

Kunpeng GCC 技术剖析——特性概览

作者：谢志恒
日期：2021.09



Security Level:



目录

1. Kunpeng GCC 介绍

2. Kunpeng GCC 功能特性概述

3. Kunpeng GCC 优化特性概述

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 和 GCC 9.3 两个二进制发行版
 - <https://www.hikunpeng.com/developer/devkit/compiler?data=hgcc>
 - > 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

目录

1. Kunpeng GCC 介绍

2. Kunpeng GCC 功能特性概述

3. Kunpeng GCC 优化特性概述

四精度数学库

- GCC aarch64 不支持 libquadmath 库的构建
 - glibc 已经有四精度数学库的支持
 - glibc 和 libquadmath 的四精度数学库符号不同, sinl .vs. sinq
 - 考虑到不同平台的迁移和用户的不同使用习惯
- Kunpeng GCC 支持了 libquadmath 库的构建
 - 需要添加 -DAARCH64_QUADMATH 宏和 -lquadmath 库

```
#include <stdio.h>
#include <quadmath.h>
#include <math.h>

int main()
{
    __float128 input = 1.23456Q;
    __float128 res = 0.0Q;
    int* i = (int*) &res;

    printf("sizeof(__float128) = %d\n", sizeof(__float128));

    res = sinl(input);
    printf("sinl = %08x-%08x-%08x-%08x\n", i[0], i[1], i[2], i[3]);

    res = sinq(input);
    printf("sinq = %08x-%08x-%08x-%08x\n", i[0], i[1], i[2], i[3]);

    return 0;
}
```

```
[xzh@localhost ~/openEuler]$ gcc test.c -O2 -g -o test -lquadmath -DAARCH64_QUADMATH
[xzh@localhost ~/openEuler]$ ./test
sizeof (__float128) = 16
sinl = 9456fce7-28583151-608f3464-3ffee354
sinq = 9456fce7-28583151-608f3464-3ffee354
```

fp-model 浮点模型

- fp-model 是用于控制浮点数计算精度的功能特性
 - 方便满足处理精度敏感性应用的用户的需求
- -fp-model=normal
 - 默认值，等同于不开任何fp-model选项，对其他选项无任何影响
- -fp-model=fast
 - 等同于打开-ffast-math，会开启各种损精度的优化，如交换结合律、强度折减等
- -fp-model=precise
 - 关闭所有影响精度的优化，保证浮点结果的正确性。包括关闭交换结合律、强度折减、向零舍入、fma指令生成等优化
- -fp-model=except
 - 开启浮点计算异常机制，编译器在此做的主要工作是考虑浮点异常的存在，阻碍不考虑浮点异常的优化
- -fp-model=strict
 - 等同于打开以上-fp-model=precise -fp-model=except

Machine Code Model

- GCC aarch64 只支持 mcmmodel=[small | large]

- 当存在符号距离寻址指令的距离大于4GB时, mcmmodel=small的寻址模式会在链接时报relocation truncated to fit错误
- mcmmodel=large不支持PIC模式
- <https://github.com/ARM-software/abi-aa/pull/107/files>

- Kunpeng GCC 支持了 mcmmodel=medium

```
#include <stdio.h>

int a[1024*1024*1024];
int b[1024*1024*1024];

int main()
{
    printf("%d,%d\n", a[0], b[0]);
    return 0;
}
```

```
[xzh@localhost ~/openEuler]$
[xzh@localhost ~/openEuler]$ gcc test.c -O2 -o test -mcmmodel=small
/tmp/cceGgS11.o: in function `main':
test.c:(.text.startup+0x8): relocation truncated to fit: R_AARCH64_ADR_PREL_PG_HI21 against symbol `a' defined in .bss section in /tmp/cceGgS11.o
collect2: error: ld returned 1 exit status
[xzh@localhost ~/openEuler]$ gcc test.c -O2 -o test -mcmmodel=medium
[xzh@localhost ~/openEuler]$ ./test
0,0
[xzh@localhost ~/openEuler]$
```

Flush to zero

- FTZ: 极小数处理为0
 - 浮点的边界在 $1.175494e-38$ ($0x00800000$)
 - `-ffast-math` 会默认启用FTZ, 但没有单独选项控制FTZ
- Kunpeng GCC 支持 `-fftz` 选项来控制启用FTZ

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

```
#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;
}
```


