Non-Standard Rechner

Peter Marwedel
Informatik 12
TU Dortmund

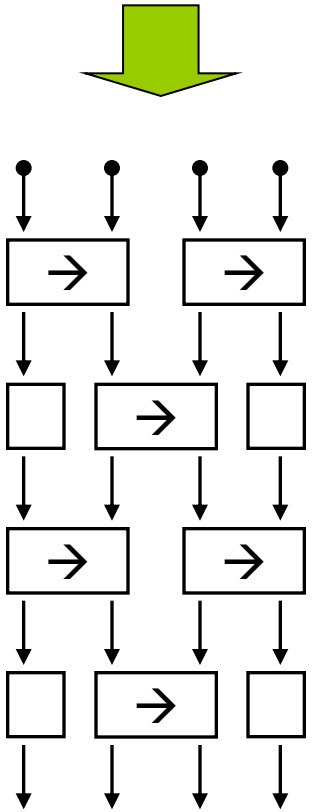
2014年04月29日

Datenflussprinzip: Systolisches Feld

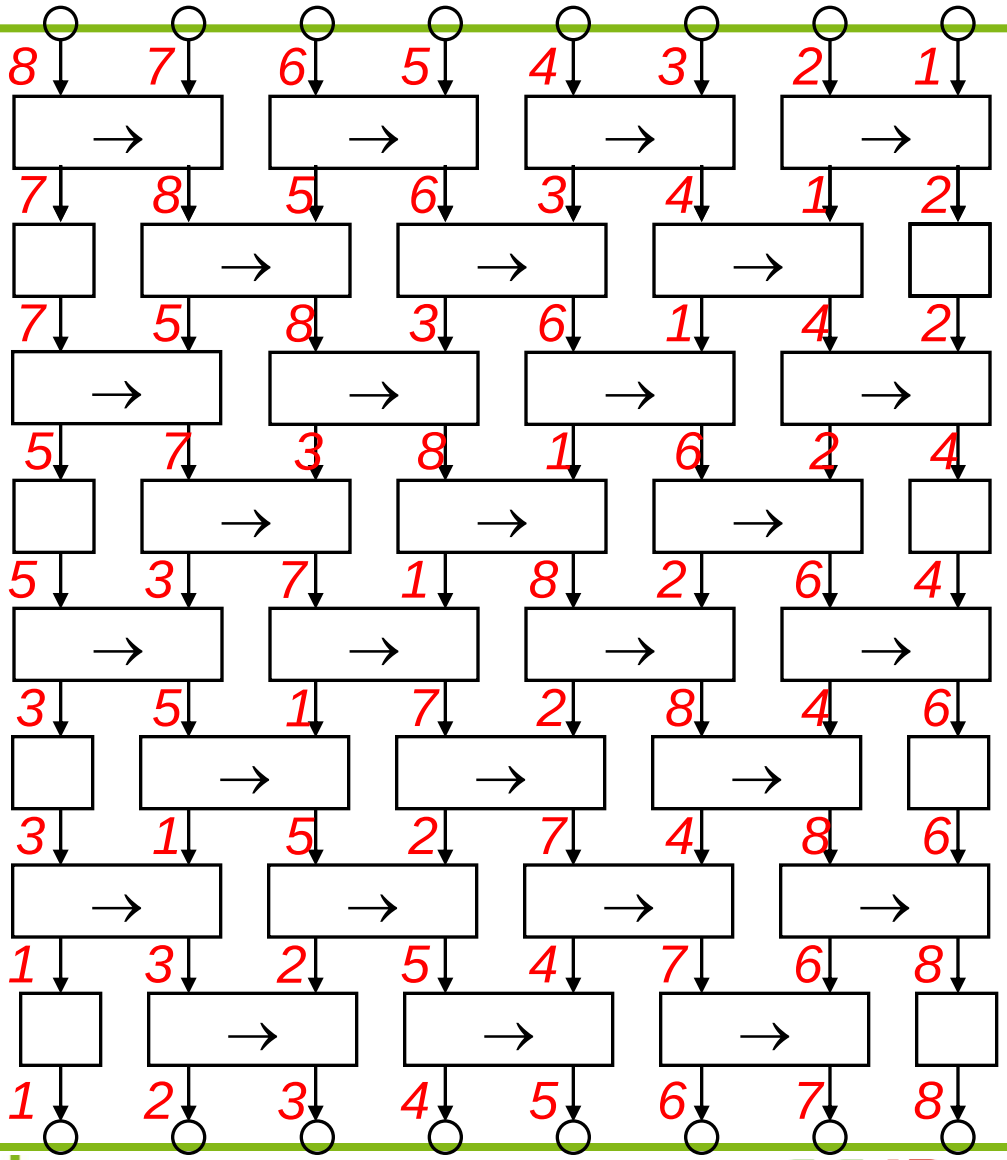
A systolic array is a special-purpose parallel device, made out of a few simple cell types which are regularly and locally connected. [☞ H.T. Kung]

