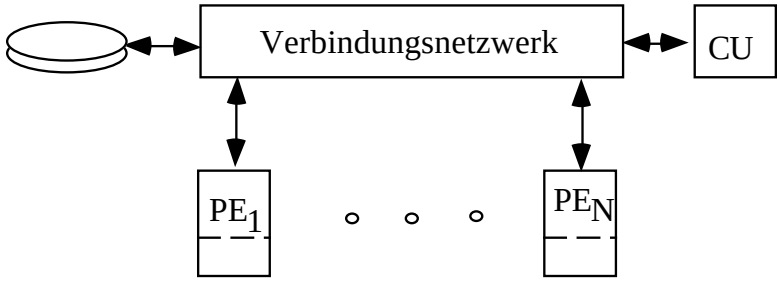
# Kapitel 5

# Feldrechner

## 5.1. Klassifikationen

Feldrechner bestehen aus einer Anzahl N von gleichartigen rechnenden Einheiten (processing elements, PE), die untereinander durch ein Verbindungsnetzwerk (VNW) Daten austauschen können und durch eine gemeinsame Steuerung CU zusammengebunden sind.




Die rechnenden Elemente können sehr unterschiedlich sein.

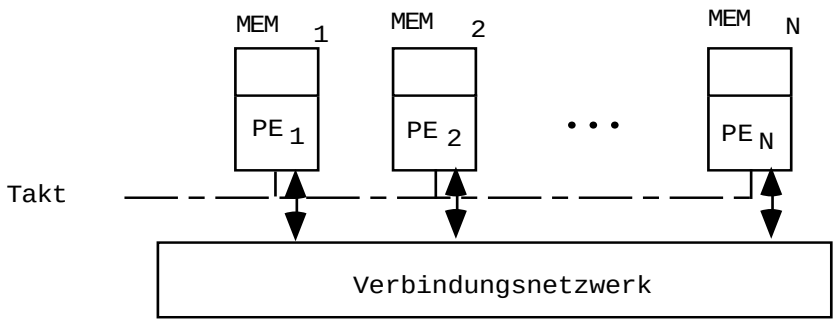
### 5.1.1. Zellulare Automaten

Die rechnenden Einheiten sind einfache Speicherautomaten. Sie werden mit einem gemeinsamen Takt angesteuert und bilden folgende Funktionen: das Element $PE_i$ hat den Zustand $z_i^{(n)}$ zum Zeitpunkt n und produziert den Folgezustand $z_i^{(n+1)}$.

$$z_i^{(n+1)} = \delta (x_{i-k}^{(n)}, \ldots x_{i-1}^{(n)} ; z_i^{(n)} ; x_{i+1}^{(n)}, \ldots, x_{i+r}^{(n)})$$

Der Ausgang $x_i^{(n)}$ ist eine Funktion des Zustands allein:

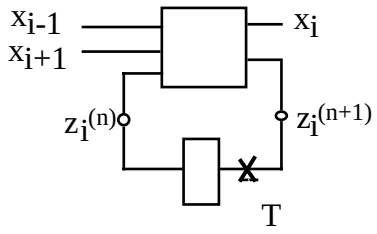
$x_i^{(n)} = \omega (z_i^{(n)})$; die Ausgänge der Nachbarn sind die Eingänge von $PE_i$



### *5.1.1.1. Linearer Automat, 1 Bit*

Es sind nur die Werte $x_{i-1}$ und $x_{i+1}$ der Nachbarn von Interesse.

Im einfachsten Fall sind die Werte $x_i$ 1 Bit.



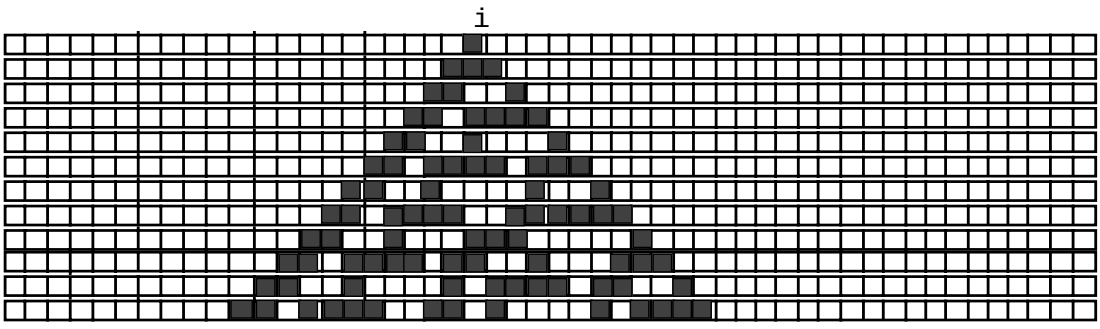

$$z_i^{(n+1)} = \delta(x_{i-1}, x_{i+1}, z_i^{(n)})$$

$$x_i = z_i^{(n)}$$

Dann hat der Automat nur zwei Zustände und wird durch eine Tafel mit 8 Einträgen beschrieben.


| $x_{i-1}$ | $x_i$ | $x_{i+1}$ | $z_i^{(n+1)}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

Diese Automatentafel produziert ein komplexes Pattern, wenn man mit einem Anfangszustand $x_j = 0 \quad \forall_j \neq i$ und $x_i = 1$ beginnt




und die Entwicklung der Automaten zeilenweise darstellt: pro Generation eine Zeile.

### *5.1.1.2. Linearer Automat mit 2-Bit-Zustand*

Sei $x_i \in \{0, 1, 2, 3\}$ und $x_i^{(n)} = z_i^{(n)}$ ( Ausgang = Zustand)

Dann ist die Zustandstafel des Automaten

| $x_{i-1}$ | $x_i$ | $x_{i+1}$ | $z_i^{(n+1)}$ |
|---|---|---|---|
| 2 Bit | 2 Bit | 2 Bit | 2 Bit |
| | | | 64 Einträge à 2 Bit |

Sei $x_i = c$ und $x_j = 0 \ \forall_j \neq i$ .

Die Entwicklung des Automaten ist nicht vorhersehbar.

Sei $x_i \in \{0, 1, 2\}$ und vereinfachend angenommen, daß bei der Berechnung des Folgezustand $z_i^{(n+1)} = x_i^{(n+1)}$ ( und des Ausgangs) nur die Summe eingeht

$$\sum_{j = i-1}^{i+1} x_j$$

Dann wäre eine Kurzbeschreibung die Codierung

$\Sigma$ : 6 5 4 3 2 1 0

$x_i^{(n+1)}$ : 1 2 0 1 1 2 0 $\quad x_i \in \{0, 1, 2\}$

Die Zustandstabelle hätte die Form

| $x_{i-1}$ | $x_i$ | $x_{i+1}$ | $\Sigma$ | $x_i^{(n+1)}$ |
|---|---|---|---|---|
| 2 | 2 | 2 | 6 | 1 |
| 2 | 2 | 1 | 5 | 2 |
| 2 | 1 | 2 | | |
| 1 | 2 | 2 | | |
| 2 | 2 | 0 | 4 | 0 |
| 2 | 1 | 1 | | |
| 2 | 0 | 2 | | |
| 0 | 2 | 2 | | |
| 1 | 1 | 2 | | |
| 1 | 2 | 1 | | |
| 1 | 1 | 1 | 3 | 1 |
| 2 | 1 | 0 | | |
| 2 | 0 | 1 | | |
| 1 | 2 | 0 | | |
| 1 | 0 | 2 | | |
| 0 | 1 | 2 | | |
| 0 | 2 | 1 | | |
| 1 | 1 | 0 | 2 | 1 |
| 2 | 0 | 0 | | |
| 1 | 0 | 1 | | |
| 0 | 1 | 1 | | |
| 0 | 2 | 0 | | |
| 0 | 0 | 2 | | |
| 1 | 0 | 0 | 1 | 2 |
| 0 | 1 | 0 | | |
| 0 | 0 | 1 | | |
| 0 | 0 | 0 | 0 | 0 |

Beispiel

```
0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 2 1 1 1 2 0 0 0 0 0 0
0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0
0 0 0 2 1 1 1 2 1 1 1 2 0 0 0 0
0 0 1 1 0 1 0 0 0 1 0 1 1 0 0 0
0 2 1 1 1 2 2 0 2 2 1 1 1 2 0 0
1 1 0 1 0 2 0 0 0 2 0 1 0 1 1 1
```

Sei $x_i \in \{0, 1, 2, 3\}$ und ein Automat habe die Codierung

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\sum$ | : | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| $x_i^{(n+1)}$ | : | 3 | 3 | 1 | 1 | 1 | 0 | 0 | 3 | 2 | 0 |

$x_i \in \{0, 1, 2, 3\}$

Dann zeigt das nächste Bild das Verhalten mit den Anfangsbelegungen

0 ... 0 1 0 ... 0 | ... 0 2 3 0 ... | ... 0 1 1 0 ... |

Im ersten Fall geht nach 16 Schritten der Automat in den Zustand 0 , ... , 0 , ... 0 über, im zweiten dauert es 1016 Schritte und für die dritte Anfangsbelegung ist ein Ende nicht absehbar. Der Übergang in den Zustand 0,…, 0 entspricht dem "Halt" eines Programms und ist nicht entscheidbar - äquivalent dem Halteproblem beim Rechner.

### *5.1.1.3. Zweidimensionaler zellularer Automat*

Betrachtet werde ein Automat mit nur zwei Zuständen: $z^{(n+1)} \in \{0, 1\}$ . Sei die betrachtete Nachbarschaft ein quadratisches Gitter und die 8 Nachbarn von Interesse.

$x_{i,k}^{(n+1)} = z_{i,k}^{(n)}$ und

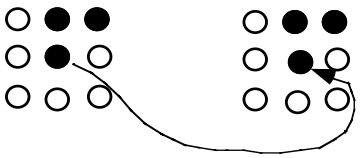
$z_{i,k}^{(n+1)} = f( x_{i,k+1} , x_{i,k-1} , x_{i+1,k} , x_{i-1,k} , x_{i+1,k+1} , x_{i-1,k+1} , x_{i+1,k-1} , x_{i-1,k-1} , z_{i,k}^{(n)} )$

Wieder werden die Nachbarn pauschal behandelt: nur ihre Summe interessiert.

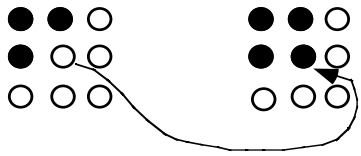
Besonders gut ist untersucht worden das **Game of Life** (John Horton Conway, 1968)

Es gibt nur zwei Regeln:

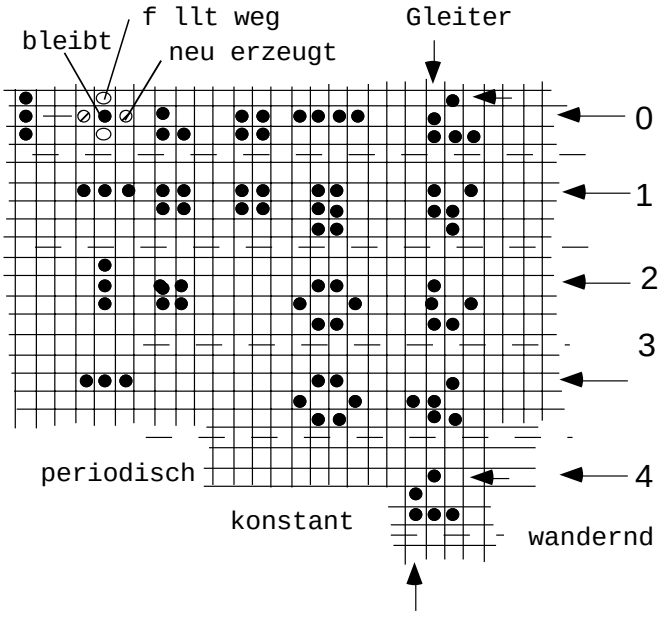
1. Sei $z_{i,k}^{(n)} = 1$
   $\exists$ 2 oder 3 Nachbarn mit $x_{i,k} = 1$
   $\Rightarrow z_{i,k}^{(n+1)} = 1 \Rightarrow x_{i,k} := 1$
   sonst $z_{i,k}^{(n+1)} = 0 \Rightarrow x_{i,k} := 0$



2. Sei $z_{i,k}^{(n)} = 0$
   $\exists$ genau drei Nachbarn mit $x_{i,k} = 1$
   $\Rightarrow z_{i,k}^{(n+1)} = 1 \Rightarrow x_{i,k} := 1$
   sonst $z_{i,k}^{(n+1)} = 0 \Rightarrow x_{i,k} := 0$



Das folgende Bild zeigt einige einfache Anfangspattern und ihre Entwicklung.



Interessant ist eine Kombination aus 5 Anfangswerten $z_{i,k} = 1$ , ein Gleiter (glider): nach 4 Schritten hat sich das Pattern repliziert und sich einen Schritt in der Diagonalen fortbewegt.

Man hat Gebilde gefunden, die nach einer Reihe von Schritten sich replizieren, aber dazu einen Gleiter produzieren (Gleiterkanone).

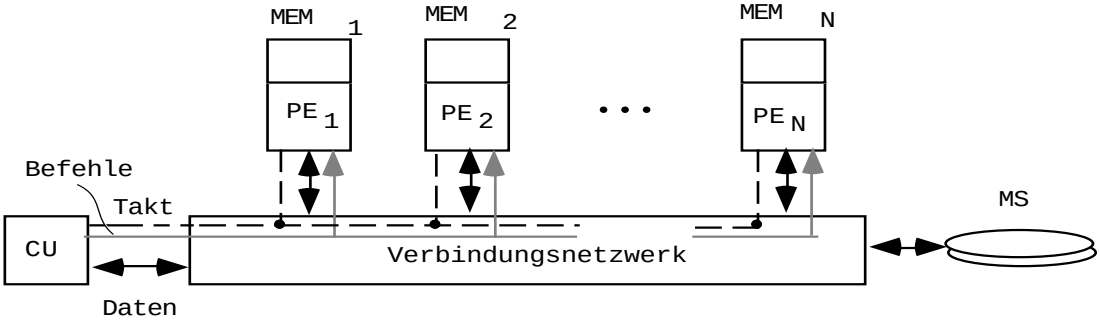
Andere Pattern zerstören einen Gleiter (Gleiterfresser). Damit läßt sich ein Strom von Gleitern (entspricht 1) und fehlenden Gleitern (entspricht 0) erzeugen.

Andere Gebilde verknüpfen zwei Gleiterströme zu einem resultierenden nach Art von UND oder ODER oder invertieren einen Gleiterstrom.

Damit sind alle Elemente zum Aufbau eines Rechners vorhanden, der mit Gleiterströmen rechnet. D. h. "Game of Life" hat die Mächtigkeit der Turing-Maschine.

### 5.1.2. SIMD-Rechner (single instruction, multiple data)

Die rechnenden Elemente sind Prozessoren mit Speicher an Bord, die mit einem gemeinsamen Takt und einem gemeinsamen Befehlsstrom versorgt werden.




Die rechnenden Einheiten können untereinander Daten austauschen.

Das ganze System hat genau einen Befehlszeiger, der auf einen Befehl in der Steuereinheit (control unit, CU) weist, der gemeinsam an alle PE's gegeben wird.

Wenn der an alle PE's gemeinsam verteilte Befehlsstrom datenabhängige Verzweigungen enthält, dann dürfen einzelne Programmzweige in einigen PE's nicht gerechnet werden.

Zur Steuerung des Befehlsflusses bedarf es spezieller globaler Kommandos und einem Array von Flags in jedem PE .

Sei durch eine reservierte Variable i : integer die Schachtelungstiefe in einem PE festgehalten und durch ein Array f(i) : boolean die Ausführungserlaubnis in dem PE gekennzeichnet, dann soll die Ausführungserlaubnis nur dann geändert werden, wenn auf der unmittelbar darunterliegenden Ebene die Ausführungserlaubnis bestand.

Globale Befehle sind

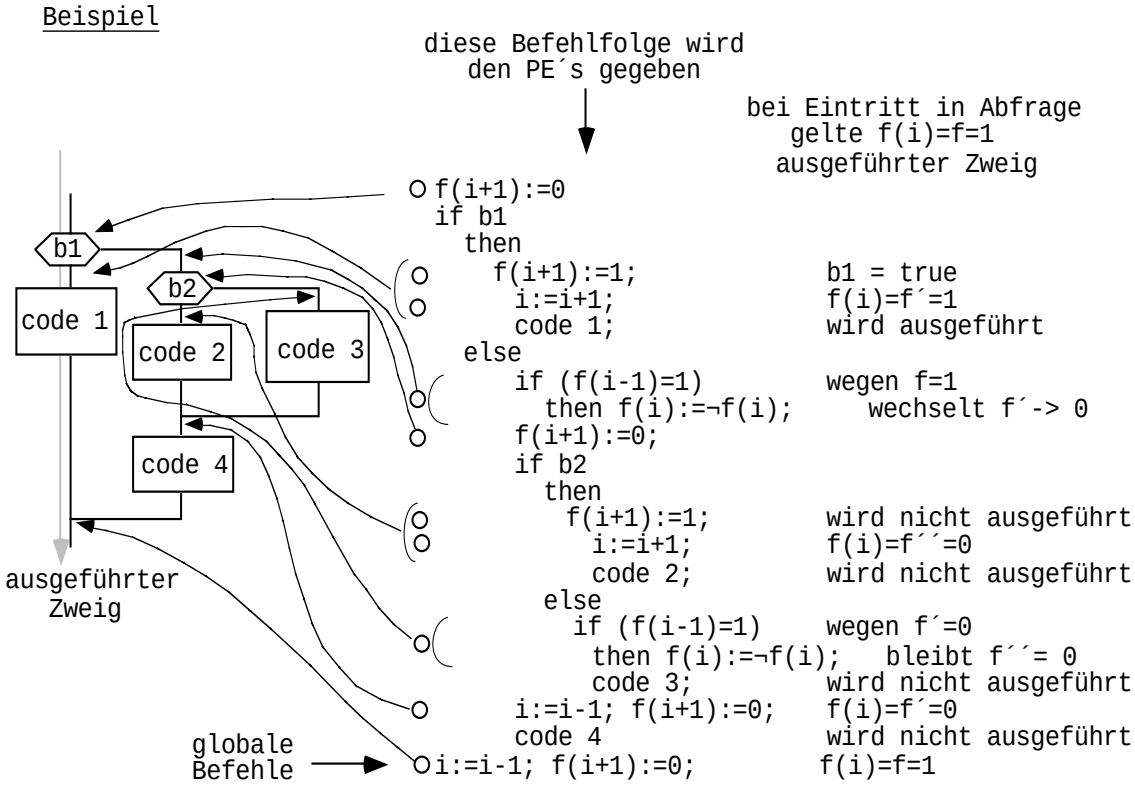
- i := i+1 Erhöhen der Schachtelungstiefe
- i := i-1 Rückkehr
- f(i+1) := 0 Sperren der Ausführungserlaubnis in der nächst höheren Ebene
- if (f(i-1) = 1) then f(i) := ¬ f(i)
  Invertieren der Ausführungserlaubnis .

Wenn f(i) = 0 , dann werden alle Befehle außer den globalen Befehlen übergangen.

