Indian Institute of Technology, New Delhi – 110016

COL-731

Course Presentation

# Superoptimizer -- A Look at the Smallest Program

Henry Massalin

# Introduction

- Finding Optimal Algorithm to compute a function is a key challenge
  - Example - Matrix Multiplication
  - Lower Bound
  - Does not establish optimality based on the program's execution time
- Instead superoptimizer finds the shortest program in the program space
- In general proving the equivalence between 2 programs is undecidable
- But we are restricting the things

# Motivating Example

```
signum:
    cmp.l #0, D0        ; Compare x with 0
    bgt.s greater       ; Branch if x > 0
    blt.s less          ; Branch if x < 0
    move.l #0, D0       ; x is 0, set result to 0
    rts                 ; Return

greater:
    move.l #1, D0       ; x > 0, set result to 1
    rts                 ; Return

less:
    move.l #-1, D0      ; x < 0, set result to -1
    rts                 ; Return
```

# Motivating Example

```
(x in dO)
add.1 d0,d0   |add dO to itself
subx.1 d1,d1  |subtract (d1 + Carry) from d1
negx.1 dO     |put (0 - dO - Carry) into dO
addx.1 d1,d1  |add (d1 + Carry) to d1
(signum(x) in d1}
```

# Motivating Example

```
cwd           | Sign extends the value in register AX into DX:AX
neg ax        | Negates the value in register AX
adc dx, dx    | Adds the value in register DX to DX with carry
```

# Interesting Example

```
int myAbs(int x) {
    return (x < 0) ? -x : x;
}
```

# Interesting Example

```
(x in d0)
move.l d0,d1
add.l  d1,d1
subx.l d1,d1
eor.l  d1,d0
sub.l  d1,d0
(abs(x) in d0)
```

# Interesting Examples

Min-Max function

```
(d0=X, d1=Y) | Flag, Reg | If d1>d0 |If d1←d0
sub.1 d1,d0  | (C,d0)   = | (1, X-Y) |(0, X-Y)
subx.1 d2,d2 | d2 =       |  11...11 |000...000
and.1 d0, d2 | d2 =       |  X-Y     |0
eor.1 d2, d0 | d0 =       |  0       |X-Y
add.1 d1,d0  | d0  =      |  Y       |X
add.1 d2, d1 | d1  =      |  X       |Y
(dO = max(X, Y), d1 = min(X, Y))
```

# Interesting Examples

```
d0 = 0        | if d0 = 0
   = -1       | if d0 ≠0
neg.1  d0
subx.1  d0
```

```
d0 = 0 if d0 == 0
   = 1 if d0 ≠ 0
neg.1  d0
subx.1 d0,d0
neg.1 d0
```

```
d0 = -1 if d0 == 0
   =  0 if d0 ≠ 0
neg.1  d0
subx.1 d0, d0
not.1  d0
```

```
d0 == 1 if d0 == 0
   == 0 if d0 ≠ 0
neg.1 d0
subx.1 d0, d0
addq.1 1, d0
```

# Basic Algorithm

- Input – Program In Machine Language
- Objective-  shortest program computing same function
- Method- Exhaustive Search over all possible programs
- What is the search space of the exhaustive search?
  - Subset of the machine's instruction set
  - Op-codes are stored in a table for consultation
- Generates all combinations of these instructions
- Each generated program undergoes testing
- Halts after the function-matching criterion met

# Boolean Test

- Determines whether two code segments compute the same function
- Employs a technique called the boolean program verifier
- Function output using boolean-logic operations on the input argument
- Two programs were considered equivalent iff their boolean expressions matched minterm for minterm.
- MUL, ADD instructions have a large number of minterms
- In spite of memory reduction efforts, computing boolean expressions for every generated program proved time-consuming
- The initial superoptimizer version tested around 40 programs/sec
- Allowed the generation of programs with up to 3 instructions

# Probabilistic Test

- On same inputs, one has same outputs on source and target
- Most programs are expected to fail the simple test
- Program verification test when the above test fails
- Can currently test 50,000 programs/second, making the exhaustive search approach practical
- Test vectors are manually chosen to maximize the probability of catching errors early
- For example, test vectors for the signum function include -1000, 0, and 456

# Probabilistic Test

- A sequential test is performed on a range of numbers from -1024 to 1024
- Low probability of passing the execution test and failing the boolean verification test
- Advantageous as bool expression are too large to fit into memory in general
- Check manually without generating the Boolean expression

# Pruning

- Filter out instructions which are redundant
  - “move X,Y; move X,Y”
- Filtering is done with N-dimensional bit tables
- Each instruction in the sequence we wish to test indexes one dimension of the bit table
- A lookup value of ' 1' causes the program to be rejected as nonoptimal
- 2 ways to fill the lookup table
  - Manually fill up the table
  - Using boolean test

# Limitations

- Exhaustive search grows exponentially with time
- Pointers cause exponential blow-up in terms
- Restriction because instruction set

# Applications

- Can be used to analyze instruction sets
  - Western Electric WE32000 microprocessor and in every case the resulting program was longer than the 68020 programs.
  - National Semiconductor NS32032 was also found to suffer from flag problems
  - Reason- Lack of an add-with-carry instruction and the fact that the flags are set according to the 32 bit result, even for byte sized operands
  - Here the difficulty is that extra instructions are needed to test the outcome of an operation because few instructions set the flags.

# Applications

- RISC architecture design
  - Coding function in terms of Boolean expressions
  - Run the superoptimizer and find if a shorter equivalent version of the instruction
- Identify table containing the equivalent program sequences
- Have been able to rewrite printf using 500 bytes

# Thank You