Datawhale

Datawhale 开源社区

Datawhale open source community

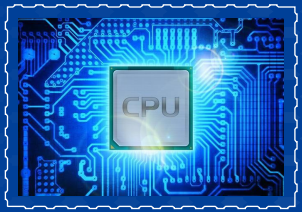
深入理解计算机系统（2）

Computer Systems A Programmer's Perspective CSAPP

李岳昆、易远哲

realyurk@gmail.com、yuanzhe.yi@outlook.com

2021 年 12 月 1 日



第 I 部分

计算机系统漫游-II

操作系统的作用

无论是 shell 程序还是 hello 程序都没有直接访问键盘、显示器、磁盘这些硬件设备，真正操挫硬件的是操作系统，我们可以把操作系统看成是应用程序和硬件之间的中间层，所有的应用程序对硬件的操作必须通过操作系统来完成。

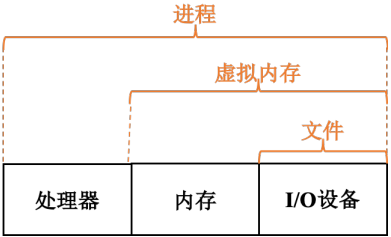
这样设计的目的主要有两个：

1. 防止硬件被失控的应用程序滥用；
2. 操作系统提供统一的机制来控制这些复杂的底层硬件.

操作系统的作用

Datawhale

为了实现上述的功能，操作系统引入了几个抽象的概念。例如：文件是对 IO 设备的抽象；虚拟内存是对内存和磁盘 IO 的抽象；进程是对处理器、内存以及 IO 设备的抽象。

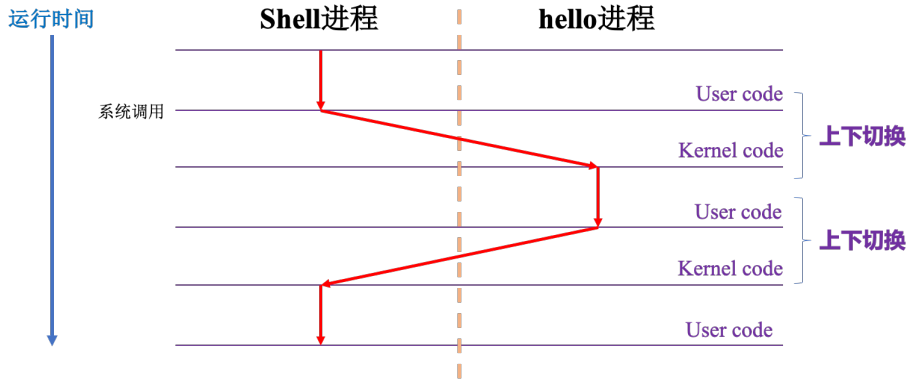


Datawhale

假设示例场景中只有两个并发的进程：shell 进程和 hello 进程

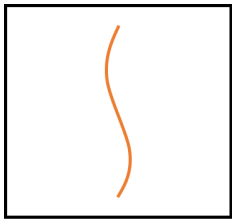
1. 最开始的时候，只有 shell 进程在运行，即 shell 在等待命合行的输入。
2. 当我们通过 shell 进程加载 hello 进程时，shell 进程通过系统调用来执行我们的请求，系统调用会将控制权从 shell 进程传递给操作系统，操作系统保存 shell 进程的上下文，然后创建一个新的 hello 进程及其上下文，然后将控制权转交给新的 hello 进程。
3. hello 进程执行完之后，操作系统就会恢复 shell 进程的上下文，并将控制权交给 shell 进程，之后 shell 进程继续等待下一个命令的输入。
4. 操作系统会跟踪进程运行所需要的所有状态信息，这种状态，称为上下文（Context）。例如当前 PC 和寄存器的值，以及内存中的内容等等。

进程角度

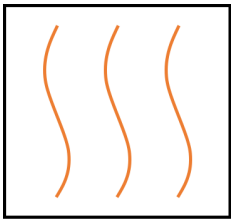

运行时间
Shell进程
hello进程
系统调用
User code
Kernel code
上下切换
User code
Kernel code
上下切换
User code

