Kunpeng GCC特性详解——FTZ

作者：谢志恒
日期：2021.12

Security Level:

目录

关于 Kunpeng GCC

- Kunpeng GCC 是基于开源 GCC 版本进行了二次开发
  > 发布了 GCC 7.3、GCC 9.3 和 GCC 10.3 三个二进制发行版
    - https://www.hikunpeng.com/developer/devkit/compiler?data=hgcc
    - https://mirror.iscas.ac.cn/kunpeng/archive/compiler/kunpeng_gcc （目前只保留了9.3和10.3）
  > 目前只发布了 Linux/aarch64 版本
  > 质量稳定，经过公司内大量测试验证，满足商用稳定质量
  > 性能优化&功能扩展特性
  > 同步安全更新，紧跟 GCC 社区的安全补丁，及时提供安全漏洞修复

  > 在 openEuler 操作系统中支持 GCC 7.3、GCC 9.3 和 GCC 10.3 的 rpm 包发行版
  > 在 openEuler 已开源源码
    - https://gitee.com/openeuler/gcc
  > Kunpeng GCC 用户指南
    - https://support.huaweicloud.com/ug-hgcc-kunpengdevps/kunpenghgcc_06_0001.html
    - https://mp.weixin.qq.com/s/J4zbcozue5Cjnfmd79BBlg

目录

背景介绍

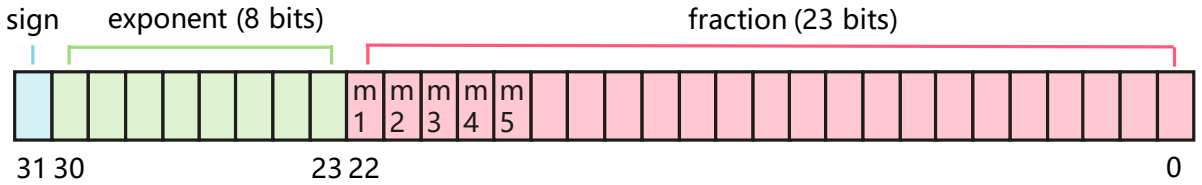
- 在 X86/ICC/Linux 开发环境中，用户或者应用时常会使用-ftz（flush-to-zero）选项，将一些极小数当作 0 处理，可以提升浮点数值计算运算的效率。

- 在 ARM/GCC/Linux 开发环境中，GCC 并未提供单独的选项使能或关闭FTZ功能，而是将其默认包含在-Ofast/-ffast-math/-funsafe-math-optimizations选项当中，使用时并不方便。
  > 用户有时并不想使能过于激进的浮点优化，此时将没有选项单独开启FTZ功能
  > 用户在打开-ffast-math的时候无法单独关闭FTZ功能

- 因此在 Kunpeng GCC 编译器中引入-fftz选项来单独控制FTZ功能

什么是极小数（Subnormal number）？

- 浮点的二进制表示[1]



- normal number
  > exponent >= 1
  > num = $1.m_1m_2m_3m_4...m_{23} * 2^{(exponent - 127)}$
  > float 正数的最小值 0x00800000 (exponent=1, fraction=0) (1.175494e-38)

- subnormal number (denormalized number/denormals)[2]
  > exponent = 0
  > num = $0.m_1m_2m_3m_4...m_{23} * 2^{-127}$

FTZ 功能

- X86 下的 FTZ 功能[2]
  > DAZ（denormals-are-zero）：将计算输入的极小数当作 0
  > FTZ（flush-to-zero）：将计算输出的极小数当作 0

- ARM 下的 FTZ 功能
  > 相当于是 X86 的 DAZ+FTZ（本次介绍提到的FTZ主要指这种模式）

- 极小数带来的性能影响[2]
  > 如果使用系统软件来处理极小数，这将带来巨大的性能损失
  > 如果使用硬件来处理极小数，普遍来说，性能会比处理普通数低一些，在极端场景下，也有可能会有几倍的性能差距

硬件控制

- AArch64 FPCR, Floating-point Control Register[3]

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RES0																															
RES0					AHP	DN	FZ	RMode		Stride		FZ16	Len			IDE	RES0		IXE	UFE	OFE	DZE	IOE	RES0					NEP	AH	FIZ
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

FZ, bit [24]

Flushing denormalized numbers to zero control bit.

