden, um abzuleiten, welche Speichertransformationen, z. B. Transponierungen, Kacheln, Zusammenführen usw., zu verwenden sind, um sie zum Erzeugen von Code, der den Hardware-beschleunigten Befehl aufruft, zu verwenden.

[0148] Ein ausführliches Beispiel für eine Implementierung einer Hyper-Rechteck-Bedingung ist als Nächstes bereitgestellt. Der Algorithmus funktioniert auch für partiell zugewiesene Rechtecke. Die Informationen, die während der Inferenz gefunden wurden, können direkt verwendet werden, um die Domänen der verbleibenden Werte durch Berechnen eines Begrenzungsrahmens, der das größte mögliche Rechteck ist, das durch die anstehenden Zuweisungen gebildet werden kann, zu propagieren.

[0149] Der Algorithmus startet durch Betrachten an dem ersten Schritt, und falls es nicht an einer Achse ausgerichtet ist, gibt er einen Fehler zurück. Dann entfernt er den ausgeführten Schritt aus der Vektorbasis, so dass jede Tensordimension einmal verwendet wird. Dann wird der Rest der Punkte iteriert, und jeder Schritt wird tabelliert. Die live_tbl zeichnet die Schritte auf, die unternommen werden. Sobald ein Dimensionssprung zu d_k stattfindet, werden die Schritte, die für d_{k-1} ausgeführt sind, in der dim_table aufgezeichnet. Jeder Schritt, der stattfindet, wird geprüft, ob er bereits aufgezeichnet ist oder nicht. Falls er bereits stattgefunden hat, überprüft counters_valid, ob der Schritt zu der korrekten Zeit stattgefunden hat, und setzt alle Zähler der inneren Dimensionen zurück. In dem Beispiel von **Fig. 5** müssen vier Schritte in der x-Dimension für den Schritt in der y-Dimension sein. Sobald der y-Schritt stattgefunden hat, wird der Zähler für x zurückgesetzt.

Algorithmus. Hyper-Rechteck-Bedingung

```
V = <Tupel von Punkten>
VB = <Vektorbasis des Tensors>

u_hat, scale = unit_vector(V[1] – V[0])
if (u_hat) not in VB:        // falls der erste Schritt nicht an einer Achse ausgerichtet ist break
        return false
VB = VB - u_hat              // entferne u_hat aus der Basis, um diese dim niemals wieder zu
verwenden
V = V - (V[0], V[1])                                        // entferne diese Werte aus der Liste

dim_table = {}                                              // verfolge die beendeten Dimensionen
live_tbl[u_hat] = 1                                 // halte den Zählwert in der aktuellen Traversalen
inner_dims[u_hat] = {}                                      // verfolge die inneren Dims einer Dimension

for (v_n in V[1::]):                                                        // scanne jetzt den Rest



        s = v_n – v_n-1
        if step in inner_dims:                              // ein bereits vorher unternommener Schritt
                update_counters(live_tbl)
                // überprüfe, ob wir uns in diese Richtung zum richtigen Zeitpunkt bewegt
haben
                if not counters_valid(live_tbl, dim_table, inner_dims):
                        return false
        else:
                // überprüfe, ob wir einen "dimension_jump" machen können
                if not counters_valid(live_tbl, dim_table, inner_dims):
                        return false
                if is_dimension_jump(s):          // ist eine gültige Bewegung in dem Rechteck
                        update_tables(dim_table, live_table, inner_dims, s)
                else
                        return false
return dim_table                        // Informationen zum Berechnen des Begrenzungsrahmens
```

[0150] Es sind jetzt Einzelheiten von Experimenten bereitgestellt, in denen die diskutierten Techniken angewandt werden, um Code für den VTA-Hardware-Beschleuniger zu erzeugen, der einen Hardware-beschleunigten Matrixmultiplizier- (GEMM-) Befehl bereitstellt. Die Hardware ist auf einem ZynqUltrascale+ FPGA mit einem 256 kb Gewichtspuffer, einem 128 kb Datenpuffer und einem GEMM-Kern, der int8-Eingaben ver-

braucht und float32-Ergebnisse produziert, instanziiert. Die GEMM-Einheit berechnet $C_{xy} += A_{xx} \cdot B_{xy}^{T}$ mit (x, y, z) = (1,16,16). Ihre Ergebnisse können beispielsweise durch eine Vektor-Skalar-Einheit zur Aktivierung und Quantisierung von Operationen verarbeitet werden. Es wird darauf hingewiesen, dass der Matrixoperand B transponiert ist. Die Hardware weist eine Lade/Speicher-Einheit für unabhängige Speicherzugriffe von Matrixoperanden auf. Sie kann vollständige 2D-Operandenmatrizen, die fortlaufend in dem Speicher gespeichert sind, laden und speichern. Die Experimente verwenden Berechnungen aus der „Baidu DeepBench Inference Benchmark“.

[0151] Code, der gemäß der vorliegenden Technik erzeugt ist, wird mit der conv2d-Referenzimplementierung von RVM verglichen, die drei Achsen x, y, z der GEMM-Einheit statisch auf die Batch n = x-, Ausgabekanal oc = y- und Eingabekanal ic = z- Dimensionen in der Faltung abbildet. Die TVM-Faltungsimplementierung erwartet ein NCHW-Speicherlayout (N = Batch-Größe, C = Kanal, H = Höhe, W = Breite).

