

Great ideas in CA: Abstraction; Make common case faster; Parallelism; Pipeline; Redundancy for accuracy; Performance Measurement; Amdahl's law: $\text{加速比 } S(W) = \frac{1}{1 - p + p\%}$. p : 并行化比例

Noore's Law: chips \downarrow transistors 数量 \uparrow 延时 \downarrow Δ : prediction, not law

Memory Hierarchy: CPU \rightarrow CPU cache \rightarrow physical memory \rightarrow solid state memory \rightarrow virtual mem

reg: $\#$ on-chip cache(L1,2,3), main mem: RAM, 地址转换; SSD, 内存 \uparrow 速度 \downarrow , 但

Info-representation: LSB: 最右; MSB: 最左: $ob[0 \dots 1]$

表示负数: 2's complement: MSB \rightarrow sign bit; 其余位翻转后加1: $(a_0, a_1, \dots, a_n)_2 = -$

*: 不论 signed or unsigned, 操作 + - 结果 可能造成 overflow! underflow!(\downarrow 在 subnormal 附近)

fP32: $(-1)^S \times \text{Mantissa} \times 2^E$, MLE 在不同 type 下算法不同: normal: bias ≥ -127

Exp Fraction Value

$M = (1.\text{Fraction})_2$, $E = (\text{Exp})_{10} - 127$ Exp 不等于 1 !!

[Range]: normal: $\pm 1.00 \dots \times 2^{-126} \sim \pm 1.11 \dots \times 2^{127} \uparrow$

全0 非0 Subnormal: $\text{subnormal: } \pm 0.00 \dots 1 \times 2^{-126} \sim \pm 0.11 \dots 1 \times 2^{-126} \uparrow E = 2^{-2} t^{-1}$

全0 非0 Null

Zero

全0 非0 Subnormal: $\text{subnormal: } \pm 0.00 \dots 1 \times 2^{-126} \sim \pm 0.11 \dots 1 \times 2^{-126} \uparrow E = 2^{-2} t^{-1}$

Core: ① Exp 全1 与 全0, 有特殊意义. **牢记!** ② bias: $\text{if Exp} \neq tbit, \text{则 } bias = 2^{-t} - 1; \text{subt: } \Rightarrow$

C: ① 宏的括号问题 ② True or False' in C: False \Leftrightarrow 0<int; True \Leftrightarrow False.

③ Sizeof type: char short int long int unsigned int void* size_t float double

④ Little Endian: $\boxed{\begin{array}{ccccccccc} 1 & 1 & 2 & 2 & 4 & 4 & 8 & 4 & 4 \end{array}} \sim \boxed{\begin{array}{ccccccccc} 3 & 2 & 1 & 7 & 8 & 7 & 6 & 5 & 4 \end{array}}$ 地址从低到高, HLSB至MSB放bit! 如左, 该函数: $0x12345678$

⑤ structure: alignment ⑥ func 传递参数 改善于 func, 使其指针

⑦ Mem management: stack: func() 中局部变量 heap: malloc/calloc/realloc, free

⑧ 只有 Array 或 指向转化成 pointer, 反之不行! func 传递 arr, arr 转化为 pointer

RISC-V: R-type: XRSU XRSU 间加减位运算(比较, 将 XRSU 返回给 rd; 还可是 shift 操作

shift中: SLL, 左移, 必用0填; SRL, 右移前面必用0填, 而 SRA, 前面用符号位填充!!

移动多少位? RS2 中低位所代表的十进制数 (unsigned); 除此之外, 操作如右移, + - 等, 都是进行 signed operation (除非指针声明是视作 unsigned 来操作)

I-type: instr rd, rs1, imm; instr rd, imm(rs1) 后者为 load 指令, 前者并非 load 指令

Warning **⚠️** RISC-V 中所有 imm 都是 signed。0.00 (2byte, 1byte)

指令操作 imm 时, 0.00 用 imm; 1.00 用 imm (I-type 中只有 s(l)t(c)); 1.16 用 byte

5 halfword 5 MSB 填充至32bit, 4byte, 布入 rd, rs1, ls1, ls2 用0填充

S-type: swl/sw/sh RS2, imm(rs1). 将 RS2 打到 word/bit/byte/word 并进 addr. 而

addr = $\pi[\text{RS2}] + \text{imm}$; Δ : sb, sh 不进行 sign-extension! 直接从4,8位直接去, 0填充

B-type: instr rs1, rs2, L, imm(label), go to PC+offset if ...条件, else PC+4

offset = imm / offset = (label - current PC)

Δ : 为了保证跳转是 2-byte aligned, 故 offset 最后一位会强制 0, 把 [12:1] 编码

这样, 跳转指令范围为: $\pm 2^{10}$ 条 offset 3位, MSB 表示, 12位 $\Rightarrow 2^{12}$ byte $\Rightarrow 2^{10}$ 条

offset = imm or (label - PC); 为保证 2-byte aligned, offset 最后一位归 0, [20:1] 编码 \Rightarrow

jal rd, label PC+4 给 rd, jump to label, i.e., PC = PC + offset, offset = label - PC

Δ : 相当于 jal rd $\&$ 译在 RS1 中地址上跳转, jal 只能在 PC 基础上 jump

Pseudo-jlabel \Rightarrow jal x0 label; jal label \Rightarrow jal ra label; jr rs1 \Rightarrow jal x0 rs1 0

U-type: lui/lui/auipc rd, imm, lui: rd \leftarrow imm << 12; auipc, rd \leftarrow PC + imm << 12

Δ : li xi, imm \Rightarrow lui xi, imm[31:12] + addi xi, xi, imm[1:0] OK!

