北京大学计算机学院本科生课程

计算机组成与系统结构实习

Review 1

北京大学微处理器研发中心

易江芳

2023-09-25

What is review

- 实习课的一部分，每次大约20-30分钟
- 之前是由学生回顾理论课内容
- 更新模式，由教师或者助教来引导学生回顾理论课内容，或者进行了开放式讨论
- 可以是知识拓展、专题讨论、经验交流
- 将成为学期成绩的一部分

处理器性能铁律

"Iron Law" of Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

- Instructions per program depends on source code, compiler technology, and ISA
- Cycles per instructions (CPI) depends on ISA and µarchitecture
- Time per cycle depends upon the µarchitecture and base technology

– Krste Asanovic (MIT/UCB cs152)

处理器性能铁律

$$CPU时间 = \frac{秒数}{程序} = \frac{指令数}{程序} \times \frac{时钟数}{指令} \times \frac{秒数}{周期}$$

	指令总数	CPI	时钟周期
算法	X	(X)	
编程语言	X	X	
编译	X	X	
指令系统	X	X	(X)
组成		X	X
实现			X

程序中的指令数目

- 编译工具的影响

	X86-64	ARMv8	Alpha	RISC-V 64G
GCC	7.1	7.1	4.3.2	7.2.0
G++	7.1	7.1	4.3.2	7.2.0
GLibC	2.23	2.15	2.6	2.26
gFortran	7.1	7.1	4.3.2	7.2.0
option	-O3 -Static	-O3 -Static --with-arch=ARMv8-a	-O3 -Static	-O3 -Static -march=RISC-V 64g

- 工具链和标准库
- 版本
- 编译选项

编译选项

- **-O0**： 不做任何优化，这是默认的编译选项

- -O和-O1： 对程序做部分编译优化，编译器会尝试减小生成代码的尺寸以及缩短执行时间，但并不执行需要占用大量编译时间的优化
  - 举例：-floop-optimize，-fif-conversion，-fdelayed-branch

编译进行的Code motion（-O1）

```
void set_row(double *a, double *b,
   long i, long n)
{
    long j;
    for (j = 0; j < n; j++)
        a[n*i+j] = b[j];
}
```

```
    long j;
    long ni = n*i;
    double *rowp = a+ni;
    for (j = 0; j < n; j++)
        *rowp++ = b[j];
```

```
set_row:
        testq   %rcx, %rcx              # Test n
        jle     .L1                     # If 0, goto done
        imulq   %rcx, %rdx              # ni = n*i
        leaq    (%rdi,%rdx,8), %rdx     # rowp = A + ni*8
        movl    $0, %eax                # j = 0
.L3:                                    # loop:
        movsd   (%rsi,%rax,8), %xmm0    # t = b[j]
        movsd   %xmm0, (%rdx,%rax,8)    # M[A+ni*8 + j*8] = t
        addq    $1, %rax                # j++
        cmpq    %rcx, %rax              # j:n
        jne     .L3                     # if !=, goto loop
.L1:                                    # done:
        rep ; ret
```

编译选项

- -O2： 是比O1更高级的选项，gcc将执行几乎所有的不包含时间和空间折中的优化，但编译器不进行循环展开以及函数内联。与O1比较而言，O2优化增加了编译时间的基础上，提高了生成代码的执行效率。
  - 举例：-fforce-mem，fschedule-insns
  - 对齐系列：-falign-functions，-falign-jumps，-falign-loops，-falign-labels

- -O3：该选项除了执行-O2所有的优化选项之外，一般都是采取很多向量化算法，提高代码的并行执行程度，利用现代CPU中的流水线，Cache等
  - 举例：-finline-functions，-funswitch-loops

代码优化带来的问题

- 调试问题
- 内存操作顺序改变所带来的问题

处理器性能铁律

$$CPU时间 = \frac{秒数}{程序} = \frac{指令数}{程序} \times \frac{时钟数}{指令} \times \frac{秒数}{周期}$$

	指令总数	CPI	时钟周期
算法	X	(X)	
编程语言	X	X	
编译	X	X	
指令系统	X	X	(X)
组成		X	X
实现			X

指令系统

- ISA：指令系统体系结构，有专门的topic介绍
- 既然ISA是软硬件接口，那么如何定义接口规范就是很重要且关键的
- 某个功能，由软件完成还是由硬件完成，这不是对错问题，是优劣问题
  - 平衡：性能vs功耗vs成本vs兼容vs易用

RISC vs CISC

Table 1. Summary of RISC and CISC Trends.