[0152] In den Experimenten wurden maschinenausführbare Generierungen unter Verwendung des TVM-Maschinenlern-Compiler-Framework erzeugt. Wenn TVM auf einen Operator trifft, der durch die Hardware beschleunigt werden kann, wird die Implementierung durch eine spezifizierte Einsatzstrategie behandelt. Eine Strategie implementiert den Operator im IR des TVM, optimiert für das Hardware-Ziel. Der Codeerzeugungs-Werkzeugablauf von TVM kann durch Integrieren unserer Herangehensweise in die Strategie von VTA verwendet werden. Der Befehls-DFG G_i wird basierend auf der Hardware-Konfiguration erzeugt. Basierend darauf wird ein Bedingungserfüllungsproblem wie beschrieben definiert. Die Knoten von G_i sind Variablen, die dynamische Instanzmenge des Operators bildet die Domäne.

[0153] In einem Experiment wurde der Bedingungsauflöser verwendet, um mögliche Aufrufe ähnlich der Referenzimplementierung zu bestimmen. Zu diesem Zweck wurden die folgenden Bedingungen verwendet:

- Datenflussbedingungen zum Abgleichen des Datenflusses der GEMM-Befehle;

- Hyper-Rechteck-Bedingungen, um sicherzustellen, dass Eingabe- und AusgabeElemente in eine an einer Achse ausgerichtete Form abgebildet werden, was einfachere Speichertransformationen basierend auf Transponier-und Umformungsoperationen erlaubt;

- eine allDiff-Bedingung, um zu verhindern, die die gleiche dynamische Ausführungsinstanz in demselben Befehlsaufruf mehrfach auftritt.

- eine Fester-Ursprung-Bedingung, die erzwingt, dass die erste Übereinstimmung aller Eingabe- und Ausgabe-Tensoren für den Ursprung der entsprechenden Domäne fest sind;

- eine Dichte-Bedingung, die Schrittweiten in irgendeiner Dimension für die Eingabe- und Ausgabe-Tensoren verbietet;

- eine Speicherzugriffs-Bedingung, die nur Übereinstimmungen in Arbeitslastdimensionen mit Zugriffsmustern, die mit dem Befehl übereinstimmen, erlaubt, z. B. keine Schrittweiten oder Schablonenmuster.

[0154] Für eine gegebene Berechnung kann der Bedingungsauflöser verwendet werden, um einen möglichen Aufruf zu erhalten, der beschreibt, wie jede/s Operation und Datenelement des Befehls G_i auf einen Knoten der Berechnung G_o abbildet. Aufgrund der Regelmäßigkeit von DNN-Arbeitslasten wie Faltungen ist es machbar, den gefundenen Aufruf auf eine Implementierung zu extrapolieren, wie den Hardware-beschleunigten Befehl wiederholt aufruft. In der Lösung werden die Variablen, die den Eingabe- und Ausgabewerten zugeordnet sind, bewertet, um zu berechnen, welche Dimension des Befehls mit welcher Dimension der Arbeitslast übereinstimmt und was die Kachelungsfaktoren sind. Diese Informationen ermöglichen, dass der Code erzeugt wird. Die übereinstimmenden Dimensionen werden mit den gefundenen Faktoren gekachelt und in die innersten Dimensionen verlagert. Diese Kachelung und Umordnungen sind für das Einbetten fest. Andere Schleifen- und Tensor-Dimensionen sind frei, um für weitere Leistungsoptimierungen transformiert zu werden. Diese Optimierungen enthalten Schleifenkacheln, Umordnen oder Zusammenführen. AutoTVM wurde verwendet, um automatisch die besten Optimierungsparameter für jede conv2d-Schicht zu bestimmen. Code mit eingebetteten Befehlen wurde durch den VTA-Programmierwerkzeugablauf von TVM erzeugt.

[0155] Die Experimente zeigten, dass es möglich war, Implementierungen mit ähnlicher Struktur und Leistung wie eine von einem Experten hergestellte Referenzimplementierung automatisch zu erzeugen. Interessanterweise erreichte für einige der Benchmark-Berechnungen Code, der basierend auf den bestimmten möglichen Aufrufen erzeugt wurde, eine erhebliche Beschleunigung im Vergleich zu der Referenzimplementierung. Und zwar erreichte die Referenzimplementierung für einige Benchmark-Berechnungen eine sehr geringe Hardware-Nutzung aufgrund ihrer statischen Einbettungsstrategie. Für diese Berechnungen wurde

ein Beschleunigungsfaktor von bis zu × 2.49 erreicht, und für individuelle Operatoren wurden Verbesserungen bis zu × 238 beobachtet. Somit stellte in diesen Szenarios Code, der gemäß den bestimmten Aufrufen automatisch erzeugt wurde, eine erhebliche Leistungsverbesserung bereit.

[0156] In einem weiteren Experiment wurden die bereitgestellten Techniken verwendet, um das Berechnungstensorlayout eines DNN dynamisch zu ändern. Die Lösung für das Bedingungserfüllungsproblem gibt an, welche Dimensionen notwendig sind, und somit, welche frei sind, was es erlaubt, dass das Speicherlayout der freien Dimensionen während der Codeerzeugung geändert wird. In diesem Experiment wurde der Bedingungsauflöser verwendet, um mögliche Aufrufe gemäß dem NHWC-Layout zu erzeugen (N = Batchgröße, H = Höhe, W = Breite, C = Kanal). In vielen Fällen wurde gefunden, dass das die Leistung im Vergleich zu dem NCHW-Layout, das durch die Referenzimplementierung verwendet wurde, verbessert.