Misc: imm 范围: 可以看 greenland inst组成中, imm 显示的最高位是多少, 便对应

Calling convention: 约定: ① 传参数 A 用 reg 系列 ② 返回值放在 A0/A1 reg

③ caller & callee: A func 调用 B func 中的运算值, 然后按照①, 把给 B 的参数放在 A 中

值存起来 (SP&SW), Δ A func 中的运算值, 然后把它们便供 B 自由

之后调用并进入 B, -进 B 要把 callee-saved reg 中存起来 (SP&SW), 它们便供 B 自由

使用, 然后把返回值放入 A0, A1; 恢复 callee-saved reg (SP&SW) 返回至 A; A 恢复

caller-saved reg (SP&SW), 可使用之前 A0, A1 中的 B 返回值; 跳至 n 中的 PC

Prologue: 存 ra, 有 callee-saved reg, SPV; Epilogue: 恢复 ra, SP callee-saved reg, SPV,

imm 英国补: I-type: 除 SLL, SRL, SRA 为 signed/unsigned, 其余均 12 位 signed

Δ : 在转换为机器码时, 它们有类似于 function 的编码, 体现了 imm[11:5]

S-type: 12位 signed; SB/B-type: 13位 signed; U: 20位 unsigned; V: 21位 signed

ISA: X86-32 (IA-32) X86-64 (AMD64). ARM (Advanced RISC Machine)

RISC; ISA 定义 特定 CPU 支持怎样的 operation, 以及如何 implement, 即: 什么 CPU 支持

什么汇编, 如 X86 汇编可在 X86 架构 CPU 跑, 但不能在 ARM CPU 跑。X86-arm.

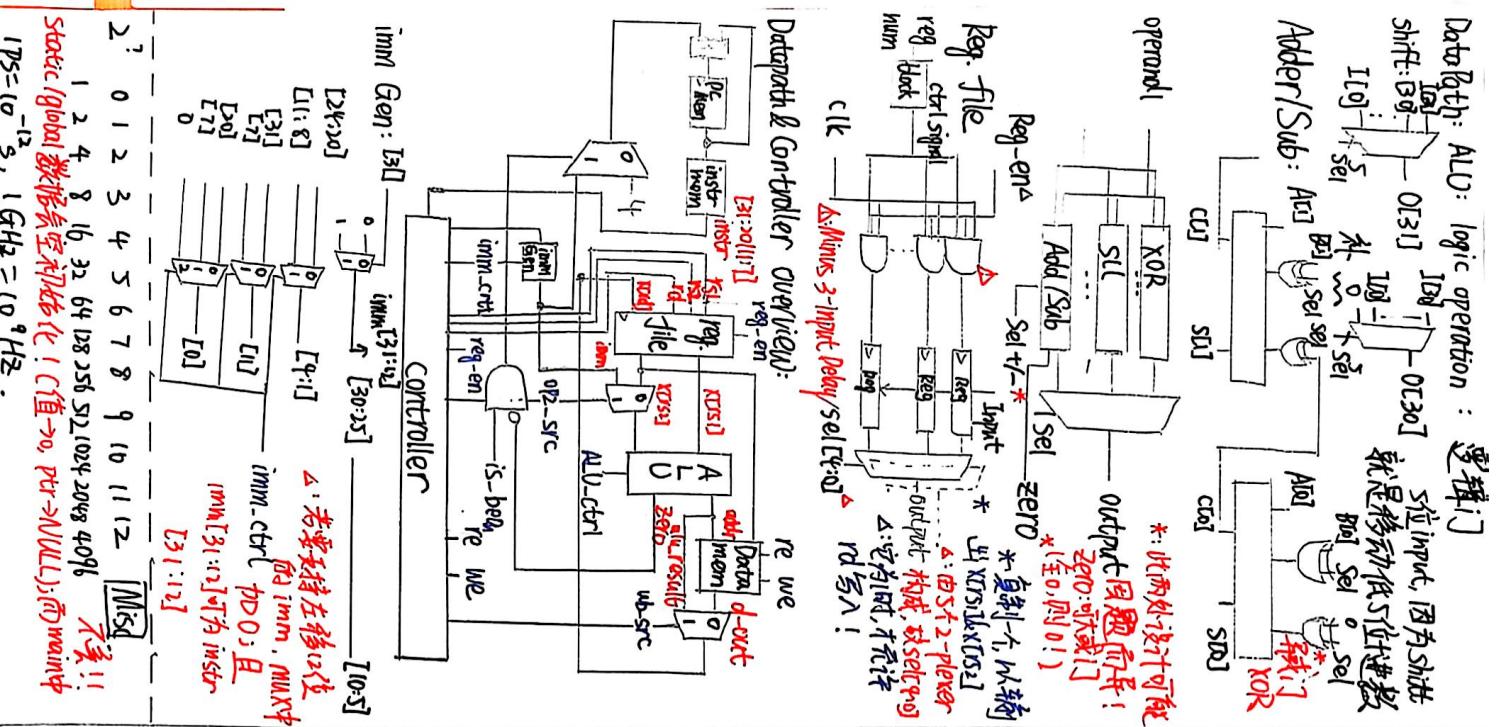
指令集 \Rightarrow CPU 架构 = ISA \Rightarrow 搭配 OS 和 Linux 支持 RISC-V ISA