目录

1. Kunpeng GCC 介绍

2. Kunpeng GCC 功能特性概述

3. Kunpeng GCC 优化特性概述

矢量化数学库

- Kunpeng GCC 使能矢量化数学库，在矢量化阶段合理矢量化数学函数的计算
 - 需要提供支持矢量化函数的数学库，如mathlib
 - 矢量化数学函数的命令需符合标准命名格式
 - 开启 -fsimdmath 选项 (gcc test.c -O3 -S -fsimdmath)

```
mov    x19, x0
mov    x20, x1
bl     sinf
str    s0, [x19]
ldr    s0, [x20, 4]
bl     sinf
str    s0, [x19, 4]
ldr    s0, [x20, 8]
bl     sinf
str    s0, [x19, 8]
ldr    s0, [x20, 12]
bl     sinf
str    s0, [x19, 12]
ldr    s0, [x20, 16]
```

```
mov    x19, x0
add    x0, x1, #4
sub    x0, x19, x0
mov    x20, x1
cmp    x0, #8
bls    .L2
ldr    q0, [x1]
bl     _ZGVnN4v_sinf
str    q0, [x19]
ldr    q0, [x20, 16]
bl     _ZGVnN4v_sinf
str    q0, [x19, 16]
ldr    q0, [x20, 32]
bl     _ZGVnN4v_sinf
str    q0, [x19, 32]
ldr    q0, [x20, 48]
bl     _ZGVnN4v_sinf
str    q0, [x19, 48]
ldp   x19, x20, [sp, 16]
ldp   x29, x30, [sp], 32
```

```
#include <mathlib.h>
#include <math.h>

void vec_sin(float a[16], float b[16])
{
    int i;
    for (i = 0; i < 16; i++)
        a[i] = sinf(b[i]);
}
```

冗余循环优化

- 目标：消除冗余的 while 循环

- string_length 函数的主要作用是计算字符串长度
- 当进入 IF 分支时，长度为0
- 当进入 ELSE 分支时，长度非0
- 计算返回值 len 我们只需要知道它是否为0即可，并不需要实际计算长度
- 此时，我们可以不需要 while 来实际计算长度

- string_length 函数需要内联
- -floop-elim 使能冗余循环消除，-ffinite-loops 假设非死循环
- gcc test.c -O2 -floop-elim -ffinite-loops -S

```
.type foo, @function
foo:
.LFB1:
        .cfi_startproc
        cbz     x0, .L2
        ldrsh  w0, [x0]
        cbnz   w0, .L1
.L2:
        b      bar
        .p2align 2,,3
.L1:
        ret
        .cfi_endproc
```

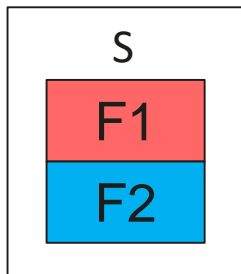
```
inline unsigned int
string_length(const short* const src)
{
    if (src == 0 || *src == 0)
        return 0;
    else
    {
        const short* tmp = src + 1;
        while (*tmp)
            ++tmp;
        return (unsigned int) (tmp - src);
    }
}


extern void bar();


void
foo(const short* const src)
{
    unsigned int len = string_length(src);
    if (!len)
        bar();
}
```

内存空间布局优化 —— 结构体split

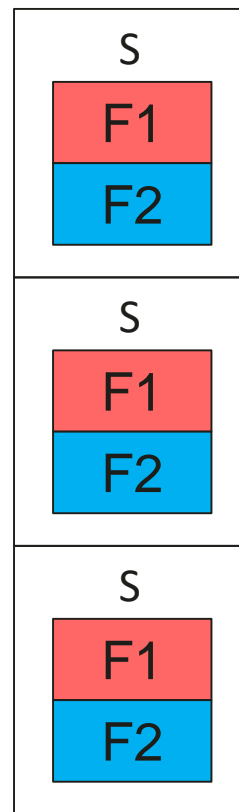
```
结构体 S  
{  
  Field F1;  
  Field F2;  
}
```



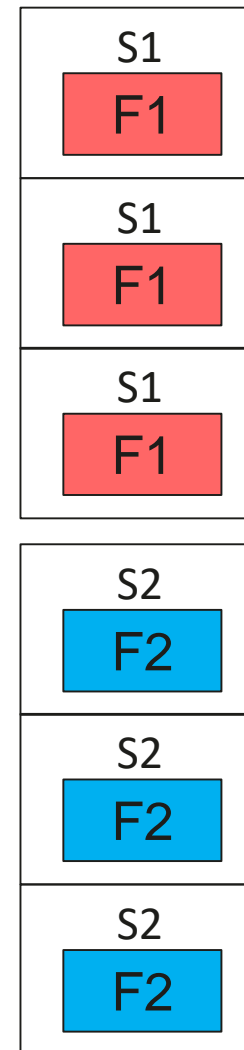
 结构体中**经常**被访问的域

 结构体中**不常**被访问的域

数组 S[N] 在内存中的布局

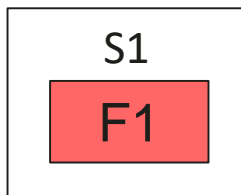


数组 S1[N] 和 S2[N] 在内存中的布局

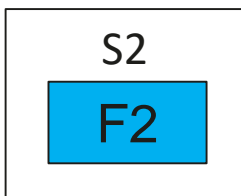


冷热结构体分离

```
结构体 S1  
{  
  Field F1;  
}
```



```
结构体 S2  
{  
  Field F2;  
}
```