[0157] In einem weiteren Experiment wurden Speicherzugriffsbedingungen im Vergleich zu den früheren Experimenten gelockert. Das erhöht die Anzahl von Lösungen für das Bedingungserfüllungsproblem. Wie diskutiert wurden in den vorstehenden Experimenten Bedingungen aufgenommen, um Lösungen ähnlich der Referenzimplementierung zu erhalten. Für einige der Benchmark-Berechnungen führten diese Bedingungen zu einer Notwendigkeit, Auffüllen mit Nullen in der ic-Dimension auszuführen, wegen ic/z < 1, was zu geringerer Nutzung und größeren Tensoren führte. Interessanterweise wurde gefunden, dass es durch Lockern dieser Bedingungen möglich ist, mögliche Aufrufe zu bestimmen, die weniger oder kein Auffüllen erforderten und somit eine verbesserte Leistung bereitzustellen. Insbesondere wurde eine Lockerung angewandt, die Abbildungen in mehreren Dimensionen der Faltung, die die Filterschablone enthält, ermöglicht. Das Lockern der Bedingungen bedeutet, dass mehr unterschiedliche Fälle behandelt werden müssen, wenn Code erzeugt wird, um die Berechnung zu implementieren, sie jedoch immer noch machbar ist. Insbesondere wurde die Codeerzeugung angepasst, um Zusammenführung mehrerer Tensordimensionen in eine und Linearisierung des Datenzugriffs von Schablonenberechnungen zu unterstützen. Letzteres beinhaltet die Replizierung des Zugriffsmusters, das durch eine Schablone produziert ist, wie z. B. W[h + kh], explizit im Speicher. Während der neue Fußabdruck der entrollten Dimension h' = hl/stride * kh ist, bleibt die Gesamtzahl von Operationen, um das Ergebnis zu berechnen, gleich. Um den erzeugten Overhead zu minimieren, wurden nur Schablonendimensionen, die für eine Einbettung notwendig sind, entrollt. Um Code für VTA zu erzeugen, wurden Tensoraufspaltungsfaktoren verwendet, die gleichmäßige Teiler der ursprünglichen Dimension sind. Wo eine Implementierung ohne irgendein Auffüllen nicht möglich ist, wurden die Reduktionsdimensionen automatisch dem nächsten gleichmäßigen Teiler der Befehlsgröße hinzugefügt, falls notwendig.

[0158] Insbesondere wurden Speicherlayout-Transformationen mit Relay-Funktionen implementiert. Der Operator, der die Schablone entrollt, verwendet ‚relay.take()‘, eine Sammelfunktion, die eine Indexliste verwendet, die bestimmt, welche Werte zu kopieren sind, da kein direkter im2col-Operator in Relay verfügbar ist.

[0159] Die Leistung der gemäß den Techniken erzeugten Implementierungen wurde hinsichtlich sowohl der Inferenz und Speichertransformationsleistung als auch des Datenfußabdrucks bewertet. Im Vergleich zu der TVM-Referenzimplementierung mit Auffüllen ist es in vielen Fällen möglich, den Speicherfußabdruck, die Operatorleistung und/oder die Gesamtleistung zu verbessern. In einigen Fällen beinhaltet das Kompromisse, und abhängig davon, für welche Leistungsparameter optimiert wird, können unterschiedliche optimale Implementierungen gefunden werden. Insbesondere war in vielen Fällen effektive Hardware-Nutzung, gesteuert durch Auffüllen, ein wichtiger Faktor, um die Leistung zu verbessern, insbesondere in Schichten mit ic = 1: falls ic < z, dann tragen nur ic/z · (h · w) Elemente in dem Eingabebild bedeutend zu dem Ergebnis bei, was die effektive Hardware-Nutzung herunter fährt und die präsentierte Technik besonders vorteilhaft macht. Andere Situationen, in denen erzeugte Aufrufe eine besonders gute Leistung zeigten, sind, wenn das Auffüllen Gewichtstensoren erzeugt, die die Kapazität des Gewichtspuffers des Beschleunigers übersteigen. Implementierungen, die mit den bereitgestellten Techniken erzeugt werden, können das vermeiden, was den Effekt besitzen kann, dass der Beschleuniger den vollständigen Gewichtstentor in dem On-Chip-Puffer halten kann. In solchen Fällen wurden ganz besonders große Leistungsbeschleunigungen gefunden.

[0160] **Fig. 6** zeigt ein Blockdiagramm eines computerimplementierten Verfahrens 600 zum Implementieren einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls eines Prozessorsystems. Das Verfahren 600 kann einer Operation des Systems 100 von **Fig. 1** entsprechen. Das ist jedoch keine Einschränkung, da das Verfahren 600 auch unter Verwendung eines anderen/einer Systems, Einrichtung oder Vorrichtung ausgeführt werden kann.

[0161] Das Verfahren 600 kann in einer Operation mit dem Titel „BERECHNUNGSDATEN ERHALTEN“ das Erhalten 610 von Berechnungsdaten umfassen. Die Berechnungsdaten können einen Berechnungsdatenf-

lussgraphen, der die Berechnung repräsentiert, definieren. Ein Knoten des Berechnungsdatenflussgraphen repräsentiert eine Eingabe oder eine Operation der Berechnung.

