



(10) **DE 10 2022 210 228 A1** 2024.03.28

(12) **Offenlegungsschrift**

(21) Aktenzeichen: **10 2022 210 228.2**

(22) Anmeldetag: **27.09.2022**

(43) Offenlegungstag: **28.03.2024**

(51) Int Cl.: **G06N 3/10** (2006.01)

(71) Anmelder:

**Robert Bosch Gesellschaft mit beschränkter
Haftung, 70469 Stuttgart, DE**

(72) Erfinder:

**Klaiber, Michael, 70176 Stuttgart, DE; Braatz,
Yannick, 27337 Blender, DE**

(56) Ermittelter Stand der Technik:

**BRAATZ, Yannick; KLAIBER, Michael J.: End-
to-End Automation Frameworks for Mapping
Neural Networks onto Embedded Devices and
Early Performance Predictions: A Survey. In:
2021 Smart Systems Integration (SSI). IEEE, 2021.
S. 1-4**

**CHEN, Tianqi [et al.]: TVM: An automated end-
to-end optimizing compiler for deep learning.
arXiv preprint arXiv:1802.04799, 2018**

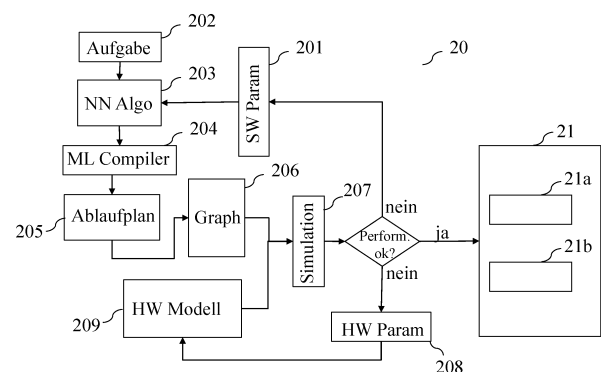
**KLAIBER, Michael J. [et al.]: An end-to-end
HW/SW co-design methodology to design
efficient deep neural network systems using
virtual models. In: INTESA2019 Proceed-ings of
the INTElligent Embedded Systems Architectures
and Applications Workshop 2019. 2019. S. 18-22**
**List of discrete event simulation software. In:
Wikipedia, Die freie Enzyklopädie.
Bearbeitungsstand: 26. September 2022. URL:
[https://en.wikipedia.org/w/index.php?
title=List_of_discrete_event_simulation_softwar-
e&oldid=1112487607](https://en.wikipedia.org/w/index.php?title=List_of_discrete_event_simulation_software&oldid=1112487607) [abgerufen am 16.01.2024]**
**SimPy. In: Wikipedia, Die freie Enzyklopädie.
Bearbeitungsstand: 24. Juni 2022. URL: [https://
en.wikipedia.org/w/index.php?
title=SimPy&oldid=1094692193](https://en.wikipedia.org/w/index.php?title=SimPy&oldid=1094692193) [abgerufen am
06.06.2023]**

Prüfungsantrag gemäß § 44 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen.

(54) Bezeichnung: **Verfahren und Vorrichtung zum gemeinsamen Optimieren eines neuronalen Netzes und einer für
das neuronale Netz dedizierten Hardware**

(57) Zusammenfassung: Verfahren zum Ermitteln einer Performance eines maschinellen Lernsystems auf einer Recheneinheit, umfassend folgende Schritte: Erstellen eines Hardwaremodells der Recheneinheit aus bereitgestellten technischen Spezifikation der Recheneinheit und Erstellen eines Simulationsgraphen basierend auf dem maschinellen Lernsystem. Simulieren eines Ausführens des maschinellen Lernsystemes auf der Recheneinheit unter Verwendung des Hardwaremodells und des Graphen, wobei die Simulation eine ereignisbasierte Simulation ist und Ermitteln der Performance abhängig von dem Ergebnis der Simulation.



Beschreibung

[0001] Die Erfindung betrifft ein Verfahren zum Ermitteln einer Performance eines neuronalen Netzes auf einer Recheneinheit und ein Verfahren zum gemeinsamen Optimieren einer neuronalen Netzwerkkonfiguration und einer Recheneinheit zum Ausführen des neuronalen Netzes hinsichtlich einer optimalen Hardwareperformance, sowie eine Vorrichtung, ein Computerprogramm und ein maschinenlesbares Speichermedium.

Stand der Technik

[0002] Der Einsatz von Deep Neural Networks (DNNs) auf Eingebettete Systeme (engl. Embedded Devices) wird aufgrund einer zunehmenden Komplexität der Netzwerkarchitekturen der DNN immer schwieriger. Insbesondere die Automatisierung der Optimierung neuronaler Netze (NN) im Rahmen der hardwareberücksichtigenden neuronalen Architektursuche (NAS) ermöglicht leistungsfähigere DNN auf Kosten großer Mengen von Netzwerkparametern. Das Ausbalancieren der begrenzten On-Chip-Ressourcen zur Ausführung von DNN auf Geräten unter Berücksichtigung von Millionen von Designentscheidungen macht es schwierig, die Entwicklung von Machine-Learning-Beschleunigerplattformen (ML) manuell zu optimieren. Design Space Exploration (DSE) Tools leiten den Suchprozess durch den großen Raum der Parameterauswahl und Architekturentscheidungen und ermöglichen HW/SW Co-Design des Algorithmus und des Beschleunigers. Um einzelne Designpunkte genau zu bewerten, nutzen diese Tools Vorhersagemethoden, um abzuschätzen wie sich bestimmte Parameterentscheidungen auf den gesamten Key Performance Indicator (KPI) auswirken.

[0003] Analytische Modelle liefern schnelle Vorhersagen der Leistung von DNN auf ML-Beschleunigern, wie bspw. Rechenlatenz, Ressourcennutzung oder Energieverbrauch durch Kombination von NN Operator Attributen, insbesondere Bedienerattributen, mit quantitativen Hardwareparametern wie bspw. Bandbreite oder verfügbar Chipressourcen. Solche Modelle sind jedoch auf einen engen Anwendungsbereich beschränkt, in welchem das Hardwareverhalten durch einfache Gleichungen beschrieben wird. Für komplexere Architekturen können MLbasierte Ansätze eingesetzt werden, die statistische Modelle basierend auf realen Hardwaremessungen trainieren. Durch das Erlernen dynamischer Hardwareeffekte sind diese Modelle in der Lage, die Beschleunigerleistung für eine Vielzahl von Workloads und Architekturen präzise vorherzusagen. Dieser Ansatz weist jedoch den Nachteil auf, dass ein zeitaufwendiges Training notwendig ist, dass reale Messdaten benötigt, die nicht immer zugänglich

sind, insbesondere in frühen Phasen des DNN-Systemdesigns.

[0004] Um eine optimale Hardware-performance/-leistung (z. B. Latenz, Energieverbrauch, Durchsatz) für die Ausführung neuronaler Netze auf Machine-Learning-Beschleunigern zu erreichen, müssen die begrenzten On-Chip-Ressourcen (Speicher, Verarbeitungselemente, Busbreite usw.) hinsichtlich der Netzanforderungen abgewogen werden. Da auf HW- und SW-Ebene in einem Suchraum bis zu Milliarden von Entscheidungen zu berücksichtigen sind, existieren Tools, die den Prozess automatisieren, indem bestimmte Parameterkombinationen von HW und SW bewertet werden. Dies geschieht üblicherweise durch die Formulierung von Gleichungen, die gelöst werden können, um Schätzungen zur Systemleistung in Bezug auf Latenz, Energieverbrauch oder Ressourcennutzung zu liefern. Solche analytischen Modelle berücksichtigen Hardware-Parameter wie Bandbreite, Taktfrequenz oder verfügbare Ressourcen und SW-Parameter der DNNs wie Schichten-/typen oder Operator Dimensionen oder Quantisierungen. Komplexeres Systemverhalten kann durch Simulationen vorhergesagt werden, aber der Aufbau für die Simulationsumgebungen kann erheblichen Aufwand bei der manuellen Modellierung der einzelnen Systemkomponenten verursachen.

