阿姆达尔定律

原文地址 作者：Jakob Jenkov 译者：张坤

阿姆达尔定律可以用来计算处理器平行运算之后效率提升的能力。阿姆达尔定律因Gene Amdal 在1967年提出这个定律而得名。绝大多数使用并行或并发系统的开发者有一种并发或并行可能会带来提速的感觉，甚至不知道阿姆达尔定律。不管怎样，了解阿姆达尔定律还是有用的。

我会首先以算术的方式介绍阿姆达尔定律定律，然后再用图表演示一下。

阿姆达尔定律定义

一个程序（或者一个算法）可以按照是否可以被并行化分为下面两个部分：

可以被并行化的部分
不可以被并行化的部分

假设一个程序处理磁盘上的文件。这个程序的一小部分用来扫描路径和在内存中创建文件目录。做完这些后，每个文件交个一个单独的线程去处理。扫描路径和创建文件目录的部分不可以被并行化，不过处理文件的过程可以。

程序串行（非并行）执行的总时间我们记为T。时间T包括不可以被并行和可以被并行部分的时间。不可以被并行的部分我们记为B。那么可以被并行的部分就是T-B。下面的列表总结了这些定义：

T = 串行执行的总时间
B = 不可以并行的总时间
T- B = 并行部分的总时间

从上面可以得出：

T = B + (T – B)

首先，这个看起来可能有一点奇怪，程序的可并行部分在上面这个公式中并没有自己的标识。然而，由于这个公式中可并行可以用总时间T 和 B（不可并行部分）表示出来，这个公式实际上已经从概念上得到了简化，也即是指以这种方式减少了变量的个数。

T- B 是可并行化的部分，以并行的方式执行可以提高程序的运行速度。可以提速多少取决于有多少线程或者多少个CPU来执行。线程或者CPU的个数我们记为N。可并行化部分被执行的最快时间可以通过下面的公式计算出来：

(T – B) / N

或者通过这种方式

(1 / N) * (T – B)

维基中使用的是第二种方式。

根据阿姆达尔定律，当一个程序的可并行部分使用N个线程或CPU执行时，执行的总时间为：

T(N) = B + (T – B) / N

T(N)指的是在并行因子为N时的总执行时间。因此，T(1)就执行在并行因子为1时程序的总执行时间。使用T(1)代替T，阿姆达尔定律定律看起来像这样：

T(N) = B + (T(1) – B) / N

表达的意思都是是一样的。

一个计算例子

为了更好的理解阿姆达尔定律，让我们来看一个计算的例子。执行一个程序的总时间设为1.程序的不可并行化占40%，按总时间1计算，就是0.4.可并行部分就是1 – 0.4 = 0.6.

在并行因子为2的情况下，程序的执行时间将会是：

```
T(2) = 0.4 + ( 1 - 0.4 ) / 2
 = 0.4 + 0.6 / 2
 = 0.4 + 0.3
 = 0.7
```

