2024 编译系统设计赛答辩

答辩人：汤翔晟 侯华玮 简泽鑫 杨俯众

国防科技大学计算机学院

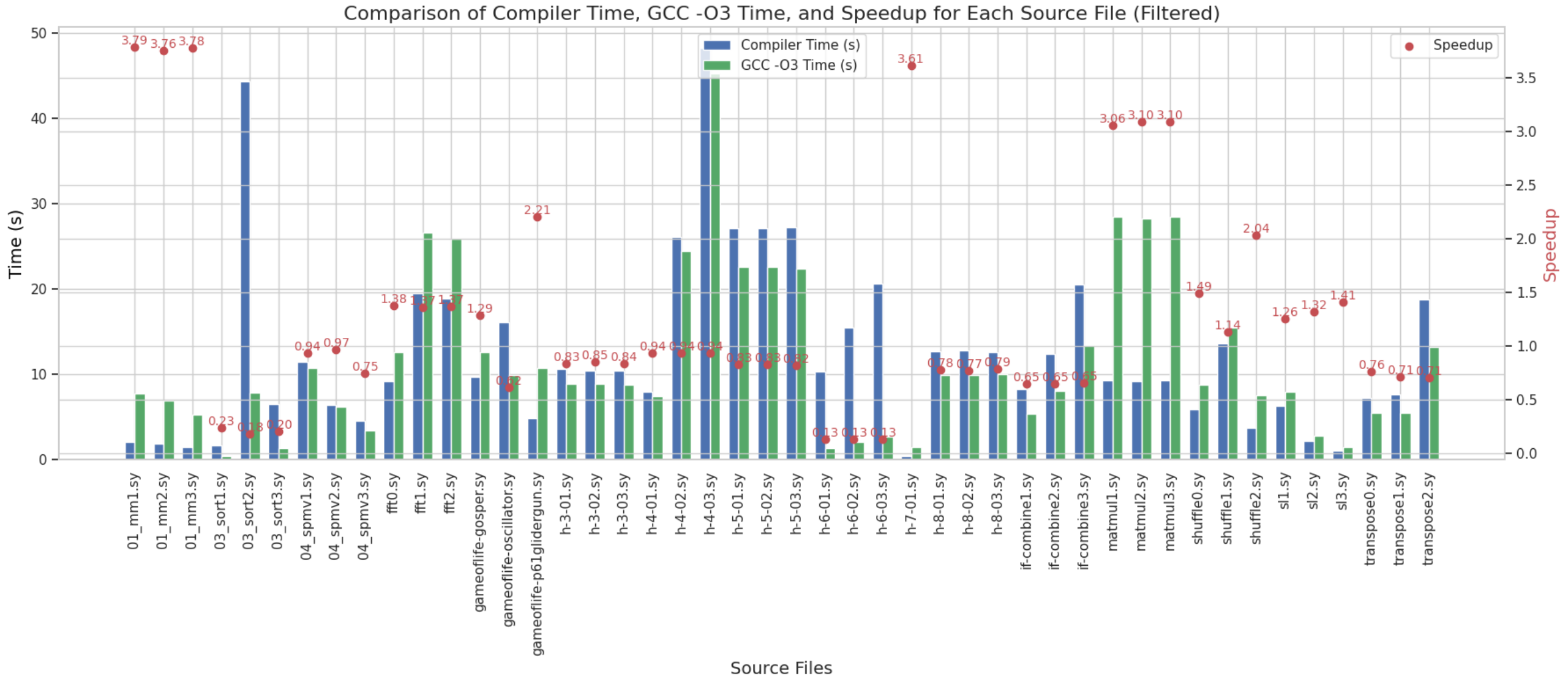
时间：2024 年 8 月 21 日

目录 CONTENT

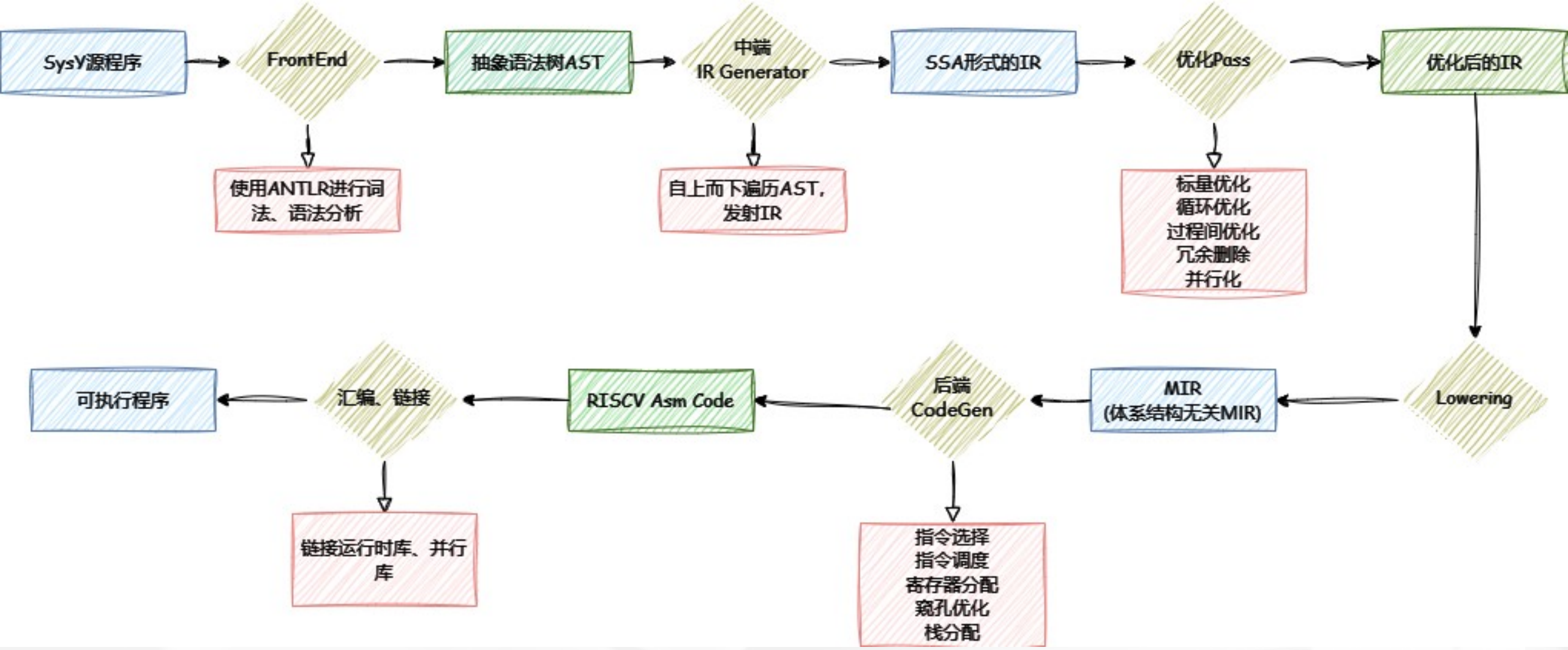
团队分工

- 汤翔晟
  组长，负责中端的功能实现和优化
- 侯华玮
  负责指令选择、指令调度、并行化、测试
- 简泽鑫
  负责数组、块调度、寄存器分配
- 杨俯众
  负责中端的功能实现和优化

