Kapitel 6

Unkonventionelle Konzepte

6.1. Datenflussrechner

6.1.1. Darstellung von Datenflüssen

Die Verfügbarkeit von Daten bestimmt den Gang einer Rechnung, nicht primär ein sequentiell formuliertes Programm.

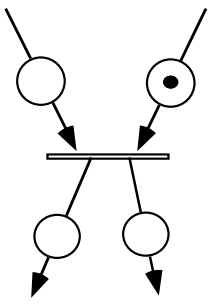
Bei den Daten wird ein Token mitgeführt, das die Gültigkeit der Daten anzeigt. Wenn die Token aller Daten einer Rechnung gesetzt sind, kann die Rechnung ablaufen. Man hat dann eine datengetriebene Organisation. Operationen laufen asynchron und parallel ab.

Daten können einfache oder komplexe Variable sein; die Operanden die Mächtigkeit von Funktionen oder Prozeduren haben.

Daten werden samt ihren Token an die Operationen übergeben. In einem Data-Token steht u. a. der Verweis auf die nächsten Operationen, die mit den Daten ausgeführt werden können.

Über Zeitpunkt und Aufeinanderfolge wird primär nichts gesagt.

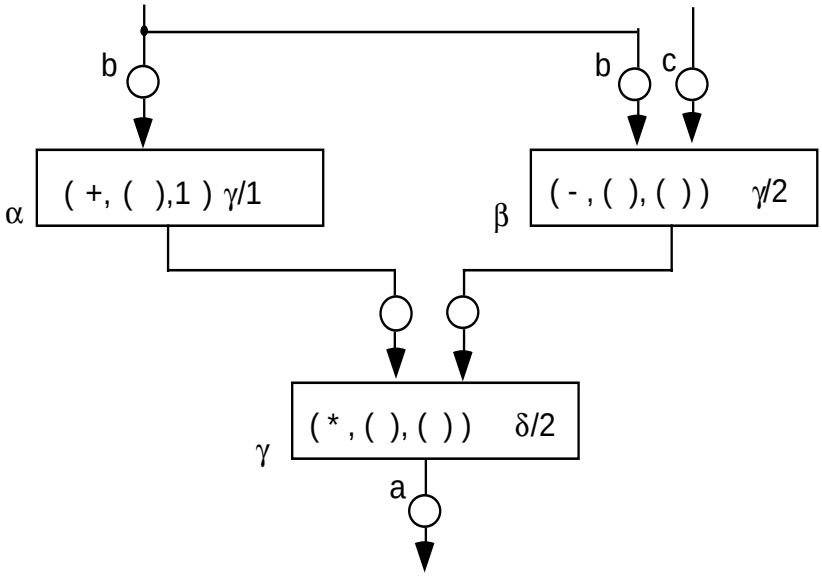
Die Programmstruktur kann auf ein Petrinetz abgebildet werden, dessen Markenplätze die Variablentypen sind und bei denen eine Marke anzeigt, daß ein gültiges Exemplar dieser Variablen zur Rechnung bereitsteht. Die Transitionen sind die Funktionen oder Prozeduren, die die Variablen bearbeiten. Die Feuerungsregeln eines Petrinetzes beschreiben den Ablauf der Operationen eines Datenflussrechners.



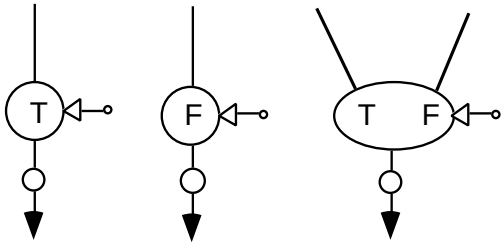
Der Datenflussgraph entspricht dann einem Petrinetz.

In den Transitionen werden die Operationen, die Variablen und die Namen der Folgeoperationen angegeben, wo das Rechnungsergebnis weiter verarbeitet werden soll.

Beispiel: eine Operation a := (b+1) (b-c) wird dargestellt als



Zur Darstellung bedingter Sprünge in Programmen wird man das Petrinetz zu einem Koordinationsnetz erweitern durch Hinzunahme boolescher Bedingungen bei Transitionen: erst eine Bedingung lässt das Token einer Variablen passieren.



Man kann die parallel ausführbaren Operationen auf verschiedene Abstraktionsebenen legen:

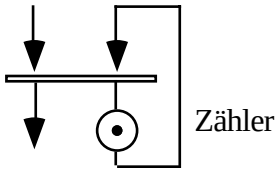
Befehl - Subtask - Task - Prozedur - Modul - Job .

Die Realisierung durch parallel arbeitende Prozessoren wird i. allg. die Parallelität auf der Ebene von Prozeduren suchen.

Die Formulierung einer Rechnung als Datenflussgraph (Petrinetz) legt Anforderungen an eine sprachliche Formulierung in Form einer Datenflusssprache mit fest:

- Freiheit von Seiteneffekten
  call-by-value statt call-by-reference
  (kostet Kopiervorgänge bei komplexen Variablen)

- Single assignment-rule
  Übergabeparameter werden nicht verändert, d. h. Input und Output sind getrennt. Es gibt keine Zuweisung an eine Variable, die auf der rechten Seite aufscheint.
  x := x +y wird ersetzt durch x_1 := x + y
- Sonderstellung von Zählern
  Sie prägt sich aus in einer rücklaufenden Kante im Petrinetz; bei der Initialisierung muß das Token gesetzt sein und die Zählervariable initialisiert werden.



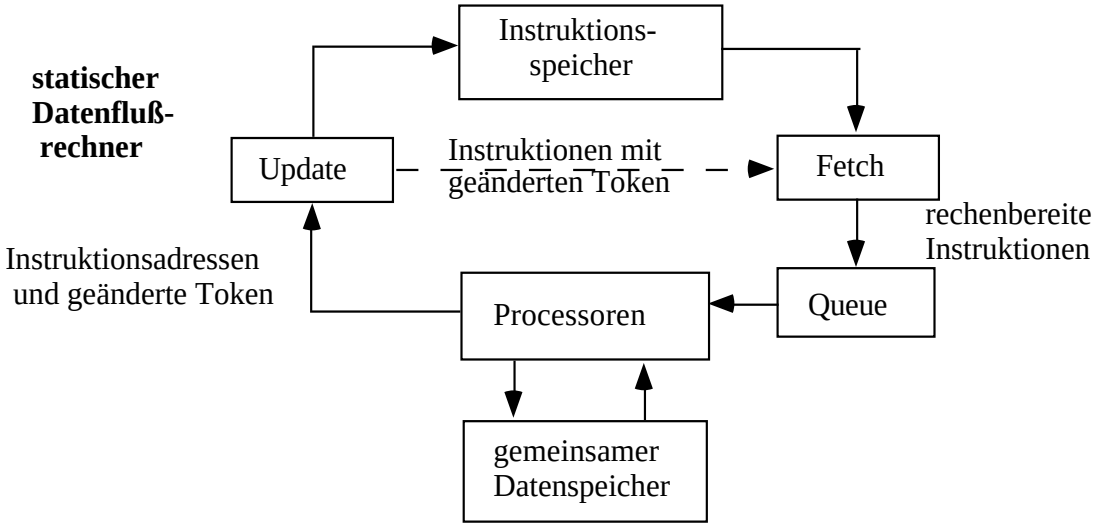
- Auflösen von Iterationen in rekursive Strukturen.

6.1.2. Architekturen für Datenflussrechner

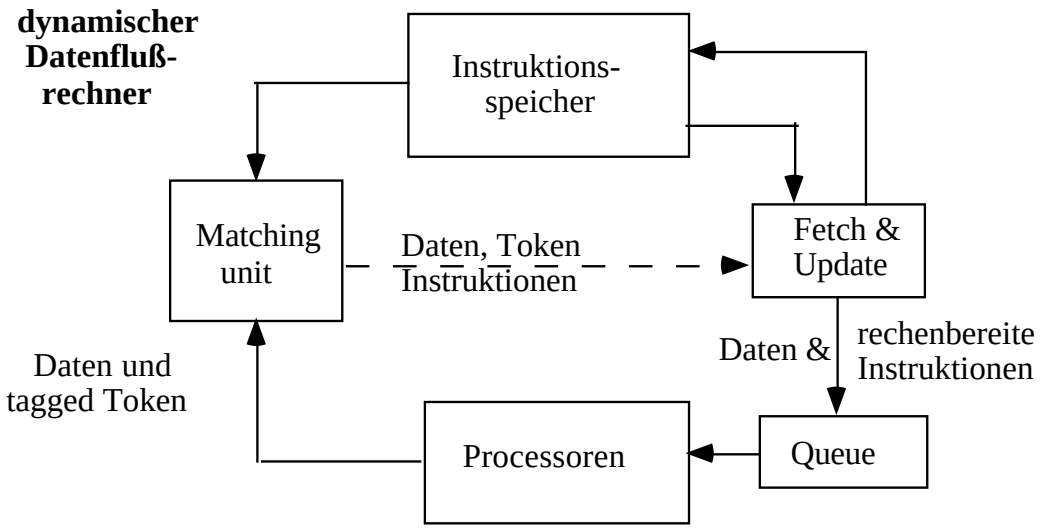
Man unterscheidet statische und dynamische Maschinen. Sie unterscheiden sich in der Art der Behandlung der Token.

