Introduction to Data Flow Analysis

Outline

- Motivation
- Live Variables Analysis
- Available Expressions Analysis
- Pointer Analysis

Part 2

Motivation
Dead Code Elimination

• No uses for variables \(a_3, b_4, c_5,\) and \(n_6\)

• Assignments to these variables can be deleted

How can we conclude this systematically?
Liveness Analysis of Variables

Find out at each program point $p$, the variables that are used beyond $p$

\[
\begin{align*}
B2: & \quad a_3 = 1; \quad b_4 = 2 \\
& \quad c_5 = 3; \quad n_6 = 6 \\
B4: & \quad a_1 = \phi (1, a_7) \\
& \quad \text{if} \quad a_1 < 6 \\
& \quad B3: \quad a_7 = a_1 + 1 \\
B5: & \quad \text{if} \quad a_1 \leq 11 \\
& \quad D.1200.B = a_1 + 2 \\
& \quad a_9 = D.1200.B + 3 \\
B6: & \quad D.1200.B = a_1 + 2 \\
& \quad a_9 = D.1200.B + 3 \\
B7: & \quad a_2 = \phi (a_1, a_9) \\
& \quad \text{return} \quad a_2
\end{align*}
\]
Liveness Analysis of Variables

Find out at each program point \( p \), the variables that are used beyond \( p \)

\[
\begin{align*}
B2: & \quad a_3 = 1; b_4 = 2 \\
& \quad c_5 = 3; n_6 = 6 \\
B4: & \quad a_1 = \phi (1, a_7) \\
& \quad \text{if } a_1 \leq 6 \\
& \quad F \quad B3 \\
& \quad a_7 = a_1 + 1 \\
B5: & \quad \text{if } a_1 \leq 11 \\
& \quad F \quad B7 \\
& \quad a_2 = \phi (a_1, a_9) \\
& \quad \text{return } a_2 \\
B6: & \quad D.1200.B_8 = a_1 + 2 \\
& \quad a_9 = D.1200.B_8 + 3 \\
B7: & \quad a_2 = \phi (a_1, a_9) \\
& \quad \text{return } a_2 \\
\end{align*}
\]

Which variables are used beyond this point?
\{a_1, a_9\}

What about \( a_2 \)?
Liveness Analysis of Variables

Find out at each program point $p$, the variables that are used beyond $p$

\[
\begin{align*}
\text{B2:} & \quad a_3 = 1; b_4 = 2 \\
& \quad c_5 = 3; n_6 = 6 \\
\text{B4:} & \quad a_1 = \phi (1, a_7) \\
& \quad \text{if } a_1 < 6 \\
\text{B3:} & \quad a_7 = a_1 + 1 \\
\text{B5:} & \quad \text{if } a_1 < 11 \\
\text{B6:} & \quad D.1200_B = a_1 + 2 \\
& \quad a_9 = D.1200_B + 3 \\
\text{B7:} & \quad a_2 = \phi (a_1, a_9) \\
& \quad \text{return } a_2
\end{align*}
\]

Which variables are used beyond this point?

\{a_1\}

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
Liveness Analysis of Variables

Find out at each program point \( p \), the variables that are used beyond \( p \):

\[
\begin{align*}
& \text{B2: } a_3 = 1; b_4 = 2 \\
& \quad c_5 = 3; n_6 = 6 \\
& \text{B4: } a_1 = \phi (1, a_7) \\
& \quad \text{if } a_1 \leq 6 \\
& \quad \text{F} \\
& \text{B3: } a_7 = a_1 + 1 \\
& \text{B5: } \text{if } a_1 \leq 11 \\
& \quad \text{T} \\
& \quad \text{F} \\
& \text{B6: } \text{D.1200}_8 = a_1 + 2 \\
& \quad a_9 = D.1200_8 + 3 \\
& \text{B7: } a_2 = \phi (a_1, a_9) \\
& \quad \text{return } a_2
\end{align*}
\]

Which variables are used beyond this point?

\{a_1, a_9\}

\( \emptyset \) (Conservative assumption)
Liveness Analysis of Variables

Find out at each program point $p$, the variables that are used beyond $p$.