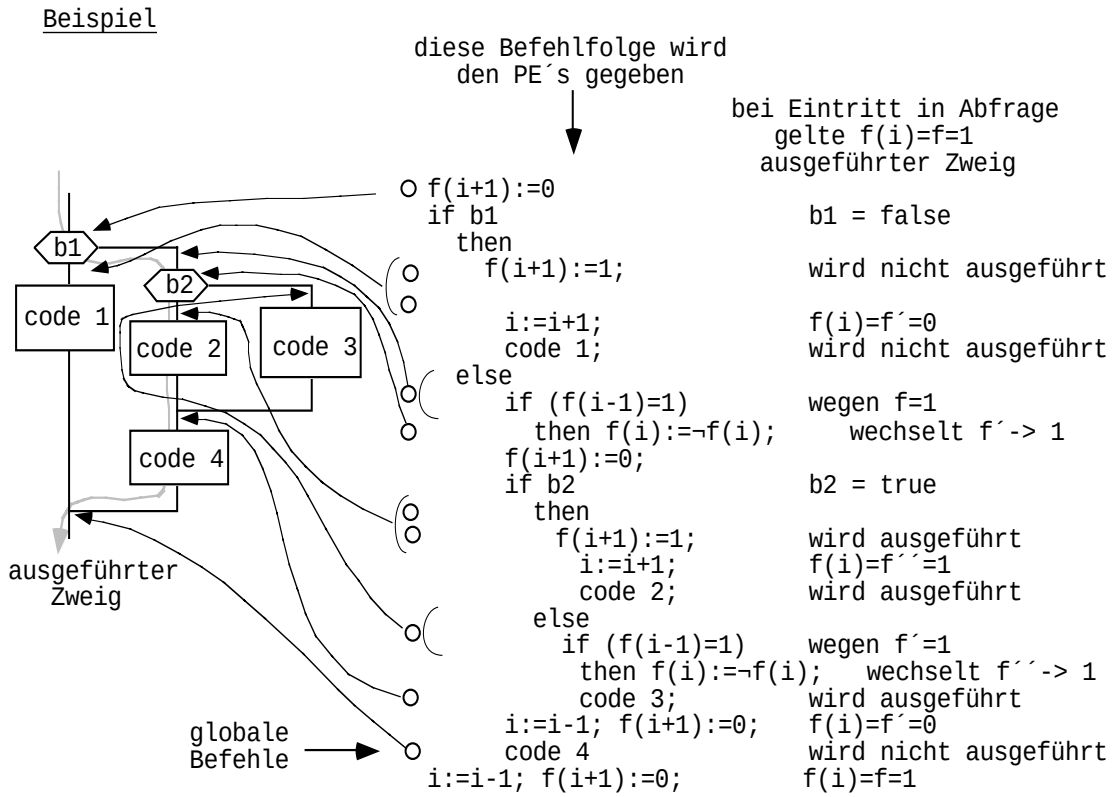
Eine if-then-else-Konstruktion muß dann so umgesetzt werden:

| Kontrollstruktur: | zusätzliche Befehle | |
|---|---|---|
| Initialisierung | i:= 0;<br>f(i):= 1 | |
| if <bedingung> | f(i+1):=0; | Vorbereitung |
| then<br><br><statement> | f(i+1):=1;<br>i:=i+1; | wenn f(i)=1<br>Weiterschalten |
| else<br><br><statement> | if (f(i-1)=1)<br>then f(i):= ¬f(i); | Umschalten nur<br>wenn Abfrage<br>ausgeführt wurde |
| Zusammenführung: | i:= i-1; f(i+1):=0; | f(i) wird gültig<br>f(i+1) rückgesetzt |

Ein Beispiel zeigt verschachtelte if-then-else-Abfragen und das Verhalten wenn der erste then-Zweig durchlaufen wird

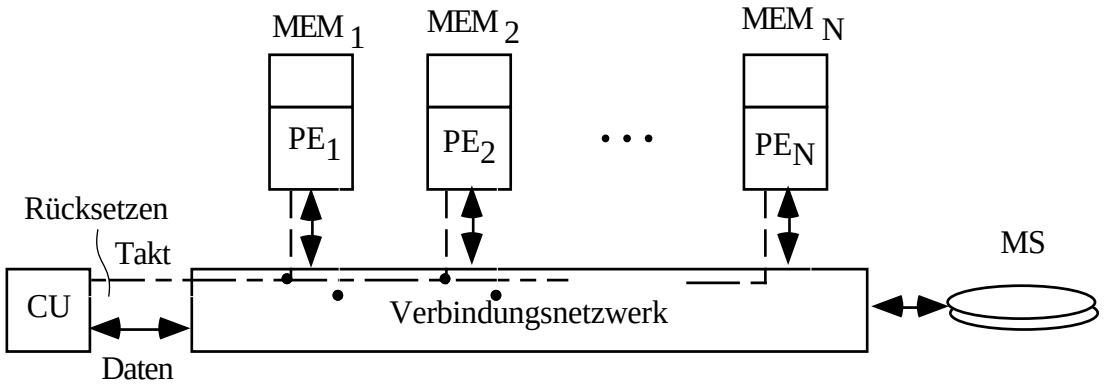



oder der mittlere Ast (in einem anderen PE) durchlaufen wird.



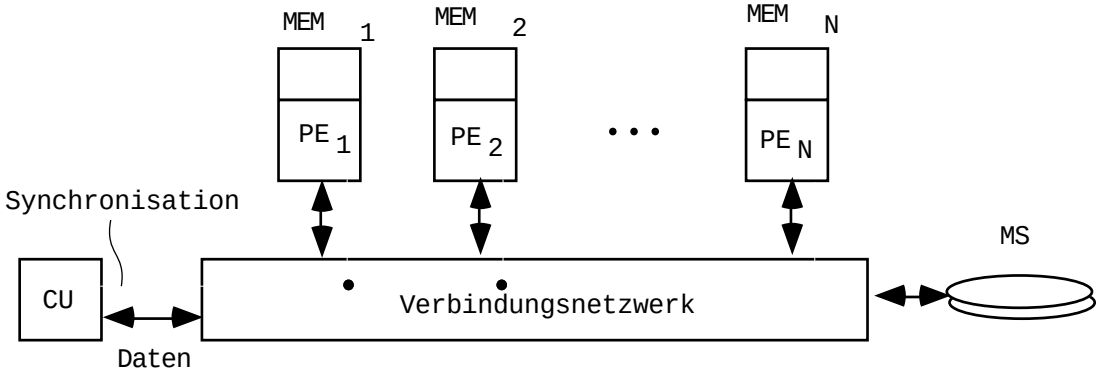
### 5.1.3. SIMD/MIMD-Rechner

Die rechnenden Einheiten sind Rechner, die gemeinsam getaktet werden. In allen läuft das gleiche Programm ab. Zu einer Zeit können datenabhängig ganz unterschiedliche Programmzweige erreicht worden sein. Die PE's müssen sich aber zum Datenaustausch synchronisieren.




### 5.1.4. MIMD-Rechner (multiple instruction, multiple data)

In den PE's laufen verschiedene Programme. Die Rechner synchronisieren sich gelegentlich zum Datenaustausch über das Verbindungsnetzwerk.




Sie sind nicht notwendig miteinander durch einen gemeinsamen Takt verbunden.

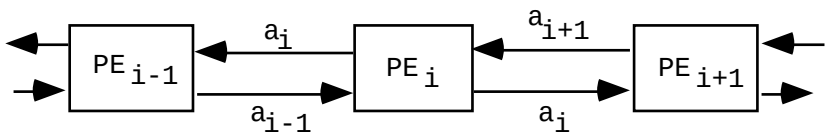
## 5.2. Synchronisation der Datenübertragung

Die Parallelarbeit in Feldrechnern erfordert von gelegentlich bis bei jedem Schritt den Austausch von Daten zwischen den rechnenden Einheiten.

Sie müssen sich dazu synchronisieren:
Wenn $PE_i$ von $PE_k$ Daten übernehmen soll, muß $PE_i$ wissen, wann die Daten am Ausgang von $PE_k$ gültig sind, und $PE_k$ muß wissen, wann er die Daten gültig und wieder ungültig schreiben darf.

### 5.2.1. Gemeinsamer Takt und identische Befehle in allen PE's

Der Datenaustausch von $PE_i$ mit seinen Nachbarn $PE_{i-1}$ und $PE_{i+1}$ folgt diesem Schema für $PE_i \leftarrow PE_{i-1}$ .




$\forall_i$ : $PE_i$ legt seinen Ausgangswert $a_i$ nach rechts heraus

$PE_i$ übernimmt vom linken Nachbarn dessen Ausgangswert $a_{i-1}$ .

Bei zellularen Automaten erfolgt die Übernahme synchron von allen Nachbarn und geschieht in der Zeit zwischen zwei Takten: mit Takt n wird ein neuer Wert $z_i$ $\forall_i$ in das Zustandsregister geschrieben. Mit $x_i = z_i$ stehen nun der Zustand und die Werte der Nachbarn zur Verfügung und $z_i^{(n+1)} = \delta$ (Nachbarn, $z_i^{(n)}$) kann gerechnet werden. Mit dem Takt (n+1) wird dann der neue Wert in das Zustandsregister übernommen.

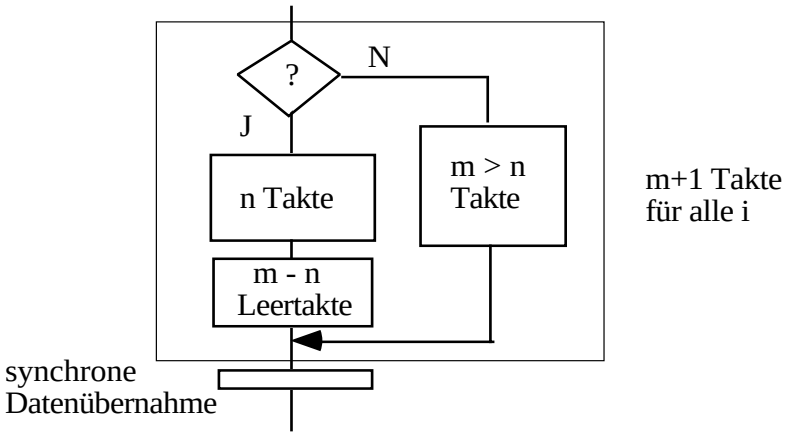
Sind die PE's keine Automaten sondern Rechner, dann können viele Befehle gebraucht werden, um synchron in allen PE's Ausgangswerte zu rechnen. Werden sie nacheinander in Ausgangsregister eingeschrieben, kann die Datenübernahme im Prinzip wieder parallel für alle Eingänge erfolgen.

Sind die Verbindungsleitungen zwischen $PE_i$ und $PE_k$ unidirektional kann parallel Datenaustausch erfolgen: $PE_i \rightarrow PE_k$ und $PE_k \rightarrow PE_i$

Bei einer bidirektionalen Leitung kann zu einer Zeit nur eine Richtung betrieben werden: $PE_i \rightarrow PE_k$ und $PE_k \rightarrow PE_i$ brauchen (mindestens) zwei Takte.

### 5.2.2. Gemeinsamer Takt und gleiches Programm in allen PE's mit bekannter Befehlsanzahl

Um synchronen Datenaustausch vornehmen zu können, müssen bei Verzweigungen im Programm ggf. NOP's eingefügt werden, um alle Zweige auf die gleiche Anzahl von Befehlen (Takten) zu bringen.

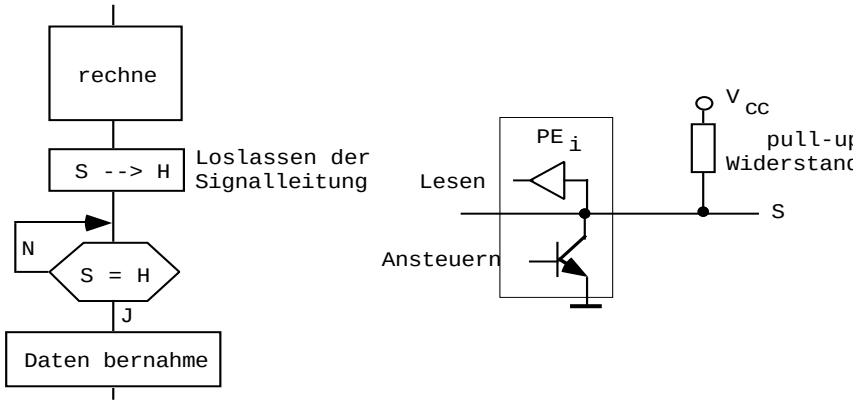



Schleifen vorab bekannter Durchlaufzahl (for-Schleifen in Pascal) müssen in anderen Programmzweigen durch entsprechend viele NOP's kompensiert werden.

### 5.2.3. Gemeinsamer Takt und separate Rückmeldeleitung

Jedes PE rechnet seine Ausgangsdaten, legt sie bereit und gibt am Ende eines Rechenschritts die Rückkopplungsleitung frei: über ein wired-AND hängt die Leitung an einem pull-up-Widerstand.

Dann wartet der PE bis die Leitung auf Eins läuft und übernimmt die Daten der Nachbarn.




Die Programme können unterschiedlich lange laufen. Probleme entstehen bei sehr vielen PE's : sei $N = 10^4$ , der pull-up-Widerstand $R = 1\ k\Omega$ und der off-Widerstand eines Transistors $10\ M\Omega$ .

Dann sind $10^4$ Transistoren parallel geschaltet und $R_{||} = 1\ k\Omega$ , d. h. auf der Leitung ist kein eindeutiger logischer Pegel mehr identifizierbar.

### 5.2.4. Selbständig arbeitende PE's ohne gemeinsame Rückmeldeleitung

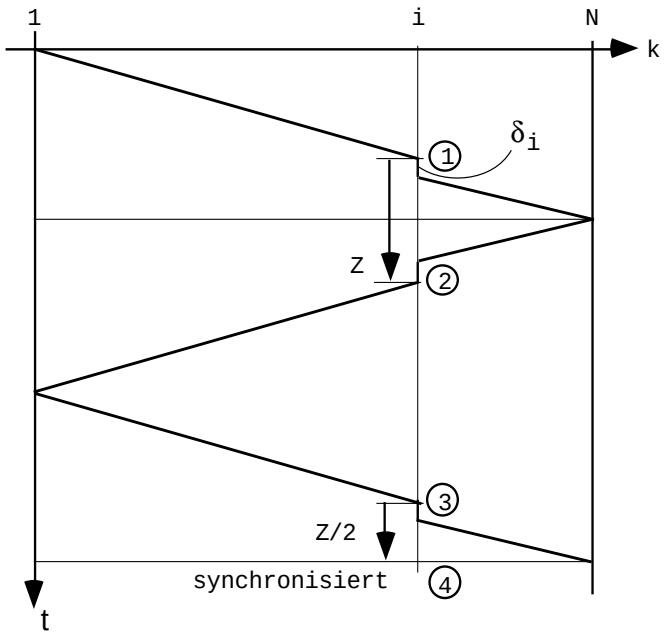
Die Programme enthalten datenabhängige Schleifen und laufen unterschiedlich lange.

Die PE's mögen logisch in einer Kette angeordnet sein: jedes PE kennt seinen Nachbarn zur Rechten und Linken, und $PE_1$ und $PE_N$ wissen um ihre Randposition. Sie haben einen gemeinsamen Takt. Die Synchronisation wird auf das **"firing squad"** Problem zurückgeführt.

Jedes PE hat einen Zähler z , initiiert auf Null. Beginnend mit $PE_1$ über $PE_2$ , ..., $PE_{i-1}$, $PE_i$, $PE_{i+1}$, ... $PE_N$ laufen Erregungswellen:

1. Wenn $PE_{i+1}$ von links angestoßen wird, startet er seinen Zähler und gibt den Anstoß weiter. $PE_N$ reflektiert den Anstoß und zählt nicht.
2. Wenn $PE_i$ von rechts angestoßen wird, stoppt er den Zähler, bildet z := z/2 , und gibt den Anstoß nach links weiter.
3. Wenn $PE_1$ angestoßen wird, halbiert er seinen Zählerstand und startet den Zähler erneut. Er gibt den Anstoß nach rechts.
   Wenn $PE_i$ von links angestoßen wird, startet er seinen Zähler und gibt den Anstoß weiter nach rechts.
   Wenn $PE_N$ angestoßen wird sind zugleich alle Zähler = 0 und die Kette ist synchronisiert.

Ist $PE_i$ beim ersten Anstoß noch nicht fertig, zählt er die Zeit $\delta t$ bis zum Ende der Rechnung separat und schlägt sie beim Rücklauf auf den Zählerstand drauf, ehe z := z/2 gerechnet wird. $PE_i$ wartet diese Zeit bis zur Weitergabe des Impulses.



## 5.3. Verbindungsnetzwerke (VNW)

### 5.3.1. Maße

Für die Leistungsfähigkeit der parallelen Datenverarbeitung spielt das Verbindungsnetzwerk zwischen den PE's eine entscheidende Rolle, wenn paralleler Datenaustausch notwendig wird. Die Chrakterisierung eines VNW erfolgt mit Hilfe von Maßen, die die Leistungsfähigkeit und/oder den Verbindungsaufwand beschreiben.

#### *5.3.1.1. Anzahl der Verbindungen pro PE : P(PE)*

Zwischen einem PE und dem VNW sind Datenverbindungen möglich. Sei eine Verbindung ein Leitungsbündel von n Bit (n = 1, 8, 16, 32). Die Verbindung sei bidirektional (lesen und schreiben) oder unidirektional; dann werden 2 x n Leitungen gebraucht.

Die Begrenzung in der Zahl der Verbindungen ist gegeben durch die Pinanzahl auf dem Chip. Chips mit pin-grid-array (PGA) haben ca. 400 Pins.

Rechnet man mit 32 Bit/Verbindung dann ist P(PE) = 12 (12 x 32 = 384 Leitungen).

Typische Werte: P(PE) = 1 ... 16 ... (32)

P(PE) ist eine **kleine** ganze Zahl.

#### *5.3.1.2. Maximaler Abstand $n_{max}$*

Der "Abstand" zwischen PE's wird so definiert:

$PE_i$ und $PE_k$ haben den Abstand n
$\Leftrightarrow$ die Verbindung $PE_i \rightarrow PE_k$ läuft über
n-1 Zwischenstationen $PE_j$
man schreibt $a (PE_i, PE_k) = n$ .

Im allgemeinen gibt es mehrere mögliche Verbindungen zwischen $PE_i$ und $PE_k$ .

Dann ist der maximale Abstand $n_{max}$ definiert als

$$n_{max} = \max_{\forall_{i,k}} (\min a (PE_i , PE_k))$$

#### *5.3.1.3. Anzahl PE's im Abstand n um $PE_i$*

A(n) ist die Anzahl der im Abstand n erreichbaren rechnenden Elemente ausgehend von einem $PE_i$ . Wird der Abstand vergrößert von n-1 $\rightarrow$ n werden genau A(n) weitere PE's erreichbar.

Dann gilt

$$\sum_{i=1}^{n_{max}} A(i) = N - 1$$

### 5.3.2. Klassifikation von Verbindungsnetzwerken

#### *5.3.2.1. Starre VNW's*

Jedes PE ist über das VNW mit P(PE) Nachbarn starr verbunden.

Die Güte der Verbindung wird durch den Algorithmus bestimmt, der die Art des Datenaustausches festlegt. Zu jedem starren VNW gibt es einen Algorithmus, der "schlecht" paßt und andere, die günstig sind.

Nummeriert man die PE's durch, dann ist $PE_i$ verbunden mit $PE_{k1}, PE_{k2}, ..., PE_{kr}$ .

Die Art der Verbindung kann sein:

- Einzelverbindung $PE_i \rightarrow PE_k$ für ein Paar (i, k)
- multiple Verbindung $PE_i \rightarrow PE_{k(i)} \ \forall_i$ (z. B. $PE_i \rightarrow PE_{i+1} \ \forall_i$
- Mehrfachverbindung $\underbrace{PE_i \rightarrow PE_{k1(i)}, ..., PE_{kr(i)}}_{\text{P(PE) Verbindungen}} \ \forall_i$

  (z. B. $i \rightarrow 2_i, 2_{i+1}, \forall_i$
  oder $i \rightarrow i+1, i-1, i+2^m, i-2^m$)

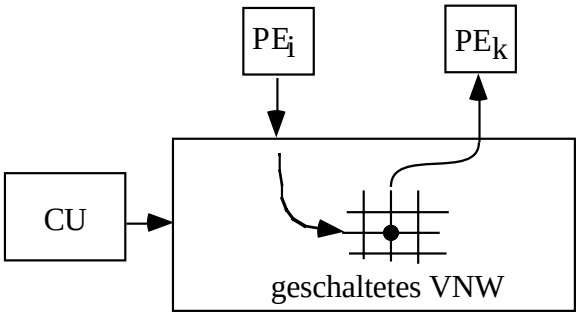
- Broadcast: Es gibt ein ausgezeichnetes $PE_i$ , das Nachrichten an alle verteilen kann:
  $PE_i \rightarrow PE_j \ \forall_{j \neq i}$
- Permutation: Eine fest eingestellte Permutation ist als Verbindungsstruktur realisiert:

  $$\begin{pmatrix} 1 & \ldots & N \\ k_1 & \ldots & k_N \end{pmatrix} \ ; \ k_i \neq k_j \quad \forall_{(i,j)}$$

  $PE_i \rightarrow PE_{ki}$

#### *5.3.2.2. Geschaltete VNW's*

Eine Steuereinheit (control unit, CU) erzeugt die Schaltinformationen für einen Verteiler, der $PE_i \rightarrow PE_{ki}$ durchschaltet.




Damit läßt sich das Verbindungsnetzwerk an den jeweils behandelten Algorithmus anpassen: in einer Vorbereitungsphase werden die Daten aufbereitet und geladen und das VNW geschaltet. Dann läuft die Rechnung mit parallelem Datenaustausch. Da der Aufwand zum Einstellen einer Permutation hoch ist (N Zahlen von 1,…, N brauchen zur Darstellung N · ld N Bit) wird man bisweilen nicht alle möglichen Permutationen einstellen können wollen, sondern sich mit einer Reihe gängiger Verbindungen begnügen. Diese Konfigurationen ruft man dann aus einem Speicher ab.

