Angewandte Computer Architektur

A. Gunzinger, HS2020, Noah Hütter, 01.2021
ETH Zürich

1. Applications

Linear Equation Solving Sy:

Pivoting $\boxed{O = 2n^3}$

$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \Rightarrow \begin{pmatrix} 1/a & -b/a \\ c/a & d-\frac{bc}{a} \end{pmatrix}$

1. Select pivot
2. inverse it
3. row: $-1/$pivot
4. col: $1/$pivot
5. rest: octagonal rule

Gauss elim. $\boxed{O = \frac{2}{3}n^3}$

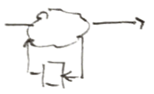
1. Solve 1st eqn
2. insert
3. repeat
4. substitute back

Gradient Decent $\boxed{O = 2n^2}$ per iter $x(t+1) = A' x(t) + b'$

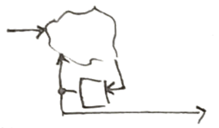
Performance of computers is measured by running LINPACK benchmarks: Solve big dense lin. eqn sys.

2 Basics of Digital Logic

Mealy



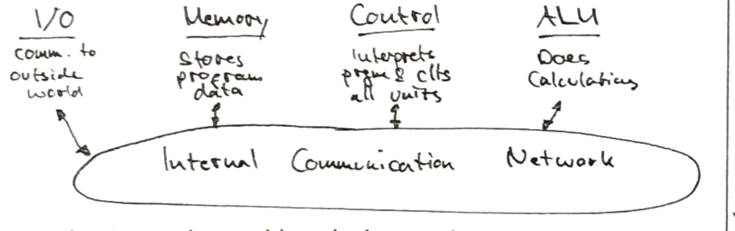
Moore



+ Fast − Not glitch free

+ Glitch free − 1 clk delay

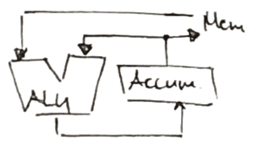
3 Basics of Computer Architecture



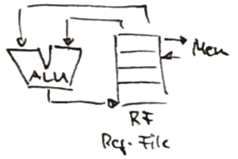
Exec of single Insn:

IF	Instr.	Fetch
ID	Instr.	Decode
OF	Opvand	Fetch
EX	Execute	
WB	Write	Back

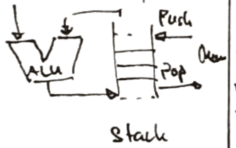
Accumulator Machine



Register Machine

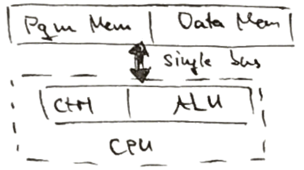


Stack Machine



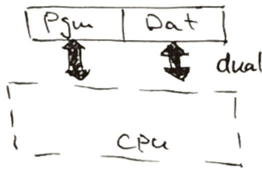
Acc	Reg	Stack
+ si[illegible] impl.	+ good perf.	+ Good model for SW
+ s[illegible] insn	+ low mem traffic	+ short ins[illegible]
− high mem traffic	− complicate SW (reg. alloc.)	+ simple excep[illegible] handle
	− Variable insn len.	− no random access
		− performance

Von Neumann



+ single bus
+ optimum pgm/data allocation
− performance

Harvard



+ performance
− 2 bus
− bus-opt mem. alloc.

Performance Measurements

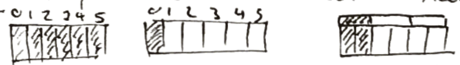
MFLOP	.	Mio float point ops /s
MIPS	.	Mio int ops /s
MOPS	.	Mio ops /s
SPEC	.	Standard perf eval corp. (int / fp)
Whetstone	.	synth benchmark
Dhrystone	.	like Whetstone w/o float

