Scalar Evolution

Nikhil Hegde

Compiler Optimizations course @ QUALCOMM India Pvt. Ltd.

Scalar Evolution in LLVM

- Symbolic execution of scalar variables inside the loop
- SCEV – LLVM's implementation of Scalar Evolution
- Passes in LLVM using SCEV
  - Loop strength reduction
  - Induction variable simplification
  - Loop vectorizer
  - SCEV-AA, Memory dependence analysis, etc.

Optimize Loops –Strength Reduction

- Like strength reduction in peephole optimization
  - E.g. replace `a*2` with `a<<1`
- Applies to uses of induction variable in loops
  - Basic induction variable (`I`) – only definition within the loop is of the form `I = I ± S`, (S is loop invariant)

    `I` *usually determines number of iterations*
  - Mutual induction variable (`J`) – defined within the loop, its value is linear function of other induction variable, `I`, such that

    `J = I * C ± D` (C, D are loop invariants)

Optimize Loops –Strength Reduction

- Suppose induction variable I takes on values I_o, I_o+S, I_o+2S, I_o+3S... in iterations 1, 2, 3, 4, and so on…
- Then, in consecutive iterations, Expression I*C+D takes on values

I_o*C+D

(I_o+S)*C+D = I_o*C+S*C+D

(I_o+2S)*C+D = I_o*C+2S*C+D

... ...

- The expression changes by a constant S*C
- Therefore, we have replaced a * and + with a +

Scalar Evolution in LLVM – Formalism

- Recurrences
  - Recurrence example:

$$n! = \begin{cases} n \times (n-1)! & \text{when } n \geq 1 \\ 1 & \text{when } n = 0 \\ \text{undefined} & \text{when } n < 0 \end{cases}$$

```
int f=k0
for(int i=0;i<n;i++){
    ... = f;
    f = f + k1;
}
```

$$f(i) = \begin{cases} k0 & \text{when } i = 0 \\ f(i-1) + k1 & \text{when } i > 0 \end{cases}$$

SCEV notation: {k0, +, k1}

E.g., {2, +, 5} denotes the sequence of values for f :
2, 7, 12, 17, …

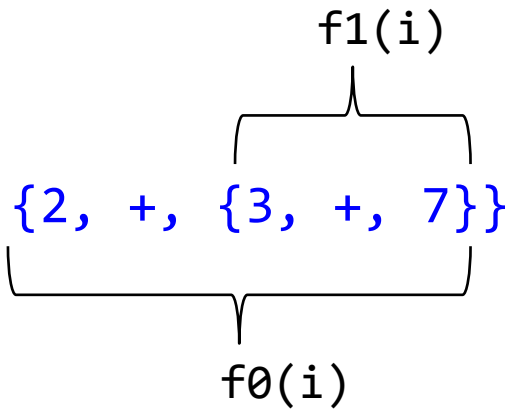
```
int f=2;
for(int i=0;i<n;i++){
    ... = f;
    f = f + 5;
}
```

Scalar Evolution in LLVM – Demo1

- `clang –emit-llvm scevex1.c –S –O1`
- `opt -passes="print<scalar-evolution>" -disable-output scevex1.ll`

Scalar Evolution in LLVM – Formalism

- Chain of Recurrences
  - example:



$$f1(i) = \begin{cases} 3 & \text{when } i=0 \\ f1(i-1) + 7 & \text{when } i>0 \end{cases}$$

$$f0(i) = \begin{cases} 3 & \text{when } i=0 \\ f0(i-1) + f1(i-1) & \text{when } i>0 \end{cases}$$

Alternative notation: {2, +, 3, +, 7}

Scalar Evolution in LLVM – Formalism

- Chain of Recurrences
  - example:

```
for(int i=0;i<n;i++){
    a[i] = i*i*i + 2*i*i + 3*i + 7;
}
```

i	0	1	2	3	4	5
a[i]	7	13	29	61	115	197
D_i (a[i]-a[i-1])	-	6	16	32	54	82
$D' = D_i - D_{i-1}$	-	-	10	16	22	28
$D'' = D'_i - D'_{i-1}$	-	-	-	6	6	6

$\{7, +, 6, +, 10, +, 6\}$

Scalar Evolution in LLVM – Demo3

- clang –emit-llvm scevex3.c –S –O1
- opt -passes="print<scalar-evolution>" -disable-output scevex3.ll

{7, +, 6, +, 10, +, 6}

```
for(int i=0;i<n;i++)
   a[i] = i*i*i + 2*i*i + 3*i + 7;
```

```
t0=7
t1=6
t2=10
for(i=0;i<n;i++)
    a[i]=t0;
    t0 += t1;
    t1 += t2;
    t2 += 6;
```