Beispiel:

- Parallele Eingabe:
- Einfache Zellen („CondSwap“, „Nop“):
- Reguläre Struktur:



Beispiel für ein systolisches Feld: *odd-even transposition sort (OETS), n=8*

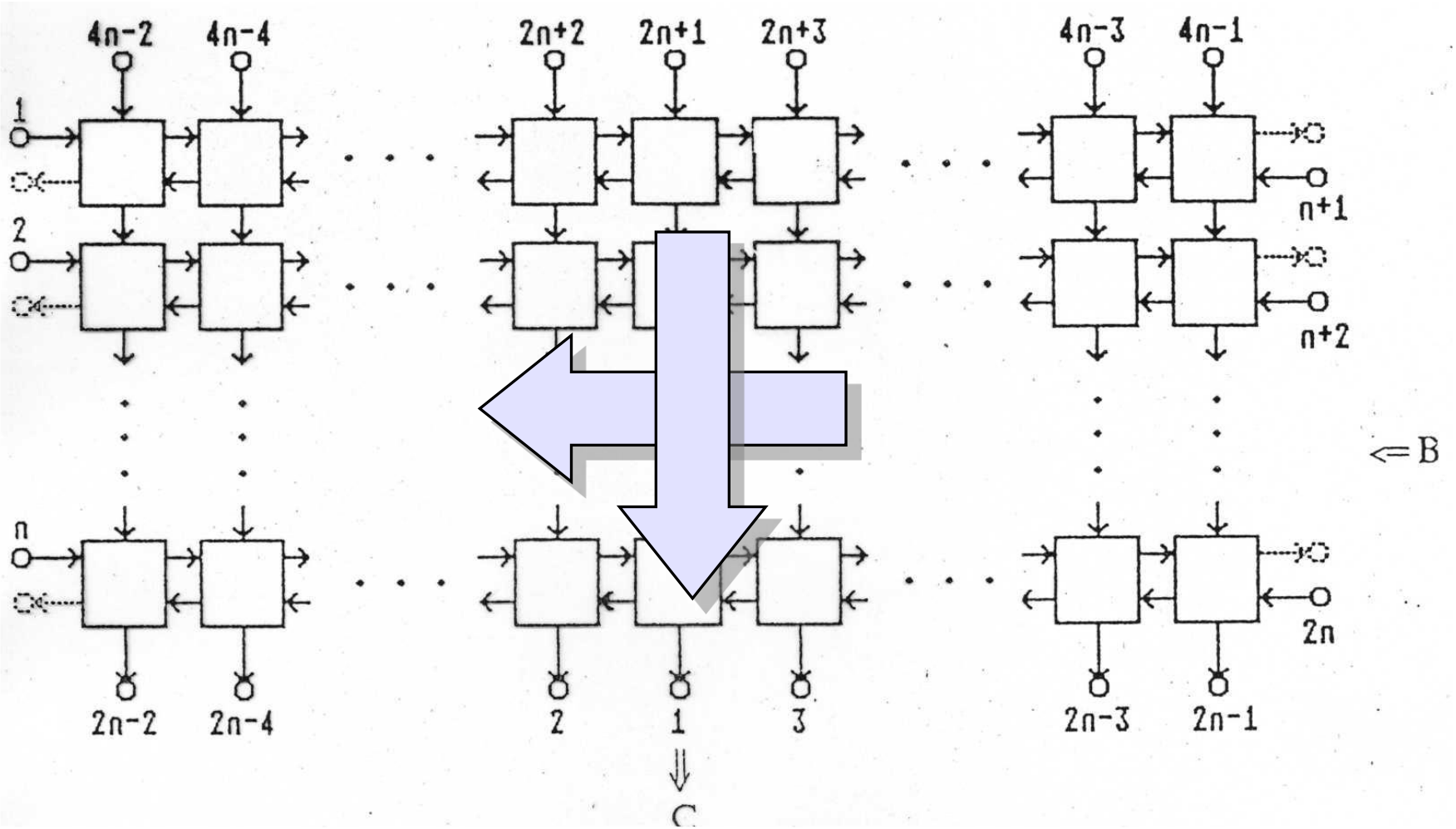


Daten werden synchron durch ein- oder zweidimensionale Felder "gepumpt" und dabei werden auf diesen Daten Berechnungen ausgeführt.

