Vektorrechner

1.1. z.B. 64-bit CPU nutzen, um 8 Operationen gleichzeitig auszuführen.

1.2 Pipelining (instruction pipelining)

1 Maschinenbefehl (z.B.

12 ADD $64 [REG]

sind in real mehrere CPU-Takte, z.B.

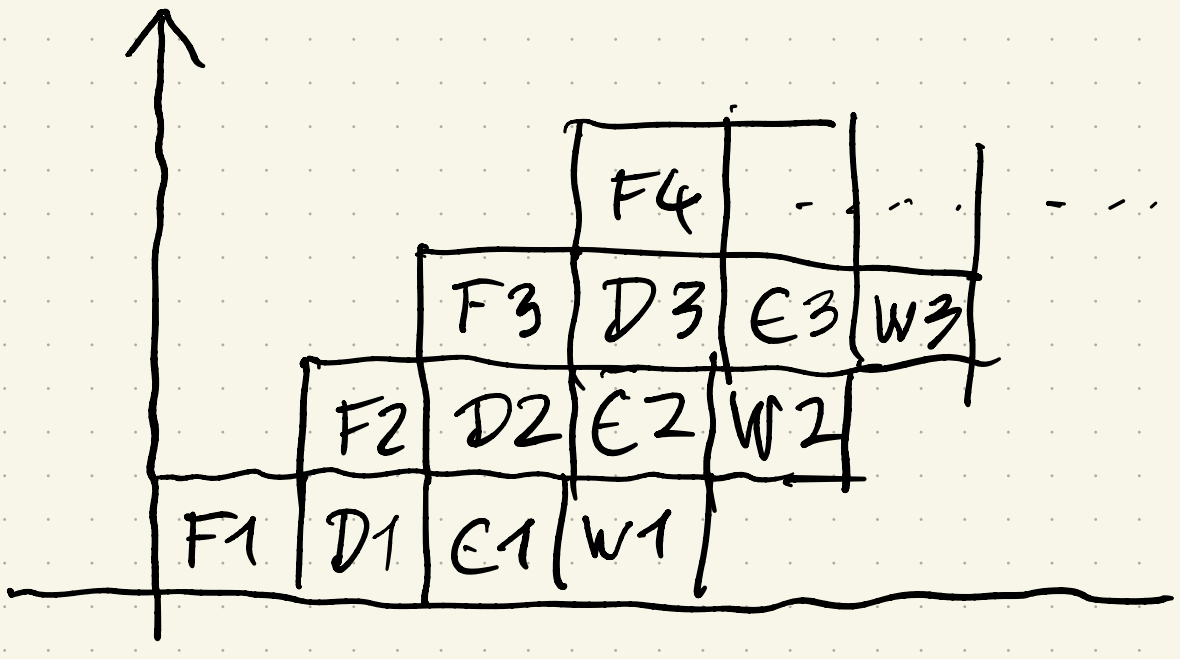
Fetch: lese Befehl

Decode: dekodiere

Execute: hole Daten aus Speicher und führe aus.

Write: schreibe Ergebnis

Trick: diese Schritte verschachteln



Probleme

bei 1.1: • Datentransport wird zum Engpass (Bandbreite)

