

Rechnerarchitektur & Betriebssysteme

Klaus Kusche

Inhalt

- **Rechnerarchitektur**

(diese Folien, Folien HW-Prog PI, mehrere alte Skripten)

- **Betriebssysteme**

(Skript Betriebssysteme IK, diverse alte Vorträge)

- **Interprozesskommunikation**

(Folien Vortrag Mutex & Folien Sysprog PI)

- **Linux, Shell-Programmierung**

(mein Übungs-Spickzettel, altes HTL-Skript Shell)

Abgrenzung RA (1)

Wir beschränken uns auf:

- ***Elektrisch***
- ***Digital*** (es gibt auch Analogrechner, z.B. Signalverarbeitung: Multiplikation zweier Pegel, Integration, ...)
- ***Programmierbar***,
nicht “hart verdrahtete” Funktion
(d.h. Funktion durch gespeichertes / änderbares
Programm gesteuert)
- ***Mit “Zustand”*** (= Speicher)
(keine reine “Rechenlogik”)

Abgrenzung RA (2)

- **“von Neumann”** im weitesten Sinne:
Schrittweise Ausführung einzelner Befehle,
gesteuert durch ihre Reihenfolge im Programm:
 - Befehl holen
 - Befehl dekodieren
 - Operanden holen
 - Operation ausführen
 - Ergebnis speichern
 - Befehlszähler erhöhen
 - nächsten Befehl holen ...
- ==> Durch Adressen angesprochener Speicher
für Programm & Daten**

Abgrenzung RA (3)

Hier nicht: Alternative Berechnungsmodelle z.B.:

- ***Datenfluß-Maschine***
(für rein funktionale Sprachen):
Einzelbefehle ähnlich “von Neumann”
Ablauf Datenverfügbarkeits-gesteuert & parallel
- ***Inferenz-Maschine*** (für logische Sprachen):
Findet Lösungen für logische Formeln
durch systematisches Erzeugen / Durchprobieren
von Variablen-Belegungen
- ***(neuronale Netze)***

Geschichte

Siehe altes Skript:

- Mechanische Rechenmaschinen
- Mechanische programmierbare Rechner
- Elektromechanische Lochkartenzähler
- Elektromechanische programmierbare Rechner
- Elektronenröhren-Rechner
- Transistorisierte Rechner
- Rechner mit IC's
- Mikroprozessoren = Ganze CPU's auf 1 Chip
Am Anfang: 8 bit, 1 MHz, rund 1000 Transistoren

Geschichte, Speicher

Intern (“RAM” und Register):

- Elektrisch
- Spezielle Röhren (u.a. optisch), Laufzeitspeicher, ...
- Magnet-Trommeln
- Kernspeicher (Magnetringe)

Extern (Platte):

- Lochkarte / Lochstreifen (Film!)
- Magnetisch (Platte, Band, Floppy, ...)
- Film-Optisch (Mikrofilm, nur als Archiv)
- CD, ...

Zukunft

“Rechen- und Leitungs-Technologie”:

- Optisch
- Quanten
- Andere??? (z.B. Kohlenstoff-Nanotechnologie)

Weil:

- “Moore’s Law” ist ziemlich am Ende
- Klassische Halbleitertechnik nähert sich physikalischen Grenzen

(betr. Miniaturisierung, Geschwindigkeit, Kosten, elektrischer Leistungsaufnahme, ...)

Zukunft, Speicher (1)

Auf “Chips” (schon verfügbar, noch zu teuer):

- Magnetoresistive Speicherung (MRAM)
- Ferroelektrische Effekte (FRAM)
- Phasenwechsel-Materialien (PCRAM)

Ziel:

- Mindestens so schnell, billig, klein, ... wie RAM
- Nichtflüchtig: Hält Daten “ewig” ohne Strom
- Keine mechanischen / bewegten Teile
- Keine Alterung (Flash altert beim Schreiben!)

Zukunft, Speicher (2)

“Im Großen”:

- ***Thermomagnetische*** Effekte
(HAMR, “dichtere” Festplatten)
- ***Magneto-optische*** Effekte
(um Jahr 2000 üblich, veraltet,
jetzt “neu erfunden”?)
- ***Optische*** Speicherbänder (“Tesspeicher”)?
- ***Holografische*** Speicher???

Bestandteile eines Rechners

- *CPU* (ev. mehrere)
- *RAM, ROM* (direkt adressierter Speicher)
- *Schnittstellen, I/O-Devices, Timer, ...*

Verbunden durch “Busse”

(stimmt heute nur mehr “logisch”,
nicht mehr physikalisch)

“Klassischer” Bus

Byteweise, parallele Datenübertragung ...