### 5.3.3. Verbindungsnetzwerke starrer Struktur mit $n_{max}$ = const unabhängig von N

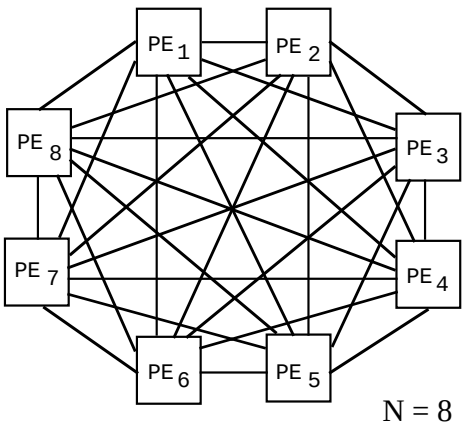
Diese in 5.3.2.1. charakterisierten VNW sollen nun näher betrachtet werden.

Sie werden nach dem maximalen Abstand klassifiziert.

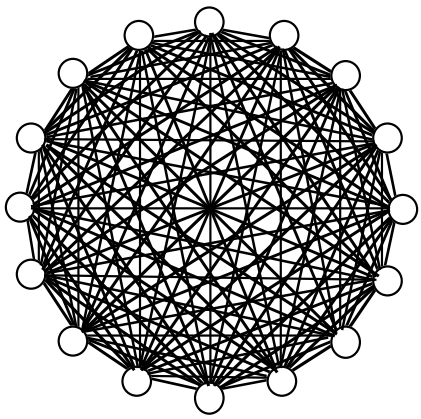
#### *5.3.3.1. Vollständiger Graph*

Jedes PE ist mit jedem verbunden.

$n_{max}$ = 1 (direkte Nachbarn)
A(n) = A(1) = N-1
P(PE) = N-1




N = 8



N = 16

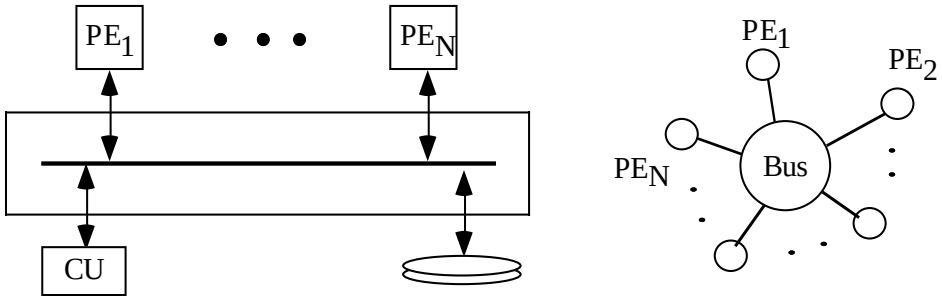
Die praktische Realisierung ist nur für wenige PE's machbar bis N = 16 , wegen P(PE) = N-1 und der Gesamtzahl der Verbindungen

$\frac{N(N-1)}{2} \sim N^2$ .

Dafür ist paralleler Datenaustausch möglich mit jeder Permutation inklusive Broadcast.

*5.3.3.2. Bussystem*

Jedes PE ist mit allen anderen über einen Bus verbunden.



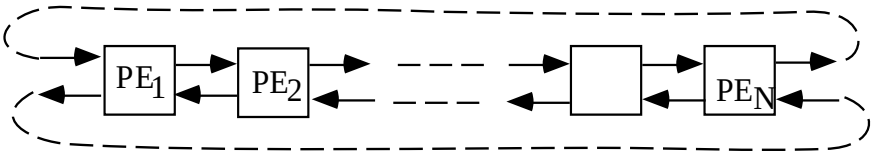

Da für jeden Transfer der Bus angefordert werden muß, ist der Abstand zwischen zwei PE's $n_{max} = 2$ .

Zugleich ist A(2) = N-1 , A(1) = 0 und P(PE) = 1.

Paralleler Datenaustausch ist nicht möglich, wohl aber die Verteilung von Daten durch ein Broadcast.

### 5.3.4. Starre Struktur mit $n_{max} \sim N^{\alpha}$ ($\alpha$ = 1, 1/2, 1/3, ...) P(PE) = const

*5.3.4.1. Lineare Kette*




Es sei die Kette zum Ring geschlossen. Dann ist A(1) = 2 , A(2) = 2 , ..., A(n) = 2 und

$n_{max} = \left\lceil \frac{N-1}{2} \right\rceil$

Wenn die lineare Kette offen ist, dann ist $n_{max} = N-1$ .

Von den parallelen Algorithmen, die sich mit einer linearen Kette gut parallelisieren lassen sind unter den Iterationen der Art

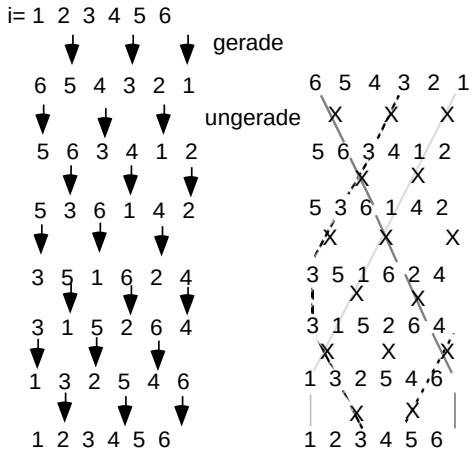
$$x_i^{(n+1)} = f\,(x_i^{(n)}, x_{i-1}^{(n)}, x_{i+1}^{(n)})$$

auch Sortiervorgänge. Das folgende Bild zeigt ein Beispiel.

```
for k=1 to N/2 do
begin
für alle PE(i) mit i(mod2)=0 do
vertausche mit linkem Nachbarn
für alle PE(i) mit i(mod2)=1 do
vertausche mit linkem Nachbarn
end
```

vertausche mit linkem Nachbarn
für (un)gerade PE´s

lies Wert vom linken Nachbarn
vergleiche
behalte größeren Wert
stelle den kleineren Wert
am linken Ausgang bereit
für alle PE(i) mit i(mod2)=1 (0) do
übernimm Wert vom rechten Nachbarn



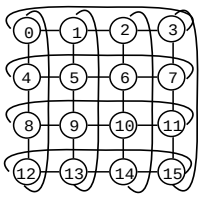

**Sortieren mit linearer Kette**

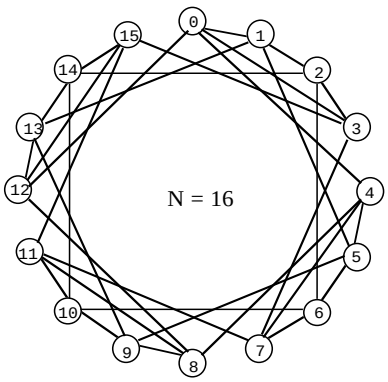
Der Aufwand ist N Schritte im Gegensatz zum sequentiellen Fall mit N ld N Schritten.

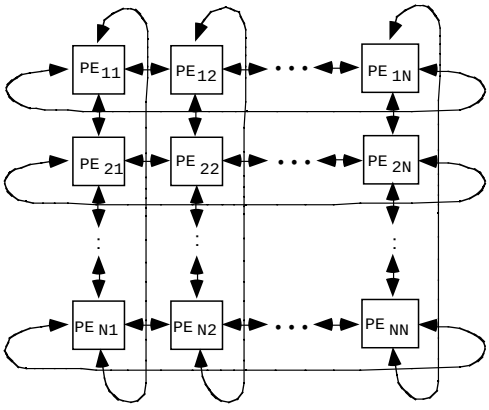
Mit einer linearen Kette als VNW lassen sich auch Algorithmen behandeln, die eigentlich ein quadratisches Gitter brauchen: In jedem PE ist ein Zeilen- oder Spaltenvektor einer Matrix gespeichert und parallel kann eine Zeile oder Spalte einer Matrix gerechnet werden.

Der Aufwand sinkt von $O(N^3)$ im sequentiellen Fall auf $O(N^2)$ .

### *5.3.4.2. Quadratisches Gitter*




Es ist P(PE) = 4 : Verbindungen zu den Nachbarn im Osten, Süden, Westen und Norden im Gitter.

Es ist $A(1) = 4$ , $A(2) = 8$ , $A(3) = 12$
allgemein : $A(n) = 4\,n$ .

Im allgemeinen arbeitet man auf einer torusförmigen Konfiguration:

Sei $N = 2^m$ und damit die Seitenlänge $2^{m/2}$ .

Dann wird $PE_j = PE_{(i,k)}$ mit

$i = 1, ..., 2^{m/2}$ und $i = 1 \equiv 2^{m/2} + 1$
$k = 1, ..., 2^{m/2}$ und $k = 1 \equiv 2^{m/2} + 1$

Mit $A(n) = 4\,n$ für große Gitter, da sonst Clipping erfolgt, wird

$$\sum_{i=1}^{n_{max}} A(i) = N-1$$

$$4 \cdot \sum_{i=1}^{n_{max}} i = 4\,\frac{n_{max}\,(n_{max}-1)}{2} = N-1$$

$$\Rightarrow 2\,n_{max}^2 - 2\,n_{max} = N-1$$

$$n_{max} = -\frac{1}{2} \pm \sqrt{\frac{1}{4} + \frac{N-1}{2}} \sim N^{1/2}$$

Ein quadratisches Gitter ist geeignet für Algorithmen der Form

$$x_{i,k}^{(n+1)} = f\,(x_{i,k}^{(n)}\,,\ x_{i-1,k}^{(n)}\,,\ x_{i+1,k}^{(n)}\,,\ x_{i,k-1}^{(n)}\,,\ x_{i,k+1}^{(n)})$$

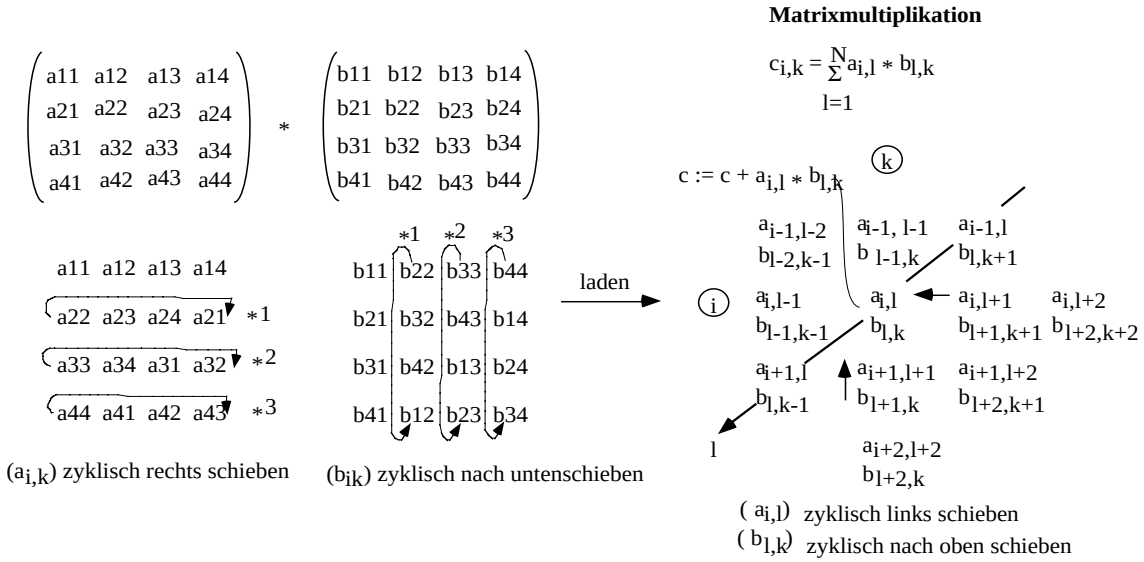
Prominentestes Beispiel ist die Matrixmultiplikation.

Beispiel:
Gegeben seien zwei $m \times m$ - Matrizen $A$ und $B$ , gesucht sei das Produkt $C = A \cdot B$ .
Sei $m = n$ und $N = n^2$ .

Dann ist

$$c_{i,k} = \sum_{l=1}^{n} a_{i,l} \cdot b_{l,k}$$

Lade die Matrizen in die Rechner $PE$ nach folgender Anfangskonfiguration.




Bei der Stelle (i,k) ist $L = (i+k) \bmod n + 1$ .

Bilde in jedem Knoten das Produkt und summiere auf. Schiebe a nach Westen, b nach Norden jeweils mod n (Toruskonfiguration).

Wiederhole n-mal . Dann steht in $PE_{i,k}$ das Resultat $c_{i,k}$ .

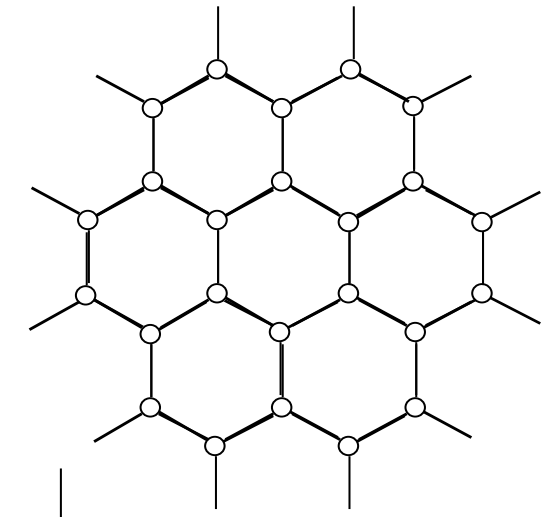
Wenn $m > n$ ist, dann muß in $PE_{i,k}$ mehr als je ein Wert von A und B gespeichert werden und das Resultat als Summe von Teilprodukten behandelt werden.

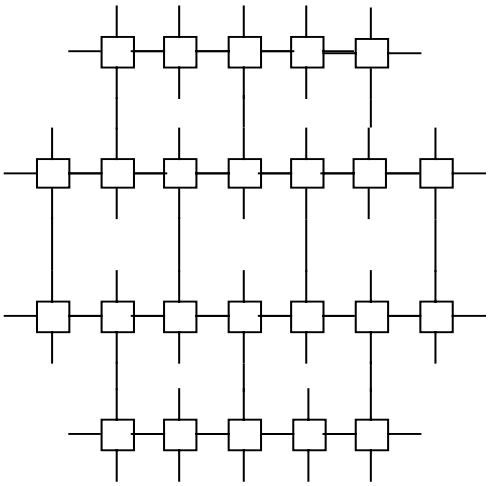
Eine andere Anwendung ist die iterative Lösung partieller Differentialgleichungen.

Beispiel:
Mehrgitterverfahren (Gabriel Wittum, Spektrum der Wissenschaften, 4,
April 1990, pp. 78-90)
Man löst die partielle Differentialgleichung durch ein System von Differenzengleichungen für die einzelnen Fourierkomponenten, die parallel gerechnet werden.

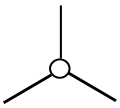
### *5.3.4.3. Hexagonales Gitter*

Dieses Gitter läßt sich auf ein quadratisches Gitter zurückführen, es ist hier zwar P(PE) = 3 aber im Endeffekt $n_{max} \sim N^{1/2}$ wie beim quadratischen Gitter.

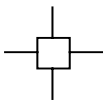




**hexagonales Gitter**



P(PE) = 3



P(PE) = 4

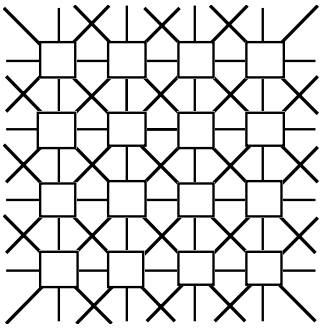
A(1) = 3
A(2) = 6
A(3) = 9
A(n) = 3 n

### *5.3.4.4. Oktogonales Gitter*

Hier ist jedes PE mit seinen 8 nächsten Nachbarn verbunden: P(PE) = 8 .

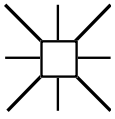
Damit werden Algorithmen wie das "Game of Life" parallel rechenbar.

Da es sich aber nach wie vor um eine planare Struktur handelt (zur Toruskonfiguration gefaltet) ist $n_{max} \sim N^{1/2}$ .



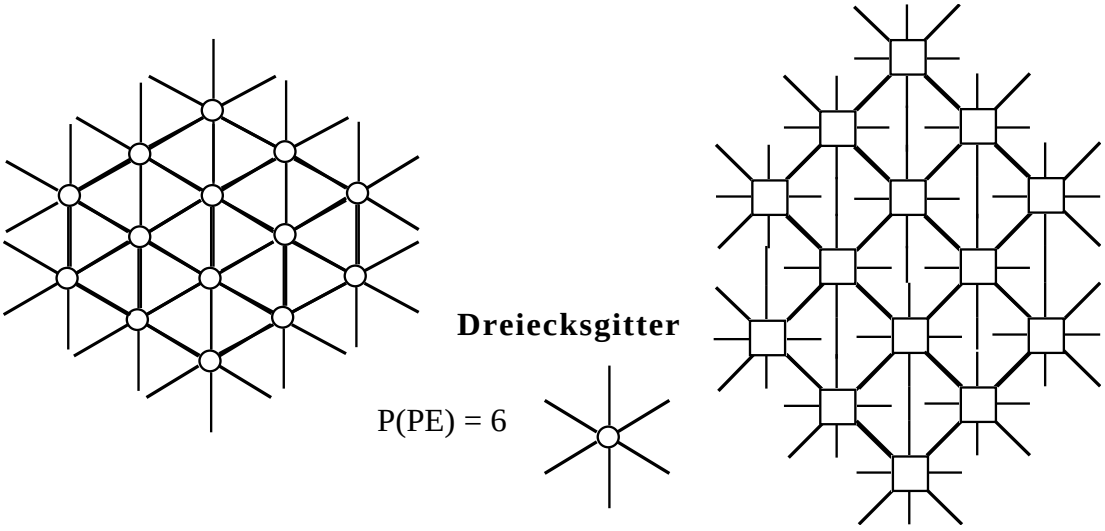
**oktogonales Gitter**

P(PE) = 8



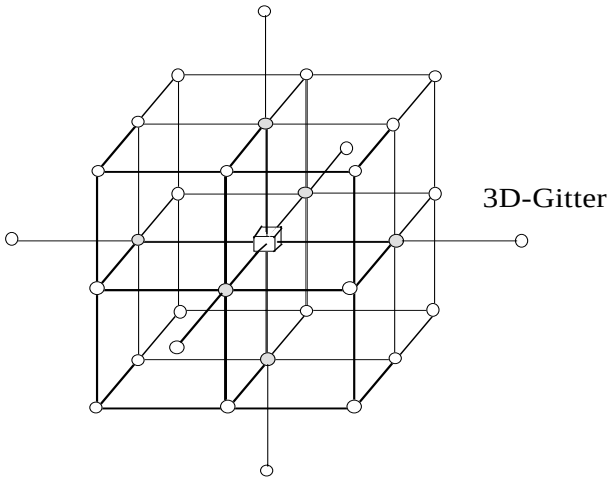
### *5.3.4.5. Dreiecksgitter*

Hier wird die Ebene tesseliert in Dreiecke: P(PE) = 6 . Ein Dreiecksgitter läßt sich durch ein oktogonales Gitter simulieren.




### *5.3.4.6. Kubisches Gitter*

Jedes PE hat 6 Nachbarn: P(PE) = 6




Es ist $A(1) = 6$
$A(2) = 26$
$A(n) \sim n^2$ (Kugeloberfläche)

und $n_{max} \sim N^{1/3}$ (Kugelvolumen)

Da für eine gegebene Seitenlänge n die Anzahl der PE's mit $n^3$ skaliert, wird der Aufwand rasch sehr groß. Dann wird man ein 3-D-Gitter der Seitenlänge n simulieren durch ein quadratisches Gitter und alle Daten mit j = 1, ... n an der Stelle $PE_{i,k}$ speichern.

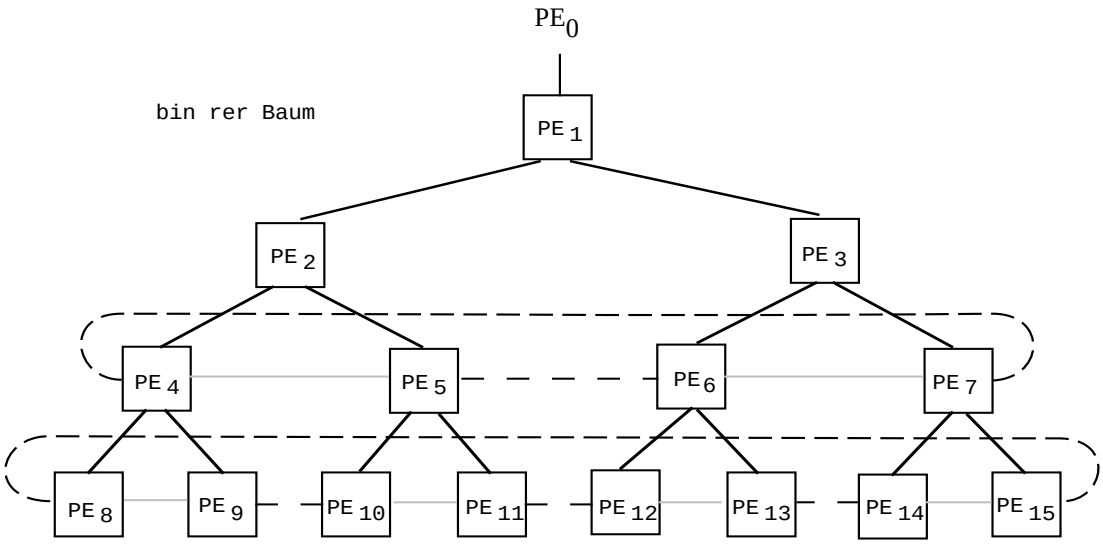
### 5.3.5. Starres VNW mit $n_{max} \sim ld\ N$ und P(PE) = const

Hier wächst der maximale Abstand nur schwach mit der Anzahl der PE's , erlaubt durch eine spezielle Struktur der Verbindungen.

