知乎日报 - 知乎

https://daily.zhihu.com/story/9728943

doodlewind, Thu Oct, 15 13:54

真正最难修复的 bug，其解决靠的已经不是个人英雄主义的单打独斗，而是全世界顶尖高手集体智慧的「饱和式抢救」了。

这种 bug 的解决，甚至能直接使其解决者自此一战而扬名天下。

1994 年著名的 Intel CPU 浮点运算 bug，就是这样的传奇 bug。

缘起

当时，Intel 为奔腾 CPU 的浮点除法指令 `FDIV` 加入了一种新型的实现。这是 Sweeney-Robertson-Tocher（SRT）算法的一种高性能变体，依赖了一个共有 2048 项的硬件查找表。因为这种算法只会访问整个 128 x16 尺寸查找表中的一个梯形子集，所以这 2048 项中只有略多于一半的项会被用到。由于一些意外，这 1066 项中有 5 项的值被错误地设置为 0（而不是正确的 2），因此可能导致运算结果的错误。但是，这些错误的索引只会在极少数情况下被访问到，以至于这个问题没有被 Intel 研发流程中的随机测试所发现。更可怕的是，**在除法算法的前 8 个执行步骤中，错误的这几项还永远不会被访问到**，因此错误结果与真实结果之间仅有轻微的差异——这种差异对于高精度计算来说可能非常关键，但普通场景下几乎不可能发现（据称概率是每 90 亿次运算出现一次，相当于七百年一遇）。

> 这是人们事后从上帝视角给出的复盘。假如你根本不知道硬件电路中埋着这样的一个雷，你觉得写应用层业务遇到问题时该从何下手呢?

察觉

这个 bug 虽然非常隐蔽，但却没有躲过美国林奇堡学院 Thomas Nicely 教授善于察觉要素的眼睛。他在多台计算机上运行同样的算法来对孪生质数的商进行求和时，发现计算结果在不同机器之间存在差异。

Nicely 花了几个月的时间（注意时间单位是月）来检查可能的差异原因，最终认为问题来自于使用了奔腾 CPU 的系统。发现问题之后，他在 1994 年的 10 月 24 日（程序员节）向 Intel 提出了反馈，并于 10 月 30 日向其他的一些联系人发送了报告问题的电子邮件，其中有一名收件人将其内容转发到了 CompuServe 网络上。电子工程时报的记者 Alex Wolfe 发现了这个帖子，并将其转发给了挪威工程师 Terje Mathisen。在收到消息后的几个小时内，Mathisen 就成功复现了 Nicely 教授的例子。他用汇编语言写了一个简单的测试用例，于 11 月 3 日在新闻组

`comp.sys.intel` 内发布了一系列关于 `FDIV` 指令错误的帖子。一天后，德国的 Andreas Kaiser 找到了 20 多个特殊的数字，这些数字的倒数在奔腾 CPU 上的计算精度只达到了单精度（也就是 32 位 float 的水平，精确到小数点后 7 位）。

定位

与此同时，加州 Vitesse 半导体公司的 FPU（浮点单元）设计师 Tim Coe 从 Kaiser 给出的列表中找到了线索，分析推断出了 Intel 的 FPU 设计师们是如何设计除法电路的。他正确地推测，奔腾 CPU 的除法指令采用了基数为 4 的 SRT 算法，每个时钟周期会产生两个 bit 的商。这样可以让奔腾 CPU 的除法速度达到过去相同时钟速率下 Intel 芯片的两倍。

Coe 创建了一个模型，以此解释了 Kaiser 所报告的误差。他还发现，如果对于分子不为 1 的除法运算，这还可能带来更大的相对误差。基于这个模型，他找到了一对七位整数，它们的商 4195835/3145727 可能是最坏情况下的错误实例。Coe 于 11 月 14 日将这个例子发布到了 `comp.sys.intel` 新闻组上。

在 Coe 发帖前几天，美国麻省一家公司的老板 Cleve Moler 从另一个渠道得知了 `FDIV` 的错误。Moler 起初只是对此感到好奇，并未实际参与。但 Coe 发布的问题定位，使 Moler 的兴趣大大提升。11 月 15 日，Moler 在新闻组上发帖，总结了截至当时 Nicely 和 Coe 各自案例中的已知情况，并找到了这两种情况下的 bug 规律，**即除数都略少于 3 乘以 2 的某个整数次幂**——这个 Moler 可不是土老板，他当过斯坦福的数学教授。

