LLVM & HPSSA Hot Path SSA Form in LLVM

Presented By Abhay¹ & Muzzammil¹

¹IIT Kanpur PRAISE Group

Dr. Subhajit Roy, Dr. Awanish Pandey, Mr. Sumit Lahiri

What we modified in LLVM Source?

• New llvm::intrinsic signature, "llvm.tau" to support addition and removal of τ -functions to the LLVM SSA IR representation.

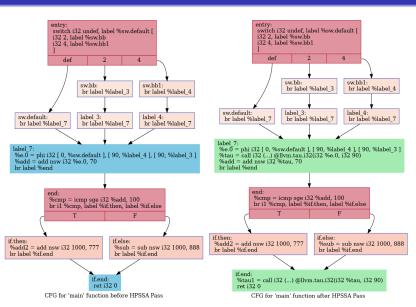
What we modified in LLVM Source?

 Modified Verifier::verifyDominatesUse() function since we don't want our intrinsic to interfere with dominators computation.

HPSSAPass: Overview

- class HPSSAPass : public PassInfoMixin<HPSSAPass>
 - Implemented llvm::HPSSAPass pass using the new LLVM Pass Manager.
 - Function HPSSAPass::run(Function \&F, ...) runs over a llvm::Function and inserts "llvm.tau" intrinsic calls with speculative and safe arguments at strategic positions in the LLVM IR and handles argument allocation for "llvm.tau" intrinsic calls as described in the previous slides.
- Key HPSSA Data Structures :
 - Hot Path Set using llvm::BitVector for maintaining hot paths in the program.
 - Definition Accumulator, defAccumulator(op, currBB) function.
 The argument "op" is a phi argument that reaches basic-block "currBB" via hot path.
 - A stack of map values std::map<Value*, Value*> to store the
 most "recent" tau definition encountered so far corresponding for a
 tau variable used later in variable renaming.

HPSSA Transformation



HPSSAPass: Main Pass

- $\bullet \ \ \, \mathtt{HPSSAPass::run}(\mathtt{Function} \ \ \underline{\mathsf{N}} \& \mathtt{F}, \ \, \mathtt{FunctionAnalysisManager} \ \ \underline{\mathsf{N}} \& \mathtt{AM}) \\$
 - Invoke HPSSAPass::getProfileInfo(Function \(\lambda \kappa F \) function to get a compact representation of all the hot paths in the program and then call HPSSAPass::getCaloricConnector(Function \(\lambda \kappa F \)) to get all the caloric connectors from the hot path information. This is a precursor to finding strategic positions to place "llvm.tau" intrinsic calls in the LLVM IR.
 - Runs over each basic block in the function "F" in topological order using iterator returned from llvm::Function::RPOT() call.
 - Uses the llvm::dominates() function from llvm::DominatorTreeAnalysis to check for dominance frontier while processing the child nodes of the current basic block. This step is a part of correctly placing "llvm.tau" intrinsic calls in the LLVM IR.
 - Uses the renaming stack and HPSSAPass::Search(BasicBlock N&BB, DomTreeNode N&DTN) function to search and replace all use of PHI result operand with that returned by the "llvm.tau" intrinsic call.

HPSSAPass: Destruction Pass

- Out of HPSSA Form.
 - A seperate pass using the new LLVM Pass Manager.
 class TDSTRPass : public PassInfoMixin<TDSTRPass>
 - Using TDSTRPass::run(Function \\delta F, ...), we replace all use of existing tau operands with first argument of "llvm.tau" intrinsic (corresponds to the safe argument) and remove the "llvm.tau" intrinsic call from the LLVM IR.
 - The LLVM IR becomes identical to what it was before running the HPSSA Pass.

HPSSAPass: Usage [It is easy!]

- Include llvm::HPSSAPass header file.
- Load shared object using opt tool. opt -load HPSSA.cpp.so ...

```
#include <HPSSA.h> // import the header.
1
      class MyExamplePass : public PassInfoMixin<MyExamplePass> {
        public: PreservedAnalyses run(Function &F,
3
          FunctionAnalysisManager &AM);
4
      };
5
6
      PreservedAnalyses MyExamplePass::run(Function &F,
        FunctionAnalysisManager &AM) {
8
        if (F.getName() != "main")
9
          return PreservedAnalyses::all();
10
        HPSSAPass hpssaUtil; // Make a HPSSAPass Object.
11
        hpssaUtil.run(F, AM); // Call the HPSSAPass::run() function.
12
        std::vector<Instruction *> TauInsts // Calling utility function.
13
          = hpssaUtil.getAllTauInstrunctions(F);
14
15
16
      /// [output] Total Tau Instructions : 7
17
                                                  4 D > 4 B > 4 B > 4 B > 9 Q P
```

New Additions to SCCP Pass

- Modified the existing SCCP Pass to add in SCCPInstVisitor::visitTauNode() function similar to SCCPInstVisitor::visitPHINode(), which handles the special "lvm.tau" intrinsic instructions added for τ-functions.
- Added a new lattice element type "spec_constant" in ValueLattice class supporting operations on speculative constants.
- Added new functions in the SCCPInstVisitor and SCCPSolver class to handle operations on speculative constants using markSpeculativeConstant() function.

Further Modifications

- Modified the SCCPInstVisitor::mergeIn() function to handle lattice "meet" operation for the new speculative constants introduced.
- Since we added the τ -functions as an "llvm.tau" intrinsic which is essentially an llvm:CallInst type, we modified all appropriate visit and marking functions in SCCPInstVisitor, SCCPSolver and SCCPPass to handle this case separately by calling visitTauNode().
- Modified utility functions in SCCPInstVisitor and SCCPSolver class to print marking of speculative constants and related operations for debugging purpose.

```
// Example for SSCCP Pass
    int main() {
2
      int a = 1000, z, c, e = 0;
3
4
      switch(c) {
5
        case 2 : goto label_3; break;
        case 4 : goto label_4; break;
6
        default : goto label_7; }
7
      label_3:
8
        e = 90:
9
        goto label_7;
10
      label_4:
11
        e = 100 - 10:
12
        goto label_7;
13
      label_7:
14
        e = e + 70; // e in rhs is 90.
15
        goto end;
16
17
      end:
        if (e \ge 100) { // e is greater than 100 always}
18
           a = a + 777;
19
        } else {
20
           a = a - 888;
21
        } return 0; }
22
```

SSCCP with an Example

