RIFS: Run-time Invariant Function Specialization

Saba Jamilan, Snehasish Kumar, Heiner Litz
Center for Research in Systems and Storage
University of California, Santa Cruz

UC SANTA CRUZ
BaskinEngineering

Center for Research
in Systems and Storage

NSF

Motivation

- **Workloads in both industry and academia are becoming increasingly complex**
  - Sophisticated control and data flows
- **Software complexity leads to unexpected inefficiencies**
  - Hard to detect and locate manually
  - Prevents achieving optimal performance
- **Compiler optimization techniques**
  - Improve the performance
  - Reduce the power consumption of applications
- **In this project, we focus on optimizing runtime value invariant function calls and we introduce RIFS.**

Motivating Example

```
int foo(int a, int b) {
   int c,d;
    a = 76;
   compute(a, c);
   ..
   compute(a, d);
}
int compute(int x, int y) {

  for(int i =0 ; i< y; i++){
      if(x % 2 != 1){
         x++;
         y= x+2*y;
         ….
      }
   } return x + y;}
```

Constant Propagation

✓

Motivating Example

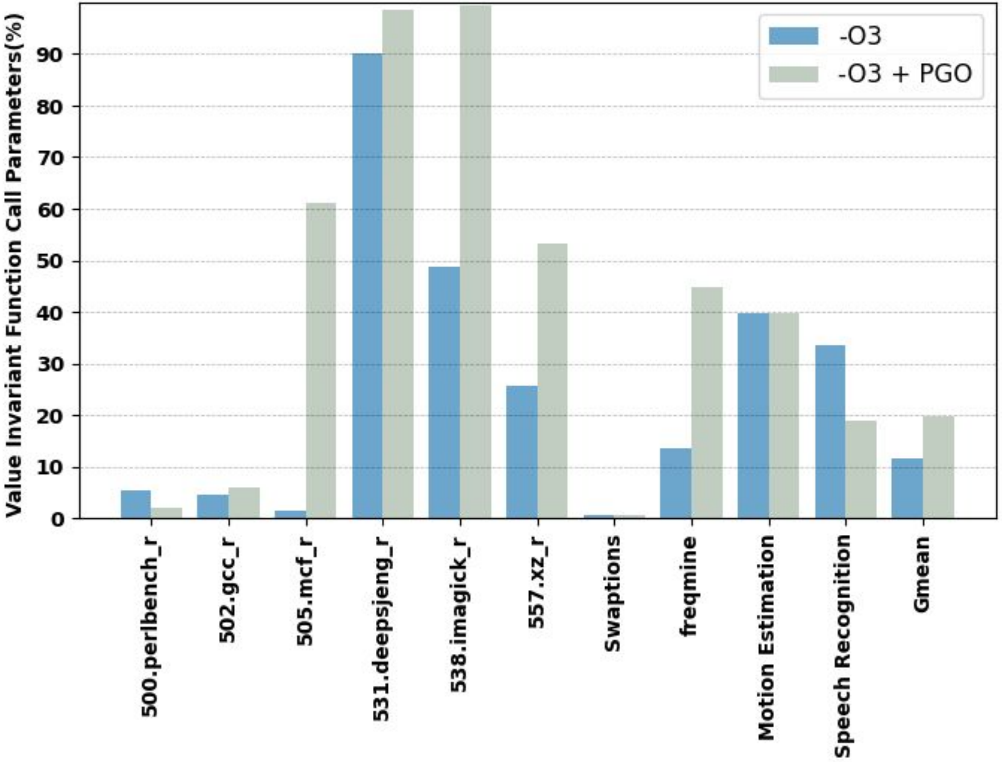
```
int foo(int a, int b) {
    int c,d;
    compute(a, c);      a = 76
    ..
    compute(a, d);
}
int compute(int x, int y) {
  for(int i =0 ; i< y; i++){
      if(x % 2 != 1){
          x++;
          y= x+2*y;
          ….
      }
  }
  return x + y;}
```

Are existing compilers able to optimize the redundant operations in compute function?