Which variables are used beyond this point?

\{a_1\}

\[a_2 = \phi (a_1, a_9)\]

return $a_2$
Liveness Analysis of Variables

Find out at each program point $p$, the variables that are used beyond $p$

$\begin{align*}
B2: & \quad a_3 = 1; b_4 = 2 \\
& \quad c_5 = 3; n_6 = 6 \\
B4: & \quad a_1 = \phi(1, a_7) \\
& \quad \text{if } a_1 \leq 6 \\
& \quad T \\
& \quad B3: \quad a_7 = a_1 + 1 \\
& \quad F \\
B5: & \quad \text{if } a_1 \leq 11 \\
& \quad F \\
& \quad B6: \quad D.1200_8 = a_1 + 2 \\
& \quad a_9 = D.1200_8 + 3 \\
& \quad T \\
& \quad F \\
B7: & \quad a_2 = \phi(a_1, a_9) \\
& \quad \text{return } a_2 \\
\end{align*}$

Which variables are used beyond this point?

$\{a_7, a_9\}$
Liveness Analysis of Variables

Find out at each program point $p$, the variables that are used beyond $p$

$$\{a_7, a_9\}$$

$$a_3 = 1; b_4 = 2$$
$$c_5 = 3; n_6 = 6$$

$$a_1 = \phi (1, a_7)$$
if $a_1 \leq 6$

$$a_7 = a_1 + 1$$

if $a_1 \leq 11$

$$D.1200.8 = a_1 + 2$$

$$a_9 = D.1200.8 + 3$$

$$a_2 = \phi (a_1, a_9)$$
return $a_2$
Liveness Analysis of Variables: Iteration 2

Find out at each program point \( p \), the variables that are used beyond \( p \)

\[
\begin{align*}
\text{B2:} & \quad a_3 = 1; b_4 = 2 \quad c_5 = 3; n_6 = 6 \\
\text{B4:} & \quad a_1 = \phi (1, a_7) \\
& \text{if } a_1 \leq 6 \\
\text{F:} & \quad a_7 = a_1 + 1 \\
\text{B5:} & \quad \text{if } a_1 \leq 11 \\
\text{F:} & \quad D.1200.8 = a_1 + 2 \\
& \quad a_9 = D.1200.8 + 3 \\
\text{B6:} & \quad \emptyset \text{ (Conservative assumption)} \\
\text{B7:} & \quad a_2 = \phi (a_1, a_9) \\
& \text{return } a_2 \\
\end{align*}
\]
Using Liveness Analysis for Dead Code Elimination

Values of $a_3$, $a_4$, $c_5$, and $n_6$ are guaranteed not to be used.

Why are the values of $a_7$ and $a_9$ meaningful at the exit of B2?

We have assumed a $\phi$ function to be an ordinary expression in which operands are computed along every path reaching the computation.

• Values of $a_3$, $a_4$, $c_5$, and $n_6$ are guaranteed not to be used.
• Why are the values of $a_7$ and $a_9$ meaningful at the exit of B2?
• We have assumed a $\phi$ function to be an ordinary expression in which operands are computed along every path reaching the computation.
Defining Live Variables Analysis

A variable $v$ is live at a program point $p$, if some path from $p$ to program exit contains an r-value occurrence of $v$ which is not preceded by an l-value occurrence of $v$. 

Path based specification

$v$ is live at $p$

$v$ is not live at $p$

$v$ is live at $p$
Defining Data Flow Analysis for Live Variables Analysis

Basic Blocks ≡ Single statements or Maximal groups of sequentially executed statements

Genₖ, Killₖ

Genᵢ, Killᵢ

Genᵢ, Killᵢ

Control Transfer

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay

Local Data Flow Properties for Live Variables Analysis

Genₙ = \{ v | variable v is used within basic block n and is not preceded by a definition of v \}

Killₙ = \{ v | basic block n contains a definition of v \}

r-value occurrence
Value is only read, e.g. x,y,z in
x.sum = y.data + z.data

l-value occurrence
Value is modified, e.g. y in
y = x.lptr

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Local Data Flow Properties for Live Variables Analysis