Prozessoren führen feste Befehle aus.

Sortieren mit Aufwand von n^2, Latenz n und Durchsatz $\approx O(1)$.

Behauptung:
systolisches Array für Matrixmultiplikation

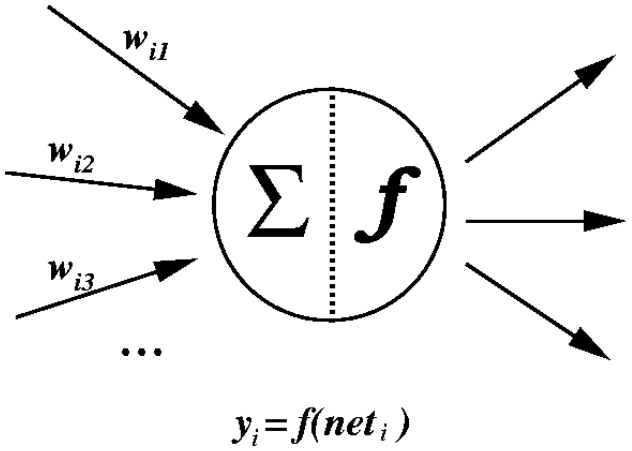


Wie kann man systolische Arrays systematisch konstruieren?

Neuronale Netze

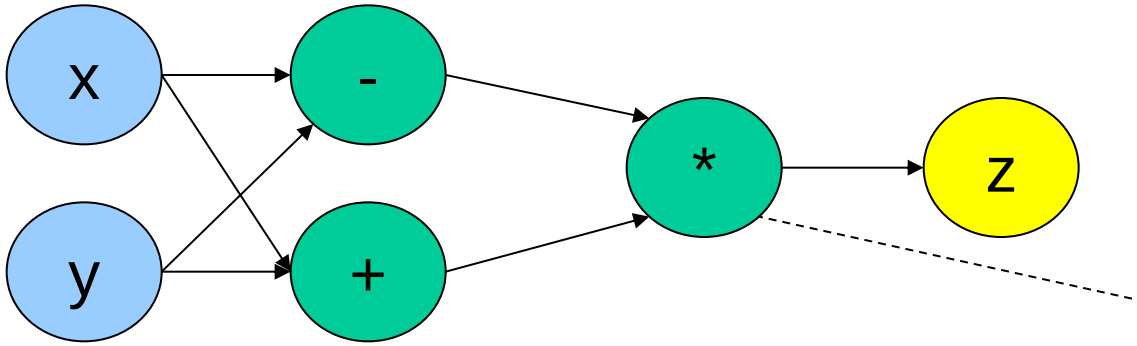
- Neuronale Netze emulieren Netze von Neuronen in Lebewesen
- Einsatz zur Klassifikation von Mustern und als nicht-lineare adaptive Filter
- Neuronale Netze erfordern eine Anlernphase zum Einstellen der Parameter
- Sind geeignet, wenn sonst wenig Erfahrung zur Lösung des Problems vorliegt.

$$y_i = f\left(\sum_j w_{ij} y_j\right)$$



2.9 Datenflussmaschinen

Verfügbarkeit von Daten veranlasst Ausführung von Operationen: Beispiel: **z = (x + y) * (x - y);**
x, y, z: Benennungen für Werte.