进程角度

现代操作系统中，一个进程实际上由多个线程组成，每个线程都运行在进程的上下文中，共享代码和数据。由于网络服务器对并行处理的需求，线程成为越来越重要的编程模型。



单进程，单线程



单进程，多线程

我们肯定是希望能够同时进行多种线程，而不希望单线程占据全部操作系统. 举个例子：一个微信程序就是一个进程，而我们在微信里一边文字聊天、一边视频聊天、一边传输文件就是三个线程同时进行.

Datawhale

虚拟内存

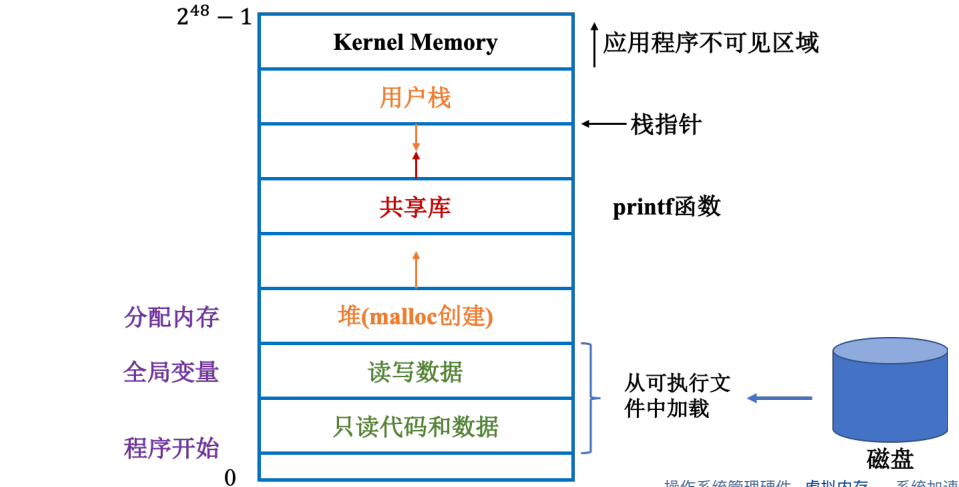
操作系统为每个进程提供了一个假象，就是每个进程都在独自占用整个内存空间，每个进程看到的内存都是一样的，我们称之为**虚拟地址空间**[1].

[1]对于更加详细的操作系统及进程调度相关内容，请参考《操作系统》

虚拟内存

Datawhale

下图为 Linux 的虚拟地址空间，从下往上看，地址是增大的。最下面是 0 地址。



- 第一个区域是用来存放程序的代码和数据的，这个区域的内容是从可执行目标文件中加载而来的，例如我们多次提到的 hello 程序。对所有的进程来讲，代码都是从固定的地址开始。至于这个读写数据区域放的是什么数据呢？例如在 C 语言中，全局变量就是存放在这个区域.
- 顺着地址增大的方向，继续往上看就是堆（heap），学过 C 语言的同学应该用过 malloc 函数，程序中 malloc 所申请的内存空间就在这个堆中。程序的代码和数据区在程序一开始的时候就被指定了大小，但是堆可以在运行时动态的扩展和收缩.