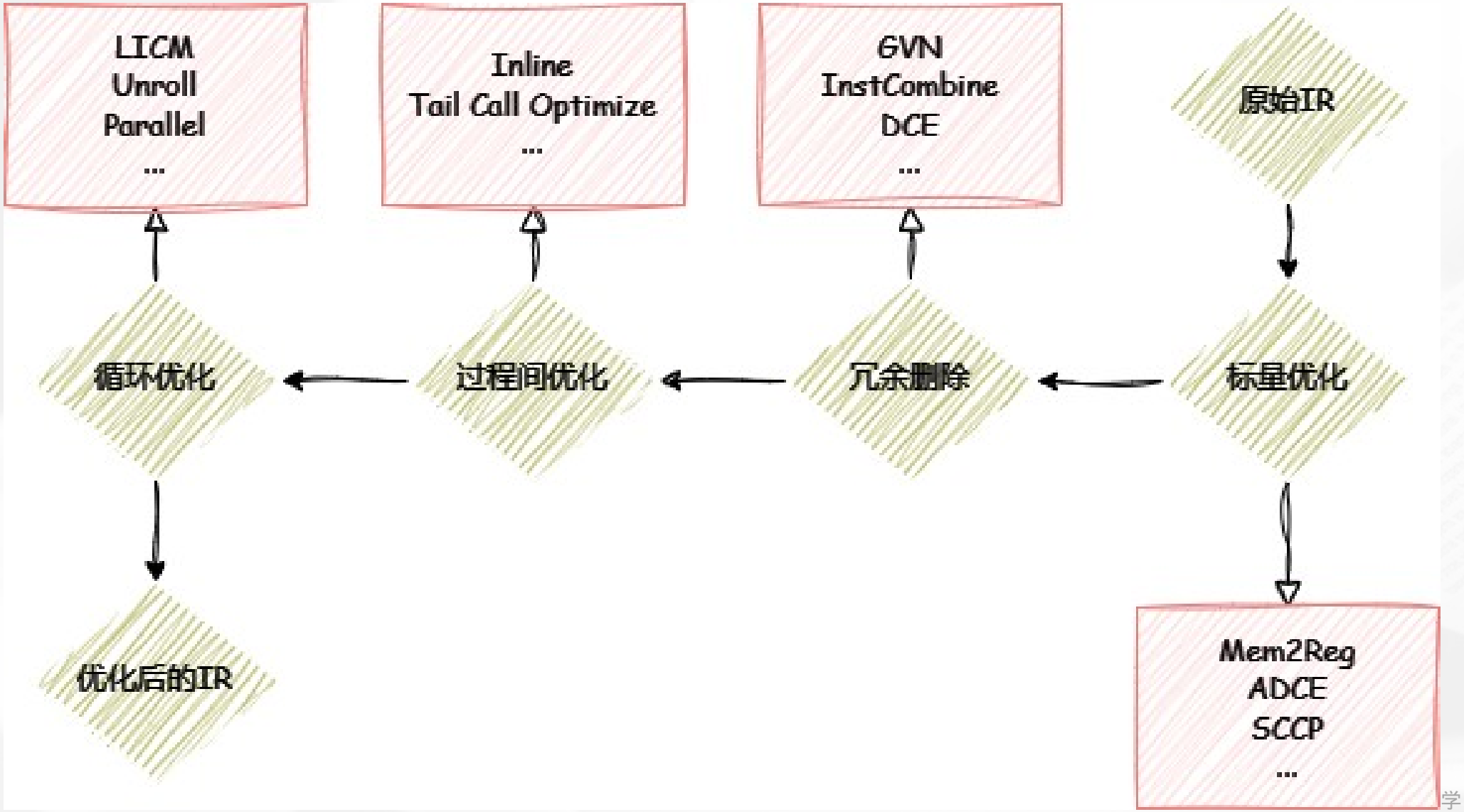
优化效果总览


Comparison of Compiler Time, GCC -O3 Time, and Speedup for Each Source File (Filtered)
Compiler Time (s)
GCC -O3 Time (s)
Speedup
Time (s)
Speedup
Source Files
01_mm1.sy
01_mm2.sy
01_mm3.sy
03_sort1.sy
03_sort2.sy
03_sort3.sy
04_spmv1.sy
04_spmv2.sy
04_spmv3.sy
fft0.sy
fft1.sy
fft2.sy
gameoflife-gosper.sy
gameoflife-oscillator.sy
gameoflife-p61glidergun.sy
h-3-01.sy
h-3-02.sy
h-3-03.sy
h-4-01.sy
h-4-02.sy
h-4-03.sy
h-5-01.sy
h-5-02.sy
h-5-03.sy
h-6-01.sy
h-6-02.sy
h-6-03.sy
h-7-01.sy
h-8-01.sy
h-8-02.sy
h-8-03.sy
if-combine1.sy
if-combine2.sy
if-combine3.sy
matmul1.sy
matmul2.sy
matmul3.sy
shuffle0.sy
shuffle1.sy
shuffle2.sy
sl1.sy
sl2.sy
sl3.sy
transpose0.sy
transpose1.sy
transpose2.sy