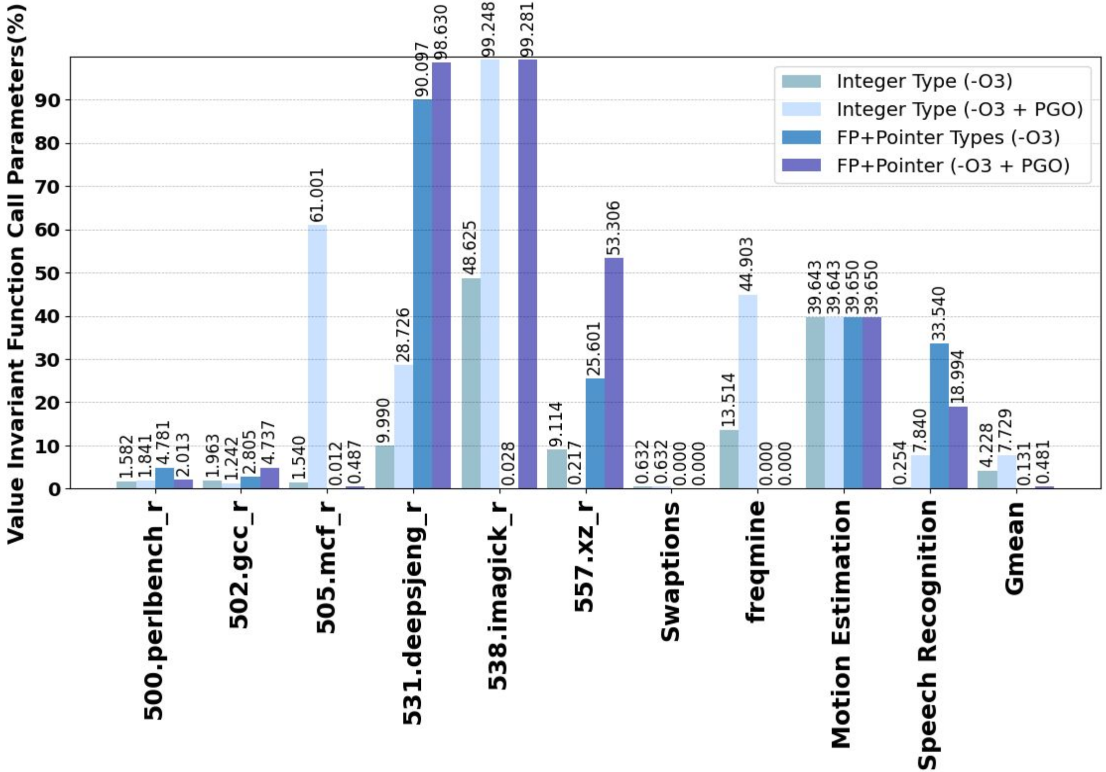
Existing Compiler Optimizations

- **PGO = Profile-Guided Optimization**
  - More information about application behavior → better optimization opportunities
    - **Profiled data**
      - Control Flow, execution counts, object types, values …
    - **Some example of optimizations:**
      - Block layout
      - Inlining heuristics
      - …
- **LTO = Link-time Optimization (BOLT, Propeller)**
  - Function and basic-block reordering to optimize instruction cache performance

Value Invariant Function call parameters



Data type of value invariant function call parameters



Value Invariant Function call parameters

Application	#Function	#Call Sites	#Dynamic Calls	#Dynamic Invariant Calls	#Fully Invariant Arguments	#Semi Invariant Arguments
500.perlbench_r	2373	3529	550202562	10128257	330	37
502.gcc_r	5973	23412	462328286	5741878	2371	126
505.mcf_r	36	106	89561282	54633640	10	3
531.deepsjeng_r	96	220	211391314	60723797	95	27
538.imagick_r	1950	1111	24028558	23847907	209	12
557.xz_r	307	256	13818099	29972	68	4
Swaptions	24	59	474601872	3000060	8	0
freqmine	42	117	116915331	52499017	6	1
Motion Estimation	11	38	29467837	11681808	5	0
Spectral Clustering	12	58	2017367	4	6	0
Kmeans Clustering	5	39	22530853	1	3	0
Speech Recognition	771	1211	21015711	1647539	252	12