Statische Maschinen schreiben als Ergebnis von Rechnungen die neuen gültigen Token in die Transitionen in einem Instruktionsspeicher ein und überprüfen dann diese Transitionen, ob alle Token gesetzt sind. Wenn ja wird die Transition zur Ausführung in eine Warteschlange gepackt und wartet auf einen freien Prozessor.

Alle Prozessoren greifen gemeinsam auf den Datenspeicher zu (homogener Adressraum), laden die Eingabeparameter und schreiben die Ausgaben zurück. Die Daten werden aus dem Speicher gelesen und zurückgeschrieben.



Bei **dynamischen Maschinen** werden die Daten mit bewegt. Man vermeidet damit den gemeinsamen Speicher als Flaschenhals, erkauft mit dem Transport der Daten aus dem Prozessor durch eine Matching Einheit zur Update/Fetch Einheit und in den jeweiligen Prozessor.



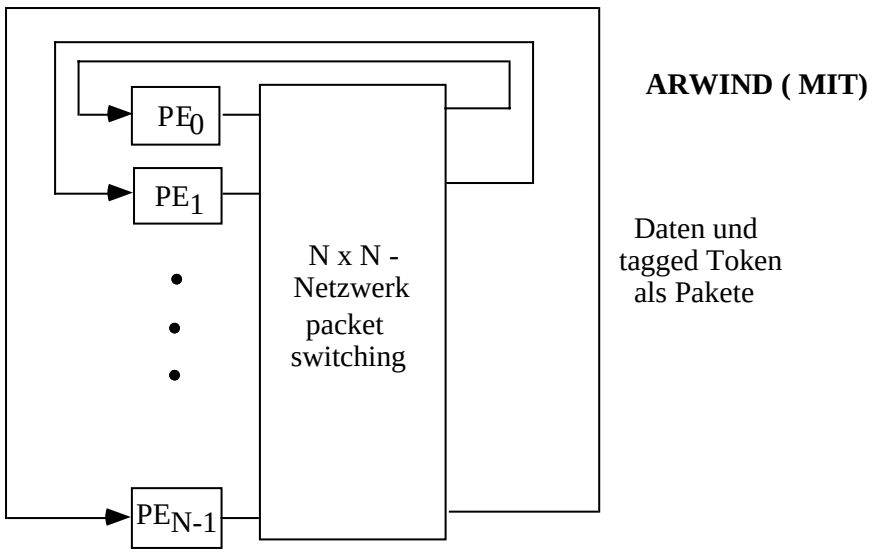
Die Suche nach ausführbaren Operationen geschieht auch hier über einen Instruktionsspeicher in der Update/Fetch Einheit.

Die Matching Einheit schreibt die Token in die jeweiligen Instruktionen ein.

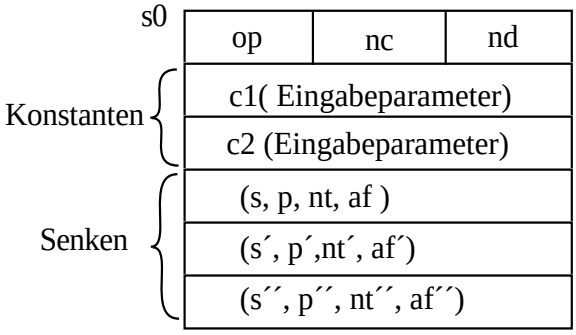
6.1.3. Beispiele für Datenflussrechner

Arvind (MIT)

Es ist eine dynamische Architektur, Daten und Token (tagged Token) werden transportiert.



Die Prozeduren in der Maschine haben einen Header der Form



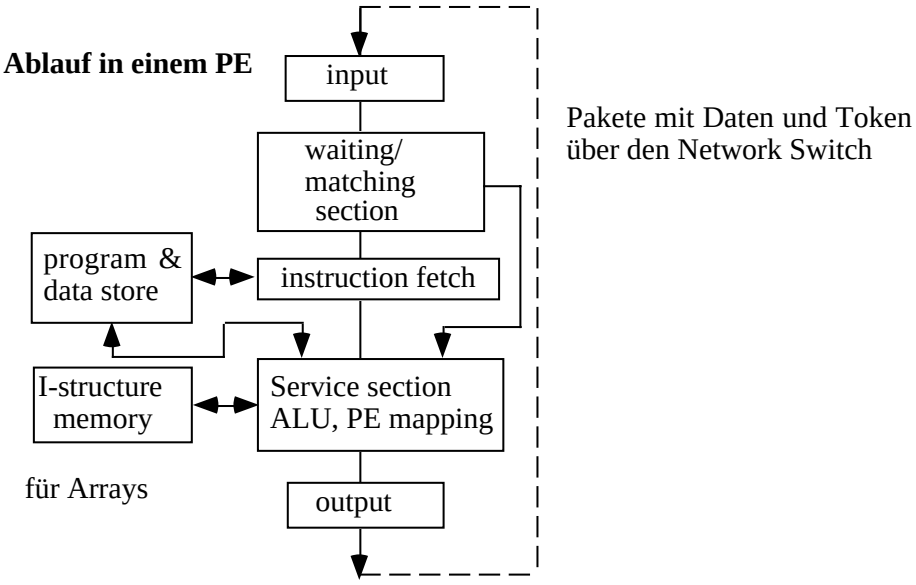
Instruktion bei Adresse s_0

op: opcode (procedure call)
nc: # Konstanten (=2); call by value
nd: # Senken (=3)

s: Senkenadresse
p: Eingangsport der Senke
nt: # Token zum Aktivieren
der Senkenadresse
af: Zuordnung zu einer Gruppe von PE´s

Wird eine Prozedur ausgeführt, dann stehen im Header die Senken, die die Daten verarbeiten sollen.

Die Prozessoren, auf denen die Senken liegen können - angegeben durch die Zuordnung af - können dann die Überprüfung auf die Vollständigkeit der Token machen.

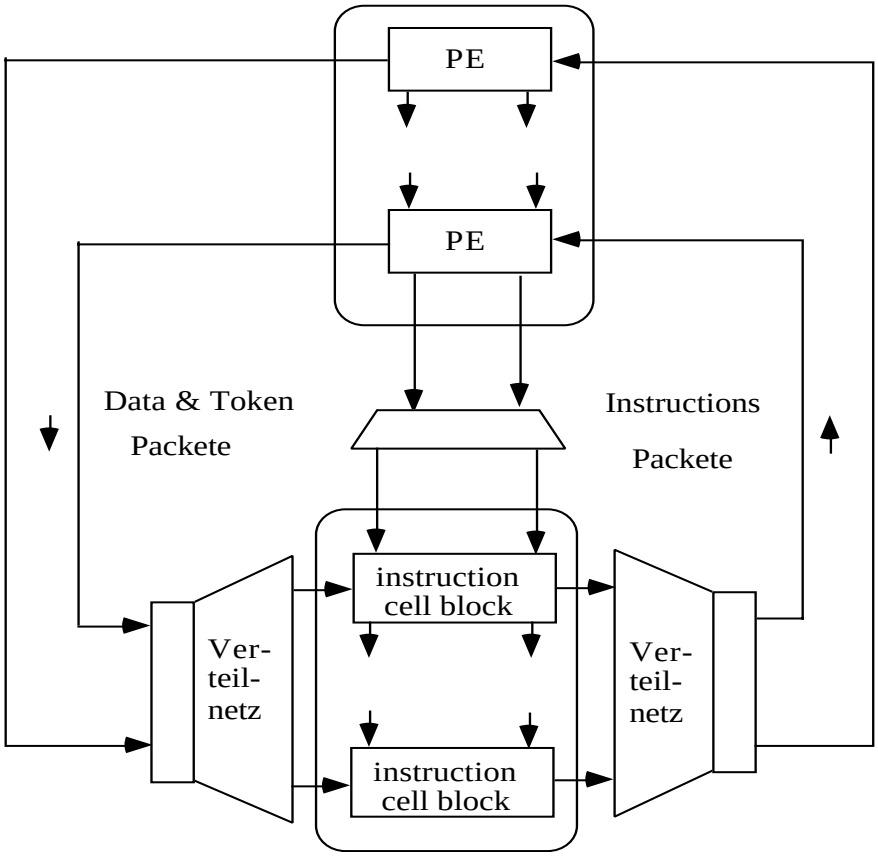


Für eine Prozedur ist jeweils eine Gruppe von PE's reserviert (physical domain).

Weitergereichte Token enthalten die Statementnummer mit.

Dennis-Maschine (MIT)

Dies ist eine statische Maschine.