[0162] Das Verfahren 600 kann in einer Operation mit dem Titel „BEFEHLSDATEN ERHALTEN“ das Erhalten 620 von Befehlsdaten umfassen. Die Befehlsdaten können einen Befehlsdatenflussgraphen, der den Hardware-beschleunigten Befehl repräsentiert, definieren. Ein Knoten des Befehlsdatenflussgraphen kann eine Eingabe oder eine Operation des Hardware-beschleunigten Befehls repräsentieren.

[0163] Das Verfahren 600 kann in einer Operation mit dem Titel „BEFEHL -> BERECHNUNGS-CSP DEFINIEREN“ basierend auf den Berechnungsdaten und den Befehlsdaten das Definieren 630 eines Bedingungserfüllungsproblems umfassen. Eine Lösung für das Bedingungserfüllungsproblem kann einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung repräsentieren. Das Bedingungserfüllungsproblem kann Knoten des Berechnungsdatenflussgraphen Knoten des Befehlsdatenflussgraphen zuweisen. Das Bedingungserfüllungsproblem kann eine oder mehrere Datenflussbedingungen umfassen, die erzwingen, dass die zugewiesenen Knoten des Berechnungsdatenflussgraphen einen dem Befehlsdatenflussgraphen äquivalenten Datenfluss aufweisen. Das Bedingungserfüllungsproblem kann ferner eine oder mehrere Eingabebedingungen umfassen, die einschränken, welche Knoten des Berechnungsdatenflussgraphen den Eingaben des Hardware-beschleunigten Befehls zugewiesen werden können. Diese Einschränkungen können durch den Hardware-beschleunigten Befehl und/oder seine Programmierschnittstelle auferlegt werden. Das Bedingungserfüllungsproblem kann zusätzliche Bedingungen umfassen.

[0164] Das Verfahren 600 kann in einer Operation mit dem Titel „CSP LÖSEN, UM AUFRUF ZU BESTIMMEN“ das Lösen 640 des Bedingungserfüllungsproblems, um einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung zu bestimmen, umfassen. Das Verfahren 600 kann in einer Operation mit dem Titel „AUFRUF AUSGEBEN“ das Ausgeben 650 von Daten, die den möglichen Aufruf definieren, umfassen.

[0165] Es ist zu verstehen, dass im Allgemeinen die Operationen des Verfahrens 600 von **Fig. 6** in irgendeiner geeigneten Reihenfolge ausgeführt werden können, z. B. nacheinander, gleichzeitig oder einer Kombination daraus, als Gegenstand, wenn erforderlich, einer speziellen Reihenfolge, die notwendig gemacht wird, z. B. durch Eingabe/Ausgabe-Beziehungen.

[0166] Das/die Verfahren können auf einem Computer als ein computerimplementiertes Verfahren, als dedizierte Hardware oder als eine Kombination aus beiden implementiert sein. Wie auch in **Fig. 7** dargestellt ist, können Befehle für den Computer, z. B. ausführbarer Code, auf einem computerlesbaren Medium 700 gespeichert sein, z. B. in der Form einer Reihe 710 von maschinenlesbaren physikalischen Markierungen und/oder als eine Reihe von Elementen, die unterschiedliche elektrische, z. B. magnetische, oder optische Eigenschaften oder Werte aufweisen. Der ausführbare Code kann auf eine transitorische oder nicht-transitorische Weise gespeichert sein. Beispiele für computerlesbare Medien enthalten Speichervorrichtungen, optische Speichervorrichtungen, integrierte Schaltungen, Server, Online-Software usw. **Fig. 7** zeigt eine optische Platte 700.

[0167] Alternativ kann das computerlesbare Medium 700 transitorische oder nicht-transitorische Daten 710 umfassen, die Befehle repräsentieren, die dann, wenn sie durch ein Prozessorsystem, das Hardware-beschleunigten Befehl unterstützt, z. B. das System 200 von **Fig. 2**, ausgeführt werden, das Prozessorsystem veranlassen, eine Berechnung auszuführen, wobei die Befehle den Hardware-beschleunigten Befehl gemäß einem möglichen Aufruf, der wie hier beschrieben bestimmt ist, aufrufen.

[0168] Darüber hinaus ist ein computerimplementiertes Verfahren vorstellbar, das Ausführen von Befehlen, die den Hardware-beschleunigten Befehl gemäß einem bestimmten möglichen Aufruf aufrufen, durch ein Prozessorsystem, das einen Hardware-beschleunigten Befehl unterstützt, umfasst.

[0169] Beispiele, Ausführungsformen oder optional Merkmale, seien sie als nicht einschränkend angegeben oder nicht, sind nicht so zu verstehen, dass sie die beanspruchte Erfindung einschränken.

