

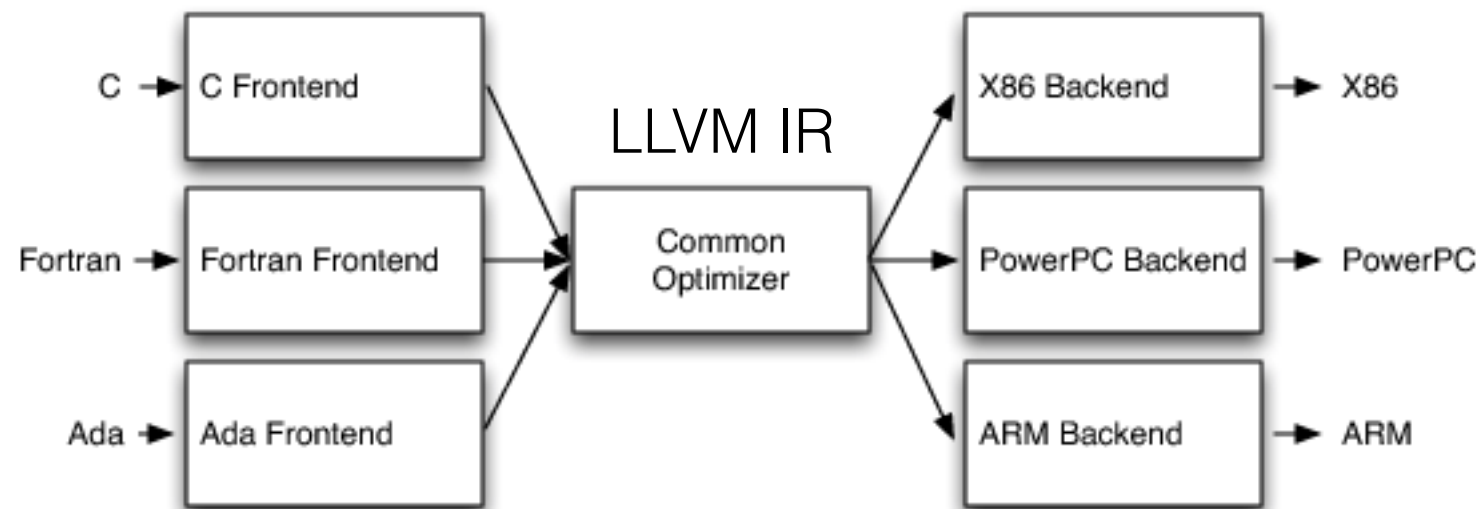
Array Access Analysis by Ilvmpy

Dong Chen



LLVM infrastructure

- Three Phase Design of LLVM infrastructure^[1]



LLVM Intermediate Representation (IR)

- Features:
 - Three address form
 - Static single assignment
 - low-level RISC like instruction set with high-level type information

- Example:

```
int max(int a, int b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```

```
define i32 @max(i32 %a, i32 %b) #0 {  
    %1 = alloca i32, align 4  
    %2 = alloca i32, align 4  
    %3 = alloca i32, align 4  
    store i32 %a, i32* %2, align 4  
    store i32 %b, i32* %3, align 4  
    %4 = load i32* %2, align 4  
    %5 = load i32* %3, align 4  
    %6 = icmp sgt i32 %4, %5  
    br i1 %6, label %7, label %9  
  
; <label>:7                                ; preds = %0  
    %8 = load i32* %2, align 4  
    store i32 %8, i32* %1  
    br label %11  
  
; <label>:9                                ; preds = %0  
    %10 = load i32* %3, align 4  
    store i32 %10, i32* %1  
    br label %11  
  
; <label>:11                               ; preds = %9, %7  
    %12 = load i32* %1  
    ret i32 %12  
}
```

LLVM API

- Features:
 - C++ implementation
 - templates are everywhere