编译流程

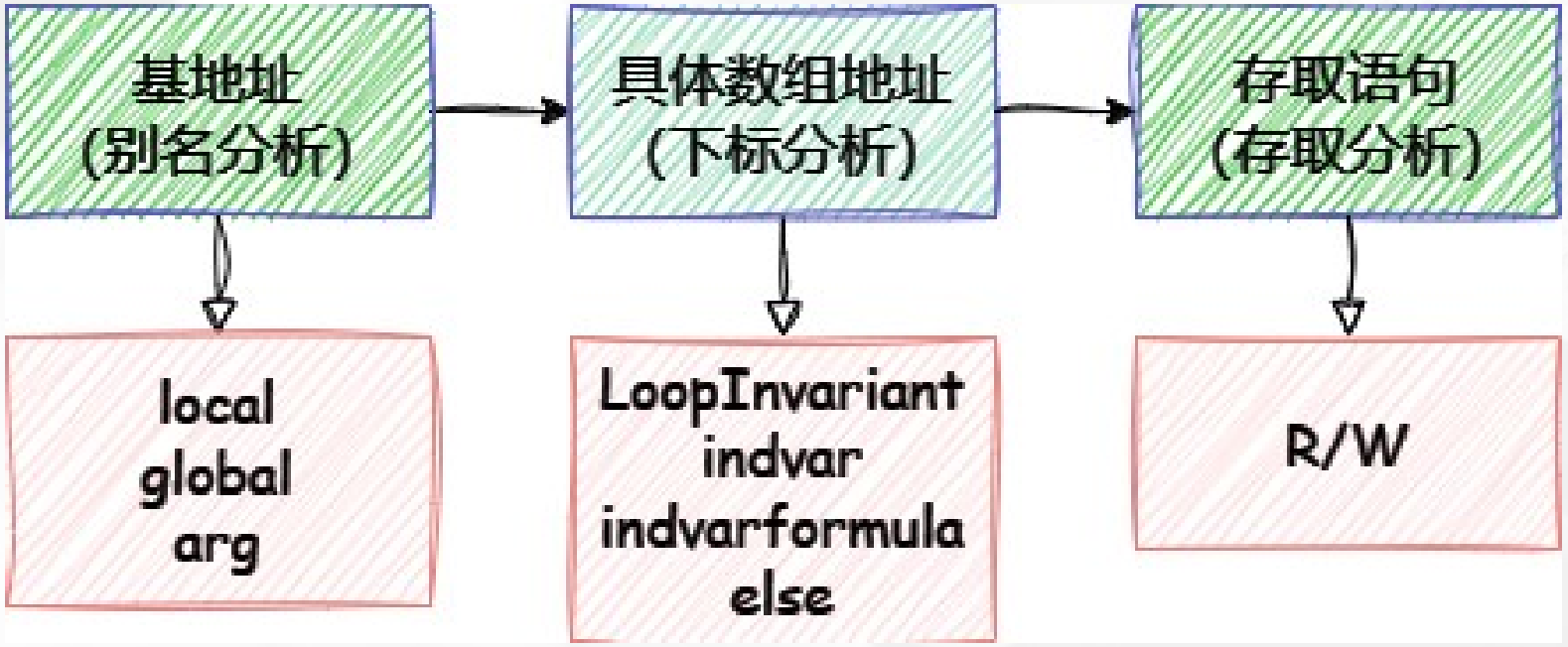

SysY源程序
FrontEnd
抽象语法树AST
中端
IR Generator
SSA形式的IR
优化Pass
优化后的IR
使用ANTLR进行词法、语法分析
自上而下遍历AST，发射IR
标量优化
循环优化
过程间优化
冗余删除
并行化
Lowering
MIR
(体系结构无关MIR)
后端
CodeGen
RISCV Asm Code
汇编、链接
可执行程序
指令选择
指令调度
寄存器分配
窥孔优化
栈分配
链接运行时库、并行库

优化总览


LICM
Unroll
Parallel
...
Inline
Tail Call Optimize
...
GVN
InstCombine
DCE
...
原始IR
循环优化
过程间优化
冗余删除
标量优化
优化后的IR
Mem2Reg
ADCE
SCCP
...

