



Karlsruhe Institute of Technology
Fakultät für Elektrotechnik



Institut für Technik der
Informationsverarbeitung (ITIV)

Implementierung und Evaluation von Systolic Arrays für maschinelles Lernen

Bachelorarbeit von

cand. el. Ahmet Narin

21 Dezember 2019

Institutsleitung: Prof. Dr.-Ing. Dr. h.c. J. Becker
Prof. Dr.-Ing. Eric Sax
Prof. Dr. rer. nat W. Stork

Betreuer: M. Sc. Tim Hotfilter

Zusammenfassung

Unser Alltag und Geschäftsleben wird immer mehr von machine Learning (ML) beeinflusst. Beispiele für den Einsatz begegnen uns im täglichen Leben bereits überall. Einige davon sind Gesichts-und Spracherkennung, personalisierte Produktempfehlungen bei Amazon oder Börsenprognose für Spekulanten etc. Aber auch in sicherheitskritischen Systeme wie autonomes Fahren wird das ML-Verfahren eingesetzt. Das Ziel von dieser Arbeit ist, dass die Datenverarbeitung von solchen sicherheitskritischen Anwendungen beschleunigt und parallele Multiplikation-und Akkumulations-Operationen ausgeführt werden. Somit wird die Rechenleistung von Embedded Systems und IoT-Geräten erhöht und

Urheberrecht

ARM[®], AMBA[®], AXI[™], Cortex[™], TrustZone[™], SecurCore[™], DSTREAM[™] und weitere im Text erwähnte ARM-Produkte sowie die entsprechenden Logos sind Marken oder eingetragene Marken der Advanced RISC Machines Ltd.

Xilinx[®], Zynq[™] und weitere im Text erwähnte Xilinx-Produkte sowie die entsprechenden Logos sind Marken oder eingetragene Marken der Xilinx Inc.

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit (bzw. Masterarbeit) selbständig und unter Beachtung der Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) in der aktuellen Fassung angefertigt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich übernommene Stellen als solche kenntlich gemacht.

Karlsruhe, den 21. Dezember 2019

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Zielsetzung	2
1.3	Das Mooresche Gesetz	2
2	Grundlagen	3
2.1	Matrix-Matrix Multiplikation	3
2.2	1D-Faltung und 2D-Faltung	4
2.2.1	1D-Faltung bzw. Faltung	4
2.2.2	2D-Faltung	4
2.3	Maschinelles Lernen	5
2.4	Kuenstliche neuronale Netze	5
2.4.1	Aktivierungsfunktionen	7
2.5	Lernphase	9
2.5.1	Supervised Learning	9
2.5.2	Unsupervised Learning	9
2.6	Grundlage der Hardwarebeschreibungssprache VHDL	10
3	Konzept	11
3.1	Grundlagen der Systolic Array Architektur	11
3.2	Why Systolic Architectures?	12
3.2.1	Grundidee	12
4	Implementierung	15
5	Bewertung des Gesamtsystems	16
6	Fazit	17

Kapitel 1

Einleitung

1.1 Motivation

Neue innovative Projekte wie autonomes Fahren oder auch selbstfliegende Autos in der Zukunft und industrielle Automatisierung müssen immer mehr mit umfangreichen Datenmenge umgehen. ML-Methoden werden in solchen Anwendungen häufig verwendet, damit die Daten analysiert werden können bzw. die Maschinen etwas neues aus Daten ohne explizite Programmierung lernen. Insbesondere im Kontext von Echtzeitsystemen stellt sich die Latenz während der Datenverarbeitung als Flaschenhals dar.

Die CPU-Hardwarearchitektur kann dieses Problem leider nicht lösen, da CPUs für allgemeinen Gebrauch vorgesehen sind. GPU (graphics processing unit) ist für solchen rechenintensiven Anwendungen besser geeignet, um die Datenverarbeitung zu beschleunigen. Aber sie verbrauchen viel Energie und für mobile Geräte und Embedded Systems nicht einsetzbar. Eine spezielle Hardwarearchitektur soll für diesen Zweck eingesetzt werden. Es soll wenig Strom verbrauchen und trotz dieser Anforderung auch schnell und mehrfache vorkommende Multiplikation-und Addition Operationen bearbeiten.