- $Gen_i$: Use not preceded by definition
  - Upwards exposed use
- $Kill_i$: Definition anywhere in a block
  - Stop the effect from being propagated across a block

Defining Data Flow Analysis for Live Variables Analysis

Global Data Flow Properties

\[
\text{In}_k = Gen_k \cup (Out_k - Kill_k)
\]

\[
Out_k = In_i \cup In_j
\]

Edge based specifications

Notes
Data Flow Equations For Live Variables Analysis

\[ \text{In}_n = (\text{Out}_n - \text{Kill}_n) \cup \text{Gen}_n \]

\[ \text{Out}_n = \begin{cases} \text{BL} & n \text{ is End block} \\ \bigcup_{s \in \text{succ}(n)} \text{In}_s & \text{otherwise} \end{cases} \]

\text{In}_n \text{ and } \text{Out}_n \text{ are sets of variables.}
A variable \( v \) is strongly live if it is used in:
- in statement other than assignment statement, or (this case is same as simple liveness analysis)
- in defining other strongly live variables in an assignment statement (this case is different from simple liveness analysis)
Comparision of Simple and Strong Liveness for our Example

**Simple Liveness**

\[ \{a_7, a_9\} \]

\[ a_3 = 1; b_4 = 2 \]

\[ c_5 = 3; n_6 = 6 \]

\[ \{a_7, a_9\} \]

\[ a_1 = \phi (1, a_7) \]

if \( a_1 \leq 6 \)

\[ \{a_1, a_9\} \]

\[ a_2 = \phi (a_1, a_9) \]

print "Hi"

\[ \{a_1, a_9\} \]

\[ D.1200 = a_1 + 2 \]

\[ a_9 = D.1200 + 3 \]

print "Hello"

\[ \{a_1, a_9\} \]

\[ \{a_1, a_9\} \]

\[ \{a_1, a_9\} \]

\[ \{a_1, a_9\} \]

**Strong Liveness**

\[ \{a_7\} \]

\[ a_3 = 1; b_4 = 2 \]

\[ c_5 = 3; n_6 = 6 \]

\[ \{a_7\} \]

\[ a_1 = \phi (1, a_7) \]

if \( a_1 \leq 6 \)

\[ \{a_1\} \]

\[ a_2 = \phi (a_1, a_9) \]

print "Hi"

\[ \{a_1\} \]

\[ D.1200 = a_1 + 2 \]

\[ a_9 = D.1200 + 3 \]

print "Hello"
• Used for register allocation.
  If variable $x$ is live in a basic block $b$, it is a potential candidate for register allocation.

• Used for dead code elimination.
  If variable $x$ is not live after an assignment $x = \ldots$, then the assignment is redundant and can be deleted as dead code.
Defining Available Expressions Analysis

An expression \( e \) is available at a program point \( p \), if every path from program entry to \( p \) contains an evaluation of \( e \) which is not followed by a definition of any operand of \( e \).

Local Data Flow Properties for Available Expressions Analysis

\[ \text{Gen}_n = \{ e \mid \text{expression } e \text{ is evaluated in basic block } n \text{ and this evaluation is not followed by a definition of any operand of } e \} \]

\[ \text{Kill}_n = \{ e \mid \text{basic block } n \text{ contains a definition of an operand of } e \} \]

<table>
<thead>
<tr>
<th>Entity</th>
<th>Manipulation</th>
<th>Exposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gen(_n)</td>
<td>Expression</td>
<td>Use</td>
</tr>
<tr>
<td>Kill(_n)</td>
<td>Expression</td>
<td>Modification</td>
</tr>
</tbody>
</table>
Data Flow Equations For Available Expressions Analysis

\[
\begin{align*}
\text{In}_n &= \begin{cases} 
BI & \text{n is Start block} \\
\bigcap_{p \in \text{pred}(n)} \text{Out}_p & \text{otherwise}
\end{cases} \\
\text{Out}_n &= \text{Gen}_n \cup (\text{In}_n - \text{Kill}_n)
\end{align*}
\]

Alternatively,

\[
\text{Out}_n = f_n(\text{In}_n), \quad \text{where}
\]

\[
f_n(X) = \text{Gen}_n \cup (X - \text{Kill}_n)
\]

\[
\text{In}_n \text{ and Out}_n \text{ are sets of expressions.}
\]