关键技术：依赖关系分析

- 分析一个循环能否并行——有无跨迭代的依赖
  - 依赖的关键问题是地址，
  - 分别从基址、具体数组地址、存取语句分析

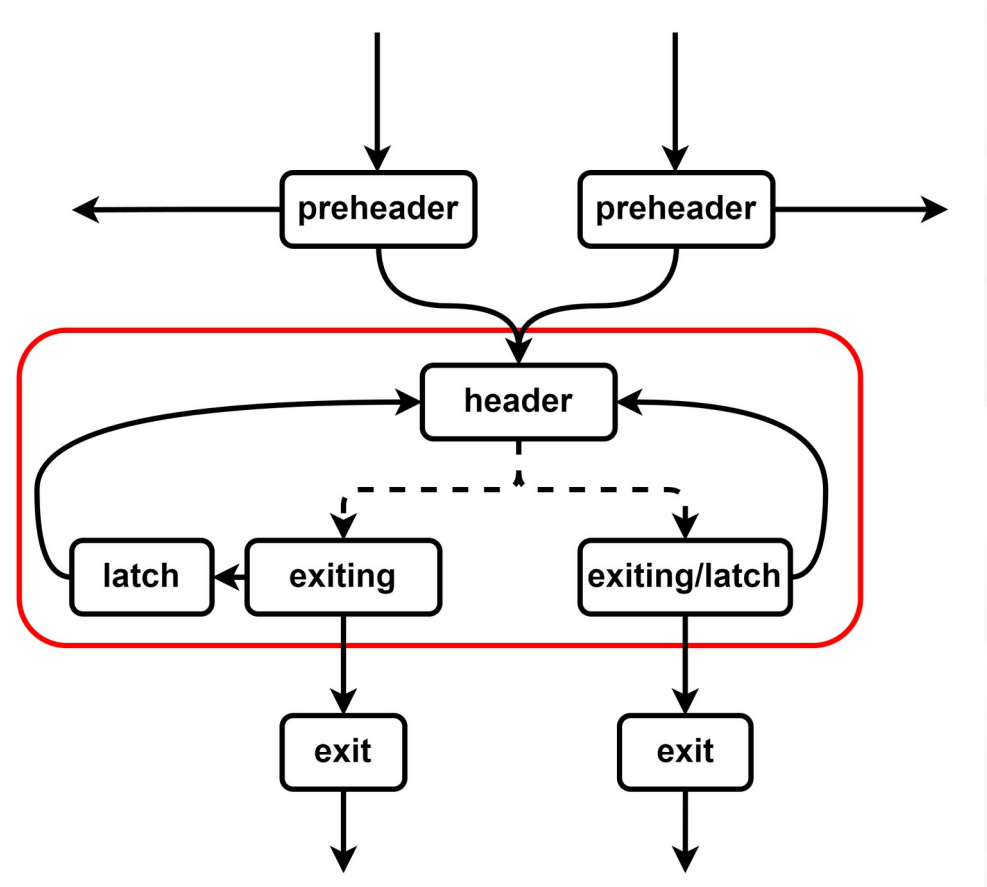


关键技术： 并行化

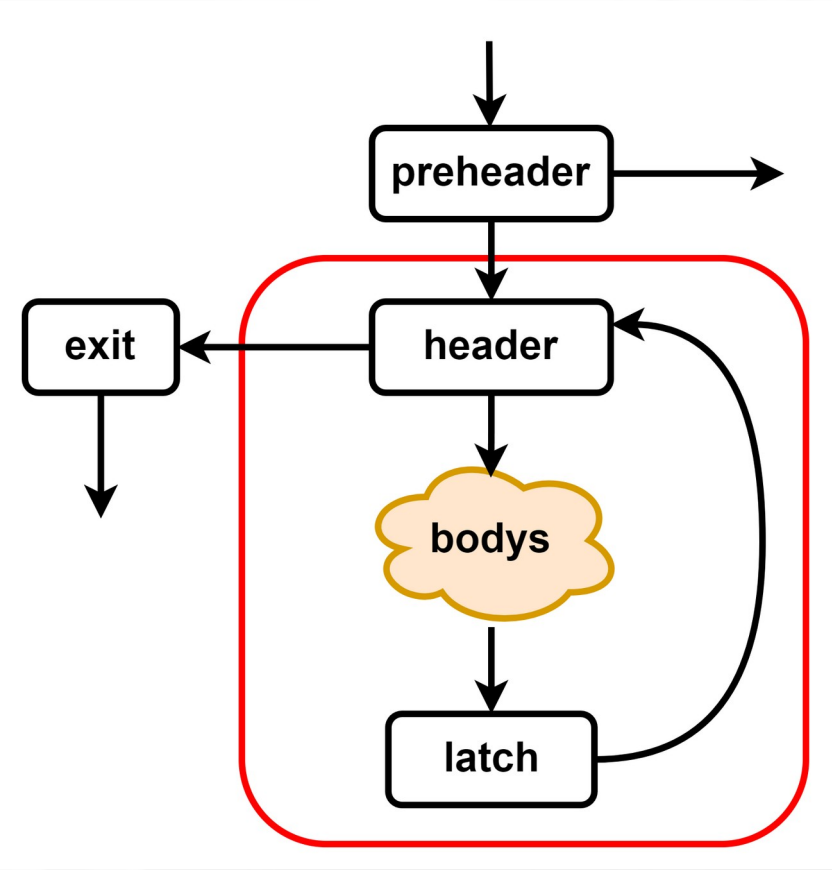
- **LoopSimplify -> LoopBodyExtract -> ParallelBodyExtract -> LoopParallel**
- **LoopSimplify**：循环标准化（ 1 preheader， 1 single backedge, dedicated exits ）
- **LoopBodyExtract** ：将循环的 body 提取成一个函数 ：
  - void loop_body(i, othersargs...)
  - giv loop_body(i, giv, otherargs...)
- **ParallelBodyExtract** ：将整个循环封装成一个函数：
  - void parallel_body(beg, end) { for (int i = beg, i < end; i++) loop_body(i, otherargs) }
- **LoopParallel** ：
  - 插入线程创建和回收的系统调用（clone, waittid），计算确定每个线程的 beg, end
- **LoopInterChange** ：循环交换， 访存局部性，并行更外层循环

关键技术： 并行化

- **LoopSimplify**：循环标准化（ 1 preheader，1 single backedge, dedicated exits ）



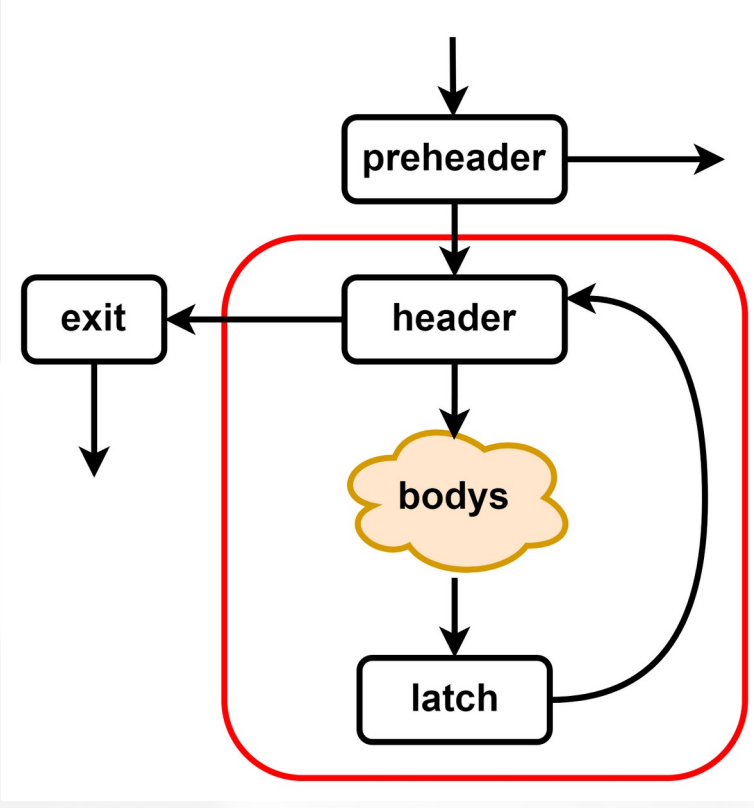
Common Loop



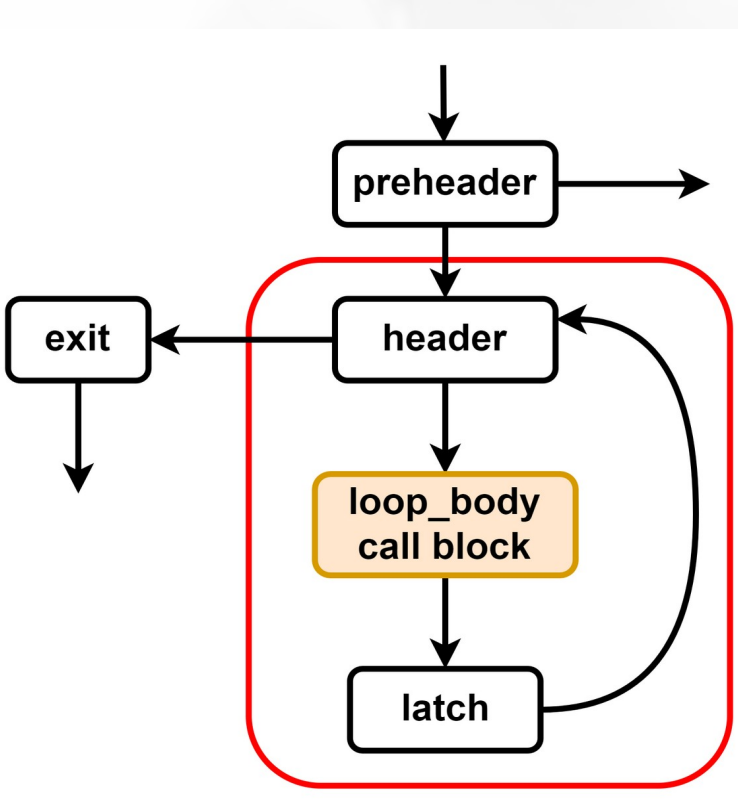
Canonical Forms Loop

关键技术： 并行化

- **LoopBodyExtract** ：将循环的 body 提取成一个函数 ：
  - void loop_body(i, othersargs...)
  - giv loop_body(i, giv, otherargs...)