虚拟内存

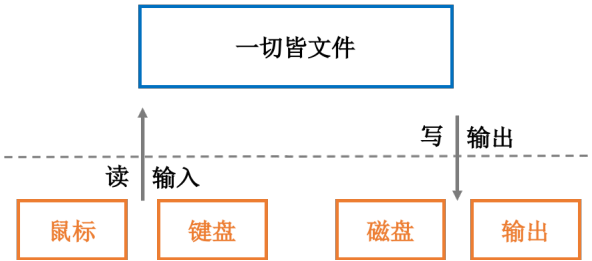
Datawhale

- 接下来，就是共享库的存放区域。这个区域主要存放像 C 语言的标准库和数学库这种共享库的代码和数据，例如 hello 程序中的 printf 函数就是存放在这里.
- 继续往上看，这个区域称为用户栈（user stack），我们在写程序的时候都使用过函数调用，实际上函数调用的本质就是压栈。这句话的意思是：每一次当程序进行函数调用的时候，栈就会增长，函数执行完毕返回时，栈就会收缩。需要注意的是　栈的增长方向是从高地址到低地址.
- 最后，我们看一下地址空间的最顶部的区域，这个区域是为内核保留的区域，应用程序代码不能读写这个区域的数据，也不能直接调用内核中定义的函数，也就是说，这个区域对应用程序是不可见的.

文件

Datawhale

Linux 系统的哲学思想是：一切皆为文件。

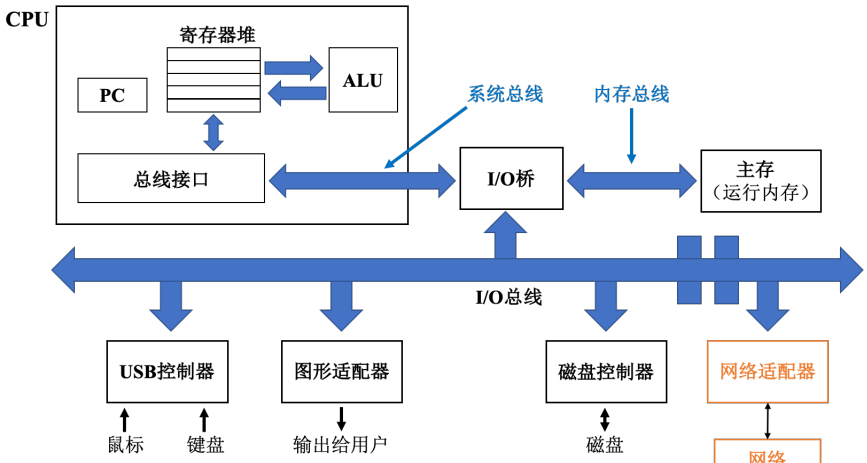


- 所有的 IO 设备，包括键盘，磁盘，显示器，甚至网络，这些都可以看成文件，系统中所有的输入和输出都可以通过读写文件来完成.
- 虽然文件的概念非常简单，但却非常强大。例如：当程序员需要处理读写磁盘上的文件时，他们不需要了解具体的磁盘技术，同一个程序，可以在不同磁盘技术上的不同系统上运行.

系统之间利用网络通信

Datawhale

系统来看，网络也可以视为一个 IO 设备。



系统之间利用网络通信

- 随着互联网的发展，从一台计算机发送消息到另外一台计算机已经成为非常普遍的应用。《深入理解计算机系统》中讲述了如何使用本地计算机上的 telnet 客户端连接远程主机上的 telnet 服务器。
- 由于 telnet 的安全性问题，目前 ssh 的连接方式的更加普遍。当我们在 ssh 的客户端中输入 hello 字符串并且敲下回车之后，客户端的软件就会通过网络将字符串发送到 ssh 服务端，ssh 服务端从网络端接收到这个字符串以后，会将这个字符串传递给远程主机上的 shell 程序，然后 shell 负责 hello 程序的加载，运行结果返回给 ssh 的服务端，最后 ssh 的服务端通过网络将程序的运行结果发送给 ssh 的客户端，ssh 客户端在屏幕上显示运行结果。

系统之间利用网络通信


1.用户在键盘敲出“hello”
3.服务器发送“hello”给shell
2.客户端发送“hello”给ssh服务器
本地ssh客户端
远程ssh服务器
4.ssh发送“hello,world/n”给客户端
5.客户端输出“hello,world/n”

一些约定/假定

Datawhale

- 任务 (task)：并行计算所处理的对象.
- 工作量 (workload)：处理某任务的所需的各种开销的总和.
- 处理器 (processor)：并行计算中所使用的最基本的处理器单元.
- 执行率 (execution rat)：每个处理器单位时间能完成的工作量[2].
- 执行时间 (execution time)：处理某任务所需的时间.
- 加速比 (scalability)：当处理器个数增多时，完成某固定工作量任务所需执行时间的减少倍数.
- 理想加速比 (ideal scalability)：处理器个数增多的比例.
- 并行效率 (parallel efficiency): 加速比 $\div$ 理想加速比 $\times 100\%$.