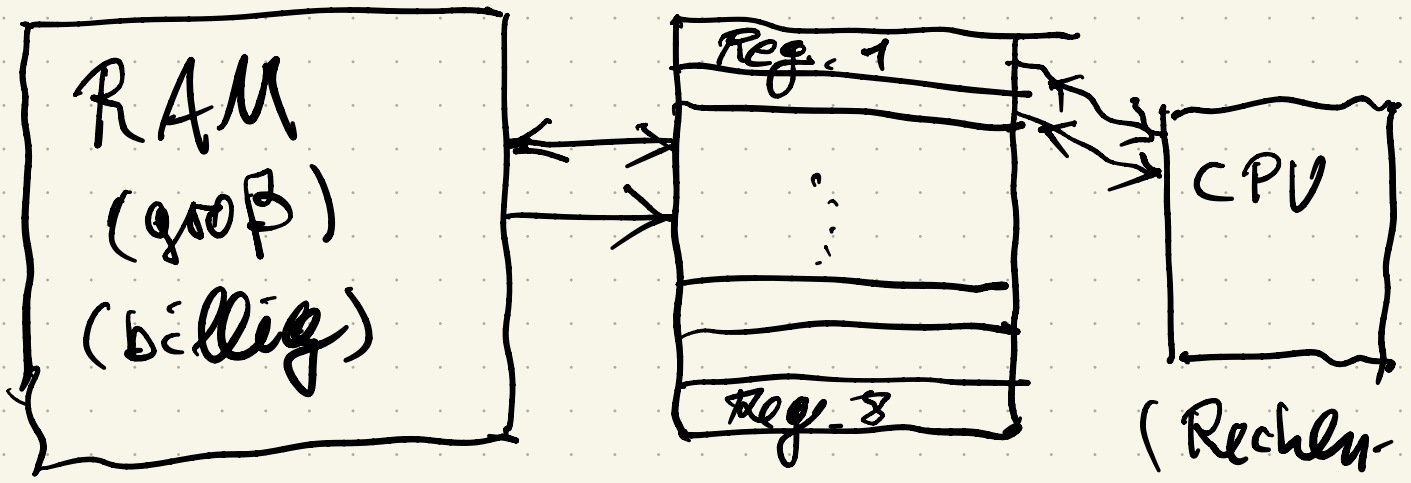
• Daten müssen hintereinander im Speicher stehen (oder so tun als ob)

• Hin- & Herschalten zwischen Steuer- & Daten-Operationen (IF vs. ADD)

bei 1.2: • Warten: bei $a_i = a_{i-1} + 5$ warten, bis a_{i-1} geschrieben ist

---

Crays lösen das, indem schnelle Steuerbefehle realisiert werden und optimiert Speicherzugriff durch viele Registerspeicher:



48 ns Zugriffszeit

6 ns (teuer)

Anderen oft viele Operationen auf denselben Daten: z.B. $a_i = 2a_{i-1} + 5 - a_{i-2}$

Also Daten lange im Register halten
[real steckt in Crays noch viel mehr; aber OK...]

C Mehrere Prozessoren bzw. Kerne

Oben läuft ja zunächst immer noch nur ein Programm ab. Versuche das wie oben zu beschleunigen:

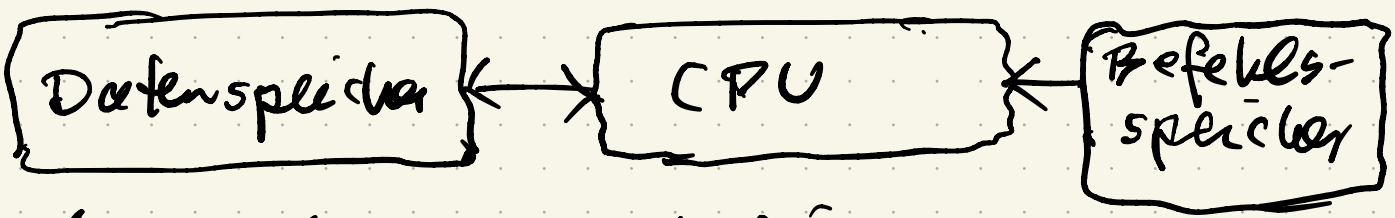
- Experimente zeigen, dass jenseits von 4 Prozessoren kaum Verbesserungen entstehen.
- Erfordert komplexes Scheduling

Also: Mehrkernprozessoren
(mehrere vollwertige CPUs auf einem Chip)

Flynnsche Klassifizierung

Ein Vorschlag der Klassifizierung (SISD, SIMD...)