[0005] Diese Erfindung zielt darauf ab, den Prozess der Erstellung von Simulationsumgebungen aus der neuronalen Netzwerkbeschreibung sowie das Ausführen eines HW/SW Co Design zu verbessern.

Vorteile der Erfindung

[0006] Die vorliegende Erfindung schlägt eine end-to-end Methodik für simulationsbasierte Leistungsvorhersagen von DNN(-Workloads) auf ML-Beschleunigern vor. Simulationsmodelle können genauere Vorhersagen liefern als mathematische Gleichungen. Die Erfindung ermöglicht eine Automatisierung des Erstellungsprozesses solcher Modelle zur Verwendung im Rahmen von HW/SW Co-Design. Insbesondere bei komplexeren HW/SW-Architekturen kann damit ein Zugewinn an Prognosegenauigkeit erzielt werden. Die erhaltenen Simulationsergebnisse des HW/SW Co Design können zur Optimierung der neuronalen Netzwerkparameter und/oder der Beschleunigerarchitektur verwendet werden.

[0007] Ein weiterer Vorteil ist, dass die Automatisierung des Prozesses der Erstellung von Simulationsumgebungen aus der Definition der neuronalen Netzwerk-Software es ermöglicht, den Algorithmus und das spezialisierte Embedded-Gerät, auf dem er eingesetzt wird, gemeinsam zu optimieren. Änderungen von Parametern auf HW- oder SW-Ebene können direkt in Simulationssetups übersetzt werden,

um die Systemleistung zu überprüfen. Dies führt zu maßgeschneiderten Hardware-Software-Systemen für spezifische Anwendungsfälle.

[0008] Ferner ermöglicht die Erfindung durch den geschlossenen Simulationskreis durch ein Anpassen von HW-Parameter des Beschleunigers und/oder SW-Parameter des DNN abhängig von den Simulationsergebnissen eine Validierung mehrerer neuronaler Netzwerkoperatoren -Ausführungen auf mehreren Designkandidaten von DNN.

[0009] Weitere Aspekte der Erfindung sind Gegenstand der nebengeordneten Ansprüche. Vorteilhafte Weiterbildungen sind Gegenstand der abhängigen Ansprüche.

Offenbarung der Erfindung

[0010] In einem ersten Aspekt betrifft die Erfindung ein computerimplementiertes Verfahren zum Ermitteln einer Performance eines maschinellen Lernsystems auf einer Recheneinheit. Die Recheneinheit kann ein Hardwarebeschleuniger für das maschinelle Lernsystem sein. Mit einem Hardwarebeschleuniger kann eine Entlastung eines Hauptprozessors der Recheneinheit durch Delegation spezieller rechenintensiver Aufgaben an auf diese Aufgaben spezialisierte Hardware erzielt werden. Das Verfahren kann mit einem optionalen Schritt umfassend ein Erhalten des maschinellen Lernsystems und einer technischen Spezifikation der Recheneinheit beginnen.

[0011] Daraufhin folgt ein Erstellen eines Hardwaremodells der Recheneinheit aus bereitgestellten technischen Spezifikation der Recheneinheit, sowie ein Erstellen eines Simulationsgraphen basierend auf dem maschinellen Lernsystem. Unter einem Simulationsgraphen kann ein Graphen verstanden werden, der die Ausführung der Simulationsabläufe charakterisiert bzw. kontrolliert. Das Hardwaremodell kann abhängig von Hardwarebausteinen, die in der technischen Spezifikation definiert sind, für die einzelnen Hardware-Komponenten wie DMAs, ALUs, Busse oder Speicher, aufgesetzt werden. D.h. es wird eine automatische Generierung von Simulationsgraphen für Optimierungen neuronaler Netze auf dedizierten Hardwarebeschleunigern durchgeführt.

[0012] Daraufhin folgt ein Simulieren eines Ausführungs des maschinellen Lernsystems auf der Recheneinheit unter Verwendung des Hardwaremodells und des Simulationsgraphen, wobei die Simulation eine ereignisbasierte Simulation ist. Die Modellierung durch ein ereignisbasiertes Simulationsframework hat den Vorteil, dass das Blockierverhalten einzelner Operationen auf der Hardware und deren Ausführung auf Hardware modelliert werden kann.

[0013] Daraufhin folgt ein Ermitteln der Performance abhängig von dem Ergebnis der Simulation. Die Performance kann eine Ausführungslatenz der Simulation sein. Operationen auf der Hardware werden Latenzen entsprechend ihrer Konfiguration zugeordnet, die beim Simulationsablauf aufsummiert werden. Am Ende der Simulation kann diese Summe als Performanz vorhersage ausgegeben werden.

[0014] Es wird vorgeschlagen, dass das maschinelle Lernsystem ein neuronales Netz ist, wobei beim Schritt des Erstellen des Simulationsgraphen ein neuronaler Netzwerkgraph in eine Baumdarstellung durch einen Machine Learning Compiler umgewandelt wird. Die Baumdarstellung charakterisiert eine Ausführungsreihenfolge aller zugewiesenen Netzwerk-Graph-Hardware-Operationen, während die Leafs Operationen Mikrodetails beschreiben, wie die Art der Hardware-Operation (z. B. laden, speichern, berechnen) zusammen mit relevanten Informationen (Eingänge, Ausgänge, Adressen, Datentypen). Die Baumdarstellung wird vorzugsweise in ein Graph umgewandelt, welcher ein Petri-Netz Graph ist.

[0015] Petri-Netz Graphen ermöglichen es, die Ausführungsreihenfolge von Prozessen zu modellieren, indem diese als Knoten in einer baumartigen Struktur ausdrücken, die über Kanten verbunden sind, die Token speichern können. Knoten müssen warten, bis eine feste Anzahl von Token an ihrer Eingangsverbindung verfügbar ist, bevor sie aktiviert werden, indem sie diese von ihr übernehmen. Nach Beendigung der zugewiesenen Aufgabe legen sie einige Token auf die Ausgangsverbindung, so dass nachfolgende Knoten ausgeführt werden können. Das Blockierverhalten von Hardware-Prozessen kann durch diese Design-Methodik zum Ausdruck gebracht werden.

[0016] Machine Learning Compiler kann wie TVM den ursprünglichen neuronalen Netzwerkgraphen in einer internen Baumdarstellung ausdrücken. Die Benutzung der Baumdarstellung zusammen mit allen zugehörigen Informationen ermöglicht eine Erstellung von Simulationsgraphen für eine geeignete Hardware-Ausführungsbeschreibung.

[0017] Weiterhin wird vorgeschlagen, dass interne Hardwareprozesse mit einer ereignisbasierten Simulation simuliert werden, wodurch Hardware-Blöcke zum Stillstand kommen, wenn Funktionen anderer Komponenten beendet werden müssen, bevor die eigentliche Ausführung gestartet werden kann. Unter Hardwareprozesse kann ein Laden/Speichern von Daten aus/in verschiedenen Bereichen der Speicherhierarchie, oder das Verarbeiten dieser Daten in Recheneinheiten verstanden werden.