Scalar Evolution in LLVM – Formalism

- Chain of Recurrences (rewriting and folding)
  - example:

Iteration	0	1	2
i: {7, +, 3}	7	10	13
j: {1, +, 1}	1	2	3
k=i+j	8	12	16

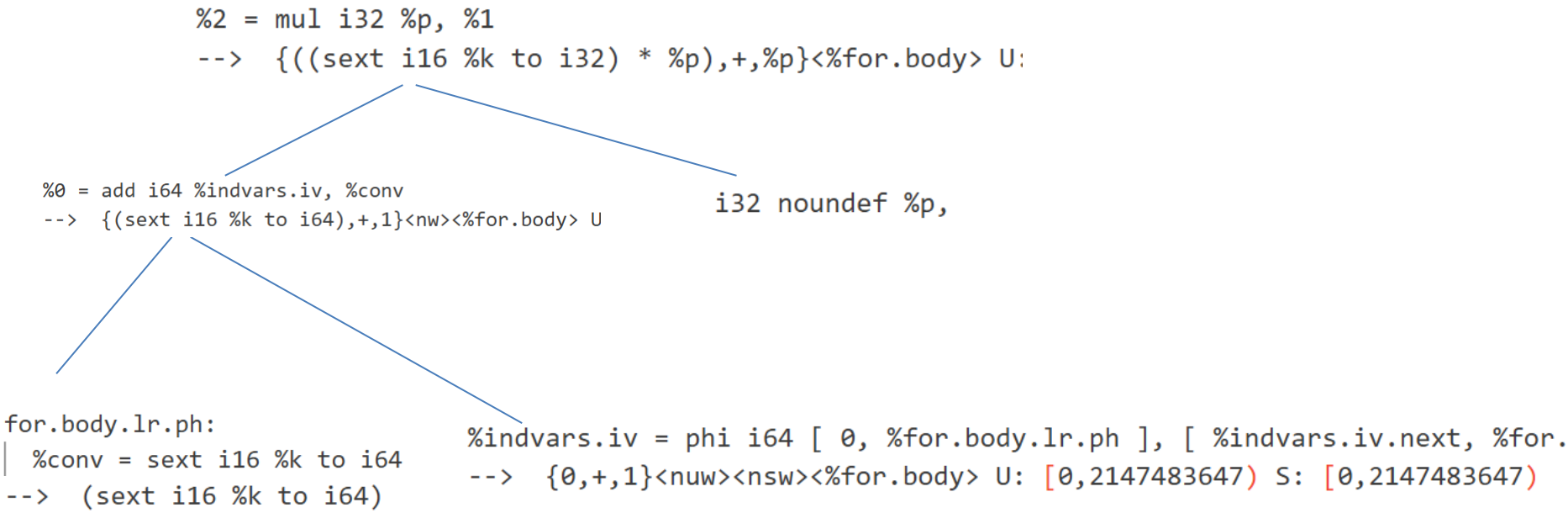
```
{7, +, 3, +, 1, +, 1} = {8, +, 4}
```

```
{{e, +, f}, +, {g, +, h}} = {e+g, +, f+h}
```

This is a rule. Other such rules are applied during rewriting and folding.

Scalar Evolution in LLVM – Demo4

- clang –emit-llvm scevex4.c –S –O1
- opt -passes="print<scalar-evolution>" -disable-output scevex4.ll



Rules

Expression		Rewrite	Example		
$G + \{e, +, f\}$	$\Rightarrow$	$\{G + e, +, f\}$	$12 + \{7, +, 3\}$	$\Rightarrow$	$\{19, +, 3\}$
$G * \{e, +, f\}$	$\Rightarrow$	$\{G * e, +, G * f\}$	$12 * \{7, +, 3\}$	$\Rightarrow$	$\{84, +, 36\}$
$\{e, +, f\} + \{g, +, h\}$	$\Rightarrow$	$\{e + g, +, f + h\}$	$\{7, +, 3\} + \{1, +, 1\}$	$\Rightarrow$	$\{8, +, 4\}$
$\{e, +, f\} * \{g, +, h\}$	$\Rightarrow$	$\left\{\begin{matrix} e * g, +, \\ e * h + f * g + f * h, \\ +, 2 * f * h \end{matrix}\right\}$	$\{0, +, 1\} * \{0, +, 1\}$	$\Rightarrow$	$\{0, +, 1, +, 2\}$

source: Javed Absar – Scalar Evolution - Demystified