Δ : Python 在不同 OS 不同 ISA 上运行, 只要有合适的 Python 解释器

Δ : 一个 OS 上的可执行文件一般不能在另一个 OS 上运行 2'signed [-1048576, 1048575]

Imm: 12signed: [-2048, 2047] 3unsigned: [0, 3] 13signed [-4096, 4095] 2unsigned [0, 1048575]





Pipeline: $\frac{\text{Time}}{\text{Program}} = \frac{\text{Instr}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instr}} \cdot \frac{\text{Time}}{\text{Cycle}}$ CPI

5-stage: IF ID EX MEM WB, Timing ~~最好~~ 唯延时最长

两个 stage 间加一层 reg, 但有 hazard: ① Structural: 如果 reg-file 能否双读写, IF/DMEM 何时双读又写 \Rightarrow 硬件解 type; IF/DMEM 何时双写 \Rightarrow 硬件解 type; ② Data: reg-file write after read (未刷 updated), 种是 NOP (无操作) 等一种是 ~~死锁~~ 死锁

Type 2: LW 后一条指令就读 (load 来源不缓冲), 种是等 (nop) + Dmem load 的数直接给 ALU, 另一种是按 instruction order, load 与 EX 使用 interlock 中间至少隔一条 (rescheduling)

③ Control: 关键: 满足条件则跳转至..., 若满足, 则在 branch 后四年岂不就行了? 一种是等 4 年 NOP; 另一种是若跳转到那四年, flush 掉, RP restore state; 另一种是将 2010 线程作用由阶段没在 EX/MEM (P: zero -> 直接 WB 从 CP; R: MEM 从 CP)

Timing: R-type: IF + DEC + EX + WB C DEC = ID

I-type: 其中 DEC = max { treg, tImm + tMux }

IF + DEC + EX + WB + ALU

S-type: IF + DEC + EX + WB = max { treg + tMux + tALU }

B-type: IF + DEC + EX + WB = max { treg + tMux + tALU + tCond }

*: IF & WB \rightarrow no; B&WB: imm \rightarrow PC reg WB = tMux + tALU

MISC: C 中位运算, 与 R 或 C 并或八取反 ~ 左右移 >><<

STH 指令 add/subt signed/unsigned 数域中是否 overflow:

unsigned : { add x2, x1, x3 $\Delta: a = b \oplus c$ (unsigned),
 sth x4, x2, x1 or x3. $\Delta: a \leftarrow b \oplus c \Rightarrow$ 送
 signed : { add to, t1, t2 $\Delta: signed: a \leftarrow b \oplus c: if t1 < 0, a \leftarrow b \oplus c: if t1 \geq 0, if t1 \neq 0 \Rightarrow overflow$
 sti t3, t2, 0.
 sti t4, to, tr
 bne t3, t4, overflow
 or (xor ts, t3, t4; ts = 1: overflow)
 宏定义中一定要符号充分打上!! 如
 #define MIN(a,b) ((a)>(b)) ? (b):(a)
 NMOS
 PMOS

Digital Circuit: $\neg D$ = $\neg D$ 与 $\neg D$ * 或 $\neg D$ 同或 [* 相同为不相同]

Combinational Logic: $\neg D$ = $\neg D$ 与 $\neg D$ * 或 $\neg D$ 同或 [* 相同为不相同]

CFB: Combinational Function Block

Timing: $t_{\text{clk-to-Q}} + t_{\text{CFB}} + t_{\text{setup}} \leq \text{min period} = 1/\text{fns}$

$t_{\text{clk-to-Q}} = 10^{-9} \text{ s}$
 $1000 \text{ MHz} = 1 \text{ GHz}$

CALL: ~~代码生成~~ → ~~汇编器~~ → ~~链接器~~ → ~~机器码~~

loader → 放至内存; pieces of obj file

More: 输出仅于当前状态有关; Mealy: 与先前状态与输入有关

它们相互依赖; Their outputs are correct in one step circuit

△ 进内存后, 分离依赖于绝对路径的 instr & data words

step 2: pseudo→normal instr & tail call instr: 主要由器完成

同时, track label addr & data transfer instr → tail summary table

③ Data: static ④ Symbol table: files label ⑤ Reallocation table

step 1: read use directive. Text → ②, data → ③, etc. 不产生新表

linker: obj-file/symbol table → 链接阶段: 将许多.0 文件整合至一个执行文件; ~~将 code & data module 打包起来~~

