

总览

前端

- ANTLR语法分析
- 转为自定义的AST (Decorated AST)
- 在DST上进行语义分析/类型检查
 - 数组初始化的展开
 - 类型转换节点

其他

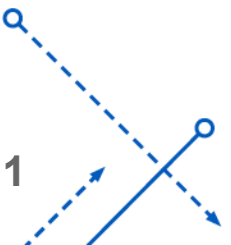
- 较完善的文档
- 转换过程尽量保留信息和注释便于Debug

中端

- 参照LLVM IR文本格式设计，可直接打印为LLVM IR
- 简洁的SSA构建
- GVN、DCE、控制流化简（迭代进行）
- 窥孔优化

后端

- 指令选择：单个IR遍历展开为多个汇编，窥孔优化消除冗余
- 进入后端前Split Critical Edge，指令选择时解构SSA。
- Local / 图着色寄存器分配
- 简洁的栈内存管理
- 尾递归优化
- BSS段的使用



生成代码注释和文档

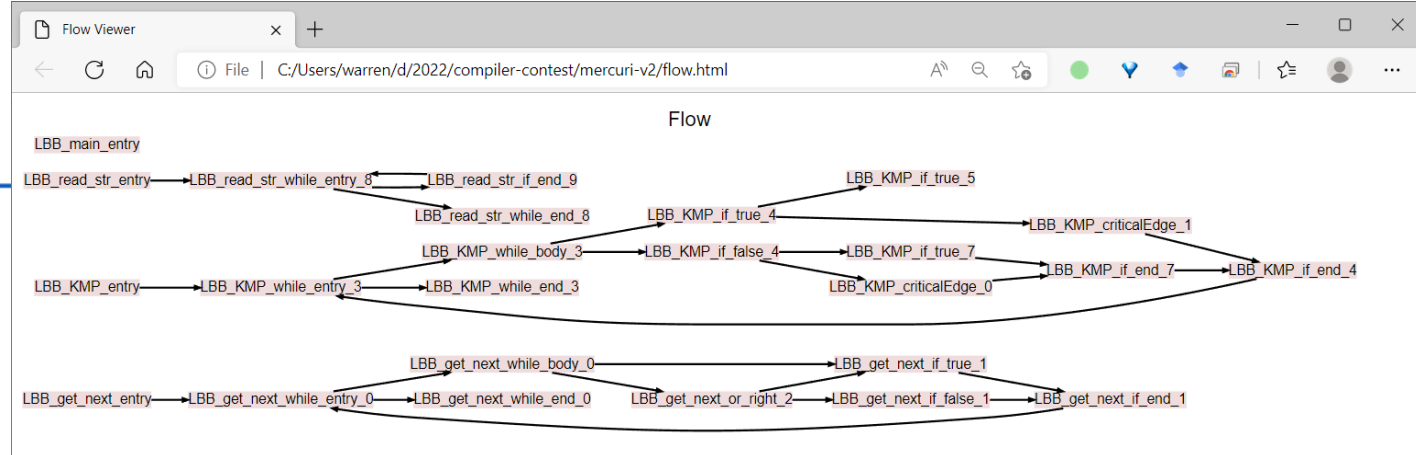
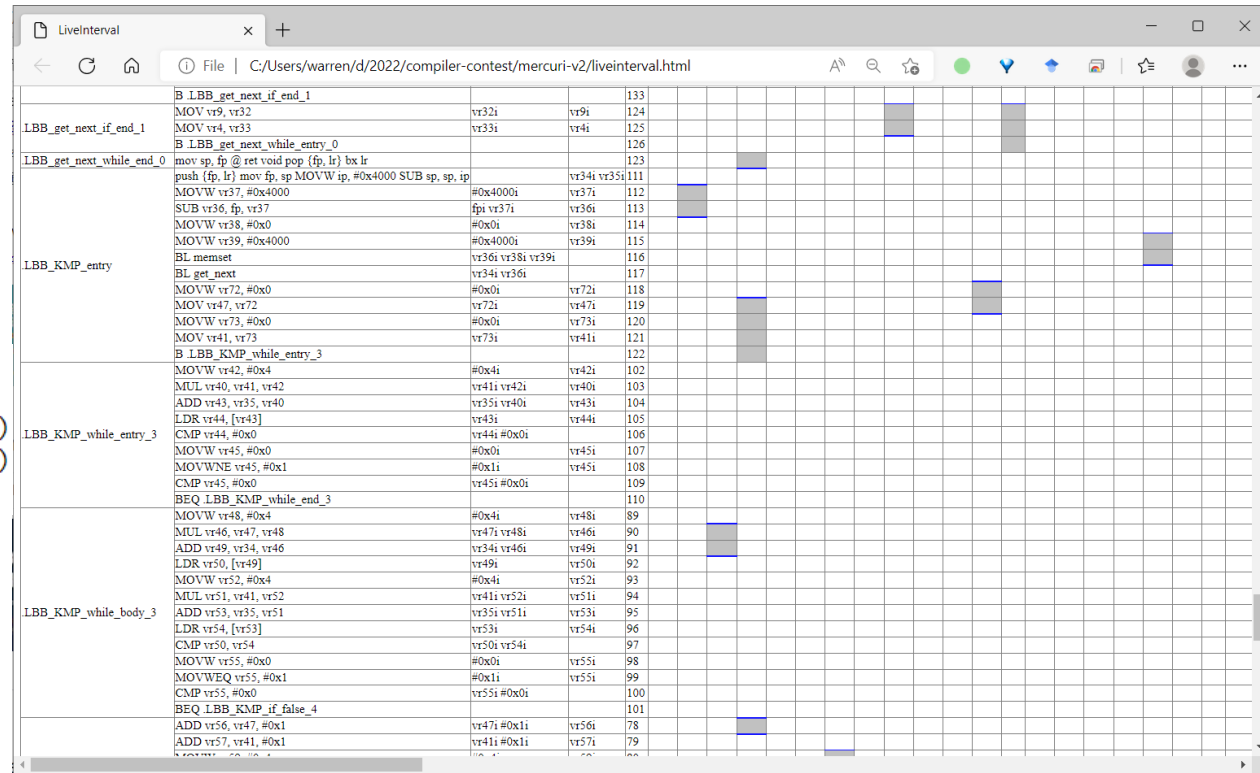
- 较为完善的文档：backend.md 15405字符，SSA-ir.md 5774字符
- 代码在转换过程中尽量保留信息，便于debug。
- 生成LLVM IR保留原变量名

```
%j_1_1 = phi i32 [ %2, %entry ], [ %j_1_5, %if_end_1 ]
%i_0_0 = phi i32 [ 0, %entry ], [ %i_0_4, %if_end_1 ]
%3 = getelementptr i32, i32* %str, i32%i_0_0
```

