VxWorks on RISC-V:
切换 LLVM 编译器遇到的一些问题分享

VxWorks 7 对 RISC-V 体系架构的支持情况

风河即将在7月17日发布的 VxWorks 7 SR0650 Release 正式支持 RISC-V 体系架构

1. 支持 RV32I / RV64I 加可选扩展（MAFDC）的任意组合，SV32/SV39/SV48 + ASID MMU
2. 支持 Workbench 图形化调试内核态及用户态应用程序（单步、断点）
3. LLVM 10.0.0 + GNU binutils 2.34 工具链（GCC 在去年的 EAR 版本中支持，正式版本不再支持）
4. 支持 QEMU 5.0.0 版本 virt / sifive_u 进行快速模拟仿真
5. 支持 OpenSBI v0.7 + U-Boot v2020.07

切换 LLVM 遇到的问题 1

去年11月底的时候，我们发布过一个支持 RISC-V 的 EAR (Early Access Release) 版本。
当时支持的编译器是 GCC 8.3.0，从今年3月初左右开始切换 LLVM 的工作。

1. 最开始我们用的是 LLVM 9.0.0，是当时最新的 LLVM release。这个版本是第一个正式支持 RISC-V 的 LLVM 编译器。

2. 9.0.0 这个版本有一些问题，比如下面这个代码：

```
void foo (char* cond)
{
    *cond = 0;
}

$ clang --target=riscv64 -march=rv64imafdc -mabi=lp64d -funwind-tables -c -o a.o a.c
$ ldriscv -m elf64lriscv --oformat=elf64-littleriscv a.o
ldriscv: error in a.o(.eh_frame); no .eh_frame_hdr table will be created
```

另外 LLVM 支持的 -mcmodel 选项名称跟 GCC 不兼容，GCC 使用的 medlow / medany，LLVM 使用 small / medium。

3. 上面这些问题经我们后来验证，发现在 10.0.0 版本都已经修复了。
LLVM 10.0.0 (3月24日) 正式 release 后，我们大概在4月下旬左右切换到最新的 10.0.0 版本继续开发工作。

4. LLVM 在处理初值赋 0 的全局变量的时候，跟 GCC 不一样。GCC 都放到数据段了，而 LLVM 很“聪明”的放到了 bss 段。

切换 LLVM 遇到的问题 2

Clang 编译带 __builtin_thread_pointer() 的代码直接崩溃

1. 我们代码中有如下获取 TP 寄存器值的代码，用到了 __builtin_thread_pointer()

```
void * _start()
{
    char * tp = __builtin_thread_pointer();
    return tp;
}

$ clang --target=riscv64 -march=rv64imafdc -mabi=lp64d -c -o a.o a.c
```

```
fatal error: error in backend: Cannot select: intrinsic %llvm.thread.pointer
...
clang: error: clang frontend command failed with exit code 70 (use -v to
see invocation)
clang: note: diagnostic msg:
********************
PLEASE ATTACH THE FOLLOWING FILES TO THE BUG REPORT:
Preprocessed source(s) and associated run script(s) are located at:
clang: note: diagnostic msg: /tmp/tp-86ca29.c
clang: note: diagnostic msg: /tmp/tp-86ca29.sh
clang: note: diagnostic msg:
********************
```

2. 向 LLVM 社区报告了 bug，详见：
https://bugs.llvm.org/show_bug.cgi?id=45303
并且提交了 bug fix (https://reviews.llvm.org/D76828) (merged)

3. 来自 LLVM 社区的 @lenary 和 GCC 社区的 @kito-cheng 讨论后，一致同意 GCC 和 LLVM 对 RISC-V 目标架构支持 __builtin_thread_pointer()。

切换 LLVM 遇到的问题 3

Clang 开 O2 优化在某些特定条件下会错误使用 lwu 指令

1

```
int _start(int * pAddr)
{
    int val;
    int count;

    val = *pAddr;
    count = val & 0x7FFFFFFF;
    __sync_bool_compare_and_swap(pAddr, val, 1);
    return count;
}

$ clang --target=riscv64 -march=rv64imafdc -mabi=lp64d -O2 -c a.c -o a.o
$ ldriscv -m elf64lriscv --oformat=elf64-littleriscv -m elf64lriscv a.o -o a.out
```

```
00000000000100b0 <_start>:
   100b0:   00056583        lwu     a1,0(a0)
   100b4:   80000637        lui     a2,0x80000
   100b8:   4685            li      a3,1
   100ba:   1605272f        lr.w.aqrl a4,(a0)
   100be:   00b71563        bne     a4,a1,100c8 <_start+0x18>
   100c2:   1ed527af        sc.w.aqrl a5,a3,(a0)
   100c6:   fbf5            bnez    a5,100ba <_start+0xa>
   100c8:   fff6051b        addiw   a0,a2,-1
   100cc:   8d6d            and     a0,a0,a1
   100ce:   8082            ret
```