#### *5.3.5.1. Binärer Baum*

Es ist ein inhomogenes, hierarchisch gegliedertes Netzwerk mit P(PE) = 3 .

$PE_i$ ist verbunden mit $PE_{2i}$ und $PE_{2i+1}$ und $PE_{idiv2}$ ; i = 0 , ... , N-1 , $N = 2^m$ .




Der Wurzelknoten $PE_1$ ist ausgezeichnet; $PE_0$ ist der Einstieg in den Baum.

Der maximale Abstand zwischen zwei PE's ist $n_{max} = 2\ (m-1) \sim ld\ N$ .

A(1) = 3 (max)
A(2) = 6 (max)
A(3) = 12 (max)
$A(n) = 3 \cdot 2^{n-1}$ .

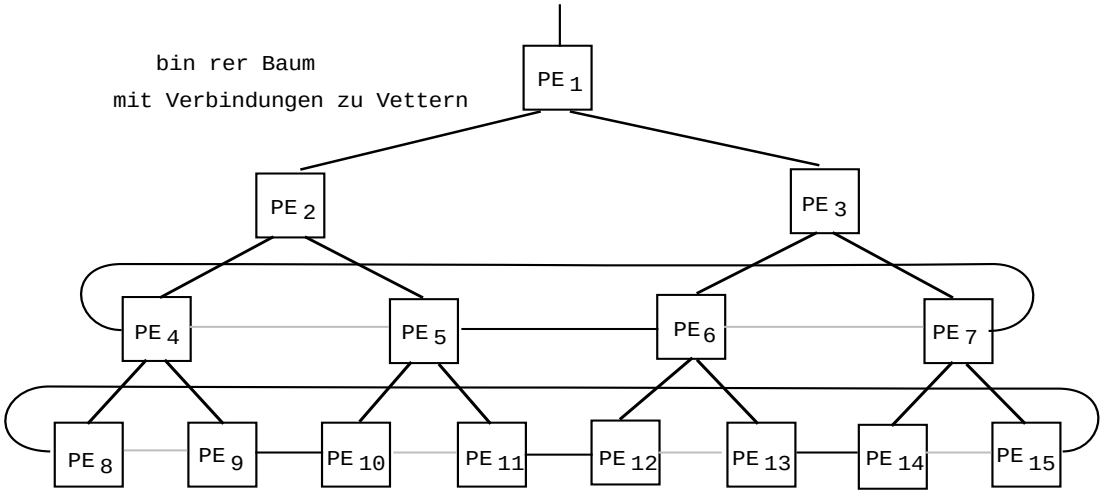
Im schlimmsten Fall muß man von einem Blatt zum Wurzelknoten laufen (m-1 Schritte) und wieder zu dem entfernten Blatt (m-1 Schritte).

Binäre Bäume sind geeignete Verbindungsstrukturen für

- Verteilvorgänge:
  Einspeisen in den Einstiegsknoten $PE_0$, von dort aus Verteilung in m Schritten auf $2^m - 1$ Knoten.
- Ermittlung eines Extremwertes (max. Wert)
  In den $N = 2^m$ Knoten stehen N Werte.
  In jedem Schritt
  lade Werte der Nachfolgerknoten
  speichere den größeren im Vaterknoten .
  Nach m Schritten steht im Einstiegsknoten der Extremwert aus allen N Werten.

- Summation von N Werten
  In jedem Schritt tue
  summiere zum Wert im Knoten die Werte der Nachfolgerknoten.
  Nach m Schritten steht im Einstiegsknoten die Gesamtsumme.

*5.3.5.2. Binärer Baum mit Verbindungen zwischen Vettern*




Hier ist P(PE) = 4
$n_{max}$ reduziert sich auf 2(m-1) - 2 oder in günstigen Fällen auf 2(m-1) - 3 .

*5.3.5.3. Binärer Baum mit Querverbindungen*

Hier ist P(PE) = 5 und $n_{max} = 2(m-1) - 2$

In beiden Fällen ist paralleler Transfer $PE_i \rightarrow PE_{k(i)}$ $\forall_i$ nur eingeschränkt möglich.

Paralleler Datentransfer ist eher möglich mit

*5.3.5.4. Pyramidennetzwerk*

Zu jedem $PE_i$ sind zugeordnet ein quadratisches Gitter mit denen PE verbunden ist und vier Verbindungen zu PE's der nächsten Stufe.

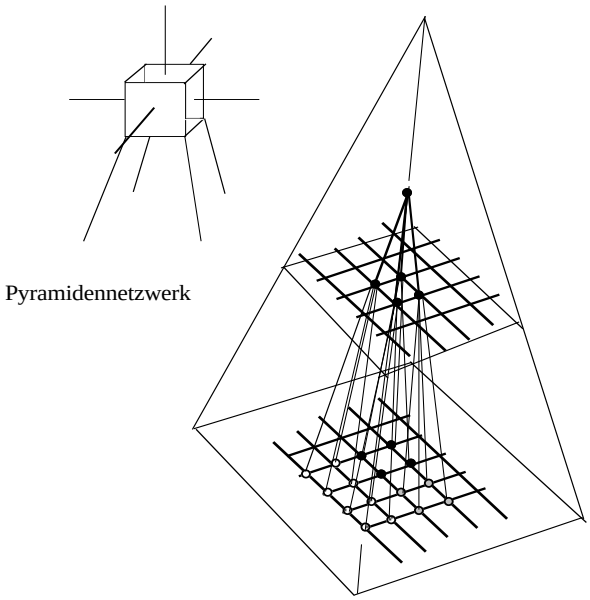
$PE_i$ ist in der nächsten Stufe verbunden mit $PE_{4i+1}$, $PE_{4i+2}$, $PE_{4i+3}$, $PE_{4i+4}$ und eine Verbindung zur darüberliegenden Schicht zu $PE_{idiv4}$.

Innerhalb seiner Schicht ist PE eingebunden in ein quadratisches Gitter und seine Nachbarn sind implizit durch i festgelegt.

Sei $i = k_0 \cdot 4^p + k_1 \cdot 4^{p-1} + k_2 \cdot 4^{p-2} + k_3$

d. h. $PE_i$ liegt auf Stufe p = 3 .

Dann sind unmittelbare Nachbarn $PE_{k3+1}$ , $PE_{k3+2}$ , $PE_{k3+3}$ jeweils mod 4 gerechnet (0 entspricht 4) und die Vettern $Pe_{k0}$ , $PE_{k1}$ , $PE_{k2} \in \{1, 2, 3, 4\}$ .



Pyramidennetzwerk

Jedes PE ist mit 9 Nachbarn verbunden, d. h. P(PE) = 9 und der maximale Abstand ist nicht durch das quadratische Gitter bestimmt, sondern durch die Quadtreedarstellung.

Sei p die maximale Stufenzahl, dann ist

$$N = \sum_{i=0}^{p} 4^p \ ;$$

p = 3 $\Rightarrow$ N = 84
p = 4 $\Rightarrow$ N = 341
p = 5 $\Rightarrow$ N = 1365
p = 6 $\Rightarrow$ N = 5461

und $n_{max} = 2 \cdot p$ .

*5.3.5.5. Perfect Shuffle (PS)-Netzwerk*

Sei $j = (p_{m-1} , \dots , p_0)_2$ die Adresse von $PE_j$ , $j = 0 , \dots , 2^m - 1$ .

Dann sind für $PE_j$ zwei Verbindungen definiert:

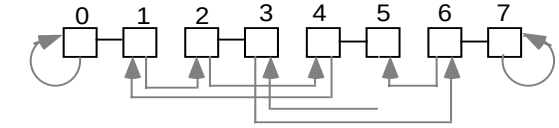
**shuf** $(p_{m-1} , p_{m-2} , \dots , p_1 , p_0) = (p_{m-2} , p_{m-3} , \dots , p_1 , p_0 , p_{m-1})_2$

und

**exch** $(p_{m-1} , p_{m-2} , \dots , p_1 , p_0) = (p_{m-1} , p_{m-2} , \dots , p_1 , \neg p_0)$ .

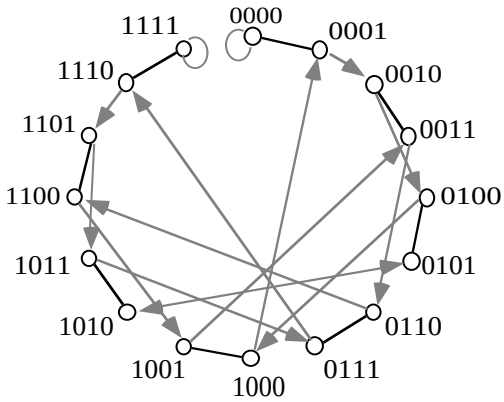
Während exch(j) = k und exch(k) = j gilt ist i. allg. shuf(j) = k und shuf(k) $\neq$ j .

Damit ist P(PE) = 3 .




shuf →

exch ——




Mit $N = 2^m$ ist $n_{max} = 2m - 1$ .

Im ungünstigsten Fall läßt sich eine Adresse $(p_{m-1} , ... , p_0)_2$ umformen in die Adresse $(\neg p_{m-1} , ... , \neg p_0)_2$ in $2m - 1$ Schritten shuf und exch.

shuf $(p_{m-1} , ... , p_0) = (p_{m-2} , ... , p_0 , p_{m-1})$

exch $(p_{m-2} , ... , p_0 , p_{m-1}) = (p_{m-2} , ... , p_0 , \neg p_{m-1})$

shuf $(p_{m-2} , ... , p_0 , \neg p_{m-1}) = (p_{m-3} , ... , p_0 , \neg p_{m-1} , p_{m-2})$

exch $(p_{m-3} , ... , p_0 , \neg p_{m-1} , p_{m-2}) = (p_{m-3} , ... , p_0 , \neg p_{m-1} , \neg p_{m-2})$

…

### 5.3.6. Starres Netzwerk mit $n_{max} \sim ld\ N$ und $P(PE) \sim ld\ N$

Diese Netzwerke sind sehr untersucht worden, da sie einen günstigen Kompromiß bilden zwischen der begrenzten Anzahl von Verbindungen und geringem maximalem Abstand.

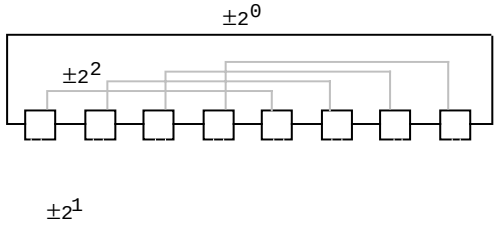
#### *5.3.6.1. Plus-minus-$2^i$ Netzwerke (PM 2I)*

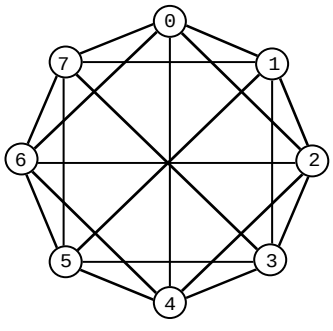
Sei $N = 2^m$ die Zahl der PE's.

Ein $PE_j$ ist verbunden mit Nachbarn $PE_k$ mit $k = j \pm 2^i$ , $i = 0 , ... , m-1$ .

Das erste Bild zeigt die Verbindungen für $N = 8$

**PM2i**

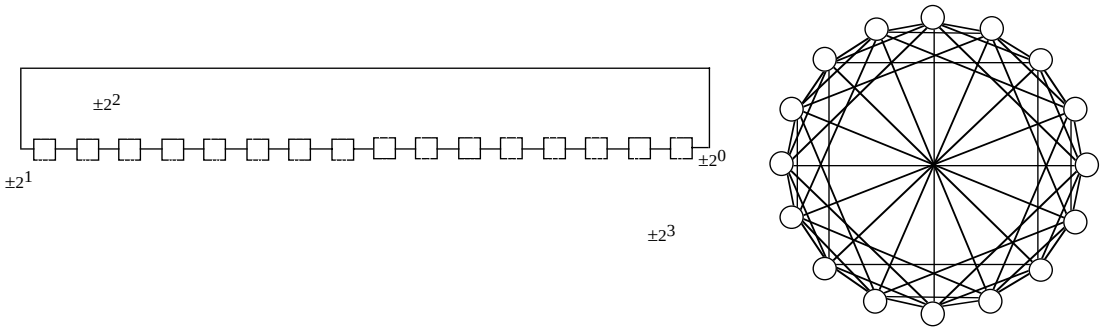



und das zweite Bild die Verbindungen für $N = 16$ .

Der Übersichtlichkeit halber und um die Symmetrien zu zeigen ist eine Kreisdarstellung jeweils mit angegeben.

Es ist $P(PE) = 2m - 1$ und $n_{max} = \lceil m/2 \rceil$.




*5.3.6.2. Hypercube*

Man nennt dieses Netzwerk auch booleschen Kubus oder kubisches Netzwerk.

Sei $j = (p_{m-1}, \ldots p_0)_2$ die Nummer des $PE_j$ mit $j = 0, \ldots, N-1$ ; $n = 2^m$

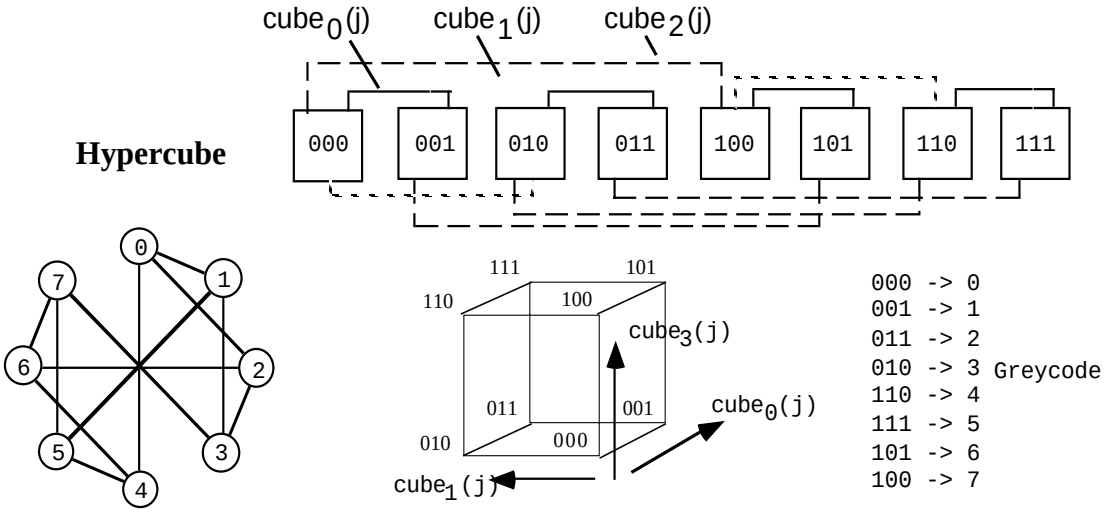
dann sind m Funktionen

$cube_i(j)$ definiert als Verbindungsfunktionen

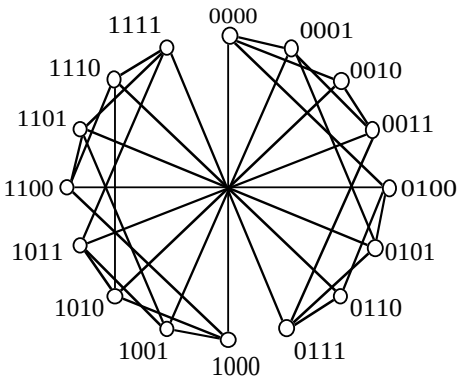
$$cube_i(j) = cube_i(p_{m-1}, \ldots, p_i, \ldots, p_0)_2$$
$$= (p_{m-1}, \ldots, \neg p_i, \ldots, p_0)_2 = k .$$

Die folgende Abbildung zeigt für N = 8 drei Darstellungen des Hypercube:

als Kette PE's , als Ecken eines Kubus der mit von 000 bis 111 durchnummeriert ist und als Kreisdarstellung.




Eine weitere Abbildung zeigt den Hypercube für N = 16 wobei die Nummern in natürlicher Binärdarstellung angegeben sind.




Hypercube

Aus der Definition der Funktion $cube_i\ (j)$ liest man ab:

$P(PE) = m$ ; $A(1) = m$ und $n_{max} = m$

Man braucht $m$ Hintereinanderanwendungen von $cube_i\ (j)$ um von $j = (0, ..., 0)_2$ nach $k = (1, ..., 1)_2$ zu kommen.

$cube_0\ (j) = (0, ..., 0, 1)_2$
$cube_1\ (cube_0\ (j)) = (0, ..., 0, 11)_2$
$cube_m\ (cube_{m-1}\ (... (cube_0\ (j)) ... )) = (1, ..., 1)_2$ .

Da man $N = 2^m$ Zahlen so umordnen kann, daß sich benachbarte nur in einem Bit unterscheiden (Gray-codierte Darstellung) ist diese Anordnung äquivalent einer linearen Kette, die auf dem Hypercube realisiert wird.

Mit dieser Umordnung ist parallel ausführbar

$$PE_z \rightarrow PE_{z+1} \quad \forall\ z = 1, ..., N\ .$$

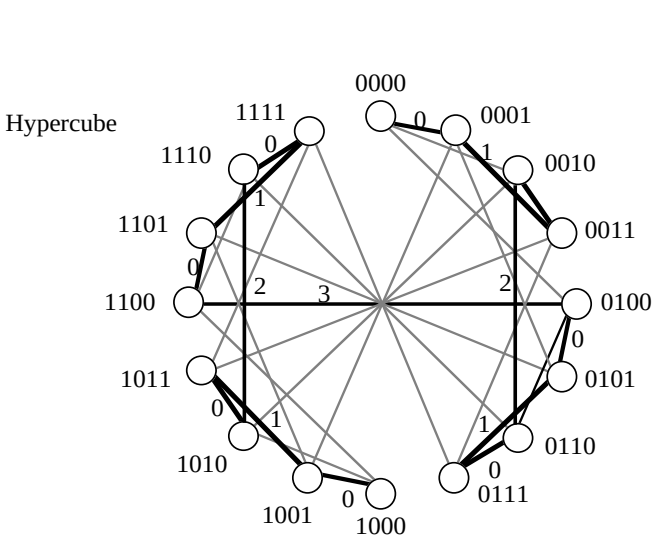
Sei $PE_j$ vorgegeben mit $j = (p_{m-1}, ... , p_0)_2$ , welche Funktion $cube_i\ (j)$ ist dann zu aktivieren?

- Interpretiere $(p_{m-1} , ... , p_0)$ als Gray-codierte Zahl $z$
- Suche ihre Binärdarstellung $q = (b_{m-1} , ... , b_0)_2$ mit $b_{m-1} = p_{m-1}$ und $b_i = b_{i+1} \oplus p_i$
- Bilde $q + 1 = (b_{m-1}^{(+)} , ... , b_0^{(+)})$ mit $b_i^{(+)} = b_i \cdot \neg P_i + \neg b_i \cdot P_i$ ; $P_0 = 1$ ;

  $$P_i = \prod_{i=0}^{m-1} b_i$$

- Stelle $q + 1$ als Gray-codierte Zahl dar
  $(z + 1) = (g_{m-1}^{(+)} , ... , g_0^{(+)})$ mit $g_{m-1}^{(+)} = b_{m-1}^{(+)}$ ; $g_i^{(+)} = b_{i+1}^{(+)} \oplus b_i^{(+)}$
- Bilde $g_i^{(+)} \oplus p_i \quad \forall\ i$ . Dann gibt es genau ein $k$ mit $g_k^{(+)} \neq p_k$ .
  $\Rightarrow$ $cube_k\ (j)$ stellt die richtige Verbindung dar.