Das **FPGA** (field-programable gate array) kann diese Anforderungen erfüllen. Das FPGA besteht aus internen Hardware-Blöcken, die mit anderen von einem Fachperson programmiert werden können, um eine spezielle Anforderungen zu erfüllen. Der Hauptvorteil von FPGA im Gegensatz zu den anderen Hardwarearchitekturen ist, dass die Verbindungen von internen Hardware-Blöcken leicht umprogrammiert und während dem Einsatz Änderungen oder Verbesserungen vorgenommen werden. Verschiedene ML-Algorithmen können damit implementiert und evaluiert werden.

Kapitel 2

Grundlagen

2.1 Matrix-Matrix Multiplikation

Bei der Matrix-Matrix Multiplikation geht es um zwei Matrizen, die miteinander multipliziert werden. Aber nicht jede Matrizen sind miteinander multiplizierbar. D.h die Spaltenmatrix der ersten Matrix muss mit der Zeilenzahl der zweiten Matrix übereinstimmen. Das Ergebnis ist auch eine Matrix, die dann als Produktmatrix genannt wird.

Gegeben seien zwei Matrizen. A ist eine $m \times n$ -Matrix.

B ist eine $n \times p$ -Matrix.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{np} \end{pmatrix}$$

Die Produktmatrix C ergibt sich:

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ c_{21} & c_{22} & \dots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mp} \end{pmatrix}$$

Die Produktmatrix C besteht aus viele Multiplikationen und Additionen.

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj} = \sum_{k=1}^n a_{ik}b_{kj}$$

$$\mathbf{C} = \begin{pmatrix} a_{11}b_{11} + \dots + a_{1n}b_{n1} & a_{11}b_{12} + \dots + a_{1n}b_{n2} & \dots & a_{11}b_{1p} + \dots + a_{1n}b_{np} \\ a_{21}b_{11} + \dots + a_{2n}b_{n1} & a_{21}b_{12} + \dots + a_{2n}b_{n2} & \dots & a_{21}b_{1p} + \dots + a_{2n}b_{np} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \dots + a_{mn}b_{n1} & a_{m1}b_{12} + \dots + a_{mn}b_{n2} & \dots & a_{m1}b_{1p} + \dots + a_{mn}b_{np} \end{pmatrix}$$

Bei einer MM-Multiplikation findet mpn -Multiplikationen und $mp(n-1)$ -Additionen statt.

2.2 1D-Faltung und 2D-Faltung

2.2.1 1D-Faltung bzw. Faltung

Die Faltung mit eindimensionalen Signalen wird als 1D-Faltung oder nur Faltung bezeichnet. In der digitalen Bildverarbeitung und Signalverarbeitung findet meistens diskrete Faltung statt.

Die Faltung von zwei diskreten Funktionen wird durch folgende Formel berechnet:

$$f[n] = a[n] * b[n] = \sum_{k=-\infty}^{\infty} a[k]b[n - k]$$

Die Faltung erfolgt durch Multiplizieren und Akkumulieren der Momentanwerte der überlappenden Abtastwerte. Dabei soll ein Signal umgedreht sein.

2.2.2 2D-Faltung

Dieses Grundkonzept für 1D-Faltung gilt auch für die 2D-Faltung, wenn die Signale 2 Dimensionen haben.

Analog kann die Faltung in 2D definiert werden.

$$f[x, y] = a[x, y] * b[x, y] = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} a[j, k]b[x - j, y - k]$$

2.2.2.1 Beispiel für 2D-Faltung

Hier ist ein Beispiel für 2D-Faltung. Es gibt ein 7x7 Eingangsmatrix I (inputmatrix) und eine Filtermatrix K (kernel) 3x3. Die Werte der Ausgangsmatrix werden durch Multiplikation und Addition von entsprechenden Pixelwerte von I und K berechnet. Siehe Beispiel:

$$\begin{pmatrix}
 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
 1 & 1 & 0 & 0 & 0 & 0 & 0
 \end{pmatrix}
 \begin{pmatrix}
 1 & 0 & 1 \\
 0 & 1 & 0 \\
 1 & 0 & 1
 \end{pmatrix}
 =
 \begin{pmatrix}
 1 & 4 & 3 & 4 & 1 \\
 1 & 2 & 4 & 3 & 3 \\
 1 & 2 & 3 & 4 & 1 \\
 1 & 3 & 3 & 1 & 1 \\
 3 & 3 & 1 & 1 & 0
 \end{pmatrix}$$