Anhand der Daten-Token und der Steuer-Token, die freie Maschinen kennzeichnen, werden die rechenbereiten Instruktionen auf die Prozessoren verteilt.

Das Verteilungsnetz schreibt die Daten-Token in die relevanten Instruktionen und reicht sie zum Arbitration-Netz weiter.

Im Arbitration-Netzwerk werden die rechenbereiten Operationen und die freien Prozessoren in jeweils einer eigenen Warteschlange gehalten und zusammengeführt.

6.2. Prolog-Maschinen

6.2.1. Problemstellung

Die Sprache Prolog (**Pro**grammieren in **Log**ik) versucht Fragen der Form
" ? L_1, L_2, ..., L_n " mit L_i = Literale (= syntaktische Form von Fakten) und der Semantik "sind L_1 und L_2 und ... und L_n wahr ?" durch Rückführen der Literale L_i auf Fakten der Form "name (O_1, ..., O_n)" mit der Semantik "die Objekte O_1, O_2, ..., O_n stehen in der Bezeichnung "name" zueinander" zu beantworten.

Beispiel: verheiratet (Mann, Frau) ; verheiratet (heinz, emma)

Bei der Auflösung von Fragen greift Prolog auf Regeln der Form " L :- L_1, L_2, ..., L_n" zurück mit der Semantik: "wenn L_1 und L_2, und , ..., und L_n wahr sind, dann ist auch L wahr".

Beispiel: ist Vater (Vater, Kind):- ist Kind (Kind, Mutter), verheiratet (Vater, Mutter)
(Variable werden mit Großbuchstaben angegeben).

Die Sprachunterstützung durch Hardware geschieht an zwei Stellen:

Bei der Feststellung, ob Regeln feuern können, d. h. ob Literale L_1, ..., L_n wahr sind.

Sei $p_j \in \{\mathbb{L}\}$ und $\mathbb{L}$ die Menge aller Literale in dem Prolog-Programm. Dann wird eine Regel R_i genau dann feuern, wenn

$$R_i = \sum_k \left(\prod_j c_{jk}\, p_j \right) d_{ki}$$

Die Matrix c_{jk} mit $c_{jk} \in \mathbb{B}$ erfaßt alle Literale L_1, ..., L_n für die Regel L_k mit $c_{jk} = 1$;

Es mag darüber hinaus noch andere Kombinationen von Literalen L_j geben, die ebenfalls L feuern lassen. Sie werden durch den Index d_{ki} erfasst.

Die Auflösung einer Anfrage geschieht durch Anwenden von folgenden Operationen:

Gegeben sei die Anfrage (A_i sind Literale) .

? A_1, ..., A_{k-1}, A_k, A_{k+1}, ..., A_n

- Es sei A_k mit einem Faktum identifizierbar (unification) .
  Dann reduziert sich die Anfrage auf
  ? A_1, ..., A_{k-1}, A_{k+1}, ..., A_n
- Sei A_k unifizierbar mit der linken Seite einer Regel L :- L_1, ..., L_p
  dann erweitert sich die Anfrage auf
  ? A_1, ..., A_{k-1}, L_1, ..., L_p, A_{k+1}, ..., A_n
- Wird die Anfrage auf die leere Menge reduziert, ist die Behauptung wahr.

Durch das Einsetzen von Regeln kann sich der Suchbaum einer Anfrage aufblähen.

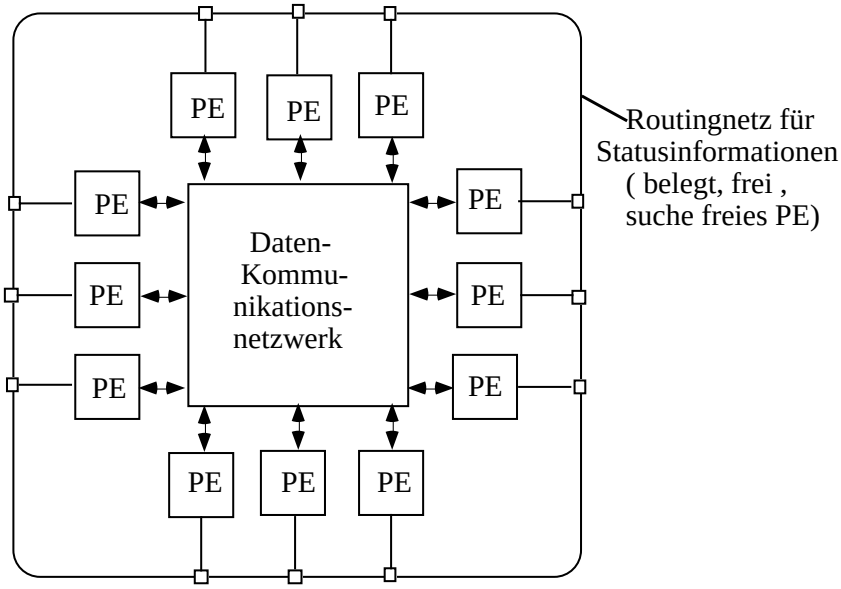
6.2.2. Hardwareunterstützung

Die Hardwareunterstützung besteht dann in einem EPLD (erasable programming logic device), das mit den Fakten L_i gespeist wird und die gefeuert habenden Regeln und die feuernden Regeln R_i beschreibt.

Die zweite Form der Hardwareunterstützung hilft beim Aufbau des Suchbaumes, der durch eine Anfrage aufgespannt wird.

Da ggf. ein Backtracking nötig wird, muss bei Verzweigungen das Environment gerettet werden und das geschieht durch Übergabe an ein freies PE.

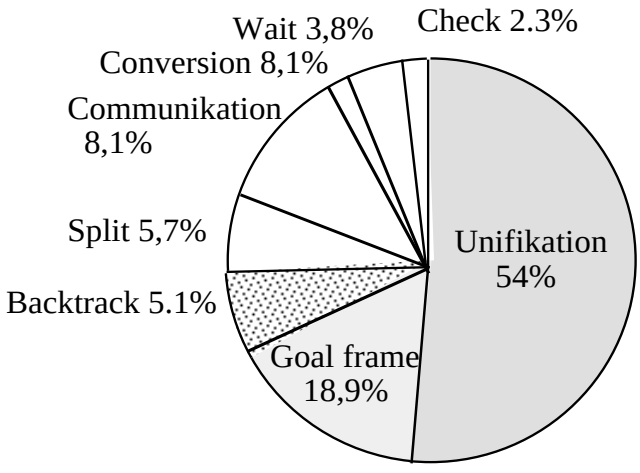
Das kann durch folgende Hardware geschehen:



Bei typischen Prolog-Systemen ergibt sich ein in etwa linearer Zusammenhang zwischen Geschwindigkeit und Anzahl der PE's.

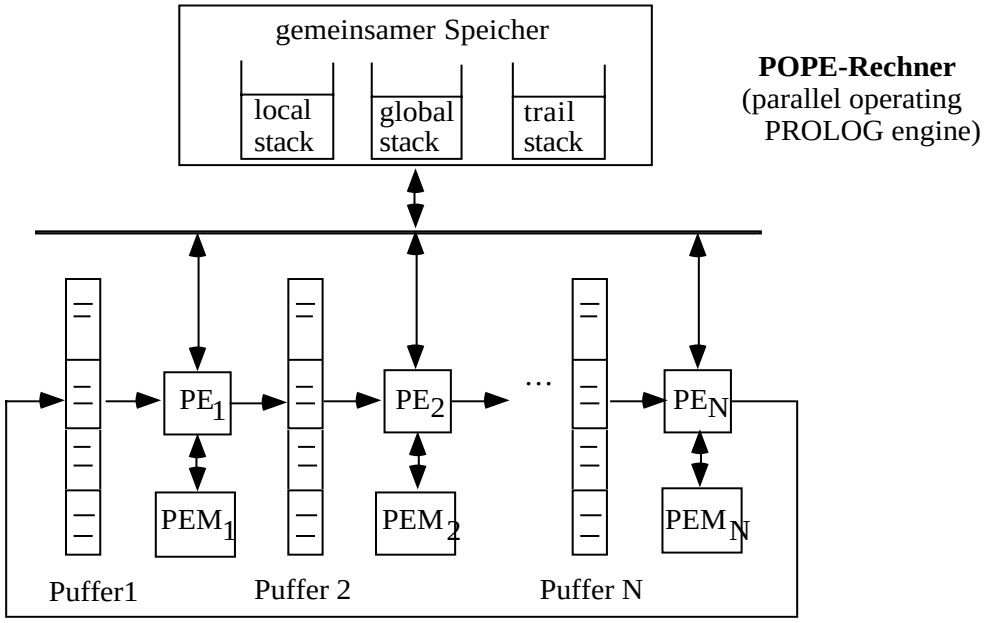
Ein typisches Beispiel ist ein 7-Damen-Problem gelöst mit 13 PE's.