Using Data Flow Information of Available Expressions Analysis

- Common subexpression elimination
  - If an expression is available at the entry of a block \( b \) \textbf{and}
  - a computation of the expression exists in \( b \) \textbf{such that}
    - it is not preceded by a definition of any of its operands
  
  Then the expression is redundant
- Redundant expression must be \textbf{upwards exposed}
- Expressions in \( \text{Gen}_n \) are \textbf{downwards exposed}
**Code Optimization In Presence of Pointers**

<table>
<thead>
<tr>
<th>Program</th>
<th>Memory graph at statement 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>q = p;</td>
<td></td>
</tr>
<tr>
<td>while (...) {</td>
<td></td>
</tr>
<tr>
<td>q = q-&gt;next;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>p-&gt;data = r1;</td>
<td></td>
</tr>
<tr>
<td>print (q-&gt;data);</td>
<td></td>
</tr>
<tr>
<td>p-&gt;data = r2;</td>
<td></td>
</tr>
</tbody>
</table>

- Is p->data live at the exit of line 5? Can we delete line 5?
  - No, if p and q can be possibly aliased
    (while loop or do-while loop with a circular list)
  - Yes, if p and q are definitely not aliased
    (do-while loop without a circular list)
1 July 2012
Introduction to DFA: Introduction to Pointer Analysis
18/34

**Code Optimization In Presence of Pointers**

```
\begin{align*}
a &= 5 \\
x &= \&a \\
b &= \ast x \\
\end{align*}
```

- Original Program
- Constant Propagation without aliasing
- Constant Propagation with aliasing

**The World of Pointer Analysis**

- Alias Analysis
- Pointer Analysis

- Alias analysis of reference parameters, fields of unions, array indices
- Alias analysis of data pointers
- Points-to analysis of data and function pointers

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
Alias Information Vs. Points-To Information

1. \( x = &a \)
   - "x Points-To a" denoted \( x \rightarrow a \)
   - Neither Symmetric nor Reflexive

2. \( b = x \)
   - "x and b are Aliases" denoted \( x \triangleright= b \)
   - Symmetric and Reflexive

- What about transitivity?
  - Points-To: No.
  - Alias: Depends.

Introduction

Two important dimensions for precise pointer analysis are

- Flow Sensitivity
- Context Sensitivity
A flow-sensitive analysis computes the data flow information at each program point according to the control-flow of a program.

At the exit of node $n_4$

Flow insensitive information:
\{a \rightarrow b, a \rightarrow c, a \rightarrow d\}

Flow sensitive information:
\{a \rightarrow d\}

Context Sensitivity in Interprocedural Analysis

Notes
Context Sensitivity in Interprocedural Analysis

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Issues with Pointer Analysis

- For precise pointer information, we require flow and context sensitive pointer analysis.

- Flow and context sensitive pointer analysis computes a large size of information.

Example of Points-to Analysis

\[
\begin{align*}
&n_1 \quad x = &y \\
&y = &z \\
&z = &u \\
&\{x \rightarrow y, y \rightarrow z, z \rightarrow u\} \\

&n_2 \quad z = y \\
&\{x \rightarrow y, y \rightarrow z, z \rightarrow u\} \\

&n_3 \\
&\{x \rightarrow y, y \rightarrow z, z \rightarrow u\} \\

&n_4 \quad use \ u \\
&use \ x \\
&\{x \rightarrow y, y \rightarrow z, z \rightarrow u, z \rightarrow z, u \rightarrow z\} \\
\end{align*}
\]
Is All This Information Useful?

\[
\begin{align*}
&n_1 \\
&x = \&y \\
&y = \&z \\
&z = \&u \\
&\{x \rightarrow y, y \rightarrow z, z \rightarrow u\}
\end{align*}
\]

\[
\begin{align*}
&n_2 \\
&x = y \\
&z = y \\
&\{x \rightarrow y, y \rightarrow z, u \rightarrow z\}
\end{align*}
\]

\[
\begin{align*}
&n_3 \\
&z = y \\
&\{x \rightarrow y, y \rightarrow z, z \rightarrow u\}
\end{align*}
\]

\[
\begin{align*}
&n_4 \\
&use u \\
&use x \\
&\{x \rightarrow y, y \rightarrow z, u \rightarrow z, u \rightarrow u\}
\end{align*}
\]

---

Improving pointer analysis

For a fast flow and context sensitive pointer analysis, we can reduce the number of computations done at a program point. This can be done in following ways:

- Computing pointer information for only those variables that are being used at some later program point.