I
 K
 $I * K$

$$\begin{array}{ccc}
 \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} & * & \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 4 & 3 & 4 & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 1 & 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 & 0 \end{pmatrix} \\
 I & K & I * K
 \end{array}$$

$$\begin{array}{ccc}
 \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} & * & \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 4 & 3 & 4 & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 1 & 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 & 0 \end{pmatrix} \\
 I & K & I * K
 \end{array}$$

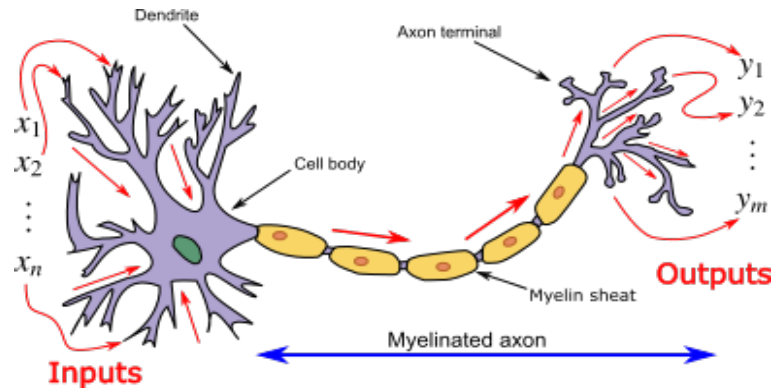
2.3 Maschinelles Lernen

Beim ML wird das System so programmiert, damit das aus eingegebenen Daten automatisch lernt und sich mit der “Erfahrung” verbessert. Das heißt, das System wird mit Trainingsdaten trainiert. Hier bedeutet das Lernen: die eingegebenen Daten zu erkennen, verstehen und auf der Grundlage der gelieferten Daten eine sinnvolle Entscheidung zu treffen.

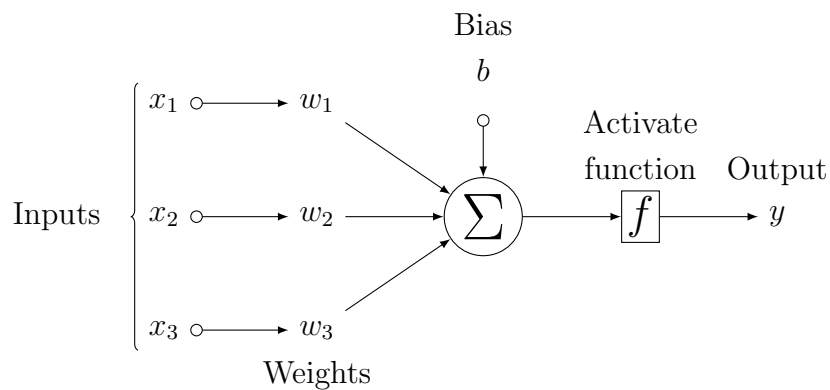
Aber nicht alle Entscheidungen können auf der Grundlage aller möglichen Eingaben berücksichtigt werden. Um dieses Problem zu lösen, werden hier Algorithmen entwickelt, die das maschinelle Lernen optimieren und schneller machen.

2.4 Kuenstliche neuronale Netze

Kuenstliche neuronale Netze sind beliebte Technik des maschinellen Lernens, die den Mechanismus des Lernens in biologischen Organismen simulieren. Das menschliche Nervensystem enthält Zellen, die als Neuron bezeichnet wird. Die Neuronen sind unter Verwendung von Axonen miteinander verbunden.



