Compiler Support For Linker Relaxation in RISC-V

Shiva Chen
Hsiangkai Wang

2019 RISC-V Workshop Taiwan

The Idea of Linker Relaxation in RISC-V

- **To reduce the number of instructions to access symbols**
  - Function call can be generated by
    - jal(jump and link)
      - jal can jump to the symbol within +/- 1Mib
    - auipc and jalr pair
      - The pair can jump to the symbol within 32-bit pc-relative address
  - If the symbol and the call site are not in the same compile unit
    - Compiler can not know the symbol offsets
      - auipc and jalr pair will be generated to guarantee the function call can reach the symbol
    - Linker know the symbol address if it's a static linking
      - Linker can transfer the pair to single jal

What is Relocation?

- **The process that linker will fill in the symbol offsets that compiler can not know**
  - Linker will rewrite the offsets according to relocation records
  - Relocation record contain the information
    - Which instructions need to be relocated the offsets
    - Which symbols involved with the relocations
    - How to relocate the fields relative to symbols

```
00000000 <foo>:
   0:                auipc   t1,0x0
   4:                jalr    t1 # 0 <foo>
```

```
Relocation section '.rela.text'
Offset          Type                Sym.Value     Sym. Name + Addend
00000000   R_RISCV_CALL  00000000      bar + 0
```

Relaxation Relocation Type

- **With the relocation types, linker can eliminate the instructions if the offset can fit in single instruction**
  - Emit extra relocation type "R_RISCV_RELAX"
    - To indicate the instructions can do the relaxation

Linker could relax auipc and jalr to jal

```
00000000 <foo>:
   0:                auipc   t1,0x0
   4:                jalr    t1 # 0 <foo>
```

```
Relocation section '.rela.text'
Offset      Type                Sym.Value     Sym. Name + Addend
00000000    R_RISCV_CALL        00000000       bar + 0
00000000    R_RISCV_RELAX                0
```

GP Base Relaxation

- **Could we relax to single instruction which is not pc-relative?**
  - Relax to gp base instructions
    - If the offsets between gp and the symbols could fit in the instructions

```
int sym;
void foo (int *a) {
  *a = sym;
}
```

```
foo:
        lui     a5,%hi(sym)
        lwi     a5,%lo(sym)(a5)
        sw      a5,0(a0)
        ret
```

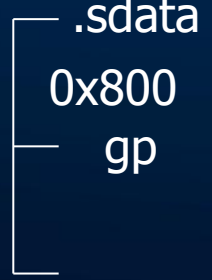
```
foo:
        lwi     a5, offset_from_gp_to_sym(gp)
        sw      a5,0(a0)
        ret
```

GP Base Relaxation

How to assign the gp value for linker?

- gp = __global_pointer$ = start_address_of_sdata + 0x800
  - start_address_of_sdata = gp – 0x800 = gp + Min(SImm12)

```
.sdata          :
 {
   . = ALIGN (32 / 8);
   __global_pointer$ = . + 0x800;
   *(.srodata.cst16) *(.srodata.cst8) *(.srodata.cst4) *(.srodata.cst2)
   *(.srodata .srodata.*) *(.sdata .sdata.* .gnu.linkonce.s.*)
 }
```



Default linker script dump by `riscv32-elf-ld –verbose`

GP Base Relaxation

- **Guard the gp initialization with .option norelax**
  - To avoid gp initial instructions relax to "mv gp, gp"

```
 # Initialize global pointer
.option push
.option norelax
1:auipc gp, %pcrel_hi(__global_pointer$)
  addi  gp, gp, %pcrel_lo(1b)
.option pop
```

RISC-V newlib crt0.S

Candidates of RISC-V Linker Relaxation

Pseudo Candidates	Candidates	After Relaxation
	lui a0, %hi(sym) addi a0, a0, %lo(sym)	addi a0, gp_offset(gp)
	lui a0, %hi(sym) load/store a1, %lo(sym)(a0)	load/store a1, gp_offset(gp)
call sym	.L1: auipc a0, %pcrel_hi(sym) jalr ra, %pcrel_lo(.L1)(a0)	jal sym
lla a0, sym	.L1: auipc a0, %pcrel_hi(sym) addi a0, a0, %pcrel_lo(.L1)	addi a0, gp_offset(gp)
lla a0, sym load/store a1, 0(a0)	.L1: auipc a0, %pcrel_hi(sym) load/store a1, %pcrel_lo(.L1)(a0)	load/store a1, gp_offset(gp)
	lui a0, %tprel_hi(sym) add a0, a0, tp, %tprel_add(sym) load/store a0, %tprel_lo(sym)(a0)	load/store a0, tp_offset(tp)

