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'

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Sym.Value</th>
<th>Sym. Name + Addend</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000</td>
<td>R_RISCV_CALL</td>
<td>00000000</td>
<td>bar + 0</td>
</tr>
</tbody>
</table>
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'

<table>
<thead>
<tr>
<th>Offset</th>
<th>Type</th>
<th>Sym.Value</th>
<th>Sym. Name + Addend</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000</td>
<td>R_RISCV_CALL</td>
<td>000000000</td>
<td>bar + 0</td>
</tr>
<tr>
<td>000000000</td>
<td>R_RISCV_RELAX</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
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

```c
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?

- \( gp = \_\_\_\_t l l s t a r t \_a d d r e s s \_o f \_s d a t a + 0x800 \)
  - \( \text{start} \_\_\_\_\_\_t l l s t a r t \_a d d r e s s \_o f \_s d a t a = gp - 0x800 = gp + \text{Min(SImm12)} \)

\[
\begin{align*}
\text{.sdata} & : \\
\{ & \\
& . = \text{ALIGN} (32 / 8); \\
& \_\_\_\_\_\_\_t l l s t a r t \_a d d r e s s \_o f \_s d a t a = . + 0x800; \\
& *(.srodata.cst16) *(.srodata.cst8) *(.srodata.cst4) *(.srodata.cst2) \\
& *(.srodata .srodata.*) *(.sdata .sdata.* .gnu.linkonce.s.*) \\
\}
\end{align*}
\]

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”

```asm
# 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

<table>
<thead>
<tr>
<th>Pseudo Candidates</th>
<th>Candidates</th>
<th>After Relaxation</th>
</tr>
</thead>
<tbody>
<tr>
<td>lui a0, %hi(sym)</td>
<td>addi a0, a0, %lo(sym)</td>
<td>addi a0, gp_offset(gp)</td>
</tr>
<tr>
<td>lui a0, %hi(sym)</td>
<td>load/store a1, %lo(sym)(a0)</td>
<td>load/store a1, gp_offset(gp)</td>
</tr>
<tr>
<td>call sym</td>
<td>.L1: auipc a0, %pcrel_hi(sym) jalr ra, %pcrel_lo(.L1)(a0)</td>
<td>jal sym</td>
</tr>
<tr>
<td>lla a0, sym</td>
<td>.L1: auipc a0, %pcrel_hi(sym) addi a0, a0, %pcrel_lo(.L1)</td>
<td>addi a0, gp_offset(gp)</td>
</tr>
<tr>
<td>lla a0, sym</td>
<td>.L1: auipc a0, %pcrel_hi(sym) load/store a1, %pcrel_lo(.L1)(a0)</td>
<td>load/store a1, gp_offset(gp)</td>
</tr>
<tr>
<td></td>
<td>lli a0, %tprel_hi(sym) add a0, a0, tp, %tprel_add(sym) load/store a0, %tprel_lo(sym)(a0)</td>
<td>load/store a0, tp_offset(tp)</td>
</tr>
</tbody>
</table>
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 shouldForceRelocation target hook in RISC-V LLVM Backend
    - Return true when the relaxation enabled to preserve relocation types

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

R_RISCV_BRANCH
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
    .word  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.

Before Relaxation

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Before Relaxation</th>
<th>After Relaxation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lui a5,%hi(sym)</code></td>
<td></td>
<td><code>lwi a0, offset(gp)</code></td>
</tr>
<tr>
<td><code>lwi a0,%lo(sym)(a5)</code></td>
<td></td>
<td><code>NOP</code></td>
</tr>
<tr>
<td><code>.p2align 3</code></td>
<td></td>
<td><code>NOP</code></td>
</tr>
<tr>
<td><code>add a1, a1, a2</code></td>
<td></td>
<td><code>add a1, a1, a2</code></td>
</tr>
</tbody>
</table>

Relaxation disabled

R_RISCV_ALIGN

Taking RISC-V® Mainstream
Debug Problem After Relaxation

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

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

Link with relaxation.

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

Taking RISC-V® Mainstream
## Debug Problem After Relaxation

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Line</th>
<th>Column</th>
<th>File</th>
<th>ISA</th>
<th>Discriminator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000000101ec</td>
<td>lui a0, 24</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x0000000000010204</td>
<td>addi a0, a0, 272</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x0000000000010214</td>
<td>c.jal 436</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x000000000001022c</td>
<td>mv a0, zero</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

`llvm-dwarfdump --debug-line`
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 0xffffffff90c at 0x10216

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

- evaluateKnownAbsolute
- evaluateAsAbsolute
- evaluateAsValue
- evaluateAsRelocatable
- evaluateAsRelocatableImpl

```
InSet = true
evaluateKnownAbsolute

(InSet = false)
evaluateAsRelocatable

(InSet = true/false)
evaluateAsAbsolute

(InSet = true)
evaluateAsValue

public

protected

false

true

false

true

false

relax

Generate fixups

Evaluate

fixups

false

true
Debug Info In General

```assembly
.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.
Taking RISC-V® Mainstream

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

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

2. **MCDwarfLineEntry**
   - Convert DebugLoc to (Symbol, DebugLoc).

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

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

5. **Layout**
   - The problem is MCAssembler assumes the expression could be resolved in assemble time.
How LLVM Handle Debug Line

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

- **MCDwarfLineEntry**
  - Convert DebugLoc to (Symbol, DebugLoc).

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

- **Fragment**
  - Use MCElr::evaluateKnownAbsolute() to resolve the address difference and output actual value.

- **Layout**
  - Use fixed length encoding for address offset instead of ULEB128 encoding.

It should generate fixups.
How LLVM Handle Debug Frame

- **CFI Instructions**: Handle CFI MachineInstr.
- **MCCFIInstruction**: Convert CFI MachineInstr to MCCFIInstructions.
- **MCDwarfCallFrame Fragment**: If the address difference expression could not be resolved by `MCE::evaluateAsAbsolute()`, convert the expression to a fragment.
- **Layout**: Use `MCE::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

CFI Instructions → Handle CFI MachineInstr.

MCCFIInstruction → Convert CFI MachineInstr to MCCFIInstructions.

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

Layout → Use MCEexpr::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