... zwischen *mehreren angeschlossenen* Bausteinen

Besteht aus vielen Leitungen:

- **Datenbus**
- **Adressbus**
(welcher Baustein, welcher Speicher?)
- **Steuerbus**
(Lesen/Schreiben, “Daten bereit”, Interrupts, ...)

==> *Früher für fast alles, heute nur mehr zum RAM!*

“Bus” heute

Intern heute vor allem *PCIe*:

- ***Seriell***, d.h. nur 1 oder 2 (differenzielle) Drähte (bzw. mehrere serielle Verbindungen parallel)
- ***Punkt zu Punkt***:
(zwischen CPU und einem I/O-Gerät, je eine eigene Verbindung pro Gerät)
- Effizienter bei ***großen Datenblöcken*** (ab 16 Bytes)

Ebenso bei ***externen Bussen*** (USB, SATA, HDMI, ...) und ***zwischen CPU's***:

Auch alle Punkt-zu-Punkt und seriell!

Schnittstellen & I/O-Devices (1)

- **Massenspeicher** (SSD, Platte, Band, CD, SD-Card,...)
bzw. Schnittstellen dafür (SATA, SAS, FC, ...)
- **Grafikkarte**
(eigener “Rechner” mit Prozessor, RAM, ...)
mit HDMI, DVI, DP, ...
- **Soundkarte, Videokarte, ...**
- **Ext. Schnittstellen:** USB, serielle Schnittstelle,
Thunderbolt = PCIe + USB + DP, ...
- **Netzwerke** (Ethernet, WLAN, BT, ...)
und Rechner-Kopplungen (Infiniband, ...)

Schnittstellen & I/O-Devices (2)

- **Timer, Watchdog, ...**
- **“Einfache” On-Board-Busse (I2C, SMBus, ...)**
(z.B. Lüftersteuerung, Temperatursensor, RAM-Kennung, ...)
- **Industrielle Feldbusse (CAN, Profibus, ...)**
(für Sensoren & Aktoren in Steuerungen)
- **Analog-I/O (AD/DA-Wandler),
digitale I/O-Pins**

*Viele Schnittstellen sind heute im “Chipsatz”
oder im Prozessor-Chip integriert!*

“Byteweiser” Speicher: RAM & ROM

- Hängt direkt an der CPU
- Byteweise adressiert,
direkter Zugriff auf einzelne Bytes
(de facto: Meist Vielfache von 64 Bits = 8 Bytes)
- **RAM**: “Hauptspeicher”, laufender Code & Daten
 - Gleich schnell & gleich einfach les- und schreibbar
 - Strom weg ==> Daten weg
- **ROM**: BIOS, Firmware
 - Nicht / nur mit speziellen Methoden schreibbar
 - Behält Daten auch ohne Strom

DRAM & SRAM

DRAM: 1 Zelle (Bit) = 1 Kondensator + 1 Transistor

- **Billiger, höher integriert**
- Relativ langsamer Zugriff (~ 50 ns),
aber dann schnelle sequentielle Transfers
- Komplexe Ansteuerung, braucht periodisch "Refresh"

SRAM: 1 Zelle (Bit) = 6-8 Transistoren

- Entweder **sehr schnell** (< 1 ns: CPU-Register, Cache)
oder **sehr sparsam** (Batterie-CMOS für BIOS)
- Hält Daten **statisch** (ohne Takt, ohne Refresh, ...)

Fehlerkorrektur RAM

- RAM ursprünglich: ***1 Parity-Bit pro 8 Bit***
==> *Erkennt alle 1-Bit-Fehler, keine Korrektur!*
- Normales RAM für PC's heute: Keine Prüfung!
(obwohl RAM definitiv gelegentlich Bits verliert!)
- RAM in Servern & Workstations:
ECC mit 8 Prüfbits pro 64 Datenbits
(deshalb ist Server-RAM 72 statt 64 Bit breit)
==> Korrigiert alle 1-Bit-Fehler,
erkennt alle 2-Bit-Fehler

Cache (1)

Cache = Zwischenspeicher zur Beschleunigung

- Ist technisch auch “**RAM**”
- Enthält die zuletzt benutzten Daten eines langsameren Speichers, spart bei erneutem Zugriff auf die Daten den Zugriff auf den langsamen Speicher
- Ist logisch transparent (d.h. aus Anwendungssicht “unsichtbar”)
- Ist viel kleiner & viel schneller als der Speicher, dessen Daten er zwischenspeichert

Cache (2)

Cache in der CPU:

- L1-, L2- und L3-Cache
- TLB (Translation Lookaside Buffer) in der MMU

Siehe separate Folien!