Canonical Forms Loop

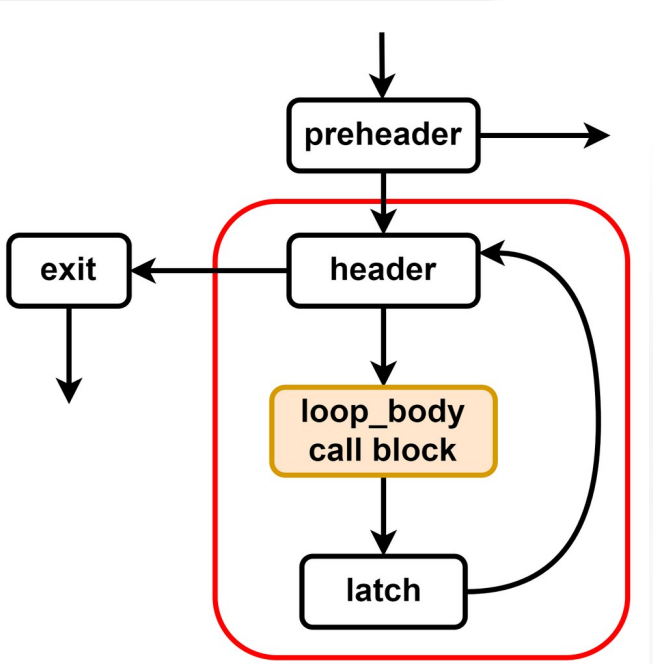


After LoopBodyExtract

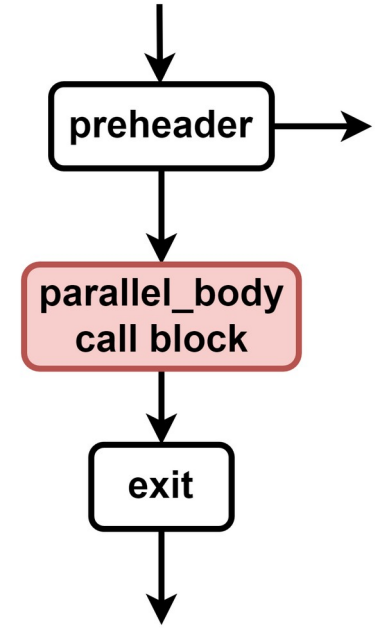
```
define i32 @sysyc_loop_body0(i32 %0, i32 %1, i32 %2) {
bb0:
    br label %bb1
bb1:
    %3 = add i32 %0, 1
    %4 = getelementptr [1024 x [1024 x i32]], [1024 x [
    br label %bb2 ; br while7_judge
bb2: ; while7_judge
    %5 = phi i32 [ %10, %bb4 ],[ %1, %bb1 ]
    %6 = phi i32 [ %11, %bb4 ],[ 0, %bb1 ]
    %7 = icmp slt i32 %6, %2
    br i1 %7, label %bb4, label %bb3
bb3:
    ret i32 %5
bb4: ; while7_loop
    %8 = getelementptr [1024 x i32], [1024 x i32]* %4,
    %9 = load i32, i32* %8
    %10 = add i32 %5, %9
    %11 = add i32 %6, 1
    br label %bb2 ; br while7_judge
}
```

关键技术： 并行化

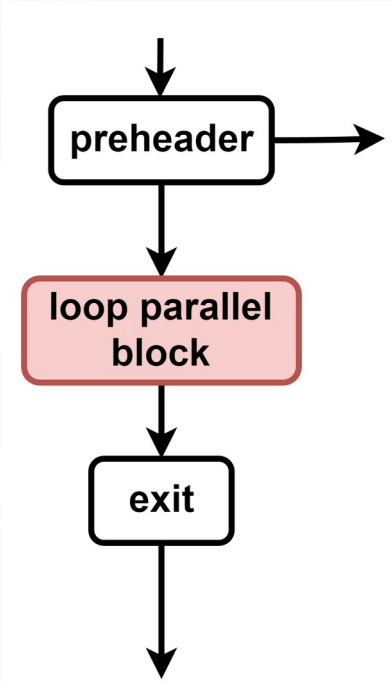
- **ParallelBodyExtract** ：将整个循环封装成一个函数：
  - void parallel_body(beg, end) { for (int i = beg, i < end; i++) loop_body(i, otherargs) }
- **LoopParallel** ：
  插入线程创建和回收的系统调用（clone, waittid），计算确定每个线程的 beg, end
  使用原子指令以避免数据竞争