然后将 data & instr labels fixaddr, 最后将所有相对 addr 转为 absolute addr

① External Ref (e.g. ijal) → always relocate

② static data ref (an ipc/lnibaddr) → reallocate

△: PC related, e.g. breq, bne, jal, never need to fill in? linker to each text data offset, no order, 则重新算 label & data 在 .o 文件中绝对地址 与 symbol table Loader: 读 .o 文件 header, 读 .text, copy, riprel, riplq, set PC

Boolean: $X\bar{X}=0$ $X\bar{O}=0$ $X\bar{I}=1$ $XX=X$ ($X\bar{Y}=X(X\bar{Y})$)

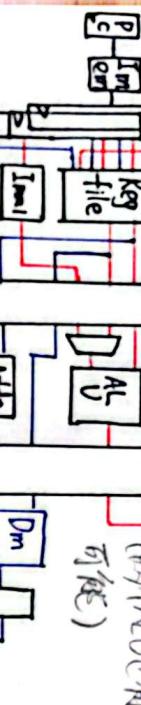
$X(X+Z)=X+XZ$ $X\bar{X}+X=X$ $\bar{X}\bar{Y}=\bar{X}+\bar{Y}$

$X\bar{X}=1$ $X\bar{I}=1$ $X+0=X$ $X+X=X$ $X(X+Z)=X(X+Z)$

$(X\bar{Y})X=X$ $X\bar{Y}=\bar{X}\bar{Y}$ 保险: $\bar{X}V$ 来推通

Multi-issue: 原来 single-issue, datapath 中最多只用一个 slot
issue 1 指令，现在一次选多条。

CPI 可能 < 1!
(两个分支有可跳)



If ID EX MEM WB

Red line: Arithmetic, logic and branch path

Blue line: Memory access (L1) path

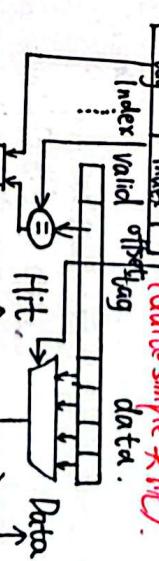
实现中：
① 硬件中，若库更新，要从头重来！且 it includes entire
② even if not all of it will be used.

Dynamic: 库文件在程序运行时会动态地加载，而非被纳入exe文件中。占用更多空间，内存更高效。库可独立更新。但运行时产生额外开销以让程序查找与加载所需库文件。

③ CALL 中 static vs. Dynamic Linking:
Static + Address resolve, 若 symbol table 中没有，则会搜 library files. 没错，库被编译进了执行文件中。若库更新，要从头重来！且 it includes entire
④ 最常使用；N-1: LRU; 选 LRU 数据最大的块换。LRU 上数据：hit 或作了 victim, LRU → 0; - 该块中未命中该 cache, LRU → 1 (LRU FIFO oldest line LRU newest line LRU)
Write Policy: Write-through: write to cache and memory at the same time (longer time)

Cache: 核心概念 - 缓存
Locality: Temporal: If a memory location is referenced, then it will tend to be referenced again soon.
Spatial: If .. referenced, addr nearby tend to be... soon.
- 一些数据结构如 linked list, tree, 以及 locality 差！
Cache Blocking: set a dirty bit to 1; when blocks get replaced, and dirty bit is 1, write to mem, dirty bit → 0.

Write-around: Directly write to mem



Cache Misses: 3Cs

Compulsory: cold start or process migration

First access to block, impossible to avoid!

Sol: Block Size ↑ (i.e., N), so fewer cold starts

Capacity: Cache can't contain all blocks accessed by program

Sol: Cache Size ↑ (but access time ↑)

Conflict (Collision): Multiple memory locations mapped to the same cache location, conflict even when the cache has not reached full capacity

Sol: Cache size ↑ Associativity (N) ↑ (But access time ↑)

④ SIMD: Data level Multi-thread: (MMO): Thread-level Parallel!: Pipeline & multi-issue : instr-level

Direct Mapping: 1-way !!

tag index offset

Other util in cache: valid bit, an indicator to tell if each entry is valid for program

LRU: (least recently used) 满员情况，

要找 victim 找成新元素 → 最不常用的先淘汰

→ 0; - 该块中未命中该 cache, LRU → 1 (LRU FIFO oldest line LRU newest line LRU)



#pragma omp barrier: Forces all threads to wait until all threads have hit the barrier

pragma omp critical: Creates a critical section within time.

pragma omp parallel for: for-loop automatic work-sharing

pragma omp parallel reduction(operation: var)

pragma omp single: code block executed by one thread only. Other threads will wait. I/O操作中常见

pragma omp master: 在线程执行，但其他线程不等它完成

pragma omp parallel: 有多个 processor，它们执行独立的指令流。且不同

processor 连动方法：① Shared Var in memory (load/store instruction) 并行。但共享 DRAM 与 L3 cache (带#). 不同

pragma omp parallel reduction(operation: var)

pragma omp parallel for: for-loop automatic work-sharing

pragma omp parallel reduction(operation: var)

同理也有 Exclusive: $L_n \cap L_m = \emptyset (n \neq m)$

不-inclusive \Rightarrow non-inclusive.