Die Rechenzeit verteilt sich dabei wie folgt:



Beispiel:

Ein Prolog-Rechner, der gebaut wurde, ist die POPE-Maschine als Coprozessor an einer Unix-Workstation (**P**arallel **O**perating **P**rolog **E**ngine).



Pipeline von Unifikationsprocessoren

Im gemeinsamen Speicher steht

- ein globaler Stack für die Datenstrukturobjekte
- ein lokaler Stack für die Rücksetzpunkte und die Ausführungsumgebung definierenden Aktivierungsrahmen
- ein Sperr-Stack (trail stack) für die Notierung der Variablenbindungen, die beim Rücksetzen rückgängig gemacht werden müssen.

Die Pipeline-Puffer dienen dem unidirektionalen Datenaustausch. Es gibt jeweils 4 Pufferblöcke.

Ein Pufferblock enthält

- Einsprungadresse für die nächste Prozedur (goal)
- # Argumente und Steuerworte
- # letzter Rücksetzpunkt (previous choice point)
- TOS local (top- of-stack)
- TOS trail
- TOS global
- Adresse der Klausel, die beim Backtracking anzuspringen ist
- Fortsetzungspointer
- Argument 1
  .
  .
  .
- Argument n

} jeweils mit full/empty Liste

Das System nutzt die UND-Parallelität aus: wenn keine Datenabhängigkeiten bestehen, werden mehrere Zwischenziele (subgoals) parallel verfolgt. Datenabhängigkeiten treten auf durch gemeinsame Variable (shared variables).

Die Prozessor-Prozessor-Kommunikation und die Synchronisation wird durch Mikroprogramme in den PE's realisiert.

Synchronisation ist nötig bei

- Lesen/Schreiben in einen Stack
- Binden einer noch nicht gebundenen Variablen
- Zugriff auf ein existierendes, vollständig instantiiertes Datenobjekt (wegen single assignment - Forderung)

Die Zugriffsberechtigung auf einen Stack geschieht durch Übergabe des Stackpointers im Puffer.

Variable werden gekennzeichnet (tagging).

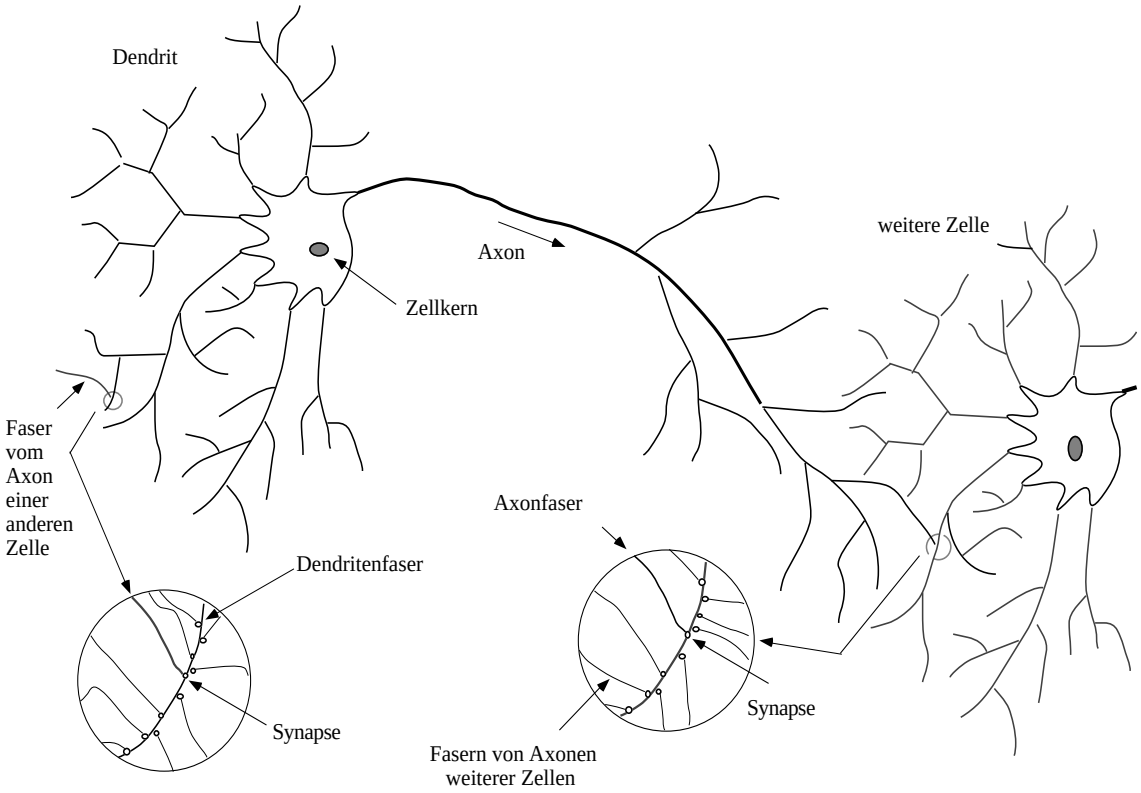
6.3. Neuronale Netze

6.3.1. Natürliche neuronale Netze

Biologische Systeme sind gekennzeichnet durch massive Parallelität: einige 10^6 einfache Sensorelemente arbeiten parallel (Sehzellen in der Retina, Tastzellen in der Haut, Hörzellen im Ohr, Chemosensoren in der Nase). Diese Parallelarbeit setzt sich fort in der Cortex.

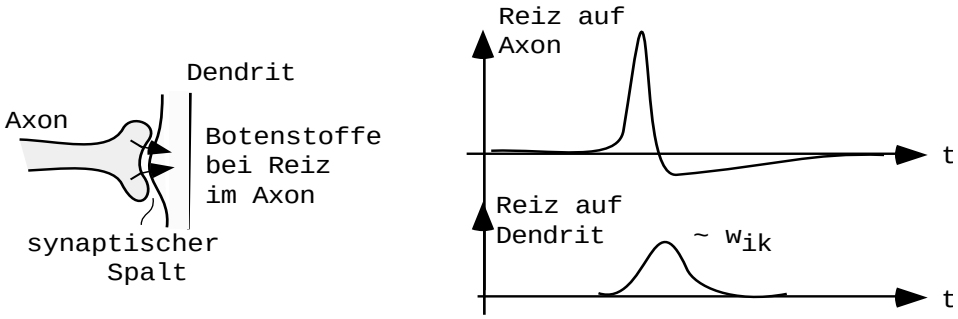
Die Datenverarbeitung geschieht in Netzen aus Neuronen. Ein Neuron besteht aus einer Zelle mit einem stark verzweigten Baum von Eingangsfasern **(Dendrit)** und einem ebenso verzweigten Baum von Ausgängen **(Axon)**. Im Gehirn eines Säugetiers haben Axone eine Reichweite von wenigen cm; vom Rückenmark zu den Extremitäten Längen von bis zu 3 m (Elefant). Die Verbindung zu benachbarten Zellen geschieht von einer Faser des Axons über eine **Synapse** zum Dendriten der anderen Zelle. Man beobachtet keine interne Rückkopplung: die Axonfasern einer Zelle haben keine synaptische Verbindung zum Dendriten der gleichen Zelle. Man findet eine schwache Kopplung der Neuronen untereinander: von einem Neuron aus gibt es zu einem anderen meist nur eine einzige synaptische Verbindung. Es gilt ein Lokalitätsprinzip mit sehr geringer Rückkopplung. Daneben gibt es Weitverbindungen (Sehnerv, Brücke zwischen den Hirnhälften) und es sind Rückkopplungen vorhanden über größere Verbände. Sie erlauben kurzfristig kreisende Erregungsmuster (Assoziationskreise, Gedanken?).

Eine Zelle ist über Synapsen mit 10^3 - 10^4 anderen Zellen verbunden. Die Organisation im Großen ist in Schichten. Die Verbindungen von Axonen einer Schicht laufen wesentlich zu Dendriten anderer Schichten. Innerhalb einer Schicht ist die Vernetzung gering.