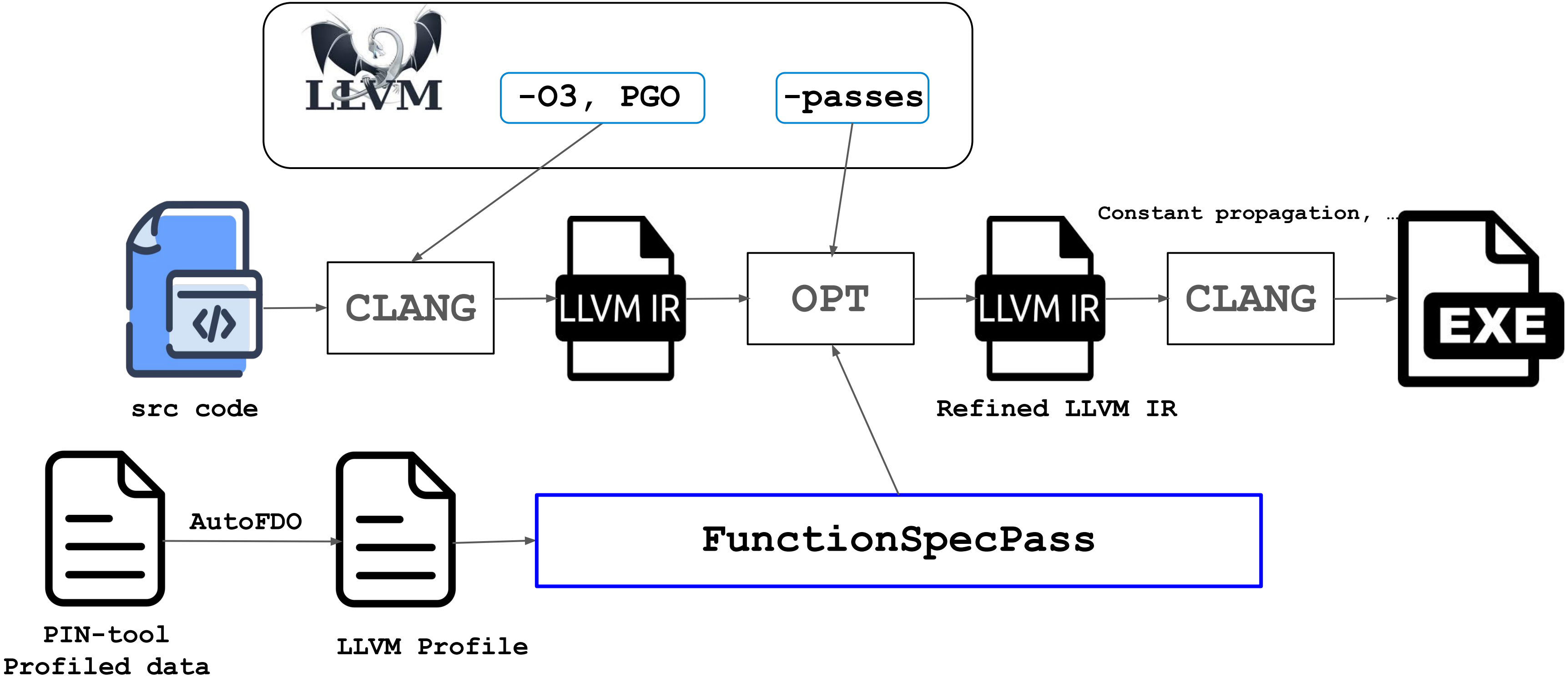
RIFS

- **We propose runtime invariant function specialization (RIFS):**
  - A technique to improve performance via value invariant function specialization
  - An application independent, generic technique that can be integrated into existing profile-guided optimization pipelines.
  - An Intel Pin-based value profiling tool to capture value invariant behavior of function call parameters
  - An LLVM function level pass utilizing profile data for automatic and safe code transformations

Profile Collection

- **Runtime profiling is required to track value-invariant variables**
- **A PIN-based value profiling tool**
  - Profiles function calls with integer, floating point, and pointer type arguments.
  - Utilizes GDB to capture the full signature of the functions including the number, type, and index of all arguments.
  - For each argument of each instrumented function, the tool caches the most frequently used N (N=8) values and their frequency of occurrence.
  - RIFS executes multiple Pin tool threads in parallel to reduce the total profiling time
  - A profile → The PC , name of the Caller, Callee, the execution frequency, the most frequent profiled values, and their occurrence ratio among all calls.

LLVM optimization pipeline



Profile-Guided Function Specialization LLVM Pass

```
Require: Profiling files (AutoFDOProfile, PinProfile)
procedure FuncSpecPass ( Function F )
    for all PC in PinProfile do                                    ▷ PC of Call
        ArgIndexVec[PC] ← argument index
        ArgValVec[PC] ← argument value
        ArgInfo←{ ArgIndexVec, ArgValVec }
    for all BB in F do
        for all I in BB do
            Found←SearchForIRInstr(I,AutoFDOProfile)
            if Found then
                Calls ← I
    for all C in Calls do
        Callee←getCalledFunction(C)
        if <isCandidateFunction(Callee)> then
            do_FuncSpecialization(C, ArgInfo)
procedure do_FuncSpecialization ( C, ArgInfo )
    FS←createSpecialization(Callee)
    for all Arg in FS do
        if <Arg is Value Invariant> then
            ArgsToUpdate ← Arg
    for all Elem in ArgsToUpdate do
        NewArg ← CreateNewArg(elem, ArgVal)
        for all dep in ArgsDepInstr[Elem] do
            NewArg ← setOperand(dep)
    SplitBlockAndInsertIfThenElse(Condition, C, ThenTerm, ElseTerm)
    Condition ← Arg value is EQUAL to the Profiled Value
    ThenTerm ← FastCallPath
    ElseTerm ← OrgCallPath
    if <Condition> then
        Jump to "FastCallPath"
        FastCall ← C.clone()
        FS ← setCalledFunction(FastCall)
    else
        Jump to "OrgCallPath"
```

(1) Determines all functions on the IR level that need to be specialized

(2) Replicates the body of all specialized functions

(3) Replaces the the function argument with a constant local variable in the replica

(4) Inserts a branch instruction to select between the original and replica function.

