hw12实验过程记录

参考资料

1. ARM and Thumb-2 Instruction Set Quick Reference Card
2. Vector Floating Point Instruction Set Quick Reference Card (arm.com)
3. SSA-based Compiler Design，3.2，Destruction

实验过程

ϕ 函数的消除

参考[3]，利用critical edge splitting算法将 ϕ 函数转变成parallel copy instruction。

具体来说，对每个 ϕ 函数，设其所在块为 B，前驱为 B_i (即 ϕ 函数形式为 $a_0 = \phi(B_1 : a_1, \cdots, B_n : a_n)$)，若 B 有多个前驱且 B_i 有多个后继，则创建一个新块 B'_i，并用 $B_i \to B'_i$，$B'_i \to B$ 来替换边 $B_i \to B$，在 B'_i 中插入 $a_0 \leftarrow a_i$；否则直接在 B_i 中插入 $a_0 \leftarrow a_i$。

在实际实现时，为每个块 B 维护一个原始前驱块label到splitting完成后的实际前驱块的映射 (`S_table parallelCopyTab`)，由此在消除 ϕ 函数时就可以直接根据原来存储的label找到对应插入Move指令的前驱块。

```c
AS_instrList SSA_destruction(AS_instrList bodyil, G_graph ssa_bg) {
  int originNodeCnt = ssa_bg->nodecount;
  for (G_nodeList p = G_nodes(ssa_bg); p && originNodeCnt; p = p->tail, --originNodeCnt)
{
    G_node v = p->head;
    AS_block b = G_nodeInfo(v);
    if (!isBlockContainPhi(b)) {
      continue;
    }

    // critical edge splitting
    S_table parallelCopyTab = S_empty();

    G_nodeList q = G_pred(v);
    while (q) {
      G_node u = q->head;
      q = q->tail;
      AS_block pred = G_nodeInfo(u);
      if (!isNodeHasMultiplePred(v) || !isNodeHasMultipleSucc(u)) {
        S_enter(parallelCopyTab, S_Symbol(Temp_labelstring(pred->label)), (void *)u);
        continue;
      } else {
        // split the edge: pred -> pred', pred' -> n
        G_rmEdge(u, v);
        G_node newNode = splitNewBlock(ssa_bg, u, v);
        S_enter(parallelCopyTab, S_Symbol(Temp_labelstring(pred->label)), (void
*)newNode);
      }
    }

    // deconstruct phi functions
    for (AS_instrList pre = b->instrs, cur = pre->tail; cur; cur = cur->tail) {
      if (!is_phi_func(cur->head)) {
        // no phi func now
        break;
      }

      Temp_temp dst = cur->head->u.OPER.dst->head;
      Temp_tempList srcs = cur->head->u.OPER.src;
      Temp_labelList labels = cur->head->u.OPER.jumps->labels;

      for (; srcs; srcs = srcs->tail, labels = labels->tail) {
        Temp_temp src = srcs->head;
        Temp_label label = labels->head;

        // find the block that contains the label
        G_node u = (G_node)S_look(parallelCopyTab, S_Symbol(Temp_labelstring(label)));
        AS_block pred = G_nodeInfo(u);

        // insert a move instruction
        AS_instrList prevLastIns = pred->instrs;
        AS_instrList lastIns = prevLastIns->tail;
        while (!isInstrCmpOrJump(lastIns->head)) {
          prevLastIns = lastIns;
          lastIns = lastIns->tail;
        }
        AS_instr moveIns = NULL;
        switch (src->type) {
          case T_int: {
            moveIns =
                AS_Move("\%`d0 = add i64 \%`s0, 0", Temp_TempList(dst, NULL),
                        Temp_TempList(src, NULL));
            break;
          }
          case T_float: {
            moveIns =
                AS_Move("\%`d0 = fadd double \%`s0, 0.0",
                        Temp_TempList(dst, NULL), Temp_TempList(src, NULL));
            break;
          }
          default:
            break;
        }
        prevLastIns->tail = AS_InstrList(moveIns, lastIns);
      }
```

```c

      // remove the phi function
      pre->tail = cur->tail;
    }
  }

  // reorder the blocks
  reorderBlocks(ssa_bg);

  AS_instrList result = NULL;
  for (G_nodeList p = G_nodes(ssa_bg); p; p = p->tail) {
    AS_block b = G_nodeInfo(p->head);
    for (AS_instrList q = b->instrs; q; q = q->tail) {
      result = AS_splice(result, AS_InstrList(q->head, NULL));
    }
  }

  return result;
}
```