AH, bit [1]
When FEAT_AFP is implemented:

Alternate Handling. Controls alternate handling of floating-point numbers.

FIZ, bit [0]
When FEAT_AFP is implemented:

Flush Inputs to Zero. Controls whether single-precision, double-precision and BFloat16 input operands that are denormalized numbers are flushed to zero.

FZ	AH	FIZ	DAZ	FTZ
0	0	0	no	no
0	0	1	yes	no
0	1	0	no	no
0	1	1	yes	no
1	0	0	yes	yes
1	0	1	yes	yes
1	1	0	no	yes
1	1	1	yes	yes

- FEAT_AFP 架构扩展[4]

2020 Architecture Extensions

Feature Name	Short description
FEAT_AFP	Alternate floating-point behavior

FTZ 实现

- 在程序 main 函数执行前修改 FPCR 寄存器
  > libgcc/config/aarch64/crtfastmath.c

```
24 #define _FPU_FPCR_FZ 0x1000000
25
26 #define _FPU_SETCW(fpcr)                                        \
27   {                                                             \
28     __asm__ __volatile__ ("msr  fpcr, %0" : : "r" (fpcr)); \
29   }
30
31 static void __attribute__((constructor))
32 set_fast_math (void)
33 {
34   /* Flush to zero, round to nearest, IEEE exceptions disabled.  */
35   _FPU_SETCW (_FPU_FPCR_FZ);
36 }
```

  > 使用 __attribute__((constructor)) 实现 main 函数执行前执行该函数
  > 内嵌汇编 __asm__ (msr) 实现对 FPCR 寄存器的修改
  > 构建应用程序时链接 crtfastmath.o 实现 FTZ 功能

-fftz 选项实现

- gcc/common.opt 定义 fftz 选项

```
1552 fftz
1553 Common Report Var(flag_ftz) Optimization
1554 Control fpcr register for flush to zero.
1555
```

- gcc/config/aarch64/aarch64-linux.h 使用 GCC SPEC 语法[5]控制 crtfastmath.o 的链接

```
51
52 #define GNU_USER_TARGET_MATHFILE_SPEC \
53   "%{Ofast|ffast-math|funsafe-math-optimizations|fp-model=fast|fftz:\
54   %{!fno-ftz:crtfastmath.o%s}}"
55
```

当编译选项里有[Ofast|ffast-math|funsafe-math-optimizations|fp-model=fast|fftz]并且没有[fno-ftz]时，链接crtfastmath.o

FTZ 测试

- 测试环境
  > CPU：Kunpeng 920
  > OS：openEuler 20.03
  > 编译器：gcc-10.3.1-2021.09-aarch64-linux

- 功能测试
  > 测试是否能够单独开启或关闭FTZ功能

- 性能测试
  > 暂未进行应用上的性能测试分析

FTZ 测试

- 功能测试
  > 用例测试浮点计算 0x00800000 / 2.0

- 构建运行测试用例：PASS
  > (1) O0-O3默认不使能FTZ
  > (2) ffast-math选项默认使能FTZ
  > (3) fftz选项开启FTZ功能
  > (4) fno-ftz选项关闭FTZ功能

```c
#include <stdio.h>

union f2i {
  float f;
  unsigned int ui;
};

int main()
{
  union f2i a, res;
  float b = 2.0;
  a.ui = 0x00800000;

  res.f = a.f / b;
  printf ("res = %x\n", res.ui);
  return 0;
}
```

```
[xzh@localhost ~/openEuler]$ gcc test.c -O0 -g -o test          (1)
[xzh@localhost ~/openEuler]$ ./test
res = 400000
[xzh@localhost ~/openEuler]$ gcc test.c -O0 -g -o test -ffast-math    (2)
[xzh@localhost ~/openEuler]$ ./test
res = 0
[xzh@localhost ~/openEuler]$ gcc test.c -O0 -g -o test -fftz    (3)
[xzh@localhost ~/openEuler]$ ./test
res = 0
[xzh@localhost ~/openEuler]$ gcc test.c -O0 -g -o test -ffast-math -fno-ftz
[xzh@localhost ~/openEuler]$ ./test
                                                                (4)
res = 400000
[xzh@localhost ~/openEuler]$
```