IR of Callsite before/after Function Specialization Pass

```
; Function Attrs: mustprogress uwtable
define dso_local noundef i32 @_Z21HJM_Swaption_BlockingPddddddiidS_PS_llii(ptr
nocapture noundef writeonly %0, double noundef %1, double noundef %2, double
noundef %3, double noundef %4, double noundef %5, i32 noundef %6, i32 noundef %7,
double noundef %8, ptr noundef %9, ptr noundef %10, i64 noundef %11, i64 noundef
%12, i32 noundef %13, i32 noundef %14) local_unnamed_addr #3 !dbg !1254 {
....

123:                                                  ; preds = %326, %93
 %124 = phi double [ 0.000000e+00, %93 ], [ %328, %326 ]
 %125 = phi double [ 0.000000e+00, %93 ], [ %327, %326 ]
 %126 = phi i64 [ 0, %93 ], [ %329, %326 ]
 %127 = call noundef i32 @_Z28HJM_SimPath_Forward_BlockingPPdiidS_S_S0_Pli(ptr
noundef %35, i32 noundef %6, i32 noundef %7, double noundef %8, ptr noundef %36,
ptr noundef %42, ptr noundef %10, ptr noundef nonnull %16, i32 noundef %13), !dbg
!1323
%128 = icmp eq i32 %127, 1, !dbg !1324
 br i1 %128, label %129, label %347, !dbg !1325

129:                                                  ; preds = %123
 br i1 %108, label %196, label %130, !dbg !1326
...
```

```
; Function Attrs: mustprogress uwtable
define dso_local noundef i32 @_Z28HJM_SimPath_Forward_BlockingPPdiidS_S_S0_Pli(ptr nocapture noundef
readonly %0, i32 noundef %1, i32 noundef %2, double noundef %3, ptr nocapture noundef readonly %4, ptr
nocapture noundef readonly %5, ptr nocapture noundef readonly %6, ptr noundef %7, i32 noundef %8)
local_unnamed_addr #3 !dbg !1090 {
123:                                                  ; preds = %330, %93
 %124 = phi double [ 0.000000e+00, %93 ], [ %332, %330 ]
 %125 = phi double [ 0.000000e+00, %93 ], [ %331, %330 ]
 %126 = phi i64 [ 0, %93 ], [ %333, %330 ]
 %127 = trunc i64 76 to i32
 %128 = icmp eq i32 %7, %127
 br i1 %128, label %FastCallPath-0, label %OrgCallPath-0
FastCallPath-0:                                       ; preds = %123
 %129 = call noundef i32 @_Z28HJM_SimPath_Forward_BlockingPPdiidS_S_S0_Pli.1(ptr noundef %35, i32
noundef %6, i32 noundef %7, double noundef %8, ptr noundef %36, ptr noundef %42, ptr noundef %10, ptr
noundef nonnull %16, i32 noundef %13), !dbg !1323
 br label %tail-0
OrgCallPath-0:                                        ; preds = %123
 %130 = call noundef i32 @_Z28HJM_SimPath_Forward_BlockingPPdiidS_S_S0_Pli(ptr noundef %35, i32
noundef %6, i32 noundef %7, double noundef %8, ptr noundef %36, ptr noundef %42, ptr noundef %10, ptr
noundef nonnull %16, i32 noundef %13), !dbg !1323
 br label %tail-0
tail-0:                                               ; preds = %OrgCallPath-0, %FastCallPath-0
 %131 = phi i32 [ %129, %FastCallPath-0 ], [ %130, %OrgCallPath-0 ], !dbg !1323
 %132 = icmp eq i32 %131, 1, !dbg !1324
 br i1 %132, label %133, label %351, !dbg !1325
133:                                                  ; preds = %tail-0
 br i1 %108, label %200, label %134, !dbg !1326
```

IR of Callee before Function Specialization Pass

CRSS

_Z28HJM_SimPath_Forward_BlockingPPdiidS_S_S0_Pli(swaptions)

```
; Function Attrs: mustprogress uwtable
define dso_local noundef i32 @_Z28HJM_SimPath_Forward_BlockingPPdiidS_S_S0_Pli(ptr
nocapture noundef readonly %0, i32 noundef %1, i32 noundef %2, double noundef %3, ptr
nocapture noundef readonly %4, ptr nocapture noundef readonly %5, ptr nocapture noundef
readonly %6, ptr noundef %7, i32 noundef %8) local_unnamed_addr #3 !dbg !1090 {
 %10 = sitofp i32 %1 to double, !dbg !1091
 %11 = fdiv double %3, %10, !dbg !1092
 %12 = tail call double @sqrt(double noundef %11) #28, !dbg !1093
 %13 = add nsw i32 %2, -1, !dbg !1094
 %14 = sext i32 %13 to i64, !dbg !1095
 %15 = mul nsw i32 %8, %1, !dbg !1096
 %16 = add nsw i32 %15, -1, !dbg !1097
 %17 = sext i32 %16 to i64, !dbg !1098
 %18 = tail call noundef ptr @_Z7dmatrixllll(i64 noundef 0, i64 noundef %14, i64 noundef
0, i64 noundef %17), !dbg !1099
 %19 = tail call noundef ptr @_Z7dmatrixllll(i64 noundef 0, i64 noundef %14, i64 noundef
0, i64 noundef %17), !dbg !1100
 %20 = icmp sgt i32 %8, 0, !dbg !1101
 %21 = icmp sgt i32 %1, 0
 %22 = and i1 %20, %21, !dbg !1103
 br i1 %22, label %23, label %167, !dbg !1103

23: …
```

```
All instructions that are dependent to the value of arg %2:
elem: i32 %2, Index: 2, val: 3
Instr:   %13 = add nsw i32 %2, -1, !dbg !32 op_index: 0
Instr:   %139 = icmp sgt i32 %2, 0 op_index: 0
Instr:   %146 = zext i32 %2 to i64 op_index: 0
Instr:   %168 = icmp sgt i32 %2, 0, !dbg !119 op_index: 0
Instr:   %171 = icmp sgt i32 %2, 0, !dbg !119 op_index: 0
Instr:   %177 = zext i32 %2 to i64, !dbg !119 op_index: 0
Instr:   %213 = zext i32 %2 to i64 op_index: 0
Instr:   %215 = icmp eq i32 %2, 1 op_index: 0
```