[0018] In einem zweiten Aspekt der Erfindung wird ein computerimplementiertes Verfahren zum gemeinsamen Optimieren einer neuronalen Netzwerkkonfiguration und einer Recheneinheit zum Ausführen des neuronalen Netzes hinsichtlich einer optimalen Hardwareperformance und vorzugsweise einer optimalen Performance des neuronalen Netzes vorgeschlagen. Die Performance des neuronalen Netzes kann eine Zuverlässigkeit und/oder Genauigkeit von ermittelten Ergebnissen des neuronalen Netzes charakterisieren.

[0019] Zusätzlich zu den Verfahrensschritten des ersten Aspektes der Erfindung wird abhängig von der ermittelten Performance bestimmt, ob eine Anpassung von Parametern charakterisierend das neuronale Netz und/oder Parameter charakterisierend die Recheneinheit innerhalb eines vorgegebenen Parameterbereichs des jeweiligen Parameters durchgeführt wird. Wenn eine Änderung der Parameter vorgenommen wurde, dann werden Schritte des Erstellens und Ausführens der Simulation und Ermittlern der Performance erneut basierend auf den geänderten Parametern ausgeführt. Denkbar ist, dass das Verfahren mehrmals wiederholt wird, bis eine vorgegebene Soll-Performance der Recheneinheit und/oder des neuronalen Netzes erzielt wird.

[0020] Unter einer Hardwareperformance kann eine Latenzzeit, ein Energieverbrauch und/oder ein Durchsatz (engl. throughput) verstanden werden. Die Parameter, die die Recheneinheit charakterisieren, sind bspw. Speicherplatzgröße, Datenverarbeitungseinheiten, Busbreite, Bandbreite und/oder eine Taktfrequenz. Die Parameter, die das neuronale Netz charakterisieren, sind zum Beispiel Schichttypen, Schichtgrößen und/oder Quantisierungen. Parameter der Recheneinheit bzw. des neuronalen Netzes können innerhalb eines festen Bereichs geändert werden, was sich direkt auf die Fähigkeiten der Bausteine und das allgemeine Hardwaremodell auswirkt.

[0021] Weiterhin wird vorgeschlagen, dass dann wenn die Soll-Performance erzielt wird, eine Recheneinheit gemäß den Parametern charakterisierend das neuronale Netz und Parameter charakterisierend die Recheneinheit, mit welchen die Simulation die Soll-Performance erreicht hat, hergestellt und/oder konfiguriert wird.

[0022] In einem dritten Aspekt ist das maschinelle Lernsystem, insbesondere neuronale Netz, des ersten oder zweiten Aspekts der Erfindung ein (Bild-) Klassifikator oder Objektdetektor oder Semantische Segmentierung.

[0023] In einem weiteren Aspekt der Erfindung wird das maschinelle Lernsystem des dritten Aspektes der Erfindung Verwendung des Klassifikators zur Klassifizierung von Sensorsignalen vorgeschlagen.

Der Klassifikator wird mit dem Verfahren nach einem der vorangehenden Aspekte der Erfindung übernommen, mit den Schritten: Empfangen eines Sensorsignals, das Daten von dem Bildsensor umfasst, Bestimmen eines Eingangssignals, das von dem Sensorsignal abhängt, und Einspeisen des Eingangssignals in den Klassifikator, um ein Ausgangssignal zu erhalten, das eine Klassifizierung des Eingangssignals charakterisiert.

[0024] Der Bildklassifikator ordnet ein Eingabebild einer oder mehreren Klassen einer vorgegebenen Klassifikation zu. Als Eingabebilder können beispielsweise Bilder von in Serie gefertigten, nominell identischen Produkten verwendet werden. Der Bildklassifikator kann beispielsweise darauf trainiert sein, die Eingabebilder einer oder mehreren von mindestens zwei möglichen Klassen zuzuordnen, die eine Qualitätsbeurteilung des jeweiligen Produkts repräsentieren.

[0025] Der Bildklassifikator, z. B. ein neuronales Netz, kann mit einem solchen Struktur ausgestattet sein, dass er trainierbar ist, um z. B. Fußgänger und/oder Fahrzeuge und/oder Verkehrszeichen und/oder Ampeln und/oder Straßenoberflächen und/oder menschliche Gesichter und/oder medizinische Anomalien in bildgebenden Sensorbildern zu identifizieren und zu unterscheiden. Alternativ kann der Klassifikator, z. B. ein neuronales Netz, mit einem solchen Struktur ausgestattet sein, die trainierbar ist, um gesprochene Befehle in Audiosensorsignalen zu identifizieren.

[0026] Der Begriff des Bildes umfasst grundsätzlich jede in einem zwei- oder mehrdimensionalen Raster angeordnete Verteilung von Informationen. Diese Informationen können z. B. Intensitätswerte von Bildpixeln sein, die mit einer beliebigen Abbildungsmodalität, wie etwa mit einer optischen Kamera, mit einer Wärmebildkamera oder mit Ultraschall, aufgenommen wurden. Es können jedoch auch beliebige andere Daten, wie z. B. Audiodaten, Radardaten oder LIDAR-Daten, in Bilder übersetzt und dann gleichermaßen klassifiziert werden.

[0027] In weiteren Aspekten betrifft die Erfindung eine Vorrichtung sowie ein Computerprogramm, die jeweils eingerichtet sind, die obigen Verfahren auszuführen und ein maschinenlesbares Speichermedium, auf dem dieses Computerprogramm gespeichert ist.

[0028] Nachfolgend werden Ausführungsformen der Erfindung unter Bezugnahme auf die beiliegenden Zeichnungen näher erläutert. In den Zeichnungen zeigen:

Fig. 1 schematisch Schritte für eine Kompilierung von Operatoren neuronaler Netze auf Software-Ebene in Simulationsumgebungen;

Fig. 2 schematisch einen Ausführungsgraphen und Petri-Net-Graph einer Schicht eines neuronalen Netzes; und

Fig. 3 schematisch ein Ausführungsbeispiel zum gemeinsamen Optimieren eines neuronalen Netzes und eines Hardwarebeschleunigers des neuronalen Netzes.

Beschreibung der Ausführungsbeispiele

[0029] Die Schätzung einer Performance beim Ausführung von DNN-Workloads auf ML-Beschleunigern erfordert eine genaue Abwägung zwischen der Prognosezeit (Geschwindigkeit), der Bandbreite der Szenarien, die das Schätzmodell bewältigen kann (Komplexität) und der Overhead beim Aufsetzen der zugrundeliegenden Techniken (Flexibilität).

[0030] Die vorgeschlagene Methodik verwendet Simulationsmodelle, um das Verhalten der Hardware-Architektur für eine Vielzahl von Anwendungen unter Berücksichtigung einer Vielzahl von unterstützten Datenflüssen, Speicherhierarchien und Plattformtopologien verschiedener DNN-Hardware-/Software-Systeme abzudecken. Flexibilität bei der Anpassung an diese Merkmale kann durch den Einsatz von Hierarchie erreicht werden. Die Hierarchien können wie folgt aufgefasst werden: Details von Hardwareprozessen werden nicht auf der Ebene von einzelnen Instruktionen beschrieben, sondern als Funktionen ausgedrückt, die dieses Verhalten im Kern wiedergeben, aber deutlich schneller sind, da nicht jeder einzelne Prozessschritt ausgeführt wird, sondern die Gesamtheit der Prozessschritte durch diese Funktion beschrieben wird.