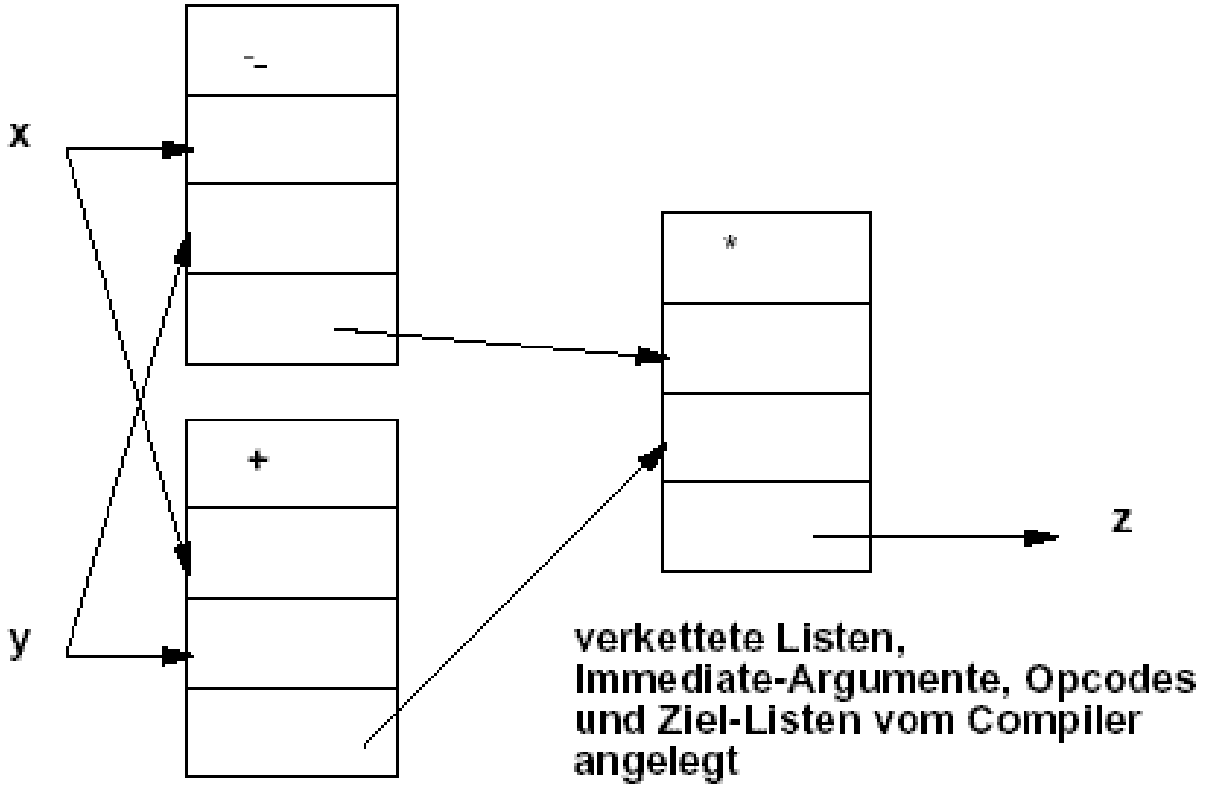


Die Operation beginnt, falls alle Argumente berechnet worden sind.

Modellierung mit Marken: Gültige Daten werden als Marken dargestellt. Eine Operation kann ausgeführt werden, falls alle ihre Argumente markiert sind. Potenziell viele Operationen gleichzeitig!

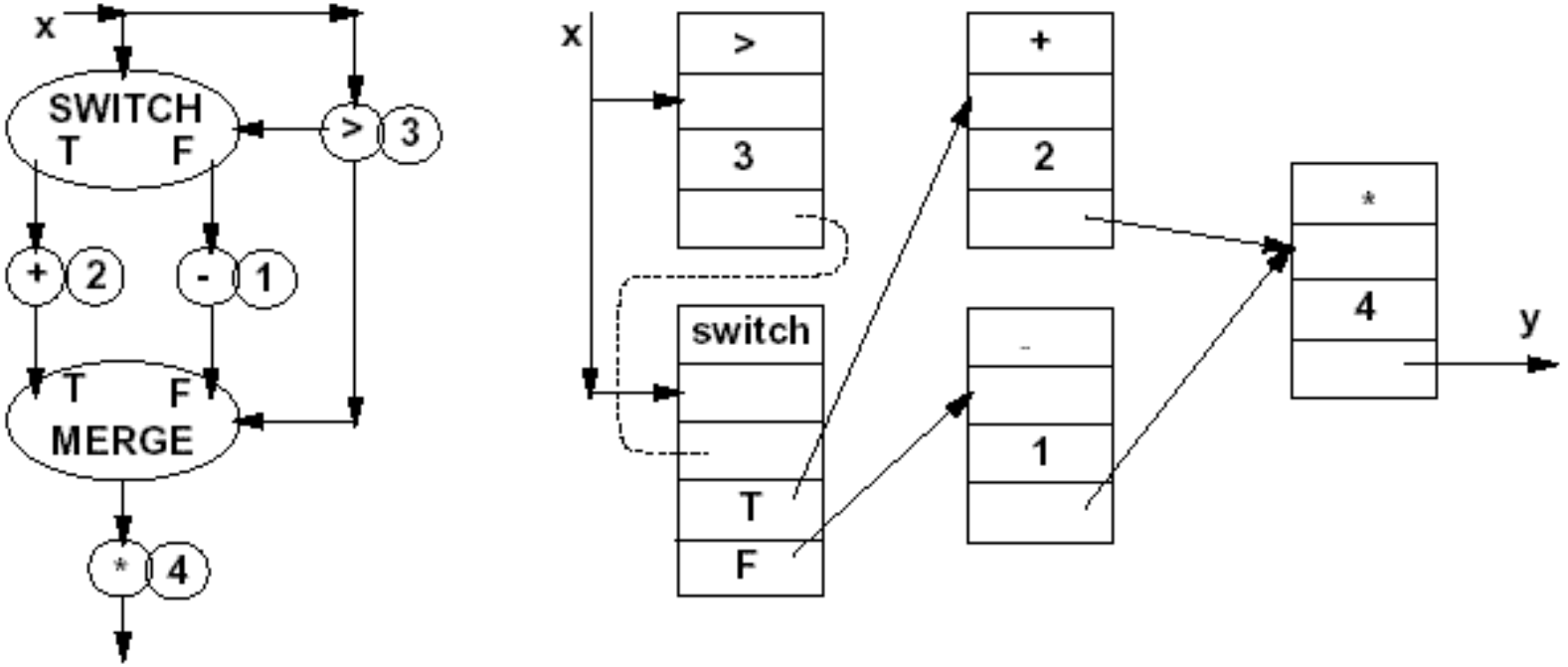
Darstellung der Befehle in der Maschine (Dennis)