Das Bild zeigt die verschiedenen Funktionen, die aktiviert werden.




lineare Kette

| j | $cube_1(j)$ |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0011 | 0 |
| 0010 | 2 |
| 0110 | 0 |
| 0111 | 1 |
| 0101 | 0 |
| 0100 | 3 |
| 1100 | 0 |
| 1101 | 1 |
| 1111 | 0 |
| 1110 | 2 |
| 1010 | 0 |
| 1011 | 1 |
| 1001 | 0 |
| 1000 | 3 |

### 5.3.7. Starres VNW mit $n_{max} \sim ld\ N$ und hybrider Verbindungsstruktur

#### *5.3.7.1. Cube-Connected-Cycles (CCC)*

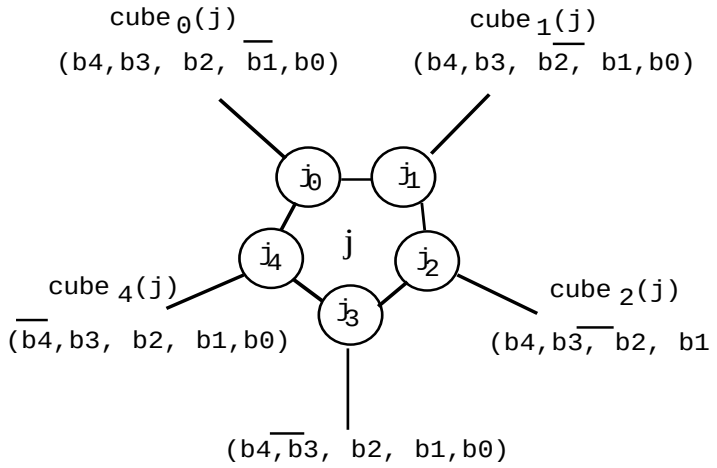
*(Preparata & Vuillemin, Proc. 20th IEEE FOCS, pp. 140-147, 1979)*

Um die bei größeren Netzwerken störende Abhängigkeit von $P(PE) \sim ld\ N$ beim Hypercube zu umgehen, ersetzt man jeden Knoten in einem Hypercube der Dimension $m$ durch einen Ring von $m$ Knoten mit $P(PE) = 3$ : zwei Verbindungen zu den $(m-1)$ anderen Knoten im Ring und eine, die $cube_i\ (j)$ realisiert. Die Anzahl der Prozessoren erhöht sich damit von $N = 2^m$ auf $m \cdot 2^m$ Knoten.

Sei $j = (p_{m-1}, \ldots, p_0)_2$ die Adresse des Knotens $PE_j$ und $j = 0, 1, \ldots, 2^m - 1$ .

Seien $k = m \cdot j, m \cdot j + 1, \ldots, m \cdot j + m - 1$ die Adressen der $m$ Prozessoren im Knoten $j$ , dann verwaltet der Prozessor $k = m \cdot j + i$ das Adressbit $p_i$ des Knotens $j$ über die Funktion $cube_i\ (j)$ .

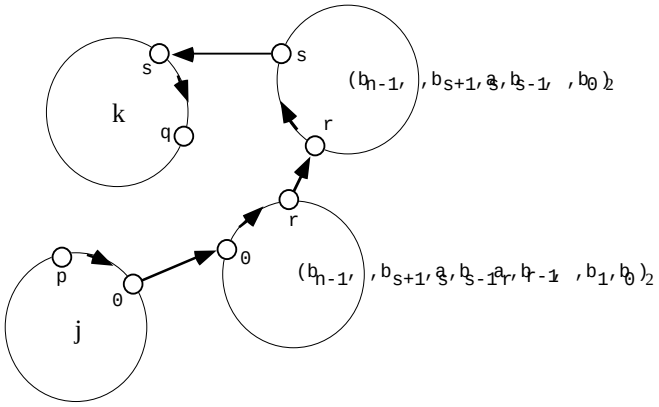
**cube connected cycles**



Es seien die Knoten $j = (p_{m-1}, \ldots, p_0)_2$ und $l = (q_{m-1}, \ldots, q_0)$ zu verbinden und sei o. B. d. A. $p_i = q_i$ für alle $i$ bis auf $r, s, t$. Sei der Ausgangspunkt des Prozessors $j \cdot m + i$. Dann erfolgt die Verbindung

$j \cdot m + i \rightarrow j \cdot m + r$ im Knoten $j$

von dort aus über $cube_r(j)$ zum Prozessor $cube_r(j) + r$ weiter innerhalb des Knotens $cube_r(j)$ nach $cube_r(j) + s$ und mit $cube_s(cube_r(j))$ zum Knoten $cube_s(cube_r(j))$ und dem Prozessor $cube_s(cube_r(j)) + s$. Innerhalb des Knotens zum Prozessor $cube_s(cube_r(j)) + t$. Dann erreicht man den Zielknoten $l = cube_t(cube_s(cube_r(j)))$. Der maximale Abstand sind pro Knoten $m/2$ Schritte und $m$ Anwendungen der Funktion $cube_i(j)$, aber nicht beides zugleich.




Bei geschickter Organisation kann die Zahl der Schritte auf den Knoten beschränkt werden auf insgesamt $m$ Schritte mit $m$ Anwendungen von $cube_i(j)$.
Dazu ist der Weg richtig zu wählen. Dann ist $n_{max} = 2m$, um $m \cdot 2^m$ Prozessoren zu verbinden.

### *5.3.7.2. Cube-Connected-Complete-Graphs (CCCG)*

Verbindet man die $m$ Prozessoren in einem Knoten eines Hypercubes der Dimension $m$ nicht als Ring sondern als vollständigen Graphen, dann spart man die Schritte von $r \rightarrow s$ beim Übergang innerhalb eines Knotens.

Der maximale Abstand ändert sich nicht $n_{max} = 2m$, wohl aber der Abstand im Mittel: sind nur $r$ Bits zu ändern beim Übergang von $j \rightarrow j'$, braucht das auch nur $2r$ Schritte.

## 5.3.8. Vergleich der Netzwerke

Es soll untersucht werden, wieweit sich einfache Netzwerke mit homogenen Netzwerken mit $n_{max} \sim ld\ N$ simulieren lassen.

*5.3.8.1. PM 2I als homogenes Netz*

PM 2I enthält die Netze

- lineare Kette : $j \rightarrow j \pm 1 = j \pm 2^0$
- Hypercube : $j \rightarrow cube_i(j)$

$j = (p_{m-1}, \cdots p_0)_2$
$p_i = 0 \Rightarrow cube_i(j) = j + 2^i$
$p_i = 1 \Rightarrow cube_i(j) = j - 2^i$

PM 2I simuliert ein quadratisches Gitter in 2 Schritten; $N = 2^m$ mit $m$ gerade

$j \rightarrow j \pm 2^{m/2} \mod \sqrt{N} \mathrel{\hat{=}} j \pm 2^{m/2}$ (1 Schritt)

$j \rightarrow (j + 1) \mod \sqrt{N}$ : if $j + 1 = j' = r \cdot \sqrt{N}$ then $j' \rightarrow j' - 2^{m/2}$ (2 Schritte)

$j \rightarrow (j - 1) \mod \sqrt{N}$ : if $j = r \cdot \sqrt{N}$ then $j \rightarrow j - 1 = j'$ ; $j' \rightarrow j' + 2^{m/2}$ (2 Schritte)

PM 2I simuliert einen binären Baum in $m$ Schritten:

$j \rightarrow 2j + 1$ ; $j = (p_{m-1}, \cdots, p_0)_2$

oder

$$j \rightarrow 2j \quad ; \quad 2j = j + \left( \sum_{i=0}^{m-1} p_i \, 2^i \right)$$

Die Summenbildung erfordert nicht mehr als $m$ Schritte im Netzwerk PM 2I.

*5.3.8.2. Hypercube als homogenes Netz*

Der Hypercube enthält die lineare Kette vermöge der Interpretation von $j$ als Gray-codierte Zahl $z$ und suche nach $z + 1$ (s. 5.3.6.2.).

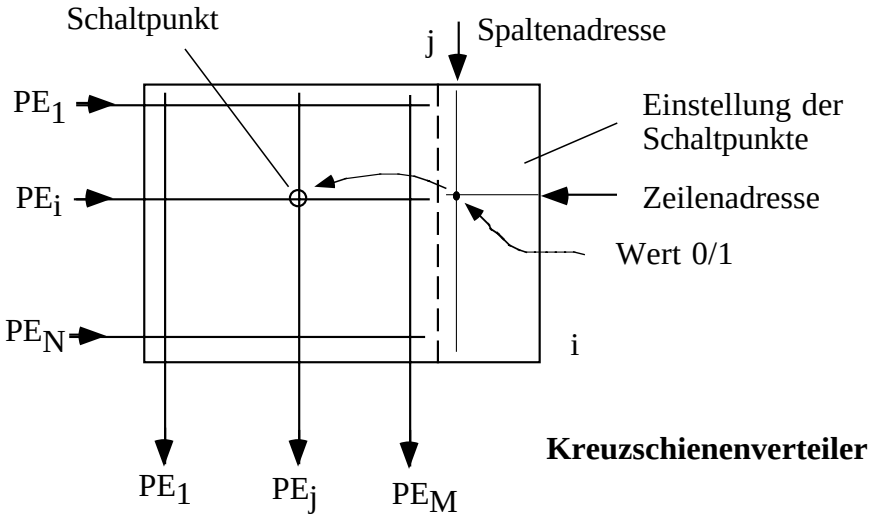
Der Hypercube simuliert ein quadratisches Gitter in $m$ Schritten.

Der Hypercube simuliert einen binären Baum in $m$ Schritten.

## 5.4. Geschaltete Verbindungsnetzwerke

### 5.4.1. Kreuzschienenverteiler (KSV)

Sie stellen unidirektional Verbindungen her zwischen N Eingängen und M Ausgängen.




Im Prinzip können auf einen Eingang i mehrere Ausgänge j aufgeschaltet sein. Üm eindeutig zu sein, wird man aber sicherstellen müssen, daß ein Ausgang j nur auf einen Eingang i geschaltet ist.

Im wichtigen Sonderfall M=N ist die Anzahl der Schalter $N^2$ und man realisiert eine Permutation: $\begin{pmatrix} 1 & .... & N \\ j_1 & .... & j_N \end{pmatrix}$

Jedem Eingang i ist genau ein Ausgang $j_i$ zugeordnet.

Für den Einsatz eines Kreuzschienenverteilers als Verbindungsnetzwerk ist die Verbindung $P_i \rightarrow P_i$ ausgeschlossen und die Anzahl der Schaltpunkte vermindert sich auf $N^2 - N$ .

Die Größe eines KSV ist weniger bestimmt durch die Anzahl der Schaltpunkte als vielmehr durch die begrenzte Anzahl von Pins des VLSI-Schaltkreises, der einen KSV realisiert.

Neben N Eingängen und N Ausgängen als KSV muß von außen die gewählte Permutation eingestellt werden können.

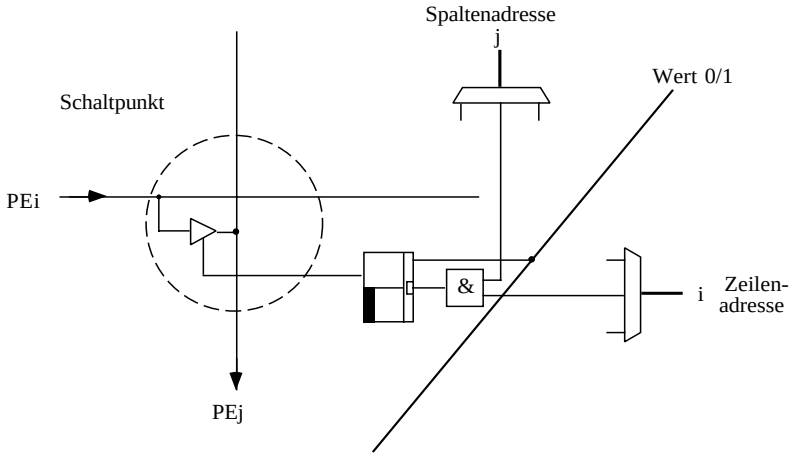
Die Beschreibung einer Permutation erfordert $N \cdot ld\ N$ Bit. Für den Zweck der Verbindung von PE's wird es nur eine kleine Anzahl von Permutationen geben, die einzustellen sind, z. B. $j_i = (i + 1) \bmod N$ oder $j_i = (i + 2^k) \bmod N$ oder $j_i = cube_k\ (i)$ .

Die Auswahl erfolge durch k Bit, mit denen eine von $2^k$ Permutationen ausgewählt werden kann.

Die Auswahl spricht Speicher auf dem Chip an, in denen die Schaltinformation steht.

Zum Laden einer Permutation kann die Schaltinformation nacheinander in N Schritten mit N Bit Wortlänge in den Speicher der Schaltpunkte eingebracht werden. Der Schaltpunktspeicher an Bord des Chip enthält dann $2^k$ Matrizen von N x N Bit.

Das Einstellen einer Permutation geschieht ebenfalls in N Schritten: mit k wird die gewünschte Schaltmatrix aderessiert, mit READ das Einlesen aktviert und mit N Takten die Matrix spaltenweise in die Latches bei den Schaltpunkten eingetaktet.

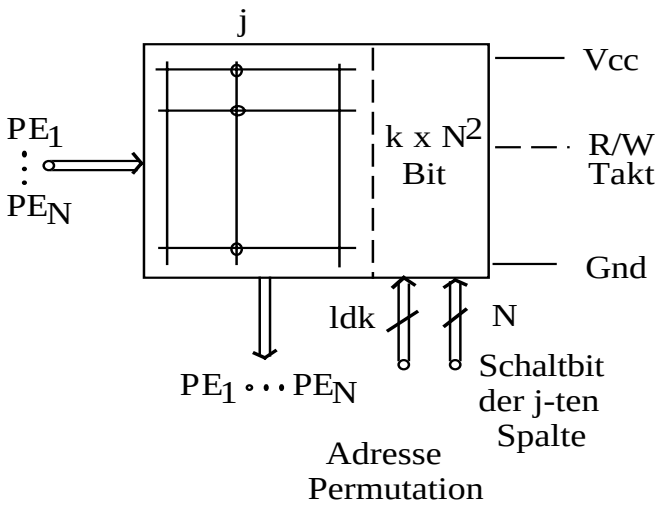



Der KSV hat somit 3 N Eingänge, einen Adresseingang von k Bit, eine Schreib-Leseleitung , die Versorgungsleitungen $V_{cc}$ und $G_{nd}$ und einen Takteingang zum Schieben der eingelesenen Information.

Beispiel: Pinanzahl 400 (pin-grid-array)
N = 128 $\Rightarrow$ 3 x 128
k = 8 (256 mögliche Permutationen)
$\Rightarrow$ 374 + 8 + 4 = 386 Pins.

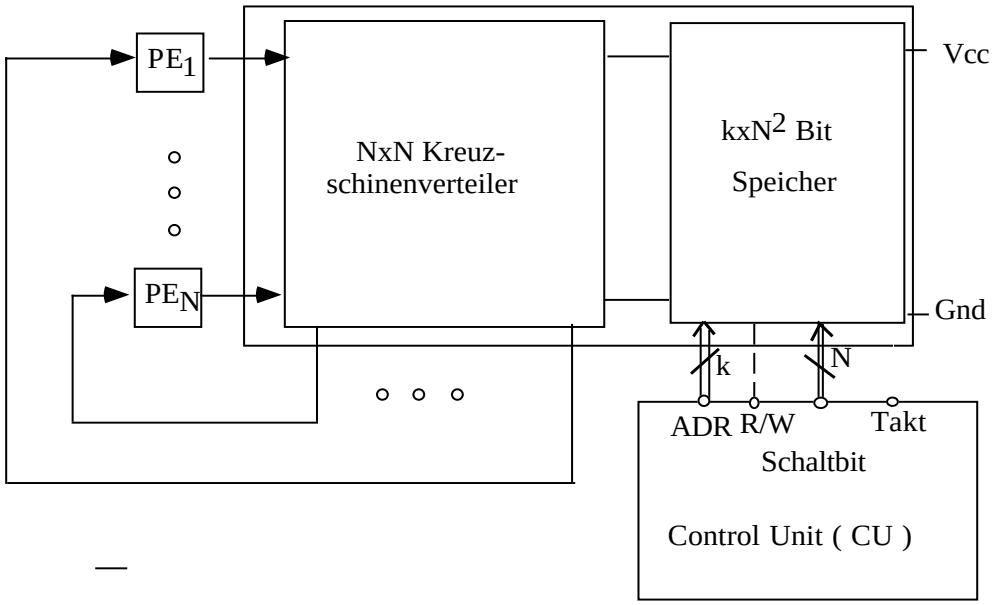
Das realisiert einen KSV mit 128 x 128 und 256 abrufbaren Permutationen. Die Wortbreite ist 1 Bit. Der Schaltpunktspeicher hat $2^7 \times 2^7 \times 2^8 = 2^{22} \approx 4$ Mio Bit.

Das nächste Bild zeigt die Realisierung eines N x N KSV.



Das Setzen der Schaltpunkte und das Einlesen einer neuen Permutation erfordert je N Schritte; sind die Schaltpunkte gesetzt, bleibt die Permutation erhalten. Die Umschaltung auf eine andere Permutation im Speicher dauert weniger als 10 µs, wenn der Auslesvorgang einer Spalte etwa 50 ns dauert. ( Takt = 20 MHz)

Die Zusammenschaltung mit PE's zu einem Feldrechner zeigt die nächste Abbildung, in der eine lineare Kette und ein binärer Baum als VNW simuliert wird.




lineare Kette $x_i := f( x_{i-1}, x_i, x_{i+1} )$ {*Kette zum Ring geschlossen*}

$x_{i'} := f_1( x_i, x_{\pi 1(i)} )$ mit $\pi_1(i) = i-1$

$x_i := f_2( x_{i'}, x_{\pi 2(i)} )$ mit $\pi_2(i) = i+1$

Binärer Baum $x_i := f( x_{2i}, x_i, x_{2i+1} )$ ( zusammenfassen )

$x_{i'} := f_1(x_i, x_{2i})$ mit $\pi_1(i) = 2i$

$x_i := f_2(x_{i'}, x_{2i+1})$ mit $\pi_2(i) = 2i+1$

$x_i := f( x_i, x_{\lfloor i/2 \rfloor} )$ ( verteilen )

$x_{i'} := f_1( x_i, x_{\pi 1(i)} )$ mit $\pi_1(i) = i \text{ div } 2$ für $i \bmod 2 = 0$,
$\pi_1(i) = 0$ sonst

$x_i := f_2( x_{i'}, x_{\pi 2(i)} )$ mit $\pi_2(i) = i \text{ div } 2$ für $i \bmod 2 = 1$
$\pi_2(i) = 0$ sonst

### 5.4.2. Delta Netzwerk

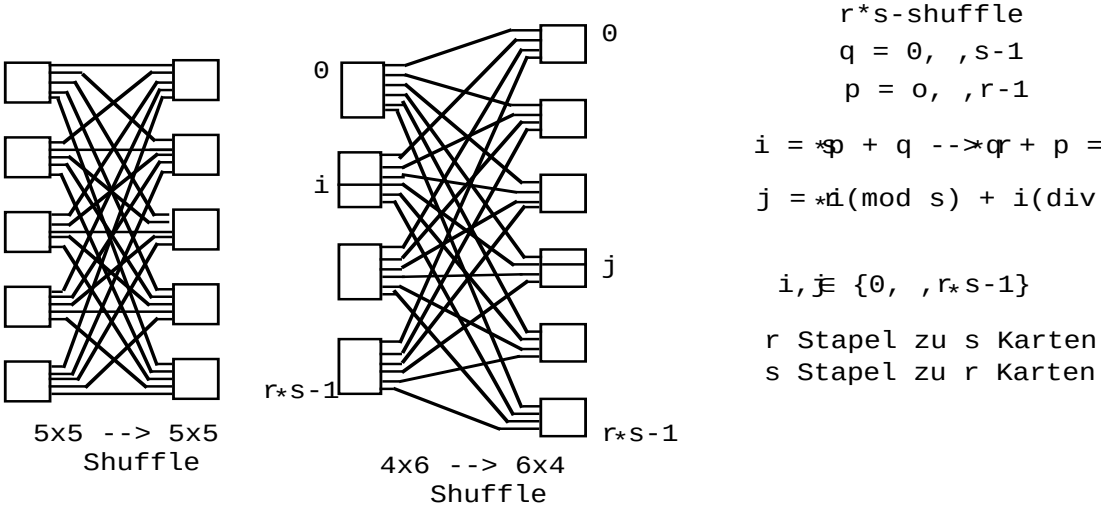
Während in einem N x N Kreuzschienenverteiler jede beliebige Permutation einstellbar und $2^k$ verschiedene abrufbar sind, ist die Anzahl N begrenzt. Bei größeren N wird man einen Verteiler aus kleinen KSV's zusammensetzen. Die Ausgänge der KSV's werden durch Shuffle-Netzwerke vermischt.