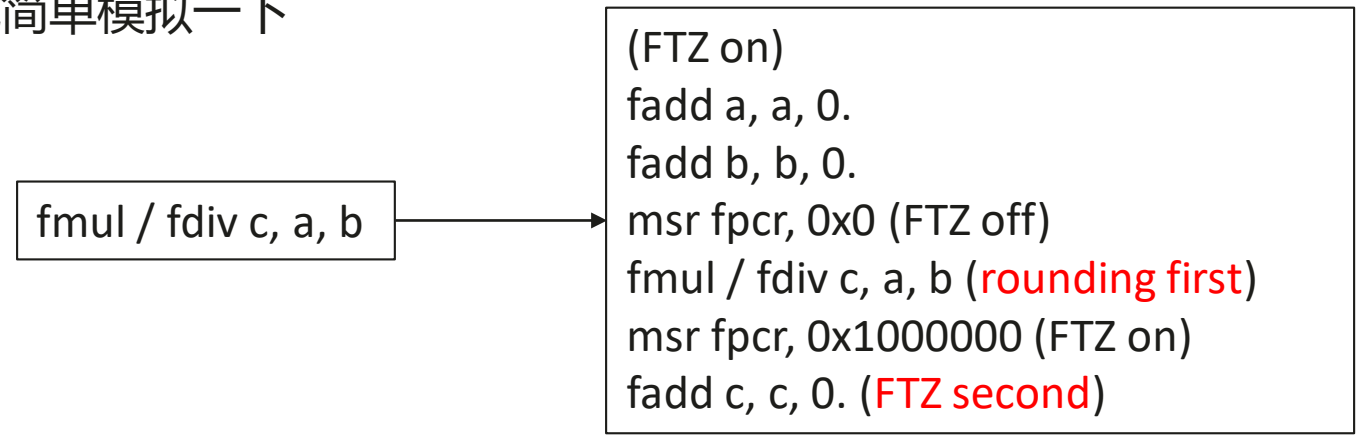
一些其他的思考

- X86/AArch64 下 FTZ 行为的分析
  > 浮点计算例子：0x08b626a1 * 0x3733e512

AArch64-noFTZ	X86-noFTZ	AArch64-FTZ	X86-FTZ
0x800000	0x800000	0x0	0x800000

- 推测 X86 先做 rounding 再做 FTZ，AArch64 先做 FTZ 再做 rounding
  > 0x08b626a1 * 0x3733e512 rounding 前小于边界 0x800000，rounding 后等于边界 0x800000

- 我们可以用以下方式简单模拟一下



一些其他的思考 2

- X86/AArch64 下 FTZ 行为的分析（AArch64 加入了前面的模拟）
  > 浮点计算例子：0x08b44c50 * 0x3735be53

AArch64-noFTZ	X86-noFTZ	AArch64-ALT-FTZ	X86-FTZ
0x800000	0x800000	0x800000	0x0

- 重新推测 X86 在计算中间数据表示中有另一个决定是否是极小数的边界，而不是简单的 float 的 0x800000，而 X86 和 AArch64 对于该值可能不同
  > 推测 X86 尝试将两个 float 操作数转化为 double 后进行乘除法，并使用边界 0x380ffffff0000000 来判断极小数

- 实际 X86 是如何处理计算输出是否需要 FTZ？

参考资料

- [1] https://en.wikipedia.org/wiki/Floating-point_arithmetic
- [2] https://en.wikipedia.org/wiki/Subnormal_number
- [3] https://developer.arm.com/documentation/ddi0595/2021-06/AArch64-Registers/FPCR--Floating-point-Control-Register
- [4] https://developer.arm.com/architectures/cpu-architecture/a-profile/exploration-tools/feature-names-for-a-profile
- [5] https://gcc.gnu.org/onlinedocs/gcc/Spec-Files.html

Thank you.