[2]这里假定每个处理器的执行率相同。

阿姆达尔定律

Datawhale

阿姆达尔定律 (Amdahl's Law, 1967)

记 $\alpha \in [0,1]$ 是某任务无法并行处理部分所占的比例. 假设该任务的工作量固定，则对任意 n 个处理器，相比于 1 个处理器，能够取得的加速比满足:$S(n) < \frac{1}{\alpha}$.

证明：设每个处理器的执行率是 R，处理该任务的总工作量是 W. 记 $T(1)$ 和 $T(n)$ 分别为使用 1 个和 n 个处理器处理该任务所需要的时间，则有

$$T(1) = \frac{W}{R}, \quad T(n) = \frac{\alpha W}{R} + \frac{(1-\alpha)W}{nR}.$$

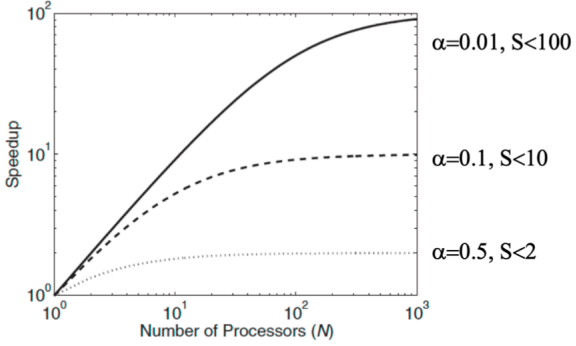
据此，可计算出使用 n 个处理器相比于 1 个处理器的加速比

$$S(n) = \frac{T(1)}{T(n)} = \frac{1}{\alpha + \frac{1-\alpha}{n}} < \frac{1}{\alpha}.$$

阿姆达尔定律的推论

Datawhale

Gene M. Amdahl
(1922-2015)



484 Spring Joint Computer Conf., 1967

by DR. GENE M. AMDAHL
International Business Machines Corporation
Sunnyvale, California

Validity of the single processor approach to achieving large scale computing capabilities

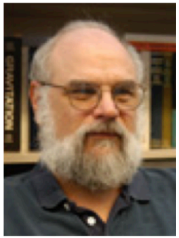
卡普挑战

NCUBE 10 (1,024 processors)

Background:

In 1983, there was a new tech trend of **massively parallel computing** with over 1,000 processors.

People started to wonder how to use them efficiently.



Alan H. Karp

卡普挑战 (Karp Challenge, 1985)

打赌 100 美元没人能在十年内为三个实际应用实现 200 倍的并行加速.

- 结果: 两年之内，没有胜者 (大多是模型问题，加速比最多数十).

戈登贝尔奖

Datawhale

Gordon Bell
(1934-)

In 1987, Alan Karp was approached by Gordon Bell, who was a founding assistant director of the **CISE Directorate at NSF**.

Bell persuaded Karp to replace the **Karp Challenge** with the **Gordon Bell Prize**.

The Gordon Bell Prize **does not require a speedup of 200X** or any particular number but emphasizes on **technical innovations in HPC applications**.

戈登贝尔奖 (Gordon Bell Prize, 1987)

每年奖励 1000 美元给在高性能计算应用取得杰出成就的团队.

- 2006 年起由 ACM 负责，2011 年起奖金提升至 1 万美元.

首届戈登贝尔奖

Datawhale

John L. Gustafson
(1955-)

In 1987 the first Gordon Bell Prize was awarded to **Robert Benner, John Gustafson and Gary Montry** from Sandia National Laboratories.