#### *5.4.2.1. r x s - Shuffle*

Sei $N = r \cdot s$ (r Stapel zu s Karten)
Sie werden verteilt auf s Stapel mit jeweils r Karten und es gilt

$S_{qxr}(i) = (s \cdot i) \bmod (r \cdot s - 1)$ für $0 \le i < r \cdot s - 1$
$= i$ für $i = r \cdot s - 1$

**Shuffle**



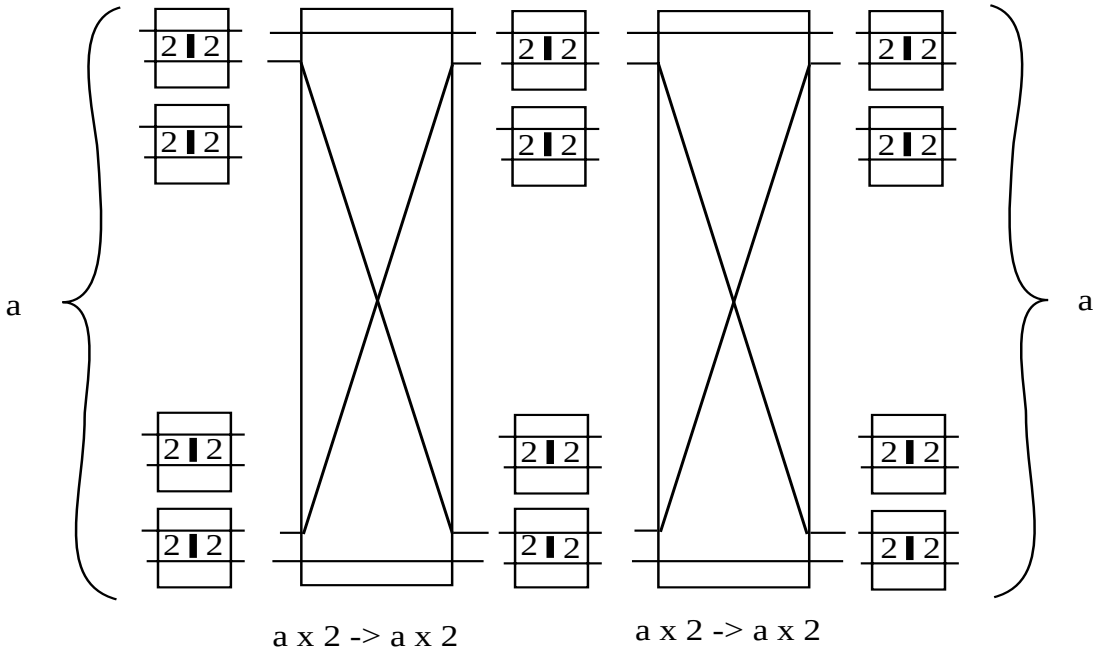

Shuffle-Netzwerke sind symmetrisch: $S_{qxr}(S_{rxq}(i)) = i$ für $0 \le i \le q \cdot r - 1$

#### *5.4.2.2. $a^n$ x $b^n$ - Delta-Netzwerk*

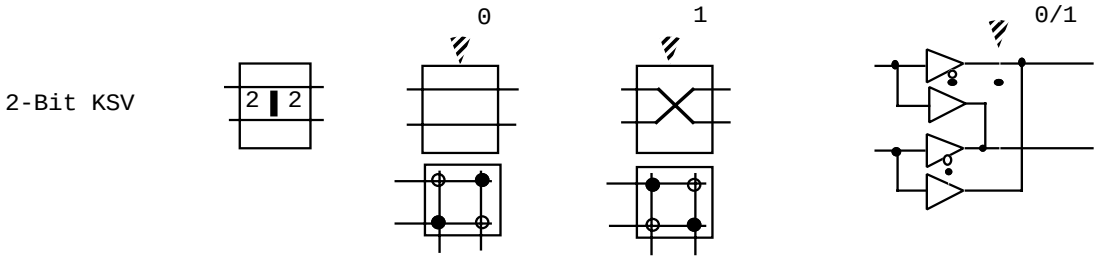
Ein $a^n$ x $b^n$ - Delta-Netzwerk realisiert einen N x N Kreuzschienenverteiler durch kleine Kreuzschienenverteiler.

Sei $N = a \cdot m$ . Dann werden kleine KSV's verwendet mit a Eingängen und b Ausgängen.

Sie werden über Shuffle-Netze zusammengehängt mit insgesamt n Stufen.

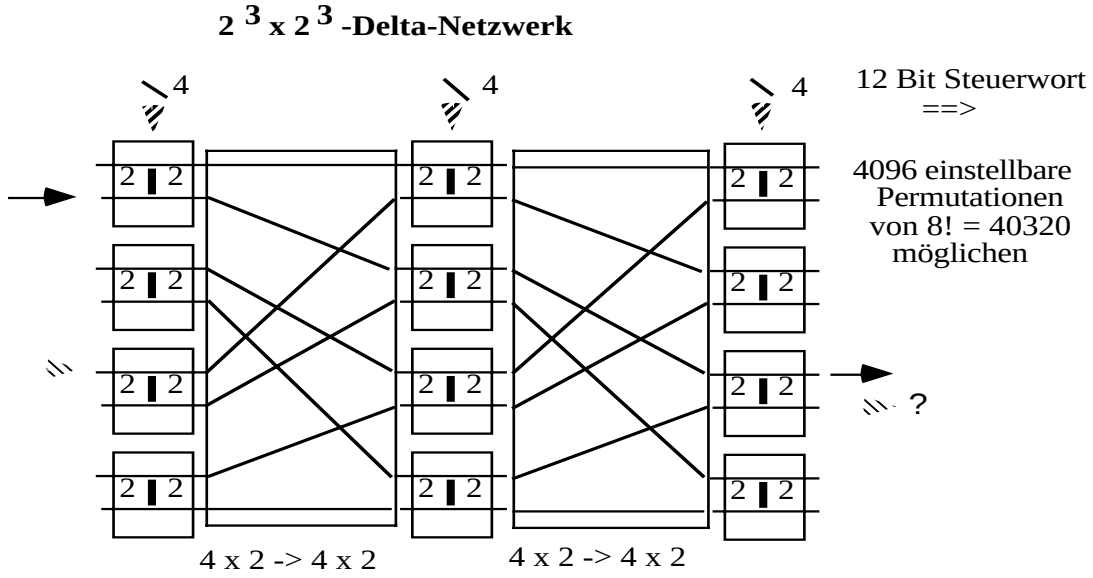



Besonders einfach sind 2 x 2 Kreuzschienenverteiler, die mit einem Bit umgesteuert werden können.




Sei $N = 2^n$ , dann ist das Netzwerk ein $2^n$ x $2^n$ - Delta-Netzwerk. Es enthält $n \cdot 2^{n-1}$ Kreuzschienenverteiler und n-1 Shuffle-Netzwerke.

Beispiel: N = 8




Bei einem $2^n$ x $2^n$ - Delta-Netzwerk ist jede Verbindung einstellbar, aber nicht jede Permutation.

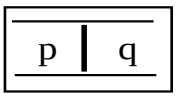
Das Netzwerk ist **nicht blockierungsfrei** .

Z. B. lassen sich nicht gleichzeitig verbinden $1 \rightarrow 4$ und $4 \rightarrow 5$ .

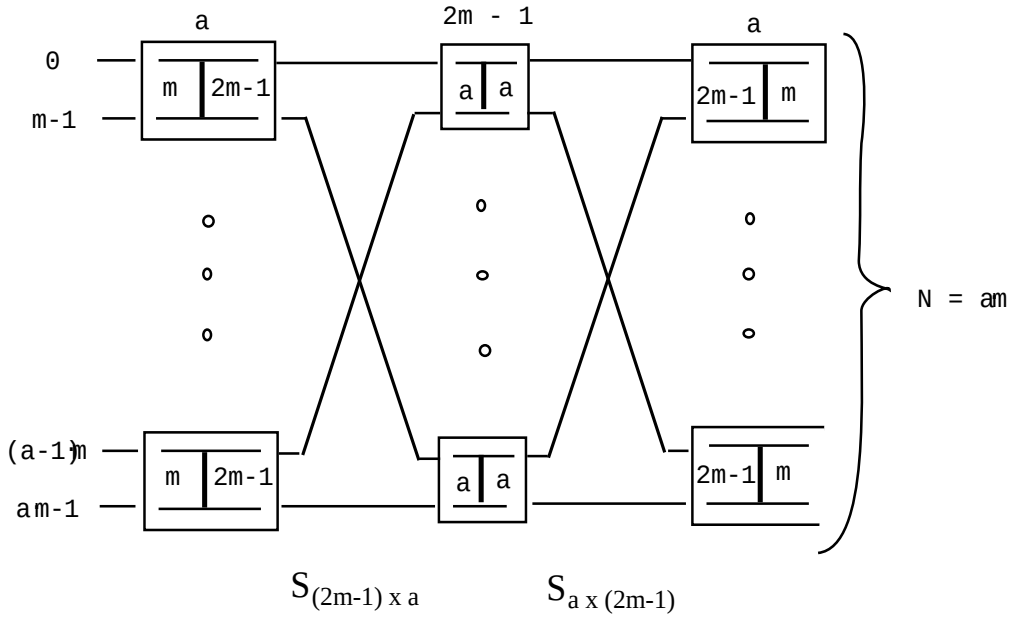
### 5.4.3. Clossches Netzwerk

Die Blockierungsfreiheit kann erreicht werden, indem man ausreichend viele mögliche Verbindungspfade im Inneren des Netzes vorsieht.

Sei ein Koppelfeld von p Eingängen und q Ausgängen. Es hat $p \cdot q$ Koppelpunkte und wird gekennzeichnet durch




Sei $N = a \cdot m$ . Dann hat der Eingang des Netzes a Koppelfelder mit jeweils m Eingängen. Haben diese Koppelfelder jeweils $2m - 1$ Ausgänge, dann läßt sich ein 3-stufiges Netz aufbauen, das blockierungsfrei ist.

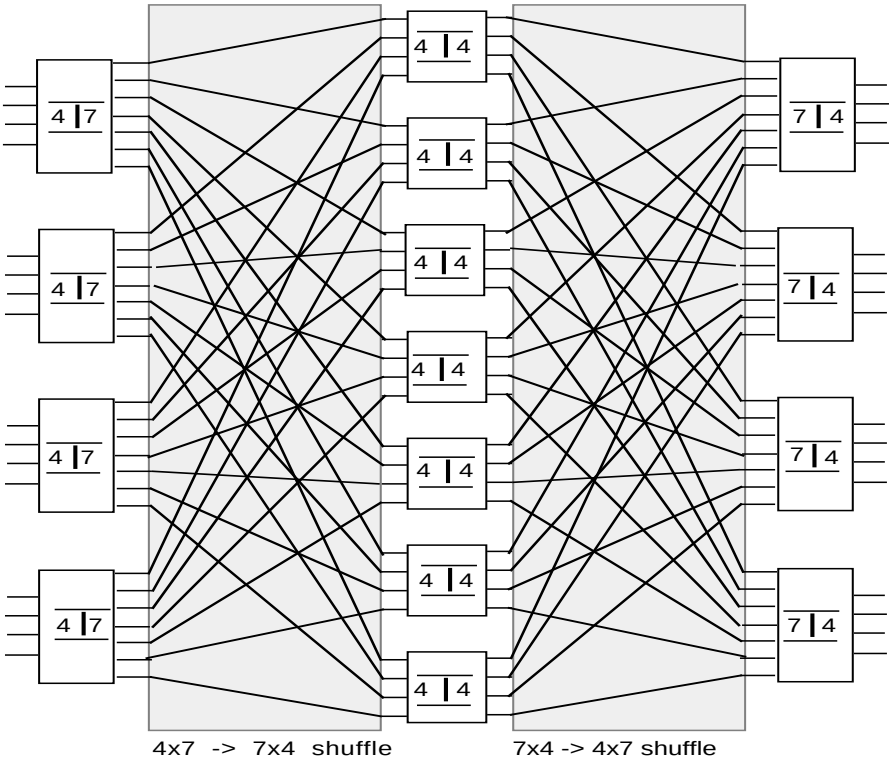



Sei $i = p \cdot a + q$ ein Eingang, der mit dem Ausgang $j = r \cdot a + s$ verbunden werden soll. Seien bei dem Eingangskoppelfeld $p$ alle Eingänge bis auf einen belegt, dann hat der Ausgang noch $2m - 1 - m - 1 = m$ freie Leitungen, die auf die $2m - 1$ Koppelfelder mit $a$ Eingängen und $a$ Ausgängen in der Zwischenstufe geschaltet werden können.

Die gleiche Argumentation gilt für den Ausgang $j$. Seien in dem gesuchten Koppelfeld $r$ alle Ausgänge bis auf $s$ belegt, dann können $m$ freie Leitungen von der Zwischenstufe auf das Koppelfeld $r$ gelegt werden.

Dann gibt es mindestens ein Koppelfeld in der Zwischenstufe, auf das von beiden Seiten her eine freie Leitung läuft, da $2m > 2m - 1$ .

Ein Beispiel zeigt ein clossches Netz mit 16 Eingängen und 16 Ausgängen.




Die Anzahl der Koppelpunkte eines N x N - Netzwerks ist mit $N = a \cdot m$

$$A = 2 \cdot m\,(2m - 1) \cdot a + a^2 (2m - 1)$$

$$A = (2 \cdot a \cdot m + a^2)(2m - 1) \;;\quad a = \frac{N}{m}$$

$$A = \left(\frac{N^2}{m^2} + 2N\right)(2m - 1) = f(m\,;N)$$

Es wird das Minimum für A gesucht bei festem N .

$$\frac{dA}{dm} = (2m-1)\left(-\frac{2N}{m^3}\right) + 2\left(\frac{N}{m^2} + 2N\right)$$

$$\frac{dA}{dm} = -4\frac{N^2}{m^2} + 2\frac{N^2}{m^3} + 2\frac{N^2}{m^2} + 4N$$

$$\frac{dA}{dm} = 0 \quad \Rightarrow \quad 2m^3 - N \cdot m + N = 0$$

$$\frac{m^3}{m-1} = \frac{N}{2}$$

Für große N ist m >> 1 und $\frac{m^3}{m-1} \approx m^2$

$\Rightarrow$

$$m = \left(\frac{N}{2}\right)^{1/2}$$

Setzt man $m = \sqrt{\frac{N}{2}}$ in A ein wird

$$A = (2m-1)(\frac{N^2}{m^2} + 2N)$$

$$A = \frac{2N^2}{m} + 4mN - \frac{N^2}{m^2} - 2N$$

$$A = 2 \cdot \sqrt{2} \cdot N^{3/2} + \frac{4}{\sqrt{2}} \cdot N^{3/2} - 2N - 2N$$

$$A = 4 \cdot \sqrt{2} \cdot N^{3/2} - 4N \quad ; \quad N >> 1$$

$$\Rightarrow$$

$$\boxed{A \approx 4 \cdot \sqrt{2} \cdot N^{3/2}}$$

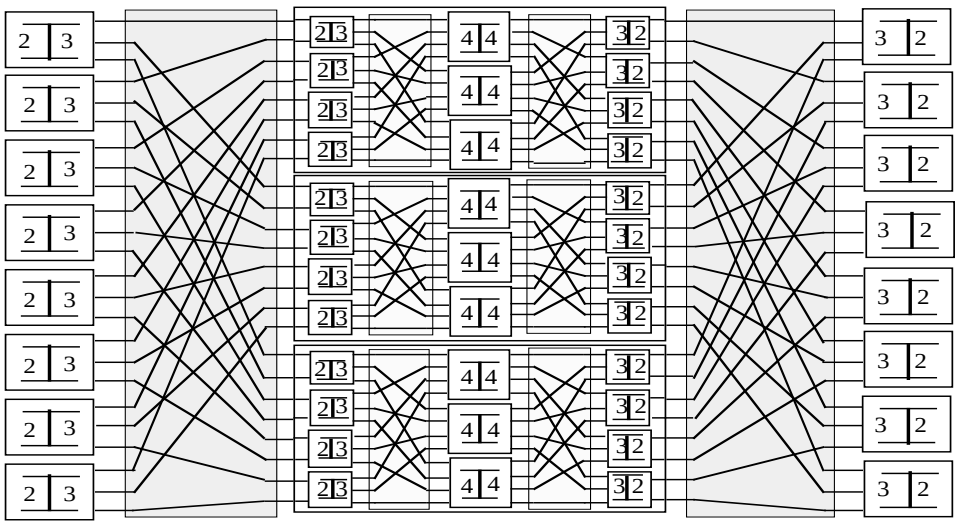
Damit spart man Schaltpunkte gegenüber einem N x N - Koppelnetzwerk.

(Das war die ursprüngliche Motivation für Clos, da Schaltpunkte seinerzeit durch Relais realisiert wurden und entsprechend teuer waren.)

(C. Clos; A Study of Non-Blocking Switching Networks
Bell Syst. Tech. J. **32** (1953) pp. 406 - 424)

Ein Beispiel für eine 5-stufige Anordnung zeigt die nächste Abbildung:
die a x a - Koppelfelder in der Zwischenstufe werden ihrerseits durch clossche Netze ersetzt.

**5 - stufiges Clos´sches Netz 16 x 16**

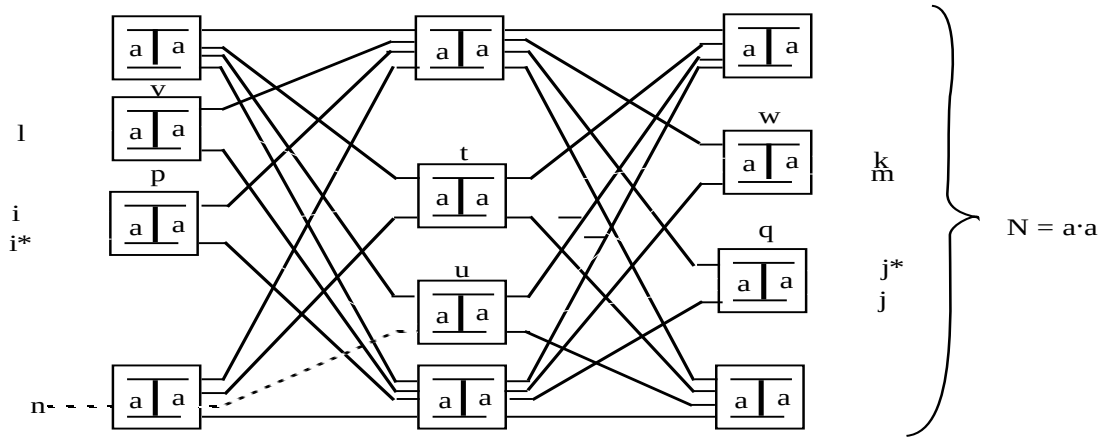


| | | | | | | |
|---|---|---|---|---|---|---|
| # | 8 | 12 | 9 | 12 | 8 | |
| Schalt-punkte | 48 | 72 | 144 | 72 | 48 | = 384 |

Um den Aufbau zu vereinfachen, baut man für $N = a^2$ Netzwerke der Art, daß man nur 3a KSV's mit a x a Ein/Ausgängen verwendet.

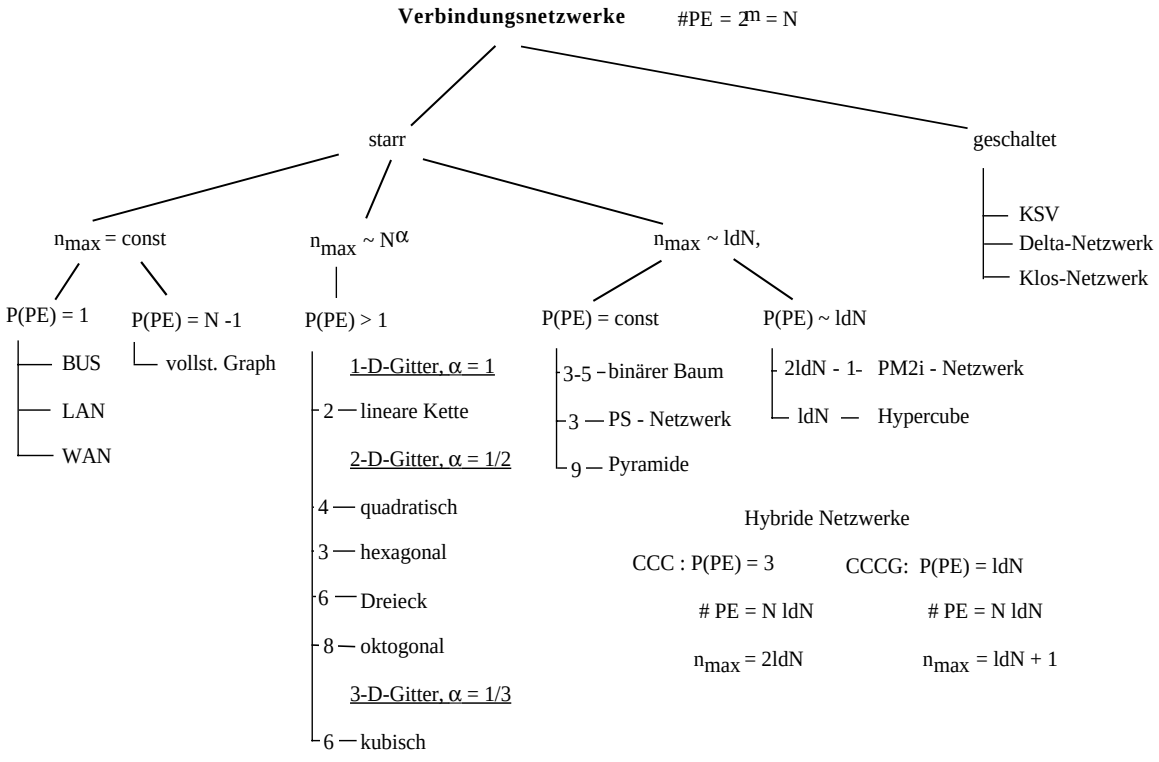
Man verliert den Vorteil der closschen Anordnung, daß unabhängig voneinander $i \rightarrow j$ geschaltet werden kann.