Als Tupel (Opcode, Plätze für Argumente, Ziel-Liste).
Die Ziel-Liste ist die Liste der Tupel, die das Ergebnis als Argument benötigen.



Komplizierterer Fall

y := (IF x > 3 THEN x+2 ELSE x-1) * 4



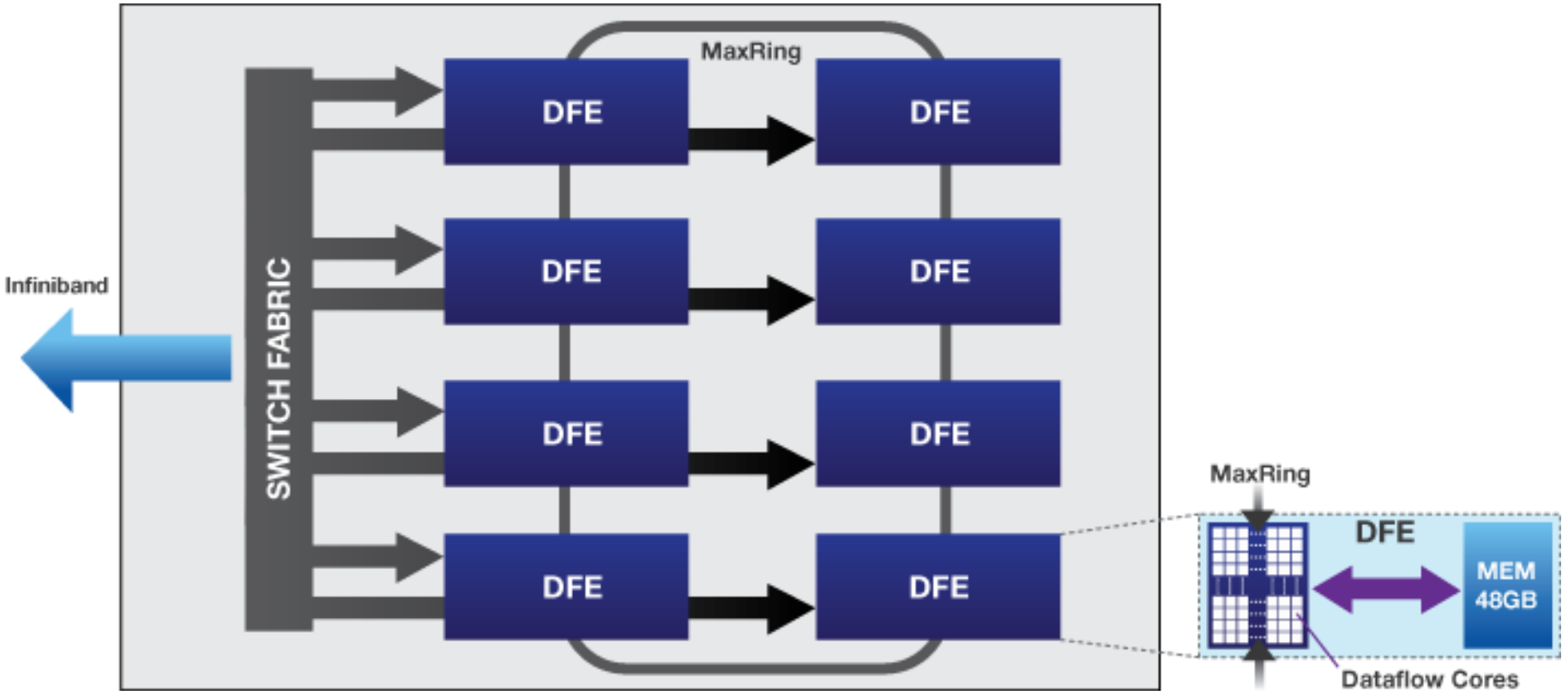
Suche nach Treffern ähnlich „*reservation stations*“ bei Tomasulo Algorithmus (nächstes Kapitel)