Achieved **400~600** speedup on NCUBE 10 in 3 applications:
* **Beam Stress Analysis**
* **Surface Wave Simulation**
* **Unstable Fluid Flow Modeling**

In fact, they also won the Karp Challenge!

REEVALUATING AMDAHL'S LAW JOHN L. GUSTAFSON

532 *Communications of the ACM* *May 1988 Volume 31 Number 5*

古斯塔法森定律

Datawhale

古斯塔法森定律 (Gustafson's Law, 1988)

记 $\alpha \in [0,1]$ 是某任务无法并行处理部分所占的比例. 假设该任务的工作量可以随着处理器个数缩放，从而保持处理时间固定. 则对任意 n 个处理器，相比于 1 个处理器，能够取得的加速比 $S(n)$ 不存在上界.

证明：设每个处理器的执行率是 R，单处理器情况下该任务的基准工作量是 W. 在处理时间固定的情况下，可以得知采用 n 个处理器时该任务的缩放工作量 W 应为

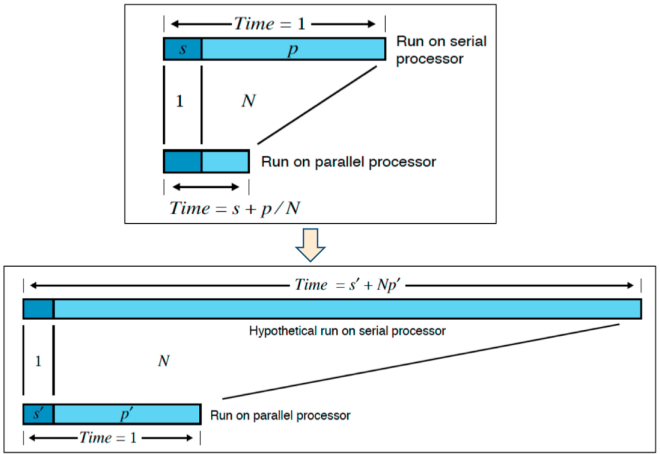
$$W' = \alpha W + (1-\alpha)nW.$$

据此，可计算出使用 n 个处理器相比于 1 个处理器的加速比

$$S'(n) = \frac{\frac{W'}{R}}{\frac{W}{R}} = \alpha + (1-\alpha)n.$$

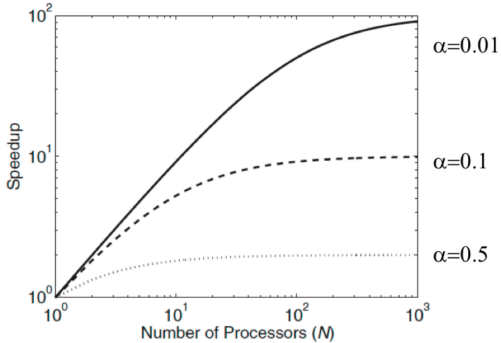
从阿姆达尔加速比到古斯塔法森加速比

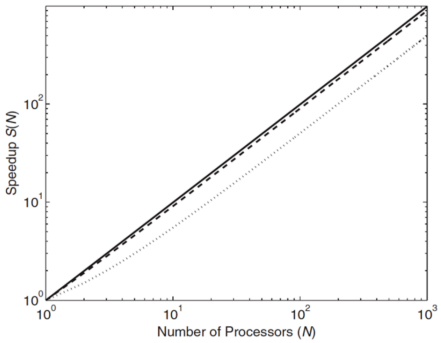
Datawhale



从阿姆达尔加速比到古斯塔法森加速比

Datawhale





孙-倪定律

Datawhale

孙-倪定律 (Sun-Ni's Law, 1990)