Trends Moors law / Transistors p. chip / clock / performan

4. Cache & Virtual Memory

\$: Leverage Temporal & Spatial locality

Org



Blk 12 can be placed anywhere

Blk 12 placed @ 12 % 6 = 0

Blk 12 placed in any of set 12 % 3 = 0

Victim Strategy: Block has to be replaced

LRU - least recently used FIFO · first in first out

Random · as good as LRU for large caches

Write
- Back: W data to \$, set dirty, is written to main Mem on \$ replace
- Through: W to \$ & Memory, CPU stall, always consistent, can be accelerated by FIFO

Snooping Invalidate \$ if main memory changed

VM · Seemingly large RAM · Swap to disk
· needs MMU and PT

5. Pipelining

Maximize Usage of computation units

```
IF ID OF EX WB
   IF ID OF EX WB
      IF ID OF EX WB
         ...
```

Speedup k: num stages n: num instructions
τ_i: time for stage i τ_a: add. delay
$\tau = \tau_a + \max \tau_i$

Max speedup $S_k = \frac{n \sum \tau_i}{\tau(k+n-1)}$ Efficiency $\eta = \frac{n}{k+n-1}$

Max freq. $f = \frac{1}{\tau}$ Max throughput $w = \frac{\eta}{\tau} = \frac{nf}{k+n-1} = \frac{n}{\tau(k+n-1)}$

Hazards

- Control: Instr. flow unknown bcs conditional insn
  → stall or execute speculatively
- Structural: Resource busy
  → stall or duplicate resource
- Data: RAW (read after write)
  WAW (write after write), overwrite sth.
  WAR (write after read)
  → stall or bypass

Branch Prediction Helps mitigate control hazards

- static / dynamic: either at compile time (s) or the CPU decides (d)
- tournament BP · predictor for the predictor
- branch delay slot: insert insns that are executed anyway

Clock distribution is important → same len, glitches

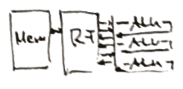
6. Vector Processors

Operates on vectors instead of scalars.

Ops sqrt, sum, min, max, VADD, SADD (r_{scalar}), VMUL, ...

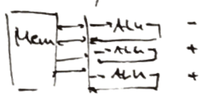
Bit vector mask select src/dst on merge, compress, expand

Register-Machine



+ fast
− Vect. splitting
− expensive

Memory Machine



− slower
+ easier
+ less expensive

Mem Interleaving separate memories in banks to be accessed in parallel

7. Parallel Processing Basics

Advantages Performance, low cost, size & weight, modular, scalable

Parameters Perf, cost, SW, programming model, power, space, maintenance, ..

Classification

- Multi processor
  - single adr. space
  - shared mem comm.
  - Distributed Mem → scalable
  - Centralized Mem → not scalable
- Multi Computer
  - multi adr. space
  - message passing
  - Distributed → scalable network
  - Centralized → fixed network

Speedup Optimal speed-up often not achieved because

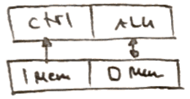
- Serial part of code (Amdahl)
- Communication
- Sync & coordinate
- Distribution of comp.

Synchronization Attention: Deadlocks

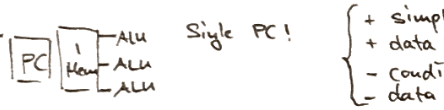
Sol: counter, HW synch

8. Parallel Programming Models

Uni-Processor



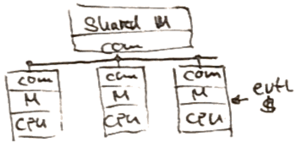
SIMD



Single PC!

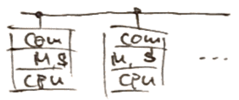
- + simple
- + data parallel s.a. Audio/Img
- − conditionals
- − data depend.

SMP Sym. Multi Proc.



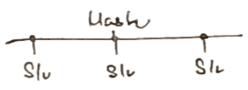
- + good for < 32 Proc.
- + simple sync & programming
- − bad scalable
- − data consistency

MP essage assing



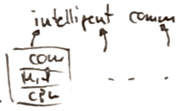
- + scalable
- + MPI programming standard
- + local communication
- − lots of HW

C/S lient erver



- + simple
- + can use ethernet
- + flexible
- − slow for tasks with intense com.