Da aber immer noch ein Eingang i mit einem Ausgang j über a verschiedene Wege verbunden werden kann, kann jede Permutation als Ganzes vorab gerechnet und eingestellt werden: die typische Forderung bei Parallelrechnern.




Ein solches Netzwerk mit a = 24 wurde als **Memphis-switch** in einem Rechner realisiert.

Eine Übersicht über die verschiedenen Verbindungsnetzwerke gibt das folgende Bild.



## 5.5. Beispiele von Feldrechnern

### 5.5.1. Klassifikation und Granularität

Man kann Feldrechner grob klassifizieren nach der Anzahl der PE's in einer Maschine und nach der Wortlänge: 4 - $10^6$ PE´s und 64 - 1 Bit Wortlänge. Viele Prozessoren großer Wortlänge ergeben sehr teure Maschinen.

Daneben kann man nach der Granularität klassifizieren:

viele einfache Prozessoren, verbunden über ein komplexes VNW
⇒ feine Granularität

wenige leistungsfähige Prozessoren, verbunden über ein einfaches VNW
⇒ grobe Granularität.

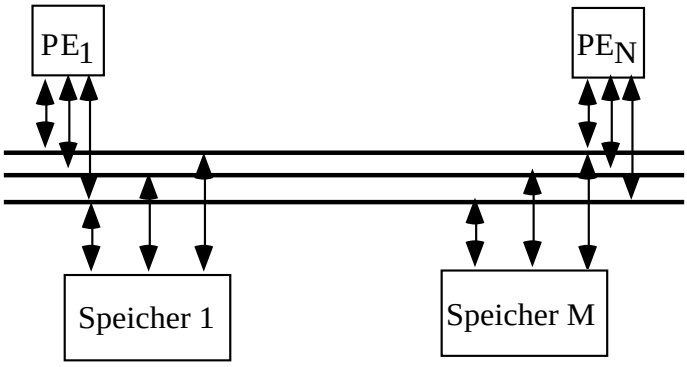
Welcher Typ von Rechner günstiger ist hängt von der Aufgabenstellung ab. Bei einer gegebenen Aufgabenstellung ist die Frage: wieweit läßt sich die Rechnung auf viele Rechner verteilen?

Wenn die vorhandene Anzahl von PE's etwas größer ist als die Anzahl der natürlicherweise parallel durchführbaren Operationen, sind Algorithmus und Rechner in ihrer Granularität aufeinander angepaßt.

Man kann die Rechner auch nach der Art des Zugriffs auf die Speicher klassifizieren:

Symmetric Multi Processing (SMP)

Die Rechner greifen über Busse auf Speicherbänke zu; der Speicher ist ein Shared Memory, d. h. es gibt einen gemeinsamen Adressraum.




Scalable Parallel Processing (SPP)

Die Prozessoren haben eigene Speicher und sind untereinander verbunden.

(Bus oder ein anderes Verbindungsnetzwerk)


Massively Parallel Processing (MPP)

Die Rechner haben eigene Speicher und die Verbindung geschieht über ein Nachrichtentransportsystem.


### 5.5.2. Multiprozessoranlagen

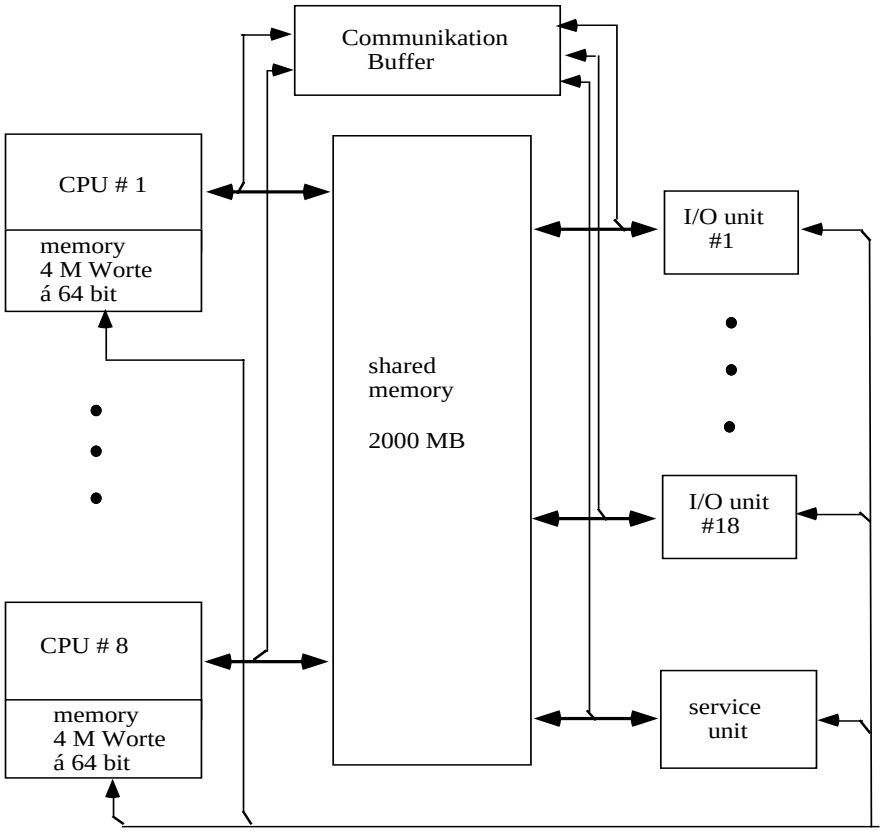
Wenige große Rechner (2 - 5) werden über Bussysteme zusammengeschlossen.

Beispiele: Cray YMP (4 Prozessoren)
Großrechner in Rechenzentren der 80er-Jahre

### 5.5.3. Feldrechner mit einigen Dutzend Prozessoren

Beispiel: ETA 10 (Control Data, 1987)

Die gemeinsame Verbindung geschieht über einen Speicher, der 8 CPU's zusammenbindet.




**ETA10**

Der Tradition der Maschinen von Control Data entsprechend hat das System bis zu 18 Input-Output-Prozessoren zum Verkehr mit der Außenwelt.

Die Wortlänge ist 64 Bit.

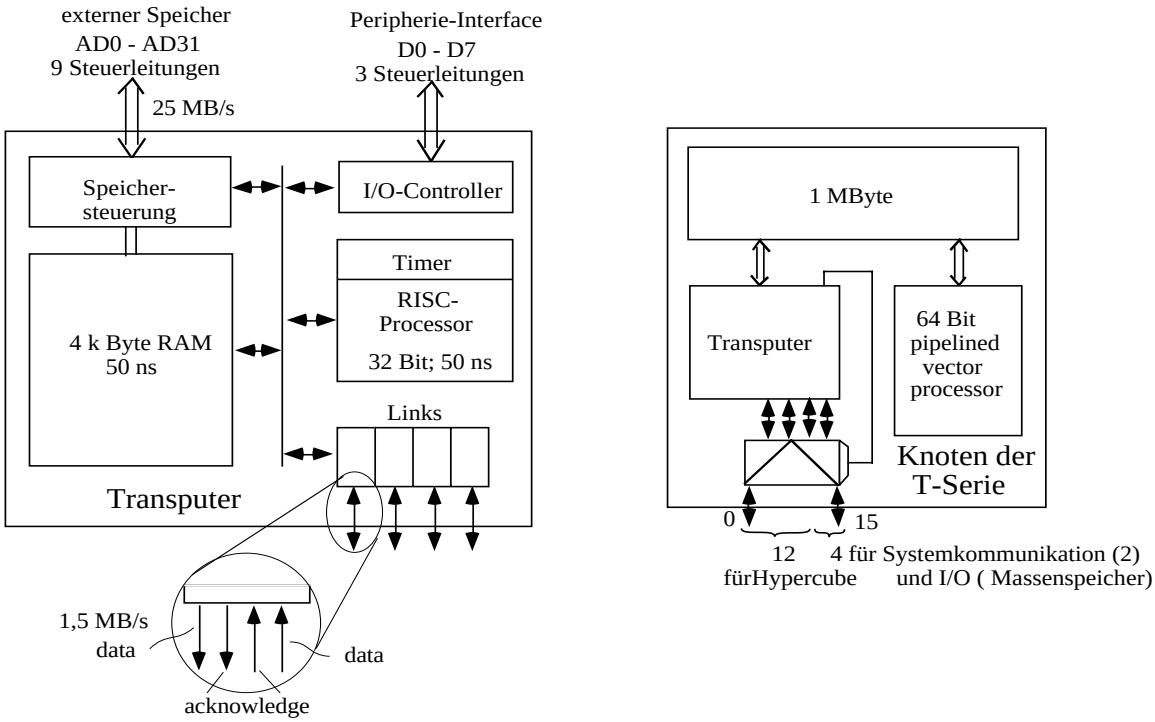
Der Communication Buffer dient als schneller Zwischenspeicher. Von den CPU's zum gemeinsamen Speicher können $9.1 \cdot 10^9$ Bit/s transportiert werden (8 x 64 Bit in 50 ns).

### 5.5.4. Rechner mit einigen Hundert Prozessoren

Hier spielt das Verbindungsnetzwerk schon eine dominierende Rolle.

Beispiel: T-Serie (Floating Point Systems) mit Transputern (Fa. Inmos)

Transputer sind Rechner auf einem Chip, die neben normalen Daten- und Adreßleitungen nach außen 4 serielle Schnittstellen haben, die Voll-Duplex arbeiten und unabhängig 4 Verbindungen zu Nachbarn aufbauen können.




In der T-Serie sind Transputer Teil der Knoten aus denen sich der Rechner zusammensetzt.

16 - 4096 Knoten können über einen seriellen 12-dimensionalen Hypercube miteinander verbunden werden.

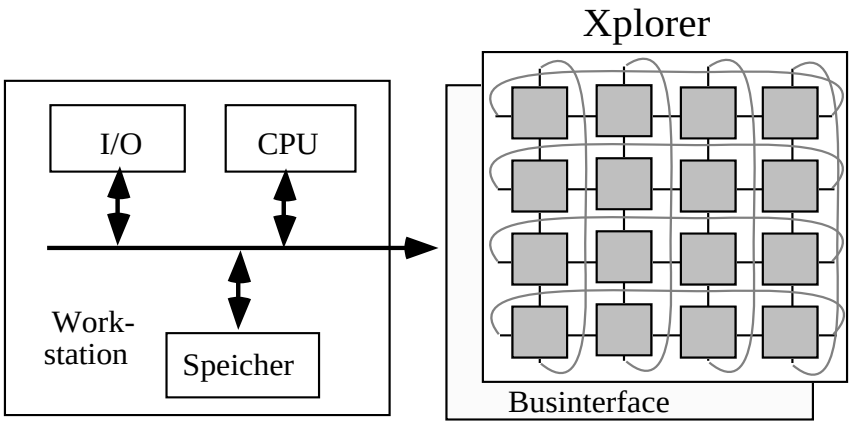
Das System wird in der Sprache OCCAM programmiert, die für Parallelarbeit und Rechnungen auf Vektoren angepaßt ist.

Beispiel: XPLORER (Parsytec, 1993)

Unter Verwendung von T9000 Transputern der Fa. INMOS (superskalar RISC-processor) mit 8 - 32 MByte per Knoten und 16 Knoten in einem Board erreicht das System 400 M Flops mit einer 50 MHz Clock.

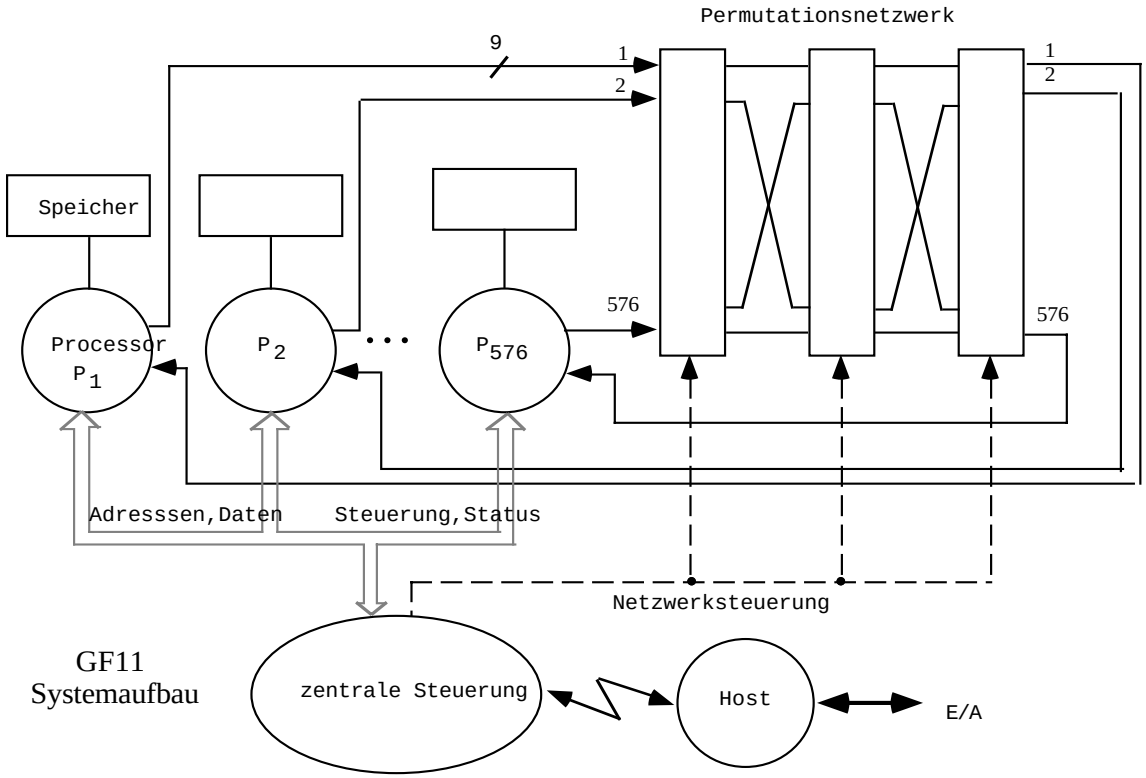
Die 16 Knoten sind in einem quadratischen Gitter (Torus) miteinander verbunden.

Intern hat das System mehr als 1000 MByte/s Kommunikationsbandbreite, nach außen können 16 Millionen Nachrichten/s übertragen werden und die I/O-Bandbreite ist 160 MByte/s.



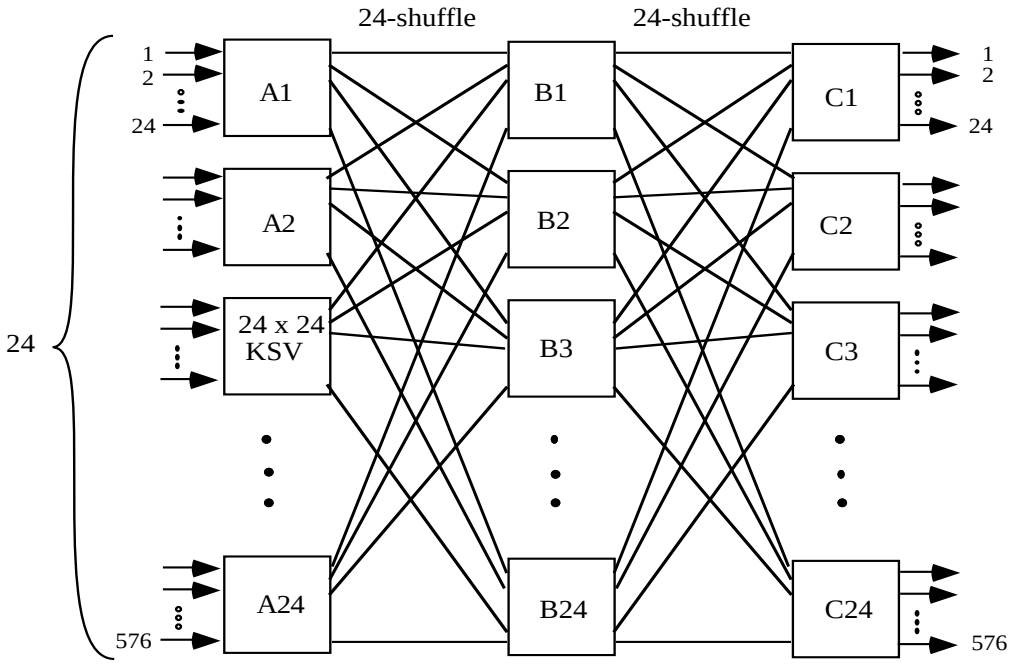

Beispiel: GF 11 (IBM, 1980)

In diesem Rechner wurde ein Permutationsnetzwerk eingesetzt, um die 576 PE's der Maschine miteinander zu verbinden.




Das Permutationsnetzwerk, der Memphis-switch, ist ein dreistufiges Netz aus 3 x 24 Kreuzschienenverteilern mit je 24 Ein- Ausgängen.

Das Netzwerk kann jede Permutation realisieren, aber eine Änderung ist nicht unabhängig durchführbar; ggf. müssen die Schalterstellungen für sehr viele KSV's neu gerechnet werden, um die Blockierungen aufzulösen, die bei einer Änderung entstehen.




Memphis-Switch

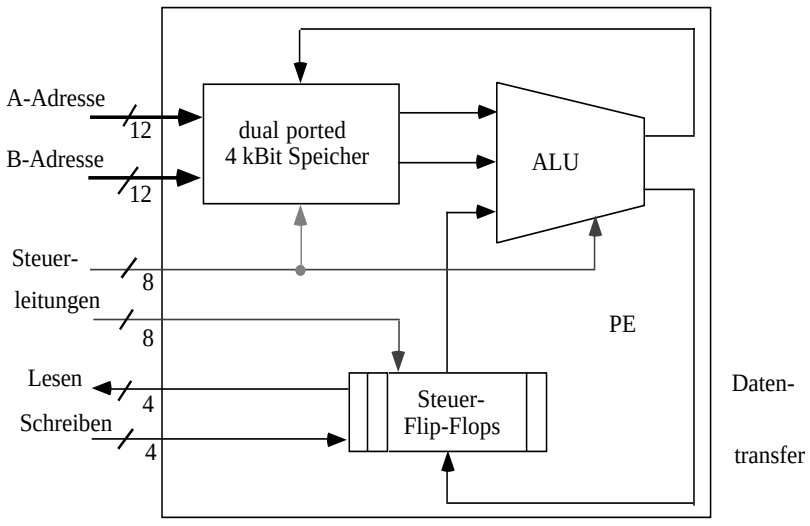
### 5.5.5. Rechner mit sehr vielen Prozessoren

Hier spielt nun das Verbindungsnetzwerk die dominierende Rolle.