记 α 是某任务无法并行处理部分所占的比例. 假设该任务的可并行部分随着处理器个数 n 按照因子 $G(n)$ 缩放，则对任意 n，相比于 1 个处理器，能够取得的加速比 $S^*(n)$ 满足

$$S^*(n) = \frac{\alpha + (1-\alpha)G(n)}{\alpha + (1-\alpha)\frac{G(n)}{n}}$$

证明：设每个处理器的执行率是 R，单处理器情况下该任务的基准工作量是 W. 使用 n 个处理器时该任务的缩放工作量为

$$W^* = \alpha W + (1-\alpha)G(n)W.$$

据此，可计算出使用 n 个处理器相比于 1 个处理器的加速比

$$S^*(n) = \frac{\alpha W_{\frac{1}{R}} + (1-\alpha)G(n)W_{\frac{1}{R}}}{\alpha W_{\frac{1}{R}} + (1-\alpha)G(n)W_{\frac{1}{nR}}} = \frac{\alpha + (1-\alpha)G(n)}{\alpha + (1-\alpha)\frac{G(n)}{n}}.$$

Datawhale

加速比分析

$$S^*(n) = \frac{\alpha + (1-\alpha)G(n)}{\alpha + (1-\alpha)\frac{G(n)}{n}} \begin{cases} = \frac{1}{\alpha + \frac{1-\alpha}{n}}, & \text{if } G(n) = 1, \\ = \alpha + (1-\alpha)n, & \text{if } G(n) = n, \\ > \alpha + (1-\alpha)n, & \text{if } G(n) > n. \end{cases}$$

	加速比 $(n \to \infty)$	并行效率 $(n \to \infty)$
阿姆达尔	$S(n) \to \frac{1}{\alpha}$	$E(n) \to 0$
古斯塔法森	$S'(n) \to \infty$	$E'(n) \to 1-\alpha$
孙-倪 $(G(n) > O(n))$	$S^*(n) \to \infty$	$E^*(n) \to 1$

应用举例

Datawhale

例: 矩阵乘 C = AB，这里 A, B, C 都是 N × N 阶矩阵.

- 计算复杂度:$C(N) = 2N^3$.
- 存储复杂度:$S(N) = 3N^2$.

由 $S(N) = x$ 可得 $S^{-1}(x) = (\frac{x}{3})^{\frac{1}{2}}$，因此

$$G(n) = \frac{C(S^{-1}(nx))}{C(S^{-1}(x))} = \frac{2(\frac{nx}{3})^{\frac{1}{2}\times 3}}{2(\frac{x}{3})^{\frac{1}{2}\times 3}} = n^{\frac{3}{2}},$$

从而

$$S^*(n) = \frac{\alpha + (1-\alpha)G(n)}{\alpha + (1-\alpha)\frac{G(n)}{n}} = \frac{\alpha + (1-\alpha)n^{\frac{3}{2}}}{\alpha + (1-\alpha)n^{\frac{1}{2}}}.$$

并发和并行

Datawhale

如何获得更高的计算能力呢？可以通过以下三种途径：

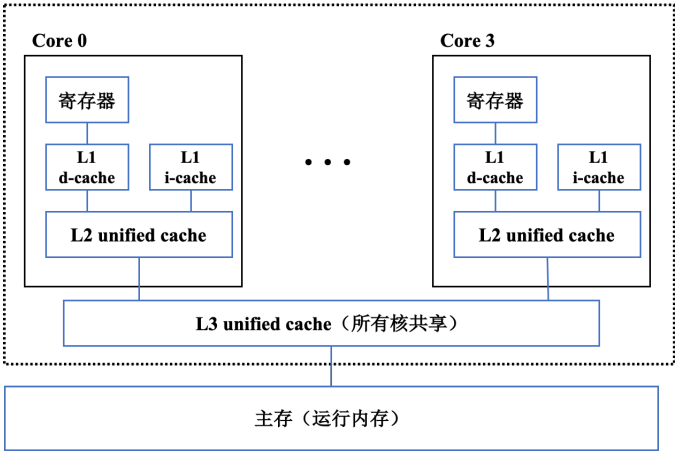
1. 线程级并发;
2. 指令级并行;
3. 单指令多数据并行.