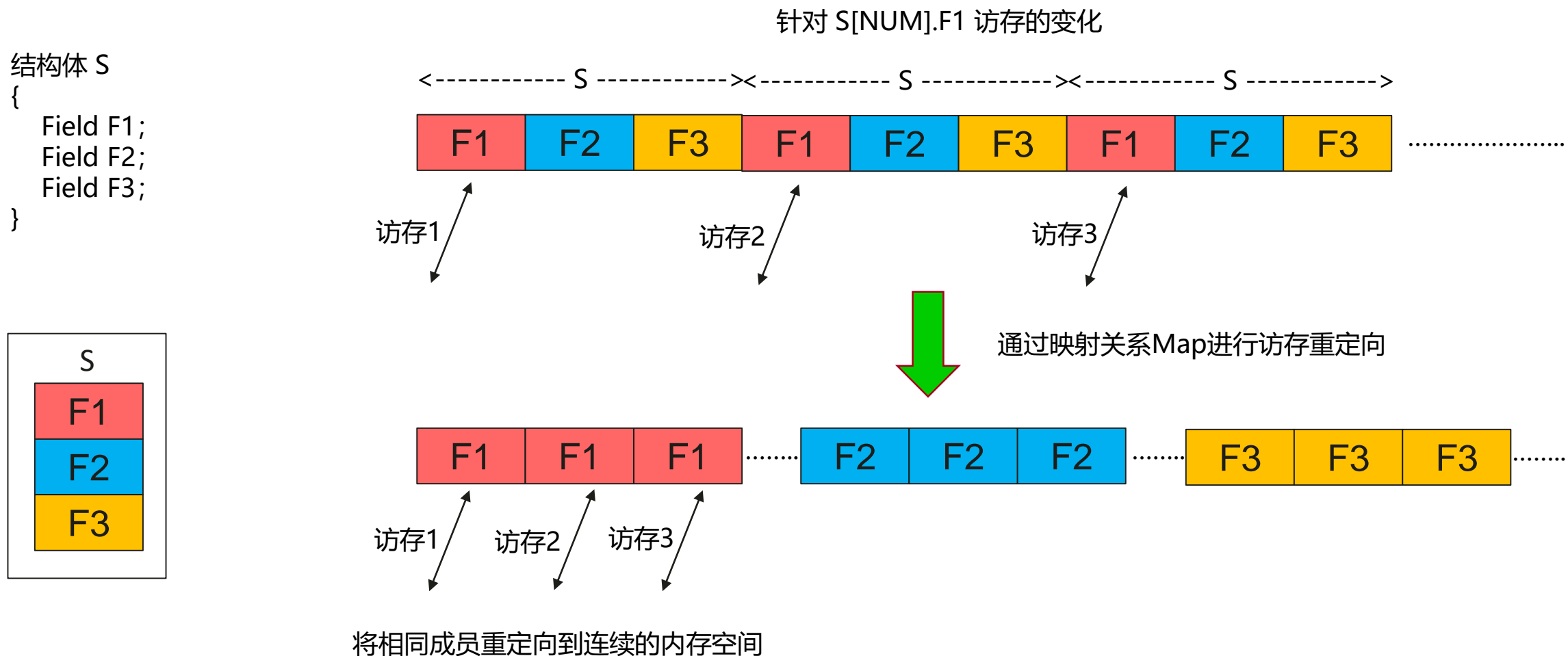


冷热结构体分离



- 把一个结构体拆分成了两个结构体

内存空间布局优化 —— 结构体relayout

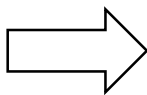


- 把结构体的数组转变成了数组的结构体

内存空间布局优化 —— 结构体reorder

- 把结构体的成员重新排列
 - 可以减少结构体占用体积，如在一次cacheline存取时得到更多的成员

```
typedef struct _test
{
    int a; // 0 4
    double b; // 8
    double c; // 16
    double d; // 24
    short e; // 32 4
    double f; // 40
    double g; // 48
    double h; // 56
    double i; // 64
} test;
```



```
typedef struct _test2
{
    int a; // 0
    short e; // 4
    double b; // 8
    double c; // 16
    double d; // 24
    double f; // 40
    double g; // 48
    double h; // 56
    double i; // 64
} test2;
```

相关的库

- Kunpeng GCC 提供 jemalloc 库
 - -ljemalloc
- Kunpeng GCC 提供算法优化版本的 mathlib 库
 - -lmathlib
- Kunpeng GCC 提供 stringlib 库, 内置了高性能的memset函数
 - -Wl,--wrap=memset -lstringlib

Q & A

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and
organization for a fully connected,
intelligent world.

**Copyright©2018 Huawei Technologies Co., Ltd.
All Rights Reserved.**

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.