IR of Callee before Function Specialization Pass

CRSS

_Z28HJM_SimPath_Forward_BlockingPPdiidS_S_S0_Pli(swaptions)

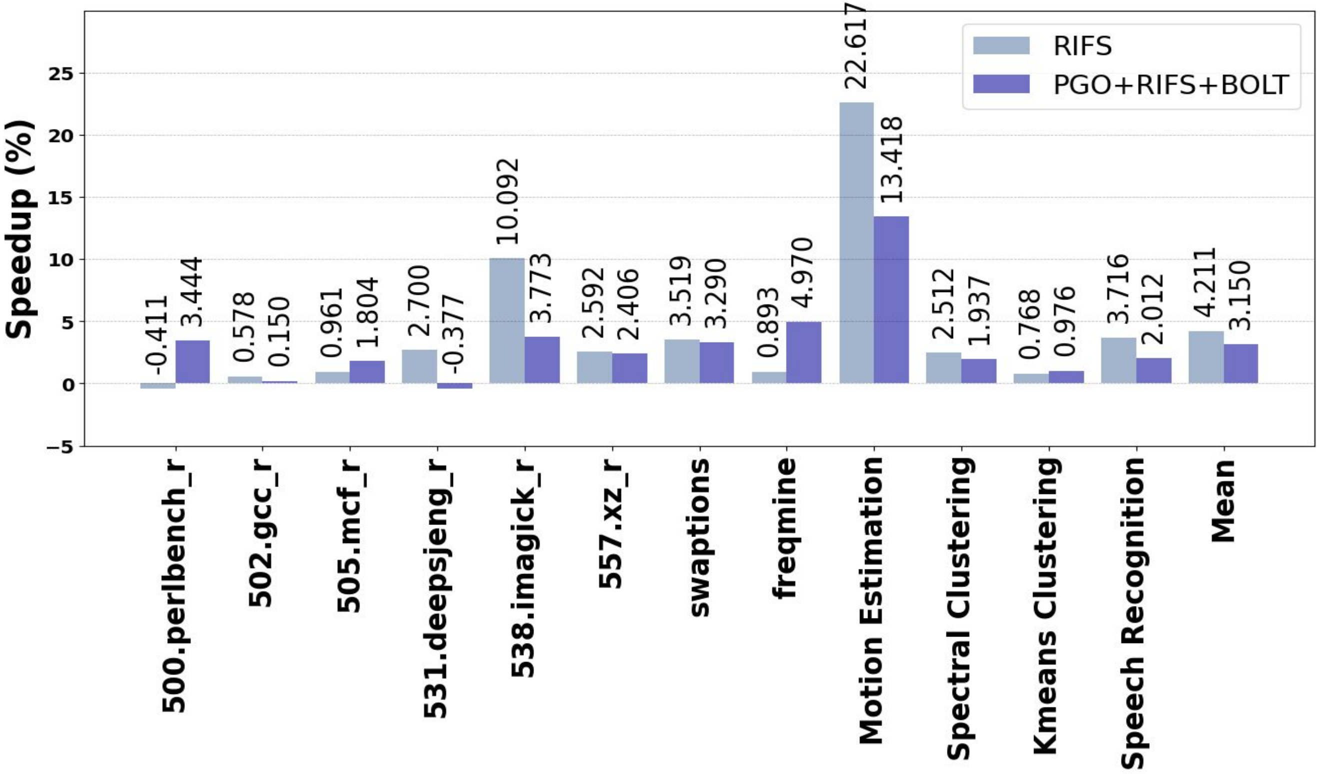
```
; Function Attrs: mustprogress uwtable
define dso_local noundef i32 @_Z28HJM_SimPath_Forward_BlockingPPdiidS_S_S0_Pli(ptr
nocapture noundef readonly %0, i32 noundef %1, i32 noundef %2, double noundef %3, ptr
nocapture noundef readonly %4, ptr nocapture noundef readonly %5, ptr nocapture noundef
readonly %6, ptr noundef %7, i32 noundef %8) local_unnamed_addr #3 !dbg !1090 {
 %10 = sitofp i32 %1 to double, !dbg !1091
 %11 = fdiv double %3, %10, !dbg !1092
 %12 = tail call double @sqrt(double noundef %11) #28, !dbg !1093
 %13 = add nsw i32 %2, -1, !dbg !1094
 %14 = sext i32 %13 to i64, !dbg !1095
 %15 = mul nsw i32 %8, %1, !dbg !1096
 %16 = add nsw i32 %15, -1, !dbg !1097
 %17 = sext i32 %16 to i64, !dbg !1098
 %18 = tail call noundef ptr @_Z7dmatrixllll(i64 noundef 0, i64 noundef %14, i64 noundef
0, i64 noundef %17), !dbg !1099
 %19 = tail call noundef ptr @_Z7dmatrixllll(i64 noundef 0, i64 noundef %14, i64 noundef
0, i64 noundef %17), !dbg !1100
 %20 = icmp sgt i32 %8, 0, !dbg !1101
 %21 = icmp sgt i32 %1, 0
 %22 = and i1 %20, %21, !dbg !1103
 br i1 %22, label %23, label %167, !dbg !1103

23: …
```