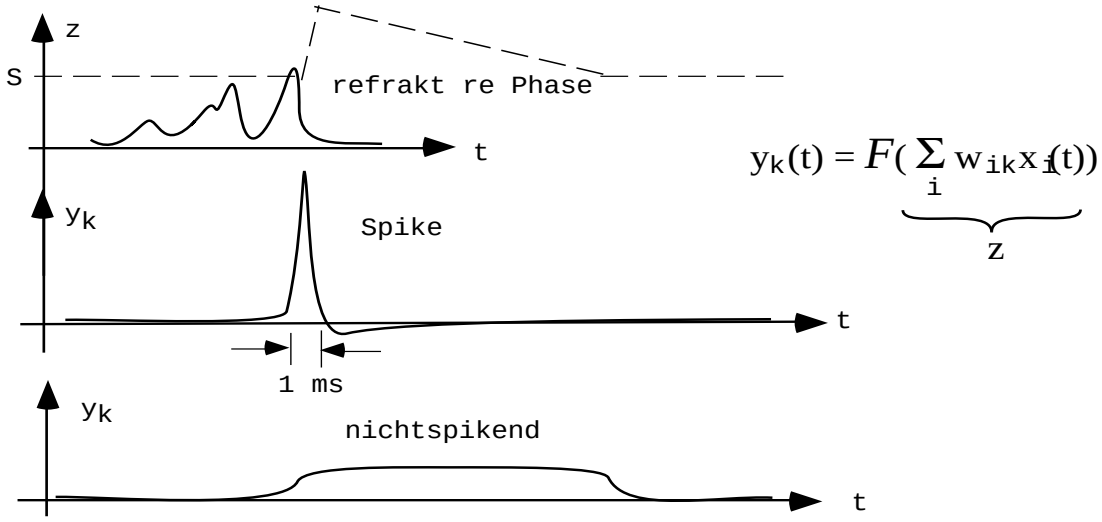


Die Weiterleitung von Erregungen erfolgt über Synapsen. Dort wird das elektrochemische Signal auf einem Axon gewandelt in Botenstoffe, die über den synaptischen Spalt in den Dendriten gelangen und dort ggf. die Zelle zum Feuern veranlassen (exitatorische Erregung), oder das Feuern behindern (inhibitorische Erregung). Wenn die Summe der Erregungen einen Schwellwert überschreitet, feuert die Zelle.

Die Wirkungsdauer einer synaptischen Erregung ist kurz (~ ms). Eine Zelle reagiert somit auf Koinzidenzen von Erregungen.Der Einfluss einer Synapse auf den zugehörenden Dendriten wird durch ein **synaptisches Gewicht** w bestimmt.



Beim Feuern kann entweder ein Spike erzeugt werden oder ein Aktivierungspotential angehoben werden (nicht spikendes Neuron).



Der Schwellwert eines Neurons ist abhängig von der chemischen Umgebung.

Lernen erfolgt durch Änderung der synaptischen Gewichte beim Feuern von Zellen. Der Einfluss einer Synapse, die einen Spike ausgelöst hat, wird verstärkt, der aller anderen Synapsen abgeschwächt.

Die Rinde der menschlichen Cortex ist ein Gebilde von ca. 1000 cm^2, ca. 2 mm dick, stark gefaltet (graue Hirnsubstanz) mit Verbindungen innerhalb der 6 - 8 Schichten in der Dicke und Bündeln weitreichender Verbindungen zwischen den einzelnen Arealen (weiße Substanz). Die Cortex enthält ca. 10^{10} Neuronen mit ca. 10^{14} Synapsen, d. h. 10^4 Synapsen/Neuron.

Auf der Rinde der Cortex existiert eine topologische Abbildung der Sensoren (Retina, Haut, (Nase), (Ohr)).

Von der großen Zahl einkommender sensorischer Erregungen werden die allermeisten in den ersten Schichten der cortikalen Weiterverarbeitung inhibitorisch unterdrückt. Nur wenige "schlagen durch" und veranlassen Ensembles zum Feuern. Das sind dann wahrgenommene äußere Ereignisse.

6.3.2. Nachbildung neuronaler Netze

Die Nachbildung neuronaler Netze bildet den Feuerungsmechanismus eines Neurons nach: Ein neuronales Netz wird zu diskreten Zeitpunkten n betrachtet. Seien $x_i(n)$ die Eingänge des Neurons k zum Zeitpunkt n und w_{ik} der Einfluss der Aktivierung x_i auf Neuron k (Synaptisches Gewicht). Die Produkte $w_{ik} \cdot x_i$ werden über alle i sumiert und einer nichtlinearen Schwellwertfunktion F zugeleitet, die $y_k^{(n)}$ setzt.

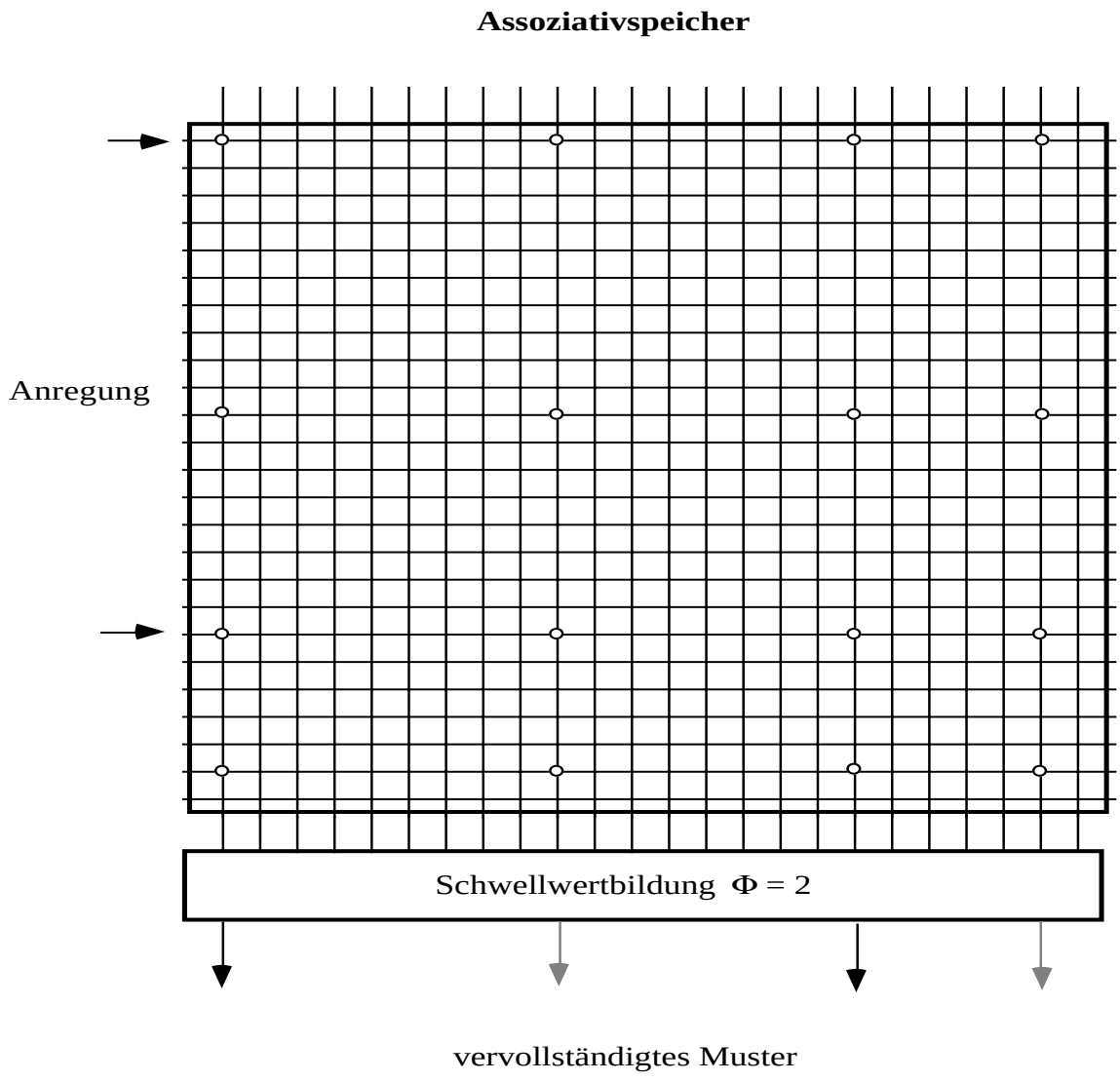
$$y_k^{(n)} = F(\sum_i w_{ik}\, x_i^{(n)})$$

Die synaptischen Gewichte sind dabei entweder $w_{ik} \in \mathbb{B}$ oder $w_{ik} \in \{0, 1\}$ oder $w_{ik} \in \{-1, ... 0, ..., +1\}$.

Beschränkt man sich auf Gewichte $w_{ik} \in \mathbb{B}$ gelangt man zu Assoziativspeichern.