Statischer Datenfluss: Zuordnung Token/Operation sind fest

Dynamischer Datenfluss

- Es wird dynamisch nach zueinander passenden Daten gesucht
- Erst im Fall eines Treffers werden die dazugehörigen Befehle geholt.
- Erlaubt, für mehrere Daten Befehle gemeinsam zu nutzen, z.B. für alle Komponenten eines Arrays
- Wegen der dynamische Suche nach passenden Daten (über Assoziativspeicher) relativ hoher Aufwand

At last …: Kommerzielles Angebot



www.maxeler.com

Benutzt u.a. zur Erderkundung (Ölsuche) und *high speed trading*

Beispiel für die Programmierung

```
7   public class MovingAverageKernel extends Kernel {
8
9       public MovingAverageKernel (KernelParameters parameters, int N) {
10          super (parameters);
11
12          // Input
13          HWVar x = io.input ("x", hwFloat(8, 24));
14
15          // Data
16          HWVar prev = stream.offset(x, -1);
17
18          HWVar next = stream.offset(x, -1);
19
20          HWVar sum = prev+x+next;
21
22          HWVar result = sum/3;
23
24          // Output
25          io.output("y" , result , hwFloat(8, 24));
26      }
27  }
```

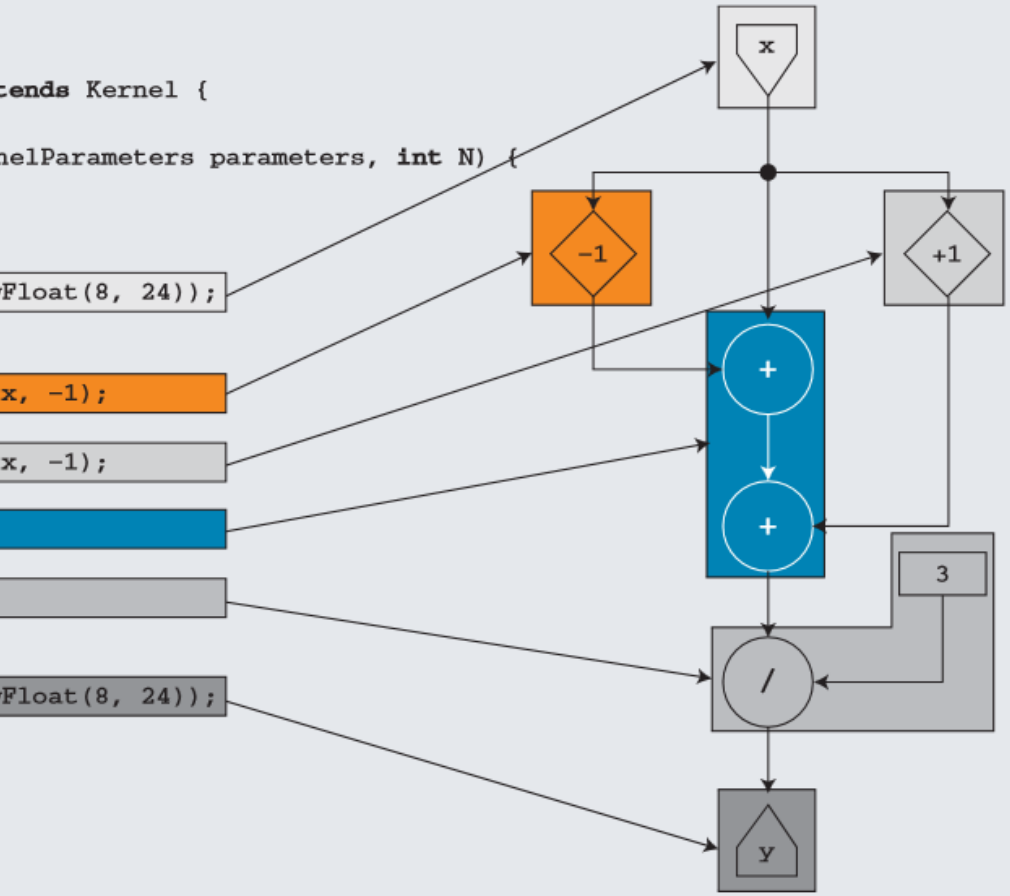


Figure A. Source code and corresponding graph for a simple moving average dataflow kernel. The current, previous, and next values in the input stream are accessed using stream offsets relative to the current position in the stream, and the three values are then averaged.