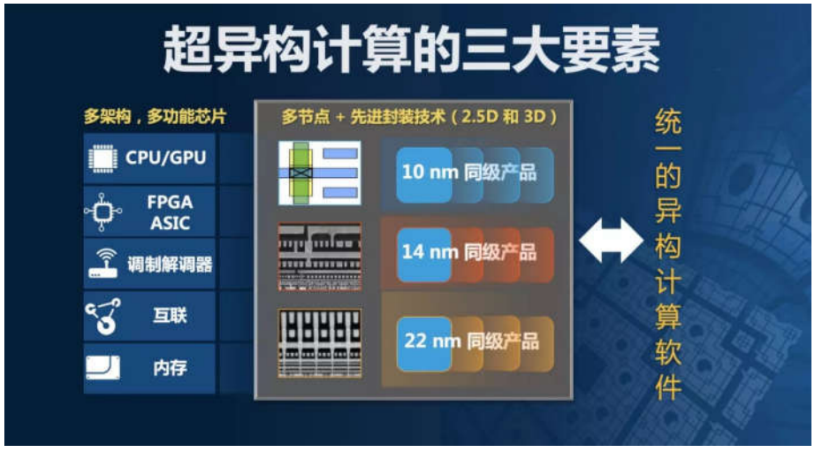
	Format	Operations	Operands
RISC / ARM	○ Fixed length instructions ○ Relatively simple encoding ○ ARM: 4B, THUMB(2B, optional)	○ Simple, single function operations ○ Single cycle	○ Operands: registers, immediates ○ Few addressing modes ○ ARM: 16 general purpose registers
CISC / x86	○ Variable length instructions ○ Common insts shorter/simpler ○ Special insts longer/complex ○ x86: from 1B to 16B long	○ Complex, multi-cycle instructions ○ Transcendentals ○ Encryption ○ String manipulation	○ Operands: memory, registers, immediates ○ Many addressing modes ○ x86: 8 32b & 6 16b registers
Historical Contrasts	○ CISC decode latency prevents pipelining ○ CISC decoders slower/more area ○ Code density: RISC < CISC	○ Even w/ μcode, pipelining hard ○ CISC latency may be longer than compiler's RISC equivalent	○ CISC decoder complexity higher ○ CISC has more per inst work, longer cycles ○ Static code size: RISC > CISC
Convergence Trends	○ **μ-op cache minimizes decoding overheads** ○ **x86 decode optimized for common insts** ○ **I-cache minimizes code density impact**	○ **CISC insts split into RISC-like micro-ops; optimizations eliminated inefficiencies** ○ **Modern compilers pick mostly RISC insts; μ-op counts similar for ARM and x86**	○ **x86 decode optimized for common insts** ○ **CISC insts split into RISC-like micro-ops; x86 and ARM μ-op latencies similar** ○ **Number of data cache accesses similar**
Empirical Questions	○ How much variance in x86 inst length? Low variance ⇒ common insts optimized ○ Are ARM and x86 code densities similar? Similar density ⇒ No ISA effect ○ What are instruction cache miss rates? Low ⇒ caches hide low code densities	○ Are macro-op counts similar? Similar ⇒ RISC-like on both ○ Are complex instructions used by x86 ISA? Few complex ⇒ Compiler picks RISC-like ○ Are μ-op counts similar? Similar ⇒ CISC split into RISC-like μ-ops	○ Number of data accesses similar? Similar ⇒ no data access inefficiencies

ISA之争

- 按照时间排序：MIPS and VAX [7], Pentium-Pro and Alpha 21164 [6], Power5+ and Intel Woodcrest [22], ARM and x86[1,2]。这些之前的ISA研究都主要关注性能，考虑商业化实现。
- 随着时间变化，共识也在变化：
  - MIPS and VAX [7]：MIPS等RISC ISA提供了显著的性能优势
  - Pentium-Pro and Alpha 21164 [6]：CISC还是RISC，对性能会产生根本性差异。CISC可以通过激进的微结构设计来弥补差距。
  - Power5+ and Intel Woodcrest [22]：使用激进的微结构设计，RISC和CISC可以获得相似的性能
  - ARM and x86[1,2]：随着微结构的调整，RISC和CISC的性能已经接近，但CISC的功耗成为棘手问题。
  - ARM and x86: 在这场功耗战争（power struggle）中，从根本上说没有哪个ISA比另一个更节能，RISC还是CISC看上去并不很重要。

异构ISA

- 2018年以来，人工智能、深度学习火爆全球，各类定制加速器层出不穷
- 如今，在一颗芯片中集成各种专用加速部件、可编程逻辑、控制单元
- 为应对应用市场的多变，将各种工艺节点的芯片封装在一起
- 针对未来多元化的计算需求，需要“异构”处理器