OSR 10: OS 功能: ①开机后提供 service to file system

② Loads, runs and manages programs

③ I/O with the rest of computer

④ File I/O: CPU 从地址执行指令 (in flash ROM 中存的)

Memory mapped I/O: certain addresses are not regular mem, but correspond to registers in I/O device. I/O 需通过读写字节流来进行操作。部分处理器有特定 I/O 指令。

挑战: 处理器 I/O 周期与设备速率差 9 个数量级

解决方案: ① Polling: CPU 循环读取控制寄存器，等待 ready bit (0 \rightarrow 1)，然后读写 ② Interrupt: I/O 准备好时

Term: Interrupt: 外部事件 (如按键), 干扰: 可能被处理器捕获，暫停程序，转至 OS 陷阱处理程序

Exception: 处理中事件 (如 page fault). 同步: 必在弓箭射出前，暂停程序，转至 OS 陷阱处理程序

异常: 异常 (trap handler) 处理 Exception

中断: 中断 (异步暂停指令). 跳转至处理器并保持状态。陷阱前指令完成，后指令未执行。SPEC 规定中断地址，简化处理

Demanding Page: Only load a page into memory if user requests it.

Status Bit: 访问时，先检查 PT 该 entry 的 status bit

若 valid，则往; 若 invalid, page fault exception. 再若 DRAM 空间不够, evict, victim status bit \rightarrow 0, 且收回 disk; 然后, 磁盘响应 page 赋入 DRAM, 且 VPN 对应 PPN 更新, status bit 为 1 (valid)

* Page table & Disk copy 来自 Page table in main memory

Dirty Bit: 与入 disk, 采取 write-back 缓存。Page table 时, 若 Dirty Bit 为 1, 则把新内容写回 Disk 及其物理页表

每个进程都有独立的 PT, 限制其虚拟地址空间, PT 由 OS 管理。

限制范围: PT 由 OS 管理。

context switching; 用户进程请求 OS 服务如文件操作、进程创建, 其调用内容由内核执行; 同时防止

进程互相覆盖内存, 虚拟内存提供特权进程间通信

Virtual Memory: Process 用 virtual address, memory 用 physical address. Memory manager: virtual \rightarrow Physical 地址: Memory Fragmentation: 程序 Ta, Storage Hierarchies

在层级内存管理中: DRAM \rightarrow Disk: V \rightarrow P address 在硬件下帮助实现 (TLB, a cache).

TLB, Cache some 地址都存 in TLB, 保存的是 VPN - PPN mapping. TLB is much closer to CPU

and caches. While TLB 有 PPN+Flags!

Paged/Memory: DRAM 分割为固定大小的页面。通

过页机制从磁盘中加载到内存 (整个页面); Page Table: 表, 将 Virtual 地址映射为 Physical 地址。表中则 Page-Table Walk, 并将虚地址写入 TLB (命中更新后)。

若所有高级缓存块也都存在缓存中, 则称低级缓存 inclusive.

则那个 VPN 就 \rightarrow PPN (Physical Page Number). 若还未对应用, 则让 OS trigger page fault to load page from disk. VPN \rightarrow PPN, addr 中还有 offset. 直接 copy, 与 PPN 结合为 Physical Addr. (PA)

△ Each User has a page table

▲ ▲ 普通 Page Table 中, 只是必要操作; DRAM

(物理大小: 2 VM-bit \times 4Byte)。而 PPN 是 index+offset

TLB 是 cached \rightarrow 一个 VPN 先看 TLB 是否 Hit, 若 miss

则 Page-Table Walk, 并将虚地址写入 TLB (命中更新后)。

TLB 有 Fully Associativity, 策略有 FIFO, random 等

TLB Reach: How many VA 可被映射到物理地址:#TLB entry size
Only one TLB per core, but page tables are per process
VA management 允许把程序加载到任意 physical memory space.
Single-core processor 无需维持 cache-coherence.

Misc: More I/O: CPU 与 I/O 设备同步方式
高开销
Polling: CPU 不断查询, Interrupt: 设备通知 CPU, 但须先由 CPU 处理
Reduce: 中间键值对聚合到一个 key
数据从设备传输至 DRAM ① 使用数据计算, 但 CPU 被占用
直接内存访问 (DMA): DMA (I/O 设备直接访问 DRAM, 避免 I/O 次数)
CPU 负担。DMA 由 CPU 的 register。传输过程:
① CPU 接受设备中断, 初始化 DMA 引擎 ② 它处理数据传输
CPU 执行其它任务 ③ 传输完成后, 再次中断 CPU
优势: 释放 CPU 资源, 适合 **高数据率设备**

Redundancy for Accuracy: 依 Amdahl: 系统可靠性和冗余决定最高系统可靠性。冗余方式: 空间、时间、信息冗余
错误检测: 奇偶校验: 附加位使 i 为奇数或偶数 (偶校验) 或偶 (偶校验) \Rightarrow 单比特错误
Hamming ECC: Hamming 距离是对应数不同位的最小数
加额外校验已足够 \Rightarrow 1-bit 错误 2-bit 错误
磁盘冗余: 优势: 提高数据可用性, 防止硬件问题
异构计算: 利用多种处理器或核心通过加速器提供统一 API
FPGA: 现场可编程门阵列, 通过硬件描述语言实现特定逻辑 (从硬件电路下手!)