6.3.2.1. Assoziativspeicher

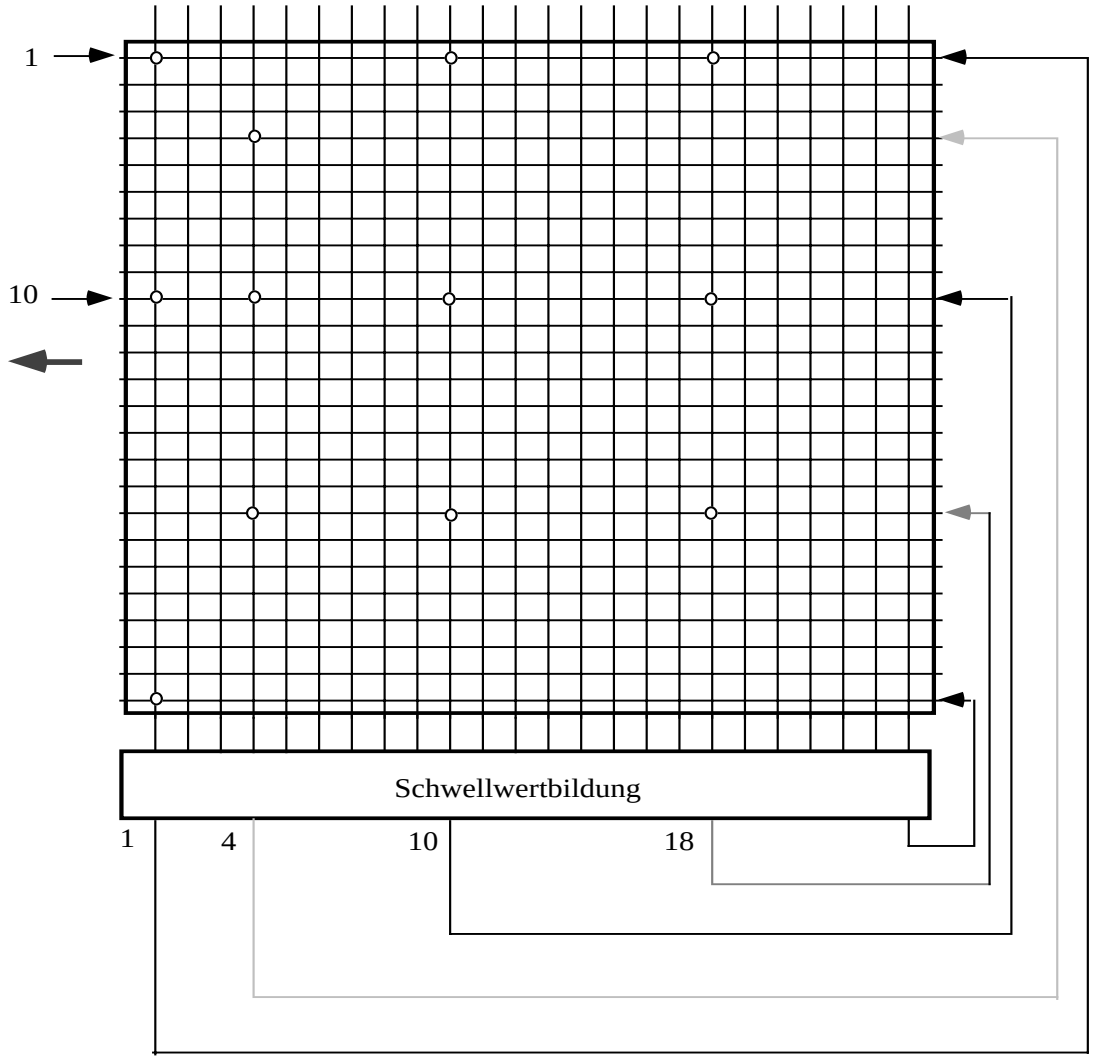
Eine schwach besetzte Matrix aus binären Punkten wird zeilenweise angesprochen und spaltenweise mit einer integrierten Schwellwertbildung ausgelesen.



Es kann ein unvollständig eingegebenes Muster vervollständigt werden.

Auf eine Anregung hin kann sich durch Rückkopplung ein stabiles Muster einstellen.

Assoziativspeicher mit Rückkopplung



(1, 10) ==> (1, 10, 18) ==> (1, 4, 10, 18) ==> (1, 4, 10, 18)
stabil

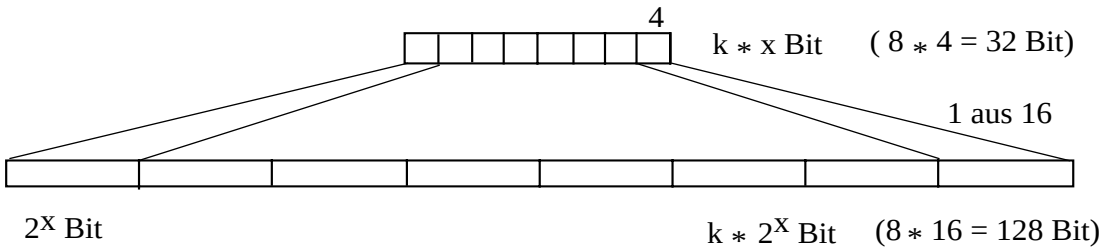
Die Matrix soll schwach besetzt sein, um viele Muster eindeutig speichern zu können.

Z. B. kann eine 1000 x 1000 - Matrix ca. 8000 Muster speichern.

Die Codierung sollte mit spärlichen Einsen erfolgen (sparse coding).

Z. B. wird eine Folge von N Bit, die eine Zahl $0 \le z \le 2^N - 1$ repräsentiert, unterteilt in k Abschnitte von jeweils x Bit. Jeder Abschnitt x stellt eine Zahl dar zwischen $0 \le z_k \le 2^x - 1$. Er wird codiert als eine Folge von 2^x Bit mit $b_j = \delta_{jzk}$: nur an der Stelle z_k ist eine Eins, sonst Null.

Damit wird die Darstellung von N Bit aufgeweitet auf $k * 2^x$ Bit.



Beispiel: 32 Bit in 4 Gruppen à 8 Bit
aufgelöst in 4 Gruppen zu 256 Bit = 1024 Bit

oder 8 Gruppen à 4 Bit
aufgelöst in 8 Gruppen zu 16 Bit = 128 Bit.

6.3.2.2. Technische Realisierung von neuronalen Netzen

Der Vergleich zwischen Biologie und Technik zeigt die durch die technischen Möglichkeiten gezogenen Grenzen.

	biolog.neuronale Netze	technische neuronale Netze
Material:	Proteine	Silizium
Struktur:	3 D, ~ 1000 cm^3	planar, ~ 1 cm^2
Strukturbreite:	≈ 0.1 µm	~ 0.2 µm ↓ (Tendenz fallend)
Geschwindigkeit:	~ 1 ms	≈ 100 ns ↓ (für $y_k := y_k + w_{ik} x_i$)
Anzahl:	10^7 - 10^{10} Neuronen	≈ 5 Mio. Transistoren/Chip ↑
Netz:	P(PE) ~ 10^3	einfache Verbindungsstruktur

Konsequenz:
Tradeoff für die Berechnungsdauer der Basisfunktion

$$y_k = F(\sum_{i=1}^{N} w_{ik} x_i) \forall k = 1,\ldots,N$$

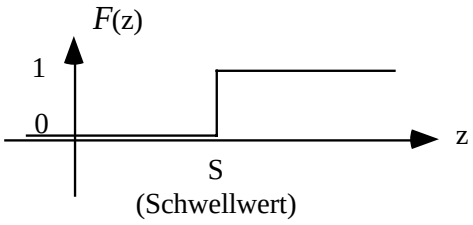
zwischen biologischem System mit P(PE) >> 1 langsam und hochparallel mit O(1)
und technischem System mit P(PE) < 20 und schnell aber seriell mit $O(N^2)$

Die Anwendung paralleler Strukturen versucht die Zeitkomplexität zu drücken von $O(N^2) \rightarrow O(N)$.

Die Basisfunktion ist die Anwendung einer nichtlinearen Schwellwertfunktion *F* auf die gewichtete Summe der Eingänge und das Fortschreiben der Gewichte nach einer Lernregel.

Aktivierung des Neurons k zum Zeitpunkt n

$$y_k^{(n)} = F(\underbrace{\sum_i w_{ik}\, x_i^{(n)}}_{z})$$

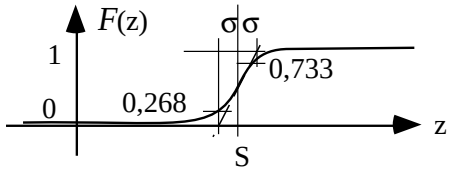


Lernregel (Hebb´sche Regel)

$$|w_{ik}| := |w_{ik}| + (\alpha\,(y_k * x_i) - \beta(\bar{y}_k * x_i)(1-|w_{ik}|)$$

lernen vergessen

$$\beta << \alpha << 1 \; ; \; -1 \le w_{ik} \le 1 \; ; \; x_i, y_k \in \mathbb{B}$$



$$F(z) = \frac{1}{1 + e^{(S-z)/\sigma}} \qquad x_i, y_k \in \{0,1\}$$

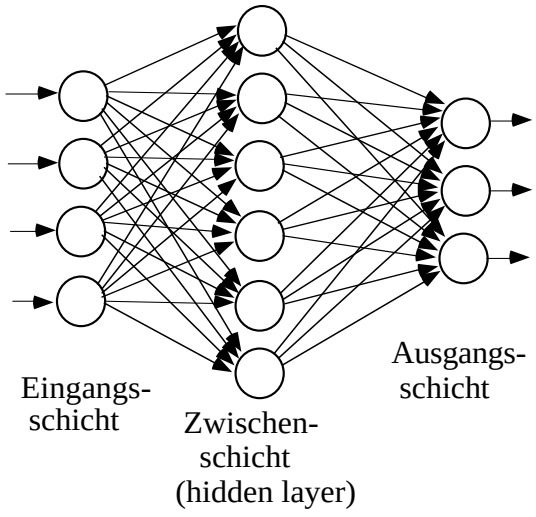
sigmoide Funktion

Das Gewicht w_{ik} beschreibt das Gewicht auf der Verbindung von Neuron i $\rightarrow$ Neuron k.