线程级并发

Datawhale

- 首先我们看一个多核处理器的组织结构，下图的处理器芯片具有四个 CPU 核心，由于篇幅限制，另外两个用省略号代替了。每个 CPU 核心都有自己的 LI cache 和 L2 cache ，四个 CPU 核心共享 L3 cache，这 4 个 CPU 核心集成在一颗芯片上.
- 对于许多高性能的服务器芯片，单颗芯片集成的 CPU 数量高达几十个，甚至上百个。通过增加 CPU 的核心数，可以提高系统的性能.

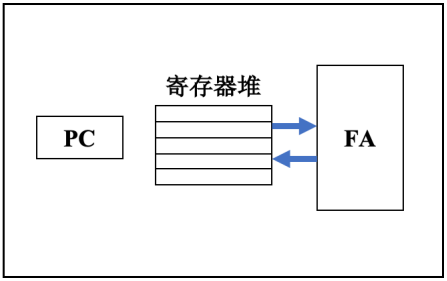
线程级并发

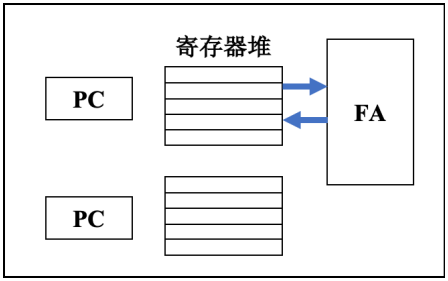

Core 0
寄存器
L1
d-cache
L1
i-cache
L2 unified cache
Core 3
寄存器
L1
d-cache
L1
i-cache
L2 unified cache
L3 unified cache（所有核共享）
主存（运行内存）

线程级并发

Datawhale

除此之外，还有一个技术就是**超线程（hyperthreading）**，也称同时多线程。如果每个 CPU 核心可以执行两个线程，那么四个核心就可以并行的执行 8 个线程，那么，单个 CPU 核心是如何实现超线程的呢?





线程级并发

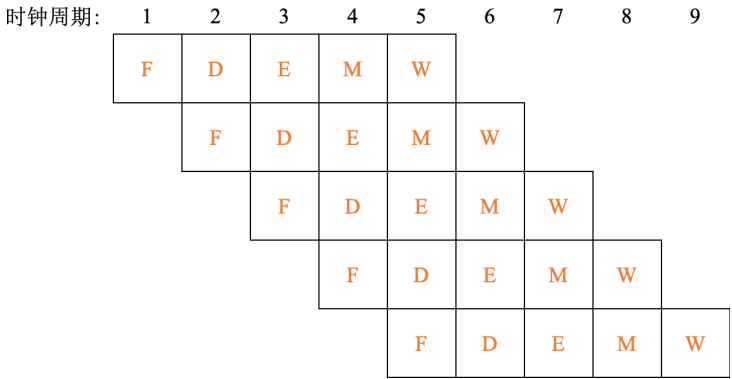
Datawhale

在 CPU 内部，像程序计数器和寄存器文件这样的硬件部件有多个备份，而像浮点运算部件这个样的硬件还是只有一份，常规单线程处理器在做线程切换时，大概需要 20000 个时钟周期，而超线程处理器可以在单周期的基础上决定执行哪一个线程，这样一来，CPU 可以更好地利用它的处理资源。当一个线程因为读取数据而进入等待状态时，CPU 可以去执行另外一个线程，其中线程之间的切换只需要极少的时间代价.

指令级并行

Datawhale

现代处理器可以同时执行多条指令的属性称为指令级并行，每条指令从开始到结束大概需要 20 个时钟周期或者更多，但是处理器采用了非常多的技巧可以同时处理多达 100 条指命，因此，近几年的处理器可以保持每个周期 2 4 条指令的执行速率。



单指令多数据并行

现代处理器拥有特殊的硬件部件，允许一条指令产生多个并行的操作，这种方式称为单指令多数据（Single Instruction Multiple Data）。SIMD 的指令多是为了提高处理视频、以及声音这类数据的执行速度，比较新的 Intel 以及 AMD 的处理器都是支持 SIMD 指令加速。