- Propagating only the new data flow values obtained in current iteration to the next iteration.
Liveness Based Pointer analysis (L-FCPA)

- A flow and context sensitive pointer analysis
- Pointer information is not computed unless a variable becomes live.
- Strong liveness is used for computing liveness information.
  If basic block contains statement like $x = y$, then $y$ is said to be live, if $x$ is live at the exit of basic block.
- Pointer information is propagated only in live range of the pointer

First Round of Liveness Analysis and Points-to Analysis

```
x = &y
y = &z
z = &u
```

```
\{u\} \quad \{u, x, z\} \quad n_1
\{z\} \quad \{u, x\} \quad n_2
+z = y \quad \{u, x\} \quad n_3
\{u, x\} \quad \{u, x\}
```

```
use u
use x
```

```
\{u\} \quad \{u, x\} \quad n_4
```

Notes
**First Round of Liveness Analysis and Points-to Analysis**

\[
x = &y \\
y = &z \\
z = &u
\]

\[
\{u\} \{u\rightarrow?\}
\]

\[
\{z\rightarrow u\} \{z\}
\]

\[
+z = y \quad n_2
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{z = y\} \quad n_3
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{z = y\} \quad n_3
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

**Second Round of Liveness Analysis and Points-to Analysis**

\[
x = &y \\
y = &z \\
z = &u
\]

\[
\{u\}
\]

\[
\{z\}
\]

\[
+z = y \quad n_2
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{z = y\} \quad n_3
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]

\[
\{u, x\} \quad \{u\rightarrow?, x\rightarrow y\}
\]
Second Round of Liveness Analysis and Points-to Analysis

\[
x = &y \\
y = &z \\
z = &u \\
\{u\} \\
\{u, x, z\}y \}
\]

\[
x = y \\
{z \rightarrow y} \\
{u, x} \\
\{u, x\} \\
\{z \rightarrow y, x \rightarrow y\}
\]

\[
+ z = y \\
{z = y} \\
{u, x} \\
\{u, x\} \\
\{u, x\}
\]

\[
use u \\
use x \\
n_4
\]

\[
\{u\} \{u \rightarrow ?\} \\
\{u \rightarrow ?, x \rightarrow y, z \rightarrow u\} \cup \{y \rightarrow z\}
\]

\[
{z \rightarrow u} \cup \{y \rightarrow z, x \rightarrow y\} \\
{u \rightarrow ?, x \rightarrow y} \\
{u, x} \\
\{u, x\} \\
\{u, x\}
\]

\[
\{u, x\} \{u \rightarrow ?, z \rightarrow u\} \cup \{y \rightarrow z\}
\]

\[
use u \\
use x \\
n_4
\]
Observation

- L-FCPA has 2 fixed point computations:
  - Strong Liveness analysis
  - Points-to analysis

- Liveness and Points-to passes are interdependent.

- Both the computations are done alternatively until final value converges.

Conclusions: New Insights in Pointer Analysis

- Usable pointer information is very small and sparse
- Earlier approaches reported inefficiency and non-scalability because they computed far more information than the actual usable information
- Triumph of *The Genius of AND over the Tyranny of OR*
- Future work
  - Redesign data structures by hiding them behind APIs
    - Current version uses linked lists and linear search
  - Incremental version
  - Using precise pointer information in other passes in GCC
### Precise Context Information is Small and Sparse