[0031] Auf Hardware-Ebene ermöglicht die Verwendung von Virtual Prototyps zur Beschreibung der Beschleunigerarchitektur eine schnelle Ausführung von Hardware-Operationen im Vergleich zur Geräteimplementierung oder RTL-Simulation, während die Änderung der Systemkonfiguration nur geringen Aufwand erfordert.

[0032] Die Einbindung von ML-Compilern in den Build-Prozess von Simulationsumgebungen ermöglicht die Erstellung von Ausführungsgraphen zur Ausführung der Simulation auf der virtuellen Hardware direkt aus der Beschreibung des DNN.

[0033] **Fig. 1** zeigt beispielhaft eine Methode für die Kompilierung von neuronalen Netzwerkarchitekturen/-beschleuniger in Software für eine Simulationsumgebungen, die ein Ausführen des neuronalen Netzes auf virtuellen Hardwaremodellen des Beschleunigers simuliert. Der Compiler ermöglicht hardwarebezogene Änderungen an den neuronalen Netzkernels, wie Loop-Tiling, Unrolling, etc. Diese Kernels in Hardware-fähige Ausführungsgraphen übersetzen und dabei automatisiert zwischen

Graph-Knoten und Komponenten virtueller Hardware-Plattformen abbilden, ermöglicht eine durchgängige Zusammenstellung von der Bedienerdefinition bis hin zu kompletten Simulationssetups.

[0034] Die Darstellung eines neuronalen Netzkernflusses durch Berechnungs- und Kommunikationseinheiten einer Beschleunigerarchitektur mit Simulationsgraphen liefert Ebenen von Prozessdetails wie Parallelisierung oder Pipelining, dass reine Software-Beschreibungen nicht ausdrücken können. Es wird vorgeschlagen, Petri-Netze als Basis für den Aufbau solcher Graphen zu nutzen, da sich herausgestellt hat, dass das Blockierverhalten mit der zugrunde liegenden Mechanik präzise ausdrücken lässt. Für weitere Details zu Petri-Netzen siehe zum Beispiel die Publikation von Alexandre V Yakovlev und Albert M Koelmans. 1996. Petrinetze und digitales Hardware-Design. In Advanced Course on Petri Nets. Springer, 154-236.

[0035] Diese Design-Methodik ermöglicht es, eine Vielzahl von Datenflüssen zu beschreiben, wobei hierfür vorzugsweise das Open-Source-Framework SimPy für die Implementierung verwendet wird.

[0036] SimPy ist ein ereignisbasiertes Simulationsframework in Python, das Ereignisse ähnlich wie Systeme nutzt, um Funktionsausführungen zu synchronisieren. Prozesse sind Funktionen, die an Ereignisse gekoppelt sind, die bei anderen Funktionen entstehen. Ressourcen sind Objekte, auf die Prozesse gegenseitig zugreifen können. Ereignisse und Prozesse, die gleichzeitig auf gemeinsame Ressourcen zugreifen können, werden in Simulationsumgebungen eingereiht, wobei die Simulation durchgeführt wird, indem die in der Warteschlange befindlichen Elemente ähnlich dem Verhalten des SystemC-Simulationskernels ausgeführt werden.

[0037] In einer bevorzugten Ausführungsform werden „places“ anhand von SimPy-Ressourcen zum Laden/Speichern von Token von/bis und Blockieren, wenn diese Übertragungen verweigert werden definiert. Zusätzlich können „transitions“ in Verbindung mit den „places“ definiert werden, die Blockierprozesse zum Ablegen oder Mitnehmen von Token an/von benachbarten Orten ausführen. Dies bietet eine effektive Modellierungsmethode zur Darstellung verschiedener Operatorspläne (engl. operator schedules). Durch die Zuordnung von Übergängen zu Hardware-Operationen von neuronalen Netzkernels, wobei die Struktur ihrer Verbindungen über Stellen die Ausführungsreihenfolge und das Timing definiert, wird der Simulationsgraph erstellt, der das Verhalten des Kerns bei Verarbeitung auf der Hardware aus der Software-Perspektive darstellt.

[0038] Fig. 2 zeigt, wie ein Faltungskern eines neuronalen Netzes durch einen Ausführungsgraphen (11) ausgedrückt werden kann. Es sei angemerkt, dass der Einfachheit halber in diesem Beispiel nur die innerste Schleife (kernel_w) dargestellt wird. Höhere Loop-Ebenen würden als zusätzliche Stellen an den Graphen Anfang und Ende erscheinen, die über eine Kante auf dem Rückweg verbunden sind. In diesem Beispiel werden Daten durch DMA-Übertragungen direkt aus dem Hauptspeicher in die Recheneinheit verschoben, die die Matrix-Multiplikation durchführt. Das Laden der Eingabe- und Kernel-Daten wird parallelisiert, wodurch der Compute-Node-Block solange blockiert wird, bis Daten aus beiden DMA-Transaktionen verfügbar sind. Nach Durchführung des Rechengangs speichert ein DMA die berechneten Ausgabedaten zurück in den Hauptspeicher. Der gesamte Ablauf des Load-Compute-Speichers wird kernel_w mal durchgeführt, angetrieben von der rekursiven Schleife, die die Hardware-Knoten umgibt. Kreise innerhalb des Diagramms repräsentieren Orte einer bestimmten Größe, was bedeutet, dass sie bis zu einer festen Menge (=Größe) von Token speichern können, die durch vorherige Übergänge bereitgestellt wurden. Alle Übergänge (Quadrat) sind durch Orte verbunden. Um aktiviert zu werden, versucht ein Übergang, eine feste Anzahl von Token vom Eingabeort(n) zu nehmen, die durch den Parameter get und stalls definiert wird, bis die angegebene Anzahl von Token am Ort(n) verfügbar ist. Nach erfolgreichem Erhalt aller notwendigen Token führt der Übergang den zugewiesenen Hardwarevorgang aus und überträgt die durch put angegebene Anzahl von Token an den Ausgabeort, um zu warten, bis neue Token auf der Eingabe erneut ausgeführt werden. Falls der Ausgabeort bereits die maximale Menge an Token speichert, wird der Übergang am Ausgang gestoppt, bis genügend Platz im Ort vorhanden ist. Dieses Blockierverhalten kann z. B. auftreten, wenn der Rechenprozess viel langsamer ist als die Belastung neuer Daten, was normalerweise bei rechengebundenen Systemen der Fall ist. Im Beispiel sind kernel_w-Token direkt am Eingabeknoten der Lastübergänge verfügbar, so dass sie theoretisch kernel_w mal ausführen könnten. Da es aber auch einen Rückwärtspfad vom Ausgangsknoten gibt, müssen an allen Stellen, die mit dem Eingangsknoten verbunden sind, Token verfügbar sein, bevor die Datenlast aktiviert wird. Da der Ort auf dem Rückweg mit 1 Token initialisiert wird, werden die Lasten an der Eingabe einmal und danach nur dann aktiviert, wenn die gesamte Abfolge von Laden, Rechnen und Speichern beendet ist. Nach dieser Struktur kann die Ausführung von Load-Compute-Store-Sequenzen modelliert werden, die für das gesamte Loop-Nest wiederholt werden. Komplexere Ausführungsszenarien können durch Änderung der Loop-Nest-Struktur ausgedrückt werden. Das Einfügen von Lastvorgängen in verschiedenen Stufen des Loop-Nests kann die Bewegung von

Daten zwischen verschiedenen Ebenen der Speicherhierarchie darstellen, während das Abrollen von Schleifen zur parallelen Ausführung von Loop-Body-Vorgängen führt. Auch die Wahl der Placesizes, zusammen mit Put-and-Get-Werten, hat Auswirkungen auf das Hardwareverhalten. Wenn z. B. der Rückwärtspfad in 2 mit einem Wert von 3 angestoßen würde, können statt reiner sequentieller Ausführung die Prozesse Laden, Rechnen und Speichern als Pipeline laufen, da die Eingangslast 3-mal ohne Stocken ausgeführt werden kann und nachfolgende Vorgänge bei erneuter Aktivierung starten kann.