[0170] Es wird darauf hingewiesen, dass die vorstehend genannten Ausführungsformen die Erfindung vielmehr darstellen als sie einschränken, und dass Fachleute fähig sein werden, viele alternative Ausführungsformen zu konstruieren, ohne von dem Schutzbereich der beigefügten Ansprüche abzuweichen. In den Ansprüchen sollen irgendwelche Bezugszeichen, die zwischen Klammern platziert sind, nicht so gedeutet werden, dass sie den Anspruch einschränken. Die Verwendung des Verbs „umfassen“ und seiner Konjugatio-

nen schließt das Vorhandensein von Elementen oder Stufen, die nicht die in einem Anspruch genannt sind, nicht aus. Der Artikel „ein/e“, der einem Element vorausgeht, schließt das Vorhandensein mehrerer solcher Elemente nicht aus. Ausdrücke wie z. B. „wenigstens eines von“, wenn sie einer Liste oder Gruppe von Elementen vorausgehen, repräsentieren eine Auswahl aller oder irgendeiner Teilmenge von Elementen aus der Liste oder Gruppe. Beispielsweise sollte der Ausdruck „wenigstens eines aus A, B und C“ so verstanden werden, dass er nur A, nur B, nur C, sowohl A als auch B, sowohl A als auch C, sowohl B als auch C oder alle aus A, B und C enthält. Die Erfindung kann mit Hilfe von Hardware, die mehrere unterscheidbare Elemente umfasst, und mit Hilfe eines geeignet programmierten Computers implementiert sein. In dem Vorrichtungsanspruch bedeutet das Aufzählen von mehreren Mitteln, dass mehrerer dieser Mittel durch ein und dasselbe Element der Hardware verwirklicht sein können. Die reine Tatsache, dass spezielle Maßnahmen in wechselseitig abhängigen unterschiedlichen Ansprüchen vorgetragen sind, gibt nicht an, dass eine Kombination dieser Maßnahmen nicht vorteilhaft verwendet werden kann.

Patentansprüche

1. Computerimplementiertes Verfahren (600) zum Implementieren einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls eines Prozessorsystems, wobei das Verfahren Folgendes umfasst:
- Erhalten (610) von Berechnungsdaten, wobei die Berechnungsdaten einen Berechnungsdatenflussgraphen, der die Berechnung repräsentiert, definieren, wobei ein Knoten des Berechnungsdatenflussgraphen eine Eingabe oder eine Operation der Berechnung repräsentiert;
- Erhalten (620) von Befehlsdaten, wobei die Befehlsdaten einen Befehlsdatenflussgraphen, der den Hardware-beschleunigten Befehl repräsentiert, definieren, wobei ein Knoten des Befehlsdatenflussgraphen eine Eingabe oder eine Operation des Hardware-beschleunigten Befehls repräsentiert;
- basierend auf den Berechnungsdaten und den Befehlsdaten Definieren (630) eines Bedingungserfüllungsproblems, wobei eine Lösung für das Bedingungserfüllungsproblem einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung repräsentiert, wobei das Bedingungserfüllungsproblem Knoten des Berechnungsdatenflussgraphen Knoten des Befehlsdatenflussgraphen zuweist und wobei das Bedingungserfüllungsproblem wenigstens Folgendes umfasst:
- eine oder mehrere Bedingungen, die erzwingen, dass die zugewiesenen Knoten des Berechnungsdatenflussgraphen einen dem Befehlsdatenflussgraphen äquivalenten Datenfluss aufweisen;
- eine oder mehrere Bedingungen, die einschränken, welche Knoten des Berechnungsdatenflussgraphen den Eingaben des Hardware-beschleunigten Befehls zugewiesen werden können, wobei die Einschränkungen durch den Hardware-beschleunigten Befehl und/oder seine Programmierschnittstelle auferlegt werden;
- Lösen (640) des Bedingungserfüllungsproblems, um einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung zu bestimmen, und Ausgeben (650) von Daten, die den möglichen Aufruf definieren.

2. Verfahren (600) nach Anspruch 1, das ferner das Erzeugen von maschinenausführbaren Befehlen, die die Berechnung implementieren, umfasst, wobei die Befehle den Hardware-beschleunigten Befehl gemäß dem bestimmten möglichen Aufruf aufrufen.

3. Verfahren (600) nach Anspruch 1 oder 2, wobei die Operation der Berechnung und/oder die Operation des Hardware-beschleunigten Befehls eine Skalaroperation ist.

4. Verfahren (600) nach Anspruch 3, wobei der Hardware-beschleunigte Befehl eines oder mehrere aus einer Matrixmultiplikation, einer Faltung, einem Skalarprodukt, einem Matrix-Vektor-Produkt, einer kumulativen Summe, einem Pooling und einem Hadamard-Produkt implementiert.

5. Verfahren (600) nach Anspruch 4, wobei die Berechnung das Anwenden eines oder mehrerer aus einer Faltung, einer Matrixmultiplikation und einer Pooling-Operation einer Schicht eines neuronalen Netzes umfasst.

6. Verfahren (600) nach einem vorhergehenden Anspruch, wobei der Befehlsdatenflussgraph Knoten für nur eine Teilmenge von Ausgaben des Hardware-beschleunigten Befehls enthält, wobei das Verfahren ferner das Ableiten von dem möglichen Aufruf einer Abbildung weiterer Ausgaben des Hardware-beschleunigten Befehls auf Operationen der Berechnung umfasst.

7. Verfahren (600) nach einem vorhergehenden Anspruch, wobei die Berechnungsdaten eine polyedrische Repräsentation umfassen, die eine Menge von Knoten des Berechnungsdatenflussgraphen symbolisch definiert, und wobei das Lösen des Bedingungserfüllungsproblems das Instanziieren der polyedrischen

Repräsentation, um einen Knoten aus der Menge von Knoten zu erhalten, und Abbilden eines Knotens des Befehlsdatenflussgraphen auf den erhaltenen Knoten umfasst.