CPU

* 绝足够能资源, FPGA 可实现任意逻辑, 包括 RISC-V

后三章核心: DMA: 除了上述之外: 访问冲突解决方式:

Burst: 传 Data Block 间, CPU 无法访问内存

Cycle Stealing: DMA 在 1 个字节后释放总线权与 CPU 交替访问

Transparent Mode: DMA 仅在 CPU 不使用总线时传输

Networking: 类型: shared: -> device switched: -> device

其 Software protocol: Send: Application data, OS buffer, OS 会话, 后续定时器, Buffer Data, 网络接口

receive: 网络接收到 Buffer Data, 算校验和, if OK send: ACK

否则丢弃并重传; Buffer Data 地址空间, 且通知网

More Parallelism: Request-level: Web 服务, 每个请求并发

Data level: SIMD, on WSC, MapReduce & Scalable file sys

Map Reduce: 用于 large scale data 行处理, 将 Data 处理分为 Map 与 Reduce 两个阶段

Map: 输入 Data 为 key-value pairs, 并行处理中间

Reduce: 中间键值对聚合到一个 key

Map Reduce 中间有 shuffle: 将 map 侧的中间 key 按键分组, 确保同一 key pair 发送给同一个 reduce 任务

Hamming ECC: 相信我的大脑已经全满程 (左→右)

RAID: 独立磁盘冗余阵列。RAID 0: 无冗余

RAID 1: 镜像 RAID 3: parity Disk 3: strip RAID 4: parity + block level RAID 5: 4+上 + interleaved

IAFR = $\frac{N \text{ disks} \times 8760 \text{ hrs/year}}{\text{MTTF} \times \text{N disks}}$

MTB between F = MTTF * IAFR + MTTR

Availability = MTTF / (MTTF + MTTR).

Security: Heartbleed: OpenSSL 漏洞, 可读内存缓

冲区, 泄露密钥; Rookhammer: 寻找 DRAM 漏洞, 通过物理干扰引起位翻转; Side-channel: 利用系统中物理行为 (如时钟, 缓存访问, 功耗, 声音) 泄露信息

诱导处理器加载受害者数据至缓存, 再利用道提取

访问的内存。Spectre: 利用推测执行漏洞, 允许程序读取不应

该读取的内存。Meltdown: 利用乱序执行漏洞, 允许程序读取不应

该读取的内存。Speculative Execution 攻击, 分支预测, 道提取

中: 宏元类型; const 与 #define 速度大致一样

Map: 将输入 Data 为 key-value pairs, 并行处理中间

Reduce: 中间键值对聚合到一个 key

Map Reduce 中间有 shuffle: 将 map 侧的中间 key 按键分

组, 确保同一 key pair 发送给同一个 reduce 任务

Hamming ECC: 相信我的大脑已经全满程 (左→右)

RAID: 独立磁盘冗余阵列。RAID 0: 无冗余

RAID 1: 镜像 RAID 3: parity Disk 3: strip RAID 4: parity + block level RAID 5: 4+上 + interleaved

IAFR = $\frac{N \text{ disks} \times 8760 \text{ hrs/year}}{\text{MTTF} \times \text{N disks}}$

MTB between F = MTTF * IAFR + MTTR

Availability = MTTF / (MTTF + MTTR).

Security: Heartbleed: OpenSSL 漏洞, 可读内存缓

冲区, 泄露密钥; Rookhammer: 寻找 DRAM 漏洞, 通过物理

干扰引起位翻转; Side-channel: 利用系统中物理

行为 (如时钟, 缓存访问, 功耗, 声音) 泄露信息

诱导处理器加载受害者数据至缓存, 再利用道提取

访问的内存。Spectre: 利用推测执行漏洞, 允许程序读取不应

该读取的内存。Meltdown: 利用乱序执行漏洞, 允许程序读取不应

该读取的内存。Speculative Execution 攻击, 分支预测, 道提取

中: 宏元类型; const 与 #define 速度大致一样

Map: 将输入 Data 为 key-value pairs, 并行处理中间

Reduce: 中间键值对聚合到一个 key

Map Reduce 中间有 shuffle: 将 map 侧的中间 key 按键分

组, 确保同一 key pair 发送给同一个 reduce 任务

Hamming ECC: 相信我的大脑已经全满程 (左→右)

RAID: 独立磁盘冗余阵列。RAID 0: 无冗余

RAID 1: 镜像 RAID 3: parity Disk 3: strip RAID 4: parity + block level RAID 5: 4+上 + interleaved

IAFR = $\frac{N \text{ disks} \times 8760 \text{ hrs/year}}{\text{MTTF} \times \text{N disks}}$

MTB between F = MTTF * IAFR + MTTR

Availability = MTTF / (MTTF + MTTR).

