Implementing machine code optimizations for RISC-V

Lewis Revill
Shrink wrapping

foo:
    addi sp, sp, -16
    sw ra, 12(sp)
    ...
    beq a0, a1, .Lend
    call bar
    ...
.Lend:
    ...
    lw ra, 12(sp)
    addi sp, sp, 16
    ret

‘ra’ used for call
Shrink wrapping

```
foo:
  addi sp, sp, -16
  sw ra, 12(sp)
  ...
  beq a0, a1, .Lend

  call bar
  ...

.Lend:
  ...
  lw ra, 12(sp)
  addi sp, sp, 16
  ret
```
Shrink wrapping

foo:
  addi sp, sp, -16
  sw ra, 12(sp)
  ...
  beq a0, a1, .Lend

  call bar
  ...

.Lend:
  ...

  lw ra, 12(sp)
  addi sp, sp, 16
  ret

Actually, ‘ra’ is only used if execution reaches this block...
Shrink wrapping

```
 foo:
   addi sp, sp, -16
   sw ra, 12(sp)
   ...
   beq a0, a1, .Lend
   call bar
   ...
 .Lend:
   ...
   lw ra, 12(sp)
   addi sp, sp, 16
   ret
```

```
 foo:
   ...
   beq a0, a1, .Lend
   addi sp, sp, -16
   sw ra, 12(sp)
   call bar
   ...
   lw ra, 12(sp)
   addi sp, sp, 16
 .Lend:
   ...
   ret
```
Shrink wrapping

- Minimal changes necessary in RISC-V backend.
  - Remove ‘shrink wrapping is not yet supported’ assertion
  - Correctly calculate epilogue insertion point

```cpp
// If this is not a terminator, the actual insert location should be after the last instruction.
if (!MBBI->isTerminator())
    MBBI = std::next(MBBI);
```
Save/restore

foo:
```assembly
call foo
```
```assembly
addi sp, sp, -16
sw ra, 12(sp)
...
```
```assembly
lw ra, 12(sp)
addi sp, sp, 16
ret
```

bar:
```assembly
call bar
```
```assembly
addi sp, sp, -16
sw ra, 12(sp)
sw s0, 8(sp)
...
```
```assembly
lw s0, 8(sp)
lw ra, 12(sp)
addi sp, sp, 16
ret
```

Save and restore code may be needed a lot throughout a program, and can start to look very familiar.
Save/restore

...__riscv_save_3:__riscv_save_2:__riscv_save_1:__riscv_save_0:
    addi sp, sp, -16
    sw s2, 0(sp)
    sw s1, 4(sp)
    sw s0, 8(sp)
    sw ra, 12(sp)
    jr t0
...

People have noticed this before. Minimized routines available in libgcc.

...__riscv_restore_3:__riscv_restore_2:__riscv_restore_1:__riscv_restore_0:
    lw s2, 0(sp)
    lw s1, 4(sp)
    lw s0, 8(sp)
    lw ra, 12(sp)
    addi sp, sp, 16
    ret
...

...
Save/restore

foo:
    addi sp, sp, -16
    sw ra, 12(sp)
    ...
    lw ra, 12(sp)
    addi sp, sp, 16
    ret

bar:
    addi sp, sp, -16
    sw ra, 12(sp)
    sw s0, 8(sp)
    ...
    lw s0, 8(sp)
    lw ra, 12(sp)
    addi sp, sp, 16
    ret

...
Save/restore

- Save/restore is not a pass, but needs to be manually implemented.
  - For each register saved by libcall, indicate that it shouldn’t have save/restore code automatically generated
  - Insert call via t0 to the save libcall at the beginning of the prologue
  - Insert tail call to the restore libcall at the end of the epilogue
Save/restore

• Using libcalls introduces various complexities.
  - Inserting a tail call to a libcall means we cannot have any other tail calls in the function
  - A frame may now have additional stack adjustment within a libcall to keep track of

```c
uint64_t StackSize = MFI.getStackSize();
uint64_t RealStackSize = StackSize + RVFI->getLibCallStackSize();
```
Machine outlining

... or a2, a2, a1
addi a0, a0, -2
srli a1, a1, 1
sw a0, 0(a2)
sw a1, 4(a2)
lw a2, 0(a1)
...
... or a2, a2, a1
addi a0, a0, -2
srli a1, a1, 1
sw a0, 0(a2)
sw a1, 4(a2)
lw a2, 0(a1)
...
Machine outlining

```assembly
... or a2, a2, a1 addi a0, a0, -2 srl a1, a1, 1 sw a0, 0(a2) sw a1, 4(a2) lw a2, 0(a1) ...

or a2, a2, a1 addi a0, a0, -2 srl a1, a1, 1 sw a0, 0(a2) sw a1, 4(a2) lw a2, 0(a1) ...

OUTLINED_FUNCTION_0:
or a2, a2, a1 addi a0, a0, -2 srl a1, a1, 1 sw a0, 0(a2) sw a1, 4(a2) lw a2, 0(a1) jr t0
```

... call t0, OUTLINED_FUNCTION_0 ...

... call t0, OUTLINED_FUNCTION_0 ...

...
Machine outlining

• Use target hooks to provide details to the machine outliner.
  - Which functions/basic blocks are safe to outline from?
  - Which instructions can be outlined?
  - What is the benefit of outlining a set of candidates?

• Build the outlined function, and insert calls
  - Same ‘call through t0’ approach as save/restore
Results
Results
Thank you!

www.embecosm.com