[0039] Fig. 2 zeigt ein einfaches Beispiel, wie Hardwarekonzepte aus einer Beschreibung auf Software-Ebene mit Hilfe von Petri-Net-Graphen (12) ausgedrückt werden können. Die Übersetzung zwischen diesen Darstellungen kann durch den Einsatz von Machine Learning Compiler erreicht werden. Es wird vorgeschlagen, das Opensource-Framework TVM für den gesamten Kompilierungsstack von der Operator-Definition bis hin zum Ausführungsgraphen zu nutzen.

[0040] TVM ist ein Open-Source-Compiler-Stack zur Anwendung hardwarebezogener Modifikationen auf neuronale Netzwerkbeschreibungen aus häufig verwendeten Machine-Learning-Frameworks. Mithilfe von zwei internen Repräsentationen (IR) und IR-spezifischen Bibliotheken können Transformationen auf den gesamten Netzwerkgraphen angewendet werden, wie z. B. Operator Fusion, sowie auf einzelne Betreiber, wobei die Loop-Nest-Struktur gegenüber dem vorgesehenen Hardware-Backend optimiert werden kann.

[0041] Die Abbildung von Graph-Operationen auf Simulationskerne, die bereits im Kompilierungsfluss enthalten sind, ermöglicht es, die Ausführungszyklen der Kernel auf virtuellen Beschleunigerplattformen end-to-end vorherzusagen. Die Terminierung neuronaler Netzbetreiber mittels TVM-spezifischer Bibliotheken wird als Software-Level-Modifikation bezeichnet. Der Begriff Scheduling wird wie von TVM verwendet, um eine Operatorberechnung zu beschreiben, indem Tensoren variabler Größen für Ein- und Ausgänge zusammen mit ihren mathematischen Beziehungen wie Addition oder Multiplikation definiert werden. Schleifen-Level-Modifikation anwenden, z. B. Halide-Sprache (Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. 2020. Internationale Konferenz zur Architekturunterstützung für Programmiersprachen und Betriebssysteme. 369-383), zum Kacheln, Abrollen oder Nachbestellen, kann man eine Vielzahl unterschiedlicher Hardware-Zeitpläne beschreiben, ausgehend von derselben unveränderten Berechnungsdefinition, während die Verwendung von TVMs-Planungsbibliotheken die Richtigkeit der

Ergebnisse sicherstellt. Zeitpläne werden als abstrakte Syntaxbäume (AST) dargestellt, mit For-Loops der Betreiberbeschreibung als Knoten und Hardware-Operationen, als Load und Compute, an ihrem Körper. Das Durchlaufen der Graphen-Knoten und -Kanten ermöglicht den Zugriff auf die Hardware-Operationen sowie Adressinformationen. In dem in **Abb. 2** gezeigten Szenario wird die For-Loop der Variablen `kernel_w` als Graph-Node dargestellt, während der Loop-Body als Speichervorgang dargestellt wird und die Ausgabe der Multiplikation von 2 Datenlasten zurück in den Hauptspeicher schreibt. Aus den Operatorindizes können Adressinformationen von Lasten und Speichern berechnet werden, während die Schleifengrenze die Wiederholungen des Vorgangs und die folgenden Adresswerte bestimmt. Das Kompilieren des Bedienerzeitplans in das Ausführungsdiagramm erzeugt Übergänge für die Hardware-Operationen am Loop-Body, umgeben von einem Rückwärtspfad, wobei `put`, `get` und `place_size`-Werte von den AST-Knoten abgeleitet werden. Die Erfassung der Informationen für die Durchführung von Hardware-Operationen, während die korrekte Ausführungsreihenfolge bereits durch die AST-Struktur definiert ist, ermöglicht die Erstellung des Ausführungsdiagramms direkt aus der Baumdefinition, den Prozess vollständig automatisiert über die TVMs-Pass-Infrastruktur durchführen. Durch das Hinzufügen spezieller Konstrukte für Parallelität und Pipelining zur bestehenden TVM-Bibliothek kann die Anzahl der Zeitpläne, die ausgedrückt werden können, erweitert werden, während der Bereitstellung von methodenabhängigen Interpretationen für die Erstellung des Simulationsgraphen in Bezug auf diese Szenarien.

[0042] Die tatsächliche Verhaltensnäherung erfolgt durch Ausführen des Workload-Graphen auf virtuellen Prototypen, die die Hardware-Funktionalität nachahmen. Die Abbildung der Graphenoperationen auf den Virtuellen Prototypen (VP) erfolgt beim Aufbau des Simulationsgraphen. Die Durchführung des Mappings in dieser Phase ermöglicht die Überprüfung auf gültige Mappings, bevor der vollständige Graph erstellt oder die Simulation durchgeführt wird. Dies spart Zeit, da nur gültige Zeitplan-/Hardware-Konfigurationen vorausgewählt werden. Wenn die Anforderungen an die Diagrammoperationen nicht mit den tatsächlichen Hardwarefunktionen übereinstimmen, z. B. ist die Menge der zu übertragenden parallelen Daten höher als die Bandbreite der Ladeeinheiten, wird dies sofort erkannt.

[0043] Virtuelle Hardwareklassen werden auf bspw. Python-Ebene definiert, was nicht nur Flexibilität bei der Konfiguration der Hardwareattribute bietet, sondern auch eine hohe Kompatibilität zu TVM und den Schnittstellen der Simulationsgraphen. Wenn alle Hauptkomponenten in **Fig. 1** in der gleichen Abstraktion dargestellt werden, entsteht beim Ausführen des

End-to-End-Flows nur ein geringer Overhead, da keine sprachübergreifenden Übersetzungen erzwungen werden. Durch die Verwendung von SimPy-Bibliothekskonstrukten für die Implementierung kann die Hardwarefunktionalität ähnlich wie bei der Transactionlevel-Modellierung von Systeme ausgedrückt werden. Prozesse von Hardware-Elementen können warten, bis die Prozesse anderer Plattform-Komponenten abgeschlossen sind, um das Blockierverhalten wie Bus-Verkehr oder Speicherzugriff nachzuahmen. Der Umfang, in dem diese Prozesse wirken können, wird durch einen flexiblen Satz von Hardware-Attributen begrenzt, die alle mit der Art der Hardware zusammenhängen. Diese sind aus einer benutzerdefinierten Konfigurationsdatei konfigurierbar, die auf eine Reihe von Hardwarevorlagen für Kommunikation, Berechnung, Vernetzung und Datenspeicherung in Bezug auf moderne Machine-Learning-Beschleunigerarchitekturen anwendbar ist. Die Zuordnung von Taktzyklen zu Hardware-Prozessen, bezogen auf die Hardware- und Workload-Attribute, generiert die Zyklenzahl für die Simulationsausgabe, während aktive und Stillstand-Zyklen von Hardware-Komponenten für jede Klasse registriert werden können. Die Simulation in der vorgeschlagenen Methodik basiert auf dem Simulationsgraphen, der den auszuführenden Software-Arbeitsaufwand als Datenfluss und die Ausführungsreihenfolge der Hardwareprozesse darstellt. Operationen des Simulationsgraphen, die einem bestimmten Hardware-Block zugeordnet sind, können ihn innerhalb der eigentlichen Simulation belegen, um die vorgesehene Graph-Operation der Übertragung oder Berechnung von Daten durchzuführen. Es kann vorgesehen sein, dass die Hardware für eine bestimmte Aufgabe zur Laufzeit programmiert und aufgesetzt ist, z.B. können Adressen für die Datenübertragung entsprechend definiert werden, während die Hardware-Attribute den Umfang definieren. Alle Prozesse der Graph-Operationen oder der gemappten Hardware-Elemente werden in einer gemeinsamen Simulationsumgebung in die Warteschlange gestellt und werden in der Reihenfolge des SimPy-Simulationskernels ausgeführt.