- Eigene Hardware (SRAM) auf dem Prozessor-Chip (großer Flächen- bzw. Kosten-Anteil!)
- Speichert Daten aus dem normalen RAM
- Größe 16 KB - 16 MB, bei Servern auch mehr
- Ist rein in Hardware verwaltet

Cache (3)

Disk-Cache des Betriebssystems:

- Rein in Software vom Betriebssystem verwaltet
- Liegt im Hauptspeicher (normalen RAM):
Nutzt den Rest des RAM (ev. mehrere GB),
der momentan nicht für Programme & Daten
gebraucht wird
- Speichert Daten von den Platten
- Auch als Schreib-Puffer

Cache (4)

Hardware-Disk-Cache:

- Sitzt am Plattenlaufwerk / im Storage-System / am Platten-Interface bzw. RAID-Controller
- Eigene, “gewöhnliche” RAM-Chips
- 64 MB bis einige GB
- Von eigenem Prozessor (im Disk-Controller bzw. RAID-Controller) in Software verwaltet
- Speichert Daten von den Platten
- Für das System “unsichtbar”

ROM

Historische Entwicklung:

- **ROM** (Maskenprogrammiert bei Fertigung)
- **PROM** (1 mal schreibbar: “Schmelzdrähte”)
- **EPROM** (1*schreibbar, als Ganzes UV-löschbar)
- **EEPROM** (1*schreibbar, als Ganzes elektr. löschbar)
- **Flash** (1*schreibbar, blockweise elektr. löschbar)

Flash (SSD's, USB-Sticks, SD-Cards, ...)

ist noch kein “vollwertiger” R/W-Speicher!

- Löschen & schreiben sehr langsam & aufwändig
- Löschen nur in großen Blöcken, nicht pro Byte
- “Alert” beim Schreiben (~3000 mal schreibbar)

Massenspeicher

Platte, SSD, Band, CD, SD-Card, USB-Stick, ...

- Verbunden über I/O-Controller & eigene Schnittstellen (SATA, SAS, FC, USB, ...)
- Speichern Daten in Sektoren bzw. Blöcken, Sektoren nur als Ganzes lesbar / schreibbar
- Nur blockweiser Datentransfer (512 B / 2 KB / 4 KB), kein direkter Zugriff auf einzelne Bytes
- Adressierung / Verwaltung über Blocknummern, keine Adressen einzelner Bytes

Fehlerkorrektur Massenspeicher

ECC (Error Correcting Code):

- Große Prüfsummen (32-256 Bits) pro Block
- Erkennen mit sehr hoher Wahrscheinlichkeit praktisch alle Fehlerarten
- Können einige (aufeinanderfolgende) falsche Bits reparieren

Früher: CRC (Cyclic Redundancy Check)

- 16 Bit pro Sektor
- Nur Erkennung

CPU

- Ein Rechner enthält
ein oder mehrere Prozessor-Chips
(= CPU's ?!)
- Ein Prozessor-Chip enthält
ein oder mehrere Cores
(+ Cache, Busse, Schnittstellen, ev. Grafik, ...)
- Ein Core führt
zu einem Zeitpunkt ein Programm aus
(ev. 2 bis 8, "Hyperthreading")

Core (1)

Ein Core besteht aus:

- **Steuerwerk:**

Holt & dekodiert Befehle

enthält IP-Register (“Instruction Pointer”)

steuert Rechenwerk & Speicherzugriffe

- **Rechenwerk (*ALU = Arithmetic Logic Unit*):**

Rechnet

- **Register:**

Speichern gerade benötigte Daten

(8-32 Stück, + FP-Register, Spezialregister, ...)

Core (2)

- ***MMU = “Memory Management Unit”***
Umrechnung virtuelle / physische Adressen
Speicher-Zugriffs-Schutz
- L1-Instruktions- und L1-Daten-***Cache***

CPU-Architektur (1)

Einteilung:

- “*von Neumann*”-Architektur im engeren Sinn:
Ein gemeinsames RAM, ein gemeinsamer Bus,
ein gemeinsamer Cache für Code & Daten
- “*Harvard*”-Architektur:
Separates RAM, separate Busse, separate Caches
für Code & Daten

Heute:

- bis incl. L1-Cache “Harvard”
- ab L2-Cache, ext. Busse, RAM “von Neumann”

CPU-Architektur (2)

Eine Prozessor-Architektur bzw. -Familie hat

- Einen ***Befehlssatz*** (Welche Befehle gibt es?)
- Einen ***Registersatz*** (Wie viele Register für welchen Zweck & für welche Typen?)

Die “***Breite***” eines Prozessors (z.B. “32 bit”) ist

- ... die Breite seiner allgemeinen Register (nicht FP)
- ... die Breite der Integer-Werte, die die ALU mit einem Befehl verarbeiten kann
- ... die Breite der Adressen (==> max. RAM-Größe)

Befehlssatz (1)

Proprietär, pro Prozessor-Familie

1 Befehl codiert eine Operation & deren Operanden

- **Arithmetische & logische Operationen, Bit-Shifts, ...**
- **Vergleiche & Bit-Tests, ...**
- **Laden & Speichern (CPU-Register $\langle == \rangle$ RAM)**
- **Bedingte & unbedingte Sprünge, Call & Return, indirekte Calls**

... als 1-16 Bytes langes Bitmuster

Befehlssatz (2)

Einteilung:

RISC = “Reduced Instruction Set Computer”
(ARM, Power, Sparc, ...)