_Z28HJM_SimPath_Forward_BlockingPPdiidS_S_S0_Pli.1(swaptions)

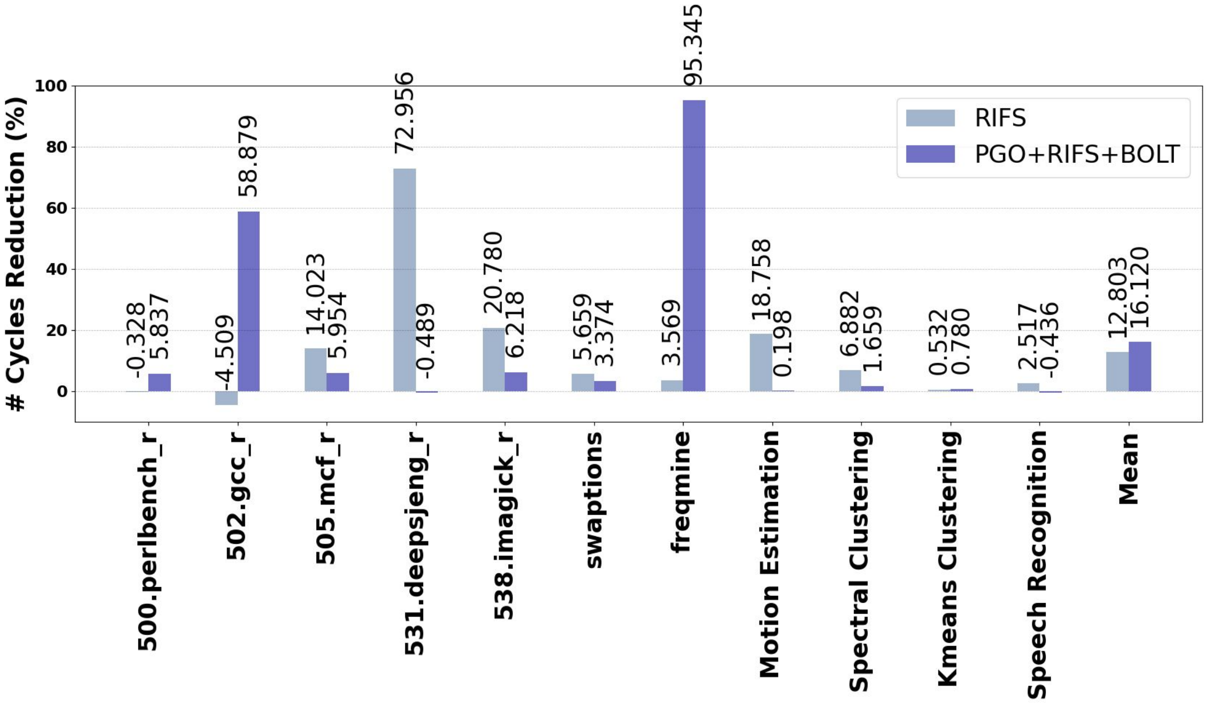
```
; Function Attrs: mustprogress uwtable
define dso_local noundef i32 @_Z28HJM_SimPath_Forward_BlockingPPdiidS_S_S0_Pli.1(ptr nocapture
noundef readonly %0, i32 noundef %1, i32 noundef %2, double noundef %3, ptr nocapture noundef
readonly %4, ptr nocapture noundef readonly %5, ptr nocapture noundef readonly %6, ptr noundef
%7, i32 noundef %8) local_unnamed_addr #3 !dbg !1946 {
 %arg2 = alloca i32, align 4, !dbg !1947
 store i32 76, ptr %arg2, align 4, !dbg !1947
 %argLoaded2 = load i32, ptr %arg2, align 4, !dbg !1947
 %10 = sitofp i32 %1 to double, !dbg !1947
 %11 = fdiv double %3, %10, !dbg !1948
 %12 = tail call double @sqrt(double noundef %11) #28, !dbg !1949
 %13 = add nsw i32 %argLoaded2, -1, !dbg !1950
 %14 = sext i32 %13 to i64, !dbg !1951
 %15 = mul nsw i32 %8, %1, !dbg !1952
 %16 = add nsw i32 %15, -1, !dbg !1953
 %17 = sext i32 %16 to i64, !dbg !1954
 %18 = tail call noundef ptr @_Z7dmatrixllll(i64 noundef 0, i64 noundef %14, i64 noundef 0, i64
noundef %17), !dbg !1955
 %19 = tail call noundef ptr @_Z7dmatrixllll(i64 noundef 0, i64 noundef %14, i64 noundef 0, i64
noundef %17), !dbg !1956
 %20 = icmp sgt i32 %8, 0, !dbg !1957
 %21 = icmp sgt i32 %1, 0
 %22 = and i1 %20, %21, !dbg !1959
 br i1 %22, label %23, label %167, !dbg !1959
23: …
```