Pell, O.; Averbukh, V., "Maximum Performance Computing with Dataflow Engines," *Computing in Science & Engineering* , vol.14, no.4, pp.98,103, July-Aug. 2012

Vorteile der Datenflussrechner

Vorteile

- eingebaute Parallelität, eingebaute Synchronisation; vereinfachte Parallelisierung
- Adressen sind nach außen nicht sichtbar
- vorteilhaft bei gemeinsamen Teilausdrücken
- beliebige Reihenfolge der Abarbeitung bereiter Befehle

Historie der Datenflussrechner

- Euphorie Ende der 70er Jahre
- Stille Ende der 80er Jahre
- Prinzipien später teilweise in andere Systeme integriert
  - Datenflussrechner können leicht asynchron realisiert werden, benötigen so wenig Energie (z.B. Entwicklung . Sharp/U.Kochi/U.Osaka für Videokameras)
  - Von-Neumann-Rechner mit dynamischen *Scheduling* (*Scoreboarding*, Tomasulo-Algorithmus) [Kap.3] haben das Datenflussprinzip intern übernommen.
  - Datenflusssprachen: LabView, Simulink, ….

☞ „Datenflussrechner“ sind Realität, aber nicht in der ursprünglich angedachten Form

2.10 Funktionale Programmierung und Reduktionsmaschinen

- Maschinen unterstützen die Auswertung funktionaler Programme direkt.
- Reduktionsmaschinen akzeptieren Ausdrücke einer funktionalen Sprache.
- Auswertung mittels **Baumtransformationen**, bis ein **Wert** erhalten wird.
- Die Bedeutung eines Programms = abgelieferter Wert.
- Einfachere Verifikation
- **Vereinfachte Parallelverarbeitung**

Auswertungsstrategien

1. ***string reduction***

   Jede Operation, die auf eine bestimmte Definition (auf einen bestimmten Teilbaum) zugreift, erhält und bearbeitet eine Kopie des Teilbaums. ➔erleichterte Realisierung, schnelle Bearbeitung skalarer Werte, ineffiziente Behandlung gemeinsamer Teilausdrücke.

2. ***graph reduction***

   Jede Operation, die auf eine bestimmte Definition zugreift, arbeitet über Verweise auf der Original-Definition.
   ➔ schwieriger zu realisieren, falls parallel bearbeitet wird; effizient auch für strukturierte Werte und gemeinsame Teilausdrücke; komplizierte Haldenverwaltung (*garbage collection*).

Steuerung der Berechnungs-Reihenfolge

1. **Die „*outermost*”- oder „*demand driven*”-Strategie:**

   Operationen werden selektiert, wenn der Wert, den sie produzieren, von einer bereits selektierten Operation benötigt wird.

   Erlaubt *lazy evaluation*.

   Erlaubt konzeptuell unendliche Listen, solange nur endl. Teilmengen referenziert werden (vgl. Streams).

2. **Die „*innermost*”- oder „*data-driven*” - Strategie:**

   Operationen werden selektiert, wenn alle Argumente verfügbar sind.

Vorteile

- Programmierung auf höherer Ebene
- kompakte Programme
- keine Seiteneffekte
- kein *Aliasing*, keine unerwartete Speichermodifikation
- keine Unterscheidung zwischen *call-by-name*, *call-by-value* und *call-by-reference* notwendig
- einfachere Verifikation, da nur Funktionen benutzt werden
- das von-Neumann-Modell von Speicherzellen und Programmzählern ist überflüssig
- beliebige Berechnungsreihenfolge für Argumente (außer bei nicht-terminierenden Berechnungen)
- Debugging einfach: *Trace* des aktuellen Ausdrucks.
- Kommerziell bislang nicht erfolgreich

Ansätze zur Nutzung von funktionaler Programmierung bei Multi-Core-Prozessoren

- Verschiedene funktionale Sprachen
- **MapReduce** (Google): Definition zweier Funktionen:
  - *map*: (key,value) → (intermediate key,intermed. value)*
  - *reduce*: (intermed. key,value*) → (intermed. key,value)