翻译成ARM指令

prologue

1. 解析函数名，输出相应的label
2. 将 `old fp` 压栈，并设置当前栈帧 (`mov fp, sp`)
3. 将callee saved registers压栈保存 (包括 `lr`)
4. 根据ARM调用公约，将函数参数移到对应的temp中
    - 首先从寄存器中获取 (整型是 `r0-r3`，浮点数是 `s0-s15`)
    - 如果寄存器不够用，剩余的参数从栈中获取 (`ldr temp, [fp, #i]`, $i = 4, 8, \cdots$)

body

call

1. 将参数从temp移到对应的寄存器中
    - 顺序和之前prologue时获取参数的顺序一致
2. 用 `blx` 跳转到寄存器存储的对应位置
3. 如果该函数有返回值，将该返回值从 `r0/s0` Move到对应的temp中

ret

1. 将返回值Move到 `r0/s0` 中
2. 将 `sp` 设置为 `fp - 32`，并将栈中保存的callee saved registers弹出 (此时自动恢复了 `lr`)
3. 弹出 `old fp`
4. 用 `bx lr` 跳转

立即数

1. 整型
   - 通用：分别将该立即数的低16位和高16位用 `mov` 和 `movt` 赋值即可，其中表示方法的获取按照week12.pptx介绍的trick

     ```c
     union uf {
       int i;
       float f;
       unsigned int u;
     };
     ```

   - 对于Operand 2，先判断是否是 `<imm8m>`，如果是就直接编码到assem中，否则应用上面的通用流程
2. 负数 (整型)：可以利用 `mvn` 指令取反得到
   - 负数的反码应该是其补码减1
   - 若 `x` 为负数，先判断 `-x-1` 是否在 `<imm8m>` 的范围内，若是则可以利用 `mvn temp, -x-1` 指令得到立即数
   - 否则直接应用通用流程
3. 浮点型：直接应用上面的通用流程

opexp

对于这类指令，如果两个操作数都是常数，直接计算出结果，并将其Move到目标temp中即可。

epilogue

暂时没有需要做的。

测试结果

以 `example09.9.arm` 为例：

```
    .text
    .align 1
    .global main
main:
    push {fp}
    mov fp, sp
    push {r4, r5, r6, r7, r8, r9, r10, lr}
C1:
    mov r133, #1
    mov r134, #2
    cmp r133, r134
    bgt L0
L1:
    mov r136, #0
```