[0044] Es wurde eine Ausführungsform einer Methodik zur flexiblen Erstellung von Simulationsumgebungen für die Abschätzung der Latenz von DNN-Workloads auf ML-Beschleunigern vorgestellt. Die Integration des Compiler für maschinelles Lernen in den vorgesehenen Ablauf ermöglichte die Erstellung hardwarefähiger Simulationsgraphen aus den Softwarekernen und die direkte Abbildung auf virtuelle Beschleunigerkomponenten in einer durchgängigen Weise. Alle Schnittstellen auf dem gleichen Sprachniveau zu haben, begünstigt das Zusammenspiel von Compiler, Graph und virtuellem Plattformmodell. Die Möglichkeit, Simulationsumgebungen für eine Vielzahl von Datenfluss- und Beschleunigerarchitekturenkombinationen zu generieren, begünstigt die

Verwendung der vorgeschlagenen Methodik im Rahmen von HW/SW Co-Design. Die Kombination der vorgeschlagenen Methodik mit dem TVMs-Blackbox-Automatisierungstool AutoTVM ermöglicht die Auswertung einer großen Anzahl von Kernel-Zeitplänen, die auf einem Kostenmodell basieren, das durch Simulationsläufe trainiert wird. Da Zeitplan-Änderungen und Hardware-Plattform-Änderungen auf Python-Ebene durchgeführt werden, besteht ein großes Potenzial für die Durchführung von DSE durch Bewertung beider innerhalb der Optimierungsschleife.

[0045] Fig. 3 zeigt beispielhaft ein Informationsflussdiagramm einer Ausführungsform der Erfindung zum Optimieren von Parametern eines DNN (21a) und eines Hardwarebeschleunigers (21b) des DNN. Das DNN (21a) und der Hardwarebeschleuniger können in einem mobilen Gerät, wie einem Embedded Device (21), verwendet werden. Vorzugweise ist das mobile Gerät ein Sensor, welcher erfasste Sensorwerte direkt mit dem DNN (21a), ausgeführt auf dem Hardwarebeschleuniger (21b), verarbeitet. Der Sensor kann mit dem DNN und anhand seiner Sensordaten eine Mustererkennung ausführen.

[0046] In einer ersten Ausführungsform der Fig. 3 kann bei Block 202 begonnen werden. In diesem Block wird gegeben einer Aufgabe, für welches das DNN trainiert werden soll, dieses basierend auf einem entsprechenden Datensatz für diese Aufgabe trainiert. Beispielsweise kann das Training derart erfolgen, dass das DNN ein Bildklassifikator oder ähnliches ist. Nachdem das Training abgeschlossen ist, liegt ein trainiertes DNN mit seiner Architektur in Block 203 vor, welches einem Machine-Learning-Compiler wie TVM in Block 204 übergeben und kompiliert wird.

[0047] In einer zweiten Ausführungsform der Fig. 3 wird ein bereits trainiertes DNN in Block 203 bereitgestellt.

[0048] Durch den Einsatz von Machine-Learning-Compilern wie TVM können die Zeitpläne zum Ausführen des DNN erstellt werden. Das Einbinden des Compilers zur Übersetzung von Zeitplänen ermöglicht jedes Mal eine erneute Erstellung von Simulationsmodellen, wenn Parameter in der Software geändert werden.

[0049] Nach einem Vorliegen des Zeitplans (engl. Schedule) in Block 205 wird dieser in ein Simulationsgraphen in Block 206 umgewandelt. Wie zuvor erläutert, ist der Simulationsgraph vorzugweise ein Petri-Netz Graph (12). Dieser Simulationsgraph wird dann in Block 207 einer Simulation bereitgestellt, die das Verhalten des DNN beim Ausführen auf einer Hardware simuliert. Am Ende der Simula-

tion wird abhängig von dem Verhalten des DNN eine Performance des DNN auf der Hardware ermittelt.

[0050] Zusätzlich zu dem eben erläuterten Aufbereiten des DNN zum Simulationsgraph für die Simulation wird abhängig von Hardwareeigenschaften des Beschleunigers (21b) ein Hardwaremodell des Beschleunigers (21b) in Block 209 erstellt.

[0051] Das Hardwaremodell wird auch der Simulation in Block 207 bereitgestellt. Damit wird durch die Simulation genau das Verhalten des DNN auf dem Beschleuniger simuliert.

[0052] Vorzugsweise ist die Simulation in Block 207 ein ereignisbasierte Simulation, bspw. SimPy.

[0053] Basierend auf dem Ergebnis der Simulation kann anschließend entschieden werden, ob die Performance des Beschleunigers oder der DNN auf dem Hardwarebeschleuniger oder das System aus DNN mit Hardwarebeschleuniger genügend ist. Fall die Performance genügt, kann das mobile Gerät (21) entsprechend hergestellt und/oder konfiguriert werden. Falls die Performance nicht genügt, dann können Parameter charakterisierend das DNN und/oder Parameter charakterisierend den Hardwarebeschleuniger entsprechend angepasst werden. In anderen Worten, die Simulationsergebnisse liefern präzise Leistungsbeurteilungen bspw. zur Latenz des HW/SW-Systems (21), die zurückgeführt werden können, um Änderungen an den SW- und HW-Konfigurationen vorzunehmen. Wenn beispielsweise die Versorgung der Recheneinheiten mit Daten zu viel Zeit in Anspruch nehmen würde, könnte den Kommunikationseinheiten der Hardware mehr Bandbreite zugeordnet werden, um schneller mehr Daten zur Verfügung zu stellen.

[0054] Für den Fall, dass Parameter geändert wurden, kann das eben beschriebene Verfahren erneut ausgeführt werden, um erneut eine Simulation mit den geänderten Hardwaremodell und/oder Graphen auszuführen. Das Ergebnis ist eine maßgeschneiderte Hardware, die auf die optimierte Ausführung des Software-Workloads zugeschnitten ist.

[0055] Diese Änderungen erfordern oft Änderungen am SW-Zeitplan, der die Ausführungsreihenfolge der Datenversorgung und -berechnung festlegt. Eine mehrfache Ausführung dieser Optimierungsschleife kann die Leistung im Vergleich zu Allzweck-Hardware verbessern und gleichzeitig die Fertigungskosten senken. Die Ergebnisse des Optimierungsprozesses bestimmen eine optimale Hardwaresystemspezifikation.

[0056] Die Möglichkeit, neue Simulationsmodelle für geänderte Software-Zeitpläne zu generieren, ermög-

licht die Nutzung der Erfindung im Rahmen von HW/SW Co-Design.

[0057] Im Entwicklungsprozess von intelligenten Geräten (engl. Smart Devices) kann der KI-Algorithmus mit der eigenen Hardware kooptimiert werden, indem die Auswirkungen von Parameteränderungen auf beiden Ebenen in einer gemeinsamen Simulationsumgebung bewertet werden.