8. Verfahren (600) nach einem vorhergehenden Anspruch, wobei das Verfahren das Definieren einer oder mehrerer Bedingungen zum Erzwingen, dass Eingaben des Hardware-beschleunigten Befehls ein erlaubtes Speicherlayout und/oder Speicherzugriffsmuster aufweisen, umfasst.

9. Verfahren (600) nach einem vorhergehenden Anspruch, wobei das Verfahren das Definieren einer oder mehrerer Bedingungen zum Erzwingen, dass Paare von wechselseitig parallelisierbaren Operationen des Hardware-beschleunigten Befehls auf Paare von wechselseitig parallelisierbaren Operationen der Berechnung abgebildet werden, umfasst.

10. Verfahren (600) nach einem vorhergehenden Anspruch, wobei ein Knoten des Befehlsdatenflussgraphen eine kommutative Reduktionsoperation repräsentiert und wobei das Verfahren das Definieren einer oder mehrerer Bedingungen zum Erzwingen, dass der Knoten auf einen entsprechenden Knoten des Berechnungsdatenflussgraphen abgebildet wird, umfasst.

11. Verfahren (600) nach einem vorhergehenden Anspruch, wobei das Verfahren das Definieren des Bedingungserfüllungsproblems, um das Abbilden von Operationen des Hardware-beschleunigten Befehls auf Dummy-Operationen, die die Berechnung nicht beeinflussen, zu erlauben, umfasst.

12. Verfahren (600) nach einem vorhergehenden Anspruch, wobei das Verfahren umfasst, dass das Bedingungserfüllungsproblem eine oder mehrere Bedingungen enthält, die erzwingen, dass Eingaben des Hardware-beschleunigten Befehls ein Hyper-Rechteck bilden.

13. Verfahren (600) nach einem vorhergehenden Anspruch, das Bestimmen mehrerer Lösungen für das Bedingungserfüllungsproblem; Bewerten der Leistung der mehreren Lösungen; und Auswählen einer Lösung aus den mehreren Lösungen basierend auf der Leistungsbewertung umfasst.

14. System (100) zum Implementieren einer Berechnung unter Verwendung eines Hardware-beschleunigten Befehls eines Prozessorsystems, wobei das System Folgendes umfasst:
- eine Datenschnittstelle (120) zum Zugreifen auf Berechnungsdaten (030) und Befehlsdaten (040), wobei die Berechnungsdaten einen Berechnungsdatenflussgraphen, der die Berechnung repräsentiert, definieren, wobei ein Knoten des Berechnungsdatenflussgraphen eine Eingabe oder eine Operation der Berechnung repräsentiert; und wobei die Befehlsdaten einen Befehlsdatenflussgraphen, der den Hardware-beschleunigten Befehl repräsentiert, definieren, wobei ein Knoten des Befehlsdatenflussgraphen eine Eingabe oder eine Operation des Hardware-beschleunigten Befehls repräsentiert;
- ein Prozessorteilsystem (140), das konfiguriert ist zum:
- Erhalten der Berechnungsdaten und der Befehlsdaten;
- basierend auf den Berechnungsdaten und den Befehlsdaten, Definieren eines Bedingungserfüllungsproblems, wobei eine Lösung für das Bedingungserfüllungsproblem einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung repräsentiert, wobei das Bedingungserfüllungsproblem Knoten des Berechnungsdatenflussgraphen Knoten des Befehlsdatenflussgraphen zuweist und wobei das Bedingungserfüllungsproblem wenigstens Folgendes umfasst:
- eine oder mehrere Bedingungen, die erzwingen, dass die zugewiesenen Knoten des Berechnungsdatenflussgraphen einen dem Befehlsdatenflussgraphen äquivalenten Datenfluss aufweisen;
- eine oder mehrere Bedingungen, die einschränken, welche Knoten des Berechnungsdatenflussgraphen den Eingaben des Hardware-beschleunigten Befehls zugewiesen werden können, wobei die Einschränkungen durch den Hardware-beschleunigten Befehl und/oder seine Programmierschnittstelle auferlegt werden;
- Lösen des Bedingungserfüllungsproblems, um einen möglichen Aufruf des Hardware-beschleunigten Befehls in der Berechnung zu bestimmen, und Ausgeben von Daten, die den möglichen Aufruf definieren.

15. Computerlesbares Medium (700), das transitorische oder nicht-transitorische Daten (710) umfasst, die Folgendes repräsentieren:
- Befehle, die dann, wenn sie durch ein Prozessorsystem ausgeführt werden, das Prozessorsystem veranlassen, das computerimplementierte Verfahren nach einem der Ansprüche 1 bis 13 auszuführen; und/oder
- Befehle, die dann, wenn sie durch ein Prozessorsystem, das einen Hardware-beschleunigten Befehl unterstützt, ausgeführt werden, das Prozessorsystem veranlassen, eine Berechnung auszuführen, wobei die

Befehle den Hardware-beschleunigten Befehl gemäß einem möglichen Aufruf, der unter Verwendung des computerimplementierten Verfahrens nach einem der Ansprüche 1 bis 13 bestimmt ist, aufrufen.