CISC = “Complex Instruction Set Computer”
(x86, zSeries)

Siehe Folien Assembler-Programmierung!

Aktuelle CPU-Familien

32 / 64 Bit:

- **x86** (Intel, AMD): PC's, Server, ...
 - **ARM** (viele): Handy's, Kleinrechner, Server, embedded Systems, ...
 - **zSeries** (IBM): Großrechner, seit > 50 Jahren!
 - **Sparc** (Oracle / Sun, Fujitsu), **Power** (IBM): RISC-Server
 - **MIPS, PowerPC, RISC V, ...**: Embedded Systems
- ... + viele **Microcontroller-Familien** (8 / 16 bit)

“Klassische” Arbeitsweise der CPU

Ein Befehl nach dem anderen:

- Nächsten Befehl aus dem Speicher holen
(Adresse steht im IP-Register)
 - Befehl decodieren
 - Falls der Befehl Speicher-Operanden hat:
Deren Adresse ausrechnen, Daten aus dem Speicher holen
 - Operation ausführen
 - Falls der Befehl Daten im Speicher ablegt:
Deren Adresse ausrechnen, Daten speichern
 - Befehlszähler (IP-Register) inkrementieren
- ... und wieder nächsten Befehl holen ...

Moderne CPU's

Ein "von-Neumann-Prozessor" strikt nach Lehrbuch wäre bei gleichem Takt

einige 100 Mal langsamer

als moderne reale CPU's!

Verbesserungen siehe altes Skript:

- Pipelining
- Out-of-Order & Speculative Execution, Multi-Issue
- Sprungvorhersage
- Caches
- Multicore, Hyper-Threading
- Vektor-Befehle

Parallelrechner, Supercomputer

- **MIMD** “Multiple Instruction Multiple Data”
(pro Takt):
Mehrere unabhängige, vollwertige Prozessoren
- **SIMD** “Single Instruction Multiple Data”:
*Ein Steuerwerk steuert viele Rechenwerke
“im Gleichschritt” (AVX-Einheit, Grafikkarte, ...)*
- **Vektorrechner**:
*Dieselbe Rechenoperation wird Pipeline-artig
für alle Elemente eines Arrays durchgeführt
(1 Ergebnis pro Takt)*

UMA & NUMA (1)

Probleme bei immer größeren Rechnern:

- Zugriff aller CPU's auf *gesamtes, gemeinsames RAM* (viele TB!)
- Kommunikation aller CPU's untereinander

Physikalische Grenzen

(Leitungslänge, Anschlüsse pro Leitung, Laufzeit, ...):

==> “Bus” ist nicht möglich

==> “Schaltnetzwerk” ist ab gewisser CPU-Zahl zu groß / zu teuer / zu langsam

UMA & NUMA (2)

“Frommer Wunsch”:

UMA = “uniform memory access”

*= Jede CPU kann auf
jeden Teil des Speichers
gleichartig & gleichschnell zugreifen*

Wäre softwaremäßig am einfachsten:

- Egal, was wo gespeichert wird
(Verwaltung als ein einziger Speicher)
- Egal, was auf welchem Core gerechnet wird

UMA & NUMA (3)

Realität:

NUMA = “non-uniform memory access”

Jede CPU hat “*eigenes*” RAM

(lokal, direkt angebunden ==> sehr schnell)

und “*fremdes*” RAM

(hängt an anderer CPU ==> deutlich langsamer)

==> Logisch immer noch ein großes RAM:

1 Adressraum

==> Braucht schnelle Kommunikationsverbindungen
zwischen den CPU's

UMA & NUMA (4)

NUMA ist softwaremäßig schwieriger

- für das Betriebssystem
- für performance-kritische parallele Anwendungen
- Daten und Code wenn möglich lokal halten, nicht auf viele CPU's verstreuen
- Programme auf genau dem Core ausführen, in dessen RAM Code & Daten liegen
- Große Datenmengen & Berechnungen gleichmäßig auf CPU's und deren RAM verteilen

UMA & NUMA (5)

... und wenn's noch größer sein muss:

Cluster

- Separate Rechner mit jeweils eigenem RAM
 - Sehr schnell netzwerkartig gekoppelt
(oft mit Infiniband)
- ==> Zugriff auf **“fremdes” RAM**
ist logisch kein normaler RAM-Zugriff mehr,
sondern eine Art **Netzwerk-Transfer**