ZITATE ENTHALTEN IN DER BESCHREIBUNG

Diese Liste der vom Anmelder aufgeführten Dokumente wurde automatisiert erzeugt und ist ausschließlich zur besseren Information des Lesers aufgenommen. Die Liste ist nicht Bestandteil der deutschen Patent- bzw. Gebrauchsmusteranmeldung. Das DPMA übernimmt keinerlei Haftung für etwaige Fehler oder Auslassungen.

Zitierte Nicht-Patentliteratur

- Halide-Sprache (Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. 2020. Internationale Konferenz zur Architekturunterstützung für Programmiersprachen und Betriebssysteme. 369-383 [0041])

Patentansprüche

1. Computerimplementiertes Verfahren (20) zum Ermitteln einer Performance eines maschinellen Lernsystems (21a) auf einer Recheneinheit (21b), umfassend folgende Schritte:

Erstellen eines Hardwaremodells der Recheneinheit aus bereitgestellten technischen Spezifikation der Recheneinheit (21b);

Erstellen eines Simulationsgraphen (12) basierend auf dem maschinellen Lernsystem (21a);

Simulieren eines Ausführens des maschinellen Lernsystems (21a) auf der Recheneinheit (21b) unter Verwendung des Hardwaremodells und des Simulationsgraphen (12), wobei die Simulation eine ereignisbasierte Simulation ist;

Ermitteln der Performance abhängig von dem Ergebnis der Simulation.

2. Verfahren (20) nach einem der vorhergehenden Ansprüche, wobei das maschinelle Lernsystem (21a) ein neuronales Netz ist, wobei beim Schritt des Erstellens des Simulationsgraphen (12) ein neuronaler Netzwerkgraph in eine Baumdarstellung durch einen Maschine Learning Compiler umgewandelt wird, wobei die Baumdarstellung in ein Petri-Netz Graph (12) umgewandelt wird, welcher der Simulation bereitgestellt wird.

3. Verfahren (20) nach einem der vorhergehenden Ansprüche, wobei interne Hardwareprozesse mit einer ereignisbasierten Simulation simuliert werden.

4. Verfahren (20) nach einem der vorhergehenden Ansprüche, wobei die Recheneinheit (21b) ein Hardwarebeschleuniger für das maschinelle Lernsystem ist (21a).

5. Computerimplementiertes Verfahren (20) zum gemeinsamen Optimieren einer neuronalen Netzwerkkonfiguration und einer Recheneinheit (21b) zum Ausführen des neuronalen Netzes hinsichtlich einer optimalen Hardwareperformance umfassend die Schritte nach einem der vorhergehenden Ansprüche, wobei abhängig von der ermittelten Performance bestimmt wird, ob eine Anpassung von Parametern charakterisierend das neuronale Netz und/oder Parameter charakterisierend die Recheneinheit innerhalb eines vorgegebenen Parameterbereichs des jeweiligen Parameters durchgeführt wird, wobei die Schritte des Erstellens und Ausführens der Simulation und Ermittlern der Performance erneut basierend auf den geänderten Parametern ausgeführt wird, wobei das Vorgehen mehrmals wiederholt wird, insbesondere bis eine vorgegebene Soll-Performance erzielt wird.

6. Verfahren (20) nach Anspruch 5, wobei dann, wenn die Soll-Performance erzielt wird, wird ein

System (21) gemäß den Parametern charakterisierend das neuronale Netz und Parameter charakterisierend die Recheneinheit, mit welchen die Simulation die Soll-Performance erreicht hat, hergestellt und/oder konfiguriert.

7. Vorrichtung, welche eingerichtet ist, das Verfahren (20) nach einem der vorhergehenden Ansprüche ausführen.

8. Computerprogramm, umfassend Befehle, die bei der Ausführung des Programms durch einen Computer diesen veranlassen, das Verfahren nach Anspruch 1 bis 6 ausführen.