After LoopBodyExtract



After ParallelBodyExtract

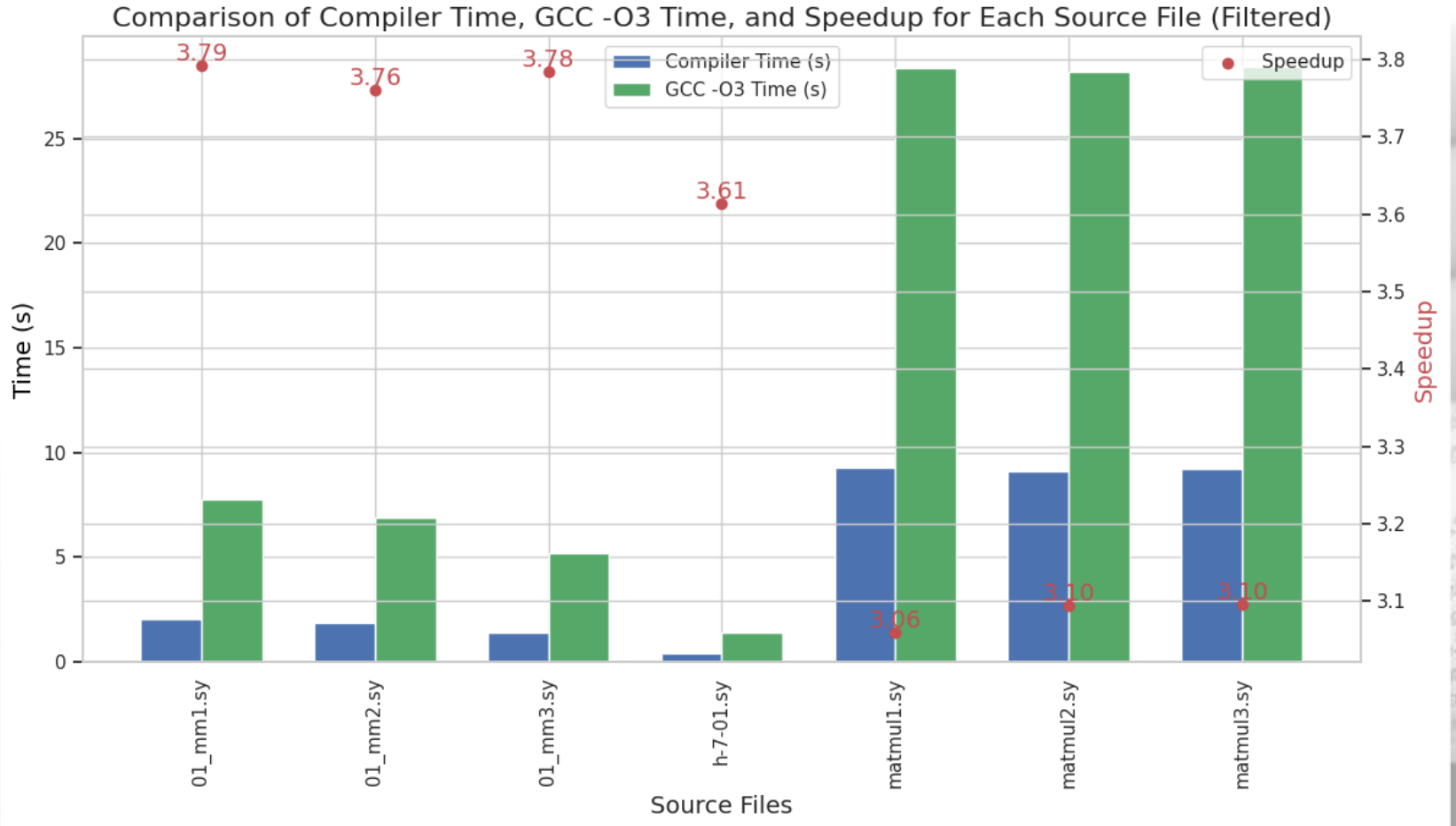


After LoopParallel

```
define void @sysyc_parallel_body1(i32 %0, i32 %1) {
bb0:
    br label %bb1
bb1:
    %2 = ptrtoint [3 x i32]* @parallel_body_payload1 to
    %3 = add i64 %2, 0
    %4 = inttoptr i64 %3 to [1024 x i32]**
    %5 = load [1024 x i32]*, [1024 x i32]** %4
    %6 = add i64 %2, 8
    %7 = inttoptr i64 %6 to i32*
    %8 = load i32, i32* %7
    br label %bb2
bb2:
    %9 = phi i32 [ %11, %bb4 ],[ %0, %bb1 ]
    %10 = icmp slt i32 %9, %1
    br i1 %10, label %bb3, label %bb5
bb3:
    call void @sysyc_loop_body1(i32 %9, [1024 x i32]* %
    br label %bb4
bb4:
    %11 = add i32 %9, 1
    br label %bb2
bb5:
    ret void
}
```

关键技术： 并行化

```
while (k < n){
    i = 0;
    while (i < n){
        if (A[i][k] == 0){
            i = i + 1;
            continue;
        }
        j = 0;
        while (j < n){
            C[i][j] = C[i][j] + A[i][k] * B[k][j];
            j = j + 1;
        }
        i = i + 1;
    }
    k = k + 1;
}
```



关键技术：基于模版的指令选择

- 通过“匹配 - 替换规则”与 模板（ python jinja2 ）自动生成指令选择 c++ 代码
- “ 数据 - 控制”分离，只需要编写对应的规则

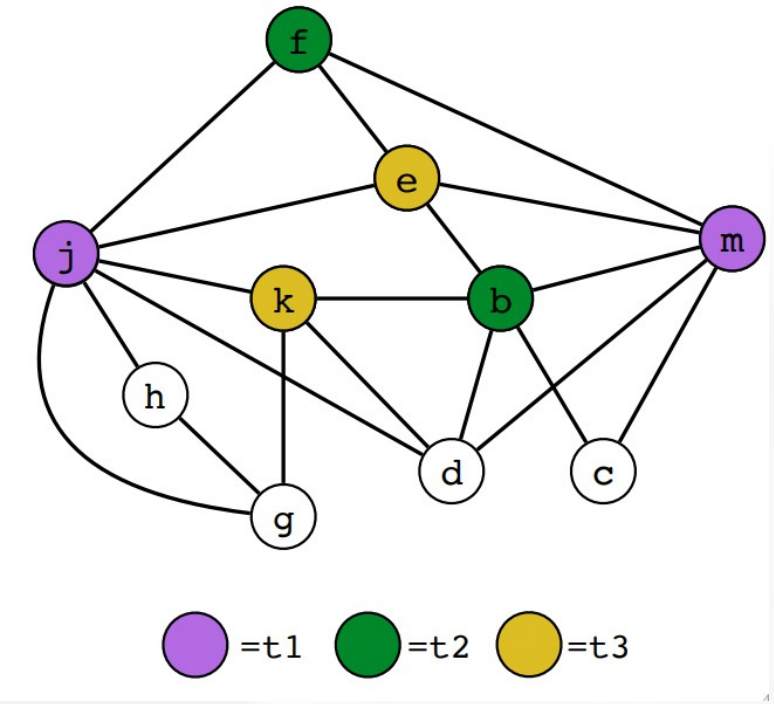
```
nstSelInfo: {
  Instances: [
    {
      # InstAdd $dst[I32], $src1, $src2[imm12] -> ADDIW $dst, $src1, $src2
      pattern:
        {
          name: InstAdd,
          dst: $dst,
          src1: $src1,
          src2: $src2,
          predicate: isOperandI32($dst) && isOperandIReg($src1) && isOperandI
        },
      replace: { name: ADDIW, rd: $dst, rs1: $src1, imm: $src2 },
    },
```