  Beispiel:
  - *map*: (*buchtitel, inhalt*) → (*wort*, 1)*
  - *reduce* erhält nach *wort* sortierte Tupel, zählt Worte und liefert Häufigkeit für ein bestimmtes Wort

  map und *reduce* sind **Funktionen**
- **CLOJURE:** Lisp-Dialekt zur parallelen Programmierung auf der Basis der JVM

2.11 Maschinen zur Realisierung von Logischen Programmiersprachen (1)

Direkte Realisierung von logischen Programmiersprachen

- **Beispiel:** PROLOG-Maschinen
- **Realisierung:** PROLOG-Maschinen basieren meist auf der Übersetzung von PROLOG in *Warren Abstract Machine*- (WAM) Code.
- **WAM-Realisierung:**
  - WAM von Maschinenprogrammen interpretiert,
  - WAM-Befehle in Maschinencode übersetzt
  - WAM in Hardware

Maschinen zur Realisierung von Logischen Programmiersprachen (2)

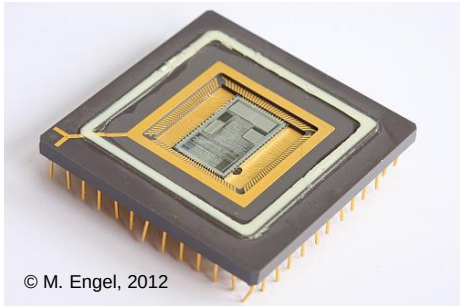
Klassen von PROLOG-Maschinen:

- Sequentielle Maschinen mit strenger Wahrung der PROLOG-Semantik
  Leiden meist unter Performanz-Problemen

- Parallele Maschinen mit unterschiedlicher Semantik (basieren nicht mehr auf formaler Mathematik)

Beispiele von Maschinen:

- PRIPs = Entwurf der Projektgruppe PRIPs.
- Entwürfe des *5th Generation Projekts*.



Problem: mit Datenmengen zunehmende Inkonsistenzen

Weitere Nicht-Von-Neumann-Maschinen: DNA-Rechner

- Im DNA-Molekül erfolgt eine Kodierung von Informationen mit 4 verschiedenen Basen.
- In einem Liter Flüssigkeit mit 6 g DNA ließen sich theoretisch ca. 3000 Exabyte an Informationen speichern.
- Die Rechenleistung läge bei ca. 10^{15} Operationen/s.
- Durch die Parallelverarbeitung könnten theoretisch kryptographische Schlüssel ermittelt werden.
- Angeblich wurde ein *traveling salesmen* –Problem gelöst – nur das Auslesen dauert lange.
- Referenz: K.H. Zimmermann, Z. Ignatova, I. Martinez-Perez: *DNA Computing Models*, Springer, 2008

Weitere Nicht-Von-Neumann-Maschinen: Quantenrechner

- Überlagerung von 0 und 1 verschiedener Lösungen in 1 Register ☞ parallele Berechnungen möglich.

  Parallelarbeit führt zu anderen Komplexitäten:

  - Algorithmus von Shor zur nicht-exponentiellen Faktorisierung auf einem Quantenrechner; Faktorisierung von 143 ist gelungen
  - Simulation von quantenmechanischen Vorgängen (bekannte Algorithmen sind schneller)
  - Datenbanksuche (beweisbar O-klassenmäßig schneller)
  - Berechenbarkeit bleibt unangetastet

- Kein Ersatz für klassische Rechner

Weitere Nicht-Von-Neumann-Maschinen: Quantenkryptographie

- Nutzung quantenmechanischer Effekte, um kryptographische Probleme zu lösen oder um kryptographische Systeme zu überwinden*. Beispiele:
  - Nutzung von Quantenrechnern, um kryptographische Codes zu knacken ☞ Interesse der NSA an Quantencomputern (Snowden, 2014)
  - Quantenmechanische Kommunikation: Beobachtung eines Zustandes bewirkt Selektion unter Zuständen Übertragung per Glasfaserkabel z.B. über 250 km° Gedacht v.a. für den Austausch von Schlüsseln

---

* Wikipedia, „Quantum computing“, Abfrage 16.4.2011

◦ D Stucki, N Walenta, F Vannel, R T Thew, N Gisin, H Zbinden, S Gray, C R Towery, S Ten: High rate, long-distance quantum key distribution over 250 km of ultra low loss fibres, New Journal of Physics, Vol. 11, 2009

Zusammenfassung

- Datenflussprinzip
  - *Systolic arrays*
  - Neuronale Netze
  - „Echte“ Datenflussmaschinen
- Funktionale Programmierung und Reduktionsmaschinen
- Realisierung logischer Programmiersprachen
- DNA-*Computing*
- *Quantum Computing*