Es folgen 5 Seiten Zeichnungen

Anhängende Zeichnungen

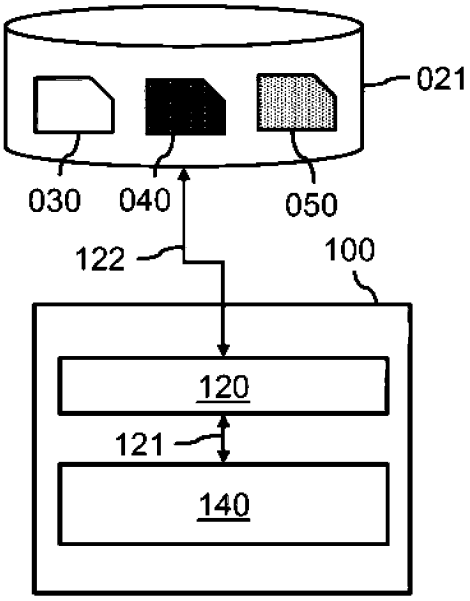


Fig. 1

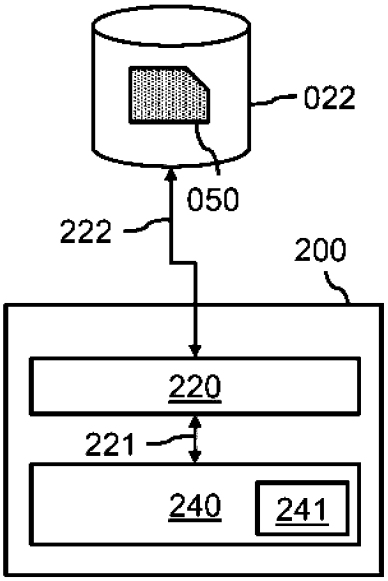


Fig. 2

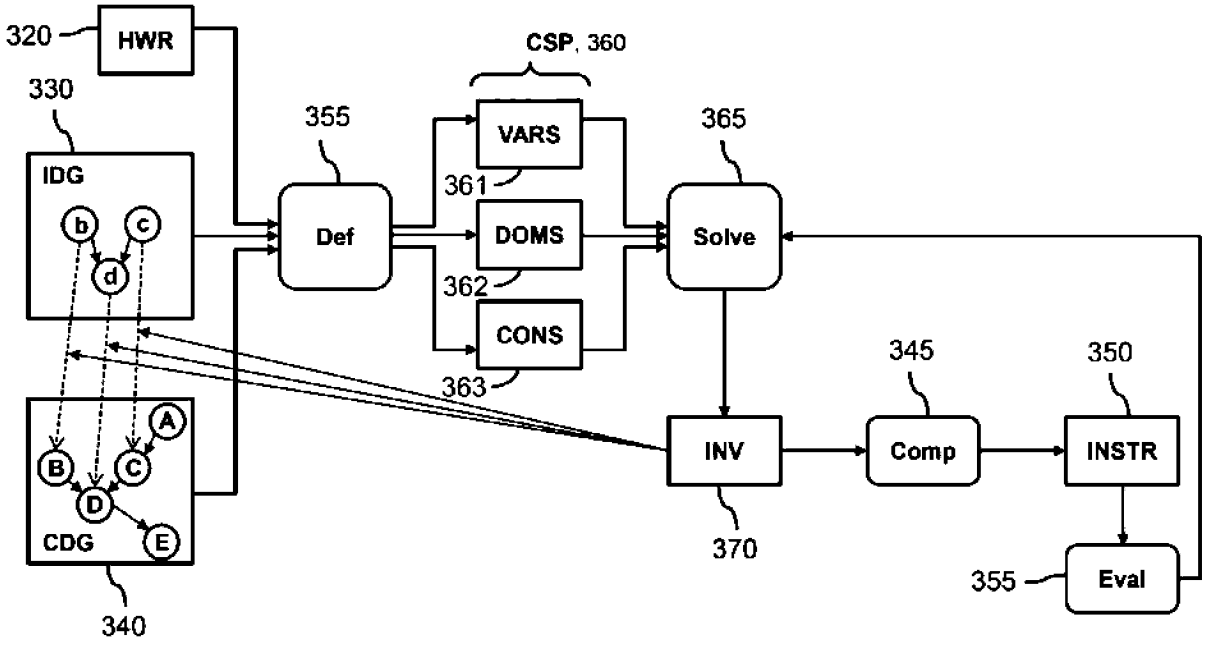

320
HWR
330
IDG
b
c
d
355
Def
CSP, 360
VARS
361
DOMS
362
CONS
363
365
Solve
345
Comp
350
INSTR
355
Eval
INV
370
A
B
C
D
E
CDG
340

Fig. 3

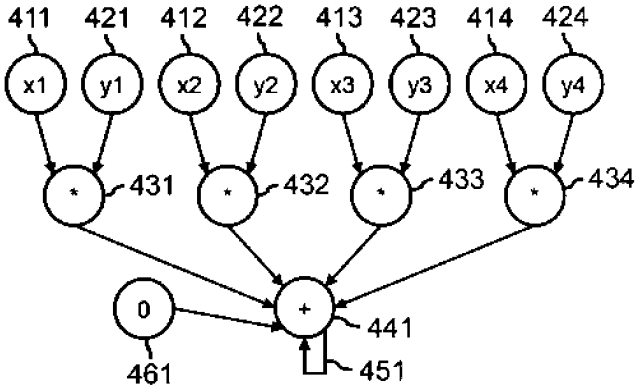

411
421
412
422
413
423
414
424
x1
y1
x2
y2
x3
y3
x4
y4
*
431
*
432
*
433
*
434
0
+
441
461
451