$w_{ik} \neq 0 \Rightarrow w_{ki} = 0$, da es keine direkte Rückkopplung gibt.

Die Gewichte sind Größen aus dem Intervall [-1, +1] ; für inhibitorische Synapsen $w_{ik} < 0$, oder exitatorische Synapsen $w_{ik} > 0$.

Das neuronale Netz ist typischerweise ein dreistufiges Netz (feed-forward-net)



mit einer verborgenen Zwischenschicht. Dieser Netztyp kann eine Funktion **y** = f (**x**) realisieren mit einem Eingangsvektor **x** und einem Ausgangsvektor **y**.

6.4. Hardwarerealisierungen

6.4.1. Paralleles Multiprozessorsystem

Unterstützung der Berechnung der Summe

$$\forall_k : \text{falls} \left(\sum_{i \neq k}^{N} w_{ik} \cdot x_i \right) > S \quad \text{dann} \quad x_k := 1$$

durch Rechner an einem Bus.

Damit läßt sich die Berechnung in N Schritten erledigen, wenn N PE's vorhanden sind.

Im Zeitschritt i legt PE_i seinen x-Wert auf den Bus. Alle PE_k mit $k \neq i$ lesen den Wert und summieren: $y_k := y_k + w_{ik} \cdot x_i$

Nach N Schritten hat jedes PE_k $\sum_{i \neq k}^{N} w_{ik} \cdot x_i$ gebildet und macht dem Vergleich:

$$\text{falls} \left(\sum_{i \neq k}^{N} w_{ik} \cdot x_i \right) > S \quad \text{dann} \quad x_k := 1 \text{ , sonst } x_k := 0 \text{ .}$$

Mit diesem Schritt sind alle PE's synchronisiert und eine weitere Runde der Berechnung kann beginnen.

Ggf. werden noch die Gewichte angepaßt:
Dazu wird das im Schritt i eingelesene x_i zwischengespeichert und dann gebildet für alle $i \neq k$.

$$|w_{ik}| := |w_{ik}| + (\alpha(x_i \cdot x_k) + \beta(\neg x_i \cdot x_k))(1 - |w_{ik}|)$$

Werden die Gewichte durch Backpropagation angepaßt, dann sind die x_i Werte $0 \leq x_i \leq 1$ und die Berechnungsfunktion ist

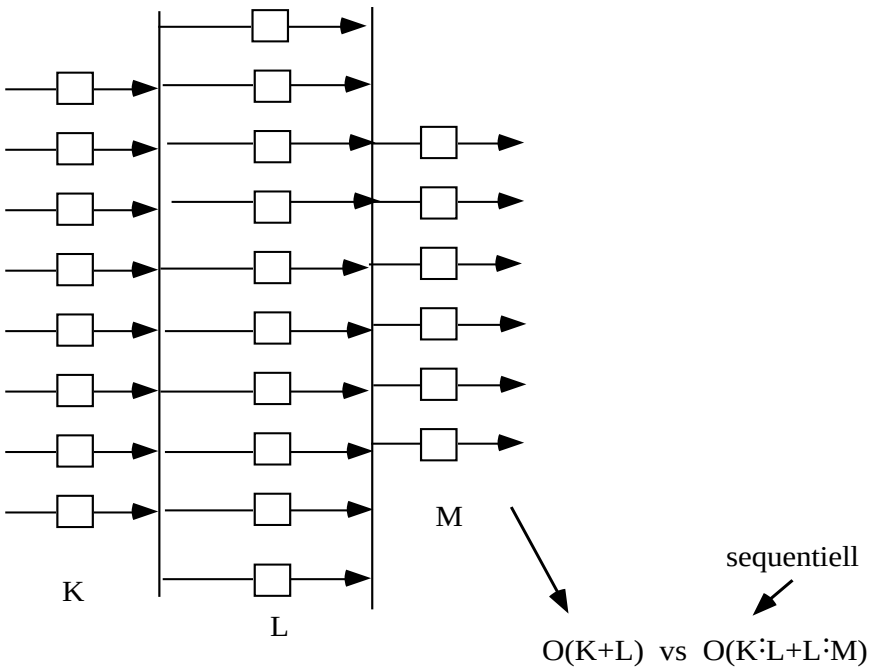
$$\forall_k : x_k := F \left(\sum_{i \neq k}^{N} w_{ik} \cdot x_i \right) \text{ mit einer sigmoiden Funktion}$$

$$F(z) = \frac{1}{1 + e^{\frac{(s-z)}{0}}}$$

Der Speicheraufwand bei jedem PE ist N Gewichte w_{ik} mit $-1 \leq w_{ik} \leq +1$.

Für ein dreischichtiges neuronales Netz mit K Eingangsneuronen, L Neuronen in der Zwischenschicht und M Ausgangsneuronen ist ein System von K + L + M PE's an einem Bus nötig:

dreischichtiges Neuronales Netz



Der Algorithmus sieht dann so aus:

```
                    || for all L PE´s  do

for i = 1 to K      for i = 1 to K
begin               begin
output x_i  <-->    read x_i
NOP                 s_k := s_k + w_ik.x_i      } K Schritte
end                 end

                    x_k := F(s_k)
                    for all i = 1 to K
                    begin
                    |w_ik| =: |w_ik|+(α·x_k·x_i+ β(¬x_i.x_k)(1-w_ik)   } K Schritte
                    end
                    s_k := 0


                                || for all M PE´s do

for i = 1 to L      for i = 1 to L
begin               begin
output x_i  <-->    read x_i
NOP                 s_k := s_k + w_ik·x_i      } L Schritte
end                 end
                    x_k := F(s_k)
                    for all i=1 to L
                    begin
L Schritte  {       |w_ik| =: |w_ik|+(α·x_k·x_i+ β(¬x_i.x_k)(1-w_ik)
                    end
                    s_k := 0
```

6.4.2. Analoge Systeme

Wenn die Struktur des Netzes bekannt ist und die Gewichte nicht verändert werden müssen und die Ausgänge nur 1 Bit sind, ist die Rechnung mit analoger Hardware einfach machbar.

Die Gewichte werden addiert und mit dem Schwellwert verglichen. Die Addition und der Vergleich kann rasch erfolgen (~ 100 ns), allerdings nur mit begrenzter Genauigkeit (4 ‰) durch maximal 8 Bit.

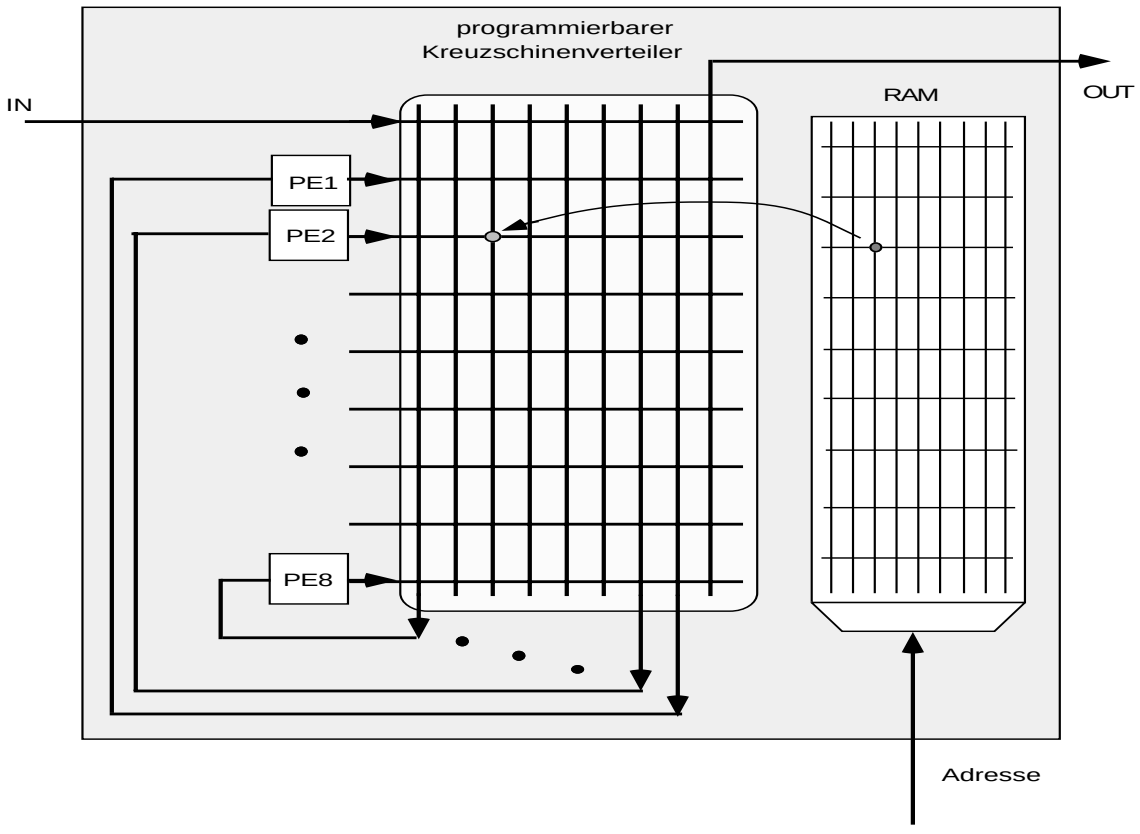
Beispiel: Neurochip von AT & T

- 32.000 1-Bit Verbindungen: der Output eines Neuron ist Input eines anderen
- analoge Summation der Gewichte
- Genauigkeit 4 ‰
- 1-Bit Verbindungen mit Schieberegister als Zwischenspeicher
- alle 100 ns parallele Evaluation
- $3 \cdot 10^{11}$ Übertragungen/s.