- 生成汇编时尽量保留源IR注释

```
.LBB_read_str_while_entry_8:
    MOVW    vr81, #0x4
    MUL vr79, vr80, vr81      @ %0 = getelementptr i32, i32* %buf, i32 %i_5_0 (i_5_0)
    ADD vr82, vr78, vr79      @ %0 = getelementptr i32, i32* %buf, i32 %i_5_0 (i_5_0)
    BL      getch             @ %1 = call i32 @getch()
```

- 可视化CFG、Liveness

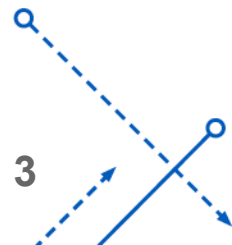
| Instruction Address | Instruction Text | Live Intervals |
|---------------------|--|----------------|
| 133 | B LBB_get_next_if_end_1 | |
| 124 | MOV vr9, vr32 | |
| 125 | MOV vr4, vr33 | |
| 126 | B LBB_get_next_while_entry_0 | |
| 123 | mov sp, fp @ ret void pop (fp, lr) bx lr | |
| 111 | push (fp, lr) mov fp, sp MOVW ip, #0x4000 SUB sp, sp, ip | |
| 112 | MOVW vr37, #0x4000 | |
| 113 | SUB vr36, fp, vr37 | |
| 114 | MOVW vr38, #0x0 | |
| 115 | MOVW vr39, #0x4000 | |
| 116 | BL memset | |
| 117 | BL get_next | |
| 118 | MOVW vr72, #0x0 | |
| 119 | MOV vr47, vr72 | |
| 120 | MOVW vr73, #0x0 | |
| 121 | MOV vr41, vr73 | |
| 122 | B LBB_KMP_while_entry_3 | |
| 102 | MOVW vr42, #0x4 | |
| 103 | MUL vr40, vr41, vr42 | |
| 104 | ADD vr43, vr35, vr40 | |
| 105 | LDR vr44, [vr43] | |
| 106 | CMP vr44, #0x0 | |
| 107 | MOVW vr45, #0x0 | |
| 108 | MOVWNE vr45, #0x1 | |
| 109 | CMP vr45, #0x0 | |
| 110 | BEQ LBB_KMP_while_end_3 | |
| 89 | MOVW vr48, #0x4 | |
| 90 | MUL vr46, vr47, vr48 | |
| 91 | ADD vr49, vr34, vr46 | |
| 92 | LDR vr50, [vr49] | |
| 93 | MOVW vr52, #0x4 | |
| 94 | MUL vr51, vr41, vr52 | |
| 95 | ADD vr53, vr35, vr51 | |
| 96 | LDR vr54, [vr53] | |
| 97 | CMP vr50, vr54 | |
| 98 | MOVW vr55, #0x0 | |
| 99 | MOVWNE vr55, #0x1 | |
| 100 | CMP vr55, #0x0 | |
| 101 | BEQ LBB_KMP_if_false_4 | |
| 78 | ADD vr56, vr47, #0x1 | |
| 79 | ADD vr57, vr41, #0x1 | |