Fig. 4a

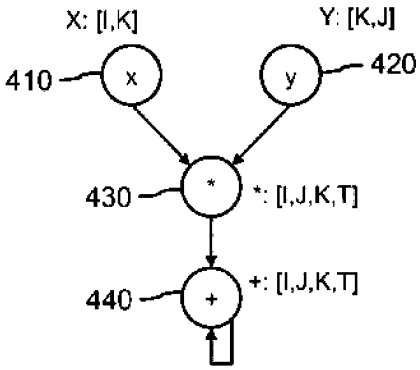

X: [I,K]
Y: [K,J]
410
x
y
420
430
*
*: [I,J,K,T]
440
+
+: [I,J,K,T]

Fig. 4b

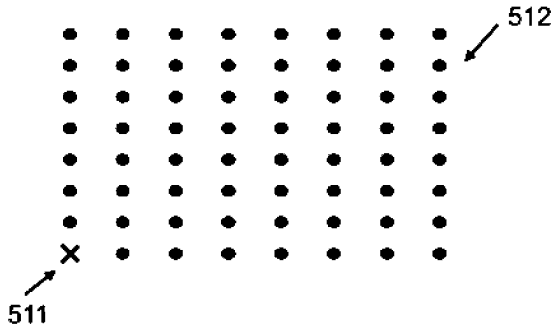


Fig. 5a

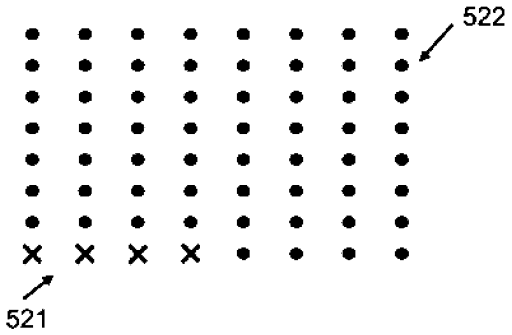


Fig. 5b

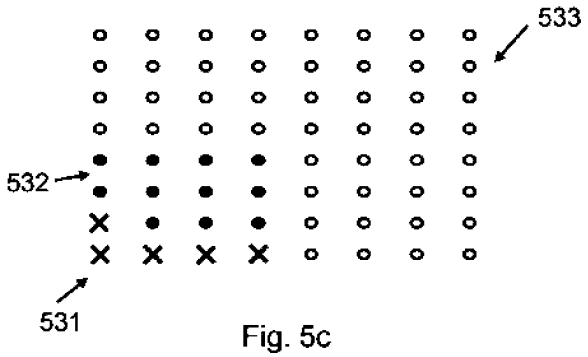


Fig. 5c

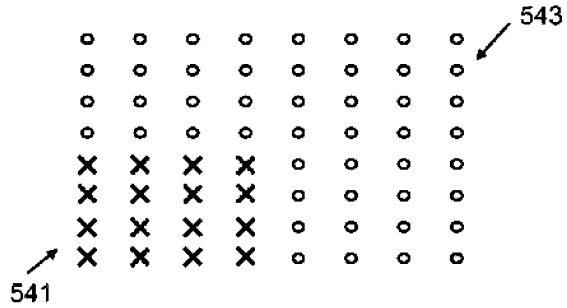


Fig. 5d

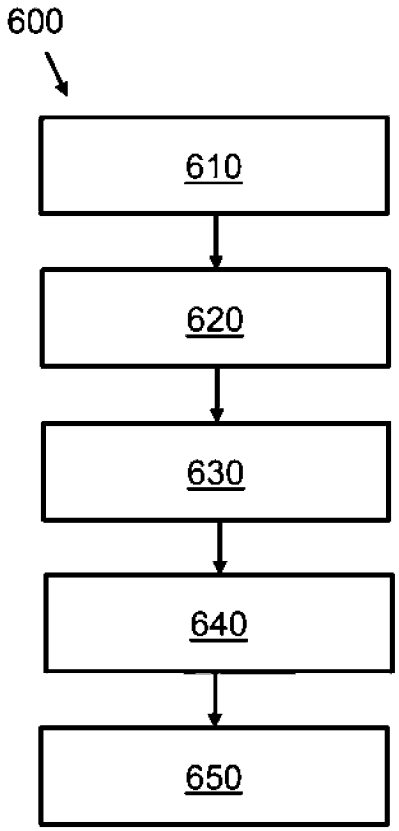


Fig. 6

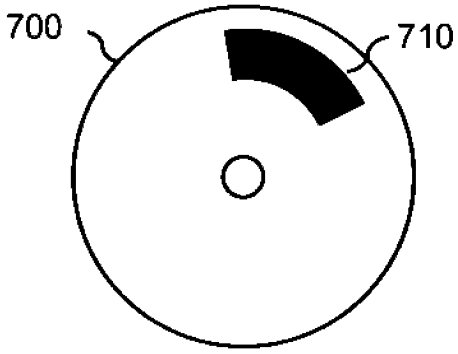


Fig. 7

 Das Dokument wurde durch die Firma Luminess hergestellt.