Security: Heartbleed: OpenSSL 漏洞, 可读内存缓

冲区, 泄露密钥; Rookhammer: 寻找 DRAM 漏洞, 通过物理

干扰引起位翻转; Side-channel: 利用系统中物理

行为 (如时钟, 缓存访问, 功耗, 声音) 泄露信息

诱导处理器加载受害者数据至缓存, 再利用道提取

访问的内存。Spectre: 利用推测执行漏洞, 允许程序读取不应

该读取的内存。Meltdown: 利用乱序执行漏洞, 允许程序读取不应

该读取的内存。Speculative Execution 攻击, 分支预测, 道提取

中: 宏元类型; const 与 #define 速度大致一样

Map: 将输入 Data 为 key-value pairs, 并行处理中间

Reduce: 中间键值对聚合到一个 key

Map Reduce 中间有 shuffle: 将 map 侧的中间 key 按键分

组, 确保同一 key pair 发送给同一个 reduce 任务

Hamming ECC: 相信我的大脑已经全满程 (左→右)

RAID: 独立磁盘冗余阵列。RAID 0: 无冗余

RAID 1: 镜像 RAID 3: parity Disk 3: strip RAID 4: parity + block level RAID 5: 4+上 + interleaved

IAFR = $\frac{N \text{ disks} \times 8760 \text{ hrs/year}}{\text{MTTF} \times \text{N disks}}$

MTB between F = MTTF * IAFR + MTTR

Availability = MTTF / (MTTF + MTTR).

Security: Heartbleed: OpenSSL 漏洞, 可读内存缓

冲区, 泄露密钥; Rookhammer: 寻找 DRAM 漏洞, 通过物理

干扰引起位翻转; Side-channel: 利用系统中物理

行为 (如时钟, 缓存访问, 功耗, 声音) 泄露信息

诱导处理器加载受害者数据至缓存, 再利用道提取

访问的内存。Spectre: 利用推测执行漏洞, 允许程序读取不应

该读取的内存。Meltdown: 利用乱序执行漏洞, 允许程序读取不应

该读取的内存。Speculative Execution 攻击, 分支预测, 道提取

中: 宏元类型; const 与 #define 速度大致一样

Map: 将输入 Data 为 key-value pairs, 并行处理中间

Reduce: 中间键值对聚合到一个 key

Map Reduce 中间有 shuffle: 将 map 侧的中间 key 按键分

组, 确保同一 key pair 发送给同一个 reduce 任务

Hamming ECC: 相信我的大脑已经全满程 (左→右)

RAID: 独立磁盘冗余阵列。RAID 0: 无冗余

RAID 1: 镜像 RAID 3: parity Disk 3: strip RAID 4: parity + block level RAID 5: 4+上 + interleaved

IAFR = $\frac{N \text{ disks} \times 8760 \text{ hrs/year}}{\text{MTTF} \times \text{N disks}}$

MTB between F = MTTF * IAFR + MTTR

Availability = MTTF / (MTTF + MTTR).

Security: Heartbleed: OpenSSL 漏洞, 可读内存缓

冲区, 泄露密钥; Rookhammer: 寻找 DRAM 漏洞, 通过物理

干扰引起位翻转; Side-channel: 利用系统中物理

行为 (如时钟, 缓存访问, 功耗, 声音) 泄露信息

诱导处理器加载受害者数据至缓存, 再利用道提取

访问的内存。Spectre: 利用推测执行漏洞, 允许程序读取不应

该读取的内存。Meltdown: 利用乱序执行漏洞, 允许程序读取不应

该读取的内存。Speculative Execution 攻击, 分支预测, 道提取

中: 宏元类型; const 与 #define 速度大致一样

Map: 将输入 Data 为 key-value pairs, 并行处理中间

Reduce: 中间键值对聚合到一个 key

Map Reduce 中间有 shuffle: 将 map 侧的中间 key 按键分

组, 确保同一 key pair 发送给同一个 reduce 任务

Hamming ECC: 相信我的大脑已经全满程 (左→右)

RAID: 独立磁盘冗余阵列。RAID 0: 无冗余

RAID 1: 镜像 RAID 3: parity Disk 3: strip RAID 4: parity + block level RAID 5: 4+上 + interleaved

IAFR = $\frac{N \text{ disks} \times 8760 \text{ hrs/year}}{\text{MTTF} \times \text{N disks}}$

MTB between F = MTTF * IAFR + MTTR

Availability = MTTF / (MTTF + MTTR).

Security: Heartbleed: OpenSSL 漏洞, 可读内存缓

冲区, 泄露密钥; Rookhammer: 寻找 DRAM 漏洞, 通过物理

干扰引起位翻转; Side-channel: 利用系统中物理

行为 (如时钟, 缓存访问, 功耗, 声音) 泄露信息

诱导处理器加载受害者数据至缓存, 再利用道提取

访问的内存。Spectre: 利用推测执行漏洞, 允许程序读取不应

该读取的内存。Meltdown: 利用乱序执行漏洞, 允许程序读取不应

该读取的内存。Speculative Execution 攻击, 分支预测, 道提取

中: 宏元类型; const 与 #define 速度大致一样

Map: 将输入 Data 为 key-value pairs, 并行处理中间

Reduce: 中间键值对聚合到一个 key

Map Reduce 中间有 shuffle: 将 map 侧的中间 key 按键分

组, 确保同一 key pair 发送给同一个 reduce 任务

Hamming ECC: 相信我的大脑已经全满程 (左→右)

RAID: 独立磁盘冗余阵列。RAID 0: 无冗余

RAID 1: 镜像 RAID 3: parity Disk 3: strip RAID 4: parity + block level RAID 5: 4+上 + interleaved

IAFR = $\frac{N \text{ disks} \times 8760 \text{ hrs/year}}{\text{MTTF} \times \text{N disks}}$

MTB between F = MTTF * IAFR + MTTR

Availability = MTTF / (MTTF + MTTR).