来自于Intel中国研究院2019技术报告

异构融合计算

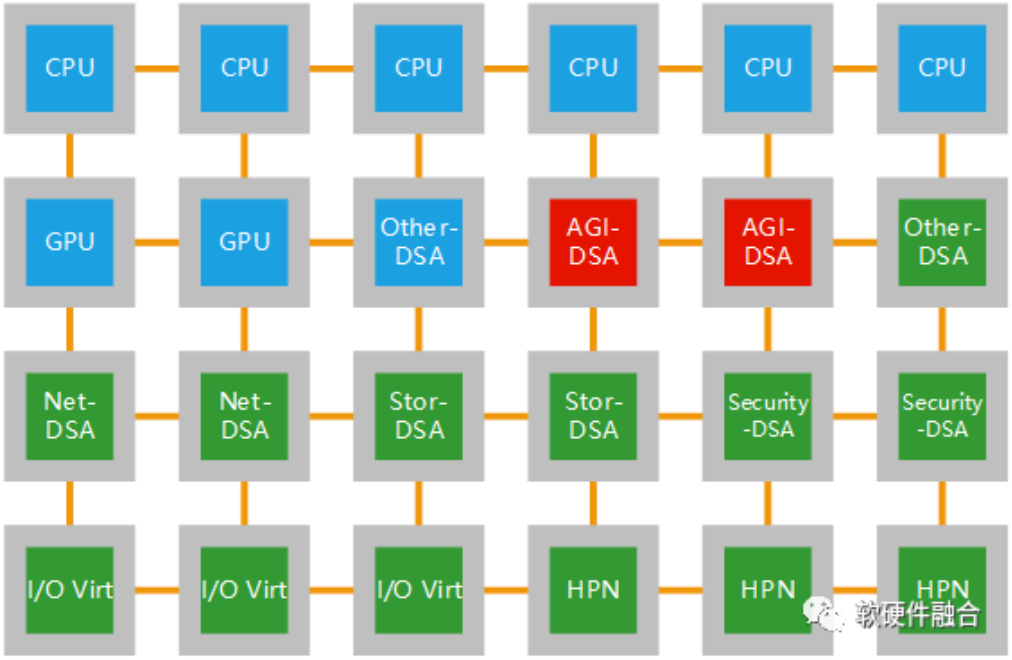
- 2019年，Intel 提出了超异构的概念，只强调了“多”异构，并没有强调异构的“融合”，且无具体产品
- NVIDIA 虽然没有提超异构或异构融合，但有许多具体的产品：GPU+CPU、GPU+DPU、未来GPU+CPU+DPU

- 动力来源：从 2018年开始，随着AI大模型应用的涌现，算力需求平均每2个月翻一倍；摩根士丹利估计2022年谷歌的 3.3万亿次搜索，平均成本约为每个 0.2美分 John Hennessy表示 基于大模型搜索的成本是标准关键词搜索的10倍。

计算架构从各自为政、孤岛式的异构计算走向异构融合计算

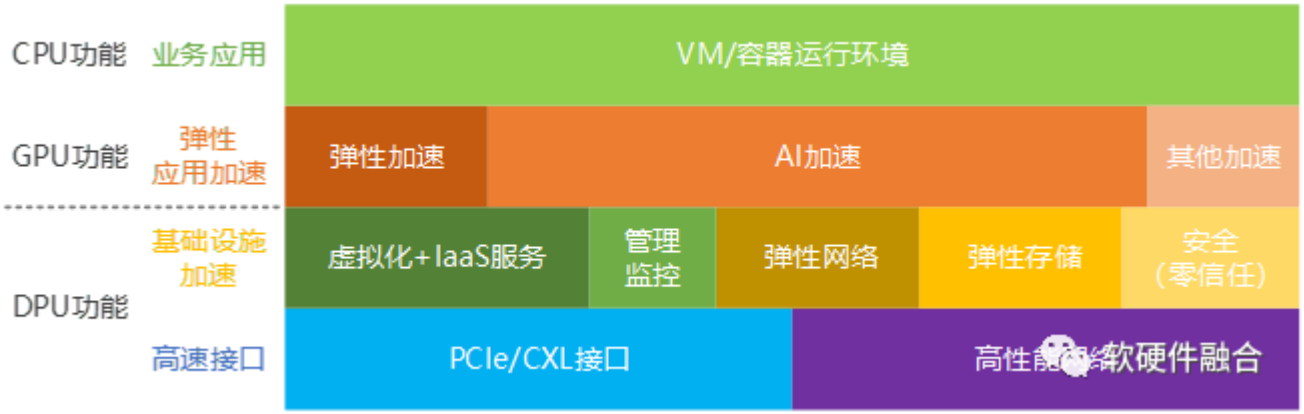
异构融合计算

- 狭义的异构融合计算，指的是多种不同类型、不同架构处理器组成的计算架构
- 广义的异构融合计算，是指通过将 **处理器、芯片、硬件设备、操作系统、编程框架、编程语言、网络通信协议、数据中心** 等不同层次、不同类型的计算技术进行整合优化，以实现多种异构计算资源的高效利用