9. Maschinenlesbares Speichermedium, auf dem das Computerprogramm nach Anspruch 8 gespeichert ist.

Es folgen 3 Seiten Zeichnungen

Anhängende Zeichnungen

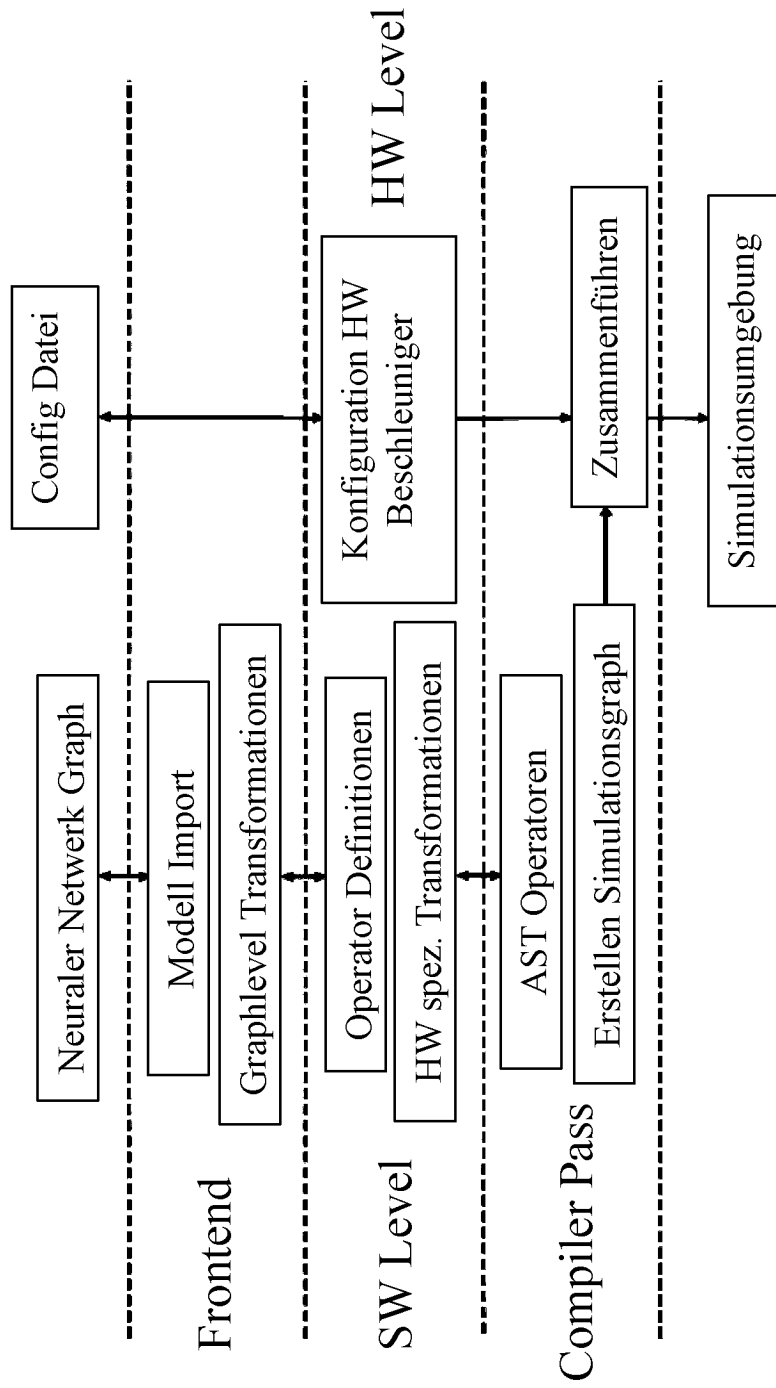


Fig. 1

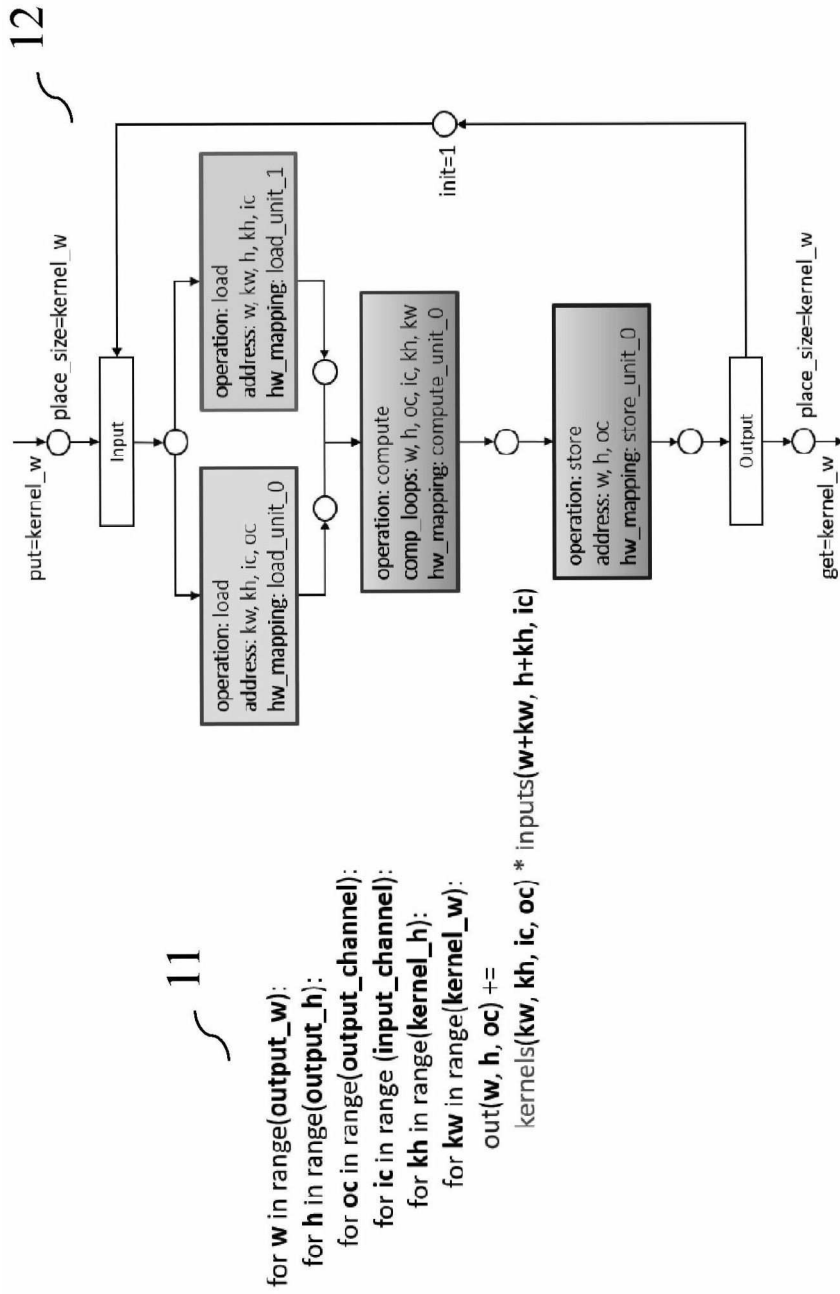


Fig. 2

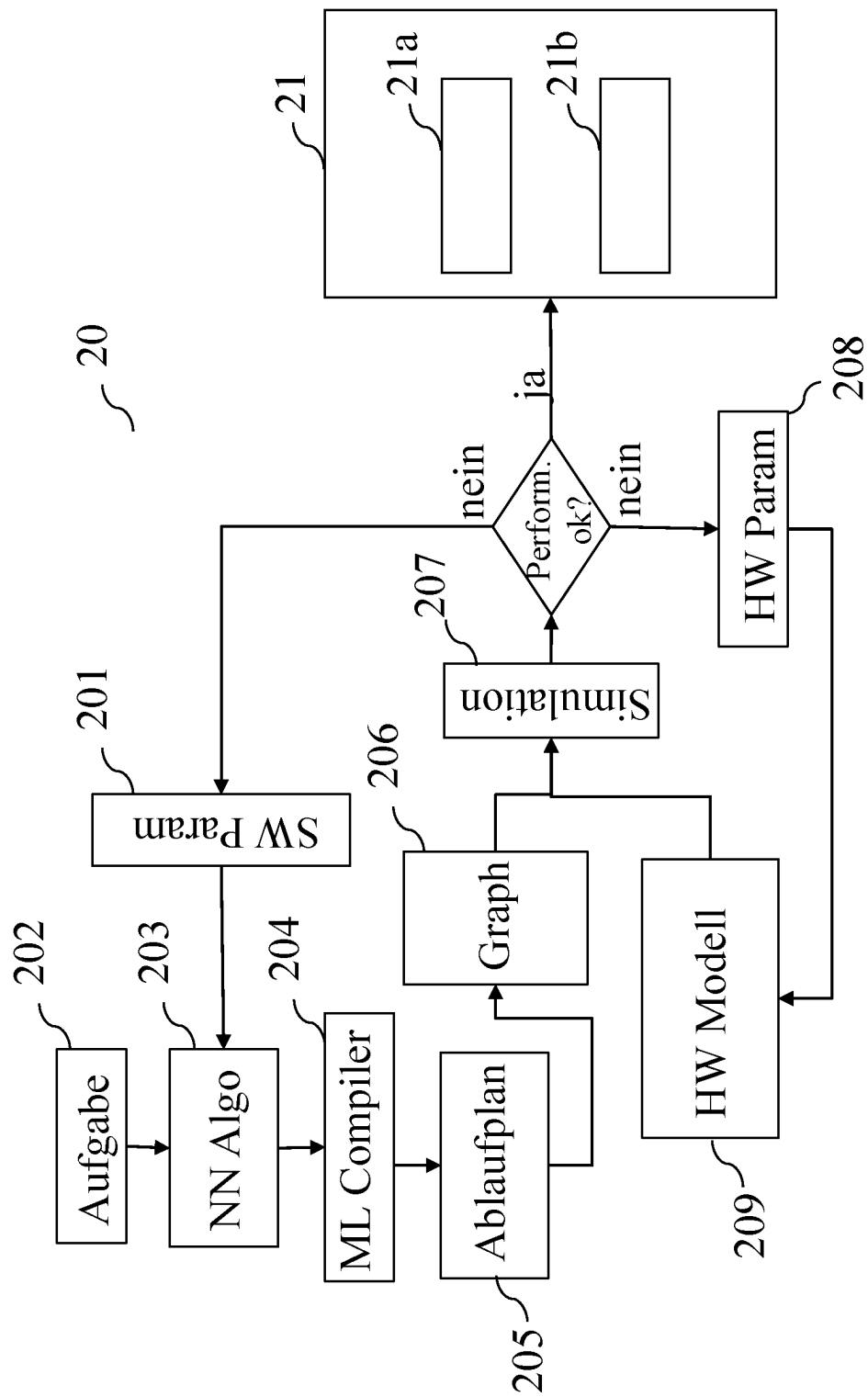


Fig. 3