Security: Heartbleed: OpenSSL 漏洞, 可读内存缓

冲区, 泄露密钥; Rookhammer: 寻找 DRAM 漏洞, 通过物理

干扰引起位翻转; Side-channel: 利用系统中物理

行为 (如时钟, 缓存访问, 功耗, 声音) 泄露信息

诱导处理器加载受害者数据至缓存, 再利用道提取

访问的内存。Spectre: 利用推测执行漏洞, 允许程序读取不应

该读取的内存。Meltdown: 利用乱序执行漏洞, 允许程序读取不应

该读取的内存。Speculative Execution 攻击, 分支预测, 道提取

中: 宏元类型; const 与 #define 速度大致一样

Map: 将输入 Data 为 key-value pairs, 并行处理中间

Reduce: 中间键值对聚合到一个 key

Map Reduce 中间有 shuffle: 将 map 侧的中间 key 按键分

组, 确保同一 key pair 发送给同一个 reduce 任务

Hamming ECC: 相信我的大脑已经全满程 (左→右)

RAID: 独立磁盘冗余阵列。RAID 0: 无冗余

RAID 1: 镜像 RAID 3: parity Disk 3: strip RAID 4: parity + block level RAID 5: 4+上 + interleaved

IAFR = $\frac{N \text{ disks} \times 8760 \text{ hrs/year}}{\text{MTTF} \times \text{N disks}}$

MTB between F = MTTF * IAFR + MTTR

Availability = MTTF / (MTTF + MTTR).

Security: Heartbleed: OpenSSL 漏洞, 可读内存缓

冲区, 泄露密钥; Rookhammer: 寻找 DRAM 漏洞, 通过物理

干扰引起位翻转; Side-channel: 利用系统中物理

行为 (如时钟, 缓存访问, 功耗, 声音) 泄露信息

诱导处理器加载受害者数据至缓存, 再利用道提取

访问的内存。Spectre: 利用推测执行漏洞, 允许程序读取不应

该读取的内存。Meltdown: 利用乱序执行漏洞, 允许程序读取不应

该读取的内存。Speculative Execution 攻击, 分支预测, 道提取

中: 宏元类型; const 与 #define 速度大致一样

Map: 将输入 Data 为 key-value pairs, 并行处理中间

Reduce: 中间键值对聚合到一个 key

Map Reduce 中间有 shuffle: 将 map 侧的中间 key 按键分

组, 确保同一 key pair 发送给同一个 reduce 任务

Hamming ECC: 相信我的大脑已经全满程 (左→右)

RAID: 独立磁盘冗余阵列。RAID 0: 无冗余

RAID 1: 镜像 RAID 3: parity Disk 3: strip RAID 4: parity + block level RAID 5: 4+上 + interleaved

IAFR = $\frac{N \text{ disks} \times 8760 \text{ hrs/year}}{\text{MTTF} \times \text{N disks}}$

MTB between F = MTTF * IAFR + MTTR

Availability = MTTF / (MTTF + MTTR).

Security: Heartbleed: OpenSSL 漏洞, 可读内存缓

冲区, 泄露密钥; Rookhammer: 寻找 DRAM 漏洞, 通过物理

干扰引起位翻转; Side-channel: 利用系统中物理

行为 (如时钟, 缓存访问, 功耗, 声音) 泄露信息

诱导处理器加载受害者数据至缓存, 再利用道提取

访问的内存。Spectre: 利用推测执行漏洞, 允许程序读取不应

该读取的内存。Meltdown: 利用乱序执行漏洞, 允许程序读取不应

该读取的内存。Speculative Execution 攻击, 分支预测, 道提取

中: 宏元类型; const 与 #define 速度大致一样

Map: 将输入 Data 为 key-value pairs, 并行处理中间

Reduce: 中间键值对聚合到一个 key

Map Reduce 中间有 shuffle: 将 map 侧的中间 key 按键分

组, 确保同一 key pair 发送给同一个 reduce 任务

Hamming ECC: 相信我的大脑已经全满程 (左→右)

RAID: 独立磁盘冗余阵列。RAID 0: 无冗余

RAID 1: 镜像 RAID 3: parity Disk 3: strip RAID 4: parity + block level RAID 5: 4+上 + interleaved

IAFR = $\frac{N \text{ disks} \times 8760 \text{ hrs/year}}{\text{MTTF} \times \text{N disks}}$

MTB between F = MTTF * IAFR + MTTR

<p