> 举个例子，2^20 = 1048576，而上面的除数 3145727 除以 3，则是 1048575.666666……像不像是给数学家玩的密室逃脱游戏？

舆情

11 月 7 日，参与反馈 bug 的 Wolfe 在电子工程时报上报道了此事，关于奔腾 CPU 这个 bug 的消息，迅速在互联网上流传了开来。

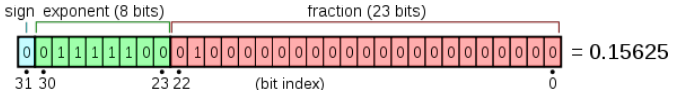
11 月 22 日，美国 NASA 喷气推进实验室（JPL）的两位工程师向采购部门提出建议，认为实验室应该停止订购使用奔腾芯片的计算机。CNN 的记者听说了 JPL 的决定，于是找到 Moler 进行了采访。当天晚上，CNN 在电视节目上报道了奔腾 CPU 的这个 bug，将事态升级为了国民级新闻。到了两天之后的感恩节，包括纽约时报和波士顿环球报在内的主流媒体都对此进行了报道。在接下来的几周内，更是出现了数百篇关于此事的文章。

> 注意这时整个社会都在 BB，但这个 bug 仍然没有解决。

拆弹

基于自己找到的规律，Moler 开始与 Coe、Mathisen、Peter Tang（来自美国阿贡国家实验室），以及 Intel 的几位软硬件工程师合作，尝试解决这个 `FDIV` 错误。只要解决了这个 bug，还能一并解决掉奔腾 CPU 上由此产生的片上正切、正交和求余指令的衍生 bug。到 12 月 5 日，他们开发出了一种巧妙的修复方法：检查除数有效位部分的的高四位（浮点数有效位部分即 fraction，如下图示例中的红色部分），如果它们是 `0001`、`0100`、`0111`、`1010` 或 `1101`，那么就在执行除法运算之前，**将除数和被除数都同乘以 15/16**。在这五种情况下，这种乘以 15/16 的效果都能使除数从「危险」状态转入「安全」状态。这时可以保证缩放后操作数的商，始终能与原始操作数的正确商相同。几天后他们进一步优化了算法，只有当除数有效位的八个高位是 `00011111`、`01001111`、`01111111`、`10101111` 或 `11011111`

时，才将操作数按 15/16 缩放，从而大大减少了额外的运算。这项优化技术被公布到了新闻组，可供全社会无偿自由使用。



32 位单精度浮点数结构，后 23 位为有效位

于是，报道「该公司修复了 Intel 奔腾 CPU 浮点数 bug」的新闻，迅速登上了包括纽约时报在内的各大主流媒体。

这家当时还名不见经传的小公司，由此正式出现在了公众视野之中。

总结

这个 FDIV bug 事件，实在有众多传奇之处：

- 极其隐蔽的 bug 来源
- 极长的定位时间
- 世界各地高手（数学家与软硬件工程师）跨领域的接力式努力
- 堪称奇技淫巧的黑魔法 fix
- 轰动性的媒体传播效应

这简直是个教科书级的程序员题材电影剧本啊……

只剩下最后一个问题了，解决 bug 的这家公司到底是什么呢？

你可能早已经看出来了，它就是出品 MATLAB 的 MathWorks。

对，就是那个中国限购的 **MATLAB** —— 很多年以后，它将以另一种形式再次出现在中国公众的茶余饭后谈资中，但那就是另一个话题了。

你看，我们天天讲的自主研发，可真不止是件喊口号的事情呀。

后记

Intel 因为这个 FDIV bug 事件亏了近 5 亿美元，但 MATLAB 则成为了此事的最大赢家。1994 年的 MathWorks 只是个 200 多人规模的小公司，而今天它已经是有超过 5000 名员工的世界巨头了。

在 MathWorks 成立近 40 年后，当年亲自下一线修 bug 的美国工程院院士 Moler，又自己动手为 MATLAB 语言撰写了历史研究文献《A History of MATLAB》。在这份去掉参考文献约 40 多页的资料中，对这个 FDIV bug 的介绍横跨了整整 3 页。这个 bug 的难度与历史地位，由此可见一斑。