IC ntelligent ommunication



- + very high performance for comm.
- + simple HW
- − no local comm
- − not widely used

9. Parallel Processing Performance

Speed-up

$$S(p) = \frac{T \text{ one PE}}{T \text{ p PE}} = \frac{t_E(1)}{t_E(p)} = \frac{t_s + t_p}{t_s + \frac{t_p}{p} + p \cdot t_c}$$

t_s: serial part
t_p: parallel part
t_c: communication

$t_e = t_E$: execution time $\qquad S(0) = S(\infty) = 0$

$$k_s = \frac{t_s}{t_E(1)} \qquad k_p = \frac{t_p}{t_E(1)} \qquad k_c = \frac{t_c}{t_E(1)}$$

Performance $P(p) = \frac{\#\text{ FP instr.}}{t_E(p)} = S(p) \cdot \frac{\#\text{FP ins}}{t_E(1)} = S(p) \cdot P(1)$

Efficiency $E(p) = \frac{S(p)}{p} = \frac{1}{\frac{k_s}{p} + \frac{k_p}{p^2} + k_c}$

Max Speedup $\max S(p) \leftrightarrow p = \sqrt{\frac{k_p}{k_c}}$, $\max S(p) = \frac{1}{k_s + 2\sqrt{k_p k_c}}$

Factors s_0: serial insn n_0: parallel ins c_0: insn per task (= granularity)

CPI: clocks per insn, t_0 : clk period, n_s/n_r : send/receive # issues

b_0: data / task B: bandwidth

exec time $P = \frac{\#\text{ insn}}{t_e} = \frac{s_0 + c_0 n_0}{(s_0 + c_0 n_0)\,CPI\,t_0} = \frac{1}{CPI\,t_0}$

Matmul : $n_0 = n$ (# order of Mtx) $\quad c_0 = 2n$

Communication $t_{com} = \frac{n_0 b_0}{B} \qquad t_{calc} = \frac{n_0 c_0\, CPI\, t_0}{p}$

p_{max} : $t_{calc} = t_{com} \rightarrow p_{max} = \frac{B c_0\, CPI\, t_0}{b_0}$

Non-optimal speedup if n_0 (parallel tasks) close to or smaller than p (# processors). Can't assign fraction of task to PE.

→ distribution loss

10. Communication

LAN / SAN : Local / System Area Network

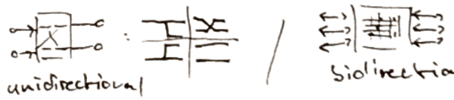
Properties: BW, latency, topology, scalability

Performance: Min/Avg/Max latency, BW, scalability

Bisectional BW: min BW between 2 ~~nodes~~ halfes of net, often $\frac{BW}{PE}$

Switch based

- store-fwd
- cut-through
- wormhole



Routing Data in packets with header for routing info

src based : pkt source tells how to route

dst based : pkt has ID and network finds destination

src: + simple SW HW + simple routing
- no multicast
- src must know the entire net

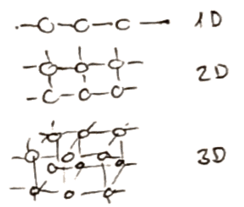
dst: + multicast + adaptive
+ src must not know topology
+ no proc on src
- complex routers

Perf incr: - Uni/Multi/Broadcast
- Channel Bundling: two or more links btw switches