2

在 2 月份的时候，社区有人提交了一个 bug fix：
[LegalizeTypes][RISCV] Correctly sign-extend comparison for ATOMIC_CMP_XCHG (https://reviews.llvm.org/D74453)

Backport 这个 fix 后问题得到了解决。

切换 LLVM 遇到的问题 4

LLVM 10.0.0 引入的优化导致暴露 dynamic linker 的一个 UB 问题
https://github.com/NetBSD/src/blob/trunk/libexec/ld.elf_so/headers.c#L404

1

```
404         } else if (use_pltrela) {
405                 obj->pltrela = (const Elf_Rela *)(obj->relocbase + pltrel);
406                 obj->pltrellim = 0;
407                 obj->pltrelalim = (const Elf_Rela *)(obj->relocbase + pltrel + pltrelsz);
408                 /* On PPC and SPARC, at least, REL(A)SZ may include JMPREL.
409                    Trim rel(a)lim to save time later. */
410                 if (obj->relalim && obj->pltrela &&
411                     obj->relalim > obj->pltrela &&
412                     obj->relalim <= obj->pltrelalim)
413                         obj->relalim = obj->pltrela;
414         }
```

```
40079f6:    beqz    a2,4007a04 <_rtld_digest_dynamic+0x314>
40079f8:    bltu    a4,a0,4007a04 <_rtld_digest_dynamic+0x314>
40079fc:    bgeu    a1,a0,4007a04 <_rtld_digest_dynamic+0x314>
4007a00:    beqz    a3,4007a04 <_rtld_digest_dynamic+0x314>
4007a02:    sd      a1,136(s1)
4007a04:    beqz    s10,4007a0e <_rtld_digest_dynamic+0x31e>
```

2 分析一下这个汇编代码，这里 a2 是 obj->rela，而 a3 是 obj->relocbase，直接对应 C 代码里的 obj->relalim 和 obj->pltrela 两个条件。

obj->relalim = (const Elf_Rela *)((caddr_t)obj->rela + relasz);
obj->pltrela = (const Elf_Rela *)(obj->relocbase + pltrel);

这里的 (caddr_t) 是 (char *)，所以是指针加上整数。看起来没啥问题，但是为啥进入下面的 if 条件判断的时候，直接变成了 obj->rela 和 obj->relocbase?
原来这里碰到了一个空指针加上非 0 偏移值的未定义行为，编译器这里正好聪明的利用了这个 UB 来做这种优化：if (ptr + offset) {...} => if (ptr) {...}
即编译器认为，如果我们判断一个指针加上一个偏移后的值不为空指针，跟这个指针本身不为空指针是等效的！

3 这个问题在 LLVM 9.0.0 版本不复现，10.0.0 合入了下面这个编译优化的改动最终导致了这个 UB 问题被发现。
[InstCombine] icmp eq/ne (gep inbounds P, Idx..), null -> icmp eq/ne P, null (https://reviews.llvm.org/D66608)
GCC 8.3.0 没有暴露这个问题，所以 GCC 的优化还有空间 (?) ☺

切换 LLVM 遇到的问题 5

禁掉 M 扩展后，LLVM Intrinsics __udivdi3 和 __udivmoddi4 互相调用形成死循环

1

```
int _start(unsigned int a, unsigned int b)
{
    if (b != 0)
        return a/b;

    return 0;
}
```

```
$ clang --target=riscv64 -march=rv64iafdc -mabi=lp64d -c a.c -o a.o
$ ldriscv -m elf64lriscv --oformat=elf64-littleriscv -lllvm a.o -o a.out
```

```
00000000000100fc <__udivdi3>:
  ...
  10116:       00e000ef          jal  ra,10124 <__udivmoddi4>
  ...

0000000000010124 <__udivmoddi4>:
  ...
  101a0:       f5dff0ef          jal  ra,100fc <__udivdi3>
  ...
```