Issues when Implement Relaxation in LLVM

- Branch Offsets Changed by Relaxation
- LLVM assembler always transfer C extension branches to 32-bit form
- Label Difference Changed by Relaxation
- Alignment Broke by Relaxation
- Incorrect Debug Information due to Relaxation

Branch Offsets Changed by Relaxation

- **Branches offsets may change by relaxation**
  - Preserve R_RISCV_BRANCH relocation in object files
    - Linker can recalculate the branch offsets
    - Define shouldForceRelcation target hook in RISC-V LLVM Backend
      - Return true when the relaxation enabled to preserve relocation types

```
foo:
        beqz    a0, .L1   <-- R_RISCV_BRANCH
        lui     a5,%hi(sym)
        lwi     a0,%lo(sym)(a5)
.L1
        ret
```

LLVM assembler always transfer C extension branches to 32-bit form

- **LLVM assembler may transfer C extension instructions to 32-bit form**
  - To make sure the offsets can fit in the instructions
  - When we force to leave relocation types for relaxation
    - LLVM will assume all the branch targets are unknown
      - Always transfer C extension branches to 32-bit form
    - Add WasForce to mark the relocation types are forced to leave for relaxation and the transformation will depend on branch offsets

```
foo:
        c.beqz    a0,  .L1
        lui       a5,%hi(sym)
.L1
        ret
```

```
foo:
        beqz      a0,  .L1
        lui       a5,%hi(sym)
.L1
        ret
```

Label Difference Changed by Relaxation

- **LLVM don't expect the code will shrink after linking**
  - Label difference will be calculated as a constant before linking
  - To fix the label difference
    - Preserve R_RISCV_ADD32 and R_RISCV_SUB32

```
.L0
        lui     a5,%hi(sym)
        lwi     a0,%lo(sym)(a5)
.L1

.data
.word .L0 - .L1   <-- R_RISCV_ADD32
                      R_RISCV_SUB32
```

NewDiff = 0
R_RISCV_ADD32:
NewDiff = NewDiff + .L0_Addr
R_RISCV_SUB32:
NewDiff = NewDIff – .L1_Addr

Alignment Broke by Relaxation

- **Alignment setting by .align may change by linker relaxation**
  - To fix the alignment
    - Insert extra NOPs when linker relaxation enabled
    - Linker can satisfy the alignment by removing NOPs.

```
        lui     a5,%hi(sym)
        lwi     a0,%lo(sym)(a5)
.p2align 3
        add     a1, a1, a2
```

Relaxation disabled

```
        lui     a5,%hi(sym)
        lwi     a0,%lo(sym)(a5)
        NOP  <-- R_RISCV_ALIGN
        NOP
        ...
.p2align 3
        add     a1, a1, a2
```

Before Relaxation

```
        lwi     a0, offset(gp)
        NOP
.p2align 3
        add     a1, a1, a2
```

After Relaxation

Debug Problem After Relaxation

```
18:         lui     a0, 0
1c:         mv      a0, a0
20:         auipc   t1, 0
24:         jalr    t1
```

Link with relaxation.

```
10204:      lui     a0, 24
10208:      addi    a0, a0, 272
1020c:      c.jal   436
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5   printf("hello world!\n");
6
7   return 0;
8 }
```

Debug Problem After Relaxation

```
10204:      lui     a0, 24
10208:      addi    a0, a0, 272
1020c:      c.jal   436
1020e:      mv      a0, zero
10212:      addi    sp, s0, -16
10216:      lw      ra, 12(sp)
1021a:      lw      s0, 8(sp)
```

llvm-dwarfdump --debug-line

Address	Line	Column	File	ISA	Discriminator
0x00000000000101ec	4	0	1	0	0
0x0000000000010204	5	3	1	0	0
0x0000000000010214	7	3	1	0	0
0x000000000001022c	7	3	1	0	0