SSA构建

- 参考《Simple and Efficient SSA Construction》
- 较为简洁，不需要计算繁杂的支配边界等信息。不到300行。

GVN、DCE、控制流 迭代化简

- GVN参考《Global Code Motion Global Value Numbering》（时间原因，GCM未实现）
 - 常量传播，指令本身的化简，查表减少重复计算
- DCE：标记有效指令，删除无用指令
- BranchMerge：化简GVN后存在的已知跳转结果的有条件跳转，删除基本块
- 迭代运行化简

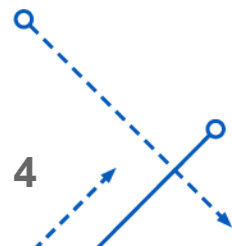
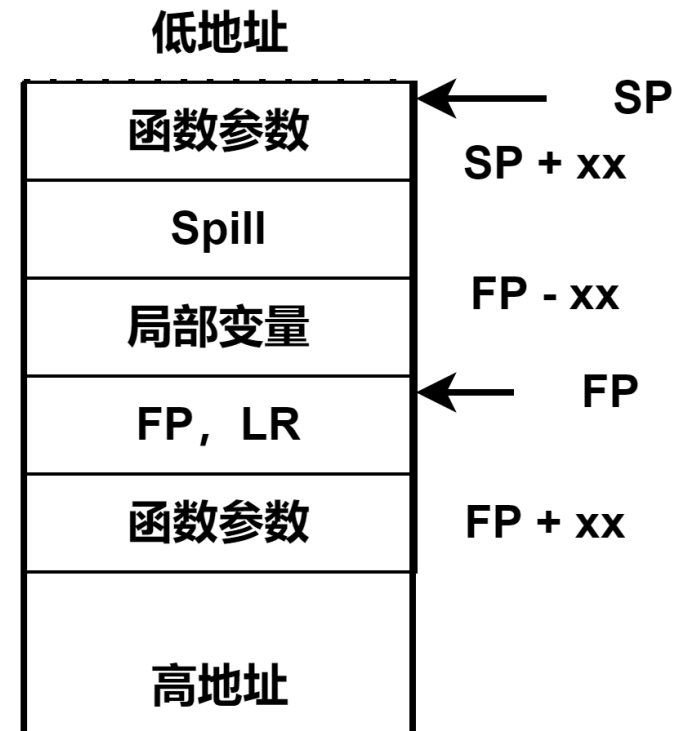


后端简洁的内存管理

- 使用FP和SP，FP向下寻址参数，向上寻址局部变量，SP向下寻址传参。
- 动态维护：局部变量空间，Spill空间和传参空间的大小。
- 保留Prologue和Ret作为抽象指令，在最后生成汇编时自动根据之前分配的空间减少SP。
- 无需回头修复之前指令的offset。

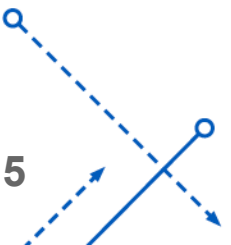
尾递归优化

- 解决medium2.sy用例爆栈问题。
- 使用中端Pass标记可以优化的函数调用：需要额外识别是否函数传入了自己栈上的数组



寄存器分配

- 旧：Local寄存器分配，spill：距离下一次使用最远的寄存器。
- LSRA寄存器分配（未成功实现）
- 新：简单Global寄存器分配：
 - 截止前一天学习了基于SSA的图着色寄存器分配，但时间来不及
 - 简单方案：额外保留两个寄存器解决spill和各种问题，构建冲突图后直接着色
- 提交截止前一个小时时完成，性能接近翻倍



如果还有时间...

- 实现基于SSA的寄存器分配：《SSA Destruction after Register Allocation》
- 完成函数内联
- 完成GCM (Global Code Motion)实现
- 完成LCSSA和循环相关优化