(a) SISD Single Instruction Single Data

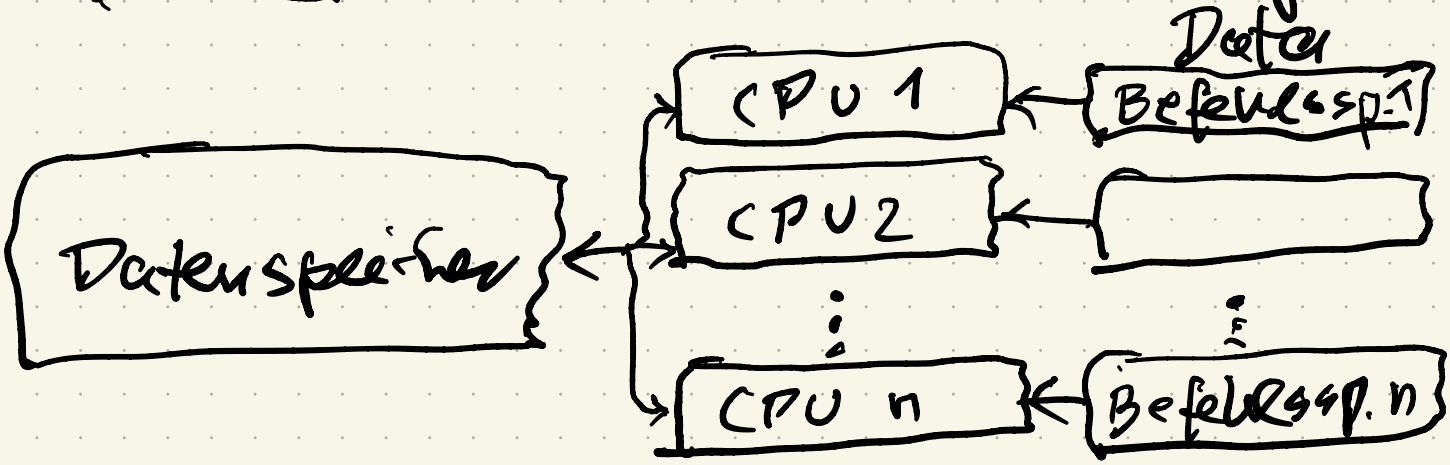


(von-Neumann-Architekturen)

Vorteil: • einfach zu benutzen & realisieren

Nachteil: siehe oben

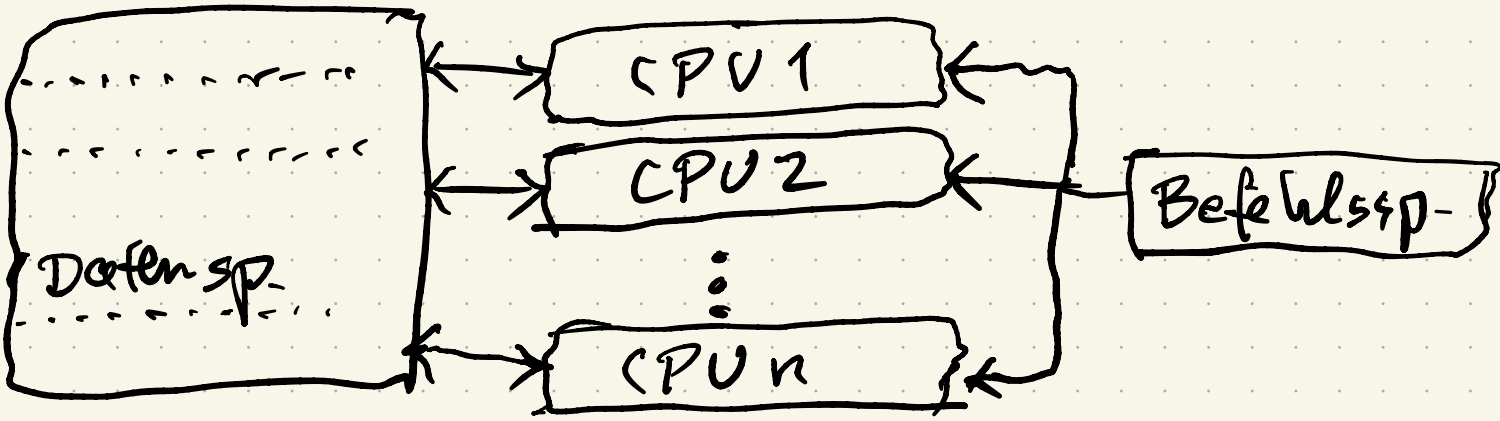
(b) MISD Multiple Instruction Single Data