Speedup(x)

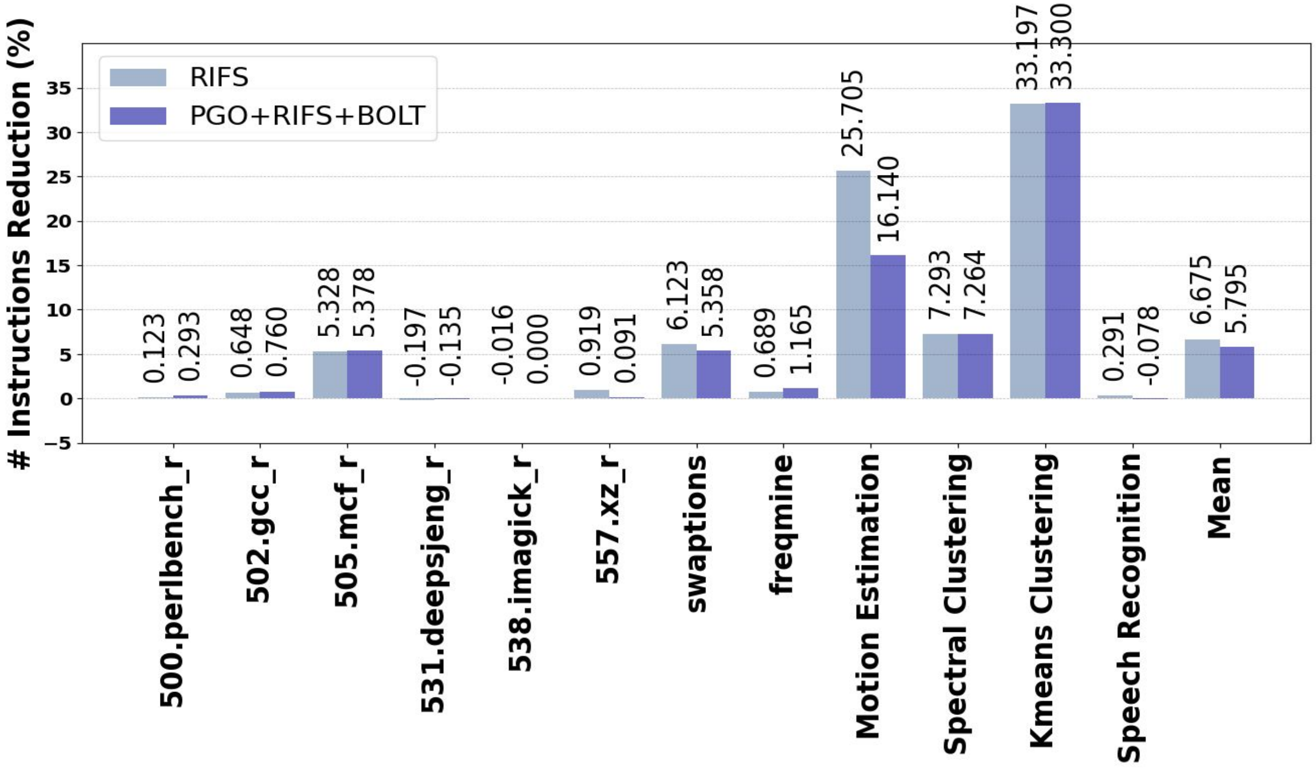


RIFS
PGO+RIFS+BOLT
Speedup (%)
25
20
15
10
5
0
−5
-0.411
3.444
0.578
0.150
0.961
1.804
2.700
-0.377
10.092
3.773
2.592
2.406
3.519
3.290
0.893
4.970
22.617
13.418
2.512
1.937
0.768
0.976
3.716
2.012
4.211
3.150
500.perlbench_r
502.gcc_r
505.mcf_r
531.deepsjeng_r
538.imagick_r
557.xz_r
swaptions
freqmine
Motion Estimation
Spectral Clustering
Kmeans Clustering
Speech Recognition
Mean