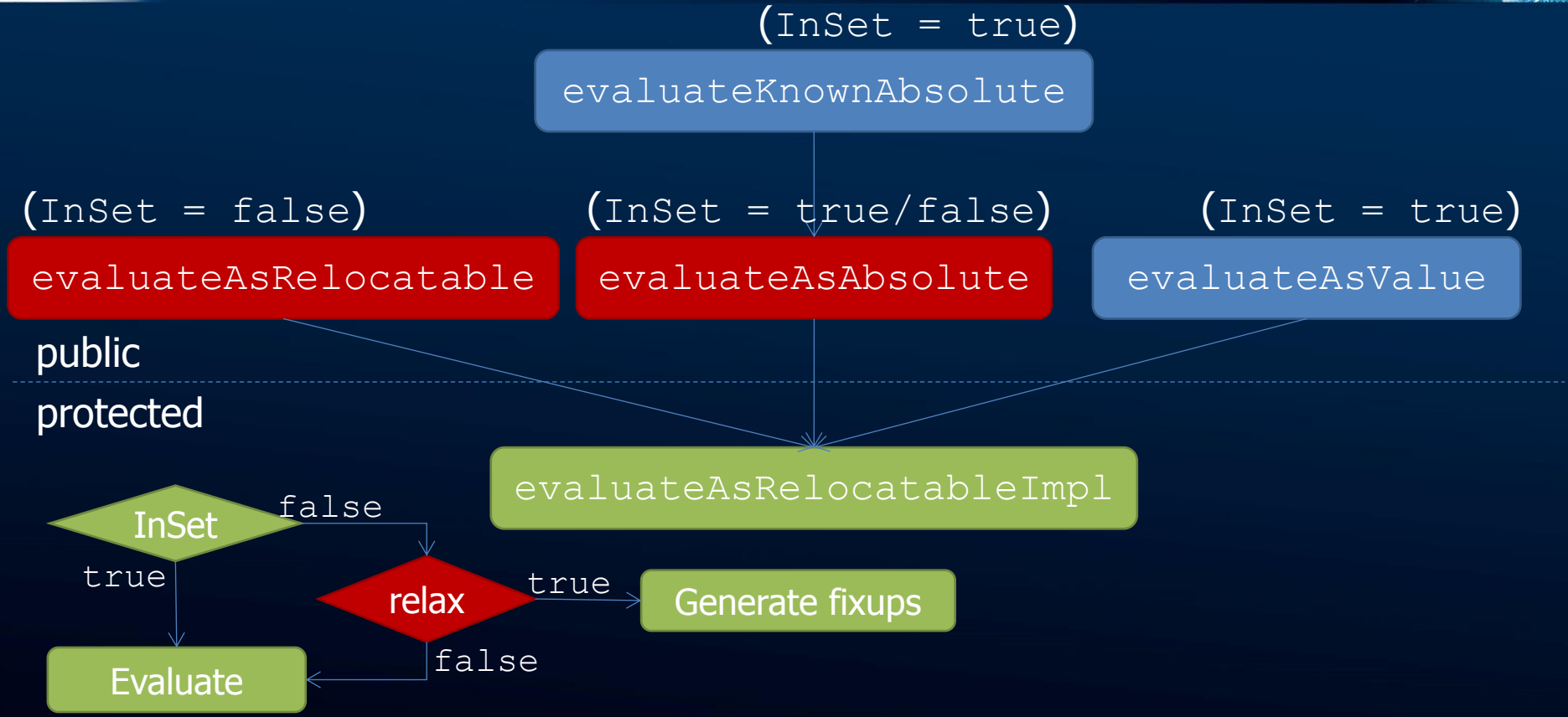
Debug Problem After Relaxation

```
1020e:        mv      a0, zero
10212:        addi    sp, s0, -16
10216:        lw      ra, 12(sp)
1021a:        lw      s0, 8(sp)
```

```
(gdb) b 7
Breakpoint 1 at 0x10214: file main.c, line 7.
(gdb) r
Starting program: /home/users/kai/sandbox/llvm/debug-line/relax
hello world!
core: 4 byte read to unmapped address 0xfffff90c at 0x10216

Program received signal SIGSEGV, Segmentation fault.
0x00010216 in main () at main.c:7
7           return 0;
```