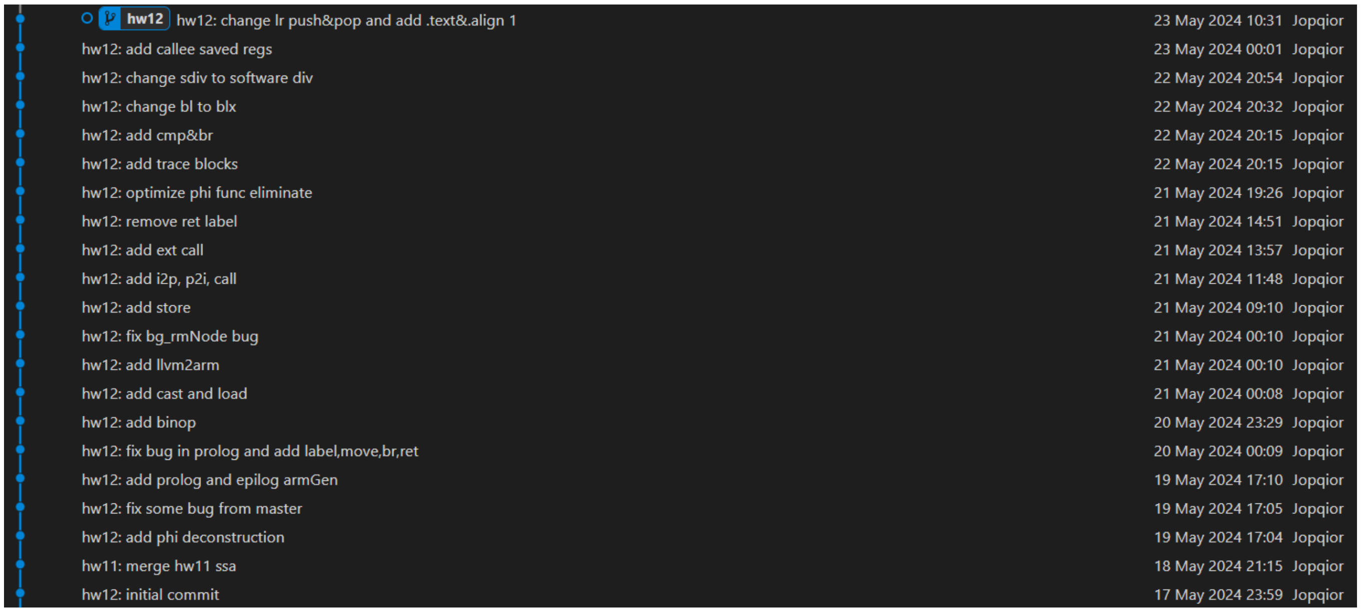
```
    mov r137, r136
    b L2
L2:
    vcvt.f32.s32 r138, r137
    mov r0, #24
    blx malloc
    mov r139, r0
    mov r140, r139
    mov r141, r140
    mov r162, #0
    movt r162, #16256
    vmov.f32 r142, r162
    vstr.f32 r142, [r141]
    mov r143, #16
    add r144, r140, r143
    mov r145, r144
    mov r163, #0
    movt r163, #16256
    vmov.f32 r146, r163
    vstr.f32 r146, [r145]
    mov r147, #8
    add r148, r140, r147
    mov r149, r148
    ldr r150, = c1$m
    str r150, [r149]
    mov r151, #8
    add r152, r140, r151
    mov r153, r152
    ldr r154, [r153]
    mov r155, r154
    mov r156, #3
    vcvt.f32.s32 r157, r156
    mov r0, r140
    mov r1, #1
    mov r2, #2
    vmov.f32 s0, r157
    blx r155
    vmov.f32 r158, s0
    vmul.f32 r159, r138, r158
    vcvt.s32.f32 r160, r159
    mov r0, r160
    sub sp, fp, #32
    pop {r4, r5, r6, r7, r8, r9, r10, lr}
    pop {fp}
    bx lr
L0:
    mov r161, #1
    mov r137, r161
    b L2

    .text
    .align 1
    .global c0$m
c0$m:
    push {fp}
    mov fp, sp
    push {r4, r5, r6, r7, r8, r9, r10, lr}
    mov r99, r0
    mov r100, r1
    mov r101, r2
    vmov.f32 r102, s0
c6:
    vmov.f32 s0, r102
    sub sp, fp, #32
    pop {r4, r5, r6, r7, r8, r9, r10, lr}
    pop {fp}
    bx lr

    .text
    .align 1
    .global c1$m
c1$m:
    push {fp}
    mov fp, sp
    push {r4, r5, r6, r7, r8, r9, r10, lr}
    mov r99, r0
    mov r103, r1
    mov r104, r2
    vmov.f32 r105, s0
c11:
    mov r164, r99
    vldr.f32 r165, [r164]
    vadd.f32 r166, r165, r105
    vmov.f32 s0, r166
    sub sp, fp, #32
    pop {r4, r5, r6, r7, r9, r10, lr}
    pop {fp}
    bx lr
```

开发过程

git提交记录如下：



```
hw12  hw12: change lr push&pop and add .text&.align 1      23 May 2024 10:31  Jopqior
hw12: add callee saved regs                                23 May 2024 00:01  Jopqior
hw12: change sdiv to software div                          22 May 2024 20:54  Jopqior
hw12: change bl to blx                                     22 May 2024 20:32  Jopqior
hw12: add cmp&br                                           22 May 2024 20:15  Jopqior
hw12: add trace blocks                                     22 May 2024 20:15  Jopqior
hw12: optimize phi func eliminate                          21 May 2024 19:26  Jopqior
hw12: remove ret label                                     21 May 2024 14:51  Jopqior
hw12: add ext call                                         21 May 2024 13:57  Jopqior
hw12: add i2p, p2i, call                                   21 May 2024 11:48  Jopqior
hw12: add store                                            21 May 2024 09:10  Jopqior
hw12: fix bg_rmNode bug                                    21 May 2024 00:10  Jopqior
hw12: add llvm2arm                                         21 May 2024 00:10  Jopqior
hw12: add cast and load                                    21 May 2024 00:08  Jopqior
hw12: add binop                                            20 May 2024 23:29  Jopqior
hw12: fix bug in prolog and add label,move,br,ret          20 May 2024 00:09  Jopqior
hw12: add prolog and epilog armGen                         19 May 2024 17:10  Jopqior
hw12: fix some bug from master                             19 May 2024 17:05  Jopqior
hw12: add phi deconstruction                               19 May 2024 17:04  Jopqior
hw11: merge hw11 ssa                                       18 May 2024 21:15  Jopqior
hw12: initial commit                                       17 May 2024 23:59  Jopqior
```