6.4.3. Digitale Neurochips

6.4.3.1. XILINX LCA (logic cell array)

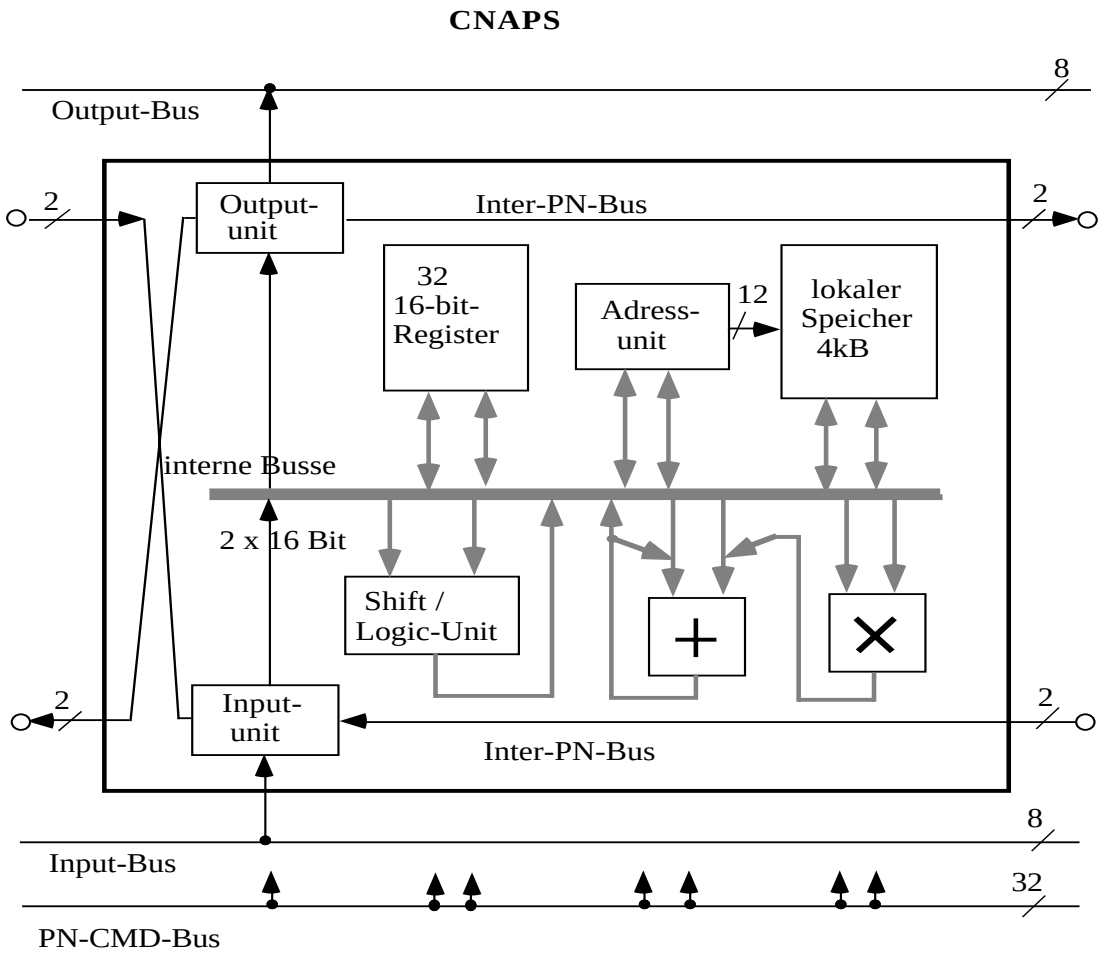
Es werden mehrere Neuronen als PE's auf einem Chip mit einem Kreuzschienenverteiler und einem RAM für die Steuerung des KSV integriert.



Mit mehreren derartigen Chips und einer globalen Steuerung kann dann ein neuronales Netz realisiert werden (n-MANN von Texas Instruments).

6.4.3.2. CNAPS

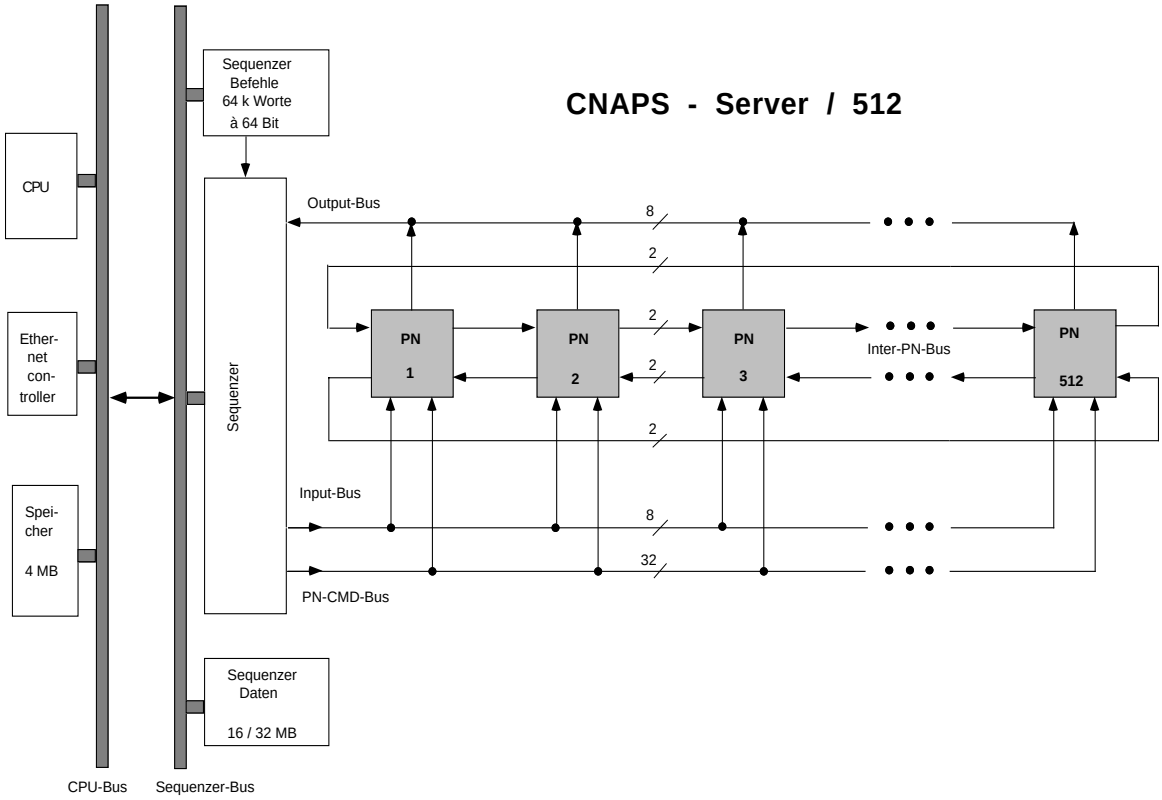
Das neuronale Netz besteht aus einfachen Zellen die über zwei lineare Ketten und zwei 8–Bit Parallelbusse zusammengeschaltet sind.



Da alle Prozessoren das Gleiche tun, erhalten sie ihre Befehle extern (SIMD).

Intern haben sie einen 4 kB-Speicher für die Gewichte, 32 16-Bit-Register, einen Addierer, Multiplizierer und eine Logik-Schiebe-Einheit.

Auf einem Chip sind bis zu 64 derartige Prozessoren integriert. 16 Chips bilden den Kern eines CNAPS-Servers.



Die linearen Ketten sind dann zum Ring geschlossen.

6.4.4. Optische Realisierungen

Mit optischen Mitteln versucht man, neuronale Netze zu realisieren, die die Rechnungen in O(1) durchführen (siehe Kapitel 7).