Evaluation Interface in MCExpr



Debug Info In General

```
.byte   2                               # Abbrev [2]
0x26:0x15 DW_TAG_subprogram
.long   .Lfunc_begin0                   # DW_AT_low_pc
.long   .Lfunc_end0-.Lfunc_begin0       # DW_AT_high_pc
.byte   1                               # DW_AT_frame_base
.byte   156
.long   .Linfo_string3                  # DW_AT_name
.byte   1                               # DW_AT_decl_file
.byte   3                               # DW_AT_decl_line
.long   59                              # DW_AT_type
```

Use `MCExpr::evaluateAsAbsolute(InSet = false)` to evaluate the expression. Generate as `R_RISCV_ADD32` and `R_RISCV_SUB32` relocations.

Incorrect Debug Info

- **LLVM does not take care of relaxation in debug information.**
  - Debug line information.
  - Debug frame information.

Debug Line and Debug Frame

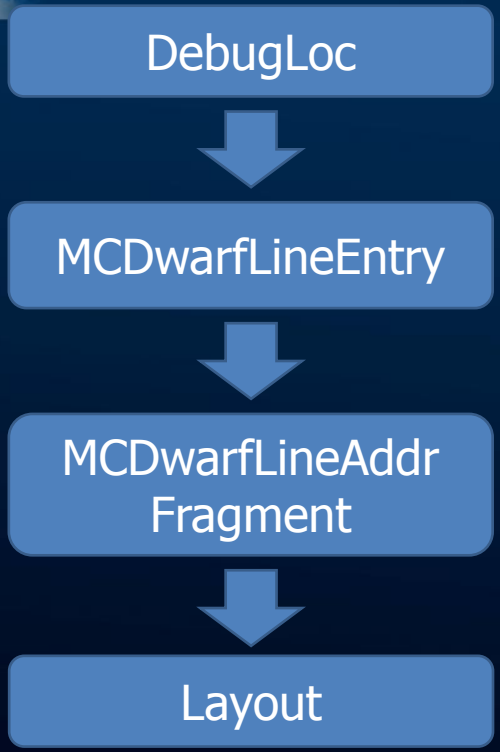
```
addi     s0, sp, 16
.cfi_def_cfa s0, 0
mv       a0, zero
sw       a0, -12(s0)

.loc     1 5 3 prologue_end
lui      a0, %hi(.L.str)
addi     a0, a0, %lo(.L.str)
call     printf
.loc     1 7 3
mv       a0, zero
addi     sp, s0, -16
lw       ra, 12(sp)
lw       s0, 8(sp)
.cfi_def_cfa sp, 16
```

Address difference between .loc.

Address difference between CFI instructions.

How LLVM Handle Debug Line



DwarfDebug::beginInstruction() to determine if the instruction is located at a new line.

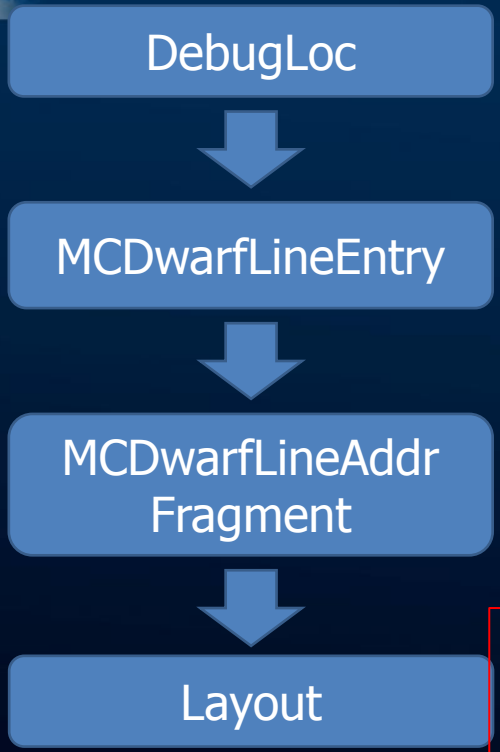
Convert DebugLoc to (Symbol, DebugLoc).