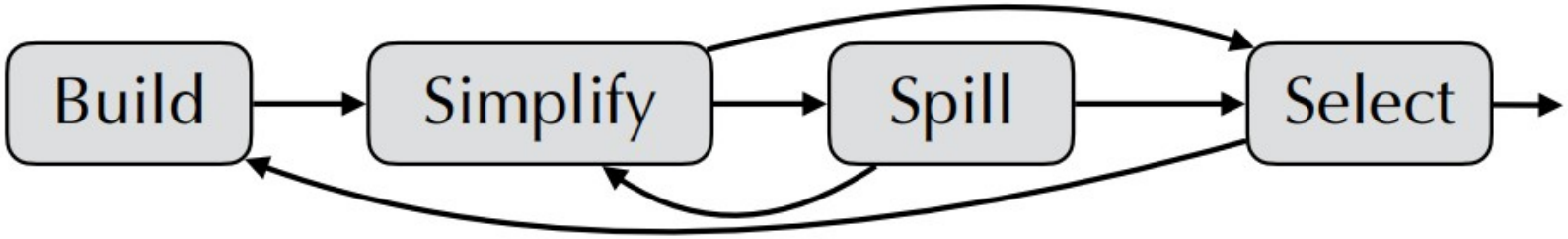
```
static bool matchAndSelectPattern21(MIRInst* inst1, ISelContext& ctx) {
  uint32_t rootOpcode = InstAdd;
  /** Match Inst **/
  /* match inst InstAdd */
  MIROperand op1;
  MIROperand op2;
  MIROperand op3;
  if (not matchInstAdd(inst1, op1, op2, op3)) {
    return false;
  }

  /* match predicate for operands  */
  if (not(isOperandI32(op1) && isOperandIReg(op2) && isOperandIReg(op3))) {
    return false;
  }

  /** Select Inst **/
  auto op5 = (op1);
  auto op6 = (op2);
  auto op7 = (op3);
  /* select inst ADDW */
  auto inst2 = ctx.insertMIRInst(ADDW, {op5, op6, op7});

  /* Replace Operand */
  ctx.replace_operand(ctx.getInstDefOperand(inst1), ctx.getInstDefOperand(inst2));
  ctx.remove_inst(inst1);
  return true;
}
```

关键技术：图着色寄存器分配

- 图着色寄存器分配算法
  - 基本块的排序影响了活跃变量的计算，从而影响其分配的好坏。
  - 我们使用了如下排序方式：

    以支配树顺序为基础，每个基本块都在所有支配它的基本块后





关键技术：基本块内指令调度 - 表调度

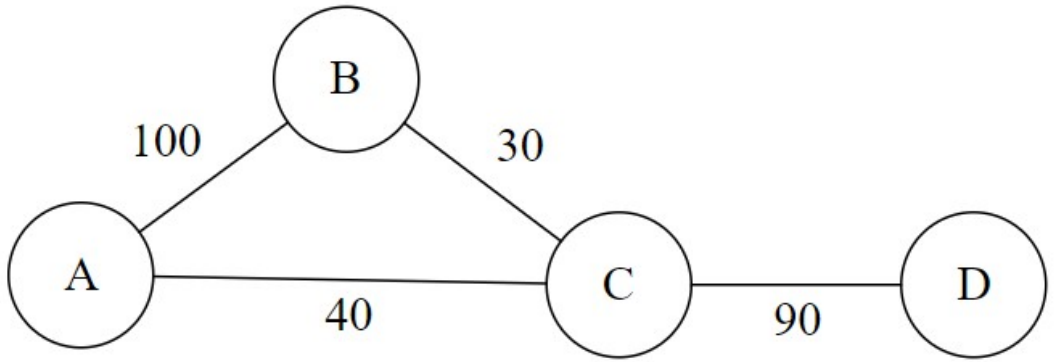
- VisionFive2 Processer: Sifive-U74MC Core
  - SiFive U74-MC Core Complex Manual
  - 流水线信息， 指令延迟信息， 指令资源使用信息
- 构建依赖图
- 每一个周期调度可被调度的指令：
  - 依赖全部满足
  - 资源满足

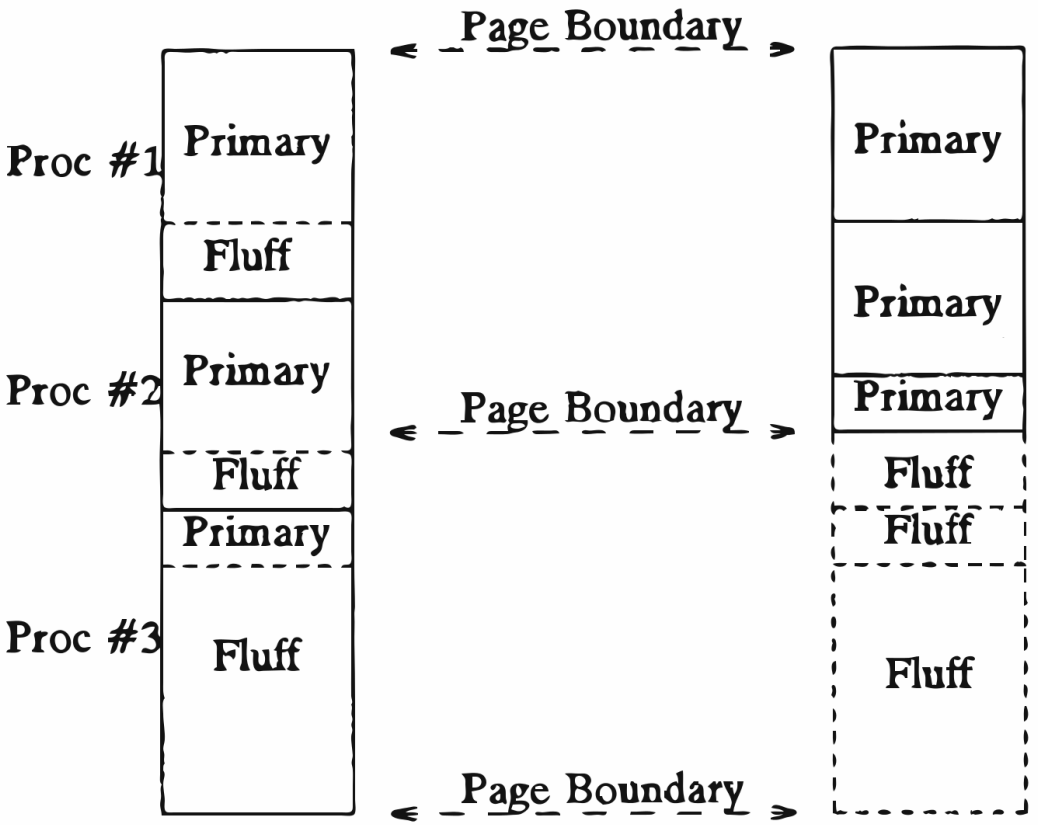
```
lla a2, sysyc_parallel_body4
jal parallelFor
lla a1, parallel_body_payload3
sd s2, 0(a1)
addi a2, a1, 8
sw s0, 0(a2)
addi a0, a1, 12
sd s1, 0(a0)
addi a2, a1, 20
sd s3, 0(a2)
mv a0, zero
mv a1, s0
```

```
972  lla a2, sysyc_parallel_body4
973  mv a1, s0
974  jal parallelFor
975  lla a1, parallel_body_payload3
976  mv a0, zero
977  addi a4, a1, 20
978  addi a3, a1, 12
979  addi a2, a1, 8
980  sd s2, 0(a1)
981  sw s0, 0(a2)
982  lla a2, sysyc_parallel_body3
983  sd s1, 0(a3)
```

