清華大學
Tsinghua University

小林家的编译器

于剑　蔡承泽　刘一苁　杨耀良

设计架构

- 使用 C++ 开发，使用 ANTLR 解析
- 前中后端解耦合
  - Frontend
  - Optimizer
  - Backend
- 批量评测系统

中端优化

- Mem2Reg
- Array SSA
- Function Inline
- Loop Unroll / Loop Parallel
- Array to Struct
- GVN
- Dead Code Elimination
- Instruction Scheduling

MEM2REG
ARRAY SSA

- Mem2reg
  - 仅在 ***main*** 函数使用的全局 ***int*** 变量变为局部变量
  - 局部 ***int*** 变量变为寄存器

- Array SSA
  - 对数组进行别名分析，将 Load/Store 和 IO 全部转为 SSA 形式
  - 将每次初始化值相同的局部数组变为全局数组
  - Load-Store 的化简

LOOP UNROLL

- 循环展开
  - `for (i=L; i<R; ++i) F(i);`
    - `for (i=L; i<R-4; i+=4) F(i), F(i+1), F(i+2), F(i+3);`
    - `for (; i<R; ++i) F(i);`
  - `do f(); while(g());`
    - `f(); while(g()) f();`
  - `for (i=0; i<5; ++i) F(i);`
    - `F(0); F(1); F(2); F(3); F(4);`

LOOP PARALLEL

- 循环自动并行化条件
  - 读写内存无冲突
  - 修改的寄存器都是归纳变量与累加变量
  - 循环条件简单
- 并行化后的行为
  - `#pragma omp parallel for reduction(+:s) schedule(static)`
  - `for (i=L; i<R; ++i) s+=F(i);`

ARRAY TO STRUCT

- 估计每个基本块的访问频率
- 基于马尔科夫链
- 假设分支跳出循环的概率较低
- 查找访问下标经常相同的数组
  - `int a[1000], b[1000];`
  - `struct {int a, b} ab[1000];`

GVN
DCE

- GVN
  - 常量传播、拷贝传播
  - 公共子表达式合并
  - 代数恒等式化简
- DCE
  - 删除无副作用的循环 `for (i=0; i<1e8; ++i);`
  - 消除条件为常数的分支 `if (true) F(); else G(); F();`
  - 删除结果未使用的计算 `a=b*c; a=0; a=0;`
  - 删除不可达的函数或基本块
  - 基本块合并 `{xxx;} {yyy;} {xxx; yyy;}`

INSTRUCTION SCHEDULING

- 调整基本块内的指令顺序，减少 *live-range* 较大的寄存器
- 调整基本块的顺序，减少不必要的长距离跳转

- 跨越基本块移动指令 GCM
  - `for (i=0; i<n; ++i) a[0][i]=a[1][i];`
    - `int *b=a[0], *c=a[1];`
    - `for (i=0; i<n; ++i) b[i]=c[i];`
  - `a=c*d; if (cond) b=a;`
    - `if (cond) {a=c*d; b=a;}`

后端优化

- 寄存器分配
- 常量嵌入
- 指令合并

寄存器分配

- Reg Allocation
- 优化 pass 分为寄存器分配前后两类

```
void optimize_before_reg_alloc(Program *prog) {
    for (auto &f:prog->funcs) more_constant_info(f.get());
    for (auto &f:prog->funcs) inline_constant(f.get());
    for (auto &f:prog->funcs) merge_shift_binary_op(f.get());
    for (auto &f:prog->funcs) merge_add_ldr_str(f.get());
    for (auto &f:prog->funcs) remove_unused(f.get());
}
```

```
void optimize_after_reg_alloc(Func *func) {
    remove_unused(func);
    remove_identical_move(func);
    remove_no_effect(func);
    direct_jump(func);
    eliminate_branch(func);
}
```

寄存器分配

- 实现了 *George, L., & Appel, A.W. (1996). Iterated register coalescing.* 中的算法
- 选择 spill 的寄存器的估价
  - 加载常量的代价为 1 或 2（*mov* / *movw* + *movt*）
  - 访存代价设为 5
  - 配合中端对基本块执行频率的估计
- 还会统计成功消除的 *move* 数量

```
register allocation for function main
reg_n = 168
using ColoringAllocator
Register allocation:
spill: 2
move instructions eliminated: 10
Callee-save registers used: 9
```

常量嵌入

- 在 IR 中，所有操作数都是伪寄存器，常量要提前用 *LoadConst* 加载
- 对于 *add* / *sub* / *cmp* 这样的指令，它们的第二操作数可以是 **有限制条件的** 常量
- 乘形如 2^k 2^k-1 2^k+1 的常量时，可以替换成移位（与加法 / 减法）
- 除 2^k 时替换成 *cmp, addlt, asr*
- 当访存地址是栈上的固定偏移时，替换成基于 *sp* 的偏移访存
- 做这样的替换可以节省用于加载常量的伪寄存器，大大降低寄存器分配的压力

指令合并

- 合并移位与之后的 *add / sub / rsb / ldr /str*
- 合并 *add* 与之后的 *ldr / str*
- 我们并未把它实现成窥孔优化，而且是在寄存器分配前进行
  - 因为伪寄存器形式是几乎 SSA 的，这样方便合并不连续的指令

```
mov r2, r1, LSL #2
add r4, r3, r2
ldr r0, [r4]

ldr r0, [r3, r1, LSL #2]
```

决赛成绩

初赛 决赛

比赛提交到排行榜更新有20秒左右的延迟

详情

#	用户名	队伍	提交次数(ASC)	最后提交时间(ASC)	正确分	性能分	总分
1	coltyang	小林家的编译器/ 清华大学	15	2021-08-17 16:17:38	100	80.9403	85.1753
2	buaa18373446	No Segmentation Fault Work/ 北京航空航天大学	44	2021-08-17 17:44:15	100	76.3155	81.5782
3	sad	沙梨酱耶/ 湖南大学	23	2021-08-17 17:42:45	100	68.0722	75.1666
4	ywh2000	TINBAC Is Not Building A Compiler/ 华南理工大学	20	2021-08-17 17:52:28	100	61.3255	69.9190
5	Forever518	早安！白给人/ 北京航空航天大学	16	2021-08-17 16:38:18	100	51.7883	62.5009
6	luooofan	胡编乱造不队/ 西北工业大学	23	2021-08-17 17:53:50	100	51.6515	62.3945
7	18373636	真实匿名队/ 北京航空航天大学	10	2021-08-17 18:22:02	100	50.9626	61.8587
8	wildoranges	Maho_shojo/ 中国科学技术大学	9	2021-08-17 17:52:53	100	42.6953	55.4284
9	pku1800012941	全场景分布式优化队/ 北京大学	24	2021-08-17 17:54:34	100	42.5322	55.3015

问答环节

问答环节

- 感谢评委老师指导