Dieser biologische Mechanismus wird in kuenstlichen neuronalen Netzen übertragen. Ein Neuron besitzt viele Eingaben und jede Eingabe des Neurons wird mit einem Gewicht versehen. Die gewichteten Eingaben werden dann durch eine Übertragungsfunktion akkumuliert und daraus resultierende Netzeingabe in eine Aktivierungsfunktion gegeben, die die Ausgabe bestimmt.



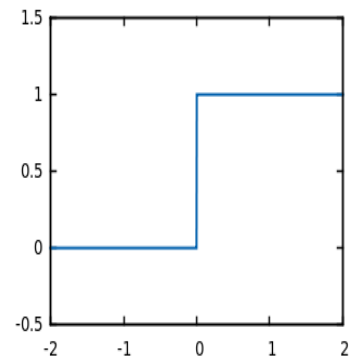
2.4.1 Aktivierungsfunktionen

Eine Aktivierungsfunktion ist eine mathematische Funktion und entscheidet, ob ein Neuron aktiviert werden soll oder nicht, indem sie die gewichtete Summe berechnet und eine Bias hinzufügt. Diese Funktionen sind:

2.4.1.1 Schwellenwertfunktion

Die Schwellenwertfunktion (engl. heaviside step function oder auch binary step function) nimmt nur die Werte 0 und 1 an. Liegt der Eingabewert über oder unter einem bestimmten Schwellenwert, wird das Neuron aktiviert und sendet das Eingangssignal. Für die Eingabe $e \geq 0$ nimmt der Ausgangswert 1, ansonsten 0.

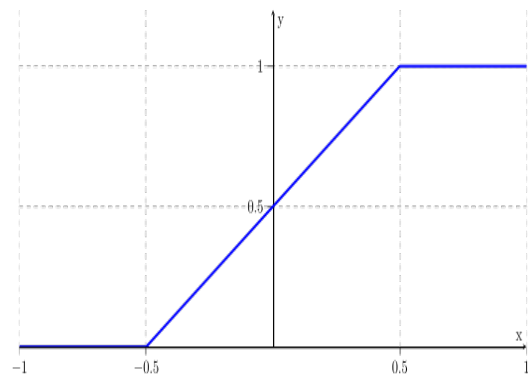
$$\varphi(e) = \begin{cases} 0 & \text{wenn } e < 0 \\ 1 & \text{wenn } e \geq 0 \end{cases}$$



2.4.1.2 Stückweise lineare Funktion

Stückweise lineare Funktion läuft in einem begrenzten Intervall linear ab. Außerhalb des Intervalls besitzt die Funktion einen konstanten Wert.

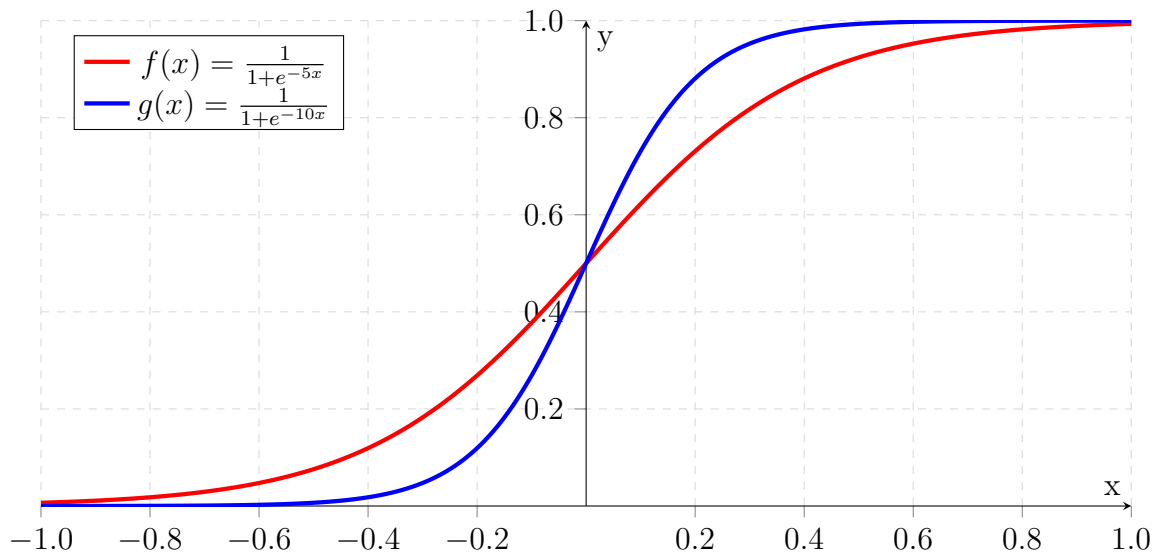
$$\varphi(e) = \begin{cases} 0 & \text{wenn } e \leq -\frac{1}{2} \\ 1 & \text{wenn } e \geq \frac{1}{2} \\ e + 1/2 & \text{wenn } -\frac{1}{2} < e < \frac{1}{2} \end{cases}$$