异构融合计算

- 系统任务主要分为三类：
  - 不经常变化的任务，归属基础设施层，由DPU覆盖；
  - 业务应用加速部分，归属到弹性应用加速层，由GPU等业务加速芯片覆盖；
  - 业务应用不可加速部分，以及其他没有加速支持的任务，归属到业务应用层，由CPU覆盖。从功能视角，可以看作是CPU、GPU和DPU功能的集合。



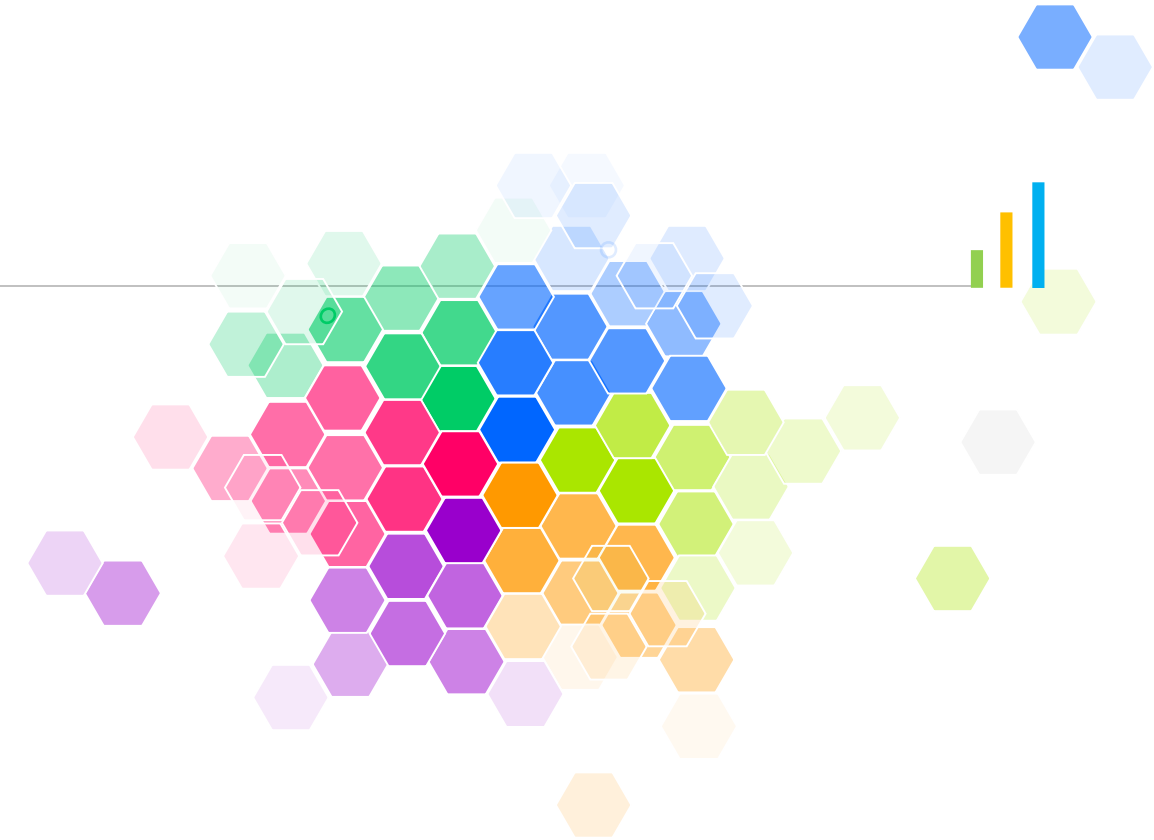
下一代超强处理单元的雏形？

Quiz

- 下列哪些是在 ISA 中描述的？
  - 寄存器堆的大小和数量
  - 栈式结构中的栈
  - 流水线中的寄存器
  - 转移延迟/加载延迟槽
  - NOP
  - 流水线中的气泡
  - 条件码
  - 存储单元的地址宽度
  - 指令/数据 Cache

Quiz

- 如果改用软件实现浮点指令，分析性能影响
  - Instruction/Program
  - CPI
  - Cycle Time



欢迎提问