If the address difference expression could not be resolved by MCExpr::evaluateAsAbsolute(), convert the expression to a fragment.

Use MCExpr::.evaluateKnownAbsolute() to resolve the address difference and output actual value.

The problem is MCAssembler assumes the expression could be resolved in assemble time.

How LLVM Handle Debug Line



DwarfDebug::beginInstruction() to determine if the instruction is located at a new line.

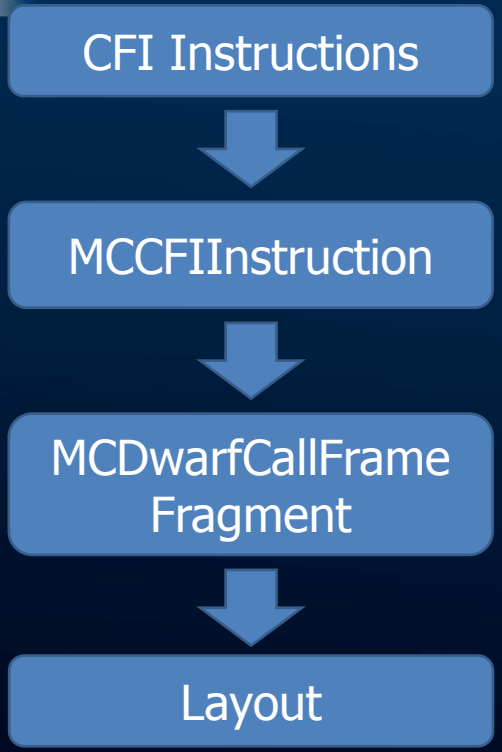
Convert DebugLoc to (Symbol, DebugLoc).

If the address difference expression could not be resolved by MCExpr::evaluateAsAbsolute(), convert the expression to a fragment.

Use MCExpr::.evaluateKnownAbsolute() to resolve the address difference and output actual value. It should generate fixups.

Use fixed length encoding for address offset instead of ULEB128 encoding.

How LLVM Handle Debug Frame



Handle CFI MachineInstr.

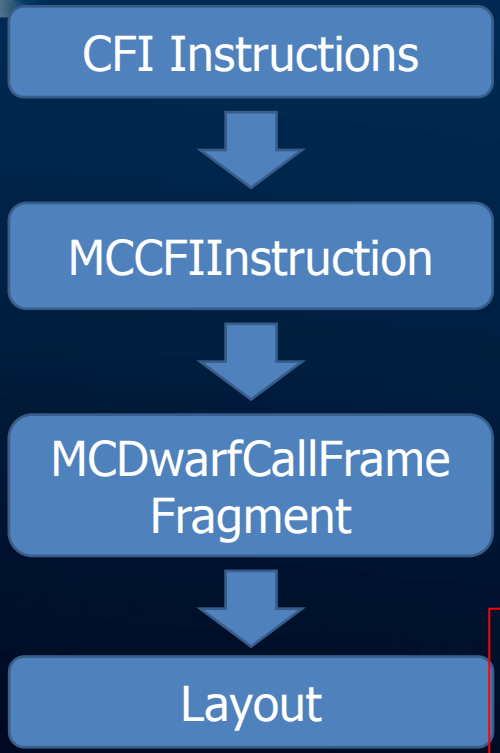
Convert CFI MachineInstr to MCCFIInstructions.

If the address difference expression could not be resolved by MCExpr::evaluateAsAbsolute(), convert the expression to a fragment.

Use MCExpr::.evaluateKnownAbsolute() to resolve the address difference and output actual value.

The problem is MCAssembler assumes the expression could be resolved in assemble time.

How LLVM Handle Debug Frame



Handle CFI MachineInstr.

Convert CFI MachineInstr to MCCFIInstructions.

If the address difference expression could not be resolved by MCExpr::evaluateAsAbsolute(), convert the expression to a fragment.

Use MCExpr::.evaluateKnownAbsolute() to resolve the address difference and output actual value. It should generate fixups.

Need a new fixup type for 6-bits offset.

Thank you