2.4.1.3 Sigmoidfunktion

Sigmoide Funktionen sind sehr häufig verwendete Funktionen. Sie besitzen ein variables Steigungsmaß α , dass die Krümmung des Funktionsgraphen beeinflusst. Differenzierbarkeit der Sigmoidfunktion ist für einige ML-Verfahren wie Backpropagation-Algorithmus ein Vorteil. Der Nachteil ist ein starker negativer Eingangssignal, damit kann es während des Trainings hängen bleiben.

Sie ist definiert durch:



$$\varphi_{\alpha}^{sig}(e) = \frac{1}{1 + \exp(-\alpha e)}$$

2.4.1.4 Rectifier (ReLU)

Es wird besonders in Deep-Learning Modellen wie CNN(Convolutional Neural Networks) eingesetzt. Dies ist auch als Rampenfunktion bekannt. ReLU Funktion ist definiert durch:

$$\varphi(e) = \max(0, e)$$



2.5 Lernphase

2.5.1 Supervised Learning

Beim supervised Learning brauchen die Trainingsdaten Label (Lösungen). D.h. man muss als Entwickler dem Modell vorher sagen, was die Lösung ist. Wenn man beispielsweise einem Bildererkenner beibringen will, Hunde-und Katzenbilder zu unterscheiden, muss vorher ein Mensch alle Trainingsbilder anschauen und notieren was zu sehen ist. Hier ist der Entwickler als "Lehrer" vorgesehen, der das ML-Modell beibringt. Ansonsten weiss der Algorithmus nicht, ob er falsch oder richtig entscheidet.

2.5.2 Unsupervised Learning

Beim Unsupervised Learning soll das ML-Modell aus Daten "lernen", die Bedeutungen von denen noch unbekannt sind. Hier wird dem Algorithmus die Daten ohne Zielvorgabe eingespeist und trainiert. Ohne Daten gibt es natürlich nichts zu "lernen".

Unsupervised Learning wird verwendet, um einen bestimmten Datensatz in verschiedene Gruppen zu gruppieren. Dies wird häufig verwendet, um Kunden für bestimmte Eingriffe in verschiedene Gruppen zu segmentieren.

2.6 Grundlage der Hardwarebeschreibungssprache VHDL

Kapitel 3

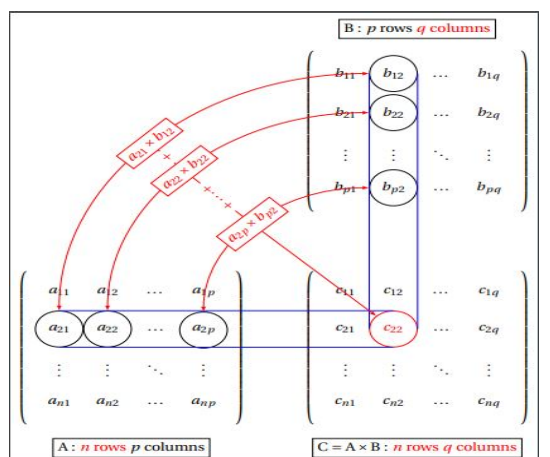
Konzept

3.1 Grundlagen der Systolic Array Architektur

Machine Learning basiert stark auf Berechnungen von Eingabewerten bzw. Daten. Die künstliche neuronale Netze bestehen aus vielen kuenstlichen Neuronen, indem viele Berechnungen stattfinden, die den Ausgabewert des ML-Modells beeinflussen.