关键技术：块调度

- 基于 Pettis-Hansen 的启发式方法
  - 静态估计分支指令指向 true/false 块跳转的频率
  - 在控制流图上估算每条边的跳转频率
  - 每次选取流图中的频率最长链，转为线性顺序
    - 尽可能最大程度减少跳转





教训与思考

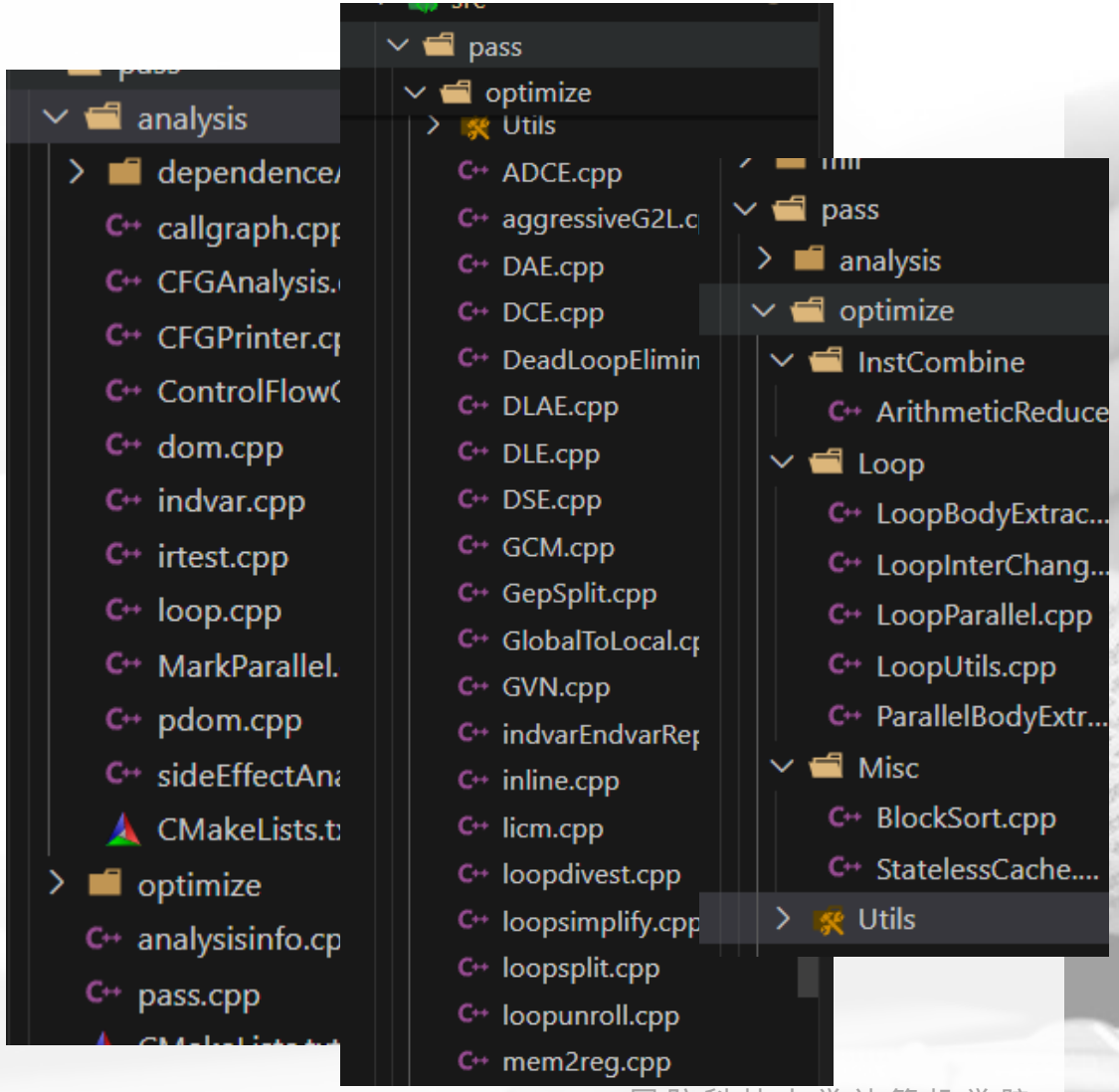
- IR 设计：什么是好的中间表示？
  - LLVM IR: User – Use – Value
  - 应尽量减少变换过程中的信息损失，方便分析和优化
  - GetElementPtr:
    - 一条 GEP 指令应保留 base + indices，从而方便进行基址分析与别名分析
    - 但我们在 emitIR 时预先将 GEP Split 了，导致每一条 GEP 都只有一个 index
    - 后续分析和变换时又要进行收集
  - 具有交换律的运算：
    - 我们将运算构建成二叉树，不便于进行具有交换律的匹配和变换
    - 是否可以使用多元运算指令？Try Later
- 测试与验证：凭什么认为你的程序是对的？
  - 熟练掌握 c++ ，现代 c++ 特性，遵守一定的编程规范
  - 有意识增强程序的鲁棒性，对代码质量有高的要求
  - 设计模式与设计经验： visitor ， builder ， flyweight, register ...
  - 测试：单元测试，集成测试，系统测试，模糊测试，回归测试
    - IR Verify, IR Check
    - 使用 Csmith 等模糊测试工具， challenge your compiler

```
// #include <algorith
namespace ir {
class Use;
class User;
class Value;

class ConstantValue;
class Instruction;
class BasicBlock;
class Argument;

class Function;
class Module;
```

```
namespace ir {

class AllocaInst;
class LoadInst;
class StoreInst;
class GetElementPtrInst;

class ReturnInst;
class BranchInst;

class UnaryInst;
class BinaryInst;

class ICmpInst;
class FCmpInst;
class CallInst;

class PhiInst;
class AtomicrmwInst;

class IndVar;
```

收获：系统能力的真实提升

- 深入理解计算机系统：
  - 程序性能优化：访存 or 计算？
  - RISCV ISA
  - Micro Architecture: 流水线，缓存
- 代码行数 3W+ lines
- 熟练使用各种开发工具，解决各种开发问题
- 程序设计能力，软件设计能力 ++
- 团队协作与分工 ++



请各位老师批评指正！