Our contributions: Value based termination, liveness

<table>
<thead>
<tr>
<th>Program</th>
<th>Total no. of functions</th>
<th>No. and percentage of functions for call-string counts</th>
<th>0 call strings</th>
<th>1-4 call strings</th>
<th>5-8 call strings</th>
<th>9+ call strings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>L-FCPA FCPA L-FCPA FCPA L-FCPA FCPA L-FCPA FCPA L-FCPA FCPA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lbm</td>
<td>22</td>
<td>16 (72.7%) 3 (13.6%) 6 (27.3%) 19 (86.4%) 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mcf</td>
<td>25</td>
<td>16 (64.0%) 3 (12.0%) 9 (36.0%) 22 (88.0%) 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bzip2</td>
<td>100</td>
<td>88 (88.0%) 38 (38.0%) 12 (62.0%) 62 (62.0%) 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>libquantum</td>
<td>118</td>
<td>100 (84.7%) 56 (47.5%) 22 (18.0%) 6 (5.2%) 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sjeng</td>
<td>151</td>
<td>96 (63.6%) 37 (24.5%) 43 (28.5%) 45 (29.8%) 12 (9.9%) 15 (9.9%) 0 (0.0%) 54 (35.6%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hmmer</td>
<td>584</td>
<td>548 (93.8%) 330 (56.5%) 32 (5.5%) 175 (30.0%) 4 (0.7%) 29 (4.5%) 0 (0.0%) 53 (9.1%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parser</td>
<td>372</td>
<td>246 (66.1%) 76 (20.4%) 118 (31.7%) 135 (36.3%) 4 (1.1%) 63 (16.9%) 0 (0.0%) 98 (26.3%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9+ call strings in L-FCPA: Tot 4, Min 10, Max 52, Mean 32.5, Median 29, Mode 10

### Precise Usable Pointer Information is Small and Sparse

Our contribution: liveness

<table>
<thead>
<tr>
<th>Program</th>
<th>Total no. of BBs</th>
<th>No. and percentage of basic blocks (BBs) for points-to (pt) pair counts</th>
<th>0 pt pairs</th>
<th>1-4 pt pairs</th>
<th>5-8 pt pairs</th>
<th>9+ pt pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>L-FCPA FCPA L-FCPA FCPA L-FCPA FCPA L-FCPA FCPA L-FCPA FCPA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lbm</td>
<td>252</td>
<td>229 (90.9%) 61 (24.2%) 23 (9.1%) 82 (32.5%) 0 (26.2%) 0 (17.1%) 43 (17.1%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mcf</td>
<td>472</td>
<td>396 (83.0%) 180 (38.2%) 116 (24.6%) 16 (0.4%) 66 (26.2%) 0 (26.2%) 0 (0.2%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>libquantum</td>
<td>1642</td>
<td>1520 (92.6%) 793 (48.3%) 119 (7.2%) 796 (48.5%) 3 (0.2%) 46 (2.8%) 0 (0.4%) 7 (0.4%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sjeng</td>
<td>6000</td>
<td>4571 (76.2%) 3239 (54.0%) 1208 (20.1%) 12 (0.2%) 221 (3.7%) 41 (0.7%) 0 (0.0%) 2708 (45.1%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hmmer</td>
<td>14418</td>
<td>13463 (93.5%) 8357 (58.0%) 896 (6.2%) 21 (0.1%) 24 (1.6%) 91 (6.3%) 15 (0.1%) 5949 (41.3%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parser</td>
<td>6875</td>
<td>4823 (70.2%) 1821 (26.5%) 1591 (23.1%) 21 (0.3%) 252 (3.7%) 154 (2.2%) 209 (3.0%) 4875 (70.9%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h264ref</td>
<td>21315</td>
<td>13729 (64.4%) 7160 (33.5%) 6489 (30.1%) 320 (1.5%) 0 (0.0%) 4760 (22.3%) 309 (1.5%) 2708 (45.1%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9+ pt pairs in L-FCPA: Tot 1, Min 12, Max 12, Mean 12.0, Median 12, Mode 12

9+ pt pairs in L-FCPA: Tot 6, Min 10, Max 16, Mean 13.1, Median 13, Mode 10

9+ pt pairs in L-FCPA: Tot 13, Min 9, Max 51, Mean 27.9, Median 18, Mode 9

9+ pt pairs in L-FCPA: Tot 44, Min 9, Max 98, Mean 36.3, Median 31, Mode 9