在并行因子为5的情况下，程序的执行时间将会是：

```
T(5) = 0.4 + ( 1 - 0.4 ) / 5
 = 0.4 + 0.6 / 6
```

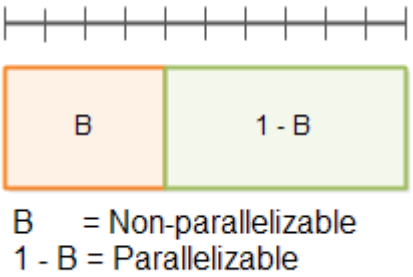
$$= 0.4 + 0.12$$

$$= 0.52$$

阿姆达尔定律图示

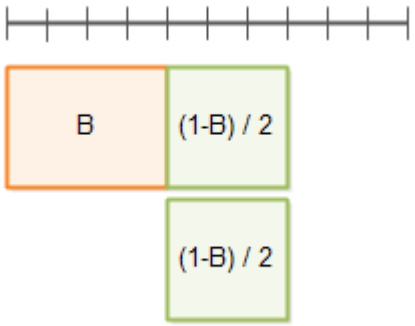
为了更好地理解阿姆达尔定律，我会尝试演示这个定定律是如何诞生的。

首先，一个程序可以被分割为两部分，一部分为不可并行部分B，一部分为可并行部分1 – B。如下图：

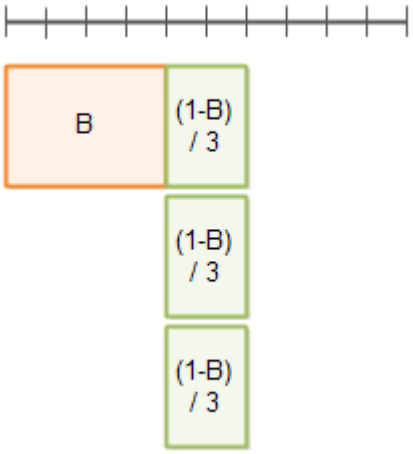


在顶部被带有分割线的那条直线代表总时间 T(1)。

下面你可以看到在并行因子为2的情况下的执行时间：



并行因子为3的情况：



优化算法

从阿姆达尔定律可以看出，程序的可并行化部分可以通过使用更多的硬件（更多的线程或CPU）运行更快。对于不可并行化的部分，只能通过优化代码来达到提速的目的。因此，你可以通过优化不可并行化部分来提高你的程序的运行速度和并行能力。你可以对不可并行化在算法上做一点改动，如果有可能，你也可以把一些移到可并行化放的部分。

优化串行分量

如果你优化一个程序的串行化部分，你也可以使用阿姆达尔定律来计算程序优化后的执行时间。如果不可并行部分通过一个因子O来优化，那么阿姆达尔定律看起来就像这样：

```
T(O, N) = B / O + (1 - B / O) / N
```

记住，现在程序的不可并行化部分占了`B / O`的时间，所以，可并行化部分就占了`1 - B / O`的时间.

如果B为0.1，O为2，N为5，计算看起来就像这样：

```
T(2,5) = 0.4 / 2 + (1 - 0.4 / 2) / 5
   = 0.2 + (1 - 0.4 / 2) / 5
   = 0.2 + (1 - 0.2) / 5
   = 0.2 + 0.8 / 5
   = 0.2 + 0.16
   = 0.36
```

运行时间 vs. 加速

到目前为止，我们只用阿姆达尔定律计算了一个程序或算法在优化后或者并行化后的执行时间。我们也可以使用阿姆达尔定律计算加速比（speedup），也就是经过优化后或者串行化后的程序或算法比原来快了多少。

如果旧版本的程序或算法的执行时间为T，那么增速比就是：

```
Speedup = T / T(O , N);
```

为了计算执行时间，我们常常把T设为1，加速比为原来时间的一个分数。公式大致像下面这样：

```
Speedup = 1 / T (O,N)
```

如果我们使用阿姆达尔定律来代替T(O,N)，我们可以得到下面的公式：

```
Speedup = 1 / ( B / O + (1 - B / O) / N)
```

如果B = 0.4， O = 2， N = 5， 计算变成下面这样：

```
Speedup = 1 / ( 0.4 / 2 + (1 - 0.4 / 2) / 5)
    = 1 / ( 0.2 + (1 - 0.4 / 2) / 5)
    = 1 / ( 0.2 + (1 - 0.2) / 5 )
    = 1 / ( 0.2 + 0.8 / 5 )
    = 1 / ( 0.2 + 0.16 )
    = 1 / 0.36
    = 2.77777 ...
```

上面的计算结果可以看出，如果你通过一个因子2来优化不可并行化部分，一个因子5来并行化可并行化部分，这个程序或算法的最新优化版本最多可以比原来的版本快2.77777倍。

测量，不要仅是计算

虽然阿姆达尔定律允许你并行化一个算法的理论加速比，但是不要过度依赖这样的计算。在实际场景中，当你优化或并行化一个算法时，可以有很多的因子可以被考虑进来。

内存的速度，CPU缓存，磁盘，网卡等可能都是一个限制因子。如果一个算法的最新版本是并行化的，但是导致了大量的CPU缓存浪费，你可能不会再使用x N个CPU来获得x N的期望加速。如果你的内存总线

（memory bus），磁盘，网卡或者网络连接都处于高负载状态，也是一样的情况。

我们的建议是，使用阿姆达尔定律定律来指导我们优化程序，而不是用来测量优化带来的实际加速比。记住，有时候一个高度串行化的算法胜过一个并行化的算法，因为串行化版本不需要进行协调管理（上下文切换），而且一个单个的CPU在底层硬件工作（CPU管道、CPU缓存等）上的一致性可能更好。

Cooler007
2015/09/01 10:56上午

阿姆达尔定律可以用来计算处理器 平行 运算之后效率提升的能力, 应该是并行。 虽然很小 但真心希望 大家 一起把它做好。

zhili
2018/06/13 4:51下午

总算看完了, 原文和译文, 原文已经改了好多了 ....

Hammer
2021/03/30 5:06下午

刚看完,不得不说讲的太好了,从模糊到现在略懂,很开心,很充实