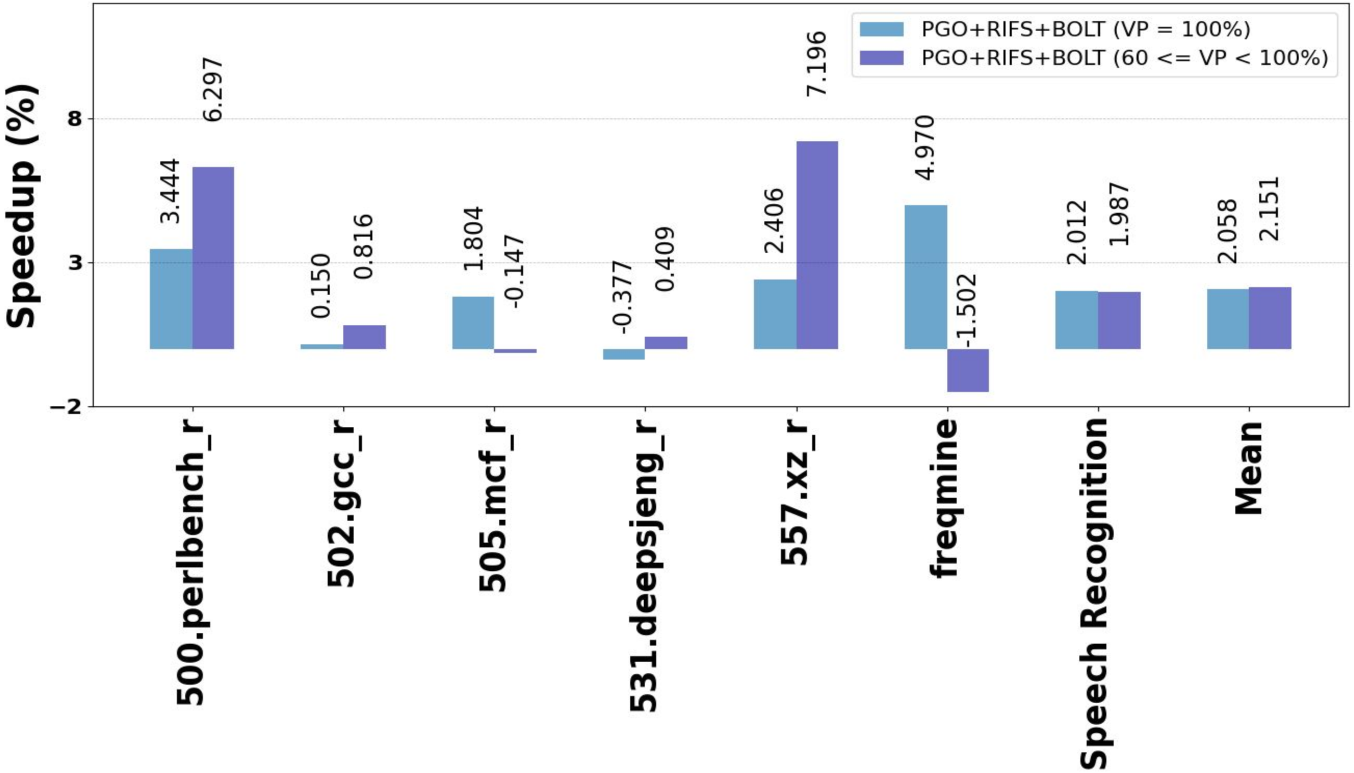
Cycles reduction (x) consumed by original and specialized functions



Instructions Reduction(x)



Speedup(x) for semi-invariant function call parameters



Conclusion

- RIFS leverages a binary instrumentation profiling tool to learn invariant function call arguments
- We have shown that profiling data can be used to optimize code by enabling constant-value propagation opportunities.
- We introduce a fully automatic and safe LLVM code transformation pass that can be easily integrated into existing compilation pipelines.
- In the context of 12 real-world applications, RIFS achieves a speedup of 4.21% and instructions reduction of 6.67% on average over the LLVM baseline (-O3).
- RIFS improves execution time by 3.15% over PGO+BOLT and reduces instructions by 5.79% on average.

Future Works

- ❖ Increase the coverage of performing Function specialization for RIFS by enabling call stack analysis
- ❖ We want to study more when specializing value-invariant function calls can be beneficial for performance and when it is not beneficial → cost-model

Thank You

Questions?

Thank you to our sponsors!

arm

cerabyte

intel

MARVELL

NUTANIX

NSF

Speedup (x) for different optimization candidates policies