2

其实社区早在2019年9月份的时候就有人报告了这个bug：
https://bugs.llvm.org/show_bug.cgi?id=43388
可惜的是一直没有人处理这个问题。

我们提交了 bug fix (https://reviews.llvm.org/D80465) (merged)

切换 LLVM 遇到的问题 6

禁掉 F 和 D 扩展后，LLVM Intrinsics __fixunsdfdi 调用 __fixunsdfdi 形成死循环

1

```
unsigned long long cvt(double x)
{
    return (unsigned long long) x;
}

unsigned long long _start()
{
    return cvt(2.5);
}

$ clang --target=riscv64 -march=rv64imac -mabi=lp64 -c a.c -o a.o
$ ldriscv -m elf64lriscv --oformat=elf64-littleriscv -lllvm a.o -o a.out
```

```
000000000001012c <__fixunsdfdi>:
   ...
   1016e:       15d2                slli    a1,a1,0x34
   10170:       fcb43023            sd      a1,-64(s0)
   10174:       454010ef            jal     ra,115c8 <__divdf3>
   10178:       fb5ff0ef            jal     ra,1012c <__fixunsdfdi>
   1017c:       fca42e23            sw      a0,-36(s0)
   10180:       fe043503            ld      a0,-32(s0)
   10184:       fdc46583            lwu     a1,-36(s0)
   ...
```

2

我们提交了 bug fix 到 LLVM 社区：
[RISCV64] emit correct lib call for fp(double) to ui/si：(https://reviews.llvm.org/D80526) (merged)

@asb 大神亲自接受了这个改动：
“Thanks, this looks good to me. I wasn't aware of MakeLibCallOptions and IsSoften - I think I've wanted something like that before.”

切换 LLVM 遇到的问题 7

1. 在最新的 LLVM 11.0.0 开发分支上，仍然没有 libunwind 32位 RISC-V 的支持。

   只能自己动手了 ^_^
   [RISCV] Support libunwind for riscv32：(https://reviews.llvm.org/D80690)

   目前这个 review 正在进行中。

2. -ftrapping-math 引起 clang RISC-V 后端崩溃
   这个是测试 -ffast-math 选项发现的。-ffast-math 会使能 -fno-trapping-math，意味着生成的代码不会触发浮点异常（除0错或者溢出等）。

   提交了 bug fix 到 LLVM 社区
   [RISCV] Do not crash when using -ftrapping-math：(https://reviews.llvm.org/D81391)

   同样这个 review 也正在进行中。

非 RISC-V 相关的一个 bug：__SOFT_FP__ ？

compiler-rt/lib/builtins/fixunsdfdi.c

1

```
#ifndef __SOFT_FP__
// Support for systems that have hardware floating-point; can set the invalid
// flag as a side-effect of computation.
...

#else
// Support for systems that don't have hardware floating-point; there are no
// flags to set, and we don't want to code-gen to an unknown soft-float
// implementation.
...

#endif
```

2

我们发了一封信到 LLVM 的邮件列表讨论：
__SOFT_FP__ not defined when building compiler-rt for Soft Float
(http://lists.llvm.org/pipermail/llvm-dev/2020-June/142129.html)

Saleem Abdulrasool 回信说他相信这应该是一个笔误 ... 正确的宏应该是 __SOFTFP__
因此我们提了一个 bug，详见 https://bugs.llvm.org/show_bug.cgi?id=46294，并提交了 bug fix (https://reviews.llvm.org/D82014)

总结

GCC 切换到 LLVM 后

1. 性能测试方面，GCC 编译出来的 kernel 和 LLVM 编译出来的 kernel 跑我们的 kernel performance benchmarking 测试结果相差无几。

   静态代码尺寸（text + data + bss）方面，LLVM 编译出来的 kernel / 用户态应用程序相比 GCC 有小小的增加。

2. 为什么切换到 LLVM?

   - 维护两套编译器的成本比较高。
   - 同一个可执行程序，可以交叉编译处不同体系架构的目标码，大幅减小我们发布产品二进制文件的大小。

Q&A

Wind River Press Release for RISC-V

My GitHub Homepage

Bin Meng, VxWorks Engineering Team, Wind River