```
for (Function::iterator b = F.begin(), be = F.end(); b != be; ++b) {  
    for (BasicBlock::iterator i = b->begin(), ie = b->end(); i != ie; ++i) {  
        if (CallInst* callInst = dyn_cast<CallInst>(&*i)) {  
            if (callInst->getCalledFunction() == targetFunc)  
                ++callCounter;  
        }  
    }  
}
```

LLVM API

- Features:
 - C++ implementation
 - templates are everywhere

```
for (Function::iterator b = F.begin(), be = F.end(); b != be; ++b) {  
    for (BasicBlock::iterator i = b->begin(), ie = b->end(); i != ie; ++i) {  
        if (CallInst* callInst = dyn_cast<CallInst>(&*i)) {  
            if (callInst->getCalledFunction() == targetFunc)  
                ++callCounter;  
        }  
    }  
}
```

- LLVMPY

```
for bb in f.basic_blocks:  
    for istr in bb.instructions:  
        if istr.opcode == OPCODE_CALL:  
            if its.operands[0] == targetFunc:  
                callCounter=callCounter+1
```

Array access analysis of GPU kernel program

- Kernel Program

- blockIdx
- threadIdx

```
__global__ void add( int *a, int *b, int *c ) {  
    int tid = blockIdx.x; // handle the data at this index  
    if (tid < N)  
        c[tid] = a[tid] + b[tid];  
}
```

- Array Access Analysis

- Extract the index
 - tid

- Calculate the range

- thread block |- 0 -|- 1 -|- 2 -|...
- threads |-0-|-1-|-2-|-3-|-0-|-1-|-2-|-3-|-0-|-1-|-2-|-3-|...
- array |-0-|-1-|-2-|-3-|-4-|-5-|-6-|-7-|-8-|-9-|10|11|...

Implementation

- Algorithm (Array index analysis)
 - locating array access instruction 'getelementptr'
 - construct prefix expression by define-use chain
 - construct infix expression from prefix expression
- Algorithm (Induction Variable analysis)
 - locating store to induction variable instructions
 - construct expression
- Algorithm (Constrain analysis)
 - locating the basic blocks contain array access
 - construct Execution Path Tree from control flow graph
 - extract branch conditions to constrains

Test

- MatrixMul

Array access analysis result=====

A = a + wA * ty + tx

B = b + wB * ty + tx

C = wB * 32 * by + 32 * bx + wB * ty + tx

induction variables analysis result=====

a = wA * 32 * blockIdx.y

a = a + 32

b = 32 * blockIdx.x

b = b + 32 * wB

constrains analysis result=====

full execution path:

Start From Basic Block 0 : [[0]]

Start From Basic Block 1 : [[1, 0], [1, 7, 6, 3, 2, 1], [1, 7, 6, 3, 5, 4, 3]]

Start From Basic Block 2 : [[2, 1, 0], [2, 1, 7, 6, 3, 2], [2, 1, 7, 6, 3, 2, 5, 4, 3]]

Start From Basic Block 3 : [[3, 5, 4, 3], [3, 2, 1, 0], [3, 2, 1, 7, 6, 3]]

Start From Basic Block 4 : [[4, 3, 5, 4], [4, 3, 2, 1, 0], [4, 3, 2, 1, 7, 6, 3]]

Start From Basic Block 5 : [[5, 4, 3, 5], [5, 4, 3, 2, 1, 0], [5, 4, 3, 2, 1, 7, 6, 3]]

Start From Basic Block 6 : [[6, 3, 5, 4, 3], [6, 3, 2, 1, 0], [6, 3, 2, 1, 7, 6]]

Start From Basic Block 7 : [[7, 6, 3, 2, 1, 7], [7, 6, 3, 5, 4, 3], [7, 6, 3, 2, 1, 0]]

Start From Basic Block 8 : [[8, 1, 0], [8, 1, 7, 6, 3, 2, 1], [8, 1, 7, 6, 3, 5, 4, 3]]

execution path of a,b:[2, 1, 0]

execution path of c:[8, 1, 0]

a,b constrains: $a \leq wA * 32 * blockIdx.y + wA - 1$

c constrains: $a > wA * 32 * blockIdx.y + wA - 1$

- Reference:
- [1] Chris Lattner. The Architecture of Open Source Applications: LLVM.