Die Frage ist, wie können die ML-Modells noch schneller "lernen" bzw. wie kann man die Berechnungen noch schneller und effektiver durchführen?

Beispiel:Bei herkömmlichen Vorgehensweise der Matrix Multiplikationen wird zuerst überprüft, ob die Matrizen miteinander multipliziert werden kann. Wenn die Spaltenmatrix der ersten Matrix mit der Zeilenzahl der zweiten Matrix übereinstimmen, dann kann es mit der Berechnung fortgesetzt werden. Der Nachteil dieser Berechnungen ist die sequentielle Ausführung. Es findet keine parallele Berechnung statt. Das Ziel ist aber ein Algorithmus zu entwickeln, dass parallele Berechnung und sogar mehrere parallele Berechnungen durchführt.

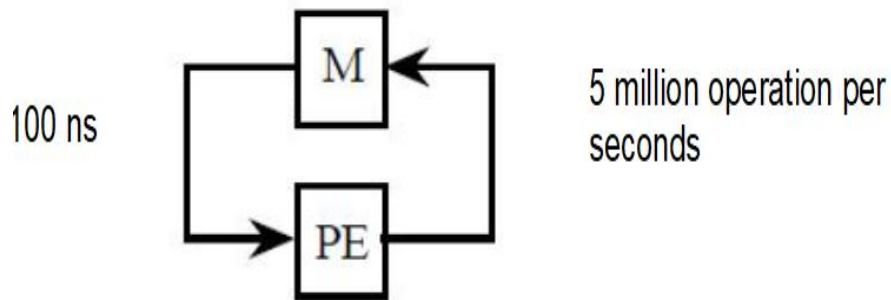


¹<http://www.texample.net/media/tikz/examples/PDF/matrix-multiplication.pdf>

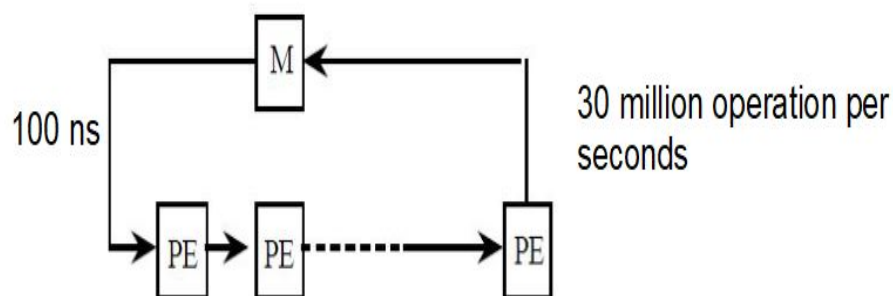
3.2 Why Systolic Architectures?

3.2.1 Grundidee

Die Grundidee des systolischen Arrays ist, dass die Daten rhythmisch aus dem Computerspeicher fließen und laufen durch viele PEs (processing elements), bevor sie in den Speicher zurückkehren.



Wenn es nur ein einziges PE gibt, das aus dem Speicher ein Data holt, verarbeitet und das Ergebnis in den Speicher speichert. Das ist ineffizient und zeitaufwändig.



Aber wenn es mehrere PEs gibt, die Daten mehrmals verarbeiten und das Ergebnis in den Speicher legt. Das ist die Idee, die das System beschleunigt.

Der Begriff "systolisches Array" kann nach einer neuen Innovation in der Rechnerarchitektur klingen. Aber es ist fast 40 Jahre alt. H.T. Kung kam auf diese Idee in den 80er Jahren und veröffentlichte seine Idee im Jahr 1982.²

²<http://www.eecs.harvard.edu/htk/publication/1982-kung-why-systolic-architecture.pdf>

Kapitel 4

Implementierung

Kapitel 5

Bewertung des Gesamtsystems

Kapitel 6

Fazit

Literaturverzeichnis

[Kun] KUNG, H.T.: Why Systolic Architectures?

Tabellenverzeichnis

Abbildungsverzeichnis