Vorteil: keine (verschiedene Befehle müssen auf denselben Daten arbeiten)

Nachteil: sinnlos

(c) SIMD: Single Instruction Multiple Data



(z.B Vektorrechner)

Vorteil:
- einfach zu programmieren/realisieren
- gut für „number crunching" (gleiche Operationen auf großen Datenmengen)

Nachteil - Vorteile ausreizen ist evtl. aufwändig bzw nicht möglich

Bsp:

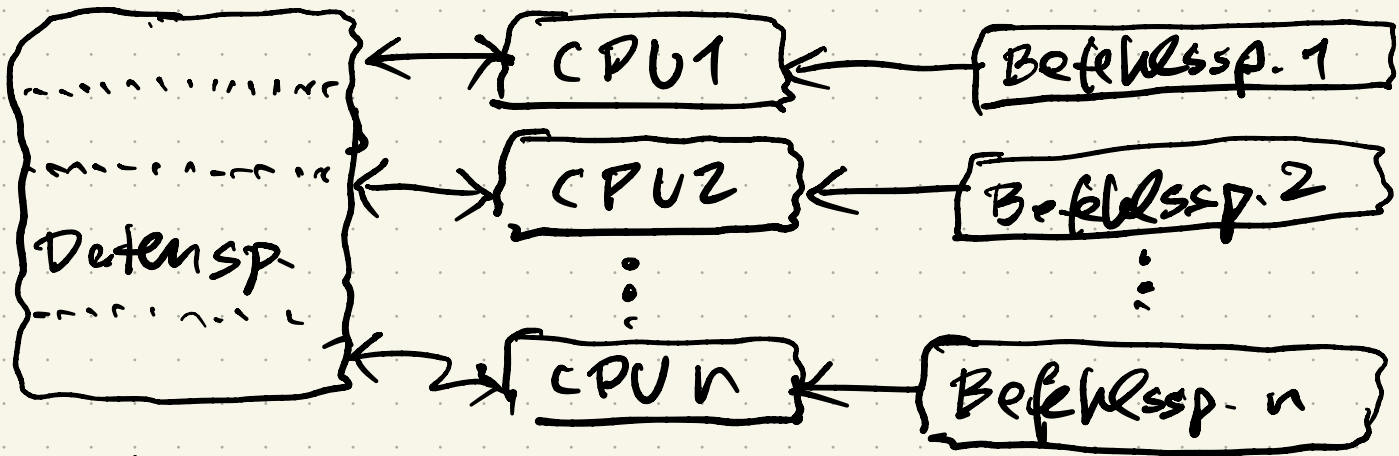
```
a = 2
b = 3
③ a = 2 * b
c = a   ← also 2 oder 6, je nachdem, ob ③ schon erledigt
```

Bsp

```
if b_i == 0   c_i = a_i   else c = a_i / b_i
   ①              ↖              ↗
```

„single instr.", also: geht nicht gleichzeitig (falls das für viele b_i parallel ausgeführt werden soll)

MIMD Multiple Instruction Multiple Data



Vorteil • n Prozesse unabhängig, also potenziell n-mal schneller,

Nachteil • Vorteile ausreizen erfordert

Kluge Organisation.

Praktisch alle Geräte (PC, Laptop, Handy) heute sind MIMD.
Auch „Supercomputer": Cluster mit mit 1000en GPUs

2. Netzwerktopologie

Beispiele oben zeigen bereits, dass der Speicherzugriff klug organisiert werden muss.
Teil dessen: Wie genau CPUs mit dem Speicher vernetzt sind.

- „Ideal": jeder mit jedem: physisch nicht machbar.

Also anders regeln:

- Topologie (Gestalt)
- Routing (wer wo lang)
- Switching (stückeln, „Ampeln", „Weichen")

Zur Topologie: [nächste Vorlesung]