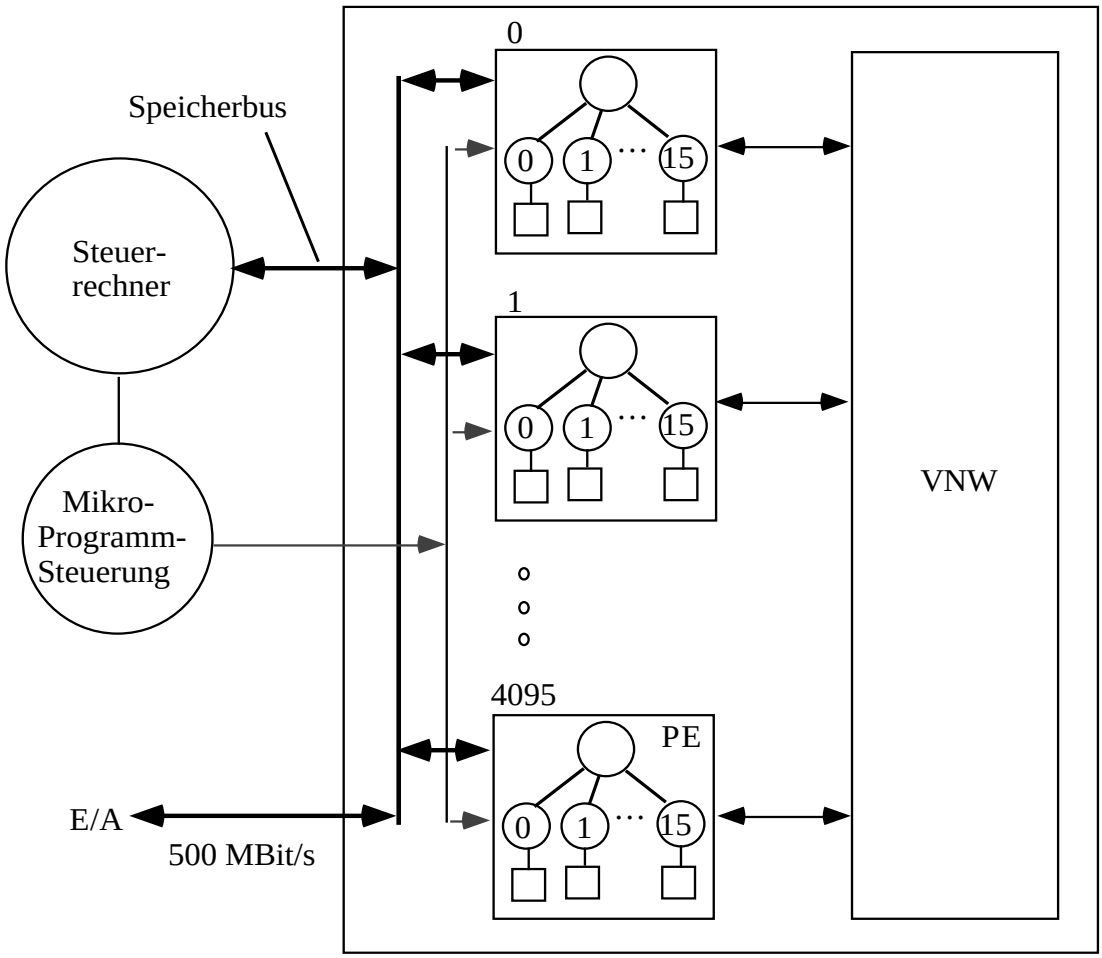
Die Tendenz geht hin zu leistungsfähigen Rechnern als PE's.

Beispiel: Connection Machine (Thinking Machines Corp, 1987)

Hier sind bis zu 64 k Prozessoren zusammengefaßt. Jedes Prozessorelement ist ein Ein-Bit-Rechner, der 4 k Bit Speicher an Bord hat und zwei Adressen parallel verarbeitet.



Die Zusammenschaltung der Rechner ist in zwei Stufen organisiert:
Je 16 PE's sind in einem vollständigen Graphen miteinander vernetzt und bilden einen Knoten der Connection Machine. Sie sind als ein separater Chip organisiert. Die Connection Machine besteht dann aus bis zu 4096 dieser Chips, einem 12-dimensionalen Hypercube als Verbindungsnetzwerk und einem Steuerrechner, der die Mikroprogrammsteuerung betätigt.




Beispiel: MASPAR MP-2 (MASPAR, 1992)

Die Anzahl der PE's ist hier $1 \leq N \leq 2^{14}$ .

Die PE's sind Risc-Prozessoren mit 16 kB oder 64 kB Speicher an Bord. Sie haben 40 32-Bit Register, und verarbeiten 36 oder 64-Bit IEEE-format Gleitkommazahlen oder Integer von 1-8-16-32-64 Bit.

Jedes PE hat Verbindung zu 8 Nachbarn (P(PE) = 8) und kann insgesamt 20 GByte/s übertragen.

Dazu hat jedes PE noch einen Daten- und Steuerpfad in die Array-Control-Unit (ACU).

Jedes PE kann seine Daten routen mit 1.3 GByte/s über die Router, die globale Verbindungen erlauben. Dabei gibt es einen Router-Kanal für je 16 PE's.

Die maximale Leistung ist 68.000 MIPS, 6.300 MFlops in Single Precision, 2.400 MFlops in Double Precision, im Vollausbau mit $2^{14}$ PE's.

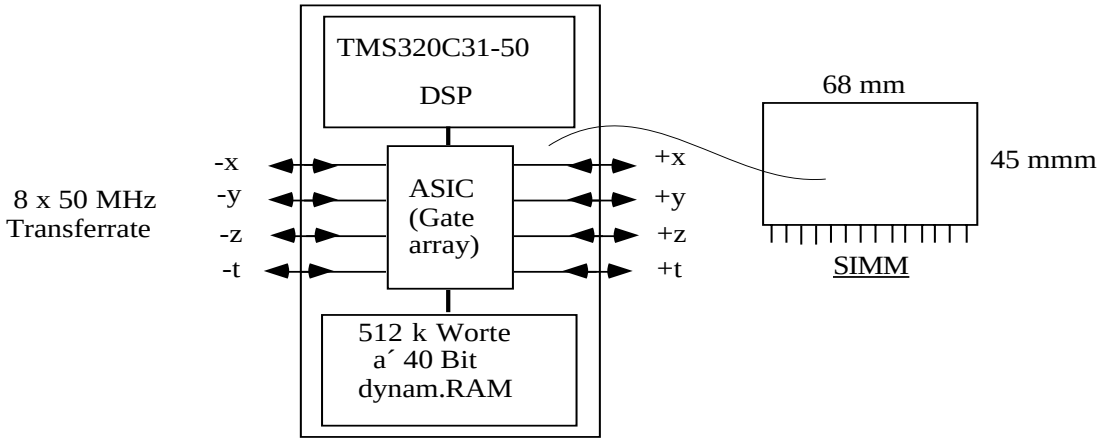
Die Leistungen der MP-2 im Vollausbau mit $2^{14}$ Prozessoren zeigt die Tabelle.

| **MP-2216** | |
|---|---|
| **Anzahl Prozessoren** | 16.384 |
| **Spitzenleistung** | |
| 32-Bit Integer MIPS | 68.000 |
| 64-Bit Integer MIPS | 34.000 |
| 32-Bit GK MFLOPS | 6.300 |
| 64-Bit GK MFLOPS | 2.400 |
| **Speicher** | |
| Kapazität ( MBytes) | 1.024 |
| Bandbreite direkt(MBytes/s) | 20.000 |
| Bandbreite indirekt(MBytes/s) | 7.800 |
| **Register** | |
| Kapazität(64-Bit Register) | 327.680 |
| Bandbreite (GBytes/s) | 800 |
| **2D-Torus mit 8-fach Verbindung zu nächsten Nachbarn** | |
| Bandbreite (MBytes/s) | 20.000 |
| **Globaler Router** | |
| Vebindungszeit ( µs) | 3 |
| Bandbreite senden (MBytes/s) | 1.300 |
| Bandbreite empfangen (MBytes/s) | 1.300 |
| **I/O Subsystem** | |
| I/O Controller Kapazität ( MBytes) | 8 |
| PE-I/O Controller BW Spitze (MBytes/s) | 64 |
| max. IORAM Kapazität (MBytes) | 1.024 |
| max. PE-IORAM BW (MBytes/s) | 1.024 |
| I/O-Kanal Spitzengeschw. (MBytes/s) | 200 |
| Parallel VME Spitzengeschw. (MBytes/s) | 16 |
| # I/O Slots | 14 |

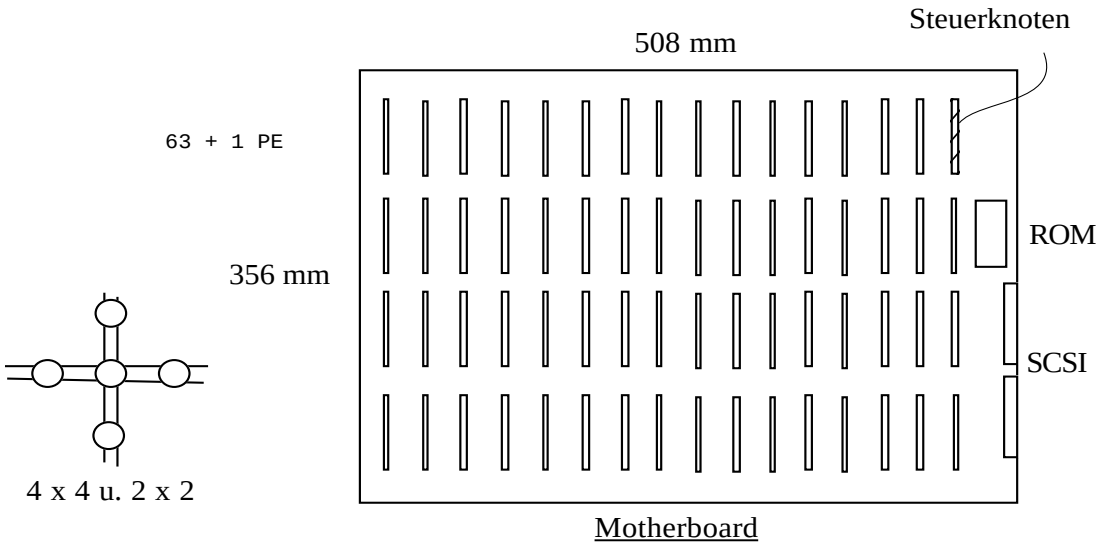
Beispiel: Columbia-Universitätsrechner (1999)

Ein recht leistungsfähiger Rechner mit DSP's als Prozessoren ist an der Columbia-Universität aufgebaut worden.

Jedes PE hat einen Speicher von 512 k Worten à 40 Bit als dynamischen RAM an Bord und 8 bidirektionale serielle Verbindungen nach außen mit 8 x 50 MHz Transferrate, aufgebaut auf einer Steckkarte in SIMD-Technik als SIMM-Baustein.




Je 63 PE's und ein Steuer-PE sind auf einem Board aufgebaut, das ROM an Bord hat und über zwei SCSI-Schnittstellen nach außen verfügt. Die PE's sind auf dem Board in zwei quadratischen Gittern miteinander verbunden.




8 Boards passen in einen Überrahmen, der Stromversorgung und Kühlung zur Verfügung stellt.

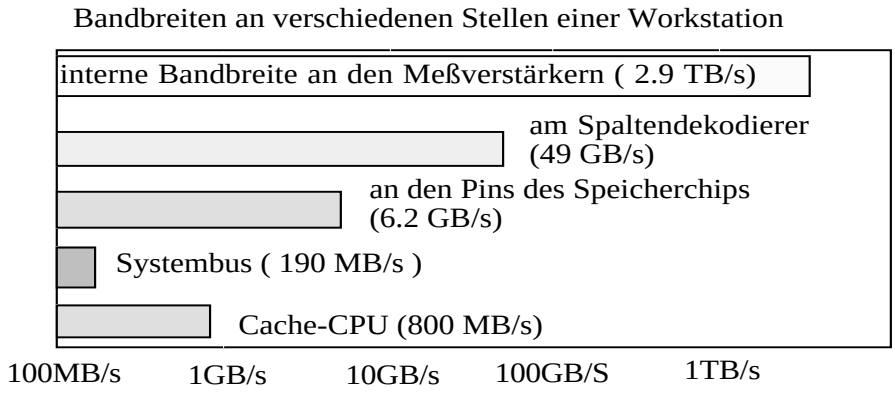
Mit 24 Überrahmen ist ein Rechner mit 12.288 PE's realisiert worden, der 600 GFlops leistet (Brookhaven, 1999).

Mit schnellen Standard-PC's als rechnenden Elementen kann man noch schnellere Systeme aufbauen: 1999 liegt der Rekord bei 7.500 Pentiumprozessoren, zusammengeschlossen zu einem ASCI-Red-Rechner (Advanced Scientific Computing Initiative), der im November 1998 31.568 GFlops leistet.

Literatur: TOP-500-Liste in: http://parallel.rz.uni-mannheim.de/top500.html

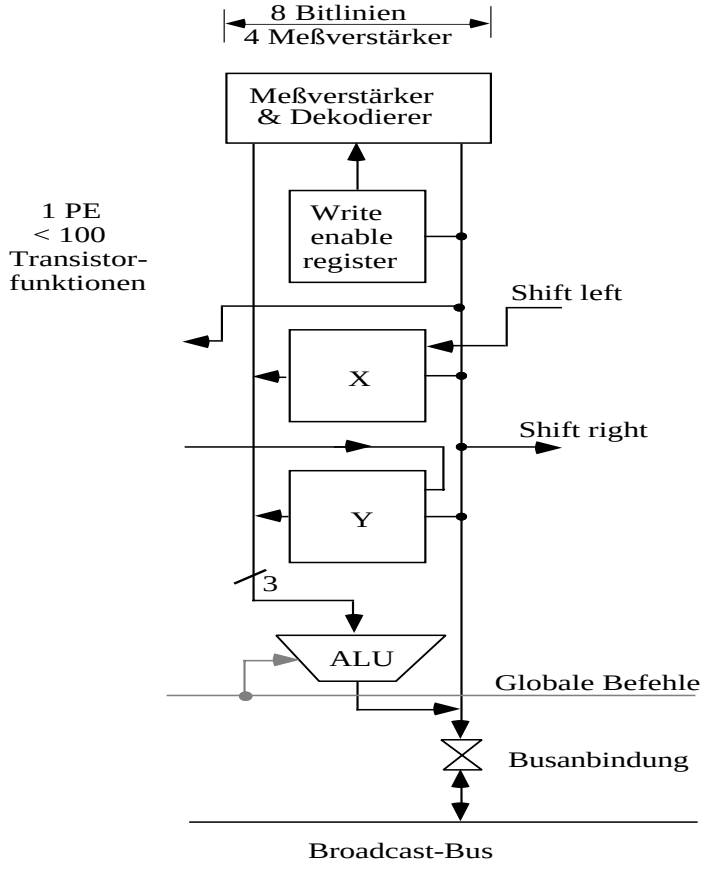
## 5.6. Computational RAM

Mit der Möglichkeit, viele 10 Millionen Transistorfunktionen auf einem Chip zu haben kann man die Frage nach SIMD-Rechnern umkehren: man ordne nicht PE's auch Speicher zu, sondern verzahne auf dem Chip Speicher und rechnende Einheiten. Ein Speicherchip hat intern sehr hohe Bandbreiten durch den parallelen Zugriff auf die Spalten einer Speichermatrix.

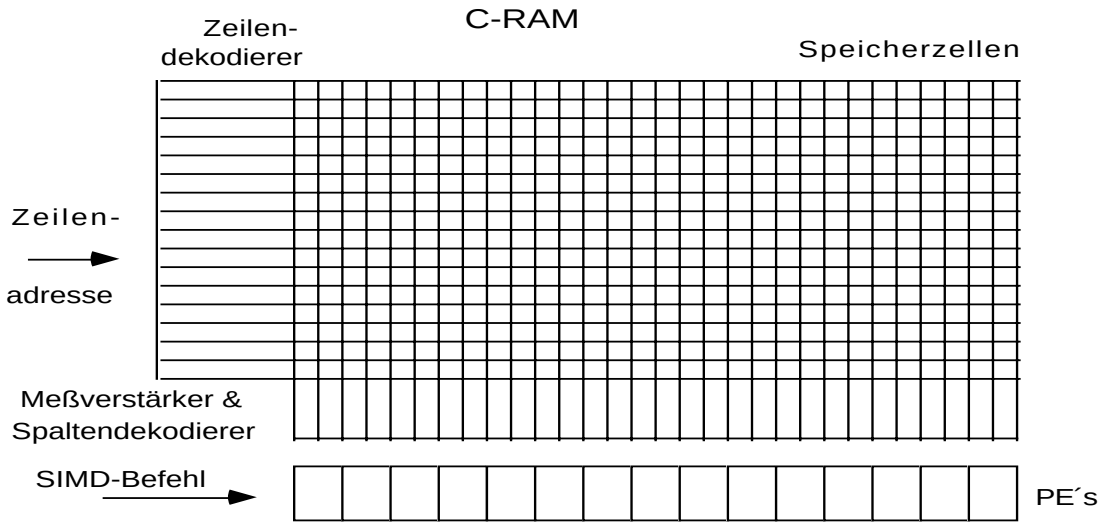



Noch an den Spaltendekodierern erreicht man einige GB/s.

Das kann man nutzen, um jeweils einige Spalten eines Speicherchips mit einem PE zu versorgen. Die PE's passen sich in ihrer Struktur an die Geometrie des Speichers an.



Der Speicherchip mit PE's hat dann eine Gestalt nach dem folgenden Bild.




Ein C-RAM mit 480 Bit Speicher/PE, 512 PE's auf dem Chip und einer Datenbreite von 1 Bit bildet den Baustein eines C-RAM-Rechners.

Simulation eines Rechners mit 32 MByte C-RAM aus 128 k rechnenden Einheiten mit einem (konservativen) 150 ns Takt gegenüber einer 70 MHz micro Sparc zeigt die große Geschwindigkeit dieses Ansatzes.

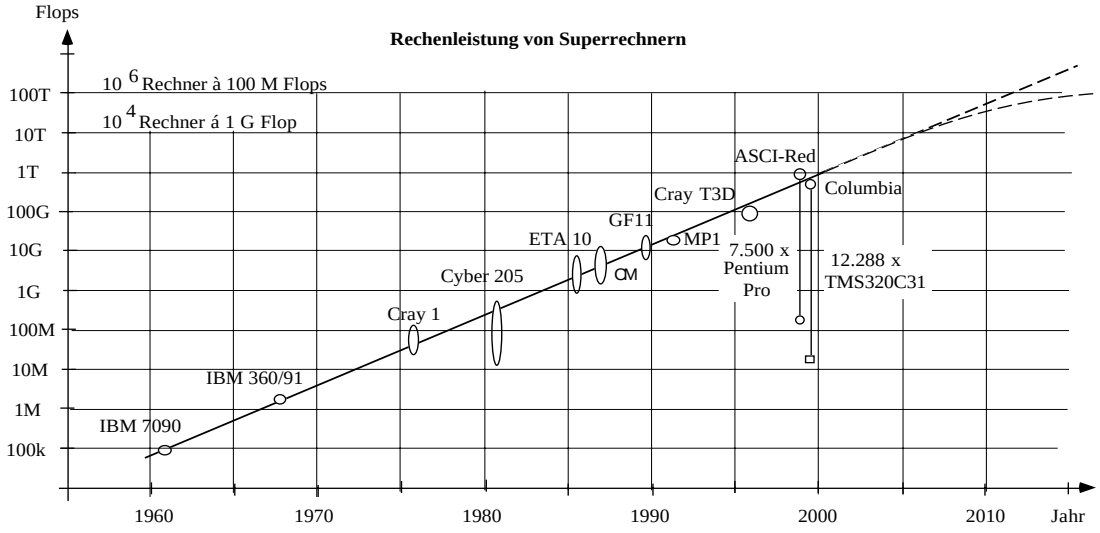
| **Programm** | **C-RAM Laufzeit** | **Sun Sparc Laufzeit** | **C-RAM Beschleunigung** | **Parallelismus** |
|---|---|---|---|---|
| Vektorquantisierung | 25,7 ms | 33,8 s | 1.312 | Bildraum |
| Maskierter blt | 18,2 µs | 443 ms | 24.310 | Bildraum |
| 3x3 Konvolution 16 M | 17,6 ms | 113 s | 6.404 | Bildraum |
| FIR 128 tap, 40 Bit | 99 µs | 312 ms | 3.144 | Koeffizienten |
| FIR 4M tap, 16 Bit | 1,04 ms | 5,14 s | 4.929 | Koeffizienten |
| LMS Matching | 0,20 ms | 251 ms | 1.253 | Records |
| Data Mining | 70,6 ms | 192 s | 2.742 | Raum der Regeln |
| Fehlersimulation | 89 µs | 3,9 s | 43.631 | Fehlerraum |
| Erfüllbarkeit | 23 µs | 959 ms | 41.391 | Lösungsraum |
| Löschen des Speichers | 1,6 µs | 8,8 ms | 5.493 | Speicher |

Literatur: D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, R. Mckencie
Computational RAM: Implementing Processors in Memory
IEEE Design & Test of Computers, pp. 32 - 41, JAN - MAR 1999

## 5.7. Vergleich von SIMD-Rechnern und Superrechnern

Die Rechenleistung von Parallelrechnern ist kontinuierlich gestiegen, vor allem durch die Parallelschaltung vieler Rechner.

Der Trend geht hin zu sehr vielen Maschinen relativ hoher Komplexität.




Die Tabelle vergleicht Maschinen, die kommerziell erfolgreich waren.

Die Darstellung ist halblogarithmisch: auf der Ordinate ist die Leistung logarithmisch aufgetragen. Die Gerade, die die Rechner verbindet bedeutet eine Exponentialkurve in der Realität.

Da eine e-Funktion divergiert, kann die Entwicklung nur noch wenige Jahre so weiterlaufen, dann muß sich die Kurve abflachen.