

# 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

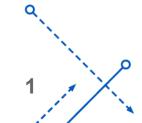
- 较完善的文档
- 转换过程尽量保留信息和注释便于Debug

#### 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





# Flow Flow Viewer Flow Flow Flow Flow Flow Flow Flow LBB\_main\_entry LBB\_read\_str\_while\_entry LBB\_read\_str\_while\_end\_8 LBB\_KMP\_if\_true\_5 LBB\_KMP\_while\_body\_3 LBB\_KMP\_if\_false\_4 LBB\_KMP\_if\_true\_7 LBB\_KMP\_if\_false\_4 LBB\_KMP\_if\_true\_7 LBB\_KMP\_while\_entry\_3 LBB\_KMP\_while\_end\_3 LBB\_KMP\_criticalEdge\_0 LBB\_KMP\_criticalEdge\_0 LBB\_KMP\_criticalEdge\_0 LBB\_KMP\_if\_false\_4 LBB\_KMP\_criticalEdge\_0 LBB\_KMP\_if\_false\_4 LBB\_KMP\_criticalEdge\_0 LBB\_KMP\_if\_end\_7 LBB\_KMP\_if

#### 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

	LiveInterval	× +															-		×
	← C A	i File   C:/Users/warren/d/2022/compiler	-contest/mercur	i-v2/livein	terval.h	ntml			A <sup>3</sup>	y (	2 9	ò	Y	1		5	₹=	9	
		B LBB_get_next_if_end_1			133														
1		MOV vr9, vr32	vr32i		124														
-    -	LBB_get_next_if_end_1	MOV vr4, vr33	vr33i		125														
		B .LBB_get_next_while_entry_0			126														
	LBB_get_next_while_end_0	mov sp, fp @ ret void pop {fp, lr} bx lr			123														
	LBB_KMP_entry	push {fp, lr} mov fp, sp MOVW ip, #0x4000 SUB sp, sp,		vr34i vr35i															
		MOVW vr37, #0x4000	#0x4000i		112														
		SUB vr36, fp, vr37	fpi vr37i		113														
		MOVW vr38, #0x0	#0x0i	vr38i	114														
		MOVW vr39, #0x4000	#0x4000i	vr39i	115														
		BL memset	vr36i vr38i vr39i		116														
		BL get_next	vr34i vr36i		117														
-11		MOVW vr72, #0x0	#0x0i	vr72i	118														
		MOV vr47, vr72	vr72i	vr47i	119														
		MOVW vr73, #0x0	#0x0i	vr73i	120														
		MOV vr41, vr73	vr73i		121														
		B LBB_KMP_while_entry_3			122														
		MOVW vr42, #0x4	#0x4i	vr42i	102														
		MUL vr40, vr41, vr42	vr41i vr42i	vr40i	103														
		ADD vr43, vr35, vr40	vr35i vr40i	vr43i	104														
۱I.		LDR vr44, [vr43]	vr43i	vr44i	105														
)∥	LBB KMP while entry 3	CMP vr44, #0x0	vr44i #0x0i		106														
)	/-	MOVW vr45, #0x0	#0x0i	vr45i	107														
		MOVWNE vr45, #0x1	#0x1i	vr45i	108														
		CMP vr45, #0x0	vr45i #0x0i		109														
		BEQ.LBB KMP while end 3			110														
- 11		MOVW vr48, #0x4	#0x4i	vr48i	89														
		MUL vr46, vr47, vr48	vr47i vr48i	vr46i	90													$\overline{}$	
		ADD vr49, vr34, vr46	vr34i vr46i	vr49i	91														
- 11		LDR vr50, [vr49]	vr49i	vr50i	92														
	LBB_KMP_while_body_3	MOVW vr52, #0x4	#0x4i		93													$\overline{}$	
-11		MUL vr51, vr41, vr52	vr41i vr52i	vr51i	94														
11.		ADD vr53, vr35, vr51	vr35i vr51i		95														
		LDR vr54, [vr53]	vr53i		96													$\Box$	
		CMP vr50, vr54	vr50i vr54i		97													$\Box$	
		MOVW vr55, #0x0	#0x0i		98					+	+	_		+					
		MOVWEQ vr55, #0x1	#0x1i		99					+	+							$\vdash$	_
		CMP vr55, #0x0	vr55i #0x0i		100														
		BEQ.LBB KMP if false 4			101														
Th		ADD vr56, vr47, #0x1	vr47i #0x1i		78						_								+-
Ш		ADD vr57, vr41, #0x1	vr41i #0x1i		79		_				_	_						-	_
Ш		MODERATE CO. IIO. 4	VITII WOXII		00	-	_	_			_	+	+	-	-			_	
4																			<b>&gt;</b>

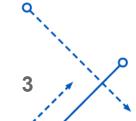


## SSA构建

- 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



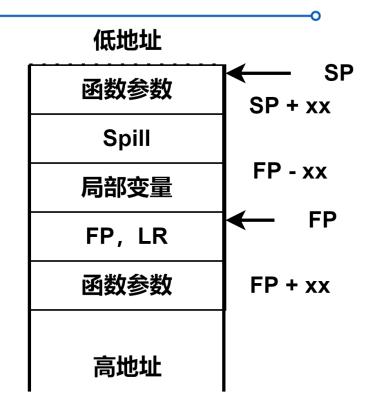


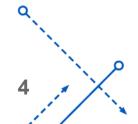
## 后端简洁的内存管理

- 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

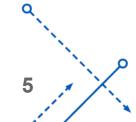






## 寄存器分配

- 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