Problems: Deadlocks, fault handling

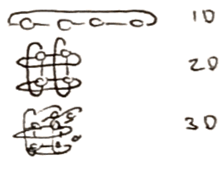
Topologies

n-Dimensional mesh



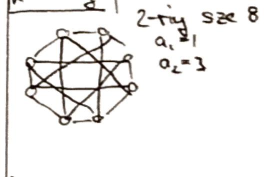
- + short phys distances
- + no size limits
- + latency scales with "d-th root of net size
- + cost scales w/ size
- + multi path to dest
- + size incr. in small steps
- − BW scales with nth root of net
- − multicast → heavy processing
- − deadlock prevention necessary

d-Dim. Torus



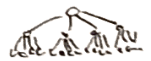
- + no size limit
- + latency scales n-th root of size
- + cost scale w/ size
- + multi path to dest
- + size incr. in small steps
- − BW scales w/ n-th root of net size
- − multicast heavy proc
- − deadlock prevent
- − long cable for wrap-around

k-Ring



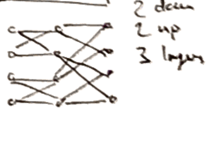
- + no size limit
- + latency scales w/ size
- + cost scales w/ size
- + multi path to dest
- + small step size incr.
- − BW scales not with size
- − multicast heavy proc
- − long cables
- − monolithic, hard to partition

tree



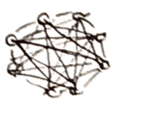
- + no size lim
- + latency scales w/ size
- + no deadlock
- + fast & simple broadcast
- − BW does not scale w/ size
- − single point of failure
- − cost does not scale w/ size
- − long cable at tree root
- − size incr only in large steps

fat tree



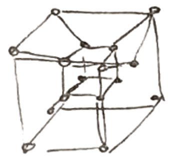
- + no size limit
- + latency scales w/ size
- + BW scales w/ size
- + Multi path to dst
- + no deadlock
- + fast + simple multicast
- − cost not scale w/ size
- − long cable at tree root
- − size incr only in large step
- − expensive

fully connected



- + const low latency
- + cost scale w/ size
- + size incr. in small steps
- + no deadlock
- + BW scales w/ size
- + multi path to dest
- + fast simple broadcast
- − limited size
- − very expensive
- − trade off : nodes ↔ net size

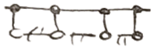
Hypercube



+ no size limit
+ latency scale w size
+ BW scale w size
+ cost scale w size
+ multi paths
+ simple multicast

- BW not scale
- Deadlock prev
- size incr in large steps
- long cables
- trade off nodes <-> size

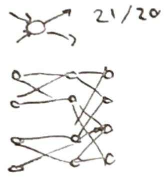
Bus



+ low cost
+ fast stable latency
+ small step incr
+ fast simple broadcast

- BW not scale
- limited to short interconn
- single point of fail
- limited size

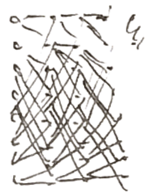
Multistage



+ no size lim
+ latency scale
+ BW scale
+ no deadlock
+ fast simple broad

- cost x scale
- long cable
- every SW point of fail
- large step incr.

Omega/Shuffle



+ no size lim
+ lat scale w size
+ BW scale w size
+ no deadlocks
+ fast simple broad

- cost not scale with size
- long con
- every SW is point of fail
- large step size incr.

		Mesch	Torus	k-Ring	Tree	Fat tree	Fully connected	Hypercube	Bus	Multistage	Omega/shuffle
Scales with size	Cost	✓	✓	✓			✓	✓	✓		
	Latency	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Bandwidth					✓	✓	✓		✓	✓
No size limit		✓	✓	✓	✓	✓		✓		✓	✓
No deadlocks					✓	✓	✓			✓	✓
Fast+Simple Multicast					✓	✓	✓	✓	✓	✓	✓
Short cables		✓									
No single point of failure		✓	✓	✓				✓			
Small step increments		✓	✓	✓			✓		✓		
Multi path to hosts		✓	✓	✓		✓	✓	✓			
No node<->size tradeoff		✓	✓	✓	✓	✓					

13. Fehlerredundante Computerarchitektur

Fehlerunterscheidung

Beu[illegible]: Rechner/Sensor liefert keine oder klar ungültige D[illegible]n

Byzantine: —"— liefert falsche, plausible Daten

Messages

OM (Oral): können gefälscht werden, einfach implementiert

SM (signed): nicht fälsch- aber unterdrückbar